Repository: open-im-server

Source: /home/cg/code/golang/study-im/open-im-server
Generated: 2026-02-03

Directory

```
open-im-server/
  .golangci.yml
  bootstrap.sh
  docker-compose.yml
  install.sh
  magefile.go
  magefile_unix.go
  magefile_windows.go
  start-config.yml
  _output/
    bin/
      platforms/
        linux/
          amd64/
      tools/
        linux/
          amd64/
    tmp/
    logs/
    tools/
  pkg/
    apistruct/
      config_manager.go
      doc.go
      manage.go
      msg.go
      public.go
    callbackstruct/
      common.go
      constant.go
      conversation.go
      doc.go
      friend.go
      group.go
      message.go
      msg_gateway.go
      push.go
      revoke.go
      user.go
    statistics/
      doc.go
      statistics.go
    rpccache/
      auth.go
      common.go
      conversation.go
      doc.go
      friend.go
      group.go
      online.go
      subscriber.go
      user.go
    mqbuild/
      builder.go
    common/
      convert/
        auth.go
```

```
  black.go
  conversation.go
  doc.go
  friend.go
  group.go
  msg.go
  user.go
  user_test.go
discovery/
  discoveryregister.go
  discoveryregister_test.go
  doc.go
  direct/
    direct_resolver.go
    directconn.go
    doc.go
  kubernetes/
    doc.go
    kubernetes.go
  etcd/
    config_manager.go
    const.go
storage/
  database/
    black.go
    cache.go
    client_config.go
    conversation.go
    doc.go
    friend.go
    friend_request.go
    group.go
    group_member.go
    group_request.go
    log.go
    msg.go
    name.go
    object.go
    seq.go
    seq_user.go
    user.go
    version_log.go
    mgo/
      black.go
      cache.go
      cache_test.go
      client_config.go
      conversation.go
      doc.go
      friend.go
      friend_request.go
      group.go
      group_member.go
      group_request.go
      helpers.go
      log.go
      msg.go
      msg_test.go
      object.go
      seq_conversation.go
      seq_conversation_test.go
      seq_user.go
      user.go
      version_log.go
      version_test.go
  common/
```

```
  types.go
controller/
  auth.go
  black.go
  client_config.go
  conversation.go
  doc.go
  friend.go
  group.go
  msg.go
  msg_transfer.go
  push.go
  s3.go
  third.go
  user.go
model/
  application.go
  black.go
  cache.go
  client_config.go
  conversation.go
  doc.go
  friend.go
  friend_request.go
  group.go
  group_member.go
  group_request.go
  log.go
  msg.go
  object.go
  seq.go
  seq_user.go
  subscribe.go
  user.go
  version_log.go
kafka/
  config.go
  consumer_group.go
  producer.go
  sarama.go
  tls.go
  util.go
  verify.go
cache/
  batch_handler.go
  black.go
  client_config.go
  conversation.go
  doc.go
  friend.go
  group.go
  msg.go
  online.go
  s3.go
  seq_conversation.go
  seq_user.go
  third.go
  token.go
  user.go
  cachekey/
    black.go
    client_config.go
    conversation.go
    doc.go
    friend.go
    group.go
```

```
      msg.go
      online.go
      s3.go
      seq.go
      third.go
      token.go
      user.go
    mcache/
      minio.go
      msg_cache.go
      online.go
      seq_conversation.go
      third.go
      token.go
      tools.go
    redis/
      batch.go
      batch_handler.go
      batch_test.go
      black.go
      client_config.go
      conversation.go
      friend.go
      group.go
      lua_script.go
      lua_script_test.go
      minio.go
      msg.go
      online.go
      online_test.go
      redis_shard_manager.go
      s3.go
      seq_conversation.go
      seq_conversation_test.go
      seq_user.go
      seq_user_test.go
      third.go
      token.go
      user.go
  versionctx/
    rpc.go
    version.go
webhook/
  condition.go
  doc.go
  http_client.go
  http_client_test.go
cmd/
  api.go
  auth.go
  conversation.go
  cron_task.go
  doc.go
  friend.go
  group.go
  msg.go
  msg_gateway.go
  msg_gateway_test.go
  msg_transfer.go
  msg_utils.go
  push.go
  root.go
  third.go
  user.go
prommetrics/
  api.go
```

```
    grpc_auth.go
    grpc_msg.go
    grpc_msggateway.go
    grpc_push.go
    grpc_user.go
    prommetrics.go
    prommetrics_test.go
    rpc.go
    transfer.go
  ginprometheus/
    doc.go
    ginprometheus.go
  servererrs/
    code.go
    doc.go
    predefine.go
    relation.go
  startrpc/
    circuitbreaker.go
    mw.go
    ratelimit.go
    start.go
    tools.go
  config/
    config.go
    constant.go
    doc.go
    env.go
    global.go
    load_config.go
    load_config_test.go
    parse.go
util/
  hashutil/
    id.go
  conversationutil/
    conversationutil.go
    doc.go
  useronline/
    split.go
tools/
  batcher/
    batcher.go
    batcher_test.go
authverify/
  doc.go
  token.go
rpcli/
  auth.go
  conversation.go
  group.go
  msg.go
  msggateway.go
  push.go
  relation.go
  rtc.go
  third.go
  tool.go
  user.go
msgprocessor/
  conversation.go
  doc.go
  options.go
notification/
  msg.go
  common_user/
```

```
      common.go
    grouphash/
      grouphash.go
  localcache/
    cache.go
    cache_test.go
    doc.go
    init.go
    option.go
    tool.go
    link/
      doc.go
      link.go
      link_test.go
    lru/
      doc.go
      lru.go
      lru_expiration.go
      lru_lazy.go
      lru_lazy_test.go
      lru_slot.go
  dbbuild/
    builder.go
    microservices.go
    standalone.go
docs/
  contrib/
  readme/
  images/
  contributing/
version/
  version.go
internal/
  msgtransfer/
    callback.go
    init.go
    online_history_msg_handler.go
    online_msg_to_mongo_handler.go
  push/
    callback.go
    offlinepush_handler.go
    onlinepusher.go
    push.go
    push_handler.go
    offlinepush/
      offlinepusher.go
      fcm/
        push.go
      jpush/
        push.go
        body/
          audience.go
          message.go
          notification.go
          options.go
          platform.go
          pushobj.go
      dummy/
        push.go
      options/
        options.go
      getui/
        body.go
        push.go
  tools/
    cron/
```

```
    cron_task.go
    cron_test.go
    dist_look.go
    msg.go
    s3.go
    user_msg.go
rpc/
  third/
    log.go
    s3.go
    third.go
    tool.go
  user/
    callback.go
    config.go
    notification.go
    online.go
    statistics.go
    user.go
  incrversion/
    batch_option.go
    option.go
  auth/
    auth.go
  conversation/
    callback.go
    conversation.go
    db_map.go
    notification.go
    sync.go
  group/
    cache.go
    callback.go
    convert.go
    db_map.go
    fill.go
    group.go
    notification.go
    statistics.go
    sync.go
  relation/
    black.go
    callback.go
    friend.go
    notification.go
    sync.go
  msg/
    as_read.go
    callback.go
    clear.go
    delete.go
    filter.go
    msg_status.go
    notification.go
    revoke.go
    send.go
    seq.go
    server.go
    statistics.go
    sync_msg.go
    utils.go
    verify.go
api/
  auth.go
  config_manager.go
  conversation.go
```

```
    custom_validator.go
    friend.go
    group.go
    init.go
    msg.go
    prometheus_discovery.go
    ratelimit.go
    router.go
    third.go
    user.go
    jssdk/
      jssdk.go
      sort.go
      tools.go
  msggateway/
    callback.go
    client.go
    client_conn.go
    compressor.go
    compressor_test.go
    constant.go
    context.go
    encoder.go
    http_error.go
    hub_server.go
    init.go
    message_handler.go
    online.go
    options.go
    subscription.go
    user_map.go
    ws_server.go
tools/
  imctl/
    main.go
  versionchecker/
    main.go
  seq/
    main.go
    internal/
      seq.go
  ncpu/
    main.go
    main_test.go
  check-component/
    main.go
  changelog/
    changelog.go
  stress-test-v2/
    main.go
  yamlfmt/
    main.go
    main_test.go
  url2im/
    main.go
    pkg/
      api.go
      buffer.go
      config.go
      http.go
      manage.go
      md5.go
      progress.go
  s3/
    main.go
    internal/
```

```
      conversion.go
  infra/
    main.go
  check-free-memory/
    main.go
cmd/
  main.go
  openim-msgtransfer/
    main.go
  openim-cmdutils/
    main.go
  openim-msggateway/
    main.go
  openim-push/
    main.go
  openim-api/
    main.go
  openim-rpc/
    openim-rpc-auth/
      main.go
    openim-rpc-third/
      main.go
    openim-rpc-group/
      main.go
    openim-rpc-user/
      main.go
    openim-rpc-conversation/
      main.go
    openim-rpc-msg/
      main.go
    openim-rpc-friend/
      main.go
  openim-crontask/
    main.go
scripts/
  template/
test/
  stress-test/
    main.go
  jwt/
    main.go
  webhook/
    msgmodify/
      main.go
  stress-test-v2/
    main.go
  testdata/
    responses/
      login.json
      register.json
      sendMessage.json
    db/
      messages.json
      users.json
    requests/
      login.json
      register.json
      send-message.json
  e2e/
    e2e.go
    e2e_test.go
    upgrade/
    page/
      chat_page.go
      login_page.go
    conformance/
```

```
    rpc/
      message/
      auth/
      conversation/
      friend/
      group/
    web/
    framework/
      ginkgowrapper/
        ginkgowrapper.go
        ginkgowrapper_test.go
      config/
        config.go
        config_test.go
      helpers/
        chat/
          chat.go
    performance/
    scalability/
    api/
      user/
        curd.go
        user.go
      token/
        token.go
study-doc/
deployments/
  deploy/
    clusterRole.yml
    ingress.yml
    kafka-secret.yml
    kafka-service.yml
    kafka-statefulset.yml
    minio-secret.yml
    minio-service.yml
    minio-statefulset.yml
    mongo-secret.yml
    mongo-service.yml
    mongo-statefulset.yml
    openim-api-deployment.yml
    openim-api-service.yml
    openim-config.yml
    openim-crontask-deployment.yml
    openim-msggateway-deployment.yml
    openim-msggateway-service.yml
    openim-msgtransfer-deployment.yml
    openim-msgtransfer-service.yml
    openim-push-deployment.yml
    openim-push-service.yml
    openim-rpc-auth-deployment.yml
    openim-rpc-auth-service.yml
    openim-rpc-conversation-deployment.yml
    openim-rpc-conversation-service.yml
    openim-rpc-friend-deployment.yml
    openim-rpc-friend-service.yml
    openim-rpc-group-deployment.yml
    openim-rpc-group-service.yml
    openim-rpc-msg-deployment.yml
    openim-rpc-msg-service.yml
    openim-rpc-third-deployment.yml
    openim-rpc-third-service.yml
    openim-rpc-user-deployment.yml
    openim-rpc-user-service.yml
    redis-secret.yml
    redis-service.yml
    redis-statefulset.yml
```

```
.github/
  .codecov.yml
  sync-release.yml
  workflows/
    auto-assign-issue.yml
    auto-invite-comment.yml
    changelog.yml
    cla-assistant.yml
    cleanup-after-milestone-prs-merged.yml
    codeql-analysis.yml
    comment-check.yml
    docker-build-and-release-services-images.yml
    go-build-test.yml
    help-comment-issue.yml
    issue-translator.yml
    merge-from-milestone.yml
    publish-docker-image.yml
    remove-unused-labels.yml
    reopen-issue.yml
    update-version-file-on-release.yml
    user-first-interaction.yml
  ISSUE_TEMPLATE/
    bug-report.yml
    config.yml
    deployment.yml
    feature-request.yml
    other.yml
config/
  alertmanager.yml
  discovery.yml
  instance-down-rules.yml
  kafka.yml
  local-cache.yml
  log.yml
  minio.yml
  mongodb.yml
  notification.yml
  openim-api.yml
  openim-crontask.yml
  openim-msggateway.yml
  openim-msgtransfer.yml
  openim-push.yml
  openim-rpc-auth.yml
  openim-rpc-conversation.yml
  openim-rpc-friend.yml
  openim-rpc-group.yml
  openim-rpc-msg.yml
  openim-rpc-third.yml
  openim-rpc-user.yml
  prometheus.yml
  redis.yml
  share.yml
  webhooks.yml
  grafana-template/
    Demo.json
CHANGELOG/
assets/
  logo-gif/
  demo/
  logo/
```

Table of Contents

.golangci.yml

```
# options for analysis running
run:
  # default concurrency is a available CPU number
  concurrency: 4

  # timeout for analysis, e.g. 30s, 5m, default is 1m
  timeout: 5m

  # exit code when at least one issue was found, default is 1
  issues-exit-code: 1

  # include test files or not, default is true
  tests: true

  # list of build tags, all linters use it. Default is empty list.
  build-tags:
    - mytag

  # which dirs to skip: issues from them won't be reported;
  # can use regexp here: generated.*, regexp is applied on full path;
  # default value is empty list, but default dirs are skipped independently
  # from this option's value (see skip-dirs-use-default).
  # "/" will be replaced by current OS file path separator to properly work
  # on Windows.
  # skip-dirs:
  #   - components
  #   - docs
  #   - util
  #   - .*~
  #   - api/swagger/docs


  #   - server/docs
  #   - components/mnt/config/certs
  #   - logs

  # default is true. Enables skipping of directories:
  #   vendor$, third_party$, testdata$, examples$, Godeps$, builtin$
  # skip-dirs-use-default: true

  # which files to skip: they will be analyzed, but issues from them
  # won't be reported. Default value is empty list, but there is
  # no need to include all autogenerated files, we confidently recognize
  # autogenerated files. If it's not please let us know.
  # "/" will be replaced by current OS file path separator to properly work
  # on Windows.
  # skip-files:
  #   - ".*\\.my\\.go$"
  #   - _test.go
  #   - ".*_test.go"
  #   - "mocks/"
  #   - ".github/"
  #   - "logs/"
  #   - "_output/"
  #   - "components/"

  # by default isn't set. If set we pass it to "go list -mod={option}". From "go help modules":
  # If invoked with -mod=readonly, the go command is disallowed from the implicit
  # automatic updating of go.mod described above. Instead, it fails when any changes
  # to go.mod are needed. This setting is most useful to check that go.mod does
  # not need updates, such as in a continuous integration and testing system.
  # If invoked with -mod=vendor, the go command assumes that the vendor
  # directory holds the correct copies of dependencies and ignores
  # the dependency descriptions in go.mod.
```

```
  #modules-download-mode: release|readonly|vendor

  # Allow multiple parallel golangci-lint instances running.
  # If false (default) - golangci-lint acquires file lock on start.
  allow-parallel-runners: true


# output configuration options
output:
  # colored-line-number|line-number|json|tab|checkstyle|code-climate, default is "colored-line-number"
  # format: colored-line-number

  # print lines of code with issue, default is true
  print-issued-lines: true

  # print linter name in the end of issue text, default is true
  print-linter-name: true

  # make issues output unique by line, default is true
  uniq-by-line: true

  # add a prefix to the output file references; default is no prefix
  path-prefix: ""

  # sorts results by: filepath, line and column
  sort-results: true

# all available settings of specific linters
linters-settings:
  bidichk:
    # The following configurations check for all mentioned invisible unicode
    # runes. It can be omitted because all runes are enabled by default.
    left-to-right-embedding: true
    right-to-left-embedding: true
    pop-directional-formatting: true
    left-to-right-override: true
    right-to-left-override: true
    left-to-right-isolate: true
    right-to-left-isolate: true
    first-strong-isolate: true
    pop-directional-isolate: true

  dupl:
    # tokens count to trigger issue, 150 by default
    threshold: 200
  errcheck:
    # report about not checking of errors in type assertions: `a := b.(MyStruct)`;
    # default is false: such cases aren't reported by default.
    check-type-assertions: false

    # report about assignment of errors to blank identifier: `num, _ := strconv.Atoi(numStr)`;
    # default is false: such cases aren't reported by default.
    check-blank: false

    # [deprecated] comma-separated list of pairs of the form pkg:regex
    # the regex is used to ignore names within pkg. (default "fmt:.*").
    # see https://github.com/kisielk/errcheck#the-deprecated-method for details
    #ignore: GenMarkdownTree,os:.*,BindPFlags,WriteTo,Help
    #ignore: (os\.)?std(out|err)\..*|.*Close|.*Flush|os\.Remove(All)?|.*print(f|ln)?|os\.(Un)?Setenv

    # path to a file containing a list of functions to exclude from checking
    # see https://github.com/kisielk/errcheck#excluding-functions for details
    # exclude: errcheck.txt

  errorlint:
    # Check whether fmt.Errorf uses the %w verb for formatting errors. See the readme for caveats
```

```
  errorf: true
  # Check for plain type assertions and type switches
  asserts: true
  # Check for plain error comparisons
  comparison: true

exhaustive:
  # Program elements to check for exhaustiveness.
  # Default: [ switch ]
  check:
    - switch
    - map
  # check switch statements in generated files also
  check-generated: false
  # indicates that switch statements are to be considered exhaustive if a
  # 'default' case is present, even if all enum members aren't listed in the
  # switch
  default-signifies-exhaustive: false
  # enum members matching the supplied regex do not have to be listed in
  # switch statements to satisfy exhaustiveness
  ignore-enum-members: ""
  # consider enums only in package scopes, not in inner scopes
  package-scope-only: false


forbidigo:
#   # Forbid the following identifiers (identifiers are written using regexp):
  forbid:
    # - ^print.*$
    - 'fmt\.Print.*'
    - fmt.Println.* # too much log noise
    - ^unsafe\..*$
    - ^init$
    - ^os.Exit$
    - ^fmt.Print.*$
    - errors.New.*$
    - ^fmt.Println.*$
    - ^panic$
    - painc
#     - ginkgo\\.F.* # these are used just for local development
#   # Exclude godoc examples from forbidigo checks.  Default is true.
#   exclude_godoc_examples: false

funlen:
  lines: 220
  statements: 80

gocognit:
  # minimal code complexity to report, 30 by default (but we recommend 10-20)
  min-complexity: 30

goconst:
  # minimal length of string constant, 3 by default
  min-len: 3
  # minimal occurrences count to trigger, 3 by default
  min-occurrences: 3
  # ignore test files, false by default
  ignore-tests: false
  # look for existing constants matching the values, true by default
  match-constant: true
  # search also for duplicated numbers, false by default
  numbers: false
  # minimum value, only works with goconst.numbers, 3 by default
  min: 3
  # maximum value, only works with goconst.numbers, 3 by default
  max: 3
```

```
  # ignore when constant is not used as function argument, true by default
  ignore-calls: true

gocritic:
  # Which checks should be enabled; can't be combined with 'disabled-checks';
  # See https://go-critic.github.io/overview#checks-overview
  # To check which checks are enabled run `GL_DEBUG=gocritic golangci-lint run`
  # By default list of stable checks is used.
  enabled-checks:
    #- rangeValCopy
    - ruleguard

  # Which checks should be disabled; can't be combined with 'enabled-checks'; default is empty
  disabled-checks:
    - regexpMust
    - ifElseChain
    #- exitAfterDefer

  # Enable multiple checks by tags, run `GL_DEBUG=gocritic golangci-lint run` to see all tags and checks.
  # Empty list by default. See https://github.com/go-critic/go-critic#usage -> section "Tags".
  enabled-tags:
    - performance
  disabled-tags:
    - experimental

  # Settings passed to gocritic.
  # The settings key is the name of a supported gocritic checker.
  # The list of supported checkers can be find in https://go-critic.github.io/overview.
  settings:
    captLocal: # must be valid enabled check name
      # whether to restrict checker to params only (default true)
      paramsOnly: true
    elseif:
      # whether to skip balanced if-else pairs (default true)
      skipBalanced: true
    hugeParam:
      # size in bytes that makes the warning trigger (default 80)
      sizeThreshold: 80
    rangeExprCopy:
      # size in bytes that makes the warning trigger (default 512)
      sizeThreshold: 512
      # whether to check test functions (default true)
      skipTestFuncs: true
    rangeValCopy:
      # size in bytes that makes the warning trigger (default 128)
      sizeThreshold: 32
      # whether to check test functions (default true)
      skipTestFuncs: true
    ruleguard:
      # path to a gorules file for the ruleguard checker
      rules: ''
    underef:
      # whether to skip (*x).method() calls where x is a pointer receiver (default true)
      skipRecvDeref: true

gocyclo:
  # minimal code complexity to report, 30 by default (but we recommend 10-20)
  min-complexity: 30
cyclop:
  # the maximal code complexity to report
  max-complexity: 50
  # the maximal average package complexity. If it's higher than 0.0 (float) the check is enabled (default 0.0)
  package-average: 0.0
  # should ignore tests (default false)
  skip-tests: false
godot:
```

```
  # comments to be checked: `declarations`, `toplevel`, or `all`
  scope: declarations
  # list of regexps for excluding particular comment lines from check
  exclude:
    # example: exclude comments which contain numbers
    - '[0-9]+'
    - 'func\s+\w+'
    - 'FIXME:'
    - '.*func.*'
  # check that each sentence starts with a capital letter
  capital: true
godox:
  # report any comments starting with keywords, this is useful for TODO or FIXME comments that
  # might be left in the code accidentally and should be resolved before merging
  keywords: # default keywords are TODO, BUG, and FIXME, these can be overwritten by this setting
    #- TODO
    - BUG
    - FIXME
    #- NOTE
    - OPTIMIZE # marks code that should be optimized before merging
    - HACK # marks hack-arounds that should be removed before merging
gofmt:
  # simplify code: gofmt with `-s` option, true by default
  simplify: true

gofumpt:
  # Select the Go version to target. The default is `1.18`.
  go-version: "1.21"

  # Choose whether or not to use the extra rules that are disabled
  # by default
  extra-rules: false

# goheader:
  # values:
    # const:
      # define here const type values in format k:v, for example:
      # COMPANY: MY COMPANY
    # regexp:
      # define here regexp type values, for example
      # AUTHOR: .*@mycompany\.com
  # template: # |-
    # put here copyright header template for source code files, for example:
    # Note: {{ YEAR }} is a builtin value that returns the year relative to the current machine time.
    #
    # {{ AUTHOR }} {{ COMPANY }} {{ YEAR }}
    # SPDX-License-Identifier: Apache-2.0

    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at:

    #   http://www.apache.org/licenses/LICENSE-2.0

    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
  # template-path:
    # also as alternative of directive 'template' you may put the path to file with the template source

goimports:
  # put imports beginning with prefix after 3rd-party packages;
  # it's a comma-separated list of prefixes
  local-prefixes: github.com/openimsdk/open-im-server
```

```yaml
gomnd:
  # List of enabled checks, see https://github.com/tommy-muehle/go-mnd/#checks for description.
  # Default: ["argument", "case", "condition", "operation", "return", "assign"]
  checks:
    - argument
    - case
    - condition
    - operation
    - return
    - assign
  # List of numbers to exclude from analysis.
  # The numbers should be written as string.
  # Values always ignored: "1", "1.0", "0" and "0.0"
  # Default: []
  ignored-numbers:
    - '0666'
    - '0755'
    - '42'
  # List of file patterns to exclude from analysis.
  # Values always ignored: `.+_test.go`
  # Default: []
  ignored-files:
    - 'magic1_.+\.go$'
  # List of function patterns to exclude from analysis.
  # Following functions are always ignored: `time.Date`,
  # `strconv.FormatInt`, `strconv.FormatUint`, `strconv.FormatFloat`,
  # `strconv.ParseInt`, `strconv.ParseUint`, `strconv.ParseFloat`.
  # Default: []
  ignored-functions:
    - '^math\.'
    - '^webhook\.StatusText$'
gomoddirectives:
  # Allow local `replace` directives. Default is false.
  replace-local: true
  # List of allowed `replace` directives. Default is empty.
  replace-allow-list:
    - google.golang.org/grpc

  # Allow to not explain why the version has been retracted in the `retract` directives. Default is false.
  retract-allow-no-explanation: false
  # Forbid the use of the `exclude` directives. Default is false.
  exclude-forbidden: false

gomodguard:
  allowed:
    modules:
      - gorm.io/gen                                                 # List of allowed modules
      - gorm.io/gorm
      - gorm.io/driver/mysql
      - k8s.io/klog
      - github.com/allowed/module
      - go.mongodb.org/mongo-driver/mongo
      # - gopkg.in/yaml.v2
    domains:                                                      # List of allowed module domains
      - google.golang.org
      - gopkg.in
      - golang.org
      - github.com
      - go.mongodb.org
      - go.uber.org
      - openim.io
      - go.etcd.io
  blocked:
    versions:
      - github.com/MakeNowJust/heredoc:
```

```
          version: "> 2.0.9"
          reason: "use the latest version"
    local_replace_directives: false     # Set to true to raise lint issues for packages that are loaded from a loc

gosec:
  # To select a subset of rules to run.
  # Available rules: https://github.com/securego/gosec#available-rules
  includes:
    - G401
    - G306
    - G101
  # To specify a set of rules to explicitly exclude.
  # Available rules: https://github.com/securego/gosec#available-rules
  excludes:
    - G204
  # Exclude generated files
  exclude-generated: true
  # Filter out the issues with a lower severity than the given value. Valid options are: low, medium, high.
  severity: "low"
  # Filter out the issues with a lower confidence than the given value. Valid options are: low, medium, high.
  confidence: "low"
  # To specify the configuration of rules.
  # The configuration of rules is not fully documented by gosec:
  # https://github.com/securego/gosec#configuration
  # https://github.com/securego/gosec/blob/569328eade2ccbad4ce2d0f21ee158ab5356a5cf/rules/rulelist.go#L60-L102
  config:
    G306: "0600"
    G101:
      pattern: "(?i)example"
      ignore_entropy: false
      entropy_threshold: "80.0"
      per_char_threshold: "3.0"
      truncate: "32"

gosimple:
  # Select the Go version to target. The default is '1.13'.
  go: "1.20"
  # https://staticcheck.io/docs/options#checks
  checks: [ "all" ]

govet:
  # settings per analyzer
  settings:
    printf: # analyzer name, run `go tool vet help` to see all analyzers
      funcs: # run `go tool vet help printf` to see available settings for `printf` analyzer
        - (github.com/golangci/golangci-lint/pkg/logutils.Log).Infof
        - (github.com/golangci/golangci-lint/pkg/logutils.Log).Warnf
        - (github.com/golangci/golangci-lint/pkg/logutils.Log).Errorf
        - (github.com/golangci/golangci-lint/pkg/logutils.Log).Fatalf

  # enable or disable analyzers by name
  enable:
    - atomicalign
  enable-all: false
  disable:
    - shadow
  disable-all: false

depguard:
  rules:
    prevent_unmaintained_packages:
      list-mode: lax # allow unless explicitely denied
      files:
        - $all
        - "!$test"
      allow:
```

```
        - $gostd
      deny:
        - pkg: io/ioutil
          desc: "replaced by io and os packages since Go 1.16: https://tip.golang.org/doc/go1.16#ioutil"
        - pkg: github.com/OpenIMSDK
          desc: "The OpenIM organization has been replaced with lowercase, please do not use uppercase organizatio
        - pkg: log
          desc: "We have a wrapped log package at openim, we recommend you to use our wrapped log package, https:/
        - pkg: errors
          desc: "We have a wrapped errors package at openim, we recommend you to use our wrapped errors package, h

importas:
  # if set to `true`, force to use alias.
  no-unaliased: true
  # List of aliases
  alias:
    # using `servingv1` alias for `knative.dev/serving/pkg/apis/serving/v1` package
    - pkg: knative.dev/serving/pkg/apis/serving/v1
      alias: servingv1
    - pkg: gopkg.in/yaml.v2
      alias: yaml
    # using `autoscalingv1alpha1` alias for `knative.dev/serving/pkg/apis/autoscaling/v1alpha1` package
    - pkg: knative.dev/serving/pkg/apis/autoscaling/v1alpha1
      alias: autoscalingv1alpha1
    # You can specify the package path by regular expression,
    # and alias by regular expression expansion syntax like below.
    # see https://github.com/julz/importas#use-regular-expression for details
    - pkg: knative.dev/serving/pkg/apis/(\w+)/(v[\w\d]+)
      alias: $1$2

ireturn:
  # ireturn allows using `allow` and `reject` settings at the same time.
  # Both settings are lists of the keywords and regular expressions matched to interface or package names.
  # keywords:
  # - `empty` for `interface{}`
  # - `error` for errors
  # - `stdlib` for standard library
  # - `anon` for anonymous interfaces

  # By default, it allows using errors, empty interfaces, anonymous interfaces,
  # and interfaces provided by the standard library.
  allow:
  - anon
  - error
  - empty
  - stdlib
  # You can specify idiomatic endings for interface
  - (or|er)$

  # Reject patterns
  reject:
  - github.com\/user\/package\/v4\.Type

lll:
  # max line length, lines longer will be reported. Default is 250.
  # '\t' is counted as 1 character by default, and can be changed with the tab-width option
  line-length: 250
  # tab width in spaces. Default to 1.
  tab-width: 4
misspell:
  # Correct spellings using locale preferences for US or UK.
  # Default is to use a neutral variety of English.
  # Setting locale to US will correct the British spelling of 'colour' to 'color'.
  locale: US
  ignore-words:
    - someword
```

```yaml
nakedret:
  # make an issue if func has more lines of code than this setting and it has naked returns; default is 30
  max-func-lines: 30

nestif:
  # minimal complexity of if statements to report, 5 by default
  min-complexity: 4

nilnil:
  # By default, nilnil checks all returned types below.
  checked-types:
    - ptr
    - func
    - iface
    - map
    - chan

nlreturn:
  # size of the block (including return statement that is still "OK")
  # so no return split required.
  block-size: 1

nolintlint:
  # Disable to ensure that all nolint directives actually have an effect. Default is true.
  allow-unused: false
  # Exclude following linters from requiring an explanation.  Default is [].
  allow-no-explanation: [ ]
  # Enable to require an explanation of nonzero length after each nolint directive. Default is false.
  require-explanation: false
  # Enable to require nolint directives to mention the specific linter being suppressed. Default is false.
  require-specific: true

prealloc:
  # XXX: we don't recommend using this linter before doing performance profiling.
  # For most programs usage of prealloc will be a premature optimization.

  # Report preallocation suggestions only on simple loops that have no returns/breaks/continues/gotos in them.
  # True by default.
  simple: true
  range-loops: true # Report preallocation suggestions on range loops, true by default
  for-loops: false # Report preallocation suggestions on for loops, false by default

promlinter:
  # Promlinter cannot infer all metrics name in static analysis.
  # Enable strict mode will also include the errors caused by failing to parse the args.
  strict: false
  # Please refer to https://github.com/yeya24/promlinter#usage for detailed usage.
  disabled-linters:
   - "Help"
   - "MetricUnits"
   - "Counter"
   - "HistogramSummaryReserved"
   - "MetricTypeInName"
   - "ReservedChars"
   - "CamelCase"

predeclared:
  # comma-separated list of predeclared identifiers to not report on
  ignore: ""
  # include method names and field names (i.e., qualified names) in checks
  q: false
rowserrcheck:
  packages:
    - github.com/jmoiron/sqlx

revive:
```

```
  # see https://github.com/mgechev/revive#available-rules for details.
  ignore-generated-header: true
  severity: warning
  rules:
    - name: indent-error-flow
      severity: warning
    - name: exported
      severity: warning
    - name: var-naming
      arguments: [ [ "OpenIM"] ]
      # arguments: [ ["ID", "HTTP", "URL", "URI", "API", "APIKey", "Token", "TokenID", "TokenSecret", "TokenKey",
    - name: atomic
    - name: line-length-limit
      severity: error
      arguments: [200]
    - name: unhandled-error
      arguments : ["fmt.Printf", "myFunction"]

staticcheck:
  # Select the Go version to target. The default is '1.13'.
  go: "1.20"
  # https://staticcheck.io/docs/options#checks
  checks: [ "all" ]

stylecheck:
  # Select the Go version to target. The default is '1.13'.
  go: "1.20"

  # https://staticcheck.io/docs/options#checks
  checks: [ "all", "-ST1000", "-ST1003", "-ST1016", "-ST1020", "-ST1021", "-ST1022" ]
  # https://staticcheck.io/docs/options#dot_import_whitelist
  dot-import-whitelist:
    - fmt
  # https://staticcheck.io/docs/options#initialisms
  initialisms: [ "ACL", "API", "ASCII", "CPU", "CSS", "DNS", "EOF", "GUID", "HTML", "HTTP", "HTTPS", "ID", "IP", "
  # https://staticcheck.io/docs/options#http_status_code_whitelist
  http-status-code-whitelist: [ "200", "400", "404", "500" ]

tagliatelle:
  # check the struck tag name case
  case:
    # use the struct field name to check the name of the struct tag
    use-field-name: true
    rules:
      # any struct tag type can be used.
      # support string case: `camel`, `pascal`, `kebab`, `snake`, `goCamel`, `goPascal`, `goKebab`, `goSnake`, `up
      json: camel
      yaml: camel
      xml: camel
      bson: camel
      avro: snake
      mapstructure: kebab

testpackage:
  # regexp pattern to skip files
  skip-regexp: (id|export|internal)_test\.go
thelper:
  # The following configurations enable all checks. It can be omitted because all checks are enabled by default.
  # You can enable only required checks deleting unnecessary checks.
  test:
    first: true
    name: true
    begin: true
  benchmark:
    first: true
    name: true
```

```
    begin: true
  tb:
    first: true
    name: true
    begin: true

tenv:
  # The option `all` will run against whole test files (`_test.go`) regardless of method/function signatures.
  # By default, only methods that take `*testing.T`, `*testing.B`, and `testing.TB` as arguments are checked.
  all: false

unparam:
  # Inspect exported functions, default is false. Set to true if no external program/library imports your code.
  # XXX: if you enable this setting, unparam will report a lot of false-positives in text editors:
  # if it's called for subdir of a project it can't find external interfaces. All text editor integrations
  # with golangci-lint call it on a directory with the changed file.
  check-exported: false
# unused:
  # treat code as a program (not a library) and report unused exported identifiers; default is false.
  # XXX: if you enable this setting, unused will report a lot of false-positives in text editors:
  # if it's called for subdir of a project it can't find funcs usages. All text editor integrations
  # with golangci-lint call it on a directory with the changed file.
whitespace:
  multi-if: false   # Enforces newlines (or comments) after every multi-line if statement
  multi-func: false # Enforces newlines (or comments) after every multi-line function signature

wrapcheck:
  # An array of strings that specify substrings of signatures to ignore.
  # If this set, it will override the default set of ignored signatures.
  # See https://github.com/tomarrell/wrapcheck#configuration for more information.
  ignoreSigs:
    - .Errorf(
    - errors.New(
    - errors.Unwrap(
    - .Wrap(
    - .WrapMsg(
    - .Wrapf(
    - .WithMessage(
    - .WithMessagef(
    - .WithStack(
  ignorePackageGlobs:
      - encoding/*
      - github.com/pkg/*
      - github.com/openimsdk/*
      - github.com/OpenIMSDK/*

wsl:
  # If true append is only allowed to be cuddled if appending value is
  # matching variables, fields or types on line above. Default is true.
  strict-append: true
  # Allow calls and assignments to be cuddled as long as the lines have any
  # matching variables, fields or types. Default is true.
  allow-assign-and-call: true
  # Allow assignments to be cuddled with anything. Default is false.
  allow-assign-and-anything: false
  # Allow multiline assignments to be cuddled. Default is true.
  allow-multiline-assign: true
  # Allow declarations (var) to be cuddled.
  allow-cuddle-declarations: false
  # Allow trailing comments in ending of blocks
  allow-trailing-comment: false
  # Force newlines in end of case at this limit (0 = never).
  force-case-trailing-whitespace: 0
  # Force cuddling of err checks with err var assignment
  force-err-cuddling: false
  # Allow leading comments to be separated with empty liens
```

```
    allow-separated-leading-comment: false
  makezero:
    # Allow only slices initialized with a length of zero. Default is false.
    always: false

  # The custom section can be used to define linter plugins to be loaded at runtime. See README doc
  #  for more info.
  #custom:
    # Each custom linter should have a unique name.
    #example:
      # The path to the plugin *.so. Can be absolute or local. Required for each custom linter
      #path: /path/to/example.so
      # The description of the linter. Optional, just for documentation purposes.
      #description: This is an example usage of a plugin linter.
      # Intended to point to the repo location of the linter. Optional, just for documentation purposes.
      #original-url: github.com/golangci/example-linter

linters:
  # please, do not use `enable-all`: it's deprecated and will be removed soon.
  # inverted configuration with `enable-all` and `disable` is not scalable during updates of golangci-lint
  # enable-all: true
  disable-all: true
  enable:
    - typecheck     # Basic type checking
    - gofmt         # Format check
    - govet         # Go's standard linting tool
    - gosimple      # Suggestions for simplifying code
    - errcheck
    - decorder
    - ineffassign
    - forbidigo
    - revive
    - reassign
    - tparallel
    - unconvert
    - fieldalignment
    - dupl
    - dupword
    - errname
    - gci
    - exhaustive
    - gocritic
    - goprintffuncname
    - gomnd
    - goconst
    - gosec
    - misspell      # Spelling mistakes
    - staticcheck   # Static analysis
    - unused        # Checks for unused code
    # - goimports     # Checks if imports are correctly sorted and formatted
    - godot         # Checks for comment punctuation
    - bodyclose     # Ensures HTTP response body is closed
    - stylecheck    # Style checker for Go code
    - unused        # Checks for unused code
    - errcheck      # Checks for missed error returns
  fast: true

issues:
  # List of regexps of issue texts to exclude, empty list by default.
  # But independently from this option we use default exclude patterns,
  # it can be disabled by `exclude-use-default: false`. To list all
  # excluded by default patterns execute `golangci-lint run --help`
  exclude:
    - tools/.*
    - test/.*
    - components/*
```

```
  - third_party/.*

# Excluding configuration per-path, per-linter, per-text and per-source
exclude-rules:
  - linters:
    - revive
    path: (log/.*)\.go

  - linters:
    - wrapcheck
    path: (cmd/.*|pkg/.*)\.go

  - linters:
    - typecheck
      #path: (pkg/storage/.*)\.go
    path: (internal/.*|pkg/.*)\.go

  - path: (cmd/.*|test/.*|tools/.*|internal/pump/pumps/.*)\.go
    linters:
      - forbidigo

  - path: (cmd/[a-z]*/.*|store/.*)\.go
    linters:
      - dupl

  - linters:
      - gocritic
    text: (hugeParam:|rangeValCopy:)

  - path: (cmd/[a-z]*/.*)\.go
    linters:
      - lll

  - path: (validator/.*|code/.*|validator/.*|watcher/watcher/.*)
    linters:
      - gochecknoinits

  - path: (internal/.*/options|internal/pump|pkg/log/options.go|internal/authzserver|tools/)
    linters:
      - tagliatelle

  - path: (pkg/app/.*)\.go
    linters:
      - unused
      - forbidigo

  # Exclude some staticcheck messages
  - linters:
      - staticcheck
    text: "SA9003:"

  # Exclude lll issues for long lines with go:generate
  - linters:
      - lll
    source: "^//go:generate "

  - text: ".*[\u4e00-\u9fa5]+.*"
    linters:
      - golint
    source: "^//.*$"

# Independently from option `exclude` we use default exclude patterns,
# it can be disabled by this option. To list all
# excluded by default patterns execute `golangci-lint run --help`.
# Default value for this option is true.
exclude-use-default: true
```

```
  # The default value is false. If set to true exclude and exclude-rules
  # regular expressions become case sensitive.
  exclude-case-sensitive: false

  # The list of ids of default excludes to include or disable. By default it's empty.
  include:
    - EXC0002 # disable excluding of issues about comments from golint

  # Maximum issues count per one linter. Set to 0 to disable. Default is 50.
  max-issues-per-linter: 0

  # Maximum count of issues with the same text. Set to 0 to disable. Default is 3.
  max-same-issues: 0

  # Show only new issues: if there are unstaged changes or untracked files,
  # only those changes are analyzed, else only changes in HEAD~ are analyzed.
  # It's a super-useful option for integration of golangci-lint into existing
  # large codebase. It's not practical to fix all existing issues at the moment
  # of integration: much better don't allow issues in new code.
  # Default is false.
  new: false

  # Show only new issues created after git revision `REV`
  # new-from-rev: REV

  # Show only new issues created in git patch with set file path.
  #new-from-patch: path/to/patch/file

  # Fix found issues (if it's supported by the linter)
  fix: true

severity:
  # Default value is empty string.
  # Set the default severity for issues. If severity rules are defined and the issues
  # do not match or no severity is provided to the rule this will be the default
  # severity applied. Severities should match the supported severity names of the
  # selected out format.
  # - Code climate: https://docs.codeclimate.com/docs/issues#issue-severity
  # -   Checkstyle: https://checkstyle.sourceforge.io/property_types.html#severity
  # -       Github: https://help.github.com/en/actions/reference/workflow-commands-for-github-actions#setting-an-err
  default-severity: error

  # The default value is false.
  # If set to true severity-rules regular expressions become case sensitive.
  case-sensitive: false

  # Default value is empty list.
  # When a list of severity rules are provided, severity information will be added to lint
  # issues. Severity rules have the same filtering capability as exclude rules except you
  # are allowed to specify one matcher per severity rule.
  # Only affects out formats that support setting severity information.
  rules:
    - linters:
      - dupl
      severity: info
```

bootstrap.sh

```
#!/bin/bash

if [[ ":$PATH:" == *":$HOME/.local/bin:"* ]]; then
    TARGET_DIR="$HOME/.local/bin"
else
    TARGET_DIR="/usr/local/bin"
    echo "Using /usr/local/bin as the installation directory. Might require sudo permissions."
fi

if ! command -v mage &> /dev/null; then
    echo "Installing Mage to $TARGET_DIR ..."
    GOBIN=$TARGET_DIR go install github.com/magefile/mage@latest
fi

if ! command -v mage &> /dev/null; then
    echo "Mage installation failed."
    echo "Please ensure that $TARGET_DIR is in your \$PATH."
    exit 1
fi

echo "Mage installed successfully."

go mod download
```

docker-compose.yml

```yaml
networks:
  openim:
    driver: bridge

services:
  mongodb:
    image: "${MONGO_IMAGE}"
    ports:
      - "37017:27017"
    container_name: mongo
    command: >
      bash -c '
      docker-entrypoint.sh mongod --wiredTigerCacheSizeGB $$wiredTigerCacheSizeGB --auth &
      until mongosh -u $$MONGO_INITDB_ROOT_USERNAME -p $$MONGO_INITDB_ROOT_PASSWORD --authenticationDatabase admin -
        echo "Waiting for MongoDB to start..."
        sleep 1
      done &&
      mongosh -u $$MONGO_INITDB_ROOT_USERNAME -p $$MONGO_INITDB_ROOT_PASSWORD --authenticationDatabase admin --eval
      db = db.getSiblingDB(\"$$MONGO_INITDB_DATABASE\");
      if (!db.getUser(\"$$MONGO_OPENIM_USERNAME\")) {
        db.createUser({
          user: \"$$MONGO_OPENIM_USERNAME\",
          pwd: \"$$MONGO_OPENIM_PASSWORD\",
          roles: [{role: \"readWrite\", db: \"$$MONGO_INITDB_DATABASE\"}]
        });
        print(\"User created successfully: \");
        print(\"Username: $$MONGO_OPENIM_USERNAME\");
        print(\"Password: $$MONGO_OPENIM_PASSWORD\");
        print(\"Database: $$MONGO_INITDB_DATABASE\");
      } else {
        print(\"User already exists in database: $$MONGO_INITDB_DATABASE, Username: $$MONGO_OPENIM_USERNAME\");
      }
      " &&
      tail -f /dev/null
      '
    volumes:
      - "${DATA_DIR}/components/mongodb/data/db:/data/db"
      - "${DATA_DIR}/components/mongodb/data/logs:/data/logs"
      - "${DATA_DIR}/components/mongodb/data/conf:/etc/mongo"
      - "${MONGO_BACKUP_DIR}:/data/backup"
    environment:
      - TZ=Asia/Shanghai
      - wiredTigerCacheSizeGB=1
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=openIM123
      - MONGO_INITDB_DATABASE=openim_v3
      - MONGO_OPENIM_USERNAME=openIM
      - MONGO_OPENIM_PASSWORD=openIM123
    restart: always
    networks:
      - openim

  redis:
    image: "${REDIS_IMAGE}"
    container_name: redis
    ports:
      - "16379:6379"
    volumes:
      - "${DATA_DIR}/components/redis/data:/data"
      - "${DATA_DIR}/components/redis/config/redis.conf:/usr/local/redis/config/redis.conf"
    environment:
      TZ: Asia/Shanghai
    restart: always
    sysctls:
```

```
    net.core.somaxconn: 1024
  command: >
    redis-server
    --requirepass openIM123
    --appendonly yes
    --aof-use-rdb-preamble yes
    --save ""
  networks:
    - openim

etcd:
  image: "${ETCD_IMAGE}"
  container_name: etcd
  ports:
    - "12379:2379"
    - "12380:2380"
  environment:
    - ETCD_NAME=s1
    - ETCD_DATA_DIR=/etcd-data
    - ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379
    - ETCD_ADVERTISE_CLIENT_URLS=http://0.0.0.0:2379
    - ETCD_LISTEN_PEER_URLS=http://0.0.0.0:2380
    - ETCD_INITIAL_ADVERTISE_PEER_URLS=http://0.0.0.0:2380
    - ETCD_INITIAL_CLUSTER=s1=http://0.0.0.0:2380
    - ETCD_INITIAL_CLUSTER_TOKEN=tkn
    - ETCD_INITIAL_CLUSTER_STATE=new
    - ALLOW_NONE_AUTHENTICATION=no

    ## Optional: Enable etcd authentication by setting the following credentials
    # - ETCD_ROOT_USER=root
    # - ETCD_ROOT_PASSWORD=openIM123
    # - ETCD_USERNAME=openIM
    # - ETCD_PASSWORD=openIM123
  volumes:
    - "${DATA_DIR}/components/etcd:/etcd-data"
  command: >
    /bin/sh -c '
      etcd &
      export ETCDCTL_API=3
      echo "Waiting for etcd to become healthy..."
      until etcdctl --endpoints=http://127.0.0.1:2379 endpoint health &>/dev/null; do
        echo "Waiting for ETCD to start..."
        sleep 1
      done

      echo "etcd is healthy."

      if [ -n "$${ETCD_ROOT_USER}" ] && [ -n "$${ETCD_ROOT_PASSWORD}" ] && [ -n "$${ETCD_USERNAME}" ] && [ -n "$${
        echo "Authentication credentials provided. Setting up authentication..."

      echo "Checking authentication status..."
      if ! etcdctl --endpoints=http://127.0.0.1:2379 auth status | grep -q "Authentication Status: true"; then
        echo "Authentication is disabled. Creating users and enabling..."

        # Create users and setup permissions
        etcdctl --endpoints=http://127.0.0.1:2379 user add $${ETCD_ROOT_USER} --new-user-password=$${ETCD_ROOT_PAS
        etcdctl --endpoints=http://127.0.0.1:2379 user add $${ETCD_USERNAME} --new-user-password=$${ETCD_PASSWORD}

        etcdctl --endpoints=http://127.0.0.1:2379 role add openim-role || true
        etcdctl --endpoints=http://127.0.0.1:2379 role grant-permission openim-role --prefix=true readwrite / || t
        etcdctl --endpoints=http://127.0.0.1:2379 role grant-permission openim-role --prefix=true readwrite "" ||
        etcdctl --endpoints=http://127.0.0.1:2379 user grant-role $${ETCD_USERNAME} openim-role || true

        etcdctl --endpoints=http://127.0.0.1:2379 user grant-role $${ETCD_ROOT_USER} $${ETCD_USERNAME} root || tru

        echo "Enabling authentication..."
```

```
        etcdctl --endpoints=http://127.0.0.1:2379 auth enable
        echo "Authentication enabled successfully"
      else
        echo "Authentication is already enabled. Checking OpenIM user..."

        # Check if openIM user exists and can perform operations
        if ! etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_USERNAME}:$${ETCD_PASSWORD} put /test/auth "
          echo "OpenIM user test failed. Recreating user with root credentials..."

          # Try to create/update the openIM user using root credentials
          etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_ROOT_USER}:$${ETCD_ROOT_PASSWORD} user add $${E
          etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_ROOT_USER}:$${ETCD_ROOT_PASSWORD} role add open
          etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_ROOT_USER}:$${ETCD_ROOT_PASSWORD} role grant-pe
          etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_ROOT_USER}:$${ETCD_ROOT_PASSWORD} role grant-pe
          etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_ROOT_USER}:$${ETCD_ROOT_PASSWORD} user grant-ro
          etcdctl --endpoints=http://127.0.0.1:2379 user grant-role $${ETCD_ROOT_USER} $${ETCD_USERNAME} root || t

          echo "OpenIM user recreated with required permissions"
        else
          echo "OpenIM user exists and has correct permissions"
          etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_USERNAME}:$${ETCD_PASSWORD} del /test/auth &>/d
        fi
      fi
      echo "Testing authentication with OpenIM user..."
      if etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_USERNAME}:$${ETCD_PASSWORD} put /test/auth "auth
        echo "Authentication working properly"
        etcdctl --endpoints=http://127.0.0.1:2379 --user=$${ETCD_USERNAME}:$${ETCD_PASSWORD} del /test/auth
      else
        echo "WARNING: Authentication test failed"
        fi
      else
        echo "No authentication credentials provided. Running in no-auth mode."
        echo "To enable authentication, set ETCD_ROOT_USER, ETCD_ROOT_PASSWORD, ETCD_USERNAME, and ETCD_PASSWORD e
      fi

      tail -f /dev/null
    '
  restart: always
  networks:
    - openim

kafka:
  image: "${KAFKA_IMAGE}"
  container_name: kafka
  user: root
  restart: always
  ports:
    - "19094:9094"
  volumes:
    - "${DATA_DIR}/components/kafka:/bitnami/kafka"
  environment:
    #KAFKA_HEAP_OPTS: "-Xms128m -Xmx256m"
    TZ: Asia/Shanghai
    # Unique identifier for the Kafka node (required in controller mode)
    KAFKA_CFG_NODE_ID: 0
    # Defines the roles this Kafka node plays: broker, controller, or both
    KAFKA_CFG_PROCESS_ROLES: controller,broker
    # Specifies which nodes are controller nodes for quorum voting.
    # The syntax follows the KRaft mode (no ZooKeeper): node.id@host:port
    # The controller listener endpoint here is kafka:9093
    KAFKA_CFG_CONTROLLER_QUORUM_VOTERS: 0@kafka:9093
    # Specifies which listener is used for controller-to-controller communication
    KAFKA_CFG_CONTROLLER_LISTENER_NAMES: CONTROLLER
    # Default number of partitions for new topics
    KAFKA_NUM_PARTITIONS: 8
    # Whether to enable automatic topic creation
```

```
    KAFKA_CFG_AUTO_CREATE_TOPICS_ENABLE: "true"
    # Kafka internal listeners; Kafka supports multiple ports with different protocols
    # Each port is used for a specific purpose: INTERNAL for internal broker communication,
    # CONTROLLER for controller communication, EXTERNAL for external client connections.
    # These logical listener names are mapped to actual protocols via KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP
    # In short, Kafka is listening on three logical ports: 9092 for internal communication,
    # 9093 for controller traffic, and 9094 for external access.
    KAFKA_CFG_LISTENERS: "INTERNAL://:9092,CONTROLLER://:9093,EXTERNAL://:9094"
    # Addresses advertised to clients. INTERNAL://kafka:9092 uses the internal Docker service name 'kafka',
    # so other containers can access Kafka via kafka:9092.
    # EXTERNAL://localhost:19094 is the address external clients (e.g., in the LAN) should use to connect.
    # If Kafka is deployed on a different machine than IM, 'localhost' should be replaced with the LAN IP.
    KAFKA_CFG_ADVERTISED_LISTENERS: "INTERNAL://kafka:9092,EXTERNAL://localhost:19094"
    # Maps logical listener names to actual protocols.
    # Supported protocols include: PLAINTEXT, SSL, SASL_PLAINTEXT, SASL_SSL
    KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP: "CONTROLLER:PLAINTEXT,EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT"
    # Defines which listener is used for inter-broker communication within the Kafka cluster
    KAFKA_CFG_INTER_BROKER_LISTENER_NAME: "INTERNAL"

    # Authentication configuration variables - comment out to disable auth
    # KAFKA_USERNAME: "openIM"
    # KAFKA_PASSWORD: "openIM123"
  command: >
    /bin/sh -c '
      if [ -n "$${KAFKA_USERNAME}" ] && [ -n "$${KAFKA_PASSWORD}" ]; then
        echo "=== Kafka SASL Authentication ENABLED ==="
        echo "Username: $${KAFKA_USERNAME}"

        # Set environment variables for SASL authentication
        export KAFKA_CFG_LISTENERS="SASL_PLAINTEXT://:9092,CONTROLLER://:9093,EXTERNAL://:9094"
        export KAFKA_CFG_ADVERTISED_LISTENERS="SASL_PLAINTEXT://kafka:9092,EXTERNAL://localhost:19094"
        export KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP="CONTROLLER:PLAINTEXT,EXTERNAL:SASL_PLAINTEXT,SASL_PLAINTE
        export KAFKA_CFG_SASL_ENABLED_MECHANISMS="PLAIN"
        export KAFKA_CFG_SASL_MECHANISM_INTER_BROKER_PROTOCOL="PLAIN"
        export KAFKA_CFG_INTER_BROKER_LISTENER_NAME="SASL_PLAINTEXT"
        export KAFKA_CLIENT_USERS="$${KAFKA_USERNAME}"
        export KAFKA_CLIENT_PASSWORDS="$${KAFKA_PASSWORD}"
      fi

      # Start Kafka with the configured environment
      exec /opt/bitnami/scripts/kafka/entrypoint.sh /opt/bitnami/scripts/kafka/run.sh
    '
  networks:
    - openim

minio:
  image: "${MINIO_IMAGE}"
  ports:
    - "10005:9000"
    - "19090:9090"
  container_name: minio
  volumes:
    - "${DATA_DIR}/components/mnt/data:/data"
    - "${DATA_DIR}/components/mnt/config:/root/.minio"
  environment:
    TZ: Asia/Shanghai
    MINIO_ROOT_USER: root
    MINIO_ROOT_PASSWORD: openIM123
  restart: always
  command: minio server /data --console-address ':9090'
  networks:
    - openim

openim-web-front:
  image: ${OPENIM_WEB_FRONT_IMAGE}
  container_name: openim-web-front
```

```
  restart: always
  ports:
    - "11001:80"
  networks:
    - openim

# openim-admin-front:
#   image: ${OPENIM_ADMIN_FRONT_IMAGE}
#   container_name: openim-admin-front
#   restart: always
#   ports:
#     - "11002:80"
#   networks:
#     - openim

prometheus:
  image: ${PROMETHEUS_IMAGE}
  container_name: prometheus
  restart: always
  user: root
  profiles:
    - m
  volumes:
    - ./config/prometheus.yml:/etc/prometheus/prometheus.yml
    - ./config/instance-down-rules.yml:/etc/prometheus/instance-down-rules.yml
    - ${DATA_DIR}/components/prometheus/data:/prometheus
  command:
    - "--config.file=/etc/prometheus/prometheus.yml"
    - "--storage.tsdb.path=/prometheus"
    - "--web.listen-address=:${PROMETHEUS_PORT}"
  network_mode: host

alertmanager:
  image: ${ALERTMANAGER_IMAGE}
  container_name: alertmanager
  restart: always
  profiles:
    - m
  volumes:
    - ./config/alertmanager.yml:/etc/alertmanager/alertmanager.yml
    - ./config/email.tmpl:/etc/alertmanager/email.tmpl
  command:
    - "--config.file=/etc/alertmanager/alertmanager.yml"
    - "--web.listen-address=:${ALERTMANAGER_PORT}"
  network_mode: host

grafana:
  image: ${GRAFANA_IMAGE}
  container_name: grafana
  user: root
  restart: always
  profiles:
    - m
  environment:
    - GF_SECURITY_ALLOW_EMBEDDING=true
    - GF_SESSION_COOKIE_SAMESITE=none
    - GF_SESSION_COOKIE_SECURE=true
    - GF_AUTH_ANONYMOUS_ENABLED=true
    - GF_AUTH_ANONYMOUS_ORG_ROLE=Admin
    - GF_SERVER_HTTP_PORT=${GRAFANA_PORT}
  volumes:
    - ${DATA_DIR:-./}/components/grafana:/var/lib/grafana
  network_mode: host

node-exporter:
  image: ${NODE_EXPORTER_IMAGE}
```

```
container_name: node-exporter
restart: always
profiles:
  - m
volumes:
  - /proc:/host/proc:ro
  - /sys:/host/sys:ro
  - /:/rootfs:ro
command:
  - "--path.procfs=/host/proc"
  - "--path.sysfs=/host/sys"
  - "--path.rootfs=/rootfs"
  - "--web.listen-address=:19100"
network_mode: host
```

install.sh

```
#!/usr/bin/env bash

# Copyright © 2023 OpenIM. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# https://gist.github.com/cubxxw/28f997f2c9aff408630b072f010c1d64
#

set -e


############################### OpenIM Github ###############################
# ... rest of the script ...

# TODO
# You can configure this script in three ways.
# 1. First, set the variables in this column with more comments.
# 2. The second is to pass an environment variable via a flag such as --help.
# 3. The third way is to set the variable externally, or pass it in as an environment variable

# Default configuration for OpenIM Repo
# The OpenIM Repo settings can be customized according to your needs.

# OpenIM Repo owner, by default it's set to "OpenIMSDK". If you're using a different owner, replace accordingly.
OWNER="OpenIMSDK"

# The repository name, by default it's "Open-IM-Server". If you're using a different repository, replace accordingly
REPO="Open-IM-Server"

# Version of Go you want to use, make sure it is compatible with your OpenIM-Server requirements.
# Default is 1.18, if you want to use a different version, replace accordingly.
GO_VERSION="1.20"

# Default HTTP_PORT is 80. If you want to use a different port, uncomment and replace the value.
# HTTP_PORT=80

# CPU core number for concurrent execution. By default it's determined automatically.
# Uncomment the next line if you want to set it manually.
# CPU=$(grep -c ^processor /proc/cpuinfo)

# By default, the script uses the latest tag from OpenIM-Server releases.
# If you want to use a specific tag, uncomment and replace "v3.0.0" with the desired tag.
# LATEST_TAG=v3.0.0

# Default OpenIM install directory is /tmp. If you want to use a different directory, uncomment and replace "/test".
# DOWNLOAD_OPENIM_DIR="/test"

# GitHub proxy settings. If you are using a proxy, uncomment and replace the empty field with your proxy URL.
PROXY=

# If you have a GitHub token, replace the empty field with your token.
GITHUB_TOKEN=
```

```
# Default user is "root". If you need to modify it, uncomment and replace accordingly.
# OPENIM_USER=root

# Default password for redis, mysql, mongo, as well as accessSecret in config/config.yaml.
# Remember, it should be a combination of 8 or more numbers and letters. If you want to set a different password, un
# PASSWORD=openIM123

# Default endpoint for minio's external service IP and port. If you want to use a different endpoint, uncomment and
# ENDPOINT=http://127.0.0.1:10005

# Default API_URL, replace if necessary.
# API_URL=http://127.0.0.1:10002/object/

# Default data directory. If you want to specify a different directory, uncomment and replace "./".
# DATA_DIR=./

############################## OpenIM Functions ##############################
# Install horizon of the script
#
# Pre-requisites:
#   - git
#   - make
#   - jq
#   - docker
#   - docker-compose
#   - go
#

# Check if the script is run as root
function check_isroot() {
    if [ "$EUID" -ne 0 ]; then
    fatal "Please run the script as root or use sudo."
    fi
}

# check if the current directory is a OpenIM git repository
function check_git_repo() {
    if git rev-parse --is-inside-work-tree >/dev/null 2>&1; then
        # Inside a git repository
        for remote in $(git remote); do
            repo_url=$(git remote get-url $remote)
            if [[ $repo_url == "https://github.com/openimsdk/open-im-server.git" || \
                  $repo_url == "https://github.com/openimsdk/open-im-server" || \
                  $repo_url == "git@github.com:openimsdk/open-im-server.git" ]]; then
                # If it's OpenIMSDK repository
                info "Current directory is OpenIMSDK git repository."
                info "Executing installation directly."
                install_openim
                exit 0
            fi
            debug "Remote: $remote, URL: $repo_url"
        done
        # If it's not OpenIMSDK repository
        debug "Current directory is not OpenIMSDK git repository."
    fi
    info "Current directory is not a git repository."
}

# Function to update and install necessary tools
function install_tools() {
    info "Checking and installing necessary tools, about git, make, jq, docker, docker-compose."
    local tools=("git" "make" "jq" "docker" "docker-compose")
    local install_cmd update_cmd os

    if grep -qEi "debian|buntu|mint" /etc/os-release; then
        os="Ubuntu"
```

```
        install_cmd="sudo apt install -y"
        update_cmd="sudo apt update"
    elif grep -qEi "fedora|rhel" /etc/os-release; then
        os="CentOS"
        install_cmd="sudo yum install -y"
        update_cmd="sudo yum update"
    else
        fatal "Unsupported OS, please use Ubuntu or CentOS."
    fi

    debug "Detected OS: $os"
    info "Updating system package repositories..."
    $update_cmd

    for tool in "${tools[@]}"; do
        if ! command -v $tool &> /dev/null; then
            warn "$tool is not installed. Installing now..."
            $install_cmd $tool
            success "$tool has been installed successfully."
        else
            info "$tool is already installed."
        fi
    done
}

# Function to check if Docker and Docker Compose are installed
function check_docker() {
    if ! command -v docker &> /dev/null; then
        fatal "Docker is not installed. Please install Docker first."
    fi
    if ! command -v docker-compose &> /dev/null; then
        fatal "Docker Compose is not installed. Please install Docker Compose first."
    fi
}

# Function to download and install Go if it's not already installed
function install_go() {
    command -v go >/dev/null 2>&1
    # Determines if GO_VERSION is defined
    if [ -z "$GO_VERSION" ]; then
        GO_VERSION="1.20"
    fi

    if [[ $? -ne 0 ]]; then
        warn "Go is not installed. Installing now..."
        curl -LO "https://golang.org/dl/go${GO_VERSION}.linux-amd64.tar.gz"
        if [ $? -ne 0 ]; then
            fatal "Download failed! Please check your network connectivity."
        fi
        sudo tar -C /usr/local -xzf "go${GO_VERSION}.linux-amd64.tar.gz"
        echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bashrc
        source ~/.bashrc
        success "Go has been installed successfully."
    else
        info "Go is already installed."
    fi
}

function download_source_code() {

    # If LATEST_TAG was not defined outside the function, get it here example: v3.0.1-beta.1
    if [ -z "$LATEST_TAG" ]; then
        LATEST_TAG=$(curl -s "https://api.github.com/repos/$OWNER/$REPO/tags" | jq -r '.[0].name')
    fi

    # If LATEST_TAG is still empty, set a default value
```

```
    local DEFAULT_TAG="v3.0.0"

    LATEST_TAG="${LATEST_TAG:-$DEFAULT_TAG}"

    debug "DEFAULT_TAG: $DEFAULT_TAG"
    info "Use OpenIM Version LATEST_TAG: $LATEST_TAG"

    # If MODIFIED_TAG was not defined outside the function, modify it here,example: 3.0.1-beta.1
    if [ -z "$MODIFIED_TAG" ]; then
        MODIFIED_TAG=$(echo $LATEST_TAG | sed 's/v//')
    fi

    # If MODIFIED_TAG is still empty, set a default value
    local DEFAULT_MODIFIED_TAG="${DEFAULT_TAG#v}"
    MODIFIED_TAG="${MODIFIED_TAG:-$DEFAULT_MODIFIED_TAG}"

    debug "MODIFIED_TAG: $MODIFIED_TAG"

    # Construct the tarball URL
    TARBALL_URL="${PROXY}https://github.com/$OWNER/$REPO/archive/refs/tags/$LATEST_TAG.tar.gz"

    info "Downloaded OpenIM TARBALL_URL: $TARBALL_URL"

    info "Starting the OpenIM automated one-click deployment script."

    # Set the download and extract directory to /tmp
    if [ -z "$DOWNLOAD_OPENIM_DIR" ]; then
        DOWNLOAD_OPENIM_DIR="/tmp"
    fi

    # Check if /tmp directory exists
    if [ ! -d "$DOWNLOAD_OPENIM_DIR" ]; then
        warn "$DOWNLOAD_OPENIM_DIR does not exist. Creating it..."
        mkdir -p "$DOWNLOAD_OPENIM_DIR"
    fi

    info "Downloading OpenIM source code from $TARBALL_URL to $DOWNLOAD_OPENIM_DIR"

    curl -L -o "${DOWNLOAD_OPENIM_DIR}/${MODIFIED_TAG}.tar.gz" $TARBALL_URL

    tar -xzvf "${DOWNLOAD_OPENIM_DIR}/${MODIFIED_TAG}.tar.gz" -C "$DOWNLOAD_OPENIM_DIR"
    cd "$DOWNLOAD_OPENIM_DIR/$REPO-$MODIFIED_TAG"
    git init && git add . && git commit -m "init"  --no-verify

    success "Source code downloaded and extracted to $REPO-$MODIFIED_TAG"
}

function set_openim_env() {
    warn "This command can only be executed once. It will modify the component passwords in docker-compose based on
    # Set default values for user input
    # If the OPENIM_USER environment variable is not set, it defaults to 'root'
    if [ -z "$OPENIM_USER" ]; then
        OPENIM_USER="root"
        debug "OPENIM_USER is not set. Defaulting to 'root'."
    fi

    # If the PASSWORD environment variable is not set, it defaults to 'openIM123'
    # This password applies to redis, mysql, mongo, as well as accessSecret in config/config.yaml
    if [ -z "$PASSWORD" ]; then
        PASSWORD="openIM123"
        debug "PASSWORD is not set. Defaulting to 'openIM123'."
    fi

    # If the ENDPOINT environment variable is not set, it defaults to 'http://127.0.0.1:10005'
    # This is minio's external service IP and port, or it could be a domain like storage.xx.xx
    # The app must be able to access this IP and port or domain
```

```
    if [ -z "$ENDPOINT" ]; then
        ENDPOINT="http://127.0.0.1:10005"
        debug "ENDPOINT is not set. Defaulting to 'http://127.0.0.1:10005'."
    fi

    # If the API_URL environment variable is not set, it defaults to 'http://127.0.0.1:10002/object/'
    # The app must be able to access this IP and port or domain
    if [ -z "$API_URL" ]; then
        API_URL="http://127.0.0.1:10002/object/"
        debug "API_URL is not set. Defaulting to 'http://127.0.0.1:10002/object/'."
    fi

    # If the DATA_DIR environment variable is not set, it defaults to the current directory './'
    # This can be set to a directory with large disk space
    if [ -z "$DATA_DIR" ]; then
        DATA_DIR="./"
        debug "DATA_DIR is not set. Defaulting to './'."
    fi
}

function install_openim() {
    info "Installing OpenIM"
    make -j${CPU} install V=1

    info "Checking installation"
    make check

    success "OpenIM installation completed successfully. Happy chatting!"
}

############################### OpenIM Help ###############################

# Function to display help message
function cmd_help() {
    openim_color
    color_echo ${BRIGHT_GREEN_PREFIX} "Usage: $0 [options]"
    color_echo ${BRIGHT_GREEN_PREFIX} "Options:"
    echo
    color_echo ${BLUE_PREFIX} "-i,  --install       ${CYAN_PREFIX}Execute the installation logic of the script${COLO
    color_echo ${BLUE_PREFIX} "-u,  --user          ${CYAN_PREFIX}set user (default: root)${COLOR_SUFFIX}"
    color_echo ${BLUE_PREFIX} "-p,  --password      ${CYAN_PREFIX}set password (default: openIM123)${COLOR_SUFFIX}"
    color_echo ${BLUE_PREFIX} "-e,  --endpoint      ${CYAN_PREFIX}set endpoint (default: http://127.0.0.1:10005)${CO
    color_echo ${BLUE_PREFIX} "-a,  --api           ${CYAN_PREFIX}set API URL (default: http://127.0.0.1:10002/objec
    color_echo ${BLUE_PREFIX} "-d,  --directory     ${CYAN_PREFIX}set directory for large disk space (default: ./)${
    color_echo ${BLUE_PREFIX} "-h,  --help          ${CYAN_PREFIX}display this help message and exit${COLOR_SUFFIX}"
    color_echo ${BLUE_PREFIX} "-cn, --china         ${CYAN_PREFIX}set to use the Chinese domestic proxy${COLOR_SUFFI
    color_echo ${BLUE_PREFIX} "-t,  --tag           ${CYAN_PREFIX}specify the tag (default option, set to latest if
    color_echo ${BLUE_PREFIX} "-r,  --release       ${CYAN_PREFIX}specify the release branch (cannot be used with th
    color_echo ${BLUE_PREFIX} "-gt, --github-token  ${CYAN_PREFIX}set the GITHUB_TOKEN (default: not set)${COLOR_SUF
    color_echo ${BLUE_PREFIX} "-g,  --go-version    ${CYAN_PREFIX}set the Go language version (default: GO_VERSION=\
    color_echo ${BLUE_PREFIX} "--install-dir        ${CYAN_PREFIX}set the OpenIM installation directory (default: /t
    color_echo ${BLUE_PREFIX} "--cpu                ${CYAN_PREFIX}set the number of concurrent processes${COLOR_SUFF
    echo
    color_echo ${RED_PREFIX} "Note: Only one of the -t/--tag or -r/--release options can be used at a time.${COLOR_S
    color_echo ${RED_PREFIX} "If both are used or none of them are used, the -t/--tag option will be prioritized.${C
    echo
    exit 1
}

function parseinput() {
    # set default values
    # OPENIM_USER=root
    # PASSWORD=openIM123
    # ENDPOINT=http://127.0.0.1:10005
    # API=http://127.0.0.1:10002/object/
    # DIRECTORY=./
```

```
# CHINA=false
# TAG=latest
# RELEASE=""
# GO_VERSION=1.20
# INSTALL_DIR=/tmp
# GITHUB_TOKEN=""
# CPU=$(nproc)

if [ $# -eq 0 ]; then
    cmd_help
    exit 1
fi

while [ $# -gt 0 ]; do
    case $1 in
        -h|--help)
            cmd_help
            exit
            ;;
        -u|--user)
            shift
            OPENIM_USER=$1
            ;;
        -p|--password)
            shift
            PASSWORD=$1
            ;;
        -e|--endpoint)
            shift
            ENDPOINT=$1
            ;;
        -a|--api)
            shift
            API=$1
            ;;
        -d|--directory)
            shift
            DIRECTORY=$1
            ;;
        -cn|--china)
            CHINA=true
            ;;
        -t|--tag)
            shift
            TAG=$1
            ;;
        -r|--release)
            shift
            RELEASE=$1
            ;;
        -g|--go-version)
            shift
            GO_VERSION=$1
            ;;
        --install-dir)
            shift
            INSTALL_DIR=$1
            ;;
        -gt|--github-token)
            shift
            GITHUB_TOKEN=$1
            ;;
        --cpu)
            shift
            CPU=$1
            ;;
```

```
            -i|--install)
                openim_main
                exit
                ;;
            *)
                echo "Unknown option: $1"
                cmd_help
                exit 1
                ;;
        esac
        shift
    done
}

############################## OpenIM LOG ###############################
# Set text color to cyan for header and URL
print_with_delay() {
  text="$1"
  delay="$2"

  for i in $(seq 0 $((${#text}-1))); do
    printf "${text:$i:1}"
    sleep $delay
  done
  printf "\n"
}

print_progress() {
  total="$1"
  delay="$2"

  printf "["
  for i in $(seq 1 $total); do
    printf "#"
    sleep $delay
  done
  printf "]\n"
}

# Function for colored echo
color_echo() {
    COLOR=$1
    shift
    echo -e "${COLOR} $* ${COLOR_SUFFIX}"
}

# Color definitions
function openim_color() {
    COLOR_SUFFIX="\033[0m"      # End all colors and special effects

    BLACK_PREFIX="\033[30m"     # Black prefix
    RED_PREFIX="\033[31m"       # Red prefix
    GREEN_PREFIX="\033[32m"     # Green prefix
    YELLOW_PREFIX="\033[33m"    # Yellow prefix
    BLUE_PREFIX="\033[34m"      # Blue prefix
    SKY_BLUE_PREFIX="\033[36m"  # Sky blue prefix
    WHITE_PREFIX="\033[37m"     # White prefix
    BOLD_PREFIX="\033[1m"       # Bold prefix
    UNDERLINE_PREFIX="\033[4m"  # Underline prefix
    ITALIC_PREFIX="\033[3m"     # Italic prefix
    BRIGHT_GREEN_PREFIX='\033[1;32m' # Bright green prefix

    CYAN_PREFIX="\033[0;36m"     # Cyan prefix
}

# --- helper functions for logs ---
```

```
info() {
    echo -e "[${GREEN_PREFIX}INFO${COLOR_SUFFIX}] " "$@"
}
warn() {
    echo -e "[${YELLOW_PREFIX}WARN${COLOR_SUFFIX}] " "$@" >&2
}
fatal() {
    echo -e "[${RED_PREFIX}ERROR${COLOR_SUFFIX}] " "$@" >&2
    exit 1
}
debug() {
    echo -e "[${BLUE_PREFIX}DEBUG${COLOR_SUFFIX}]===> " "$@"
}
success() {
    echo -e "${BRIGHT_GREEN_PREFIX}=== [SUCCESS] ===${COLOR_SUFFIX}\n=> " "$@"
}

function openim_logo() {
    # Set text color to cyan for header and URL
    echo -e "\033[0;36m"

    # Display fancy ASCII Art logo
    # look http://patorjk.com/software/taag/#p=display&h=1&v=1&f=Doh&t=OpenIM
    print_with_delay '


     OOOOOOOOO                                                                  IIIIIIIIIIMMMMMMMM              MMMMMM
   OO:::::::::OO                                                                I::::::::IM::::::::M            M::::::
 OO:::::::::::::OO                                                              I::::::::IM::::::::M          M::::::
O:::::::OOO:::::::O                                                             II::::::IIM::::::::::M         M::::::
O::::::O   O::::::Opppppp   ppppppppp       eeeeeeeeeeee    nnnn  nnnnnnnn      I::::I  M::::::::::M       M::::::
O:::::O     O:::::Op::::ppp:::::::::p    ee::::::::::::ee  n::::nn::::::::nn    I::::I  M::::::::::M     M::::::
O:::::O     O:::::Op:::::::::::::::::p  e::::::eeeee:::::een:::::::::::::::nn   I::::I  M::::::M:::M   M:::M::::::
O:::::O     O:::::Opp::::::ppppp::::::pe::::::e     e:::::enn:::::::::::::::n  I::::I  M::::::M M:::M M:::M M::::::
O:::::O     O:::::O p:::::p     p:::::pe:::::::eeeee:::::e  n::::::nnnn:::::n  I::::I  M::::::M  M:::M:::M  M::::::
O:::::O     O:::::O p:::::p     p:::::pe:::::::::::::::::e   n::::n    n::::n  I::::I  M::::::M   M:::::M   M::::::
O:::::O     O:::::O p:::::p     p:::::pe::::::eeeeeeeeeee    n::::n    n::::n  I::::I  M::::::M    M:::::M    M::::::
O::::::O   O::::::O p:::::p    p::::::pe:::::::e             n::::n    n::::n  I::::I  M::::::M     MMMMM     M::::::
O:::::::OOO:::::::O p:::::pppppp:::::::pe::::::::e            n::::n    n::::nII::::::IIM::::::M               M::::::
 OO:::::::::::::OO  p::::::::::::::::p  e::::::::eeeeeeee    n::::n    n::::nI::::::::IM::::::M               M::::::
   OO:::::::::OO    p::::::::::::::pp    ee::::::::::::e    n::::n    n::::nI::::::::IM::::::M               M::::::
     OOOOOOOOO      p::::::pppppppp        eeeeeeeeeeeeee    nnnnnn    nnnnnnIIIIIIIIIIMMMMMMMM               MMMMMM
                    p:::::p
                    p:::::p
                   p:::::::p
                   p:::::::p
                   p:::::::p
                   pppppppp

    ' 0.0001

    # Display product URL
    print_with_delay "Discover more and contribute at: https://github.com/openimsdk/open-im-server" 0.01

    # Reset text color back to normal
    echo -e "\033[0m"

    # Set text color to green for product description
    echo -e "\033[1;32m"

    print_with_delay "Open-IM-Server: Reinventing Instant Messaging" 0.01
    print_progress 50 0.02

    print_with_delay "Open-IM-Server is not just a product; it's a revolution. It's about bringing the power of seam

    print_progress 50 0.02
```

```
    # Reset text color back to normal
    echo -e "\033[0m"

    # Set text color to yellow for the Slack link
    echo -e "\033[1;33m"

    print_with_delay "Join our developer community on Slack: https://join.slack.com/t/openimsdk/shared_invite/zt-2ij

    # Reset text color back to normal
    echo -e "\033[0m"
}

# Main function to run the script
function openim_main() {
    check_git_repo
    check_isroot
    openim_color
    install_tools
    check_docker
    install_go
    download_source_code
    set_openim_env
    install_openim
    openim_logo

}

parseinput "$@"
```

magefile.go

```
//go:build mage
// +build mage

package main

import (
	"flag"
	"os"

	"github.com/openimsdk/gomake/mageutil"
)

var Default = Build

var Aliases = map[string]any{
	"buildcc": BuildWithCustomConfig,
	"startcc": StartWithCustomConfig,
}

var (
	customRootDir   = "."       // workDir in mage, default is "./"(project root directory)
	customSrcDir    = "cmd"     // source code directory, default is "cmd"
	customOutputDir = "_output" // output directory, default is "_output"
	customConfigDir = "config"  // configuration directory, default is "config"
	customToolsDir  = "tools"   // tools source code directory, default is "tools"
)


// Build support specifical binary build.
//
// Example: `mage build openim-api openim-rpc-user seq`
func Build() {
	flag.Parse()
	bin := flag.Args()
	if len(bin) != 0 {
		bin = bin[1:]
	}

	mageutil.Build(bin, nil)
}

func BuildWithCustomConfig() {
	flag.Parse()
	bin := flag.Args()
	if len(bin) != 0 {
		bin = bin[1:]
	}

	config := &mageutil.PathOptions{
		RootDir:   &customRootDir,
		OutputDir: &customOutputDir,
		SrcDir:    &customSrcDir,
		ToolsDir:  &customToolsDir,
	}

	mageutil.Build(bin, config)
}

func Start() {
	mageutil.InitForSSC()
	err := setMaxOpenFiles()
	if err != nil {
		mageutil.PrintRed("setMaxOpenFiles failed " + err.Error())
		os.Exit(1)
```

```
    }

    flag.Parse()
    bin := flag.Args()
    if len(bin) != 0 {
        bin = bin[1:]
    }

    mageutil.StartToolsAndServices(bin, nil)
}

func StartWithCustomConfig() {
    mageutil.InitForSSC()
    err := setMaxOpenFiles()
    if err != nil {
        mageutil.PrintRed("setMaxOpenFiles failed " + err.Error())
        os.Exit(1)
    }

    flag.Parse()
    bin := flag.Args()
    if len(bin) != 0 {
        bin = bin[1:]
    }

    config := &mageutil.PathOptions{
        RootDir:   &customRootDir,
        OutputDir: &customOutputDir,
        ConfigDir: &customConfigDir,
    }

    mageutil.StartToolsAndServices(bin, config)
}

func Stop() {
    mageutil.StopAndCheckBinaries()
}

func Check() {
    mageutil.CheckAndReportBinariesStatus()
}
```

magefile_unix.go

```
//go:build mage && !windows
// +build mage,!windows

package main

import (
	"syscall"

	"github.com/openimsdk/gomake/mageutil"
)

func setMaxOpenFiles() error {
	var rLimit syscall.Rlimit
	err := syscall.Getrlimit(syscall.RLIMIT_NOFILE, &rLimit)
	if err != nil {
		return err
	}
	rLimit.Max = uint64(mageutil.MaxFileDescriptors)
	rLimit.Cur = uint64(mageutil.MaxFileDescriptors)
	return syscall.Setrlimit(syscall.RLIMIT_NOFILE, &rLimit)
}
```

magefile_windows.go

```
//go:build mage
// +build mage

package main

func setMaxOpenFiles() error {
	return nil
}
```

start-config.yml

```yaml
serviceBinaries:
  openim-api: 1
  openim-crontask: 4
  openim-rpc-user: 1
  openim-msggateway: 1
  openim-push: 8
  openim-msgtransfer: 8
  openim-rpc-conversation: 1
  openim-rpc-auth: 1
  openim-rpc-group: 1
  openim-rpc-friend: 1
  openim-rpc-msg: 1
  openim-rpc-third: 1
toolBinaries:
  - check-free-memory
  - check-component
  - seq
maxFileDescriptors: 10000
```

pkg

pkg/apistruct

pkg/apistruct/config_manager.go

```go
package apistruct

type GetConfigReq struct {
	ConfigName string `json:"configName"`
}

type GetConfigListResp struct {
	Environment string   `json:"environment"`
	Version     string   `json:"version"`
	ConfigNames []string `json:"configNames"`
}

type SetConfigReq struct {
	ConfigName string `json:"configName"`
	Data       string `json:"data"`
}

type SetConfigsReq struct {
	Configs []SetConfigReq `json:"configs"`
}

type SetEnableConfigManagerReq struct {
	Enable bool `json:"enable"`
}

type GetEnableConfigManagerResp struct {
	Enable bool `json:"enable"`
}
```

pkg/apistruct/doc.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package apistruct // import "github.com/openimsdk/open-im-server/v3/pkg/apistruct"
```

pkg/apistruct/manage.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package apistruct

import (
	pbmsg "github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
)

// SendMsg defines the structure for sending messages with various metadata.
type SendMsg struct {
	// SendID uniquely identifies the sender.
	SendID string `json:"sendID" binding:"required"`

	// GroupID is the identifier for the group, required if SessionType is 2 or 3.
	GroupID string `json:"groupID" binding:"required_if=SessionType 2|required_if=SessionType 3"`

	// SenderNickname is the nickname of the sender.
	SenderNickname string `json:"senderNickname"`

	// SenderFaceURL is the URL to the sender's avatar.
	SenderFaceURL string `json:"senderFaceURL"`

	// SenderPlatformID is an integer identifier for the sender's platform.
	SenderPlatformID int32 `json:"senderPlatformID"`

	// Content is the actual content of the message, required and excluded from Swagger documentation.
	Content map[string]any `json:"content" binding:"required" swaggerignore:"true"`

	// ContentType is an integer that represents the type of the content.
	ContentType int32 `json:"contentType" binding:"required"`

	// SessionType is an integer that represents the type of session for the message.
	SessionType int32 `json:"sessionType" binding:"required"`

	// IsOnlineOnly specifies if the message is only sent when the receiver is online.
	IsOnlineOnly bool `json:"isOnlineOnly"`

	// NotOfflinePush specifies if the message should not trigger offline push notifications.
	NotOfflinePush bool `json:"notOfflinePush"`

	// SendTime is a timestamp indicating when the message was sent.
	SendTime int64 `json:"sendTime"`

	// OfflinePushInfo contains information for offline push notifications.
	OfflinePushInfo *sdkws.OfflinePushInfo `json:"offlinePushInfo"`

	// Ex stores extended fields
	Ex string `json:"ex"`
}

// SendMsgReq extends SendMsg with the requirement of RecvID when SessionType indicates a one-on-one or notification
```

```go
type SendMsgReq struct {
	// RecvID uniquely identifies the receiver and is required for one-on-one or notification chat types.
	RecvID string `json:"recvID" binding:"required_if" message:"recvID is required if sessionType is SingleChatType or
	SendMsg
}

type GetConversationListReq struct {
	// userID uniquely identifies the user.
	UserID string `protobuf:"bytes,1,opt,name=userID,proto3" json:"userID,omitempty" binding:"required"`

	// ConversationIDs contains a list of unique identifiers for conversations.
	ConversationIDs []string `protobuf:"bytes,2,rep,name=conversationIDs,proto3" json:"conversationIDs,omitempty"`
}

type GetConversationListResp struct {
	// ConversationElems is a map that associates conversation IDs with their respective details.
	ConversationElems map[string]*ConversationElem `protobuf:"bytes,1,rep,name=conversationElems,proto3" json:"conversa
}

type ConversationElem struct {
	// MaxSeq represents the maximum sequence number within the conversation.
	MaxSeq int64 `protobuf:"varint,1,opt,name=maxSeq,proto3" json:"maxSeq,omitempty"`

	// UnreadSeq represents the number of unread messages in the conversation.
	UnreadSeq int64 `protobuf:"varint,2,opt,name=unreadSeq,proto3" json:"unreadSeq,omitempty"`

	// LastSeqTime represents the timestamp of the last sequence in the conversation.
	LastSeqTime int64 `protobuf:"varint,3,opt,name=LastSeqTime,proto3" json:"LastSeqTime,omitempty"`
}

// BatchSendMsgReq defines the structure for sending a message to multiple recipients.
type BatchSendMsgReq struct {
	SendMsg

	// IsSendAll indicates whether the message should be sent to all users.
	IsSendAll bool `json:"isSendAll"`

	// RecvIDs is a slice of receiver identifiers to whom the message will be sent, required field.
	RecvIDs []string `json:"recvIDs" binding:"required"`
}

// BatchSendMsgResp contains the results of a batch message send operation.
type BatchSendMsgResp struct {
	// Results is a slice of SingleReturnResult, representing the outcome of each message sent.
	Results []*SingleReturnResult `json:"results"`

	// FailedIDs is a slice of user IDs for whom the message send failed.
	FailedIDs []string `json:"failedUserIDs"`
}

// SendSingleMsgReq defines the structure for sending a message to multiple recipients.
type SendSingleMsgReq struct {
	// groupMsg should appoint sendID
	SendID          string                 `json:"sendID"`
	Content         string                 `json:"content" binding:"required"`
	OfflinePushInfo *sdkws.OfflinePushInfo `json:"offlinePushInfo"`
	Ex              string                 `json:"ex"`
}

type KeyMsgData struct {
	SendID  string `json:"sendID"`
	RecvID  string `json:"recvID"`
	GroupID string `json:"groupID"`
}

// SingleReturnResult encapsulates the result of a single message send attempt.
```

```go
type SingleReturnResult struct {
    // ServerMsgID is the message identifier on the server-side.
    ServerMsgID string `json:"serverMsgID"`

    // ClientMsgID is the message identifier on the client-side.
    ClientMsgID string `json:"clientMsgID"`

    // SendTime is the timestamp of when the message was sent.
    SendTime int64 `json:"sendTime"`

    // RecvID uniquely identifies the receiver of the message.
    RecvID string `json:"recvID"`

    // Modify fields modified via webhook.
    Modify map[string]any `json:"modify,omitempty"`
}

type SendMsgResp struct {
    // SendMsgResp original response.
    *pbmsg.SendMsgResp

    // Modify fields modified via webhook.
    Modify map[string]any `json:"modify,omitempty"`
}
```

pkg/apistruct/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package apistruct

import "github.com/openimsdk/protocol/sdkws"

type PictureBaseInfo struct {
    UUID   string `mapstructure:"uuid"`
    Type   string `mapstructure:"type"   validate:"required"`
    Size   int64  `mapstructure:"size"`
    Width  int32  `mapstructure:"width"  validate:"required"`
    Height int32  `mapstructure:"height" validate:"required"`
    Url    string `mapstructure:"url"    validate:"required"`
}

type PictureElem struct {
    SourcePath      string          `mapstructure:"sourcePath"`
    SourcePicture   PictureBaseInfo `mapstructure:"sourcePicture"   validate:"required"`
    BigPicture      PictureBaseInfo `mapstructure:"bigPicture"      validate:"required"`
    SnapshotPicture PictureBaseInfo `mapstructure:"snapshotPicture" validate:"required"`
}

type SoundElem struct {
    UUID      string `mapstructure:"uuid"`
    SoundPath string `mapstructure:"soundPath"`
    SourceURL string `mapstructure:"sourceUrl" validate:"required"`
    DataSize  int64  `mapstructure:"dataSize"`
    Duration  int64  `mapstructure:"duration"  validate:"required,min=1"`
}

type VideoElem struct {
    VideoPath      string `mapstructure:"videoPath"`
    VideoUUID      string `mapstructure:"videoUUID"`
    VideoURL       string `mapstructure:"videoUrl"       validate:"required"`
    VideoType      string `mapstructure:"videoType"      validate:"required"`
    VideoSize      int64  `mapstructure:"videoSize"      validate:"required"`
    Duration       int64  `mapstructure:"duration"       validate:"required"`
    SnapshotPath   string `mapstructure:"snapshotPath"`
    SnapshotUUID   string `mapstructure:"snapshotUUID"`
    SnapshotSize   int64  `mapstructure:"snapshotSize"`
    SnapshotURL    string `mapstructure:"snapshotUrl"    validate:"required"`
    SnapshotWidth  int32  `mapstructure:"snapshotWidth"  validate:"required"`
    SnapshotHeight int32  `mapstructure:"snapshotHeight" validate:"required"`
}

type FileElem struct {
    FilePath  string `mapstructure:"filePath"`
    UUID      string `mapstructure:"uuid"`
    SourceURL string `mapstructure:"sourceUrl" validate:"required"`
    FileName  string `mapstructure:"fileName"  validate:"required"`
    FileSize  int64  `mapstructure:"fileSize"  validate:"required"`
}
```

```
type AtElem struct {
    Text         string     `mapstructure:"text"`
    AtUserList   []string   `mapstructure:"atUserList" validate:"required,max=1000"`
    AtUsersInfo  []*AtInfo  `json:"atUsersInfo"`
    QuoteMessage *MsgStruct `json:"quoteMessage"`
    IsAtSelf     bool       `mapstructure:"isAtSelf"`
}
type LocationElem struct {
    Description string  `mapstructure:"description"`
    Longitude   float64 `mapstructure:"longitude"   validate:"required"`
    Latitude    float64 `mapstructure:"latitude"    validate:"required"`
}

type CustomElem struct {
    Data        string `mapstructure:"data"        validate:"required"`
    Description string `mapstructure:"description"`
    Extension   string `mapstructure:"extension"`
}

type TextElem struct {
    Content string `json:"content" validate:"required"`
}

type MarkdownTextElem struct {
    Content string `mapstructure:"content" validate:"required"`
}

type StreamMsgElem struct {
    Type    string `mapstructure:"type" validate:"required"`
    Content string `mapstructure:"content" validate:"required"`
}

type RevokeElem struct {
    RevokeMsgClientID string `mapstructure:"revokeMsgClientID" validate:"required"`
}

type QuoteElem struct {
    Text         string     `json:"text,omitempty"`
    QuoteMessage *MsgStruct `json:"quoteMessage,omitempty"`
}

type OANotificationElem struct {
    NotificationName    string       `mapstructure:"notificationName"    json:"notificationName"    validate:"required"
    NotificationFaceURL string       `mapstructure:"notificationFaceURL" json:"notificationFaceURL"`
    NotificationType    int32        `mapstructure:"notificationType"    json:"notificationType"    validate:"required"
    Text                string       `mapstructure:"text"                json:"text"                validate:"required"
    Url                 string       `mapstructure:"url"                 json:"url"`
    MixType             int32        `mapstructure:"mixType"             json:"mixType"             validate:"gte=0,lte
    PictureElem         *PictureElem `mapstructure:"pictureElem"         json:"pictureElem"`
    SoundElem           *SoundElem   `mapstructure:"soundElem"           json:"soundElem"`
    VideoElem           *VideoElem   `mapstructure:"videoElem"           json:"videoElem"`
    FileElem            *FileElem    `mapstructure:"fileElem"            json:"fileElem"`
    Ex                  string       `mapstructure:"ex"                  json:"ex"`
}

type MessageRevoked struct {
    RevokerID       string `mapstructure:"revokerID"       json:"revokerID"       validate:"required"`
    RevokerRole     int32  `mapstructure:"revokerRole"     json:"revokerRole"     validate:"required"`
    ClientMsgID     string `mapstructure:"clientMsgID"     json:"clientMsgID"     validate:"required"`
    RevokerNickname string `mapstructure:"revokerNickname" json:"revokerNickname"`
    SessionType     int32  `mapstructure:"sessionType"     json:"sessionType"     validate:"required"`
    Seq             uint32 `mapstructure:"seq"             json:"seq"             validate:"required"`
}

type MsgStruct struct {
    ClientMsgID          string                `json:"clientMsgID,omitempty"`
```

```
	ServerMsgID          string                `json:"serverMsgID,omitempty"`
	CreateTime           int64                 `json:"createTime"`
	SendTime             int64                 `json:"sendTime"`
	SessionType          int32                 `json:"sessionType"`
	SendID               string                `json:"sendID,omitempty"`
	RecvID               string                `json:"recvID,omitempty"`
	MsgFrom              int32                 `json:"msgFrom"`
	ContentType          int32                 `json:"contentType"`
	SenderPlatformID     int32                 `json:"senderPlatformID"`
	SenderNickname       string                `json:"senderNickname,omitempty"`
	SenderFaceURL        string                `json:"senderFaceUrl,omitempty"`
	GroupID              string                `json:"groupID,omitempty"`
	Content              string                `json:"content,omitempty"`
	Seq                  int64                 `json:"seq"`
	IsRead               bool                  `json:"isRead"`
	Status               int32                 `json:"status"`
	IsReact              bool                  `json:"isReact,omitempty"`
	IsExternalExtensions bool                  `json:"isExternalExtensions,omitempty"`
	OfflinePush          *sdkws.OfflinePushInfo `json:"offlinePush,omitempty"`
	AttachedInfo         string                `json:"attachedInfo,omitempty"`
	Ex                   string                `json:"ex,omitempty"`
	LocalEx              string                `json:"localEx,omitempty"`
	TextElem             *TextElem             `json:"textElem,omitempty"`
	PictureElem          *PictureElem          `json:"pictureElem,omitempty"`
	SoundElem            *SoundElem            `json:"soundElem,omitempty"`
	VideoElem            *VideoElem            `json:"videoElem,omitempty"`
	FileElem             *FileElem             `json:"fileElem,omitempty"`
	AtTextElem           *AtElem               `json:"atTextElem,omitempty"`
	LocationElem         *LocationElem         `json:"locationElem,omitempty"`
	CustomElem           *CustomElem           `json:"customElem,omitempty"`
	QuoteElem            *QuoteElem            `json:"quoteElem,omitempty"`
}

type AtInfo struct {
	AtUserID      string `json:"atUserID,omitempty"`
	GroupNickname string `json:"groupNickname,omitempty"`
}
```

pkg/apistruct/public.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package apistruct

type GroupAddMemberInfo struct {
    UserID    string `json:"userID"    binding:"required"`
    RoleLevel int32  `json:"roleLevel" binding:"required,oneof= 1 3"`
}
```

pkg/callbackstruct

pkg/callbackstruct/common.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

import (
"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
"github.com/openimsdk/tools/errs"
)

const (
Next = 1
)

type CommonCallbackReq struct {
SendID           string   `json:"sendID"`
CallbackCommand  string   `json:"callbackCommand"`
ServerMsgID      string   `json:"serverMsgID"`
ClientMsgID      string   `json:"clientMsgID"`
OperationID      string   `json:"operationID"`
SenderPlatformID int32    `json:"senderPlatformID"`
SenderNickname   string   `json:"senderNickname"`
SessionType      int32    `json:"sessionType"`
MsgFrom          int32    `json:"msgFrom"`
ContentType      int32    `json:"contentType"`
Status           int32    `json:"status"`
SendTime         int64    `json:"sendTime"`
CreateTime       int64    `json:"createTime"`
Content          string   `json:"content"`
Seq              uint32   `json:"seq"`
AtUserIDList     []string `json:"atUserList"`
SenderFaceURL    string   `json:"faceURL"`
Ex               string   `json:"ex"`
}

func (c *CommonCallbackReq) GetCallbackCommand() string {
return c.CallbackCommand
}

type CallbackReq interface {
GetCallbackCommand() string
}

type CallbackResp interface {
Parse() (err error)
}

type CommonCallbackResp struct {
ActionCode int32  `json:"actionCode"`
ErrCode    int32  `json:"errCode"`
```

```
	ErrMsg     string `json:"errMsg"`
	ErrDlt     string `json:"errDlt"`
	NextCode   int32  `json:"nextCode"`
}

func (c CommonCallbackResp) Parse() error {
	if c.ActionCode == servererrs.NoError && c.NextCode == Next {
		return errs.NewCodeError(int(c.ErrCode), c.ErrMsg).WithDetail(c.ErrDlt)
	}
	return nil
}

type UserStatusBaseCallback struct {
	CallbackCommand string `json:"callbackCommand"`
	OperationID     string `json:"operationID"`
	PlatformID      int    `json:"platformID"`
	Platform        string `json:"platform"`
}

func (c UserStatusBaseCallback) GetCallbackCommand() string {
	return c.CallbackCommand
}

type UserStatusCallbackReq struct {
	UserStatusBaseCallback
	UserID string `json:"userID"`
}

type UserStatusBatchCallbackReq struct {
	UserStatusBaseCallback
	UserIDList []string `json:"userIDList"`
}
```

pkg/callbackstruct/constant.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

const (
    CallbackBeforeInviteJoinGroupCommand          = "callbackBeforeInviteJoinGroupCommand"
    CallbackAfterJoinGroupCommand                 = "callbackAfterJoinGroupCommand"
    CallbackAfterSetGroupInfoCommand              = "callbackAfterSetGroupInfoCommand"
    CallbackAfterSetGroupInfoExCommand            = "callbackAfterSetGroupInfoExCommand"
    CallbackBeforeSetGroupInfoCommand             = "callbackBeforeSetGroupInfoCommand"
    CallbackBeforeSetGroupInfoExCommand           = "callbackBeforeSetGroupInfoExCommand"
    CallbackAfterRevokeMsgCommand                 = "callbackBeforeAfterMsgCommand"
    CallbackBeforeAddBlackCommand                 = "callbackBeforeAddBlackCommand"
    CallbackAfterAddFriendCommand                 = "callbackAfterAddFriendCommand"
    CallbackBeforeAddFriendAgreeCommand           = "callbackBeforeAddFriendAgreeCommand"
    CallbackAfterAddFriendAgreeCommand            = "callbackAfterAddFriendAgreeCommand"
    CallbackAfterDeleteFriendCommand              = "callbackAfterDeleteFriendCommand"
    CallbackBeforeImportFriendsCommand            = "callbackBeforeImportFriendsCommand"
    CallbackAfterImportFriendsCommand             = "callbackAfterImportFriendsCommand"
    CallbackAfterRemoveBlackCommand               = "callbackAfterRemoveBlackCommand"
    CallbackAfterQuitGroupCommand                 = "callbackAfterQuitGroupCommand"
    CallbackAfterKickGroupCommand                 = "callbackAfterKickGroupCommand"
    CallbackAfterDisMissGroupCommand              = "callbackAfterDisMissGroupCommand"
    CallbackBeforeJoinGroupCommand                = "callbackBeforeJoinGroupCommand"
    CallbackAfterGroupMsgReadCommand              = "callbackAfterGroupMsgReadCommand"
    CallbackBeforeMsgModifyCommand                = "callbackBeforeMsgModifyCommand"
    CallbackAfterUpdateUserInfoCommand            = "callbackAfterUpdateUserInfoCommand"
    CallbackAfterUpdateUserInfoExCommand          = "callbackAfterUpdateUserInfoExCommand"
    CallbackBeforeUpdateUserInfoExCommand         = "callbackBeforeUpdateUserInfoExCommand"
    CallbackBeforeUserRegisterCommand             = "callbackBeforeUserRegisterCommand"
    CallbackAfterUserRegisterCommand              = "callbackAfterUserRegisterCommand"
    CallbackAfterTransferGroupOwnerCommand        = "callbackAfterTransferGroupOwnerCommand"
    CallbackBeforeSetFriendRemarkCommand          = "callbackBeforeSetFriendRemarkCommand"
    CallbackAfterSetFriendRemarkCommand           = "callbackAfterSetFriendRemarkCommand"
    CallbackAfterSingleMsgReadCommand             = "callbackAfterSingleMsgReadCommand"
    CallbackBeforeSendSingleMsgCommand            = "callbackBeforeSendSingleMsgCommand"
    CallbackAfterSendSingleMsgCommand             = "callbackAfterSendSingleMsgCommand"
    CallbackBeforeSendGroupMsgCommand             = "callbackBeforeSendGroupMsgCommand"
    CallbackAfterSendGroupMsgCommand              = "callbackAfterSendGroupMsgCommand"
    CallbackAfterUserOnlineCommand                = "callbackAfterUserOnlineCommand"
    CallbackAfterUserOfflineCommand               = "callbackAfterUserOfflineCommand"
    CallbackAfterUserKickOffCommand               = "callbackAfterUserKickOffCommand"
    CallbackBeforeOfflinePushCommand              = "callbackBeforeOfflinePushCommand"
    CallbackBeforeOnlinePushCommand               = "callbackBeforeOnlinePushCommand"
    CallbackBeforeGroupOnlinePushCommand          = "callbackBeforeGroupOnlinePushCommand"
    CallbackBeforeAddFriendCommand                = "callbackBeforeAddFriendCommand"
    CallbackBeforeUpdateUserInfoCommand           = "callbackBeforeUpdateUserInfoCommand"
    CallbackBeforeCreateGroupCommand              = "callbackBeforeCreateGroupCommand"
    CallbackAfterCreateGroupCommand               = "callbackAfterCreateGroupCommand"
    CallbackBeforeMembersJoinGroupCommand         = "callbackBeforeMembersJoinGroupCommand"
    CallbackBeforeSetGroupMemberInfoCommand       = "callbackBeforeSetGroupMemberInfoCommand"
    CallbackAfterSetGroupMemberInfoCommand        = "callbackAfterSetGroupMemberInfoCommand"
```

```
    CallbackBeforeCreateSingleChatConversationsCommand = "callbackBeforeCreateSingleChatConversationsCommand"
    CallbackAfterCreateSingleChatConversationsCommand  = "callbackAfterCreateSingleChatConversationsCommand"
    CallbackBeforeCreateGroupChatConversationsCommand  = "callbackBeforeCreateGroupChatConversationsCommand"
    CallbackAfterCreateGroupChatConversationsCommand   = "callbackAfterCreateGroupChatConversationsCommand"
    CallbackAfterMsgSaveDBCommand                      = "callbackAfterMsgSaveDBCommand"
)
```

pkg/callbackstruct/conversation.go

```go
package callbackstruct

type CallbackBeforeCreateSingleChatConversationsReq struct {
	CallbackCommand  `json:"callbackCommand"`
	OwnerUserID      string `json:"ownerUserId"`
	ConversationID   string `json:"conversationId"`
	ConversationType int32  `json:"conversationType"`
	UserID           string `json:"userId"`
	RecvMsgOpt       int32  `json:"recvMsgOpt"`
	IsPinned         bool   `json:"isPinned"`
	IsPrivateChat    bool   `json:"isPrivateChat"`
	BurnDuration     int32  `json:"burnDuration"`
	GroupAtType      int32  `json:"groupAtType"`
	AttachedInfo     string `json:"attachedInfo"`
	Ex               string `json:"ex"`
}

type CallbackBeforeCreateSingleChatConversationsResp struct {
	CommonCallbackResp
	RecvMsgOpt    *int32  `json:"recvMsgOpt"`
	IsPinned      *bool   `json:"isPinned"`
	IsPrivateChat *bool   `json:"isPrivateChat"`
	BurnDuration  *int32  `json:"burnDuration"`
	GroupAtType   *int32  `json:"groupAtType"`
	AttachedInfo  *string `json:"attachedInfo"`
	Ex            *string `json:"ex"`
}

type CallbackAfterCreateSingleChatConversationsReq struct {
	CallbackCommand  `json:"callbackCommand"`
	OwnerUserID      string `json:"ownerUserId"`
	ConversationID   string `json:"conversationId"`
	ConversationType int32  `json:"conversationType"`
	UserID           string `json:"userId"`
	RecvMsgOpt       int32  `json:"recvMsgOpt"`
	IsPinned         bool   `json:"isPinned"`
	IsPrivateChat    bool   `json:"isPrivateChat"`
	BurnDuration     int32  `json:"burnDuration"`
	GroupAtType      int32  `json:"groupAtType"`
	AttachedInfo     string `json:"attachedInfo"`
	Ex               string `json:"ex"`
}

type CallbackAfterCreateSingleChatConversationsResp struct {
	CommonCallbackResp
}

type CallbackBeforeCreateGroupChatConversationsReq struct {
	CallbackCommand  `json:"callbackCommand"`
	OwnerUserID      string `json:"ownerUserId"`
	ConversationID   string `json:"conversationId"`
	ConversationType int32  `json:"conversationType"`
	GroupID          string `json:"groupId"`
	RecvMsgOpt       int32  `json:"recvMsgOpt"`
	IsPinned         bool   `json:"isPinned"`
	IsPrivateChat    bool   `json:"isPrivateChat"`
	BurnDuration     int32  `json:"burnDuration"`
	GroupAtType      int32  `json:"groupAtType"`
	AttachedInfo     string `json:"attachedInfo"`
	Ex               string `json:"ex"`
}

type CallbackBeforeCreateGroupChatConversationsResp struct {
	CommonCallbackResp
```

```
	RecvMsgOpt    *int32  `json:"recvMsgOpt"`
	IsPinned      *bool   `json:"isPinned"`
	IsPrivateChat *bool   `json:"isPrivateChat"`
	BurnDuration  *int32  `json:"burnDuration"`
	GroupAtType   *int32  `json:"groupAtType"`
	AttachedInfo  *string `json:"attachedInfo"`
	Ex            *string `json:"ex"`
}

type CallbackAfterCreateGroupChatConversationsReq struct {
	CallbackCommand  `json:"callbackCommand"`
	OwnerUserID      string `json:"ownerUserId"`
	ConversationID   string `json:"conversationId"`
	ConversationType int32  `json:"conversationType"`
	GroupID          string `json:"groupId"`
	RecvMsgOpt       int32  `json:"recvMsgOpt"`
	IsPinned         bool   `json:"isPinned"`
	IsPrivateChat    bool   `json:"isPrivateChat"`
	BurnDuration     int32  `json:"burnDuration"`
	GroupAtType      int32  `json:"groupAtType"`
	AttachedInfo     string `json:"attachedInfo"`
	Ex               string `json:"ex"`
}

type CallbackAfterCreateGroupChatConversationsResp struct {
	CommonCallbackResp
}
```

pkg/callbackstruct/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct // import "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
```

pkg/callbackstruct/friend.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

type CallbackBeforeAddFriendReq struct {
    CallbackCommand `json:"callbackCommand"`
    FromUserID      string `json:"fromUserID" `
    ToUserID        string `json:"toUserID"`
    ReqMsg          string `json:"reqMsg"`
    Ex              string `json:"ex"`
}

type CallbackBeforeAddFriendResp struct {
    CommonCallbackResp
}

type CallBackAddFriendReplyBeforeReq struct {
    CallbackCommand `json:"callbackCommand"`
    FromUserID      string `json:"fromUserID" `
    ToUserID        string `json:"toUserID"`
}

type CallBackAddFriendReplyBeforeResp struct {
    CommonCallbackResp
}

type CallbackBeforeSetFriendRemarkReq struct {
    CallbackCommand `json:"callbackCommand"`
    OwnerUserID     string `json:"ownerUserID"`
    FriendUserID    string `json:"friendUserID"`
    Remark          string `json:"remark"`
}

type CallbackBeforeSetFriendRemarkResp struct {
    CommonCallbackResp
    Remark string `json:"remark"`
}

type CallbackAfterSetFriendRemarkReq struct {
    CallbackCommand `json:"callbackCommand"`
    OwnerUserID     string `json:"ownerUserID"`
    FriendUserID    string `json:"friendUserID"`
    Remark          string `json:"remark"`
}

type CallbackAfterSetFriendRemarkResp struct {
    CommonCallbackResp
}
type CallbackAfterAddFriendReq struct {
    CallbackCommand `json:"callbackCommand"`
    FromUserID      string `json:"fromUserID" `
    ToUserID        string `json:"toUserID"`
```

```
	ReqMsg          string `json:"reqMsg"`
}

type CallbackAfterAddFriendResp struct {
	CommonCallbackResp
}
type CallbackBeforeAddBlackReq struct {
	CallbackCommand `json:"callbackCommand"`
	OwnerUserID     string `json:"ownerUserID" `
	BlackUserID     string `json:"blackUserID"`
}

type CallbackBeforeAddBlackResp struct {
	CommonCallbackResp
}

type CallbackBeforeAddFriendAgreeReq struct {
	CallbackCommand `json:"callbackCommand"`
	FromUserID      string `json:"fromUserID" `
	ToUserID        string `json:"blackUserID"`
	HandleResult    int32  `json:"HandleResult"`
	HandleMsg       string `json:"HandleMsg"`
}

type CallbackBeforeAddFriendAgreeResp struct {
	CommonCallbackResp
}

type CallbackAfterAddFriendAgreeReq struct {
	CallbackCommand `json:"callbackCommand"`
	FromUserID      string `json:"fromUserID" `
	ToUserID        string `json:"blackUserID"`
	HandleResult    int32  `json:"HandleResult"`
	HandleMsg       string `json:"HandleMsg"`
}

type CallbackAfterAddFriendAgreeResp struct {
	CommonCallbackResp
}

type CallbackAfterDeleteFriendReq struct {
	CallbackCommand `json:"callbackCommand"`
	OwnerUserID     string `json:"ownerUserID" `
	FriendUserID    string `json:"friendUserID"`
}
type CallbackAfterDeleteFriendResp struct {
	CommonCallbackResp
}

type CallbackBeforeImportFriendsReq struct {
	CallbackCommand `json:"callbackCommand"`
	OwnerUserID     string   `json:"ownerUserID" `
	FriendUserIDs   []string `json:"friendUserIDs"`
}
type CallbackBeforeImportFriendsResp struct {
	CommonCallbackResp
	FriendUserIDs []string `json:"friendUserIDs"`
}
type CallbackAfterImportFriendsReq struct {
	CallbackCommand `json:"callbackCommand"`
	OwnerUserID     string   `json:"ownerUserID" `
	FriendUserIDs   []string `json:"friendUserIDs"`
}
type CallbackAfterImportFriendsResp struct {
	CommonCallbackResp
}
```

```go
type CallbackAfterRemoveBlackReq struct {
    CallbackCommand `json:"callbackCommand"`
    OwnerUserID     string `json:"ownerUserID"`
    BlackUserID     string `json:"blackUserID"`
}
type CallbackAfterRemoveBlackResp struct {
    CommonCallbackResp
}
```

pkg/callbackstruct/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

import (
    "github.com/openimsdk/open-im-server/v3/pkg/apistruct"
    common "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/protocol/wrapperspb"
)

type CallbackCommand string

func (c CallbackCommand) GetCallbackCommand() string {
    return string(c)
}

type CallbackBeforeCreateGroupReq struct {
    OperationID     string `json:"operationID"`
    CallbackCommand `json:"callbackCommand"`
    *common.GroupInfo
    InitMemberList []*apistruct.GroupAddMemberInfo `json:"initMemberList"`
}

type CallbackBeforeCreateGroupResp struct {
    CommonCallbackResp
    GroupID           *string `json:"groupID"`
    GroupName         *string `json:"groupName"`
    Notification      *string `json:"notification"`
    Introduction      *string `json:"introduction"`
    FaceURL           *string `json:"faceURL"`
    OwnerUserID       *string `json:"ownerUserID"`
    Ex                *string `json:"ex"`
    Status            *int32  `json:"status"`
    CreatorUserID     *string `json:"creatorUserID"`
    GroupType         *int32  `json:"groupType"`
    NeedVerification  *int32  `json:"needVerification"`
    LookMemberInfo    *int32  `json:"lookMemberInfo"`
    ApplyMemberFriend *int32  `json:"applyMemberFriend"`
}

type CallbackAfterCreateGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    *common.GroupInfo
    InitMemberList []*apistruct.GroupAddMemberInfo `json:"initMemberList"`
}

type CallbackAfterCreateGroupResp struct {
    CommonCallbackResp
}

type CallbackGroupMember struct {
    UserID string `json:"userID"`
```

```
    Ex     string `json:"ex"`
}

type CallbackBeforeMembersJoinGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string                 `json:"groupID"`
    MembersList     []*CallbackGroupMember `json:"memberList"`
    GroupEx         string                 `json:"groupEx"`
}

type MemberJoinGroupCallBack struct {
    UserID      *string `json:"userID"`
    Nickname    *string `json:"nickname"`
    FaceURL     *string `json:"faceURL"`
    RoleLevel   *int32  `json:"roleLevel"`
    MuteEndTime *int64  `json:"muteEndTime"`
    Ex          *string `json:"ex"`
}

type CallbackBeforeMembersJoinGroupResp struct {
    CommonCallbackResp
    MemberCallbackList []*MemberJoinGroupCallBack `json:"memberCallbackList"`
}

type CallbackBeforeSetGroupMemberInfoReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string  `json:"groupID"`
    UserID          string  `json:"userID"`
    Nickname        *string `json:"nickName"`
    FaceURL         *string `json:"faceURL"`
    RoleLevel       *int32  `json:"roleLevel"`
    Ex              *string `json:"ex"`
}

type CallbackBeforeSetGroupMemberInfoResp struct {
    CommonCallbackResp
    Ex        *string `json:"ex"`
    Nickname  *string `json:"nickName"`
    FaceURL   *string `json:"faceURL"`
    RoleLevel *int32  `json:"roleLevel"`
}

type CallbackAfterSetGroupMemberInfoReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string  `json:"groupID"`
    UserID          string  `json:"userID"`
    Nickname        *string `json:"nickName"`
    FaceURL         *string `json:"faceURL"`
    RoleLevel       *int32  `json:"roleLevel"`
    Ex              *string `json:"ex"`
}

type CallbackAfterSetGroupMemberInfoResp struct {
    CommonCallbackResp
}

type CallbackQuitGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string `json:"groupID"`
    UserID          string `json:"userID"`
}

type CallbackQuitGroupResp struct {
    CommonCallbackResp
}
```

```
type CallbackKillGroupMemberReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string   `json:"groupID"`
    KickedUserIDs   []string `json:"kickedUserIDs"`
    Reason          string   `json:"reason"`
}

type CallbackKillGroupMemberResp struct {
    CommonCallbackResp
}

type CallbackDisMissGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string   `json:"groupID"`
    OwnerID         string   `json:"ownerID"`
    GroupType       string   `json:"groupType"`
    MembersID       []string `json:"membersID"`
}

type CallbackDisMissGroupResp struct {
    CommonCallbackResp
}

type CallbackJoinGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string `json:"groupID"`
    GroupType       string `json:"groupType"`
    ApplyID         string `json:"applyID"`
    ReqMessage      string `json:"reqMessage"`
    Ex              string `json:"ex"`
}

type CallbackJoinGroupResp struct {
    CommonCallbackResp
}

type CallbackTransferGroupOwnerReq struct {
    CallbackCommand `json:"callbackCommand"`
    GroupID         string `json:"groupID"`
    OldOwnerUserID  string `json:"oldOwnerUserID"`
    NewOwnerUserID  string `json:"newOwnerUserID"`
}

type CallbackTransferGroupOwnerResp struct {
    CommonCallbackResp
}

type CallbackBeforeInviteUserToGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    OperationID     string   `json:"operationID"`
    GroupID         string   `json:"groupID"`
    Reason          string   `json:"reason"`
    InvitedUserIDs  []string `json:"invitedUserIDs"`
}
type CallbackBeforeInviteUserToGroupResp struct {
    CommonCallbackResp
    RefusedMembersAccount []string `json:"refusedMembersAccount,omitempty"` // Optional field to list members whose inv
}

type CallbackAfterJoinGroupReq struct {
    CallbackCommand `json:"callbackCommand"`
    OperationID     string `json:"operationID"`
    GroupID         string `json:"groupID"`
    ReqMessage      string `json:"reqMessage"`
    JoinSource      int32  `json:"joinSource"`
    InviterUserID   string `json:"inviterUserID"`
```

```
}
type CallbackAfterJoinGroupResp struct {
    CommonCallbackResp
}

type CallbackBeforeSetGroupInfoReq struct {
    CallbackCommand   `json:"callbackCommand"`
    OperationID       string `json:"operationID"`
    GroupID           string `json:"groupID"`
    GroupName         string `json:"groupName"`
    Notification      string `json:"notification"`
    Introduction      string `json:"introduction"`
    FaceURL           string `json:"faceURL"`
    Ex                string `json:"ex"`
    NeedVerification  int32  `json:"needVerification"`
    LookMemberInfo    int32  `json:"lookMemberInfo"`
    ApplyMemberFriend int32  `json:"applyMemberFriend"`
}

type CallbackBeforeSetGroupInfoResp struct {
    CommonCallbackResp
    GroupID           string  ` json:"groupID"`
    GroupName         string  `json:"groupName"`
    Notification      string  `json:"notification"`
    Introduction      string  `json:"introduction"`
    FaceURL           string  `json:"faceURL"`
    Ex                *string `json:"ex"`
    NeedVerification  *int32  `json:"needVerification"`
    LookMemberInfo    *int32  `json:"lookMemberInfo"`
    ApplyMemberFriend *int32  `json:"applyMemberFriend"`
}

type CallbackAfterSetGroupInfoReq struct {
    CallbackCommand   `json:"callbackCommand"`
    OperationID       string  `json:"operationID"`
    GroupID           string  `json:"groupID"`
    GroupName         string  `json:"groupName"`
    Notification      string  `json:"notification"`
    Introduction      string  `json:"introduction"`
    FaceURL           string  `json:"faceURL"`
    Ex                *string `json:"ex"`
    NeedVerification  *int32  `json:"needVerification"`
    LookMemberInfo    *int32  `json:"lookMemberInfo"`
    ApplyMemberFriend *int32  `json:"applyMemberFriend"`
}

type CallbackAfterSetGroupInfoResp struct {
    CommonCallbackResp
}

type CallbackBeforeSetGroupInfoExReq struct {
    CallbackCommand   `json:"callbackCommand"`
    OperationID       string                    `json:"operationID"`
    GroupID           string                    `json:"groupID"`
    GroupName         *wrapperspb.StringValue   `json:"groupName"`
    Notification      *wrapperspb.StringValue   `json:"notification"`
    Introduction      *wrapperspb.StringValue   `json:"introduction"`
    FaceURL           *wrapperspb.StringValue   `json:"faceURL"`
    Ex                *wrapperspb.StringValue   `json:"ex"`
    NeedVerification  *wrapperspb.Int32Value    `json:"needVerification"`
    LookMemberInfo    *wrapperspb.Int32Value    `json:"lookMemberInfo"`
    ApplyMemberFriend *wrapperspb.Int32Value    `json:"applyMemberFriend"`
}

type CallbackBeforeSetGroupInfoExResp struct {
    CommonCallbackResp
```

```
    GroupID           string                  `json:"groupID"`
    GroupName         *wrapperspb.StringValue `json:"groupName"`
    Notification      *wrapperspb.StringValue `json:"notification"`
    Introduction      *wrapperspb.StringValue `json:"introduction"`
    FaceURL           *wrapperspb.StringValue `json:"faceURL"`
    Ex                *wrapperspb.StringValue `json:"ex"`
    NeedVerification  *wrapperspb.Int32Value  `json:"needVerification"`
    LookMemberInfo    *wrapperspb.Int32Value  `json:"lookMemberInfo"`
    ApplyMemberFriend *wrapperspb.Int32Value  `json:"applyMemberFriend"`
}

type CallbackAfterSetGroupInfoExReq struct {
    CallbackCommand   `json:"callbackCommand"`
    OperationID       string                  `json:"operationID"`
    GroupID           string                  `json:"groupID"`
    GroupName         *wrapperspb.StringValue `json:"groupName"`
    Notification      *wrapperspb.StringValue `json:"notification"`
    Introduction      *wrapperspb.StringValue `json:"introduction"`
    FaceURL           *wrapperspb.StringValue `json:"faceURL"`
    Ex                *wrapperspb.StringValue `json:"ex"`
    NeedVerification  *wrapperspb.Int32Value  `json:"needVerification"`
    LookMemberInfo    *wrapperspb.Int32Value  `json:"lookMemberInfo"`
    ApplyMemberFriend *wrapperspb.Int32Value  `json:"applyMemberFriend"`
}

type CallbackAfterSetGroupInfoExResp struct {
    CommonCallbackResp
}
```

pkg/callbackstruct/message.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

import (
    sdkws "github.com/openimsdk/protocol/sdkws"
)

type CallbackBeforeSendSingleMsgReq struct {
    CommonCallbackReq
    RecvID string `json:"recvID"`
}

type CallbackBeforeSendSingleMsgResp struct {
    CommonCallbackResp
}

type CallbackAfterSendSingleMsgReq struct {
    CommonCallbackReq
    RecvID string `json:"recvID"`
}

type CallbackAfterSendSingleMsgResp struct {
    CommonCallbackResp
}

type CallbackBeforeSendGroupMsgReq struct {
    CommonCallbackReq
    GroupID string `json:"groupID"`
}

type CallbackBeforeSendGroupMsgResp struct {
    CommonCallbackResp
}

type CallbackAfterSendGroupMsgReq struct {
    CommonCallbackReq
    GroupID string `json:"groupID"`
}

type CallbackAfterSendGroupMsgResp struct {
    CommonCallbackResp
}

type CallbackMsgModifyCommandReq struct {
    CommonCallbackReq
}

type CallbackMsgModifyCommandResp struct {
    CommonCallbackResp
    Content          *string                `json:"content"`
    RecvID           *string                `json:"recvID"`
```

```
    GroupID          *string                `json:"groupID"`
    ClientMsgID      *string                `json:"clientMsgID"`
    ServerMsgID      *string                `json:"serverMsgID"`
    SenderPlatformID *int32                 `json:"senderPlatformID"`
    SenderNickname   *string                `json:"senderNickname"`
    SenderFaceURL    *string                `json:"senderFaceURL"`
    SessionType      *int32                 `json:"sessionType"`
    MsgFrom          *int32                 `json:"msgFrom"`
    ContentType      *int32                 `json:"contentType"`
    Status           *int32                 `json:"status"`
    Options          *map[string]bool       `json:"options"`
    OfflinePushInfo  *sdkws.OfflinePushInfo `json:"offlinePushInfo"`
    AtUserIDList     *[]string              `json:"atUserIDList"`
    MsgDataList      *[]byte                `json:"msgDataList"`
    AttachedInfo     *string                `json:"attachedInfo"`
    Ex               *string                `json:"ex"`
}

type CallbackGroupMsgReadReq struct {
    CallbackCommand `json:"callbackCommand"`
    SendID          string `json:"sendID"`
    ReceiveID       string `json:"receiveID"`
    UnreadMsgNum    int64  `json:"unreadMsgNum"`
    ContentType     int64  `json:"contentType"`
}

type CallbackGroupMsgReadResp struct {
    CommonCallbackResp
}

type CallbackSingleMsgReadReq struct {
    CallbackCommand `json:"callbackCommand"`
    ConversationID  string  `json:"conversationID"`
    UserID          string  `json:"userID"`
    Seqs            []int64 `json:"Seqs"`
    ContentType     int32   `json:"contentType"`
}

type CallbackSingleMsgReadResp struct {
    CommonCallbackResp
}

type CallbackAfterMsgSaveDBReq struct {
    CommonCallbackReq
    RecvID  string `json:"recvID"`
    GroupID string `json:"groupID"`
}

type CallbackAfterMsgSaveDBResp struct {
    CommonCallbackResp
}
```

pkg/callbackstruct/msg_gateway.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

type CallbackUserOnlineReq struct {
    UserStatusCallbackReq
    // Token           string `json:"token"`
    Seq             int64  `json:"seq"`
    IsAppBackground bool   `json:"isAppBackground"`
    ConnID          string `json:"connID"`
}

type CallbackUserOnlineResp struct {
    CommonCallbackResp
}

type CallbackUserOfflineReq struct {
    UserStatusCallbackReq
    Seq    int64  `json:"seq"`
    ConnID string `json:"connID"`
}

type CallbackUserOfflineResp struct {
    CommonCallbackResp
}

type CallbackUserKickOffReq struct {
    UserStatusCallbackReq
    Seq int64 `json:"seq"`
}

type CallbackUserKickOffResp struct {
    CommonCallbackResp
}
```

pkg/callbackstruct/push.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

import common "github.com/openimsdk/protocol/sdkws"

type CallbackBeforePushReq struct {
    UserStatusBatchCallbackReq
    *common.OfflinePushInfo
    ClientMsgID string   `json:"clientMsgID"`
    SendID      string   `json:"sendID"`
    GroupID     string   `json:"groupID"`
    ContentType int32    `json:"contentType"`
    SessionType int32    `json:"sessionType"`
    AtUserIDs   []string `json:"atUserIDList"`
    Content     string   `json:"content"`
}

type CallbackBeforePushResp struct {
    CommonCallbackResp
    UserIDs         []string                `json:"userIDList"`
    OfflinePushInfo *common.OfflinePushInfo `json:"offlinePushInfo"`
}

type CallbackBeforeSuperGroupOnlinePushReq struct {
    UserStatusBaseCallback
    ClientMsgID string   `json:"clientMsgID"`
    SendID      string   `json:"sendID"`
    GroupID     string   `json:"groupID"`
    ContentType int32    `json:"contentType"`
    SessionType int32    `json:"sessionType"`
    AtUserIDs   []string `json:"atUserIDList"`
    Content     string   `json:"content"`
    Seq         int64    `json:"seq"`
}

type CallbackBeforeSuperGroupOnlinePushResp struct {
    CommonCallbackResp
    UserIDs         []string                `json:"userIDList"`
    OfflinePushInfo *common.OfflinePushInfo `json:"offlinePushInfo"`
}
```

pkg/callbackstruct/revoke.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

type CallbackAfterRevokeMsgReq struct {
    CallbackCommand `json:"callbackCommand"`
    ConversationID  string `json:"conversationID"`
    Seq             int64  `json:"seq"`
    UserID          string `json:"userID"`
}

type CallbackAfterRevokeMsgResp struct {
    CommonCallbackResp
}
```

pkg/callbackstruct/user.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package callbackstruct

import (
"github.com/openimsdk/protocol/sdkws"
"github.com/openimsdk/protocol/wrapperspb"
)

type CallbackBeforeUpdateUserInfoReq struct {
CallbackCommand `json:"callbackCommand"`
UserID          string  `json:"userID"`
Nickname        *string `json:"nickName"`
FaceURL         *string `json:"faceURL"`
Ex              *string `json:"ex"`
}

type CallbackBeforeUpdateUserInfoResp struct {
CommonCallbackResp
Nickname *string `json:"nickName"`
FaceURL  *string `json:"faceURL"`
Ex       *string `json:"ex"`
}

type CallbackAfterUpdateUserInfoReq struct {
CallbackCommand `json:"callbackCommand"`
UserID          string `json:"userID"`
Nickname        string `json:"nickName"`
FaceURL         string `json:"faceURL"`
Ex              string `json:"ex"`
}
type CallbackAfterUpdateUserInfoResp struct {
CommonCallbackResp
}

type CallbackBeforeUpdateUserInfoExReq struct {
CallbackCommand `json:"callbackCommand"`
UserID          string                  `json:"userID"`
Nickname        *wrapperspb.StringValue `json:"nickName"`
FaceURL         *wrapperspb.StringValue `json:"faceURL"`
Ex              *wrapperspb.StringValue `json:"ex"`
}
type CallbackBeforeUpdateUserInfoExResp struct {
CommonCallbackResp
Nickname *wrapperspb.StringValue `json:"nickName"`
FaceURL  *wrapperspb.StringValue `json:"faceURL"`
Ex       *wrapperspb.StringValue `json:"ex"`
}

type CallbackAfterUpdateUserInfoExReq struct {
CallbackCommand `json:"callbackCommand"`
UserID          string                  `json:"userID"`
```

```
	Nickname        *wrapperspb.StringValue `json:"nickName"`
	FaceURL         *wrapperspb.StringValue `json:"faceURL"`
	Ex              *wrapperspb.StringValue `json:"ex"`
}
type CallbackAfterUpdateUserInfoExResp struct {
	CommonCallbackResp
}

type CallbackBeforeUserRegisterReq struct {
	CallbackCommand `json:"callbackCommand"`
	Users           []*sdkws.UserInfo `json:"users"`
}

type CallbackBeforeUserRegisterResp struct {
	CommonCallbackResp
	Users []*sdkws.UserInfo `json:"users"`
}

type CallbackAfterUserRegisterReq struct {
	CallbackCommand `json:"callbackCommand"`
	Users           []*sdkws.UserInfo `json:"users"`
}

type CallbackAfterUserRegisterResp struct {
	CommonCallbackResp
}
```

pkg/statistics

pkg/statistics/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package statistics // import "github.com/openimsdk/open-im-server/v3/pkg/statistics"
```

pkg/statistics/statistics.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package statistics

import (
	"context"
	"time"

	"github.com/openimsdk/tools/log"
)

type Statistics struct {
	AllCount   *uint64
	ModuleName string
	PrintArgs  string
	SleepTime  uint64
}

func (s *Statistics) output() {
	var intervalCount uint64
	t := time.NewTicker(time.Duration(s.SleepTime) * time.Second)
	defer t.Stop()
	var sum uint64
	var timeIntervalNum uint64
	for {
		sum = *s.AllCount
		<-t.C
		if *s.AllCount-sum <= 0 {
			intervalCount = 0
		} else {
			intervalCount = *s.AllCount - sum
		}
		timeIntervalNum++
		log.ZWarn(
			context.Background(),
			" system stat ",
			nil,
			"args",
			s.PrintArgs,
			"intervalCount",
			intervalCount,
			"total:",
			*s.AllCount,
			"intervalNum",
			timeIntervalNum,
			"avg",
			(*s.AllCount)/(timeIntervalNum)/s.SleepTime,
		)
	}
}

func NewStatistics(allCount *uint64, moduleName, printArgs string, sleepTime int) *Statistics {
```

```
    p := &Statistics{AllCount: allCount, ModuleName: moduleName, SleepTime: uint64(sleepTime), PrintArgs: printArgs}
    go p.output()
    return p
}
```

pkg/rpccache

pkg/rpccache/auth.go

```
package rpccache

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

func NewAuthLocalCache(client *rpcli.AuthClient, localCache *config.LocalCache, cli redis.UniversalClient) *AuthLoca
	lc := localCache.Auth
	log.ZDebug(context.Background(), "AuthLocalCache", "topic", lc.Topic, "slotNum", lc.SlotNum, "slotSize", lc.SlotSiz
	x := &AuthLocalCache{
		client: client,
		local: localcache.New[[]byte](
			localcache.WithLocalSlotNum(lc.SlotNum),
			localcache.WithLocalSlotSize(lc.SlotSize),
			localcache.WithLinkSlotNum(lc.SlotNum),
			localcache.WithLocalSuccessTTL(lc.Success()),
			localcache.WithLocalFailedTTL(lc.Failed()),
		),
	}
	if lc.Enable() {
		go subscriberRedisDeleteCache(context.Background(), cli, lc.Topic, x.local.DelLocal)
	}
	return x
}

type AuthLocalCache struct {
	client *rpcli.AuthClient
	local  localcache.Cache[[]byte]
}

func (a *AuthLocalCache) GetExistingToken(ctx context.Context, userID string, platformID int) (val map[string]int, e
	resp, err := a.getExistingToken(ctx, userID, platformID)
	if err != nil {
		return nil, err
	}

	res := convert.TokenMapPb2DB(resp.TokenStates)

	return res, nil
}

func (a *AuthLocalCache) getExistingToken(ctx context.Context, userID string, platformID int) (val *auth.GetExisting
	start := time.Now()
	log.ZDebug(ctx, "AuthLocalCache GetExistingToken req", "userID", userID, "platformID", platformID)
	defer func() {
		if err != nil {
			log.ZError(ctx, "AuthLocalCache GetExistingToken error", err, "cost", time.Since(start), "userID", userID, "plat
		} else {
			log.ZDebug(ctx, "AuthLocalCache GetExistingToken resp", "cost", time.Since(start), "userID", userID, "platformID
		}
	}()
```

```
    var cache cacheProto[auth.GetExistingTokenResp]

    return cache.Unmarshal(a.local.Get(ctx, cachekey.GetTokenKey(userID, platformID), func(ctx context.Context) ([]byte
        log.ZDebug(ctx, "AuthLocalCache GetExistingToken call rpc", "userID", userID, "platformID", platformID)
        return cache.Marshal(a.client.AuthClient.GetExistingToken(ctx, &auth.GetExistingTokenReq{UserID: userID, Platform
    }))
}
```

pkg/rpccache/common.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache

import (
    "github.com/openimsdk/tools/errs"
    "google.golang.org/protobuf/proto"
)

func newListMap[V comparable](values []V, err error) (*listMap[V], error) {
    if err != nil {
        return nil, err
    }
    lm := &listMap[V]{
        List: values,
        Map:  make(map[V]struct{}, len(values)),
    }
    for _, value := range values {
        lm.Map[value] = struct{}{}
    }
    return lm, nil
}

type listMap[V comparable] struct {
    List []V
    Map  map[V]struct{}
}

func respProtoMarshal(resp proto.Message, err error) ([]byte, error) {
    if err != nil {
        return nil, err
    }
    return proto.Marshal(resp)
}

func cacheUnmarshal[V any](resp []byte, err error) (*V, error) {
    if err != nil {
        return nil, err
    }
    var val V
    if err := proto.Unmarshal(resp, any(&val).(proto.Message)); err != nil {
        return nil, errs.WrapMsg(err, "local cache proto.Unmarshal error")
    }
    return &val, nil
}

type cacheProto[V any] struct{}

func (cacheProto[V]) Marshal(resp *V, err error) ([]byte, error) {
    if err != nil {
        return nil, err
    }
```

```
    return proto.Marshal(any(resp).(proto.Message))
}

func (cacheProto[V]) Unmarshal(resp []byte, err error) (*V, error) {
    if err != nil {
        return nil, err
    }
    var val V
    if err := proto.Unmarshal(resp, any(&val).(proto.Message)); err != nil {
        return nil, errs.WrapMsg(err, "local cache proto.Unmarshal error")
    }
    return &val, nil
}
```

pkg/rpccache/conversation.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	pbconversation "github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
	"golang.org/x/sync/errgroup"
)

const (
	conversationWorkerCount = 20
)

func NewConversationLocalCache(client *rpcli.ConversationClient, localCache *config.LocalCache, cli redis.UniversalC
	lc := localCache.Conversation
	log.ZDebug(context.Background(), "ConversationLocalCache", "topic", lc.Topic, "slotNum", lc.SlotNum, "slotSize", lc
	x := &ConversationLocalCache{
		client: client,
		local: localcache.New[[]byte](
			localcache.WithLocalSlotNum(lc.SlotNum),
			localcache.WithLocalSlotSize(lc.SlotSize),
			localcache.WithLinkSlotNum(lc.SlotNum),
			localcache.WithLocalSuccessTTL(lc.Success()),
			localcache.WithLocalFailedTTL(lc.Failed()),
		),
	}
	if lc.Enable() {
		go subscriberRedisDeleteCache(context.Background(), cli, lc.Topic, x.local.DelLocal)
	}
	return x
}

type ConversationLocalCache struct {
	client *rpcli.ConversationClient
	local  localcache.Cache[[]byte]
}

func (c *ConversationLocalCache) GetConversationIDs(ctx context.Context, ownerUserID string) (val []string, err erro
	resp, err := c.getConversationIDs(ctx, ownerUserID)
	if err != nil {
		return nil, err
	}
```

```
	return resp.ConversationIDs, nil
}

func (c *ConversationLocalCache) getConversationIDs(ctx context.Context, ownerUserID string) (val *pbconversation.Ge
	log.ZDebug(ctx, "ConversationLocalCache getConversationIDs req", "ownerUserID", ownerUserID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "ConversationLocalCache getConversationIDs return", "ownerUserID", ownerUserID, "value", val)
		} else {
			log.ZError(ctx, "ConversationLocalCache getConversationIDs return", err, "ownerUserID", ownerUserID)
		}
	}()
	var cache cacheProto[pbconversation.GetConversationIDsResp]
	return cache.Unmarshal(c.local.Get(ctx, cachekey.GetConversationIDsKey(ownerUserID), func(ctx context.Context) ([]b
		log.ZDebug(ctx, "ConversationLocalCache getConversationIDs rpc", "ownerUserID", ownerUserID)
		return cache.Marshal(c.client.ConversationClient.GetConversationIDs(ctx, &pbconversation.GetConversationIDsReq{Use
	}))
}

func (c *ConversationLocalCache) GetConversation(ctx context.Context, userID, conversationID string) (val *pbconvers
	log.ZDebug(ctx, "ConversationLocalCache GetConversation req", "userID", userID, "conversationID", conversationID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "ConversationLocalCache GetConversation return", "userID", userID, "conversationID", conversatio
		} else {
			log.ZWarn(ctx, "ConversationLocalCache GetConversation return", err, "userID", userID, "conversationID", convers
		}
	}()
	var cache cacheProto[pbconversation.Conversation]
	return cache.Unmarshal(c.local.Get(ctx, cachekey.GetConversationKey(userID, conversationID), func(ctx context.Conte
		log.ZDebug(ctx, "ConversationLocalCache GetConversation rpc", "userID", userID, "conversationID", conversationID)
		return cache.Marshal(c.client.GetConversation(ctx, conversationID, userID))
	}))
}

func (c *ConversationLocalCache) GetSingleConversationRecvMsgOpt(ctx context.Context, userID, conversationID string)
	conv, err := c.GetConversation(ctx, userID, conversationID)
	if err != nil {
		return 0, err
	}
	return conv.RecvMsgOpt, nil
}

func (c *ConversationLocalCache) GetConversations(ctx context.Context, ownerUserID string, conversationIDs []string)
	var (
		conversations     = make([]*pbconversation.Conversation, 0, len(conversationIDs))
		conversationsChan = make(chan *pbconversation.Conversation, len(conversationIDs))
	)

	g, ctx := errgroup.WithContext(ctx)
	g.SetLimit(conversationWorkerCount)

	for _, conversationID := range conversationIDs {
		conversationID := conversationID
		g.Go(func() error {
			conversation, err := c.GetConversation(ctx, ownerUserID, conversationID)
			if err != nil {
				if errs.ErrRecordNotFound.Is(err) {
					return nil
				}
				return err
			}
			conversationsChan <- conversation
			return nil
		})
	}
```

```
	if err := g.Wait(); err != nil {
		return nil, err
	}
	close(conversationsChan)
	for conversation := range conversationsChan {
		conversations = append(conversations, conversation)
	}
	return conversations, nil
}

func (c *ConversationLocalCache) getConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string)
	log.ZDebug(ctx, "ConversationLocalCache getConversationNotReceiveMessageUserIDs req", "conversationID", conversatio
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "ConversationLocalCache getConversationNotReceiveMessageUserIDs return", "conversationID", conve
		} else {
			log.ZError(ctx, "ConversationLocalCache getConversationNotReceiveMessageUserIDs return", err, "conversationID",
		}
	}()
	var cache cacheProto[pbconversation.GetConversationNotReceiveMessageUserIDsResp]
	return cache.Unmarshal(c.local.Get(ctx, cachekey.GetConversationNotReceiveMessageUserIDsKey(conversationID), func(c
		log.ZDebug(ctx, "ConversationLocalCache getConversationNotReceiveMessageUserIDs rpc", "conversationID", conversati
		return cache.Marshal(c.client.ConversationClient.GetConversationNotReceiveMessageUserIDs(ctx, &pbconversation.Get
	}))
}

func (c *ConversationLocalCache) getPinnedConversationIDs(ctx context.Context, userID string) (val []string, err err
	log.ZDebug(ctx, "ConversationLocalCache getPinnedConversations req", "userID", userID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "ConversationLocalCache getPinnedConversations return", "userID", userID, "value", val)
		} else {
			log.ZError(ctx, "ConversationLocalCache getPinnedConversations return", err, "userID", userID)
		}
	}()
	var cache cacheProto[pbconversation.GetPinnedConversationIDsResp]
	resp, err := cache.Unmarshal(c.local.Get(ctx, cachekey.GetPinnedConversationIDs(userID), func(ctx context.Context)
		log.ZDebug(ctx, "ConversationLocalCache getConversationNotReceiveMessageUserIDs rpc", "userID", userID)
		return cache.Marshal(c.client.ConversationClient.GetPinnedConversationIDs(ctx, &pbconversation.GetPinnedConversati
	}))
	if err != nil {
		return nil, err
	}
	return resp.ConversationIDs, nil
}

func (c *ConversationLocalCache) GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string)
	res, err := c.getConversationNotReceiveMessageUserIDs(ctx, conversationID)
	if err != nil {
		return nil, err
	}
	return res.UserIDs, nil
}

func (c *ConversationLocalCache) GetConversationNotReceiveMessageUserIDMap(ctx context.Context, conversationID strin
	res, err := c.getConversationNotReceiveMessageUserIDs(ctx, conversationID)
	if err != nil {
		return nil, err
	}
	return datautil.SliceSet(res.UserIDs), nil
}

func (c *ConversationLocalCache) GetPinnedConversationIDs(ctx context.Context, userID string) ([]string, error) {
	return c.getPinnedConversationIDs(ctx, userID)
}
```

pkg/rpccache/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache // import "github.com/openimsdk/open-im-server/v3/pkg/rpccache"
```

pkg/rpccache/friend.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/relation"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

func NewFriendLocalCache(client *rpcli.RelationClient, localCache *config.LocalCache, cli redis.UniversalClient) *Fr
	lc := localCache.Friend
	log.ZDebug(context.Background(), "FriendLocalCache", "topic", lc.Topic, "slotNum", lc.SlotNum, "slotSize", lc.SlotS
	x := &FriendLocalCache{
		client: client,
		local: localcache.New[[]byte](
			localcache.WithLocalSlotNum(lc.SlotNum),
			localcache.WithLocalSlotSize(lc.SlotSize),
			localcache.WithLinkSlotNum(lc.SlotNum),
			localcache.WithLocalSuccessTTL(lc.Success()),
			localcache.WithLocalFailedTTL(lc.Failed()),
		),
	}
	if lc.Enable() {
		go subscriberRedisDeleteCache(context.Background(), cli, lc.Topic, x.local.DelLocal)
	}
	return x
}

type FriendLocalCache struct {
	client *rpcli.RelationClient
	local  localcache.Cache[[]byte]
}

func (f *FriendLocalCache) IsFriend(ctx context.Context, possibleFriendUserID, userID string) (val bool, err error)
	res, err := f.isFriend(ctx, possibleFriendUserID, userID)
	if err != nil {
		return false, err
	}
	return res.InUser1Friends, nil
}

func (f *FriendLocalCache) isFriend(ctx context.Context, possibleFriendUserID, userID string) (val *relation.IsFrien
	log.ZDebug(ctx, "FriendLocalCache isFriend req", "possibleFriendUserID", possibleFriendUserID, "userID", userID)
	defer func() {
		if err == nil {
```

```
        log.ZDebug(ctx, "FriendLocalCache isFriend return", "possibleFriendUserID", possibleFriendUserID, "userID", user
    } else {
        log.ZError(ctx, "FriendLocalCache isFriend return", err, "possibleFriendUserID", possibleFriendUserID, "userID",
    }
}()
var cache cacheProto[relation.IsFriendResp]
return cache.Unmarshal(f.local.GetLink(ctx, cachekey.GetIsFriendKey(possibleFriendUserID, userID), func(ctx context
    log.ZDebug(ctx, "FriendLocalCache isFriend rpc", "possibleFriendUserID", possibleFriendUserID, "userID", userID)
    return cache.Marshal(f.client.FriendClient.IsFriend(ctx, &relation.IsFriendReq{UserID1: userID, UserID2: possible
}, cachekey.GetFriendIDsKey(possibleFriendUserID)))
}

// IsBlack possibleBlackUserID selfUserID.
func (f *FriendLocalCache) IsBlack(ctx context.Context, possibleBlackUserID, userID string) (val bool, err error) {
res, err := f.isBlack(ctx, possibleBlackUserID, userID)
if err != nil {
    return false, err
}
return res.InUser2Blacks, nil
}

// IsBlack possibleBlackUserID selfUserID.
func (f *FriendLocalCache) isBlack(ctx context.Context, possibleBlackUserID, userID string) (val *relation.IsBlackRe
log.ZDebug(ctx, "FriendLocalCache isBlack req", "possibleBlackUserID", possibleBlackUserID, "userID", userID)
defer func() {
    if err == nil {
        log.ZDebug(ctx, "FriendLocalCache isBlack return", "possibleBlackUserID", possibleBlackUserID, "userID", userID,
    } else {
        log.ZError(ctx, "FriendLocalCache isBlack return", err, "possibleBlackUserID", possibleBlackUserID, "userID", us
    }
}()
var cache cacheProto[relation.IsBlackResp]
return cache.Unmarshal(f.local.GetLink(ctx, cachekey.GetIsBlackIDsKey(possibleBlackUserID, userID), func(ctx contex
    log.ZDebug(ctx, "FriendLocalCache IsBlack rpc", "possibleBlackUserID", possibleBlackUserID, "userID", userID)
    return cache.Marshal(f.client.FriendClient.IsBlack(ctx, &relation.IsBlackReq{UserID1: possibleBlackUserID, UserID
}, cachekey.GetBlackIDsKey(userID)))
}
```

pkg/rpccache/group.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/tools/utils/datautil"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

func NewGroupLocalCache(client *rpcli.GroupClient, localCache *config.LocalCache, cli redis.UniversalClient) *GroupL
	lc := localCache.Group
	log.ZDebug(context.Background(), "GroupLocalCache", "topic", lc.Topic, "slotNum", lc.SlotNum, "slotSize", lc.SlotSi
	x := &GroupLocalCache{
		client: client,
		local: localcache.New[[]byte](
			localcache.WithLocalSlotNum(lc.SlotNum),
			localcache.WithLocalSlotSize(lc.SlotSize),
			localcache.WithLinkSlotNum(lc.SlotNum),
			localcache.WithLocalSuccessTTL(lc.Success()),
			localcache.WithLocalFailedTTL(lc.Failed()),
		),
	}
	if lc.Enable() {
		go subscriberRedisDeleteCache(context.Background(), cli, lc.Topic, x.local.DelLocal)
	}
	return x
}

type GroupLocalCache struct {
	client *rpcli.GroupClient
	local  localcache.Cache[[]byte]
}

func (g *GroupLocalCache) getGroupMemberIDs(ctx context.Context, groupID string) (val *group.GetGroupMemberUserIDsRe
	log.ZDebug(ctx, "GroupLocalCache getGroupMemberIDs req", "groupID", groupID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "GroupLocalCache getGroupMemberIDs return", "groupID", groupID, "value", val)
		} else {
			log.ZError(ctx, "GroupLocalCache getGroupMemberIDs return", err, "groupID", groupID)
		}
	}()
```

```
	var cache cacheProto[group.GetGroupMemberUserIDsResp]
	return cache.Unmarshal(g.local.Get(ctx, cachekey.GetGroupMemberIDsKey(groupID), func(ctx context.Context) ([]byte,
		log.ZDebug(ctx, "GroupLocalCache getGroupMemberIDs rpc", "groupID", groupID)
		return cache.Marshal(g.client.GroupClient.GetGroupMemberUserIDs(ctx, &group.GetGroupMemberUserIDsReq{GroupID: gro
	}))
}

func (g *GroupLocalCache) GetGroupMember(ctx context.Context, groupID, userID string) (val *sdkws.GroupMemberFullInf
	log.ZDebug(ctx, "GroupLocalCache GetGroupInfo req", "groupID", groupID, "userID", userID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "GroupLocalCache GetGroupInfo return", "groupID", groupID, "userID", userID, "value", val)
		} else {
			log.ZError(ctx, "GroupLocalCache GetGroupInfo return", err, "groupID", groupID, "userID", userID)
		}
	}()
	var cache cacheProto[sdkws.GroupMemberFullInfo]
	return cache.Unmarshal(g.local.Get(ctx, cachekey.GetGroupMemberInfoKey(groupID, userID), func(ctx context.Context)
		log.ZDebug(ctx, "GroupLocalCache GetGroupInfo rpc", "groupID", groupID, "userID", userID)
		return cache.Marshal(g.client.GetGroupMemberCache(ctx, groupID, userID))
	}))
}

func (g *GroupLocalCache) GetGroupInfo(ctx context.Context, groupID string) (val *sdkws.GroupInfo, err error) {
	log.ZDebug(ctx, "GroupLocalCache GetGroupInfo req", "groupID", groupID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "GroupLocalCache GetGroupInfo return", "groupID", groupID, "value", val)
		} else {
			log.ZError(ctx, "GroupLocalCache GetGroupInfo return", err, "groupID", groupID)
		}
	}()
	var cache cacheProto[sdkws.GroupInfo]
	return cache.Unmarshal(g.local.Get(ctx, cachekey.GetGroupInfoKey(groupID), func(ctx context.Context) ([]byte, error
		log.ZDebug(ctx, "GroupLocalCache GetGroupInfo rpc", "groupID", groupID)
		return cache.Marshal(g.client.GetGroupInfoCache(ctx, groupID))
	}))
}

func (g *GroupLocalCache) GetGroupMemberIDs(ctx context.Context, groupID string) ([]string, error) {
	res, err := g.getGroupMemberIDs(ctx, groupID)
	if err != nil {
		return nil, err
	}
	return res.UserIDs, nil
}

func (g *GroupLocalCache) GetGroupMemberIDMap(ctx context.Context, groupID string) (map[string]struct{}, error) {
	res, err := g.getGroupMemberIDs(ctx, groupID)
	if err != nil {
		return nil, err
	}
	return datautil.SliceSet(res.UserIDs), nil
}

func (g *GroupLocalCache) GetGroupInfos(ctx context.Context, groupIDs []string) ([]*sdkws.GroupInfo, error) {
	groupInfos := make([]*sdkws.GroupInfo, 0, len(groupIDs))
	for _, groupID := range groupIDs {
		groupInfo, err := g.GetGroupInfo(ctx, groupID)
		if err != nil {
			if errs.ErrRecordNotFound.Is(err) {
				continue
			}
			return nil, err
		}
		groupInfos = append(groupInfos, groupInfo)
```

```
	}
	return groupInfos, nil
}

func (g *GroupLocalCache) GetGroupMembers(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupMemb
	members := make([]*sdkws.GroupMemberFullInfo, 0, len(userIDs))
	for _, userID := range userIDs {
		member, err := g.GetGroupMember(ctx, groupID, userID)
		if err != nil {
			if errs.ErrRecordNotFound.Is(err) {
				continue
			}
			return nil, err
		}
		members = append(members, member)
	}
	return members, nil
}

func (g *GroupLocalCache) GetGroupMemberInfoMap(ctx context.Context, groupID string, userIDs []string) (map[string]*
	members := make(map[string]*sdkws.GroupMemberFullInfo)
	for _, userID := range userIDs {
		member, err := g.GetGroupMember(ctx, groupID, userID)
		if err != nil {
			if errs.ErrRecordNotFound.Is(err) {
				continue
			}
			return nil, err
		}
		members[userID] = member
	}
	return members, nil
}
```

pkg/rpccache/online.go

```
package rpccache

import (
	"context"
	"fmt"
	"math/rand"
	"strconv"
	"sync"
	"sync/atomic"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/user"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache/lru"
	"github.com/openimsdk/open-im-server/v3/pkg/util/useronline"
	"github.com/openimsdk/tools/db/cacheutil"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/redis/go-redis/v9"
)

const (
	Begin uint32 = iota
	DoOnlineStatusOver
	DoSubscribeOver
)

type OnlineCache interface {
	GetUserOnlinePlatform(ctx context.Context, userID string) ([]int32, error)
	GetUserOnline(ctx context.Context, userID string) (bool, error)
	GetUsersOnline(ctx context.Context, userIDs []string) ([]string, []string, error)
	WaitCache()
}

func NewOnlineCache(client *rpcli.UserClient, group *GroupLocalCache, rdb redis.UniversalClient, fullUserCache bool,
	if config.Standalone() {
		return disableOnlineCache{client: client}, nil
	}
	l := &sync.Mutex{}
	x := &defaultOnlineCache{
		client:        client,
		group:         group,
		fullUserCache: fullUserCache,
		Lock:          l,
		Cond:          sync.NewCond(l),
	}

	ctx := mcontext.SetOperationID(context.TODO(), strconv.FormatInt(time.Now().UnixNano()+int64(rand.Uint32()), 10))

	switch x.fullUserCache {
	case true:
		log.ZDebug(ctx, "fullUserCache is true")
		x.mapCache = cacheutil.NewCache[string, []int32]()
		go func() {
			if err := x.initUsersOnlineStatus(ctx); err != nil {
				log.ZError(ctx, "initUsersOnlineStatus failed", err)
			}
		}()
	case false:
```

```
    log.ZDebug(ctx, "fullUserCache is false")
    x.lruCache = lru.NewSlotLRU(1024, localcache.LRUStringHash, func() lru.LRU[string, []int32] {
        return lru.NewLazyLRU[string, []int32](2048, cachekey.OnlineExpire/2, time.Second*3, localcache.EmptyTarget{}, f
    })
    x.CurrentPhase.Store(DoSubscribeOver)
    x.Cond.Broadcast()
  }
  if rdb != nil {
    go func() {
      x.doSubscribe(ctx, rdb, fn)
    }()
  }
  return x, nil
}

type defaultOnlineCache struct {
  client *rpcli.UserClient
  group  *GroupLocalCache

  // fullUserCache if enabled, caches the online status of all users using mapCache;
  // otherwise, only a portion of users' online statuses (regardless of whether they are online) will be cached using
  fullUserCache bool

  lruCache lru.LRU[string, []int32]
  mapCache *cacheutil.Cache[string, []int32]

  Lock         *sync.Mutex
  Cond         *sync.Cond
  CurrentPhase atomic.Uint32
}

func (o *defaultOnlineCache) initUsersOnlineStatus(ctx context.Context) (err error) {
  log.ZDebug(ctx, "init users online status begin")

  var (
    totalSet      atomic.Int64
    maxTries      = 5
    retryInterval = time.Second * 5

    resp *user.GetAllOnlineUsersResp
  )

  defer func(t time.Time) {
    log.ZInfo(ctx, "init users online status end", "cost", time.Since(t), "totalSet", totalSet.Load())
    o.CurrentPhase.Store(DoOnlineStatusOver)
    o.Cond.Broadcast()
  }(time.Now())

  retryOperation := func(operation func() error, operationName string) error {
    for i := 0; i < maxTries; i++ {
      if err = operation(); err != nil {
        log.ZWarn(ctx, fmt.Sprintf("initUsersOnlineStatus: %s failed", operationName), err)
        time.Sleep(retryInterval)
      } else {
        return nil
      }
    }
    return err
  }

  cursor := uint64(0)
  for resp == nil || resp.NextCursor != 0 {
    if err = retryOperation(func() error {
      resp, err = o.client.GetAllOnlineUsers(ctx, cursor)
      if err != nil {
        return err
```

```
    }

    for _, u := range resp.StatusList {
        if u.Status == constant.Online {
            o.setUserOnline(u.UserID, u.PlatformIDs)
        }
        totalSet.Add(1)
    }
    cursor = resp.NextCursor
    return nil
  }, "getAllOnlineUsers"); err != nil {
    return err
  }
 }

 return nil
}

func (o *defaultOnlineCache) doSubscribe(ctx context.Context, rdb redis.UniversalClient, fn func(ctx context.Context
 o.Lock.Lock()
 ch := rdb.Subscribe(ctx, cachekey.OnlineChannel).Channel()
 for o.CurrentPhase.Load() < DoOnlineStatusOver {
  o.Cond.Wait()
 }
 o.Lock.Unlock()
 log.ZInfo(ctx, "begin doSubscribe")

 doMessage := func(message *redis.Message) {
  userID, platformIDs, err := useronline.ParseUserOnlineStatus(message.Payload)
  if err != nil {
    log.ZError(ctx, "OnlineCache setHasUserOnline redis subscribe parseUserOnlineStatus", err, "payload", message.Pa
    return
  }
  log.ZDebug(ctx, fmt.Sprintf("get subscribe %s message", cachekey.OnlineChannel), "useID", userID, "platformIDs", p
  switch o.fullUserCache {
  case true:
    if len(platformIDs) == 0 {
      // offline
      o.mapCache.Delete(userID)
    } else {
      o.mapCache.Store(userID, platformIDs)
    }
  case false:
    storageCache := o.setHasUserOnline(userID, platformIDs)
    log.ZDebug(ctx, "OnlineCache setHasUserOnline", "userID", userID, "platformIDs", platformIDs, "payload", message
    if fn != nil {
      fn(ctx, userID, platformIDs)
    }
  }
 }

 if o.CurrentPhase.Load() == DoOnlineStatusOver {
  for done := false; !done; {
    select {
    case message := <-ch:
      doMessage(message)
    default:
      o.CurrentPhase.Store(DoSubscribeOver)
      o.Cond.Broadcast()
      done = true
    }
  }
 }

 for message := range ch {
  doMessage(message)
```

```
	}
}

func (o *defaultOnlineCache) getUserOnlinePlatform(ctx context.Context, userID string) ([]int32, error) {
	platformIDs, err := o.lruCache.Get(userID, func() ([]int32, error) {
		return o.client.GetUserOnlinePlatform(ctx, userID)
	})
	if err != nil {
		log.ZError(ctx, "OnlineCache GetUserOnlinePlatform", err, "userID", userID)
		return nil, err
	}
	//log.ZDebug(ctx, "OnlineCache GetUserOnlinePlatform", "userID", userID, "platformIDs", platformIDs)
	return platformIDs, nil
}

func (o *defaultOnlineCache) GetUserOnlinePlatform(ctx context.Context, userID string) ([]int32, error) {
	platformIDs, err := o.getUserOnlinePlatform(ctx, userID)
	if err != nil {
		return nil, err
	}
	tmp := make([]int32, len(platformIDs))
	copy(tmp, platformIDs)
	return platformIDs, nil
}

func (o *defaultOnlineCache) GetUserOnline(ctx context.Context, userID string) (bool, error) {
	platformIDs, err := o.getUserOnlinePlatform(ctx, userID)
	if err != nil {
		return false, err
	}
	return len(platformIDs) > 0, nil
}

func (o *defaultOnlineCache) getUserOnlinePlatformBatch(ctx context.Context, userIDs []string) (map[string][]int32,
	if len(userIDs) == 0 {
		return nil, nil
	}
	platformIDsMap, err := o.lruCache.GetBatch(userIDs, func(missingUsers []string) (map[string][]int32, error) {
		platformIDsMap := make(map[string][]int32)
		usersStatus, err := o.client.GetUsersOnlinePlatform(ctx, missingUsers)
		if err != nil {
			return nil, err
		}

		for _, u := range usersStatus {
			platformIDsMap[u.UserID] = u.PlatformIDs
		}

		return platformIDsMap, nil
	})
	if err != nil {
		log.ZError(ctx, "OnlineCache GetUserOnlinePlatform", err, "userID", userIDs)
		return nil, err
	}
	return platformIDsMap, nil
}

func (o *defaultOnlineCache) GetUsersOnline(ctx context.Context, userIDs []string) ([]string, []string, error) {
	t := time.Now()

	var (
		onlineUserIDs  = make([]string, 0, len(userIDs))
		offlineUserIDs = make([]string, 0, len(userIDs))
	)

	switch o.fullUserCache {
```

```
	case true:
		for _, userID := range userIDs {
			if _, ok := o.mapCache.Load(userID); ok {
				onlineUserIDs = append(onlineUserIDs, userID)
			} else {
				offlineUserIDs = append(offlineUserIDs, userID)
			}
		}
	case false:
		userOnlineMap, err := o.getUserOnlinePlatformBatch(ctx, userIDs)
		if err != nil {
			return nil, nil, err
		}

		for key, value := range userOnlineMap {
			if len(value) > 0 {
				onlineUserIDs = append(onlineUserIDs, key)
			} else {
				offlineUserIDs = append(offlineUserIDs, key)
			}
		}
	}

	log.ZInfo(ctx, "get users online", "online users length", len(onlineUserIDs), "offline users length", len(offlineUs
	return onlineUserIDs, offlineUserIDs, nil
}

func (o *defaultOnlineCache) setUserOnline(userID string, platformIDs []int32) {
	switch o.fullUserCache {
	case true:
		o.mapCache.Store(userID, platformIDs)
	case false:
		o.lruCache.Set(userID, platformIDs)
	}
}

func (o *defaultOnlineCache) setHasUserOnline(userID string, platformIDs []int32) bool {
	return o.lruCache.SetHas(userID, platformIDs)
}

func (o *defaultOnlineCache) WaitCache() {
	o.Lock.Lock()
	for o.CurrentPhase.Load() < DoSubscribeOver {
		o.Cond.Wait()
	}
	o.Lock.Unlock()
}

type disableOnlineCache struct {
	client *rpcli.UserClient
}

func (o disableOnlineCache) GetUserOnlinePlatform(ctx context.Context, userID string) ([]int32, error) {
	return o.client.GetUserOnlinePlatform(ctx, userID)
}

func (o disableOnlineCache) GetUserOnline(ctx context.Context, userID string) (bool, error) {
	onlinePlatform, err := o.client.GetUserOnlinePlatform(ctx, userID)
	if err != nil {
		return false, err
	}
	return len(onlinePlatform) > 0, err
}

func (o disableOnlineCache) GetUsersOnline(ctx context.Context, userIDs []string) ([]string, []string, error) {
	var (
```

```
    onlineUserIDs  = make([]string, 0, len(userIDs))
    offlineUserIDs = make([]string, 0, len(userIDs))
  )
  for _, userID := range userIDs {
    online, err := o.GetUserOnline(ctx, userID)
    if err != nil {
      return nil, nil, err
    }
    if online {
      onlineUserIDs = append(onlineUserIDs, userID)
    } else {
      offlineUserIDs = append(offlineUserIDs, userID)
    }
  }
  return onlineUserIDs, offlineUserIDs, nil
}

func (o disableOnlineCache) WaitCache() {}
```

pkg/rpccache/subscriber.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache

import (
	"context"
	"encoding/json"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

func subscriberRedisDeleteCache(ctx context.Context, client redis.UniversalClient, channel string, del func(ctx cont
	defer func() {
		if r := recover(); r != nil {
			log.ZPanic(ctx, "subscriberRedisDeleteCache Panic", errs.ErrPanic(r))
		}
	}()
	for message := range client.Subscribe(ctx, channel).Channel() {
		log.ZDebug(ctx, "subscriberRedisDeleteCache", "channel", channel, "payload", message.Payload)
		var keys []string
		if err := json.Unmarshal([]byte(message.Payload), &keys); err != nil {
			log.ZError(ctx, "subscriberRedisDeleteCache json.Unmarshal error", err)
			continue
		}
		if len(keys) == 0 {
			continue
		}
		del(ctx, keys...)
	}
}
```

pkg/rpccache/user.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package rpccache

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

func NewUserLocalCache(client *rpcli.UserClient, localCache *config.LocalCache, cli redis.UniversalClient) *UserLoca
	lc := localCache.User
	log.ZDebug(context.Background(), "UserLocalCache", "topic", lc.Topic, "slotNum", lc.SlotNum, "slotSize", lc.SlotSiz
	x := &UserLocalCache{
		client: client,
		local: localcache.New[[]byte](
			localcache.WithLocalSlotNum(lc.SlotNum),
			localcache.WithLocalSlotSize(lc.SlotSize),
			localcache.WithLinkSlotNum(lc.SlotNum),
			localcache.WithLocalSuccessTTL(lc.Success()),
			localcache.WithLocalFailedTTL(lc.Failed()),
		),
	}
	if lc.Enable() {
		go subscriberRedisDeleteCache(context.Background(), cli, lc.Topic, x.local.DelLocal)
	}
	return x
}

type UserLocalCache struct {
	client *rpcli.UserClient
	local  localcache.Cache[[]byte]
}

func (u *UserLocalCache) GetUserInfo(ctx context.Context, userID string) (val *sdkws.UserInfo, err error) {
	log.ZDebug(ctx, "UserLocalCache GetUserInfo req", "userID", userID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "UserLocalCache GetUserInfo return", "value", val)
		} else {
			log.ZError(ctx, "UserLocalCache GetUserInfo return", err)
		}
	}()
	var cache cacheProto[sdkws.UserInfo]
```

```
	return cache.Unmarshal(u.local.Get(ctx, cachekey.GetUserInfoKey(userID), func(ctx context.Context) ([]byte, error)
		log.ZDebug(ctx, "UserLocalCache GetUserInfo rpc", "userID", userID)
		return cache.Marshal(u.client.GetUserInfo(ctx, userID))
	}))
}

func (u *UserLocalCache) GetUserGlobalMsgRecvOpt(ctx context.Context, userID string) (val int32, err error) {
	resp, err := u.getUserGlobalMsgRecvOpt(ctx, userID)
	if err != nil {
		return 0, err
	}
	return resp.GlobalRecvMsgOpt, nil
}

func (u *UserLocalCache) getUserGlobalMsgRecvOpt(ctx context.Context, userID string) (val *user.GetGlobalRecvMessage
	log.ZDebug(ctx, "UserLocalCache getUserGlobalMsgRecvOpt req", "userID", userID)
	defer func() {
		if err == nil {
			log.ZDebug(ctx, "UserLocalCache getUserGlobalMsgRecvOpt return", "value", val)
		} else {
			log.ZError(ctx, "UserLocalCache getUserGlobalMsgRecvOpt return", err)
		}
	}()
	var cache cacheProto[user.GetGlobalRecvMessageOptResp]
	return cache.Unmarshal(u.local.Get(ctx, cachekey.GetUserGlobalRecvMsgOptKey(userID), func(ctx context.Context) ([]b
		log.ZDebug(ctx, "UserLocalCache GetUserGlobalMsgRecvOpt rpc", "userID", userID)
		return cache.Marshal(u.client.UserClient.GetGlobalRecvMessageOpt(ctx, &user.GetGlobalRecvMessageOptReq{UserID: us
	}))
}

func (u *UserLocalCache) GetUsersInfo(ctx context.Context, userIDs []string) ([]*sdkws.UserInfo, error) {
	users := make([]*sdkws.UserInfo, 0, len(userIDs))
	for _, userID := range userIDs {
		user, err := u.GetUserInfo(ctx, userID)
		if err != nil {
			if errs.ErrRecordNotFound.Is(err) {
				log.ZWarn(ctx, "User info notFound", err, "userID", userID)
				continue
			}
			return nil, err
		}
		users = append(users, user)
	}
	return users, nil
}

func (u *UserLocalCache) GetUsersInfoMap(ctx context.Context, userIDs []string) (map[string]*sdkws.UserInfo, error)
	users := make(map[string]*sdkws.UserInfo, len(userIDs))
	for _, userID := range userIDs {
		user, err := u.GetUserInfo(ctx, userID)
		if err != nil {
			if errs.ErrRecordNotFound.Is(err) {
				continue
			}
			return nil, err
		}
		users[userID] = user
	}
	return users, nil
}
```

pkg/mqbuild

pkg/mqbuild/builder.go

```
package mqbuild

import (
	"context"
	"fmt"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/mq"
	"github.com/openimsdk/tools/mq/kafka"
	"github.com/openimsdk/tools/mq/simmq"
)

type Builder interface {
	GetTopicProducer(ctx context.Context, topic string) (mq.Producer, error)
	GetTopicConsumer(ctx context.Context, topic string) (mq.Consumer, error)
}

func NewBuilder(kafka *config.Kafka) Builder {
	if config.Standalone() {
		return standaloneBuilder{}
	}
	return &kafkaBuilder{
		addr:   kafka.Address,
		config: kafka.Build(),
		topicGroupID: map[string]string{
			kafka.ToRedisTopic:       kafka.ToRedisGroupID,
			kafka.ToMongoTopic:       kafka.ToMongoGroupID,
			kafka.ToPushTopic:        kafka.ToPushGroupID,
			kafka.ToOfflinePushTopic: kafka.ToOfflineGroupID,
		},
	}
}

type standaloneBuilder struct{}

func (standaloneBuilder) GetTopicProducer(ctx context.Context, topic string) (mq.Producer, error) {
	return simmq.GetTopicProducer(topic), nil
}

func (standaloneBuilder) GetTopicConsumer(ctx context.Context, topic string) (mq.Consumer, error) {
	return simmq.GetTopicConsumer(topic), nil
}

type kafkaBuilder struct {
	addr         []string
	config       *kafka.Config
	topicGroupID map[string]string
}

func (x *kafkaBuilder) GetTopicProducer(ctx context.Context, topic string) (mq.Producer, error) {
	return kafka.NewKafkaProducerV2(x.config, x.addr, topic)
}

func (x *kafkaBuilder) GetTopicConsumer(ctx context.Context, topic string) (mq.Consumer, error) {
	groupID, ok := x.topicGroupID[topic]
	if !ok {
		return nil, fmt.Errorf("topic %s groupID not found", topic)
	}
	return kafka.NewMConsumerGroupV2(ctx, x.config, groupID, []string{topic}, true)
}
```

pkg/common

pkg/common/convert

pkg/common/convert/auth.go

```go
package convert

func TokenMapDB2Pb(tokenMapDB map[string]int) map[string]int32 {
    if tokenMapDB == nil {
        return nil
    }

    tokenMapPB := make(map[string]int32, len(tokenMapDB))
    for k, v := range tokenMapDB {
        tokenMapPB[k] = int32(v)
    }
    return tokenMapPB
}

func TokenMapPb2DB(tokenMapPB map[string]int32) map[string]int {
    if tokenMapPB == nil {
        return nil
    }

    tokenMapDB := make(map[string]int, len(tokenMapPB))
    for k, v := range tokenMapPB {
        tokenMapDB[k] = int(v)
    }
    return tokenMapDB
}
```

pkg/common/convert/black.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/openimsdk/protocol/sdkws"
	sdk "github.com/openimsdk/protocol/sdkws"
)

func BlackDB2Pb(ctx context.Context, blackDBs []*model.Black, f func(ctx context.Context, userIDs []string) (map[str
	if len(blackDBs) == 0 {
		return nil, nil
	}
	var userIDs []string
	for _, blackDB := range blackDBs {
		userIDs = append(userIDs, blackDB.BlockUserID)
	}
	userInfos, err := f(ctx, userIDs)
	if err != nil {
		return nil, err
	}
	for _, blackDB := range blackDBs {
		blackPb := &sdk.BlackInfo{
			OwnerUserID:    blackDB.OwnerUserID,
			CreateTime:     blackDB.CreateTime.Unix(),
			AddSource:      blackDB.AddSource,
			Ex:             blackDB.Ex,
			OperatorUserID: blackDB.OperatorUserID,
			BlackUserInfo: &sdkws.PublicUserInfo{
				UserID:   userInfos[blackDB.BlockUserID].UserID,
				Nickname: userInfos[blackDB.BlockUserID].Nickname,
				FaceURL:  userInfos[blackDB.BlockUserID].FaceURL,
				Ex:       userInfos[blackDB.BlockUserID].Ex,
			},
		}
		blackPbs = append(blackPbs, blackPb)
	}
	return blackPbs, nil
}
```

pkg/common/convert/conversation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/tools/utils/datautil"
)

func ConversationDB2Pb(conversationDB *model.Conversation) *conversation.Conversation {
	conversationPB := &conversation.Conversation{}
	conversationPB.LatestMsgDestructTime = conversationDB.LatestMsgDestructTime.UnixMilli()
	if err := datautil.CopyStructFields(conversationPB, conversationDB); err != nil {
		return nil
	}
	return conversationPB
}

func ConversationsDB2Pb(conversationsDB []*model.Conversation) (conversationsPB []*conversation.Conversation) {
	for _, conversationDB := range conversationsDB {
		conversationPB := &conversation.Conversation{}
		if err := datautil.CopyStructFields(conversationPB, conversationDB); err != nil {
			continue
		}
		conversationPB.LatestMsgDestructTime = conversationDB.LatestMsgDestructTime.UnixMilli()
		conversationsPB = append(conversationsPB, conversationPB)
	}
	return conversationsPB
}

func ConversationPb2DB(conversationPB *conversation.Conversation) *model.Conversation {
	conversationDB := &model.Conversation{}
	if err := datautil.CopyStructFields(conversationDB, conversationPB); err != nil {
		return nil
	}
	return conversationDB
}

func ConversationsPb2DB(conversationsPB []*conversation.Conversation) (conversationsDB []*model.Conversation) {
	for _, conversationPB := range conversationsPB {
		conversationDB := &model.Conversation{}
		if err := datautil.CopyStructFields(conversationDB, conversationPB); err != nil {
			continue
		}
		conversationsDB = append(conversationsDB, conversationDB)
	}
	return conversationsDB
}
```

pkg/common/convert/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert // import "github.com/openimsdk/open-im-server/v3/pkg/common/convert"
```

pkg/common/convert/friend.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
■"context"
■"fmt"

■"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
■"github.com/openimsdk/open-im-server/v3/pkg/notification/common_user"
■"github.com/openimsdk/protocol/relation"

■"github.com/openimsdk/protocol/sdkws"
■"github.com/openimsdk/tools/utils/datautil"
■"github.com/openimsdk/tools/utils/timeutil"
)

func FriendPb2DB(friend *sdkws.FriendInfo) *model.Friend {
■dbFriend := &model.Friend{}
■err := datautil.CopyStructFields(dbFriend, friend)
■if err != nil {
■■return nil
■}
■dbFriend.FriendUserID = friend.FriendUser.UserID
■dbFriend.CreateTime = timeutil.UnixSecondToTime(friend.CreateTime)
■return dbFriend
}

func FriendDB2Pb(ctx context.Context, friendDB *model.Friend, getUsers func(ctx context.Context, userIDs []string) (
■users, err := getUsers(ctx, []string{friendDB.FriendUserID})
■if err != nil {
■■return nil, err
■}
■user, ok := users[friendDB.FriendUserID]
■if !ok {
■■return nil, fmt.Errorf("user not found: %s", friendDB.FriendUserID)
■}

■return &sdkws.FriendInfo{
■■FriendUser: user,
■■CreateTime: friendDB.CreateTime.Unix(),
■}, nil
}

func FriendsDB2Pb(ctx context.Context, friendsDB []*model.Friend, getUsers func(ctx context.Context, userIDs []strin
■if len(friendsDB) == 0 {
■■return nil, nil
■}
■var userID []string
■for _, friendDB := range friendsDB {
■■userID = append(userID, friendDB.FriendUserID)
■}
```

```
	users, err := getUsers(ctx, userID)
	if err != nil {
		return nil, err
	}
	for _, friend := range friendsDB {
		friendPb := &sdkws.FriendInfo{FriendUser: &sdkws.UserInfo{}}
		err := datautil.CopyStructFields(friendPb, friend)
		if err != nil {
			return nil, err
		}

		friendPb.FriendUser.UserID = users[friend.FriendUserID].UserID
		friendPb.FriendUser.Nickname = users[friend.FriendUserID].Nickname
		friendPb.FriendUser.FaceURL = users[friend.FriendUserID].FaceURL
		friendPb.FriendUser.Ex = users[friend.FriendUserID].Ex
		friendPb.CreateTime = friend.CreateTime.Unix()
		friendPb.IsPinned = friend.IsPinned
		friendsPb = append(friendsPb, friendPb)
	}
	return friendsPb, nil
}

func FriendOnlyDB2PbOnly(friendsDB []*model.Friend) []*relation.FriendInfoOnly {
	return datautil.Slice(friendsDB, func(f *model.Friend) *relation.FriendInfoOnly {
		return &relation.FriendInfoOnly{
			OwnerUserID:    f.OwnerUserID,
			FriendUserID:   f.FriendUserID,
			Remark:         f.Remark,
			CreateTime:     f.CreateTime.UnixMilli(),
			AddSource:      f.AddSource,
			OperatorUserID: f.OperatorUserID,
			Ex:             f.Ex,
			IsPinned:       f.IsPinned,
		}
	})
}

func FriendRequestDB2Pb(ctx context.Context, friendRequests []*model.FriendRequest, getUsers func(ctx context.Contex
	if len(friendRequests) == 0 {
		return nil, nil
	}
	userIDMap := make(map[string]struct{})
	for _, friendRequest := range friendRequests {
		userIDMap[friendRequest.ToUserID] = struct{}{}
		userIDMap[friendRequest.FromUserID] = struct{}{}
	}
	users, err := getUsers(ctx, datautil.Keys(userIDMap))
	if err != nil {
		return nil, err
	}
	res := make([]*sdkws.FriendRequest, 0, len(friendRequests))
	for _, friendRequest := range friendRequests {
		toUser := users[friendRequest.ToUserID]
		fromUser := users[friendRequest.FromUserID]
		res = append(res, &sdkws.FriendRequest{
			FromUserID:    friendRequest.FromUserID,
			FromNickname:  fromUser.GetNickname(),
			FromFaceURL:   fromUser.GetFaceURL(),
			ToUserID:      friendRequest.ToUserID,
			ToNickname:    toUser.GetNickname(),
			ToFaceURL:     toUser.GetFaceURL(),
			HandleResult:  friendRequest.HandleResult,
			ReqMsg:        friendRequest.ReqMsg,
			CreateTime:    friendRequest.CreateTime.UnixMilli(),
			HandlerUserID: friendRequest.HandlerUserID,
```

```
		HandleMsg:     friendRequest.HandleMsg,
		HandleTime:    friendRequest.HandleTime.UnixMilli(),
		Ex:            friendRequest.Ex,
	})
	}
	return res, nil
}

// FriendPb2DBMap converts a FriendInfo protobuf object to a map suitable for database operations.
// It only includes non-zero or non-empty fields in the map.
func FriendPb2DBMap(friend *sdkws.FriendInfo) map[string]any {
	if friend == nil {
		return nil
	}

	val := make(map[string]any)

	// Assuming FriendInfo has similar fields to those in Friend.
	// Add or remove fields based on your actual FriendInfo and Friend structures.
	if friend.FriendUser != nil {
		if friend.FriendUser.UserID != "" {
			val["friend_user_id"] = friend.FriendUser.UserID
		}
		if friend.FriendUser.Nickname != "" {
			val["nickname"] = friend.FriendUser.Nickname
		}
		if friend.FriendUser.FaceURL != "" {
			val["face_url"] = friend.FriendUser.FaceURL
		}
		if friend.FriendUser.Ex != "" {
			val["ex"] = friend.FriendUser.Ex
		}
	}
	if friend.CreateTime != 0 {
		val["create_time"] = friend.CreateTime // You might need to convert this to a proper time format.
	}

	// Include other fields from FriendInfo as needed, similar to the above pattern.

	return val
}
```

pkg/common/convert/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"time"

	pbgroup "github.com/openimsdk/protocol/group"
	sdkws "github.com/openimsdk/protocol/sdkws"
)

func Db2PbGroupInfo(m *model.Group, ownerUserID string, memberCount uint32) *sdkws.GroupInfo {
	return &sdkws.GroupInfo{
		GroupID:                m.GroupID,
		GroupName:              m.GroupName,
		Notification:           m.Notification,
		Introduction:           m.Introduction,
		FaceURL:                m.FaceURL,
		OwnerUserID:            ownerUserID,
		CreateTime:             m.CreateTime.UnixMilli(),
		MemberCount:            memberCount,
		Ex:                     m.Ex,
		Status:                 m.Status,
		CreatorUserID:          m.CreatorUserID,
		GroupType:              m.GroupType,
		NeedVerification:       m.NeedVerification,
		LookMemberInfo:         m.LookMemberInfo,
		ApplyMemberFriend:      m.ApplyMemberFriend,
		NotificationUpdateTime: m.NotificationUpdateTime.UnixMilli(),
		NotificationUserID:     m.NotificationUserID,
	}
}

func Pb2DbGroupRequest(req *pbgroup.GroupApplicationResponseReq, handleUserID string) *model.GroupRequest {
	return &model.GroupRequest{
		UserID:       req.FromUserID,
		GroupID:      req.GroupID,
		HandleResult: req.HandleResult,
		HandledMsg:   req.HandledMsg,
		HandleUserID: handleUserID,
		HandledTime:  time.Now(),
	}
}

func Db2PbCMSGroup(m *model.Group, ownerUserID string, ownerUserName string, memberCount uint32) *pbgroup.CMSGroup {
	return &pbgroup.CMSGroup{
		GroupInfo:          Db2PbGroupInfo(m, ownerUserID, memberCount),
		GroupOwnerUserID:   ownerUserID,
		GroupOwnerUserName: ownerUserName,
	}
}
```

```
func Db2PbGroupMember(m *model.GroupMember) *sdkws.GroupMemberFullInfo {
	return &sdkws.GroupMemberFullInfo{
		GroupID:   m.GroupID,
		UserID:    m.UserID,
		RoleLevel: m.RoleLevel,
		JoinTime:  m.JoinTime.UnixMilli(),
		Nickname:  m.Nickname,
		FaceURL:   m.FaceURL,
		// AppMangerLevel: m.AppMangerLevel,
		JoinSource:     m.JoinSource,
		OperatorUserID: m.OperatorUserID,
		Ex:             m.Ex,
		MuteEndTime:    m.MuteEndTime.UnixMilli(),
		InviterUserID:  m.InviterUserID,
	}
}

func Db2PbGroupRequest(m *model.GroupRequest, user *sdkws.UserInfo, group *sdkws.GroupInfo) *sdkws.GroupRequest {
	var pu *sdkws.PublicUserInfo
	if user != nil {
		pu = &sdkws.PublicUserInfo{
			UserID:   user.UserID,
			Nickname: user.Nickname,
			FaceURL:  user.FaceURL,
			Ex:       user.Ex,
		}
	}
	return &sdkws.GroupRequest{
		UserInfo:      pu,
		GroupInfo:     group,
		HandleResult:  m.HandleResult,
		ReqMsg:        m.ReqMsg,
		HandleMsg:     m.HandledMsg,
		ReqTime:       m.ReqTime.UnixMilli(),
		HandleUserID:  m.HandleUserID,
		HandleTime:    m.HandledTime.UnixMilli(),
		Ex:            m.Ex,
		JoinSource:    m.JoinSource,
		InviterUserID: m.InviterUserID,
	}
}

func Db2PbGroupAbstractInfo(
	groupID string,
	groupMemberNumber uint32,
	groupMemberListHash uint64,
) *pbgroup.GroupAbstractInfo {
	return &pbgroup.GroupAbstractInfo{
		GroupID:             groupID,
		GroupMemberNumber:   groupMemberNumber,
		GroupMemberListHash: groupMemberListHash,
	}
}

func Pb2DBGroupInfo(m *sdkws.GroupInfo) *model.Group {
	return &model.Group{
		GroupID:                m.GroupID,
		GroupName:              m.GroupName,
		Notification:           m.Notification,
		Introduction:           m.Introduction,
		FaceURL:                m.FaceURL,
		CreateTime:             time.Now(),
		Ex:                     m.Ex,
		Status:                 m.Status,
		CreatorUserID:          m.CreatorUserID,
```

```
    GroupType:              m.GroupType,
    NeedVerification:       m.NeedVerification,
    LookMemberInfo:         m.LookMemberInfo,
    ApplyMemberFriend:      m.ApplyMemberFriend,
    NotificationUpdateTime: time.UnixMilli(m.NotificationUpdateTime),
    NotificationUserID:     m.NotificationUserID,
  }
}

// func Pb2DbGroupMember(m *sdkws.UserInfo) *relation.GroupMember {
//  return &relation.GroupMember{
//    UserID:   m.UserID,
//    Nickname: m.Nickname,
//    FaceURL:  m.FaceURL,
//    Ex:       m.Ex,
//  }
//}
```

pkg/common/convert/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/sdkws"
)

func MsgPb2DB(msg *sdkws.MsgData) *model.MsgDataModel {
    if msg == nil {
        return nil
    }
    var msgDataModel model.MsgDataModel
    msgDataModel.SendID = msg.SendID
    msgDataModel.RecvID = msg.RecvID
    msgDataModel.GroupID = msg.GroupID
    msgDataModel.ClientMsgID = msg.ClientMsgID
    msgDataModel.ServerMsgID = msg.ServerMsgID
    msgDataModel.SenderPlatformID = msg.SenderPlatformID
    msgDataModel.SenderNickname = msg.SenderNickname
    msgDataModel.SenderFaceURL = msg.SenderFaceURL
    msgDataModel.SessionType = msg.SessionType
    msgDataModel.MsgFrom = msg.MsgFrom
    msgDataModel.ContentType = msg.ContentType
    msgDataModel.Content = string(msg.Content)
    msgDataModel.Seq = msg.Seq
    msgDataModel.SendTime = msg.SendTime
    msgDataModel.CreateTime = msg.CreateTime
    msgDataModel.Status = msg.Status
    msgDataModel.Options = msg.Options
    if msg.OfflinePushInfo != nil {
        msgDataModel.OfflinePush = &model.OfflinePushModel{
            Title:         msg.OfflinePushInfo.Title,
            Desc:          msg.OfflinePushInfo.Desc,
            Ex:            msg.OfflinePushInfo.Ex,
            IOSPushSound:  msg.OfflinePushInfo.IOSPushSound,
            IOSBadgeCount: msg.OfflinePushInfo.IOSBadgeCount,
        }
    }
    msgDataModel.AtUserIDList = msg.AtUserIDList
    msgDataModel.AttachedInfo = msg.AttachedInfo
    msgDataModel.Ex = msg.Ex
    return &msgDataModel
}

func MsgDB2Pb(msgModel *model.MsgDataModel) *sdkws.MsgData {
    if msgModel == nil {
        return nil
    }
    var msg sdkws.MsgData
```

```
    msg.SendID = msgModel.SendID
    msg.RecvID = msgModel.RecvID
    msg.GroupID = msgModel.GroupID
    msg.ClientMsgID = msgModel.ClientMsgID
    msg.ServerMsgID = msgModel.ServerMsgID
    msg.SenderPlatformID = msgModel.SenderPlatformID
    msg.SenderNickname = msgModel.SenderNickname
    msg.SenderFaceURL = msgModel.SenderFaceURL
    msg.SessionType = msgModel.SessionType
    msg.MsgFrom = msgModel.MsgFrom
    msg.ContentType = msgModel.ContentType
    msg.Content = []byte(msgModel.Content)
    msg.Seq = msgModel.Seq
    msg.SendTime = msgModel.SendTime
    msg.CreateTime = msgModel.CreateTime
    msg.Status = msgModel.Status
    if msgModel.SessionType == constant.SingleChatType {
        msg.IsRead = msgModel.IsRead
    }
    msg.Options = msgModel.Options
    if msgModel.OfflinePush != nil {
        msg.OfflinePushInfo = &sdkws.OfflinePushInfo{
            Title:         msgModel.OfflinePush.Title,
            Desc:          msgModel.OfflinePush.Desc,
            Ex:            msgModel.OfflinePush.Ex,
            IOSPushSound:  msgModel.OfflinePush.IOSPushSound,
            IOSBadgeCount: msgModel.OfflinePush.IOSBadgeCount,
        }
    }
    msg.AtUserIDList = msgModel.AtUserIDList
    msg.AttachedInfo = msgModel.AttachedInfo
    msg.Ex = msgModel.Ex
    return &msg
}
```

pkg/common/convert/user.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/utils/datautil"
	"time"

	"github.com/openimsdk/protocol/sdkws"
)

func UserDB2Pb(user *relationtb.User) *sdkws.UserInfo {
	return &sdkws.UserInfo{
		UserID:           user.UserID,
		Nickname:         user.Nickname,
		FaceURL:          user.FaceURL,
		Ex:               user.Ex,
		CreateTime:       user.CreateTime.UnixMilli(),
		AppMangerLevel:   user.AppMangerLevel,
		GlobalRecvMsgOpt: user.GlobalRecvMsgOpt,
	}
}

func UsersDB2Pb(users []*relationtb.User) []*sdkws.UserInfo {
	return datautil.Slice(users, UserDB2Pb)
}

func UserPb2DB(user *sdkws.UserInfo) *relationtb.User {
	return &relationtb.User{
		UserID:           user.UserID,
		Nickname:         user.Nickname,
		FaceURL:          user.FaceURL,
		Ex:               user.Ex,
		CreateTime:       time.UnixMilli(user.CreateTime),
		AppMangerLevel:   user.AppMangerLevel,
		GlobalRecvMsgOpt: user.GlobalRecvMsgOpt,
	}
}

func UserPb2DBMap(user *sdkws.UserInfo) map[string]any {
	if user == nil {
		return nil
	}
	val := make(map[string]any)
	fields := map[string]any{
		"nickname":            user.Nickname,
		"face_url":            user.FaceURL,
		"ex":                  user.Ex,
		"app_manager_level":   user.AppMangerLevel,
		"global_recv_msg_opt": user.GlobalRecvMsgOpt,
	}
```

```
	for key, value := range fields {
		if v, ok := value.(string); ok && v != "" {
			val[key] = v
		} else if v, ok := value.(int32); ok && v != 0 {
			val[key] = v
		}
	}
	return val
}
func UserPb2DBMapEx(user *sdkws.UserInfoWithEx) map[string]any {
	if user == nil {
		return nil
	}
	val := make(map[string]any)

	// Map fields from UserInfoWithEx to val
	if user.Nickname != nil {
		val["nickname"] = user.Nickname.Value
	}
	if user.FaceURL != nil {
		val["face_url"] = user.FaceURL.Value
	}
	if user.Ex != nil {
		val["ex"] = user.Ex.Value
	}
	if user.GlobalRecvMsgOpt != nil {
		val["global_recv_msg_opt"] = user.GlobalRecvMsgOpt.Value
	}

	return val
}
```

pkg/common/convert/user_test.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package convert

import (
	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"reflect"
	"testing"

	"github.com/openimsdk/protocol/sdkws"
)

func TestUsersDB2Pb(t *testing.T) {
	type args struct {
		users []*relationtb.User
	}
	tests := []struct {
		name       string
		args       args
		wantResult []*sdkws.UserInfo
	}{
		// TODO: Add test cases.
	}
	for _, tt := range tests {
		t.Run(tt.name, func(t *testing.T) {
			if gotResult := UsersDB2Pb(tt.args.users); !reflect.DeepEqual(gotResult, tt.wantResult) {
				t.Errorf("UsersDB2Pb() = %v, want %v", gotResult, tt.wantResult)
			}
		})
	}
}

func TestUserPb2DB(t *testing.T) {
	type args struct {
		user *sdkws.UserInfo
	}
	tests := []struct {
		name string
		args args
		want *relationtb.User
	}{
		// TODO: Add test cases.
	}
	for _, tt := range tests {
		t.Run(tt.name, func(t *testing.T) {
			if got := UserPb2DB(tt.args.user); !reflect.DeepEqual(got, tt.want) {
				t.Errorf("UserPb2DB() = %v, want %v", got, tt.want)
			}
		})
	}
}
```

```go
func TestUserPb2DBMap(t *testing.T) {
	user := &sdkws.UserInfo{
		Nickname:         "TestUser",
		FaceURL:          "http://openim.io/logo.jpg",
		Ex:               "Extra Data",
		AppMangerLevel:   1,
		GlobalRecvMsgOpt: 2,
	}

	expected := map[string]any{
		"nickname":            "TestUser",
		"face_url":            "http://openim.io/logo.jpg",
		"ex":                  "Extra Data",
		"app_manager_level":   int32(1),
		"global_recv_msg_opt": int32(2),
	}

	result := UserPb2DBMap(user)
	if !reflect.DeepEqual(result, expected) {
		t.Errorf("UserPb2DBMap returned unexpected map. Got %v, want %v", result, expected)
	}
}
```

pkg/common/discovery

pkg/common/discovery/discoveryregister.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package discovery

import (
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/discovery/standalone"
	"github.com/openimsdk/tools/utils/runtimeenv"
	"google.golang.org/grpc"

	"github.com/openimsdk/tools/discovery/kubernetes"

	"github.com/openimsdk/tools/discovery/etcd"
	"github.com/openimsdk/tools/errs"
)

// NewDiscoveryRegister creates a new service discovery and registry client based on the provided environment type.
func NewDiscoveryRegister(discovery *config.Discovery, watchNames []string) (discovery.SvcDiscoveryRegistry, error) {
	if config.Standalone() {
		return standalone.GetSvcDiscoveryRegistry(), nil
	}
	if runtimeenv.RuntimeEnvironment() == config.KUBERNETES {
		return kubernetes.NewConnManager(discovery.Kubernetes.Namespace, nil,
			grpc.WithDefaultCallOptions(
				grpc.MaxCallSendMsgSize(1024*1024*20),
			),
		)
	}

	switch discovery.Enable {
	case config.ETCD:
		return etcd.NewSvcDiscoveryRegistry(
			discovery.Etcd.RootDirectory,
			discovery.Etcd.Address,
			watchNames,
			etcd.WithDialTimeout(10*time.Second),
			etcd.WithMaxCallSendMsgSize(20*1024*1024),
			etcd.WithUsernameAndPassword(discovery.Etcd.Username, discovery.Etcd.Password))
	default:
		return nil, errs.New("unsupported discovery type", "type", discovery.Enable).Wrap()
	}
}
```

pkg/common/discovery/discoveryregister_test.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package discovery

import (
	"os"
)

func setupTestEnvironment() {
	os.Setenv("ZOOKEEPER_SCHEMA", "openim")
	os.Setenv("ZOOKEEPER_ADDRESS", "172.28.0.1")
	os.Setenv("ZOOKEEPER_PORT", "12181")
	os.Setenv("ZOOKEEPER_USERNAME", "")
	os.Setenv("ZOOKEEPER_PASSWORD", "")
}

//func TestNewDiscoveryRegister(t *testing.T) {
//	setupTestEnvironment()
//	conf := config.NewGlobalConfig()
//	tests := []struct {
//		envType        string
//		gatewayName    string
//		expectedError  bool
//		expectedResult bool
//	}{
//		{"zookeeper", "MessageGateway", false, true},
//		{"k8s", "MessageGateway", false, true},
//		{"direct", "MessageGateway", false, true},
//		{"invalid", "MessageGateway", true, false},
//	}
//
//	for _, test := range tests {
//		conf.Envs.Discovery = test.envType
//		conf.RpcRegisterName.OpenImMessageGatewayName = test.gatewayName
//		client, err := NewDiscoveryRegister(conf)
//
//		if test.expectedError {
//			assert.Error(t, err)
//		} else {
//			assert.NoError(t, err)
//			if test.expectedResult {
//				assert.Implements(t, (*discovery.SvcDiscoveryRegistry)(nil), client)
//			} else {
//				assert.Nil(t, client)
//			}
//		}
//	}
//}
```

pkg/common/discovery/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package discovery // import "github.com/openimsdk/open-im-server/v3/pkg/common/discovery"
```

pkg/common/discovery/direct

pkg/common/discovery/direct/direct_resolver.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package direct

import (
	"context"
	"math/rand"
	"strings"

	"github.com/openimsdk/tools/log"
	"google.golang.org/grpc/resolver"
)

const (
	slashSeparator = "/"
	// EndpointSepChar is the separator char in endpoints.
	EndpointSepChar = ','

	subsetSize = 32
	scheme     = "direct"
)

type ResolverDirect struct {
}

func NewResolverDirect() *ResolverDirect {
	return &ResolverDirect{}
}

func (rd *ResolverDirect) Build(target resolver.Target, cc resolver.ClientConn, _ resolver.BuildOptions) (
	resolver.Resolver, error) {
	log.ZDebug(context.Background(), "Build", "target", target)
	endpoints := strings.FieldsFunc(GetEndpoints(target), func(r rune) bool {
		return r == EndpointSepChar
	})
	endpoints = subset(endpoints, subsetSize)
	addrs := make([]resolver.Address, 0, len(endpoints))

	for _, val := range endpoints {
		addrs = append(addrs, resolver.Address{
			Addr: val,
		})
	}
	if err := cc.UpdateState(resolver.State{
		Addresses: addrs,
	}); err != nil {
		return nil, err
	}
```

```
	return &nopResolver{cc: cc}, nil
}
func init() {
	resolver.Register(&ResolverDirect{})
}
func (rd *ResolverDirect) Scheme() string {
	return scheme // return your custom scheme name
}

// GetEndpoints returns the endpoints from the given target.
func GetEndpoints(target resolver.Target) string {
	return strings.Trim(target.URL.Path, slashSeparator)
}
func subset(set []string, sub int) []string {
	rand.Shuffle(len(set), func(i, j int) {
		set[i], set[j] = set[j], set[i]
	})
	if len(set) <= sub {
		return set
	}

	return set[:sub]
}

type nopResolver struct {
	cc resolver.ClientConn
}

func (n nopResolver) ResolveNow(options resolver.ResolveNowOptions) {

}

func (n nopResolver) Close() {

}
```

pkg/common/discovery/direct/directconn.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package direct

//import (
//■"context"
//■"fmt"
//
//■config2 "github.com/openimsdk/open-im-server/v3/pkg/common/config"
//■"github.com/openimsdk/tools/errs"
//■"google.golang.org/grpc"
//■"google.golang.org/grpc/credentials/insecure"
//)
//
//type ServiceAddresses map[string][]int
//
//func getServiceAddresses(rpcRegisterName *config2.RpcRegisterName,
//■rpcPort *config2.RpcPort, longConnSvrPort []int) ServiceAddresses {
//■return ServiceAddresses{
//■■rpcRegisterName.OpenImUserName:           rpcPort.OpenImUserPort,
//■■rpcRegisterName.OpenImFriendName:         rpcPort.OpenImFriendPort,
//■■rpcRegisterName.OpenImMsgName:            rpcPort.OpenImMessagePort,
//■■rpcRegisterName.OpenImMessageGatewayName: longConnSvrPort,
//■■rpcRegisterName.OpenImGroupName:          rpcPort.OpenImGroupPort,
//■■rpcRegisterName.OpenImAuthName:           rpcPort.OpenImAuthPort,
//■■rpcRegisterName.OpenImPushName:           rpcPort.OpenImPushPort,
//■■rpcRegisterName.OpenImConversationName:   rpcPort.OpenImConversationPort,
//■■rpcRegisterName.OpenImThirdName:          rpcPort.OpenImThirdPort,
//■}
//}
//
//type ConnDirect struct {
//■additionalOpts        []grpc.DialOption
//■currentServiceAddress string
//■conns                 map[string][]*grpc.ClientConn
//■resolverDirect        *ResolverDirect
//■config                *config2.GlobalConfig
//}
//
//func (cd *ConnDirect) GetClientLocalConns() map[string][]*grpc.ClientConn {
//■return nil
//}
//
//func (cd *ConnDirect) GetUserIdHashGatewayHost(ctx context.Context, userId string) (string, error) {
//■return "", nil
//}
//
//func (cd *ConnDirect) Register(serviceName, host string, port int, opts ...grpc.DialOption) error {
//■return nil
//}
//
//func (cd *ConnDirect) UnRegister() error {
```

```
//■return nil
//}
//
//func (cd *ConnDirect) CreateRpcRootNodes(serviceNames []string) error {
//■return nil
//}
//
//func (cd *ConnDirect) RegisterConf2Registry(key string, conf []byte) error {
//■return nil
//}
//
//func (cd *ConnDirect) GetConfFromRegistry(key string) ([]byte, error) {
//■return nil, nil
//}
//
//func (cd *ConnDirect) Close() {
//
//}
//
//func NewConnDirect(config *config2.GlobalConfig) (*ConnDirect, error) {
//■return &ConnDirect{
//■■conns:          make(map[string][]*grpc.ClientConn),
//■■resolverDirect: NewResolverDirect(),
//■■config:         config,
//■}, nil
//}
//
//func (cd *ConnDirect) GetConns(ctx context.Context,
//■serviceName string, opts ...grpc.DialOption) ([]*grpc.ClientConn, error) {
//
//■if conns, exists := cd.conns[serviceName]; exists {
//■■return conns, nil
//■}
//■ports := getServiceAddresses(&cd.config.RpcRegisterName,
//■■&cd.config.RpcPort, cd.config.LongConnSvr.OpenImMessageGatewayPort)[serviceName]
//■var connections []*grpc.ClientConn
//■for _, port := range ports {
//■■conn, err := cd.dialServiceWithoutResolver(ctx, fmt.Sprintf(cd.config.Rpc.ListenIP+":%d", port), append(cd.addi
//■■if err != nil {
//■■■return nil, errs.Wrap(fmt.Errorf("connect to port %d failed,serviceName %s, IP %s", port, serviceName, cd.conf
//■■}
//■■connections = append(connections, conn)
//■}
//
//■if len(connections) == 0 {
//■■return nil, errs.New("no connections found for service", "serviceName", serviceName).Wrap()
//■}
//■return connections, nil
//}
//
//func (cd *ConnDirect) GetConn(ctx context.Context, serviceName string, opts ...grpc.DialOption) (*grpc.ClientConn,
//■// Get service addresses
//■addresses := getServiceAddresses(&cd.config.RpcRegisterName,
//■■&cd.config.RpcPort, cd.config.LongConnSvr.OpenImMessageGatewayPort)
//■address, ok := addresses[serviceName]
//■if !ok {
//■■return nil, errs.New("unknown service name", "serviceName", serviceName).Wrap()
//■}
//■var result string
//■for _, addr := range address {
//■■if result != "" {
//■■■result = result + "," + fmt.Sprintf(cd.config.Rpc.ListenIP+":%d", addr)
//■■} else {
//■■■result = fmt.Sprintf(cd.config.Rpc.ListenIP+":%d", addr)
//■■}
//■}
```

```
//■// Try to dial a new connection
//■conn, err := cd.dialService(ctx, result, append(cd.additionalOpts, opts...)...)
//■if err != nil {
//■■return nil, errs.WrapMsg(err, "address", result)
//■}
//
//■// Store the new connection
//■cd.conns[serviceName] = append(cd.conns[serviceName], conn)
//■return conn, nil
//}
//
//func (cd *ConnDirect) GetSelfConnTarget() string {
//■return cd.currentServiceAddress
//}
//
//func (cd *ConnDirect) AddOption(opts ...grpc.DialOption) {
//■cd.additionalOpts = append(cd.additionalOpts, opts...)
//}
//
//func (cd *ConnDirect) CloseConn(conn *grpc.ClientConn) {
//■if conn != nil {
//■■conn.Close()
//■}
//}
//
//func (cd *ConnDirect) dialService(ctx context.Context, address string, opts ...grpc.DialOption) (*grpc.ClientConn,
//■options := append(opts, grpc.WithTransportCredentials(insecure.NewCredentials()))
//■conn, err := grpc.DialContext(ctx, cd.resolverDirect.Scheme()+":///"+address, options...)
//
//■if err != nil {
//■■return nil, errs.WrapMsg(err, "address", address)
//■}
//■return conn, nil
//}
//
//func (cd *ConnDirect) dialServiceWithoutResolver(ctx context.Context, address string, opts ...grpc.DialOption) (*g
//■options := append(opts, grpc.WithTransportCredentials(insecure.NewCredentials()))
//■conn, err := grpc.DialContext(ctx, address, options...)
//
//■if err != nil {
//■■return nil, errs.Wrap(err)
//■}
//■return conn, nil
//}
```

pkg/common/discovery/direct/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package direct // import "github.com/openimsdk/open-im-server/v3/pkg/common/discovery/direct"
```

pkg/common/discovery/kubernetes

pkg/common/discovery/kubernetes/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package kubernetes // import "github.com/openimsdk/open-im-server/v3/pkg/common/discovery/kubernetes"
```

pkg/common/discovery/kubernetes/kubernetes.go

```
package kubernetes

import (
	"context"
	"fmt"
	"log"
	"os"
	"sync"
	"time"

	"google.golang.org/grpc"
	"google.golang.org/grpc/credentials/insecure"
	v1 "k8s.io/api/core/v1"
	metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
	"k8s.io/client-go/informers"
	"k8s.io/client-go/kubernetes"
	"k8s.io/client-go/rest"
	"k8s.io/client-go/tools/cache"
)

type KubernetesConnManager struct {
	clientset   *kubernetes.Clientset
	namespace   string
	dialOptions []grpc.DialOption

	rpcTargets map[string]string
	selfTarget string

	mu      sync.RWMutex
	connMap map[string][]*grpc.ClientConn
}

// NewKubernetesConnManager creates a new connection manager that uses Kubernetes services for service discovery.
func NewKubernetesConnManager(namespace string, options ...grpc.DialOption) (*KubernetesConnManager, error) {
	config, err := rest.InClusterConfig()
	if err != nil {
		return nil, fmt.Errorf("failed to create in-cluster config: %v", err)
	}

	clientset, err := kubernetes.NewForConfig(config)
	if err != nil {
		return nil, fmt.Errorf("failed to create clientset: %v", err)
	}

	k := &KubernetesConnManager{
		clientset:   clientset,
		namespace:   namespace,
		dialOptions: options,
		connMap:     make(map[string][]*grpc.ClientConn),
	}

	go k.watchEndpoints()

	return k, nil
}

func (k *KubernetesConnManager) initializeConns(serviceName string) error {
	port, err := k.getServicePort(serviceName)
	if err != nil {
		return err
	}

	endpoints, err := k.clientset.CoreV1().Endpoints(k.namespace).Get(context.Background(), serviceName, metav1.GetOpti
	if err != nil {
```

```
    return fmt.Errorf("failed to get endpoints for service %s: %v", serviceName, err)
  }

  // fmt.Println("Endpoints:", endpoints, "endpoints.Subsets:", endpoints.Subsets)

  var conns []*grpc.ClientConn
  for _, subset := range endpoints.Subsets {
    for _, address := range subset.Addresses {
      target := fmt.Sprintf("%s:%d", address.IP, port)
      // fmt.Println("IP target:", target)
      conn, err := grpc.Dial(target, append(k.dialOptions, grpc.WithTransportCredentials(insecure.NewCredentials()))..
      if err != nil {
        return fmt.Errorf("failed to dial endpoint %s: %v", target, err)
      }
      conns = append(conns, conn)
    }
  }

  k.mu.Lock()
  k.connMap[serviceName] = conns
  k.mu.Unlock()

  return nil
}

// GetConns returns gRPC client connections for a given Kubernetes service name.
func (k *KubernetesConnManager) GetConns(ctx context.Context, serviceName string, opts ...grpc.DialOption) ([]*grpc.
  k.mu.RLock()

  conns, exists := k.connMap[serviceName]
  k.mu.RUnlock()
  if exists {
    return conns, nil
  }

  k.mu.Lock()
  // Check if another goroutine has already initialized the connections when we released the read lock
  conns, exists = k.connMap[serviceName]
  if exists {
    return conns, nil
  }
  k.mu.Unlock()

  if err := k.initializeConns(serviceName); err != nil {
    fmt.Println("Failed to initialize connections:", err)
    return nil, fmt.Errorf("failed to initialize connections for service %s: %v", serviceName, err)
  }

  return k.connMap[serviceName], nil
}

// GetConn returns a single gRPC client connection for a given Kubernetes service name.
func (k *KubernetesConnManager) GetConn(ctx context.Context, serviceName string, opts ...grpc.DialOption) (*grpc.Cli
  var target string

  if k.rpcTargets[serviceName] == "" {
    var err error

    svcPort, err := k.getServicePort(serviceName)
    if err != nil {
      return nil, err
    }

    target = fmt.Sprintf("%s.%s.svc.cluster.local:%d", serviceName, k.namespace, svcPort)

    // fmt.Println("SVC target:", target)
```

```
	} else {
		target = k.rpcTargets[serviceName]
	}

	return grpc.DialContext(
		ctx,
		target,
		append([]grpc.DialOption{
			grpc.WithTransportCredentials(insecure.NewCredentials()),
			grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(1024*1024*10), grpc.MaxCallSendMsgSize(1024*1024*20)),
		}, k.dialOptions...)...,
	)
}

// GetSelfConnTarget returns the connection target for the current service.
func (k *KubernetesConnManager) GetSelfConnTarget() string {
	if k.selfTarget == "" {
		hostName := os.Getenv("HOSTNAME")

		pod, err := k.clientset.CoreV1().Pods(k.namespace).Get(context.Background(), hostName, metav1.GetOptions{})
		if err != nil {
			log.Printf("failed to get pod %s: %v \n", hostName, err)
		}

		for pod.Status.PodIP == "" {
			pod, err = k.clientset.CoreV1().Pods(k.namespace).Get(context.TODO(), hostName, metav1.GetOptions{})
			if err != nil {
				log.Printf("Error getting pod: %v \n", err)
			}

			time.Sleep(3 * time.Second)
		}

		var selfPort int32

		for _, port := range pod.Spec.Containers[0].Ports {
			if port.ContainerPort != 10001 {
				selfPort = port.ContainerPort
				break
			}
		}

		k.selfTarget = fmt.Sprintf("%s:%d", pod.Status.PodIP, selfPort)
	}

	return k.selfTarget
}

// AddOption appends gRPC dial options to the existing options.
func (k *KubernetesConnManager) AddOption(opts ...grpc.DialOption) {
	k.mu.Lock()
	defer k.mu.Unlock()
	k.dialOptions = append(k.dialOptions, opts...)
}

// CloseConn closes a given gRPC client connection.
func (k *KubernetesConnManager) CloseConn(conn *grpc.ClientConn) {
	conn.Close()
}

// Close closes all gRPC connections managed by KubernetesConnManager.
func (k *KubernetesConnManager) Close() {
	k.mu.Lock()
	defer k.mu.Unlock()
	for _, conns := range k.connMap {
		for _, conn := range conns {
```

```
		_ = conn.Close()
		}
	}
	k.connMap = make(map[string][]*grpc.ClientConn)
}

func (k *KubernetesConnManager) Register(serviceName, host string, port int, opts ...grpc.DialOption) error {
	return nil
}

func (k *KubernetesConnManager) UnRegister() error {
	return nil
}

func (k *KubernetesConnManager) GetUserIdHashGatewayHost(ctx context.Context, userId string) (string, error) {
	return "", nil
}

func (k *KubernetesConnManager) getServicePort(serviceName string) (int32, error) {
	var svcPort int32

	svc, err := k.clientset.CoreV1().Services(k.namespace).Get(context.Background(), serviceName, metav1.GetOptions{})
	if err != nil {
		fmt.Print("namespace:", k.namespace)
		return 0, fmt.Errorf("failed to get service %s: %v", serviceName, err)
	}

	if len(svc.Spec.Ports) == 0 {
		return 0, fmt.Errorf("service %s has no ports defined", serviceName)
	}

	for _, port := range svc.Spec.Ports {
		// fmt.Println(serviceName, " Now Get Port:", port.Port)
		if port.Port != 10001 {
			svcPort = port.Port
			break
		}
	}

	return svcPort, nil
}

// watchEndpoints listens for changes in Pod resources.
func (k *KubernetesConnManager) watchEndpoints() {
	informerFactory := informers.NewSharedInformerFactory(k.clientset, time.Minute*10)
	informer := informerFactory.Core().V1().Pods().Informer()

	// Watch for Pod changes (add, update, delete)
	informer.AddEventHandler(cache.ResourceEventHandlerFuncs{
		AddFunc: func(obj interface{}) {
			k.handleEndpointChange(obj)
		},
		UpdateFunc: func(oldObj, newObj interface{}) {
			k.handleEndpointChange(newObj)
		},
		DeleteFunc: func(obj interface{}) {
			k.handleEndpointChange(obj)
		},
	})

	informerFactory.Start(context.Background().Done())
	<-context.Background().Done() // Block forever
}

func (k *KubernetesConnManager) handleEndpointChange(obj interface{}) {
	endpoint, ok := obj.(*v1.Endpoints)
```

```
    if !ok {
        return
    }
    serviceName := endpoint.Name
    if err := k.initializeConns(serviceName); err != nil {
        fmt.Printf("Error initializing connections for %s: %v\n", serviceName, err)
    }
}
```

pkg/common/discovery/etcd

pkg/common/discovery/etcd/config_manager.go

```
package etcd

import (
	"context"
	"os"
	"os/exec"
	"runtime"
	"sync"
	"syscall"

	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	clientv3 "go.etcd.io/etcd/client/v3"
)

var (
	ShutDowns []func() error
)

func RegisterShutDown(shutDown ...func() error) {
	ShutDowns = append(ShutDowns, shutDown...)
}

type ConfigManager struct {
	client           *clientv3.Client
	watchConfigNames []string
	lock             sync.Mutex
}

func BuildKey(s string) string {
	return ConfigKeyPrefix + s
}

func NewConfigManager(client *clientv3.Client, configNames []string) *ConfigManager {
	return &ConfigManager{
		client:           client,
		watchConfigNames: datautil.Batch(func(s string) string { return BuildKey(s) }, append(configNames, RestartKey))}
}

func (c *ConfigManager) Watch(ctx context.Context) {
	chans := make([]clientv3.WatchChan, 0, len(c.watchConfigNames))
	for _, name := range c.watchConfigNames {
		chans = append(chans, c.client.Watch(ctx, name, clientv3.WithPrefix()))
	}

	doWatch := func(watchChan clientv3.WatchChan) {
		for watchResp := range watchChan {
			if watchResp.Err() != nil {
				log.ZError(ctx, "watch err", errs.Wrap(watchResp.Err()))
				continue
			}
			for _, event := range watchResp.Events {
				if event.IsModify() {
					if datautil.Contain(string(event.Kv.Key), c.watchConfigNames...) {
						c.lock.Lock()
						err := restartServer(ctx)
						if err != nil {
							log.ZError(ctx, "restart server err", err)
						}
						c.lock.Unlock()
```

```
        }
    }
  }
 }
 for _, ch := range chans {
  go doWatch(ch)
 }
}

func restartServer(ctx context.Context) error {
 exePath, err := os.Executable()
 if err != nil {
  return errs.New("get executable path fail").Wrap()
 }

 args := os.Args
 env := os.Environ()

 cmd := exec.Command(exePath, args[1:]...)
 cmd.Env = env
 cmd.Stdout = os.Stdout
 cmd.Stderr = os.Stderr
 cmd.Stdin = os.Stdin

 if runtime.GOOS != "windows" {
  cmd.SysProcAttr = &syscall.SysProcAttr{}
 }
 log.ZInfo(ctx, "shutdown server")
 for _, f := range ShutDowns {
  if err = f(); err != nil {
   log.ZError(ctx, "shutdown fail", err)
  }
 }

 log.ZInfo(ctx, "restart server")
 err = cmd.Start()
 if err != nil {
  return errs.New("restart server fail").Wrap()
 }
 log.ZInfo(ctx, "cmd start over")

 os.Exit(0)
 return nil
}
```

pkg/common/discovery/etcd/const.go

```
package etcd

const (
	ConfigKeyPrefix       = "/open-im/config/"
	RestartKey            = "restart"
	EnableConfigCenterKey = "enable-config-center"
	Enable                = "enable"
	Disable               = "disable"
)
```

pkg/common/storage

pkg/common/storage/database

pkg/common/storage/database/black.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/pagination"
)

type Black interface {
	Create(ctx context.Context, blacks []*model.Black) (err error)
	Delete(ctx context.Context, blacks []*model.Black) (err error)
	Find(ctx context.Context, blacks []*model.Black) (blackList []*model.Black, err error)
	Take(ctx context.Context, ownerUserID, blockUserID string) (black *model.Black, err error)
	FindOwnerBlacks(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (total int64, blacks []*
	FindOwnerBlackInfos(ctx context.Context, ownerUserID string, userIDs []string) (blacks []*model.Black, err error)
	FindBlackUserIDs(ctx context.Context, ownerUserID string) (blackUserIDs []string, err error)
}

var (
	_ Black = (*mgoImpl)(nil)
	_ Black = (*redisImpl)(nil)
)

type mgoImpl struct {
}

func (m *mgoImpl) Create(ctx context.Context, blacks []*model.Black) (err error) {
	//TODO implement me
	panic("implement me")
}

func (m *mgoImpl) Delete(ctx context.Context, blacks []*model.Black) (err error) {
	//TODO implement me
	panic("implement me")
}

func (m *mgoImpl) Find(ctx context.Context, blacks []*model.Black) (blackList []*model.Black, err error) {
	//TODO implement me
	panic("implement me")
}

func (m *mgoImpl) Take(ctx context.Context, ownerUserID, blockUserID string) (black *model.Black, err error) {
	//TODO implement me
```

```
    panic("implement me")
}

func (m *mgoImpl) FindOwnerBlacks(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (total
    //TODO implement me
    panic("implement me")
}

func (m *mgoImpl) FindOwnerBlackInfos(ctx context.Context, ownerUserID string, userIDs []string) (blacks []*model.Bl
    //TODO implement me
    panic("implement me")
}

func (m *mgoImpl) FindBlackUserIDs(ctx context.Context, ownerUserID string) (blackUserIDs []string, err error) {
    //TODO implement me
    panic("implement me")
}

type redisImpl struct {
}

func (r *redisImpl) Create(ctx context.Context, blacks []*model.Black) (err error) {

    //TODO implement me
    panic("implement me")
}

func (r *redisImpl) Delete(ctx context.Context, blacks []*model.Black) (err error) {
    //TODO implement me
    panic("implement me")
}

func (r *redisImpl) Find(ctx context.Context, blacks []*model.Black) (blackList []*model.Black, err error) {
    //TODO implement me
    panic("implement me")
}

func (r *redisImpl) Take(ctx context.Context, ownerUserID, blockUserID string) (black *model.Black, err error) {
    //TODO implement me
    panic("implement me")
}

func (r *redisImpl) FindOwnerBlacks(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (tota
    //TODO implement me
    panic("implement me")
}

func (r *redisImpl) FindOwnerBlackInfos(ctx context.Context, ownerUserID string, userIDs []string) (blacks []*model.
    //TODO implement me
    panic("implement me")
}

func (r *redisImpl) FindBlackUserIDs(ctx context.Context, ownerUserID string) (blackUserIDs []string, err error) {
    //TODO implement me
    panic("implement me")
}
```

pkg/common/storage/database/cache.go

```go
package database

import (
	"context"
	"time"
)

type Cache interface {
	Get(ctx context.Context, key []string) (map[string]string, error)
	Prefix(ctx context.Context, prefix string) (map[string]string, error)
	Set(ctx context.Context, key string, value string, expireAt time.Duration) error
	Incr(ctx context.Context, key string, value int) (int, error)
	Del(ctx context.Context, key []string) error
	Lock(ctx context.Context, key string, duration time.Duration) (string, error)
	Unlock(ctx context.Context, key string, value string) error
}
```

pkg/common/storage/database/client_config.go

```go
package database

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/pagination"
)

type ClientConfig interface {
	Set(ctx context.Context, userID string, config map[string]string) error
	Get(ctx context.Context, userID string) (map[string]string, error)
	Del(ctx context.Context, userID string, keys []string) error
	GetPage(ctx context.Context, userID string, key string, pagination pagination.Pagination) (int64, []*model.ClientCo
}
```

pkg/common/storage/database/conversation.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/pagination"
)

type Conversation interface {
	Create(ctx context.Context, conversations []*model.Conversation) (err error)
	UpdateByMap(ctx context.Context, userIDs []string, conversationID string, args map[string]any) (rows int64, err err
	UpdateUserConversations(ctx context.Context, userID string, args map[string]any) ([]*model.Conversation, error)
	Update(ctx context.Context, conversation *model.Conversation) (err error)
	Find(ctx context.Context, ownerUserID string, conversationIDs []string) (conversations []*model.Conversation, err e
	FindUserID(ctx context.Context, userIDs []string, conversationIDs []string) ([]string, error)
	FindUserIDAllConversationID(ctx context.Context, userID string) ([]string, error)
	FindUserIDAllNotNotifyConversationID(ctx context.Context, userID string) ([]string, error)
	FindUserIDAllPinnedConversationID(ctx context.Context, userID string) ([]string, error)
	Take(ctx context.Context, userID, conversationID string) (conversation *model.Conversation, err error)
	FindConversationID(ctx context.Context, userID string, conversationIDs []string) (existConversationID []string, err
	FindUserIDAllConversations(ctx context.Context, userID string) (conversations []*model.Conversation, err error)
	FindRecvMsgUserIDs(ctx context.Context, conversationID string, recvOpts []int) ([]string, error)
	GetUserRecvMsgOpt(ctx context.Context, ownerUserID, conversationID string) (opt int, err error)
	GetAllConversationIDs(ctx context.Context) ([]string, error)
	GetAllConversationIDsNumber(ctx context.Context) (int64, error)
	PageConversationIDs(ctx context.Context, pagination pagination.Pagination) (conversationIDs []string, err error)
	GetConversationsByConversationID(ctx context.Context, conversationIDs []string) ([]*model.Conversation, error)
	GetConversationIDsNeedDestruct(ctx context.Context) ([]*model.Conversation, error)
	GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string) ([]string, error)
	FindConversationUserVersion(ctx context.Context, userID string, version uint, limit int) (*model.VersionLog, error)
	FindRandConversation(ctx context.Context, ts int64, limit int) ([]*model.Conversation, error)
	DeleteUsersConversations(ctx context.Context, userID string, conversationIDs []string) (err error)
}
```

pkg/common/storage/database/doc.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database // import "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model/relation"
```

pkg/common/storage/database/friend.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
"context"

"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
"github.com/openimsdk/tools/db/pagination"
)

// Friend defines the operations for managing friends in MongoDB.
type Friend interface {
// Create inserts multiple friend records.
Create(ctx context.Context, friends []*model.Friend) (err error)
// Delete removes specified friends of the owner user.
Delete(ctx context.Context, ownerUserID string, friendUserIDs []string) (err error)
// UpdateByMap updates specific fields of a friend document using a map.
UpdateByMap(ctx context.Context, ownerUserID string, friendUserID string, args map[string]any) (err error)
// UpdateRemark modify remarks.
UpdateRemark(ctx context.Context, ownerUserID, friendUserID, remark string) (err error)
// Take retrieves a single friend document. Returns an error if not found.
Take(ctx context.Context, ownerUserID, friendUserID string) (friend *model.Friend, err error)
// FindUserState finds the friendship status between two users.
FindUserState(ctx context.Context, userID1, userID2 string) (friends []*model.Friend, err error)
// FindFriends retrieves a list of friends for a given owner. Missing friends do not cause an error.
FindFriends(ctx context.Context, ownerUserID string, friendUserIDs []string) (friends []*model.Friend, err error)
// FindReversalFriends finds users who have added the specified user as a friend.
FindReversalFriends(ctx context.Context, friendUserID string, ownerUserIDs []string) (friends []*model.Friend, err
// FindOwnerFriends retrieves a paginated list of friends for a given owner.
FindOwnerFriends(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (total int64, friends |
// FindInWhoseFriends finds users who have added the specified user as a friend, with pagination.
FindInWhoseFriends(ctx context.Context, friendUserID string, pagination pagination.Pagination) (total int64, friend
// FindFriendUserIDs retrieves a list of friend user IDs for a given owner.
FindFriendUserIDs(ctx context.Context, ownerUserID string) (friendUserIDs []string, err error)
// UpdateFriends update friends' fields
UpdateFriends(ctx context.Context, ownerUserID string, friendUserIDs []string, val map[string]any) (err error)

FindIncrVersion(ctx context.Context, ownerUserID string, version uint, limit int) (*model.VersionLog, error)

FindFriendUserID(ctx context.Context, friendUserID string) ([]string, error)

//SearchFriend(ctx context.Context, ownerUserID, keyword string, pagination pagination.Pagination) (int64, []*model

FindOwnerFriendUserIds(ctx context.Context, ownerUserID string, limit int) ([]string, error)

IncrVersion(ctx context.Context, ownerUserID string, friendUserIDs []string, state int32) error
}
```

pkg/common/storage/database/friend_request.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
"context"

"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
"github.com/openimsdk/tools/db/pagination"
)

type FriendRequest interface {
// Insert multiple records
Create(ctx context.Context, friendRequests []*model.FriendRequest) (err error)
// Delete record
Delete(ctx context.Context, fromUserID, toUserID string) (err error)
// Update with zero values
UpdateByMap(ctx context.Context, formUserID string, toUserID string, args map[string]any) (err error)
// Update multiple records (non-zero values)
Update(ctx context.Context, friendRequest *model.FriendRequest) (err error)
// Get friend requests sent to a specific user, no error returned if not found
Find(ctx context.Context, fromUserID, toUserID string) (friendRequest *model.FriendRequest, err error)
Take(ctx context.Context, fromUserID, toUserID string) (friendRequest *model.FriendRequest, err error)
// Get list of friend requests received by toUserID
FindToUserID(ctx context.Context, toUserID string, handleResults []int, pagination pagination.Pagination) (total in
// Get list of friend requests sent by fromUserID
FindFromUserID(ctx context.Context, fromUserID string, handleResults []int, pagination pagination.Pagination) (tota
FindBothFriendRequests(ctx context.Context, fromUserID, toUserID string) (friends []*model.FriendRequest, err error
GetUnhandledCount(ctx context.Context, userID string, ts int64) (int64, error)
}
```

pkg/common/storage/database/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
"context"
"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
"github.com/openimsdk/tools/db/pagination"
"time"
)

type Group interface {
Create(ctx context.Context, groups []*model.Group) (err error)
UpdateMap(ctx context.Context, groupID string, args map[string]any) (err error)
UpdateStatus(ctx context.Context, groupID string, status int32) (err error)
Find(ctx context.Context, groupIDs []string) (groups []*model.Group, err error)
Take(ctx context.Context, groupID string) (group *model.Group, err error)
Search(ctx context.Context, keyword string, pagination pagination.Pagination) (total int64, groups []*model.Group,
// Get Group total quantity
CountTotal(ctx context.Context, before *time.Time) (count int64, err error)
// Get Group total quantity every day
CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int64, error)

FindJoinSortGroupID(ctx context.Context, groupIDs []string) ([]string, error)

SearchJoin(ctx context.Context, groupIDs []string, keyword string, pagination pagination.Pagination) (int64, []*mod
}
```

pkg/common/storage/database/group_member.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
"context"

"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
"github.com/openimsdk/tools/db/pagination"
)

type GroupMember interface {
Create(ctx context.Context, groupMembers []*model.GroupMember) (err error)
Delete(ctx context.Context, groupID string, userIDs []string) (err error)
Update(ctx context.Context, groupID string, userID string, data map[string]any) (err error)
UpdateRoleLevel(ctx context.Context, groupID string, userID string, roleLevel int32) error
UpdateUserRoleLevels(ctx context.Context, groupID string, firstUserID string, firstUserRoleLevel int32, secondUserI
FindMemberUserID(ctx context.Context, groupID string) (userIDs []string, err error)
Take(ctx context.Context, groupID string, userID string) (groupMember *model.GroupMember, err error)
Find(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupMember, error)
FindInGroup(ctx context.Context, userID string, groupIDs []string) ([]*model.GroupMember, error)
TakeOwner(ctx context.Context, groupID string) (groupMember *model.GroupMember, err error)
SearchMember(ctx context.Context, keyword string, groupID string, pagination pagination.Pagination) (total int64, g
FindRoleLevelUserIDs(ctx context.Context, groupID string, roleLevel int32) ([]string, error)
FindUserJoinedGroupID(ctx context.Context, userID string) (groupIDs []string, err error)
TakeGroupMemberNum(ctx context.Context, groupID string) (count int64, err error)
FindUserManagedGroupID(ctx context.Context, userID string) (groupIDs []string, err error)
IsUpdateRoleLevel(data map[string]any) bool
JoinGroupIncrVersion(ctx context.Context, userID string, groupIDs []string, state int32) error
MemberGroupIncrVersion(ctx context.Context, groupID string, userIDs []string, state int32) error
FindMemberIncrVersion(ctx context.Context, groupID string, version uint, limit int) (*model.VersionLog, error)
BatchFindMemberIncrVersion(ctx context.Context, groupIDs []string, versions []uint, limits []int) ([]*model.Version
FindJoinIncrVersion(ctx context.Context, userID string, version uint, limit int) (*model.VersionLog, error)
}
```

pkg/common/storage/database/group_request.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/pagination"
)

type GroupRequest interface {
	Create(ctx context.Context, groupRequests []*model.GroupRequest) (err error)
	Delete(ctx context.Context, groupID string, userID string) (err error)
	UpdateHandler(ctx context.Context, groupID string, userID string, handledMsg string, handleResult int32) (err error
	Take(ctx context.Context, groupID string, userID string) (groupRequest *model.GroupRequest, err error)
	FindGroupRequests(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupRequest, error)
	Page(ctx context.Context, userID string, groupIDs []string, handleResults []int, pagination pagination.Pagination)
	PageGroup(ctx context.Context, groupIDs []string, handleResults []int, pagination pagination.Pagination) (total int
	GetUnhandledCount(ctx context.Context, groupIDs []string, ts int64) (int64, error)
}
```

pkg/common/storage/database/log.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/pagination"
	"time"
)

type Log interface {
	Create(ctx context.Context, log []*model.Log) error
	Search(ctx context.Context, keyword string, start time.Time, end time.Time, pagination pagination.Pagination) (int6
	Delete(ctx context.Context, logID []string, userID string) error
	Get(ctx context.Context, logIDs []string, userID string) ([]*model.Log, error)
}
```

pkg/common/storage/database/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/protocol/msg"
    "go.mongodb.org/mongo-driver/mongo"
)

type Msg interface {
    Create(ctx context.Context, model *model.MsgDocModel) error
    UpdateMsg(ctx context.Context, docID string, index int64, key string, value any) (*mongo.UpdateResult, error)
    PushUnique(ctx context.Context, docID string, index int64, key string, value any) (*mongo.UpdateResult, error)
    FindOneByDocID(ctx context.Context, docID string) (*model.MsgDocModel, error)
    GetMsgBySeqIndexIn1Doc(ctx context.Context, userID, docID string, seqs []int64) ([]*model.MsgInfoModel, error)
    GetNewestMsg(ctx context.Context, conversationID string) (*model.MsgInfoModel, error)
    GetOldestMsg(ctx context.Context, conversationID string) (*model.MsgInfoModel, error)
    DeleteMsgsInOneDocByIndex(ctx context.Context, docID string, indexes []int) error
    MarkSingleChatMsgsAsRead(ctx context.Context, userID string, docID string, indexes []int64) error
    SearchMessage(ctx context.Context, req *msg.SearchMessageReq) (int64, []*model.MsgInfoModel, error)
    RangeUserSendCount(ctx context.Context, start time.Time, end time.Time, group bool, ase bool, pageNumber int32, sho
    RangeGroupSendCount(ctx context.Context, start time.Time, end time.Time, ase bool, pageNumber int32, showNumber int
    DeleteDoc(ctx context.Context, docID string) error
    GetRandBeforeMsg(ctx context.Context, ts int64, limit int) ([]*model.MsgDocModel, error)
    GetLastMessageSeqByTime(ctx context.Context, conversationID string, time int64) (int64, error)
    GetLastMessage(ctx context.Context, conversationID string) (*model.MsgInfoModel, error)
    FindSeqs(ctx context.Context, conversationID string, seqs []int64) ([]*model.MsgInfoModel, error)
}
```

pkg/common/storage/database/name.go

```go
package database

const (
	BlackName               = "black"
	ConversationName        = "conversation"
	FriendName              = "friend"
	FriendVersionName       = "friend_version"
	FriendRequestName       = "friend_request"
	GroupName               = "group"
	GroupMemberName         = "group_member"
	GroupMemberVersionName  = "group_member_version"
	GroupJoinVersionName    = "group_join_version"
	ConversationVersionName = "conversation_version"
	GroupRequestName        = "group_request"
	LogName                 = "log"
	ObjectName              = "s3"
	UserName                = "user"
	SeqConversationName     = "seq"
	SeqUserName             = "seq_user"
	StreamMsgName           = "stream_msg"
	CacheName               = "cache"
)
```

pkg/common/storage/database/object.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

type ObjectInfo interface {
    SetObject(ctx context.Context, obj *model.Object) error
    Take(ctx context.Context, engine string, name string) (*model.Object, error)
    Delete(ctx context.Context, engine string, name []string) error
    FindExpirationObject(ctx context.Context, engine string, expiration time.Time, needDelType []string, count int64) (
    GetKeyCount(ctx context.Context, engine string, key string) (int64, error)

    GetEngineCount(ctx context.Context, engine string) (int64, error)
    GetEngineInfo(ctx context.Context, engine string, limit int, skip int) ([]*model.Object, error)
    UpdateEngine(ctx context.Context, oldEngine, oldName string, newEngine string) error
}
```

pkg/common/storage/database/seq.go

```go
package database

import "context"

type SeqTime struct {
	Seq  int64
	Time int64
}

type SeqConversation interface {
	Malloc(ctx context.Context, conversationID string, size int64) (int64, error)
	GetMaxSeq(ctx context.Context, conversationID string) (int64, error)
	SetMaxSeq(ctx context.Context, conversationID string, seq int64) error
	GetMinSeq(ctx context.Context, conversationID string) (int64, error)
	SetMinSeq(ctx context.Context, conversationID string, seq int64) error
}
```

pkg/common/storage/database/seq_user.go

```
package database

import "context"

type SeqUser interface {
    GetUserMaxSeq(ctx context.Context, conversationID string, userID string) (int64, error)
    SetUserMaxSeq(ctx context.Context, conversationID string, userID string, seq int64) error
    GetUserMinSeq(ctx context.Context, conversationID string, userID string) (int64, error)
    SetUserMinSeq(ctx context.Context, conversationID string, userID string, seq int64) error
    GetUserReadSeq(ctx context.Context, conversationID string, userID string) (int64, error)
    SetUserReadSeq(ctx context.Context, conversationID string, userID string, seq int64) error
    GetUserReadSeqs(ctx context.Context, userID string, conversationID []string) (map[string]int64, error)
}
```

pkg/common/storage/database/user.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package database

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/protocol/user"
    "github.com/openimsdk/tools/db/pagination"
)

type User interface {
    Create(ctx context.Context, users []*model.User) (err error)
    UpdateByMap(ctx context.Context, userID string, args map[string]any) (err error)
    Find(ctx context.Context, userIDs []string) (users []*model.User, err error)
    Take(ctx context.Context, userID string) (user *model.User, err error)
    TakeNotification(ctx context.Context, level int64) (user []*model.User, err error)
    TakeGTEAppManagerLevel(ctx context.Context, level int64) (user []*model.User, err error)
    TakeByNickname(ctx context.Context, nickname string) (user []*model.User, err error)
    Page(ctx context.Context, pagination pagination.Pagination) (count int64, users []*model.User, err error)
    PageFindUser(ctx context.Context, level1 int64, level2 int64, pagination pagination.Pagination) (count int64, users
    PageFindUserWithKeyword(ctx context.Context, level1 int64, level2 int64, userID, nickName string, pagination pagina
    Exist(ctx context.Context, userID string) (exist bool, err error)
    GetAllUserID(ctx context.Context, pagination pagination.Pagination) (count int64, userIDs []string, err error)
    GetUserGlobalRecvMsgOpt(ctx context.Context, userID string) (opt int, err error)
    // Get user total quantity
    CountTotal(ctx context.Context, before *time.Time) (count int64, err error)
    // Get user total quantity every day
    CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int64, error)

    SortQuery(ctx context.Context, userIDName map[string]string, asc bool) ([]*model.User, error)

    // CRUD user command
    AddUserCommand(ctx context.Context, userID string, Type int32, UUID string, value string, ex string) error
    DeleteUserCommand(ctx context.Context, userID string, Type int32, UUID string) error
    UpdateUserCommand(ctx context.Context, userID string, Type int32, UUID string, val map[string]any) error
    GetUserCommand(ctx context.Context, userID string, Type int32) ([]*user.CommandInfoResp, error)
    GetAllUserCommand(ctx context.Context, userID string) ([]*user.AllCommandInfoResp, error)
}
```

pkg/common/storage/database/version_log.go

```go
package database

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

const (
	FirstVersion         = 1
	DefaultDeleteVersion = 0
)

type VersionLog interface {
	IncrVersion(ctx context.Context, dId string, eIds []string, state int32) error
	FindChangeLog(ctx context.Context, dId string, version uint, limit int) (*model.VersionLog, error)
	BatchFindChangeLog(ctx context.Context, dIds []string, versions []uint, limits []int) ([]*model.VersionLog, error)
	DeleteAfterUnchangedLog(ctx context.Context, deadline time.Time) error
	Delete(ctx context.Context, dId string) error
}
```

pkg/common/storage/database/mgo

pkg/common/storage/database/mgo/black.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewBlackMongo(db *mongo.Database) (database.Black, error) {
	coll := db.Collection(database.BlackName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "owner_user_id", Value: 1},
			{Key: "block_user_id", Value: 1},
		},
		Options: options.Index().SetUnique(true),
	})
	if err != nil {
		return nil, err
	}
	return &BlackMgo{coll: coll}, nil
}

type BlackMgo struct {
	coll *mongo.Collection
}

func (b *BlackMgo) blackFilter(ownerUserID, blockUserID string) bson.M {
	return bson.M{
		"owner_user_id": ownerUserID,
		"block_user_id": blockUserID,
	}
}

func (b *BlackMgo) blacksFilter(blacks []*model.Black) bson.M {
	if len(blacks) == 0 {
		return nil
	}
	or := make(bson.A, 0, len(blacks))
	for _, black := range blacks {
		or = append(or, b.blackFilter(black.OwnerUserID, black.BlockUserID))
```

```
    }
    return bson.M{"$or": or}
}

func (b *BlackMgo) Create(ctx context.Context, blacks []*model.Black) (err error) {
    return mongoutil.InsertMany(ctx, b.coll, blacks)
}

func (b *BlackMgo) Delete(ctx context.Context, blacks []*model.Black) (err error) {
    if len(blacks) == 0 {
        return nil
    }
    return mongoutil.DeleteMany(ctx, b.coll, b.blacksFilter(blacks))
}

func (b *BlackMgo) UpdateByMap(ctx context.Context, ownerUserID, blockUserID string, args map[string]any) (err error
    if len(args) == 0 {
        return nil
    }
    return mongoutil.UpdateOne(ctx, b.coll, b.blackFilter(ownerUserID, blockUserID), bson.M{"$set": args}, false)
}

func (b *BlackMgo) Find(ctx context.Context, blacks []*model.Black) (blackList []*model.Black, err error) {
    return mongoutil.Find[*model.Black](ctx, b.coll, b.blacksFilter(blacks))
}

func (b *BlackMgo) Take(ctx context.Context, ownerUserID, blockUserID string) (black *model.Black, err error) {
    return mongoutil.FindOne[*model.Black](ctx, b.coll, b.blackFilter(ownerUserID, blockUserID))
}

func (b *BlackMgo) FindOwnerBlacks(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (total
    return mongoutil.FindPage[*model.Black](ctx, b.coll, bson.M{"owner_user_id": ownerUserID}, pagination)
}

func (b *BlackMgo) FindOwnerBlackInfos(ctx context.Context, ownerUserID string, userIDs []string) (blacks []*model.B
    if len(userIDs) == 0 {
        return mongoutil.Find[*model.Black](ctx, b.coll, bson.M{"owner_user_id": ownerUserID})
    }
    return mongoutil.Find[*model.Black](ctx, b.coll, bson.M{"owner_user_id": ownerUserID, "block_user_id": bson.M{"$in'
}

func (b *BlackMgo) FindBlackUserIDs(ctx context.Context, ownerUserID string) (blackUserIDs []string, err error) {
    return mongoutil.Find[string](ctx, b.coll, bson.M{"owner_user_id": ownerUserID}, options.Find().SetProjection(bson.
}
```

pkg/common/storage/database/mgo/cache.go

```
package mgo

import (
	"context"
	"strconv"
	"time"

	"github.com/google/uuid"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/errs"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewCacheMgo(db *mongo.Database) (*CacheMgo, error) {
	coll := db.Collection(database.CacheName)
	_, err := coll.Indexes().CreateMany(context.Background(), []mongo.IndexModel{
		{
			Keys: bson.D{
				{Key: "key", Value: 1},
			},
			Options: options.Index().SetUnique(true),
		},
		{
			Keys: bson.D{
				{Key: "expire_at", Value: 1},
			},
			Options: options.Index().SetExpireAfterSeconds(0),
		},
	})
	if err != nil {
		return nil, errs.Wrap(err)
	}
	return &CacheMgo{coll: coll}, nil
}

type CacheMgo struct {
	coll *mongo.Collection
}

func (x *CacheMgo) findToMap(res []model.Cache, now time.Time) map[string]string {
	kv := make(map[string]string)
	for _, re := range res {
		if re.ExpireAt != nil && re.ExpireAt.Before(now) {
			continue
		}
		kv[re.Key] = re.Value
	}
	return kv

}

func (x *CacheMgo) Get(ctx context.Context, key []string) (map[string]string, error) {
	if len(key) == 0 {
		return nil, nil
	}
	now := time.Now()
	res, err := mongoutil.Find[model.Cache](ctx, x.coll, bson.M{
		"key": bson.M{"$in": key},
		"$or": []bson.M{
			{"expire_at": bson.M{"$gt": now}},
```

```
        {"expire_at": nil},
    },
    })
    if err != nil {
        return nil, err
    }
    return x.findToMap(res, now), nil
}

func (x *CacheMgo) Prefix(ctx context.Context, prefix string) (map[string]string, error) {
    now := time.Now()
    res, err := mongoutil.Find[model.Cache](ctx, x.coll, bson.M{
        "key": bson.M{"$regex": "^" + prefix},
        "$or": []bson.M{
            {"expire_at": bson.M{"$gt": now}},
            {"expire_at": nil},
        },
    })
    if err != nil {
        return nil, err
    }
    return x.findToMap(res, now), nil
}

func (x *CacheMgo) Set(ctx context.Context, key string, value string, expireAt time.Duration) error {
    cv := &model.Cache{
        Key:   key,
        Value: value,
    }
    if expireAt > 0 {
        now := time.Now().Add(expireAt)
        cv.ExpireAt = &now
    }
    opt := options.Update().SetUpsert(true)
    return mongoutil.UpdateOne(ctx, x.coll, bson.M{"key": key}, bson.M{"$set": cv}, false, opt)
}

func (x *CacheMgo) Incr(ctx context.Context, key string, value int) (int, error) {
    pipeline := mongo.Pipeline{
        {
            {"$set", bson.M{
                "value": bson.M{
                    "$toString": bson.M{
                        "$add": bson.A{
                            bson.M{"$toInt": "$value"},
                            value,
                        },
                    },
                },
            }},
        },
    }
    opt := options.FindOneAndUpdate().SetReturnDocument(options.After)
    res, err := mongoutil.FindOneAndUpdate[model.Cache](ctx, x.coll, bson.M{"key": key}, pipeline, opt)
    if err != nil {
        return 0, err
    }
    return strconv.Atoi(res.Value)
}

func (x *CacheMgo) Del(ctx context.Context, key []string) error {
    if len(key) == 0 {
        return nil
    }
    _, err := x.coll.DeleteMany(ctx, bson.M{"key": bson.M{"$in": key}})
    return errs.Wrap(err)
```

```
}

func (x *CacheMgo) lockKey(key string) string {
    return "LOCK_" + key
}

func (x *CacheMgo) Lock(ctx context.Context, key string, duration time.Duration) (string, error) {
    tmp, err := uuid.NewUUID()
    if err != nil {
        return "", err
    }
    if duration <= 0 || duration > time.Minute*10 {
        duration = time.Minute * 10
    }
    cv := &model.Cache{
        Key:      x.lockKey(key),
        Value:    tmp.String(),
        ExpireAt: nil,
    }
    ctx, cancel := context.WithTimeout(ctx, time.Second*30)
    defer cancel()
    wait := func() error {
        timeout := time.NewTimer(time.Millisecond * 100)
        defer timeout.Stop()
        select {
        case <-ctx.Done():
            return ctx.Err()
        case <-timeout.C:
            return nil
        }
    }
    for {
        if err := mongoutil.DeleteOne(ctx, x.coll, bson.M{"key": key, "expire_at": bson.M{"$lt": time.Now()}}); err != ni
            return "", err
        }
        expireAt := time.Now().Add(duration)
        cv.ExpireAt = &expireAt
        if err := mongoutil.InsertMany[*model.Cache](ctx, x.coll, []*model.Cache{cv}); err != nil {
            if mongo.IsDuplicateKeyError(err) {
                if err := wait(); err != nil {
                    return "", err
                }
                continue
            }
            return "", err
        }
        return cv.Value, nil
    }
}

func (x *CacheMgo) Unlock(ctx context.Context, key string, value string) error {
    return mongoutil.DeleteOne(ctx, x.coll, bson.M{"key": x.lockKey(key), "value": value})
}
```

pkg/common/storage/database/mgo/cache_test.go

```
package mgo

import (
■"context"
■"strings"
■"sync"
■"testing"
■"time"

■"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
■"github.com/openimsdk/tools/db/mongoutil"
■"go.mongodb.org/mongo-driver/bson"
■"go.mongodb.org/mongo-driver/mongo"
■"go.mongodb.org/mongo-driver/mongo/options"
)

func TestName1111(t *testing.T) {
■coll := Mongodb().Collection("temp")

■//updatePipeline := mongo.Pipeline{
■//■{
■//■■{"$set", bson.M{
■//■■■"age": bson.M{
■//■■■■"$toString": bson.M{
■//■■■■■"$add": bson.A{
■//■■■■■■bson.M{"$toInt": "$age"},
■//■■■■■■1,
■//■■■■■},
■//■■■■},
■//■■■},
■//■■}},
■//■},
■//}

■pipeline := mongo.Pipeline{
■■{
■■■{"$set", bson.M{
■■■■"value": bson.M{
■■■■■"$toString": bson.M{
■■■■■■"$add": bson.A{
■■■■■■■bson.M{"$toInt": "$value"},
■■■■■■■1,
■■■■■■},
■■■■■},
■■■■},
■■■}},
■■},
■}

■opt := options.FindOneAndUpdate().SetUpsert(false).SetReturnDocument(options.After)
■res, err := mongoutil.FindOneAndUpdate[model.Cache](context.Background(), coll, bson.M{"key": "123456"}, pipeline,
■if err != nil {
■■panic(err)
■}
■t.Log(res)
}

func TestName33333(t *testing.T) {
■c, err := NewCacheMgo(Mongodb())
■if err != nil {
■■panic(err)
■}
■if err := c.Set(context.Background(), "123456", "123456", time.Hour); err != nil {
■■panic(err)
```

```
	}

	if err := c.Set(context.Background(), "123666", "123666", time.Hour); err != nil {
		panic(err)
	}

	res1, err := c.Get(context.Background(), []string{"123456"})
	if err != nil {
		panic(err)
	}
	t.Log(res1)

	res2, err := c.Prefix(context.Background(), "123")
	if err != nil {
		panic(err)
	}
	t.Log(res2)
}

func TestName1111aa(t *testing.T) {

	c, err := NewCacheMgo(Mongodb())
	if err != nil {
		panic(err)
	}
	var count int

	key := "123456"

	doFunc := func() {
		value, err := c.Lock(context.Background(), key, time.Second*30)
		if err != nil {
			t.Log("Lock error", err)
			return
		}
		tmp := count
		tmp++
		count = tmp
		t.Log("count", tmp)
		if err := c.Unlock(context.Background(), key, value); err != nil {
			t.Log("Unlock error", err)
			return
		}
	}

	if _, err := c.Lock(context.Background(), key, time.Second*10); err != nil {
		t.Log(err)
		return
	}

	var wg sync.WaitGroup
	for i := 0; i < 32; i++ {
		wg.Add(1)
		go func() {
			defer wg.Done()
			for i := 0; i < 100; i++ {
				doFunc()
			}
		}()
	}

	wg.Wait()

}

func TestName111111a(t *testing.T) {
```

```
	arr := strings.SplitN("1:testkakskdask:1111", ":", 2)
	t.Log(arr)
}
```

pkg/common/storage/database/mgo/client_config.go

```
// Copyright © 2023 OpenIM open source community. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/tools/db/mongoutil"
    "github.com/openimsdk/tools/db/pagination"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

    "github.com/openimsdk/tools/errs"
)

func NewClientConfig(db *mongo.Database) (database.ClientConfig, error) {
    coll := db.Collection("config")
    _, err := coll.Indexes().CreateMany(context.Background(), []mongo.IndexModel{
        {
            Keys: bson.D{
                {Key: "key", Value: 1},
                {Key: "user_id", Value: 1},
            },
            Options: options.Index().SetUnique(true),
        },
    })
    if err != nil {
        return nil, errs.Wrap(err)
    }
    return &ClientConfig{
        coll: coll,
    }, nil
}

type ClientConfig struct {
    coll *mongo.Collection
}

func (x *ClientConfig) Set(ctx context.Context, userID string, config map[string]string) error {
    if len(config) == 0 {
        return nil
    }
    for key, value := range config {
        filter := bson.M{"key": key, "user_id": userID}
        update := bson.M{
            "value": value,
        }
        err := mongoutil.UpdateOne(ctx, x.coll, filter, bson.M{"$set": update}, false, options.Update().SetUpsert(true))
        if err != nil {
```

```
            return err
        }
    }
    return nil
}

func (x *ClientConfig) Get(ctx context.Context, userID string) (map[string]string, error) {
    cs, err := mongoutil.Find[*model.ClientConfig](ctx, x.coll, bson.M{"user_id": userID})
    if err != nil {
        return nil, err
    }
    cm := make(map[string]string)
    for _, config := range cs {
        cm[config.Key] = config.Value
    }
    return cm, nil
}

func (x *ClientConfig) Del(ctx context.Context, userID string, keys []string) error {
    if len(keys) == 0 {
        return nil
    }
    return mongoutil.DeleteMany(ctx, x.coll, bson.M{"key": bson.M{"$in": keys}, "user_id": userID})
}

func (x *ClientConfig) GetPage(ctx context.Context, userID string, key string, pagination pagination.Pagination) (in
    filter := bson.M{}
    if userID != "" {
        filter["user_id"] = userID
    }
    if key != "" {
        filter["key"] = key
    }
    return mongoutil.FindPage[*model.ClientConfig](ctx, x.coll, filter, pagination)
}
```

pkg/common/storage/database/mgo/conversation.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"

	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/errs"
)

func NewConversationMongo(db *mongo.Database) (*ConversationMgo, error) {
	coll := db.Collection(database.ConversationName)
	_, err := coll.Indexes().CreateMany(context.Background(), []mongo.IndexModel{
		{
			Keys: bson.D{
				{Key: "owner_user_id", Value: 1},
				{Key: "conversation_id", Value: 1},
			},
			Options: options.Index().SetUnique(true),
		},
		{
			Keys: bson.D{
				{Key: "user_id", Value: 1},
			},
			Options: options.Index(),
		},
	})
	if err != nil {
		return nil, errs.Wrap(err)
	}
	version, err := NewVersionLog(db.Collection(database.ConversationVersionName))
	if err != nil {
		return nil, err
	}
	return &ConversationMgo{version: version, coll: coll}, nil
}

type ConversationMgo struct {
	version database.VersionLog
	coll    *mongo.Collection
}
```

```
func (c *ConversationMgo) Create(ctx context.Context, conversations []*model.Conversation) (err error) {
	return mongoutil.IncrVersion(func() error {
		return mongoutil.InsertMany(ctx, c.coll, conversations)
	}, func() error {
		userConversation := make(map[string][]string)
		for _, conversation := range conversations {
			userConversation[conversation.OwnerUserID] = append(userConversation[conversation.OwnerUserID], conversation.Con
		}
		for userID, conversationIDs := range userConversation {
			if err := c.version.IncrVersion(ctx, userID, conversationIDs, model.VersionStateInsert); err != nil {
				return err
			}
		}
		return nil
	})
}

func (c *ConversationMgo) UpdateByMap(ctx context.Context, userIDs []string, conversationID string, args map[string]
	if len(args) == 0 || len(userIDs) == 0 {
		return 0, nil
	}
	filter := bson.M{
		"conversation_id": conversationID,
		"owner_user_id":   bson.M{"$in": userIDs},
	}
	var rows int64
	err := mongoutil.IncrVersion(func() error {
		res, err := mongoutil.UpdateMany(ctx, c.coll, filter, bson.M{"$set": args})
		if err != nil {
			return err
		}
		rows = res.ModifiedCount
		return nil
	}, func() error {
		for _, userID := range userIDs {
			if err := c.version.IncrVersion(ctx, userID, []string{conversationID}, model.VersionStateUpdate); err != nil {
				return err
			}
		}
		return nil
	})
	if err != nil {
		return 0, err
	}
	return rows, nil
}

func (c *ConversationMgo) UpdateUserConversations(ctx context.Context, userID string, args map[string]any) ([]*model
	if len(args) == 0 {
		return nil, nil
	}
	filter := bson.M{
		"user_id": userID,
	}

	conversations, err := mongoutil.Find[*model.Conversation](ctx, c.coll, filter, options.Find().SetProjection(bson.M{
	if err != nil {
		return nil, err
	}
	err = mongoutil.IncrVersion(func() error {
		_, err := mongoutil.UpdateMany(ctx, c.coll, filter, bson.M{"$set": args})
		if err != nil {
			return err
		}
		return nil
```

```
	}, func() error {
		for _, conversation := range conversations {
			if err := c.version.IncrVersion(ctx, conversation.OwnerUserID, []string{conversation.ConversationID}, model.Vers
				return err
			}
		}
		return nil
	})
	if err != nil {
		return nil, err
	}
	return conversations, nil
}

func (c *ConversationMgo) Update(ctx context.Context, conversation *model.Conversation) (err error) {
	return mongoutil.IncrVersion(func() error {
		return mongoutil.UpdateOne(ctx, c.coll, bson.M{"owner_user_id": conversation.OwnerUserID, "conversation_id": conv
	}, func() error {
		return c.version.IncrVersion(ctx, conversation.OwnerUserID, []string{conversation.ConversationID}, model.VersionS
	})
}

func (c *ConversationMgo) Find(ctx context.Context, ownerUserID string, conversationIDs []string) (conversations []*
	return mongoutil.Find[*model.Conversation](ctx, c.coll, bson.M{"owner_user_id": ownerUserID, "conversation_id": bso
}

func (c *ConversationMgo) FindUserID(ctx context.Context, userIDs []string, conversationIDs []string) ([]string, err
	return mongoutil.Find[string](
		ctx,
		c.coll,
		bson.M{"owner_user_id": bson.M{"$in": userIDs}, "conversation_id": bson.M{"$in": conversationIDs}},
		options.Find().SetProjection(bson.M{"_id": 0, "owner_user_id": 1}),
	)
}
func (c *ConversationMgo) FindUserIDAllConversationID(ctx context.Context, userID string) ([]string, error) {
	return mongoutil.Find[string](ctx, c.coll, bson.M{"owner_user_id": userID}, options.Find().SetProjection(bson.M{"_i
}

func (c *ConversationMgo) FindUserIDAllNotNotifyConversationID(ctx context.Context, userID string) ([]string, error)
	return mongoutil.Find[string](ctx, c.coll, bson.M{
		"owner_user_id": userID,
		"recv_msg_opt":  constant.ReceiveNotNotifyMessage,
	}, options.Find().SetProjection(bson.M{"_id": 0, "conversation_id": 1}))
}

func (c *ConversationMgo) FindUserIDAllPinnedConversationID(ctx context.Context, userID string) ([]string, error) {
	return mongoutil.Find[string](ctx, c.coll, bson.M{
		"owner_user_id": userID,
		"is_pinned":     true,
	}, options.Find().SetProjection(bson.M{"_id": 0, "conversation_id": 1}))
}

func (c *ConversationMgo) Take(ctx context.Context, userID, conversationID string) (conversation *model.Conversation
	return mongoutil.FindOne[*model.Conversation](ctx, c.coll, bson.M{"owner_user_id": userID, "conversation_id": conve
}

func (c *ConversationMgo) FindConversationID(ctx context.Context, userID string, conversationIDs []string) (existCon
	return mongoutil.Find[string](ctx, c.coll, bson.M{"owner_user_id": userID, "conversation_id": bson.M{"$in": convers
}

func (c *ConversationMgo) FindUserIDAllConversations(ctx context.Context, userID string) (conversations []*model.Con
	return mongoutil.Find[*model.Conversation](ctx, c.coll, bson.M{"owner_user_id": userID})
}

func (c *ConversationMgo) FindRecvMsgUserIDs(ctx context.Context, conversationID string, recvOpts []int) ([]string,
	var filter any
```

```
    if len(recvOpts) == 0 {
        filter = bson.M{"conversation_id": conversationID}
    } else {
        filter = bson.M{"conversation_id": conversationID, "recv_msg_opt": bson.M{"$in": recvOpts}}
    }
    return mongoutil.Find[string](ctx, c.coll, filter, options.Find().SetProjection(bson.M{"_id": 0, "owner_user_id": 1
}

func (c *ConversationMgo) GetUserRecvMsgOpt(ctx context.Context, ownerUserID, conversationID string) (opt int, err e
    return mongoutil.FindOne[int](ctx, c.coll, bson.M{"owner_user_id": ownerUserID, "conversation_id": conversationID},
}

func (c *ConversationMgo) GetAllConversationIDs(ctx context.Context) ([]string, error) {
    return mongoutil.Aggregate[string](ctx, c.coll, []bson.M{
        {"$group": bson.M{"_id": "$conversation_id"}},
        {"$project": bson.M{"_id": 0, "conversation_id": "$_id"}},
    })
}

func (c *ConversationMgo) GetAllConversationIDsNumber(ctx context.Context) (int64, error) {
    counts, err := mongoutil.Aggregate[int64](ctx, c.coll, []bson.M{
        {"$group": bson.M{"_id": "$conversation_id"}},
        {"$group": bson.M{"_id": nil, "count": bson.M{"$sum": 1}}},
        {"$project": bson.M{"_id": 0}},
    })
    if err != nil {
        return 0, err
    }
    if len(counts) == 0 {
        return 0, nil
    }
    return counts[0], nil
}

func (c *ConversationMgo) PageConversationIDs(ctx context.Context, pagination pagination.Pagination) (conversationID
    return mongoutil.FindPageOnly[string](ctx, c.coll, bson.M{}, pagination, options.Find().SetProjection(bson.M{"conve
}

func (c *ConversationMgo) GetConversationsByConversationID(ctx context.Context, conversationIDs []string) ([]*model.
    return mongoutil.Find[*model.Conversation](ctx, c.coll, bson.M{"conversation_id": bson.M{"$in": conversationIDs}})
}

func (c *ConversationMgo) GetConversationIDsNeedDestruct(ctx context.Context) ([]*model.Conversation, error) {
    // "is_msg_destruct = 1 && msg_destruct_time != 0 && (UNIX_TIMESTAMP(NOW()) > (msg_destruct_time + UNIX_TIMESTAMP(l
    return mongoutil.Find[*model.Conversation](ctx, c.coll, bson.M{
        "is_msg_destruct":   1,
        "msg_destruct_time": bson.M{"$ne": 0},
        "$or": []bson.M{
            {
                "$expr": bson.M{
                    "$gt": []any{
                        time.Now(),
                        bson.M{"$add": []any{"$msg_destruct_time", "$latest_msg_destruct_time"}},
                    },
                },
            },
            {
                "latest_msg_destruct_time": nil,
            },
        },
    })
}

func (c *ConversationMgo) GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string) ([]str
    return mongoutil.Find[string](
        ctx,
```

```
    c.coll,
    bson.M{"conversation_id": conversationID, "recv_msg_opt": bson.M{"$ne": constant.ReceiveMessage}},
    options.Find().SetProjection(bson.M{"_id": 0, "owner_user_id": 1}),
  )
}

func (c *ConversationMgo) FindConversationUserVersion(ctx context.Context, userID string, version uint, limit int) (
  return c.version.FindChangeLog(ctx, userID, version, limit)
}

func (c *ConversationMgo) FindRandConversation(ctx context.Context, ts int64, limit int) ([]*model.Conversation, err
  pipeline := []bson.M{
    {
      "$match": bson.M{
        "is_msg_destruct":   true,
        "msg_destruct_time": bson.M{"$ne": 0},
      },
    },
    {
      "$addFields": bson.M{
        "next_msg_destruct_timestamp": bson.M{
          "$add": []any{
            bson.M{
              "$toLong": "$latest_msg_destruct_time",
            },
            bson.M{
              "$multiply": []any{
                "$msg_destruct_time",
                1000, // convert to milliseconds
              },
            },
          },
        },
      },
    },
    {
      "$match": bson.M{
        "next_msg_destruct_timestamp": bson.M{"$lt": ts},
      },
    },
    {
      "$sample": bson.M{
        "size": limit,
      },
    },
  }
  return mongoutil.Aggregate[*model.Conversation](ctx, c.coll, pipeline)
}

func (c *ConversationMgo) DeleteUsersConversations(ctx context.Context, userID string, conversationIDs []string) (er
  if len(conversationIDs) == 0 {
    return nil
  }
  return mongoutil.IncrVersion(func() error {
    err := mongoutil.DeleteMany(ctx, c.coll, bson.M{"owner_user_id": userID, "conversation_id": bson.M{"$in": conversa
    return err
  }, func() error {
    for _, conversationID := range conversationIDs {
      if err := c.version.IncrVersion(ctx, userID, []string{conversationID}, model.VersionStateDelete); err != nil {
        return err
      }
    }
    return nil
  })
}
```

pkg/common/storage/database/mgo/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo // import "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
```

pkg/common/storage/database/mgo/friend.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"go.mongodb.org/mongo-driver/bson/primitive"
	"time"

	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

// FriendMgo implements Friend using MongoDB as the storage backend.
type FriendMgo struct {
	coll  *mongo.Collection
	owner database.VersionLog
}

// NewFriendMongo creates a new instance of FriendMgo with the provided MongoDB database.
func NewFriendMongo(db *mongo.Database) (database.Friend, error) {
	coll := db.Collection(database.FriendName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "owner_user_id", Value: 1},
			{Key: "friend_user_id", Value: 1},
		},
		Options: options.Index().SetUnique(true),
	})
	if err != nil {
		return nil, err
	}
	owner, err := NewVersionLog(db.Collection(database.FriendVersionName))
	if err != nil {
		return nil, err
	}
	return &FriendMgo{coll: coll, owner: owner}, nil
}

func (f *FriendMgo) friendSort() any {
	return bson.D{{"is_pinned", -1}, {"_id", 1}}
}

// Create inserts multiple friend records.
func (f *FriendMgo) Create(ctx context.Context, friends []*model.Friend) error {
	for i, friend := range friends {
		if friend.ID.IsZero() {
```

```
        friends[i].ID = primitive.NewObjectID()
    }
    if friend.CreateTime.IsZero() {
        friends[i].CreateTime = time.Now()
    }
  }
  return mongoutil.IncrVersion(func() error {
    return mongoutil.InsertMany(ctx, f.coll, friends)
  }, func() error {
    mp := make(map[string][]string)
    for _, friend := range friends {
        mp[friend.OwnerUserID] = append(mp[friend.OwnerUserID], friend.FriendUserID)
    }
    for ownerUserID, friendUserIDs := range mp {
        if err := f.owner.IncrVersion(ctx, ownerUserID, friendUserIDs, model.VersionStateInsert); err != nil {
            return err
        }
    }
    return nil
  })
}

// Delete removes specified friends of the owner user.
func (f *FriendMgo) Delete(ctx context.Context, ownerUserID string, friendUserIDs []string) error {
  filter := bson.M{
    "owner_user_id":  ownerUserID,
    "friend_user_id": bson.M{"$in": friendUserIDs},
  }
  return mongoutil.IncrVersion(func() error {
    return mongoutil.DeleteOne(ctx, f.coll, filter)
  }, func() error {
    return f.owner.IncrVersion(ctx, ownerUserID, friendUserIDs, model.VersionStateDelete)
  })
}

// UpdateByMap updates specific fields of a friend document using a map.
func (f *FriendMgo) UpdateByMap(ctx context.Context, ownerUserID string, friendUserID string, args map[string]any) e
  if len(args) == 0 {
    return nil
  }
  filter := bson.M{
    "owner_user_id":  ownerUserID,
    "friend_user_id": friendUserID,
  }
  return mongoutil.IncrVersion(func() error {
    return mongoutil.UpdateOne(ctx, f.coll, filter, bson.M{"$set": args}, true)
  }, func() error {
    var friendUserIDs []string
    if f.IsUpdateIsPinned(args) {
        friendUserIDs = []string{model.VersionSortChangeID, friendUserID}
    } else {
        friendUserIDs = []string{friendUserID}
    }
    return f.owner.IncrVersion(ctx, ownerUserID, friendUserIDs, model.VersionStateUpdate)
  })
}

// UpdateRemark updates the remark for a specific friend.
func (f *FriendMgo) UpdateRemark(ctx context.Context, ownerUserID, friendUserID, remark string) error {
  return f.UpdateByMap(ctx, ownerUserID, friendUserID, map[string]any{"remark": remark})
}

func (f *FriendMgo) fillTime(friends ...*model.Friend) {
  for i, friend := range friends {
    if friend.CreateTime.IsZero() {
        friends[i].CreateTime = friend.ID.Timestamp()
```

```
    }
  }
}

func (f *FriendMgo) findOne(ctx context.Context, filter any) (*model.Friend, error) {
  friend, err := mongoutil.FindOne[*model.Friend](ctx, f.coll, filter)
  if err != nil {
    return nil, err
  }
  f.fillTime(friend)
  return friend, nil
}

func (f *FriendMgo) find(ctx context.Context, filter any) ([]*model.Friend, error) {
  friends, err := mongoutil.Find[*model.Friend](ctx, f.coll, filter)
  if err != nil {
    return nil, err
  }
  f.fillTime(friends...)
  return friends, nil
}

func (f *FriendMgo) findPage(ctx context.Context, filter any, pagination pagination.Pagination, opts ...*options.Fin
  return mongoutil.FindPage[*model.Friend](ctx, f.coll, filter, pagination, opts...)
}

// Take retrieves a single friend document. Returns an error if not found.
func (f *FriendMgo) Take(ctx context.Context, ownerUserID, friendUserID string) (*model.Friend, error) {
  filter := bson.M{
    "owner_user_id":  ownerUserID,
    "friend_user_id": friendUserID,
  }
  return f.findOne(ctx, filter)
}

// FindUserState finds the friendship status between two users.
func (f *FriendMgo) FindUserState(ctx context.Context, userID1, userID2 string) ([]*model.Friend, error) {
  filter := bson.M{
    "$or": []bson.M{
      {"owner_user_id": userID1, "friend_user_id": userID2},
      {"owner_user_id": userID2, "friend_user_id": userID1},
    },
  }
  return f.find(ctx, filter)
}

// FindFriends retrieves a list of friends for a given owner. Missing friends do not cause an error.
func (f *FriendMgo) FindFriends(ctx context.Context, ownerUserID string, friendUserIDs []string) ([]*model.Friend, e
  filter := bson.M{
    "owner_user_id":  ownerUserID,
    "friend_user_id": bson.M{"$in": friendUserIDs},
  }
  return f.find(ctx, filter)
}

// FindReversalFriends finds users who have added the specified user as a friend.
func (f *FriendMgo) FindReversalFriends(ctx context.Context, friendUserID string, ownerUserIDs []string) ([]*model.F
  filter := bson.M{
    "owner_user_id":  bson.M{"$in": ownerUserIDs},
    "friend_user_id": friendUserID,
  }
  return f.find(ctx, filter)
}

// FindOwnerFriends retrieves a paginated list of friends for a given owner.
func (f *FriendMgo) FindOwnerFriends(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (int
```

```
    filter := bson.M{"owner_user_id": ownerUserID}
    opt := options.Find().SetSort(f.friendSort())
    return f.findPage(ctx, filter, pagination, opt)
}

func (f *FriendMgo) FindOwnerFriendUserIds(ctx context.Context, ownerUserID string, limit int) ([]string, error) {
    filter := bson.M{"owner_user_id": ownerUserID}
    opt := options.Find().SetProjection(bson.M{"_id": 0, "friend_user_id": 1}).SetSort(f.friendSort()).SetLimit(int64(l
    return mongoutil.Find[string](ctx, f.coll, filter, opt)
}

// FindInWhoseFriends finds users who have added the specified user as a friend, with pagination.
func (f *FriendMgo) FindInWhoseFriends(ctx context.Context, friendUserID string, pagination pagination.Pagination) (
    filter := bson.M{"friend_user_id": friendUserID}
    opt := options.Find().SetSort(f.friendSort())
    return f.findPage(ctx, filter, pagination, opt)
}

// FindFriendUserIDs retrieves a list of friend user IDs for a given owner.
func (f *FriendMgo) FindFriendUserIDs(ctx context.Context, ownerUserID string) ([]string, error) {
    filter := bson.M{"owner_user_id": ownerUserID}
    return mongoutil.Find[string](ctx, f.coll, filter, options.Find().SetProjection(bson.M{"_id": 0, "friend_user_id":
}

func (f *FriendMgo) UpdateFriends(ctx context.Context, ownerUserID string, friendUserIDs []string, val map[string]an
    // Ensure there are IDs to update
    if len(friendUserIDs) == 0 || len(val) == 0 {
        return nil // Or return an error if you expect there to always be IDs
    }

    // Create a filter to match documents with the specified ownerUserID and any of the friendUserIDs
    filter := bson.M{
        "owner_user_id":  ownerUserID,
        "friend_user_id": bson.M{"$in": friendUserIDs},
    }

    // Create an update document
    update := bson.M{"$set": val}

    return mongoutil.IncrVersion(func() error {
        return mongoutil.Ignore(mongoutil.UpdateMany(ctx, f.coll, filter, update))
    }, func() error {
        var userIDs []string
        if f.IsUpdateIsPinned(val) {
            userIDs = append([]string{model.VersionSortChangeID}, friendUserIDs...)
        } else {
            userIDs = friendUserIDs
        }
        return f.owner.IncrVersion(ctx, ownerUserID, userIDs, model.VersionStateUpdate)
    })
}

func (f *FriendMgo) FindIncrVersion(ctx context.Context, ownerUserID string, version uint, limit int) (*model.Versio
    return f.owner.FindChangeLog(ctx, ownerUserID, version, limit)
}

func (f *FriendMgo) FindFriendUserID(ctx context.Context, friendUserID string) ([]string, error) {
    filter := bson.M{
        "friend_user_id": friendUserID,
    }
    return mongoutil.Find[string](ctx, f.coll, filter, options.Find().SetProjection(bson.M{"_id": 0, "owner_user_id": 1
}

func (f *FriendMgo) IncrVersion(ctx context.Context, ownerUserID string, friendUserIDs []string, state int32) error
    return f.owner.IncrVersion(ctx, ownerUserID, friendUserIDs, state)
}
```

```
func (f *FriendMgo) IsUpdateIsPinned(data map[string]any) bool {
    if data == nil {
        return false
    }
    _, ok := data["is_pinned"]
    return ok
}
```

pkg/common/storage/database/mgo/friend_request.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"time"

	"go.mongodb.org/mongo-driver/mongo/options"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"

	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
)

func NewFriendRequestMongo(db *mongo.Database) (database.FriendRequest, error) {
	coll := db.Collection(database.FriendRequestName)
	_, err := coll.Indexes().CreateMany(context.Background(), []mongo.IndexModel{
		{
			Keys: bson.D{
				{Key: "from_user_id", Value: 1},
				{Key: "to_user_id", Value: 1},
			},
			Options: options.Index().SetUnique(true),
		},
		{
			Keys: bson.D{
				{Key: "create_time", Value: -1},
			},
		},
	})
	if err != nil {
		return nil, err
	}
	return &FriendRequestMgo{coll: coll}, nil
}

type FriendRequestMgo struct {
	coll *mongo.Collection
}

func (f *FriendRequestMgo) sort() any {
	return bson.D{{Key: "create_time", Value: -1}}
}

func (f *FriendRequestMgo) FindToUserID(ctx context.Context, toUserID string, handleResults []int, pagination pagina
	filter := bson.M{"to_user_id": toUserID}
```

```
    if len(handleResults) > 0 {
        filter["handle_result"] = bson.M{"$in": handleResults}
    }
    return mongoutil.FindPage[*model.FriendRequest](ctx, f.coll, filter, pagination, options.Find().SetSort(f.sort()))
}

func (f *FriendRequestMgo) FindFromUserID(ctx context.Context, fromUserID string, handleResults []int, pagination pa
    filter := bson.M{"from_user_id": fromUserID}
    if len(handleResults) > 0 {
        filter["handle_result"] = bson.M{"$in": handleResults}
    }
    return mongoutil.FindPage[*model.FriendRequest](ctx, f.coll, filter, pagination, options.Find().SetSort(f.sort()))
}

func (f *FriendRequestMgo) FindBothFriendRequests(ctx context.Context, fromUserID, toUserID string) (friends []*mode
    filter := bson.M{"$or": []bson.M{
        {"from_user_id": fromUserID, "to_user_id": toUserID},
        {"from_user_id": toUserID, "to_user_id": fromUserID},
    }}
    return mongoutil.Find[*model.FriendRequest](ctx, f.coll, filter)
}

func (f *FriendRequestMgo) Create(ctx context.Context, friendRequests []*model.FriendRequest) error {
    return mongoutil.InsertMany(ctx, f.coll, friendRequests)
}

func (f *FriendRequestMgo) Delete(ctx context.Context, fromUserID, toUserID string) (err error) {
    return mongoutil.DeleteOne(ctx, f.coll, bson.M{"from_user_id": fromUserID, "to_user_id": toUserID})
}

func (f *FriendRequestMgo) UpdateByMap(ctx context.Context, formUserID, toUserID string, args map[string]any) (err e
    if len(args) == 0 {
        return nil
    }
    return mongoutil.UpdateOne(ctx, f.coll, bson.M{"from_user_id": formUserID, "to_user_id": toUserID}, bson.M{"$set":
}

func (f *FriendRequestMgo) Update(ctx context.Context, friendRequest *model.FriendRequest) (err error) {
    updater := bson.M{}
    if friendRequest.HandleResult != 0 {
        updater["handle_result"] = friendRequest.HandleResult
    }
    if friendRequest.ReqMsg != "" {
        updater["req_msg"] = friendRequest.ReqMsg
    }
    if friendRequest.HandlerUserID != "" {
        updater["handler_user_id"] = friendRequest.HandlerUserID
    }
    if friendRequest.HandleMsg != "" {
        updater["handle_msg"] = friendRequest.HandleMsg
    }
    if !friendRequest.HandleTime.IsZero() {
        updater["handle_time"] = friendRequest.HandleTime
    }
    if friendRequest.Ex != "" {
        updater["ex"] = friendRequest.Ex
    }
    if len(updater) == 0 {
        return nil
    }
    filter := bson.M{"from_user_id": friendRequest.FromUserID, "to_user_id": friendRequest.ToUserID}
    return mongoutil.UpdateOne(ctx, f.coll, filter, bson.M{"$set": updater}, true)
}

func (f *FriendRequestMgo) Find(ctx context.Context, fromUserID, toUserID string) (friendRequest *model.FriendReques
    return mongoutil.FindOne[*model.FriendRequest](ctx, f.coll, bson.M{"from_user_id": fromUserID, "to_user_id": toUser
```

```
}

func (f *FriendRequestMgo) Take(ctx context.Context, fromUserID, toUserID string) (friendRequest *model.FriendReques
    return f.Find(ctx, fromUserID, toUserID)
}

func (f *FriendRequestMgo) GetUnhandledCount(ctx context.Context, userID string, ts int64) (int64, error) {
    filter := bson.M{"to_user_id": userID, "handle_result": 0}
    if ts != 0 {
        filter["create_time"] = bson.M{"$gt": time.UnixMilli(ts)}
    }
    return mongoutil.Count(ctx, f.coll, filter)
}
```

pkg/common/storage/database/mgo/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"time"

	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/errs"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewGroupMongo(db *mongo.Database) (database.Group, error) {
	coll := db.Collection(database.GroupName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "group_id", Value: 1},
		},
		Options: options.Index().SetUnique(true),
	})
	if err != nil {
		return nil, errs.Wrap(err)
	}
	return &GroupMgo{coll: coll}, nil
}

type GroupMgo struct {
	coll *mongo.Collection
}

func (g *GroupMgo) sortGroup() any {
	return bson.D{{"group_name", 1}, {"create_time", 1}}
}

func (g *GroupMgo) Create(ctx context.Context, groups []*model.Group) (err error) {
	return mongoutil.InsertMany(ctx, g.coll, groups)
}

func (g *GroupMgo) UpdateStatus(ctx context.Context, groupID string, status int32) (err error) {
	return g.UpdateMap(ctx, groupID, map[string]any{"status": status})
}

func (g *GroupMgo) UpdateMap(ctx context.Context, groupID string, args map[string]any) (err error) {
	if len(args) == 0 {
		return nil
```

```
	}
	return mongoutil.UpdateOne(ctx, g.coll, bson.M{"group_id": groupID}, bson.M{"$set": args}, true)
}

func (g *GroupMgo) Find(ctx context.Context, groupIDs []string) (groups []*model.Group, err error) {
	return mongoutil.Find[*model.Group](ctx, g.coll, bson.M{"group_id": bson.M{"$in": groupIDs}})
}

func (g *GroupMgo) Take(ctx context.Context, groupID string) (group *model.Group, err error) {
	return mongoutil.FindOne[*model.Group](ctx, g.coll, bson.M{"group_id": groupID})
}

func (g *GroupMgo) Search(ctx context.Context, keyword string, pagination pagination.Pagination) (total int64, group
	// Define the sorting options
	opts := options.Find().SetSort(bson.D{{Key: "create_time", Value: -1}})

	// Perform the search with pagination and sorting
	return mongoutil.FindPage[*model.Group](ctx, g.coll, bson.M{
		"group_name": bson.M{"$regex": keyword},
		"status":     bson.M{"$ne": constant.GroupStatusDismissed},
	}, pagination, opts)
}

func (g *GroupMgo) CountTotal(ctx context.Context, before *time.Time) (count int64, err error) {
	if before == nil {
		return mongoutil.Count(ctx, g.coll, bson.M{})
	}
	return mongoutil.Count(ctx, g.coll, bson.M{"create_time": bson.M{"$lt": before}})
}

func (g *GroupMgo) CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int64, e
	pipeline := bson.A{
		bson.M{
			"$match": bson.M{
				"create_time": bson.M{
					"$gte": start,
					"$lt":  end,
				},
			},
		},
		bson.M{
			"$group": bson.M{
				"_id": bson.M{
					"$dateToString": bson.M{
						"format": "%Y-%m-%d",
						"date":   "$create_time",
					},
				},
				"count": bson.M{
					"$sum": 1,
				},
			},
		},
	}
	type Item struct {
		Date  string `bson:"_id"`
		Count int64  `bson:"count"`
	}
	items, err := mongoutil.Aggregate[Item](ctx, g.coll, pipeline)
	if err != nil {
		return nil, err
	}
	res := make(map[string]int64, len(items))
	for _, item := range items {
		res[item.Date] = item.Count
	}
```

```
    return res, nil
}

func (g *GroupMgo) FindJoinSortGroupID(ctx context.Context, groupIDs []string) ([]string, error) {
    if len(groupIDs) < 2 {
        return groupIDs, nil
    }
    filter := bson.M{
        "group_id": bson.M{"$in": groupIDs},
        "status":   bson.M{"$ne": constant.GroupStatusDismissed},
    }
    opt := options.Find().SetSort(g.sortGroup()).SetProjection(bson.M{"_id": 0, "group_id": 1})
    return mongoutil.Find[string](ctx, g.coll, filter, opt)
}

func (g *GroupMgo) SearchJoin(ctx context.Context, groupIDs []string, keyword string, pagination pagination.Paginati
    if len(groupIDs) == 0 {
        return 0, nil, nil
    }
    filter := bson.M{
        "group_id": bson.M{"$in": groupIDs},
        "status":   bson.M{"$ne": constant.GroupStatusDismissed},
    }
    if keyword != "" {
        filter["group_name"] = bson.M{"$regex": keyword}
    }
    // Define the sorting options
    opts := options.Find().SetSort(g.sortGroup())
    // Perform the search with pagination and sorting
    return mongoutil.FindPage[*model.Group](ctx, g.coll, filter, pagination, opts)
}
```

pkg/common/storage/database/mgo/group_member.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/log"

	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/errs"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewGroupMember(db *mongo.Database) (database.GroupMember, error) {
	coll := db.Collection(database.GroupMemberName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "group_id", Value: 1},
			{Key: "user_id", Value: 1},
		},
		Options: options.Index().SetUnique(true),
	})
	if err != nil {
		return nil, errs.Wrap(err)
	}
	member, err := NewVersionLog(db.Collection(database.GroupMemberVersionName))
	if err != nil {
		return nil, err
	}
	join, err := NewVersionLog(db.Collection(database.GroupJoinVersionName))
	if err != nil {
		return nil, err
	}
	return &GroupMemberMgo{coll: coll, member: member, join: join}, nil
}

type GroupMemberMgo struct {
	coll   *mongo.Collection
	member database.VersionLog
	join   database.VersionLog
}

func (g *GroupMemberMgo) memberSort() any {
	return bson.D{{Key: "role_level", Value: -1}, {Key: "create_time", Value: 1}}
}
```

```go
func (g *GroupMemberMgo) Create(ctx context.Context, groupMembers []*model.GroupMember) (err error) {
    return mongoutil.IncrVersion(func() error {
        return mongoutil.InsertMany(ctx, g.coll, groupMembers)
    }, func() error {
        gms := make(map[string][]string)
        for _, member := range groupMembers {
            gms[member.GroupID] = append(gms[member.GroupID], member.UserID)
        }
        for groupID, userIDs := range gms {
            if err := g.member.IncrVersion(ctx, groupID, userIDs, model.VersionStateInsert); err != nil {
                return err
            }
        }
        return nil
    }, func() error {
        gms := make(map[string][]string)
        for _, member := range groupMembers {
            gms[member.UserID] = append(gms[member.UserID], member.GroupID)
        }
        for userID, groupIDs := range gms {
            if err := g.join.IncrVersion(ctx, userID, groupIDs, model.VersionStateInsert); err != nil {
                return err
            }
        }
        return nil
    })
}

func (g *GroupMemberMgo) Delete(ctx context.Context, groupID string, userIDs []string) (err error) {
    filter := bson.M{"group_id": groupID}
    if len(userIDs) > 0 {
        filter["user_id"] = bson.M{"$in": userIDs}
    }
    return mongoutil.IncrVersion(func() error {
        return mongoutil.DeleteMany(ctx, g.coll, filter)
    }, func() error {
        if len(userIDs) == 0 {
            return g.member.Delete(ctx, groupID)
        } else {
            return g.member.IncrVersion(ctx, groupID, userIDs, model.VersionStateDelete)
        }
    }, func() error {
        for _, userID := range userIDs {
            if err := g.join.IncrVersion(ctx, userID, []string{groupID}, model.VersionStateDelete); err != nil {
                return err
            }
        }
        return nil
    })
}

func (g *GroupMemberMgo) UpdateRoleLevel(ctx context.Context, groupID string, userID string, roleLevel int32) error
    return mongoutil.IncrVersion(func() error {
        return mongoutil.UpdateOne(ctx, g.coll, bson.M{"group_id": groupID, "user_id": userID},
            bson.M{"$set": bson.M{"role_level": roleLevel}}, true)
    }, func() error {
        return g.member.IncrVersion(ctx, groupID, []string{model.VersionSortChangeID, userID}, model.VersionStateUpdate)
    })
}
func (g *GroupMemberMgo) UpdateUserRoleLevels(ctx context.Context, groupID string, firstUserID string, firstUserRole
    return mongoutil.IncrVersion(func() error {
        if err := mongoutil.UpdateOne(ctx, g.coll, bson.M{"group_id": groupID, "user_id": firstUserID},
            bson.M{"$set": bson.M{"role_level": firstUserRoleLevel}}, true); err != nil {
            return err
        }
```

```
    if err := mongoutil.UpdateOne(ctx, g.coll, bson.M{"group_id": groupID, "user_id": secondUserID},
        bson.M{"$set": bson.M{"role_level": secondUserRoleLevel}}, true); err != nil {
        return err
    }
    return nil
}, func() error {
    return g.member.IncrVersion(ctx, groupID, []string{model.VersionSortChangeID, firstUserID, secondUserID}, model.Ve
})
}

func (g *GroupMemberMgo) Update(ctx context.Context, groupID string, userID string, data map[string]any) (err error)
if len(data) == 0 {
    return nil
}
return mongoutil.IncrVersion(func() error {
    return mongoutil.UpdateOne(ctx, g.coll, bson.M{"group_id": groupID, "user_id": userID}, bson.M{"$set": data}, true
}, func() error {
    var userIDs []string
    if g.IsUpdateRoleLevel(data) {
        userIDs = []string{model.VersionSortChangeID, userID}
    } else {
        userIDs = []string{userID}
    }
    return g.member.IncrVersion(ctx, groupID, userIDs, model.VersionStateUpdate)
})
}

func (g *GroupMemberMgo) FindMemberUserID(ctx context.Context, groupID string) (userIDs []string, err error) {
return mongoutil.Find[string](ctx, g.coll, bson.M{"group_id": groupID}, options.Find().SetProjection(bson.M{"_id":
}

func (g *GroupMemberMgo) Find(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupMember, error) {
filter := bson.M{"group_id": groupID}
if len(userIDs) > 0 {
    filter["user_id"] = bson.M{"$in": userIDs}
}
return mongoutil.Find[*model.GroupMember](ctx, g.coll, filter)
}

func (g *GroupMemberMgo) FindInGroup(ctx context.Context, userID string, groupIDs []string) ([]*model.GroupMember, e
filter := bson.M{"user_id": userID}
if len(groupIDs) > 0 {
    filter["group_id"] = bson.M{"$in": groupIDs}
}
return mongoutil.Find[*model.GroupMember](ctx, g.coll, filter)
}

func (g *GroupMemberMgo) Take(ctx context.Context, groupID string, userID string) (groupMember *model.GroupMember, e
return mongoutil.FindOne[*model.GroupMember](ctx, g.coll, bson.M{"group_id": groupID, "user_id": userID})
}

func (g *GroupMemberMgo) TakeOwner(ctx context.Context, groupID string) (groupMember *model.GroupMember, err error)
return mongoutil.FindOne[*model.GroupMember](ctx, g.coll, bson.M{"group_id": groupID, "role_level": constant.GroupO
}

func (g *GroupMemberMgo) FindRoleLevelUserIDs(ctx context.Context, groupID string, roleLevel int32) ([]string, error
return mongoutil.Find[string](ctx, g.coll, bson.M{"group_id": groupID, "role_level": roleLevel}, options.Find().Set
}

func (g *GroupMemberMgo) SearchMember(ctx context.Context, keyword string, groupID string, pagination pagination.Pag
filter := bson.M{"group_id": groupID, "nickname": bson.M{"$regex": keyword}}
return mongoutil.FindPage[*model.GroupMember](ctx, g.coll, filter, pagination, options.Find().SetSort(g.memberSort(
}

func (g *GroupMemberMgo) FindUserJoinedGroupID(ctx context.Context, userID string) (groupIDs []string, err error) {
return mongoutil.Find[string](ctx, g.coll, bson.M{"user_id": userID}, options.Find().SetProjection(bson.M{"_id": 0,
```

```
}

func (g *GroupMemberMgo) TakeGroupMemberNum(ctx context.Context, groupID string) (count int64, err error) {
    return mongoutil.Count(ctx, g.coll, bson.M{"group_id": groupID})
}

func (g *GroupMemberMgo) FindUserManagedGroupID(ctx context.Context, userID string) (groupIDs []string, err error) {
    filter := bson.M{
        "user_id": userID,
        "role_level": bson.M{
            "$in": []int{constant.GroupOwner, constant.GroupAdmin},
        },
    }
    return mongoutil.Find[string](ctx, g.coll, filter, options.Find().SetProjection(bson.M{"_id": 0, "group_id": 1}))
}

func (g *GroupMemberMgo) IsUpdateRoleLevel(data map[string]any) bool {
    if len(data) == 0 {
        return false
    }
    _, ok := data["role_level"]
    return ok
}

func (g *GroupMemberMgo) JoinGroupIncrVersion(ctx context.Context, userID string, groupIDs []string, state int32) er
    return g.join.IncrVersion(ctx, userID, groupIDs, state)
}

func (g *GroupMemberMgo) MemberGroupIncrVersion(ctx context.Context, groupID string, userIDs []string, state int32)
    return g.member.IncrVersion(ctx, groupID, userIDs, state)
}

func (g *GroupMemberMgo) FindMemberIncrVersion(ctx context.Context, groupID string, version uint, limit int) (*model
    log.ZDebug(ctx, "find member incr version", "groupID", groupID, "version", version)
    return g.member.FindChangeLog(ctx, groupID, version, limit)
}

func (g *GroupMemberMgo) BatchFindMemberIncrVersion(ctx context.Context, groupIDs []string, versions []uint, limits
    log.ZDebug(ctx, "Batch find member incr version", "groupIDs", groupIDs, "versions", versions)
    return g.member.BatchFindChangeLog(ctx, groupIDs, versions, limits)
}

func (g *GroupMemberMgo) FindJoinIncrVersion(ctx context.Context, userID string, version uint, limit int) (*model.Ve
    log.ZDebug(ctx, "find join incr version", "userID", userID, "version", version)
    return g.join.FindChangeLog(ctx, userID, version, limit)
}
```

pkg/common/storage/database/mgo/group_request.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/tools/utils/datautil"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

    "github.com/openimsdk/tools/db/mongoutil"
    "github.com/openimsdk/tools/db/pagination"
    "github.com/openimsdk/tools/errs"
)

func NewGroupRequestMgo(db *mongo.Database) (database.GroupRequest, error) {
    coll := db.Collection(database.GroupRequestName)
    _, err := coll.Indexes().CreateMany(context.Background(), []mongo.IndexModel{
        {
            Keys: bson.D{
                {Key: "group_id", Value: 1},
                {Key: "user_id", Value: 1},
            },
            Options: options.Index().SetUnique(true),
        },
        {
            Keys: bson.D{
                {Key: "req_time", Value: -1},
            },
        },
    })
    if err != nil {
        return nil, errs.Wrap(err)
    }
    return &GroupRequestMgo{coll: coll}, nil
}

type GroupRequestMgo struct {
    coll *mongo.Collection
}

func (g *GroupRequestMgo) Create(ctx context.Context, groupRequests []*model.GroupRequest) (err error) {
    return mongoutil.InsertMany(ctx, g.coll, groupRequests)
}

func (g *GroupRequestMgo) Delete(ctx context.Context, groupID string, userID string) (err error) {
```

```
    return mongoutil.DeleteOne(ctx, g.coll, bson.M{"group_id": groupID, "user_id": userID})
}

func (g *GroupRequestMgo) UpdateHandler(ctx context.Context, groupID string, userID string, handledMsg string, handl
    return mongoutil.UpdateOne(ctx, g.coll, bson.M{"group_id": groupID, "user_id": userID}, bson.M{"$set": bson.M{"hand
}

func (g *GroupRequestMgo) Take(ctx context.Context, groupID string, userID string) (groupRequest *model.GroupRequest
    return mongoutil.FindOne[*model.GroupRequest](ctx, g.coll, bson.M{"group_id": groupID, "user_id": userID})
}

func (g *GroupRequestMgo) FindGroupRequests(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupRe
    return mongoutil.Find[*model.GroupRequest](ctx, g.coll, bson.M{"group_id": groupID, "user_id": bson.M{"$in": userID
}

func (g *GroupRequestMgo) sort() any {
    return bson.D{{Key: "req_time", Value: -1}}
}

func (g *GroupRequestMgo) Page(ctx context.Context, userID string, groupIDs []string, handleResults []int, paginatio
    filter := bson.M{"user_id": userID}
    if len(groupIDs) > 0 {
        filter["group_id"] = bson.M{"$in": datautil.Distinct(groupIDs)}
    }
    if len(handleResults) > 0 {
        filter["handle_result"] = bson.M{"$in": handleResults}
    }
    return mongoutil.FindPage[*model.GroupRequest](ctx, g.coll, filter, pagination, options.Find().SetSort(g.sort()))
}

func (g *GroupRequestMgo) PageGroup(ctx context.Context, groupIDs []string, handleResults []int, pagination paginati
    if len(groupIDs) == 0 {
        return 0, nil, nil
    }
    filter := bson.M{"group_id": bson.M{"$in": groupIDs}}
    if len(handleResults) > 0 {
        filter["handle_result"] = bson.M{"$in": handleResults}
    }
    return mongoutil.FindPage[*model.GroupRequest](ctx, g.coll, filter, pagination, options.Find().SetSort(g.sort()))
}

func (g *GroupRequestMgo) GetUnhandledCount(ctx context.Context, groupIDs []string, ts int64) (int64, error) {
    if len(groupIDs) == 0 {
        return 0, nil
    }
    filter := bson.M{"group_id": bson.M{"$in": groupIDs}, "handle_result": 0}
    if ts != 0 {
        filter["req_time"] = bson.M{"$gt": time.UnixMilli(ts)}
    }
    return mongoutil.Count(ctx, g.coll, filter)
}
```

pkg/common/storage/database/mgo/helpers.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
    "github.com/openimsdk/tools/errs"
    "go.mongodb.org/mongo-driver/mongo"
)

func IsNotFound(err error) bool {
    return errs.Unwrap(err) == mongo.ErrNoDocuments
}
```

pkg/common/storage/database/mgo/log.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"time"

	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewLogMongo(db *mongo.Database) (database.Log, error) {
	coll := db.Collection(database.LogName)
	_, err := coll.Indexes().CreateMany(context.Background(), []mongo.IndexModel{
		{
			Keys: bson.D{
				{Key: "log_id", Value: 1},
			},
			Options: options.Index().SetUnique(true),
		},
		{
			Keys: bson.D{
				{Key: "user_id", Value: 1},
			},
		},
		{
			Keys: bson.D{
				{Key: "create_time", Value: -1},
			},
		},
	})
	if err != nil {
		return nil, err
	}
	return &LogMgo{coll: coll}, nil
}

type LogMgo struct {
	coll *mongo.Collection
}

func (l *LogMgo) Create(ctx context.Context, log []*model.Log) error {
	return mongoutil.InsertMany(ctx, l.coll, log)
}

func (l *LogMgo) Search(ctx context.Context, keyword string, start time.Time, end time.Time, pagination pagination.P
```

```
    filter := bson.M{"create_time": bson.M{"$gte": start, "$lte": end}}
    if keyword != "" {
        filter["user_id"] = bson.M{"$regex": keyword}
    }
    return mongoutil.FindPage[*model.Log](ctx, l.coll, filter, pagination, options.Find().SetSort(bson.M{"create_time":
}

func (l *LogMgo) Delete(ctx context.Context, logID []string, userID string) error {
    if userID == "" {
        return mongoutil.DeleteMany(ctx, l.coll, bson.M{"log_id": bson.M{"$in": logID}})
    }
    return mongoutil.DeleteMany(ctx, l.coll, bson.M{"log_id": bson.M{"$in": logID}, "user_id": userID})
}

func (l *LogMgo) Get(ctx context.Context, logIDs []string, userID string) ([]*model.Log, error) {
    if userID == "" {
        return mongoutil.Find[*model.Log](ctx, l.coll, bson.M{"log_id": bson.M{"$in": logIDs}})
    }
    return mongoutil.Find[*model.Log](ctx, l.coll, bson.M{"log_id": bson.M{"$in": logIDs}, "user_id": userID})
}
```

pkg/common/storage/database/mgo/msg.go

```
package mgo

import (
	"context"
	"fmt"
	"time"

	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/bson/primitive"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/jsonutil"
)

func NewMsgMongo(db *mongo.Database) (database.Msg, error) {
	coll := db.Collection(new(model.MsgDocModel).TableName())
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "doc_id", Value: 1},
		},
		Options: options.Index().SetUnique(true),
	})
	if err != nil {
		return nil, errs.Wrap(err)
	}
	return &MsgMgo{coll: coll}, nil
}

type MsgMgo struct {
	coll  *mongo.Collection
	model model.MsgDocModel
}

func (m *MsgMgo) Create(ctx context.Context, msg *model.MsgDocModel) error {
	return mongoutil.InsertMany(ctx, m.coll, []*model.MsgDocModel{msg})
}

func (m *MsgMgo) UpdateMsg(ctx context.Context, docID string, index int64, key string, value any) (*mongo.UpdateResu
	var field string
	if key == "" {
		field = fmt.Sprintf("msgs.%d", index)
	} else {
		field = fmt.Sprintf("msgs.%d.%s", index, key)
	}
	filter := bson.M{"doc_id": docID}
	update := bson.M{"$set": bson.M{field: value}}
	return mongoutil.UpdateOneResult(ctx, m.coll, filter, update)
}

func (m *MsgMgo) PushUnique(ctx context.Context, docID string, index int64, key string, value any) (*mongo.UpdateRes
	var field string
	if key == "" {
		field = fmt.Sprintf("msgs.%d", index)
	} else {
		field = fmt.Sprintf("msgs.%d.%s", index, key)
```

```
    }
    filter := bson.M{"doc_id": docID}
    update := bson.M{
        "$addToSet": bson.M{
            field: bson.M{"$each": value},
        },
    }
    return mongoutil.UpdateOneResult(ctx, m.coll, filter, update)
}

func (m *MsgMgo) FindOneByDocID(ctx context.Context, docID string) (*model.MsgDocModel, error) {
    return mongoutil.FindOne[*model.MsgDocModel](ctx, m.coll, bson.M{"doc_id": docID})
}

func (m *MsgMgo) GetMsgBySeqIndexIn1Doc(ctx context.Context, userID, docID string, seqs []int64) ([]*model.MsgInfoMo
    msgs, err := m.getMsgBySeqIndexIn1Doc(ctx, userID, docID, seqs)
    if err != nil {
        return nil, err
    }
    if len(msgs) == len(seqs) {
        return msgs, nil
    }
    tmp := make(map[int64]*model.MsgInfoModel)
    for i, val := range msgs {
        tmp[val.Msg.Seq] = msgs[i]
    }
    res := make([]*model.MsgInfoModel, 0, len(seqs))
    for _, seq := range seqs {
        if val, ok := tmp[seq]; ok {
            res = append(res, val)
        } else {
            res = append(res, &model.MsgInfoModel{Msg: &model.MsgDataModel{Seq: seq}})
        }
    }
    return res, nil
}

func (m *MsgMgo) getMsgBySeqIndexIn1Doc(ctx context.Context, userID, docID string, seqs []int64) ([]*model.MsgInfoMo
    indexes := make([]int64, 0, len(seqs))
    for _, seq := range seqs {
        indexes = append(indexes, m.model.GetMsgIndex(seq))
    }
    pipeline := mongo.Pipeline{
        bson.D{{Key: "$match", Value: bson.D{
            {Key: "doc_id", Value: docID},
        }}},
        bson.D{{Key: "$project", Value: bson.D{
            {Key: "_id", Value: 0},
            {Key: "doc_id", Value: 1},
            {Key: "msgs", Value: bson.D{
                {Key: "$map", Value: bson.D{
                    {Key: "input", Value: indexes},
                    {Key: "as", Value: "index"},
                    {Key: "in", Value: bson.D{
                        {Key: "$arrayElemAt", Value: bson.A{"$msgs", "$$index"}},
                    }},
                }},
            }},
        }}},
    }
    msgDocModel, err := mongoutil.Aggregate[*model.MsgDocModel](ctx, m.coll, pipeline)
    if err != nil {
        return nil, err
    }
    if len(msgDocModel) == 0 {
        return nil, errs.Wrap(mongo.ErrNoDocuments)
```

```
	}
	msgs := make([]*model.MsgInfoModel, 0, len(msgDocModel[0].Msg))
	for i := range msgDocModel[0].Msg {
		msg := msgDocModel[0].Msg[i]
		if msg == nil || msg.Msg == nil {
			continue
		}
		if datautil.Contain(userID, msg.DelList...) {
			msg.Msg.Content = ""
			msg.Msg.Status = constant.MsgDeleted
		}
		if msg.Revoke != nil {
			revokeContent := sdkws.MessageRevokedContent{
				RevokerID:                  msg.Revoke.UserID,
				RevokerRole:                msg.Revoke.Role,
				ClientMsgID:                msg.Msg.ClientMsgID,
				RevokerNickname:            msg.Revoke.Nickname,
				RevokeTime:                 msg.Revoke.Time,
				SourceMessageSendTime:      msg.Msg.SendTime,
				SourceMessageSendID:        msg.Msg.SendID,
				SourceMessageSenderNickname: msg.Msg.SenderNickname,
				SessionType:                msg.Msg.SessionType,
				Seq:                        msg.Msg.Seq,
				Ex:                         msg.Msg.Ex,
			}
			data, err := jsonutil.JsonMarshal(&revokeContent)
			if err != nil {
				return nil, errs.WrapMsg(err, fmt.Sprintf("docID is %s, seqs is %v", docID, seqs))
			}
			elem := sdkws.NotificationElem{
				Detail: string(data),
			}
			content, err := jsonutil.JsonMarshal(&elem)
			if err != nil {
				return nil, errs.WrapMsg(err, fmt.Sprintf("docID is %s, seqs is %v", docID, seqs))
			}
			msg.Msg.ContentType = constant.MsgRevokeNotification
			msg.Msg.Content = string(content)
		}
		msgs = append(msgs, msg)
	}
	return msgs, nil
}

func (m *MsgMgo) GetNewestMsg(ctx context.Context, conversationID string) (*model.MsgInfoModel, error) {
	for skip := int64(0); ; skip++ {
		msgDocModel, err := m.GetMsgDocModelByIndex(ctx, conversationID, skip, -1)
		if err != nil {
			return nil, err
		}
		for i := len(msgDocModel.Msg) - 1; i >= 0; i-- {
			if msgDocModel.Msg[i].Msg != nil {
				return msgDocModel.Msg[i], nil
			}
		}
	}
}

func (m *MsgMgo) GetOldestMsg(ctx context.Context, conversationID string) (*model.MsgInfoModel, error) {
	for skip := int64(0); ; skip++ {
		msgDocModel, err := m.GetMsgDocModelByIndex(ctx, conversationID, skip, 1)
		if err != nil {
			return nil, err
		}
		for i, v := range msgDocModel.Msg {
			if v.Msg != nil {
```

```
        return msgDocModel.Msg[i], nil
    }
  }
 }
}

func (m *MsgMgo) GetMsgDocModelByIndex(ctx context.Context, conversationID string, index, sort int64) (*model.MsgDoc
 if sort != 1 && sort != -1 {
  return nil, errs.ErrArgs.WrapMsg("mongo sort must be 1 or -1")
 }
 opt := options.Find().SetSkip(index).SetSort(bson.M{"_id": sort}).SetLimit(1)
 filter := bson.M{"doc_id": primitive.Regex{Pattern: fmt.Sprintf("^%s:", conversationID)}}
 msgs, err := mongoutil.Find[*model.MsgDocModel](ctx, m.coll, filter, opt)
 if err != nil {
  return nil, err
 }
 if len(msgs) > 0 {
  return msgs[0], nil
 }
 return nil, errs.Wrap(model.ErrMsgListNotExist)
}

func (m *MsgMgo) DeleteMsgsInOneDocByIndex(ctx context.Context, docID string, indexes []int) error {
 update := bson.M{
  "$set": bson.M{},
 }
 for _, index := range indexes {
  update["$set"].(bson.M)[fmt.Sprintf("msgs.%d", index)] = bson.M{
   "msg": nil,
  }
 }
 _, err := mongoutil.UpdateMany(ctx, m.coll, bson.M{"doc_id": docID}, update)
 return err
}

func (m *MsgMgo) MarkSingleChatMsgsAsRead(ctx context.Context, userID string, docID string, indexes []int64) error {
 var updates []mongo.WriteModel
 for _, index := range indexes {
  filter := bson.M{
   "doc_id": docID,
   fmt.Sprintf("msgs.%d.msg.send_id", index): bson.M{
    "$ne": userID,
   },
  }
  update := bson.M{
   "$set": bson.M{
    fmt.Sprintf("msgs.%d.is_read", index): true,
   },
  }
  updateModel := mongo.NewUpdateManyModel().
   SetFilter(filter).
   SetUpdate(update)
  updates = append(updates, updateModel)
 }
 if _, err := m.coll.BulkWrite(ctx, updates); err != nil {
  return errs.WrapMsg(err, fmt.Sprintf("docID is %s, indexes is %v", docID, indexes))
 }
 return nil
}

type searchMessageIndex struct {
 ID    primitive.ObjectID `bson:"_id"`
 Index []int64            `bson:"index"`
}

func (m *MsgMgo) searchMessageIndex(ctx context.Context, filter any, nextID primitive.ObjectID, limit int) ([]search
```

```
	var pipeline bson.A
	if !nextID.IsZero() {
		pipeline = append(pipeline, bson.M{"$match": bson.M{"_id": bson.M{"$gt": nextID}}})
	}
	coarseFilter := bson.M{
		"$or": bson.A{
			bson.M{
				"doc_id": primitive.Regex{Pattern: "^sg_"},
			},
			bson.M{
				"doc_id": primitive.Regex{Pattern: "^si_"},
			},
		},
	}
	pipeline = append(pipeline,
		bson.M{"$sort": bson.M{"_id": 1}},
		bson.M{"$match": coarseFilter},
		bson.M{"$match": filter},
		bson.M{"$limit": limit},
		bson.M{
			"$project": bson.M{
				"_id": 1,
				"msgs": bson.M{
					"$map": bson.M{
						"input": "$msgs",
						"as":    "msg",
						"in": bson.M{
							"$mergeObjects": bson.A{
								"$$msg",
								bson.M{
									"_search_temp_index": bson.M{
										"$indexOfArray": bson.A{
											"$msgs", "$$msg",
										},
									},
								},
							},
						},
					},
				},
			},
		},
		bson.M{"$unwind": "$msgs"},
		bson.M{"$match": filter},
		bson.M{
			"$project": bson.M{
				"_id":                     1,
				"msgs._search_temp_index": 1,
			},
		},
		bson.M{
			"$group": bson.M{
				"_id":   "$_id",
				"index": bson.M{"$push": "$msgs._search_temp_index"},
			},
		},
		bson.M{"$sort": bson.M{"_id": 1}},
	)
	return mongoutil.Aggregate[searchMessageIndex](ctx, m.coll, pipeline)
}

func (m *MsgMgo) searchMessage(ctx context.Context, req *msg.SearchMessageReq) (int64, []searchMessageIndex, error) {
	filter := bson.M{
		"msgs.msg": bson.M{
			"$exists": true,
			"$type":   "object",
```

```
    },
  }
  if req.RecvID != "" {
    filter["$or"] = bson.A{
      bson.M{"msgs.msg.recv_id": req.RecvID},
      bson.M{"msgs.msg.group_id": req.RecvID},
    }
  }
  if req.SendID != "" {
    filter["msgs.msg.send_id"] = req.SendID
  }
  if req.ContentType != 0 {
    filter["msgs.msg.content_type"] = req.ContentType
  }
  if req.SessionType != 0 {
    filter["msgs.msg.session_type"] = req.SessionType
  }
  if req.SendTime != "" {
    sendTime, err := time.Parse(time.DateOnly, req.SendTime)
    if err != nil {
      return 0, nil, errs.ErrArgs.WrapMsg("invalid sendTime", "req", req.SendTime, "format", time.DateOnly, "cause", e
    }
    filter["$and"] = bson.A{
      bson.M{"msgs.msg.send_time": bson.M{
        "$gte": sendTime.UnixMilli(),
      }},
      bson.M{
        "msgs.msg.send_time": bson.M{
          "$lt": sendTime.Add(time.Hour * 24).UnixMilli(),
        },
      },
    }
  }

  var (
    nextID    primitive.ObjectID
    count     int
    dataRange []searchMessageIndex
    skip      = int((req.Pagination.GetPageNumber() - 1) * req.Pagination.GetShowNumber())
  )
  _, _ = dataRange, skip
  const maxDoc = 50
  data := make([]searchMessageIndex, 0, req.Pagination.GetShowNumber())
  push := cap(data)
  for i := 0; ; i++ {
    res, err := m.searchMessageIndex(ctx, filter, nextID, maxDoc)
    if err != nil {
      return 0, nil, err
    }
    if len(res) > 0 {
      nextID = res[len(res)-1].ID
    }
    for _, r := range res {
      var dataIndex []int64
      for _, index := range r.Index {
        if push > 0 && count >= skip {
          dataIndex = append(dataIndex, index)
          push--
        }
        count++
      }
      if len(dataIndex) > 0 {
        data = append(data, searchMessageIndex{
          ID:    r.ID,
          Index: dataIndex,
        })
```

```
        }
    }
    if push <= 0 {
        push--
    }
    if len(res) < maxDoc || push < -10 {
        return int64(count), data, nil
    }
}
}

func (m *MsgMgo) SearchMessage(ctx context.Context, req *msg.SearchMessageReq) (int64, []*model.MsgInfoModel, error)
count, data, err := m.searchMessage(ctx, req)
if err != nil {
    return 0, nil, err
}
var msgs []*model.MsgInfoModel
if len(data) > 0 {
    var n int
    for _, d := range data {
        n += len(d.Index)
    }
    msgs = make([]*model.MsgInfoModel, 0, n)
}
for _, val := range data {
    res, err := mongoutil.FindOne[*model.MsgDocModel](ctx, m.coll, bson.M{"_id": val.ID})
    if err != nil {
        return 0, nil, err
    }
    for _, i := range val.Index {
        if i >= int64(len(res.Msg)) {
            continue
        }
        msgs = append(msgs, res.Msg[i])
    }
}
return count, msgs, nil
}

func (m *MsgMgo) RangeUserSendCount(ctx context.Context, start time.Time, end time.Time, group bool, ase bool, pageN
var sort int
if ase {
    sort = 1
} else {
    sort = -1
}
type Result struct {
    MsgCount  int64 `bson:"msg_count"`
    UserCount int64 `bson:"user_count"`
    Users     []struct {
        UserID string `bson:"_id"`
        Count  int64  `bson:"count"`
    } `bson:"users"`
    Dates []struct {
        Date  string `bson:"_id"`
        Count int64  `bson:"count"`
    } `bson:"dates"`
}
or := bson.A{
    bson.M{
        "doc_id": bson.M{
            "$regex":   "^si_",
            "$options": "i",
        },
    },
}
```

```
    if group {
        or = append(or,
            bson.M{
                "doc_id": bson.M{
                    "$regex":   "^g_",
                    "$options": "i",
                },
            },
            bson.M{
                "doc_id": bson.M{
                    "$regex":   "^sg_",
                    "$options": "i",
                },
            },
        )
    }
    pipeline := bson.A{
        bson.M{
            "$match": bson.M{
                "$and": bson.A{
                    bson.M{
                        "msgs.msg.send_time": bson.M{
                            "$gte": start.UnixMilli(),
                            "$lt":  end.UnixMilli(),
                        },
                    },
                    bson.M{
                        "$or": or,
                    },
                },
            },
        },
        bson.M{
            "$addFields": bson.M{
                "msgs": bson.M{
                    "$filter": bson.M{
                        "input": "$msgs",
                        "as":    "item",
                        "cond": bson.M{
                            "$and": bson.A{
                                bson.M{
                                    "$gte": bson.A{
                                        "$$item.msg.send_time", start.UnixMilli(),
                                    },
                                },
                                bson.M{
                                    "$lt": bson.A{
                                        "$$item.msg.send_time", end.UnixMilli(),
                                    },
                                },
                            },
                        },
                    },
                },
            },
        },
        bson.M{
            "$project": bson.M{
                "_id": 0,
            },
        },
        bson.M{
            "$project": bson.M{
                "result": bson.M{
                    "$map": bson.M{
                        "input": "$msgs",
```

```
        "as":    "item",
        "in": bson.M{
            "user_id": "$$item.msg.send_id",
            "send_date": bson.M{
                "$dateToString": bson.M{
                    "format": "%Y-%m-%d",
                    "date": bson.M{
                        "$toDate": "$$item.msg.send_time", // Millisecond timestamp
                    },
                },
            },
        },
    },
    bson.M{
        "$unwind": "$result",
    },
    bson.M{
        "$group": bson.M{
            "_id": "$result.send_date",
            "count": bson.M{
                "$sum": 1,
            },
            "original": bson.M{
                "$push": "$$ROOT",
            },
        },
    },
    bson.M{
        "$addFields": bson.M{
            "dates": "$$ROOT",
        },
    },
    bson.M{
        "$project": bson.M{
            "_id":            0,
            "count":          0,
            "dates.original": 0,
        },
    },
    bson.M{
        "$group": bson.M{
            "_id": nil,
            "count": bson.M{
                "$sum": 1,
            },
            "dates": bson.M{
                "$push": "$dates",
            },
            "original": bson.M{
                "$push": "$original",
            },
        },
    },
    bson.M{
        "$unwind": "$original",
    },
    bson.M{
        "$unwind": "$original",
    },
    bson.M{
        "$group": bson.M{
            "_id": "$original.result.user_id",
            "count": bson.M{
```

```
            "$sum": 1,
        },
        "original": bson.M{
            "$push": "$dates",
        },
    },
},
bson.M{
    "$addFields": bson.M{
        "dates": bson.M{
            "$arrayElemAt": bson.A{"$original", 0},
        },
    },
},
bson.M{
    "$project": bson.M{
        "original": 0,
    },
},
bson.M{
    "$sort": bson.M{
        "count": sort,
    },
},
bson.M{
    "$group": bson.M{
        "_id": nil,
        "user_count": bson.M{
            "$sum": 1,
        },
        "users": bson.M{
            "$push": "$$ROOT",
        },
    },
},
bson.M{
    "$addFields": bson.M{
        "dates": bson.M{
            "$arrayElemAt": bson.A{"$users", 0},
        },
    },
},
bson.M{
    "$addFields": bson.M{
        "dates": "$dates.dates",
    },
},
bson.M{
    "$project": bson.M{
        "_id":         0,
        "users.dates": 0,
    },
},
bson.M{
    "$addFields": bson.M{
        "msg_count": bson.M{
            "$sum": "$users.count",
        },
    },
},
bson.M{
    "$addFields": bson.M{
        "users": bson.M{
            "$slice": bson.A{"$users", pageNumber - 1, showNumber},
        },
    },
},
```

```
    },
  }
  result, err := mongoutil.Aggregate[*Result](ctx, m.coll, pipeline, options.Aggregate().SetAllowDiskUse(true))
  if err != nil {
    return 0, 0, nil, nil, err
  }
  if len(result) == 0 {
    return 0, 0, nil, nil, errs.Wrap(err)
  }
  users = make([]*model.UserCount, len(result[0].Users))
  for i, r := range result[0].Users {
    users[i] = &model.UserCount{
      UserID: r.UserID,
      Count:  r.Count,
    }
  }
  dateCount = make(map[string]int64)
  for _, r := range result[0].Dates {
    dateCount[r.Date] = r.Count
  }
  return result[0].MsgCount, result[0].UserCount, users, dateCount, nil
}

func (m *MsgMgo) RangeGroupSendCount(ctx context.Context, start time.Time, end time.Time, ase bool, pageNumber int32
  var sort int
  if ase {
    sort = 1
  } else {
    sort = -1
  }
  type Result struct {
    MsgCount  int64 `bson:"msg_count"`
    UserCount int64 `bson:"user_count"`
    Groups    []struct {
      GroupID string `bson:"_id"`
      Count   int64  `bson:"count"`
    } `bson:"groups"`
    Dates []struct {
      Date  string `bson:"_id"`
      Count int64  `bson:"count"`
    } `bson:"dates"`
  }
  pipeline := bson.A{
    bson.M{
      "$match": bson.M{
        "$and": bson.A{
          bson.M{
            "msgs.msg.send_time": bson.M{
              "$gte": start.UnixMilli(),
              "$lt":  end.UnixMilli(),
            },
          },
          bson.M{
            "$or": bson.A{
              bson.M{
                "doc_id": bson.M{
                  "$regex":   "^g_",
                  "$options": "i",
                },
              },
              bson.M{
                "doc_id": bson.M{
                  "$regex":   "^sg_",
                  "$options": "i",
                },
              },
```

```
        },
      },
    },
  },
  bson.M{
    "$addFields": bson.M{
      "msgs": bson.M{
        "$filter": bson.M{
          "input": "$msgs",
          "as":    "item",
          "cond": bson.M{
            "$and": bson.A{
              bson.M{
                "$gte": bson.A{
                  "$$item.msg.send_time", start.UnixMilli(),
                },
              },
              bson.M{
                "$lt": bson.A{
                  "$$item.msg.send_time", end.UnixMilli(),
                },
              },
            },
          },
        },
      },
    },
  },
  bson.M{
    "$project": bson.M{
      "_id": 0,
    },
  },
  bson.M{
    "$project": bson.M{
      "result": bson.M{
        "$map": bson.M{
          "input": "$msgs",
          "as":    "item",
          "in": bson.M{
            "group_id": "$$item.msg.group_id",
            "send_date": bson.M{
              "$dateToString": bson.M{
                "format": "%Y-%m-%d",
                "date": bson.M{
                  "$toDate": "$$item.msg.send_time", // Millisecond timestamp
                },
              },
            },
          },
        },
      },
    },
  },
  bson.M{
    "$unwind": "$result",
  },
  bson.M{
    "$group": bson.M{
      "_id": "$result.send_date",
      "count": bson.M{
        "$sum": 1,
      },
      "original": bson.M{
        "$push": "$$ROOT",
```

```
        },
    },
},
bson.M{
    "$addFields": bson.M{
        "dates": "$$ROOT",
    },
},
bson.M{
    "$project": bson.M{
        "_id":            0,
        "count":          0,
        "dates.original": 0,
    },
},
bson.M{
    "$group": bson.M{
        "_id": nil,
        "count": bson.M{
            "$sum": 1,
        },
        "dates": bson.M{
            "$push": "$dates",
        },
        "original": bson.M{
            "$push": "$original",
        },
    },
},
bson.M{
    "$unwind": "$original",
},
bson.M{
    "$unwind": "$original",
},
bson.M{
    "$group": bson.M{
        "_id": "$original.result.group_id",
        "count": bson.M{
            "$sum": 1,
        },
        "original": bson.M{
            "$push": "$dates",
        },
    },
},
bson.M{
    "$addFields": bson.M{
        "dates": bson.M{
            "$arrayElemAt": bson.A{"$original", 0},
        },
    },
},
bson.M{
    "$project": bson.M{
        "original": 0,
    },
},
bson.M{
    "$sort": bson.M{
        "count": sort,
    },
},
bson.M{
    "$group": bson.M{
        "_id": nil,
```

```
        "user_count": bson.M{
            "$sum": 1,
        },
        "groups": bson.M{
            "$push": "$$ROOT",
        },
    },
    bson.M{
        "$addFields": bson.M{
            "dates": bson.M{
                "$arrayElemAt": bson.A{"$groups", 0},
            },
        },
    },
    bson.M{
        "$addFields": bson.M{
            "dates": "$dates.dates",
        },
    },
    bson.M{
        "$project": bson.M{
            "_id":          0,
            "groups.dates": 0,
        },
    },
    bson.M{
        "$addFields": bson.M{
            "msg_count": bson.M{
                "$sum": "$groups.count",
            },
        },
    },
    bson.M{
        "$addFields": bson.M{
            "groups": bson.M{
                "$slice": bson.A{"$groups", pageNumber - 1, showNumber},
            },
        },
    },
}
result, err := mongoutil.Aggregate[*Result](ctx, m.coll, pipeline, options.Aggregate().SetAllowDiskUse(true))
if err != nil {
    return 0, 0, nil, nil, err
}
if len(result) == 0 {
    return 0, 0, nil, nil, errs.Wrap(err)
}
groups = make([]*model.GroupCount, len(result[0].Groups))
for i, r := range result[0].Groups {
    groups[i] = &model.GroupCount{
        GroupID: r.GroupID,
        Count:   r.Count,
    }
}
dateCount = make(map[string]int64)
for _, r := range result[0].Dates {
    dateCount[r.Date] = r.Count
}
return result[0].MsgCount, result[0].UserCount, groups, dateCount, nil
}

func (m *MsgMgo) GetRandBeforeMsg(ctx context.Context, ts int64, limit int) ([]*model.MsgDocModel, error) {
    return mongoutil.Aggregate[*model.MsgDocModel](ctx, m.coll, []bson.M{
        {
            "$match": bson.M{
```

```
        "msgs": bson.M{
            "$not": bson.M{
                "$elemMatch": bson.M{
                    "msg.send_time": bson.M{
                        "$gt": ts,
                    },
                },
            },
        },
    },
    {
        "$project": bson.M{
            "_id":                0,
            "doc_id":             1,
            "msgs.msg.send_time": 1,
            "msgs.msg.seq":       1,
        },
    },
    {
        "$sample": bson.M{
            "size": limit,
        },
    },
})
}

func (m *MsgMgo) DeleteDoc(ctx context.Context, docID string) error {
    return mongoutil.DeleteOne(ctx, m.coll, bson.M{"doc_id": docID})
}

func (m *MsgMgo) GetLastMessageSeqByTime(ctx context.Context, conversationID string, time int64) (int64, error) {
    pipeline := []bson.M{
        {
            "$match": bson.M{
                "doc_id": bson.M{
                    "$regex": fmt.Sprintf("^%s", conversationID),
                },
            },
        },
        {
            "$match": bson.M{
                "msgs.msg.send_time": bson.M{
                    "$lte": time,
                },
            },
        },
        {
            "$sort": bson.M{
                "_id": -1,
            },
        },
        {
            "$limit": 1,
        },
        {
            "$project": bson.M{
                "_id":                0,
                "doc_id":             1,
                "msgs.msg.send_time": 1,
                "msgs.msg.seq":       1,
            },
        },
    }
    res, err := mongoutil.Aggregate[*model.MsgDocModel](ctx, m.coll, pipeline)
    if err != nil {
```

```
    return 0, err
  }
  if len(res) == 0 {
    return 0, nil
  }
  var seq int64
  for _, v := range res[0].Msg {
    if v.Msg == nil {
      continue
    }
    if v.Msg.SendTime <= time {
      seq = v.Msg.Seq
    }
  }
  return seq, nil
}

func (m *MsgMgo) GetLastMessage(ctx context.Context, conversationID string) (*model.MsgInfoModel, error) {
  pipeline := []bson.M{
    {
      "$match": bson.M{
        "doc_id": bson.M{
          "$regex": fmt.Sprintf("^%s", conversationID),
        },
      },
    },
    {
      "$match": bson.M{
        "msgs.msg.status": bson.M{
          "$lt": constant.MsgStatusHasDeleted,
        },
      },
    },
    {
      "$sort": bson.M{
        "_id": -1,
      },
    },
    {
      "$limit": 1,
    },
    {
      "$project": bson.M{
        "_id":    0,
        "doc_id": 0,
      },
    },
    {
      "$unwind": "$msgs",
    },
    {
      "$match": bson.M{
        "msgs.msg.status": bson.M{
          "$lt": constant.MsgStatusHasDeleted,
        },
      },
    },
    {
      "$sort": bson.M{
        "msgs.msg.seq": -1,
      },
    },
    {
      "$limit": 1,
    },
  }
```

```
	type Result struct {
		Msgs *model.MsgInfoModel `bson:"msgs"`
	}
	res, err := mongoutil.Aggregate[*Result](ctx, m.coll, pipeline)
	if err != nil {
		return nil, err
	}
	if len(res) == 0 {
		return nil, errs.Wrap(mongo.ErrNoDocuments)
	}
	return res[0].Msgs, nil
}

func (m *MsgMgo) onlyFindDocIndex(ctx context.Context, docID string, indexes []int64) ([]*model.MsgInfoModel, error)
	if len(indexes) == 0 {
		return nil, nil
	}
	pipeline := mongo.Pipeline{
		bson.D{{Key: "$match", Value: bson.D{
			{Key: "doc_id", Value: docID},
		}}},
		bson.D{{Key: "$project", Value: bson.D{
			{Key: "_id", Value: 0},
			{Key: "doc_id", Value: 1},
			{Key: "msgs", Value: bson.D{
				{Key: "$map", Value: bson.D{
					{Key: "input", Value: indexes},
					{Key: "as", Value: "index"},
					{Key: "in", Value: bson.D{
						{Key: "$arrayElemAt", Value: bson.A{"$msgs", "$$index"}},
					}},
				}},
			}},
		}}},
	}
	msgDocModel, err := mongoutil.Aggregate[*model.MsgDocModel](ctx, m.coll, pipeline)
	if err != nil {
		return nil, err
	}
	if len(msgDocModel) == 0 {
		return nil, nil
	}
	return msgDocModel[0].Msg, nil
}

//func (m *MsgMgo) FindSeqs(ctx context.Context, conversationID string, seqs []int64) ([]*model.MsgInfoModel, error)
//	if len(seqs) == 0 {
//		return nil, nil
//	}
//	result := make([]*model.MsgInfoModel, 0, len(seqs))
//	for docID, seqs := range m.model.GetDocIDSeqsMap(conversationID, seqs) {
//		res, err := m.onlyFindDocIndex(ctx, docID, datautil.Slice(seqs, m.model.GetMsgIndex))
//		if err != nil {
//			return nil, err
//		}
//		for i, re := range res {
//			if re == nil || re.Msg == nil {
//				continue
//			}
//			result = append(result, res[i])
//		}
//	}
//	return result, nil
//}

func (m *MsgMgo) findBeforeDocSendTime(ctx context.Context, docID string, limit int64) (int64, int64, error) {
```

```
    if limit == 0 {
        return 0, 0, nil
    }
    pipeline := []bson.M{
        {
            "$match": bson.M{
                "doc_id": docID,
            },
        },
        {
            "$project": bson.M{
                "_id":    0,
                "doc_id": 0,
            },
        },
        {
            "$unwind": "$msgs",
        },
        {
            "$project": bson.M{
                //"_id":                0,
                //"doc_id":             0,
                "msgs.msg.send_time": 1,
                "msgs.msg.seq":       1,
            },
        },
    }
    if limit > 0 {
        pipeline = append(pipeline, bson.M{"$limit": limit})
    }
    type Result struct {
        Msgs *model.MsgInfoModel `bson:"msgs"`
    }
    res, err := mongoutil.Aggregate[Result](ctx, m.coll, pipeline)
    if err != nil {
        return 0, 0, err
    }
    for i := len(res) - 1; i >= 0; i-- {
        v := res[i]
        if v.Msgs != nil && v.Msgs.Msg != nil && v.Msgs.Msg.SendTime > 0 {
            return v.Msgs.Msg.Seq, v.Msgs.Msg.SendTime, nil
        }
    }
    return 0, 0, nil
}

func (m *MsgMgo) findBeforeSendTime(ctx context.Context, conversationID string, seq int64) (int64, int64, error) {
    first := true
    for i := m.model.GetDocIndex(seq); i >= 0; i-- {
        limit := int64(-1)
        if first {
            first = false
            limit = m.model.GetLimitForSingleDoc(seq)
        }
        docID := m.model.BuildDocIDByIndex(conversationID, i)
        msgSeq, msgSendTime, err := m.findBeforeDocSendTime(ctx, docID, limit)
        if err != nil {
            return 0, 0, err
        }
        if msgSendTime > 0 {
            return msgSeq, msgSendTime, nil
        }
    }
    return 0, 0, nil
}
```

```go
func (m *MsgMgo) FindSeqs(ctx context.Context, conversationID string, seqs []int64) ([]*model.MsgInfoModel, error) {
	if len(seqs) == 0 {
		return nil, nil
	}
	var abnormalSeq []int64
	result := make([]*model.MsgInfoModel, 0, len(seqs))
	for docID, docSeqs := range m.model.GetDocIDSeqsMap(conversationID, seqs) {
		res, err := m.onlyFindDocIndex(ctx, docID, datautil.Slice(docSeqs, m.model.GetMsgIndex))
		if err != nil {
			return nil, err
		}
		if len(res) == 0 {
			abnormalSeq = append(abnormalSeq, docSeqs...)
			continue
		}
		for i, re := range res {
			if re == nil || re.Msg == nil || re.Msg.SendTime == 0 {
				abnormalSeq = append(abnormalSeq, docSeqs[i])
				continue
			}
			result = append(result, res[i])
		}
	}
	if len(abnormalSeq) > 0 {
		datautil.Sort(abnormalSeq, false)
		sendTime := make(map[int64]int64)
		var (
			lastSeq      int64
			lastSendTime int64
		)
		for _, seq := range abnormalSeq {
			if lastSendTime > 0 && lastSeq <= seq {
				sendTime[seq] = lastSendTime
				continue
			}
			msgSeq, msgSendTime, err := m.findBeforeSendTime(ctx, conversationID, seq)
			if err != nil {
				return nil, err
			}
			if msgSendTime <= 0 {
				break
			}
			sendTime[seq] = msgSendTime
			lastSeq = msgSeq
			lastSendTime = msgSendTime
		}
		for _, seq := range abnormalSeq {
			result = append(result, &model.MsgInfoModel{
				Msg: &model.MsgDataModel{
					Seq:      seq,
					Status:   constant.MsgStatusHasDeleted,
					SendTime: sendTime[seq],
				},
			})
		}
	}
	return result, nil
}
```

pkg/common/storage/database/mgo/msg_test.go

```
package mgo

import (
	"context"
	"math"
	"math/rand"
	"strconv"
	"testing"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/db/mongoutil"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func TestName1(t *testing.T) {
	//ctx, cancel := context.WithTimeout(context.Background(), time.Second*300)
	//defer cancel()
	//cli := Result(mongo.Connect(ctx, options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.66:37017/openim_v
	//
	//v := &MsgMgo{
	//	coll: cli.Database("openim_v3").Collection("msg3"),
	//}
	//
	//req := &msg.SearchMessageReq{
	//	//RecvID: "3187706596",
	//	//SendID:      "7009965934",
	//	ContentType: 101,
	//	//SendTime:    "2024-05-06",
	//	//SessionType: 3,
	//	Pagination: &sdkws.RequestPagination{
	//		PageNumber: 1,
	//		ShowNumber: 10,
	//	},
	//}
	//total, res, err := v.SearchMessage(ctx, req)
	//if err != nil {
	//	panic(err)
	//}
	//
	//for i, re := range res {
	//	t.Logf("%d => %d | %+v", i+1, re.Msg.Seq, re.Msg.Content)
	//}
	//
	//t.Log(total)
	//
	//msg, err := NewMsgMongo(cli.Database("openim_v3"))
	//if err != nil {
	//	panic(err)
	//}
	//res, err := msg.GetBeforeMsg(ctx, time.Now().UnixMilli(), []string{"1:0"}, 1000)
	//if err != nil {
	//	panic(err)
	//}
	//t.Log(len(res))
}

func TestName10(t *testing.T) {
	ctx, cancel := context.WithTimeout(context.Background(), time.Second*10)
	defer cancel()
```

```
	cli := Result(mongo.Connect(ctx, options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.48:37017/openim_v3
	v := &MsgMgo{
		coll: cli.Database("openim_v3").Collection("msg3"),
	}
	opt := options.Find().SetLimit(1000)

	res, err := mongoutil.Find[model.MsgDocModel](ctx, v.coll, bson.M{}, opt)
	if err != nil {
		panic(err)
	}
	ctx = context.Background()
	for i := 0; i < 100000; i++ {
		for j := range res {
			res[j].DocID = strconv.FormatUint(rand.Uint64(), 10) + ":0"
		}
		if err := mongoutil.InsertMany(ctx, v.coll, res); err != nil {
			panic(err)
		}
		t.Log("====>", time.Now(), i)
	}

}

func TestName3(t *testing.T) {
	t.Log(uint64(math.MaxUint64))
	t.Log(int64(math.MaxInt64))

	t.Log(int64(math.MinInt64))
}

func TestName4(t *testing.T) {
	ctx, cancel := context.WithTimeout(context.Background(), time.Second*300)
	defer cancel()
	cli := Result(mongo.Connect(ctx, options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.135:37017/openim_v3

	msg, err := NewMsgMongo(cli.Database("openim_v3"))
	if err != nil {
		panic(err)
	}
	ts := time.Now().Add(-time.Hour * 24 * 5).UnixMilli()
	t.Log(ts)
	res, err := msg.GetLastMessageSeqByTime(ctx, "sg_1523453548", ts)
	if err != nil {
		panic(err)
	}
	t.Log(res)
}

func TestName5(t *testing.T) {
	ctx, cancel := context.WithTimeout(context.Background(), time.Second*300)
	defer cancel()
	cli := Result(mongo.Connect(ctx, options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.135:37017/openim_v3

	tmp, err := NewMsgMongo(cli.Database("openim_v3"))
	if err != nil {
		panic(err)
	}
	msg := tmp.(*MsgMgo)
	ts := time.Now().Add(-time.Hour * 24 * 5).UnixMilli()
	t.Log(ts)
	var seqs []int64
	for i := 1; i < 256; i++ {
		seqs = append(seqs, int64(i))
	}
	res, err := msg.FindSeqs(ctx, "si_4924054191_9511766539", seqs)
```

```
	if err != nil {
		panic(err)
	}
	t.Log(res)
}

//func TestName6(t *testing.T) {
//	ctx, cancel := context.WithTimeout(context.Background(), time.Second*300)
//	defer cancel()
//	cli := Result(mongo.Connect(ctx, options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.135:37017/openim_
//
//	tmp, err := NewMsgMongo(cli.Database("openim_v3"))
//	if err != nil {
//		panic(err)
//	}
//	msg := tmp.(*MsgMgo)
//	seq, sendTime, err := msg.findBeforeSendTime(ctx, "si_4924054191_9511766539", 1144)
//	if err != nil {
//		panic(err)
//	}
//	t.Log(seq, sendTime)
//}

func TestSearchMessage(t *testing.T) {
	ctx, cancel := context.WithTimeout(context.Background(), time.Second*300)
	defer cancel()
	cli := Result(mongo.Connect(ctx, options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.135:37017/openim_v3

	msgMongo, err := NewMsgMongo(cli.Database("openim_v3"))
	if err != nil {
		panic(err)
	}
	ts := time.Now().Add(-time.Hour * 24 * 5).UnixMilli()
	t.Log(ts)
	req := &msg.SearchMessageReq{
		//SendID: "yjz",
		//RecvID: "aibot",
		Pagination: &sdkws.RequestPagination{
			PageNumber: 1,
			ShowNumber: 20,
		},
	}
	count, resp, err := msgMongo.SearchMessage(ctx, req)
	if err != nil {
		panic(err)
	}
	t.Log(resp, count)
}
```

pkg/common/storage/database/mgo/object.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

    "github.com/openimsdk/tools/db/mongoutil"
    "github.com/openimsdk/tools/errs"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

func NewS3Mongo(db *mongo.Database) (database.ObjectInfo, error) {
    coll := db.Collection(database.ObjectName)

    // Create index for name
    _, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
        Keys: bson.D{
            {Key: "name", Value: 1},
        },
        Options: options.Index().SetUnique(true),
    })
    if err != nil {
        return nil, errs.Wrap(err)
    }

    // Create index for create_time
    _, err = coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
        Keys: bson.D{
            {Key: "create_time", Value: 1},
        },
    })
    if err != nil {
        return nil, errs.Wrap(err)
    }

    // Create index for key
    _, err = coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
        Keys: bson.D{
            {Key: "key", Value: 1},
        },
    })
    if err != nil {
        return nil, errs.Wrap(err)
    }
```

```
    return &S3Mongo{coll: coll}, nil
}

type S3Mongo struct {
    coll *mongo.Collection
}

func (o *S3Mongo) SetObject(ctx context.Context, obj *model.Object) error {
    filter := bson.M{"name": obj.Name, "engine": obj.Engine}
    update := bson.M{
        "name":         obj.Name,
        "engine":       obj.Engine,
        "key":          obj.Key,
        "size":         obj.Size,
        "content_type": obj.ContentType,
        "group":        obj.Group,
        "create_time":  obj.CreateTime,
    }
    return mongoutil.UpdateOne(ctx, o.coll, filter, bson.M{"$set": update}, false, options.Update().SetUpsert(true))
}

func (o *S3Mongo) Take(ctx context.Context, engine string, name string) (*model.Object, error) {
    if engine == "" {
        return mongoutil.FindOne[*model.Object](ctx, o.coll, bson.M{"name": name})
    }
    return mongoutil.FindOne[*model.Object](ctx, o.coll, bson.M{"name": name, "engine": engine})
}

func (o *S3Mongo) Delete(ctx context.Context, engine string, name []string) error {
    if len(name) == 0 {
        return nil
    }
    return mongoutil.DeleteOne(ctx, o.coll, bson.M{"engine": engine, "name": bson.M{"$in": name}})
}

func (o *S3Mongo) FindExpirationObject(ctx context.Context, engine string, expiration time.Time, needDelType []strin
    opt := options.Find()
    if count > 0 {
        opt.SetLimit(count)
    }
    return mongoutil.Find[*model.Object](ctx, o.coll, bson.M{
        "engine":      engine,
        "create_time": bson.M{"$lt": expiration},
        "group":       bson.M{"$in": needDelType},
    }, opt)
}

func (o *S3Mongo) GetKeyCount(ctx context.Context, engine string, key string) (int64, error) {
    return mongoutil.Count(ctx, o.coll, bson.M{"engine": engine, "key": key})
}

func (o *S3Mongo) GetEngineCount(ctx context.Context, engine string) (int64, error) {
    return mongoutil.Count(ctx, o.coll, bson.M{"engine": engine})
}

func (o *S3Mongo) GetEngineInfo(ctx context.Context, engine string, limit int, skip int) ([]*model.Object, error) {
    return mongoutil.Find[*model.Object](ctx, o.coll, bson.M{"engine": engine}, options.Find().SetLimit(int64(limit)).S
}

func (o *S3Mongo) UpdateEngine(ctx context.Context, oldEngine, oldName string, newEngine string) error {
    return mongoutil.UpdateOne(ctx, o.coll, bson.M{"engine": oldEngine, "name": oldName}, bson.M{"$set": bson.M{"engine
}
```

pkg/common/storage/database/mgo/seq_conversation.go

```
package mgo

import (
	"context"
	"errors"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/mongoutil"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewSeqConversationMongo(db *mongo.Database) (database.SeqConversation, error) {
	coll := db.Collection(database.SeqConversationName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "conversation_id", Value: 1},
		},
	})
	if err != nil {
		return nil, err
	}
	return &seqConversationMongo{coll: coll}, nil
}

type seqConversationMongo struct {
	coll *mongo.Collection
}

func (s *seqConversationMongo) setSeq(ctx context.Context, conversationID string, seq int64, field string) error {
	filter := map[string]any{
		"conversation_id": conversationID,
	}
	insert := bson.M{
		"conversation_id": conversationID,
		"min_seq":         0,
		"max_seq":         0,
	}
	delete(insert, field)
	update := map[string]any{
		"$set": bson.M{
			field: seq,
		},
		"$setOnInsert": insert,
	}
	opt := options.Update().SetUpsert(true)
	return mongoutil.UpdateOne(ctx, s.coll, filter, update, false, opt)
}

func (s *seqConversationMongo) Malloc(ctx context.Context, conversationID string, size int64) (int64, error) {
	if size < 0 {
		return 0, errors.New("size must be greater than 0")
	}
	if size == 0 {
		return s.GetMaxSeq(ctx, conversationID)
	}
	filter := map[string]any{"conversation_id": conversationID}
	update := map[string]any{
		"$inc":         map[string]any{"max_seq": size},
		"$setOnInsert": map[string]any{"min_seq": int64(0)},
	}
	opt := options.FindOneAndUpdate().SetUpsert(true).SetReturnDocument(options.After).SetProjection(map[string]any{"_i
	lastSeq, err := mongoutil.FindOneAndUpdate[int64](ctx, s.coll, filter, update, opt)
```

```
    if err != nil {
        return 0, err
    }
    return lastSeq - size, nil
}

func (s *seqConversationMongo) SetMaxSeq(ctx context.Context, conversationID string, seq int64) error {
    return s.setSeq(ctx, conversationID, seq, "max_seq")
}

func (s *seqConversationMongo) GetMaxSeq(ctx context.Context, conversationID string) (int64, error) {
    seq, err := mongoutil.FindOne[int64](ctx, s.coll, bson.M{"conversation_id": conversationID}, options.FindOne().SetP
    if err == nil {
        return seq, nil
    } else if IsNotFound(err) {
        return 0, nil
    } else {
        return 0, err
    }
}

func (s *seqConversationMongo) GetMinSeq(ctx context.Context, conversationID string) (int64, error) {
    seq, err := mongoutil.FindOne[int64](ctx, s.coll, bson.M{"conversation_id": conversationID}, options.FindOne().SetP
    if err == nil {
        return seq, nil
    } else if IsNotFound(err) {
        return 0, nil
    } else {
        return 0, err
    }
}

func (s *seqConversationMongo) SetMinSeq(ctx context.Context, conversationID string, seq int64) error {
    return s.setSeq(ctx, conversationID, seq, "min_seq")
}

func (s *seqConversationMongo) GetConversation(ctx context.Context, conversationID string) (*model.SeqConversation,
    return mongoutil.FindOne[*model.SeqConversation](ctx, s.coll, bson.M{"conversation_id": conversationID})
}
```

pkg/common/storage/database/mgo/seq_conversation_test.go

```go
package mgo

import (
	"context"
	"testing"
	"time"

	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func Result[V any](val V, err error) V {
	if err != nil {
		panic(err)
	}
	return val
}

func Mongodb() *mongo.Database {
	return Result(
		mongo.Connect(context.Background(),
			options.Client().
				ApplyURI("mongodb://openIM:openIM123@172.16.8.135:37017/openim_v3?maxPoolSize=100").
				SetConnectTimeout(5*time.Second)),
	).Database("openim_v3")
}

func TestUserSeq(t *testing.T) {
	uSeq := Result(NewSeqUserMongo(Mongodb())).(*seqUserMongo)
	t.Log(uSeq.SetUserMinSeq(context.Background(), "1000", "2000", 4))
}

func TestConversationSeq(t *testing.T) {
	cSeq := Result(NewSeqConversationMongo(Mongodb())).(*seqConversationMongo)
	t.Log(cSeq.SetMaxSeq(context.Background(), "2000", 10))
	t.Log(cSeq.Malloc(context.Background(), "2000", 10))
	t.Log(cSeq.GetMaxSeq(context.Background(), "2000"))
}

func TestUserGetUserReadSeqs(t *testing.T) {
	uSeq := Result(NewSeqUserMongo(Mongodb())).(*seqUserMongo)
	t.Log(uSeq.GetUserReadSeqs(context.Background(), "2110910952", []string{"sg_345762580", "2000", "3000"}))
}
```

pkg/common/storage/database/mgo/seq_user.go

```go
package mgo

import (
	"context"
	"errors"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/mongoutil"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewSeqUserMongo(db *mongo.Database) (database.SeqUser, error) {
	coll := db.Collection(database.SeqUserName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "user_id", Value: 1},
			{Key: "conversation_id", Value: 1},
		},
	})
	if err != nil {
		return nil, err
	}
	return &seqUserMongo{coll: coll}, nil
}

type seqUserMongo struct {
	coll *mongo.Collection
}

func (s *seqUserMongo) setSeq(ctx context.Context, conversationID string, userID string, seq int64, field string) er
	filter := map[string]any{
		"user_id":         userID,
		"conversation_id": conversationID,
	}
	insert := bson.M{
		"user_id":         userID,
		"conversation_id": conversationID,
		"min_seq":         0,
		"max_seq":         0,
		"read_seq":        0,
	}
	delete(insert, field)
	update := map[string]any{
		"$set": bson.M{
			field: seq,
		},
		"$setOnInsert": insert,
	}
	opt := options.Update().SetUpsert(true)
	return mongoutil.UpdateOne(ctx, s.coll, filter, update, false, opt)
}

func (s *seqUserMongo) getSeq(ctx context.Context, conversationID string, userID string, failed string) (int64, erro
	filter := map[string]any{
		"user_id":         userID,
		"conversation_id": conversationID,
	}
	opt := options.FindOne().SetProjection(bson.M{"_id": 0, failed: 1})
	seq, err := mongoutil.FindOne[int64](ctx, s.coll, filter, opt)
	if err == nil {
		return seq, nil
	} else if errors.Is(err, mongo.ErrNoDocuments) {
```

```
		return 0, nil
	} else {
		return 0, err
	}
}

func (s *seqUserMongo) GetUserMaxSeq(ctx context.Context, conversationID string, userID string) (int64, error) {
	return s.getSeq(ctx, conversationID, userID, "max_seq")
}

func (s *seqUserMongo) SetUserMaxSeq(ctx context.Context, conversationID string, userID string, seq int64) error {
	return s.setSeq(ctx, conversationID, userID, seq, "max_seq")
}

func (s *seqUserMongo) GetUserMinSeq(ctx context.Context, conversationID string, userID string) (int64, error) {
	return s.getSeq(ctx, conversationID, userID, "min_seq")
}

func (s *seqUserMongo) SetUserMinSeq(ctx context.Context, conversationID string, userID string, seq int64) error {
	return s.setSeq(ctx, conversationID, userID, seq, "min_seq")
}

func (s *seqUserMongo) GetUserReadSeq(ctx context.Context, conversationID string, userID string) (int64, error) {
	return s.getSeq(ctx, conversationID, userID, "read_seq")
}

func (s *seqUserMongo) notFoundSet0(seq map[string]int64, conversationIDs []string) {
	for _, conversationID := range conversationIDs {
		if _, ok := seq[conversationID]; !ok {
			seq[conversationID] = 0
		}
	}
}

func (s *seqUserMongo) GetUserReadSeqs(ctx context.Context, userID string, conversationID []string) (map[string]int6
	if len(conversationID) == 0 {
		return map[string]int64{}, nil
	}
	filter := bson.M{"user_id": userID, "conversation_id": bson.M{"$in": conversationID}}
	opt := options.Find().SetProjection(bson.M{"_id": 0, "conversation_id": 1, "read_seq": 1})
	seqs, err := mongoutil.Find[*model.SeqUser](ctx, s.coll, filter, opt)
	if err != nil {
		return nil, err
	}
	res := make(map[string]int64)
	for _, seq := range seqs {
		res[seq.ConversationID] = seq.ReadSeq
	}
	s.notFoundSet0(res, conversationID)
	return res, nil
}

func (s *seqUserMongo) SetUserReadSeq(ctx context.Context, conversationID string, userID string, seq int64) error {
	dbSeq, err := s.GetUserReadSeq(ctx, conversationID, userID)
	if err != nil {
		return err
	}
	if dbSeq > seq {
		return nil
	}
	return s.setSeq(ctx, conversationID, userID, seq, "read_seq")
}
```

pkg/common/storage/database/mgo/user.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package mgo

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/errs"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/bson/primitive"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewUserMongo(db *mongo.Database) (database.User, error) {
	coll := db.Collection(database.UserName)
	_, err := coll.Indexes().CreateOne(context.Background(), mongo.IndexModel{
		Keys: bson.D{
			{Key: "user_id", Value: 1},
		},
		Options: options.Index().SetUnique(true),
	})
	if err != nil {
		return nil, errs.Wrap(err)
	}
	return &UserMgo{coll: coll}, nil
}

type UserMgo struct {
	coll *mongo.Collection
}

func (u *UserMgo) Create(ctx context.Context, users []*model.User) error {
	return mongoutil.InsertMany(ctx, u.coll, users)
}

func (u *UserMgo) UpdateByMap(ctx context.Context, userID string, args map[string]any) (err error) {
	if len(args) == 0 {
		return nil
	}
	return mongoutil.UpdateOne(ctx, u.coll, bson.M{"user_id": userID}, bson.M{"$set": args}, true)
}

func (u *UserMgo) Find(ctx context.Context, userIDs []string) (users []*model.User, err error) {
	return mongoutil.Find[*model.User](ctx, u.coll, bson.M{"user_id": bson.M{"$in": userIDs}})
```

```go
}

func (u *UserMgo) Take(ctx context.Context, userID string) (user *model.User, err error) {
    return mongoutil.FindOne[*model.User](ctx, u.coll, bson.M{"user_id": userID})
}

func (u *UserMgo) TakeNotification(ctx context.Context, level int64) (user []*model.User, err error) {
    return mongoutil.Find[*model.User](ctx, u.coll, bson.M{"app_manger_level": level})
}

func (u *UserMgo) TakeGTEAppManagerLevel(ctx context.Context, level int64) (user []*model.User, err error) {
    return mongoutil.Find[*model.User](ctx, u.coll, bson.M{"app_manger_level": bson.M{"$gte": level}})
}

func (u *UserMgo) TakeByNickname(ctx context.Context, nickname string) (user []*model.User, err error) {
    return mongoutil.Find[*model.User](ctx, u.coll, bson.M{"nickname": nickname})
}

func (u *UserMgo) Page(ctx context.Context, pagination pagination.Pagination) (count int64, users []*model.User, err
    return mongoutil.FindPage[*model.User](ctx, u.coll, bson.M{}, pagination)
}

func (u *UserMgo) PageFindUser(ctx context.Context, level1 int64, level2 int64, pagination pagination.Pagination) (c
    query := bson.M{
        "$or": []bson.M{
            {"app_manger_level": level1},
            {"app_manger_level": level2},
        },
    }

    return mongoutil.FindPage[*model.User](ctx, u.coll, query, pagination)
}

func (u *UserMgo) PageFindUserWithKeyword(
    ctx context.Context,
    level1 int64,
    level2 int64,
    userID string,
    nickName string,
    pagination pagination.Pagination,
) (count int64, users []*model.User, err error) {
    // Initialize the base query with level conditions
    query := bson.M{
        "$and": []bson.M{
            {"app_manger_level": bson.M{"$in": []int64{level1, level2}}},
        },
    }

    // Add userID and userName conditions to the query if they are provided
    if userID != "" || nickName != "" {
        userConditions := []bson.M{}
        if userID != "" {
            // Use regex for userID
            regexPattern := primitive.Regex{Pattern: userID, Options: "i"} // 'i' for case-insensitive matching
            userConditions = append(userConditions, bson.M{"user_id": regexPattern})
        }
        if nickName != "" {
            // Use regex for userName
            regexPattern := primitive.Regex{Pattern: nickName, Options: "i"} // 'i' for case-insensitive matching
            userConditions = append(userConditions, bson.M{"nickname": regexPattern})
        }
        query["$and"] = append(query["$and"].([]bson.M), bson.M{"$or": userConditions})
    }

    // Perform the paginated search
    return mongoutil.FindPage[*model.User](ctx, u.coll, query, pagination)
```

```
}

func (u *UserMgo) GetAllUserID(ctx context.Context, pagination pagination.Pagination) (int64, []string, error) {
	return mongoutil.FindPage[string](ctx, u.coll, bson.M{}, pagination, options.Find().SetProjection(bson.M{"_id": 0,
}

func (u *UserMgo) Exist(ctx context.Context, userID string) (exist bool, err error) {
	return mongoutil.Exist(ctx, u.coll, bson.M{"user_id": userID})
}

func (u *UserMgo) GetUserGlobalRecvMsgOpt(ctx context.Context, userID string) (opt int, err error) {
	return mongoutil.FindOne[int](ctx, u.coll, bson.M{"user_id": userID}, options.FindOne().SetProjection(bson.M{"_id":
}

func (u *UserMgo) CountTotal(ctx context.Context, before *time.Time) (count int64, err error) {
	if before == nil {
		return mongoutil.Count(ctx, u.coll, bson.M{})
	}
	return mongoutil.Count(ctx, u.coll, bson.M{"create_time": bson.M{"$lt": before}})
}

func (u *UserMgo) AddUserCommand(ctx context.Context, userID string, Type int32, UUID string, value string, ex strin
	collection := u.coll.Database().Collection("userCommands")

	// Create a new document instead of updating an existing one
	doc := bson.M{
		"userID":     userID,
		"type":       Type,
		"uuid":       UUID,
		"createTime": time.Now().Unix(), // assuming you want the creation time in Unix timestamp
		"value":      value,
		"ex":         ex,
	}

	_, err := collection.InsertOne(ctx, doc)
	return errs.Wrap(err)
}

func (u *UserMgo) DeleteUserCommand(ctx context.Context, userID string, Type int32, UUID string) error {
	collection := u.coll.Database().Collection("userCommands")

	filter := bson.M{"userID": userID, "type": Type, "uuid": UUID}

	result, err := collection.DeleteOne(ctx, filter)
	// when err is not nil, result might be nil
	if err != nil {
		return errs.Wrap(err)
	}
	if result.DeletedCount == 0 {
		// No records found to update
		return errs.Wrap(errs.ErrRecordNotFound)
	}
	return errs.Wrap(err)
}
func (u *UserMgo) UpdateUserCommand(ctx context.Context, userID string, Type int32, UUID string, val map[string]any)
	if len(val) == 0 {
		return nil
	}

	collection := u.coll.Database().Collection("userCommands")

	filter := bson.M{"userID": userID, "type": Type, "uuid": UUID}
	update := bson.M{"$set": val}

	result, err := collection.UpdateOne(ctx, filter, update)
	if err != nil {
```

```
    return errs.Wrap(err)
  }

  if result.MatchedCount == 0 {
    // No records found to update
    return errs.Wrap(errs.ErrRecordNotFound)
  }

  return nil
}

func (u *UserMgo) GetUserCommand(ctx context.Context, userID string, Type int32) ([]*user.CommandInfoResp, error) {
  collection := u.coll.Database().Collection("userCommands")
  filter := bson.M{"userID": userID, "type": Type}

  cursor, err := collection.Find(ctx, filter)
  if err != nil {
    return nil, err
  }
  defer cursor.Close(ctx)

  // Initialize commands as a slice of pointers
  commands := []*user.CommandInfoResp{}

  for cursor.Next(ctx) {
    var document struct {
      Type       int32  `bson:"type"`
      UUID       string `bson:"uuid"`
      Value      string `bson:"value"`
      CreateTime int64  `bson:"createTime"`
      Ex         string `bson:"ex"`
    }

    if err := cursor.Decode(&document); err != nil {
      return nil, err
    }

    commandInfo := &user.CommandInfoResp{
      Type:       document.Type,
      Uuid:       document.UUID,
      Value:      document.Value,
      CreateTime: document.CreateTime,
      Ex:         document.Ex,
    }

    commands = append(commands, commandInfo)
  }

  if err := cursor.Err(); err != nil {
    return nil, errs.Wrap(err)
  }

  return commands, nil
}
func (u *UserMgo) GetAllUserCommand(ctx context.Context, userID string) ([]*user.AllCommandInfoResp, error) {
  collection := u.coll.Database().Collection("userCommands")
  filter := bson.M{"userID": userID}

  cursor, err := collection.Find(ctx, filter)
  if err != nil {
    return nil, errs.Wrap(err)
  }
  defer cursor.Close(ctx)

  // Initialize commands as a slice of pointers
  commands := []*user.AllCommandInfoResp{}
```

```
    for cursor.Next(ctx) {
        var document struct {
            Type       int32  `bson:"type"`
            UUID       string `bson:"uuid"`
            Value      string `bson:"value"`
            CreateTime int64  `bson:"createTime"`
            Ex         string `bson:"ex"`
        }

        if err := cursor.Decode(&document); err != nil {
            return nil, errs.Wrap(err)
        }

        commandInfo := &user.AllCommandInfoResp{
            Type:       document.Type,
            Uuid:       document.UUID,
            Value:      document.Value,
            CreateTime: document.CreateTime,
            Ex:         document.Ex,
        }

        commands = append(commands, commandInfo)
    }

    if err := cursor.Err(); err != nil {
        return nil, errs.Wrap(err)
    }
    return commands, nil
}
func (u *UserMgo) CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int64, er
    pipeline := bson.A{
        bson.M{
            "$match": bson.M{
                "create_time": bson.M{
                    "$gte": start,
                    "$lt":  end,
                },
            },
        },
        bson.M{
            "$group": bson.M{
                "_id": bson.M{
                    "$dateToString": bson.M{
                        "format": "%Y-%m-%d",
                        "date":   "$create_time",
                    },
                },
                "count": bson.M{
                    "$sum": 1,
                },
            },
        },
    }
    type Item struct {
        Date  string `bson:"_id"`
        Count int64  `bson:"count"`
    }
    items, err := mongoutil.Aggregate[Item](ctx, u.coll, pipeline)
    if err != nil {
        return nil, err
    }
    res := make(map[string]int64, len(items))
    for _, item := range items {
        res[item.Date] = item.Count
    }
```

```
    return res, nil
}

func (u *UserMgo) SortQuery(ctx context.Context, userIDName map[string]string, asc bool) ([]*model.User, error) {
    if len(userIDName) == 0 {
        return nil, nil
    }
    userIDs := make([]string, 0, len(userIDName))
    attached := make(map[string]string)
    for userID, name := range userIDName {
        userIDs = append(userIDs, userID)
        if name == "" {
            continue
        }
        attached[userID] = name
    }
    var sortValue int
    if asc {
        sortValue = 1
    } else {
        sortValue = -1
    }
    if len(attached) == 0 {
        filter := bson.M{"user_id": bson.M{"$in": userIDs}}
        opt := options.Find().SetSort(bson.M{"nickname": sortValue})
        return mongoutil.Find[*model.User](ctx, u.coll, filter, opt)
    }
    pipeline := []bson.M{
        {
            "$match": bson.M{
                "user_id": bson.M{"$in": userIDs},
            },
        },
        {
            "$addFields": bson.M{
                "_query_sort_name": bson.M{
                    "$arrayElemAt": []any{
                        bson.M{
                            "$filter": bson.M{
                                "input": bson.M{
                                    "$objectToArray": attached,
                                },
                                "as": "item",
                                "cond": bson.M{
                                    "$eq": []any{"$$item.k", "$user_id"},
                                },
                            },
                        },
                        0,
                    },
                },
            },
        },
        {
            "$addFields": bson.M{
                "_query_sort_name": bson.M{
                    "$ifNull": []any{"$_query_sort_name.v", "$nickname"},
                },
            },
        },
        {
            "$sort": bson.M{
                "_query_sort_name": sortValue,
            },
        },
    }
```

```
    return mongoutil.Aggregate[*model.User](ctx, u.coll, pipeline)
}
```

pkg/common/storage/database/mgo/version_log.go

```
package mgo

import (
	"context"
	"errors"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/versionctx"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/bson/primitive"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

func NewVersionLog(coll *mongo.Collection) (database.VersionLog, error) {
	lm := &VersionLogMgo{coll: coll}
	if err := lm.initIndex(context.Background()); err != nil {
		return nil, errs.WrapMsg(err, "init version log index failed", "coll", coll.Name())
	}
	return lm, nil
}

type VersionLogMgo struct {
	coll *mongo.Collection
}

func (l *VersionLogMgo) initIndex(ctx context.Context) error {
	_, err := l.coll.Indexes().CreateOne(ctx, mongo.IndexModel{
		Keys: bson.M{
			"d_id": 1,
		},
		Options: options.Index().SetUnique(true),
	})

	return err
}

func (l *VersionLogMgo) IncrVersion(ctx context.Context, dId string, eIds []string, state int32) error {
	_, err := l.IncrVersionResult(ctx, dId, eIds, state)
	return err
}

func (l *VersionLogMgo) IncrVersionResult(ctx context.Context, dId string, eIds []string, state int32) (*model.Versi
	vl, err := l.incrVersionResult(ctx, dId, eIds, state)
	if err != nil {
		return nil, err
	}
	versionctx.GetVersionLog(ctx).Append(versionctx.Collection{
		Name: l.coll.Name(),
		Doc:  vl,
	})
	return vl, nil
}

func (l *VersionLogMgo) incrVersionResult(ctx context.Context, dId string, eIds []string, state int32) (*model.Versi
	if len(eIds) == 0 {
		return nil, errs.ErrArgs.WrapMsg("elem id is empty", "dId", dId)
	}
	now := time.Now()
```

```
	if res, err := l.writeLogBatch2(ctx, dId, eIds, state, now); err == nil {
		return res, nil
	} else if !errors.Is(err, mongo.ErrNoDocuments) {
		return nil, err
	}
	if res, err := l.initDoc(ctx, dId, eIds, state, now); err == nil {
		return res, nil
	} else if !mongo.IsDuplicateKeyError(err) {
		return nil, err
	}
	return l.writeLogBatch2(ctx, dId, eIds, state, now)
}

func (l *VersionLogMgo) initDoc(ctx context.Context, dId string, eIds []string, state int32, now time.Time) (*model.
	wl := model.VersionLogTable{
		ID:         primitive.NewObjectID(),
		DID:        dId,
		Logs:       make([]model.VersionLogElem, 0, len(eIds)),
		Version:    database.FirstVersion,
		Deleted:    database.DefaultDeleteVersion,
		LastUpdate: now,
	}
	for _, eId := range eIds {
		wl.Logs = append(wl.Logs, model.VersionLogElem{
			EID:        eId,
			State:      state,
			Version:    database.FirstVersion,
			LastUpdate: now,
		})
	}
	if _, err := l.coll.InsertOne(ctx, &wl); err != nil {
		return nil, err
	}
	return wl.VersionLog(), nil
}

func (l *VersionLogMgo) writeLogBatch2(ctx context.Context, dId string, eIds []string, state int32, now time.Time) (
	if eIds == nil {
		eIds = []string{}
	}
	filter := bson.M{
		"d_id": dId,
	}
	elems := make([]bson.M, 0, len(eIds))
	for _, eId := range eIds {
		elems = append(elems, bson.M{
			"e_id":        eId,
			"version":     "$version",
			"state":       state,
			"last_update": now,
		})
	}
	pipeline := []bson.M{
		{
			"$addFields": bson.M{
				"delete_e_ids": eIds,
			},
		},
		{
			"$set": bson.M{
				"version":     bson.M{"$add": []any{"$version", 1}},
				"last_update": now,
			},
		},
		{
			"$set": bson.M{
```

```
        "logs": bson.M{
            "$filter": bson.M{
                "input": "$logs",
                "as":    "log",
                "cond": bson.M{
                    "$not": bson.M{
                        "$in": []any{"$$log.e_id", "$delete_e_ids"},
                    },
                },
            },
        },
    },
    {
        "$set": bson.M{
            "logs": bson.M{
                "$concatArrays": []any{
                    "$logs",
                    elems,
                },
            },
        },
    },
    {
        "$unset": "delete_e_ids",
    },
}
projection := bson.M{
    "logs": 0,
}
opt := options.FindOneAndUpdate().SetUpsert(false).SetReturnDocument(options.After).SetProjection(projection)
res, err := mongoutil.FindOneAndUpdate[*model.VersionLog](ctx, l.coll, filter, pipeline, opt)
if err != nil {
    return nil, err
}
res.Logs = make([]model.VersionLogElem, 0, len(eIds))
for _, id := range eIds {
    res.Logs = append(res.Logs, model.VersionLogElem{
        EID:        id,
        State:      state,
        Version:    res.Version,
        LastUpdate: res.LastUpdate,
    })
}
return res, nil
}

func (l *VersionLogMgo) findDoc(ctx context.Context, dId string) (*model.VersionLog, error) {
vl, err := mongoutil.FindOne[*model.VersionLogTable](ctx, l.coll, bson.M{"d_id": dId}, options.FindOne().SetProject
if err != nil {
    return nil, err
}
return vl.VersionLog(), nil
}

func (l *VersionLogMgo) FindChangeLog(ctx context.Context, dId string, version uint, limit int) (*model.VersionLog,
if wl, err := l.findChangeLog(ctx, dId, version, limit); err == nil {
    return wl, nil
} else if !errors.Is(err, mongo.ErrNoDocuments) {
    return nil, err
}
log.ZDebug(ctx, "init doc", "dId", dId)
if res, err := l.initDoc(ctx, dId, nil, 0, time.Now()); err == nil {
    log.ZDebug(ctx, "init doc success", "dId", dId)
    return res, nil
} else if mongo.IsDuplicateKeyError(err) {
```

```
    return l.findChangeLog(ctx, dId, version, limit)
  } else {
    return nil, err
  }
}

func (l *VersionLogMgo) BatchFindChangeLog(ctx context.Context, dIds []string, versions []uint, limits []int) (vLogs
  for i := 0; i < len(dIds); i++ {
    if vLog, err := l.findChangeLog(ctx, dIds[i], versions[i], limits[i]); err == nil {
      vLogs = append(vLogs, vLog)
    } else if !errors.Is(err, mongo.ErrNoDocuments) {
      log.ZError(ctx, "findChangeLog error:", errs.Wrap(err))
    }
    log.ZDebug(ctx, "init doc", "dId", dIds[i])
    if res, err := l.initDoc(ctx, dIds[i], nil, 0, time.Now()); err == nil {
      log.ZDebug(ctx, "init doc success", "dId", dIds[i])
      vLogs = append(vLogs, res)
    } else if mongo.IsDuplicateKeyError(err) {
      l.findChangeLog(ctx, dIds[i], versions[i], limits[i])
    } else {
      log.ZError(ctx, "init doc error:", errs.Wrap(err))
    }
  }
  return vLogs, errs.Wrap(err)
}

func (l *VersionLogMgo) findChangeLog(ctx context.Context, dId string, version uint, limit int) (*model.VersionLog,
  if version == 0 && limit == 0 {
    return l.findDoc(ctx, dId)
  }
  pipeline := []bson.M{
    {
      "$match": bson.M{
        "d_id": dId,
      },
    },
    {
      "$addFields": bson.M{
        "logs": bson.M{
          "$cond": bson.M{
            "if": bson.M{
              "$or": []bson.M{
                {"$lt": []any{"$version", version}},
                {"$gte": []any{"$deleted", version}},
              },
            },
            "then": []any{},
            "else": "$logs",
          },
        },
      },
    },
    {
      "$addFields": bson.M{
        "logs": bson.M{
          "$filter": bson.M{
            "input": "$logs",
            "as":    "l",
            "cond": bson.M{
              "$gt": []any{"$$l.version", version},
            },
          },
        },
      },
    },
    {
```

```
        "$addFields": bson.M{
            "log_len": bson.M{"$size": "$logs"},
        },
    },
    {
        "$addFields": bson.M{
            "logs": bson.M{
                "$cond": bson.M{
                    "if": bson.M{
                        "$gt": []any{"$log_len", limit},
                    },
                    "then": []any{},
                    "else": "$logs",
                },
            },
        },
    },
}
if limit <= 0 {
    pipeline = pipeline[:len(pipeline)-1]
}
vl, err := mongoutil.Aggregate[*model.VersionLog](ctx, l.coll, pipeline)
if err != nil {
    return nil, err
}
if len(vl) == 0 {
    return nil, mongo.ErrNoDocuments
}
return vl[0], nil
}

func (l *VersionLogMgo) DeleteAfterUnchangedLog(ctx context.Context, deadline time.Time) error {
return mongoutil.DeleteMany(ctx, l.coll, bson.M{
    "last_update": bson.M{
        "$lt": deadline,
    },
})
}

func (l *VersionLogMgo) Delete(ctx context.Context, dId string) error {
return mongoutil.DeleteOne(ctx, l.coll, bson.M{"d_id": dId})
}
```

pkg/common/storage/database/mgo/version_test.go

```go
package mgo

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
	"testing"
	"time"
)

//func Result[V any](val V, err error) V {
//	if err != nil {
//		panic(err)
//	}
//	return val
//}

func Check(err error) {
	if err != nil {
		panic(err)
	}
}

func TestName(t *testing.T) {
	cli := Result(mongo.Connect(context.Background(), options.Client().ApplyURI("mongodb://openIM:openIM123@172.16.8.48
	coll := cli.Database("openim_v3").Collection("version_test")
	tmp, err := NewVersionLog(coll)
	if err != nil {
		panic(err)
	}
	vl := tmp.(*VersionLogMgo)
	res, err := vl.incrVersionResult(context.Background(), "100", []string{"1000", "1001", "1003"}, model.VersionStateI
	if err != nil {
		t.Log(err)
		return
	}
	t.Logf("%+v", res)
}
```

pkg/common/storage/common

pkg/common/storage/common/types.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package common

type BatchUpdateGroupMember struct {
	GroupID string
	UserID  string
	Map     map[string]any
}

type GroupSimpleUserID struct {
	Hash      uint64
	MemberNum uint32
}
```

pkg/common/storage/controller

pkg/common/storage/controller/auth.go

```
package controller

import (
	"context"

	"github.com/golang-jwt/jwt/v4"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/tokenverify"
)

type AuthDatabase interface {
	// If the result is empty, no error is returned.
	GetTokensWithoutError(ctx context.Context, userID string, platformID int) (map[string]int, error)

	GetTemporaryTokensWithoutError(ctx context.Context, userID string, platformID int, token string) error
	// Create token
	CreateToken(ctx context.Context, userID string, platformID int) (string, error)

	BatchSetTokenMapByUidPid(ctx context.Context, tokens []string) error

	SetTokenMapByUidPid(ctx context.Context, userID string, platformID int, m map[string]int) error
}

type multiLoginConfig struct {
	Policy       int
	MaxNumOneEnd int
}

type authDatabase struct {
	cache        cache.TokenModel
	accessSecret string
	accessExpire int64
	multiLogin   multiLoginConfig
	adminUserIDs []string
}

func NewAuthDatabase(cache cache.TokenModel, accessSecret string, accessExpire int64, multiLogin config.MultiLogin,
	return &authDatabase{cache: cache, accessSecret: accessSecret, accessExpire: accessExpire, multiLogin: multiLoginCo
		Policy:       multiLogin.Policy,
		MaxNumOneEnd: multiLogin.MaxNumOneEnd,
	},
		adminUserIDs: adminUserIDs,
	}
}

// If the result is empty.
func (a *authDatabase) GetTokensWithoutError(ctx context.Context, userID string, platformID int) (map[string]int, er
	return a.cache.GetTokensWithoutError(ctx, userID, platformID)
}

func (a *authDatabase) GetTemporaryTokensWithoutError(ctx context.Context, userID string, platformID int, token stri
	return a.cache.HasTemporaryToken(ctx, userID, platformID, token)
}

func (a *authDatabase) SetTokenMapByUidPid(ctx context.Context, userID string, platformID int, m map[string]int) err
```

```
	return a.cache.SetTokenMapByUidPid(ctx, userID, platformID, m)
}

func (a *authDatabase) BatchSetTokenMapByUidPid(ctx context.Context, tokens []string) error {
	setMap := make(map[string]map[string]any)
	for _, token := range tokens {
		claims, err := tokenverify.GetClaimFromToken(token, authverify.Secret(a.accessSecret))
		if err != nil {
			continue
		}
		key := cachekey.GetTokenKey(claims.UserID, claims.PlatformID)
		if v, ok := setMap[key]; ok {
			v[token] = constant.KickedToken
		} else {
			setMap[key] = map[string]any{
				token: constant.KickedToken,
			}
		}
	}
	if err := a.cache.BatchSetTokenMapByUidPid(ctx, setMap); err != nil {
		return err
	}
	return nil
}

// Create Token.
func (a *authDatabase) CreateToken(ctx context.Context, userID string, platformID int) (string, error) {
	tokens, err := a.cache.GetAllTokensWithoutError(ctx, userID)
	if err != nil {
		return "", err
	}

	deleteTokenKey, kickedTokenKey, adminTokens, err := a.checkToken(ctx, tokens, platformID)
	if err != nil {
		return "", err
	}
	if len(deleteTokenKey) != 0 {
		err = a.cache.DeleteTokenByTokenMap(ctx, userID, deleteTokenKey)
		if err != nil {
			return "", err
		}
	}
	if len(kickedTokenKey) != 0 {
		for plt, ks := range kickedTokenKey {
			for _, k := range ks {
				err := a.cache.SetTokenFlagEx(ctx, userID, plt, k, constant.KickedToken)
				if err != nil {
					return "", err
				}
				log.ZDebug(ctx, "kicked token in create token", "token", k)
			}
		}
	}
	if len(adminTokens) != 0 {
		if err = a.cache.DeleteAndSetTemporary(ctx, userID, constant.AdminPlatformID, adminTokens); err != nil {
			return "", err
		}
	}

	claims := tokenverify.BuildClaims(userID, platformID, a.accessExpire)
	token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
	tokenString, err := token.SignedString([]byte(a.accessSecret))
	if err != nil {
		return "", errs.WrapMsg(err, "token.SignedString")
	}
```

```
	if err = a.cache.SetTokenFlagEx(ctx, userID, platformID, tokenString, constant.NormalToken); err != nil {
		return "", err
	}

	return tokenString, nil
}

// checkToken will check token by tokenPolicy and return deleteToken,kickToken,deleteAdminToken
func (a *authDatabase) checkToken(ctx context.Context, tokens map[int]map[string]int, platformID int) (map[int][]str
	// todo: Asynchronous deletion of old data.
	var (
		loginTokenMap  = make(map[int][]string) // The length of the value of the map must be greater than 0
		deleteToken    = make(map[int][]string)
		kickToken      = make(map[int][]string)
		adminToken     = make([]string, 0)
		unkickTerminal = ""
	)

	for plfID, tks := range tokens {
		for k, v := range tks {
			_, err := tokenverify.GetClaimFromToken(k, authverify.Secret(a.accessSecret))
			if err != nil || v != constant.NormalToken {
				deleteToken[plfID] = append(deleteToken[plfID], k)
			} else {
				if plfID != constant.AdminPlatformID {
					loginTokenMap[plfID] = append(loginTokenMap[plfID], k)
				} else {
					adminToken = append(adminToken, k)
				}
			}
		}
	}

	switch a.multiLogin.Policy {
	case constant.DefalutNotKick:
		for plt, ts := range loginTokenMap {
			l := len(ts)
			if platformID == plt {
				l++
			}
			limit := a.multiLogin.MaxNumOneEnd
			if l > limit {
				kickToken[plt] = ts[:l-limit]
			}
		}
	case constant.AllLoginButSameTermKick:
		for plt, ts := range loginTokenMap {
			kickToken[plt] = ts[:len(ts)-1]

			if plt == platformID {
				kickToken[plt] = append(kickToken[plt], ts[len(ts)-1])
			}
		}
	case constant.PCAndOther:
		unkickTerminal = constant.TerminalPC
		if constant.PlatformIDToClass(platformID) != unkickTerminal {
			for plt, ts := range loginTokenMap {
				if constant.PlatformIDToClass(plt) != unkickTerminal {
					kickToken[plt] = ts
				}
			}
		} else {
			var (
				preKickToken string
				preKickPlt   int
				reserveToken = false
```

```
    )
    for plt, ts := range loginTokenMap {
        if constant.PlatformIDToClass(plt) != unkickTerminal {
            // Keep a token from another end
            if !reserveToken {
                reserveToken = true
                kickToken[plt] = ts[:len(ts)-1]
                preKickToken = ts[len(ts)-1]
                preKickPlt = plt
                continue
            } else {
                // Prioritize keeping Android
                if plt == constant.AndroidPlatformID {
                    if preKickToken != "" {
                        kickToken[preKickPlt] = append(kickToken[preKickPlt], preKickToken)
                    }
                    kickToken[plt] = ts[:len(ts)-1]
                } else {
                    kickToken[plt] = ts
                }
            }
        }
    }
  case constant.AllLoginButSameClassKick:
    var (
        reserved = make(map[string]struct{})
    )

    for plt, ts := range loginTokenMap {
        if constant.PlatformIDToClass(plt) == constant.PlatformIDToClass(platformID) {
            kickToken[plt] = ts
        } else {
            if _, ok := reserved[constant.PlatformIDToClass(plt)]; !ok {
                reserved[constant.PlatformIDToClass(plt)] = struct{}{}
                kickToken[plt] = ts[:len(ts)-1]
                continue
            } else {
                kickToken[plt] = ts
            }
        }
    }
  default:
    return nil, nil, nil, errs.New("unknown multiLogin policy").Wrap()
  }

  //var adminTokenMaxNum = a.multiLogin.MaxNumOneEnd
  //l := len(adminToken)
  //if platformID == constant.AdminPlatformID {
  //  l++
  //}
  //if l > adminTokenMaxNum {
  //  kickToken = append(kickToken, adminToken[:l-adminTokenMaxNum]...)
  //}
  var deleteAdminToken []string
  if platformID == constant.AdminPlatformID {
    deleteAdminToken = adminToken
  }
  return deleteToken, kickToken, deleteAdminToken, nil
}
```

pkg/common/storage/controller/black.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
    "context"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/tools/db/pagination"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/datautil"
)

type BlackDatabase interface {
    // Create add BlackList
    Create(ctx context.Context, blacks []*model.Black) (err error)
    // Delete delete BlackList
    Delete(ctx context.Context, blacks []*model.Black) (err error)
    // FindOwnerBlacks get BlackList list
    FindOwnerBlacks(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (total int64, blacks []*
    FindBlackInfos(ctx context.Context, ownerUserID string, userIDs []string) (blacks []*model.Black, err error)
    // CheckIn Check whether user2 is in the black list of user1 (inUser1Blacks==true) Check whether user1 is in the bl
    CheckIn(ctx context.Context, userID1, userID2 string) (inUser1Blacks bool, inUser2Blacks bool, err error)
}

type blackDatabase struct {
    black database.Black
    cache cache.BlackCache
}

func NewBlackDatabase(black database.Black, cache cache.BlackCache) BlackDatabase {
    return &blackDatabase{black, cache}
}

// Create Add Blacklist.
func (b *blackDatabase) Create(ctx context.Context, blacks []*model.Black) (err error) {
    if err := b.black.Create(ctx, blacks); err != nil {
        return err
    }
    return b.deleteBlackIDsCache(ctx, blacks)
}

// Delete Delete Blacklist.
func (b *blackDatabase) Delete(ctx context.Context, blacks []*model.Black) (err error) {
    if err := b.black.Delete(ctx, blacks); err != nil {
        return err
    }
    return b.deleteBlackIDsCache(ctx, blacks)
}

// FindOwnerBlacks Get Blacklist List.
```

```go
func (b *blackDatabase) deleteBlackIDsCache(ctx context.Context, blacks []*model.Black) (err error) {
    cache := b.cache.CloneBlackCache()
    for _, black := range blacks {
        cache = cache.DelBlackIDs(ctx, black.OwnerUserID)
    }
    return cache.ChainExecDel(ctx)
}

// FindOwnerBlacks Get Blacklist List.
func (b *blackDatabase) FindOwnerBlacks(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (
    return b.black.FindOwnerBlacks(ctx, ownerUserID, pagination)
}

// FindOwnerBlacks Get Blacklist List.
func (b *blackDatabase) CheckIn(ctx context.Context, userID1, userID2 string) (inUser1Blacks bool, inUser2Blacks boo
    userID1BlackIDs, err := b.cache.GetBlackIDs(ctx, userID1)
    if err != nil {
        return
    }
    userID2BlackIDs, err := b.cache.GetBlackIDs(ctx, userID2)
    if err != nil {
        return
    }
    log.ZDebug(ctx, "blackIDs", "user1BlackIDs", userID1BlackIDs, "user2BlackIDs", userID2BlackIDs)
    return datautil.Contain(userID2, userID1BlackIDs...), datautil.Contain(userID1, userID2BlackIDs...), nil
}

// FindBlackIDs Get Blacklist List.
func (b *blackDatabase) FindBlackIDs(ctx context.Context, ownerUserID string) (blackIDs []string, err error) {
    return b.cache.GetBlackIDs(ctx, ownerUserID)
}

// FindBlackInfos Get Blacklist List.
func (b *blackDatabase) FindBlackInfos(ctx context.Context, ownerUserID string, userIDs []string) (blacks []*model.B
    return b.black.FindOwnerBlackInfos(ctx, ownerUserID, userIDs)
}
```

pkg/common/storage/controller/client_config.go

```
package controller

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/db/tx"
)

type ClientConfigDatabase interface {
	SetUserConfig(ctx context.Context, userID string, config map[string]string) error
	GetUserConfig(ctx context.Context, userID string) (map[string]string, error)
	DelUserConfig(ctx context.Context, userID string, keys []string) error
	GetUserConfigPage(ctx context.Context, userID string, key string, pagination pagination.Pagination) (int64, []*mode
}

func NewClientConfigDatabase(db database.ClientConfig, cache cache.ClientConfigCache, tx tx.Tx) ClientConfigDatabase
	return &clientConfigDatabase{
		tx:    tx,
		db:    db,
		cache: cache,
	}
}

type clientConfigDatabase struct {
	tx    tx.Tx
	db    database.ClientConfig
	cache cache.ClientConfigCache
}

func (x *clientConfigDatabase) SetUserConfig(ctx context.Context, userID string, config map[string]string) error {
	return x.tx.Transaction(ctx, func(ctx context.Context) error {
		if err := x.db.Set(ctx, userID, config); err != nil {
			return err
		}
		return x.cache.DeleteUserCache(ctx, []string{userID})
	})
}

func (x *clientConfigDatabase) GetUserConfig(ctx context.Context, userID string) (map[string]string, error) {
	return x.cache.GetUserConfig(ctx, userID)
}

func (x *clientConfigDatabase) DelUserConfig(ctx context.Context, userID string, keys []string) error {
	return x.tx.Transaction(ctx, func(ctx context.Context) error {
		if err := x.db.Del(ctx, userID, keys); err != nil {
			return err
		}
		return x.cache.DeleteUserCache(ctx, []string{userID})
	})
}

func (x *clientConfigDatabase) GetUserConfigPage(ctx context.Context, userID string, key string, pagination paginati
	return x.db.GetPage(ctx, userID, key, pagination)
}
```

pkg/common/storage/controller/conversation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/tools/db/pagination"
    "github.com/openimsdk/tools/db/tx"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/datautil"
    "github.com/openimsdk/tools/utils/stringutil"
)

type ConversationDatabase interface {
    // UpdateUsersConversationField updates the properties of a conversation for specified users.
    UpdateUsersConversationField(ctx context.Context, userIDs []string, conversationID string, args map[string]any) err
    // CreateConversation creates a batch of new conversations.
    CreateConversation(ctx context.Context, conversations []*relationtb.Conversation) error
    // SyncPeerUserPrivateConversationTx ensures transactional operation while syncing private conversations between pe
    SyncPeerUserPrivateConversationTx(ctx context.Context, conversation []*relationtb.Conversation) error
    // FindConversations retrieves multiple conversations of a user by conversation IDs.
    FindConversations(ctx context.Context, ownerUserID string, conversationIDs []string) ([]*relationtb.Conversation, e
    // GetUserAllConversation fetches all conversations of a user on the server.
    GetUserAllConversation(ctx context.Context, ownerUserID string) ([]*relationtb.Conversation, error)
    // SetUserConversations sets multiple conversation properties for a user, creates new conversations if they do not
    SetUserConversations(ctx context.Context, ownerUserID string, conversations []*relationtb.Conversation) error
    // SetUsersConversationFieldTx updates a specific field for multiple users' conversations, creating new conversatio
    // transactional.
    SetUsersConversationFieldTx(ctx context.Context, userIDs []string, conversation *relationtb.Conversation, fieldMap
    // UpdateUserConversations updates all conversations related to a specified user.
    // This function does NOT update the user's own conversations but rather the conversations where this user is invol
    UpdateUserConversations(ctx context.Context, userID string, args map[string]any) error
    // CreateGroupChatConversation creates a group chat conversation for the specified group ID and user IDs.
    CreateGroupChatConversation(ctx context.Context, groupID string, userIDs []string, conversations *relationtb.Conver
    // GetConversationIDs retrieves conversation IDs for a given user.
    GetConversationIDs(ctx context.Context, userID string) ([]string, error)
    // GetUserConversationIDsHash gets the hash of conversation IDs for a given user.
    GetUserConversationIDsHash(ctx context.Context, ownerUserID string) (hash uint64, err error)
    // GetAllConversationIDs fetches all conversation IDs.
    GetAllConversationIDs(ctx context.Context) ([]string, error)
    // GetAllConversationIDsNumber returns the number of all conversation IDs.
    GetAllConversationIDsNumber(ctx context.Context) (int64, error)
    // PageConversationIDs paginates through conversation IDs based on the specified pagination settings.
    PageConversationIDs(ctx context.Context, pagination pagination.Pagination) (conversationIDs []string, err error)
    // GetConversationsByConversationID retrieves conversations by their IDs.
```

```
	GetConversationsByConversationID(ctx context.Context, conversationIDs []string) ([]*relationtb.Conversation, error)
	// GetConversationIDsNeedDestruct fetches conversations that need to be destructed based on specific criteria.
	GetConversationIDsNeedDestruct(ctx context.Context) ([]*relationtb.Conversation, error)
	// GetConversationNotReceiveMessageUserIDs gets user IDs for users in a conversation who have not received messages
	GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string) ([]string, error)
	// GetUserAllHasReadSeqs(ctx context.Context, ownerUserID string) (map[string]int64, error)
	// FindRecvMsgNotNotifyUserIDs(ctx context.Context, groupID string) ([]string, error)
	FindConversationUserVersion(ctx context.Context, userID string, version uint, limit int) (*relationtb.VersionLog, e
	FindMaxConversationUserVersionCache(ctx context.Context, userID string) (*relationtb.VersionLog, error)
	GetOwnerConversation(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (int64, []*relation
	// GetNotNotifyConversationIDs gets not notify conversationIDs by userID
	GetNotNotifyConversationIDs(ctx context.Context, userID string) ([]string, error)
	// GetPinnedConversationIDs gets pinned conversationIDs by userID
	GetPinnedConversationIDs(ctx context.Context, userID string) ([]string, error)
	// FindRandConversation finds random conversations based on the specified timestamp and limit.
	FindRandConversation(ctx context.Context, ts int64, limit int) ([]*relationtb.Conversation, error)

	DeleteUsersConversations(ctx context.Context, userID string, conversationIDs []string) (err error)
}

func NewConversationDatabase(conversation database.Conversation, cache cache.ConversationCache, tx tx.Tx) Conversati
	return &conversationDatabase{
		conversationDB: conversation,
		cache:          cache,
		tx:             tx,
	}
}

type conversationDatabase struct {
	conversationDB database.Conversation
	cache          cache.ConversationCache
	tx             tx.Tx
}

func (c *conversationDatabase) SetUsersConversationFieldTx(ctx context.Context, userIDs []string, conversation *rela
	return c.tx.Transaction(ctx, func(ctx context.Context) error {
		cache := c.cache.CloneConversationCache()
		if conversation.GroupID != "" {
			cache = cache.DelSuperGroupRecvMsgNotNotifyUserIDs(conversation.GroupID).DelSuperGroupRecvMsgNotNotifyUserIDsHas
		}
		haveUserIDs, err := c.conversationDB.FindUserID(ctx, userIDs, []string{conversation.ConversationID})
		if err != nil {
			return err
		}
		if len(haveUserIDs) > 0 {
			_, err = c.conversationDB.UpdateByMap(ctx, haveUserIDs, conversation.ConversationID, fieldMap)
			if err != nil {
				return err
			}
			cache = cache.DelUsersConversation(conversation.ConversationID, haveUserIDs...)
			if _, ok := fieldMap["has_read_seq"]; ok {
				for _, userID := range haveUserIDs {
					cache = cache.DelUserAllHasReadSeqs(userID, conversation.ConversationID)
				}
			}
			if _, ok := fieldMap["recv_msg_opt"]; ok {
				cache = cache.DelConversationNotReceiveMessageUserIDs(conversation.ConversationID)
				cache = cache.DelConversationNotNotifyMessageUserIDs(userIDs...)
			}
			if _, ok := fieldMap["is_pinned"]; ok {
				cache = cache.DelUserPinnedConversations(userIDs...)
			}
			cache = cache.DelConversationVersionUserIDs(haveUserIDs...)
		}
		NotUserIDs := stringutil.DifferenceString(haveUserIDs, userIDs)
		log.ZDebug(ctx, "SetUsersConversationFieldTx", "NotUserIDs", NotUserIDs, "haveUserIDs", haveUserIDs, "userIDs", us
```

```
    var conversations []*relationtb.Conversation
    now := time.Now()
    for _, v := range NotUserIDs {
        temp := new(relationtb.Conversation)
        if err = datautil.CopyStructFields(temp, conversation); err != nil {
            return err
        }
        temp.OwnerUserID = v
        temp.CreateTime = now
        conversations = append(conversations, temp)
    }
    if len(conversations) > 0 {
        err = c.conversationDB.Create(ctx, conversations)
        if err != nil {
            return err
        }
        cache = cache.DelConversationIDs(NotUserIDs...).DelUserConversationIDsHash(NotUserIDs...).DelConversations(conve
    }
    return cache.ChainExecDel(ctx)
  })
}

func (c *conversationDatabase) UpdateUserConversations(ctx context.Context, userID string, args map[string]any) erro
  conversations, err := c.conversationDB.UpdateUserConversations(ctx, userID, args)
  if err != nil {
    return err
  }
  cache := c.cache.CloneConversationCache()
  for _, conversation := range conversations {
    cache = cache.DelUsersConversation(conversation.ConversationID, conversation.OwnerUserID).DelConversationVersionUs
  }
  return cache.ChainExecDel(ctx)
}

func (c *conversationDatabase) UpdateUsersConversationField(ctx context.Context, userIDs []string, conversationID st
  _, err := c.conversationDB.UpdateByMap(ctx, userIDs, conversationID, args)
  if err != nil {
    return err
  }
  cache := c.cache.CloneConversationCache()
  cache = cache.DelUsersConversation(conversationID, userIDs...).DelConversationVersionUserIDs(userIDs...)
  if _, ok := args["recv_msg_opt"]; ok {
    cache = cache.DelConversationNotReceiveMessageUserIDs(conversationID)
    cache = cache.DelConversationNotNotifyMessageUserIDs(userIDs...)
  }
  if _, ok := args["is_pinned"]; ok {
    cache = cache.DelUserPinnedConversations(userIDs...)
  }
  return cache.ChainExecDel(ctx)
}

func (c *conversationDatabase) CreateConversation(ctx context.Context, conversations []*relationtb.Conversation) err
  if err := c.conversationDB.Create(ctx, conversations); err != nil {
    return err
  }
  var (
    userIDs          []string
    notNotifyUserIDs []string
    pinnedUserIDs    []string
  )

  cache := c.cache.CloneConversationCache()
  for _, conversation := range conversations {
    cache = cache.DelConversations(conversation.OwnerUserID, conversation.ConversationID)
    cache = cache.DelConversationNotReceiveMessageUserIDs(conversation.ConversationID)
    userIDs = append(userIDs, conversation.OwnerUserID)
```

```
    if conversation.RecvMsgOpt == constant.ReceiveNotNotifyMessage {
        notNotifyUserIDs = append(notNotifyUserIDs, conversation.OwnerUserID)
    }
    if conversation.IsPinned {
        pinnedUserIDs = append(pinnedUserIDs, conversation.OwnerUserID)
    }
}
return cache.DelConversationIDs(userIDs...).
    DelUserConversationIDsHash(userIDs...).
    DelConversationVersionUserIDs(userIDs...).
    DelConversationNotNotifyMessageUserIDs(notNotifyUserIDs...).
    DelUserPinnedConversations(pinnedUserIDs...).
    ChainExecDel(ctx)
}

func (c *conversationDatabase) SyncPeerUserPrivateConversationTx(ctx context.Context, conversations []*relationtb.Co
return c.tx.Transaction(ctx, func(ctx context.Context) error {
    cache := c.cache.CloneConversationCache()
    for _, conversation := range conversations {
        cache = cache.DelConversationVersionUserIDs(conversation.OwnerUserID, conversation.UserID)
        for _, v := range [][2]string{{conversation.OwnerUserID, conversation.UserID}, {conversation.UserID, conversatio
            ownerUserID := v[0]
            userID := v[1]
            haveUserIDs, err := c.conversationDB.FindUserID(ctx, []string{ownerUserID}, []string{conversation.ConversationI
            if err != nil {
                return err
            }
            if len(haveUserIDs) > 0 {
                _, err := c.conversationDB.UpdateByMap(ctx, []string{ownerUserID}, conversation.ConversationID, map[string]any
                if err != nil {
                    return err
                }
                cache = cache.DelUsersConversation(conversation.ConversationID, ownerUserID)
            } else {
                newConversation := *conversation
                newConversation.OwnerUserID = ownerUserID
                newConversation.UserID = userID
                newConversation.ConversationID = conversation.ConversationID
                newConversation.IsPrivateChat = conversation.IsPrivateChat
                if err := c.conversationDB.Create(ctx, []*relationtb.Conversation{&newConversation}); err != nil {
                    return err
                }
                cache = cache.DelConversationIDs(ownerUserID).DelUserConversationIDsHash(ownerUserID)
            }
        }
    }
    return cache.ChainExecDel(ctx)
})
}

func (c *conversationDatabase) FindConversations(ctx context.Context, ownerUserID string, conversationIDs []string)
return c.cache.GetConversations(ctx, ownerUserID, conversationIDs)
}

func (c *conversationDatabase) GetConversation(ctx context.Context, ownerUserID string, conversationID string) (*rel
return c.cache.GetConversation(ctx, ownerUserID, conversationID)
}

func (c *conversationDatabase) GetUserAllConversation(ctx context.Context, ownerUserID string) ([]*relationtb.Conver
return c.cache.GetUserAllConversations(ctx, ownerUserID)
}

func (c *conversationDatabase) SetUserConversations(ctx context.Context, ownerUserID string, conversations []*relati
return c.tx.Transaction(ctx, func(ctx context.Context) error {
    cache := c.cache.CloneConversationCache()
    cache = cache.DelConversationVersionUserIDs(ownerUserID).
```

```
    DelConversationNotNotifyMessageUserIDs(ownerUserID).
    DelUserPinnedConversations(ownerUserID)

  groupIDs := datautil.Distinct(datautil.Filter(conversations, func(e *relationtb.Conversation) (string, bool) {
    return e.GroupID, e.GroupID != ""
  }))
  for _, groupID := range groupIDs {
    cache = cache.DelSuperGroupRecvMsgNotNotifyUserIDs(groupID).DelSuperGroupRecvMsgNotNotifyUserIDsHash(groupID)
  }
  var conversationIDs []string
  for _, conversation := range conversations {
    conversationIDs = append(conversationIDs, conversation.ConversationID)
    cache = cache.DelConversations(conversation.OwnerUserID, conversation.ConversationID)
  }
  existConversations, err := c.conversationDB.Find(ctx, ownerUserID, conversationIDs)
  if err != nil {
    return err
  }
  if len(existConversations) > 0 {
    for _, conversation := range conversations {
      err = c.conversationDB.Update(ctx, conversation)
      if err != nil {
        return err
      }
    }
  }
  var existConversationIDs []string
  for _, conversation := range existConversations {
    existConversationIDs = append(existConversationIDs, conversation.ConversationID)
  }

  var notExistConversations []*relationtb.Conversation
  for _, conversation := range conversations {
    if !datautil.Contain(conversation.ConversationID, existConversationIDs...) {
      notExistConversations = append(notExistConversations, conversation)
    }
  }
  if len(notExistConversations) > 0 {
    err = c.conversationDB.Create(ctx, notExistConversations)
    if err != nil {
      return err
    }
    cache = cache.DelConversationIDs(ownerUserID).
      DelUserConversationIDsHash(ownerUserID).
      DelConversationNotReceiveMessageUserIDs(datautil.Slice(notExistConversations, func(e *relationtb.Conversation)
  }
  return cache.ChainExecDel(ctx)
 })
}

// func (c *conversationDatabase) FindRecvMsgNotNotifyUserIDs(ctx context.Context, groupID string) ([]string, error)
//  return c.cache.GetSuperGroupRecvMsgNotNotifyUserIDs(ctx, groupID)
//}

func (c *conversationDatabase) CreateGroupChatConversation(ctx context.Context, groupID string, userIDs []string, co
 return c.tx.Transaction(ctx, func(ctx context.Context) error {
  cache := c.cache.CloneConversationCache()
  conversationID := conversation.ConversationID
  existConversationUserIDs, err := c.conversationDB.FindUserID(ctx, userIDs, []string{conversationID})
  if err != nil {
    return err
  }
  notExistUserIDs := stringutil.DifferenceString(userIDs, existConversationUserIDs)
  var conversations []*relationtb.Conversation
  for _, v := range notExistUserIDs {
    conversation := relationtb.Conversation{
```

```
        ConversationType: conversation.ConversationType, GroupID: groupID, OwnerUserID: v, ConversationID: conversation
        // the parameters have default value
        RecvMsgOpt: conversation.RecvMsgOpt, IsPinned: conversation.IsPinned, IsPrivateChat: conversation.IsPrivateChat
        BurnDuration: conversation.BurnDuration, GroupAtType: conversation.GroupAtType, AttachedInfo: conversation.Atta
        Ex: conversation.Ex, MaxSeq: conversation.MaxSeq, MinSeq: conversation.MinSeq, CreateTime: conversation.CreateT
        MsgDestructTime: conversation.MsgDestructTime, IsMsgDestruct: conversation.IsMsgDestruct, LatestMsgDestructTime
    }

    conversations = append(conversations, &conversation)
    cache = cache.DelConversations(v, conversationID).DelConversationNotReceiveMessageUserIDs(conversationID)
  }
  cache = cache.DelConversationIDs(notExistUserIDs...).DelUserConversationIDsHash(notExistUserIDs...)
  if len(conversations) > 0 {
    err = c.conversationDB.Create(ctx, conversations)
    if err != nil {
      return err
    }
  }
  _, err = c.conversationDB.UpdateByMap(ctx, existConversationUserIDs, conversationID, map[string]any{"max_seq": co
  if err != nil {
    return err
  }
  for _, v := range existConversationUserIDs {
    cache = cache.DelConversations(v, conversationID)
  }
  return cache.ChainExecDel(ctx)
 })
}

func (c *conversationDatabase) GetConversationIDs(ctx context.Context, userID string) ([]string, error) {
 return c.cache.GetUserConversationIDs(ctx, userID)
}

func (c *conversationDatabase) GetUserConversationIDsHash(ctx context.Context, ownerUserID string) (hash uint64, err
 return c.cache.GetUserConversationIDsHash(ctx, ownerUserID)
}

func (c *conversationDatabase) GetAllConversationIDs(ctx context.Context) ([]string, error) {
 return c.conversationDB.GetAllConversationIDs(ctx)
}

func (c *conversationDatabase) GetAllConversationIDsNumber(ctx context.Context) (int64, error) {
 return c.conversationDB.GetAllConversationIDsNumber(ctx)
}

func (c *conversationDatabase) PageConversationIDs(ctx context.Context, pagination pagination.Pagination) ([]string,
 return c.conversationDB.PageConversationIDs(ctx, pagination)
}

func (c *conversationDatabase) GetConversationsByConversationID(ctx context.Context, conversationIDs []string) ([]*r
 return c.conversationDB.GetConversationsByConversationID(ctx, conversationIDs)
}

func (c *conversationDatabase) GetConversationIDsNeedDestruct(ctx context.Context) ([]*relationtb.Conversation, erro
 return c.conversationDB.GetConversationIDsNeedDestruct(ctx)
}

func (c *conversationDatabase) GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string) (
 return c.cache.GetConversationNotReceiveMessageUserIDs(ctx, conversationID)
}

func (c *conversationDatabase) FindConversationUserVersion(ctx context.Context, userID string, version uint, limit i
 return c.conversationDB.FindConversationUserVersion(ctx, userID, version, limit)
}

func (c *conversationDatabase) FindMaxConversationUserVersionCache(ctx context.Context, userID string) (*relationtb.
```

```
	return c.cache.FindMaxConversationUserVersion(ctx, userID)
}

func (c *conversationDatabase) GetOwnerConversation(ctx context.Context, ownerUserID string, pagination pagination.P
	conversationIDs, err := c.cache.GetUserConversationIDs(ctx, ownerUserID)
	if err != nil {
		return 0, nil, err
	}
	findConversationIDs := datautil.Paginate(conversationIDs, int(pagination.GetPageNumber()), int(pagination.GetShowNu
	conversations := make([]*relationtb.Conversation, 0, len(findConversationIDs))
	for _, conversationID := range findConversationIDs {
		conversation, err := c.cache.GetConversation(ctx, ownerUserID, conversationID)
		if err != nil {
			return 0, nil, err
		}
		conversations = append(conversations, conversation)
	}
	return int64(len(conversationIDs)), conversations, nil
}

func (c *conversationDatabase) GetNotNotifyConversationIDs(ctx context.Context, userID string) ([]string, error) {
	conversationIDs, err := c.cache.GetUserNotNotifyConversationIDs(ctx, userID)
	if err != nil {
		return nil, err
	}
	return conversationIDs, nil
}

func (c *conversationDatabase) GetPinnedConversationIDs(ctx context.Context, userID string) ([]string, error) {
	conversationIDs, err := c.cache.GetPinnedConversationIDs(ctx, userID)
	if err != nil {
		return nil, err
	}
	return conversationIDs, nil
}

func (c *conversationDatabase) FindRandConversation(ctx context.Context, ts int64, limit int) ([]*relationtb.Convers
	return c.conversationDB.FindRandConversation(ctx, ts, limit)
}

func (c *conversationDatabase) DeleteUsersConversations(ctx context.Context, userID string, conversationIDs []string
	return c.tx.Transaction(ctx, func(ctx context.Context) error {
		err = c.conversationDB.DeleteUsersConversations(ctx, userID, conversationIDs)
		if err != nil {
			return err
		}
		cache := c.cache.CloneConversationCache()
		cache = cache.DelConversations(userID, conversationIDs...).
			DelConversationVersionUserIDs(userID).
			DelConversationIDs(userID).
			DelUserConversationIDsHash(userID).
			DelConversationNotNotifyMessageUserIDs(userID).
			DelUserPinnedConversations(userID)

		return cache.ChainExecDel(ctx)
	})
}
```

pkg/common/storage/controller/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller // import "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
```

pkg/common/storage/controller/friend.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
	"context"
	"fmt"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/db/tx"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
)

type FriendDatabase interface {
	// CheckIn checks if user2 is in user1's friend list (inUser1Friends==true) and if user1 is in user2's friend list
	CheckIn(ctx context.Context, user1, user2 string) (inUser1Friends bool, inUser2Friends bool, err error)

	// AddFriendRequest adds or updates a friend request
	AddFriendRequest(ctx context.Context, fromUserID, toUserID string, reqMsg string, ex string) (err error)

	// BecomeFriends first checks if the users are already in the friends model; if not, it inserts them as friends
	BecomeFriends(ctx context.Context, ownerUserID string, friendUserIDs []string, addSource int32) (err error)

	// RefuseFriendRequest refuses a friend request
	RefuseFriendRequest(ctx context.Context, friendRequest *model.FriendRequest) (err error)

	// AgreeFriendRequest accepts a friend request
	AgreeFriendRequest(ctx context.Context, friendRequest *model.FriendRequest) (err error)

	// Delete removes a friend or friends from the owner's friend list
	Delete(ctx context.Context, ownerUserID string, friendUserIDs []string) (err error)

	// UpdateRemark updates the remark for a friend
	UpdateRemark(ctx context.Context, ownerUserID, friendUserID, remark string) (err error)

	// PageOwnerFriends retrieves the friend list of ownerUserID with pagination
	PageOwnerFriends(ctx context.Context, ownerUserID string, pagination pagination.Pagination) (total int64, friends |

	// PageInWhoseFriends finds the users who have friendUserID in their friend list with pagination
	PageInWhoseFriends(ctx context.Context, friendUserID string, pagination pagination.Pagination) (total int64, friend

	// PageFriendRequestFromMe retrieves the friend requests sent by the user with pagination
```

```
	PageFriendRequestFromMe(ctx context.Context, userID string, handleResults []int, pagination pagination.Pagination)

	// PageFriendRequestToMe retrieves the friend requests received by the user with pagination
	PageFriendRequestToMe(ctx context.Context, userID string, handleResults []int, pagination pagination.Pagination) (t

	// FindFriendsWithError fetches specified friends of a user and returns an error if any do not exist
	FindFriendsWithError(ctx context.Context, ownerUserID string, friendUserIDs []string) (friends []*model.Friend, err

	// FindFriendUserIDs retrieves the friend IDs of a user
	FindFriendUserIDs(ctx context.Context, ownerUserID string) (friendUserIDs []string, err error)

	// FindBothFriendRequests finds friend requests sent and received
	FindBothFriendRequests(ctx context.Context, fromUserID, toUserID string) (friends []*model.FriendRequest, err error

	// UpdateFriends updates fields for friends
	UpdateFriends(ctx context.Context, ownerUserID string, friendUserIDs []string, val map[string]any) (err error)

	//FindSortFriendUserIDs(ctx context.Context, ownerUserID string) ([]string, error)

	FindFriendIncrVersion(ctx context.Context, ownerUserID string, version uint, limit int) (*model.VersionLog, error)

	FindMaxFriendVersionCache(ctx context.Context, ownerUserID string) (*model.VersionLog, error)

	FindFriendUserID(ctx context.Context, friendUserID string) ([]string, error)

	OwnerIncrVersion(ctx context.Context, ownerUserID string, friendUserIDs []string, state int32) error

	GetUnhandledCount(ctx context.Context, userID string, ts int64) (int64, error)
}

type friendDatabase struct {
	friend        database.Friend
	friendRequest database.FriendRequest
	tx            tx.Tx
	cache         cache.FriendCache
}

func NewFriendDatabase(friend database.Friend, friendRequest database.FriendRequest, cache cache.FriendCache, tx tx.
	return &friendDatabase{friend: friend, friendRequest: friendRequest, cache: cache, tx: tx}
}

// CheckIn verifies if user2 is in user1's friend list (inUser1Friends returns true) and
// if user1 is in user2's friend list (inUser2Friends returns true).
func (f *friendDatabase) CheckIn(ctx context.Context, userID1, userID2 string) (inUser1Friends bool, inUser2Friends
	// Retrieve friend IDs of userID1 from the cache
	userID1FriendIDs, err := f.cache.GetFriendIDs(ctx, userID1)
	if err != nil {
		err = fmt.Errorf("error retrieving friend IDs for user %s: %w", userID1, err)
		return
	}

	// Retrieve friend IDs of userID2 from the cache
	userID2FriendIDs, err := f.cache.GetFriendIDs(ctx, userID2)
	if err != nil {
		err = fmt.Errorf("error retrieving friend IDs for user %s: %w", userID2, err)
		return
	}

	// Check if userID2 is in userID1's friend list and vice versa
	inUser1Friends = datautil.Contain(userID2, userID1FriendIDs...)
	inUser2Friends = datautil.Contain(userID1, userID2FriendIDs...)
	return inUser1Friends, inUser2Friends, nil
}

// AddFriendRequest adds or updates a friend request.
func (f *friendDatabase) AddFriendRequest(ctx context.Context, fromUserID, toUserID string, reqMsg string, ex string
```

```
    return f.tx.Transaction(ctx, func(ctx context.Context) error {
        _, err := f.friendRequest.Take(ctx, fromUserID, toUserID)
        switch {
        case err == nil:
            m := make(map[string]any, 1)
            m["handle_result"] = 0
            m["handle_msg"] = ""
            m["req_msg"] = reqMsg
            m["ex"] = ex
            m["create_time"] = time.Now()
            return f.friendRequest.UpdateByMap(ctx, fromUserID, toUserID, m)
        case mgo.IsNotFound(err):
            return f.friendRequest.Create(
                ctx,
                []*model.FriendRequest{{FromUserID: fromUserID, ToUserID: toUserID, ReqMsg: reqMsg, Ex: ex, CreateTime: time.No
            )
        default:
            return err
        }
    })
}

// (1) First determine whether it is in the friends list (in or out does not return an error) (2) for not in the fri
func (f *friendDatabase) BecomeFriends(ctx context.Context, ownerUserID string, friendUserIDs []string, addSource in
    return f.tx.Transaction(ctx, func(ctx context.Context) error {
        cache := f.cache.CloneFriendCache()
        // user find friends
        myFriends, err := f.friend.FindFriends(ctx, ownerUserID, friendUserIDs)
        if err != nil {
            return err
        }
        addOwners, err := f.friend.FindReversalFriends(ctx, ownerUserID, friendUserIDs)
        if err != nil {
            return err
        }
        opUserID := mcontext.GetOpUserID(ctx)
        friends := make([]*model.Friend, 0, len(friendUserIDs)*2)
        myFriendsSet := datautil.SliceSetAny(myFriends, func(friend *model.Friend) string {
            return friend.FriendUserID
        })
        addOwnersSet := datautil.SliceSetAny(addOwners, func(friend *model.Friend) string {
            return friend.OwnerUserID
        })
        newMyFriendIDs := make([]string, 0, len(friendUserIDs))
        newMyOwnerIDs := make([]string, 0, len(friendUserIDs))
        for _, userID := range friendUserIDs {
            if ownerUserID == userID {
                continue
            }
            if _, ok := myFriendsSet[userID]; !ok {
                myFriendsSet[userID] = struct{}{}
                newMyFriendIDs = append(newMyFriendIDs, userID)
                friends = append(friends, &model.Friend{OwnerUserID: ownerUserID, FriendUserID: userID, AddSource: addSource, C
            }
            if _, ok := addOwnersSet[userID]; !ok {
                addOwnersSet[userID] = struct{}{}
                newMyOwnerIDs = append(newMyOwnerIDs, userID)
                friends = append(friends, &model.Friend{OwnerUserID: userID, FriendUserID: ownerUserID, AddSource: addSource, C
            }
        }
        if len(friends) == 0 {
            return nil
        }
        err = f.friend.Create(ctx, friends)
        if err != nil {
            return err
```

```
  }
  cache = cache.DelFriendIDs(ownerUserID).DelMaxFriendVersion(ownerUserID)
  if len(newMyFriendIDs) > 0 {
    cache = cache.DelFriendIDs(newMyFriendIDs...)
    cache = cache.DelFriends(ownerUserID, newMyFriendIDs).DelMaxFriendVersion(newMyFriendIDs...)
  }
  if len(newMyOwnerIDs) > 0 {
    cache = cache.DelFriendIDs(newMyOwnerIDs...)
    cache = cache.DelOwner(ownerUserID, newMyOwnerIDs).DelMaxFriendVersion(newMyOwnerIDs...)
  }
  return cache.ChainExecDel(ctx)
 })
}

// RefuseFriendRequest rejects a friend request. It first checks for an existing, unprocessed request.
// If no such request exists, it returns an error. Otherwise, it marks the request as refused.
func (f *friendDatabase) RefuseFriendRequest(ctx context.Context, friendRequest *model.FriendRequest) error {
 // Attempt to retrieve the friend request from the database.
 fr, err := f.friendRequest.Take(ctx, friendRequest.FromUserID, friendRequest.ToUserID)
 if err != nil {
  return fmt.Errorf("failed to retrieve friend request from %s to %s: %w", friendRequest.FromUserID, friendRequest.
 }

 // Check if the friend request has already been handled.
 if fr.HandleResult != 0 {
  return fmt.Errorf("friend request from %s to %s has already been processed", friendRequest.FromUserID, friendRequ
 }

 // Log the action of refusing the friend request for debugging and auditing purposes.
 log.ZDebug(ctx, "Refusing friend request", map[string]interface{}{
  "DB_FriendRequest":  fr,
  "Arg_FriendRequest": friendRequest,
 })

 // Mark the friend request as refused and update the handle time.
 friendRequest.HandleResult = constant.FriendResponseRefuse
 friendRequest.HandleTime = time.Now()
 if err := f.friendRequest.Update(ctx, friendRequest); err != nil {
  return fmt.Errorf("failed to update friend request from %s to %s as refused: %w", friendRequest.FromUserID, frien
 }

 return nil
}

// AgreeFriendRequest accepts a friend request. It first checks for an existing, unprocessed request.
func (f *friendDatabase) AgreeFriendRequest(ctx context.Context, friendRequest *model.FriendRequest) (err error) {
 return f.tx.Transaction(ctx, func(ctx context.Context) error {
  now := time.Now()
  fr, err := f.friendRequest.Take(ctx, friendRequest.FromUserID, friendRequest.ToUserID)
  if err != nil {
    return err
  }
  if fr.HandleResult != 0 {
    return errs.ErrArgs.WrapMsg("the friend request has been processed")
  }
  friendRequest.HandlerUserID = mcontext.GetOpUserID(ctx)
  friendRequest.HandleResult = constant.FriendResponseAgree
  friendRequest.HandleTime = now
  err = f.friendRequest.Update(ctx, friendRequest)
  if err != nil {
    return err
  }

  fr2, err := f.friendRequest.Take(ctx, friendRequest.ToUserID, friendRequest.FromUserID)
  if err == nil && fr2.HandleResult == constant.FriendResponseNotHandle {
    fr2.HandlerUserID = mcontext.GetOpUserID(ctx)
```

```
    fr2.HandleResult = constant.FriendResponseAgree
    fr2.HandleTime = now
    err = f.friendRequest.Update(ctx, fr2)
    if err != nil {
        return err
    }
  } else if err != nil && (!mgo.IsNotFound(err)) {
    return err
  }

  exists, err := f.friend.FindUserState(ctx, friendRequest.FromUserID, friendRequest.ToUserID)
  if err != nil {
    return err
  }
  existsMap := datautil.SliceSet(datautil.Slice(exists, func(friend *model.Friend) [2]string {
    return [...]string{friend.OwnerUserID, friend.FriendUserID} // My - Friend
  }))
  var adds []*model.Friend
  if _, ok := existsMap[[...]string{friendRequest.ToUserID, friendRequest.FromUserID}]; !ok { // My - Friend
    adds = append(
      adds,
      &model.Friend{
        OwnerUserID:    friendRequest.ToUserID,
        FriendUserID:   friendRequest.FromUserID,
        AddSource:      int32(constant.BecomeFriendByApply),
        OperatorUserID: friendRequest.FromUserID,
      },
    )
  }
  if _, ok := existsMap[[...]string{friendRequest.FromUserID, friendRequest.ToUserID}]; !ok { // My - Friend
    adds = append(
      adds,
      &model.Friend{
        OwnerUserID:    friendRequest.FromUserID,
        FriendUserID:   friendRequest.ToUserID,
        AddSource:      int32(constant.BecomeFriendByApply),
        OperatorUserID: friendRequest.FromUserID,
      },
    )
  }
  if len(adds) > 0 {
    if err := f.friend.Create(ctx, adds); err != nil {
      return err
    }
  }
  return f.cache.DelFriendIDs(friendRequest.ToUserID, friendRequest.FromUserID).DelMaxFriendVersion(friendRequest.T
 })
}

// Delete removes a friend relationship. It is assumed that the external caller has verified the friendship status.
func (f *friendDatabase) Delete(ctx context.Context, ownerUserID string, friendUserIDs []string) (err error) {
 if err := f.friend.Delete(ctx, ownerUserID, friendUserIDs); err != nil {
  return err
 }
 userIds := append(friendUserIDs, ownerUserID)
 return f.cache.DelFriendIDs(userIds...).DelMaxFriendVersion(userIds...).ChainExecDel(ctx)
}

// UpdateRemark updates the remark for a friend. Zero value for remark is also supported.
func (f *friendDatabase) UpdateRemark(ctx context.Context, ownerUserID, friendUserID, remark string) (err error) {
 if err := f.friend.UpdateRemark(ctx, ownerUserID, friendUserID, remark); err != nil {
  return err
 }
 return f.cache.DelFriend(ownerUserID, friendUserID).DelMaxFriendVersion(ownerUserID).ChainExecDel(ctx)
}
```

```go
// PageOwnerFriends retrieves the list of friends for the ownerUserID. It does not return an error if the result is
func (f *friendDatabase) PageOwnerFriends(ctx context.Context, ownerUserID string, pagination pagination.Pagination)
    return f.friend.FindOwnerFriends(ctx, ownerUserID, pagination)
}

// PageInWhoseFriends identifies in whose friend lists the friendUserID appears.
func (f *friendDatabase) PageInWhoseFriends(ctx context.Context, friendUserID string, pagination pagination.Paginati
    return f.friend.FindInWhoseFriends(ctx, friendUserID, pagination)
}

// PageFriendRequestFromMe retrieves friend requests sent by me. It does not return an error if the result is empty.
func (f *friendDatabase) PageFriendRequestFromMe(ctx context.Context, userID string, handleResults []int, pagination
    return f.friendRequest.FindFromUserID(ctx, userID, handleResults, pagination)
}

// PageFriendRequestToMe retrieves friend requests received by me. It does not return an error if the result is empt
func (f *friendDatabase) PageFriendRequestToMe(ctx context.Context, userID string, handleResults []int, pagination p
    return f.friendRequest.FindToUserID(ctx, userID, handleResults, pagination)
}

// FindFriendsWithError retrieves specified friends' information for ownerUserID. Returns an error if any friend doe
func (f *friendDatabase) FindFriendsWithError(ctx context.Context, ownerUserID string, friendUserIDs []string) (frie
    friends, err = f.friend.FindFriends(ctx, ownerUserID, friendUserIDs)
    if err != nil {
        return
    }
    return
}

func (f *friendDatabase) FindFriendUserIDs(ctx context.Context, ownerUserID string) (friendUserIDs []string, err err
    return f.cache.GetFriendIDs(ctx, ownerUserID)
}

func (f *friendDatabase) FindBothFriendRequests(ctx context.Context, fromUserID, toUserID string) (friends []*model.
    return f.friendRequest.FindBothFriendRequests(ctx, fromUserID, toUserID)
}
func (f *friendDatabase) UpdateFriends(ctx context.Context, ownerUserID string, friendUserIDs []string, val map[stri
    if len(val) == 0 {
        return nil
    }
    return f.tx.Transaction(ctx, func(ctx context.Context) error {
        if err := f.friend.UpdateFriends(ctx, ownerUserID, friendUserIDs, val); err != nil {
            return err
        }
        return f.cache.DelFriends(ownerUserID, friendUserIDs).DelMaxFriendVersion(ownerUserID).ChainExecDel(ctx)
    })
}

//func (f *friendDatabase) FindSortFriendUserIDs(ctx context.Context, ownerUserID string) ([]string, error) {
//    return f.cache.FindSortFriendUserIDs(ctx, ownerUserID)
//}

func (f *friendDatabase) FindFriendIncrVersion(ctx context.Context, ownerUserID string, version uint, limit int) (*m
    return f.friend.FindIncrVersion(ctx, ownerUserID, version, limit)
}

func (f *friendDatabase) FindMaxFriendVersionCache(ctx context.Context, ownerUserID string) (*model.VersionLog, erro
    return f.cache.FindMaxFriendVersion(ctx, ownerUserID)
}

func (f *friendDatabase) FindFriendUserID(ctx context.Context, friendUserID string) ([]string, error) {
    return f.friend.FindFriendUserID(ctx, friendUserID)
}

//func (f *friendDatabase) SearchFriend(ctx context.Context, ownerUserID, keyword string, pagination pagination.Pagi
//    return f.friend.SearchFriend(ctx, ownerUserID, keyword, pagination)
```

```
//}

func (f *friendDatabase) OwnerIncrVersion(ctx context.Context, ownerUserID string, friendUserIDs []string, state int
	if err := f.friend.IncrVersion(ctx, ownerUserID, friendUserIDs, state); err != nil {
		return err
	}
	return f.cache.DelMaxFriendVersion(ownerUserID).ChainExecDel(ctx)
}

func (f *friendDatabase) GetUnhandledCount(ctx context.Context, userID string, ts int64) (int64, error) {
	return f.friendRequest.GetUnhandledCount(ctx, userID, ts)
}
```

pkg/common/storage/controller/group.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	redis2 "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/common"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/db/tx"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
)

type GroupDatabase interface {
	// CreateGroup creates new groups along with their members.
	CreateGroup(ctx context.Context, groups []*model.Group, groupMembers []*model.GroupMember) error
	// TakeGroup retrieves a single group by its ID.
	TakeGroup(ctx context.Context, groupID string) (group *model.Group, err error)
	// FindGroup retrieves multiple groups by their IDs.
	FindGroup(ctx context.Context, groupIDs []string) (groups []*model.Group, err error)
	// SearchGroup searches for groups based on a keyword and pagination settings, returns total count and groups.
	SearchGroup(ctx context.Context, keyword string, pagination pagination.Pagination) (int64, []*model.Group, error)
	// UpdateGroup updates the properties of a group identified by its ID.
	UpdateGroup(ctx context.Context, groupID string, data map[string]any) error
	// DismissGroup disbands a group and optionally removes its members based on the deleteMember flag.
	DismissGroup(ctx context.Context, groupID string, deleteMember bool) error

	// TakeGroupMember retrieves a specific group member by group ID and user ID.
	TakeGroupMember(ctx context.Context, groupID string, userID string) (groupMember *model.GroupMember, err error)
	// TakeGroupOwner retrieves the owner of a group by group ID.
	TakeGroupOwner(ctx context.Context, groupID string) (*model.GroupMember, error)
	// FindGroupMembers retrieves members of a group filtered by user IDs.
	FindGroupMembers(ctx context.Context, groupID string, userIDs []string) (groupMembers []*model.GroupMember, err er
	// FindGroupMemberUser retrieves groups that a user is a member of, filtered by group IDs.
	FindGroupMemberUser(ctx context.Context, groupIDs []string, userID string) (groupMembers []*model.GroupMember, err
	// FindGroupMemberRoleLevels retrieves group members filtered by their role levels within a group.
	FindGroupMemberRoleLevels(ctx context.Context, groupID string, roleLevels []int32) (groupMembers []*model.GroupMemb
	// FindGroupMemberAll retrieves all members of a group.
	FindGroupMemberAll(ctx context.Context, groupID string) (groupMembers []*model.GroupMember, err error)
	// FindGroupsOwner retrieves the owners for multiple groups.
	FindGroupsOwner(ctx context.Context, groupIDs []string) ([]*model.GroupMember, error)
	// FindGroupMemberUserID retrieves the user IDs of all members in a group.
```

```
    FindGroupMemberUserID(ctx context.Context, groupID string) ([]string, error)
    // FindGroupMemberNum retrieves the number of members in a group.
    FindGroupMemberNum(ctx context.Context, groupID string) (uint32, error)
    // FindUserManagedGroupID retrieves group IDs managed by a user.
    FindUserManagedGroupID(ctx context.Context, userID string) (groupIDs []string, err error)
    // PageGroupRequest paginates through group requests for specified groups.
    PageGroupRequest(ctx context.Context, groupIDs []string, handleResults []int, pagination pagination.Pagination) (in
    // GetGroupRoleLevelMemberIDs retrieves user IDs of group members with a specific role level.
    GetGroupRoleLevelMemberIDs(ctx context.Context, groupID string, roleLevel int32) ([]string, error)

    // PageGetJoinGroup paginates through groups that a user has joined.
    PageGetJoinGroup(ctx context.Context, userID string, pagination pagination.Pagination) (total int64, totalGroupMemb
    // PageGetGroupMember paginates through members of a group.
    PageGetGroupMember(ctx context.Context, groupID string, pagination pagination.Pagination) (total int64, totalGroupM
    // SearchGroupMember searches for group members based on a keyword, group ID, and pagination settings.
    SearchGroupMember(ctx context.Context, keyword string, groupID string, pagination pagination.Pagination) (int64, []
    // HandlerGroupRequest processes a group join request with a specified result.
    HandlerGroupRequest(ctx context.Context, groupID string, userID string, handledMsg string, handleResult int32, memb
    // DeleteGroupMember removes specified users from a group.
    DeleteGroupMember(ctx context.Context, groupID string, userIDs []string) error
    // MapGroupMemberUserID maps group IDs to their members' simplified user IDs.
    MapGroupMemberUserID(ctx context.Context, groupIDs []string) (map[string]*common.GroupSimpleUserID, error)
    // MapGroupMemberNum maps group IDs to their member count.
    MapGroupMemberNum(ctx context.Context, groupIDs []string) (map[string]uint32, error)
    // TransferGroupOwner transfers the ownership of a group to another user.
    TransferGroupOwner(ctx context.Context, groupID string, oldOwnerUserID, newOwnerUserID string, roleLevel int32) err
    // UpdateGroupMember updates properties of a group member.
    UpdateGroupMember(ctx context.Context, groupID string, userID string, data map[string]any) error
    // UpdateGroupMembers batch updates properties of group members.
    UpdateGroupMembers(ctx context.Context, data []*common.BatchUpdateGroupMember) error

    // CreateGroupRequest creates new group join requests.
    CreateGroupRequest(ctx context.Context, requests []*model.GroupRequest) error
    // TakeGroupRequest retrieves a specific group join request.
    TakeGroupRequest(ctx context.Context, groupID string, userID string) (*model.GroupRequest, error)
    // FindGroupRequests retrieves multiple group join requests.
    FindGroupRequests(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupRequest, error)
    // PageGroupRequestUser paginates through group join requests made by a user.
    PageGroupRequestUser(ctx context.Context, userID string, groupIDs []string, handleResults []int, pagination paginat

    // CountTotal counts the total number of groups as of a certain date.
    CountTotal(ctx context.Context, before *time.Time) (count int64, err error)
    // CountRangeEverydayTotal counts the daily group creation total within a specified date range.
    CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int64, error)
    // DeleteGroupMemberHash deletes the hash entries for group members in specified groups.
    DeleteGroupMemberHash(ctx context.Context, groupIDs []string) error

    FindMemberIncrVersion(ctx context.Context, groupID string, version uint, limit int) (*model.VersionLog, error)
    BatchFindMemberIncrVersion(ctx context.Context, groupIDs []string, versions []uint64, limits []int) (map[string]*mo
    FindJoinIncrVersion(ctx context.Context, userID string, version uint, limit int) (*model.VersionLog, error)
    MemberGroupIncrVersion(ctx context.Context, groupID string, userIDs []string, state int32) error

    //FindSortGroupMemberUserIDs(ctx context.Context, groupID string) ([]string, error)
    //FindSortJoinGroupIDs(ctx context.Context, userID string) ([]string, error)

    FindMaxGroupMemberVersionCache(ctx context.Context, groupID string) (*model.VersionLog, error)
    BatchFindMaxGroupMemberVersionCache(ctx context.Context, groupIDs []string) (map[string]*model.VersionLog, error)
    FindMaxJoinGroupVersionCache(ctx context.Context, userID string) (*model.VersionLog, error)

    SearchJoinGroup(ctx context.Context, userID string, keyword string, pagination pagination.Pagination) (int64, []*mo

    FindJoinGroupID(ctx context.Context, userID string) ([]string, error)

    GetGroupApplicationUnhandledCount(ctx context.Context, groupIDs []string, ts int64) (int64, error)
}
```

```
func NewGroupDatabase(
	rdb redis.UniversalClient,
	localCache *config.LocalCache,
	groupDB database.Group,
	groupMemberDB database.GroupMember,
	groupRequestDB database.GroupRequest,
	ctxTx tx.Tx,
	groupHash cache.GroupHash,
) GroupDatabase {
	return &groupDatabase{
		groupDB:        groupDB,
		groupMemberDB:  groupMemberDB,
		groupRequestDB: groupRequestDB,
		ctxTx:          ctxTx,
		cache:          redis2.NewGroupCacheRedis(rdb, localCache, groupDB, groupMemberDB, groupRequestDB, groupHash),
	}
}

type groupDatabase struct {
	groupDB        database.Group
	groupMemberDB  database.GroupMember
	groupRequestDB database.GroupRequest
	ctxTx          tx.Tx
	cache          cache.GroupCache
}

func (g *groupDatabase) FindJoinGroupID(ctx context.Context, userID string) ([]string, error) {
	return g.cache.GetJoinedGroupIDs(ctx, userID)
}

func (g *groupDatabase) FindGroupMembers(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupMembe
	return g.cache.GetGroupMembersInfo(ctx, groupID, userIDs)
}

func (g *groupDatabase) FindGroupMemberUser(ctx context.Context, groupIDs []string, userID string) ([]*model.GroupMe
	return g.cache.FindGroupMemberUser(ctx, groupIDs, userID)
}

func (g *groupDatabase) FindGroupMemberRoleLevels(ctx context.Context, groupID string, roleLevels []int32) ([]*model
	return g.cache.GetGroupRolesLevelMemberInfo(ctx, groupID, roleLevels)
}

func (g *groupDatabase) FindGroupMemberAll(ctx context.Context, groupID string) ([]*model.GroupMember, error) {
	return g.cache.GetAllGroupMembersInfo(ctx, groupID)
}

func (g *groupDatabase) FindGroupsOwner(ctx context.Context, groupIDs []string) ([]*model.GroupMember, error) {
	return g.cache.GetGroupsOwner(ctx, groupIDs)
}

func (g *groupDatabase) GetGroupRoleLevelMemberIDs(ctx context.Context, groupID string, roleLevel int32) ([]string,
	return g.cache.GetGroupRoleLevelMemberIDs(ctx, groupID, roleLevel)
}

func (g *groupDatabase) CreateGroup(ctx context.Context, groups []*model.Group, groupMembers []*model.GroupMember) e
	if len(groups)+len(groupMembers) == 0 {
		return nil
	}
	return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
		c := g.cache.CloneGroupCache()
		if len(groups) > 0 {
			if err := g.groupDB.Create(ctx, groups); err != nil {
				return err
			}
			for _, group := range groups {
				c = c.DelGroupsInfo(group.GroupID).
```

```
        DelGroupMembersHash(group.GroupID).
        DelGroupsMemberNum(group.GroupID).
        DelGroupMemberIDs(group.GroupID).
        DelGroupAllRoleLevel(group.GroupID).
        DelMaxGroupMemberVersion(group.GroupID)
    }
  }
  if len(groupMembers) > 0 {
    if err := g.groupMemberDB.Create(ctx, groupMembers); err != nil {
      return err
    }
    for _, groupMember := range groupMembers {
      c = c.DelGroupMembersHash(groupMember.GroupID).
        DelGroupsMemberNum(groupMember.GroupID).
        DelGroupMemberIDs(groupMember.GroupID).
        DelJoinedGroupID(groupMember.UserID).
        DelGroupMembersInfo(groupMember.GroupID, groupMember.UserID).
        DelGroupAllRoleLevel(groupMember.GroupID).
        DelMaxJoinGroupVersion(groupMember.UserID).
        DelMaxGroupMemberVersion(groupMember.GroupID)
    }
  }
  return c.ChainExecDel(ctx)
 })
}

func (g *groupDatabase) FindGroupMemberUserID(ctx context.Context, groupID string) ([]string, error) {
 return g.cache.GetGroupMemberIDs(ctx, groupID)
}

func (g *groupDatabase) FindGroupMemberNum(ctx context.Context, groupID string) (uint32, error) {
 num, err := g.cache.GetGroupMemberNum(ctx, groupID)
 if err != nil {
  return 0, err
 }
 return uint32(num), nil
}

func (g *groupDatabase) TakeGroup(ctx context.Context, groupID string) (*model.Group, error) {
 return g.cache.GetGroupInfo(ctx, groupID)
}

func (g *groupDatabase) FindGroup(ctx context.Context, groupIDs []string) ([]*model.Group, error) {
 return g.cache.GetGroupsInfo(ctx, groupIDs)
}

func (g *groupDatabase) SearchGroup(ctx context.Context, keyword string, pagination pagination.Pagination) (int64, [
 return g.groupDB.Search(ctx, keyword, pagination)
}

func (g *groupDatabase) UpdateGroup(ctx context.Context, groupID string, data map[string]any) error {
 return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
  if err := g.groupDB.UpdateMap(ctx, groupID, data); err != nil {
   return err
  }
  if err := g.groupMemberDB.MemberGroupIncrVersion(ctx, groupID, []string{""}, model.VersionStateUpdate); err != ni
   return err
  }
  return g.cache.CloneGroupCache().DelGroupsInfo(groupID).DelMaxGroupMemberVersion(groupID).ChainExecDel(ctx)
 })
}

func (g *groupDatabase) DismissGroup(ctx context.Context, groupID string, deleteMember bool) error {
 return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
  c := g.cache.CloneGroupCache()
  if err := g.groupDB.UpdateStatus(ctx, groupID, constant.GroupStatusDismissed); err != nil {
```

```
        return err
    }
    if deleteMember {
        userIDs, err := g.cache.GetGroupMemberIDs(ctx, groupID)
        if err != nil {
            return err
        }
        if err := g.groupMemberDB.Delete(ctx, groupID, nil); err != nil {
            return err
        }
        c = c.DelJoinedGroupID(userIDs...).
            DelGroupMemberIDs(groupID).
            DelGroupsMemberNum(groupID).
            DelGroupMembersHash(groupID).
            DelGroupAllRoleLevel(groupID).
            DelGroupMembersInfo(groupID, userIDs...).
            DelMaxGroupMemberVersion(groupID).
            DelMaxJoinGroupVersion(userIDs...)
        for _, userID := range userIDs {
            if err := g.groupMemberDB.JoinGroupIncrVersion(ctx, userID, []string{groupID}, model.VersionStateDelete); err !
                return err
            }
        }
    } else {
        if err := g.groupMemberDB.MemberGroupIncrVersion(ctx, groupID, []string{""}, model.VersionStateUpdate); err != n
            return err
        }
        c = c.DelMaxGroupMemberVersion(groupID)
    }
    return c.DelGroupsInfo(groupID).ChainExecDel(ctx)
})
}

func (g *groupDatabase) TakeGroupMember(ctx context.Context, groupID string, userID string) (*model.GroupMember, err
    return g.cache.GetGroupMemberInfo(ctx, groupID, userID)
}

func (g *groupDatabase) TakeGroupOwner(ctx context.Context, groupID string) (*model.GroupMember, error) {
    return g.cache.GetGroupOwner(ctx, groupID)
}

func (g *groupDatabase) FindUserManagedGroupID(ctx context.Context, userID string) (groupIDs []string, err error) {
    return g.groupMemberDB.FindUserManagedGroupID(ctx, userID)
}

func (g *groupDatabase) PageGroupRequest(ctx context.Context, groupIDs []string, handleResults []int, pagination pag
    return g.groupRequestDB.PageGroup(ctx, groupIDs, handleResults, pagination)
}

func (g *groupDatabase) PageGetJoinGroup(ctx context.Context, userID string, pagination pagination.Pagination) (tota
    groupIDs, err := g.cache.GetJoinedGroupIDs(ctx, userID)
    if err != nil {
        return 0, nil, err
    }
    for _, groupID := range datautil.Paginate(groupIDs, int(pagination.GetPageNumber()), int(pagination.GetShowNumber()
        groupMembers, err := g.cache.GetGroupMembersInfo(ctx, groupID, []string{userID})
        if err != nil {
            return 0, nil, err
        }
        totalGroupMembers = append(totalGroupMembers, groupMembers...)
    }
    return int64(len(groupIDs)), totalGroupMembers, nil
}

func (g *groupDatabase) PageGetGroupMember(ctx context.Context, groupID string, pagination pagination.Pagination) (t
    groupMemberIDs, err := g.cache.GetGroupMemberIDs(ctx, groupID)
```

```
	if err != nil {
		return 0, nil, err
	}
	pageIDs := datautil.Paginate(groupMemberIDs, int(pagination.GetPageNumber()), int(pagination.GetShowNumber()))
	if len(pageIDs) == 0 {
		return int64(len(groupMemberIDs)), nil, nil
	}
	members, err := g.cache.GetGroupMembersInfo(ctx, groupID, pageIDs)
	if err != nil {
		return 0, nil, err
	}
	return int64(len(groupMemberIDs)), members, nil
}

func (g *groupDatabase) SearchGroupMember(ctx context.Context, keyword string, groupID string, pagination pagination
	return g.groupMemberDB.SearchMember(ctx, keyword, groupID, pagination)
}

func (g *groupDatabase) HandlerGroupRequest(ctx context.Context, groupID string, userID string, handledMsg string, h
	return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
		if err := g.groupRequestDB.UpdateHandler(ctx, groupID, userID, handledMsg, handleResult); err != nil {
			return err
		}
		if member != nil {
			c := g.cache.CloneGroupCache()
			if err := g.groupMemberDB.Create(ctx, []*model.GroupMember{member}); err != nil {
				return err
			}
			c = c.DelGroupMembersHash(groupID).
				DelGroupMembersInfo(groupID, member.UserID).
				DelGroupMemberIDs(groupID).
				DelGroupsMemberNum(groupID).
				DelJoinedGroupID(member.UserID).
				DelGroupRoleLevel(groupID, []int32{member.RoleLevel}).
				DelMaxJoinGroupVersion(userID).
				DelMaxGroupMemberVersion(groupID)
			if err := c.ChainExecDel(ctx); err != nil {
				return err
			}
		}
		return nil
	})
}

func (g *groupDatabase) DeleteGroupMember(ctx context.Context, groupID string, userIDs []string) error {
	return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
		if err := g.groupMemberDB.Delete(ctx, groupID, userIDs); err != nil {
			return err
		}
		c := g.cache.CloneGroupCache()
		return c.DelGroupMembersHash(groupID).
			DelGroupMemberIDs(groupID).
			DelGroupsMemberNum(groupID).
			DelJoinedGroupID(userIDs...).
			DelGroupMembersInfo(groupID, userIDs...).
			DelGroupAllRoleLevel(groupID).
			DelMaxGroupMemberVersion(groupID).
			DelMaxJoinGroupVersion(userIDs...).
			ChainExecDel(ctx)
	})
}

func (g *groupDatabase) MapGroupMemberUserID(ctx context.Context, groupIDs []string) (map[string]*common.GroupSimple
	return g.cache.GetGroupMemberHashMap(ctx, groupIDs)
}
```

```go
func (g *groupDatabase) MapGroupMemberNum(ctx context.Context, groupIDs []string) (m map[string]uint32, err error) {
    m = make(map[string]uint32)
    for _, groupID := range groupIDs {
        num, err := g.cache.GetGroupMemberNum(ctx, groupID)
        if err != nil {
            return nil, err
        }
        m[groupID] = uint32(num)
    }
    return m, nil
}

func (g *groupDatabase) TransferGroupOwner(ctx context.Context, groupID string, oldOwnerUserID, newOwnerUserID strin
    return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
        if err := g.groupMemberDB.UpdateUserRoleLevels(ctx, groupID, oldOwnerUserID, roleLevel, newOwnerUserID, constant.
            return err
        }
        c := g.cache.CloneGroupCache()
        return c.DelGroupMembersInfo(groupID, oldOwnerUserID, newOwnerUserID).
            DelGroupAllRoleLevel(groupID).
            DelGroupMembersHash(groupID).
            DelMaxGroupMemberVersion(groupID).
            DelGroupMemberIDs(groupID).
            ChainExecDel(ctx)
    })
}

func (g *groupDatabase) UpdateGroupMember(ctx context.Context, groupID string, userID string, data map[string]any) e
    if len(data) == 0 {
        return nil
    }
    return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
        if err := g.groupMemberDB.Update(ctx, groupID, userID, data); err != nil {
            return err
        }
        c := g.cache.CloneGroupCache()
        c = c.DelGroupMembersInfo(groupID, userID)
        if g.groupMemberDB.IsUpdateRoleLevel(data) {
            c = c.DelGroupAllRoleLevel(groupID).DelGroupMemberIDs(groupID)
        }
        c = c.DelMaxGroupMemberVersion(groupID)
        return c.ChainExecDel(ctx)
    })
}

func (g *groupDatabase) UpdateGroupMembers(ctx context.Context, data []*common.BatchUpdateGroupMember) error {
    return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
        c := g.cache.CloneGroupCache()
        for _, item := range data {
            if err := g.groupMemberDB.Update(ctx, item.GroupID, item.UserID, item.Map); err != nil {
                return err
            }
            if g.groupMemberDB.IsUpdateRoleLevel(item.Map) {
                c = c.DelGroupAllRoleLevel(item.GroupID).DelGroupMemberIDs(item.GroupID)
            }
            c = c.DelGroupMembersInfo(item.GroupID, item.UserID).DelMaxGroupMemberVersion(item.GroupID).DelGroupMembersHash(
        }
        return c.ChainExecDel(ctx)
    })
}

func (g *groupDatabase) CreateGroupRequest(ctx context.Context, requests []*model.GroupRequest) error {
    return g.ctxTx.Transaction(ctx, func(ctx context.Context) error {
        for _, request := range requests {
            if err := g.groupRequestDB.Delete(ctx, request.GroupID, request.UserID); err != nil {
                return err
```

```
		}
	}
	return g.groupRequestDB.Create(ctx, requests)
	})
}

func (g *groupDatabase) TakeGroupRequest(ctx context.Context, groupID string, userID string) (*model.GroupRequest, e
	return g.groupRequestDB.Take(ctx, groupID, userID)
}

func (g *groupDatabase) PageGroupRequestUser(ctx context.Context, userID string, groupIDs []string, handleResults []
	return g.groupRequestDB.Page(ctx, userID, groupIDs, handleResults, pagination)
}

func (g *groupDatabase) CountTotal(ctx context.Context, before *time.Time) (count int64, err error) {
	return g.groupDB.CountTotal(ctx, before)
}

func (g *groupDatabase) CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int
	return g.groupDB.CountRangeEverydayTotal(ctx, start, end)
}

func (g *groupDatabase) FindGroupRequests(ctx context.Context, groupID string, userIDs []string) ([]*model.GroupRequ
	return g.groupRequestDB.FindGroupRequests(ctx, groupID, userIDs)
}

func (g *groupDatabase) DeleteGroupMemberHash(ctx context.Context, groupIDs []string) error {
	if len(groupIDs) == 0 {
		return nil
	}
	c := g.cache.CloneGroupCache()
	for _, groupID := range groupIDs {
		c = c.DelGroupMembersHash(groupID)
	}
	return c.ChainExecDel(ctx)
}

func (g *groupDatabase) FindMemberIncrVersion(ctx context.Context, groupID string, version uint, limit int) (*model.
	return g.groupMemberDB.FindMemberIncrVersion(ctx, groupID, version, limit)
}

func (g *groupDatabase) BatchFindMemberIncrVersion(ctx context.Context, groupIDs []string, versions []uint64, limits
	if len(groupIDs) == 0 {
		return nil, errs.Wrap(errs.New("groupIDs is nil."))
	}

	// convert []uint64 to []uint
	var uintVersions []uint
	for _, version := range versions {
		uintVersions = append(uintVersions, uint(version))
	}

	versionLogs, err := g.groupMemberDB.BatchFindMemberIncrVersion(ctx, groupIDs, uintVersions, limits)
	if err != nil {
		return nil, errs.Wrap(err)
	}

	groupMemberIncrVersionsMap := datautil.SliceToMap(versionLogs, func(e *model.VersionLog) string {
		return e.DID
	})

	return groupMemberIncrVersionsMap, nil
}

func (g *groupDatabase) FindJoinIncrVersion(ctx context.Context, userID string, version uint, limit int) (*model.Ver
	return g.groupMemberDB.FindJoinIncrVersion(ctx, userID, version, limit)
```

```
}

func (g *groupDatabase) FindMaxGroupMemberVersionCache(ctx context.Context, groupID string) (*model.VersionLog, erro
    return g.cache.FindMaxGroupMemberVersion(ctx, groupID)
}

func (g *groupDatabase) BatchFindMaxGroupMemberVersionCache(ctx context.Context, groupIDs []string) (map[string]*mod
    if len(groupIDs) == 0 {
        return nil, errs.Wrap(errs.New("groupIDs is nil in Cache."))
    }
    versionLogs, err := g.cache.BatchFindMaxGroupMemberVersion(ctx, groupIDs)
    if err != nil {
        return nil, errs.Wrap(err)
    }
    maxGroupMemberVersionsMap := datautil.SliceToMap(versionLogs, func(e *model.VersionLog) string {
        return e.DID
    })
    return maxGroupMemberVersionsMap, nil
}

func (g *groupDatabase) FindMaxJoinGroupVersionCache(ctx context.Context, userID string) (*model.VersionLog, error)
    return g.cache.FindMaxJoinGroupVersion(ctx, userID)
}

func (g *groupDatabase) SearchJoinGroup(ctx context.Context, userID string, keyword string, pagination pagination.Pa
    groupIDs, err := g.cache.GetJoinedGroupIDs(ctx, userID)
    if err != nil {
        return 0, nil, err
    }
    return g.groupDB.SearchJoin(ctx, groupIDs, keyword, pagination)
}

func (g *groupDatabase) MemberGroupIncrVersion(ctx context.Context, groupID string, userIDs []string, state int32) e
    if err := g.groupMemberDB.MemberGroupIncrVersion(ctx, groupID, userIDs, state); err != nil {
        return err
    }
    return g.cache.DelMaxGroupMemberVersion(groupID).ChainExecDel(ctx)
}

func (g *groupDatabase) GetGroupApplicationUnhandledCount(ctx context.Context, groupIDs []string, ts int64) (int64,
    return g.groupRequestDB.GetUnhandledCount(ctx, groupIDs, ts)
}
```

pkg/common/storage/controller/msg.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
	"context"
	"encoding/json"
	"errors"

	"github.com/openimsdk/tools/mq"
	"github.com/openimsdk/tools/utils/jsonutil"
	"google.golang.org/protobuf/proto"

	"strconv"
	"strings"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/redis/go-redis/v9"
	"go.mongodb.org/mongo-driver/mongo"

	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/protocol/constant"
	pbmsg "github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
)

const (
	updateKeyMsg = iota
	updateKeyRevoke
)

// CommonMsgDatabase defines the interface for message database operations.
type CommonMsgDatabase interface {
	// RevokeMsg revokes a message in a conversation.
	RevokeMsg(ctx context.Context, conversationID string, seq int64, revoke *model.RevokeModel) error
	// MarkSingleChatMsgsAsRead marks messages as read for a single chat by sequence numbers.
	MarkSingleChatMsgsAsRead(ctx context.Context, userID string, conversationID string, seqs []int64) error
	// GetMsgBySeqsRange retrieves messages from MongoDB by a range of sequence numbers.
	GetMsgBySeqsRange(ctx context.Context, userID string, conversationID string, begin, end, num, userMaxSeq int64) (mi
	// GetMsgBySeqs retrieves messages for large groups from MongoDB by sequence numbers.
	GetMsgBySeqs(ctx context.Context, userID string, conversationID string, seqs []int64) (minSeq int64, maxSeq int64,

	GetMessagesBySeqWithBounds(ctx context.Context, userID string, conversationID string, seqs []int64, pullOrder sdkws
	// DeleteUserMsgsBySeqs allows a user to delete messages based on sequence numbers.
	DeleteUserMsgsBySeqs(ctx context.Context, userID string, conversationID string, seqs []int64) error
```

```
	// DeleteMsgsPhysicalBySeqs physically deletes messages by emptying them based on sequence numbers.
	DeleteMsgsPhysicalBySeqs(ctx context.Context, conversationID string, seqs []int64) error
	//SetMaxSeq(ctx context.Context, conversationID string, maxSeq int64) error
	GetMaxSeqs(ctx context.Context, conversationIDs []string) (map[string]int64, error)
	GetMaxSeq(ctx context.Context, conversationID string) (int64, error)
	SetMinSeqs(ctx context.Context, seqs map[string]int64) error
	SetMinSeq(ctx context.Context, conversationID string, seq int64) error

	SetUserConversationsMinSeqs(ctx context.Context, userID string, seqs map[string]int64) (err error)
	SetHasReadSeq(ctx context.Context, userID string, conversationID string, hasReadSeq int64) error
	GetHasReadSeqs(ctx context.Context, userID string, conversationIDs []string) (map[string]int64, error)
	GetHasReadSeq(ctx context.Context, userID string, conversationID string) (int64, error)
	UserSetHasReadSeqs(ctx context.Context, userID string, hasReadSeqs map[string]int64) error

	GetMaxSeqsWithTime(ctx context.Context, conversationIDs []string) (map[string]database.SeqTime, error)
	GetMaxSeqWithTime(ctx context.Context, conversationID string) (database.SeqTime, error)
	GetCacheMaxSeqWithTime(ctx context.Context, conversationIDs []string) (map[string]database.SeqTime, error)

	SetSendMsgStatus(ctx context.Context, id string, status int32) error
	GetSendMsgStatus(ctx context.Context, id string) (int32, error)
	SearchMessage(ctx context.Context, req *pbmsg.SearchMessageReq) (total int64, msgData []*pbmsg.SearchedMsgData, err
	FindOneByDocIDs(ctx context.Context, docIDs []string, seqs map[string]int64) (map[string]*sdkws.MsgData, error)

	// to mq
	MsgToMQ(ctx context.Context, key string, msg2mq *sdkws.MsgData) error

	RangeUserSendCount(ctx context.Context, start time.Time, end time.Time, group bool, ase bool, pageNumber int32, sho
	RangeGroupSendCount(ctx context.Context, start time.Time, end time.Time, ase bool, pageNumber int32, showNumber int

	GetRandBeforeMsg(ctx context.Context, ts int64, limit int) ([]*model.MsgDocModel, error)

	SetUserConversationsMaxSeq(ctx context.Context, conversationID string, userID string, seq int64) error
	SetUserConversationsMinSeq(ctx context.Context, conversationID string, userID string, seq int64) error

	DeleteDoc(ctx context.Context, docID string) error

	GetLastMessageSeqByTime(ctx context.Context, conversationID string, time int64) (int64, error)

	GetLastMessage(ctx context.Context, conversationIDS []string, userID string) (map[string]*sdkws.MsgData, error)
}

func NewCommonMsgDatabase(msgDocModel database.Msg, msg cache.MsgCache, seqUser cache.SeqUser, seqConversation cache
	return &commonMsgDatabase{
		msgDocDatabase:  msgDocModel,
		msgCache:        msg,
		seqUser:         seqUser,
		seqConversation: seqConversation,
		producer:        producer,
	}
}

type commonMsgDatabase struct {
	msgDocDatabase  database.Msg
	msgTable        model.MsgDocModel
	msgCache        cache.MsgCache
	seqConversation cache.SeqConversationCache
	seqUser         cache.SeqUser
	producer        mq.Producer
}

func (db *commonMsgDatabase) MsgToMQ(ctx context.Context, key string, msg2mq *sdkws.MsgData) error {
	data, err := proto.Marshal(msg2mq)
	if err != nil {
		return err
	}
	return db.producer.SendMessage(ctx, key, data)
```

```
}

func (db *commonMsgDatabase) batchInsertBlock(ctx context.Context, conversationID string, fields []any, key int8, fi
    if len(fields) == 0 {
        return nil
    }
    num := db.msgTable.GetSingleGocMsgNum()
    // num = 100
    for i, field := range fields { // Check the type of the field
        var ok bool
        switch key {
        case updateKeyMsg:
            var msg *model.MsgDataModel
            msg, ok = field.(*model.MsgDataModel)
            if msg != nil && msg.Seq != firstSeq+int64(i) {
                return errs.ErrInternalServer.WrapMsg("seq is invalid")
            }
        case updateKeyRevoke:
            _, ok = field.(*model.RevokeModel)
        default:
            return errs.ErrInternalServer.WrapMsg("key is invalid")
        }
        if !ok {
            return errs.ErrInternalServer.WrapMsg("field type is invalid")
        }
    }
    // Returns true if the document exists in the database, false if the document does not exist in the database
    updateMsgModel := func(seq int64, i int) (bool, error) {
        var (
            res *mongo.UpdateResult
            err error
        )
        docID := db.msgTable.GetDocID(conversationID, seq)
        index := db.msgTable.GetMsgIndex(seq)
        field := fields[i]
        switch key {
        case updateKeyMsg:
            res, err = db.msgDocDatabase.UpdateMsg(ctx, docID, index, "msg", field)
        case updateKeyRevoke:
            res, err = db.msgDocDatabase.UpdateMsg(ctx, docID, index, "revoke", field)
        }
        if err != nil {
            return false, err
        }
        return res.MatchedCount > 0, nil
    }
    tryUpdate := true
    for i := 0; i < len(fields); i++ {
        seq := firstSeq + int64(i) // Current sequence number
        if tryUpdate {
            matched, err := updateMsgModel(seq, i)
            if err != nil {
                return err
            }
            if matched {
                continue // The current data has been updated, skip the current data
            }
        }
        doc := model.MsgDocModel{
            DocID: db.msgTable.GetDocID(conversationID, seq),
            Msg:   make([]*model.MsgInfoModel, num),
        }
        var insert int // Inserted data number
        for j := i; j < len(fields); j++ {
            seq = firstSeq + int64(j)
            if db.msgTable.GetDocID(conversationID, seq) != doc.DocID {
```

```
        break
    }
    insert++
    switch key {
    case updateKeyMsg:
        doc.Msg[db.msgTable.GetMsgIndex(seq)] = &model.MsgInfoModel{
            Msg: fields[j].(*model.MsgDataModel),
        }
    case updateKeyRevoke:
        doc.Msg[db.msgTable.GetMsgIndex(seq)] = &model.MsgInfoModel{
            Revoke: fields[j].(*model.RevokeModel),
        }
    }
  }
  for i, msgInfo := range doc.Msg {
    if msgInfo == nil {
        msgInfo = &model.MsgInfoModel{}
        doc.Msg[i] = msgInfo
    }
    if msgInfo.DelList == nil {
        doc.Msg[i].DelList = []string{}
    }
  }
  if err := db.msgDocDatabase.Create(ctx, &doc); err != nil {
    if mongo.IsDuplicateKeyError(err) {
        i--              // already inserted
        tryUpdate = true // next block use update mode
        continue
    }
    return err
  }
  tryUpdate = false // The current block is inserted successfully, and the next block is inserted preferentially
  i += insert - 1   // Skip the inserted data
 }

 return nil
}

func (db *commonMsgDatabase) RevokeMsg(ctx context.Context, conversationID string, seq int64, revoke *model.RevokeMo
 if err := db.batchInsertBlock(ctx, conversationID, []any{revoke}, updateKeyRevoke, seq); err != nil {
  return err
 }
 return db.msgCache.DelMessageBySeqs(ctx, conversationID, []int64{seq})
}

func (db *commonMsgDatabase) MarkSingleChatMsgsAsRead(ctx context.Context, userID string, conversationID string, tot
 for docID, seqs := range db.msgTable.GetDocIDSeqsMap(conversationID, totalSeqs) {
  var indexes []int64
  for _, seq := range seqs {
    indexes = append(indexes, db.msgTable.GetMsgIndex(seq))
  }
  log.ZDebug(ctx, "MarkSingleChatMsgsAsRead", "userID", userID, "docID", docID, "indexes", indexes)
  if err := db.msgDocDatabase.MarkSingleChatMsgsAsRead(ctx, userID, docID, indexes); err != nil {
    log.ZError(ctx, "MarkSingleChatMsgsAsRead", err, "userID", userID, "docID", docID, "indexes", indexes)
    return err
  }
 }
 return db.msgCache.DelMessageBySeqs(ctx, conversationID, totalSeqs)
}

func (db *commonMsgDatabase) getMsgBySeqs(ctx context.Context, userID, conversationID string, seqs []int64) (totalMs
 return db.GetMessageBySeqs(ctx, conversationID, userID, seqs)
}

func (db *commonMsgDatabase) handlerDBMsg(ctx context.Context, cache map[int64][]*model.MsgInfoModel, userID, conver
 if msg == nil || msg.Msg == nil {
```

```
		return
	}
	if msg.IsRead {
		msg.Msg.IsRead = true
	}
	if msg.Msg.ContentType != constant.Quote {
		return
	}
	if msg.Msg.Content == "" {
		return
	}
	type MsgData struct {
		SendID           string                 `json:"sendID"`
		RecvID           string                 `json:"recvID"`
		GroupID          string                 `json:"groupID"`
		ClientMsgID      string                 `json:"clientMsgID"`
		ServerMsgID      string                 `json:"serverMsgID"`
		SenderPlatformID int32                  `json:"senderPlatformID"`
		SenderNickname   string                 `json:"senderNickname"`
		SenderFaceURL    string                 `json:"senderFaceURL"`
		SessionType      int32                  `json:"sessionType"`
		MsgFrom          int32                  `json:"msgFrom"`
		ContentType      int32                  `json:"contentType"`
		Content          string                 `json:"content"`
		Seq              int64                  `json:"seq"`
		SendTime         int64                  `json:"sendTime"`
		CreateTime       int64                  `json:"createTime"`
		Status           int32                  `json:"status"`
		IsRead           bool                   `json:"isRead"`
		Options          map[string]bool        `json:"options,omitempty"`
		OfflinePushInfo  *sdkws.OfflinePushInfo `json:"offlinePushInfo"`
		AtUserIDList     []string               `json:"atUserIDList"`
		AttachedInfo     string                 `json:"attachedInfo"`
		Ex               string                 `json:"ex"`
		KeyVersion       int32                  `json:"keyVersion"`
		DstUserIDs       []string               `json:"dstUserIDs"`
	}
	var quoteMsg struct {
		Text               string          `json:"text,omitempty"`
		QuoteMessage       *MsgData        `json:"quoteMessage,omitempty"`
		MessageEntityList json.RawMessage `json:"messageEntityList,omitempty"`
	}
	if err := json.Unmarshal([]byte(msg.Msg.Content), &quoteMsg); err != nil {
		log.ZError(ctx, "json.Unmarshal", err)
		return
	}
	if quoteMsg.QuoteMessage == nil {
		return
	}
	if quoteMsg.QuoteMessage.Content == "e30=" {
		quoteMsg.QuoteMessage.Content = "{}"
		data, err := json.Marshal(&quoteMsg)
		if err != nil {
			return
		}
		msg.Msg.Content = string(data)
	}
	if quoteMsg.QuoteMessage.Seq <= 0 && quoteMsg.QuoteMessage.ContentType == constant.MsgRevokeNotification {
		return
	}
	var msgs []*model.MsgInfoModel
	if v, ok := cache[quoteMsg.QuoteMessage.Seq]; ok {
		msgs = v
	} else {
		if quoteMsg.QuoteMessage.Seq > 0 {
			ms, err := db.msgDocDatabase.GetMsgBySeqIndexIn1Doc(ctx, userID, db.msgTable.GetDocID(conversationID, quoteMsg.Q
```

```
        if err != nil {
            log.ZError(ctx, "GetMsgBySeqIndexIn1Doc", err, "conversationID", conversationID, "seq", quoteMsg.QuoteMessage.S
            return
        }
        msgs = ms
        cache[quoteMsg.QuoteMessage.Seq] = ms
    }
}
if len(msgs) != 0 && msgs[0].Msg.ContentType != constant.MsgRevokeNotification {
    return
}
quoteMsg.QuoteMessage.ContentType = constant.MsgRevokeNotification
if len(msgs) > 0 {
    quoteMsg.QuoteMessage.Content = msgs[0].Msg.Content
} else {
    quoteMsg.QuoteMessage.Content = "{}"
}
data, err := json.Marshal(&quoteMsg)
if err != nil {
    log.ZError(ctx, "json.Marshal", err)
    return
}
msg.Msg.Content = string(data)
}

func (db *commonMsgDatabase) findMsgInfoBySeq(ctx context.Context, userID, docID string, conversationID string, seqs
msgs, err := db.msgDocDatabase.GetMsgBySeqIndexIn1Doc(ctx, userID, docID, seqs)
if err != nil {
    return nil, err
}
tempCache := make(map[int64][]*model.MsgInfoModel)
for _, msg := range msgs {
    db.handlerDBMsg(ctx, tempCache, userID, conversationID, msg)
}
return msgs, err
}

// GetMsgBySeqsRange In the context of group chat, we have the following parameters:
//
// "maxSeq" of a conversation: It represents the maximum value of messages in the group conversation.
// "minSeq" of a conversation (default: 1): It represents the minimum value of messages in the group conversation.
//
// For a user's perspective regarding the group conversation, we have the following parameters:
//
// "userMaxSeq": It represents the user's upper limit for message retrieval in the group. If not set (default: 0),
// it means the upper limit is the same as the conversation's "maxSeq".
// "userMinSeq": It represents the user's starting point for message retrieval in the group. If not set (default: 0)
// it means the starting point is the same as the conversation's "minSeq".
//
// The scenarios for these parameters are as follows:
//
// For users who have been kicked out of the group, "userMaxSeq" can be set as the maximum value they had before
// being kicked out. This limits their ability to retrieve messages up to a certain point.
// For new users joining the group, if they don't need to receive old messages,
// "userMinSeq" can be set as the same value as the conversation's "maxSeq" at the moment they join the group.
// This ensures that their message retrieval starts from the point they joined.
func (db *commonMsgDatabase) GetMsgBySeqsRange(ctx context.Context, userID string, conversationID string, begin, end
userMinSeq, err := db.seqUser.GetUserMinSeq(ctx, conversationID, userID)
if err != nil && !errors.Is(err, redis.Nil) {
    return 0, 0, nil, err
}
minSeq, err := db.seqConversation.GetMinSeq(ctx, conversationID)
if err != nil {
    return 0, 0, nil, err
}
if userMinSeq > minSeq {
```

```
    minSeq = userMinSeq
  }
  // "minSeq" represents the startSeq value that the user can retrieve.
  if minSeq > end {
    log.ZWarn(ctx, "minSeq > end", errs.New("minSeq>end"), "minSeq", minSeq, "end", end)
    return 0, 0, nil, nil
  }
  maxSeq, err := db.seqConversation.GetMaxSeq(ctx, conversationID)
  if err != nil {
    return 0, 0, nil, err
  }
  log.ZDebug(ctx, "GetMsgBySeqsRange", "userMinSeq", userMinSeq, "conMinSeq", minSeq, "conMaxSeq", maxSeq, "userMaxSe
  if userMaxSeq != 0 {
    if userMaxSeq < maxSeq {
      maxSeq = userMaxSeq
    }
  }
  // "maxSeq" represents the endSeq value that the user can retrieve.

  if begin < minSeq {
    begin = minSeq
  }
  if end > maxSeq {
    end = maxSeq
  }
  // "begin" and "end" represent the actual startSeq and endSeq values that the user can retrieve.
  if end < begin {
    return 0, 0, nil, errs.ErrArgs.WrapMsg("seq end < begin")
  }
  var seqs []int64
  if end-begin+1 <= num {
    for i := begin; i <= end; i++ {
      seqs = append(seqs, i)
    }
  } else {
    for i := end - num + 1; i <= end; i++ {
      seqs = append(seqs, i)
    }
  }
  successMsgs, err := db.GetMessageBySeqs(ctx, conversationID, userID, seqs)
  if err != nil {
    return 0, 0, nil, err
  }
  return minSeq, maxSeq, successMsgs, nil
}

func (db *commonMsgDatabase) GetMsgBySeqs(ctx context.Context, userID string, conversationID string, seqs []int64) (
  userMinSeq, err := db.seqUser.GetUserMinSeq(ctx, conversationID, userID)
  if err != nil {
    return 0, 0, nil, err
  }
  minSeq, err := db.seqConversation.GetMinSeq(ctx, conversationID)
  if err != nil {
    return 0, 0, nil, err
  }
  maxSeq, err := db.seqConversation.GetMaxSeq(ctx, conversationID)
  if err != nil {
    return 0, 0, nil, err
  }
  userMaxSeq, err := db.seqUser.GetUserMaxSeq(ctx, conversationID, userID)
  if err != nil {
    return 0, 0, nil, err
  }
  if userMinSeq > minSeq {
    minSeq = userMinSeq
  }
```

```
    if userMaxSeq > 0 && userMaxSeq < maxSeq {
        maxSeq = userMaxSeq
    }
    newSeqs := make([]int64, 0, len(seqs))
    for _, seq := range seqs {
        if seq <= 0 {
            continue
        }
        if seq >= minSeq && seq <= maxSeq {
            newSeqs = append(newSeqs, seq)
        }
    }
    successMsgs, err := db.GetMessageBySeqs(ctx, conversationID, userID, newSeqs)
    if err != nil {
        return 0, 0, nil, err
    }
    return minSeq, maxSeq, successMsgs, nil
}

func (db *commonMsgDatabase) GetMessagesBySeqWithBounds(ctx context.Context, userID string, conversationID string, s
    var endSeq int64
    var isEnd bool
    userMinSeq, err := db.seqUser.GetUserMinSeq(ctx, conversationID, userID)
    if err != nil {
        return false, 0, nil, err
    }
    minSeq, err := db.seqConversation.GetMinSeq(ctx, conversationID)
    if err != nil {
        return false, 0, nil, err
    }
    maxSeq, err := db.seqConversation.GetMaxSeq(ctx, conversationID)
    if err != nil {
        return false, 0, nil, err
    }
    userMaxSeq, err := db.seqUser.GetUserMaxSeq(ctx, conversationID, userID)
    if err != nil {
        return false, 0, nil, err
    }
    if userMinSeq > minSeq {
        minSeq = userMinSeq
    }
    if userMaxSeq > 0 && userMaxSeq < maxSeq {
        maxSeq = userMaxSeq
    }
    newSeqs := make([]int64, 0, len(seqs))
    for _, seq := range seqs {
        if seq <= 0 {
            continue
        }
        // The normal range and can fetch messages
        if seq >= minSeq && seq <= maxSeq {
            newSeqs = append(newSeqs, seq)
            continue
        }
        // If the requested seq is smaller than the minimum seq and the pull order is descending (pulling older messages)
        if seq < minSeq && pullOrder == sdkws.PullOrder_PullOrderDesc {
            isEnd = true
            endSeq = minSeq
        }
        // If the requested seq is larger than the maximum seq and the pull order is ascending (pulling newer messages)
        if seq > maxSeq && pullOrder == sdkws.PullOrder_PullOrderAsc {
            isEnd = true
            endSeq = maxSeq
        }
    }
    if len(newSeqs) == 0 {
```

```
		return isEnd, endSeq, nil, nil
	}
	successMsgs, err := db.GetMessageBySeqs(ctx, conversationID, userID, newSeqs)
	if err != nil {
		return false, 0, nil, err
	}
	return isEnd, endSeq, successMsgs, nil
}

func (db *commonMsgDatabase) DeleteMsgsPhysicalBySeqs(ctx context.Context, conversationID string, allSeqs []int64) e
	for docID, seqs := range db.msgTable.GetDocIDSeqsMap(conversationID, allSeqs) {
		var indexes []int
		for _, seq := range seqs {
			indexes = append(indexes, int(db.msgTable.GetMsgIndex(seq)))
		}
		if err := db.msgDocDatabase.DeleteMsgsInOneDocByIndex(ctx, docID, indexes); err != nil {
			return err
		}
	}
	return db.msgCache.DelMessageBySeqs(ctx, conversationID, allSeqs)
}

func (db *commonMsgDatabase) DeleteUserMsgsBySeqs(ctx context.Context, userID string, conversationID string, seqs []
	for docID, seqs := range db.msgTable.GetDocIDSeqsMap(conversationID, seqs) {
		for _, seq := range seqs {
			if _, err := db.msgDocDatabase.PushUnique(ctx, docID, db.msgTable.GetMsgIndex(seq), "del_list", []string{userID}
				return err
			}
		}
	}
	return db.msgCache.DelMessageBySeqs(ctx, conversationID, seqs)
}

func (db *commonMsgDatabase) GetMaxSeqs(ctx context.Context, conversationIDs []string) (map[string]int64, error) {
	return db.seqConversation.GetMaxSeqs(ctx, conversationIDs)
}

func (db *commonMsgDatabase) GetMaxSeq(ctx context.Context, conversationID string) (int64, error) {
	return db.seqConversation.GetMaxSeq(ctx, conversationID)
}

func (db *commonMsgDatabase) SetMinSeqs(ctx context.Context, seqs map[string]int64) error {
	return db.seqConversation.SetMinSeqs(ctx, seqs)
}

func (db *commonMsgDatabase) SetUserConversationsMinSeqs(ctx context.Context, userID string, seqs map[string]int64)
	return db.seqUser.SetUserMinSeqs(ctx, userID, seqs)
}

func (db *commonMsgDatabase) SetUserConversationsMaxSeq(ctx context.Context, conversationID string, userID string, s
	return db.seqUser.SetUserMaxSeq(ctx, conversationID, userID, seq)
}

func (db *commonMsgDatabase) SetUserConversationsMinSeq(ctx context.Context, conversationID string, userID string, s
	return db.seqUser.SetUserMinSeq(ctx, conversationID, userID, seq)
}

func (db *commonMsgDatabase) UserSetHasReadSeqs(ctx context.Context, userID string, hasReadSeqs map[string]int64) er
	return db.seqUser.SetUserReadSeqs(ctx, userID, hasReadSeqs)
}

func (db *commonMsgDatabase) SetHasReadSeq(ctx context.Context, userID string, conversationID string, hasReadSeq int
	return db.seqUser.SetUserReadSeq(ctx, conversationID, userID, hasReadSeq)
}

func (db *commonMsgDatabase) GetHasReadSeqs(ctx context.Context, userID string, conversationIDs []string) (map[strin
```

```
	return db.seqUser.GetUserReadSeqs(ctx, userID, conversationIDs)
}

func (db *commonMsgDatabase) GetHasReadSeq(ctx context.Context, userID string, conversationID string) (int64, error)
	return db.seqUser.GetUserReadSeq(ctx, conversationID, userID)
}

func (db *commonMsgDatabase) SetSendMsgStatus(ctx context.Context, id string, status int32) error {
	return db.msgCache.SetSendMsgStatus(ctx, id, status)
}

func (db *commonMsgDatabase) GetSendMsgStatus(ctx context.Context, id string) (int32, error) {
	return db.msgCache.GetSendMsgStatus(ctx, id)
}

func (db *commonMsgDatabase) GetConversationMinMaxSeqInMongoAndCache(ctx context.Context, conversationID string) (mi
	minSeqMongo, maxSeqMongo, err = db.GetMinMaxSeqMongo(ctx, conversationID)
	if err != nil {
		return
	}
	minSeqCache, err = db.seqConversation.GetMinSeq(ctx, conversationID)
	if err != nil {
		return
	}
	maxSeqCache, err = db.seqConversation.GetMaxSeq(ctx, conversationID)
	if err != nil {
		return
	}
	return
}

func (db *commonMsgDatabase) GetMongoMaxAndMinSeq(ctx context.Context, conversationID string) (minSeqMongo, maxSeqMo
	return db.GetMinMaxSeqMongo(ctx, conversationID)
}

func (db *commonMsgDatabase) GetMinMaxSeqMongo(ctx context.Context, conversationID string) (minSeqMongo, maxSeqMongo
	oldestMsgMongo, err := db.msgDocDatabase.GetOldestMsg(ctx, conversationID)
	if err != nil {
		return
	}
	minSeqMongo = oldestMsgMongo.Msg.Seq
	newestMsgMongo, err := db.msgDocDatabase.GetNewestMsg(ctx, conversationID)
	if err != nil {
		return
	}
	maxSeqMongo = newestMsgMongo.Msg.Seq
	return
}

func (db *commonMsgDatabase) RangeUserSendCount(ctx context.Context, start time.Time, end time.Time, group bool, ase
	return db.msgDocDatabase.RangeUserSendCount(ctx, start, end, group, ase, pageNumber, showNumber)
}

func (db *commonMsgDatabase) RangeGroupSendCount(ctx context.Context, start time.Time, end time.Time, ase bool, page
	return db.msgDocDatabase.RangeGroupSendCount(ctx, start, end, ase, pageNumber, showNumber)
}

func (db *commonMsgDatabase) SearchMessage(ctx context.Context, req *pbmsg.SearchMessageReq) (total int64, msgData [
	var totalMsgs []*pbmsg.SearchedMsgData
	total, msgs, err := db.msgDocDatabase.SearchMessage(ctx, req)
	if err != nil {
		return 0, nil, err
	}
	for _, msg := range msgs {
		if msg.IsRead {
			msg.Msg.IsRead = true
```

```
    }
    searchedMsgData := &pbmsg.SearchedMsgData{MsgData: convert.MsgDB2Pb(msg.Msg)}

    if msg.Revoke != nil {
        searchedMsgData.IsRevoked = true
    }

    totalMsgs = append(totalMsgs, searchedMsgData)
  }
  return total, totalMsgs, nil
}

func (db *commonMsgDatabase) FindOneByDocIDs(ctx context.Context, conversationIDs []string, seqs map[string]int64) (
  totalMsgs := make(map[string]*sdkws.MsgData)
  for _, conversationID := range conversationIDs {
    seq, ok := seqs[conversationID]
    if !ok {
        log.ZWarn(ctx, "seq not found for conversationID", errs.New("seq not found for conversation"), "conversationID",
        continue
    }
    docID := db.msgTable.GetDocID(conversationID, seq)
    msgs, err := db.msgDocDatabase.FindOneByDocID(ctx, docID)
    if err != nil {
        log.ZWarn(ctx, "FindOneByDocID failed", err, "conversationID", conversationID, "docID", docID, "seq", seq)
        continue
    }

    index := db.msgTable.GetMsgIndex(seq)
    totalMsgs[conversationID] = convert.MsgDB2Pb(msgs.Msg[index].Msg)
  }
  return totalMsgs, nil
}

func (db *commonMsgDatabase) GetRandBeforeMsg(ctx context.Context, ts int64, limit int) ([]*model.MsgDocModel, error
  return db.msgDocDatabase.GetRandBeforeMsg(ctx, ts, limit)
}

func (db *commonMsgDatabase) SetMinSeq(ctx context.Context, conversationID string, seq int64) error {
  dbSeq, err := db.seqConversation.GetMinSeq(ctx, conversationID)
  if err != nil {
    if errors.Is(errs.Unwrap(err), redis.Nil) {
        return nil
    }
    return err
  }
  if dbSeq >= seq {
    return nil
  }
  return db.seqConversation.SetMinSeq(ctx, conversationID, seq)
}

func (db *commonMsgDatabase) GetCacheMaxSeqWithTime(ctx context.Context, conversationIDs []string) (map[string]datab
  return db.seqConversation.GetCacheMaxSeqWithTime(ctx, conversationIDs)
}

func (db *commonMsgDatabase) GetMaxSeqWithTime(ctx context.Context, conversationID string) (database.SeqTime, error)
  return db.seqConversation.GetMaxSeqWithTime(ctx, conversationID)
}

func (db *commonMsgDatabase) GetMaxSeqsWithTime(ctx context.Context, conversationIDs []string) (map[string]database.
  // todo: only the time in the redis cache will be taken, not the message time
  return db.seqConversation.GetMaxSeqsWithTime(ctx, conversationIDs)
}

func (db *commonMsgDatabase) DeleteDoc(ctx context.Context, docID string) error {
  index := strings.LastIndex(docID, ":")
```

```
	if index <= 0 {
		return errs.ErrInternalServer.WrapMsg("docID is invalid", "docID", docID)
	}
	docIndex, err := strconv.Atoi(docID[index+1:])
	if err != nil {
		return errs.WrapMsg(err, "strconv.Atoi", "docID", docID)
	}
	conversationID := docID[:index]
	seqs := make([]int64, db.msgTable.GetSingleGocMsgNum())
	minSeq := db.msgTable.GetMinSeq(docIndex)
	for i := range seqs {
		seqs[i] = minSeq + int64(i)
	}
	if err := db.msgDocDatabase.DeleteDoc(ctx, docID); err != nil {
		return err
	}
	return db.msgCache.DelMessageBySeqs(ctx, conversationID, seqs)
}

func (db *commonMsgDatabase) GetLastMessageSeqByTime(ctx context.Context, conversationID string, time int64) (int64,
	return db.msgDocDatabase.GetLastMessageSeqByTime(ctx, conversationID, time)
}

func (db *commonMsgDatabase) handlerDeleteAndRevoked(ctx context.Context, userID string, msgs []*model.MsgInfoModel)
	for i := range msgs {
		msg := msgs[i]
		if msg == nil || msg.Msg == nil {
			continue
		}
		msg.Msg.IsRead = msg.IsRead
		if datautil.Contain(userID, msg.DelList...) {
			msg.Msg.Content = ""
			msg.Msg.Status = constant.MsgDeleted
		}
		if msg.Revoke == nil {
			continue
		}
		msg.Msg.ContentType = constant.MsgRevokeNotification
		revokeContent := sdkws.MessageRevokedContent{
			RevokerID:                  msg.Revoke.UserID,
			RevokerRole:                msg.Revoke.Role,
			ClientMsgID:                msg.Msg.ClientMsgID,
			RevokerNickname:            msg.Revoke.Nickname,
			RevokeTime:                 msg.Revoke.Time,
			SourceMessageSendTime:      msg.Msg.SendTime,
			SourceMessageSendID:        msg.Msg.SendID,
			SourceMessageSenderNickname: msg.Msg.SenderNickname,
			SessionType:                msg.Msg.SessionType,
			Seq:                        msg.Msg.Seq,
			Ex:                         msg.Msg.Ex,
		}
		data, err := jsonutil.JsonMarshal(&revokeContent)
		if err != nil {
			log.ZWarn(ctx, "handlerDeleteAndRevoked JsonMarshal MessageRevokedContent", err, "msg", msg)
			continue
		}
		elem := sdkws.NotificationElem{
			Detail: string(data),
		}
		content, err := jsonutil.JsonMarshal(&elem)
		if err != nil {
			log.ZWarn(ctx, "handlerDeleteAndRevoked JsonMarshal NotificationElem", err, "msg", msg)
			continue
		}
		msg.Msg.Content = string(content)
	}
```

```
}

func (db *commonMsgDatabase) handlerQuote(ctx context.Context, userID, conversationID string, msgs []*model.MsgInfoM
	temp := make(map[int64][]*model.MsgInfoModel)
	for i := range msgs {
		db.handlerDBMsg(ctx, temp, userID, conversationID, msgs[i])
	}
}

func (db *commonMsgDatabase) GetMessageBySeqs(ctx context.Context, conversationID string, userID string, seqs []int6
	msgs, err := db.msgCache.GetMessageBySeqs(ctx, conversationID, seqs)
	if err != nil {
		return nil, err
	}
	db.handlerDeleteAndRevoked(ctx, userID, msgs)
	db.handlerQuote(ctx, userID, conversationID, msgs)
	seqMsgs := make(map[int64]*model.MsgInfoModel)
	for i, msg := range msgs {
		if msg.Msg == nil {
			continue
		}
		seqMsgs[msg.Msg.Seq] = msgs[i]
	}
	res := make([]*sdkws.MsgData, 0, len(seqs))
	for _, seq := range seqs {
		if v, ok := seqMsgs[seq]; ok {
			res = append(res, convert.MsgDB2Pb(v.Msg))
		} else {
			res = append(res, &sdkws.MsgData{Seq: seq, Status: constant.MsgStatusHasDeleted})
		}
	}
	return res, nil
}

func (db *commonMsgDatabase) GetLastMessage(ctx context.Context, conversationIDs []string, userID string) (map[strin
	res := make(map[string]*sdkws.MsgData)
	for _, conversationID := range conversationIDs {
		if _, ok := res[conversationID]; ok {
			continue
		}
		msg, err := db.msgDocDatabase.GetLastMessage(ctx, conversationID)
		if err != nil {
			if errs.Unwrap(err) == mongo.ErrNoDocuments {
				continue
			}
			return nil, err
		}
		tmp := []*model.MsgInfoModel{msg}
		db.handlerDeleteAndRevoked(ctx, userID, tmp)
		db.handlerQuote(ctx, userID, conversationID, tmp)
		res[conversationID] = convert.MsgDB2Pb(msg.Msg)
	}
	return res, nil
}
```

pkg/common/storage/controller/msg_transfer.go

```
package controller

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/mq"
	"github.com/openimsdk/tools/utils/datautil"
	"google.golang.org/protobuf/proto"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	pbmsg "github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"go.mongodb.org/mongo-driver/mongo"
)

type MsgTransferDatabase interface {
	// BatchInsertChat2DB inserts a batch of messages into the database for a specific conversation.
	BatchInsertChat2DB(ctx context.Context, conversationID string, msgs []*sdkws.MsgData, currentMaxSeq int64) error
	// DeleteMessagesFromCache deletes message caches from Redis by sequence numbers.
	DeleteMessagesFromCache(ctx context.Context, conversationID string, seqs []int64) error

	// BatchInsertChat2Cache increments the sequence number and then batch inserts messages into the cache.
	BatchInsertChat2Cache(ctx context.Context, conversationID string, msgs []*sdkws.MsgData) (seq int64, isNewConversat

	SetHasReadSeqs(ctx context.Context, conversationID string, userSeqMap map[string]int64) error

	SetHasReadSeqToDB(ctx context.Context, conversationID string, userSeqMap map[string]int64) error

	// to mq
	MsgToPushMQ(ctx context.Context, key, conversationID string, msg2mq *sdkws.MsgData) error
	MsgToMongoMQ(ctx context.Context, key, conversationID string, msgs []*sdkws.MsgData, lastSeq int64) error
}

func NewMsgTransferDatabase(msgDocModel database.Msg, msg cache.MsgCache, seqUser cache.SeqUser, seqConversation cac
	//conf, err := kafka.BuildProducerConfig(*kafkaConf.Build())
	//if err != nil {
	//	return nil, err
	//}
	//producerToMongo, err := kafka.NewKafkaProducerV2(conf, kafkaConf.Address, kafkaConf.ToMongoTopic)
	//if err != nil {
	//	return nil, err
	//}
	//producerToPush, err := kafka.NewKafkaProducerV2(conf, kafkaConf.Address, kafkaConf.ToPushTopic)
	//if err != nil {
	//	return nil, err
	//}
	return &msgTransferDatabase{
		msgDocDatabase:  msgDocModel,
		msgCache:        msg,
		seqUser:         seqUser,
		seqConversation: seqConversation,
		producerToMongo: mongoProducer,
		producerToPush:  pushProducer,
	}, nil
}

type msgTransferDatabase struct {
	msgDocDatabase  database.Msg
```

```
msgTable        model.MsgDocModel
msgCache        cache.MsgCache
seqConversation cache.SeqConversationCache
seqUser         cache.SeqUser
producerToMongo mq.Producer
producerToPush  mq.Producer
}

func (db *msgTransferDatabase) BatchInsertChat2DB(ctx context.Context, conversationID string, msgList []*sdkws.MsgDa
if len(msgList) == 0 {
    return errs.ErrArgs.WrapMsg("msgList is empty")
}
msgs := make([]any, len(msgList))
seqs := make([]int64, len(msgList))
for i, msg := range msgList {
    if msg == nil {
        continue
    }
    seqs[i] = msg.Seq
    if msg.Status == constant.MsgStatusSending {
        msg.Status = constant.MsgStatusSendSuccess
    }
    msgs[i] = convert.MsgPb2DB(msg)
}
if err := db.BatchInsertBlock(ctx, conversationID, msgs, updateKeyMsg, msgList[0].Seq); err != nil {
    return err
}
//return db.msgCache.DelMessageBySeqs(ctx, conversationID, seqs)
return nil
}

func (db *msgTransferDatabase) BatchInsertBlock(ctx context.Context, conversationID string, fields []any, key int8,
if len(fields) == 0 {
    return nil
}
num := db.msgTable.GetSingleGocMsgNum()
// num = 100
for i, field := range fields { // Check the type of the field
    var ok bool
    switch key {
    case updateKeyMsg:
        var msg *model.MsgDataModel
        msg, ok = field.(*model.MsgDataModel)
        if msg != nil && msg.Seq != firstSeq+int64(i) {
            return errs.ErrInternalServer.WrapMsg("seq is invalid")
        }
    case updateKeyRevoke:
        _, ok = field.(*model.RevokeModel)
    default:
        return errs.ErrInternalServer.WrapMsg("key is invalid")
    }
    if !ok {
        return errs.ErrInternalServer.WrapMsg("field type is invalid")
    }
}
// Returns true if the document exists in the database, false if the document does not exist in the database
updateMsgModel := func(seq int64, i int) (bool, error) {
    var (
        res *mongo.UpdateResult
        err error
    )
    docID := db.msgTable.GetDocID(conversationID, seq)
    index := db.msgTable.GetMsgIndex(seq)
    field := fields[i]
    switch key {
    case updateKeyMsg:
```

```
        res, err = db.msgDocDatabase.UpdateMsg(ctx, docID, index, "msg", field)
    case updateKeyRevoke:
        res, err = db.msgDocDatabase.UpdateMsg(ctx, docID, index, "revoke", field)
    }
    if err != nil {
        return false, err
    }
    return res.MatchedCount > 0, nil
  }
  tryUpdate := true
  for i := 0; i < len(fields); i++ {
    seq := firstSeq + int64(i) // Current sequence number
    if tryUpdate {
        matched, err := updateMsgModel(seq, i)
        if err != nil {
            return err
        }
        if matched {
            continue // The current data has been updated, skip the current data
        }
    }
    doc := model.MsgDocModel{
        DocID: db.msgTable.GetDocID(conversationID, seq),
        Msg:   make([]*model.MsgInfoModel, num),
    }
    var insert int // Inserted data number
    for j := i; j < len(fields); j++ {
        seq = firstSeq + int64(j)
        if db.msgTable.GetDocID(conversationID, seq) != doc.DocID {
            break
        }
        insert++
        switch key {
        case updateKeyMsg:
            doc.Msg[db.msgTable.GetMsgIndex(seq)] = &model.MsgInfoModel{
                Msg: fields[j].(*model.MsgDataModel),
            }
        case updateKeyRevoke:
            doc.Msg[db.msgTable.GetMsgIndex(seq)] = &model.MsgInfoModel{
                Revoke: fields[j].(*model.RevokeModel),
            }
        }
    }
    for i, msgInfo := range doc.Msg {
        if msgInfo == nil {
            msgInfo = &model.MsgInfoModel{}
            doc.Msg[i] = msgInfo
        }
        if msgInfo.DelList == nil {
            doc.Msg[i].DelList = []string{}
        }
    }
    if err := db.msgDocDatabase.Create(ctx, &doc); err != nil {
        if mongo.IsDuplicateKeyError(err) {
            i--              // already inserted
            tryUpdate = true // next block use update mode
            continue
        }
        return err
    }
    tryUpdate = false // The current block is inserted successfully, and the next block is inserted preferentially
    i += insert - 1   // Skip the inserted data
  }
  return nil
}
```

```go
func (db *msgTransferDatabase) DeleteMessagesFromCache(ctx context.Context, conversationID string, seqs []int64) err
	return db.msgCache.DelMessageBySeqs(ctx, conversationID, seqs)
}

func (db *msgTransferDatabase) BatchInsertChat2Cache(ctx context.Context, conversationID string, msgs []*sdkws.MsgDa
	lenList := len(msgs)
	if int64(lenList) > db.msgTable.GetSingleGocMsgNum() {
		return 0, false, nil, errs.New("message count exceeds limit", "limit", db.msgTable.GetSingleGocMsgNum()).Wrap()
	}
	if lenList < 1 {
		return 0, false, nil, errs.New("no messages to insert", "minCount", 1).Wrap()
	}
	currentMaxSeq, err := db.seqConversation.Malloc(ctx, conversationID, int64(len(msgs)))
	if err != nil {
		log.ZError(ctx, "storage.seq.Malloc", err)
		return 0, false, nil, err
	}
	isNew = currentMaxSeq == 0
	lastMaxSeq := currentMaxSeq
	userSeqMap := make(map[string]int64)
	seqs := make([]int64, 0, lenList)
	for _, m := range msgs {
		currentMaxSeq++
		m.Seq = currentMaxSeq
		userSeqMap[m.SendID] = m.Seq
		seqs = append(seqs, m.Seq)
	}
	msgToDB := func(msg *sdkws.MsgData) *model.MsgInfoModel {
		return &model.MsgInfoModel{
			Msg: convert.MsgPb2DB(msg),
		}
	}
	if err := db.msgCache.SetMessageBySeqs(ctx, conversationID, datautil.Slice(msgs, msgToDB)); err != nil {
		return 0, false, nil, err
	}
	return lastMaxSeq, isNew, userSeqMap, nil
}

func (db *msgTransferDatabase) SetHasReadSeqs(ctx context.Context, conversationID string, userSeqMap map[string]int6
	for userID, seq := range userSeqMap {
		if err := db.seqUser.SetUserReadSeq(ctx, conversationID, userID, seq); err != nil {
			return err
		}
	}
	return nil
}

func (db *msgTransferDatabase) SetHasReadSeqToDB(ctx context.Context, conversationID string, userSeqMap map[string]i
	for userID, seq := range userSeqMap {
		if err := db.seqUser.SetUserReadSeqToDB(ctx, conversationID, userID, seq); err != nil {
			return err
		}
	}
	return nil
}

func (db *msgTransferDatabase) MsgToPushMQ(ctx context.Context, key, conversationID string, msg2mq *sdkws.MsgData) e
	data, err := proto.Marshal(&pbmsg.PushMsgDataToMQ{MsgData: msg2mq, ConversationID: conversationID})
	if err != nil {
		return err
	}
	if err := db.producerToPush.SendMessage(ctx, key, data); err != nil {
		log.ZError(ctx, "MsgToPushMQ", err, "key", key, "conversationID", conversationID)
		return err
	}
	return nil
```

```
}

func (db *msgTransferDatabase) MsgToMongoMQ(ctx context.Context, key, conversationID string, messages []*sdkws.MsgDa
  if len(messages) > 0 {
    data, err := proto.Marshal(&pbmsg.MsgDataToMongoByMQ{LastSeq: lastSeq, ConversationID: conversationID, MsgData: m
    if err != nil {
      return err
    }
    if err := db.producerToMongo.SendMessage(ctx, key, data); err != nil {
      log.ZError(ctx, "MsgToMongoMQ", err, "key", key, "conversationID", conversationID, "lastSeq", lastSeq)
      return err
    }
  }
  return nil
}
```

pkg/common/storage/controller/push.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/protocol/push"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mq"
	"google.golang.org/protobuf/proto"
)

type PushDatabase interface {
	DelFcmToken(ctx context.Context, userID string, platformID int) error
	MsgToOfflinePushMQ(ctx context.Context, key string, userIDs []string, msg2mq *sdkws.MsgData) error
}

type pushDataBase struct {
	cache                 cache.ThirdCache
	producerToOfflinePush mq.Producer
}

func NewPushDatabase(cache cache.ThirdCache, offlinePushProducer mq.Producer) PushDatabase {
	return &pushDataBase{
		cache:                 cache,
		producerToOfflinePush: offlinePushProducer,
	}
}

func (p *pushDataBase) DelFcmToken(ctx context.Context, userID string, platformID int) error {
	return p.cache.DelFcmToken(ctx, userID, platformID)
}

func (p *pushDataBase) MsgToOfflinePushMQ(ctx context.Context, key string, userIDs []string, msg2mq *sdkws.MsgData)
	data, err := proto.Marshal(&push.PushMsgReq{MsgData: msg2mq, UserIDs: userIDs})
	if err != nil {
		return err
	}
	if err := p.producerToOfflinePush.SendMessage(ctx, key, data); err != nil {
		log.ZError(ctx, "message is push to offlinePush topic", err, "key", key, "userIDs", userIDs, "msg", msg2mq.String
	}
	return err
}
```

pkg/common/storage/controller/s3.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
    "context"
    "path/filepath"
    "time"

    redisCache "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/tools/s3"
    "github.com/openimsdk/tools/s3/cont"
    "github.com/redis/go-redis/v9"
)

type S3Database interface {
    PartLimit() (*s3.PartLimit, error)
    PartSize(ctx context.Context, size int64) (int64, error)
    AuthSign(ctx context.Context, uploadID string, partNumbers []int) (*s3.AuthSignResult, error)
    InitiateMultipartUpload(ctx context.Context, hash string, size int64, expire time.Duration, maxParts int, contentTy
    CompleteMultipartUpload(ctx context.Context, uploadID string, parts []string) (*cont.UploadResult, error)
    AccessURL(ctx context.Context, name string, expire time.Duration, opt *s3.AccessURLOption) (time.Time, string, erro
    SetObject(ctx context.Context, info *model.Object) error
    StatObject(ctx context.Context, name string) (*s3.ObjectInfo, error)
    FormData(ctx context.Context, name string, size int64, contentType string, duration time.Duration) (*s3.FormData, e
    FindExpirationObject(ctx context.Context, engine string, expiration time.Time, needDelType []string, count int64) (
    DeleteSpecifiedData(ctx context.Context, engine string, name []string) error
    DelS3Key(ctx context.Context, engine string, keys ...string) error
    GetKeyCount(ctx context.Context, engine string, key string) (int64, error)
}

func NewS3Database(rdb redis.UniversalClient, s3 s3.Interface, obj database.ObjectInfo) S3Database {
    return &s3Database{
        s3:      cont.New(redisCache.NewS3Cache(rdb, s3), s3),
        cache:   redisCache.NewObjectCacheRedis(rdb, obj),
        s3cache: redisCache.NewS3Cache(rdb, s3),
        db:      obj,
    }
}

type s3Database struct {
    s3      *cont.Controller
    cache   cache.ObjectCache
    s3cache cont.S3Cache
    db      database.ObjectInfo
}

func (s *s3Database) PartSize(ctx context.Context, size int64) (int64, error) {
```

```
	return s.s3.PartSize(ctx, size)
}

func (s *s3Database) PartLimit() (*s3.PartLimit, error) {
	return s.s3.PartLimit()
}

func (s *s3Database) AuthSign(ctx context.Context, uploadID string, partNumbers []int) (*s3.AuthSignResult, error) {
	return s.s3.AuthSign(ctx, uploadID, partNumbers)
}

func (s *s3Database) InitiateMultipartUpload(ctx context.Context, hash string, size int64, expire time.Duration, max
	return s.s3.InitiateUploadContentType(ctx, hash, size, expire, maxParts, contentType)
}

func (s *s3Database) CompleteMultipartUpload(ctx context.Context, uploadID string, parts []string) (*cont.UploadResu
	return s.s3.CompleteUpload(ctx, uploadID, parts)
}

func (s *s3Database) SetObject(ctx context.Context, info *model.Object) error {
	info.Engine = s.s3.Engine()
	if err := s.db.SetObject(ctx, info); err != nil {
		return err
	}
	return s.cache.DelObjectName(info.Engine, info.Name).ChainExecDel(ctx)
}

func (s *s3Database) AccessURL(ctx context.Context, name string, expire time.Duration, opt *s3.AccessURLOption) (tim
	obj, err := s.cache.GetName(ctx, s.s3.Engine(), name)
	if err != nil {
		return time.Time{}, "", err
	}
	if opt == nil {
		opt = &s3.AccessURLOption{}
	}
	if opt.ContentType == "" {
		opt.ContentType = obj.ContentType
	}
	if opt.Filename == "" {
		opt.Filename = filepath.Base(obj.Name)
	}
	expireTime := time.Now().Add(expire)
	rawURL, err := s.s3.AccessURL(ctx, obj.Key, expire, opt)
	if err != nil {
		return time.Time{}, "", err
	}
	return expireTime, rawURL, nil
}

func (s *s3Database) StatObject(ctx context.Context, name string) (*s3.ObjectInfo, error) {
	return s.s3.StatObject(ctx, name)
}

func (s *s3Database) FormData(ctx context.Context, name string, size int64, contentType string, duration time.Durati
	return s.s3.FormData(ctx, name, size, contentType, duration)
}

func (s *s3Database) FindExpirationObject(ctx context.Context, engine string, expiration time.Time, needDelType []st
	return s.db.FindExpirationObject(ctx, engine, expiration, needDelType, count)
}

func (s *s3Database) GetKeyCount(ctx context.Context, engine string, key string) (int64, error) {
	return s.db.GetKeyCount(ctx, engine, key)
}

func (s *s3Database) DeleteSpecifiedData(ctx context.Context, engine string, name []string) error {
```

```
    return s.db.Delete(ctx, engine, name)
}

func (s *s3Database) DelS3Key(ctx context.Context, engine string, keys ...string) error {
    return s.s3cache.DelS3Key(ctx, engine, keys...)
}
```

pkg/common/storage/controller/third.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/tools/db/pagination"
)

type ThirdDatabase interface {
	FcmUpdateToken(ctx context.Context, account string, platformID int, fcmToken string, expireTime int64) error
	SetAppBadge(ctx context.Context, userID string, value int) error
	// about log for debug
	UploadLogs(ctx context.Context, logs []*model.Log) error
	DeleteLogs(ctx context.Context, logID []string, userID string) error
	SearchLogs(ctx context.Context, keyword string, start time.Time, end time.Time, pagination pagination.Pagination) (
	GetLogs(ctx context.Context, LogIDs []string, userID string) ([]*model.Log, error)
}

type thirdDatabase struct {
	cache cache.ThirdCache
	logdb database.Log
}

// DeleteLogs implements ThirdDatabase.
func (t *thirdDatabase) DeleteLogs(ctx context.Context, logID []string, userID string) error {
	return t.logdb.Delete(ctx, logID, userID)
}

// GetLogs implements ThirdDatabase.
func (t *thirdDatabase) GetLogs(ctx context.Context, LogIDs []string, userID string) ([]*model.Log, error) {
	return t.logdb.Get(ctx, LogIDs, userID)
}

// SearchLogs implements ThirdDatabase.
func (t *thirdDatabase) SearchLogs(ctx context.Context, keyword string, start time.Time, end time.Time, pagination p
	return t.logdb.Search(ctx, keyword, start, end, pagination)
}

// UploadLogs implements ThirdDatabase.
func (t *thirdDatabase) UploadLogs(ctx context.Context, logs []*model.Log) error {
	return t.logdb.Create(ctx, logs)
}

func NewThirdDatabase(cache cache.ThirdCache, logdb database.Log) ThirdDatabase {
	return &thirdDatabase{cache: cache, logdb: logdb}
```

```
}

func (t *thirdDatabase) FcmUpdateToken(ctx context.Context, account string, platformID int, fcmToken string, expireT
	return t.cache.SetFcmToken(ctx, account, platformID, fcmToken, expireTime)
}

func (t *thirdDatabase) SetAppBadge(ctx context.Context, userID string, value int) error {
	return t.cache.SetUserBadgeUnreadCountSum(ctx, userID, value)
}
```

pkg/common/storage/controller/user.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package controller

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/db/pagination"
	"github.com/openimsdk/tools/db/tx"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
)

type UserDatabase interface {
	// FindWithError Get the information of the specified user. If the userID is not found, it will also return an erro
	FindWithError(ctx context.Context, userIDs []string) (users []*model.User, err error)
	// Find Get the information of the specified user If the userID is not found, no error will be returned
	Find(ctx context.Context, userIDs []string) (users []*model.User, err error)
	// Find userInfo By Nickname
	FindByNickname(ctx context.Context, nickname string) (users []*model.User, err error)
	// FindNotification find system account by level
	FindNotification(ctx context.Context, level int64) (users []*model.User, err error)
	// FindSystemAccount find all system account
	FindSystemAccount(ctx context.Context) (users []*model.User, err error)
	// Create Insert multiple external guarantees that the userID is not repeated and does not exist in the storage
	Create(ctx context.Context, users []*model.User) (err error)
	// UpdateByMap update (zero value) external guarantee userID exists
	UpdateByMap(ctx context.Context, userID string, args map[string]any) (err error)
	// FindUser
	PageFindUser(ctx context.Context, level1 int64, level2 int64, pagination pagination.Pagination) (count int64, users
	// FindUser with keyword
	PageFindUserWithKeyword(ctx context.Context, level1 int64, level2 int64, userID string, nickName string, paginatior
	// Page If not found, no error is returned
	Page(ctx context.Context, pagination pagination.Pagination) (count int64, users []*model.User, err error)
	// IsExist true as long as one exists
	IsExist(ctx context.Context, userIDs []string) (exist bool, err error)
	// GetAllUserID Get all user IDs
	GetAllUserID(ctx context.Context, pagination pagination.Pagination) (int64, []string, error)
	// Get user by userID
	GetUserByID(ctx context.Context, userID string) (user *model.User, err error)
	// InitOnce Inside the function, first query whether it exists in the storage, if it exists, do nothing; if it does
	InitOnce(ctx context.Context, users []*model.User) (err error)
	// CountTotal Get the total number of users
	CountTotal(ctx context.Context, before *time.Time) (int64, error)
	// CountRangeEverydayTotal Get the user increment in the range
```

```
    CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int64, error)

    SortQuery(ctx context.Context, userIDName map[string]string, asc bool) ([]*model.User, error)

    // CRUD user command
    AddUserCommand(ctx context.Context, userID string, Type int32, UUID string, value string, ex string) error
    DeleteUserCommand(ctx context.Context, userID string, Type int32, UUID string) error
    UpdateUserCommand(ctx context.Context, userID string, Type int32, UUID string, val map[string]any) error
    GetUserCommands(ctx context.Context, userID string, Type int32) ([]*user.CommandInfoResp, error)
    GetAllUserCommands(ctx context.Context, userID string) ([]*user.AllCommandInfoResp, error)
}

type userDatabase struct {
    tx     tx.Tx
    userDB database.User
    cache  cache.UserCache
}

func NewUserDatabase(userDB database.User, cache cache.UserCache, tx tx.Tx) UserDatabase {
    return &userDatabase{userDB: userDB, cache: cache, tx: tx}
}

func (u *userDatabase) InitOnce(ctx context.Context, users []*model.User) error {
    // Extract user IDs from the given user models.
    userIDs := datautil.Slice(users, func(e *model.User) string {
        return e.UserID
    })

    // Find existing users in the database.
    existingUsers, err := u.userDB.Find(ctx, userIDs)
    if err != nil {
        return err
    }

    // Determine which users are missing from the database.
    var (
        missing, update []*model.User
    )
    existMap := datautil.SliceToMap(existingUsers, func(e *model.User) string {
        return e.UserID
    })
    orgMap := datautil.SliceToMap(users, func(e *model.User) string { return e.UserID })
    for k, u1 := range orgMap {
        if u2, ok := existMap[k]; !ok {
            missing = append(missing, u1)
        } else if u1.Nickname != u2.Nickname {
            update = append(update, u1)
        }
    }

    // Create records for missing users.
    if len(missing) > 0 {
        if err := u.userDB.Create(ctx, missing); err != nil {
            return err
        }
    }
    if len(update) > 0 {
        for i := range update {
            if err := u.userDB.UpdateByMap(ctx, update[i].UserID, map[string]any{"nickname": update[i].Nickname}); err != ni
                return err
            }
        }
    }

    return nil
}
```

```
// FindWithError Get the information of the specified user and return an error if the userID is not found.
func (u *userDatabase) FindWithError(ctx context.Context, userIDs []string) (users []*model.User, err error) {
    userIDs = datautil.Distinct(userIDs)

    // TODO: Add logic to identify which user IDs are distinct and which user IDs were not found.

    users, err = u.cache.GetUsersInfo(ctx, userIDs)
    if err != nil {
        return
    }

    if len(users) != len(userIDs) {
        err = errs.ErrRecordNotFound.WrapMsg("userID not found")
    }
    return
}

// Find Get the information of the specified user. If the userID is not found, no error will be returned.
func (u *userDatabase) Find(ctx context.Context, userIDs []string) (users []*model.User, err error) {
    return u.cache.GetUsersInfo(ctx, userIDs)
}

func (u *userDatabase) FindByNickname(ctx context.Context, nickname string) (users []*model.User, err error) {
    return u.userDB.TakeByNickname(ctx, nickname)
}

func (u *userDatabase) FindNotification(ctx context.Context, level int64) (users []*model.User, err error) {
    return u.userDB.TakeNotification(ctx, level)
}

func (u *userDatabase) FindSystemAccount(ctx context.Context) (users []*model.User, err error) {
    return u.userDB.TakeGTEAppManagerLevel(ctx, constant.AppNotificationAdmin)
}

// Create Insert multiple external guarantees that the userID is not repeated and does not exist in the storage.
func (u *userDatabase) Create(ctx context.Context, users []*model.User) (err error) {
    return u.tx.Transaction(ctx, func(ctx context.Context) error {
        if err = u.userDB.Create(ctx, users); err != nil {
            return err
        }
        return u.cache.DelUsersInfo(datautil.Slice(users, func(e *model.User) string {
            return e.UserID
        })...).ChainExecDel(ctx)
    })
}

// UpdateByMap update (zero value) externally guarantees that userID exists.
func (u *userDatabase) UpdateByMap(ctx context.Context, userID string, args map[string]any) (err error) {
    return u.tx.Transaction(ctx, func(ctx context.Context) error {
        if err := u.userDB.UpdateByMap(ctx, userID, args); err != nil {
            return err
        }
        return u.cache.DelUsersInfo(userID).ChainExecDel(ctx)
    })
}

// Page Gets, returns no error if not found.
func (u *userDatabase) Page(ctx context.Context, pagination pagination.Pagination) (count int64, users []*model.User
    return u.userDB.Page(ctx, pagination)
}

func (u *userDatabase) PageFindUser(ctx context.Context, level1 int64, level2 int64, pagination pagination.Paginatio
    return u.userDB.PageFindUser(ctx, level1, level2, pagination)
}
```

```go
func (u *userDatabase) PageFindUserWithKeyword(ctx context.Context, level1 int64, level2 int64, userID, nickName str
    return u.userDB.PageFindUserWithKeyword(ctx, level1, level2, userID, nickName, pagination)
}

// IsExist Does userIDs exist? As long as there is one, it will be true.
func (u *userDatabase) IsExist(ctx context.Context, userIDs []string) (exist bool, err error) {
    users, err := u.userDB.Find(ctx, userIDs)
    if err != nil {
        return false, err
    }
    if len(users) > 0 {
        return true, nil
    }
    return false, nil
}

// GetAllUserID Get all user IDs.
func (u *userDatabase) GetAllUserID(ctx context.Context, pagination pagination.Pagination) (total int64, userIDs []s
    return u.userDB.GetAllUserID(ctx, pagination)
}

func (u *userDatabase) GetUserByID(ctx context.Context, userID string) (user *model.User, err error) {
    return u.cache.GetUserInfo(ctx, userID)
}

// CountTotal Get the total number of users.
func (u *userDatabase) CountTotal(ctx context.Context, before *time.Time) (count int64, err error) {
    return u.userDB.CountTotal(ctx, before)
}

// CountRangeEverydayTotal Get the user increment in the range.
func (u *userDatabase) CountRangeEverydayTotal(ctx context.Context, start time.Time, end time.Time) (map[string]int6
    return u.userDB.CountRangeEverydayTotal(ctx, start, end)
}

func (u *userDatabase) SortQuery(ctx context.Context, userIDName map[string]string, asc bool) ([]*model.User, error)
    return u.userDB.SortQuery(ctx, userIDName, asc)
}

func (u *userDatabase) AddUserCommand(ctx context.Context, userID string, Type int32, UUID string, value string, ex
    return u.userDB.AddUserCommand(ctx, userID, Type, UUID, value, ex)
}

func (u *userDatabase) DeleteUserCommand(ctx context.Context, userID string, Type int32, UUID string) error {
    return u.userDB.DeleteUserCommand(ctx, userID, Type, UUID)
}

func (u *userDatabase) UpdateUserCommand(ctx context.Context, userID string, Type int32, UUID string, val map[string
    return u.userDB.UpdateUserCommand(ctx, userID, Type, UUID, val)
}

func (u *userDatabase) GetUserCommands(ctx context.Context, userID string, Type int32) ([]*user.CommandInfoResp, err
    commands, err := u.userDB.GetUserCommand(ctx, userID, Type)
    return commands, err
}

func (u *userDatabase) GetAllUserCommands(ctx context.Context, userID string) ([]*user.AllCommandInfoResp, error) {
    commands, err := u.userDB.GetAllUserCommand(ctx, userID)
    return commands, err
}
```

pkg/common/storage/model

pkg/common/storage/model/application.go

```
package model

import (
	"go.mongodb.org/mongo-driver/bson/primitive"
	"time"
)

type Application struct {
	ID         primitive.ObjectID `bson:"_id"`
	Platform   string             `bson:"platform"`
	Hot        bool               `bson:"hot"`
	Version    string             `bson:"version"`
	Url        string             `bson:"url"`
	Text       string             `bson:"text"`
	Force      bool               `bson:"force"`
	Latest     bool               `bson:"latest"`
	CreateTime time.Time          `bson:"create_time"`
}
```

pkg/common/storage/model/black.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
    "time"
)

type Black struct {
    OwnerUserID    string    `bson:"owner_user_id"`
    BlockUserID    string    `bson:"block_user_id"`
    CreateTime     time.Time `bson:"create_time"`
    AddSource      int32     `bson:"add_source"`
    OperatorUserID string    `bson:"operator_user_id"`
    Ex             string    `bson:"ex"`
}
```

pkg/common/storage/model/cache.go

```go
package model

import "time"

type Cache struct {
	Key      string     `bson:"key"`
	Value    string     `bson:"value"`
	ExpireAt *time.Time `bson:"expire_at"`
}
```

pkg/common/storage/model/client_config.go

```
package model

type ClientConfig struct {
    Key    string `bson:"key"`
    UserID string `bson:"user_id"`
    Value  string `bson:"value"`
}
```

pkg/common/storage/model/conversation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"time"
)

type Conversation struct {
	OwnerUserID           string    `bson:"owner_user_id"`
	ConversationID        string    `bson:"conversation_id"`
	ConversationType      int32     `bson:"conversation_type"`
	UserID                string    `bson:"user_id"`
	GroupID               string    `bson:"group_id"`
	RecvMsgOpt            int32     `bson:"recv_msg_opt"`
	IsPinned              bool      `bson:"is_pinned"`
	IsPrivateChat         bool      `bson:"is_private_chat"`
	BurnDuration          int32     `bson:"burn_duration"`
	GroupAtType           int32     `bson:"group_at_type"`
	AttachedInfo          string    `bson:"attached_info"`
	Ex                    string    `bson:"ex"`
	MaxSeq                int64     `bson:"max_seq"`
	MinSeq                int64     `bson:"min_seq"`
	CreateTime            time.Time `bson:"create_time"`
	IsMsgDestruct         bool      `bson:"is_msg_destruct"`
	MsgDestructTime       int64     `bson:"msg_destruct_time"`
	LatestMsgDestructTime time.Time `bson:"latest_msg_destruct_time"`
}
```

pkg/common/storage/model/doc.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model // import "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model/relation"
```

pkg/common/storage/model/friend.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"go.mongodb.org/mongo-driver/bson/primitive"
	"time"
)

// Friend represents the data structure for a friend relationship in MongoDB.
type Friend struct {
	ID             primitive.ObjectID `bson:"_id"`
	OwnerUserID    string             `bson:"owner_user_id"`
	FriendUserID   string             `bson:"friend_user_id"`
	Remark         string             `bson:"remark"`
	CreateTime     time.Time          `bson:"create_time"`
	AddSource      int32              `bson:"add_source"`
	OperatorUserID string             `bson:"operator_user_id"`
	Ex             string             `bson:"ex"`
	IsPinned       bool               `bson:"is_pinned"`
}
```

pkg/common/storage/model/friend_request.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
    "time"
)

type FriendRequest struct {
    FromUserID    string    `bson:"from_user_id"`
    ToUserID      string    `bson:"to_user_id"`
    HandleResult  int32     `bson:"handle_result"`
    ReqMsg        string    `bson:"req_msg"`
    CreateTime    time.Time `bson:"create_time"`
    HandlerUserID string    `bson:"handler_user_id"`
    HandleMsg     string    `bson:"handle_msg"`
    HandleTime    time.Time `bson:"handle_time"`
    Ex            string    `bson:"ex"`
}
```

pkg/common/storage/model/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"time"
)

type Group struct {
	GroupID                string    `bson:"group_id"`
	GroupName              string    `bson:"group_name"`
	Notification           string    `bson:"notification"`
	Introduction           string    `bson:"introduction"`
	FaceURL                string    `bson:"face_url"`
	CreateTime             time.Time `bson:"create_time"`
	Ex                     string    `bson:"ex"`
	Status                 int32     `bson:"status"`
	CreatorUserID          string    `bson:"creator_user_id"`
	GroupType              int32     `bson:"group_type"`
	NeedVerification       int32     `bson:"need_verification"`
	LookMemberInfo         int32     `bson:"look_member_info"`
	ApplyMemberFriend      int32     `bson:"apply_member_friend"`
	NotificationUpdateTime time.Time `bson:"notification_update_time"`
	NotificationUserID     string    `bson:"notification_user_id"`
}
```

pkg/common/storage/model/group_member.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"time"
)

type GroupMember struct {
	GroupID        string    `bson:"group_id"`
	UserID         string    `bson:"user_id"`
	Nickname       string    `bson:"nickname"`
	FaceURL        string    `bson:"face_url"`
	RoleLevel      int32     `bson:"role_level"`
	JoinTime       time.Time `bson:"join_time"`
	JoinSource     int32     `bson:"join_source"`
	InviterUserID  string    `bson:"inviter_user_id"`
	OperatorUserID string    `bson:"operator_user_id"`
	MuteEndTime    time.Time `bson:"mute_end_time"`
	Ex             string    `bson:"ex"`
}
```

pkg/common/storage/model/group_request.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"time"
)

type GroupRequest struct {
	UserID        string    `bson:"user_id"`
	GroupID       string    `bson:"group_id"`
	HandleResult  int32     `bson:"handle_result"`
	ReqMsg        string    `bson:"req_msg"`
	HandledMsg    string    `bson:"handled_msg"`
	ReqTime       time.Time `bson:"req_time"`
	HandleUserID  string    `bson:"handle_user_id"`
	HandledTime   time.Time `bson:"handled_time"`
	JoinSource    int32     `bson:"join_source"`
	InviterUserID string    `bson:"inviter_user_id"`
	Ex            string    `bson:"ex"`
}
```

pkg/common/storage/model/log.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
    "time"
)

type Log struct {
    LogID        string    `bson:"log_id"`
    Platform     string    `bson:"platform"`
    UserID       string    `bson:"user_id"`
    CreateTime   time.Time `bson:"create_time"`
    Url          string    `bson:"url"`
    FileName     string    `bson:"file_name"`
    SystemType   string    `bson:"system_type"`
    AppFramework string    `bson:"app_framework"`
    Version      string    `bson:"version"`
    Ex           string    `bson:"ex"`
}
```

pkg/common/storage/model/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
    "strconv"

    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/errs"
)

const (
    singleGocMsgNum     = 100
    singleGocMsgNum5000 = 5000
    MsgTableName        = "msg"
    OldestList          = 0
    NewestList          = -1
)

var ErrMsgListNotExist = errs.New("user not have msg in mongoDB")

type MsgDocModel struct {
    DocID string          `bson:"doc_id"`
    Msg   []*MsgInfoModel `bson:"msgs"`
}

type RevokeModel struct {
    Role     int32  `bson:"role"`
    UserID   string `bson:"user_id"`
    Nickname string `bson:"nickname"`
    Time     int64  `bson:"time"`
}

type OfflinePushModel struct {
    Title         string `bson:"title"`
    Desc          string `bson:"desc"`
    Ex            string `bson:"ex"`
    IOSPushSound  string `bson:"ios_push_sound"`
    IOSBadgeCount bool   `bson:"ios_badge_count"`
}

type MsgDataModel struct {
    SendID           string            `bson:"send_id"`
    RecvID           string            `bson:"recv_id"`
    GroupID          string            `bson:"group_id"`
    ClientMsgID      string            `bson:"client_msg_id"`
    ServerMsgID      string            `bson:"server_msg_id"`
    SenderPlatformID int32             `bson:"sender_platform_id"`
    SenderNickname   string            `bson:"sender_nickname"`
    SenderFaceURL    string            `bson:"sender_face_url"`
    SessionType      int32             `bson:"session_type"`
    MsgFrom          int32             `bson:"msg_from"`
```

```
	ContentType      int32             `bson:"content_type"`
	Content          string            `bson:"content"`
	Seq              int64             `bson:"seq"`
	SendTime         int64             `bson:"send_time"`
	CreateTime       int64             `bson:"create_time"`
	Status           int32             `bson:"status"`
	IsRead           bool              `bson:"is_read"`
	Options          map[string]bool   `bson:"options"`
	OfflinePush      *OfflinePushModel `bson:"offline_push"`
	AtUserIDList     []string          `bson:"at_user_id_list"`
	AttachedInfo     string            `bson:"attached_info"`
	Ex               string            `bson:"ex"`
}

type MsgInfoModel struct {
	Msg     *MsgDataModel `bson:"msg"`
	Revoke  *RevokeModel  `bson:"revoke"`
	DelList []string      `bson:"del_list"`
	IsRead  bool          `bson:"is_read"`
}

type UserCount struct {
	UserID string `bson:"user_id"`
	Count  int64  `bson:"count"`
}

type GroupCount struct {
	GroupID string `bson:"group_id"`
	Count   int64  `bson:"count"`
}

func (*MsgDocModel) TableName() string {
	return MsgTableName
}

func (*MsgDocModel) GetSingleGocMsgNum() int64 {
	return singleGocMsgNum
}

func (*MsgDocModel) GetSingleGocMsgNum5000() int64 {
	return singleGocMsgNum5000
}

func (m *MsgDocModel) IsFull() bool {
	return m.Msg[len(m.Msg)-1].Msg != nil
}

func (m *MsgDocModel) GetDocIndex(seq int64) int64 {
	return (seq - 1) / singleGocMsgNum
}

func (m *MsgDocModel) GetDocID(conversationID string, seq int64) string {
	seqSuffix := (seq - 1) / singleGocMsgNum
	return m.indexGen(conversationID, seqSuffix)
}

func (m *MsgDocModel) GetDocIDSeqsMap(conversationID string, seqs []int64) map[string][]int64 {
	t := make(map[string][]int64)
	for _, seq := range seqs {
		docID := m.GetDocID(conversationID, seq)
		t[docID] = append(t[docID], seq)
	}

	return t
}
```

```go
func (*MsgDocModel) GetMsgIndex(seq int64) int64 {
	return (seq - 1) % singleGocMsgNum
}

func (*MsgDocModel) GetLimitForSingleDoc(seq int64) int64 {
	return seq % singleGocMsgNum
}

func (*MsgDocModel) indexGen(conversationID string, seqSuffix int64) string {
	return conversationID + ":" + strconv.FormatInt(seqSuffix, 10)
}

func (*MsgDocModel) BuildDocIDByIndex(conversationID string, index int64) string {
	return conversationID + ":" + strconv.FormatInt(index, 10)
}

func (*MsgDocModel) GenExceptionMessageBySeqs(seqs []int64) (exceptionMsg []*sdkws.MsgData) {
	for _, v := range seqs {
		msgModel := new(sdkws.MsgData)
		msgModel.Seq = v
		exceptionMsg = append(exceptionMsg, msgModel)
	}
	return exceptionMsg
}

func (*MsgDocModel) GetMinSeq(index int) int64 {
	return int64(index*singleGocMsgNum) + 1
}
```

pkg/common/storage/model/object.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"time"
)

type Object struct {
	Name        string    `bson:"name"`
	UserID      string    `bson:"user_id"`
	Hash        string    `bson:"hash"`
	Engine      string    `bson:"engine"`
	Key         string    `bson:"key"`
	Size        int64     `bson:"size"`
	ContentType string    `bson:"content_type"`
	Group       string    `bson:"group"`
	CreateTime  time.Time `bson:"create_time"`
}
```

pkg/common/storage/model/seq.go

```
package model

type SeqConversation struct {
    ConversationID string `bson:"conversation_id"`
    MaxSeq         int64  `bson:"max_seq"`
    MinSeq         int64  `bson:"min_seq"`
}
```

pkg/common/storage/model/seq_user.go

```
package model

type SeqUser struct {
    UserID         string `bson:"user_id"`
    ConversationID string `bson:"conversation_id"`
    MinSeq         int64  `bson:"min_seq"`
    MaxSeq         int64  `bson:"max_seq"`
    ReadSeq        int64  `bson:"read_seq"`
}
```

pkg/common/storage/model/subscribe.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

// SubscribeUserTableName collection constant.
const (
	SubscribeUserTableName = "subscribe_user"
)

// SubscribeUser collection structure.
type SubscribeUser struct {
	UserID     string   `bson:"user_id"      json:"userID"`
	UserIDList []string `bson:"user_id_list" json:"userIDList"`
}

func (SubscribeUser) TableName() string {
	return SubscribeUserTableName
}
```

pkg/common/storage/model/user.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package model

import (
	"time"
)

type User struct {
	UserID           string    `bson:"user_id"`
	Nickname         string    `bson:"nickname"`
	FaceURL          string    `bson:"face_url"`
	Ex               string    `bson:"ex"`
	AppMangerLevel   int32     `bson:"app_manger_level"`
	GlobalRecvMsgOpt int32     `bson:"global_recv_msg_opt"`
	CreateTime       time.Time `bson:"create_time"`
}

func (u *User) GetNickname() string {
	return u.Nickname
}

func (u *User) GetFaceURL() string {
	return u.FaceURL
}

func (u *User) GetUserID() string {
	return u.UserID
}

func (u *User) GetEx() string {
	return u.Ex
}
```

pkg/common/storage/model/version_log.go

```
package model

import (
	"context"
	"errors"
	"github.com/openimsdk/tools/log"
	"go.mongodb.org/mongo-driver/bson/primitive"
	"time"
)

const (
	VersionStateInsert = iota + 1
	VersionStateDelete
	VersionStateUpdate
)

const (
	VersionGroupChangeID = ""
	VersionSortChangeID  = "____S_O_R_T_I_D____"
)

type VersionLogElem struct {
	EID        string    `bson:"e_id"`
	State      int32     `bson:"state"`
	Version    uint      `bson:"version"`
	LastUpdate time.Time `bson:"last_update"`
}

type VersionLogTable struct {
	ID         primitive.ObjectID `bson:"_id"`
	DID        string             `bson:"d_id"`
	Logs       []VersionLogElem   `bson:"logs"`
	Version    uint               `bson:"version"`
	Deleted    uint               `bson:"deleted"`
	LastUpdate time.Time          `bson:"last_update"`
}

func (v *VersionLogTable) VersionLog() *VersionLog {
	return &VersionLog{
		ID:         v.ID,
		DID:        v.DID,
		Logs:       v.Logs,
		Version:    v.Version,
		Deleted:    v.Deleted,
		LastUpdate: v.LastUpdate,
		LogLen:     len(v.Logs),
	}
}

type VersionLog struct {
	ID         primitive.ObjectID `bson:"_id"`
	DID        string             `bson:"d_id"`
	Logs       []VersionLogElem   `bson:"logs"`
	Version    uint               `bson:"version"`
	Deleted    uint               `bson:"deleted"`
	LastUpdate time.Time          `bson:"last_update"`
	LogLen     int                `bson:"log_len"`
}

func (v *VersionLog) DeleteAndChangeIDs() (insertIds, deleteIds, updateIds []string) {
	for _, l := range v.Logs {
		switch l.State {
		case VersionStateInsert:
			insertIds = append(insertIds, l.EID)
```

```
		case VersionStateDelete:
			deleteIds = append(deleteIds, l.EID)
		case VersionStateUpdate:
			updateIds = append(updateIds, l.EID)
		default:
			log.ZError(context.Background(), "invalid version status found", errors.New("dirty database data"), "objID", v.I
		}
	}
	return
}
```

pkg/common/storage/kafka

pkg/common/storage/kafka/config.go

```go
// Copyright © 2024 OpenIM open source community. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package kafka

type TLSConfig struct {
	EnableTLS          bool   `yaml:"enableTLS"`
	CACrt              string `yaml:"caCrt"`
	ClientCrt          string `yaml:"clientCrt"`
	ClientKey          string `yaml:"clientKey"`
	ClientKeyPwd       string `yaml:"clientKeyPwd"`
	InsecureSkipVerify bool   `yaml:"insecureSkipVerify"`
}

type Config struct {
	Username     string    `yaml:"username"`
	Password     string    `yaml:"password"`
	ProducerAck  string    `yaml:"producerAck"`
	CompressType string    `yaml:"compressType"`
	Addr         []string  `yaml:"addr"`
	TLS          TLSConfig `yaml:"tls"`
}
```

pkg/common/storage/kafka/consumer_group.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package kafka

import (
	"context"
	"errors"

	"github.com/IBM/sarama"
	"github.com/openimsdk/tools/log"
)

type MConsumerGroup struct {
	sarama.ConsumerGroup
	groupID string
	topics  []string
}

func NewMConsumerGroup(conf *Config, groupID string, topics []string, autoCommitEnable bool) (*MConsumerGroup, error
	config, err := BuildConsumerGroupConfig(conf, sarama.OffsetNewest, autoCommitEnable)
	if err != nil {
		return nil, err
	}
	group, err := NewConsumerGroup(config, conf.Addr, groupID)
	if err != nil {
		return nil, err
	}
	return &MConsumerGroup{
		ConsumerGroup: group,
		groupID:       groupID,
		topics:        topics,
	}, nil
}

func (mc *MConsumerGroup) GetContextFromMsg(cMsg *sarama.ConsumerMessage) context.Context {
	return GetContextWithMQHeader(cMsg.Headers)
}

func (mc *MConsumerGroup) RegisterHandleAndConsumer(ctx context.Context, handler sarama.ConsumerGroupHandler) {
	for {
		err := mc.ConsumerGroup.Consume(ctx, mc.topics, handler)
		if errors.Is(err, sarama.ErrClosedConsumerGroup) {
			return
		}
		if errors.Is(err, context.Canceled) {
			return
		}
		if err != nil {
			log.ZWarn(ctx, "consume err", err, "topic", mc.topics, "groupID", mc.groupID)
		}
	}
}
```

```go
func (mc *MConsumerGroup) Close() error {
    return mc.ConsumerGroup.Close()
}
```

pkg/common/storage/kafka/producer.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package kafka

import (
    "context"
    "github.com/IBM/sarama"
    "github.com/openimsdk/tools/errs"
    "google.golang.org/protobuf/proto"
)

// Producer represents a Kafka producer.
type Producer struct {
    addr     []string
    topic    string
    config   *sarama.Config
    producer sarama.SyncProducer
}

func NewKafkaProducer(config *sarama.Config, addr []string, topic string) (*Producer, error) {
    producer, err := NewProducer(config, addr)
    if err != nil {
        return nil, err
    }
    return &Producer{
        addr:     addr,
        topic:    topic,
        config:   config,
        producer: producer,
    }, nil
}

// SendMessage sends a message to the Kafka topic configured in the Producer.
func (p *Producer) SendMessage(ctx context.Context, key string, msg proto.Message) (int32, int64, error) {
    // Marshal the protobuf message
    bMsg, err := proto.Marshal(msg)
    if err != nil {
        return 0, 0, errs.WrapMsg(err, "kafka proto Marshal err")
    }
    if len(bMsg) == 0 {
        return 0, 0, errs.WrapMsg(errEmptyMsg, "kafka proto Marshal err")
    }

    // Prepare Kafka message
    kMsg := &sarama.ProducerMessage{
        Topic: p.topic,
        Key:   sarama.StringEncoder(key),
        Value: sarama.ByteEncoder(bMsg),
    }

    // Validate message key and value
    if kMsg.Key.Length() == 0 || kMsg.Value.Length() == 0 {
```

```
    return 0, 0, errs.Wrap(errEmptyMsg)
  }

  // Attach context metadata as headers
  header, err := GetMQHeaderWithContext(ctx)
  if err != nil {
    return 0, 0, err
  }
  kMsg.Headers = header

  // Send the message
  partition, offset, err := p.producer.SendMessage(kMsg)
  if err != nil {
    return 0, 0, errs.WrapMsg(err, "p.producer.SendMessage error")
  }

  return partition, offset, nil
}
```

pkg/common/storage/kafka/sarama.go

```
package kafka

import (
    "bytes"
    "strings"

    "github.com/IBM/sarama"
    "github.com/openimsdk/tools/errs"
)

func BuildConsumerGroupConfig(conf *Config, initial int64, autoCommitEnable bool) (*sarama.Config, error) {
    kfk := sarama.NewConfig()
    kfk.Version = sarama.V2_0_0_0
    kfk.Consumer.Offsets.Initial = initial
    kfk.Consumer.Offsets.AutoCommit.Enable = autoCommitEnable
    kfk.Consumer.Return.Errors = false
    if conf.Username != "" || conf.Password != "" {
        kfk.Net.SASL.Enable = true
        kfk.Net.SASL.User = conf.Username
        kfk.Net.SASL.Password = conf.Password
    }
    if conf.TLS.EnableTLS {
        tls, err := newTLSConfig(conf.TLS.ClientCrt, conf.TLS.ClientKey, conf.TLS.CACrt, []byte(conf.TLS.ClientKeyPwd), c
        if err != nil {
            return nil, err
        }
        kfk.Net.TLS.Config = tls
        kfk.Net.TLS.Enable = true
    }
    return kfk, nil
}

func NewConsumerGroup(conf *sarama.Config, addr []string, groupID string) (sarama.ConsumerGroup, error) {
    cg, err := sarama.NewConsumerGroup(addr, groupID, conf)
    if err != nil {
        return nil, errs.WrapMsg(err, "NewConsumerGroup failed", "addr", addr, "groupID", groupID, "conf", *conf)
    }
    return cg, nil
}

func BuildProducerConfig(conf Config) (*sarama.Config, error) {
    kfk := sarama.NewConfig()
    kfk.Producer.Return.Successes = true
    kfk.Producer.Return.Errors = true
    kfk.Producer.Partitioner = sarama.NewHashPartitioner
    if conf.Username != "" || conf.Password != "" {
        kfk.Net.SASL.Enable = true
        kfk.Net.SASL.User = conf.Username
        kfk.Net.SASL.Password = conf.Password
    }
    switch strings.ToLower(conf.ProducerAck) {
    case "no_response":
        kfk.Producer.RequiredAcks = sarama.NoResponse
    case "wait_for_local":
        kfk.Producer.RequiredAcks = sarama.WaitForLocal
    case "wait_for_all":
        kfk.Producer.RequiredAcks = sarama.WaitForAll
    default:
        kfk.Producer.RequiredAcks = sarama.WaitForAll
    }
    if conf.CompressType == "" {
        kfk.Producer.Compression = sarama.CompressionNone
    } else {
        if err := kfk.Producer.Compression.UnmarshalText(bytes.ToLower([]byte(conf.CompressType))); err != nil {
```

```
        return nil, errs.WrapMsg(err, "UnmarshalText failed", "compressType", conf.CompressType)
    }
  }
  if conf.TLS.EnableTLS {
    tls, err := newTLSConfig(conf.TLS.ClientCrt, conf.TLS.ClientKey, conf.TLS.CACrt, []byte(conf.TLS.ClientKeyPwd), c
    if err != nil {
        return nil, err
    }
    kfk.Net.TLS.Config = tls
    kfk.Net.TLS.Enable = true
  }
  return kfk, nil
}

func NewProducer(conf *sarama.Config, addr []string) (sarama.SyncProducer, error) {
  producer, err := sarama.NewSyncProducer(addr, conf)
  if err != nil {
    return nil, errs.WrapMsg(err, "NewSyncProducer failed", "addr", addr, "conf", *conf)
  }
  return producer, nil
}
```

pkg/common/storage/kafka/tls.go

```
// Copyright © 2024 OpenIM open source community. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package kafka

import (
	"crypto/tls"
	"crypto/x509"
	"encoding/pem"
	"os"

	"github.com/openimsdk/tools/errs"
)

// decryptPEM decrypts a PEM block using a password.
func decryptPEM(data []byte, passphrase []byte) ([]byte, error) {
	if len(passphrase) == 0 {
		return data, nil
	}
	b, _ := pem.Decode(data)
	d, err := x509.DecryptPEMBlock(b, passphrase)
	if err != nil {
		return nil, errs.WrapMsg(err, "DecryptPEMBlock failed")
	}
	return pem.EncodeToMemory(&pem.Block{
		Type:  b.Type,
		Bytes: d,
	}), nil
}

func readEncryptablePEMBlock(path string, pwd []byte) ([]byte, error) {
	data, err := os.ReadFile(path)
	if err != nil {
		return nil, errs.WrapMsg(err, "ReadFile failed", "path", path)
	}
	return decryptPEM(data, pwd)
}

// newTLSConfig setup the TLS config from general config file.
func newTLSConfig(clientCertFile, clientKeyFile, caCertFile string, keyPwd []byte, insecureSkipVerify bool) (*tls.Co
	var tlsConfig tls.Config
	if clientCertFile != "" && clientKeyFile != "" {
		certPEMBlock, err := os.ReadFile(clientCertFile)
		if err != nil {
			return nil, errs.WrapMsg(err, "ReadFile failed", "clientCertFile", clientCertFile)
		}
		keyPEMBlock, err := readEncryptablePEMBlock(clientKeyFile, keyPwd)
		if err != nil {
			return nil, err
		}

		cert, err := tls.X509KeyPair(certPEMBlock, keyPEMBlock)
		if err != nil {
```

```
        return nil, errs.WrapMsg(err, "X509KeyPair failed")
    }
    tlsConfig.Certificates = []tls.Certificate{cert}
  }

  if caCertFile != "" {
    caCert, err := os.ReadFile(caCertFile)
    if err != nil {
        return nil, errs.WrapMsg(err, "ReadFile failed", "caCertFile", caCertFile)
    }
    caCertPool := x509.NewCertPool()
    if ok := caCertPool.AppendCertsFromPEM(caCert); !ok {
        return nil, errs.New("AppendCertsFromPEM failed")
    }
    tlsConfig.RootCAs = caCertPool
  }
  tlsConfig.InsecureSkipVerify = insecureSkipVerify
  return &tlsConfig, nil
}
```

pkg/common/storage/kafka/util.go

```
package kafka

import (
	"context"
	"errors"
	"github.com/IBM/sarama"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/mcontext"
)

var errEmptyMsg = errors.New("kafka binary msg is empty")

// GetMQHeaderWithContext extracts message queue headers from the context.
func GetMQHeaderWithContext(ctx context.Context) ([]sarama.RecordHeader, error) {
	operationID, opUserID, platform, connID, err := mcontext.GetCtxInfos(ctx)
	if err != nil {
		return nil, err
	}
	return []sarama.RecordHeader{
		{Key: []byte(constant.OperationID), Value: []byte(operationID)},
		{Key: []byte(constant.OpUserID), Value: []byte(opUserID)},
		{Key: []byte(constant.OpUserPlatform), Value: []byte(platform)},
		{Key: []byte(constant.ConnID), Value: []byte(connID)},
	}, nil
}

// GetContextWithMQHeader creates a context from message queue headers.
func GetContextWithMQHeader(header []*sarama.RecordHeader) context.Context {
	var values []string
	for _, recordHeader := range header {
		values = append(values, string(recordHeader.Value))
	}
	return mcontext.WithMustInfoCtx(values) // Attach extracted values to context
}
```

pkg/common/storage/kafka/verify.go

```
// Copyright © 2024 OpenIM open source community. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package kafka

import (
	"context"
	"fmt"

	"github.com/IBM/sarama"
	"github.com/openimsdk/tools/errs"
)

func CheckTopics(ctx context.Context, conf *Config, topics []string) error {
	kfk, err := BuildConsumerGroupConfig(conf, sarama.OffsetNewest, false)
	if err != nil {
		return err
	}
	cli, err := sarama.NewClient(conf.Addr, kfk)
	if err != nil {
		return errs.WrapMsg(err, "NewClient failed", "config: ", fmt.Sprintf("%+v", conf))
	}
	defer cli.Close()

	existingTopics, err := cli.Topics()
	if err != nil {
		return errs.WrapMsg(err, "Failed to list topics")
	}

	existingTopicsMap := make(map[string]bool)
	for _, t := range existingTopics {
		existingTopicsMap[t] = true
	}

	for _, topic := range topics {
		if !existingTopicsMap[topic] {
			return errs.New("topic not exist", "topic", topic).Wrap()
		}
	}
	return nil
}

func CheckHealth(ctx context.Context, conf *Config) error {
	kfk, err := BuildConsumerGroupConfig(conf, sarama.OffsetNewest, false)
	if err != nil {
		return err
	}
	cli, err := sarama.NewClient(conf.Addr, kfk)
	if err != nil {
		return errs.WrapMsg(err, "NewClient failed", "config: ", fmt.Sprintf("%+v", conf))
	}
	defer cli.Close()
```

```
    // Get broker list
    brokers := cli.Brokers()
    if len(brokers) == 0 {
        return errs.New("no brokers found").Wrap()
    }

    // Check if all brokers are reachable
    for _, broker := range brokers {
        if err := broker.Open(kfk); err != nil {
            return errs.WrapMsg(err, "failed to open broker", "broker", broker.Addr())
        }
    }

    return nil
}
```

pkg/common/storage/cache

pkg/common/storage/cache/batch_handler.go

```go
package cache

import (
	"context"
)

// BatchDeleter interface defines a set of methods for batch deleting cache and publishing deletion information.
type BatchDeleter interface {
	//ChainExecDel method is used for chain calls and must call Clone to prevent memory pollution.
	ChainExecDel(ctx context.Context) error
	//ExecDelWithKeys method directly takes keys for deletion.
	ExecDelWithKeys(ctx context.Context, keys []string) error
	//Clone method creates a copy of the BatchDeleter to avoid modifying the original object.
	Clone() BatchDeleter
	//AddKeys method adds keys to be deleted.
	AddKeys(keys ...string)
}
```

pkg/common/storage/cache/black.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
	"context"
)

type BlackCache interface {
	BatchDeleter
	CloneBlackCache() BlackCache
	GetBlackIDs(ctx context.Context, userID string) (blackIDs []string, err error)
	// del user's blackIDs msgCache, exec when a user's black list changed
	DelBlackIDs(ctx context.Context, userID string) BlackCache
}
```

pkg/common/storage/cache/client_config.go

```
package cache

import "context"

type ClientConfigCache interface {
    DeleteUserCache(ctx context.Context, userIDs []string) error
    GetUserConfig(ctx context.Context, userID string) (map[string]string, error)
}
```

pkg/common/storage/cache/conversation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
	"context"

	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

// arg fn will exec when no data in msgCache.
type ConversationCache interface {
	BatchDeleter
	CloneConversationCache() ConversationCache
	// get user's conversationIDs from msgCache
	GetUserConversationIDs(ctx context.Context, ownerUserID string) ([]string, error)
	GetUserNotNotifyConversationIDs(ctx context.Context, userID string) ([]string, error)
	GetPinnedConversationIDs(ctx context.Context, userID string) ([]string, error)
	DelConversationIDs(userIDs ...string) ConversationCache

	GetUserConversationIDsHash(ctx context.Context, ownerUserID string) (hash uint64, err error)
	DelUserConversationIDsHash(ownerUserIDs ...string) ConversationCache

	// get one conversation from msgCache
	GetConversation(ctx context.Context, ownerUserID, conversationID string) (*relationtb.Conversation, error)
	DelConversations(ownerUserID string, conversationIDs ...string) ConversationCache
	DelUsersConversation(conversationID string, ownerUserIDs ...string) ConversationCache
	// get one conversation from msgCache
	GetConversations(ctx context.Context, ownerUserID string,
		conversationIDs []string) ([]*relationtb.Conversation, error)
	// get one user's all conversations from msgCache
	GetUserAllConversations(ctx context.Context, ownerUserID string) ([]*relationtb.Conversation, error)
	// get user conversation recv msg from msgCache
	GetUserRecvMsgOpt(ctx context.Context, ownerUserID, conversationID string) (opt int, err error)
	DelUserRecvMsgOpt(ownerUserID, conversationID string) ConversationCache
	// get one super group recv msg but do not notification userID list
	// GetSuperGroupRecvMsgNotNotifyUserIDs(ctx context.Context, groupID string) (userIDs []string, err error)
	DelSuperGroupRecvMsgNotNotifyUserIDs(groupID string) ConversationCache
	// get one super group recv msg but do not notification userID list hash
	// GetSuperGroupRecvMsgNotNotifyUserIDsHash(ctx context.Context, groupID string) (hash uint64, err error)
	DelSuperGroupRecvMsgNotNotifyUserIDsHash(groupID string) ConversationCache

	// GetUserAllHasReadSeqs(ctx context.Context, ownerUserID string) (map[string]int64, error)
	DelUserAllHasReadSeqs(ownerUserID string, conversationIDs ...string) ConversationCache

	GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string) ([]string, error)
	DelConversationNotReceiveMessageUserIDs(conversationIDs ...string) ConversationCache
	DelConversationNotNotifyMessageUserIDs(userIDs ...string) ConversationCache
	DelUserPinnedConversations(userIDs ...string) ConversationCache
	DelConversationVersionUserIDs(userIDs ...string) ConversationCache

	FindMaxConversationUserVersion(ctx context.Context, userID string) (*relationtb.VersionLog, error)
```

```
}
```

pkg/common/storage/cache/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache // import "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
```

pkg/common/storage/cache/friend.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
	"context"
	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

// FriendCache is an interface for caching friend-related data.
type FriendCache interface {
	BatchDeleter
	CloneFriendCache() FriendCache
	GetFriendIDs(ctx context.Context, ownerUserID string) (friendIDs []string, err error)
	// Called when friendID list changed
	DelFriendIDs(ownerUserID ...string) FriendCache
	// Get single friendInfo from the cache
	GetFriend(ctx context.Context, ownerUserID, friendUserID string) (friend *relationtb.Friend, err error)
	// Delete friend when friend info changed
	DelFriend(ownerUserID, friendUserID string) FriendCache
	// Delete friends when friends' info changed
	DelFriends(ownerUserID string, friendUserIDs []string) FriendCache

	DelOwner(friendUserID string, ownerUserIDs []string) FriendCache

	DelMaxFriendVersion(ownerUserIDs ...string) FriendCache

	//DelSortFriendUserIDs(ownerUserIDs ...string) FriendCache

	//FindSortFriendUserIDs(ctx context.Context, ownerUserID string) ([]string, error)

	//FindFriendIncrVersion(ctx context.Context, ownerUserID string, version uint, limit int) (*relationtb.VersionLog,

	FindMaxFriendVersion(ctx context.Context, ownerUserID string) (*relationtb.VersionLog, error)
}
```

pkg/common/storage/cache/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/common"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

type GroupHash interface {
    GetGroupHash(ctx context.Context, groupID string) (uint64, error)
}

type GroupCache interface {
    BatchDeleter
    CloneGroupCache() GroupCache
    GetGroupsInfo(ctx context.Context, groupIDs []string) (groups []*model.Group, err error)
    GetGroupInfo(ctx context.Context, groupID string) (group *model.Group, err error)
    DelGroupsInfo(groupIDs ...string) GroupCache

    GetGroupMembersHash(ctx context.Context, groupID string) (hashCode uint64, err error)
    GetGroupMemberHashMap(ctx context.Context, groupIDs []string) (map[string]*common.GroupSimpleUserID, error)
    DelGroupMembersHash(groupID string) GroupCache

    GetGroupMemberIDs(ctx context.Context, groupID string) (groupMemberIDs []string, err error)

    DelGroupMemberIDs(groupID string) GroupCache

    GetJoinedGroupIDs(ctx context.Context, userID string) (joinedGroupIDs []string, err error)
    DelJoinedGroupID(userID ...string) GroupCache

    GetGroupMemberInfo(ctx context.Context, groupID, userID string) (groupMember *model.GroupMember, err error)
    GetGroupMembersInfo(ctx context.Context, groupID string, userID []string) (groupMembers []*model.GroupMember, err e
    GetAllGroupMembersInfo(ctx context.Context, groupID string) (groupMembers []*model.GroupMember, err error)
    FindGroupMemberUser(ctx context.Context, groupIDs []string, userID string) ([]*model.GroupMember, error)

    GetGroupRoleLevelMemberIDs(ctx context.Context, groupID string, roleLevel int32) ([]string, error)
    GetGroupOwner(ctx context.Context, groupID string) (*model.GroupMember, error)
    GetGroupsOwner(ctx context.Context, groupIDs []string) ([]*model.GroupMember, error)
    DelGroupRoleLevel(groupID string, roleLevel []int32) GroupCache
    DelGroupAllRoleLevel(groupID string) GroupCache
    DelGroupMembersInfo(groupID string, userID ...string) GroupCache
    GetGroupRoleLevelMemberInfo(ctx context.Context, groupID string, roleLevel int32) ([]*model.GroupMember, error)
    GetGroupRolesLevelMemberInfo(ctx context.Context, groupID string, roleLevels []int32) ([]*model.GroupMember, error)
    GetGroupMemberNum(ctx context.Context, groupID string) (memberNum int64, err error)
    DelGroupsMemberNum(groupID ...string) GroupCache

    //FindSortGroupMemberUserIDs(ctx context.Context, groupID string) ([]string, error)
    //FindSortJoinGroupIDs(ctx context.Context, userID string) ([]string, error)
```

```
    DelMaxGroupMemberVersion(groupIDs ...string) GroupCache
    DelMaxJoinGroupVersion(userIDs ...string) GroupCache
    FindMaxGroupMemberVersion(ctx context.Context, groupID string) (*model.VersionLog, error)
    BatchFindMaxGroupMemberVersion(ctx context.Context, groupIDs []string) ([]*model.VersionLog, error)
    FindMaxJoinGroupVersion(ctx context.Context, userID string) (*model.VersionLog, error)
}
```

pkg/common/storage/cache/msg.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

type MsgCache interface {
	SetSendMsgStatus(ctx context.Context, id string, status int32) error
	GetSendMsgStatus(ctx context.Context, id string) (int32, error)

	GetMessageBySeqs(ctx context.Context, conversationID string, seqs []int64) ([]*model.MsgInfoModel, error)
	DelMessageBySeqs(ctx context.Context, conversationID string, seqs []int64) error
	SetMessageBySeqs(ctx context.Context, conversationID string, msgs []*model.MsgInfoModel) error
}
```

pkg/common/storage/cache/online.go

```
package cache

import "context"

type OnlineCache interface {
    GetOnline(ctx context.Context, userID string) ([]int32, error)
    SetUserOnline(ctx context.Context, userID string, online, offline []int32) error
    GetAllOnlineUsers(ctx context.Context, cursor uint64) (map[string][]int32, uint64, error)
}
```

pkg/common/storage/cache/s3.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
	"context"
	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/s3"
)

type ObjectCache interface {
	BatchDeleter
	CloneObjectCache() ObjectCache
	GetName(ctx context.Context, engine string, name string) (*relationtb.Object, error)
	DelObjectName(engine string, names ...string) ObjectCache
}

type S3Cache interface {
	BatchDeleter
	GetKey(ctx context.Context, engine string, key string) (*s3.ObjectInfo, error)
	DelS3Key(engine string, keys ...string) S3Cache
}

// TODO integrating minio.Cache and MinioCache interfaces.
type MinioCache interface {
	BatchDeleter
	GetImageObjectKeyInfo(ctx context.Context, key string, fn func(ctx context.Context) (*MinioImageInfo, error)) (*Mi
	GetThumbnailKey(ctx context.Context, key string, format string, width int, height int, minioCache func(ctx context.
	DelObjectImageInfoKey(keys ...string) MinioCache
	DelImageThumbnailKey(key string, format string, width int, height int) MinioCache
}

type MinioImageInfo struct {
	IsImg  bool   `json:"isImg"`
	Width  int    `json:"width"`
	Height int    `json:"height"`
	Format string `json:"format"`
	Etag   string `json:"etag"`
}
```

pkg/common/storage/cache/seq_conversation.go

```go
package cache

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
)

type SeqConversationCache interface {
	Malloc(ctx context.Context, conversationID string, size int64) (int64, error)
	GetMaxSeq(ctx context.Context, conversationID string) (int64, error)
	SetMinSeq(ctx context.Context, conversationID string, seq int64) error
	GetMinSeq(ctx context.Context, conversationID string) (int64, error)
	GetMaxSeqs(ctx context.Context, conversationIDs []string) (map[string]int64, error)
	SetMinSeqs(ctx context.Context, seqs map[string]int64) error
	GetCacheMaxSeqWithTime(ctx context.Context, conversationIDs []string) (map[string]database.SeqTime, error)
	GetMaxSeqsWithTime(ctx context.Context, conversationIDs []string) (map[string]database.SeqTime, error)
	GetMaxSeqWithTime(ctx context.Context, conversationID string) (database.SeqTime, error)
}
```

pkg/common/storage/cache/seq_user.go

```
package cache

import "context"

type SeqUser interface {
    GetUserMaxSeq(ctx context.Context, conversationID string, userID string) (int64, error)
    SetUserMaxSeq(ctx context.Context, conversationID string, userID string, seq int64) error
    GetUserMinSeq(ctx context.Context, conversationID string, userID string) (int64, error)
    SetUserMinSeq(ctx context.Context, conversationID string, userID string, seq int64) error
    GetUserReadSeq(ctx context.Context, conversationID string, userID string) (int64, error)
    SetUserReadSeq(ctx context.Context, conversationID string, userID string, seq int64) error
    SetUserReadSeqToDB(ctx context.Context, conversationID string, userID string, seq int64) error
    SetUserMinSeqs(ctx context.Context, userID string, seqs map[string]int64) error
    SetUserReadSeqs(ctx context.Context, userID string, seqs map[string]int64) error
    GetUserReadSeqs(ctx context.Context, userID string, conversationIDs []string) (map[string]int64, error)
}
```

pkg/common/storage/cache/third.go

```go
package cache

import (
	"context"
)

type ThirdCache interface {
	SetFcmToken(ctx context.Context, account string, platformID int, fcmToken string, expireTime int64) (err error)
	GetFcmToken(ctx context.Context, account string, platformID int) (string, error)
	DelFcmToken(ctx context.Context, account string, platformID int) error
	IncrUserBadgeUnreadCountSum(ctx context.Context, userID string) (int, error)
	SetUserBadgeUnreadCountSum(ctx context.Context, userID string, value int) error
	GetUserBadgeUnreadCountSum(ctx context.Context, userID string) (int, error)
	SetGetuiToken(ctx context.Context, token string, expireTime int64) error
	GetGetuiToken(ctx context.Context) (string, error)
	SetGetuiTaskID(ctx context.Context, taskID string, expireTime int64) error
	GetGetuiTaskID(ctx context.Context) (string, error)
}
```

pkg/common/storage/cache/token.go

```go
package cache

import (
	"context"
)

type TokenModel interface {
	SetTokenFlag(ctx context.Context, userID string, platformID int, token string, flag int) error
	// SetTokenFlagEx set token and flag with expire time
	SetTokenFlagEx(ctx context.Context, userID string, platformID int, token string, flag int) error
	GetTokensWithoutError(ctx context.Context, userID string, platformID int) (map[string]int, error)
	HasTemporaryToken(ctx context.Context, userID string, platformID int, token string) error
	GetAllTokensWithoutError(ctx context.Context, userID string) (map[int]map[string]int, error)
	SetTokenMapByUidPid(ctx context.Context, userID string, platformID int, m map[string]int) error
	BatchSetTokenMapByUidPid(ctx context.Context, tokens map[string]map[string]any) error
	DeleteTokenByUidPid(ctx context.Context, userID string, platformID int, fields []string) error
	DeleteTokenByTokenMap(ctx context.Context, userID string, tokens map[int][]string) error
	DeleteAndSetTemporary(ctx context.Context, userID string, platformID int, fields []string) error
}
```

pkg/common/storage/cache/user.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cache

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

type UserCache interface {
	BatchDeleter
	CloneUserCache() UserCache
	GetUserInfo(ctx context.Context, userID string) (userInfo *model.User, err error)
	GetUsersInfo(ctx context.Context, userIDs []string) ([]*model.User, error)
	DelUsersInfo(userIDs ...string) UserCache
	GetUserGlobalRecvMsgOpt(ctx context.Context, userID string) (opt int, err error)
	DelUsersGlobalRecvMsgOpt(userIDs ...string) UserCache
	//GetUserStatus(ctx context.Context, userIDs []string) ([]*user.OnlineStatus, error)
	//SetUserStatus(ctx context.Context, userID string, status, platformID int32) error
}
```

pkg/common/storage/cache/cachekey

pkg/common/storage/cache/cachekey/black.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

const (
	BlackIDsKey = "BLACK_IDS:"
	IsBlackKey  = "IS_BLACK:" // local cache
)

func GetBlackIDsKey(ownerUserID string) string {
	return BlackIDsKey + ownerUserID

}

func GetIsBlackIDsKey(possibleBlackUserID, userID string) string {
	return IsBlackKey + userID + "-" + possibleBlackUserID
}
```

pkg/common/storage/cache/cachekey/client_config.go

```
package cachekey

const ClientConfig = "CLIENT_CONFIG"

func GetClientConfigKey(userID string) string {
    if userID == "" {
        return ClientConfig
    }
    return ClientConfig + ":" + userID
}
```

pkg/common/storage/cache/cachekey/conversation.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

const (
	ConversationKey                          = "CONVERSATION:"
	ConversationIDsKey                       = "CONVERSATION_IDS:"
	NotNotifyConversationIDsKey              = "NOT_NOTIFY_CONVERSATION_IDS:"
	PinnedConversationIDsKey                 = "PINNED_CONVERSATION_IDS:"
	ConversationIDsHashKey                   = "CONVERSATION_IDS_HASH:"
	ConversationHasReadSeqKey                = "CONVERSATION_HAS_READ_SEQ:"
	RecvMsgOptKey                            = "RECV_MSG_OPT:"
	SuperGroupRecvMsgNotNotifyUserIDsKey     = "SUPER_GROUP_RECV_MSG_NOT_NOTIFY_USER_IDS:"
	SuperGroupRecvMsgNotNotifyUserIDsHashKey = "SUPER_GROUP_RECV_MSG_NOT_NOTIFY_USER_IDS_HASH:"
	ConversationNotReceiveMessageUserIDsKey  = "CONVERSATION_NOT_RECEIVE_MESSAGE_USER_IDS:"
	ConversationUserMaxKey                   = "CONVERSATION_USER_MAX:"
)

func GetConversationKey(ownerUserID, conversationID string) string {
	return ConversationKey + ownerUserID + ":" + conversationID
}

func GetConversationIDsKey(ownerUserID string) string {
	return ConversationIDsKey + ownerUserID
}

func GetNotNotifyConversationIDsKey(ownerUserID string) string {
	return NotNotifyConversationIDsKey + ownerUserID
}

func GetPinnedConversationIDs(ownerUserID string) string {
	return PinnedConversationIDsKey + ownerUserID
}

func GetSuperGroupRecvNotNotifyUserIDsKey(groupID string) string {
	return SuperGroupRecvMsgNotNotifyUserIDsKey + groupID
}

func GetRecvMsgOptKey(ownerUserID, conversationID string) string {
	return RecvMsgOptKey + ownerUserID + ":" + conversationID
}

func GetSuperGroupRecvNotNotifyUserIDsHashKey(groupID string) string {
	return SuperGroupRecvMsgNotNotifyUserIDsHashKey + groupID
}

func GetConversationHasReadSeqKey(ownerUserID, conversationID string) string {
	return ConversationHasReadSeqKey + ownerUserID + ":" + conversationID
}

func GetConversationNotReceiveMessageUserIDsKey(conversationID string) string {
	return ConversationNotReceiveMessageUserIDsKey + conversationID
```

```
}

func GetUserConversationIDsHashKey(ownerUserID string) string {
	return ConversationIDsHashKey + ownerUserID
}

func GetConversationUserMaxVersionKey(userID string) string {
	return ConversationUserMaxKey + userID
}
```

pkg/common/storage/cache/cachekey/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey // import "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cachekey"
```

pkg/common/storage/cache/cachekey/friend.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

const (
	FriendIDsKey        = "FRIEND_IDS:"
	TwoWayFriendsIDsKey = "COMMON_FRIENDS_IDS:"
	FriendKey           = "FRIEND_INFO:"
	IsFriendKey         = "IS_FRIEND:" // local cache key
	//FriendSyncSortUserIDsKey = "FRIEND_SYNC_SORT_USER_IDS:"
	FriendMaxVersionKey = "FRIEND_MAX_VERSION:"
)

func GetFriendIDsKey(ownerUserID string) string {
	return FriendIDsKey + ownerUserID
}

func GetTwoWayFriendsIDsKey(ownerUserID string) string {
	return TwoWayFriendsIDsKey + ownerUserID
}

func GetFriendKey(ownerUserID, friendUserID string) string {
	return FriendKey + ownerUserID + "-" + friendUserID
}

func GetFriendMaxVersionKey(ownerUserID string) string {
	return FriendMaxVersionKey + ownerUserID
}

func GetIsFriendKey(possibleFriendUserID, userID string) string {
	return IsFriendKey + possibleFriendUserID + "-" + userID
}

//func GetFriendSyncSortUserIDsKey(ownerUserID string, count int) string {
//	return FriendSyncSortUserIDsKey + strconv.Itoa(count) + ":" + ownerUserID
//}
```

pkg/common/storage/cache/cachekey/group.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

import (
	"strconv"
	"time"
)

const (
	groupExpireTime             = time.Second * 60 * 60 * 12
	GroupInfoKey                = "GROUP_INFO:"
	GroupMemberIDsKey           = "GROUP_MEMBER_IDS:"
	GroupMembersHashKey         = "GROUP_MEMBERS_HASH2:"
	GroupMemberInfoKey          = "GROUP_MEMBER_INFO:"
	JoinedGroupsKey             = "JOIN_GROUPS_KEY:"
	GroupMemberNumKey           = "GROUP_MEMBER_NUM_CACHE:"
	GroupRoleLevelMemberIDsKey  = "GROUP_ROLE_LEVEL_MEMBER_IDS:"
	GroupAdminLevelMemberIDsKey = "GROUP_ADMIN_LEVEL_MEMBER_IDS:"
	GroupMemberMaxVersionKey    = "GROUP_MEMBER_MAX_VERSION:"
	GroupJoinMaxVersionKey      = "GROUP_JOIN_MAX_VERSION:"
)

func GetGroupInfoKey(groupID string) string {
	return GroupInfoKey + groupID
}

func GetJoinedGroupsKey(userID string) string {
	return JoinedGroupsKey + userID
}

func GetGroupMembersHashKey(groupID string) string {
	return GroupMembersHashKey + groupID
}

func GetGroupMemberIDsKey(groupID string) string {
	return GroupMemberIDsKey + groupID
}

func GetGroupMemberInfoKey(groupID, userID string) string {
	return GroupMemberInfoKey + groupID + "-" + userID
}

func GetGroupMemberNumKey(groupID string) string {
	return GroupMemberNumKey + groupID
}

func GetGroupRoleLevelMemberIDsKey(groupID string, roleLevel int32) string {
	return GroupRoleLevelMemberIDsKey + groupID + "-" + strconv.Itoa(int(roleLevel))
}

func GetGroupMemberMaxVersionKey(groupID string) string {
```

```
    return GroupMemberMaxVersionKey + groupID
}

func GetJoinGroupMaxVersionKey(userID string) string {
    return GroupJoinMaxVersionKey + userID
}
```

pkg/common/storage/cache/cachekey/msg.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

import (
	"strconv"
)

const (
	sendMsgFailedFlag = "SEND_MSG_FAILED_FLAG:"
	messageCache      = "MSG_CACHE:"
)

func GetMsgCacheKey(conversationID string, seq int64) string {
	return messageCache + conversationID + ":" + strconv.Itoa(int(seq))
}

func GetSendMsgKey(id string) string {
	return sendMsgFailedFlag + id
}
```

pkg/common/storage/cache/cachekey/online.go

```go
package cachekey

import (
	"strings"
	"time"
)

const (
	OnlineKey     = "ONLINE:"
	OnlineChannel = "online_change"
	OnlineExpire  = time.Hour / 2
)

func GetOnlineKey(userID string) string {
	return OnlineKey + userID
}

func GetOnlineKeyUserID(key string) string {
	return strings.TrimPrefix(key, OnlineKey)
}
```

pkg/common/storage/cache/cachekey/s3.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

import "strconv"

const (
	object         = "OBJECT:"
	s3             = "S3:"
	minioImageInfo = "MINIO:IMAGE:"
	minioThumbnail = "MINIO:THUMBNAIL:"
)

func GetObjectKey(engine string, name string) string {
	return object + engine + ":" + name
}

func GetS3Key(engine string, name string) string {
	return s3 + engine + ":" + name
}

func GetObjectImageInfoKey(key string) string {
	return minioImageInfo + key
}

func GetMinioImageThumbnailKey(key string, format string, width int, height int) string {
	return minioThumbnail + format + ":w" + strconv.Itoa(width) + ":h" + strconv.Itoa(height) + ":" + key
}
```

pkg/common/storage/cache/cachekey/seq.go

```go
package cachekey

const (
	MallocSeq        = "MALLOC_SEQ:"
	MallocMinSeqLock = "MALLOC_MIN_SEQ:"

	SeqUserMaxSeq  = "SEQ_USER_MAX:"
	SeqUserMinSeq  = "SEQ_USER_MIN:"
	SeqUserReadSeq = "SEQ_USER_READ:"
)

func GetMallocSeqKey(conversationID string) string {
	return MallocSeq + conversationID
}

func GetMallocMinSeqKey(conversationID string) string {
	return MallocMinSeqLock + conversationID
}

func GetSeqUserMaxSeqKey(conversationID string, userID string) string {
	return SeqUserMaxSeq + conversationID + ":" + userID
}

func GetSeqUserMinSeqKey(conversationID string, userID string) string {
	return SeqUserMinSeq + conversationID + ":" + userID
}

func GetSeqUserReadSeqKey(conversationID string, userID string) string {
	return SeqUserReadSeq + conversationID + ":" + userID
}
```

pkg/common/storage/cache/cachekey/third.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

import (
	"strconv"
)

const (
	getuiToken              = "GETUI_TOKEN"
	getuiTaskID             = "GETUI_TASK_ID"
	fmcToken                = "FCM_TOKEN:"
	userBadgeUnreadCountSum = "USER_BADGE_UNREAD_COUNT_SUM:"
)

func GetFcmAccountTokenKey(account string, platformID int) string {
	return fmcToken + account + ":" + strconv.Itoa(platformID)
}

func GetUserBadgeUnreadCountSumKey(userID string) string {
	return userBadgeUnreadCountSum + userID
}

func GetGetuiTokenKey() string {
	return getuiToken
}
func GetGetuiTaskIDKey() string {
	return getuiTaskID
}
```

pkg/common/storage/cache/cachekey/token.go

```go
package cachekey

import (
	"strings"

	"github.com/openimsdk/protocol/constant"
)

const (
	UidPidToken = "UID_PID_TOKEN_STATUS:"
)

func GetTokenKey(userID string, platformID int) string {
	return UidPidToken + userID + ":" + constant.PlatformIDToName(platformID)
}

func GetTemporaryTokenKey(userID string, platformID int, token string) string {
	return UidPidToken + ":TEMPORARY:" + userID + ":" + constant.PlatformIDToName(platformID) + ":" + token
}

func GetAllPlatformTokenKey(userID string) []string {
	res := make([]string, len(constant.PlatformID2Name))
	for k := range constant.PlatformID2Name {
		res[k-1] = GetTokenKey(userID, k)
	}
	return res
}

func GetPlatformIDByTokenKey(key string) int {
	splitKey := strings.Split(key, ":")
	platform := splitKey[len(splitKey)-1]
	return constant.PlatformNameToID(platform)
}
```

pkg/common/storage/cache/cachekey/user.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cachekey

const (
	UserInfoKey             = "USER_INFO:"
	UserGlobalRecvMsgOptKey = "USER_GLOBAL_RECV_MSG_OPT_KEY:"
)

func GetUserInfoKey(userID string) string {
	return UserInfoKey + userID
}

func GetUserGlobalRecvMsgOptKey(userID string) string {
	return UserGlobalRecvMsgOptKey + userID
}
```

pkg/common/storage/cache/mcache

pkg/common/storage/cache/mcache/minio.go

```go
package mcache

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/tools/s3/minio"
)

func NewMinioCache(cache database.Cache) minio.Cache {
	return &minioCache{
		cache:      cache,
		expireTime: time.Hour * 24 * 7,
	}
}

type minioCache struct {
	cache      database.Cache
	expireTime time.Duration
}

func (g *minioCache) getObjectImageInfoKey(key string) string {
	return cachekey.GetObjectImageInfoKey(key)
}

func (g *minioCache) getMinioImageThumbnailKey(key string, format string, width int, height int) string {
	return cachekey.GetMinioImageThumbnailKey(key, format, width, height)
}

func (g *minioCache) DelObjectImageInfoKey(ctx context.Context, keys ...string) error {
	ks := make([]string, 0, len(keys))
	for _, key := range keys {
		ks = append(ks, g.getObjectImageInfoKey(key))
	}
	return g.cache.Del(ctx, ks)
}

func (g *minioCache) DelImageThumbnailKey(ctx context.Context, key string, format string, width int, height int) err
	return g.cache.Del(ctx, []string{g.getMinioImageThumbnailKey(key, format, width, height)})
}

func (g *minioCache) GetImageObjectKeyInfo(ctx context.Context, key string, fn func(ctx context.Context) (*minio.Ima
	return getCache[*minio.ImageInfo](ctx, g.cache, g.getObjectImageInfoKey(key), g.expireTime, fn)
}

func (g *minioCache) GetThumbnailKey(ctx context.Context, key string, format string, width int, height int, minioCac
	return getCache[string](ctx, g.cache, g.getMinioImageThumbnailKey(key, format, width, height), g.expireTime, minioC
}
```

pkg/common/storage/cache/mcache/msg_cache.go

```
package mcache

import (
	"context"
	"strconv"
	"sync"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache/lru"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
)

var (
	memMsgCache     lru.LRU[string, *model.MsgInfoModel]
	initMemMsgCache sync.Once
)

func NewMsgCache(cache database.Cache, msgDocDatabase database.Msg) cache.MsgCache {
	initMemMsgCache.Do(func() {
		memMsgCache = lru.NewLazyLRU[string, *model.MsgInfoModel](1024*8, time.Hour, time.Second*10, localcache.EmptyTarg
	})
	return &msgCache{
		cache:          cache,
		msgDocDatabase: msgDocDatabase,
		memMsgCache:    memMsgCache,
	}
}

type msgCache struct {
	cache          database.Cache
	msgDocDatabase database.Msg
	memMsgCache    lru.LRU[string, *model.MsgInfoModel]
}

func (x *msgCache) getSendMsgKey(id string) string {
	return cachekey.GetSendMsgKey(id)
}

func (x *msgCache) SetSendMsgStatus(ctx context.Context, id string, status int32) error {
	return x.cache.Set(ctx, x.getSendMsgKey(id), strconv.Itoa(int(status)), time.Hour*24)
}

func (x *msgCache) GetSendMsgStatus(ctx context.Context, id string) (int32, error) {
	key := x.getSendMsgKey(id)
	res, err := x.cache.Get(ctx, []string{key})
	if err != nil {
		return 0, err
	}
	val, ok := res[key]
	if !ok {
		return 0, errs.Wrap(redis.Nil)
	}
	status, err := strconv.Atoi(val)
	if err != nil {
		return 0, errs.WrapMsg(err, "GetSendMsgStatus strconv.Atoi error", "val", val)
	}
	return int32(status), nil
```

```
}

func (x *msgCache) getMsgCacheKey(conversationID string, seq int64) string {
	return cachekey.GetMsgCacheKey(conversationID, seq)

}

func (x *msgCache) GetMessageBySeqs(ctx context.Context, conversationID string, seqs []int64) ([]*model.MsgInfoModel
	if len(seqs) == 0 {
		return nil, nil
	}
	keys := make([]string, 0, len(seqs))
	keySeq := make(map[string]int64, len(seqs))
	for _, seq := range seqs {
		key := x.getMsgCacheKey(conversationID, seq)
		keys = append(keys, key)
		keySeq[key] = seq
	}
	res, err := x.memMsgCache.GetBatch(keys, func(keys []string) (map[string]*model.MsgInfoModel, error) {
		findSeqs := make([]int64, 0, len(keys))
		for _, key := range keys {
			seq, ok := keySeq[key]
			if !ok {
				continue
			}
			findSeqs = append(findSeqs, seq)
		}
		res, err := x.msgDocDatabase.FindSeqs(ctx, conversationID, seqs)
		if err != nil {
			return nil, err
		}
		kv := make(map[string]*model.MsgInfoModel)
		for i := range res {
			msg := res[i]
			if msg == nil || msg.Msg == nil || msg.Msg.Seq <= 0 {
				continue
			}
			key := x.getMsgCacheKey(conversationID, msg.Msg.Seq)
			kv[key] = msg
		}
		return kv, nil
	})
	if err != nil {
		return nil, err
	}
	return datautil.Values(res), nil
}

func (x msgCache) DelMessageBySeqs(ctx context.Context, conversationID string, seqs []int64) error {
	if len(seqs) == 0 {
		return nil
	}
	for _, seq := range seqs {
		x.memMsgCache.Del(x.getMsgCacheKey(conversationID, seq))
	}
	return nil
}

func (x *msgCache) SetMessageBySeqs(ctx context.Context, conversationID string, msgs []*model.MsgInfoModel) error {
	for i := range msgs {
		msg := msgs[i]
		if msg == nil || msg.Msg == nil || msg.Msg.Seq <= 0 {
			continue
		}
		x.memMsgCache.Set(x.getMsgCacheKey(conversationID, msg.Msg.Seq), msg)
	}
```

```
    return nil
}
```

pkg/common/storage/cache/mcache/online.go

```
package mcache

import (
	"context"
	"sync"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
)

var (
	globalOnlineCache cache.OnlineCache
	globalOnlineOnce  sync.Once
)

func NewOnlineCache() cache.OnlineCache {
	globalOnlineOnce.Do(func() {
		globalOnlineCache = &onlineCache{
			user: make(map[string]map[int32]struct{}),
		}
	})
	return globalOnlineCache
}

type onlineCache struct {
	lock sync.RWMutex
	user map[string]map[int32]struct{}
}

func (x *onlineCache) GetOnline(ctx context.Context, userID string) ([]int32, error) {
	x.lock.RLock()
	defer x.lock.RUnlock()
	pSet, ok := x.user[userID]
	if !ok {
		return nil, nil
	}
	res := make([]int32, 0, len(pSet))
	for k := range pSet {
		res = append(res, k)
	}
	return res, nil
}

func (x *onlineCache) SetUserOnline(ctx context.Context, userID string, online, offline []int32) error {
	x.lock.Lock()
	defer x.lock.Unlock()
	pSet, ok := x.user[userID]
	if ok {
		for _, p := range offline {
			delete(pSet, p)
		}
	}
	if len(online) > 0 {
		if !ok {
			pSet = make(map[int32]struct{})
			x.user[userID] = pSet
		}
		for _, p := range online {
			pSet[p] = struct{}{}
		}
	}
	if len(pSet) == 0 {
		delete(x.user, userID)
	}
	return nil
```

```
}

func (x *onlineCache) GetAllOnlineUsers(ctx context.Context, cursor uint64) (map[string][]int32, uint64, error) {
    if cursor != 0 {
        return nil, 0, nil
    }
    x.lock.RLock()
    defer x.lock.RUnlock()
    res := make(map[string][]int32)
    for k, v := range x.user {
        pSet := make([]int32, 0, len(v))
        for p := range v {
            pSet = append(pSet, p)
        }
        res[k] = pSet
    }
    return res, 0, nil
}
```

pkg/common/storage/cache/mcache/seq_conversation.go

```
package mcache

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
)

func NewSeqConversationCache(sc database.SeqConversation) cache.SeqConversationCache {
	return &seqConversationCache{
		sc: sc,
	}
}

type seqConversationCache struct {
	sc database.SeqConversation
}

func (x *seqConversationCache) Malloc(ctx context.Context, conversationID string, size int64) (int64, error) {
	return x.sc.Malloc(ctx, conversationID, size)
}

func (x *seqConversationCache) SetMinSeq(ctx context.Context, conversationID string, seq int64) error {
	return x.sc.SetMinSeq(ctx, conversationID, seq)
}

func (x *seqConversationCache) GetMinSeq(ctx context.Context, conversationID string) (int64, error) {
	return x.sc.GetMinSeq(ctx, conversationID)
}

func (x *seqConversationCache) GetMaxSeqs(ctx context.Context, conversationIDs []string) (map[string]int64, error) {
	res := make(map[string]int64)
	for _, conversationID := range conversationIDs {
		seq, err := x.GetMinSeq(ctx, conversationID)
		if err != nil {
			return nil, err
		}
		res[conversationID] = seq
	}
	return res, nil
}

func (x *seqConversationCache) GetMaxSeqsWithTime(ctx context.Context, conversationIDs []string) (map[string]databas
	res := make(map[string]database.SeqTime)
	for _, conversationID := range conversationIDs {
		seq, err := x.GetMinSeq(ctx, conversationID)
		if err != nil {
			return nil, err
		}
		res[conversationID] = database.SeqTime{Seq: seq}
	}
	return res, nil
}

func (x *seqConversationCache) GetMaxSeq(ctx context.Context, conversationID string) (int64, error) {
	return x.sc.GetMaxSeq(ctx, conversationID)
}

func (x *seqConversationCache) GetMaxSeqWithTime(ctx context.Context, conversationID string) (database.SeqTime, erro
	seq, err := x.GetMinSeq(ctx, conversationID)
	if err != nil {
		return database.SeqTime{}, err
	}
```

```
	return database.SeqTime{Seq: seq}, nil
}

func (x *seqConversationCache) SetMinSeqs(ctx context.Context, seqs map[string]int64) error {
	for conversationID, seq := range seqs {
		if err := x.sc.SetMinSeq(ctx, conversationID, seq); err != nil {
			return err
		}
	}
	return nil
}

func (x *seqConversationCache) GetCacheMaxSeqWithTime(ctx context.Context, conversationIDs []string) (map[string]dat
	return x.GetMaxSeqsWithTime(ctx, conversationIDs)
}
```

pkg/common/storage/cache/mcache/third.go

```
package mcache

import (
	"context"
	"strconv"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/tools/errs"
	"github.com/redis/go-redis/v9"
)

func NewThirdCache(cache database.Cache) cache.ThirdCache {
	return &thirdCache{
		cache: cache,
	}
}

type thirdCache struct {
	cache database.Cache
}

func (c *thirdCache) getGetuiTokenKey() string {
	return cachekey.GetGetuiTokenKey()
}

func (c *thirdCache) getGetuiTaskIDKey() string {
	return cachekey.GetGetuiTaskIDKey()
}

func (c *thirdCache) getUserBadgeUnreadCountSumKey(userID string) string {
	return cachekey.GetUserBadgeUnreadCountSumKey(userID)
}

func (c *thirdCache) getFcmAccountTokenKey(account string, platformID int) string {
	return cachekey.GetFcmAccountTokenKey(account, platformID)
}

func (c *thirdCache) get(ctx context.Context, key string) (string, error) {
	res, err := c.cache.Get(ctx, []string{key})
	if err != nil {
		return "", err
	}
	if val, ok := res[key]; ok {
		return val, nil
	}
	return "", errs.Wrap(redis.Nil)
}

func (c *thirdCache) SetFcmToken(ctx context.Context, account string, platformID int, fcmToken string, expireTime in
	return errs.Wrap(c.cache.Set(ctx, c.getFcmAccountTokenKey(account, platformID), fcmToken, time.Duration(expireTime)
}

func (c *thirdCache) GetFcmToken(ctx context.Context, account string, platformID int) (string, error) {
	return c.get(ctx, c.getFcmAccountTokenKey(account, platformID))
}

func (c *thirdCache) DelFcmToken(ctx context.Context, account string, platformID int) error {
	return c.cache.Del(ctx, []string{c.getFcmAccountTokenKey(account, platformID)})
}

func (c *thirdCache) IncrUserBadgeUnreadCountSum(ctx context.Context, userID string) (int, error) {
```

```
	return c.cache.Incr(ctx, c.getUserBadgeUnreadCountSumKey(userID), 1)
}

func (c *thirdCache) SetUserBadgeUnreadCountSum(ctx context.Context, userID string, value int) error {
	return c.cache.Set(ctx, c.getUserBadgeUnreadCountSumKey(userID), strconv.Itoa(value), 0)
}

func (c *thirdCache) GetUserBadgeUnreadCountSum(ctx context.Context, userID string) (int, error) {
	str, err := c.get(ctx, c.getUserBadgeUnreadCountSumKey(userID))
	if err != nil {
		return 0, err
	}
	val, err := strconv.Atoi(str)
	if err != nil {
		return 0, errs.WrapMsg(err, "strconv.Atoi", "str", str)
	}
	return val, nil
}

func (c *thirdCache) SetGetuiToken(ctx context.Context, token string, expireTime int64) error {
	return c.cache.Set(ctx, c.getGetuiTokenKey(), token, time.Duration(expireTime)*time.Second)
}

func (c *thirdCache) GetGetuiToken(ctx context.Context) (string, error) {
	return c.get(ctx, c.getGetuiTokenKey())
}

func (c *thirdCache) SetGetuiTaskID(ctx context.Context, taskID string, expireTime int64) error {
	return c.cache.Set(ctx, c.getGetuiTaskIDKey(), taskID, time.Duration(expireTime)*time.Second)
}

func (c *thirdCache) GetGetuiTaskID(ctx context.Context) (string, error) {
	return c.get(ctx, c.getGetuiTaskIDKey())
}
```

pkg/common/storage/cache/mcache/token.go

```
package mcache

import (
	"context"
	"fmt"
	"strconv"
	"strings"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
)

func NewTokenCacheModel(cache database.Cache, accessExpire int64) cache.TokenModel {
	c := &tokenCache{cache: cache}
	c.accessExpire = c.getExpireTime(accessExpire)
	return c
}

type tokenCache struct {
	cache        database.Cache
	accessExpire time.Duration
}

func (x *tokenCache) getTokenKey(userID string, platformID int, token string) string {
	return cachekey.GetTokenKey(userID, platformID) + ":" + token
}

func (x *tokenCache) SetTokenFlag(ctx context.Context, userID string, platformID int, token string, flag int) error
	return x.cache.Set(ctx, x.getTokenKey(userID, platformID, token), strconv.Itoa(flag), x.accessExpire)
}

// SetTokenFlagEx set token and flag with expire time
func (x *tokenCache) SetTokenFlagEx(ctx context.Context, userID string, platformID int, token string, flag int) erro
	return x.SetTokenFlag(ctx, userID, platformID, token, flag)
}

func (x *tokenCache) GetTokensWithoutError(ctx context.Context, userID string, platformID int) (map[string]int, erro
	prefix := x.getTokenKey(userID, platformID, "")
	m, err := x.cache.Prefix(ctx, prefix)
	if err != nil {
		return nil, errs.Wrap(err)
	}
	mm := make(map[string]int)
	for k, v := range m {
		state, err := strconv.Atoi(v)
		if err != nil {
			log.ZError(ctx, "token value is not int", err, "value", v, "userID", userID, "platformID", platformID)
			continue
		}
		mm[strings.TrimPrefix(k, prefix)] = state
	}
	return mm, nil
}

func (x *tokenCache) HasTemporaryToken(ctx context.Context, userID string, platformID int, token string) error {
	key := cachekey.GetTemporaryTokenKey(userID, platformID, token)
	if _, err := x.cache.Get(ctx, []string{key}); err != nil {
		return err
	}
	return nil
```

```
}

func (x *tokenCache) GetAllTokensWithoutError(ctx context.Context, userID string) (map[int]map[string]int, error) {
	prefix := cachekey.UidPidToken + userID + ":"
	tokens, err := x.cache.Prefix(ctx, prefix)
	if err != nil {
		return nil, err
	}
	res := make(map[int]map[string]int)
	for key, flagStr := range tokens {
		flag, err := strconv.Atoi(flagStr)
		if err != nil {
			log.ZError(ctx, "token value is not int", err, "key", key, "value", flagStr, "userID", userID)
			continue
		}
		arr := strings.SplitN(strings.TrimPrefix(key, prefix), ":", 2)
		if len(arr) != 2 {
			log.ZError(ctx, "token value is not int", err, "key", key, "value", flagStr, "userID", userID)
			continue
		}
		platformID, err := strconv.Atoi(arr[0])
		if err != nil {
			log.ZError(ctx, "token value is not int", err, "key", key, "value", flagStr, "userID", userID)
			continue
		}
		token := arr[1]
		if token == "" {
			log.ZError(ctx, "token value is not int", err, "key", key, "value", flagStr, "userID", userID)
			continue
		}
		tk, ok := res[platformID]
		if !ok {
			tk = make(map[string]int)
			res[platformID] = tk
		}
		tk[token] = flag
	}
	return res, nil
}

func (x *tokenCache) SetTokenMapByUidPid(ctx context.Context, userID string, platformID int, m map[string]int) error
	for token, flag := range m {
		err := x.SetTokenFlag(ctx, userID, platformID, token, flag)
		if err != nil {
			return err
		}
	}
	return nil
}

func (x *tokenCache) BatchSetTokenMapByUidPid(ctx context.Context, tokens map[string]map[string]any) error {
	for prefix, tokenFlag := range tokens {
		for token, flag := range tokenFlag {
			flagStr := fmt.Sprintf("%v", flag)
			if err := x.cache.Set(ctx, prefix+":"+token, flagStr, x.accessExpire); err != nil {
				return err
			}
		}
	}
	return nil
}

func (x *tokenCache) DeleteTokenByUidPid(ctx context.Context, userID string, platformID int, fields []string) error
	keys := make([]string, 0, len(fields))
	for _, token := range fields {
		keys = append(keys, x.getTokenKey(userID, platformID, token))
```

```
	}
	return x.cache.Del(ctx, keys)
}

func (x *tokenCache) getExpireTime(t int64) time.Duration {
	return time.Hour * 24 * time.Duration(t)
}

func (x *tokenCache) DeleteTokenByTokenMap(ctx context.Context, userID string, tokens map[int][]string) error {
	keys := make([]string, 0, len(tokens))
	for platformID, ts := range tokens {
		for _, t := range ts {
			keys = append(keys, x.getTokenKey(userID, platformID, t))
		}
	}
	return x.cache.Del(ctx, keys)
}

func (x *tokenCache) DeleteAndSetTemporary(ctx context.Context, userID string, platformID int, fields []string) erro
	keys := make([]string, 0, len(fields))
	for _, f := range fields {
		keys = append(keys, x.getTokenKey(userID, platformID, f))
	}
	if err := x.cache.Del(ctx, keys); err != nil {
		return err
	}

	for _, f := range fields {
		k := cachekey.GetTemporaryTokenKey(userID, platformID, f)
		if err := x.cache.Set(ctx, k, "", time.Minute*5); err != nil {
			return errs.Wrap(err)
		}
	}

	return nil
}
```

pkg/common/storage/cache/mcache/tools.go

```go
package mcache

import (
	"context"
	"encoding/json"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/tools/log"
)

func getCache[V any](ctx context.Context, cache database.Cache, key string, expireTime time.Duration, fn func(ctx co
	getDB := func() (V, bool, error) {
		res, err := cache.Get(ctx, []string{key})
		if err != nil {
			var val V
			return val, false, err
		}
		var val V
		if str, ok := res[key]; ok {
			if json.Unmarshal([]byte(str), &val) != nil {
				return val, false, err
			}
			return val, true, nil
		}
		return val, false, nil
	}
	dbVal, ok, err := getDB()
	if err != nil {
		return dbVal, err
	}
	if ok {
		return dbVal, nil
	}
	lockValue, err := cache.Lock(ctx, key, time.Minute)
	if err != nil {
		return dbVal, err
	}
	defer func() {
		if err := cache.Unlock(ctx, key, lockValue); err != nil {
			log.ZError(ctx, "unlock cache key", err, "key", key, "value", lockValue)
		}
	}()
	dbVal, ok, err = getDB()
	if err != nil {
		return dbVal, err
	}
	if ok {
		return dbVal, nil
	}
	val, err := fn(ctx)
	if err != nil {
		return val, err
	}
	data, err := json.Marshal(val)
	if err != nil {
		return val, err
	}
	if err := cache.Set(ctx, key, string(data), expireTime); err != nil {
		return val, err
	}
	return val, nil
}
```

pkg/common/storage/cache/redis

pkg/common/storage/cache/redis/batch.go

```
package redis

import (
	"context"
	"encoding/json"
	"time"

	"github.com/dtm-labs/rockscache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

// GetRocksCacheOptions returns the default configuration options for RocksCache.
func GetRocksCacheOptions() *rockscache.Options {
	opts := rockscache.NewDefaultOptions()
	opts.LockExpire = rocksCacheTimeout
	opts.WaitReplicasTimeout = rocksCacheTimeout
	opts.StrongConsistency = true
	opts.RandomExpireAdjustment = 0.2

	return &opts
}

func newRocksCacheClient(rdb redis.UniversalClient) *rocksCacheClient {
	if rdb == nil {
		return &rocksCacheClient{}
	}
	rc := &rocksCacheClient{
		rdb:    rdb,
		client: rockscache.NewClient(rdb, *GetRocksCacheOptions()),
	}
	return rc
}

type rocksCacheClient struct {
	rdb    redis.UniversalClient
	client *rockscache.Client
}

func (x *rocksCacheClient) GetClient() *rockscache.Client {
	return x.client
}

func (x *rocksCacheClient) Disable() bool {
	return x.client == nil
}

func (x *rocksCacheClient) GetRedis() redis.UniversalClient {
	return x.rdb
}

func (x *rocksCacheClient) GetBatchDeleter(topics ...string) cache.BatchDeleter {
	return NewBatchDeleterRedis(x, topics)
}

func batchGetCache2[K comparable, V any](ctx context.Context, rcClient *rocksCacheClient, expire time.Duration, ids
	if len(ids) == 0 {
		return nil, nil
	}
```

```
	if rcClient.Disable() {
		return fn(ctx, ids)
	}
	findKeys := make([]string, 0, len(ids))
	keyId := make(map[string]K)
	for _, id := range ids {
		key := idKey(id)
		if _, ok := keyId[key]; ok {
			continue
		}
		keyId[key] = id
		findKeys = append(findKeys, key)
	}
	slotKeys, err := groupKeysBySlot(ctx, rcClient.GetRedis(), findKeys)
	if err != nil {
		return nil, err
	}
	result := make([]*V, 0, len(findKeys))
	for _, keys := range slotKeys {
		indexCache, err := rcClient.GetClient().FetchBatch2(ctx, keys, expire, func(idx []int) (map[int]string, error) {
			queryIds := make([]K, 0, len(idx))
			idIndex := make(map[K]int)
			for _, index := range idx {
				id := keyId[keys[index]]
				idIndex[id] = index
				queryIds = append(queryIds, id)
			}
			values, err := fn(ctx, queryIds)
			if err != nil {
				log.ZError(ctx, "batchGetCache query database failed", err, "keys", keys, "queryIds", queryIds)
				return nil, err
			}
			if len(values) == 0 {
				return map[int]string{}, nil
			}
			cacheIndex := make(map[int]string)
			for _, value := range values {
				id := vId(value)
				index, ok := idIndex[id]
				if !ok {
					continue
				}
				bs, err := json.Marshal(value)
				if err != nil {
					log.ZError(ctx, "marshal failed", err)
					return nil, err
				}
				cacheIndex[index] = string(bs)
			}
			return cacheIndex, nil
		})
		if err != nil {
			return nil, errs.WrapMsg(err, "FetchBatch2 failed")
		}
		for index, data := range indexCache {
			if data == "" {
				continue
			}
			var value V
			if err := json.Unmarshal([]byte(data), &value); err != nil {
				return nil, errs.WrapMsg(err, "Unmarshal failed")
			}
			if cb, ok := any(&value).(BatchCacheCallback[K]); ok {
				cb.BatchCache(keyId[keys[index]])
			}
			result = append(result, &value)
```

```
        }
    }
    return result, nil
}

type BatchCacheCallback[K comparable] interface {
    BatchCache(id K)
}
```

pkg/common/storage/cache/redis/batch_handler.go

```go
package redis

import (
	"context"
	"encoding/json"
	"fmt"
	"time"

	"github.com/dtm-labs/rockscache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
)

const (
	rocksCacheTimeout = 11 * time.Second
)

// BatchDeleterRedis is a concrete implementation of the BatchDeleter interface based on Redis and RocksCache.
type BatchDeleterRedis struct {
	redisClient    redis.UniversalClient
	keys           []string
	rocksClient    *rockscache.Client
	redisPubTopics []string
}

// NewBatchDeleterRedis creates a new BatchDeleterRedis instance.
func NewBatchDeleterRedis(rcClient *rocksCacheClient, redisPubTopics []string) *BatchDeleterRedis {
	return &BatchDeleterRedis{
		redisClient:    rcClient.GetRedis(),
		rocksClient:    rcClient.GetClient(),
		redisPubTopics: redisPubTopics,
	}
}

// ExecDelWithKeys directly takes keys for batch deletion and publishes deletion information.
func (c *BatchDeleterRedis) ExecDelWithKeys(ctx context.Context, keys []string) error {
	distinctKeys := datautil.Distinct(keys)
	return c.execDel(ctx, distinctKeys)
}

// ChainExecDel is used for chain calls for batch deletion. It must call Clone to prevent memory pollution.
func (c *BatchDeleterRedis) ChainExecDel(ctx context.Context) error {
	distinctKeys := datautil.Distinct(c.keys)
	return c.execDel(ctx, distinctKeys)
}

// execDel performs batch deletion and publishes the keys that have been deleted to update the local cache informati
func (c *BatchDeleterRedis) execDel(ctx context.Context, keys []string) error {
	if len(keys) > 0 {
		log.ZDebug(ctx, "delete cache", "topic", c.redisPubTopics, "keys", keys)
		// Batch delete keys
		err := ProcessKeysBySlot(ctx, c.redisClient, keys, func(ctx context.Context, slot int64, keys []string) error {
			return c.rocksClient.TagAsDeletedBatch2(ctx, keys)
		})
		if err != nil {
			return err
		}
		// Publish the keys that have been deleted to Redis to update the local cache information of other nodes
		if len(c.redisPubTopics) > 0 && len(keys) > 0 {
			keysByTopic := localcache.GetPublishKeysByTopic(c.redisPubTopics, keys)
```

```
    for topic, keys := range keysByTopic {
        if len(keys) > 0 {
            data, err := json.Marshal(keys)
            if err != nil {
                log.ZWarn(ctx, "keys json marshal failed", err, "topic", topic, "keys", keys)
            } else {
                if err := c.redisClient.Publish(ctx, topic, string(data)).Err(); err != nil {
                    log.ZWarn(ctx, "redis publish cache delete error", err, "topic", topic, "keys", keys)
                }
            }
        }
    }
  }
  return nil
}

// Clone creates a copy of BatchDeleterRedis for chain calls to prevent memory pollution.
func (c *BatchDeleterRedis) Clone() cache.BatchDeleter {
  return &BatchDeleterRedis{
    redisClient:    c.redisClient,
    keys:           c.keys,
    rocksClient:    c.rocksClient,
    redisPubTopics: c.redisPubTopics,
  }
}

// AddKeys adds keys to be deleted.
func (c *BatchDeleterRedis) AddKeys(keys ...string) {
  c.keys = append(c.keys, keys...)
}

type disableBatchDeleter struct{}

func (x disableBatchDeleter) ChainExecDel(ctx context.Context) error {
  return nil
}

func (x disableBatchDeleter) ExecDelWithKeys(ctx context.Context, keys []string) error {
  return nil
}

func (x disableBatchDeleter) Clone() cache.BatchDeleter {
  return x
}

func (x disableBatchDeleter) AddKeys(keys ...string) {}

func getCache[T any](ctx context.Context, rcClient *rocksCacheClient, key string, expire time.Duration, fn func(ctx
  if rcClient.Disable() {
    return fn(ctx)
  }
  var t T
  var write bool
  v, err := rcClient.GetClient().Fetch2(ctx, key, expire, func() (s string, err error) {
    t, err = fn(ctx)
    if err != nil {
      //log.ZError(ctx, "getCache query database failed", err, "key", key)
      return "", err
    }
    bs, err := json.Marshal(t)
    if err != nil {
      return "", errs.WrapMsg(err, "marshal failed")
    }
    write = true
```

```
        return string(bs), nil
    })
    if err != nil {
        return t, errs.Wrap(err)
    }
    if write {
        return t, nil
    }
    if v == "" {
        return t, errs.ErrRecordNotFound.WrapMsg("cache is not found")
    }
    err = json.Unmarshal([]byte(v), &t)
    if err != nil {
        errInfo := fmt.Sprintf("cache json.Unmarshal failed, key:%s, value:%s, expire:%s", key, v, expire)
        return t, errs.WrapMsg(err, errInfo)
    }

    return t, nil
}
```

pkg/common/storage/cache/redis/batch_test.go

```
package redis

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"testing"
)

func TestName(t *testing.T) {
	//var rocks rockscache.Client
	//rdb := getRocksCacheRedisClient(&rocks)
	//t.Log(rdb == nil)

	ctx := context.Background()
	rdb, err := redisutil.NewRedisClient(ctx, (&config.Redis{
		Address:  []string{"172.16.8.48:16379"},
		Password: "openIM123",
		DB:       3,
	}).Build())
	if err != nil {
		panic(err)
	}
	mgocli, err := mongoutil.NewMongoDB(ctx, (&config.Mongo{
		Address:     []string{"172.16.8.48:37017"},
		Database:    "openim_v3",
		Username:    "openIM",
		Password:    "openIM123",
		MaxPoolSize: 100,
		MaxRetry:    1,
	}).Build())
	if err != nil {
		panic(err)
	}
	//userMgo, err := mgo.NewUserMongo(mgocli.GetDB())
	//if err != nil {
	//	panic(err)
	//}
	//rock := rockscache.NewClient(rdb, rockscache.NewDefaultOptions())
	mgoSeqUser, err := mgo.NewSeqUserMongo(mgocli.GetDB())
	if err != nil {
		panic(err)
	}
	seqUser := NewSeqUserCacheRedis(rdb, mgoSeqUser)

	res, err := seqUser.GetUserReadSeqs(ctx, "2110910952", []string{"sg_2920732023", "sg_345762580"})
	if err != nil {
		panic(err)
	}

	t.Log(res)

}
```

pkg/common/storage/cache/redis/black.go

```
package redis

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/redis/go-redis/v9"
)

const (
	blackExpireTime = time.Second * 60 * 60 * 12
)

type BlackCacheRedis struct {
	cache.BatchDeleter
	expireTime time.Duration
	rcClient   *rocksCacheClient
	blackDB    database.Black
}

func NewBlackCacheRedis(rdb redis.UniversalClient, localCache *config.LocalCache, blackDB database.Black) cache.Blac
	rc := newRocksCacheClient(rdb)
	return &BlackCacheRedis{
		BatchDeleter: rc.GetBatchDeleter(localCache.Friend.Topic),
		expireTime:   blackExpireTime,
		rcClient:     rc,
		blackDB:      blackDB,
	}
}

func (b *BlackCacheRedis) CloneBlackCache() cache.BlackCache {
	return &BlackCacheRedis{
		BatchDeleter: b.BatchDeleter.Clone(),
		expireTime:   b.expireTime,
		rcClient:     b.rcClient,
		blackDB:      b.blackDB,
	}
}

func (b *BlackCacheRedis) getBlackIDsKey(ownerUserID string) string {
	return cachekey.GetBlackIDsKey(ownerUserID)
}

func (b *BlackCacheRedis) GetBlackIDs(ctx context.Context, userID string) (blackIDs []string, err error) {
	return getCache(
		ctx,
		b.rcClient,
		b.getBlackIDsKey(userID),
		b.expireTime,
		func(ctx context.Context) ([]string, error) {
			return b.blackDB.FindBlackUserIDs(ctx, userID)
		},
	)
}

func (b *BlackCacheRedis) DelBlackIDs(_ context.Context, userID string) cache.BlackCache {
	cache := b.CloneBlackCache()
	cache.AddKeys(b.getBlackIDsKey(userID))

	return cache
```

```
}
```

pkg/common/storage/cache/redis/client_config.go

```
package redis

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/redis/go-redis/v9"
)

func NewClientConfigCache(rdb redis.UniversalClient, mgo database.ClientConfig) cache.ClientConfigCache {
	rc := newRocksCacheClient(rdb)
	return &ClientConfigCache{
		mgo:      mgo,
		rcClient: rc,
		delete:   rc.GetBatchDeleter(),
	}
}

type ClientConfigCache struct {
	mgo      database.ClientConfig
	rcClient *rocksCacheClient
	delete   cache.BatchDeleter
}

func (x *ClientConfigCache) getExpireTime(userID string) time.Duration {
	if userID == "" {
		return time.Hour * 24
	} else {
		return time.Hour
	}
}

func (x *ClientConfigCache) getClientConfigKey(userID string) string {
	return cachekey.GetClientConfigKey(userID)
}

func (x *ClientConfigCache) GetConfig(ctx context.Context, userID string) (map[string]string, error) {
	return getCache(ctx, x.rcClient, x.getClientConfigKey(userID), x.getExpireTime(userID), func(ctx context.Context) (
		return x.mgo.Get(ctx, userID)
	})
}

func (x *ClientConfigCache) DeleteUserCache(ctx context.Context, userIDs []string) error {
	keys := make([]string, 0, len(userIDs))
	for _, userID := range userIDs {
		keys = append(keys, x.getClientConfigKey(userID))
	}
	return x.delete.ExecDelWithKeys(ctx, keys)
}

func (x *ClientConfigCache) GetUserConfig(ctx context.Context, userID string) (map[string]string, error) {
	config, err := x.GetConfig(ctx, "")
	if err != nil {
		return nil, err
	}
	if userID != "" {
		userConfig, err := x.GetConfig(ctx, userID)
		if err != nil {
			return nil, err
		}
		for k, v := range userConfig {
```

```
            config[k] = v
        }
    }
    return config, nil
}
```

pkg/common/storage/cache/redis/conversation.go

```
package redis

import (
	"context"
	"math/big"
	"strings"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/encrypt"
	"github.com/redis/go-redis/v9"
)

const (
	conversationExpireTime = time.Second * 60 * 60 * 12
)

func NewConversationRedis(rdb redis.UniversalClient, localCache *config.LocalCache, db database.Conversation) cache.
	rc := newRocksCacheClient(rdb)
	return &ConversationRedisCache{
		BatchDeleter:   rc.GetBatchDeleter(localCache.Conversation.Topic),
		rcClient:       rc,
		conversationDB: db,
		expireTime:     conversationExpireTime,
	}
}

type ConversationRedisCache struct {
	cache.BatchDeleter
	rcClient       *rocksCacheClient
	conversationDB database.Conversation
	expireTime     time.Duration
}

func (c *ConversationRedisCache) CloneConversationCache() cache.ConversationCache {
	return &ConversationRedisCache{
		BatchDeleter:   c.BatchDeleter.Clone(),
		rcClient:       c.rcClient,
		conversationDB: c.conversationDB,
		expireTime:     c.expireTime,
	}
}

func (c *ConversationRedisCache) getConversationKey(ownerUserID, conversationID string) string {
	return cachekey.GetConversationKey(ownerUserID, conversationID)
}

func (c *ConversationRedisCache) getConversationIDsKey(ownerUserID string) string {
	return cachekey.GetConversationIDsKey(ownerUserID)
}

func (c *ConversationRedisCache) getNotNotifyConversationIDsKey(ownerUserID string) string {
	return cachekey.GetNotNotifyConversationIDsKey(ownerUserID)
}

func (c *ConversationRedisCache) getPinnedConversationIDsKey(ownerUserID string) string {
	return cachekey.GetPinnedConversationIDs(ownerUserID)
}
```

```go
func (c *ConversationRedisCache) getSuperGroupRecvNotNotifyUserIDsKey(groupID string) string {
	return cachekey.GetSuperGroupRecvNotNotifyUserIDsKey(groupID)
}

func (c *ConversationRedisCache) getRecvMsgOptKey(ownerUserID, conversationID string) string {
	return cachekey.GetRecvMsgOptKey(ownerUserID, conversationID)
}

func (c *ConversationRedisCache) getSuperGroupRecvNotNotifyUserIDsHashKey(groupID string) string {
	return cachekey.GetSuperGroupRecvNotNotifyUserIDsHashKey(groupID)
}

func (c *ConversationRedisCache) getConversationHasReadSeqKey(ownerUserID, conversationID string) string {
	return cachekey.GetConversationHasReadSeqKey(ownerUserID, conversationID)
}

func (c *ConversationRedisCache) getConversationNotReceiveMessageUserIDsKey(conversationID string) string {
	return cachekey.GetConversationNotReceiveMessageUserIDsKey(conversationID)
}

func (c *ConversationRedisCache) getUserConversationIDsHashKey(ownerUserID string) string {
	return cachekey.GetUserConversationIDsHashKey(ownerUserID)
}

func (c *ConversationRedisCache) getConversationUserMaxVersionKey(ownerUserID string) string {
	return cachekey.GetConversationUserMaxVersionKey(ownerUserID)
}

func (c *ConversationRedisCache) GetUserConversationIDs(ctx context.Context, ownerUserID string) ([]string, error) {
	return getCache(ctx, c.rcClient, c.getConversationIDsKey(ownerUserID), c.expireTime, func(ctx context.Context) ([]s
		return c.conversationDB.FindUserIDAllConversationID(ctx, ownerUserID)
	})
}

func (c *ConversationRedisCache) GetUserNotNotifyConversationIDs(ctx context.Context, userID string) ([]string, erro
	return getCache(ctx, c.rcClient, c.getNotNotifyConversationIDsKey(userID), c.expireTime, func(ctx context.Context)
		return c.conversationDB.FindUserIDAllNotNotifyConversationID(ctx, userID)
	})
}

func (c *ConversationRedisCache) GetPinnedConversationIDs(ctx context.Context, userID string) ([]string, error) {
	return getCache(ctx, c.rcClient, c.getPinnedConversationIDsKey(userID), c.expireTime, func(ctx context.Context) ([]
		return c.conversationDB.FindUserIDAllPinnedConversationID(ctx, userID)
	})
}

func (c *ConversationRedisCache) DelConversationIDs(userIDs ...string) cache.ConversationCache {
	keys := make([]string, 0, len(userIDs))
	for _, userID := range userIDs {
		keys = append(keys, c.getConversationIDsKey(userID))
	}
	cache := c.CloneConversationCache()
	cache.AddKeys(keys...)

	return cache
}

func (c *ConversationRedisCache) GetUserConversationIDsHash(ctx context.Context, ownerUserID string) (hash uint64, e
	return getCache(
		ctx,
		c.rcClient,
		c.getUserConversationIDsHashKey(ownerUserID),
		c.expireTime,
		func(ctx context.Context) (uint64, error) {
			conversationIDs, err := c.GetUserConversationIDs(ctx, ownerUserID)
			if err != nil {
```

```
        return 0, err
    }
    datautil.Sort(conversationIDs, true)
    bi := big.NewInt(0)
    bi.SetString(encrypt.Md5(strings.Join(conversationIDs, ";"))[0:8], 16)
    return bi.Uint64(), nil
  },
 )
}

func (c *ConversationRedisCache) DelUserConversationIDsHash(ownerUserIDs ...string) cache.ConversationCache {
 keys := make([]string, 0, len(ownerUserIDs))
 for _, ownerUserID := range ownerUserIDs {
  keys = append(keys, c.getUserConversationIDsHashKey(ownerUserID))
 }
 cache := c.CloneConversationCache()
 cache.AddKeys(keys...)

 return cache
}

func (c *ConversationRedisCache) GetConversation(ctx context.Context, ownerUserID, conversationID string) (*model.Co
 return getCache(ctx, c.rcClient, c.getConversationKey(ownerUserID, conversationID), c.expireTime, func(ctx context.
  return c.conversationDB.Take(ctx, ownerUserID, conversationID)
 })
}

func (c *ConversationRedisCache) DelConversations(ownerUserID string, conversationIDs ...string) cache.ConversationC
 keys := make([]string, 0, len(conversationIDs))
 for _, conversationID := range conversationIDs {
  keys = append(keys, c.getConversationKey(ownerUserID, conversationID))
 }
 cache := c.CloneConversationCache()
 cache.AddKeys(keys...)

 return cache
}

func (c *ConversationRedisCache) GetConversations(ctx context.Context, ownerUserID string, conversationIDs []string)
 return batchGetCache2(ctx, c.rcClient, c.expireTime, conversationIDs, func(conversationID string) string {
  return c.getConversationKey(ownerUserID, conversationID)
 }, func(conversation *model.Conversation) string {
  return conversation.ConversationID
 }, func(ctx context.Context, conversationIDs []string) ([]*model.Conversation, error) {
  return c.conversationDB.Find(ctx, ownerUserID, conversationIDs)
 })
}

func (c *ConversationRedisCache) GetUserAllConversations(ctx context.Context, ownerUserID string) ([]*model.Conversa
 conversationIDs, err := c.GetUserConversationIDs(ctx, ownerUserID)
 if err != nil {
  return nil, err
 }
 return c.GetConversations(ctx, ownerUserID, conversationIDs)
}

func (c *ConversationRedisCache) GetUserRecvMsgOpt(ctx context.Context, ownerUserID, conversationID string) (opt int
 return getCache(ctx, c.rcClient, c.getRecvMsgOptKey(ownerUserID, conversationID), c.expireTime, func(ctx context.Co
  return c.conversationDB.GetUserRecvMsgOpt(ctx, ownerUserID, conversationID)
 })
}

func (c *ConversationRedisCache) DelUsersConversation(conversationID string, ownerUserIDs ...string) cache.Conversat
 keys := make([]string, 0, len(ownerUserIDs))
 for _, ownerUserID := range ownerUserIDs {
  keys = append(keys, c.getConversationKey(ownerUserID, conversationID))
```

```
	}
	cache := c.CloneConversationCache()
	cache.AddKeys(keys...)

	return cache
}

func (c *ConversationRedisCache) DelUserRecvMsgOpt(ownerUserID, conversationID string) cache.ConversationCache {
	cache := c.CloneConversationCache()
	cache.AddKeys(c.getRecvMsgOptKey(ownerUserID, conversationID))

	return cache
}

func (c *ConversationRedisCache) DelSuperGroupRecvMsgNotNotifyUserIDs(groupID string) cache.ConversationCache {
	cache := c.CloneConversationCache()
	cache.AddKeys(c.getSuperGroupRecvNotNotifyUserIDsKey(groupID))

	return cache
}

func (c *ConversationRedisCache) DelSuperGroupRecvMsgNotNotifyUserIDsHash(groupID string) cache.ConversationCache {
	cache := c.CloneConversationCache()
	cache.AddKeys(c.getSuperGroupRecvNotNotifyUserIDsHashKey(groupID))

	return cache
}

func (c *ConversationRedisCache) DelUserAllHasReadSeqs(ownerUserID string, conversationIDs ...string) cache.Conversa
	cache := c.CloneConversationCache()
	for _, conversationID := range conversationIDs {
		cache.AddKeys(c.getConversationHasReadSeqKey(ownerUserID, conversationID))
	}

	return cache
}

func (c *ConversationRedisCache) GetConversationNotReceiveMessageUserIDs(ctx context.Context, conversationID string)
	return getCache(ctx, c.rcClient, c.getConversationNotReceiveMessageUserIDsKey(conversationID), c.expireTime, func(c
		return c.conversationDB.GetConversationNotReceiveMessageUserIDs(ctx, conversationID)
	})
}

func (c *ConversationRedisCache) DelConversationNotReceiveMessageUserIDs(conversationIDs ...string) cache.Conversati
	cache := c.CloneConversationCache()
	for _, conversationID := range conversationIDs {
		cache.AddKeys(c.getConversationNotReceiveMessageUserIDsKey(conversationID))
	}
	return cache
}

func (c *ConversationRedisCache) DelConversationNotNotifyMessageUserIDs(userIDs ...string) cache.ConversationCache {
	cache := c.CloneConversationCache()
	for _, userID := range userIDs {
		cache.AddKeys(c.getNotNotifyConversationIDsKey(userID))
	}
	return cache
}

func (c *ConversationRedisCache) DelUserPinnedConversations(userIDs ...string) cache.ConversationCache {
	cache := c.CloneConversationCache()
	for _, userID := range userIDs {
		cache.AddKeys(c.getPinnedConversationIDsKey(userID))
	}
	return cache
}
```

```
func (c *ConversationRedisCache) DelConversationVersionUserIDs(userIDs ...string) cache.ConversationCache {
    cache := c.CloneConversationCache()
    for _, userID := range userIDs {
        cache.AddKeys(c.getConversationUserMaxVersionKey(userID))
    }
    return cache
}

func (c *ConversationRedisCache) FindMaxConversationUserVersion(ctx context.Context, userID string) (*model.VersionL
    return getCache(ctx, c.rcClient, c.getConversationUserMaxVersionKey(userID), c.expireTime, func(ctx context.Context
        return c.conversationDB.FindConversationUserVersion(ctx, userID, 0, 0)
    })
}
```

pkg/common/storage/cache/redis/friend.go

```
package redis

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
)

const (
	friendExpireTime = time.Second * 60 * 60 * 12
)

// FriendCacheRedis is an implementation of the FriendCache interface using Redis.
type FriendCacheRedis struct {
	cache.BatchDeleter
	friendDB   database.Friend
	expireTime time.Duration
	rcClient   *rocksCacheClient
	syncCount  int
}

// NewFriendCacheRedis creates a new instance of FriendCacheRedis.
func NewFriendCacheRedis(rdb redis.UniversalClient, localCache *config.LocalCache, friendDB database.Friend) cache.F
	rc := newRocksCacheClient(rdb)
	return &FriendCacheRedis{
		BatchDeleter: rc.GetBatchDeleter(localCache.Friend.Topic),
		friendDB:     friendDB,
		expireTime:   friendExpireTime,
		rcClient:     rc,
	}
}

func (f *FriendCacheRedis) CloneFriendCache() cache.FriendCache {
	return &FriendCacheRedis{
		BatchDeleter: f.BatchDeleter.Clone(),
		friendDB:     f.friendDB,
		expireTime:   f.expireTime,
		rcClient:     f.rcClient,
	}
}

// getFriendIDsKey returns the key for storing friend IDs in the cache.
func (f *FriendCacheRedis) getFriendIDsKey(ownerUserID string) string {
	return cachekey.GetFriendIDsKey(ownerUserID)
}

func (f *FriendCacheRedis) getFriendMaxVersionKey(ownerUserID string) string {
	return cachekey.GetFriendMaxVersionKey(ownerUserID)
}

// getTwoWayFriendsIDsKey returns the key for storing two-way friend IDs in the cache.
func (f *FriendCacheRedis) getTwoWayFriendsIDsKey(ownerUserID string) string {
	return cachekey.GetTwoWayFriendsIDsKey(ownerUserID)
}

// getFriendKey returns the key for storing friend info in the cache.
func (f *FriendCacheRedis) getFriendKey(ownerUserID, friendUserID string) string {
```

```
	return cachekey.GetFriendKey(ownerUserID, friendUserID)
}

// GetFriendIDs retrieves friend IDs from the cache or the database if not found.
func (f *FriendCacheRedis) GetFriendIDs(ctx context.Context, ownerUserID string) (friendIDs []string, err error) {
	return getCache(ctx, f.rcClient, f.getFriendIDsKey(ownerUserID), f.expireTime, func(ctx context.Context) ([]string,
		return f.friendDB.FindFriendUserIDs(ctx, ownerUserID)
	})
}

// DelFriendIDs deletes friend IDs from the cache.
func (f *FriendCacheRedis) DelFriendIDs(ownerUserIDs ...string) cache.FriendCache {
	newFriendCache := f.CloneFriendCache()
	keys := make([]string, 0, len(ownerUserIDs))
	for _, userID := range ownerUserIDs {
		keys = append(keys, f.getFriendIDsKey(userID))
	}
	newFriendCache.AddKeys(keys...)

	return newFriendCache
}

// GetTwoWayFriendIDs retrieves two-way friend IDs from the cache.
func (f *FriendCacheRedis) GetTwoWayFriendIDs(ctx context.Context, ownerUserID string) (twoWayFriendIDs []string, er
	friendIDs, err := f.GetFriendIDs(ctx, ownerUserID)
	if err != nil {
		return nil, err
	}
	for _, friendID := range friendIDs {
		friendFriendID, err := f.GetFriendIDs(ctx, friendID)
		if err != nil {
			return nil, err
		}
		if datautil.Contain(ownerUserID, friendFriendID...) {
			twoWayFriendIDs = append(twoWayFriendIDs, ownerUserID)
		}
	}

	return twoWayFriendIDs, nil
}

// DelTwoWayFriendIDs deletes two-way friend IDs from the cache.
func (f *FriendCacheRedis) DelTwoWayFriendIDs(ctx context.Context, ownerUserID string) cache.FriendCache {
	newFriendCache := f.CloneFriendCache()
	newFriendCache.AddKeys(f.getTwoWayFriendsIDsKey(ownerUserID))

	return newFriendCache
}

// GetFriend retrieves friend info from the cache or the database if not found.
func (f *FriendCacheRedis) GetFriend(ctx context.Context, ownerUserID, friendUserID string) (friend *model.Friend, e
	return getCache(ctx, f.rcClient, f.getFriendKey(ownerUserID,
		friendUserID), f.expireTime, func(ctx context.Context) (*model.Friend, error) {
		return f.friendDB.Take(ctx, ownerUserID, friendUserID)
	})
}

// DelFriend deletes friend info from the cache.
func (f *FriendCacheRedis) DelFriend(ownerUserID, friendUserID string) cache.FriendCache {
	newFriendCache := f.CloneFriendCache()
	newFriendCache.AddKeys(f.getFriendKey(ownerUserID, friendUserID))

	return newFriendCache
}

// DelFriends deletes multiple friend infos from the cache.
```

```go
func (f *FriendCacheRedis) DelFriends(ownerUserID string, friendUserIDs []string) cache.FriendCache {
    newFriendCache := f.CloneFriendCache()

    for _, friendUserID := range friendUserIDs {
        key := f.getFriendKey(ownerUserID, friendUserID)
        newFriendCache.AddKeys(key) // Assuming AddKeys marks the keys for deletion
    }

    return newFriendCache
}

func (f *FriendCacheRedis) DelOwner(friendUserID string, ownerUserIDs []string) cache.FriendCache {
    newFriendCache := f.CloneFriendCache()

    for _, ownerUserID := range ownerUserIDs {
        key := f.getFriendKey(ownerUserID, friendUserID)
        newFriendCache.AddKeys(key) // Assuming AddKeys marks the keys for deletion
    }

    return newFriendCache
}

func (f *FriendCacheRedis) DelMaxFriendVersion(ownerUserIDs ...string) cache.FriendCache {
    newFriendCache := f.CloneFriendCache()
    for _, ownerUserID := range ownerUserIDs {
        key := f.getFriendMaxVersionKey(ownerUserID)
        newFriendCache.AddKeys(key) // Assuming AddKeys marks the keys for deletion
    }

    return newFriendCache
}

func (f *FriendCacheRedis) FindMaxFriendVersion(ctx context.Context, ownerUserID string) (*model.VersionLog, error)
    return getCache(ctx, f.rcClient, f.getFriendMaxVersionKey(ownerUserID), f.expireTime, func(ctx context.Context) (*m
        return f.friendDB.FindIncrVersion(ctx, ownerUserID, 0, 0)
    })
}
```

pkg/common/storage/cache/redis/group.go

```
package redis

import (
	"context"
	"fmt"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/common"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

const (
	groupExpireTime = time.Second * 60 * 60 * 12
)

type GroupCacheRedis struct {
	cache.BatchDeleter
	groupDB        database.Group
	groupMemberDB  database.GroupMember
	groupRequestDB database.GroupRequest
	expireTime     time.Duration
	rcClient       *rocksCacheClient
	groupHash      cache.GroupHash
}

func NewGroupCacheRedis(rdb redis.UniversalClient, localCache *config.LocalCache, groupDB database.Group, groupMembe
	rc := newRocksCacheClient(rdb)
	return &GroupCacheRedis{
		BatchDeleter:   rc.GetBatchDeleter(localCache.Group.Topic),
		rcClient:       rc,
		expireTime:     groupExpireTime,
		groupDB:        groupDB,
		groupMemberDB:  groupMemberDB,
		groupRequestDB: groupRequestDB,
		groupHash:      hashCode,
	}
}

func (g *GroupCacheRedis) CloneGroupCache() cache.GroupCache {
	return &GroupCacheRedis{
		BatchDeleter:   g.BatchDeleter.Clone(),
		rcClient:       g.rcClient,
		expireTime:     g.expireTime,
		groupDB:        g.groupDB,
		groupMemberDB:  g.groupMemberDB,
		groupRequestDB: g.groupRequestDB,
	}
}

func (g *GroupCacheRedis) getGroupInfoKey(groupID string) string {
	return cachekey.GetGroupInfoKey(groupID)
}

func (g *GroupCacheRedis) getJoinedGroupsKey(userID string) string {
	return cachekey.GetJoinedGroupsKey(userID)
}
```

```go
func (g *GroupCacheRedis) getGroupMembersHashKey(groupID string) string {
	return cachekey.GetGroupMembersHashKey(groupID)
}

func (g *GroupCacheRedis) getGroupMemberIDsKey(groupID string) string {
	return cachekey.GetGroupMemberIDsKey(groupID)
}

func (g *GroupCacheRedis) getGroupMemberInfoKey(groupID, userID string) string {
	return cachekey.GetGroupMemberInfoKey(groupID, userID)
}

func (g *GroupCacheRedis) getGroupMemberNumKey(groupID string) string {
	return cachekey.GetGroupMemberNumKey(groupID)
}

func (g *GroupCacheRedis) getGroupRoleLevelMemberIDsKey(groupID string, roleLevel int32) string {
	return cachekey.GetGroupRoleLevelMemberIDsKey(groupID, roleLevel)
}

func (g *GroupCacheRedis) getGroupMemberMaxVersionKey(groupID string) string {
	return cachekey.GetGroupMemberMaxVersionKey(groupID)
}

func (g *GroupCacheRedis) getJoinGroupMaxVersionKey(userID string) string {
	return cachekey.GetJoinGroupMaxVersionKey(userID)
}

func (g *GroupCacheRedis) getGroupID(group *model.Group) string {
	return group.GroupID
}

func (g *GroupCacheRedis) GetGroupsInfo(ctx context.Context, groupIDs []string) (groups []*model.Group, err error) {
	return batchGetCache2(ctx, g.rcClient, g.expireTime, groupIDs, g.getGroupInfoKey, g.getGroupID, g.groupDB.Find)
}

func (g *GroupCacheRedis) GetGroupInfo(ctx context.Context, groupID string) (group *model.Group, err error) {
	return getCache(ctx, g.rcClient, g.getGroupInfoKey(groupID), g.expireTime, func(ctx context.Context) (*model.Group,
		return g.groupDB.Take(ctx, groupID)
	})
}

func (g *GroupCacheRedis) DelGroupsInfo(groupIDs ...string) cache.GroupCache {
	newGroupCache := g.CloneGroupCache()
	keys := make([]string, 0, len(groupIDs))
	for _, groupID := range groupIDs {
		keys = append(keys, g.getGroupInfoKey(groupID))
	}
	newGroupCache.AddKeys(keys...)

	return newGroupCache
}

func (g *GroupCacheRedis) DelGroupsOwner(groupIDs ...string) cache.GroupCache {
	newGroupCache := g.CloneGroupCache()
	keys := make([]string, 0, len(groupIDs))
	for _, groupID := range groupIDs {
		keys = append(keys, g.getGroupRoleLevelMemberIDsKey(groupID, constant.GroupOwner))
	}
	newGroupCache.AddKeys(keys...)

	return newGroupCache
}

func (g *GroupCacheRedis) DelGroupRoleLevel(groupID string, roleLevels []int32) cache.GroupCache {
```

```
	newGroupCache := g.CloneGroupCache()
	keys := make([]string, 0, len(roleLevels))
	for _, roleLevel := range roleLevels {
		keys = append(keys, g.getGroupRoleLevelMemberIDsKey(groupID, roleLevel))
	}
	newGroupCache.AddKeys(keys...)
	return newGroupCache
}

func (g *GroupCacheRedis) DelGroupAllRoleLevel(groupID string) cache.GroupCache {
	return g.DelGroupRoleLevel(groupID, []int32{constant.GroupOwner, constant.GroupAdmin, constant.GroupOrdinaryUsers})
}

func (g *GroupCacheRedis) GetGroupMembersHash(ctx context.Context, groupID string) (hashCode uint64, err error) {
	if g.groupHash == nil {
		return 0, errs.ErrInternalServer.WrapMsg("group hash is nil")
	}
	return getCache(ctx, g.rcClient, g.getGroupMembersHashKey(groupID), g.expireTime, func(ctx context.Context) (uint64
		return g.groupHash.GetGroupHash(ctx, groupID)
	})
}

func (g *GroupCacheRedis) GetGroupMemberHashMap(ctx context.Context, groupIDs []string) (map[string]*common.GroupSim
	if g.groupHash == nil {
		return nil, errs.ErrInternalServer.WrapMsg("group hash is nil")
	}
	res := make(map[string]*common.GroupSimpleUserID)
	for _, groupID := range groupIDs {
		hash, err := g.GetGroupMembersHash(ctx, groupID)
		if err != nil {
			return nil, err
		}
		log.ZDebug(ctx, "GetGroupMemberHashMap", "groupID", groupID, "hash", hash)
		num, err := g.GetGroupMemberNum(ctx, groupID)
		if err != nil {
			return nil, err
		}
		res[groupID] = &common.GroupSimpleUserID{Hash: hash, MemberNum: uint32(num)}
	}

	return res, nil
}

func (g *GroupCacheRedis) DelGroupMembersHash(groupID string) cache.GroupCache {
	cache := g.CloneGroupCache()
	cache.AddKeys(g.getGroupMembersHashKey(groupID))

	return cache
}

func (g *GroupCacheRedis) GetGroupMemberIDs(ctx context.Context, groupID string) (groupMemberIDs []string, err error
	return getCache(ctx, g.rcClient, g.getGroupMemberIDsKey(groupID), g.expireTime, func(ctx context.Context) ([]string
		return g.groupMemberDB.FindMemberUserID(ctx, groupID)
	})
}

func (g *GroupCacheRedis) DelGroupMemberIDs(groupID string) cache.GroupCache {
	cache := g.CloneGroupCache()
	cache.AddKeys(g.getGroupMemberIDsKey(groupID))

	return cache
}

func (g *GroupCacheRedis) findUserJoinedGroupID(ctx context.Context, userID string) ([]string, error) {
	groupIDs, err := g.groupMemberDB.FindUserJoinedGroupID(ctx, userID)
	if err != nil {
```

```
		return nil, err
	}
	return g.groupDB.FindJoinSortGroupID(ctx, groupIDs)
}

func (g *GroupCacheRedis) GetJoinedGroupIDs(ctx context.Context, userID string) (joinedGroupIDs []string, err error)
	return getCache(ctx, g.rcClient, g.getJoinedGroupsKey(userID), g.expireTime, func(ctx context.Context) ([]string, e
		return g.findUserJoinedGroupID(ctx, userID)
	})
}

func (g *GroupCacheRedis) DelJoinedGroupID(userIDs ...string) cache.GroupCache {
	keys := make([]string, 0, len(userIDs))
	for _, userID := range userIDs {
		keys = append(keys, g.getJoinedGroupsKey(userID))
	}
	cache := g.CloneGroupCache()
	cache.AddKeys(keys...)

	return cache
}

func (g *GroupCacheRedis) GetGroupMemberInfo(ctx context.Context, groupID, userID string) (groupMember *model.GroupM
	return getCache(ctx, g.rcClient, g.getGroupMemberInfoKey(groupID, userID), g.expireTime, func(ctx context.Context)
		return g.groupMemberDB.Take(ctx, groupID, userID)
	})
}

func (g *GroupCacheRedis) GetGroupMembersInfo(ctx context.Context, groupID string, userIDs []string) ([]*model.Group
	return batchGetCache2(ctx, g.rcClient, g.expireTime, userIDs, func(userID string) string {
		return g.getGroupMemberInfoKey(groupID, userID)
	}, func(member *model.GroupMember) string {
		return member.UserID
	}, func(ctx context.Context, userIDs []string) ([]*model.GroupMember, error) {
		return g.groupMemberDB.Find(ctx, groupID, userIDs)
	})
}

func (g *GroupCacheRedis) GetAllGroupMembersInfo(ctx context.Context, groupID string) (groupMembers []*model.GroupMe
	groupMemberIDs, err := g.GetGroupMemberIDs(ctx, groupID)
	if err != nil {
		return nil, err
	}

	return g.GetGroupMembersInfo(ctx, groupID, groupMemberIDs)
}

func (g *GroupCacheRedis) DelGroupMembersInfo(groupID string, userIDs ...string) cache.GroupCache {
	keys := make([]string, 0, len(userIDs))
	for _, userID := range userIDs {
		keys = append(keys, g.getGroupMemberInfoKey(groupID, userID))
	}
	cache := g.CloneGroupCache()
	cache.AddKeys(keys...)

	return cache
}

func (g *GroupCacheRedis) GetGroupMemberNum(ctx context.Context, groupID string) (memberNum int64, err error) {
	return getCache(ctx, g.rcClient, g.getGroupMemberNumKey(groupID), g.expireTime, func(ctx context.Context) (int64, e
		return g.groupMemberDB.TakeGroupMemberNum(ctx, groupID)
	})
}

func (g *GroupCacheRedis) DelGroupsMemberNum(groupID ...string) cache.GroupCache {
	keys := make([]string, 0, len(groupID))
```

```
	for _, groupID := range groupID {
		keys = append(keys, g.getGroupMemberNumKey(groupID))
	}
	cache := g.CloneGroupCache()
	cache.AddKeys(keys...)

	return cache
}

func (g *GroupCacheRedis) GetGroupOwner(ctx context.Context, groupID string) (*model.GroupMember, error) {
	members, err := g.GetGroupRoleLevelMemberInfo(ctx, groupID, constant.GroupOwner)
	if err != nil {
		return nil, err
	}
	if len(members) == 0 {
		return nil, errs.ErrRecordNotFound.WrapMsg(fmt.Sprintf("group %s owner not found", groupID))
	}
	return members[0], nil
}

func (g *GroupCacheRedis) GetGroupsOwner(ctx context.Context, groupIDs []string) ([]*model.GroupMember, error) {
	members := make([]*model.GroupMember, 0, len(groupIDs))
	for _, groupID := range groupIDs {
		items, err := g.GetGroupRoleLevelMemberInfo(ctx, groupID, constant.GroupOwner)
		if err != nil {
			return nil, err
		}
		if len(items) > 0 {
			members = append(members, items[0])
		}
	}
	return members, nil
}

func (g *GroupCacheRedis) GetGroupRoleLevelMemberIDs(ctx context.Context, groupID string, roleLevel int32) ([]string
	return getCache(ctx, g.rcClient, g.getGroupRoleLevelMemberIDsKey(groupID, roleLevel), g.expireTime, func(ctx contex
		return g.groupMemberDB.FindRoleLevelUserIDs(ctx, groupID, roleLevel)
	})
}

func (g *GroupCacheRedis) GetGroupRoleLevelMemberInfo(ctx context.Context, groupID string, roleLevel int32) ([]*mode
	userIDs, err := g.GetGroupRoleLevelMemberIDs(ctx, groupID, roleLevel)
	if err != nil {
		return nil, err
	}
	return g.GetGroupMembersInfo(ctx, groupID, userIDs)
}

func (g *GroupCacheRedis) GetGroupRolesLevelMemberInfo(ctx context.Context, groupID string, roleLevels []int32) ([]*
	var userIDs []string
	for _, roleLevel := range roleLevels {
		ids, err := g.GetGroupRoleLevelMemberIDs(ctx, groupID, roleLevel)
		if err != nil {
			return nil, err
		}
		userIDs = append(userIDs, ids...)
	}
	return g.GetGroupMembersInfo(ctx, groupID, userIDs)
}

func (g *GroupCacheRedis) FindGroupMemberUser(ctx context.Context, groupIDs []string, userID string) ([]*model.Group
	if len(groupIDs) == 0 {
		var err error
		groupIDs, err = g.GetJoinedGroupIDs(ctx, userID)
		if err != nil {
			return nil, err
```

```
    }
  }
  return batchGetCache2(ctx, g.rcClient, g.expireTime, groupIDs, func(groupID string) string {
    return g.getGroupMemberInfoKey(groupID, userID)
  }, func(member *model.GroupMember) string {
    return member.GroupID
  }, func(ctx context.Context, groupIDs []string) ([]*model.GroupMember, error) {
    return g.groupMemberDB.FindInGroup(ctx, userID, groupIDs)
  })
}

func (g *GroupCacheRedis) DelMaxGroupMemberVersion(groupIDs ...string) cache.GroupCache {
  keys := make([]string, 0, len(groupIDs))
  for _, groupID := range groupIDs {
    keys = append(keys, g.getGroupMemberMaxVersionKey(groupID))
  }
  cache := g.CloneGroupCache()
  cache.AddKeys(keys...)
  return cache
}

func (g *GroupCacheRedis) DelMaxJoinGroupVersion(userIDs ...string) cache.GroupCache {
  keys := make([]string, 0, len(userIDs))
  for _, userID := range userIDs {
    keys = append(keys, g.getJoinGroupMaxVersionKey(userID))
  }
  cache := g.CloneGroupCache()
  cache.AddKeys(keys...)
  return cache
}

func (g *GroupCacheRedis) FindMaxGroupMemberVersion(ctx context.Context, groupID string) (*model.VersionLog, error)
  return getCache(ctx, g.rcClient, g.getGroupMemberMaxVersionKey(groupID), g.expireTime, func(ctx context.Context) (*
    return g.groupMemberDB.FindMemberIncrVersion(ctx, groupID, 0, 0)
  })
}

func (g *GroupCacheRedis) BatchFindMaxGroupMemberVersion(ctx context.Context, groupIDs []string) ([]*model.VersionLo
  return batchGetCache2(ctx, g.rcClient, g.expireTime, groupIDs,
    func(groupID string) string {
      return g.getGroupMemberMaxVersionKey(groupID)
    }, func(versionLog *model.VersionLog) string {
      return versionLog.DID
    }, func(ctx context.Context, groupIDs []string) ([]*model.VersionLog, error) {
      // create two slices with len is groupIDs, just need 0
      versions := make([]uint, len(groupIDs))
      limits := make([]int, len(groupIDs))

      return g.groupMemberDB.BatchFindMemberIncrVersion(ctx, groupIDs, versions, limits)
    })
}

func (g *GroupCacheRedis) FindMaxJoinGroupVersion(ctx context.Context, userID string) (*model.VersionLog, error) {
  return getCache(ctx, g.rcClient, g.getJoinGroupMaxVersionKey(userID), g.expireTime, func(ctx context.Context) (*mod
    return g.groupMemberDB.FindJoinIncrVersion(ctx, userID, 0, 0)
  })
}
```

pkg/common/storage/cache/redis/lua_script.go

```
package redis

import (
	"context"
	"errors"
	"fmt"

	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

var (
	setBatchWithCommonExpireScript = redis.NewScript(`
local expire = tonumber(ARGV[1])
for i, key in ipairs(KEYS) do
    redis.call('SET', key, ARGV[i + 1])
    redis.call('EXPIRE', key, expire)
end
return #KEYS
`)

	setBatchWithIndividualExpireScript = redis.NewScript(`
local n = #KEYS
for i = 1, n do
    redis.call('SET', KEYS[i], ARGV[i])
    redis.call('EXPIRE', KEYS[i], ARGV[i + n])
end
return n
`)

	deleteBatchScript = redis.NewScript(`
for i, key in ipairs(KEYS) do
    redis.call('DEL', key)
end
return #KEYS
`)

	getBatchScript = redis.NewScript(`
local values = {}
for i, key in ipairs(KEYS) do
    local value = redis.call('GET', key)
    table.insert(values, value)
end
return values
`)
)

func callLua(ctx context.Context, rdb redis.Scripter, script *redis.Script, keys []string, args []any) (any, error) {
	log.ZDebug(ctx, "callLua args", "scriptHash", script.Hash(), "keys", keys, "args", args)
	r := script.EvalSha(ctx, rdb, keys, args)
	if redis.HasErrorPrefix(r.Err(), "NOSCRIPT") {
		if err := script.Load(ctx, rdb).Err(); err != nil {
			r = script.Eval(ctx, rdb, keys, args)
		} else {
			r = script.EvalSha(ctx, rdb, keys, args)
		}
	}
	v, err := r.Result()
	if errors.Is(err, redis.Nil) {
		err = nil
	}
	return v, errs.WrapMsg(err, "call lua err", "scriptHash", script.Hash(), "keys", keys, "args", args)
```

```
}

func LuaSetBatchWithCommonExpire(ctx context.Context, rdb redis.Scripter, keys []string, values []string, expire int
    // Check if the lengths of keys and values match
    if len(keys) != len(values) {
        return errs.New("keys and values length mismatch").Wrap()
    }

    // Ensure allocation size does not overflow
    maxAllowedLen := (1 << 31) - 1 // 2GB limit (maximum address space for 32-bit systems)

    if len(values) > maxAllowedLen-1 {
        return fmt.Errorf("values length is too large, causing overflow")
    }
    var vals = make([]any, 0, 1+len(values))
    vals = append(vals, expire)
    for _, v := range values {
        vals = append(vals, v)
    }
    _, err := callLua(ctx, rdb, setBatchWithCommonExpireScript, keys, vals)
    return err
}

func LuaSetBatchWithIndividualExpire(ctx context.Context, rdb redis.Scripter, keys []string, values []string, expire
    // Check if the lengths of keys, values, and expires match
    if len(keys) != len(values) || len(keys) != len(expires) {
        return errs.New("keys and values length mismatch").Wrap()
    }

    // Ensure the allocation size does not overflow
    maxAllowedLen := (1 << 31) - 1 // 2GB limit (maximum address space for 32-bit systems)

    if len(values) > maxAllowedLen-1 {
        return errs.New(fmt.Sprintf("values length %d exceeds the maximum allowed length %d", len(values), maxAllowedLen-
    }
    var vals = make([]any, 0, len(values)+len(expires))
    for _, v := range values {
        vals = append(vals, v)
    }
    for _, ex := range expires {
        vals = append(vals, ex)
    }
    _, err := callLua(ctx, rdb, setBatchWithIndividualExpireScript, keys, vals)
    return err
}

func LuaDeleteBatch(ctx context.Context, rdb redis.Scripter, keys []string) error {
    _, err := callLua(ctx, rdb, deleteBatchScript, keys, nil)
    return err
}

func LuaGetBatch(ctx context.Context, rdb redis.Scripter, keys []string) ([]any, error) {
    v, err := callLua(ctx, rdb, getBatchScript, keys, nil)
    if err != nil {
        return nil, err
    }
    values, ok := v.([]any)
    if !ok {
        return nil, servererrs.ErrArgs.WrapMsg("invalid lua get batch result")
    }
    return values, nil

}
```

pkg/common/storage/cache/redis/lua_script_test.go

```
package redis

import (
	"context"
	"github.com/go-redis/redismock/v9"
	"github.com/stretchr/testify/assert"
	"github.com/stretchr/testify/require"
	"testing"
)

func TestLuaSetBatchWithCommonExpire(t *testing.T) {
	rdb, mock := redismock.NewClientMock()
	ctx := context.Background()

	keys := []string{"key1", "key2"}
	values := []string{"value1", "value2"}
	expire := 10

	mock.ExpectEvalSha(setBatchWithCommonExpireScript.Hash(), keys, []any{expire, "value1", "value2"}).SetVal(int64(len

	err := LuaSetBatchWithCommonExpire(ctx, rdb, keys, values, expire)
	require.NoError(t, err)
	assert.NoError(t, mock.ExpectationsWereMet())
}

func TestLuaSetBatchWithIndividualExpire(t *testing.T) {
	rdb, mock := redismock.NewClientMock()
	ctx := context.Background()

	keys := []string{"key1", "key2"}
	values := []string{"value1", "value2"}
	expires := []int{10, 20}

	args := make([]any, 0, len(values)+len(expires))
	for _, v := range values {
		args = append(args, v)
	}
	for _, ex := range expires {
		args = append(args, ex)
	}

	mock.ExpectEvalSha(setBatchWithIndividualExpireScript.Hash(), keys, args).SetVal(int64(len(keys)))

	err := LuaSetBatchWithIndividualExpire(ctx, rdb, keys, values, expires)
	require.NoError(t, err)
	assert.NoError(t, mock.ExpectationsWereMet())
}

func TestLuaDeleteBatch(t *testing.T) {
	rdb, mock := redismock.NewClientMock()
	ctx := context.Background()

	keys := []string{"key1", "key2"}

	mock.ExpectEvalSha(deleteBatchScript.Hash(), keys, []any{}).SetVal(int64(len(keys)))

	err := LuaDeleteBatch(ctx, rdb, keys)
	require.NoError(t, err)
	assert.NoError(t, mock.ExpectationsWereMet())
}

func TestLuaGetBatch(t *testing.T) {
	rdb, mock := redismock.NewClientMock()
	ctx := context.Background()
```

```
    keys := []string{"key1", "key2"}
    expectedValues := []any{"value1", "value2"}

    mock.ExpectEvalSha(getBatchScript.Hash(), keys, []any{}).SetVal(expectedValues)

    values, err := LuaGetBatch(ctx, rdb, keys)
    require.NoError(t, err)
    assert.NoError(t, mock.ExpectationsWereMet())
    assert.Equal(t, expectedValues, values)
}
```

pkg/common/storage/cache/redis/minio.go

```
package redis

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/tools/s3/minio"
	"github.com/redis/go-redis/v9"
)

func NewMinioCache(rdb redis.UniversalClient) minio.Cache {
	rc := newRocksCacheClient(rdb)
	return &minioCacheRedis{
		BatchDeleter: rc.GetBatchDeleter(),
		rcClient:     rc,
		expireTime:   time.Hour * 24 * 7,
	}
}

type minioCacheRedis struct {
	cache.BatchDeleter
	rcClient   *rocksCacheClient
	expireTime time.Duration
}

func (g *minioCacheRedis) getObjectImageInfoKey(key string) string {
	return cachekey.GetObjectImageInfoKey(key)
}

func (g *minioCacheRedis) getMinioImageThumbnailKey(key string, format string, width int, height int) string {
	return cachekey.GetMinioImageThumbnailKey(key, format, width, height)
}

func (g *minioCacheRedis) DelObjectImageInfoKey(ctx context.Context, keys ...string) error {
	ks := make([]string, 0, len(keys))
	for _, key := range keys {
		ks = append(ks, g.getObjectImageInfoKey(key))
	}
	return g.BatchDeleter.ExecDelWithKeys(ctx, ks)
}

func (g *minioCacheRedis) DelImageThumbnailKey(ctx context.Context, key string, format string, width int, height int
	return g.BatchDeleter.ExecDelWithKeys(ctx, []string{g.getMinioImageThumbnailKey(key, format, width, height)})

}

func (g *minioCacheRedis) GetImageObjectKeyInfo(ctx context.Context, key string, fn func(ctx context.Context) (*mini
	info, err := getCache(ctx, g.rcClient, g.getObjectImageInfoKey(key), g.expireTime, fn)
	if err != nil {
		return nil, err
	}
	return info, nil
}

func (g *minioCacheRedis) GetThumbnailKey(ctx context.Context, key string, format string, width int, height int, min
	return getCache(ctx, g.rcClient, g.getMinioImageThumbnailKey(key, format, width, height), g.expireTime, minioCache)
}
```

pkg/common/storage/cache/redis/msg.go

```
package redis

import (
	"context"
	"encoding/json"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
) //

// msgCacheTimeout is  expiration time of message cache, 86400 seconds
const msgCacheTimeout = time.Hour * 24

func NewMsgCache(client redis.UniversalClient, db database.Msg) cache.MsgCache {
	return &msgCache{
		rcClient:       newRocksCacheClient(client),
		msgDocDatabase: db,
	}
}

type msgCache struct {
	rcClient       *rocksCacheClient
	msgDocDatabase database.Msg
}

func (c *msgCache) getSendMsgKey(id string) string {
	return cachekey.GetSendMsgKey(id)
}

func (c *msgCache) SetSendMsgStatus(ctx context.Context, id string, status int32) error {
	return errs.Wrap(c.rcClient.GetRedis().Set(ctx, c.getSendMsgKey(id), status, time.Hour*24).Err())
}

func (c *msgCache) GetSendMsgStatus(ctx context.Context, id string) (int32, error) {
	result, err := c.rcClient.GetRedis().Get(ctx, c.getSendMsgKey(id)).Int()
	return int32(result), errs.Wrap(err)
}

func (c *msgCache) GetMessageBySeqs(ctx context.Context, conversationID string, seqs []int64) ([]*model.MsgInfoModel
	if len(seqs) == 0 {
		return nil, nil
	}
	getKey := func(seq int64) string {
		return cachekey.GetMsgCacheKey(conversationID, seq)
	}
	getMsgID := func(msg *model.MsgInfoModel) int64 {
		return msg.Msg.Seq
	}
	find := func(ctx context.Context, seqs []int64) ([]*model.MsgInfoModel, error) {
		return c.msgDocDatabase.FindSeqs(ctx, conversationID, seqs)
	}
	return batchGetCache2(ctx, c.rcClient, msgCacheTimeout, seqs, getKey, getMsgID, find)
}

func (c *msgCache) DelMessageBySeqs(ctx context.Context, conversationID string, seqs []int64) error {
	if len(seqs) == 0 {
		return nil
	}
```

```
	keys := datautil.Slice(seqs, func(seq int64) string {
		return cachekey.GetMsgCacheKey(conversationID, seq)
	})
	slotKeys, err := groupKeysBySlot(ctx, c.rcClient.GetRedis(), keys)
	if err != nil {
		return err
	}
	for _, keys := range slotKeys {
		if err := c.rcClient.GetClient().TagAsDeletedBatch2(ctx, keys); err != nil {
			return err
		}
	}
	return nil
}

func (c *msgCache) SetMessageBySeqs(ctx context.Context, conversationID string, msgs []*model.MsgInfoModel) error {
	for _, msg := range msgs {
		if msg == nil || msg.Msg == nil || msg.Msg.Seq <= 0 {
			continue
		}
		data, err := json.Marshal(msg)
		if err != nil {
			return err
		}
		if err := c.rcClient.GetClient().RawSet(ctx, cachekey.GetMsgCacheKey(conversationID, msg.Msg.Seq), string(data), 
			return err
		}
	}
	return nil
}
```

pkg/common/storage/cache/redis/online.go

```
package redis

import (
	"context"
	"fmt"
	"strconv"
	"strings"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
)

func NewUserOnline(rdb redis.UniversalClient) cache.OnlineCache {
	if rdb == nil || config.Standalone() {
		return mcache.NewOnlineCache()
	}
	return &userOnline{
		rdb:         rdb,
		expire:      cachekey.OnlineExpire,
		channelName: cachekey.OnlineChannel,
	}
}

type userOnline struct {
	rdb         redis.UniversalClient
	expire      time.Duration
	channelName string
}

func (s *userOnline) getUserOnlineKey(userID string) string {
	return cachekey.GetOnlineKey(userID)
}

func (s *userOnline) GetOnline(ctx context.Context, userID string) ([]int32, error) {
	members, err := s.rdb.ZRangeByScore(ctx, s.getUserOnlineKey(userID), &redis.ZRangeBy{
		Min: strconv.FormatInt(time.Now().Unix(), 10),
		Max: "+inf",
	}).Result()
	if err != nil {
		return nil, errs.Wrap(err)
	}
	platformIDs := make([]int32, 0, len(members))
	for _, member := range members {
		val, err := strconv.Atoi(member)
		if err != nil {
			return nil, errs.Wrap(err)
		}
		platformIDs = append(platformIDs, int32(val))
	}
	return platformIDs, nil
}

func (s *userOnline) GetAllOnlineUsers(ctx context.Context, cursor uint64) (map[string][]int32, uint64, error) {
	result := make(map[string][]int32)

	keys, nextCursor, err := s.rdb.Scan(ctx, cursor, fmt.Sprintf("%s*", cachekey.OnlineKey), constant.ParamMaxLength).R
	if err != nil {
```

```
		return nil, 0, err
	}

	for _, key := range keys {
		userID := cachekey.GetOnlineKeyUserID(key)
		strValues, err := s.rdb.ZRange(ctx, key, 0, -1).Result()
		if err != nil {
			return nil, 0, err
		}

		values := make([]int32, 0, len(strValues))
		for _, value := range strValues {
			intValue, err := strconv.Atoi(value)
			if err != nil {
				return nil, 0, errs.Wrap(err)
			}
			values = append(values, int32(intValue))
		}

		result[userID] = values
	}

	return result, nextCursor, nil
}

func (s *userOnline) SetUserOnline(ctx context.Context, userID string, online, offline []int32) error {
	script := `
local key = KEYS[1]
local score = ARGV[3]
local num1 = redis.call("ZCARD", key)
redis.call("ZREMRANGEBYSCORE", key, "-inf", ARGV[2])
for i = 5, tonumber(ARGV[4])+4 do
  redis.call("ZREM", key, ARGV[i])
end
local num2 = redis.call("ZCARD", key)
for i = 5+tonumber(ARGV[4]), #ARGV do
  redis.call("ZADD", key, score, ARGV[i])
end
redis.call("EXPIRE", key, ARGV[1])
local num3 = redis.call("ZCARD", key)
local change = (num1 ~= num2) or (num2 ~= num3)
if change then
  local members = redis.call("ZRANGE", key, 0, -1)
  table.insert(members, "1")
  return members
else
  return {"0"}
end
`
	now := time.Now()
	argv := make([]any, 0, 2+len(online)+len(offline))
	argv = append(argv, int32(s.expire/time.Second), now.Unix(), now.Add(s.expire).Unix(), int32(len(offline)))
	for _, platformID := range offline {
		argv = append(argv, platformID)
	}
	for _, platformID := range online {
		argv = append(argv, platformID)
	}
	keys := []string{s.getUserOnlineKey(userID)}
	platformIDs, err := s.rdb.Eval(ctx, script, keys, argv).StringSlice()
	if err != nil {
		log.ZError(ctx, "redis SetUserOnline", err, "userID", userID, "online", online, "offline", offline)
		return err
	}
	if len(platformIDs) == 0 {
		return errs.ErrInternalServer.WrapMsg("SetUserOnline redis lua invalid return value")
```

```
	}
	if platformIDs[len(platformIDs)-1] != "0" {
		log.ZDebug(ctx, "redis SetUserOnline push", "userID", userID, "online", online, "offline", offline, "platformIDs",
		platformIDs[len(platformIDs)-1] = userID
		msg := strings.Join(platformIDs, ":")
		if err := s.rdb.Publish(ctx, s.channelName, msg).Err(); err != nil {
			return errs.Wrap(err)
		}
	} else {
		log.ZDebug(ctx, "redis SetUserOnline not push", "userID", userID, "online", online, "offline", offline)
	}
	return nil
}
```

pkg/common/storage/cache/redis/online_test.go

```go
package redis

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/db/redisutil"
	"testing"
	"time"
)

/*
address: [ 172.16.8.48:7001, 172.16.8.48:7002, 172.16.8.48:7003, 172.16.8.48:7004, 172.16.8.48:7005, 172.16.8.48:700
username:
password: passwd123
clusterMode: true
db: 0
maxRetry: 10
*/
func TestName111111(t *testing.T) {
	conf := config.Redis{
		Address: []string{
			"172.16.8.124:7001",
			"172.16.8.124:7002",
			"172.16.8.124:7003",
			"172.16.8.124:7004",
			"172.16.8.124:7005",
			"172.16.8.124:7006",
		},
		RedisMode: "cluster",
		Password:    "passwd123",
		//Address:  []string{"localhost:16379"},
		//Password: "openIM123",
	}
	ctx, cancel := context.WithTimeout(context.Background(), time.Second*1000)
	defer cancel()
	rdb, err := redisutil.NewRedisClient(ctx, conf.Build())
	if err != nil {
		panic(err)
	}
	online := NewUserOnline(rdb)

	userID := "a123456"
	t.Log(online.GetOnline(ctx, userID))
	t.Log(online.SetUserOnline(ctx, userID, []int32{1, 2, 3, 4}, nil))
	t.Log(online.GetOnline(ctx, userID))

}

func TestName111(t *testing.T) {

}
```

pkg/common/storage/cache/redis/redis_shard_manager.go

```go
package redis

import (
	"context"

	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/redis/go-redis/v9"
	"golang.org/x/sync/errgroup"
)

const (
	defaultBatchSize       = 50
	defaultConcurrentLimit = 3
)

// RedisShardManager is a class for sharding and processing keys
type RedisShardManager struct {
	redisClient redis.UniversalClient
	config      *Config
}
type Config struct {
	batchSize       int
	continueOnError bool
	concurrentLimit int
}

// Option is a function type for configuring Config
type Option func(c *Config)

//// NewRedisShardManager creates a new RedisShardManager instance
//func NewRedisShardManager(redisClient redis.UniversalClient, opts ...Option) *RedisShardManager {
//	config := &Config{
//		batchSize:       defaultBatchSize, // Default batch size is 50 keys
//		continueOnError: false,
//		concurrentLimit: defaultConcurrentLimit, // Default concurrent limit is 3
//	}
//	for _, opt := range opts {
//		opt(config)
//	}
//	rsm := &RedisShardManager{
//		redisClient: redisClient,
//		config:      config,
//	}
//	return rsm
//}
//
//// WithBatchSize sets the number of keys to process per batch
//func WithBatchSize(size int) Option {
//	return func(c *Config) {
//		c.batchSize = size
//	}
//}
//
//// WithContinueOnError sets whether to continue processing on error
//func WithContinueOnError(continueOnError bool) Option {
//	return func(c *Config) {
//		c.continueOnError = continueOnError
//	}
//}
//
//// WithConcurrentLimit sets the concurrency limit
//func WithConcurrentLimit(limit int) Option {
//	return func(c *Config) {
```

```
//■■c.concurrentLimit = limit
//■}
//}
//
//// ProcessKeysBySlot groups keys by their Redis cluster hash slots and processes them using the provided function.
//func (rsm *RedisShardManager) ProcessKeysBySlot(
//■ctx context.Context,
//■keys []string,
//■processFunc func(ctx context.Context, slot int64, keys []string) error,
//) error {
//
//■// Group keys by slot
//■slots, err := groupKeysBySlot(ctx, rsm.redisClient, keys)
//■if err != nil {
//■■return err
//■}
//
//■g, ctx := errgroup.WithContext(ctx)
//■g.SetLimit(rsm.config.concurrentLimit)
//
//■// Process keys in each slot using the provided function
//■for slot, singleSlotKeys := range slots {
//■■batches := splitIntoBatches(singleSlotKeys, rsm.config.batchSize)
//■■for _, batch := range batches {
//■■■slot, batch := slot, batch // Avoid closure capture issue
//■■■g.Go(func() error {
//■■■■err := processFunc(ctx, slot, batch)
//■■■■if err != nil {
//■■■■■log.ZWarn(ctx, "Batch processFunc failed", err, "slot", slot, "keys", batch)
//■■■■■if !rsm.config.continueOnError {
//■■■■■■return err
//■■■■■}
//■■■■}
//■■■■return nil
//■■■})
//■■}
//■}
//
//■if err := g.Wait(); err != nil {
//■■return err
//■}
//■return nil
//}

// groupKeysBySlot groups keys by their Redis cluster hash slots.
func groupKeysBySlot(ctx context.Context, redisClient redis.UniversalClient, keys []string) (map[int64][]string, err
■slots := make(map[int64][]string)
■clusterClient, isCluster := redisClient.(*redis.ClusterClient)
■if isCluster && len(keys) > 1 {
■■pipe := clusterClient.Pipeline()
■■cmds := make([]*redis.IntCmd, len(keys))
■■for i, key := range keys {
■■■cmds[i] = pipe.ClusterKeySlot(ctx, key)
■■}
■■_, err := pipe.Exec(ctx)
■■if err != nil {
■■■return nil, errs.WrapMsg(err, "get slot err")
■■}

■■for i, cmd := range cmds {
■■■slot, err := cmd.Result()
■■■if err != nil {
■■■■log.ZWarn(ctx, "some key get slot err", err, "key", keys[i])
■■■■return nil, errs.WrapMsg(err, "get slot err", "key", keys[i])
■■■}
■■■slots[slot] = append(slots[slot], keys[i])
```

```
    }
  } else {
    // If not a cluster client, put all keys in the same slot (0)
    slots[0] = keys
  }

  return slots, nil
}

// splitIntoBatches splits keys into batches of the specified size
func splitIntoBatches(keys []string, batchSize int) [][]string {
  var batches [][]string
  for batchSize < len(keys) {
    keys, batches = keys[batchSize:], append(batches, keys[0:batchSize:batchSize])
  }
  return append(batches, keys)
}

// ProcessKeysBySlot groups keys by their Redis cluster hash slots and processes them using the provided function.
func ProcessKeysBySlot(
  ctx context.Context,
  redisClient redis.UniversalClient,
  keys []string,
  processFunc func(ctx context.Context, slot int64, keys []string) error,
  opts ...Option,
) error {

  config := &Config{
    batchSize:       defaultBatchSize,
    continueOnError: false,
    concurrentLimit: defaultConcurrentLimit,
  }
  for _, opt := range opts {
    opt(config)
  }

  // Group keys by slot
  slots, err := groupKeysBySlot(ctx, redisClient, keys)
  if err != nil {
    return err
  }

  g, ctx := errgroup.WithContext(ctx)
  g.SetLimit(config.concurrentLimit)

  // Process keys in each slot using the provided function
  for slot, singleSlotKeys := range slots {
    batches := splitIntoBatches(singleSlotKeys, config.batchSize)
    for _, batch := range batches {
      slot, batch := slot, batch // Avoid closure capture issue
      g.Go(func() error {
        err := processFunc(ctx, slot, batch)
        if err != nil {
          log.ZWarn(ctx, "Batch processFunc failed", err, "slot", slot, "keys", batch)
          if !config.continueOnError {
            return err
          }
        }
        return nil
      })
    }
  }

  if err := g.Wait(); err != nil {
    return err
  }
```

```
	return nil
}

func DeleteCacheBySlot(ctx context.Context, rcClient *rocksCacheClient, keys []string) error {
	switch len(keys) {
	case 0:
		return nil
	case 1:
		return rcClient.GetClient().TagAsDeletedBatch2(ctx, keys)
	default:
		return ProcessKeysBySlot(ctx, rcClient.GetRedis(), keys, func(ctx context.Context, slot int64, keys []string) err
			return rcClient.GetClient().TagAsDeletedBatch2(ctx, keys)
		})
	}
}
```

pkg/common/storage/cache/redis/s3.go

```
package redis

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/s3"
	"github.com/openimsdk/tools/s3/cont"
	"github.com/redis/go-redis/v9"
)

func NewObjectCacheRedis(rdb redis.UniversalClient, objDB database.ObjectInfo) cache.ObjectCache {
	rc := newRocksCacheClient(rdb)
	return &objectCacheRedis{
		BatchDeleter: rc.GetBatchDeleter(),
		rcClient:     rc,
		expireTime:   time.Hour * 12,
		objDB:        objDB,
	}
}

type objectCacheRedis struct {
	cache.BatchDeleter
	objDB      database.ObjectInfo
	rcClient   *rocksCacheClient
	expireTime time.Duration
}

func (g *objectCacheRedis) getObjectKey(engine string, name string) string {
	return cachekey.GetObjectKey(engine, name)
}

func (g *objectCacheRedis) CloneObjectCache() cache.ObjectCache {
	return &objectCacheRedis{
		BatchDeleter: g.BatchDeleter.Clone(),
		rcClient:     g.rcClient,
		expireTime:   g.expireTime,
		objDB:        g.objDB,
	}
}

func (g *objectCacheRedis) DelObjectName(engine string, names ...string) cache.ObjectCache {
	objectCache := g.CloneObjectCache()
	keys := make([]string, 0, len(names))
	for _, name := range names {
		keys = append(keys, g.getObjectKey(name, engine))
	}
	objectCache.AddKeys(keys...)
	return objectCache
}

func (g *objectCacheRedis) GetName(ctx context.Context, engine string, name string) (*model.Object, error) {
	return getCache(ctx, g.rcClient, g.getObjectKey(name, engine), g.expireTime, func(ctx context.Context) (*model.Obje
		return g.objDB.Take(ctx, engine, name)
	})
}

func NewS3Cache(rdb redis.UniversalClient, s3 s3.Interface) cont.S3Cache {
	rc := newRocksCacheClient(rdb)
	return &s3CacheRedis{
```

```
    BatchDeleter: rc.GetBatchDeleter(),
    rcClient:     rc,
    expireTime:   time.Hour * 12,
    s3:           s3,
  }
}

type s3CacheRedis struct {
  cache.BatchDeleter
  s3         s3.Interface
  rcClient   *rocksCacheClient
  expireTime time.Duration
}

func (g *s3CacheRedis) getS3Key(engine string, name string) string {
  return cachekey.GetS3Key(engine, name)
}

func (g *s3CacheRedis) DelS3Key(ctx context.Context, engine string, keys ...string) error {
  ks := make([]string, 0, len(keys))
  for _, key := range keys {
    ks = append(ks, g.getS3Key(engine, key))
  }
  return g.BatchDeleter.ExecDelWithKeys(ctx, ks)
}

func (g *s3CacheRedis) GetKey(ctx context.Context, engine string, name string) (*s3.ObjectInfo, error) {
  return getCache(ctx, g.rcClient, g.getS3Key(engine, name), g.expireTime, func(ctx context.Context) (*s3.ObjectInfo,
    return g.s3.StatObject(ctx, name)
  })
}
```

pkg/common/storage/cache/redis/seq_conversation.go

```
package redis

import (
    "context"
    "errors"
    "fmt"
    "strconv"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/redis/go-redis/v9"
)

func NewSeqConversationCacheRedis(rdb redis.UniversalClient, mgo database.SeqConversation) cache.SeqConversationCach
    if rdb == nil {
        return mcache.NewSeqConversationCache(mgo)
    }
    return &seqConversationCacheRedis{
        mgo:              mgo,
        lockTime:         time.Second * 3,
        dataTime:         time.Hour * 24 * 365,
        minSeqExpireTime: time.Hour,
        rcClient:         newRocksCacheClient(rdb),
    }
}

type seqConversationCacheRedis struct {
    mgo              database.SeqConversation
    rcClient         *rocksCacheClient
    lockTime         time.Duration
    dataTime         time.Duration
    minSeqExpireTime time.Duration
}

func (s *seqConversationCacheRedis) getMinSeqKey(conversationID string) string {
    return cachekey.GetMallocMinSeqKey(conversationID)
}

func (s *seqConversationCacheRedis) SetMinSeq(ctx context.Context, conversationID string, seq int64) error {
    return s.SetMinSeqs(ctx, map[string]int64{conversationID: seq})
}

func (s *seqConversationCacheRedis) GetMinSeq(ctx context.Context, conversationID string) (int64, error) {
    return getCache(ctx, s.rcClient, s.getMinSeqKey(conversationID), s.minSeqExpireTime, func(ctx context.Context) (int
        return s.mgo.GetMinSeq(ctx, conversationID)
    })
}

func (s *seqConversationCacheRedis) getSingleMaxSeq(ctx context.Context, conversationID string) (map[string]int64, e
    seq, err := s.GetMaxSeq(ctx, conversationID)
    if err != nil {
        return nil, err
    }
    return map[string]int64{conversationID: seq}, nil
}

func (s *seqConversationCacheRedis) getSingleMaxSeqWithTime(ctx context.Context, conversationID string) (map[string]
    seq, err := s.GetMaxSeqWithTime(ctx, conversationID)
```

```
	if err != nil {
		return nil, err
	}
	return map[string]database.SeqTime{conversationID: seq}, nil
}

func (s *seqConversationCacheRedis) batchGetMaxSeq(ctx context.Context, keys []string, keyConversationID map[string]
	result := make([]*redis.StringCmd, len(keys))
	pipe := s.rcClient.GetRedis().Pipeline()
	for i, key := range keys {
		result[i] = pipe.HGet(ctx, key, "CURR")
	}
	if _, err := pipe.Exec(ctx); err != nil && !errors.Is(err, redis.Nil) {
		return errs.Wrap(err)
	}
	var notFoundKey []string
	for i, r := range result {
		req, err := r.Int64()
		if err == nil {
			seqs[keyConversationID[keys[i]]] = req
		} else if errors.Is(err, redis.Nil) {
			notFoundKey = append(notFoundKey, keys[i])
		} else {
			return errs.Wrap(err)
		}
	}
	for _, key := range notFoundKey {
		conversationID := keyConversationID[key]
		seq, err := s.GetMaxSeq(ctx, conversationID)
		if err != nil {
			return err
		}
		seqs[conversationID] = seq
	}
	return nil
}

func (s *seqConversationCacheRedis) batchGetMaxSeqWithTime(ctx context.Context, keys []string, keyConversationID map
	result := make([]*redis.SliceCmd, len(keys))
	pipe := s.rcClient.GetRedis().Pipeline()
	for i, key := range keys {
		result[i] = pipe.HMGet(ctx, key, "CURR", "TIME")
	}
	if _, err := pipe.Exec(ctx); err != nil && !errors.Is(err, redis.Nil) {
		return errs.Wrap(err)
	}
	var notFoundKey []string
	for i, r := range result {
		val, err := r.Result()
		if len(val) != 2 {
			return errs.WrapMsg(err, "batchGetMaxSeqWithTime invalid result", "key", keys[i], "res", val)
		}
		if val[0] == nil {
			notFoundKey = append(notFoundKey, keys[i])
			continue
		}
		seq, err := s.parseInt64(val[0])
		if err != nil {
			return err
		}
		mill, err := s.parseInt64(val[1])
		if err != nil {
			return err
		}
		seqs[keyConversationID[keys[i]]] = database.SeqTime{Seq: seq, Time: mill}
	}
```

```
	for _, key := range notFoundKey {
		conversationID := keyConversationID[key]
		seq, err := s.GetMaxSeqWithTime(ctx, conversationID)
		if err != nil {
			return err
		}
		seqs[conversationID] = seq
	}
	return nil
}

func (s *seqConversationCacheRedis) GetMaxSeqs(ctx context.Context, conversationIDs []string) (map[string]int64, err
	switch len(conversationIDs) {
	case 0:
		return map[string]int64{}, nil
	case 1:
		return s.getSingleMaxSeq(ctx, conversationIDs[0])
	}
	keys := make([]string, 0, len(conversationIDs))
	keyConversationID := make(map[string]string, len(conversationIDs))
	for _, conversationID := range conversationIDs {
		key := s.getSeqMallocKey(conversationID)
		if _, ok := keyConversationID[key]; ok {
			continue
		}
		keys = append(keys, key)
		keyConversationID[key] = conversationID
	}
	if len(keys) == 1 {
		return s.getSingleMaxSeq(ctx, conversationIDs[0])
	}
	slotKeys, err := groupKeysBySlot(ctx, s.rcClient.GetRedis(), keys)
	if err != nil {
		return nil, err
	}
	seqs := make(map[string]int64, len(conversationIDs))
	for _, keys := range slotKeys {
		if err := s.batchGetMaxSeq(ctx, keys, keyConversationID, seqs); err != nil {
			return nil, err
		}
	}
	return seqs, nil
}

func (s *seqConversationCacheRedis) GetMaxSeqsWithTime(ctx context.Context, conversationIDs []string) (map[string]da
	switch len(conversationIDs) {
	case 0:
		return map[string]database.SeqTime{}, nil
	case 1:
		return s.getSingleMaxSeqWithTime(ctx, conversationIDs[0])
	}
	keys := make([]string, 0, len(conversationIDs))
	keyConversationID := make(map[string]string, len(conversationIDs))
	for _, conversationID := range conversationIDs {
		key := s.getSeqMallocKey(conversationID)
		if _, ok := keyConversationID[key]; ok {
			continue
		}
		keys = append(keys, key)
		keyConversationID[key] = conversationID
	}
	if len(keys) == 1 {
		return s.getSingleMaxSeqWithTime(ctx, conversationIDs[0])
	}
	slotKeys, err := groupKeysBySlot(ctx, s.rcClient.GetRedis(), keys)
	if err != nil {
```

```
		return nil, err
	}
	seqs := make(map[string]database.SeqTime, len(conversationIDs))
	for _, keys := range slotKeys {
		if err := s.batchGetMaxSeqWithTime(ctx, keys, keyConversationID, seqs); err != nil {
			return nil, err
		}
	}
	return seqs, nil
}

func (s *seqConversationCacheRedis) getSeqMallocKey(conversationID string) string {
	return cachekey.GetMallocSeqKey(conversationID)
}

func (s *seqConversationCacheRedis) setSeq(ctx context.Context, key string, owner int64, currSeq int64, lastSeq int6
	if lastSeq < currSeq {
		return 0, errs.New("lastSeq must be greater than currSeq")
	}
	// 0: success
	// 1: success the lock has expired, but has not been locked by anyone else
	// 2: already locked, but not by yourself
	script := `
local key = KEYS[1]
local lockValue = ARGV[1]
local dataSecond = ARGV[2]
local curr_seq = tonumber(ARGV[3])
local last_seq = tonumber(ARGV[4])
local mallocTime = ARGV[5]
if redis.call("EXISTS", key) == 0 then
	redis.call("HSET", key, "CURR", curr_seq, "LAST", last_seq, "TIME", mallocTime)
	redis.call("EXPIRE", key, dataSecond)
	return 1
end
if redis.call("HGET", key, "LOCK") ~= lockValue then
	return 2
end
redis.call("HDEL", key, "LOCK")
redis.call("HSET", key, "CURR", curr_seq, "LAST", last_seq, "TIME", mallocTime)
redis.call("EXPIRE", key, dataSecond)
return 0
`
	result, err := s.rcClient.GetRedis().Eval(ctx, script, []string{key}, owner, int64(s.dataTime/time.Second), currSec
	if err != nil {
		return 0, errs.Wrap(err)
	}
	return result, nil
}

// malloc size=0 is to get the current seq size>0 is to allocate seq
func (s *seqConversationCacheRedis) malloc(ctx context.Context, key string, size int64) ([]int64, error) {
	// 0: success
	// 1: need to obtain and lock
	// 2: already locked
	// 3: exceeded the maximum value and locked
	script := `
local key = KEYS[1]
local size = tonumber(ARGV[1])
local lockSecond = ARGV[2]
local dataSecond = ARGV[3]
local mallocTime = ARGV[4]
local result = {}
if redis.call("EXISTS", key) == 0 then
	local lockValue = math.random(0, 999999999)
	redis.call("HSET", key, "LOCK", lockValue)
	redis.call("EXPIRE", key, lockSecond)
```

```
■table.insert(result, 1)
■table.insert(result, lockValue)
■table.insert(result, mallocTime)
■return result
end
if redis.call("HEXISTS", key, "LOCK") == 1 then
■table.insert(result, 2)
■return result
end
local curr_seq = tonumber(redis.call("HGET", key, "CURR"))
local last_seq = tonumber(redis.call("HGET", key, "LAST"))
if size == 0 then
■redis.call("EXPIRE", key, dataSecond)
■table.insert(result, 0)
■table.insert(result, curr_seq)
■table.insert(result, last_seq)
■local setTime = redis.call("HGET", key, "TIME")
■if setTime then
■■table.insert(result, setTime)■
■else
■■table.insert(result, 0)
■end
■return result
end
local max_seq = curr_seq + size
if max_seq > last_seq then
■local lockValue = math.random(0, 999999999)
■redis.call("HSET", key, "LOCK", lockValue)
■redis.call("HSET", key, "CURR", last_seq)
■redis.call("HSET", key, "TIME", mallocTime)
■redis.call("EXPIRE", key, lockSecond)
■table.insert(result, 3)
■table.insert(result, curr_seq)
■table.insert(result, last_seq)
■table.insert(result, lockValue)
■table.insert(result, mallocTime)
■return result
end
redis.call("HSET", key, "CURR", max_seq)
redis.call("HSET", key, "TIME", ARGV[4])
redis.call("EXPIRE", key, dataSecond)
table.insert(result, 0)
table.insert(result, curr_seq)
table.insert(result, last_seq)
table.insert(result, mallocTime)
return result
`
■result, err := s.rcClient.GetRedis().Eval(ctx, script, []string{key}, size, int64(s.lockTime/time.Second), int64(s.
■if err != nil {
■■■return nil, errs.Wrap(err)
■}
■return result, nil
}

func (s *seqConversationCacheRedis) wait(ctx context.Context) error {
■timer := time.NewTimer(time.Second / 4)
■defer timer.Stop()
■select {
■case <-timer.C:
■■■return nil
■case <-ctx.Done():
■■■return ctx.Err()
■}
}

func (s *seqConversationCacheRedis) setSeqRetry(ctx context.Context, key string, owner int64, currSeq int64, lastSeq
```

```
	for i := 0; i < 10; i++ {
		state, err := s.setSeq(ctx, key, owner, currSeq, lastSeq, mill)
		if err != nil {
			log.ZError(ctx, "set seq cache failed", err, "key", key, "owner", owner, "currSeq", currSeq, "lastSeq", lastSeq,
			if err := s.wait(ctx); err != nil {
				return
			}
			continue
		}
		switch state {
		case 0: // ideal state
		case 1:
			log.ZWarn(ctx, "set seq cache lock not found", nil, "key", key, "owner", owner, "currSeq", currSeq, "lastSeq", l
		case 2:
			log.ZWarn(ctx, "set seq cache lock to be held by someone else", nil, "key", key, "owner", owner, "currSeq", curr
		default:
			log.ZError(ctx, "set seq cache lock unknown state", nil, "key", key, "owner", owner, "currSeq", currSeq, "lastSe
		}
		return
	}
	log.ZError(ctx, "set seq cache retrying still failed", nil, "key", key, "owner", owner, "currSeq", currSeq, "lastSe
}

func (s *seqConversationCacheRedis) getMallocSize(conversationID string, size int64) int64 {
	if size == 0 {
		return 0
	}
	var basicSize int64
	if msgprocessor.IsGroupConversationID(conversationID) {
		basicSize = 100
	} else {
		basicSize = 50
	}
	basicSize += size
	return basicSize
}

func (s *seqConversationCacheRedis) Malloc(ctx context.Context, conversationID string, size int64) (int64, error) {
	seq, _, err := s.mallocTime(ctx, conversationID, size)
	return seq, err
}

func (s *seqConversationCacheRedis) mallocTime(ctx context.Context, conversationID string, size int64) (int64, int64
	if size < 0 {
		return 0, 0, errs.New("size must be greater than 0")
	}
	key := s.getSeqMallocKey(conversationID)
	for i := 0; i < 10; i++ {
		states, err := s.malloc(ctx, key, size)
		if err != nil {
			return 0, 0, err
		}
		switch states[0] {
		case 0: // success
			return states[1], states[3], nil
		case 1: // not found
			mallocSize := s.getMallocSize(conversationID, size)
			seq, err := s.mgo.Malloc(ctx, conversationID, mallocSize)
			if err != nil {
				return 0, 0, err
			}
			s.setSeqRetry(ctx, key, states[1], seq+size, seq+mallocSize, states[2])
			return seq, 0, nil
		case 2: // locked
			if err := s.wait(ctx); err != nil {
				return 0, 0, err
```

```
        }
        continue
    case 3: // exceeded cache max value
        currSeq := states[1]
        lastSeq := states[2]
        mill := states[4]
        mallocSize := s.getMallocSize(conversationID, size)
        seq, err := s.mgo.Malloc(ctx, conversationID, mallocSize)
        if err != nil {
            return 0, 0, err
        }
        if lastSeq == seq {
            s.setSeqRetry(ctx, key, states[3], currSeq+size, seq+mallocSize, mill)
            return currSeq, states[4], nil
        } else {
            log.ZWarn(ctx, "malloc seq not equal cache last seq", nil, "conversationID", conversationID, "currSeq", currSeq
            s.setSeqRetry(ctx, key, states[3], seq+size, seq+mallocSize, mill)
            return seq, mill, nil
        }
    default:
        log.ZError(ctx, "malloc seq unknown state", nil, "state", states[0], "conversationID", conversationID, "size", s
        return 0, 0, errs.New(fmt.Sprintf("unknown state: %d", states[0]))
    }
  }
  log.ZError(ctx, "malloc seq retrying still failed", nil, "conversationID", conversationID, "size", size)
  return 0, 0, errs.New("malloc seq waiting for lock timeout", "conversationID", conversationID, "size", size)
}

func (s *seqConversationCacheRedis) GetMaxSeq(ctx context.Context, conversationID string) (int64, error) {
  return s.Malloc(ctx, conversationID, 0)
}

func (s *seqConversationCacheRedis) GetMaxSeqWithTime(ctx context.Context, conversationID string) (database.SeqTime,
  seq, mill, err := s.mallocTime(ctx, conversationID, 0)
  if err != nil {
    return database.SeqTime{}, err
  }
  return database.SeqTime{Seq: seq, Time: mill}, nil
}

func (s *seqConversationCacheRedis) SetMinSeqs(ctx context.Context, seqs map[string]int64) error {
  keys := make([]string, 0, len(seqs))
  for conversationID, seq := range seqs {
    keys = append(keys, s.getMinSeqKey(conversationID))
    if err := s.mgo.SetMinSeq(ctx, conversationID, seq); err != nil {
      return err
    }
  }
  return DeleteCacheBySlot(ctx, s.rcClient, keys)
}

// GetCacheMaxSeqWithTime only get the existing cache, if there is no cache, no cache will be generated
func (s *seqConversationCacheRedis) GetCacheMaxSeqWithTime(ctx context.Context, conversationIDs []string) (map[strin
  if len(conversationIDs) == 0 {
    return map[string]database.SeqTime{}, nil
  }
  key2conversationID := make(map[string]string)
  keys := make([]string, 0, len(conversationIDs))
  for _, conversationID := range conversationIDs {
    key := s.getSeqMallocKey(conversationID)
    if _, ok := key2conversationID[key]; ok {
      continue
    }
    key2conversationID[key] = conversationID
    keys = append(keys, key)
  }
```

```
	slotKeys, err := groupKeysBySlot(ctx, s.rcClient.GetRedis(), keys)
	if err != nil {
		return nil, err
	}
	res := make(map[string]database.SeqTime)
	for _, keys := range slotKeys {
		if len(keys) == 0 {
			continue
		}
		pipe := s.rcClient.GetRedis().Pipeline()
		cmds := make([]*redis.SliceCmd, 0, len(keys))
		for _, key := range keys {
			cmds = append(cmds, pipe.HMGet(ctx, key, "CURR", "TIME"))
		}
		if _, err := pipe.Exec(ctx); err != nil {
			return nil, errs.Wrap(err)
		}
		for i, cmd := range cmds {
			val, err := cmd.Result()
			if err != nil {
				return nil, err
			}
			if len(val) != 2 {
				return nil, errs.WrapMsg(err, "GetCacheMaxSeqWithTime invalid result", "key", keys[i], "res", val)
			}
			if val[0] == nil {
				continue
			}
			seq, err := s.parseInt64(val[0])
			if err != nil {
				return nil, err
			}
			mill, err := s.parseInt64(val[1])
			if err != nil {
				return nil, err
			}
			conversationID := key2conversationID[keys[i]]
			res[conversationID] = database.SeqTime{Seq: seq, Time: mill}
		}
	}
	return res, nil
}

func (s *seqConversationCacheRedis) parseInt64(val any) (int64, error) {
	switch v := val.(type) {
	case nil:
		return 0, nil
	case int:
		return int64(v), nil
	case int64:
		return v, nil
	case string:
		res, err := strconv.ParseInt(v, 10, 64)
		if err != nil {
			return 0, errs.WrapMsg(err, "invalid string not int64", "value", v)
		}
		return res, nil
	default:
		return 0, errs.New("invalid result not int64", "resType", fmt.Sprintf("%T", v), "value", v)
	}
}
```

pkg/common/storage/cache/redis/seq_conversation_test.go

```go
package redis

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/redis/go-redis/v9"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
	"strconv"
	"sync"
	"sync/atomic"
	"testing"
	"time"
)

func newTestSeq() *seqConversationCacheRedis {
	mgocli, err := mongo.Connect(context.Background(), options.Client().ApplyURI("mongodb://openIM:openIM123@127.0.0.1:
	if err != nil {
		panic(err)
	}
	model, err := mgo.NewSeqConversationMongo(mgocli.Database("openim_v3"))
	if err != nil {
		panic(err)
	}
	opt := &redis.Options{
		Addr:     "127.0.0.1:16379",
		Password: "openIM123",
		DB:       1,
	}
	rdb := redis.NewClient(opt)
	if err := rdb.Ping(context.Background()).Err(); err != nil {
		panic(err)
	}
	return NewSeqConversationCacheRedis(rdb, model).(*seqConversationCacheRedis)
}

func TestSeq(t *testing.T) {
	ts := newTestSeq()
	var (
		wg    sync.WaitGroup
		speed atomic.Int64
	)

	const count = 128
	wg.Add(count)
	for i := 0; i < count; i++ {
		index := i + 1
		go func() {
			defer wg.Done()
			var size int64 = 10
			cID := strconv.Itoa(index * 1)
			for i := 1; ; i++ {
				//first, err := ts.mgo.Malloc(context.Background(), cID, size) // mongo
				first, err := ts.Malloc(context.Background(), cID, size) // redis
				if err != nil {
					t.Logf("[%d-%d] %s %s", index, i, cID, err)
					return
				}
				speed.Add(size)
				_ = first
				//t.Logf("[%d] %d -> %d", i, first+1, first+size)
			}
		}()
	}
```

```
    done := make(chan struct{})

    go func() {
        wg.Wait()
        close(done)
    }()

    ticker := time.NewTicker(time.Second)

    for {
        select {
        case <-done:
            ticker.Stop()
            return
        case <-ticker.C:
            value := speed.Swap(0)
            t.Logf("speed: %d/s", value)
        }
    }
}

func TestDel(t *testing.T) {
    ts := newTestSeq()
    for i := 1; i < 100; i++ {
        var size int64 = 100
        first, err := ts.Malloc(context.Background(), "100", size)
        if err != nil {
            t.Logf("[%d] %s", i, err)
            return
        }
        t.Logf("[%d] %d -> %d", i, first+1, first+size)
        time.Sleep(time.Second)
    }
}

func TestSeqMalloc(t *testing.T) {
    ts := newTestSeq()
    t.Log(ts.GetMaxSeq(context.Background(), "100"))
}

func TestMinSeq(t *testing.T) {
    ts := newTestSeq()
    t.Log(ts.GetMinSeq(context.Background(), "10000000"))
}

func TestMalloc(t *testing.T) {
    ts := newTestSeq()
    t.Log(ts.mallocTime(context.Background(), "10000000", 100))
}

func TestHMGET(t *testing.T) {
    ts := newTestSeq()
    res, err := ts.GetCacheMaxSeqWithTime(context.Background(), []string{"10000000", "123456"})
    if err != nil {
        panic(err)
    }
    t.Log(res)
}

func TestGetMaxSeqWithTime(t *testing.T) {
    ts := newTestSeq()
    t.Log(ts.GetMaxSeqWithTime(context.Background(), "10000000"))
}

func TestGetMaxSeqWithTime1(t *testing.T) {
```

```
	ts := newTestSeq()
	t.Log(ts.GetMaxSeqsWithTime(context.Background(), []string{"10000000", "12345", "111"}))
}

//
//func TestHMGET(t *testing.T) {
//	ts := newTestSeq()
//	res, err := ts.rdb.HMGet(context.Background(), "MALLOC_SEQ:1", "CURR", "TIME1").Result()
//	if err != nil {
//		panic(err)
//	}
//	t.Log(res)
//}
```

pkg/common/storage/cache/redis/seq_user.go

```
package redis

import (
	"context"
	"strconv"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/tools/errs"
	"github.com/redis/go-redis/v9"
)

func NewSeqUserCacheRedis(rdb redis.UniversalClient, mgo database.SeqUser) cache.SeqUser {
	return &seqUserCacheRedis{
		mgo:               mgo,
		readSeqWriteRatio: 100,
		expireTime:        time.Hour * 24 * 7,
		readExpireTime:    time.Hour * 24 * 30,
		rocks:             newRocksCacheClient(rdb),
	}
}

type seqUserCacheRedis struct {
	mgo               database.SeqUser
	rocks             *rocksCacheClient
	expireTime        time.Duration
	readExpireTime    time.Duration
	readSeqWriteRatio int64
}

func (s *seqUserCacheRedis) getSeqUserMaxSeqKey(conversationID string, userID string) string {
	return cachekey.GetSeqUserMaxSeqKey(conversationID, userID)
}

func (s *seqUserCacheRedis) getSeqUserMinSeqKey(conversationID string, userID string) string {
	return cachekey.GetSeqUserMinSeqKey(conversationID, userID)
}

func (s *seqUserCacheRedis) getSeqUserReadSeqKey(conversationID string, userID string) string {
	return cachekey.GetSeqUserReadSeqKey(conversationID, userID)
}

func (s *seqUserCacheRedis) GetUserMaxSeq(ctx context.Context, conversationID string, userID string) (int64, error)
	return getCache(ctx, s.rocks, s.getSeqUserMaxSeqKey(conversationID, userID), s.expireTime, func(ctx context.Context
		return s.mgo.GetUserMaxSeq(ctx, conversationID, userID)
	})
}

func (s *seqUserCacheRedis) SetUserMaxSeq(ctx context.Context, conversationID string, userID string, seq int64) erro
	if err := s.mgo.SetUserMaxSeq(ctx, conversationID, userID, seq); err != nil {
		return err
	}
	return s.rocks.GetClient().TagAsDeleted2(ctx, s.getSeqUserMaxSeqKey(conversationID, userID))
}

func (s *seqUserCacheRedis) GetUserMinSeq(ctx context.Context, conversationID string, userID string) (int64, error)
	return getCache(ctx, s.rocks, s.getSeqUserMinSeqKey(conversationID, userID), s.expireTime, func(ctx context.Context
		return s.mgo.GetUserMinSeq(ctx, conversationID, userID)
	})
}

func (s *seqUserCacheRedis) SetUserMinSeq(ctx context.Context, conversationID string, userID string, seq int64) erro
```

```
	return s.SetUserMinSeqs(ctx, userID, map[string]int64{conversationID: seq})
}

func (s *seqUserCacheRedis) GetUserReadSeq(ctx context.Context, conversationID string, userID string) (int64, error)
	return getCache(ctx, s.rocks, s.getSeqUserReadSeqKey(conversationID, userID), s.readExpireTime, func(ctx context.Co
		return s.mgo.GetUserReadSeq(ctx, conversationID, userID)
	})
}

func (s *seqUserCacheRedis) SetUserReadSeq(ctx context.Context, conversationID string, userID string, seq int64) err
	if s.rocks.GetRedis() == nil {
		return s.SetUserReadSeqToDB(ctx, conversationID, userID, seq)
	}
	dbSeq, err := s.GetUserReadSeq(ctx, conversationID, userID)
	if err != nil {
		return err
	}
	if dbSeq < seq {
		if err := s.rocks.GetClient().RawSet(ctx, s.getSeqUserReadSeqKey(conversationID, userID), strconv.Itoa(int(seq)),
			return errs.Wrap(err)
		}
	}
	return nil
}

func (s *seqUserCacheRedis) SetUserReadSeqToDB(ctx context.Context, conversationID string, userID string, seq int64)
	return s.mgo.SetUserReadSeq(ctx, conversationID, userID, seq)
}

func (s *seqUserCacheRedis) SetUserMinSeqs(ctx context.Context, userID string, seqs map[string]int64) error {
	keys := make([]string, 0, len(seqs))
	for conversationID, seq := range seqs {
		if err := s.mgo.SetUserMinSeq(ctx, conversationID, userID, seq); err != nil {
			return err
		}
		keys = append(keys, s.getSeqUserMinSeqKey(conversationID, userID))
	}
	return DeleteCacheBySlot(ctx, s.rocks, keys)
}

func (s *seqUserCacheRedis) setUserRedisReadSeqs(ctx context.Context, userID string, seqs map[string]int64) error {
	keys := make([]string, 0, len(seqs))
	keySeq := make(map[string]int64)
	for conversationID, seq := range seqs {
		key := s.getSeqUserReadSeqKey(conversationID, userID)
		keys = append(keys, key)
		keySeq[key] = seq
	}
	slotKeys, err := groupKeysBySlot(ctx, s.rocks.GetRedis(), keys)
	if err != nil {
		return err
	}
	for _, keys := range slotKeys {
		pipe := s.rocks.GetRedis().Pipeline()
		for _, key := range keys {
			pipe.HSet(ctx, key, "value", strconv.FormatInt(keySeq[key], 10))
			pipe.Expire(ctx, key, s.readExpireTime)
		}
		if _, err := pipe.Exec(ctx); err != nil {
			return err
		}
	}
	return nil
}

func (s *seqUserCacheRedis) SetUserReadSeqs(ctx context.Context, userID string, seqs map[string]int64) error {
```

```
	if len(seqs) == 0 {
		return nil
	}
	if err := s.setUserRedisReadSeqs(ctx, userID, seqs); err != nil {
		return err
	}
	return nil
}

func (s *seqUserCacheRedis) GetUserReadSeqs(ctx context.Context, userID string, conversationIDs []string) (map[strin
	res, err := batchGetCache2(ctx, s.rocks, s.readExpireTime, conversationIDs, func(conversationID string) string {
		return s.getSeqUserReadSeqKey(conversationID, userID)
	}, func(v *readSeqModel) string {
		return v.ConversationID
	}, func(ctx context.Context, conversationIDs []string) ([]*readSeqModel, error) {
		seqs, err := s.mgo.GetUserReadSeqs(ctx, userID, conversationIDs)
		if err != nil {
			return nil, err
		}
		res := make([]*readSeqModel, 0, len(seqs))
		for conversationID, seq := range seqs {
			res = append(res, &readSeqModel{ConversationID: conversationID, Seq: seq})
		}
		return res, nil
	})
	if err != nil {
		return nil, err
	}
	data := make(map[string]int64)
	for _, v := range res {
		data[v.ConversationID] = v.Seq
	}
	return data, nil
}

var _ BatchCacheCallback[string] = (*readSeqModel)(nil)

type readSeqModel struct {
	ConversationID string
	Seq            int64
}

func (r *readSeqModel) BatchCache(conversationID string) {
	r.ConversationID = conversationID
}

func (r *readSeqModel) UnmarshalJSON(bytes []byte) (err error) {
	r.Seq, err = strconv.ParseInt(string(bytes), 10, 64)
	return
}

func (r *readSeqModel) MarshalJSON() ([]byte, error) {
	return []byte(strconv.FormatInt(r.Seq, 10)), nil
}
```

pkg/common/storage/cache/redis/seq_user_test.go

```go
package redis

import (
	"context"
	"fmt"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	mgo2 "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/redis/go-redis/v9"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
	"log"
	"strconv"
	"sync/atomic"
	"testing"
	"time"
)

func newTestOnline() *userOnline {
	opt := &redis.Options{
		Addr:     "172.16.8.48:16379",
		Password: "openIM123",
		DB:       0,
	}
	rdb := redis.NewClient(opt)
	if err := rdb.Ping(context.Background()).Err(); err != nil {
		panic(err)
	}
	return &userOnline{rdb: rdb, expire: time.Hour, channelName: "user_online"}
}

func TestOnline(t *testing.T) {
	ts := newTestOnline()
	var count atomic.Int64
	for i := 0; i < 64; i++ {
		go func(userID string) {
			var err error
			for i := 0; ; i++ {
				if i%2 == 0 {
					err = ts.SetUserOnline(context.Background(), userID, []int32{5, 6}, []int32{7, 8, 9})
				} else {
					err = ts.SetUserOnline(context.Background(), userID, []int32{1, 2, 3}, []int32{4, 5, 6})
				}
				if err != nil {
					panic(err)
				}
				count.Add(1)
			}
		}(strconv.Itoa(10000 + i))
	}

	ticker := time.NewTicker(time.Second)
	for range ticker.C {
		t.Log(count.Swap(0))
	}
}

func TestGetOnline(t *testing.T) {
	ts := newTestOnline()
	ctx := context.Background()
	pIDs, err := ts.GetOnline(ctx, "10000")
	if err != nil {
		panic(err)
	}
	t.Log(pIDs)
```

```
}

func TestRecvOnline(t *testing.T) {
	ts := newTestOnline()
	ctx := context.Background()
	pubsub := ts.rdb.Subscribe(ctx, cachekey.OnlineChannel)

	_, err := pubsub.Receive(ctx)
	if err != nil {
		log.Fatalf("Could not subscribe: %v", err)
	}

	ch := pubsub.Channel()

	for msg := range ch {
		fmt.Printf("Received message from channel %s: %s\n", msg.Channel, msg.Payload)
	}
}

func TestName1(t *testing.T) {
	opt := &redis.Options{
		Addr:     "172.16.8.48:16379",
		Password: "openIM123",
		DB:       0,
	}
	rdb := redis.NewClient(opt)

	mgo, err := mongo.Connect(context.Background(),
		options.Client().
			ApplyURI("mongodb://openIM:openIM123@172.16.8.48:37017/openim_v3?maxPoolSize=100").
			SetConnectTimeout(5*time.Second))
	if err != nil {
		panic(err)
	}
	model, err := mgo2.NewSeqUserMongo(mgo.Database("openim_v3"))
	if err != nil {
		panic(err)
	}
	seq := NewSeqUserCacheRedis(rdb, model)

	res, err := seq.GetUserReadSeqs(context.Background(), "2110910952", []string{"sg_345762580", "2000", "3000"})
	if err != nil {
		panic(err)
	}
	t.Log(res)

}
```

pkg/common/storage/cache/redis/third.go

```
package redis

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/tools/errs"
	"github.com/redis/go-redis/v9"
)

func NewThirdCache(rdb redis.UniversalClient) cache.ThirdCache {
	return &thirdCache{rdb: rdb}
}

type thirdCache struct {
	rdb redis.UniversalClient
}

func (c *thirdCache) getGetuiTokenKey() string {
	return cachekey.GetGetuiTokenKey()
}

func (c *thirdCache) getGetuiTaskIDKey() string {
	return cachekey.GetGetuiTaskIDKey()
}

func (c *thirdCache) getUserBadgeUnreadCountSumKey(userID string) string {
	return cachekey.GetUserBadgeUnreadCountSumKey(userID)
}

func (c *thirdCache) getFcmAccountTokenKey(account string, platformID int) string {
	return cachekey.GetFcmAccountTokenKey(account, platformID)
}

func (c *thirdCache) SetFcmToken(ctx context.Context, account string, platformID int, fcmToken string, expireTime in
	return errs.Wrap(c.rdb.Set(ctx, c.getFcmAccountTokenKey(account, platformID), fcmToken, time.Duration(expireTime)*t
}

func (c *thirdCache) GetFcmToken(ctx context.Context, account string, platformID int) (string, error) {
	val, err := c.rdb.Get(ctx, c.getFcmAccountTokenKey(account, platformID)).Result()
	if err != nil {
		return "", errs.Wrap(err)
	}
	return val, nil
}

func (c *thirdCache) DelFcmToken(ctx context.Context, account string, platformID int) error {
	return errs.Wrap(c.rdb.Del(ctx, c.getFcmAccountTokenKey(account, platformID)).Err())
}

func (c *thirdCache) IncrUserBadgeUnreadCountSum(ctx context.Context, userID string) (int, error) {
	seq, err := c.rdb.Incr(ctx, c.getUserBadgeUnreadCountSumKey(userID)).Result()

	return int(seq), errs.Wrap(err)
}

func (c *thirdCache) SetUserBadgeUnreadCountSum(ctx context.Context, userID string, value int) error {
	return errs.Wrap(c.rdb.Set(ctx, c.getUserBadgeUnreadCountSumKey(userID), value, 0).Err())
}

func (c *thirdCache) GetUserBadgeUnreadCountSum(ctx context.Context, userID string) (int, error) {
	val, err := c.rdb.Get(ctx, c.getUserBadgeUnreadCountSumKey(userID)).Int()
```

```
	return val, errs.Wrap(err)
}

func (c *thirdCache) SetGetuiToken(ctx context.Context, token string, expireTime int64) error {
	return errs.Wrap(c.rdb.Set(ctx, c.getGetuiTokenKey(), token, time.Duration(expireTime)*time.Second).Err())
}

func (c *thirdCache) GetGetuiToken(ctx context.Context) (string, error) {
	val, err := c.rdb.Get(ctx, c.getGetuiTokenKey()).Result()
	if err != nil {
		return "", errs.Wrap(err)
	}
	return val, nil
}

func (c *thirdCache) SetGetuiTaskID(ctx context.Context, taskID string, expireTime int64) error {
	return errs.Wrap(c.rdb.Set(ctx, c.getGetuiTaskIDKey(), taskID, time.Duration(expireTime)*time.Second).Err())
}

func (c *thirdCache) GetGetuiTaskID(ctx context.Context) (string, error) {
	val, err := c.rdb.Get(ctx, c.getGetuiTaskIDKey()).Result()
	if err != nil {
		return "", errs.Wrap(err)
	}
	return val, nil
}
```

pkg/common/storage/cache/redis/token.go

```
package redis

import (
	"context"
	"encoding/json"
	"strconv"
	"sync"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
)

type tokenCache struct {
	rdb          redis.UniversalClient
	accessExpire time.Duration
	localCache   *config.LocalCache
}

func NewTokenCacheModel(rdb redis.UniversalClient, localCache *config.LocalCache, accessExpire int64) cache.TokenMod
	c := &tokenCache{rdb: rdb, localCache: localCache}
	c.accessExpire = c.getExpireTime(accessExpire)
	return c
}

func (c *tokenCache) SetTokenFlag(ctx context.Context, userID string, platformID int, token string, flag int) error
	key := cachekey.GetTokenKey(userID, platformID)
	if err := c.rdb.HSet(ctx, key, token, flag).Err(); err != nil {
		return errs.Wrap(err)
	}

	if c.localCache != nil {
		c.removeLocalTokenCache(ctx, key)
	}

	return nil
}

// SetTokenFlagEx set token and flag with expire time
func (c *tokenCache) SetTokenFlagEx(ctx context.Context, userID string, platformID int, token string, flag int) erro
	key := cachekey.GetTokenKey(userID, platformID)
	if err := c.rdb.HSet(ctx, key, token, flag).Err(); err != nil {
		return errs.Wrap(err)
	}
	if err := c.rdb.Expire(ctx, key, c.accessExpire).Err(); err != nil {
		return errs.Wrap(err)
	}

	if c.localCache != nil {
		c.removeLocalTokenCache(ctx, key)
	}

	return nil
}

func (c *tokenCache) GetTokensWithoutError(ctx context.Context, userID string, platformID int) (map[string]int, erro
	m, err := c.rdb.HGetAll(ctx, cachekey.GetTokenKey(userID, platformID)).Result()
	if err != nil {
		return nil, errs.Wrap(err)
```

```
	}
	mm := make(map[string]int)
	for k, v := range m {
		state, err := strconv.Atoi(v)
		if err != nil {
			return nil, errs.WrapMsg(err, "redis token value is not int", "value", v, "userID", userID, "platformID", platfo
		}
		mm[k] = state
	}
	return mm, nil
}

func (c *tokenCache) HasTemporaryToken(ctx context.Context, userID string, platformID int, token string) error {
	err := c.rdb.Get(ctx, cachekey.GetTemporaryTokenKey(userID, platformID, token)).Err()
	if err != nil {
		return errs.Wrap(err)
	}
	return nil
}

func (c *tokenCache) GetAllTokensWithoutError(ctx context.Context, userID string) (map[int]map[string]int, error) {
	var (
		res     = make(map[int]map[string]int)
		resLock = sync.Mutex{}
	)

	keys := cachekey.GetAllPlatformTokenKey(userID)
	if err := ProcessKeysBySlot(ctx, c.rdb, keys, func(ctx context.Context, slot int64, keys []string) error {
		pipe := c.rdb.Pipeline()
		mapRes := make([]*redis.MapStringStringCmd, len(keys))
		for i, key := range keys {
			mapRes[i] = pipe.HGetAll(ctx, key)
		}
		_, err := pipe.Exec(ctx)
		if err != nil {
			return err
		}
		for i, m := range mapRes {
			mm := make(map[string]int)
			for k, v := range m.Val() {
				state, err := strconv.Atoi(v)
				if err != nil {
					return errs.WrapMsg(err, "redis token value is not int", "value", v, "userID", userID)
				}
				mm[k] = state
			}
			resLock.Lock()
			res[cachekey.GetPlatformIDByTokenKey(keys[i])] = mm
			resLock.Unlock()
		}
		return nil
	}); err != nil {
		return nil, err
	}
	return res, nil
}

func (c *tokenCache) SetTokenMapByUidPid(ctx context.Context, userID string, platformID int, m map[string]int) error
	mm := make(map[string]any)
	for k, v := range m {
		mm[k] = v
	}

	err := c.rdb.HSet(ctx, cachekey.GetTokenKey(userID, platformID), mm).Err()
	if err != nil {
		return errs.Wrap(err)
```

```
	}

	if c.localCache != nil {
		c.removeLocalTokenCache(ctx, cachekey.GetTokenKey(userID, platformID))
	}

	return nil
}

func (c *tokenCache) BatchSetTokenMapByUidPid(ctx context.Context, tokens map[string]map[string]any) error {
	keys := datautil.Keys(tokens)
	if err := ProcessKeysBySlot(ctx, c.rdb, keys, func(ctx context.Context, slot int64, keys []string) error {
		pipe := c.rdb.Pipeline()
		for k, v := range tokens {
			pipe.HSet(ctx, k, v)
		}
		_, err := pipe.Exec(ctx)
		if err != nil {
			return errs.Wrap(err)
		}
		return nil
	}); err != nil {
		return err
	}

	if c.localCache != nil {
		c.removeLocalTokenCache(ctx, keys...)
	}
	return nil
}

func (c *tokenCache) DeleteTokenByUidPid(ctx context.Context, userID string, platformID int, fields []string) error
	key := cachekey.GetTokenKey(userID, platformID)
	if err := c.rdb.HDel(ctx, key, fields...).Err(); err != nil {
		return errs.Wrap(err)
	}

	if c.localCache != nil {
		c.removeLocalTokenCache(ctx, key)
	}
	return nil
}

func (c *tokenCache) getExpireTime(t int64) time.Duration {
	return time.Hour * 24 * time.Duration(t)
}

// DeleteTokenByTokenMap tokens key is platformID, value is token slice
func (c *tokenCache) DeleteTokenByTokenMap(ctx context.Context, userID string, tokens map[int][]string) error {
	var (
		keys   = make([]string, 0, len(tokens))
		keyMap = make(map[string][]string)
	)
	for k, v := range tokens {
		k1 := cachekey.GetTokenKey(userID, k)
		keys = append(keys, k1)
		keyMap[k1] = v
	}

	if err := ProcessKeysBySlot(ctx, c.rdb, keys, func(ctx context.Context, slot int64, keys []string) error {
		pipe := c.rdb.Pipeline()
		for k, v := range tokens {
			pipe.HDel(ctx, cachekey.GetTokenKey(userID, k), v...)
		}
		_, err := pipe.Exec(ctx)
		if err != nil {
```

```
        return errs.Wrap(err)
    }
    return nil
    }); err != nil {
    return err
    }

    // Remove local cache for the token
    if c.localCache != nil {
    c.removeLocalTokenCache(ctx, keys...)
    }

    return nil
}

func (c *tokenCache) DeleteAndSetTemporary(ctx context.Context, userID string, platformID int, fields []string) erro
    for _, f := range fields {
    k := cachekey.GetTemporaryTokenKey(userID, platformID, f)
    if err := c.rdb.Set(ctx, k, "", time.Minute*5).Err(); err != nil {
        return errs.Wrap(err)
    }
    }
    key := cachekey.GetTokenKey(userID, platformID)
    if err := c.rdb.HDel(ctx, key, fields...).Err(); err != nil {
    return errs.Wrap(err)
    }
    if c.localCache != nil {
    c.removeLocalTokenCache(ctx, key)
    }
    return nil
}

func (c *tokenCache) removeLocalTokenCache(ctx context.Context, keys ...string) {
    if len(keys) == 0 {
    return
    }

    topic := c.localCache.Auth.Topic
    if topic == "" {
    return
    }

    data, err := json.Marshal(keys)
    if err != nil {
    log.ZWarn(ctx, "keys json marshal failed", err, "topic", topic, "keys", keys)
    } else {
    if err := c.rdb.Publish(ctx, topic, string(data)).Err(); err != nil {
        log.ZWarn(ctx, "redis publish cache delete error", err, "topic", topic, "keys", keys)
    }
    }
}
```

pkg/common/storage/cache/redis/user.go

```
package redis

import (
	"context"
	"time"

	"github.com/dtm-labs/rockscache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/redis/go-redis/v9"
)

const (
	userExpireTime            = time.Second * 60 * 60 * 12
	userOlineStatusExpireTime = time.Second * 60 * 60 * 24
	statusMod                 = 501
)

type UserCacheRedis struct {
	cache.BatchDeleter
	rdb        redis.UniversalClient
	userDB     database.User
	expireTime time.Duration
	rcClient   *rocksCacheClient
}

func NewUserCacheRedis(rdb redis.UniversalClient, localCache *config.LocalCache, userDB database.User, options *rock
	rc := newRocksCacheClient(rdb)
	return &UserCacheRedis{
		BatchDeleter: rc.GetBatchDeleter(localCache.User.Topic),
		rdb:          rdb,
		userDB:       userDB,
		expireTime:   userExpireTime,
		rcClient:     rc,
	}
}

func (u *UserCacheRedis) getUserID(user *model.User) string {
	return user.UserID
}

func (u *UserCacheRedis) CloneUserCache() cache.UserCache {
	return &UserCacheRedis{
		BatchDeleter: u.BatchDeleter.Clone(),
		rdb:          u.rdb,
		userDB:       u.userDB,
		expireTime:   u.expireTime,
		rcClient:     u.rcClient,
	}
}

func (u *UserCacheRedis) getUserInfoKey(userID string) string {
	return cachekey.GetUserInfoKey(userID)
}

func (u *UserCacheRedis) getUserGlobalRecvMsgOptKey(userID string) string {
	return cachekey.GetUserGlobalRecvMsgOptKey(userID)
}

func (u *UserCacheRedis) GetUserInfo(ctx context.Context, userID string) (userInfo *model.User, err error) {
	return getCache(ctx, u.rcClient, u.getUserInfoKey(userID), u.expireTime, func(ctx context.Context) (*model.User, er
```

```
		return u.userDB.Take(ctx, userID)
	})
}

func (u *UserCacheRedis) GetUsersInfo(ctx context.Context, userIDs []string) ([]*model.User, error) {
	return batchGetCache2(ctx, u.rcClient, u.expireTime, userIDs, u.getUserInfoKey, u.getUserID, u.userDB.Find)
}

func (u *UserCacheRedis) DelUsersInfo(userIDs ...string) cache.UserCache {
	keys := make([]string, 0, len(userIDs))
	for _, userID := range userIDs {
		keys = append(keys, u.getUserInfoKey(userID))
	}
	cache := u.CloneUserCache()
	cache.AddKeys(keys...)

	return cache
}

func (u *UserCacheRedis) GetUserGlobalRecvMsgOpt(ctx context.Context, userID string) (opt int, err error) {
	return getCache(
		ctx,
		u.rcClient,
		u.getUserGlobalRecvMsgOptKey(userID),
		u.expireTime,
		func(ctx context.Context) (int, error) {
			return u.userDB.GetUserGlobalRecvMsgOpt(ctx, userID)
		},
	)
}

func (u *UserCacheRedis) DelUsersGlobalRecvMsgOpt(userIDs ...string) cache.UserCache {
	keys := make([]string, 0, len(userIDs))
	for _, userID := range userIDs {
		keys = append(keys, u.getUserGlobalRecvMsgOptKey(userID))
	}
	cache := u.CloneUserCache()
	cache.AddKeys(keys...)

	return cache
}
```

pkg/common/storage/versionctx

pkg/common/storage/versionctx/rpc.go

```
package versionctx

import (
	"context"
	"google.golang.org/grpc"
)

func EnableVersionCtx() grpc.ServerOption {
	return grpc.ChainUnaryInterceptor(enableVersionCtxInterceptor)
}

func enableVersionCtxInterceptor(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.UnaryHandler
	return handler(WithVersionLog(ctx), req)
}
```

pkg/common/storage/versionctx/version.go

```go
package versionctx

import (
	"context"
	tablerelation "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"sync"
)

type Collection struct {
	Name string
	Doc  *tablerelation.VersionLog
}

type versionKey struct{}

func WithVersionLog(ctx context.Context) context.Context {
	return context.WithValue(ctx, versionKey{}, &VersionLog{})
}

func GetVersionLog(ctx context.Context) *VersionLog {
	if v, ok := ctx.Value(versionKey{}).(*VersionLog); ok {
		return v
	}
	return nil
}

type VersionLog struct {
	lock sync.Mutex
	data []Collection
}

func (v *VersionLog) Append(data ...Collection) {
	if v == nil || len(data) == 0 {
		return
	}
	v.lock.Lock()
	defer v.lock.Unlock()
	v.data = append(v.data, data...)
}

func (v *VersionLog) Get() []Collection {
	if v == nil {
		return nil
	}
	v.lock.Lock()
	defer v.lock.Unlock()
	return v.data
}
```

pkg/common/webhook

pkg/common/webhook/condition.go

```go
package webhook

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
)

func WithCondition(ctx context.Context, before *config.BeforeConfig, callback func(context.Context) error) error {
	if !before.Enable {
		return nil
	}
	return callback(ctx)
}
```

pkg/common/webhook/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package webhook // import "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
```

pkg/common/webhook/http_client.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package webhook

import (
	"context"
	"encoding/json"
	"net/http"
	"net/url"

	"github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/mq/memamq"
	"github.com/openimsdk/tools/utils/httputil"
)

type Client struct {
	client *httputil.HTTPClient
	url    string
	queue  *memamq.MemoryQueue
}

const (
	webhookWorkerCount = 2
	webhookBufferSize  = 100

	Key = "key"
)

func NewWebhookClient(url string, options ...*memamq.MemoryQueue) *Client {
	var queue *memamq.MemoryQueue
	if len(options) > 0 && options[0] != nil {
		queue = options[0]
	} else {
		queue = memamq.NewMemoryQueue(webhookWorkerCount, webhookBufferSize)
	}

	http.DefaultTransport.(*http.Transport).MaxConnsPerHost = 100 // Enhance the default number of max connections per

	return &Client{
		client: httputil.NewHTTPClient(httputil.NewClientConfig()),
		url:    url,
		queue:  queue,
	}
}

func (c *Client) SyncPost(ctx context.Context, command string, req callbackstruct.CallbackReq, resp callbackstruct.C
	return c.post(ctx, command, req, resp, before.Timeout)
```

```
}

func (c *Client) AsyncPost(ctx context.Context, command string, req callbackstruct.CallbackReq, resp callbackstruct.
    if after.Enable {
        c.queue.Push(func() { c.post(ctx, command, req, resp, after.Timeout) })
    }
}

func (c *Client) AsyncPostWithQuery(ctx context.Context, command string, req callbackstruct.CallbackReq, resp callba
    if after.Enable {
        c.queue.Push(func() { c.postWithQuery(ctx, command, req, resp, after.Timeout, queryParams) })
    }
}

func (c *Client) post(ctx context.Context, command string, input interface{}, output callbackstruct.CallbackResp, ti
    ctx = mcontext.WithMustInfoCtx([]string{mcontext.GetOperationID(ctx), mcontext.GetOpUserID(ctx), mcontext.GetOpUser
    fullURL := c.url + "/" + command
    log.ZInfo(ctx, "webhook", "url", fullURL, "input", input, "config", timeout)
    operationID, _ := ctx.Value(constant.OperationID).(string)
    b, err := c.client.Post(ctx, fullURL, map[string]string{constant.OperationID: operationID}, input, timeout)
    if err != nil {
        return servererrs.ErrNetwork.WrapMsg(err.Error(), "post url", fullURL)
    }
    if err = json.Unmarshal(b, output); err != nil {
        return servererrs.ErrData.WithDetail(err.Error() + " response format error")
    }
    if err := output.Parse(); err != nil {
        return err
    }
    log.ZInfo(ctx, "webhook success", "url", fullURL, "input", input, "response", string(b))
    return nil
}

func (c *Client) postWithQuery(ctx context.Context, command string, input interface{}, output callbackstruct.Callbac
    ctx = mcontext.WithMustInfoCtx([]string{mcontext.GetOperationID(ctx), mcontext.GetOpUserID(ctx), mcontext.GetOpUser
    fullURL := c.url + "/" + command

    parsedURL, err := url.Parse(fullURL)
    if err != nil {
        return servererrs.ErrNetwork.WrapMsg(err.Error(), "failed to parse URL", fullURL)
    }

    query := parsedURL.Query()

    operationID, _ := ctx.Value(constant.OperationID).(string)

    for key, value := range queryParams {
        query.Set(key, value)
    }

    parsedURL.RawQuery = query.Encode()

    fullURL = parsedURL.String()
    log.ZInfo(ctx, "webhook", "url", fullURL, "input", input, "config", timeout)

    b, err := c.client.Post(ctx, fullURL, map[string]string{constant.OperationID: operationID}, input, timeout)
    if err != nil {
        return servererrs.ErrNetwork.WrapMsg(err.Error(), "post url", fullURL)
    }

    if err = json.Unmarshal(b, output); err != nil {
        return servererrs.ErrData.WithDetail(err.Error() + " response format error")
    }
    if err := output.Parse(); err != nil {
        return err
    }
```

```
    log.ZInfo(ctx, "webhook success", "url", fullURL, "input", input, "response", string(b))
    return nil
}
```

pkg/common/webhook/http_client_test.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package webhook
```

pkg/common/cmd

pkg/common/cmd/api.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/internal/api"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
    "github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
    "github.com/openimsdk/open-im-server/v3/version"
    "github.com/openimsdk/tools/system/program"
    "github.com/spf13/cobra"
)

type ApiCmd struct {
    *RootCmd
    ctx       context.Context
    configMap map[string]any
    apiConfig *api.Config
}

func NewApiCmd() *ApiCmd {
    var apiConfig api.Config
    ret := &ApiCmd{apiConfig: &apiConfig}
    ret.configMap = map[string]any{
        config.DiscoveryConfigFilename:          &apiConfig.Discovery,
        config.KafkaConfigFileName:              &apiConfig.Kafka,
        config.LocalCacheConfigFileName:         &apiConfig.LocalCache,
        config.LogConfigFileName:                &apiConfig.Log,
        config.MinioConfigFileName:              &apiConfig.Minio,
        config.MongodbConfigFileName:            &apiConfig.Mongo,
        config.NotificationFileName:             &apiConfig.Notification,
        config.OpenIMAPICfgFileName:             &apiConfig.API,
        config.OpenIMCronTaskCfgFileName:        &apiConfig.CronTask,
        config.OpenIMMsgGatewayCfgFileName:      &apiConfig.MsgGateway,
        config.OpenIMMsgTransferCfgFileName:     &apiConfig.MsgTransfer,
        config.OpenIMPushCfgFileName:            &apiConfig.Push,
        config.OpenIMRPCAuthCfgFileName:         &apiConfig.Auth,
        config.OpenIMRPCConversationCfgFileName: &apiConfig.Conversation,
        config.OpenIMRPCFriendCfgFileName:       &apiConfig.Friend,
        config.OpenIMRPCGroupCfgFileName:        &apiConfig.Group,
        config.OpenIMRPCMsgCfgFileName:          &apiConfig.Msg,
        config.OpenIMRPCThirdCfgFileName:        &apiConfig.Third,
        config.OpenIMRPCUserCfgFileName:         &apiConfig.User,
        config.RedisConfigFileName:              &apiConfig.Redis,
        config.ShareFileName:                    &apiConfig.Share,
        config.WebhooksConfigFileName:           &apiConfig.Webhooks,
```

```
	}
	ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
	ret.ctx = context.WithValue(context.Background(), "version", version.Version)
	ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
		apiConfig.ConfigPath = config.Path(ret.configPath)
		return ret.runE()
	}
	return ret
}

func (a *ApiCmd) Exec() error {
	return a.Execute()
}

func (a *ApiCmd) runE() error {
	a.apiConfig.Index = config.Index(a.Index())
	prometheus := config.Prometheus{
		Enable: a.apiConfig.API.Prometheus.Enable,
		Ports:  a.apiConfig.API.Prometheus.Ports,
	}
	return startrpc.Start(
		a.ctx, &a.apiConfig.Discovery,
		nil,
		nil,
		// &a.apiConfig.API.RateLimiter,
		&prometheus,
		a.apiConfig.API.Api.ListenIP, "",
		a.apiConfig.API.Prometheus.AutoSetPorts,
		nil, int(a.apiConfig.Index),
		prommetrics.APIKeyName,
		&a.apiConfig.Notification,
		a.apiConfig,
		[]string{},
		[]string{},
		api.Start,
	)
}
```

pkg/common/cmd/auth.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/internal/rpc/auth"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
    "github.com/openimsdk/open-im-server/v3/version"
    "github.com/openimsdk/tools/system/program"
    "github.com/spf13/cobra"
)

type AuthRpcCmd struct {
    *RootCmd
    ctx        context.Context
    configMap  map[string]any
    authConfig *auth.Config
}

func NewAuthRpcCmd() *AuthRpcCmd {
    var authConfig auth.Config
    ret := &AuthRpcCmd{authConfig: &authConfig}
    ret.configMap = map[string]any{
        config.OpenIMRPCAuthCfgFileName: &authConfig.RpcConfig,
        config.RedisConfigFileName:      &authConfig.RedisConfig,
        config.MongodbConfigFileName:    &authConfig.MongoConfig,
        config.ShareFileName:            &authConfig.Share,
        config.LocalCacheConfigFileName: &authConfig.LocalCacheConfig,
        config.DiscoveryConfigFilename:  &authConfig.Discovery,
    }
    ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
    ret.ctx = context.WithValue(context.Background(), "version", version.Version)
    ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
        return ret.runE()
    }

    return ret
}

func (a *AuthRpcCmd) Exec() error {
    return a.Execute()
}

func (a *AuthRpcCmd) runE() error {
    return startrpc.Start(a.ctx, &a.authConfig.Discovery, &a.authConfig.RpcConfig.CircuitBreaker, &a.authConfig.RpcCon
        a.authConfig.RpcConfig.RPC.RegisterIP, a.authConfig.RpcConfig.RPC.AutoSetPorts, a.authConfig.RpcConfig.RPC.Ports,
        a.Index(), a.authConfig.Discovery.RpcService.Auth, nil, a.authConfig,
        []string{
            a.authConfig.RpcConfig.GetConfigFileName(),
```

```
        a.authConfig.Share.GetConfigFileName(),
        a.authConfig.RedisConfig.GetConfigFileName(),
        a.authConfig.Discovery.GetConfigFileName(),
    },
    []string{
        a.authConfig.Discovery.RpcService.MessageGateway,
    },
    auth.Start)
}
```

pkg/common/cmd/conversation.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/rpc/conversation"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/system/program"
	"github.com/spf13/cobra"
)

type ConversationRpcCmd struct {
	*RootCmd
	ctx                context.Context
	configMap          map[string]any
	conversationConfig *conversation.Config
}

func NewConversationRpcCmd() *ConversationRpcCmd {
	var conversationConfig conversation.Config
	ret := &ConversationRpcCmd{conversationConfig: &conversationConfig}
	ret.configMap = map[string]any{
		config.OpenIMRPCConversationCfgFileName: &conversationConfig.RpcConfig,
		config.RedisConfigFileName:              &conversationConfig.RedisConfig,
		config.MongodbConfigFileName:            &conversationConfig.MongodbConfig,
		config.ShareFileName:                    &conversationConfig.Share,
		config.NotificationFileName:             &conversationConfig.NotificationConfig,
		config.WebhooksConfigFileName:           &conversationConfig.WebhooksConfig,
		config.LocalCacheConfigFileName:         &conversationConfig.LocalCacheConfig,
		config.DiscoveryConfigFilename:          &conversationConfig.Discovery,
	}
	ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
	ret.ctx = context.WithValue(context.Background(), "version", version.Version)
	ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
		return ret.runE()
	}
	return ret
}

func (a *ConversationRpcCmd) Exec() error {
	return a.Execute()
}

func (a *ConversationRpcCmd) runE() error {
	return startrpc.Start(a.ctx, &a.conversationConfig.Discovery, &a.conversationConfig.RpcConfig.CircuitBreaker, &a.co
		a.conversationConfig.RpcConfig.RPC.RegisterIP, a.conversationConfig.RpcConfig.RPC.AutoSetPorts, a.conversationCon
		a.Index(), a.conversationConfig.Discovery.RpcService.Conversation, &a.conversationConfig.NotificationConfig, a.co
		[]string{
```

```
        a.conversationConfig.RpcConfig.GetConfigFileName(),
        a.conversationConfig.RedisConfig.GetConfigFileName(),
        a.conversationConfig.MongodbConfig.GetConfigFileName(),
        a.conversationConfig.NotificationConfig.GetConfigFileName(),
        a.conversationConfig.Share.GetConfigFileName(),
        a.conversationConfig.LocalCacheConfig.GetConfigFileName(),
        a.conversationConfig.WebhooksConfig.GetConfigFileName(),
        a.conversationConfig.Discovery.GetConfigFileName(),
    }, nil,
    conversation.Start)
}
```

pkg/common/cmd/cron_task.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/tools/cron"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/system/program"
	"github.com/spf13/cobra"
)

type CronTaskCmd struct {
	*RootCmd
	ctx            context.Context
	configMap      map[string]any
	cronTaskConfig *cron.Config
}

func NewCronTaskCmd() *CronTaskCmd {
	var cronTaskConfig cron.Config
	ret := &CronTaskCmd{cronTaskConfig: &cronTaskConfig}
	ret.configMap = map[string]any{
		config.OpenIMCronTaskCfgFileName: &cronTaskConfig.CronTask,
		config.ShareFileName:             &cronTaskConfig.Share,
		config.DiscoveryConfigFilename:   &cronTaskConfig.Discovery,
	}
	ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
	ret.ctx = context.WithValue(context.Background(), "version", version.Version)
	ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
		return ret.runE()
	}
	return ret
}

func (a *CronTaskCmd) Exec() error {
	return a.Execute()
}

func (a *CronTaskCmd) runE() error {
	var prometheus config.Prometheus
	return startrpc.Start(
		a.ctx, &a.cronTaskConfig.Discovery,
		nil,
		nil,
		&prometheus,
		"", "",
		true,
		nil, 0,
```

```
    "",
    nil,
    a.cronTaskConfig,
    []string{},
    []string{},
    cron.Start,
  )
}
```

pkg/common/cmd/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd // import "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
```

pkg/common/cmd/friend.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/internal/rpc/relation"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
    "github.com/openimsdk/open-im-server/v3/version"
    "github.com/openimsdk/tools/system/program"
    "github.com/spf13/cobra"
)

type FriendRpcCmd struct {
    *RootCmd
    ctx            context.Context
    configMap      map[string]any
    relationConfig *relation.Config
}

func NewFriendRpcCmd() *FriendRpcCmd {
    var relationConfig relation.Config
    ret := &FriendRpcCmd{relationConfig: &relationConfig}
    ret.configMap = map[string]any{
        config.OpenIMRPCFriendCfgFileName: &relationConfig.RpcConfig,
        config.RedisConfigFileName:        &relationConfig.RedisConfig,
        config.MongodbConfigFileName:      &relationConfig.MongodbConfig,
        config.ShareFileName:              &relationConfig.Share,
        config.NotificationFileName:       &relationConfig.NotificationConfig,
        config.WebhooksConfigFileName:     &relationConfig.WebhooksConfig,
        config.LocalCacheConfigFileName:   &relationConfig.LocalCacheConfig,
        config.DiscoveryConfigFilename:    &relationConfig.Discovery,
    }
    ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
    ret.ctx = context.WithValue(context.Background(), "version", version.Version)
    ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
        return ret.runE()
    }
    return ret
}

func (a *FriendRpcCmd) Exec() error {
    return a.Execute()
}

func (a *FriendRpcCmd) runE() error {
    return startrpc.Start(a.ctx, &a.relationConfig.Discovery, &a.relationConfig.RpcConfig.CircuitBreaker, &a.relationCo
        a.relationConfig.RpcConfig.RPC.RegisterIP, a.relationConfig.RpcConfig.RPC.AutoSetPorts, a.relationConfig.RpcConfi
        a.Index(), a.relationConfig.Discovery.RpcService.Friend, &a.relationConfig.NotificationConfig, a.relationConfig,
        []string{
```

```
        a.relationConfig.RpcConfig.GetConfigFileName(),
        a.relationConfig.RedisConfig.GetConfigFileName(),
        a.relationConfig.MongodbConfig.GetConfigFileName(),
        a.relationConfig.NotificationConfig.GetConfigFileName(),
        a.relationConfig.Share.GetConfigFileName(),
        a.relationConfig.WebhooksConfig.GetConfigFileName(),
        a.relationConfig.LocalCacheConfig.GetConfigFileName(),
        a.relationConfig.Discovery.GetConfigFileName(),
    }, nil,
    relation.Start)
}
```

pkg/common/cmd/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
■"context"

■"github.com/openimsdk/open-im-server/v3/internal/rpc/group"
■"github.com/openimsdk/open-im-server/v3/pkg/common/config"
■"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
■"github.com/openimsdk/open-im-server/v3/pkg/common/storage/versionctx"
■"github.com/openimsdk/open-im-server/v3/version"
■"github.com/openimsdk/tools/system/program"
■"github.com/spf13/cobra"
)

type GroupRpcCmd struct {
■*RootCmd
■ctx         context.Context
■configMap   map[string]any
■groupConfig *group.Config
}

func NewGroupRpcCmd() *GroupRpcCmd {
■var groupConfig group.Config
■ret := &GroupRpcCmd{groupConfig: &groupConfig}
■ret.configMap = map[string]any{
■■config.OpenIMRPCGroupCfgFileName: &groupConfig.RpcConfig,
■■config.RedisConfigFileName:       &groupConfig.RedisConfig,
■■config.MongodbConfigFileName:     &groupConfig.MongodbConfig,
■■config.ShareFileName:             &groupConfig.Share,
■■config.NotificationFileName:      &groupConfig.NotificationConfig,
■■config.WebhooksConfigFileName:    &groupConfig.WebhooksConfig,
■■config.LocalCacheConfigFileName:  &groupConfig.LocalCacheConfig,
■■config.DiscoveryConfigFilename:   &groupConfig.Discovery,
■}
■ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
■ret.ctx = context.WithValue(context.Background(), "version", version.Version)
■ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
■■return ret.runE()
■}
■return ret
}

func (a *GroupRpcCmd) Exec() error {
■return a.Execute()
}

func (a *GroupRpcCmd) runE() error {
■return startrpc.Start(a.ctx, &a.groupConfig.Discovery, &a.groupConfig.RpcConfig.CircuitBreaker, &a.groupConfig.RpcC
■■a.groupConfig.RpcConfig.RPC.RegisterIP, a.groupConfig.RpcConfig.RPC.AutoSetPorts, a.groupConfig.RpcConfig.RPC.Por
■■a.Index(), a.groupConfig.Discovery.RpcService.Group, &a.groupConfig.NotificationConfig, a.groupConfig,
```

```
  []string{
    a.groupConfig.RpcConfig.GetConfigFileName(),
    a.groupConfig.RedisConfig.GetConfigFileName(),
    a.groupConfig.MongodbConfig.GetConfigFileName(),
    a.groupConfig.NotificationConfig.GetConfigFileName(),
    a.groupConfig.Share.GetConfigFileName(),
    a.groupConfig.WebhooksConfig.GetConfigFileName(),
    a.groupConfig.LocalCacheConfig.GetConfigFileName(),
    a.groupConfig.Discovery.GetConfigFileName(),
  }, nil,
  group.Start, versionctx.EnableVersionCtx())
}
```

pkg/common/cmd/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
"context"

"github.com/openimsdk/open-im-server/v3/internal/rpc/msg"
"github.com/openimsdk/open-im-server/v3/pkg/common/config"
"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
"github.com/openimsdk/open-im-server/v3/version"
"github.com/openimsdk/tools/system/program"
"github.com/spf13/cobra"
)

type MsgRpcCmd struct {
*RootCmd
ctx       context.Context
configMap map[string]any
msgConfig *msg.Config
}

func NewMsgRpcCmd() *MsgRpcCmd {
var msgConfig msg.Config
ret := &MsgRpcCmd{msgConfig: &msgConfig}
ret.configMap = map[string]any{
config.OpenIMRPCMsgCfgFileName:  &msgConfig.RpcConfig,
config.RedisConfigFileName:      &msgConfig.RedisConfig,
config.MongodbConfigFileName:    &msgConfig.MongodbConfig,
config.KafkaConfigFileName:      &msgConfig.KafkaConfig,
config.ShareFileName:            &msgConfig.Share,
config.NotificationFileName:     &msgConfig.NotificationConfig,
config.WebhooksConfigFileName:   &msgConfig.WebhooksConfig,
config.LocalCacheConfigFileName: &msgConfig.LocalCacheConfig,
config.DiscoveryConfigFilename:  &msgConfig.Discovery,
}
ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
ret.ctx = context.WithValue(context.Background(), "version", version.Version)
ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
return ret.runE()
}
return ret
}

func (a *MsgRpcCmd) Exec() error {
return a.Execute()
}

func (a *MsgRpcCmd) runE() error {
return startrpc.Start(a.ctx, &a.msgConfig.Discovery, &a.msgConfig.RpcConfig.CircuitBreaker, &a.msgConfig.RpcConfig.
a.msgConfig.RpcConfig.RPC.RegisterIP, a.msgConfig.RpcConfig.RPC.AutoSetPorts, a.msgConfig.RpcConfig.RPC.Ports,
a.Index(), a.msgConfig.Discovery.RpcService.Msg, &a.msgConfig.NotificationConfig, a.msgConfig,
```

```
    []string{
        a.msgConfig.RpcConfig.GetConfigFileName(),
        a.msgConfig.RedisConfig.GetConfigFileName(),
        a.msgConfig.MongodbConfig.GetConfigFileName(),
        a.msgConfig.KafkaConfig.GetConfigFileName(),
        a.msgConfig.NotificationConfig.GetConfigFileName(),
        a.msgConfig.Share.GetConfigFileName(),
        a.msgConfig.WebhooksConfig.GetConfigFileName(),
        a.msgConfig.LocalCacheConfig.GetConfigFileName(),
        a.msgConfig.Discovery.GetConfigFileName(),
    }, nil,
    msg.Start)
}
```

pkg/common/cmd/msg_gateway.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/msggateway"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
	"github.com/openimsdk/open-im-server/v3/version"

	"github.com/openimsdk/tools/system/program"
	"github.com/spf13/cobra"
)

type MsgGatewayCmd struct {
	*RootCmd
	ctx              context.Context
	configMap        map[string]any
	msgGatewayConfig *msggateway.Config
}

func NewMsgGatewayCmd() *MsgGatewayCmd {
	var msgGatewayConfig msggateway.Config
	ret := &MsgGatewayCmd{msgGatewayConfig: &msgGatewayConfig}
	ret.configMap = map[string]any{
		config.OpenIMMsgGatewayCfgFileName: &msgGatewayConfig.MsgGateway,
		config.ShareFileName:               &msgGatewayConfig.Share,
		config.RedisConfigFileName:         &msgGatewayConfig.RedisConfig,
		config.WebhooksConfigFileName:      &msgGatewayConfig.WebhooksConfig,
		config.DiscoveryConfigFilename:     &msgGatewayConfig.Discovery,
	}
	ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
	ret.ctx = context.WithValue(context.Background(), "version", version.Version)
	ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
		return ret.runE()
	}
	return ret
}

func (m *MsgGatewayCmd) Exec() error {
	return m.Execute()
}

func (m *MsgGatewayCmd) runE() error {
	m.msgGatewayConfig.Index = config.Index(m.Index())
	rpc := m.msgGatewayConfig.MsgGateway.RPC
	var prometheus config.Prometheus
	return startrpc.Start(
		m.ctx, &m.msgGatewayConfig.Discovery,
		&m.msgGatewayConfig.MsgGateway.CircuitBreaker,
```

```
    &m.msgGatewayConfig.MsgGateway.RateLimiter,
    &prometheus,
    rpc.ListenIP, rpc.RegisterIP,
    rpc.AutoSetPorts,
    rpc.Ports, int(m.msgGatewayConfig.Index),
    m.msgGatewayConfig.Discovery.RpcService.MessageGateway,
    nil,
    m.msgGatewayConfig,
    []string{},
    []string{},
    msggateway.Start,
  )
}
```

pkg/common/cmd/msg_gateway_test.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
	"math"
	"testing"

	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/utils/jsonutil"
	"github.com/stretchr/testify/mock"
	"go.mongodb.org/mongo-driver/bson/primitive"
)

// MockRootCmd is a mock type for the RootCmd type
type MockRootCmd struct {
	mock.Mock
}

func (m *MockRootCmd) Execute() error {
	args := m.Called()
	return args.Error(0)
}

func TestName(t *testing.T) {
	resp := &apiresp.ApiResponse{
		ErrCode: 1234,
		ErrMsg:  "test",
		ErrDlt:  "4567",
		Data: &auth.GetUserTokenResp{
			Token:             "1234567",
			ExpireTimeSeconds: math.MaxInt64,
		},
	}
	data, err := resp.MarshalJSON()
	if err != nil {
		panic(err)
	}
	t.Log(string(data))

	var rReso apiresp.ApiResponse
	rReso.Data = &auth.GetUserTokenResp{}

	if err := jsonutil.JsonUnmarshal(data, &rReso); err != nil {
		panic(err)
	}

	t.Logf("%+v\n", rReso)

}
```

```
func TestName1(t *testing.T) {
	t.Log(primitive.NewObjectID().String())
	t.Log(primitive.NewObjectID().Hex())

}
```

pkg/common/cmd/msg_transfer.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/msgtransfer"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/system/program"
	"github.com/spf13/cobra"
)

type MsgTransferCmd struct {
	*RootCmd
	ctx               context.Context
	configMap         map[string]any
	msgTransferConfig *msgtransfer.Config
}

func NewMsgTransferCmd() *MsgTransferCmd {
	var msgTransferConfig msgtransfer.Config
	ret := &MsgTransferCmd{msgTransferConfig: &msgTransferConfig}
	ret.configMap = map[string]any{
		config.OpenIMMsgTransferCfgFileName: &msgTransferConfig.MsgTransfer,
		config.RedisConfigFileName:          &msgTransferConfig.RedisConfig,
		config.MongodbConfigFileName:        &msgTransferConfig.MongodbConfig,
		config.KafkaConfigFileName:          &msgTransferConfig.KafkaConfig,
		config.ShareFileName:                &msgTransferConfig.Share,
		config.WebhooksConfigFileName:       &msgTransferConfig.WebhooksConfig,
		config.DiscoveryConfigFilename:      &msgTransferConfig.Discovery,
	}
	ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
	ret.ctx = context.WithValue(context.Background(), "version", version.Version)
	ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
		return ret.runE()
	}
	return ret
}

func (m *MsgTransferCmd) Exec() error {
	return m.Execute()
}

func (m *MsgTransferCmd) runE() error {
	m.msgTransferConfig.Index = config.Index(m.Index())
	var prometheus config.Prometheus
	return startrpc.Start(
		m.ctx, &m.msgTransferConfig.Discovery,
```

```
    &m.msgTransferConfig.MsgTransfer.CircuitBreaker,
    &m.msgTransferConfig.MsgTransfer.RateLimiter,
    &prometheus,
    "", "",
    true,
    nil, int(m.msgTransferConfig.Index),
    prommetrics.MessageTransferKeyName,
    nil,
    m.msgTransferConfig,
    []string{},
    []string{},
    msgtransfer.Start,
  )
}
```

pkg/common/cmd/msg_utils.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/spf13/cobra"
)

type MsgUtilsCmd struct {
    cobra.Command
}

func (m *MsgUtilsCmd) AddUserIDFlag() {
    m.Command.PersistentFlags().StringP("userID", "u", "", "openIM userID")
}
func (m *MsgUtilsCmd) AddIndexFlag() {
    m.Command.PersistentFlags().IntP(config.FlagTransferIndex, "i", 0, "process startup sequence number")
}

func (m *MsgUtilsCmd) AddConfigDirFlag() {
    m.Command.PersistentFlags().StringP(config.FlagConf, "c", "", "path of config directory")

}

func (m *MsgUtilsCmd) getUserIDFlag(cmdLines *cobra.Command) string {
    userID, _ := cmdLines.Flags().GetString("userID")
    return userID
}

func (m *MsgUtilsCmd) AddFixAllFlag() {
    m.Command.PersistentFlags().BoolP("fixAll", "f", false, "openIM fix all seqs")
}

/* func (m *MsgUtilsCmd) getFixAllFlag(cmdLines *cobra.Command) bool {
    fixAll, _ := cmdLines.Flags().GetBool("fixAll")
    return fixAll
} */

func (m *MsgUtilsCmd) AddClearAllFlag() {
    m.Command.PersistentFlags().BoolP("clearAll", "", false, "openIM clear all seqs")
}

/* func (m *MsgUtilsCmd) getClearAllFlag(cmdLines *cobra.Command) bool {
    clearAll, _ := cmdLines.Flags().GetBool("clearAll")
    return clearAll
} */

func (m *MsgUtilsCmd) AddSuperGroupIDFlag() {
    m.Command.PersistentFlags().StringP("superGroupID", "g", "", "openIM superGroupID")
}
```

```
func (m *MsgUtilsCmd) getSuperGroupIDFlag(cmdLines *cobra.Command) string {
	superGroupID, _ := cmdLines.Flags().GetString("superGroupID")
	return superGroupID
}

func (m *MsgUtilsCmd) AddBeginSeqFlag() {
	m.Command.PersistentFlags().Int64P("beginSeq", "b", 0, "openIM beginSeq")
}

/* func (m *MsgUtilsCmd) getBeginSeqFlag(cmdLines *cobra.Command) int64 {
	beginSeq, _ := cmdLines.Flags().GetInt64("beginSeq")
	return beginSeq
} */

func (m *MsgUtilsCmd) AddLimitFlag() {
	m.Command.PersistentFlags().Int64P("limit", "l", 0, "openIM limit")
}

/* func (m *MsgUtilsCmd) getLimitFlag(cmdLines *cobra.Command) int64 {
	limit, _ := cmdLines.Flags().GetInt64("limit")
	return limit
} */

func (m *MsgUtilsCmd) Execute() error {
	return m.Command.Execute()
}

func NewMsgUtilsCmd(use, short string, args cobra.PositionalArgs) *MsgUtilsCmd {
	return &MsgUtilsCmd{
		Command: cobra.Command{
			Use:   use,
			Short: short,
			Args:  args,
		},
	}
}

type GetCmd struct {
	*MsgUtilsCmd
}

func NewGetCmd() *GetCmd {
	return &GetCmd{
		NewMsgUtilsCmd("get [resource]", "get action", cobra.MatchAll(cobra.ExactArgs(1), cobra.OnlyValidArgs)),
	}
}

type FixCmd struct {
	*MsgUtilsCmd
}

func NewFixCmd() *FixCmd {
	return &FixCmd{
		NewMsgUtilsCmd("fix [resource]", "fix action", cobra.MatchAll(cobra.ExactArgs(1), cobra.OnlyValidArgs)),
	}
}

type ClearCmd struct {
	*MsgUtilsCmd
}

func NewClearCmd() *ClearCmd {
	return &ClearCmd{
		NewMsgUtilsCmd("clear [resource]", "clear action", cobra.MatchAll(cobra.ExactArgs(1), cobra.OnlyValidArgs)),
	}
}
```

```
type SeqCmd struct {
	*MsgUtilsCmd
}

func NewSeqCmd() *SeqCmd {
	seqCmd := &SeqCmd{
		NewMsgUtilsCmd("seq", "seq", nil),
	}
	return seqCmd
}

func (s *SeqCmd) GetSeqCmd() *cobra.Command {
	s.Command.Run = func(cmdLines *cobra.Command, args []string) {

	}
	return &s.Command
}

func (s *SeqCmd) FixSeqCmd() *cobra.Command {
	return &s.Command
}

type MsgCmd struct {
	*MsgUtilsCmd
}

func NewMsgCmd() *MsgCmd {
	msgCmd := &MsgCmd{
		NewMsgUtilsCmd("msg", "msg", nil),
	}
	return msgCmd
}

func (m *MsgCmd) GetMsgCmd() *cobra.Command {
	return &m.Command
}

func (m *MsgCmd) ClearMsgCmd() *cobra.Command {
	return &m.Command
}
```

pkg/common/cmd/push.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/internal/push"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
    "github.com/openimsdk/open-im-server/v3/version"
    "github.com/openimsdk/tools/system/program"
    "github.com/spf13/cobra"
)

type PushRpcCmd struct {
    *RootCmd
    ctx        context.Context
    configMap  map[string]any
    pushConfig *push.Config
}

func NewPushRpcCmd() *PushRpcCmd {
    var pushConfig push.Config
    ret := &PushRpcCmd{pushConfig: &pushConfig}
    ret.configMap = map[string]any{
        config.OpenIMPushCfgFileName:    &pushConfig.RpcConfig,
        config.RedisConfigFileName:      &pushConfig.RedisConfig,
        config.MongodbConfigFileName:    &pushConfig.MongoConfig,
        config.KafkaConfigFileName:      &pushConfig.KafkaConfig,
        config.ShareFileName:            &pushConfig.Share,
        config.NotificationFileName:     &pushConfig.NotificationConfig,
        config.WebhooksConfigFileName:   &pushConfig.WebhooksConfig,
        config.LocalCacheConfigFileName: &pushConfig.LocalCacheConfig,
        config.DiscoveryConfigFilename:  &pushConfig.Discovery,
    }
    ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
    ret.ctx = context.WithValue(context.Background(), "version", version.Version)
    ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
        ret.pushConfig.FcmConfigPath = config.Path(ret.ConfigPath())
        return ret.runE()
    }
    return ret
}

func (a *PushRpcCmd) Exec() error {
    return a.Execute()
}

func (a *PushRpcCmd) runE() error {
    return startrpc.Start(a.ctx, &a.pushConfig.Discovery, &a.pushConfig.RpcConfig.CircuitBreaker, &a.pushConfig.RpcConf
        a.pushConfig.RpcConfig.RPC.RegisterIP, a.pushConfig.RpcConfig.RPC.AutoSetPorts, a.pushConfig.RpcConfig.RPC.Ports,
```

```
    a.Index(), a.pushConfig.Discovery.RpcService.Push, &a.pushConfig.NotificationConfig, a.pushConfig,
    []string{
        a.pushConfig.RpcConfig.GetConfigFileName(),
        a.pushConfig.RedisConfig.GetConfigFileName(),
        a.pushConfig.KafkaConfig.GetConfigFileName(),
        a.pushConfig.NotificationConfig.GetConfigFileName(),
        a.pushConfig.Share.GetConfigFileName(),
        a.pushConfig.WebhooksConfig.GetConfigFileName(),
        a.pushConfig.LocalCacheConfig.GetConfigFileName(),
        a.pushConfig.Discovery.GetConfigFileName(),
    },
    []string{
        a.pushConfig.Discovery.RpcService.MessageGateway,
    },
    push.Start)
}
```

pkg/common/cmd/root.go

```
package cmd

import (
	"context"
	"encoding/json"
	"fmt"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	kdisc "github.com/openimsdk/open-im-server/v3/pkg/common/discovery"
	disetcd "github.com/openimsdk/open-im-server/v3/pkg/common/discovery/etcd"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/discovery/etcd"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/spf13/cobra"
	clientv3 "go.etcd.io/etcd/client/v3"
)

type RootCmd struct {
	Command        cobra.Command
	processName    string
	port           int
	prometheusPort int
	log            config.Log
	index          int
	configPath     string
	etcdClient     *clientv3.Client
}

func (r *RootCmd) ConfigPath() string {
	return r.configPath
}

func (r *RootCmd) Index() int {
	return r.index
}

func (r *RootCmd) Port() int {
	return r.port
}

type CmdOpts struct {
	loggerPrefixName string
	configMap        map[string]any
}

func WithCronTaskLogName() func(*CmdOpts) {
	return func(opts *CmdOpts) {
		opts.loggerPrefixName = "openim-crontask"
	}
}

func WithLogName(logName string) func(*CmdOpts) {
	return func(opts *CmdOpts) {
		opts.loggerPrefixName = logName
	}
}
func WithConfigMap(configMap map[string]any) func(*CmdOpts) {
	return func(opts *CmdOpts) {
		opts.configMap = configMap
	}
}

func NewRootCmd(processName string, opts ...func(*CmdOpts)) *RootCmd {
```

```
	rootCmd := &RootCmd{processName: processName}
	cmd := cobra.Command{
		Use:  "Start openIM application",
		Long: fmt.Sprintf(`Start %s `, processName),
		PersistentPreRunE: func(cmd *cobra.Command, args []string) error {
			return rootCmd.persistentPreRun(cmd, opts...)
		},
		SilenceUsage:  true,
		SilenceErrors: false,
	}
	cmd.Flags().StringP(config.FlagConf, "c", "", "path of config directory")
	cmd.Flags().IntP(config.FlagTransferIndex, "i", 0, "process startup sequence number")

	rootCmd.Command = cmd
	return rootCmd
}

func (r *RootCmd) initEtcd() error {
	configDirectory, _, err := r.getFlag(&r.Command)
	if err != nil {
		return err
	}
	disConfig := config.Discovery{}
	err = config.Load(configDirectory, config.DiscoveryConfigFilename, config.EnvPrefixMap[config.DiscoveryConfigFilena
	if err != nil {
		return err
	}
	if disConfig.Enable == config.ETCD {
		discov, _ := kdisc.NewDiscoveryRegister(&disConfig, nil)
		r.etcdClient = discov.(*etcd.SvcDiscoveryRegistryImpl).GetClient()
	}
	return nil
}

func (r *RootCmd) persistentPreRun(cmd *cobra.Command, opts ...func(*CmdOpts)) error {
	if err := r.initEtcd(); err != nil {
		return err
	}
	cmdOpts := r.applyOptions(opts...)
	if err := r.initializeConfiguration(cmd, cmdOpts); err != nil {
		return err
	}
	if err := r.updateConfigFromEtcd(cmdOpts); err != nil {
		return err
	}
	if err := r.initializeLogger(cmdOpts); err != nil {
		return errs.WrapMsg(err, "failed to initialize logger")
	}
	if err := r.etcdClient.Close(); err != nil {
		return errs.WrapMsg(err, "failed to close etcd client")
	}
	return nil
}

func (r *RootCmd) initializeConfiguration(cmd *cobra.Command, opts *CmdOpts) error {
	configDirectory, _, err := r.getFlag(cmd)
	if err != nil {
		return err
	}

	// Load common configuration file
	//opts.configMap[ShareFileName] = StructEnvPrefix{EnvPrefix: shareEnvPrefix, ConfigStruct: &r.share}
	for configFileName, configStruct := range opts.configMap {
		err := config.Load(configDirectory, configFileName, config.EnvPrefixMap[configFileName], configStruct)
		if err != nil {
			return err
```

```
    }
  }
  // Load common log configuration file
  return config.Load(configDirectory, config.LogConfigFileName, config.EnvPrefixMap[config.LogConfigFileName], &r.log
}

func (r *RootCmd) updateConfigFromEtcd(opts *CmdOpts) error {
  if r.etcdClient == nil {
    return nil
  }
  ctx := context.TODO()

  res, err := r.etcdClient.Get(ctx, disetcd.BuildKey(disetcd.EnableConfigCenterKey))
  if err != nil {
    log.ZWarn(ctx, "root cmd updateConfigFromEtcd, etcd Get EnableConfigCenterKey err: %v", errs.Wrap(err))
    return nil
  }
  if res.Count == 0 {
    return nil
  } else {
    if string(res.Kvs[0].Value) == disetcd.Disable {
      return nil
    } else if string(res.Kvs[0].Value) != disetcd.Enable {
      return errs.New("unknown EnableConfigCenter value").Wrap()
    }
  }

  update := func(configFileName string, configStruct any) error {
    key := disetcd.BuildKey(configFileName)
    etcdRes, err := r.etcdClient.Get(ctx, key)
    if err != nil {
      log.ZWarn(ctx, "root cmd updateConfigFromEtcd, etcd Get err: %v", errs.Wrap(err))
      return nil
    }
    if etcdRes.Count == 0 {
      data, err := json.Marshal(configStruct)
      if err != nil {
        return errs.ErrArgs.WithDetail(err.Error()).Wrap()
      }
      _, err = r.etcdClient.Put(ctx, disetcd.BuildKey(configFileName), string(data))
      if err != nil {
        log.ZWarn(ctx, "root cmd updateConfigFromEtcd, etcd Put err: %v", errs.Wrap(err))
      }
      return nil
    }
    err = json.Unmarshal(etcdRes.Kvs[0].Value, configStruct)
    if err != nil {
      return errs.WrapMsg(err, "failed to unmarshal config from etcd")
    }
    return nil
  }
  for configFileName, configStruct := range opts.configMap {
    if err := update(configFileName, configStruct); err != nil {
      return err
    }
  }
  if err := update(config.LogConfigFileName, &r.log); err != nil {
    return err
  }
  // Load common log configuration file
  return nil

}

func (r *RootCmd) applyOptions(opts ...func(*CmdOpts)) *CmdOpts {
  cmdOpts := defaultCmdOpts()
```

```
	for _, opt := range opts {
		opt(cmdOpts)
	}

	return cmdOpts
}

func (r *RootCmd) initializeLogger(cmdOpts *CmdOpts) error {
	err := log.InitLoggerFromConfig(
		cmdOpts.loggerPrefixName,
		r.processName,
		"", "",
		r.log.RemainLogLevel,
		r.log.IsStdout,
		r.log.IsJson,
		r.log.StorageLocation,
		r.log.RemainRotationCount,
		r.log.RotationTime,
		version.Version,
		r.log.IsSimplify,
	)
	if err != nil {
		return errs.Wrap(err)
	}
	return errs.Wrap(log.InitConsoleLogger(r.processName, r.log.RemainLogLevel, r.log.IsJson, version.Version))

}

func defaultCmdOpts() *CmdOpts {
	return &CmdOpts{
		loggerPrefixName: "openim-service-log",
	}
}

func (r *RootCmd) getFlag(cmd *cobra.Command) (string, int, error) {
	configDirectory, err := cmd.Flags().GetString(config.FlagConf)
	if err != nil {
		return "", 0, errs.Wrap(err)
	}
	r.configPath = configDirectory
	index, err := cmd.Flags().GetInt(config.FlagTransferIndex)
	if err != nil {
		return "", 0, errs.Wrap(err)
	}
	r.index = index
	return configDirectory, index, nil
}

func (r *RootCmd) Execute() error {
	return r.Command.Execute()
}
```

pkg/common/cmd/third.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/rpc/third"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/system/program"
	"github.com/spf13/cobra"
)

type ThirdRpcCmd struct {
	*RootCmd
	ctx         context.Context
	configMap   map[string]any
	thirdConfig *third.Config
}

func NewThirdRpcCmd() *ThirdRpcCmd {
	var thirdConfig third.Config
	ret := &ThirdRpcCmd{thirdConfig: &thirdConfig}
	ret.configMap = map[string]any{
		config.OpenIMRPCThirdCfgFileName: &thirdConfig.RpcConfig,
		config.RedisConfigFileName:       &thirdConfig.RedisConfig,
		config.MongodbConfigFileName:     &thirdConfig.MongodbConfig,
		config.ShareFileName:             &thirdConfig.Share,
		config.NotificationFileName:      &thirdConfig.NotificationConfig,
		config.MinioConfigFileName:       &thirdConfig.MinioConfig,
		config.LocalCacheConfigFileName:  &thirdConfig.LocalCacheConfig,
		config.DiscoveryConfigFilename:   &thirdConfig.Discovery,
	}
	ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
	ret.ctx = context.WithValue(context.Background(), "version", version.Version)
	ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
		return ret.runE()
	}
	return ret
}

func (a *ThirdRpcCmd) Exec() error {
	return a.Execute()
}

func (a *ThirdRpcCmd) runE() error {
	return startrpc.Start(a.ctx, &a.thirdConfig.Discovery, &a.thirdConfig.RpcConfig.CircuitBreaker, &a.thirdConfig.RpcC
		a.thirdConfig.RpcConfig.RPC.RegisterIP, a.thirdConfig.RpcConfig.RPC.AutoSetPorts, a.thirdConfig.RpcConfig.RPC.Por
		a.Index(), a.thirdConfig.Discovery.RpcService.Third, &a.thirdConfig.NotificationConfig, a.thirdConfig,
		[]string{
```

```
        a.thirdConfig.RpcConfig.GetConfigFileName(),
        a.thirdConfig.RedisConfig.GetConfigFileName(),
        a.thirdConfig.MongodbConfig.GetConfigFileName(),
        a.thirdConfig.NotificationConfig.GetConfigFileName(),
        a.thirdConfig.Share.GetConfigFileName(),
        a.thirdConfig.MinioConfig.GetConfigFileName(),
        a.thirdConfig.LocalCacheConfig.GetConfigFileName(),
        a.thirdConfig.Discovery.GetConfigFileName(),
    }, nil,
    third.Start)
}
```

pkg/common/cmd/user.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package cmd

import (
"context"

"github.com/openimsdk/open-im-server/v3/internal/rpc/user"
"github.com/openimsdk/open-im-server/v3/pkg/common/config"
"github.com/openimsdk/open-im-server/v3/pkg/common/startrpc"
"github.com/openimsdk/open-im-server/v3/version"
"github.com/openimsdk/tools/system/program"
"github.com/spf13/cobra"
)

type UserRpcCmd struct {
*RootCmd
ctx        context.Context
configMap  map[string]any
userConfig *user.Config
}

func NewUserRpcCmd() *UserRpcCmd {
var userConfig user.Config
ret := &UserRpcCmd{userConfig: &userConfig}
ret.configMap = map[string]any{
config.OpenIMRPCUserCfgFileName: &userConfig.RpcConfig,
config.RedisConfigFileName:      &userConfig.RedisConfig,
config.MongodbConfigFileName:    &userConfig.MongodbConfig,
config.KafkaConfigFileName:      &userConfig.KafkaConfig,
config.ShareFileName:            &userConfig.Share,
config.NotificationFileName:     &userConfig.NotificationConfig,
config.WebhooksConfigFileName:   &userConfig.WebhooksConfig,
config.LocalCacheConfigFileName: &userConfig.LocalCacheConfig,
config.DiscoveryConfigFilename:  &userConfig.Discovery,
}
ret.RootCmd = NewRootCmd(program.GetProcessName(), WithConfigMap(ret.configMap))
ret.ctx = context.WithValue(context.Background(), "version", version.Version)
ret.Command.RunE = func(cmd *cobra.Command, args []string) error {
return ret.runE()
}
return ret
}

func (a *UserRpcCmd) Exec() error {
return a.Execute()
}

func (a *UserRpcCmd) runE() error {
return startrpc.Start(a.ctx, &a.userConfig.Discovery, &a.userConfig.RpcConfig.CircuitBreaker, &a.userConfig.RpcConf
a.userConfig.RpcConfig.RPC.RegisterIP, a.userConfig.RpcConfig.RPC.AutoSetPorts, a.userConfig.RpcConfig.RPC.Ports,
a.Index(), a.userConfig.Discovery.RpcService.User, &a.userConfig.NotificationConfig, a.userConfig,
```

```
  []string{
    a.userConfig.RpcConfig.GetConfigFileName(),
    a.userConfig.RedisConfig.GetConfigFileName(),
    a.userConfig.MongodbConfig.GetConfigFileName(),
    a.userConfig.KafkaConfig.GetConfigFileName(),
    a.userConfig.NotificationConfig.GetConfigFileName(),
    a.userConfig.Share.GetConfigFileName(),
    a.userConfig.WebhooksConfig.GetConfigFileName(),
    a.userConfig.LocalCacheConfig.GetConfigFileName(),
    a.userConfig.Discovery.GetConfigFileName(),
  }, nil,
  user.Start)
}
```

pkg/common/prommetrics

pkg/common/prommetrics/api.go

```
package prommetrics

import (
	"net"
	"strconv"

	"github.com/prometheus/client_golang/prometheus"
	"github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
	apiCounter = prometheus.NewCounterVec(
		prometheus.CounterOpts{
			Name: "api_count",
			Help: "Total number of API calls",
		},
		[]string{"path", "method", "code"},
	)
	httpCounter = prometheus.NewCounterVec(
		prometheus.CounterOpts{
			Name: "http_count",
			Help: "Total number of HTTP calls",
		},
		[]string{"path", "method", "status"},
	)
)

func RegistryApi() {
	registry.MustRegister(apiCounter, httpCounter)
}

func ApiInit(listener net.Listener) error {
	apiRegistry := prometheus.NewRegistry()
	cs := append(
		baseCollector,
		apiCounter,
		httpCounter,
	)
	return Init(apiRegistry, listener, commonPath, promhttp.HandlerFor(apiRegistry, promhttp.HandlerOpts{}), cs...)
}

func APICall(path string, method string, apiCode int) {
	apiCounter.With(prometheus.Labels{"path": path, "method": method, "code": strconv.Itoa(apiCode)}).Inc()
}

func HttpCall(path string, method string, status int) {
	httpCounter.With(prometheus.Labels{"path": path, "method": method, "status": strconv.Itoa(status)}).Inc()
}
```

pkg/common/prommetrics/grpc_auth.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import (
    "github.com/prometheus/client_golang/prometheus"
)

var (
    UserLoginCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "user_login_total",
        Help: "The number of user login",
    })
)

func RegistryAuth() {
    registry.MustRegister(UserLoginCounter)
}
```

pkg/common/prommetrics/grpc_msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import (
    "github.com/prometheus/client_golang/prometheus"
)

var (
    SingleChatMsgProcessSuccessCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "single_chat_msg_process_success_total",
        Help: "The number of single chat msg successful processed",
    })
    SingleChatMsgProcessFailedCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "single_chat_msg_process_failed_total",
        Help: "The number of single chat msg failed processed",
    })
    GroupChatMsgProcessSuccessCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "group_chat_msg_process_success_total",
        Help: "The number of group chat msg successful processed",
    })
    GroupChatMsgProcessFailedCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "group_chat_msg_process_failed_total",
        Help: "The number of group chat msg failed processed",
    })
)

func RegistryMsg() {
    registry.MustRegister(
        SingleChatMsgProcessSuccessCounter,
        SingleChatMsgProcessFailedCounter,
        GroupChatMsgProcessSuccessCounter,
        GroupChatMsgProcessFailedCounter,
    )
}
```

pkg/common/prommetrics/grpc_msggateway.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import (
    "github.com/prometheus/client_golang/prometheus"
)

var (
    OnlineUserGauge = prometheus.NewGauge(prometheus.GaugeOpts{
        Name: "online_user_num",
        Help: "The number of online user num",
    })
)

func RegistryMsgGateway() {
    registry.MustRegister(OnlineUserGauge)
}
```

pkg/common/prommetrics/grpc_push.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import (
    "github.com/prometheus/client_golang/prometheus"
)

var (
    MsgOfflinePushFailedCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "msg_offline_push_failed_total",
        Help: "The number of msg failed offline pushed",
    })
    MsgLoneTimePushCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "msg_long_time_push_total",
        Help: "The number of messages with a push time exceeding 10 seconds",
    })
)

func RegistryPush() {
    registry.MustRegister(
        MsgOfflinePushFailedCounter,
        MsgLoneTimePushCounter,
    )
}
```

pkg/common/prommetrics/grpc_user.go

```
package prommetrics

import "github.com/prometheus/client_golang/prometheus"

var (
    UserRegisterCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "user_register_total",
        Help: "The number of user login",
    })
)

func RegistryUser() {
    registry.MustRegister(UserRegisterCounter)
}
```

pkg/common/prommetrics/prommetrics.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import (
    "errors"
    "fmt"
    "net"
    "net/http"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/collectors"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

const commonPath = "/metrics"

var registry = &prometheusRegistry{prometheus.NewRegistry()}

type prometheusRegistry struct {
    *prometheus.Registry
}

func (x *prometheusRegistry) MustRegister(cs ...prometheus.Collector) {
    for _, c := range cs {
        if err := x.Registry.Register(c); err != nil {
            if errors.As(err, &prometheus.AlreadyRegisteredError{}) {
                continue
            }
            panic(err)
        }
    }
}

func init() {
    registry.MustRegister(
        collectors.NewProcessCollector(collectors.ProcessCollectorOpts{}),
        collectors.NewGoCollector(),
    )
}

var (
    baseCollector = []prometheus.Collector{
        collectors.NewProcessCollector(collectors.ProcessCollectorOpts{}),
        collectors.NewGoCollector(),
    }
)

func Init(registry *prometheus.Registry, listener net.Listener, path string, handler http.Handler, cs ...prometheus.
    registry.MustRegister(cs...)
    srv := http.NewServeMux()
    srv.Handle(path, handler)
```

```
    return http.Serve(listener, srv)
}

func RegistryAll() {
    RegistryApi()
    RegistryAuth()
    RegistryMsg()
    RegistryMsgGateway()
    RegistryPush()
    RegistryUser()
    RegistryRpc()
    RegistryTransfer()
}

func Start(listener net.Listener) error {
    srv := http.NewServeMux()
    srv.Handle(commonPath, promhttp.HandlerFor(registry, promhttp.HandlerOpts{}))
    return http.Serve(listener, srv)
}

const (
    APIKeyName             = "api"
    MessageTransferKeyName = "message-transfer"

    TTL = 300
)

type Target struct {
    Target string            `json:"target"`
    Labels map[string]string `json:"labels"`
}

type RespTarget struct {
    Targets []string          `json:"targets"`
    Labels  map[string]string `json:"labels"`
}

func BuildDiscoveryKeyPrefix(name string) string {
    return fmt.Sprintf("%s/%s/%s", "openim", "prometheus_discovery", name)
}

func BuildDiscoveryKey(name string, index int) string {
    return fmt.Sprintf("%s/%s/%s/%d", "openim", "prometheus_discovery", name, index)
}

func BuildDefaultTarget(host string, ip int) Target {
    return Target{
        Target: fmt.Sprintf("%s:%d", host, ip),
        Labels: map[string]string{
            "namespace": "default",
        },
    }
}
```

pkg/common/prommetrics/prommetrics_test.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import "testing"

//func TestNewGrpcPromObj(t *testing.T) {
//■// Create a custom metric to pass into the NewGrpcPromObj function.
//■customMetric := prometheus.NewCounter(prometheus.CounterOpts{
//■■Name: "test_metric",
//■■Help: "This is a test metric.",
//■})
//■cusMetrics := []prometheus.Collector{customMetric}
//
//■// Call NewGrpcPromObj with the custom metrics.
//■reg, grpcMetrics, err := NewGrpcPromObj(cusMetrics)
//
//■// Assert no error was returned.
//■assert.NoError(t, err)
//
//■// Assert the registry was correctly initialized.
//■assert.NotNil(t, reg)
//
//■// Assert the grpcMetrics was correctly initialized.
//■assert.NotNil(t, grpcMetrics)
//
//■// Assert that the custom metric is registered.
//■mfs, err := reg.Gather()
//■assert.NoError(t, err)
//■assert.NotEmpty(t, mfs) // Ensure some metrics are present.
//■found := false
//■for _, mf := range mfs {
//■■if *mf.Name == "test_metric" {
//■■■found = true
//■■■break
//■■}
//■}
//■assert.True(t, found, "Custom metric not found in registry")
//}

//func TestGetGrpcCusMetrics(t *testing.T) {
//■conf := config2.NewGlobalConfig()
//
//■config2.InitConfig(conf, "../../config")
//■// Test various cases based on the switch statement in the GetGrpcCusMetrics function.
//■testCases := []struct {
//■■name     string
//■■expected int // The expected number of metrics for each case.
//■}{
//■■{conf.RpcRegisterName.OpenImMessageGatewayName, 1},
//■}
//
```

```
//■for _, tc := range testCases {
//■■t.Run(tc.name, func(t *testing.T) {
//■■■metrics := GetGrpcCusMetrics(tc.name, &conf.RpcRegisterName)
//■■■assert.Len(t, metrics, tc.expected)
//■■})
//■}
//}

func TestName(t *testing.T) {
■RegistryApi()
■RegistryApi()

}
```

pkg/common/prommetrics/rpc.go

```
package prommetrics

import (
	"net"
	"strconv"

	gp "github.com/grpc-ecosystem/go-grpc-prometheus"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/prometheus/client_golang/prometheus"
	"github.com/prometheus/client_golang/prometheus/promhttp"
)

const rpcPath = commonPath

var (
	grpcMetrics *gp.ServerMetrics
	rpcCounter  = prometheus.NewCounterVec(
		prometheus.CounterOpts{
			Name: "rpc_count",
			Help: "Total number of RPC calls",
		},
		[]string{"name", "path", "code"},
	)
)

func RegistryRpc() {
	registry.MustRegister(rpcCounter)
}

func RpcInit(cs []prometheus.Collector, listener net.Listener) error {
	reg := prometheus.NewRegistry()
	cs = append(append(
		baseCollector,
		rpcCounter,
	), cs...)
	return Init(reg, listener, rpcPath, promhttp.HandlerFor(reg, promhttp.HandlerOpts{Registry: reg}), cs...)
}

func RPCCall(name string, path string, code int) {
	rpcCounter.With(prometheus.Labels{"name": name, "path": path, "code": strconv.Itoa(code)}).Inc()
}

func GetGrpcServerMetrics() *gp.ServerMetrics {
	if grpcMetrics == nil {
		grpcMetrics = gp.NewServerMetrics()
		grpcMetrics.EnableHandlingTimeHistogram()
	}
	return grpcMetrics
}

func GetGrpcCusMetrics(registerName string, discovery *config.Discovery) []prometheus.Collector {
	switch registerName {
	case discovery.RpcService.MessageGateway:
		return []prometheus.Collector{OnlineUserGauge}
	case discovery.RpcService.Msg:
		return []prometheus.Collector{
			SingleChatMsgProcessSuccessCounter,
			SingleChatMsgProcessFailedCounter,
			GroupChatMsgProcessSuccessCounter,
			GroupChatMsgProcessFailedCounter,
		}
	case discovery.RpcService.Push:
		return []prometheus.Collector{
			MsgOfflinePushFailedCounter,
```

```
        MsgLoneTimePushCounter,
    }
    case discovery.RpcService.Auth:
        return []prometheus.Collector{UserLoginCounter}
    case discovery.RpcService.User:
        return []prometheus.Collector{UserRegisterCounter}
    default:
        return nil
    }
}
```

pkg/common/prommetrics/transfer.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package prommetrics

import (
    "net"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
    MsgInsertRedisSuccessCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "msg_insert_redis_success_total",
        Help: "The number of successful insert msg to redis",
    })
    MsgInsertRedisFailedCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "msg_insert_redis_failed_total",
        Help: "The number of failed insert msg to redis",
    })
    MsgInsertMongoSuccessCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "msg_insert_mongo_success_total",
        Help: "The number of successful insert msg to mongo",
    })
    MsgInsertMongoFailedCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "msg_insert_mongo_failed_total",
        Help: "The number of failed insert msg to mongo",
    })
    SeqSetFailedCounter = prometheus.NewCounter(prometheus.CounterOpts{
        Name: "seq_set_failed_total",
        Help: "The number of failed set seq",
    })
)

func RegistryTransfer() {
    registry.MustRegister(
        MsgInsertRedisSuccessCounter,
        MsgInsertRedisFailedCounter,
        MsgInsertMongoSuccessCounter,
        MsgInsertMongoFailedCounter,
        SeqSetFailedCounter,
    )
}

func TransferInit(listener net.Listener) error {
    reg := prometheus.NewRegistry()
    cs := append(
        baseCollector,
        MsgInsertRedisSuccessCounter,
        MsgInsertRedisFailedCounter,
        MsgInsertMongoSuccessCounter,
        MsgInsertMongoFailedCounter,
```

```
    SeqSetFailedCounter,
  )
  return Init(reg, listener, commonPath, promhttp.HandlerFor(reg, promhttp.HandlerOpts{Registry: reg}), cs...)
}
```

pkg/common/ginprometheus

pkg/common/ginprometheus/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package ginprometheus // import "github.com/openimsdk/open-im-server/v3/pkg/common/ginprometheus"
```

pkg/common/ginprometheus/ginprometheus.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package ginprometheus

//
//import (
//■"bytes"
//■"fmt"
//■"io"
//■"net/http"
//■"os"
//■"strconv"
//■"time"
//
//■"github.com/gin-gonic/gin"
//■"github.com/prometheus/client_golang/prometheus"
//■"github.com/prometheus/client_golang/prometheus/promhttp"
//)
//
//var defaultMetricPath = "/metrics"
//
//// counter, counter_vec, gauge, gauge_vec,
//// histogram, histogram_vec, summary, summary_vec.
//var (
//■reqCounter = &Metric{
//■■ID:          "reqCnt",
//■■Name:        "requests_total",
//■■Description: "How many HTTP requests processed, partitioned by status code and HTTP method.",
//■■Type:        "counter_vec",
//■■Args:        []string{"code", "method", "handler", "host", "url"}}
//
//■reqDuration = &Metric{
//■■ID:          "reqDur",
//■■Name:        "request_duration_seconds",
//■■Description: "The HTTP request latencies in seconds.",
//■■Type:        "histogram_vec",
//■■Args:        []string{"code", "method", "url"},
//■}
//
//■resSize = &Metric{
//■■ID:          "resSz",
//■■Name:        "response_size_bytes",
//■■Description: "The HTTP response sizes in bytes.",
//■■Type:        "summary"}
//
//■reqSize = &Metric{
//■■ID:          "reqSz",
//■■Name:        "request_size_bytes",
//■■Description: "The HTTP request sizes in bytes.",
//■■Type:        "summary"}
//
//■standardMetrics = []*Metric{
```

```
//■■reqCounter,
//■■reqDuration,
//■■resSize,
//■■reqSize,
//■}
//)
//
///*
//RequestCounterURLLabelMappingFn is a function which can be supplied to the middleware to control
//the cardinality of the request counter's "url" label, which might be required in some contexts.
//For instance, if for a "/customer/:name" route you don't want to generate a time series for every
//possible customer name, you could use this function:
//
//■func(c *gin.Context) string {
//■■url := c.Request.URL.Path
//■■for _, p := range c.Params {
//■■■if p.Key == "name" {
//■■■■url = strings.Replace(url, p.Value, ":name", 1)
//■■■■break
//■■■}
//■■}
//■■return url
//■}
//
//which would map "/customer/alice" and "/customer/bob" to their template "/customer/:name".
//*/
//type RequestCounterURLLabelMappingFn func(c *gin.Context) string
//
//// Metric is a definition for the name, description, type, ID, and
//// prometheus.Collector type (i.e. CounterVec, Summary, etc) of each metric.
//type Metric struct {
//■MetricCollector prometheus.Collector
//■ID              string
//■Name            string
//■Description     string
//■Type            string
//■Args            []string
//}
//
//// Prometheus contains the metrics gathered by the instance and its path.
//type Prometheus struct {
//■reqCnt        *prometheus.CounterVec
//■reqDur        *prometheus.HistogramVec
//■reqSz, resSz  prometheus.Summary
//■router        *gin.Engine
//■listenAddress string
//■Ppg           PrometheusPushGateway
//
//■MetricsList []*Metric
//■MetricsPath string
//
//■ReqCntURLLabelMappingFn RequestCounterURLLabelMappingFn
//
//■// gin.Context string to use as a prometheus URL label
//■URLLabelFromContext string
//}
//
//// PrometheusPushGateway contains the configuration for pushing to a Prometheus pushgateway (optional).
//type PrometheusPushGateway struct {
//
//■// Push interval in seconds
//■PushIntervalSeconds time.Duration
//
//■// Push Gateway URL in format http://domain:port
//■// where JOBNAME can be any string of your choice
//■PushGatewayURL string
```

```
//
//■// Local metrics URL where metrics are fetched from, this could be omitted in the future
//■// if implemented using prometheus common/expfmt instead
//■MetricsURL string
//
//■// pushgateway job name, defaults to "gin"
//■Job string
//}
//
//// NewPrometheus generates a new set of metrics with a certain subsystem name.
//func NewPrometheus(subsystem string, customMetricsList ...[]*Metric) *Prometheus {
//■if subsystem == "" {
//■■subsystem = "app"
//■}
//
//■var metricsList []*Metric
//
//■if len(customMetricsList) > 1 {
//■■panic("Too many args. NewPrometheus( string, <optional []*Metric> ).")
//■} else if len(customMetricsList) == 1 {
//■■metricsList = customMetricsList[0]
//■}
//■metricsList = append(metricsList, standardMetrics...)
//
//■p := &Prometheus{
//■■MetricsList: metricsList,
//■■MetricsPath: defaultMetricPath,
//■■ReqCntURLLabelMappingFn: func(c *gin.Context) string {
//■■■return c.FullPath() // e.g. /user/:id , /user/:id/info
//■■},
//■}
//
//■p.registerMetrics(subsystem)
//
//■return p
//}
//
//// SetPushGateway sends metrics to a remote pushgateway exposed on pushGatewayURL
//// every pushIntervalSeconds. Metrics are fetched from metricsURL.
//func (p *Prometheus) SetPushGateway(pushGatewayURL, metricsURL string, pushIntervalSeconds time.Duration) {
//■p.Ppg.PushGatewayURL = pushGatewayURL
//■p.Ppg.MetricsURL = metricsURL
//■p.Ppg.PushIntervalSeconds = pushIntervalSeconds
//■p.startPushTicker()
//}
//
//// SetPushGatewayJob job name, defaults to "gin".
//func (p *Prometheus) SetPushGatewayJob(j string) {
//■p.Ppg.Job = j
//}
//
//// SetListenAddress for exposing metrics on address. If not set, it will be exposed at the
//// same address of the gin engine that is being used.
//func (p *Prometheus) SetListenAddress(address string) {
//■p.listenAddress = address
//■if p.listenAddress != "" {
//■■p.router = gin.Default()
//■}
//}
//
//// SetListenAddressWithRouter for using a separate router to expose metrics. (this keeps things like GET /metrics
//// your content's access log).
//func (p *Prometheus) SetListenAddressWithRouter(listenAddress string, r *gin.Engine) {
//■p.listenAddress = listenAddress
//■if len(p.listenAddress) > 0 {
//■■p.router = r
```

```
//■}
//}
//
//// SetMetricsPath set metrics paths.
//func (p *Prometheus) SetMetricsPath(e *gin.Engine) error {
//
//■if p.listenAddress != "" {
//■■p.router.GET(p.MetricsPath, prometheusHandler())
//■■return p.runServer()
//■} else {
//■■e.GET(p.MetricsPath, prometheusHandler())
//■■return nil
//■}
//}
//
//// SetMetricsPathWithAuth set metrics paths with authentication.
//func (p *Prometheus) SetMetricsPathWithAuth(e *gin.Engine, accounts gin.Accounts) error {
//
//■if p.listenAddress != "" {
//■■p.router.GET(p.MetricsPath, gin.BasicAuth(accounts), prometheusHandler())
//■■return p.runServer()
//■} else {
//■■e.GET(p.MetricsPath, gin.BasicAuth(accounts), prometheusHandler())
//■■return nil
//■}
//
//}
//
//func (p *Prometheus) runServer() error {
//■return p.router.Run(p.listenAddress)
//}
//
//func (p *Prometheus) getMetrics() []byte {
//■response, err := http.Get(p.Ppg.MetricsURL)
//■if err != nil {
//■■return nil
//■}
//
//■defer response.Body.Close()
//
//■body, _ := io.ReadAll(response.Body)
//■return body
//}
//
//var hostname, _ = os.Hostname()
//
//func (p *Prometheus) getPushGatewayURL() string {
//■if p.Ppg.Job == "" {
//■■p.Ppg.Job = "gin"
//■}
//■return p.Ppg.PushGatewayURL + "/metrics/job/" + p.Ppg.Job + "/instance/" + hostname
//}
//
//func (p *Prometheus) sendMetricsToPushGateway(metrics []byte) {
//■req, err := http.NewRequest("POST", p.getPushGatewayURL(), bytes.NewBuffer(metrics))
//■if err != nil {
//■■return
//■}
//
//■client := &http.Client{}
//■resp, err := client.Do(req)
//■if err != nil {
//■■fmt.Println("Error sending to push gateway error:", err.Error())
//■}
//
//■resp.Body.Close()
```

```
//}
//
//func (p *Prometheus) startPushTicker() {
//■ticker := time.NewTicker(time.Second * p.Ppg.PushIntervalSeconds)
//■go func() {
//■■for range ticker.C {
//■■■■p.sendMetricsToPushGateway(p.getMetrics())
//■■■}
//■■}()
//}
//
//// NewMetric associates prometheus.Collector based on Metric.Type.
//func NewMetric(m *Metric, subsystem string) prometheus.Collector {
//■var metric prometheus.Collector
//■switch m.Type {
//■case "counter_vec":
//■■metric = prometheus.NewCounterVec(
//■■■■prometheus.CounterOpts{
//■■■■■Subsystem: subsystem,
//■■■■■Name:      m.Name,
//■■■■■Help:      m.Description,
//■■■■},
//■■■■m.Args,
//■■■)
//■case "counter":
//■■metric = prometheus.NewCounter(
//■■■■prometheus.CounterOpts{
//■■■■■Subsystem: subsystem,
//■■■■■Name:      m.Name,
//■■■■■Help:      m.Description,
//■■■■},
//■■■)
//■case "gauge_vec":
//■■metric = prometheus.NewGaugeVec(
//■■■■prometheus.GaugeOpts{
//■■■■■Subsystem: subsystem,
//■■■■■Name:      m.Name,
//■■■■■Help:      m.Description,
//■■■■},
//■■■■m.Args,
//■■■)
//■case "gauge":
//■■metric = prometheus.NewGauge(
//■■■■prometheus.GaugeOpts{
//■■■■■Subsystem: subsystem,
//■■■■■Name:      m.Name,
//■■■■■Help:      m.Description,
//■■■■},
//■■■)
//■case "histogram_vec":
//■■metric = prometheus.NewHistogramVec(
//■■■■prometheus.HistogramOpts{
//■■■■■Subsystem: subsystem,
//■■■■■Name:      m.Name,
//■■■■■Help:      m.Description,
//■■■■},
//■■■■m.Args,
//■■■)
//■case "histogram":
//■■metric = prometheus.NewHistogram(
//■■■■prometheus.HistogramOpts{
//■■■■■Subsystem: subsystem,
//■■■■■Name:      m.Name,
//■■■■■Help:      m.Description,
//■■■■},
//■■■)
```

```
//■case "summary_vec":
//■■metric = prometheus.NewSummaryVec(
//■■■prometheus.SummaryOpts{
//■■■■Subsystem: subsystem,
//■■■■Name:      m.Name,
//■■■■Help:      m.Description,
//■■■},
//■■■m.Args,
//■■)
//■case "summary":
//■■metric = prometheus.NewSummary(
//■■■prometheus.SummaryOpts{
//■■■■Subsystem: subsystem,
//■■■■Name:      m.Name,
//■■■■Help:      m.Description,
//■■■},
//■■)
//■}
//■return metric
//}
//
//func (p *Prometheus) registerMetrics(subsystem string) {
//■for _, metricDef := range p.MetricsList {
//■■metric := NewMetric(metricDef, subsystem)
//■■if err := prometheus.Register(metric); err != nil {
//■■■fmt.Println("could not be registered in Prometheus,metricDef.Name:", metricDef.Name, "   error:", err.Error())
//■■}
//
//■■switch metricDef {
//■■case reqCounter:
//■■■p.reqCnt = metric.(*prometheus.CounterVec)
//■■case reqDuration:
//■■■p.reqDur = metric.(*prometheus.HistogramVec)
//■■case resSize:
//■■■p.resSz = metric.(prometheus.Summary)
//■■case reqSize:
//■■■p.reqSz = metric.(prometheus.Summary)
//■■}
//■■metricDef.MetricCollector = metric
//■}
//}
//
//// Use adds the middleware to a gin engine.
//func (p *Prometheus) Use(e *gin.Engine) error {
//■e.Use(p.HandlerFunc())
//■return p.SetMetricsPath(e)
//}
//
//// UseWithAuth adds the middleware to a gin engine with BasicAuth.
//func (p *Prometheus) UseWithAuth(e *gin.Engine, accounts gin.Accounts) error {
//■e.Use(p.HandlerFunc())
//■return p.SetMetricsPathWithAuth(e, accounts)
//}
//
//// HandlerFunc defines handler function for middleware.
//func (p *Prometheus) HandlerFunc() gin.HandlerFunc {
//■return func(c *gin.Context) {
//■■if c.Request.URL.Path == p.MetricsPath {
//■■■c.Next()
//■■■return
//■■}
//
//■■start := time.Now()
//■■reqSz := computeApproximateRequestSize(c.Request)
//
//■■c.Next()
```

```
//
//■■status := strconv.Itoa(c.Writer.Status())
//■■elapsed := float64(time.Since(start)) / float64(time.Second)
//■■resSz := float64(c.Writer.Size())
//
//■■url := p.ReqCntURLLabelMappingFn(c)
//■■if len(p.URLLabelFromContext) > 0 {
//■■■u, found := c.Get(p.URLLabelFromContext)
//■■■if !found {
//■■■■u = "unknown"
//■■■}
//■■■url = u.(string)
//■■}
//■■p.reqDur.WithLabelValues(status, c.Request.Method, url).Observe(elapsed)
//■■p.reqCnt.WithLabelValues(status, c.Request.Method, c.HandlerName(), c.Request.Host, url).Inc()
//■■p.reqSz.Observe(float64(reqSz))
//■■p.resSz.Observe(resSz)
//■}
//}
//
//func prometheusHandler() gin.HandlerFunc {
//■h := promhttp.Handler()
//■return func(c *gin.Context) {
//■■h.ServeHTTP(c.Writer, c.Request)
//■}
//}
//
//func computeApproximateRequestSize(r *http.Request) int {
//■var s int
//■if r.URL != nil {
//■■s = len(r.URL.Path)
//■}
//
//■s += len(r.Method)
//■s += len(r.Proto)
//■for name, values := range r.Header {
//■■s += len(name)
//■■for _, value := range values {
//■■■s += len(value)
//■■}
//■}
//■s += len(r.Host)
//
//■// r.FormData and r.MultipartForm are assumed to be included in r.URL.
//
//■if r.ContentLength != -1 {
//■■s += int(r.ContentLength)
//■}
//■return s
//}
```

pkg/common/servererrs

pkg/common/servererrs/code.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package servererrs

// UnknownCode represents the error code when code is not parsed or parsed code equals 0.
const UnknownCode = 1000

// Error codes for various error scenarios.
const (
	FormattingError      = 10001 // Error in formatting
	HasRegistered        = 10002 // user has already registered
	NotRegistered        = 10003 // user is not registered
	PasswordErr          = 10004 // Password error
	GetIMTokenErr        = 10005 // Error in getting IM token
	RepeatSendCode       = 10006 // Repeat sending code
	MailSendCodeErr      = 10007 // Error in sending code via email
	SmsSendCodeErr       = 10008 // Error in sending code via SMS
	CodeInvalidOrExpired = 10009 // Code is invalid or expired
	RegisterFailed       = 10010 // Registration failed
	ResetPasswordFailed  = 10011 // Resetting password failed
	RegisterLimit        = 10012 // Registration limit exceeded
	LoginLimit           = 10013 // Login limit exceeded
	InvitationError      = 10014 // Error in invitation
)

// General error codes.
const (
	NoError = 0 // No error

	DatabaseError = 90002 // Database error (redis/mysql, etc.)
	NetworkError  = 90004 // Network error
	DataError     = 90007 // Data error

	CallbackError = 80000

	// General error codes.
	ServerInternalError   = 500  // Server internal error
	ArgsError             = 1001 // Input parameter error
	NoPermissionError     = 1002 // Insufficient permission
	DuplicateKeyError     = 1003
	RecordNotFoundError   = 1004 // Record does not exist
	SecretNotChangedError = 1050 // secret not changed

	// Account error codes.
	UserIDNotFoundError    = 1101 // UserID does not exist or is not registered
	RegisteredAlreadyError = 1102 // user is already registered

	// Group error codes.
	GroupIDNotFoundError  = 1201 // GroupID does not exist
```

```
	GroupIDExisted        = 1202 // GroupID already exists
	NotInGroupYetError    = 1203 // Not in the group yet
	DismissedAlreadyError = 1204 // Group has already been dismissed
	GroupTypeNotSupport   = 1205
	GroupRequestHandled   = 1206

	// Relationship error codes.
	CanNotAddYourselfError   = 1301 // Cannot add yourself as a friend
	BlockedByPeer            = 1302 // Blocked by the peer
	NotPeersFriend           = 1303 // Not the peer's friend
	RelationshipAlreadyError = 1304 // Already in a friend relationship

	// Message error codes.
	MessageHasReadDisable = 1401
	MutedInGroup          = 1402 // Member muted in the group
	MutedGroup            = 1403 // Group is muted
	MsgAlreadyRevoke      = 1404 // Message already revoked

	// Token error codes.
	TokenExpiredError     = 1501
	TokenInvalidError     = 1502
	TokenMalformedError   = 1503
	TokenNotValidYetError = 1504
	TokenUnknownError     = 1505
	TokenKickedError      = 1506
	TokenNotExistError    = 1507

	// Long connection gateway error codes.
	ConnOverMaxNumLimit  = 1601
	ConnArgsErr          = 1602
	PushMsgErr           = 1603
	IOSBackgroundPushErr = 1604

	// S3 error codes.
	FileUploadedExpiredError = 1701 // Upload expired
)
```

pkg/common/servererrs/doc.go

```
package servererrs // import "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
```

pkg/common/servererrs/predefine.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package servererrs

import "github.com/openimsdk/tools/errs"

var (
	ErrSecretNotChanged = errs.NewCodeError(SecretNotChangedError, "secret not changed, please change secret in config/

	ErrDatabase         = errs.NewCodeError(DatabaseError, "DatabaseError")
	ErrNetwork          = errs.NewCodeError(NetworkError, "NetworkError")
	ErrCallback         = errs.NewCodeError(CallbackError, "CallbackError")
	ErrCallbackContinue = errs.NewCodeError(CallbackError, "ErrCallbackContinue")

	ErrInternalServer = errs.NewCodeError(ServerInternalError, "ServerInternalError")
	ErrArgs           = errs.NewCodeError(ArgsError, "ArgsError")
	ErrNoPermission   = errs.NewCodeError(NoPermissionError, "NoPermissionError")
	ErrDuplicateKey   = errs.NewCodeError(DuplicateKeyError, "DuplicateKeyError")
	ErrRecordNotFound = errs.NewCodeError(RecordNotFoundError, "RecordNotFoundError")

	ErrUserIDNotFound  = errs.NewCodeError(UserIDNotFoundError, "UserIDNotFoundError")
	ErrGroupIDNotFound = errs.NewCodeError(GroupIDNotFoundError, "GroupIDNotFoundError")
	ErrGroupIDExisted  = errs.NewCodeError(GroupIDExisted, "GroupIDExisted")

	ErrNotInGroupYet       = errs.NewCodeError(NotInGroupYetError, "NotInGroupYetError")
	ErrDismissedAlready    = errs.NewCodeError(DismissedAlreadyError, "DismissedAlreadyError")
	ErrRegisteredAlready   = errs.NewCodeError(RegisteredAlreadyError, "RegisteredAlreadyError")
	ErrGroupTypeNotSupport = errs.NewCodeError(GroupTypeNotSupport, "")
	ErrGroupRequestHandled = errs.NewCodeError(GroupRequestHandled, "GroupRequestHandled")

	ErrData             = errs.NewCodeError(DataError, "DataError")
	ErrTokenExpired     = errs.NewCodeError(TokenExpiredError, "TokenExpiredError")
	ErrTokenInvalid     = errs.NewCodeError(TokenInvalidError, "TokenInvalidError")         //
	ErrTokenMalformed   = errs.NewCodeError(TokenMalformedError, "TokenMalformedError")     //
	ErrTokenNotValidYet = errs.NewCodeError(TokenNotValidYetError, "TokenNotValidYetError") //
	ErrTokenUnknown     = errs.NewCodeError(TokenUnknownError, "TokenUnknownError")         //
	ErrTokenKicked      = errs.NewCodeError(TokenKickedError, "TokenKickedError")
	ErrTokenNotExist    = errs.NewCodeError(TokenNotExistError, "TokenNotExistError") //

	ErrMessageHasReadDisable = errs.NewCodeError(MessageHasReadDisable, "MessageHasReadDisable")

	ErrCanNotAddYourself   = errs.NewCodeError(CanNotAddYourselfError, "CanNotAddYourselfError")
	ErrBlockedByPeer       = errs.NewCodeError(BlockedByPeer, "BlockedByPeer")
	ErrNotPeersFriend      = errs.NewCodeError(NotPeersFriend, "NotPeersFriend")
	ErrRelationshipAlready = errs.NewCodeError(RelationshipAlreadyError, "RelationshipAlreadyError")

	ErrMutedInGroup     = errs.NewCodeError(MutedInGroup, "MutedInGroup")
	ErrMutedGroup       = errs.NewCodeError(MutedGroup, "MutedGroup")
	ErrMsgAlreadyRevoke = errs.NewCodeError(MsgAlreadyRevoke, "MsgAlreadyRevoke")

	ErrConnOverMaxNumLimit = errs.NewCodeError(ConnOverMaxNumLimit, "ConnOverMaxNumLimit")
```

```
	ErrConnArgsErr          = errs.NewCodeError(ConnArgsErr, "args err, need token, sendID, platformID")
	ErrPushMsgErr           = errs.NewCodeError(PushMsgErr, "push msg err")
	ErrIOSBackgroundPushErr = errs.NewCodeError(IOSBackgroundPushErr, "ios background push err")

	ErrFileUploadedExpired = errs.NewCodeError(FileUploadedExpiredError, "FileUploadedExpiredError")
)
```

pkg/common/servererrs/relation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package servererrs

import "github.com/openimsdk/tools/errs"

var Relation = &relation{m: make(map[int]map[int]struct{})}

func init() {
	Relation.Add(errs.RecordNotFoundError, UserIDNotFoundError)
	Relation.Add(errs.RecordNotFoundError, GroupIDNotFoundError)
	Relation.Add(errs.DuplicateKeyError, GroupIDExisted)
}

type relation struct {
	m map[int]map[int]struct{}
}

func (r *relation) Add(codes ...int) {
	if len(codes) < 2 {
		panic("codes length must be greater than 2")
	}
	for i := 1; i < len(codes); i++ {
		parent := codes[i-1]
		s, ok := r.m[parent]
		if !ok {
			s = make(map[int]struct{})
			r.m[parent] = s
		}
		for _, code := range codes[i:] {
			s[code] = struct{}{}
		}
	}
}

func (r *relation) Is(parent, child int) bool {
	if parent == child {
		return true
	}
	s, ok := r.m[parent]
	if !ok {
		return false
	}
	_, ok = s[child]
	return ok
}
```

pkg/common/startrpc

pkg/common/startrpc/circuitbreaker.go

```
package startrpc

import (
	"context"
	"time"

	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/stability/circuitbreaker"
	"github.com/openimsdk/tools/stability/circuitbreaker/sre"
	"google.golang.org/grpc"
	"google.golang.org/grpc/codes"
	"google.golang.org/grpc/status"
)

type CircuitBreaker struct {
	Enable  bool          `yaml:"enable"`
	Success float64       `yaml:"success"` // success rate threshold (0.0-1.0)
	Request int64         `yaml:"request"` // request threshold
	Bucket  int           `yaml:"bucket"`  // number of buckets
	Window  time.Duration `yaml:"window"`  // time window for statistics
}

func NewCircuitBreaker(config *CircuitBreaker) circuitbreaker.CircuitBreaker {
	if !config.Enable {
		return nil
	}

	return sre.NewSREBraker(
		sre.WithWindow(config.Window),
		sre.WithBucket(config.Bucket),
		sre.WithSuccess(config.Success),
		sre.WithRequest(config.Request),
	)
}

func UnaryCircuitBreakerInterceptor(breaker circuitbreaker.CircuitBreaker) grpc.ServerOption {
	if breaker == nil {
		return grpc.ChainUnaryInterceptor(func(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.Una
			return handler(ctx, req)
		})
	}

	return grpc.ChainUnaryInterceptor(func(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.Unary
		if err := breaker.Allow(); err != nil {
			log.ZWarn(ctx, "rpc circuit breaker open", err, "method", info.FullMethod)
			return nil, status.Error(codes.Unavailable, "service unavailable due to circuit breaker")
		}

		resp, err = handler(ctx, req)

		if err != nil {
			if st, ok := status.FromError(err); ok {
				switch st.Code() {
				case codes.OK:
					breaker.MarkSuccess()
				case codes.InvalidArgument, codes.NotFound, codes.AlreadyExists, codes.PermissionDenied:
					breaker.MarkSuccess()
				default:
					breaker.MarkFailed()
				}
			} else {
```

```
        breaker.MarkFailed()
    }
  } else {
    breaker.MarkSuccess()
  }

  return resp, err

 })
}

func StreamCircuitBreakerInterceptor(breaker circuitbreaker.CircuitBreaker) grpc.ServerOption {
 if breaker == nil {
  return grpc.ChainStreamInterceptor(func(srv any, ss grpc.ServerStream, info *grpc.StreamServerInfo, handler grpc.S
   return handler(srv, ss)
  })
 }

 return grpc.ChainStreamInterceptor(func(srv any, ss grpc.ServerStream, info *grpc.StreamServerInfo, handler grpc.St
  if err := breaker.Allow(); err != nil {
   log.ZWarn(ss.Context(), "rpc circuit breaker open", err, "method", info.FullMethod)
   return status.Error(codes.Unavailable, "service unavailable due to circuit breaker")
  }

  err := handler(srv, ss)

  if err != nil {
   if st, ok := status.FromError(err); ok {
    switch st.Code() {
    case codes.OK:
     breaker.MarkSuccess()
    case codes.InvalidArgument, codes.NotFound, codes.AlreadyExists, codes.PermissionDenied:
     breaker.MarkSuccess()
    default:
     breaker.MarkFailed()
    }
   } else {
    breaker.MarkFailed()
   }
  } else {
   breaker.MarkSuccess()
  }

  return err
 })
}
```

pkg/common/startrpc/mw.go

```
package startrpc

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"google.golang.org/grpc"
)

func grpcServerIMAdminUserID(imAdminUserID []string) grpc.ServerOption {
	return grpc.ChainUnaryInterceptor(func(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.Unary
		ctx = authverify.WithIMAdminUserIDs(ctx, imAdminUserID)
		return handler(ctx, req)
	})
}
```

pkg/common/startrpc/ratelimit.go

```
package startrpc

import (
	"context"
	"time"

	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/stability/ratelimit"
	"github.com/openimsdk/tools/stability/ratelimit/bbr"
	"google.golang.org/grpc"
	"google.golang.org/grpc/codes"
	"google.golang.org/grpc/status"
)

type RateLimiter struct {
	Enable       bool
	Window       time.Duration
	Bucket       int
	CPUThreshold int64
}

func NewRateLimiter(config *RateLimiter) ratelimit.Limiter {
	if !config.Enable {
		return nil
	}

	return bbr.NewBBRLimiter(
		bbr.WithWindow(config.Window),
		bbr.WithBucket(config.Bucket),
		bbr.WithCPUThreshold(config.CPUThreshold),
	)
}

func UnaryRateLimitInterceptor(limiter ratelimit.Limiter) grpc.ServerOption {
	if limiter == nil {
		return grpc.ChainUnaryInterceptor(func(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.Una
			return handler(ctx, req)
		})
	}

	return grpc.ChainUnaryInterceptor(func(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.Unary
		done, err := limiter.Allow()
		if err != nil {
			log.ZWarn(ctx, "rpc rate limited", err, "method", info.FullMethod)
			return nil, status.Errorf(codes.ResourceExhausted, "rpc request rate limit exceeded: %v, please try again later"
		}

		defer done(ratelimit.DoneInfo{})
		return handler(ctx, req)
	})
}

func StreamRateLimitInterceptor(limiter ratelimit.Limiter) grpc.ServerOption {
	if limiter == nil {
		return grpc.ChainStreamInterceptor(func(srv any, ss grpc.ServerStream, info *grpc.StreamServerInfo, handler grpc.S
			return handler(srv, ss)
		})
	}

	return grpc.ChainStreamInterceptor(func(srv any, ss grpc.ServerStream, info *grpc.StreamServerInfo, handler grpc.St
		done, err := limiter.Allow()
		if err != nil {
			log.ZWarn(ss.Context(), "rpc rate limited", err, "method", info.FullMethod)
			return status.Errorf(codes.ResourceExhausted, "rpc request rate limit exceeded: %v, please try again later", err
```

```
    }
    defer done(ratelimit.DoneInfo{})

    return handler(srv, ss)
  })
}
```

pkg/common/startrpc/start.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package startrpc

import (
	"context"
	"errors"
	"fmt"
	"net"
	"os"
	"os/signal"
	"reflect"
	"strconv"
	"syscall"
	"time"

	conf "github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/jsonutil"
	"github.com/openimsdk/tools/utils/network"
	"google.golang.org/grpc/status"

	kdisc "github.com/openimsdk/open-im-server/v3/pkg/common/discovery"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	grpccli "github.com/openimsdk/tools/mw/grpc/client"
	grpcsrv "github.com/openimsdk/tools/mw/grpc/server"
	"google.golang.org/grpc"
	"google.golang.org/grpc/credentials/insecure"
)

func init() {
	prommetrics.RegistryAll()
}

func Start[T any](ctx context.Context, disc *conf.Discovery, circuitBreakerConfig *conf.CircuitBreaker, rateLimiterC
	registerIP string, autoSetPorts bool, rpcPorts []int, index int, rpcRegisterName string, notification *conf.Notific
	watchConfigNames []string, watchServiceNames []string,
	rpcFn func(ctx context.Context, config T, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar) erro
	options ...grpc.ServerOption) error {

	if notification != nil {
		conf.InitNotification(notification)
	}

	maxRequestBody := getConfigRpcMaxRequestBody(reflect.ValueOf(config))
	shareConfig := getConfigShare(reflect.ValueOf(config))

	log.ZDebug(ctx, "rpc start", "rpcMaxRequestBody", maxRequestBody, "rpcRegisterName", rpcRegisterName, "registerIP",
```

```
	options = append(options,
		grpcsrv.GrpcServerMetadataContext(),
		grpcsrv.GrpcServerErrorConvert(),
		grpcsrv.GrpcServerLogger(),
		grpcsrv.GrpcServerRequestValidate(),
		grpcsrv.GrpcServerPanicCapture(),
	)
	if shareConfig != nil && len(shareConfig.IMAdminUser.UserIDs) > 0 {
		options = append(options, grpcServerIMAdminUserID(shareConfig.IMAdminUser.UserIDs))
	}
	var clientOptions []grpc.DialOption
	if maxRequestBody != nil {
		if maxRequestBody.RequestMaxBodySize > 0 {
			options = append(options, grpc.MaxRecvMsgSize(maxRequestBody.RequestMaxBodySize))
			clientOptions = append(clientOptions, grpc.WithDefaultCallOptions(grpc.MaxCallSendMsgSize(maxRequestBody.Request
		}
		if maxRequestBody.ResponseMaxBodySize > 0 {
			options = append(options, grpc.MaxSendMsgSize(maxRequestBody.ResponseMaxBodySize))
			clientOptions = append(clientOptions, grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(maxRequestBody.Respons
		}
	}

	if circuitBreakerConfig != nil && circuitBreakerConfig.Enable {
		cb := &CircuitBreaker{
			Enable:  circuitBreakerConfig.Enable,
			Success: circuitBreakerConfig.Success,
			Request: circuitBreakerConfig.Request,
			Bucket:  circuitBreakerConfig.Bucket,
			Window:  circuitBreakerConfig.Window,
		}

		breaker := NewCircuitBreaker(cb)

		options = append(options,
			UnaryCircuitBreakerInterceptor(breaker),
			StreamCircuitBreakerInterceptor(breaker),
		)

		log.ZInfo(ctx, "RPC circuit breaker enabled",
			"service", rpcRegisterName,
			"window", circuitBreakerConfig.Window,
			"bucket", circuitBreakerConfig.Bucket,
			"success", circuitBreakerConfig.Success,
			"requestThreshold", circuitBreakerConfig.Request)
	}

	if rateLimiterConfig != nil && rateLimiterConfig.Enable {
		limiter := NewRateLimiter((*RateLimiter)(rateLimiterConfig))

		options = append(options,
			UnaryRateLimitInterceptor(limiter),
			StreamRateLimitInterceptor(limiter),
		)

		log.ZInfo(ctx, "RPC rate limiter enabled",
			"service", rpcRegisterName,
			"window", rateLimiterConfig.Window,
			"bucket", rateLimiterConfig.Bucket,
			"cpuThreshold", rateLimiterConfig.CPUThreshold)
	}

	registerIP, err := network.GetRpcRegisterIP(registerIP)
	if err != nil {
		return err
	}
	var prometheusListenAddr string
```

```
	if autoSetPorts {
		prometheusListenAddr = net.JoinHostPort(listenIP, "0")
	} else {
		prometheusPort, err := datautil.GetElemByIndex(prometheusConfig.Ports, index)
		if err != nil {
			return err
		}
		prometheusListenAddr = net.JoinHostPort(listenIP, strconv.Itoa(prometheusPort))
	}

	watchConfigNames = append(watchConfigNames, conf.LogConfigFileName)

	client, err := kdisc.NewDiscoveryRegister(disc, watchServiceNames)
	if err != nil {
		return err
	}

	defer client.Close()
	client.AddOption(
		grpc.WithTransportCredentials(insecure.NewCredentials()),
		grpc.WithDefaultServiceConfig(fmt.Sprintf(`{"LoadBalancingPolicy": "%s"}`, "round_robin")),

		grpccli.GrpcClientLogger(),
		grpccli.GrpcClientContext(),
		grpccli.GrpcClientErrorConvert(),
	)
	if len(clientOptions) > 0 {
		client.AddOption(clientOptions...)
	}

	ctx, cancel := context.WithCancelCause(ctx)

	go func() {
		sigs := make(chan os.Signal, 1)
		signal.Notify(sigs, syscall.SIGTERM, syscall.SIGINT)
		select {
		case <-ctx.Done():
			return
		case val := <-sigs:
			log.ZDebug(ctx, "recv signal", "signal", val.String())
			cancel(fmt.Errorf("signal %s", val.String()))
		}
	}()

	if prometheusListenAddr != "" {
		options = append(
			options,
			prommetricsUnaryInterceptor(rpcRegisterName),
			prommetricsStreamInterceptor(rpcRegisterName),
		)
		prometheusListener, prometheusPort, err := listenTCP(prometheusListenAddr)
		if err != nil {
			return err
		}
		log.ZDebug(ctx, "prometheus start", "addr", prometheusListener.Addr(), "rpcRegisterName", rpcRegisterName)
		target, err := jsonutil.JsonMarshal(prommetrics.BuildDefaultTarget(registerIP, prometheusPort))
		if err != nil {
			return err
		}
		if autoSetPorts {
			if err = client.SetWithLease(ctx, prommetrics.BuildDiscoveryKey(rpcRegisterName, index), target, prommetrics.TTL
				if !errors.Is(err, discovery.ErrNotSupported) {
					return err
				}
			}
		}
	}
```

```
    go func() {
        err := prommetrics.Start(prometheusListener)
        if err == nil {
            err = fmt.Errorf("listener done")
        }
        cancel(fmt.Errorf("prommetrics %s %w", rpcRegisterName, err))
    }()
}

var (
    rpcServer       *grpc.Server
    rpcGracefulStop chan struct{}
)

onGrpcServiceRegistrar := func(desc *grpc.ServiceDesc, impl any) {
    if rpcServer != nil {
        rpcServer.RegisterService(desc, impl)
        return
    }
    var rpcListenAddr string
    if autoSetPorts {
        rpcListenAddr = net.JoinHostPort(listenIP, "0")
    } else {
        rpcPort, err := datautil.GetElemByIndex(rpcPorts, index)
        if err != nil {
            cancel(fmt.Errorf("rpcPorts index out of range %s %w", rpcRegisterName, err))
            return
        }
        rpcListenAddr = net.JoinHostPort(listenIP, strconv.Itoa(rpcPort))
    }
    rpcListener, err := net.Listen("tcp", rpcListenAddr)
    if err != nil {
        cancel(fmt.Errorf("listen rpc %s %s %w", rpcRegisterName, rpcListenAddr, err))
        return
    }

    rpcServer = grpc.NewServer(options...)
    rpcServer.RegisterService(desc, impl)
    rpcGracefulStop = make(chan struct{})
    rpcPort := rpcListener.Addr().(*net.TCPAddr).Port
    log.ZDebug(ctx, "rpc start register", "rpcRegisterName", rpcRegisterName, "registerIP", registerIP, "rpcPort", rp
    grpcOpt := grpc.WithTransportCredentials(insecure.NewCredentials())
    rpcGracefulStop = make(chan struct{})
    go func() {
        <-ctx.Done()
        rpcServer.GracefulStop()
        close(rpcGracefulStop)
    }()
    if err := client.Register(ctx, rpcRegisterName, registerIP, rpcListener.Addr().(*net.TCPAddr).Port, grpcOpt); err
        cancel(fmt.Errorf("rpc register %s %w", rpcRegisterName, err))
        return
    }

    go func() {
        err := rpcServer.Serve(rpcListener)
        if err == nil {
            err = fmt.Errorf("serve end")
        }
        cancel(fmt.Errorf("rpc %s %w", rpcRegisterName, err))
    }()
}

err = rpcFn(ctx, config, client, &grpcServiceRegistrar{onRegisterService: onGrpcServiceRegistrar})
if err != nil {
    return err
}
```

```
	<-ctx.Done()
	log.ZDebug(ctx, "cmd wait done", "err", context.Cause(ctx))
	if rpcGracefulStop != nil {
		timeout := time.NewTimer(time.Second * 15)
		defer timeout.Stop()
		select {
		case <-timeout.C:
			log.ZWarn(ctx, "rcp graceful stop timeout", nil)
		case <-rpcGracefulStop:
			log.ZDebug(ctx, "rcp graceful stop done")
		}
	}
	return context.Cause(ctx)
}

func listenTCP(addr string) (net.Listener, int, error) {
	listener, err := net.Listen("tcp", addr)
	if err != nil {
		return nil, 0, errs.WrapMsg(err, "listen err", "addr", addr)
	}
	return listener, listener.Addr().(*net.TCPAddr).Port, nil
}

func prommetricsUnaryInterceptor(rpcRegisterName string) grpc.ServerOption {
	getCode := func(err error) int {
		if err == nil {
			return 0
		}
		rpcErr, ok := err.(interface{ GRPCStatus() *status.Status })
		if !ok {
			return -1
		}
		return int(rpcErr.GRPCStatus().Code())
	}
	return grpc.ChainUnaryInterceptor(func(ctx context.Context, req any, info *grpc.UnaryServerInfo, handler grpc.Unary
		resp, err := handler(ctx, req)
		prommetrics.RPCCall(rpcRegisterName, info.FullMethod, getCode(err))
		return resp, err
	})
}

func prommetricsStreamInterceptor(rpcRegisterName string) grpc.ServerOption {
	return grpc.ChainStreamInterceptor()
}

type grpcServiceRegistrar struct {
	onRegisterService func(desc *grpc.ServiceDesc, impl any)
}

func (x *grpcServiceRegistrar) RegisterService(desc *grpc.ServiceDesc, impl any) {
	x.onRegisterService(desc, impl)
}
```

pkg/common/startrpc/tools.go

```
package startrpc

import (
	"reflect"

	conf "github.com/openimsdk/open-im-server/v3/pkg/common/config"
)

func getConfig[T any](value reflect.Value) *T {
	for value.Kind() == reflect.Pointer {
		value = value.Elem()
	}
	if value.Kind() == reflect.Struct {
		num := value.NumField()
		for i := 0; i < num; i++ {
			field := value.Field(i)
			for field.Kind() == reflect.Pointer {
				field = field.Elem()
			}
			if field.Kind() == reflect.Struct {
				if elem, ok := field.Interface().(T); ok {
					return &elem
				}
				if elem := getConfig[T](field); elem != nil {
					return elem
				}
			}
		}
	}
	return nil
}

func getConfigRpcMaxRequestBody(value reflect.Value) *conf.MaxRequestBody {
	return getConfig[conf.MaxRequestBody](value)
}

func getConfigShare(value reflect.Value) *conf.Share {
	return getConfig[conf.Share](value)
}
```

pkg/common/config

pkg/common/config/config.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package config

import (
	"strings"
	"time"

	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"github.com/openimsdk/tools/mq/kafka"
	"github.com/openimsdk/tools/s3/aws"
	"github.com/openimsdk/tools/s3/cos"
	"github.com/openimsdk/tools/s3/kodo"
	"github.com/openimsdk/tools/s3/minio"
	"github.com/openimsdk/tools/s3/oss"
)

const StructTagName = "yaml"

type Path string

type Index int

type CacheConfig struct {
	Topic         string `yaml:"topic"`
	SlotNum       int    `yaml:"slotNum"`
	SlotSize      int    `yaml:"slotSize"`
	SuccessExpire int    `yaml:"successExpire"`
	FailedExpire  int    `yaml:"failedExpire"`
}

type LocalCache struct {
	Auth         CacheConfig `yaml:"auth"`
	User         CacheConfig `yaml:"user"`
	Group        CacheConfig `yaml:"group"`
	Friend       CacheConfig `yaml:"friend"`
	Conversation CacheConfig `yaml:"conversation"`
}

type Log struct {
	StorageLocation     string `yaml:"storageLocation"`
	RotationTime        uint   `yaml:"rotationTime"`
	RemainRotationCount uint   `yaml:"remainRotationCount"`
	RemainLogLevel      int    `yaml:"remainLogLevel"`
	IsStdout            bool   `yaml:"isStdout"`
	IsJson              bool   `yaml:"isJson"`
	IsSimplify          bool   `yaml:"isSimplify"`
	WithStack           bool   `yaml:"withStack"`
```

```
}

type Minio struct {
    Bucket          string `yaml:"bucket"`
    AccessKeyID     string `yaml:"accessKeyID"`
    SecretAccessKey string `yaml:"secretAccessKey"`
    SessionToken    string `yaml:"sessionToken"`
    InternalAddress string `yaml:"internalAddress"`
    ExternalAddress string `yaml:"externalAddress"`
    PublicRead      bool   `yaml:"publicRead"`
}

type Mongo struct {
    URI            string   `yaml:"uri"`
    Address        []string `yaml:"address"`
    Database       string   `yaml:"database"`
    Username       string   `yaml:"username"`
    Password       string   `yaml:"password"`
    AuthSource     string   `yaml:"authSource"`
    MaxPoolSize    int      `yaml:"maxPoolSize"`
    MaxRetry       int      `yaml:"maxRetry"`
    MongoMode      string   `yaml:"mongoMode"`
    ReplicaSet     ReplicaSetConfig
    ReadPreference ReadPrefConfig
    WriteConcern   WriteConcernConfig
}

type ReplicaSetConfig struct {
    Name         string        `yaml:"name"`
    Hosts        []string      `yaml:"hosts"`
    ReadConcern  string        `yaml:"readConcern"`
    MaxStaleness time.Duration `yaml:"maxStaleness"`
}

type ReadPrefConfig struct {
    Mode         string              `yaml:"mode"`
    TagSets      []map[string]string `yaml:"tagSets"`
    MaxStaleness time.Duration       `yaml:"maxStaleness"`
}

type WriteConcernConfig struct {
    W        any           `yaml:"w"`
    J        bool          `yaml:"j"`
    WTimeout time.Duration `yaml:"wtimeout"`
}

type Kafka struct {
    Username           string   `yaml:"username"`
    Password           string   `yaml:"password"`
    ProducerAck        string   `yaml:"producerAck"`
    CompressType       string   `yaml:"compressType"`
    Address            []string `yaml:"address"`
    ToRedisTopic       string   `yaml:"toRedisTopic"`
    ToMongoTopic       string   `yaml:"toMongoTopic"`
    ToPushTopic        string   `yaml:"toPushTopic"`
    ToOfflinePushTopic string   `yaml:"toOfflinePushTopic"`
    ToRedisGroupID     string   `yaml:"toRedisGroupID"`
    ToMongoGroupID     string   `yaml:"toMongoGroupID"`
    ToPushGroupID      string   `yaml:"toPushGroupID"`
    ToOfflineGroupID   string   `yaml:"toOfflinePushGroupID"`

    Tls TLSConfig `yaml:"tls"`
}
type TLSConfig struct {
    EnableTLS          bool   `yaml:"enableTLS"`
    CACrt              string `yaml:"caCrt"`
```

```
	ClientCrt          string `yaml:"clientCrt"`
	ClientKey          string `yaml:"clientKey"`
	ClientKeyPwd       string `yaml:"clientKeyPwd"`
	InsecureSkipVerify bool   `yaml:"insecureSkipVerify"`
}

type API struct {
	Api struct {
		ListenIP         string `yaml:"listenIP"`
		Ports            []int  `yaml:"ports"`
		CompressionLevel int    `yaml:"compressionLevel"`
	} `yaml:"api"`
	Prometheus struct {
		Enable       bool   `yaml:"enable"`
		AutoSetPorts bool   `yaml:"autoSetPorts"`
		Ports        []int  `yaml:"ports"`
		GrafanaURL   string `yaml:"grafanaURL"`
	} `yaml:"prometheus"`

	RateLimiter RateLimiter `yaml:"rateLimiter"`
}

type RateLimiter struct {
	Enable       bool          `yaml:"enable"`
	Window       time.Duration `yaml:"window"`
	Bucket       int           `yaml:"bucket"`
	CPUThreshold int64         `yaml:"cpuThreshold"`
}

type CircuitBreaker struct {
	Enable  bool          `yaml:"enable"`
	Window  time.Duration `yaml:"window"`
	Bucket  int           `yaml:"bucket"`
	Success float64       `yaml:"success"`
	Request int64         `yaml:"request"`
}

type CronTask struct {
	CronExecuteTime   string   `yaml:"cronExecuteTime"`
	RetainChatRecords int      `yaml:"retainChatRecords"`
	FileExpireTime    int      `yaml:"fileExpireTime"`
	DeleteObjectType  []string `yaml:"deleteObjectType"`
}

type OfflinePushConfig struct {
	Enable bool   `yaml:"enable"`
	Title  string `yaml:"title"`
	Desc   string `yaml:"desc"`
	Ext    string `yaml:"ext"`
}

type NotificationConfig struct {
	IsSendMsg        bool              `yaml:"isSendMsg"`
	ReliabilityLevel int               `yaml:"reliabilityLevel"`
	UnreadCount      bool              `yaml:"unreadCount"`
	OfflinePush      OfflinePushConfig `yaml:"offlinePush"`
}

type Notification struct {
	GroupCreated              NotificationConfig `yaml:"groupCreated"`
	GroupInfoSet              NotificationConfig `yaml:"groupInfoSet"`
	JoinGroupApplication      NotificationConfig `yaml:"joinGroupApplication"`
	MemberQuit                NotificationConfig `yaml:"memberQuit"`
	GroupApplicationAccepted  NotificationConfig `yaml:"groupApplicationAccepted"`
	GroupApplicationRejected  NotificationConfig `yaml:"groupApplicationRejected"`
	GroupOwnerTransferred     NotificationConfig `yaml:"groupOwnerTransferred"`
```

```
	MemberKicked              NotificationConfig `yaml:"memberKicked"`
	MemberInvited             NotificationConfig `yaml:"memberInvited"`
	MemberEnter               NotificationConfig `yaml:"memberEnter"`
	GroupDismissed            NotificationConfig `yaml:"groupDismissed"`
	GroupMuted                NotificationConfig `yaml:"groupMuted"`
	GroupCancelMuted          NotificationConfig `yaml:"groupCancelMuted"`
	GroupMemberMuted          NotificationConfig `yaml:"groupMemberMuted"`
	GroupMemberCancelMuted    NotificationConfig `yaml:"groupMemberCancelMuted"`
	GroupMemberInfoSet        NotificationConfig `yaml:"groupMemberInfoSet"`
	GroupMemberSetToAdmin     NotificationConfig `yaml:"groupMemberSetToAdmin"`
	GroupMemberSetToOrdinary  NotificationConfig `yaml:"groupMemberSetToOrdinaryUser"`
	GroupInfoSetAnnouncement  NotificationConfig `yaml:"groupInfoSetAnnouncement"`
	GroupInfoSetName          NotificationConfig `yaml:"groupInfoSetName"`
	FriendApplicationAdded    NotificationConfig `yaml:"friendApplicationAdded"`
	FriendApplicationApproved NotificationConfig `yaml:"friendApplicationApproved"`
	FriendApplicationRejected NotificationConfig `yaml:"friendApplicationRejected"`
	FriendAdded               NotificationConfig `yaml:"friendAdded"`
	FriendDeleted             NotificationConfig `yaml:"friendDeleted"`
	FriendRemarkSet           NotificationConfig `yaml:"friendRemarkSet"`
	BlackAdded                NotificationConfig `yaml:"blackAdded"`
	BlackDeleted              NotificationConfig `yaml:"blackDeleted"`
	FriendInfoUpdated         NotificationConfig `yaml:"friendInfoUpdated"`
	UserInfoUpdated           NotificationConfig `yaml:"userInfoUpdated"`
	UserStatusChanged         NotificationConfig `yaml:"userStatusChanged"`
	ConversationChanged       NotificationConfig `yaml:"conversationChanged"`
	ConversationSetPrivate    NotificationConfig `yaml:"conversationSetPrivate"`
}

type Prometheus struct {
	Enable bool  `yaml:"enable"`
	Ports  []int `yaml:"ports"`
}

type MsgGateway struct {
	RPC         RPC        `yaml:"rpc"`
	Prometheus  Prometheus `yaml:"prometheus"`
	ListenIP    string     `yaml:"listenIP"`
	LongConnSvr struct {
		Ports               []int `yaml:"ports"`
		WebsocketMaxConnNum int   `yaml:"websocketMaxConnNum"`
		WebsocketMaxMsgLen  int   `yaml:"websocketMaxMsgLen"`
		WebsocketTimeout    int   `yaml:"websocketTimeout"`
	} `yaml:"longConnSvr"`
	RateLimiter    RateLimiter    `yaml:"rateLimiter"`
	CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type MsgTransfer struct {
	Prometheus struct {
		Enable       bool  `yaml:"enable"`
		AutoSetPorts bool  `yaml:"autoSetPorts"`
		Ports        []int `yaml:"ports"`
	} `yaml:"prometheus"`
	RateLimiter    RateLimiter    `yaml:"rateLimiter"`
	CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type Push struct {
	RPC                 RPC        `yaml:"rpc"`
	Prometheus          Prometheus `yaml:"prometheus"`
	MaxConcurrentWorkers int        `yaml:"maxConcurrentWorkers"`
	Enable              string     `yaml:"enable"`
	GeTui               struct {
		PushUrl      string `yaml:"pushUrl"`
		MasterSecret string `yaml:"masterSecret"`
		AppKey       string `yaml:"appKey"`
```

```
    Intent       string `yaml:"intent"`
    ChannelID    string `yaml:"channelID"`
    ChannelName  string `yaml:"channelName"`
  } `yaml:"geTui"`
  FCM struct {
    FilePath string `yaml:"filePath"`
    AuthURL  string `yaml:"authURL"`
  } `yaml:"fcm"`
  JPush struct {
    AppKey       string `yaml:"appKey"`
    MasterSecret string `yaml:"masterSecret"`
    PushURL      string `yaml:"pushURL"`
    PushIntent   string `yaml:"pushIntent"`
  } `yaml:"jpush"`
  IOSPush struct {
    PushSound  string `yaml:"pushSound"`
    BadgeCount bool   `yaml:"badgeCount"`
    Production bool   `yaml:"production"`
  } `yaml:"iosPush"`
  FullUserCache  bool           `yaml:"fullUserCache"`
  RateLimiter    RateLimiter    `yaml:"rateLimiter"`
  CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type Auth struct {
  RPC         RPC        `yaml:"rpc"`
  Prometheus  Prometheus `yaml:"prometheus"`
  TokenPolicy struct {
    Expire int64 `yaml:"expire"`
  } `yaml:"tokenPolicy"`
  RateLimiter    RateLimiter    `yaml:"rateLimiter"`
  CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type Conversation struct {
  RPC            RPC            `yaml:"rpc"`
  Prometheus     Prometheus     `yaml:"prometheus"`
  RateLimiter    RateLimiter    `yaml:"rateLimiter"`
  CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type Friend struct {
  RPC            RPC            `yaml:"rpc"`
  Prometheus     Prometheus     `yaml:"prometheus"`
  RateLimiter    RateLimiter    `yaml:"rateLimiter"`
  CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type Group struct {
  RPC                        RPC            `yaml:"rpc"`
  Prometheus                 Prometheus     `yaml:"prometheus"`
  EnableHistoryForNewMembers bool           `yaml:"enableHistoryForNewMembers"`
  RateLimiter                RateLimiter    `yaml:"rateLimiter"`
  CircuitBreaker             CircuitBreaker `yaml:"circuitBreaker"`
}

type Msg struct {
  RPC            RPC            `yaml:"rpc"`
  Prometheus     Prometheus     `yaml:"prometheus"`
  FriendVerify   bool           `yaml:"friendVerify"`
  RateLimiter    RateLimiter    `yaml:"rateLimiter"`
  CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type Third struct {
  RPC        RPC        `yaml:"rpc"`
```

```
    Prometheus Prometheus `yaml:"prometheus"`
    Object     struct {
        Enable string `yaml:"enable"`
        Cos    Cos    `yaml:"cos"`
        Oss    Oss    `yaml:"oss"`
        Kodo   Kodo   `yaml:"kodo"`
        Aws    Aws    `yaml:"aws"`
    } `yaml:"object"`
    RateLimiter    RateLimiter    `yaml:"rateLimiter"`
    CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}
type Cos struct {
    BucketURL    string `yaml:"bucketURL"`
    SecretID     string `yaml:"secretID"`
    SecretKey    string `yaml:"secretKey"`
    SessionToken string `yaml:"sessionToken"`
    PublicRead   bool   `yaml:"publicRead"`
}
type Oss struct {
    Endpoint        string `yaml:"endpoint"`
    Bucket          string `yaml:"bucket"`
    BucketURL       string `yaml:"bucketURL"`
    AccessKeyID     string `yaml:"accessKeyID"`
    AccessKeySecret string `yaml:"accessKeySecret"`
    SessionToken    string `yaml:"sessionToken"`
    PublicRead      bool   `yaml:"publicRead"`
}

type Kodo struct {
    Endpoint        string `yaml:"endpoint"`
    Bucket          string `yaml:"bucket"`
    BucketURL       string `yaml:"bucketURL"`
    AccessKeyID     string `yaml:"accessKeyID"`
    AccessKeySecret string `yaml:"accessKeySecret"`
    SessionToken    string `yaml:"sessionToken"`
    PublicRead      bool   `yaml:"publicRead"`
}

type Aws struct {
    Region          string `yaml:"region"`
    Bucket          string `yaml:"bucket"`
    AccessKeyID     string `yaml:"accessKeyID"`
    SecretAccessKey string `yaml:"secretAccessKey"`
    SessionToken    string `yaml:"sessionToken"`
    PublicRead      bool   `yaml:"publicRead"`
}

type User struct {
    RPC            RPC            `yaml:"rpc"`
    Prometheus     Prometheus     `yaml:"prometheus"`
    RateLimiter    RateLimiter    `yaml:"rateLimiter"`
    CircuitBreaker CircuitBreaker `yaml:"circuitBreaker"`
}

type RPC struct {
    RegisterIP   string `yaml:"registerIP"`
    ListenIP     string `yaml:"listenIP"`
    AutoSetPorts bool   `yaml:"autoSetPorts"`
    Ports        []int  `yaml:"ports"`
}

type Redis struct {
    Disable      bool     `yaml:"-"`
    Address      []string `yaml:"address"`
    Username     string   `yaml:"username"`
    Password     string   `yaml:"password"`
```

```
	RedisMode    string   `yaml:"redisMode"`
	DB           int      `yaml:"db"`
	MaxRetry     int      `yaml:"maxRetry"`
	PoolSize     int      `yaml:"poolSize"`
	SentinelMode Sentinel `yaml:"sentinelMode"`
}

type Sentinel struct {
	MasterName     string   `yaml:"masterName"`
	SentinelAddrs  []string `yaml:"sentinelsAddrs"`
	RouteByLatency bool     `yaml:"routeByLatency"`
	RouteRandomly  bool     `yaml:"routeRandomly"`
}

type BeforeConfig struct {
	Enable         bool    `yaml:"enable"`
	Timeout        int     `yaml:"timeout"`
	FailedContinue bool    `yaml:"failedContinue"`
	DeniedTypes    []int32 `yaml:"deniedTypes"`
}

type AfterConfig struct {
	Enable       bool     `yaml:"enable"`
	Timeout      int      `yaml:"timeout"`
	AttentionIds []string `yaml:"attentionIds"`
	DeniedTypes  []int32  `yaml:"deniedTypes"`
}

type Share struct {
	Secret      string `yaml:"secret"`
	IMAdminUser struct {
		UserIDs   []string `yaml:"userIDs"`
		Nicknames []string `yaml:"nicknames"`
	} `yaml:"imAdminUser"`
	MultiLogin     MultiLogin     `yaml:"multiLogin"`
	RPCMaxBodySize MaxRequestBody `yaml:"rpcMaxBodySize"`
}

type MaxRequestBody struct {
	RequestMaxBodySize  int `yaml:"requestMaxBodySize"`
	ResponseMaxBodySize int `yaml:"responseMaxBodySize"`
}

type MultiLogin struct {
	Policy       int `yaml:"policy"`
	MaxNumOneEnd int `yaml:"maxNumOneEnd"`
}

type RpcService struct {
	User           string `yaml:"user"`
	Friend         string `yaml:"friend"`
	Msg            string `yaml:"msg"`
	Push           string `yaml:"push"`
	MessageGateway string `yaml:"messageGateway"`
	Group          string `yaml:"group"`
	Auth           string `yaml:"auth"`
	Conversation   string `yaml:"conversation"`
	Third          string `yaml:"third"`
}

func (r *RpcService) GetServiceNames() []string {
	return []string{
		r.User,
		r.Friend,
		r.Msg,
		r.Push,
```

```
    r.MessageGateway,
    r.Group,
    r.Auth,
    r.Conversation,
    r.Third,
  }
}

// FullConfig stores all configurations for before and after events
type Webhooks struct {
  URL                                string       `yaml:"url"`
  BeforeSendSingleMsg                BeforeConfig `yaml:"beforeSendSingleMsg"`
  BeforeUpdateUserInfoEx             BeforeConfig `yaml:"beforeUpdateUserInfoEx"`
  AfterUpdateUserInfoEx              AfterConfig  `yaml:"afterUpdateUserInfoEx"`
  AfterSendSingleMsg                 AfterConfig  `yaml:"afterSendSingleMsg"`
  BeforeSendGroupMsg                 BeforeConfig `yaml:"beforeSendGroupMsg"`
  BeforeMsgModify                    BeforeConfig `yaml:"beforeMsgModify"`
  AfterSendGroupMsg                  AfterConfig  `yaml:"afterSendGroupMsg"`
  AfterMsgSaveDB                     AfterConfig  `yaml:"afterMsgSaveDB"`
  AfterUserOnline                    AfterConfig  `yaml:"afterUserOnline"`
  AfterUserOffline                   AfterConfig  `yaml:"afterUserOffline"`
  AfterUserKickOff                   AfterConfig  `yaml:"afterUserKickOff"`
  BeforeOfflinePush                  BeforeConfig `yaml:"beforeOfflinePush"`
  BeforeOnlinePush                   BeforeConfig `yaml:"beforeOnlinePush"`
  BeforeGroupOnlinePush              BeforeConfig `yaml:"beforeGroupOnlinePush"`
  BeforeAddFriend                    BeforeConfig `yaml:"beforeAddFriend"`
  BeforeUpdateUserInfo               BeforeConfig `yaml:"beforeUpdateUserInfo"`
  AfterUpdateUserInfo                AfterConfig  `yaml:"afterUpdateUserInfo"`
  BeforeCreateGroup                  BeforeConfig `yaml:"beforeCreateGroup"`
  AfterCreateGroup                   AfterConfig  `yaml:"afterCreateGroup"`
  BeforeMemberJoinGroup              BeforeConfig `yaml:"beforeMemberJoinGroup"`
  BeforeSetGroupMemberInfo           BeforeConfig `yaml:"beforeSetGroupMemberInfo"`
  AfterSetGroupMemberInfo            AfterConfig  `yaml:"afterSetGroupMemberInfo"`
  AfterQuitGroup                     AfterConfig  `yaml:"afterQuitGroup"`
  AfterKickGroupMember               AfterConfig  `yaml:"afterKickGroupMember"`
  AfterDismissGroup                  AfterConfig  `yaml:"afterDismissGroup"`
  BeforeApplyJoinGroup               BeforeConfig `yaml:"beforeApplyJoinGroup"`
  AfterGroupMsgRead                  AfterConfig  `yaml:"afterGroupMsgRead"`
  AfterSingleMsgRead                 AfterConfig  `yaml:"afterSingleMsgRead"`
  BeforeUserRegister                 BeforeConfig `yaml:"beforeUserRegister"`
  AfterUserRegister                  AfterConfig  `yaml:"afterUserRegister"`
  AfterTransferGroupOwner            AfterConfig  `yaml:"afterTransferGroupOwner"`
  BeforeSetFriendRemark              BeforeConfig `yaml:"beforeSetFriendRemark"`
  AfterSetFriendRemark               AfterConfig  `yaml:"afterSetFriendRemark"`
  AfterGroupMsgRevoke                AfterConfig  `yaml:"afterGroupMsgRevoke"`
  AfterJoinGroup                     AfterConfig  `yaml:"afterJoinGroup"`
  BeforeInviteUserToGroup            BeforeConfig `yaml:"beforeInviteUserToGroup"`
  AfterSetGroupInfo                  AfterConfig  `yaml:"afterSetGroupInfo"`
  BeforeSetGroupInfo                 BeforeConfig `yaml:"beforeSetGroupInfo"`
  AfterSetGroupInfoEx                AfterConfig  `yaml:"afterSetGroupInfoEx"`
  BeforeSetGroupInfoEx               BeforeConfig `yaml:"beforeSetGroupInfoEx"`
  AfterRevokeMsg                     AfterConfig  `yaml:"afterRevokeMsg"`
  BeforeAddBlack                     BeforeConfig `yaml:"beforeAddBlack"`
  AfterAddFriend                     AfterConfig  `yaml:"afterAddFriend"`
  BeforeAddFriendAgree               BeforeConfig `yaml:"beforeAddFriendAgree"`
  AfterAddFriendAgree                AfterConfig  `yaml:"afterAddFriendAgree"`
  AfterDeleteFriend                  AfterConfig  `yaml:"afterDeleteFriend"`
  BeforeImportFriends                BeforeConfig `yaml:"beforeImportFriends"`
  AfterImportFriends                 AfterConfig  `yaml:"afterImportFriends"`
  AfterRemoveBlack                   AfterConfig  `yaml:"afterRemoveBlack"`
  BeforeCreateSingleChatConversations BeforeConfig `yaml:"beforeCreateSingleChatConversations"`
  AfterCreateSingleChatConversations  AfterConfig  `yaml:"afterCreateSingleChatConversations"`
  BeforeCreateGroupChatConversations  BeforeConfig `yaml:"beforeCreateGroupChatConversations"`
  AfterCreateGroupChatConversations   AfterConfig  `yaml:"afterCreateGroupChatConversations"`
}
```

```
type ZooKeeper struct {
	Schema   string   `yaml:"schema"`
	Address  []string `yaml:"address"`
	Username string   `yaml:"username"`
	Password string   `yaml:"password"`
}

type Discovery struct {
	Enable     string     `yaml:"enable"`
	Etcd       Etcd       `yaml:"etcd"`
	Kubernetes Kubernetes `yaml:"kubernetes"`
	RpcService RpcService `yaml:"rpcService"`
}

type Kubernetes struct {
	Namespace string `yaml:"namespace"`
}

type Etcd struct {
	RootDirectory string   `yaml:"rootDirectory"`
	Address       []string `yaml:"address"`
	Username      string   `yaml:"username"`
	Password      string   `yaml:"password"`
}

func (m *Mongo) Build() *mongoutil.Config {
	return &mongoutil.Config{
		Uri:         m.URI,
		Address:     m.Address,
		Database:    m.Database,
		Username:    m.Username,
		Password:    m.Password,
		AuthSource:  m.AuthSource,
		MaxPoolSize: m.MaxPoolSize,
		MaxRetry:    m.MaxRetry,
		MongoMode:   m.MongoMode,
		ReplicaSet: &mongoutil.ReplicaSetConfig{
			Name:         m.ReplicaSet.Name,
			Hosts:        m.ReplicaSet.Hosts,
			ReadConcern:  m.ReplicaSet.ReadConcern,
			MaxStaleness: m.ReplicaSet.MaxStaleness,
		},
		ReadPreference: &mongoutil.ReadPrefConfig{
			Mode:         m.ReadPreference.Mode,
			TagSets:      m.ReadPreference.TagSets,
			MaxStaleness: m.ReadPreference.MaxStaleness,
		},
		WriteConcern: &mongoutil.WriteConcernConfig{
			W:        m.WriteConcern.W,
			J:        m.WriteConcern.J,
			WTimeout: m.WriteConcern.WTimeout,
		},
	}
}

func (r *Redis) Build() *redisutil.Config {
	return &redisutil.Config{
		RedisMode: r.RedisMode,
		Address:   r.Address,
		Username:  r.Username,
		Password:  r.Password,
		DB:        r.DB,
		MaxRetry:  r.MaxRetry,
		PoolSize:  r.PoolSize,
		Sentinel: &redisutil.Sentinel{
			MasterName:     r.SentinelMode.MasterName,
```

```
        SentinelAddrs:  r.SentinelMode.SentinelAddrs,
        RouteByLatency: r.SentinelMode.RouteByLatency,
        RouteRandomly:  r.SentinelMode.RouteRandomly,
    },
    }
}

func (k *Kafka) Build() *kafka.Config {
    return &kafka.Config{
        Username:     k.Username,
        Password:     k.Password,
        ProducerAck:  k.ProducerAck,
        CompressType: k.CompressType,
        Addr:         k.Address,
        TLS: kafka.TLSConfig{
            EnableTLS:          k.Tls.EnableTLS,
            CACrt:              k.Tls.CACrt,
            ClientCrt:          k.Tls.ClientCrt,
            ClientKey:          k.Tls.ClientKey,
            ClientKeyPwd:       k.Tls.ClientKeyPwd,
            InsecureSkipVerify: k.Tls.InsecureSkipVerify,
        },
    }
}

func (m *Minio) Build() *minio.Config {
    formatEndpoint := func(address string) string {
        if strings.HasPrefix(address, "http://") || strings.HasPrefix(address, "https://") {
            return address
        }
        return "http://" + address
    }
    return &minio.Config{
        Bucket:          m.Bucket,
        AccessKeyID:     m.AccessKeyID,
        SecretAccessKey: m.SecretAccessKey,
        SessionToken:    m.SessionToken,
        PublicRead:      m.PublicRead,
        Endpoint:        formatEndpoint(m.InternalAddress),
        SignEndpoint:    formatEndpoint(m.ExternalAddress),
    }
}

func (c *Cos) Build() *cos.Config {
    return &cos.Config{
        BucketURL:    c.BucketURL,
        SecretID:     c.SecretID,
        SecretKey:    c.SecretKey,
        SessionToken: c.SessionToken,
        PublicRead:   c.PublicRead,
    }
}

func (o *Oss) Build() *oss.Config {
    return &oss.Config{
        Endpoint:        o.Endpoint,
        Bucket:          o.Bucket,
        BucketURL:       o.BucketURL,
        AccessKeyID:     o.AccessKeyID,
        AccessKeySecret: o.AccessKeySecret,
        SessionToken:    o.SessionToken,
        PublicRead:      o.PublicRead,
    }
}

func (o *Kodo) Build() *kodo.Config {
```

```
    return &kodo.Config{
        Endpoint:        o.Endpoint,
        Bucket:          o.Bucket,
        BucketURL:       o.BucketURL,
        AccessKeyID:     o.AccessKeyID,
        AccessKeySecret: o.AccessKeySecret,
        SessionToken:    o.SessionToken,
        PublicRead:      o.PublicRead,
    }
}

func (o *Aws) Build() *aws.Config {
    return &aws.Config{
        Region:          o.Region,
        Bucket:          o.Bucket,
        AccessKeyID:     o.AccessKeyID,
        SecretAccessKey: o.SecretAccessKey,
        SessionToken:    o.SessionToken,
    }
}

func (l *CacheConfig) Failed() time.Duration {
    return time.Second * time.Duration(l.FailedExpire)
}

func (l *CacheConfig) Success() time.Duration {
    return time.Second * time.Duration(l.SuccessExpire)
}

func (l *CacheConfig) Enable() bool {
    return l.Topic != "" && l.SlotNum > 0 && l.SlotSize > 0
}

func InitNotification(notification *Notification) {
    notification.GroupCreated.UnreadCount = false
    notification.GroupCreated.ReliabilityLevel = 1
    notification.GroupInfoSet.UnreadCount = false
    notification.GroupInfoSet.ReliabilityLevel = 1
    notification.JoinGroupApplication.UnreadCount = false
    notification.JoinGroupApplication.ReliabilityLevel = 1
    notification.MemberQuit.UnreadCount = false
    notification.MemberQuit.ReliabilityLevel = 1
    notification.GroupApplicationAccepted.UnreadCount = false
    notification.GroupApplicationAccepted.ReliabilityLevel = 1
    notification.GroupApplicationRejected.UnreadCount = false
    notification.GroupApplicationRejected.ReliabilityLevel = 1
    notification.GroupOwnerTransferred.UnreadCount = false
    notification.GroupOwnerTransferred.ReliabilityLevel = 1
    notification.MemberKicked.UnreadCount = false
    notification.MemberKicked.ReliabilityLevel = 1
    notification.MemberInvited.UnreadCount = false
    notification.MemberInvited.ReliabilityLevel = 1
    notification.MemberEnter.UnreadCount = false
    notification.MemberEnter.ReliabilityLevel = 1
    notification.GroupDismissed.UnreadCount = false
    notification.GroupDismissed.ReliabilityLevel = 1
    notification.GroupMuted.UnreadCount = false
    notification.GroupMuted.ReliabilityLevel = 1
    notification.GroupCancelMuted.UnreadCount = false
    notification.GroupCancelMuted.ReliabilityLevel = 1
    notification.GroupMemberMuted.UnreadCount = false
    notification.GroupMemberMuted.ReliabilityLevel = 1
    notification.GroupMemberCancelMuted.UnreadCount = false
    notification.GroupMemberCancelMuted.ReliabilityLevel = 1
    notification.GroupMemberInfoSet.UnreadCount = false
    notification.GroupMemberInfoSet.ReliabilityLevel = 1
```

```
    notification.GroupMemberSetToAdmin.UnreadCount = false
    notification.GroupMemberSetToAdmin.ReliabilityLevel = 1
    notification.GroupMemberSetToOrdinary.UnreadCount = false
    notification.GroupMemberSetToOrdinary.ReliabilityLevel = 1
    notification.GroupInfoSetAnnouncement.UnreadCount = false
    notification.GroupInfoSetAnnouncement.ReliabilityLevel = 1
    notification.GroupInfoSetName.UnreadCount = false
    notification.GroupInfoSetName.ReliabilityLevel = 1
    notification.FriendApplicationAdded.UnreadCount = false
    notification.FriendApplicationAdded.ReliabilityLevel = 1
    notification.FriendApplicationApproved.UnreadCount = false
    notification.FriendApplicationApproved.ReliabilityLevel = 1
    notification.FriendApplicationRejected.UnreadCount = false
    notification.FriendApplicationRejected.ReliabilityLevel = 1
    notification.FriendAdded.UnreadCount = false
    notification.FriendAdded.ReliabilityLevel = 1
    notification.FriendDeleted.UnreadCount = false
    notification.FriendDeleted.ReliabilityLevel = 1
    notification.FriendRemarkSet.UnreadCount = false
    notification.FriendRemarkSet.ReliabilityLevel = 1
    notification.BlackAdded.UnreadCount = false
    notification.BlackAdded.ReliabilityLevel = 1
    notification.BlackDeleted.UnreadCount = false
    notification.BlackDeleted.ReliabilityLevel = 1
    notification.FriendInfoUpdated.UnreadCount = false
    notification.FriendInfoUpdated.ReliabilityLevel = 1
    notification.UserInfoUpdated.UnreadCount = false
    notification.UserInfoUpdated.ReliabilityLevel = 1
    notification.UserStatusChanged.UnreadCount = false
    notification.UserStatusChanged.ReliabilityLevel = 1
    notification.ConversationChanged.UnreadCount = false
    notification.ConversationChanged.ReliabilityLevel = 1
    notification.ConversationSetPrivate.UnreadCount = false
    notification.ConversationSetPrivate.ReliabilityLevel = 1
}

type AllConfig struct {
    Discovery    Discovery
    Kafka        Kafka
    LocalCache   LocalCache
    Log          Log
    Minio        Minio
    Mongo        Mongo
    Notification Notification
    API          API
    CronTask     CronTask
    MsgGateway   MsgGateway
    MsgTransfer  MsgTransfer
    Push         Push
    Auth         Auth
    Conversation Conversation
    Friend       Friend
    Group        Group
    Msg          Msg
    Third        Third
    User         User
    Redis        Redis
    Share        Share
    Webhooks     Webhooks
}

func (a *AllConfig) Name2Config(name string) any {
    switch name {
    case a.Discovery.GetConfigFileName():
        return a.Discovery
    case a.Kafka.GetConfigFileName():
```

```
    return a.Kafka
  case a.LocalCache.GetConfigFileName():
    return a.LocalCache
  case a.Log.GetConfigFileName():
    return a.Log
  case a.Minio.GetConfigFileName():
    return a.Minio
  case a.Mongo.GetConfigFileName():
    return a.Mongo
  case a.Notification.GetConfigFileName():
    return a.Notification
  case a.API.GetConfigFileName():
    return a.API
  case a.CronTask.GetConfigFileName():
    return a.CronTask
  case a.MsgGateway.GetConfigFileName():
    return a.MsgGateway
  case a.MsgTransfer.GetConfigFileName():
    return a.MsgTransfer
  case a.Push.GetConfigFileName():
    return a.Push
  case a.Auth.GetConfigFileName():
    return a.Auth
  case a.Conversation.GetConfigFileName():
    return a.Conversation
  case a.Friend.GetConfigFileName():
    return a.Friend
  case a.Group.GetConfigFileName():
    return a.Group
  case a.Msg.GetConfigFileName():
    return a.Msg
  case a.Third.GetConfigFileName():
    return a.Third
  case a.User.GetConfigFileName():
    return a.User
  case a.Redis.GetConfigFileName():
    return a.Redis
  case a.Share.GetConfigFileName():
    return a.Share
  case a.Webhooks.GetConfigFileName():
    return a.Webhooks
  default:
    return nil
  }
}

func (a *AllConfig) GetConfigNames() []string {
  return []string{
    a.Discovery.GetConfigFileName(),
    a.Kafka.GetConfigFileName(),
    a.LocalCache.GetConfigFileName(),
    a.Log.GetConfigFileName(),
    a.Minio.GetConfigFileName(),
    a.Mongo.GetConfigFileName(),
    a.Notification.GetConfigFileName(),
    a.API.GetConfigFileName(),
    a.CronTask.GetConfigFileName(),
    a.MsgGateway.GetConfigFileName(),
    a.MsgTransfer.GetConfigFileName(),
    a.Push.GetConfigFileName(),
    a.Auth.GetConfigFileName(),
    a.Conversation.GetConfigFileName(),
    a.Friend.GetConfigFileName(),
    a.Group.GetConfigFileName(),
    a.Msg.GetConfigFileName(),
    a.Third.GetConfigFileName(),
```

```
    a.User.GetConfigFileName(),
    a.Redis.GetConfigFileName(),
    a.Share.GetConfigFileName(),
    a.Webhooks.GetConfigFileName(),
  }
}

const (
  FileName                         = "config.yaml"
  DiscoveryConfigFilename          = "discovery.yml"
  KafkaConfigFileName              = "kafka.yml"
  LocalCacheConfigFileName         = "local-cache.yml"
  LogConfigFileName                = "log.yml"
  MinioConfigFileName              = "minio.yml"
  MongodbConfigFileName            = "mongodb.yml"
  NotificationFileName             = "notification.yml"
  OpenIMAPICfgFileName             = "openim-api.yml"
  OpenIMCronTaskCfgFileName        = "openim-crontask.yml"
  OpenIMMsgGatewayCfgFileName      = "openim-msggateway.yml"
  OpenIMMsgTransferCfgFileName     = "openim-msgtransfer.yml"
  OpenIMPushCfgFileName            = "openim-push.yml"
  OpenIMRPCAuthCfgFileName         = "openim-rpc-auth.yml"
  OpenIMRPCConversationCfgFileName = "openim-rpc-conversation.yml"
  OpenIMRPCFriendCfgFileName       = "openim-rpc-friend.yml"
  OpenIMRPCGroupCfgFileName        = "openim-rpc-group.yml"
  OpenIMRPCMsgCfgFileName          = "openim-rpc-msg.yml"
  OpenIMRPCThirdCfgFileName        = "openim-rpc-third.yml"
  OpenIMRPCUserCfgFileName         = "openim-rpc-user.yml"
  RedisConfigFileName              = "redis.yml"
  ShareFileName                    = "share.yml"
  WebhooksConfigFileName           = "webhooks.yml"
)

func (d *Discovery) GetConfigFileName() string {
  return DiscoveryConfigFilename
}

func (k *Kafka) GetConfigFileName() string {
  return KafkaConfigFileName
}

func (lc *LocalCache) GetConfigFileName() string {
  return LocalCacheConfigFileName
}

func (l *Log) GetConfigFileName() string {
  return LogConfigFileName
}

func (m *Minio) GetConfigFileName() string {
  return MinioConfigFileName
}

func (m *Mongo) GetConfigFileName() string {
  return MongodbConfigFileName
}

func (n *Notification) GetConfigFileName() string {
  return NotificationFileName
}

func (a *API) GetConfigFileName() string {
  return OpenIMAPICfgFileName
}

func (ct *CronTask) GetConfigFileName() string {
```

```
	return OpenIMCronTaskCfgFileName
}

func (mg *MsgGateway) GetConfigFileName() string {
	return OpenIMMsgGatewayCfgFileName
}

func (mt *MsgTransfer) GetConfigFileName() string {
	return OpenIMMsgTransferCfgFileName
}

func (p *Push) GetConfigFileName() string {
	return OpenIMPushCfgFileName
}

func (a *Auth) GetConfigFileName() string {
	return OpenIMRPCAuthCfgFileName
}

func (c *Conversation) GetConfigFileName() string {
	return OpenIMRPCConversationCfgFileName
}

func (f *Friend) GetConfigFileName() string {
	return OpenIMRPCFriendCfgFileName
}

func (g *Group) GetConfigFileName() string {
	return OpenIMRPCGroupCfgFileName
}

func (m *Msg) GetConfigFileName() string {
	return OpenIMRPCMsgCfgFileName
}

func (t *Third) GetConfigFileName() string {
	return OpenIMRPCThirdCfgFileName
}

func (u *User) GetConfigFileName() string {
	return OpenIMRPCUserCfgFileName
}

func (r *Redis) GetConfigFileName() string {
	return RedisConfigFileName
}

func (s *Share) GetConfigFileName() string {
	return ShareFileName
}

func (w *Webhooks) GetConfigFileName() string {
	return WebhooksConfigFileName
}
```

pkg/common/config/constant.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package config

import "github.com/openimsdk/tools/utils/runtimeenv"

const ConfKey = "conf"

const (
	MountConfigFilePath = "CONFIG_PATH"
	DeploymentType      = "DEPLOYMENT_TYPE"
	KUBERNETES          = runtimeenv.Kubernetes
	ETCD                = "etcd"
	//Standalone          = "standalone"
)

const (
	// DefaultDirPerm is used for creating general directories, allowing the owner to read, write, and execute,
	// while the group and others can only read and execute.
	DefaultDirPerm = 0755

	// PrivateFilePerm is used for sensitive files, allowing only the owner to read and write.
	PrivateFilePerm = 0600

	// ExecFilePerm is used for executable files, allowing the owner to read, write, and execute,
	// while the group and others can only read.
	ExecFilePerm = 0754

	// SharedDirPerm is used for shared directories, allowing the owner and group to read, write, and execute,
	// with no permissions for others.
	SharedDirPerm = 0770

	// ReadOnlyDirPerm is used for read-only directories, allowing the owner, group, and others to only read.
	ReadOnlyDirPerm = 0555
)
```

pkg/common/config/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package config // import "github.com/openimsdk/open-im-server/v3/pkg/common/config"
```

pkg/common/config/env.go

```
package config

import "strings"

var EnvPrefixMap map[string]string

func init() {
	EnvPrefixMap = make(map[string]string)
	fileNames := []string{
		FileName, NotificationFileName, ShareFileName, WebhooksConfigFileName,
		KafkaConfigFileName, RedisConfigFileName,
		MongodbConfigFileName, MinioConfigFileName, LogConfigFileName,
		OpenIMAPICfgFileName, OpenIMCronTaskCfgFileName, OpenIMMsgGatewayCfgFileName,
		OpenIMMsgTransferCfgFileName, OpenIMPushCfgFileName, OpenIMRPCAuthCfgFileName,
		OpenIMRPCConversationCfgFileName, OpenIMRPCFriendCfgFileName, OpenIMRPCGroupCfgFileName,
		OpenIMRPCMsgCfgFileName, OpenIMRPCThirdCfgFileName, OpenIMRPCUserCfgFileName, DiscoveryConfigFilename,
	}

	for _, fileName := range fileNames {
		envKey := strings.TrimSuffix(strings.TrimSuffix(fileName, ".yml"), ".yaml")
		envKey = "IMENV_" + envKey
		envKey = strings.ToUpper(strings.ReplaceAll(envKey, "-", "_"))
		EnvPrefixMap[fileName] = envKey
	}
}

const (
	FlagConf          = "config_folder_path"
	FlagTransferIndex = "index"
)
```

pkg/common/config/global.go

```
package config

var standalone bool

func SetStandalone() {
	standalone = true
}

func Standalone() bool {
	return standalone
}
```

pkg/common/config/load_config.go

```
package config

import (
	"os"
	"path/filepath"
	"strings"

	"github.com/mitchellh/mapstructure"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/runtimeenv"
	"github.com/spf13/viper"
)

func Load(configDirectory string, configFileName string, envPrefix string, config any) error {
	if runtimeenv.RuntimeEnvironment() == KUBERNETES {
		mountPath := os.Getenv(MountConfigFilePath)
		if mountPath == "" {
			return errs.ErrArgs.WrapMsg(MountConfigFilePath + " env is empty")
		}

		return loadConfig(filepath.Join(mountPath, configFileName), envPrefix, config)
	}

	return loadConfig(filepath.Join(configDirectory, configFileName), envPrefix, config)
}

func loadConfig(path string, envPrefix string, config any) error {
	v := viper.New()
	v.SetConfigFile(path)
	v.SetEnvPrefix(envPrefix)
	v.AutomaticEnv()
	v.SetEnvKeyReplacer(strings.NewReplacer(".", "_"))

	if err := v.ReadInConfig(); err != nil {
		return errs.WrapMsg(err, "failed to read config file", "path", path, "envPrefix", envPrefix)
	}

	if err := v.Unmarshal(config, func(config *mapstructure.DecoderConfig) {
		config.TagName = StructTagName
	}); err != nil {
		return errs.WrapMsg(err, "failed to unmarshal config", "path", path, "envPrefix", envPrefix)
	}
	return nil
}
```

pkg/common/config/load_config_test.go

```go
package config

import (
	"os"
	"testing"

	"github.com/stretchr/testify/assert"
)

func TestLoadLogConfig(t *testing.T) {
	var log Log
	os.Setenv("IMENV_LOG_REMAINLOGLEVEL", "5")
	err := Load("../../../config/", "log.yml", "IMENV_LOG", &log)
	assert.Nil(t, err)
	t.Log(log.RemainLogLevel)
	// assert.Equal(t, "../../../../logs/", log.StorageLocation)
}

func TestLoadMongoConfig(t *testing.T) {
	var mongo Mongo
	// os.Setenv("DEPLOYMENT_TYPE", "kubernetes")
	os.Setenv("IMENV_MONGODB_PASSWORD", "openIM1231231")
	// os.Setenv("IMENV_MONGODB_URI", "openIM123")
	// os.Setenv("IMENV_MONGODB_USERNAME", "openIM123")
	err := Load("../../../config/", "mongodb.yml", "IMENV_MONGODB", &mongo)
	// err := LoadApiConfig("../../../config/mongodb.yml", "IMENV_MONGODB", &mongo)

	assert.Nil(t, err)
	t.Log(mongo.Password)
	// assert.Equal(t, "openIM123", mongo.Password)
	t.Log(os.Getenv("IMENV_MONGODB_PASSWORD"))
	t.Log(mongo)
	// //export IMENV_OPENIM_RPC_USER_RPC_LISTENIP="0.0.0.0"
	// assert.Equal(t, "0.0.0.0", user.RPC.ListenIP)
	// //export IMENV_OPENIM_RPC_USER_RPC_PORTS="10110,10111,10112"
	// assert.Equal(t, []int{10110, 10111, 10112}, user.RPC.Ports)
}

func TestLoadMinioConfig(t *testing.T) {
	var storageConfig Minio
	err := Load("../../../config/minio.yml", "IMENV_MINIO", "", &storageConfig)
	assert.Nil(t, err)
	assert.Equal(t, "openim", storageConfig.Bucket)
}

func TestLoadWebhooksConfig(t *testing.T) {
	var webhooks Webhooks
	err := Load("../../../config/webhooks.yml", "IMENV_WEBHOOKS", "", &webhooks)
	assert.Nil(t, err)
	assert.Equal(t, 5, webhooks.BeforeAddBlack.Timeout)

}

func TestLoadOpenIMRpcUserConfig(t *testing.T) {
	var user User
	err := Load("../../../config/openim-rpc-user.yml", "IMENV_OPENIM_RPC_USER", "", &user)
	assert.Nil(t, err)
	//export IMENV_OPENIM_RPC_USER_RPC_LISTENIP="0.0.0.0"
	assert.Equal(t, "0.0.0.0", user.RPC.ListenIP)
	//export IMENV_OPENIM_RPC_USER_RPC_PORTS="10110,10111,10112"
	assert.Equal(t, []int{10110, 10111, 10112}, user.RPC.Ports)
}

func TestLoadNotificationConfig(t *testing.T) {
```

```
	var noti Notification
	err := Load("../../../config/notification.yml", "IMENV_NOTIFICATION", "", &noti)
	assert.Nil(t, err)
	assert.Equal(t, "Your friend's profile has been changed", noti.FriendRemarkSet.OfflinePush.Title)
}

func TestLoadOpenIMThirdConfig(t *testing.T) {
	var third Third
	err := Load("../../../config/openim-rpc-third.yml", "IMENV_OPENIM_RPC_THIRD", "", &third)
	assert.Nil(t, err)
	assert.Equal(t, "enabled", third.Object.Enable)
	assert.Equal(t, "https://oss-cn-chengdu.aliyuncs.com", third.Object.Oss.Endpoint)
	assert.Equal(t, "my_bucket_name", third.Object.Oss.Bucket)
	assert.Equal(t, "https://my_bucket_name.oss-cn-chengdu.aliyuncs.com", third.Object.Oss.BucketURL)
	assert.Equal(t, "AKID1234567890", third.Object.Oss.AccessKeyID)
	assert.Equal(t, "abc123xyz789", third.Object.Oss.AccessKeySecret)
	assert.Equal(t, "session_token_value", third.Object.Oss.SessionToken) // Uncomment if session token is needed
	assert.Equal(t, true, third.Object.Oss.PublicRead)

	// Environment: IMENV_OPENIM_RPC_THIRD_OBJECT_ENABLE=enabled;IMENV_OPENIM_RPC_THIRD_OBJECT_OSS_ENDPOINT=https://oss
}

func TestTransferConfig(t *testing.T) {
	var tran MsgTransfer
	err := Load("../../../config/openim-msgtransfer.yml", "IMENV_OPENIM-MSGTRANSFER", "", &tran)
	assert.Nil(t, err)
	assert.Equal(t, true, tran.Prometheus.Enable)
	assert.Equal(t, true, tran.Prometheus.AutoSetPorts)
}
```

pkg/common/config/parse.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package config

import (
    "os"
    "path/filepath"

    "gopkg.in/yaml.v3"

    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/field"
)

const (
    DefaultFolderPath = "../config/"
)

// return absolude path join ../config/, this is k8s container config path.
func GetDefaultConfigPath() (string, error) {
    executablePath, err := os.Executable()
    if err != nil {
        return "", errs.WrapMsg(err, "failed to get executable path")
    }

    configPath, err := field.OutDir(filepath.Join(filepath.Dir(executablePath), "../config/"))
    if err != nil {
        return "", errs.WrapMsg(err, "failed to get output directory", "outDir", filepath.Join(filepath.Dir(executablePath
    }
    return configPath, nil
}

// getProjectRoot returns the absolute path of the project root directory.
func GetProjectRoot() (string, error) {
    executablePath, err := os.Executable()
    if err != nil {
        return "", errs.Wrap(err)
    }
    projectRoot, err := field.OutDir(filepath.Join(filepath.Dir(executablePath), "../../../../.."))
    if err != nil {
        return "", errs.Wrap(err)
    }
    return projectRoot, nil
}

func GetOptionsByNotification(cfg NotificationConfig, sendMessage *bool) msgprocessor.Options {
    opts := msgprocessor.NewOptions()

    if sendMessage != nil {
        cfg.IsSendMsg = *sendMessage
```

```
	}
	if cfg.IsSendMsg {
		opts = msgprocessor.WithOptions(opts, msgprocessor.WithUnreadCount(true))
	}
	if cfg.OfflinePush.Enable {
		opts = msgprocessor.WithOptions(opts, msgprocessor.WithOfflinePush(true))
	}
	switch cfg.ReliabilityLevel {
	case constant.UnreliableNotification:
	case constant.ReliableNotificationNoMsg:
		opts = msgprocessor.WithOptions(opts, msgprocessor.WithHistory(true), msgprocessor.WithPersistent())
	}
	opts = msgprocessor.WithOptions(opts, msgprocessor.WithSendMsg(cfg.IsSendMsg))

	return opts
}

// initConfig loads configuration from a specified path into the provided config structure.
// If the specified config file does not exist, it attempts to load from the project's default "config" directory.
// It logs informative messages regarding the configuration path being used.
func initConfig(config any, configName, configFolderPath string) error {
	configFolderPath = filepath.Join(configFolderPath, configName)
	_, err := os.Stat(configFolderPath)
	if err != nil {
		if !os.IsNotExist(err) {
			return errs.WrapMsg(err, "stat config path error", "config Folder Path", configFolderPath)
		}
		path, err := GetProjectRoot()
		if err != nil {
			return err
		}
		configFolderPath = filepath.Join(path, "config", configName)
	}
	data, err := os.ReadFile(configFolderPath)
	if err != nil {
		return errs.WrapMsg(err, "read file error", "config Folder Path", configFolderPath)
	}
	if err = yaml.Unmarshal(data, config); err != nil {
		return errs.WrapMsg(err, "unmarshal yaml error", "config Folder Path", configFolderPath)
	}

	return nil
}
```

pkg/util

pkg/util/hashutil

pkg/util/hashutil/id.go

```
package hashutil

import (
	"crypto/md5"
	"encoding/binary"
	"encoding/json"
)

func IdHash(ids []string) uint64 {
	if len(ids) == 0 {
		return 0
	}
	data, _ := json.Marshal(ids)
	sum := md5.Sum(data)
	return binary.BigEndian.Uint64(sum[:])
}
```

pkg/util/conversationutil

pkg/util/conversationutil/conversationutil.go

```go
package conversationutil

import (
	"sort"
	"strings"
)

func GenConversationIDForSingle(sendID, recvID string) string {
	l := []string{sendID, recvID}
	sort.Strings(l)
	return "si_" + strings.Join(l, "_")
}

func GenConversationUniqueKeyForGroup(groupID string) string {
	return groupID
}

func GenGroupConversationID(groupID string) string {
	return "sg_" + groupID
}

func IsGroupConversationID(conversationID string) bool {
	return strings.HasPrefix(conversationID, "sg_")
}

func IsNotificationConversationID(conversationID string) bool {
	return strings.HasPrefix(conversationID, "n_")
}

func GenConversationUniqueKeyForSingle(sendID, recvID string) string {
	l := []string{sendID, recvID}
	sort.Strings(l)
	return strings.Join(l, "_")
}

func GetNotificationConversationIDByConversationID(conversationID string) string {
	l := strings.Split(conversationID, "_")
	if len(l) > 1 {
		l[0] = "n"
		return strings.Join(l, "_")
	}
	return ""
}

func GetSelfNotificationConversationID(userID string) string {
	return "n_" + userID + "_" + userID
}

func GetSeqsBeginEnd(seqs []int64) (int64, int64) {
	if len(seqs) == 0 {
		return 0, 0
	}
	return seqs[0], seqs[len(seqs)-1]
}
```

pkg/util/conversationutil/doc.go

```
package conversationutil // import "github.com/openimsdk/open-im-server/v3/pkg/util/conversationutil"
```

pkg/util/useronline

pkg/util/useronline/split.go

```
package useronline

import (
	"errors"
	"strconv"
	"strings"
)

func ParseUserOnlineStatus(payload string) (string, []int32, error) {
	arr := strings.Split(payload, ":")
	if len(arr) == 0 {
		return "", nil, errors.New("invalid data")
	}
	userID := arr[len(arr)-1]
	if userID == "" {
		return "", nil, errors.New("userID is empty")
	}
	platformIDs := make([]int32, len(arr)-1)
	for i := range platformIDs {
		platformID, err := strconv.Atoi(arr[i])
		if err != nil {
			return "", nil, err
		}
		platformIDs[i] = int32(platformID)
	}
	return userID, platformIDs, nil
}
```

pkg/tools

pkg/tools/batcher

pkg/tools/batcher/batcher.go

```go
package batcher

import (
	"context"
	"fmt"
	"strings"
	"sync"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/idutil"
)

var (
	DefaultDataChanSize = 1000
	DefaultSize         = 100
	DefaultBuffer       = 100
	DefaultWorker       = 5
	DefaultInterval     = time.Second
)

type Config struct {
	size       int           // Number of message aggregations
	buffer     int           // The number of caches running in a single coroutine
	dataBuffer int           // The size of the main data channel
	worker     int           // Number of coroutines processed in parallel
	interval   time.Duration // Time of message aggregations
	syncWait   bool          // Whether to wait synchronously after distributing messages have been consumed
}

type Option func(c *Config)

func WithSize(s int) Option {
	return func(c *Config) {
		c.size = s
	}
}

func WithBuffer(b int) Option {
	return func(c *Config) {
		c.buffer = b
	}
}

func WithWorker(w int) Option {
	return func(c *Config) {
		c.worker = w
	}
}

func WithInterval(i time.Duration) Option {
	return func(c *Config) {
		c.interval = i
	}
}

func WithSyncWait(wait bool) Option {
```

```
	return func(c *Config) {
		c.syncWait = wait
	}
}

func WithDataBuffer(size int) Option {
	return func(c *Config) {
		c.dataBuffer = size
	}
}

type Batcher[T any] struct {
	config *Config

	globalCtx  context.Context
	cancel     context.CancelFunc
	Do         func(ctx context.Context, channelID int, val *Msg[T])
	OnComplete func(lastMessage *T, totalCount int)
	Sharding   func(key string) int
	Key        func(data *T) string
	HookFunc   func(triggerID string, messages map[string][]*T, totalCount int, lastMessage *T)
	data       chan *T
	chArrays   []chan *Msg[T]
	wait       sync.WaitGroup
	counter    sync.WaitGroup
}

func emptyOnComplete[T any](*T, int) {}
func emptyHookFunc[T any](string, map[string][]*T, int, *T) {
}

func New[T any](opts ...Option) *Batcher[T] {
	b := &Batcher[T]{
		OnComplete: emptyOnComplete[T],
		HookFunc:   emptyHookFunc[T],
	}
	config := &Config{
		size:     DefaultSize,
		buffer:   DefaultBuffer,
		worker:   DefaultWorker,
		interval: DefaultInterval,
	}
	for _, opt := range opts {
		opt(config)
	}
	b.config = config
	b.data = make(chan *T, DefaultDataChanSize)
	b.globalCtx, b.cancel = context.WithCancel(context.Background())

	b.chArrays = make([]chan *Msg[T], b.config.worker)
	for i := 0; i < b.config.worker; i++ {
		b.chArrays[i] = make(chan *Msg[T], b.config.buffer)
	}
	return b
}

func (b *Batcher[T]) Worker() int {
	return b.config.worker
}

func (b *Batcher[T]) Start() error {
	if b.Sharding == nil {
		return errs.New("Sharding function is required").Wrap()
	}
	if b.Do == nil {
		return errs.New("Do function is required").Wrap()
```

```
	}
	if b.Key == nil {
		return errs.New("Key function is required").Wrap()
	}
	b.wait.Add(b.config.worker)
	for i := 0; i < b.config.worker; i++ {
		go b.run(i, b.chArrays[i])
	}
	b.wait.Add(1)
	go b.scheduler()
	return nil
}

func (b *Batcher[T]) Put(ctx context.Context, data *T) error {
	if data == nil {
		return errs.New("data can not be nil").Wrap()
	}
	select {
	case <-b.globalCtx.Done():
		return errs.New("data channel is closed").Wrap()
	case <-ctx.Done():
		return ctx.Err()
	case b.data <- data:
		return nil
	}
}

func (b *Batcher[T]) scheduler() {
	ticker := time.NewTicker(b.config.interval)
	defer func() {
		ticker.Stop()
		for _, ch := range b.chArrays {
			close(ch)
		}
		close(b.data)
		b.wait.Done()
	}()

	vals := make(map[string][]*T)
	count := 0
	var lastAny *T

	for {
		select {
		case data, ok := <-b.data:
			if !ok {
				// If the data channel is closed unexpectedly
				return
			}
			if data == nil {
				if count > 0 {
					b.distributeMessage(vals, count, lastAny)
				}
				return
			}

			key := b.Key(data)
			vals[key] = append(vals[key], data)
			lastAny = data

			count++
			if count >= b.config.size {

				b.distributeMessage(vals, count, lastAny)
				vals = make(map[string][]*T)
				count = 0
```

```
    }

    case <-ticker.C:
        if count > 0 {

            b.distributeMessage(vals, count, lastAny)
            vals = make(map[string][]*T)
            count = 0
        }
    }
  }
}

type Msg[T any] struct {
  key       string
  triggerID string
  val       []*T
}

func (m Msg[T]) Key() string {
  return m.key
}

func (m Msg[T]) TriggerID() string {
  return m.triggerID
}

func (m Msg[T]) Val() []*T {
  return m.val
}

func (m Msg[T]) String() string {
  var sb strings.Builder
  sb.WriteString("Key: ")
  sb.WriteString(m.key)
  sb.WriteString(", Values: [")
  for i, v := range m.val {
    if i > 0 {
      sb.WriteString(", ")
    }
    sb.WriteString(fmt.Sprintf("%v", *v))
  }
  sb.WriteString("]")
  return sb.String()
}

func (b *Batcher[T]) distributeMessage(messages map[string][]*T, totalCount int, lastMessage *T) {
  triggerID := idutil.OperationIDGenerator()
  b.HookFunc(triggerID, messages, totalCount, lastMessage)
  for key, data := range messages {
    if b.config.syncWait {
      b.counter.Add(1)
    }
    channelID := b.Sharding(key)
    b.chArrays[channelID] <- &Msg[T]{key: key, triggerID: triggerID, val: data}
  }
  if b.config.syncWait {
    b.counter.Wait()
  }
  if b.OnComplete != nil {
    b.OnComplete(lastMessage, totalCount)
  }
}

func (b *Batcher[T]) run(channelID int, ch <-chan *Msg[T]) {
  defer b.wait.Done()
```

```
	ctx := authverify.WithTempAdmin(context.Background())
	for {
		select {
		case messages, ok := <-ch:
			if !ok {
				return
			}
			b.Do(ctx, channelID, messages)
			if b.config.syncWait {
				b.counter.Done()
			}
		}
	}
}

func (b *Batcher[T]) Close() {
	b.cancel() // Signal to stop put data
	b.data <- nil
	//wait all goroutines exit
	b.wait.Wait()
}
```

pkg/tools/batcher/batcher_test.go

```go
package batcher

import (
	"context"
	"fmt"
	"github.com/openimsdk/tools/utils/stringutil"
	"testing"
	"time"
)

func TestBatcher(t *testing.T) {
	config := Config{
		size:     1000,
		buffer:   10,
		worker:   10,
		interval: 5 * time.Millisecond,
	}

	b := New[string](
		WithSize(config.size),
		WithBuffer(config.buffer),
		WithWorker(config.worker),
		WithInterval(config.interval),
		WithSyncWait(true),
	)

	// Mock Do function to simply print values for demonstration
	b.Do = func(ctx context.Context, channelID int, vals *Msg[string]) {
		t.Logf("Channel %d Processed batch: %v", channelID, vals)
	}
	b.OnComplete = func(lastMessage *string, totalCount int) {
		t.Logf("Completed processing with last message: %v, total count: %d", *lastMessage, totalCount)
	}
	b.Sharding = func(key string) int {
		hashCode := stringutil.GetHashCode(key)
		return int(hashCode) % config.worker
	}
	b.Key = func(data *string) string {
		return *data
	}

	err := b.Start()
	if err != nil {
		t.Fatal(err)
	}

	// Test normal data processing
	for i := 0; i < 10000; i++ {
		data := "data" + fmt.Sprintf("%d", i)
		if err := b.Put(context.Background(), &data); err != nil {
			t.Fatal(err)
		}
	}

	time.Sleep(time.Duration(1) * time.Second)
	start := time.Now()
	// Wait for all processing to finish
	b.Close()

	elapsed := time.Since(start)
	t.Logf("Close took %s", elapsed)

	if len(b.data) != 0 {
		t.Error("Data channel should be empty after closing")
```

```
    }
}
```

pkg/authverify

pkg/authverify/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package authverify // import "github.com/openimsdk/open-im-server/v3/pkg/authverify"
```

pkg/authverify/token.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package authverify

import (
	"context"
	"fmt"

	"github.com/golang-jwt/jwt/v4"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
)

func Secret(secret string) jwt.Keyfunc {
	return func(token *jwt.Token) (any, error) {
		return []byte(secret), nil
	}
}

func CheckAdmin(ctx context.Context) error {
	if IsAdmin(ctx) {
		return nil
	}
	return servererrs.ErrNoPermission.WrapMsg(fmt.Sprintf("user %s is not admin userID", mcontext.GetOpUserID(ctx)))
}

//func IsManagerUserID(opUserID string, imAdminUserID []string) bool {
//	return datautil.Contain(opUserID, imAdminUserID...)
//}

func CheckUserIsAdmin(ctx context.Context, userID string) bool {
	return datautil.Contain(userID, GetIMAdminUserIDs(ctx)...)
}

func CheckSystemAccount(ctx context.Context, level int32) bool {
	return level >= constant.AppAdmin
}

const (
	CtxAdminUserIDsKey = "CtxAdminUserIDsKey"
)

func WithIMAdminUserIDs(ctx context.Context, imAdminUserID []string) context.Context {
	return context.WithValue(ctx, CtxAdminUserIDsKey, imAdminUserID)
}

func GetIMAdminUserIDs(ctx context.Context) []string {
	imAdminUserID, _ := ctx.Value(CtxAdminUserIDsKey).([]string)
	return imAdminUserID
}
```

```go
func IsAdmin(ctx context.Context) bool {
	return IsTempAdmin(ctx) || IsSystemAdmin(ctx)
}

func CheckAccess(ctx context.Context, ownerUserID string) error {
	if mcontext.GetOpUserID(ctx) == ownerUserID {
		return nil
	}
	if IsAdmin(ctx) {
		return nil
	}
	return servererrs.ErrNoPermission.WrapMsg("ownerUserID", ownerUserID)
}

func CheckAccessIn(ctx context.Context, ownerUserIDs ...string) error {
	opUserID := mcontext.GetOpUserID(ctx)
	for _, userID := range ownerUserIDs {
		if opUserID == userID {
			return nil
		}
	}
	if IsAdmin(ctx) {
		return nil
	}
	return servererrs.ErrNoPermission.WrapMsg("opUser in ownerUserIDs")
}

var tempAdminValue = []string{"1"}

const ctxTempAdminKey = "ctxImTempAdminKey"

func WithTempAdmin(ctx context.Context) context.Context {
	keys, _ := ctx.Value(constant.RpcCustomHeader).([]string)
	if datautil.Contain(ctxTempAdminKey, keys...) {
		return ctx
	}
	if len(keys) > 0 {
		temp := make([]string, 0, len(keys)+1)
		temp = append(temp, keys...)
		keys = append(temp, ctxTempAdminKey)
	} else {
		keys = []string{ctxTempAdminKey}
	}
	ctx = context.WithValue(ctx, constant.RpcCustomHeader, keys)
	return context.WithValue(ctx, ctxTempAdminKey, tempAdminValue)
}

func IsTempAdmin(ctx context.Context) bool {
	values, _ := ctx.Value(ctxTempAdminKey).([]string)
	return datautil.Equal(tempAdminValue, values)
}

func IsSystemAdmin(ctx context.Context) bool {
	return datautil.Contain(mcontext.GetOpUserID(ctx), GetIMAdminUserIDs(ctx)...)
}
```

pkg/rpcli

pkg/rpcli/auth.go

```
package rpcli

import (
	"context"
	"github.com/openimsdk/protocol/auth"
	"google.golang.org/grpc"
)

func NewAuthClient(cc grpc.ClientConnInterface) *AuthClient {
	return &AuthClient{auth.NewAuthClient(cc)}
}

type AuthClient struct {
	auth.AuthClient
}

func (x *AuthClient) KickTokens(ctx context.Context, tokens []string) error {
	if len(tokens) == 0 {
		return nil
	}
	return ignoreResp(x.AuthClient.KickTokens(ctx, &auth.KickTokensReq{Tokens: tokens}))
}

func (x *AuthClient) InvalidateToken(ctx context.Context, req *auth.InvalidateTokenReq) error {
	return ignoreResp(x.AuthClient.InvalidateToken(ctx, req))
}

func (x *AuthClient) ParseToken(ctx context.Context, token string) (*auth.ParseTokenResp, error) {
	return x.AuthClient.ParseToken(ctx, &auth.ParseTokenReq{Token: token})
}
```

pkg/rpcli/conversation.go

```
package rpcli

import (
	"context"
	"github.com/openimsdk/protocol/conversation"
	"google.golang.org/grpc"
)

func NewConversationClient(cc grpc.ClientConnInterface) *ConversationClient {
	return &ConversationClient{conversation.NewConversationClient(cc)}
}

type ConversationClient struct {
	conversation.ConversationClient
}

func (x *ConversationClient) SetConversationMaxSeq(ctx context.Context, conversationID string, ownerUserIDs []string
	if len(ownerUserIDs) == 0 {
		return nil
	}
	req := &conversation.SetConversationMaxSeqReq{ConversationID: conversationID, OwnerUserID: ownerUserIDs, MaxSeq: ma
	return ignoreResp(x.ConversationClient.SetConversationMaxSeq(ctx, req))
}

func (x *ConversationClient) SetConversations(ctx context.Context, ownerUserIDs []string, info *conversation.Convers
	if len(ownerUserIDs) == 0 {
		return nil
	}
	req := &conversation.SetConversationsReq{UserIDs: ownerUserIDs, Conversation: info}
	return ignoreResp(x.ConversationClient.SetConversations(ctx, req))
}

func (x *ConversationClient) GetConversationsByConversationIDs(ctx context.Context, conversationIDs []string) ([]*co
	if len(conversationIDs) == 0 {
		return nil, nil
	}
	req := &conversation.GetConversationsByConversationIDReq{ConversationIDs: conversationIDs}
	return extractField(ctx, x.ConversationClient.GetConversationsByConversationID, req, (*conversation.GetConversation
}

func (x *ConversationClient) GetConversationsByConversationID(ctx context.Context, conversationID string) (*conversa
	return firstValue(x.GetConversationsByConversationIDs(ctx, []string{conversationID}))
}

func (x *ConversationClient) SetConversationMinSeq(ctx context.Context, conversationID string, ownerUserIDs []string
	if len(ownerUserIDs) == 0 {
		return nil
	}
	req := &conversation.SetConversationMinSeqReq{ConversationID: conversationID, OwnerUserID: ownerUserIDs, MinSeq: mi
	return ignoreResp(x.ConversationClient.SetConversationMinSeq(ctx, req))
}

func (x *ConversationClient) GetConversation(ctx context.Context, conversationID string, ownerUserID string) (*conve
	req := &conversation.GetConversationReq{ConversationID: conversationID, OwnerUserID: ownerUserID}
	return extractField(ctx, x.ConversationClient.GetConversation, req, (*conversation.GetConversationResp).GetConversa
}

func (x *ConversationClient) GetConversations(ctx context.Context, conversationIDs []string, ownerUserID string) ([]
	if len(conversationIDs) == 0 {
		return nil, nil
	}
	req := &conversation.GetConversationsReq{ConversationIDs: conversationIDs, OwnerUserID: ownerUserID}
	return extractField(ctx, x.ConversationClient.GetConversations, req, (*conversation.GetConversationsResp).GetConver
}
```

```go
func (x *ConversationClient) GetConversationIDs(ctx context.Context, ownerUserID string) ([]string, error) {
	req := &conversation.GetConversationIDsReq{UserID: ownerUserID}
	return extractField(ctx, x.ConversationClient.GetConversationIDs, req, (*conversation.GetConversationIDsResp).GetCo
}

func (x *ConversationClient) GetPinnedConversationIDs(ctx context.Context, ownerUserID string) ([]string, error) {
	req := &conversation.GetPinnedConversationIDsReq{UserID: ownerUserID}
	return extractField(ctx, x.ConversationClient.GetPinnedConversationIDs, req, (*conversation.GetPinnedConversationID
}

func (x *ConversationClient) CreateGroupChatConversations(ctx context.Context, groupID string, userIDs []string) err
	if len(userIDs) == 0 {
		return nil
	}
	req := &conversation.CreateGroupChatConversationsReq{GroupID: groupID, UserIDs: userIDs}
	return ignoreResp(x.ConversationClient.CreateGroupChatConversations(ctx, req))
}

func (x *ConversationClient) CreateSingleChatConversations(ctx context.Context, req *conversation.CreateSingleChatCo
	return ignoreResp(x.ConversationClient.CreateSingleChatConversations(ctx, req))
}

func (x *ConversationClient) GetConversationOfflinePushUserIDs(ctx context.Context, conversationID string, userIDs [
	if len(userIDs) == 0 {
		return nil, nil
	}
	req := &conversation.GetConversationOfflinePushUserIDsReq{ConversationID: conversationID, UserIDs: userIDs}
	return extractField(ctx, x.ConversationClient.GetConversationOfflinePushUserIDs, req, (*conversation.GetConversatio
}
```

pkg/rpcli/group.go

```
package rpcli

import (
	"context"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/sdkws"
	"google.golang.org/grpc"
)

func NewGroupClient(cc grpc.ClientConnInterface) *GroupClient {
	return &GroupClient{group.NewGroupClient(cc)}
}

type GroupClient struct {
	group.GroupClient
}

func (x *GroupClient) GetGroupsInfo(ctx context.Context, groupIDs []string) ([]*sdkws.GroupInfo, error) {
	if len(groupIDs) == 0 {
		return nil, nil
	}
	req := &group.GetGroupsInfoReq{GroupIDs: groupIDs}
	return extractField(ctx, x.GroupClient.GetGroupsInfo, req, (*group.GetGroupsInfoResp).GetGroupInfos)
}

func (x *GroupClient) GetGroupInfo(ctx context.Context, groupID string) (*sdkws.GroupInfo, error) {
	return firstValue(x.GetGroupsInfo(ctx, []string{groupID}))
}

func (x *GroupClient) GetGroupInfoCache(ctx context.Context, groupID string) (*sdkws.GroupInfo, error) {
	req := &group.GetGroupInfoCacheReq{GroupID: groupID}
	return extractField(ctx, x.GroupClient.GetGroupInfoCache, req, (*group.GetGroupInfoCacheResp).GetGroupInfo)
}

func (x *GroupClient) GetGroupMemberCache(ctx context.Context, groupID string, userID string) (*sdkws.GroupMemberFul
	req := &group.GetGroupMemberCacheReq{GroupID: groupID, GroupMemberID: userID}
	return extractField(ctx, x.GroupClient.GetGroupMemberCache, req, (*group.GetGroupMemberCacheResp).GetMember)
}

func (x *GroupClient) DismissGroup(ctx context.Context, groupID string, deleteMember bool) error {
	req := &group.DismissGroupReq{GroupID: groupID, DeleteMember: deleteMember}
	return ignoreResp(x.GroupClient.DismissGroup(ctx, req))
}

func (x *GroupClient) GetGroupMemberUserIDs(ctx context.Context, groupID string) ([]string, error) {
	req := &group.GetGroupMemberUserIDsReq{GroupID: groupID}
	return extractField(ctx, x.GroupClient.GetGroupMemberUserIDs, req, (*group.GetGroupMemberUserIDsResp).GetUserIDs)
}

func (x *GroupClient) GetGroupMembersInfo(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupMemb
	if len(userIDs) == 0 {
		return nil, nil
	}
	req := &group.GetGroupMembersInfoReq{GroupID: groupID, UserIDs: userIDs}
	return extractField(ctx, x.GroupClient.GetGroupMembersInfo, req, (*group.GetGroupMembersInfoResp).GetMembers)
}

func (x *GroupClient) GetGroupMemberInfo(ctx context.Context, groupID string, userID string) (*sdkws.GroupMemberFull
	return firstValue(x.GetGroupMembersInfo(ctx, groupID, []string{userID}))
}

func (x *GroupClient) GetGroupMemberMapInfo(ctx context.Context, groupID string, userIDs []string) (map[string]*sdkw
	members, err := x.GetGroupMembersInfo(ctx, groupID, userIDs)
	if err != nil {
```

```
		return nil, err
	}
	memberMap := make(map[string]*sdkws.GroupMemberFullInfo)
	for _, member := range members {
		memberMap[member.UserID] = member
	}
	return memberMap, nil
}
```

pkg/rpcli/msg.go

```
package rpcli

import (
	"context"

	"google.golang.org/grpc"

	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
)

func NewMsgClient(cc grpc.ClientConnInterface) *MsgClient {
	return &MsgClient{msg.NewMsgClient(cc)}
}

type MsgClient struct {
	msg.MsgClient
}

func (x *MsgClient) GetMaxSeqs(ctx context.Context, conversationIDs []string) (map[string]int64, error) {
	if len(conversationIDs) == 0 {
		return nil, nil
	}
	req := &msg.GetMaxSeqsReq{ConversationIDs: conversationIDs}
	return extractField(ctx, x.MsgClient.GetMaxSeqs, req, (*msg.SeqsInfoResp).GetMaxSeqs)
}

func (x *MsgClient) GetMsgByConversationIDs(ctx context.Context, conversationIDs []string, maxSeqs map[string]int64)
	if len(conversationIDs) == 0 || len(maxSeqs) == 0 {
		return nil, nil
	}
	req := &msg.GetMsgByConversationIDsReq{ConversationIDs: conversationIDs, MaxSeqs: maxSeqs}
	return extractField(ctx, x.MsgClient.GetMsgByConversationIDs, req, (*msg.GetMsgByConversationIDsResp).GetMsgDatas)
}

func (x *MsgClient) GetHasReadSeqs(ctx context.Context, conversationIDs []string, userID string) (map[string]int64,
	if len(conversationIDs) == 0 {
		return nil, nil
	}
	req := &msg.GetHasReadSeqsReq{ConversationIDs: conversationIDs, UserID: userID}
	return extractField(ctx, x.MsgClient.GetHasReadSeqs, req, (*msg.SeqsInfoResp).GetMaxSeqs)
}

func (x *MsgClient) SetUserConversationMaxSeq(ctx context.Context, conversationID string, ownerUserIDs []string, max
	if len(ownerUserIDs) == 0 {
		return nil
	}
	req := &msg.SetUserConversationMaxSeqReq{ConversationID: conversationID, OwnerUserID: ownerUserIDs, MaxSeq: maxSeq}
	return ignoreResp(x.MsgClient.SetUserConversationMaxSeq(ctx, req))
}

func (x *MsgClient) SetUserConversationMin(ctx context.Context, conversationID string, ownerUserIDs []string, minSeq
	if len(ownerUserIDs) == 0 {
		return nil
	}
	req := &msg.SetUserConversationsMinSeqReq{ConversationID: conversationID, UserIDs: ownerUserIDs, Seq: minSeq}
	return ignoreResp(x.MsgClient.SetUserConversationsMinSeq(ctx, req))
}

func (x *MsgClient) GetLastMessageSeqByTime(ctx context.Context, conversationID string, lastTime int64) (int64, erro
	req := &msg.GetLastMessageSeqByTimeReq{ConversationID: conversationID, Time: lastTime}
	return extractField(ctx, x.MsgClient.GetLastMessageSeqByTime, req, (*msg.GetLastMessageSeqByTimeResp).GetSeq)
}
```

```go
func (x *MsgClient) GetConversationMaxSeq(ctx context.Context, conversationID string) (int64, error) {
	req := &msg.GetConversationMaxSeqReq{ConversationID: conversationID}
	return extractField(ctx, x.MsgClient.GetConversationMaxSeq, req, (*msg.GetConversationMaxSeqResp).GetMaxSeq)
}

func (x *MsgClient) GetActiveConversation(ctx context.Context, conversationIDs []string) ([]*msg.ActiveConversation,
	if len(conversationIDs) == 0 {
		return nil, nil
	}
	req := &msg.GetActiveConversationReq{ConversationIDs: conversationIDs}
	return extractField(ctx, x.MsgClient.GetActiveConversation, req, (*msg.GetActiveConversationResp).GetConversations)
}

func (x *MsgClient) GetSeqMessage(ctx context.Context, userID string, conversations []*msg.ConversationSeqs) (map[st
	if len(conversations) == 0 {
		return nil, nil
	}
	req := &msg.GetSeqMessageReq{UserID: userID, Conversations: conversations}
	return extractField(ctx, x.MsgClient.GetSeqMessage, req, (*msg.GetSeqMessageResp).GetMsgs)
}

func (x *MsgClient) SetUserConversationsMinSeq(ctx context.Context, conversationID string, userIDs []string, seq int
	if len(userIDs) == 0 {
		return nil
	}
	req := &msg.SetUserConversationsMinSeqReq{ConversationID: conversationID, UserIDs: userIDs, Seq: seq}
	return ignoreResp(x.MsgClient.SetUserConversationsMinSeq(ctx, req))
}
```

pkg/rpcli/msggateway.go

```
package rpcli

import (
	"github.com/openimsdk/protocol/msggateway"
	"google.golang.org/grpc"
)

func NewMsgGatewayClient(cc grpc.ClientConnInterface) *MsgGatewayClient {
	return &MsgGatewayClient{msggateway.NewMsgGatewayClient(cc)}
}

type MsgGatewayClient struct {
	msggateway.MsgGatewayClient
}
```

pkg/rpcli/push.go

```
package rpcli

import (
	"github.com/openimsdk/protocol/push"
	"google.golang.org/grpc"
)

func NewPushMsgServiceClient(cc grpc.ClientConnInterface) *PushMsgServiceClient {
	return &PushMsgServiceClient{push.NewPushMsgServiceClient(cc)}
}

type PushMsgServiceClient struct {
	push.PushMsgServiceClient
}
```

pkg/rpcli/relation.go

```
package rpcli

import (
	"context"
	"github.com/openimsdk/protocol/relation"
	"google.golang.org/grpc"
)

func NewRelationClient(cc grpc.ClientConnInterface) *RelationClient {
	return &RelationClient{relation.NewFriendClient(cc)}
}

type RelationClient struct {
	relation.FriendClient
}

func (x *RelationClient) GetFriendsInfo(ctx context.Context, ownerUserID string, friendUserIDs []string) ([]*relatio
	if len(friendUserIDs) == 0 {
		return nil, nil
	}
	req := &relation.GetFriendInfoReq{OwnerUserID: ownerUserID, FriendUserIDs: friendUserIDs}
	return extractField(ctx, x.FriendClient.GetFriendInfo, req, (*relation.GetFriendInfoResp).GetFriendInfos)
}
```

pkg/rpcli/rtc.go

```
package rpcli

import (
	"github.com/openimsdk/protocol/rtc"
	"google.golang.org/grpc"
)

func NewRtcServiceClient(cc grpc.ClientConnInterface) *RtcServiceClient {
	return &RtcServiceClient{rtc.NewRtcServiceClient(cc)}
}

type RtcServiceClient struct {
	rtc.RtcServiceClient
}
```

pkg/rpcli/third.go

```
package rpcli

import (
	"github.com/openimsdk/protocol/third"
	"google.golang.org/grpc"
)

func NewThirdClient(cc grpc.ClientConnInterface) *ThirdClient {
	return &ThirdClient{third.NewThirdClient(cc)}
}

type ThirdClient struct {
	third.ThirdClient
}
```

pkg/rpcli/tool.go

```
package rpcli

import (
	"context"
	"github.com/openimsdk/tools/errs"
	"google.golang.org/grpc"
)

func extractField[A, B, C any](ctx context.Context, fn func(ctx context.Context, req *A, opts ...grpc.CallOption) (*
	resp, err := fn(ctx, req)
	if err != nil {
		var c C
		return c, err
	}
	return get(resp), nil
}

func firstValue[A any](val []A, err error) (A, error) {
	if err != nil {
		var a A
		return a, err
	}
	if len(val) == 0 {
		var a A
		return a, errs.ErrRecordNotFound.WrapMsg("record not found")
	}
	return val[0], nil
}

func ignoreResp(_ any, err error) error {
	return err
}
```

pkg/rpcli/user.go

```
package rpcli

import (
	"context"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"
	"google.golang.org/grpc"
)

func NewUserClient(cc grpc.ClientConnInterface) *UserClient {
	return &UserClient{user.NewUserClient(cc)}
}

type UserClient struct {
	user.UserClient
}

func (x *UserClient) GetUsersInfo(ctx context.Context, userIDs []string) ([]*sdkws.UserInfo, error) {
	if len(userIDs) == 0 {
		return nil, nil
	}
	req := &user.GetDesignateUsersReq{UserIDs: userIDs}
	return extractField(ctx, x.UserClient.GetDesignateUsers, req, (*user.GetDesignateUsersResp).GetUsersInfo)
}

func (x *UserClient) GetUserInfo(ctx context.Context, userID string) (*sdkws.UserInfo, error) {
	return firstValue(x.GetUsersInfo(ctx, []string{userID}))
}

func (x *UserClient) CheckUser(ctx context.Context, userIDs []string) error {
	if len(userIDs) == 0 {
		return nil
	}
	users, err := x.GetUsersInfo(ctx, userIDs)
	if err != nil {
		return err
	}
	if len(users) != len(userIDs) {
		return errs.ErrRecordNotFound.WrapMsg("user not found")
	}
	return nil
}

func (x *UserClient) GetUsersInfoMap(ctx context.Context, userIDs []string) (map[string]*sdkws.UserInfo, error) {
	users, err := x.GetUsersInfo(ctx, userIDs)
	if err != nil {
		return nil, err
	}
	return datautil.SliceToMap(users, func(e *sdkws.UserInfo) string {
		return e.UserID
	}), nil
}

func (x *UserClient) GetAllOnlineUsers(ctx context.Context, cursor uint64) (*user.GetAllOnlineUsersResp, error) {
	req := &user.GetAllOnlineUsersReq{Cursor: cursor}
	return x.UserClient.GetAllOnlineUsers(ctx, req)
}

func (x *UserClient) GetUsersOnlinePlatform(ctx context.Context, userIDs []string) ([]*user.OnlineStatus, error) {
	if len(userIDs) == 0 {
		return nil, nil
	}
```

```
	req := &user.GetUserStatusReq{UserIDs: userIDs}
	return extractField(ctx, x.UserClient.GetUserStatus, req, (*user.GetUserStatusResp).GetStatusList)

}

func (x *UserClient) GetUserOnlinePlatform(ctx context.Context, userID string) ([]int32, error) {
	status, err := x.GetUsersOnlinePlatform(ctx, []string{userID})
	if err != nil {
		return nil, err
	}
	if len(status) == 0 {
		return nil, nil
	}
	return status[0].PlatformIDs, nil
}

func (x *UserClient) SetUserOnlineStatus(ctx context.Context, req *user.SetUserOnlineStatusReq) error {
	if len(req.Status) == 0 {
		return nil
	}
	return ignoreResp(x.UserClient.SetUserOnlineStatus(ctx, req))
}

func (x *UserClient) GetNotificationByID(ctx context.Context, userID string) error {
	return ignoreResp(x.UserClient.GetNotificationAccount(ctx, &user.GetNotificationAccountReq{UserID: userID}))
}

func (x *UserClient) GetAllUserIDs(ctx context.Context, pageNumber, showNumber int32) ([]string, error) {
	req := &user.GetAllUserIDReq{Pagination: &sdkws.RequestPagination{PageNumber: pageNumber, ShowNumber: showNumber}}
	return extractField(ctx, x.UserClient.GetAllUserID, req, (*user.GetAllUserIDResp).GetUserIDs)
}
```

pkg/msgprocessor

pkg/msgprocessor/conversation.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msgprocessor

import (
	"sort"
	"strings"

	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"google.golang.org/protobuf/proto"
)

func IsGroupConversationID(conversationID string) bool {
	return strings.HasPrefix(conversationID, "g_") || strings.HasPrefix(conversationID, "sg_")
}

func GetNotificationConversationIDByMsg(msg *sdkws.MsgData) string {
	switch msg.SessionType {
	case constant.SingleChatType:
		l := []string{msg.SendID, msg.RecvID}
		sort.Strings(l)
		return "n_" + strings.Join(l, "_")
	case constant.WriteGroupChatType:
		return "n_" + msg.GroupID
	case constant.ReadGroupChatType:
		return "n_" + msg.GroupID
	case constant.NotificationChatType:
		l := []string{msg.SendID, msg.RecvID}
		sort.Strings(l)
		return "n_" + strings.Join(l, "_")
	}
	return ""
}

func GetChatConversationIDByMsg(msg *sdkws.MsgData) string {
	switch msg.SessionType {
	case constant.SingleChatType:
		l := []string{msg.SendID, msg.RecvID}
		sort.Strings(l)
		return "si_" + strings.Join(l, "_")
	case constant.WriteGroupChatType:
		return "g_" + msg.GroupID
	case constant.ReadGroupChatType:
		return "sg_" + msg.GroupID
	case constant.NotificationChatType:
		l := []string{msg.SendID, msg.RecvID}
		sort.Strings(l)
```

```
		return "sn_" + strings.Join(l, "_")
	}

	return ""
}

func GetConversationIDByMsg(msg *sdkws.MsgData) string {
	options := Options(msg.Options)
	switch msg.SessionType {
	case constant.SingleChatType:
		l := []string{msg.SendID, msg.RecvID}
		sort.Strings(l)
		if !options.IsNotNotification() {
			return "n_" + strings.Join(l, "_")
		}
		return "si_" + strings.Join(l, "_") // single chat
	case constant.WriteGroupChatType:
		if !options.IsNotNotification() {
			return "n_" + msg.GroupID // group chat
		}
		return "g_" + msg.GroupID // group chat
	case constant.ReadGroupChatType:
		if !options.IsNotNotification() {
			return "n_" + msg.GroupID // super group chat
		}
		return "sg_" + msg.GroupID // super group chat
	case constant.NotificationChatType:
		l := []string{msg.SendID, msg.RecvID}
		sort.Strings(l)
		if !options.IsNotNotification() {
			return "n_" + strings.Join(l, "_")
		}
		return "sn_" + strings.Join(l, "_")
	}
	return ""
}

func GetConversationIDBySessionType(sessionType int, ids ...string) string {
	sort.Strings(ids)
	if len(ids) > 2 || len(ids) < 1 {
		return ""
	}
	switch sessionType {
	case constant.SingleChatType:
		return "si_" + strings.Join(ids, "_") // single chat
	case constant.WriteGroupChatType:
		return "g_" + ids[0] // group chat
	case constant.ReadGroupChatType:
		return "sg_" + ids[0] // super group chat
	case constant.NotificationChatType:
		return "sn_" + strings.Join(ids, "_") // server notification chat
	}
	return ""
}

func IsNotification(conversationID string) bool {
	return strings.HasPrefix(conversationID, "n_")
}

func IsNotificationByMsg(msg *sdkws.MsgData) bool {
	return !Options(msg.Options).IsNotNotification()
}

type MsgBySeq []*sdkws.MsgData

func (s MsgBySeq) Len() int {
```

```
    return len(s)
}

func (s MsgBySeq) Less(i, j int) bool {
    return s[i].Seq < s[j].Seq
}

func (s MsgBySeq) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}

func Pb2String(pb proto.Message) (string, error) {
    s, err := proto.Marshal(pb)
    if err != nil {
        return "", errs.Wrap(err)
    }
    return string(s), nil
}

func String2Pb(s string, pb proto.Message) error {
    return proto.Unmarshal([]byte(s), pb)
}
```

pkg/msgprocessor/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msgprocessor // import "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
```

pkg/msgprocessor/options.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msgprocessor

import "github.com/openimsdk/protocol/constant"

type (
    Options    map[string]bool
    OptionsOpt func(Options)
)

func NewOptions(opts ...OptionsOpt) Options {
    options := make(map[string]bool, 11)
    options[constant.IsNotNotification] = false
    options[constant.IsSendMsg] = false
    options[constant.IsHistory] = false
    options[constant.IsPersistent] = false
    options[constant.IsOfflinePush] = false
    options[constant.IsUnreadCount] = false
    options[constant.IsConversationUpdate] = false
    options[constant.IsSenderSync] = true
    options[constant.IsNotPrivate] = false
    options[constant.IsSenderConversationUpdate] = false
    options[constant.IsReactionFromCache] = false
    for _, opt := range opts {
        opt(options)
    }

    return options
}

func NewMsgOptions() Options {
    options := make(map[string]bool, 11)
    options[constant.IsOfflinePush] = false
    return make(map[string]bool)
}

func WithOptions(options Options, opts ...OptionsOpt) Options {
    for _, opt := range opts {
        opt(options)
    }
    return options
}

func WithNotNotification(b bool) OptionsOpt {
    return func(options Options) {
        options[constant.IsNotNotification] = b
    }
}

func WithSendMsg(b bool) OptionsOpt {
    return func(options Options) {
```

```
    options[constant.IsSendMsg] = b
  }
}

func WithHistory(b bool) OptionsOpt {
  return func(options Options) {
    options[constant.IsHistory] = b
  }
}

func WithPersistent() OptionsOpt {
  return func(options Options) {
    options[constant.IsPersistent] = true
  }
}

func WithOfflinePush(b bool) OptionsOpt {
  return func(options Options) {
    options[constant.IsOfflinePush] = b
  }
}

func WithUnreadCount(b bool) OptionsOpt {
  return func(options Options) {
    options[constant.IsUnreadCount] = b
  }
}

func WithConversationUpdate() OptionsOpt {
  return func(options Options) {
    options[constant.IsConversationUpdate] = true
  }
}

func WithSenderSync() OptionsOpt {
  return func(options Options) {
    options[constant.IsSenderSync] = true
  }
}

func WithNotPrivate() OptionsOpt {
  return func(options Options) {
    options[constant.IsNotPrivate] = true
  }
}

func WithSenderConversationUpdate() OptionsOpt {
  return func(options Options) {
    options[constant.IsSenderConversationUpdate] = true
  }
}

func WithReactionFromCache() OptionsOpt {
  return func(options Options) {
    options[constant.IsReactionFromCache] = true
  }
}

func (o Options) Is(notification string) bool {
  v, ok := o[notification]
  if !ok || v {
    return true
  }
  return false
}
```

```
func (o Options) IsNotNotification() bool {
	return o.Is(constant.IsNotNotification)
}

func (o Options) IsSendMsg() bool {
	return o.Is(constant.IsSendMsg)
}

func (o Options) IsHistory() bool {
	return o.Is(constant.IsHistory)
}

func (o Options) IsPersistent() bool {
	return o.Is(constant.IsPersistent)
}

func (o Options) IsOfflinePush() bool {
	return o.Is(constant.IsOfflinePush)
}

func (o Options) IsUnreadCount() bool {
	return o.Is(constant.IsUnreadCount)
}

func (o Options) IsConversationUpdate() bool {
	return o.Is(constant.IsConversationUpdate)
}

func (o Options) IsSenderSync() bool {
	return o.Is(constant.IsSenderSync)
}

func (o Options) IsNotPrivate() bool {
	return o.Is(constant.IsNotPrivate)
}

func (o Options) IsSenderConversationUpdate() bool {
	return o.Is(constant.IsSenderConversationUpdate)
}

func (o Options) IsReactionFromCache() bool {
	return o.Is(constant.IsReactionFromCache)
}
```

pkg/notification

pkg/notification/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package notification

import (
	"context"
	"encoding/json"
	"time"

	"google.golang.org/protobuf/proto"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mq/memamq"
	"github.com/openimsdk/tools/utils/idutil"
	"github.com/openimsdk/tools/utils/jsonutil"
	"github.com/openimsdk/tools/utils/timeutil"
)

func newContentTypeConf(conf *config.Notification) map[int32]config.NotificationConfig {
	return map[int32]config.NotificationConfig{
		// group
		constant.GroupCreatedNotification:                 conf.GroupCreated,
		constant.GroupInfoSetNotification:                 conf.GroupInfoSet,
		constant.JoinGroupApplicationNotification:         conf.JoinGroupApplication,
		constant.MemberQuitNotification:                   conf.MemberQuit,
		constant.GroupApplicationAcceptedNotification:     conf.GroupApplicationAccepted,
		constant.GroupApplicationRejectedNotification:     conf.GroupApplicationRejected,
		constant.GroupOwnerTransferredNotification:        conf.GroupOwnerTransferred,
		constant.MemberKickedNotification:                 conf.MemberKicked,
		constant.MemberInvitedNotification:                conf.MemberInvited,
		constant.MemberEnterNotification:                  conf.MemberEnter,
		constant.GroupDismissedNotification:               conf.GroupDismissed,
		constant.GroupMutedNotification:                   conf.GroupMuted,
		constant.GroupCancelMutedNotification:             conf.GroupCancelMuted,
		constant.GroupMemberMutedNotification:             conf.GroupMemberMuted,
		constant.GroupMemberCancelMutedNotification:       conf.GroupMemberCancelMuted,
		constant.GroupMemberInfoSetNotification:           conf.GroupMemberInfoSet,
		constant.GroupMemberSetToAdminNotification:        conf.GroupMemberSetToAdmin,
		constant.GroupMemberSetToOrdinaryUserNotification: conf.GroupMemberSetToOrdinary,
		constant.GroupInfoSetAnnouncementNotification:     conf.GroupInfoSetAnnouncement,
		constant.GroupInfoSetNameNotification:             conf.GroupInfoSetName,
		// user
		constant.UserInfoUpdatedNotification:  conf.UserInfoUpdated,
		constant.UserStatusChangeNotification: conf.UserStatusChanged,
		// friend
```

```
    constant.FriendApplicationNotification:         conf.FriendApplicationAdded,
    constant.FriendApplicationApprovedNotification: conf.FriendApplicationApproved,
    constant.FriendApplicationRejectedNotification: conf.FriendApplicationRejected,
    constant.FriendAddedNotification:               conf.FriendAdded,
    constant.FriendDeletedNotification:             conf.FriendDeleted,
    constant.FriendRemarkSetNotification:           conf.FriendRemarkSet,
    constant.BlackAddedNotification:                conf.BlackAdded,
    constant.BlackDeletedNotification:              conf.BlackDeleted,
    constant.FriendInfoUpdatedNotification:         conf.FriendInfoUpdated,
    constant.FriendsInfoUpdateNotification:         conf.FriendInfoUpdated, // use the same FriendInfoUpdated
    // conversation
    constant.ConversationChangeNotification:      conf.ConversationChanged,
    constant.ConversationUnreadNotification:      conf.ConversationChanged,
    constant.ConversationPrivateChatNotification: conf.ConversationSetPrivate,
    // msg
    constant.MsgRevokeNotification:  {IsSendMsg: false, ReliabilityLevel: constant.ReliableNotificationNoMsg},
    constant.HasReadReceipt:         {IsSendMsg: false, ReliabilityLevel: constant.ReliableNotificationNoMsg},
    constant.DeleteMsgsNotification: {IsSendMsg: false, ReliabilityLevel: constant.ReliableNotificationNoMsg},
  }
}

func newSessionTypeConf() map[int32]int32 {
  return map[int32]int32{
    // group
    constant.GroupCreatedNotification:                 constant.ReadGroupChatType,
    constant.GroupInfoSetNotification:                 constant.ReadGroupChatType,
    constant.JoinGroupApplicationNotification:         constant.SingleChatType,
    constant.MemberQuitNotification:                   constant.ReadGroupChatType,
    constant.GroupApplicationAcceptedNotification:     constant.SingleChatType,
    constant.GroupApplicationRejectedNotification:     constant.SingleChatType,
    constant.GroupOwnerTransferredNotification:        constant.ReadGroupChatType,
    constant.MemberKickedNotification:                 constant.ReadGroupChatType,
    constant.MemberInvitedNotification:                constant.ReadGroupChatType,
    constant.MemberEnterNotification:                  constant.ReadGroupChatType,
    constant.GroupDismissedNotification:               constant.ReadGroupChatType,
    constant.GroupMutedNotification:                   constant.ReadGroupChatType,
    constant.GroupCancelMutedNotification:             constant.ReadGroupChatType,
    constant.GroupMemberMutedNotification:             constant.ReadGroupChatType,
    constant.GroupMemberCancelMutedNotification:       constant.ReadGroupChatType,
    constant.GroupMemberInfoSetNotification:           constant.ReadGroupChatType,
    constant.GroupMemberSetToAdminNotification:        constant.ReadGroupChatType,
    constant.GroupMemberSetToOrdinaryUserNotification: constant.ReadGroupChatType,
    constant.GroupInfoSetAnnouncementNotification:     constant.ReadGroupChatType,
    constant.GroupInfoSetNameNotification:             constant.ReadGroupChatType,
    // user
    constant.UserInfoUpdatedNotification:  constant.SingleChatType,
    constant.UserStatusChangeNotification: constant.SingleChatType,
    // friend
    constant.FriendApplicationNotification:         constant.SingleChatType,
    constant.FriendApplicationApprovedNotification: constant.SingleChatType,
    constant.FriendApplicationRejectedNotification: constant.SingleChatType,
    constant.FriendAddedNotification:               constant.SingleChatType,
    constant.FriendDeletedNotification:             constant.SingleChatType,
    constant.FriendRemarkSetNotification:           constant.SingleChatType,
    constant.BlackAddedNotification:                constant.SingleChatType,
    constant.BlackDeletedNotification:              constant.SingleChatType,
    constant.FriendInfoUpdatedNotification:         constant.SingleChatType,
    constant.FriendsInfoUpdateNotification:         constant.SingleChatType,
    // conversation
    constant.ConversationChangeNotification:      constant.SingleChatType,
    constant.ConversationUnreadNotification:      constant.SingleChatType,
    constant.ConversationPrivateChatNotification: constant.SingleChatType,
    // delete
    constant.DeleteMsgsNotification: constant.SingleChatType,
  }
}
```

```
type NotificationSender struct {
	contentTypeConf map[int32]config.NotificationConfig
	sessionTypeConf map[int32]int32
	sendMsg         func(ctx context.Context, req *msg.SendMsgReq) (*msg.SendMsgResp, error)
	getUserInfo     func(ctx context.Context, userID string) (*sdkws.UserInfo, error)
	queue           *memamq.MemoryQueue
}

func WithQueue(queue *memamq.MemoryQueue) NotificationSenderOptions {
	return func(s *NotificationSender) {
		s.queue = queue
	}
}

type NotificationSenderOptions func(*NotificationSender)

func WithLocalSendMsg(sendMsg func(ctx context.Context, req *msg.SendMsgReq) (*msg.SendMsgResp, error)) Notification
	return func(s *NotificationSender) {
		s.sendMsg = sendMsg
	}
}

func WithRpcClient(sendMsg func(ctx context.Context, req *msg.SendMsgReq) (*msg.SendMsgResp, error)) NotificationSen
	return func(s *NotificationSender) {
		s.sendMsg = func(ctx context.Context, req *msg.SendMsgReq) (*msg.SendMsgResp, error) {
			return sendMsg(ctx, req)
		}
	}
}

func WithUserRpcClient(getUserInfo func(ctx context.Context, userID string) (*sdkws.UserInfo, error)) NotificationSe
	return func(s *NotificationSender) {
		s.getUserInfo = getUserInfo
	}
}

const (
	notificationWorkerCount = 16
	notificationBufferSize  = 1024 * 1024 * 2
)

func NewNotificationSender(conf *config.Notification, opts ...NotificationSenderOptions) *NotificationSender {
	notificationSender := &NotificationSender{contentTypeConf: newContentTypeConf(conf), sessionTypeConf: newSessionTyp
	for _, opt := range opts {
		opt(notificationSender)
	}
	if notificationSender.queue == nil {
		notificationSender.queue = memamq.NewMemoryQueue(notificationWorkerCount, notificationBufferSize)
	}
	return notificationSender
}

type notificationOpt struct {
	RpcGetUsername bool
	SendMessage    *bool
}

type NotificationOptions func(*notificationOpt)

func WithRpcGetUserName() NotificationOptions {
	return func(opt *notificationOpt) {
		opt.RpcGetUsername = true
	}
}
func WithSendMessage(sendMessage *bool) NotificationOptions {
```

```
    return func(opt *notificationOpt) {
        opt.SendMessage = sendMessage
    }
}

func (s *NotificationSender) send(ctx context.Context, sendID, recvID string, contentType, sessionType int32, m prot
    ctx = context.WithoutCancel(ctx)
    ctx, cancel := context.WithTimeout(ctx, time.Second*time.Duration(5))
    defer cancel()
    n := sdkws.NotificationElem{Detail: jsonutil.StructToJsonString(m)}
    content, err := json.Marshal(&n)
    if err != nil {
        log.ZWarn(ctx, "json.Marshal failed", err, "sendID", sendID, "recvID", recvID, "contentType", contentType, "msg",
        return
    }
    notificationOpt := &notificationOpt{}
    for _, opt := range opts {
        opt(notificationOpt)
    }
    var req msg.SendMsgReq
    var msg sdkws.MsgData
    var userInfo *sdkws.UserInfo
    if notificationOpt.RpcGetUsername && s.getUserInfo != nil {
        userInfo, err = s.getUserInfo(ctx, sendID)
        if err != nil {
            log.ZWarn(ctx, "getUserInfo failed", err, "sendID", sendID)
            return
        }
        msg.SenderNickname = userInfo.Nickname
        msg.SenderFaceURL = userInfo.FaceURL
    }
    var offlineInfo sdkws.OfflinePushInfo
    msg.SendID = sendID
    msg.RecvID = recvID
    msg.Content = content
    msg.MsgFrom = constant.SysMsgType
    msg.ContentType = contentType
    msg.SessionType = sessionType
    if msg.SessionType == constant.ReadGroupChatType {
        msg.GroupID = recvID
    }
    msg.CreateTime = timeutil.GetCurrentTimestampByMill()
    msg.ClientMsgID = idutil.GetMsgIDByMD5(sendID)
    optionsConfig := s.contentTypeConf[contentType]
    if sendID == recvID && contentType == constant.HasReadReceipt {
        optionsConfig.ReliabilityLevel = constant.UnreliableNotification
    }
    options := config.GetOptionsByNotification(optionsConfig, notificationOpt.SendMessage)
    s.SetOptionsByContentType(ctx, options, contentType)
    msg.Options = options
    // fill Notification OfflinePush by config
    offlineInfo.Title = optionsConfig.OfflinePush.Title
    offlineInfo.Desc = optionsConfig.OfflinePush.Desc
    offlineInfo.Ex = optionsConfig.OfflinePush.Ext
    msg.OfflinePushInfo = &offlineInfo
    req.MsgData = &msg
    _, err = s.sendMsg(ctx, &req)
    if err != nil {
        log.ZWarn(ctx, "SendMsg failed", err, "req", req.String())
    }
}

func (s *NotificationSender) NotificationWithSessionType(ctx context.Context, sendID, recvID string, contentType, se
    if err := s.queue.Push(func() { s.send(ctx, sendID, recvID, contentType, sessionType, m, opts...) }); err != nil {
        log.ZWarn(ctx, "Push to queue failed", err, "sendID", sendID, "recvID", recvID, "msg", jsonutil.StructToJsonStrin
    }
```

```
}

func (s *NotificationSender) Notification(ctx context.Context, sendID, recvID string, contentType int32, m proto.Mes
    s.NotificationWithSessionType(ctx, sendID, recvID, contentType, s.sessionTypeConf[contentType], m, opts...)
}

func (s *NotificationSender) SetOptionsByContentType(_ context.Context, options map[string]bool, contentType int32)
    switch contentType {
    case constant.UserStatusChangeNotification:
        options[constant.IsSenderSync] = false
    default:
    }
}
```

pkg/notification/common_user

pkg/notification/common_user/common.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package common_user

type CommonUser interface {
	GetNickname() string
	GetFaceURL() string
	GetUserID() string
	GetEx() string
}

type CommonGroup interface {
	GetNickname() string
	GetFaceURL() string
	GetGroupID() string
	GetEx() string
}
```

pkg/notification/grouphash

pkg/notification/grouphash/grouphash.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package grouphash

import (
	"context"
	"crypto/md5"
	"encoding/binary"
	"encoding/json"

	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/utils/datautil"
)

func NewGroupHashFromGroupClient(x group.GroupClient) *GroupHash {
	return &GroupHash{
		getGroupAllUserIDs: func(ctx context.Context, groupID string) ([]string, error) {
			resp, err := x.GetGroupMemberUserIDs(ctx, &group.GetGroupMemberUserIDsReq{GroupID: groupID})
			if err != nil {
				return nil, err
			}
			return resp.UserIDs, nil
		},
		getGroupMemberInfo: func(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupMemberFullInfo, er
			resp, err := x.GetGroupMembersInfo(ctx, &group.GetGroupMembersInfoReq{GroupID: groupID, UserIDs: userIDs})
			if err != nil {
				return nil, err
			}
			return resp.Members, nil
		},
	}
}

func NewGroupHashFromGroupServer(x group.GroupServer) *GroupHash {
	return &GroupHash{
		getGroupAllUserIDs: func(ctx context.Context, groupID string) ([]string, error) {
			resp, err := x.GetGroupMemberUserIDs(ctx, &group.GetGroupMemberUserIDsReq{GroupID: groupID})
			if err != nil {
				return nil, err
			}
			return resp.UserIDs, nil
		},
		getGroupMemberInfo: func(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupMemberFullInfo, er
			resp, err := x.GetGroupMembersInfo(ctx, &group.GetGroupMembersInfoReq{GroupID: groupID, UserIDs: userIDs})
			if err != nil {
				return nil, err
			}
			return resp.Members, nil
```

```
    },
  }
}

type GroupHash struct {
  getGroupAllUserIDs func(ctx context.Context, groupID string) ([]string, error)
  getGroupMemberInfo func(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupMemberFullInfo, error
}

func (gh *GroupHash) GetGroupHash(ctx context.Context, groupID string) (uint64, error) {
  userIDs, err := gh.getGroupAllUserIDs(ctx, groupID)
  if err != nil {
    return 0, err
  }
  var members []*sdkws.GroupMemberFullInfo
  if len(userIDs) > 0 {
    members, err = gh.getGroupMemberInfo(ctx, groupID, userIDs)
    if err != nil {
      return 0, err
    }
    datautil.Sort(userIDs, true)
  }
  memberMap := datautil.SliceToMap(members, func(e *sdkws.GroupMemberFullInfo) string {
    return e.UserID
  })
  res := make([]*sdkws.GroupMemberFullInfo, 0, len(members))
  for _, userID := range userIDs {
    member, ok := memberMap[userID]
    if !ok {
      continue
    }
    member.AppMangerLevel = 0
    res = append(res, member)
  }
  data, err := json.Marshal(res)
  if err != nil {
    return 0, err
  }
  sum := md5.Sum(data)
  return binary.BigEndian.Uint64(sum[:]), nil
}
```

pkg/localcache

pkg/localcache/cache.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package localcache

import (
	"context"
	"hash/fnv"
	"unsafe"

	"github.com/openimsdk/open-im-server/v3/pkg/localcache/link"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache/lru"
)

type Cache[V any] interface {
	Get(ctx context.Context, key string, fetch func(ctx context.Context) (V, error)) (V, error)
	GetLink(ctx context.Context, key string, fetch func(ctx context.Context) (V, error), link ...string) (V, error)
	Del(ctx context.Context, key ...string)
	DelLocal(ctx context.Context, key ...string)
	Stop()
}

func LRUStringHash(key string) uint64 {
	h := fnv.New64a()
	h.Write(*(*[]byte)(unsafe.Pointer(&key)))
	return h.Sum64()
}

func New[V any](opts ...Option) Cache[V] {
	opt := defaultOption()
	for _, o := range opts {
		o(opt)
	}

	c := cache[V]{opt: opt}
	if opt.localSlotNum > 0 && opt.localSlotSize > 0 {
		createSimpleLRU := func() lru.LRU[string, V] {
			if opt.expirationEvict {
				return lru.NewExpirationLRU[string, V](opt.localSlotSize, opt.localSuccessTTL, opt.localFailedTTL, opt.target,
			} else {
				return lru.NewLazyLRU[string, V](opt.localSlotSize, opt.localSuccessTTL, opt.localFailedTTL, opt.target, c.onEv
			}
		}
		if opt.localSlotNum == 1 {
			c.local = createSimpleLRU()
		} else {
			c.local = lru.NewSlotLRU[string, V](opt.localSlotNum, LRUStringHash, createSimpleLRU)
		}
		if opt.linkSlotNum > 0 {
			c.link = link.New(opt.linkSlotNum)
```

```
  }
 }
 return &c
}

type cache[V any] struct {
 opt   *option
 link  link.Link
 local lru.LRU[string, V]
}

func (c *cache[V]) onEvict(key string, value V) {
 if c.link != nil {
  // Do not delete other keys while the underlying LRU still holds its lock;
  // defer linked deletions to avoid re-entering the same slot and deadlocking.
  if lks := c.link.Del(key); len(lks) > 0 {
   go c.delLinked(key, lks)
  }
 }
}

func (c *cache[V]) delLinked(src string, keys map[string]struct{}) {
 for k := range keys {
  if src != k {
   c.local.Del(k)
  }
 }
}

func (c *cache[V]) del(key ...string) {
 if c.local == nil {
  return
 }
 for _, k := range key {
  c.local.Del(k)
  if c.link != nil {
   lks := c.link.Del(k)
   for k := range lks {
    c.local.Del(k)
   }
  }
 }
}

func (c *cache[V]) Get(ctx context.Context, key string, fetch func(ctx context.Context) (V, error)) (V, error) {
 return c.GetLink(ctx, key, fetch)
}

func (c *cache[V]) GetLink(ctx context.Context, key string, fetch func(ctx context.Context) (V, error), link ...stri
 if c.local != nil {
  return c.local.Get(key, func() (V, error) {
   if len(link) > 0 && c.link != nil {
    c.link.Link(key, link...)
   }
   return fetch(ctx)
  })
 } else {
  return fetch(ctx)
 }
}

func (c *cache[V]) Del(ctx context.Context, key ...string) {
 for _, fn := range c.opt.delFn {
  fn(ctx, key...)
 }
 c.del(key...)
```

```
}

func (c *cache[V]) DelLocal(ctx context.Context, key ...string) {
	c.del(key...)
}

func (c *cache[V]) Stop() {
	c.local.Stop()
}
```

pkg/localcache/cache_test.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package localcache

import (
	"context"
	"fmt"
	"math/rand"
	"sync"
	"sync/atomic"
	"testing"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/localcache/lru"
)

func TestName(t *testing.T) {
	c := New[string](WithExpirationEvict())
	//c := New[string]()
	ctx := context.Background()

	const (
		num  = 10000
		tNum = 10000
		kNum = 100000
		pNum = 100
	)

	getKey := func(v uint64) string {
		return fmt.Sprintf("key_%d", v%kNum)
	}

	start := time.Now()
	t.Log("start", start)

	var (
		get atomic.Int64
		del atomic.Int64
	)

	incrGet := func() {
		if v := get.Add(1); v%pNum == 0 {
			//t.Log("#get count", v/pNum)
		}
	}
	incrDel := func() {
		if v := del.Add(1); v%pNum == 0 {
			//t.Log("@del count", v/pNum)
		}
	}

	var wg sync.WaitGroup
```

```
	for i := 0; i < tNum; i++ {
		wg.Add(2)
		go func() {
			defer wg.Done()
			for i := 0; i < num; i++ {
				c.Get(ctx, getKey(rand.Uint64()), func(ctx context.Context) (string, error) {
					return fmt.Sprintf("index_%d", i), nil
				})
				incrGet()
			}
		}()

		go func() {
			defer wg.Done()
			time.Sleep(time.Second / 10)
			for i := 0; i < num; i++ {
				c.Del(ctx, getKey(rand.Uint64()))
				incrDel()
			}
		}()
	}

	wg.Wait()
	end := time.Now()
	t.Log("end", end)
	t.Log("time", end.Sub(start))
	t.Log("get", get.Load())
	t.Log("del", del.Load())
	// 137.35s
}

// Test deadlock scenario when eviction callback deletes a linked key that hashes to the same slot.
func TestCacheEvictDeadlock(t *testing.T) {
	ctx := context.Background()
	c := New[string](WithLocalSlotNum(1), WithLocalSlotSize(1), WithLazy())

	if _, err := c.GetLink(ctx, "k1", func(ctx context.Context) (string, error) {
		return "v1", nil
	}, "k2"); err != nil {
		t.Fatalf("seed cache failed: %v", err)
	}

	done := make(chan struct{})
	go func() {
		defer close(done)
		_, _ = c.GetLink(ctx, "k2", func(ctx context.Context) (string, error) {
			return "v2", nil
		}, "k1")
	}()

	select {
	case <-done:
		// expected to finish quickly; current implementation deadlocks here.
	case <-time.After(time.Second):
		t.Fatal("GetLink deadlocked during eviction of linked key")
	}
}

func TestExpirationLRUGetBatch(t *testing.T) {
	l := lru.NewExpirationLRU[string, string](2, time.Minute, time.Second*5, EmptyTarget{}, nil)

	keys := []string{"a", "b"}
	values, err := l.GetBatch(keys, func(keys []string) (map[string]string, error) {
		res := make(map[string]string)
		for _, k := range keys {
```

```
        res[k] = k + "_v"
    }
    return res, nil
  })
  if err != nil {
    t.Fatalf("unexpected error: %v", err)
  }
  if len(values) != len(keys) {
    t.Fatalf("expected %d values, got %d", len(keys), len(values))
  }
  for _, k := range keys {
    if v, ok := values[k]; !ok || v != k+"_v" {
      t.Fatalf("unexpected value for %s: %q, ok=%v", k, v, ok)
    }
  }

  // second batch should hit cache
  values, err = l.GetBatch(keys, func(keys []string) (map[string]string, error) {
    t.Fatalf("should not fetch on cache hit")
    return nil, nil
  })
  if err != nil {
    t.Fatalf("unexpected error on cache hit: %v", err)
  }
  for _, k := range keys {
    if v, ok := values[k]; !ok || v != k+"_v" {
      t.Fatalf("unexpected cached value for %s: %q, ok=%v", k, v, ok)
    }
  }
}
```

pkg/localcache/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package localcache // import "github.com/openimsdk/open-im-server/v3/pkg/localcache"
```

pkg/localcache/init.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package localcache

import (
	"strings"
	"sync"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
)

var (
	once      sync.Once
	subscribe map[string][]string
)

func InitLocalCache(localCache *config.LocalCache) {
	once.Do(func() {
		list := []struct {
			Local config.CacheConfig
			Keys  []string
		}{
			{
				Local: localCache.User,
				Keys:  []string{cachekey.UserInfoKey, cachekey.UserGlobalRecvMsgOptKey},
			},
			{
				Local: localCache.Group,
				Keys:  []string{cachekey.GroupMemberIDsKey, cachekey.GroupInfoKey, cachekey.GroupMemberInfoKey},
			},
			{
				Local: localCache.Friend,
				Keys:  []string{cachekey.FriendIDsKey, cachekey.BlackIDsKey},
			},
			{
				Local: localCache.Conversation,
				Keys:  []string{cachekey.ConversationKey, cachekey.ConversationIDsKey, cachekey.ConversationNotReceiveMessageUs
			},
		}
		subscribe = make(map[string][]string)
		for _, v := range list {
			if v.Local.Enable() {
				subscribe[v.Local.Topic] = v.Keys
			}
		}
	})
}

func GetPublishKeysByTopic(topics []string, keys []string) map[string][]string {
	keysByTopic := make(map[string][]string)
	for _, topic := range topics {
```

```
        keysByTopic[topic] = []string{}
    }

    for _, key := range keys {
        for _, topic := range topics {
            prefixes, ok := subscribe[topic]
            if !ok {
                continue
            }
            for _, prefix := range prefixes {
                if strings.HasPrefix(key, prefix) {
                    keysByTopic[topic] = append(keysByTopic[topic], key)
                    break
                }
            }
        }
    }

    return keysByTopic
}
```

pkg/localcache/option.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package localcache

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/localcache/lru"
)

func defaultOption() *option {
	return &option{
		localSlotNum:    500,
		localSlotSize:   20000,
		linkSlotNum:     500,
		expirationEvict: false,
		localSuccessTTL: time.Minute,
		localFailedTTL:  time.Second * 5,
		delFn:           make([]func(ctx context.Context, key ...string), 0, 2),
		target:          EmptyTarget{},
	}
}

type option struct {
	localSlotNum  int
	localSlotSize int
	linkSlotNum   int
	// expirationEvict: true means that the cache will be actively cleared when the timer expires,
	// false means that the cache will be lazily deleted.
	expirationEvict bool
	localSuccessTTL time.Duration
	localFailedTTL  time.Duration
	delFn           []func(ctx context.Context, key ...string)
	target          lru.Target
}

type Option func(o *option)

func WithExpirationEvict() Option {
	return func(o *option) {
		o.expirationEvict = true
	}
}

func WithLazy() Option {
	return func(o *option) {
		o.expirationEvict = false
	}
}

func WithLocalDisable() Option {
```

```
	return WithLinkSlotNum(0)
}

func WithLinkDisable() Option {
	return WithLinkSlotNum(0)
}

func WithLinkSlotNum(linkSlotNum int) Option {
	return func(o *option) {
		o.linkSlotNum = linkSlotNum
	}
}

func WithLocalSlotNum(localSlotNum int) Option {
	return func(o *option) {
		o.localSlotNum = localSlotNum
	}
}

func WithLocalSlotSize(localSlotSize int) Option {
	return func(o *option) {
		o.localSlotSize = localSlotSize
	}
}

func WithLocalSuccessTTL(localSuccessTTL time.Duration) Option {
	if localSuccessTTL < 0 {
		panic("localSuccessTTL should be greater than 0")
	}
	return func(o *option) {
		o.localSuccessTTL = localSuccessTTL
	}
}

func WithLocalFailedTTL(localFailedTTL time.Duration) Option {
	if localFailedTTL < 0 {
		panic("localFailedTTL should be greater than 0")
	}
	return func(o *option) {
		o.localFailedTTL = localFailedTTL
	}
}

func WithTarget(target lru.Target) Option {
	if target == nil {
		panic("target should not be nil")
	}
	return func(o *option) {
		o.target = target
	}
}

func WithDeleteKeyBefore(fn func(ctx context.Context, key ...string)) Option {
	if fn == nil {
		panic("fn should not be nil")
	}
	return func(o *option) {
		o.delFn = append(o.delFn, fn)
	}
}

type EmptyTarget struct{}

func (e EmptyTarget) IncrGetHit() {}

func (e EmptyTarget) IncrGetSuccess() {}
```

```
func (e EmptyTarget) IncrGetFailed() {}

func (e EmptyTarget) IncrDelHit() {}

func (e EmptyTarget) IncrDelNotFound() {}
```

pkg/localcache/tool.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package localcache

func AnyValue[V any](v any, err error) (V, error) {
    if err != nil {
        var zero V
        return zero, err
    }
    return v.(V), nil
}
```

pkg/localcache/link

pkg/localcache/link/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package link // import "github.com/openimsdk/open-im-server/v3/pkg/localcache/link"
```

pkg/localcache/link/link.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package link

import (
	"hash/fnv"
	"sync"
	"unsafe"
)

type Link interface {
	Link(key string, link ...string)
	Del(key string) map[string]struct{}
}

func newLinkKey() *linkKey {
	return &linkKey{
		data: make(map[string]map[string]struct{}),
	}
}

type linkKey struct {
	lock sync.Mutex
	data map[string]map[string]struct{}
}

func (x *linkKey) link(key string, link ...string) {
	x.lock.Lock()
	defer x.lock.Unlock()
	v, ok := x.data[key]
	if !ok {
		v = make(map[string]struct{})
		x.data[key] = v
	}
	for _, k := range link {
		v[k] = struct{}{}
	}
}

func (x *linkKey) del(key string) map[string]struct{} {
	x.lock.Lock()
	defer x.lock.Unlock()
	ks, ok := x.data[key]
	if !ok {
		return nil
	}
	delete(x.data, key)
	return ks
}

func New(n int) Link {
	if n <= 0 {
```

```
		panic("must be greater than 0")
	}
	slots := make([]*linkKey, n)
	for i := 0; i < len(slots); i++ {
		slots[i] = newLinkKey()
	}
	return &slot{
		n:     uint64(n),
		slots: slots,
	}
}

type slot struct {
	n     uint64
	slots []*linkKey
}

func (x *slot) index(s string) uint64 {
	h := fnv.New64a()
	_, _ = h.Write(*(*[]byte)(unsafe.Pointer(&s)))
	return h.Sum64() % x.n
}

func (x *slot) Link(key string, link ...string) {
	if len(link) == 0 {
		return
	}
	mk := key
	lks := make([]string, len(link))
	for i, k := range link {
		lks[i] = k
	}
	x.slots[x.index(mk)].link(mk, lks...)
	for _, lk := range lks {
		x.slots[x.index(lk)].link(lk, mk)
	}
}

func (x *slot) Del(key string) map[string]struct{} {
	return x.delKey(key)
}

func (x *slot) delKey(k string) map[string]struct{} {
	del := make(map[string]struct{})
	stack := []string{k}
	for len(stack) > 0 {
		curr := stack[len(stack)-1]
		stack = stack[:len(stack)-1]
		if _, ok := del[curr]; ok {
			continue
		}
		del[curr] = struct{}{}
		childKeys := x.slots[x.index(curr)].del(curr)
		for ck := range childKeys {
			stack = append(stack, ck)
		}
	}
	return del
}
```

pkg/localcache/link/link_test.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package link

import (
	"testing"
)

func TestName(t *testing.T) {

	v := New(1)

	//v.Link("a:1", "b:1", "c:1", "d:1")
	v.Link("a:1", "b:1", "c:1")
	v.Link("z:1", "b:1")

	//v.DelKey("a:1")
	v.Del("z:1")

	t.Log(v)

}
```

pkg/localcache/lru

pkg/localcache/lru/doc.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package lru // import "github.com/openimsdk/open-im-server/v3/pkg/localcache/lru"
```

pkg/localcache/lru/lru.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package lru

import "github.com/hashicorp/golang-lru/v2/simplelru"

type EvictCallback[K comparable, V any] simplelru.EvictCallback[K, V]

type LRU[K comparable, V any] interface {
	Get(key K, fetch func() (V, error)) (V, error)
	Set(key K, value V)
	SetHas(key K, value V) bool
	GetBatch(keys []K, fetch func(keys []K) (map[K]V, error)) (map[K]V, error)
	Del(key K) bool
	Stop()
}

type Target interface {
	IncrGetHit()
	IncrGetSuccess()
	IncrGetFailed()

	IncrDelHit()
	IncrDelNotFound()
}
```

pkg/localcache/lru/lru_expiration.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package lru

import (
	"sync"
	"time"

	"github.com/hashicorp/golang-lru/v2/expirable"
)

func NewExpirationLRU[K comparable, V any](size int, successTTL, failedTTL time.Duration, target Target, onEvict Evi
	var cb expirable.EvictCallback[K, *expirationLruItem[V]]
	if onEvict != nil {
		cb = func(key K, value *expirationLruItem[V]) {
			onEvict(key, value.value)
		}
	}
	core := expirable.NewLRU[K, *expirationLruItem[V]](size, cb, successTTL)
	return &ExpirationLRU[K, V]{
		core:       core,
		successTTL: successTTL,
		failedTTL:  failedTTL,
		target:     target,
	}
}

type expirationLruItem[V any] struct {
	lock  sync.RWMutex
	err   error
	value V
}

type ExpirationLRU[K comparable, V any] struct {
	lock       sync.Mutex
	core       *expirable.LRU[K, *expirationLruItem[V]]
	successTTL time.Duration
	failedTTL  time.Duration
	target     Target
}

func (x *ExpirationLRU[K, V]) GetBatch(keys []K, fetch func(keys []K) (map[K]V, error)) (map[K]V, error) {
	var (
		err     error
		results = make(map[K]V)
		misses  = make([]K, 0, len(keys))
	)

	for _, key := range keys {
		x.lock.Lock()
		v, ok := x.core.Get(key)
		x.lock.Unlock()
```

```
    if ok {
        x.target.IncrGetHit()
        v.lock.RLock()
        results[key] = v.value
        if v.err != nil && err == nil {
            err = v.err
        }
        v.lock.RUnlock()
        continue
    }
    misses = append(misses, key)
}

if len(misses) == 0 {
    return results, err
}

fetchValues, fetchErr := fetch(misses)
if fetchErr != nil && err == nil {
    err = fetchErr
}

for key, val := range fetchValues {
    results[key] = val
    if fetchErr != nil {
        x.target.IncrGetFailed()
        continue
    }
    x.target.IncrGetSuccess()
    item := &expirationLruItem[V]{value: val}
    x.lock.Lock()
    x.core.Add(key, item)
    x.lock.Unlock()
}

// any keys not returned from fetch remain absent (no cache write)
return results, err
}

func (x *ExpirationLRU[K, V]) Get(key K, fetch func() (V, error)) (V, error) {
x.lock.Lock()
v, ok := x.core.Get(key)
if ok {
    x.lock.Unlock()
    x.target.IncrGetSuccess()
    v.lock.RLock()
    defer v.lock.RUnlock()
    return v.value, v.err
} else {
    v = &expirationLruItem[V]{}
    x.core.Add(key, v)
    v.lock.Lock()
    x.lock.Unlock()
    defer v.lock.Unlock()
    v.value, v.err = fetch()
    if v.err == nil {
        x.target.IncrGetSuccess()
    } else {
        x.target.IncrGetFailed()
        x.core.Remove(key)
    }
    return v.value, v.err
}
}

func (x *ExpirationLRU[K, V]) Del(key K) bool {
```

```
	x.lock.Lock()
	ok := x.core.Remove(key)
	x.lock.Unlock()
	if ok {
		x.target.IncrDelHit()
	} else {
		x.target.IncrDelNotFound()
	}
	return ok
}

func (x *ExpirationLRU[K, V]) SetHas(key K, value V) bool {
	x.lock.Lock()
	defer x.lock.Unlock()
	if x.core.Contains(key) {
		x.core.Add(key, &expirationLruItem[V]{value: value})
		return true
	}
	return false
}

func (x *ExpirationLRU[K, V]) Set(key K, value V) {
	x.lock.Lock()
	defer x.lock.Unlock()
	x.core.Add(key, &expirationLruItem[V]{value: value})
}

func (x *ExpirationLRU[K, V]) Stop() {
}
```

pkg/localcache/lru/lru_lazy.go

```
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package lru

import (
	"sync"
	"time"

	"github.com/hashicorp/golang-lru/v2/simplelru"
)

type lazyLruItem[V any] struct {
	lock    sync.Mutex
	expires int64
	err     error
	value   V
}

func NewLazyLRU[K comparable, V any](size int, successTTL, failedTTL time.Duration, target Target, onEvict EvictCall
	var cb simplelru.EvictCallback[K, *lazyLruItem[V]]
	if onEvict != nil {
		cb = func(key K, value *lazyLruItem[V]) {
			onEvict(key, value.value)
		}
	}
	core, err := simplelru.NewLRU[K, *lazyLruItem[V]](size, cb)
	if err != nil {
		panic(err)
	}
	return &LazyLRU[K, V]{
		core:       core,
		successTTL: successTTL,
		failedTTL:  failedTTL,
		target:     target,
	}
}

type LazyLRU[K comparable, V any] struct {
	lock       sync.Mutex
	core       *simplelru.LRU[K, *lazyLruItem[V]]
	successTTL time.Duration
	failedTTL  time.Duration
	target     Target
}

func (x *LazyLRU[K, V]) Get(key K, fetch func() (V, error)) (V, error) {
	x.lock.Lock()
	v, ok := x.core.Get(key)
	if ok {
		x.lock.Unlock()
		v.lock.Lock()
		expires, value, err := v.expires, v.value, v.err
```

```
  if expires != 0 && expires > time.Now().UnixMilli() {
    v.lock.Unlock()
    x.target.IncrGetHit()
    return value, err
  }
 } else {
  v = &lazyLruItem[V]{}
  x.core.Add(key, v)
  v.lock.Lock()
  x.lock.Unlock()
 }
 defer v.lock.Unlock()
 if v.expires > time.Now().UnixMilli() {
  return v.value, v.err
 }
 v.value, v.err = fetch()
 if v.err == nil {
  v.expires = time.Now().Add(x.successTTL).UnixMilli()
  x.target.IncrGetSuccess()
 } else {
  v.expires = time.Now().Add(x.failedTTL).UnixMilli()
  x.target.IncrGetFailed()
 }
 return v.value, v.err
}

func (x *LazyLRU[K, V]) GetBatch(keys []K, fetch func(keys []K) (map[K]V, error)) (map[K]V, error) {
 var (
  err  error
  once sync.Once
 )

 res := make(map[K]V)
 queries := make([]K, 0, len(keys))

 for _, key := range keys {
  x.lock.Lock()
  v, ok := x.core.Get(key)
  x.lock.Unlock()
  if ok {
   v.lock.Lock()
   expires, value, err1 := v.expires, v.value, v.err
   v.lock.Unlock()
   if expires != 0 && expires > time.Now().UnixMilli() {
    x.target.IncrGetHit()
    res[key] = value
    if err1 != nil {
     once.Do(func() {
      err = err1
     })
    }
    continue
   }
  }
  queries = append(queries, key)
 }

 if len(queries) == 0 {
  return res, err
 }

 values, fetchErr := fetch(queries)
 if fetchErr != nil {
  once.Do(func() {
   err = fetchErr
  })
```

```
	}

	for key, val := range values {
		v := &lazyLruItem[V]{}
		v.value = val

		if err == nil {
			v.expires = time.Now().Add(x.successTTL).UnixMilli()
			x.target.IncrGetSuccess()
		} else {
			v.expires = time.Now().Add(x.failedTTL).UnixMilli()
			x.target.IncrGetFailed()
		}

		x.lock.Lock()
		x.core.Add(key, v)
		x.lock.Unlock()
		res[key] = val
	}

	return res, err
}

//func (x *LazyLRU[K, V]) Has(key K) bool {
//	x.lock.Lock()
//	defer x.lock.Unlock()
//	return x.core.Contains(key)
//}

func (x *LazyLRU[K, V]) Set(key K, value V) {
	x.lock.Lock()
	defer x.lock.Unlock()
	x.core.Add(key, &lazyLruItem[V]{value: value, expires: time.Now().Add(x.successTTL).UnixMilli()})
}

func (x *LazyLRU[K, V]) SetHas(key K, value V) bool {
	x.lock.Lock()
	defer x.lock.Unlock()
	if x.core.Contains(key) {
		x.core.Add(key, &lazyLruItem[V]{value: value, expires: time.Now().Add(x.successTTL).UnixMilli()})
		return true
	}
	return false
}

func (x *LazyLRU[K, V]) Del(key K) bool {
	x.lock.Lock()
	ok := x.core.Remove(key)
	x.lock.Unlock()
	if ok {
		x.target.IncrDelHit()
	} else {
		x.target.IncrDelNotFound()
	}
	return ok
}

func (x *LazyLRU[K, V]) Stop() {

}
```

pkg/localcache/lru/lru_lazy_test.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package lru

import (
	"fmt"
	"hash/fnv"
	"sync"
	"sync/atomic"
	"testing"
	"time"
	"unsafe"
)

type cacheTarget struct {
	getHit      int64
	getSuccess  int64
	getFailed   int64
	delHit      int64
	delNotFound int64
}

func (r *cacheTarget) IncrGetHit() {
	atomic.AddInt64(&r.getHit, 1)
}

func (r *cacheTarget) IncrGetSuccess() {
	atomic.AddInt64(&r.getSuccess, 1)
}

func (r *cacheTarget) IncrGetFailed() {
	atomic.AddInt64(&r.getFailed, 1)
}

func (r *cacheTarget) IncrDelHit() {
	atomic.AddInt64(&r.delHit, 1)
}

func (r *cacheTarget) IncrDelNotFound() {
	atomic.AddInt64(&r.delNotFound, 1)
}

func (r *cacheTarget) String() string {
	return fmt.Sprintf("getHit: %d, getSuccess: %d, getFailed: %d, delHit: %d, delNotFound: %d", r.getHit, r.getSuccess
}

func TestName(t *testing.T) {
	target := &cacheTarget{}
	l := NewSlotLRU[string, string](100, func(k string) uint64 {
		h := fnv.New64a()
		h.Write(*(*[]byte)(unsafe.Pointer(&k)))
		return h.Sum64()
```

```
	}, func() LRU[string, string] {
		return NewExpirationLRU[string, string](100, time.Second*60, time.Second, target, nil)
	})
	//l := NewInertiaLRU[string, string](1000, time.Second*20, time.Second*5, target)

	fn := func(key string, n int, fetch func() (string, error)) {
		for i := 0; i < n; i++ {
			//v, err := l.Get(key, fetch)
			//if err == nil {
			//	t.Log("key", key, "value", v)
			//} else {
			//	t.Error("key", key, err)
			//}
			v, err := l.Get(key, fetch)
			//time.Sleep(time.Second / 100)
			func(v ...any) {}(v, err)
		}
	}

	tmp := make(map[string]struct{})

	var wg sync.WaitGroup
	for i := 0; i < 10000; i++ {
		wg.Add(1)
		key := fmt.Sprintf("key_%d", i%200)
		tmp[key] = struct{}{}
		go func() {
			defer wg.Done()
			//t.Log(key)
			fn(key, 10000, func() (string, error) {

				return "value_" + key, nil
			})
		}()

		//wg.Add(1)
		//go func() {
		//	defer wg.Done()
		//	for i := 0; i < 10; i++ {
		//		l.Del(key)
		//		time.Sleep(time.Second / 3)
		//	}
		//}()
	}
	wg.Wait()
	t.Log(len(tmp))
	t.Log(target.String())

}
```

pkg/localcache/lru/lru_slot.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package lru

func NewSlotLRU[K comparable, V any](slotNum int, hash func(K) uint64, create func() LRU[K, V]) LRU[K, V] {
	x := &slotLRU[K, V]{
		n:     uint64(slotNum),
		slots: make([]LRU[K, V], slotNum),
		hash:  hash,
	}
	for i := 0; i < slotNum; i++ {
		x.slots[i] = create()
	}
	return x
}

type slotLRU[K comparable, V any] struct {
	n     uint64
	slots []LRU[K, V]
	hash  func(k K) uint64
}

func (x *slotLRU[K, V]) GetBatch(keys []K, fetch func(keys []K) (map[K]V, error)) (map[K]V, error) {
	var (
		slotKeys = make(map[uint64][]K)
		vs       = make(map[K]V)
	)

	for _, k := range keys {
		index := x.getIndex(k)
		slotKeys[index] = append(slotKeys[index], k)
	}

	for k, v := range slotKeys {
		batches, err := x.slots[k].GetBatch(v, fetch)
		if err != nil {
			return nil, err
		}
		for key, value := range batches {
			vs[key] = value
		}
	}
	return vs, nil
}

func (x *slotLRU[K, V]) getIndex(k K) uint64 {
	return x.hash(k) % x.n
}

func (x *slotLRU[K, V]) Get(key K, fetch func() (V, error)) (V, error) {
	return x.slots[x.getIndex(key)].Get(key, fetch)
}
```

```
func (x *slotLRU[K, V]) Set(key K, value V) {
	x.slots[x.getIndex(key)].Set(key, value)
}

func (x *slotLRU[K, V]) SetHas(key K, value V) bool {
	return x.slots[x.getIndex(key)].SetHas(key, value)
}

func (x *slotLRU[K, V]) Del(key K) bool {
	return x.slots[x.getIndex(key)].Del(key)
}

func (x *slotLRU[K, V]) Stop() {
	for _, slot := range x.slots {
		slot.Stop()
	}
}
```

pkg/dbbuild

pkg/dbbuild/builder.go

```
package dbbuild

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/redis/go-redis/v9"
)

type Builder interface {
	Mongo(ctx context.Context) (*mongoutil.Client, error)
	Redis(ctx context.Context) (redis.UniversalClient, error)
}

func NewBuilder(mongoConf *config.Mongo, redisConf *config.Redis) Builder {
	if config.Standalone() {
		globalStandalone.setConfig(mongoConf, redisConf)
		return globalStandalone
	}
	return &microservices{
		mongo: mongoConf,
		redis: redisConf,
	}
}
```

pkg/dbbuild/microservices.go

```
package dbbuild

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"github.com/redis/go-redis/v9"
)

type microservices struct {
	mongo *config.Mongo
	redis *config.Redis
}

func (x *microservices) Mongo(ctx context.Context) (*mongoutil.Client, error) {
	return mongoutil.NewMongoDB(ctx, x.mongo.Build())
}

func (x *microservices) Redis(ctx context.Context) (redis.UniversalClient, error) {
	if x.redis.Disable {
		return nil, nil
	}
	return redisutil.NewRedisClient(ctx, x.redis.Build())
}
```

pkg/dbbuild/standalone.go

```
package dbbuild

import (
	"context"
	"sync"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"github.com/redis/go-redis/v9"
)

const (
	standaloneMongo = "mongo"
	standaloneRedis = "redis"
)

var globalStandalone = &standalone{}

type standaloneConn[C any] struct {
	Conn C
	Err  error
}

func (x *standaloneConn[C]) result() (C, error) {
	return x.Conn, x.Err
}

type standalone struct {
	lock  sync.Mutex
	mongo *config.Mongo
	redis *config.Redis
	conn  map[string]any
}

func (x *standalone) setConfig(mongoConf *config.Mongo, redisConf *config.Redis) {
	x.lock.Lock()
	defer x.lock.Unlock()
	x.mongo = mongoConf
	x.redis = redisConf
}

func (x *standalone) Mongo(ctx context.Context) (*mongoutil.Client, error) {
	x.lock.Lock()
	defer x.lock.Unlock()
	if x.conn == nil {
		x.conn = make(map[string]any)
	}
	v, ok := x.conn[standaloneMongo]
	if !ok {
		var val standaloneConn[*mongoutil.Client]
		val.Conn, val.Err = mongoutil.NewMongoDB(ctx, x.mongo.Build())
		v = &val
		x.conn[standaloneMongo] = v
	}
	return v.(*standaloneConn[*mongoutil.Client]).result()
}

func (x *standalone) Redis(ctx context.Context) (redis.UniversalClient, error) {
	x.lock.Lock()
	defer x.lock.Unlock()
	if x.redis.Disable {
		return nil, nil
	}
```

```
	if x.conn == nil {
		x.conn = make(map[string]any)
	}
	v, ok := x.conn[standaloneRedis]
	if !ok {
		var val standaloneConn[redis.UniversalClient]
		val.Conn, val.Err = redisutil.NewRedisClient(ctx, x.redis.Build())
		v = &val
		x.conn[standaloneRedis] = v
	}
	return v.(*standaloneConn[redis.UniversalClient]).result()
}
```

version

version/version.go

```
package version

import (
	_ "embed"
	"strings"
)

//go:embed version
var Version string

func init() {
	Version = strings.Trim(Version, "\n")
	Version = strings.TrimSpace(Version)
}
```

internal

internal/msgtransfer

internal/msgtransfer/callback.go

```
package msgtransfer

import (
	"context"
	"encoding/base64"

	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/stringutil"
	"google.golang.org/protobuf/proto"

	cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
)

func toCommonCallback(ctx context.Context, msg *sdkws.MsgData, command string) cbapi.CommonCallbackReq {
	return cbapi.CommonCallbackReq{
		SendID:           msg.SendID,
		ServerMsgID:      msg.ServerMsgID,
		CallbackCommand:  command,
		ClientMsgID:      msg.ClientMsgID,
		OperationID:      mcontext.GetOperationID(ctx),
		SenderPlatformID: msg.SenderPlatformID,
		SenderNickname:   msg.SenderNickname,
		SessionType:      msg.SessionType,
		MsgFrom:          msg.MsgFrom,
		ContentType:      msg.ContentType,
		Status:           msg.Status,
		SendTime:         msg.SendTime,
		CreateTime:       msg.CreateTime,
		AtUserIDList:     msg.AtUserIDList,
		SenderFaceURL:    msg.SenderFaceURL,
		Content:          GetContent(msg),
		Seq:              uint32(msg.Seq),
		Ex:               msg.Ex,
	}
}

func GetContent(msg *sdkws.MsgData) string {
	if msg.ContentType >= constant.NotificationBegin && msg.ContentType <= constant.NotificationEnd {
		var tips sdkws.TipsComm
		_ = proto.Unmarshal(msg.Content, &tips)
		content := tips.JsonDetail
		return content
	} else {
		return string(msg.Content)
	}
}

func (mc *OnlineHistoryMongoConsumerHandler) webhookAfterMsgSaveDB(ctx context.Context, after *config.AfterConfig, m
	if !filterAfterMsg(msg, after) {
		return
	}
```

```
    cbReq := &cbapi.CallbackAfterMsgSaveDBReq{
        CommonCallbackReq: toCommonCallback(ctx, msg, cbapi.CallbackAfterMsgSaveDBCommand),
    }

    switch msg.SessionType {
    case constant.SingleChatType, constant.NotificationChatType:
        cbReq.RecvID = msg.RecvID
    case constant.ReadGroupChatType:
        cbReq.GroupID = msg.GroupID
    default:
    }

    mc.webhookClient.AsyncPostWithQuery(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterMsgSaveDBResp{}, af
}

func buildKeyMsgDataQuery(msg *sdkws.MsgData) map[string]string {
    keyMsgData := apistruct.KeyMsgData{
        SendID:  msg.SendID,
        RecvID:  msg.RecvID,
        GroupID: msg.GroupID,
    }

    return map[string]string{
        webhook.Key: base64.StdEncoding.EncodeToString(stringutil.StructToJsonBytes(keyMsgData)),
    }
}

func filterAfterMsg(msg *sdkws.MsgData, after *config.AfterConfig) bool {
    return filterMsg(msg, after.AttentionIds, after.DeniedTypes)
}

func filterMsg(msg *sdkws.MsgData, attentionIds []string, deniedTypes []int32) bool {
    // According to the attentionIds configuration, only some users are sent
    if len(attentionIds) != 0 && msg.ContentType == constant.SingleChatType && !datautil.Contain(msg.RecvID, attentionI
        return false
    }

    if len(attentionIds) != 0 && msg.ContentType == constant.ReadGroupChatType && !datautil.Contain(msg.GroupID, attent
        return false
    }

    if defaultDeniedTypes(msg.ContentType) {
        return false
    }

    if len(deniedTypes) != 0 && datautil.Contain(msg.ContentType, deniedTypes...) {
        return false
    }

    return true
}

func defaultDeniedTypes(contentType int32) bool {
    if contentType >= constant.NotificationBegin && contentType <= constant.NotificationEnd {
        return true
    }
    if contentType == constant.Typing {
        return true
    }
    return false
}
```

internal/msgtransfer/init.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msgtransfer

import (
    "context"
    "fmt"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/mqbuild"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/mq"
    "github.com/openimsdk/tools/utils/runtimeenv"

    conf "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/tools/log"
    "google.golang.org/grpc"
)

type MsgTransfer struct {
    historyConsumer      mq.Consumer
    historyMongoConsumer mq.Consumer
    // This consumer aggregated messages, subscribed to the topic:toRedis,
    //  the message is stored in redis, Incr Redis, and then the message is sent to toPush topic for push,
    // and the message is sent to toMongo topic for persistence
    historyHandler *OnlineHistoryRedisConsumerHandler
    //This consumer handle message to mongo
    historyMongoHandler *OnlineHistoryMongoConsumerHandler
    ctx                 context.Context
    //cancel              context.CancelFunc
}

type Config struct {
    MsgTransfer    conf.MsgTransfer
    RedisConfig    conf.Redis
    MongodbConfig  conf.Mongo
    KafkaConfig    conf.Kafka
    Share          conf.Share
    WebhooksConfig conf.Webhooks
    Discovery      conf.Discovery
    Index          conf.Index
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
    builder := mqbuild.NewBuilder(&config.KafkaConfig)

    log.CInfo(ctx, "MSG-TRANSFER server is initializing", "runTimeEnv", runtimeenv.RuntimeEnvironment(), "prometheusPor
```

```
    config.MsgTransfer.Prometheus.Ports, "index", config.Index)
    dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
    mgocli, err := dbb.Mongo(ctx)
    if err != nil {
        return err
    }
    rdb, err := dbb.Redis(ctx)
    if err != nil {
        return err
    }

    //if config.Discovery.Enable == conf.ETCD {
    //    cm := disetcd.NewConfigManager(client.(*etcd.SvcDiscoveryRegistryImpl).GetClient(), []string{
    //        config.MsgTransfer.GetConfigFileName(),
    //        config.RedisConfig.GetConfigFileName(),
    //        config.MongodbConfig.GetConfigFileName(),
    //        config.KafkaConfig.GetConfigFileName(),
    //        config.Share.GetConfigFileName(),
    //        config.WebhooksConfig.GetConfigFileName(),
    //        config.Discovery.GetConfigFileName(),
    //        conf.LogConfigFileName,
    //    })
    //    cm.Watch(ctx)
    //}
    mongoProducer, err := builder.GetTopicProducer(ctx, config.KafkaConfig.ToMongoTopic)
    if err != nil {
        return err
    }
    pushProducer, err := builder.GetTopicProducer(ctx, config.KafkaConfig.ToPushTopic)
    if err != nil {
        return err
    }
    msgDocModel, err := mgo.NewMsgMongo(mgocli.GetDB())
    if err != nil {
        return err
    }
    var msgModel cache.MsgCache
    if rdb == nil {
        cm, err := mgo.NewCacheMgo(mgocli.GetDB())
        if err != nil {
            return err
        }
        msgModel = mcache.NewMsgCache(cm, msgDocModel)
    } else {
        msgModel = redis.NewMsgCache(rdb, msgDocModel)
    }
    seqConversation, err := mgo.NewSeqConversationMongo(mgocli.GetDB())
    if err != nil {
        return err
    }
    seqConversationCache := redis.NewSeqConversationCacheRedis(rdb, seqConversation)
    seqUser, err := mgo.NewSeqUserMongo(mgocli.GetDB())
    if err != nil {
        return err
    }
    seqUserCache := redis.NewSeqUserCacheRedis(rdb, seqUser)
    msgTransferDatabase, err := controller.NewMsgTransferDatabase(msgDocModel, msgModel, seqUserCache, seqConversationC
    if err != nil {
        return err
    }
    historyConsumer, err := builder.GetTopicConsumer(ctx, config.KafkaConfig.ToRedisTopic)
    if err != nil {
        return err
    }
    historyMongoConsumer, err := builder.GetTopicConsumer(ctx, config.KafkaConfig.ToMongoTopic)
    if err != nil {
```

```
    return err
  }
  historyHandler, err := NewOnlineHistoryRedisConsumerHandler(ctx, client, config, msgTransferDatabase)
  if err != nil {
    return err
  }
  historyMongoHandler := NewOnlineHistoryMongoConsumerHandler(msgTransferDatabase, config)

  msgTransfer := &MsgTransfer{
    historyConsumer:      historyConsumer,
    historyMongoConsumer: historyMongoConsumer,
    historyHandler:       historyHandler,
    historyMongoHandler:  historyMongoHandler,
  }

  return msgTransfer.Start(ctx)
}

func (m *MsgTransfer) Start(ctx context.Context) error {
  var cancel context.CancelCauseFunc
  m.ctx, cancel = context.WithCancelCause(ctx)

  go func() {
    for {
      if err := m.historyConsumer.Subscribe(m.ctx, m.historyHandler.HandlerRedisMessage); err != nil {
        cancel(fmt.Errorf("history consumer %w", err))
        log.ZError(m.ctx, "historyConsumer err", err)
        return
      }
    }
  }()

  go func() {
    fn := func(msg mq.Message) error {
      m.historyMongoHandler.HandleChatWs2Mongo(msg)
      return nil
    }
    for {
      if err := m.historyMongoConsumer.Subscribe(m.ctx, fn); err != nil {
        cancel(fmt.Errorf("history mongo consumer %w", err))
        log.ZError(m.ctx, "historyMongoConsumer err", err)
        return
      }
    }
  }()

  go m.historyHandler.HandleUserHasReadSeqMessages(m.ctx)

  err := m.historyHandler.redisMessageBatches.Start()
  if err != nil {
    return err
  }
  <-m.ctx.Done()
  return context.Cause(m.ctx)
}
```

internal/msgtransfer/online_history_msg_handler.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msgtransfer

import (
    "context"
    "encoding/json"
    "errors"

    "github.com/openimsdk/tools/mq"

    "sync"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "github.com/openimsdk/tools/discovery"

    "github.com/go-redis/redis"
    "google.golang.org/protobuf/proto"

    "github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/open-im-server/v3/pkg/tools/batcher"
    "github.com/openimsdk/protocol/constant"
    pbconv "github.com/openimsdk/protocol/conversation"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/utils/stringutil"
)

const (
    size             = 500
    mainDataBuffer   = 500
    subChanBuffer    = 50
    worker           = 50
    interval         = 100 * time.Millisecond
    hasReadChanBuffer = 1000
)

type ContextMsg struct {
    message *sdkws.MsgData
    ctx     context.Context
}

// This structure is used for asynchronously writing the sender's read sequence (seq) regarding a message into Mongo
// For example, if the sender sends a message with a seq of 10, then their own read seq for this conversation should
type userHasReadSeq struct {
    conversationID string
    userHasReadMap map[string]int64
```

```
}

type OnlineHistoryRedisConsumerHandler struct {
    redisMessageBatches *batcher.Batcher[ConsumerMessage]

    msgTransferDatabase         controller.MsgTransferDatabase
    conversationUserHasReadChan chan *userHasReadSeq
    wg                          sync.WaitGroup

    groupClient        *rpcli.GroupClient
    conversationClient *rpcli.ConversationClient
}

type ConsumerMessage struct {
    Ctx   context.Context
    Key   string
    Value []byte
    Raw   mq.Message
}

func NewOnlineHistoryRedisConsumerHandler(ctx context.Context, client discovery.Conn, config *Config, database contr
    groupConn, err := client.GetConn(ctx, config.Discovery.RpcService.Group)
    if err != nil {
        return nil, err
    }
    conversationConn, err := client.GetConn(ctx, config.Discovery.RpcService.Conversation)
    if err != nil {
        return nil, err
    }
    var och OnlineHistoryRedisConsumerHandler
    och.msgTransferDatabase = database
    och.conversationUserHasReadChan = make(chan *userHasReadSeq, hasReadChanBuffer)
    och.groupClient = rpcli.NewGroupClient(groupConn)
    och.conversationClient = rpcli.NewConversationClient(conversationConn)
    och.wg.Add(1)

    b := batcher.New[ConsumerMessage](
        batcher.WithSize(size),
        batcher.WithWorker(worker),
        batcher.WithInterval(interval),
        batcher.WithDataBuffer(mainDataBuffer),
        batcher.WithSyncWait(true),
        batcher.WithBuffer(subChanBuffer),
    )
    b.Sharding = func(key string) int {
        hashCode := stringutil.GetHashCode(key)
        return int(hashCode) % och.redisMessageBatches.Worker()
    }
    b.Key = func(consumerMessage *ConsumerMessage) string {
        return consumerMessage.Key
    }
    b.Do = och.do
    och.redisMessageBatches = b

    och.redisMessageBatches.OnComplete = func(lastMessage *ConsumerMessage, totalCount int) {
        lastMessage.Raw.Mark()
        lastMessage.Raw.Commit()
    }

    return &och, nil
}
func (och *OnlineHistoryRedisConsumerHandler) do(ctx context.Context, channelID int, val *batcher.Msg[ConsumerMessag
    ctx = mcontext.WithTriggerIDContext(ctx, val.TriggerID())
    ctxMessages := och.parseConsumerMessages(ctx, val.Val())
    ctx = withAggregationCtx(ctx, ctxMessages)
    log.ZInfo(ctx, "msg arrived channel", "channel id", channelID, "msgList length", len(ctxMessages), "key", val.Key()
```

```
    och.doSetReadSeq(ctx, ctxMessages)

    storageMsgList, notStorageMsgList, storageNotificationList, notStorageNotificationList :=
        och.categorizeMessageLists(ctxMessages)
    log.ZDebug(ctx, "number of categorized messages", "storageMsgList", len(storageMsgList), "notStorageMsgList",
        len(notStorageMsgList), "storageNotificationList", len(storageNotificationList), "notStorageNotificationList", le

    conversationIDMsg := msgprocessor.GetChatConversationIDByMsg(ctxMessages[0].message)
    conversationIDNotification := msgprocessor.GetNotificationConversationIDByMsg(ctxMessages[0].message)
    och.handleMsg(ctx, val.Key(), conversationIDMsg, storageMsgList, notStorageMsgList)
    och.handleNotification(ctx, val.Key(), conversationIDNotification, storageNotificationList, notStorageNotificationL
}

func (och *OnlineHistoryRedisConsumerHandler) doSetReadSeq(ctx context.Context, msgs []*ContextMsg) {

    // Outer map: conversationID -> (userID -> maxHasReadSeq)
    conversationUserSeq := make(map[string]map[string]int64)

    for _, msg := range msgs {
        if msg.message.ContentType != constant.HasReadReceipt {
            continue
        }
        var elem sdkws.NotificationElem
        if err := json.Unmarshal(msg.message.Content, &elem); err != nil {
            log.ZWarn(ctx, "Unmarshal NotificationElem error", err, "msg", msg)
            continue
        }
        var tips sdkws.MarkAsReadTips
        if err := json.Unmarshal([]byte(elem.Detail), &tips); err != nil {
            log.ZWarn(ctx, "Unmarshal MarkAsReadTips error", err, "msg", msg)
            continue
        }
        if len(tips.ConversationID) == 0 || tips.HasReadSeq < 0 {
            continue
        }

        // Calculate the max seq from tips.Seqs
        for _, seq := range tips.Seqs {
            if tips.HasReadSeq < seq {
                tips.HasReadSeq = seq
            }
        }

        if _, ok := conversationUserSeq[tips.ConversationID]; !ok {
            conversationUserSeq[tips.ConversationID] = make(map[string]int64)
        }
        if conversationUserSeq[tips.ConversationID][tips.MarkAsReadUserID] < tips.HasReadSeq {
            conversationUserSeq[tips.ConversationID][tips.MarkAsReadUserID] = tips.HasReadSeq
        }
    }
    log.ZInfo(ctx, "doSetReadSeq", "conversationUserSeq", conversationUserSeq)

    // persist to db
    for convID, userSeqMap := range conversationUserSeq {
        if err := och.msgTransferDatabase.SetHasReadSeqToDB(ctx, convID, userSeqMap); err != nil {
            log.ZWarn(ctx, "SetHasReadSeqToDB error", err, "conversationID", convID, "userSeqMap", userSeqMap)
        }
    }

}

func (och *OnlineHistoryRedisConsumerHandler) parseConsumerMessages(ctx context.Context, consumerMessages []*Consume
    var ctxMessages []*ContextMsg
    for i := 0; i < len(consumerMessages); i++ {
        ctxMsg := &ContextMsg{}
        msgFromMQ := &sdkws.MsgData{}
```

```
    err := proto.Unmarshal(consumerMessages[i].Value, msgFromMQ)
    if err != nil {
        log.ZWarn(ctx, "msg_transfer Unmarshal msg err", err, string(consumerMessages[i].Value))
        continue
    }
    ctxMsg.ctx = consumerMessages[i].Ctx
    ctxMsg.message = msgFromMQ
    log.ZDebug(ctx, "message parse finish", "message", msgFromMQ, "key", consumerMessages[i].Key)
    ctxMessages = append(ctxMessages, ctxMsg)
  }
  return ctxMessages
}

// Get messages/notifications stored message list, not stored and pushed message list.
func (och *OnlineHistoryRedisConsumerHandler) categorizeMessageLists(totalMsgs []*ContextMsg) (storageMsgList,
  notStorageMsgList, storageNotificationList, notStorageNotificationList []*ContextMsg) {
  for _, v := range totalMsgs {
    options := msgprocessor.Options(v.message.Options)
    if !options.IsNotNotification() {
        // clone msg from notificationMsg
        if options.IsSendMsg() {
            msg := proto.Clone(v.message).(*sdkws.MsgData)
            // message
            if v.message.Options != nil {
                msg.Options = msgprocessor.NewMsgOptions()
            }
            msg.Options = msgprocessor.WithOptions(msg.Options,
                msgprocessor.WithOfflinePush(options.IsOfflinePush()),
                msgprocessor.WithUnreadCount(options.IsUnreadCount()),
            )
            v.message.Options = msgprocessor.WithOptions(
                v.message.Options,
                msgprocessor.WithOfflinePush(false),
                msgprocessor.WithUnreadCount(false),
            )
            ctxMsg := &ContextMsg{
                message: msg,
                ctx:     v.ctx,
            }
            storageMsgList = append(storageMsgList, ctxMsg)
        }
        if options.IsHistory() {
            storageNotificationList = append(storageNotificationList, v)
        } else {
            notStorageNotificationList = append(notStorageNotificationList, v)
        }
    } else {
        if options.IsHistory() {
            storageMsgList = append(storageMsgList, v)
        } else {
            notStorageMsgList = append(notStorageMsgList, v)
        }
    }
  }
  return
}

func (och *OnlineHistoryRedisConsumerHandler) handleMsg(ctx context.Context, key, conversationID string, storageList
  log.ZInfo(ctx, "handle storage msg")
  for _, storageMsg := range storageList {
    log.ZDebug(ctx, "handle storage msg", "msg", storageMsg.message.String())
  }

  och.toPushTopic(ctx, key, conversationID, notStorageList)
  var storageMessageList []*sdkws.MsgData
  for _, msg := range storageList {
```

```
        storageMessageList = append(storageMessageList, msg.message)
    }
    if len(storageMessageList) > 0 {
        msg := storageMessageList[0]
        lastSeq, isNewConversation, userSeqMap, err := och.msgTransferDatabase.BatchInsertChat2Cache(ctx, conversationID,
        if err != nil && !errors.Is(errs.Unwrap(err), redis.Nil) {
            log.ZWarn(ctx, "batch data insert to redis err", err, "storageMsgList", storageMessageList)
            return
        }
        log.ZInfo(ctx, "BatchInsertChat2Cache end")
        err = och.msgTransferDatabase.SetHasReadSeqs(ctx, conversationID, userSeqMap)
        if err != nil {
            log.ZWarn(ctx, "SetHasReadSeqs error", err, "userSeqMap", userSeqMap, "conversationID", conversationID)
            prommetrics.SeqSetFailedCounter.Inc()
        }
        och.conversationUserHasReadChan <- &userHasReadSeq{
            conversationID: conversationID,
            userHasReadMap: userSeqMap,
        }

        if isNewConversation {
            switch msg.SessionType {
            case constant.ReadGroupChatType:
                log.ZDebug(ctx, "group chat first create conversation", "conversationID",
                    conversationID)

                userIDs, err := och.groupClient.GetGroupMemberUserIDs(ctx, msg.GroupID)
                if err != nil {
                    log.ZWarn(ctx, "get group member ids error", err, "conversationID",
                        conversationID)
                } else {
                    log.ZInfo(ctx, "GetGroupMemberIDs end")

                    if err := och.conversationClient.CreateGroupChatConversations(ctx, msg.GroupID, userIDs); err != nil {
                        log.ZWarn(ctx, "single chat first create conversation error", err,
                            "conversationID", conversationID)
                    }
                }
            case constant.SingleChatType, constant.NotificationChatType:
                req := &pbconv.CreateSingleChatConversationsReq{
                    RecvID:           msg.RecvID,
                    SendID:           msg.SendID,
                    ConversationID:   conversationID,
                    ConversationType: msg.SessionType,
                }
                if err := och.conversationClient.CreateSingleChatConversations(ctx, req); err != nil {
                    log.ZWarn(ctx, "single chat or notification first create conversation error", err,
                        "conversationID", conversationID, "sessionType", msg.SessionType)
                }
            default:
                log.ZWarn(ctx, "unknown session type", nil, "sessionType",
                    msg.SessionType)
            }
        }

        log.ZInfo(ctx, "success incr to next topic")
        err = och.msgTransferDatabase.MsgToMongoMQ(ctx, key, conversationID, storageMessageList, lastSeq)
        if err != nil {
            log.ZError(ctx, "Msg To MongoDB MQ error", err, "conversationID",
                conversationID, "storageList", storageMessageList, "lastSeq", lastSeq)
        }
        log.ZInfo(ctx, "MsgToMongoMQ end")

        och.toPushTopic(ctx, key, conversationID, storageList)
        log.ZInfo(ctx, "toPushTopic end")
    }
```

```
}

func (och *OnlineHistoryRedisConsumerHandler) handleNotification(ctx context.Context, key, conversationID string,
 storageList, notStorageList []*ContextMsg) {
 och.toPushTopic(ctx, key, conversationID, notStorageList)
 var storageMessageList []*sdkws.MsgData
 for _, msg := range storageList {
  storageMessageList = append(storageMessageList, msg.message)
 }
 if len(storageMessageList) > 0 {
  lastSeq, _, _, err := och.msgTransferDatabase.BatchInsertChat2Cache(ctx, conversationID, storageMessageList)
  if err != nil {
   log.ZError(ctx, "notification batch insert to redis error", err, "conversationID", conversationID,
    "storageList", storageMessageList)
   return
  }
  log.ZDebug(ctx, "success to next topic", "conversationID", conversationID)
  err = och.msgTransferDatabase.MsgToMongoMQ(ctx, key, conversationID, storageMessageList, lastSeq)
  if err != nil {
   log.ZError(ctx, "Msg To MongoDB MQ error", err, "conversationID",
    conversationID, "storageList", storageMessageList, "lastSeq", lastSeq)
  }
  och.toPushTopic(ctx, key, conversationID, storageList)
 }
}
func (och *OnlineHistoryRedisConsumerHandler) HandleUserHasReadSeqMessages(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.ZPanic(ctx, "HandleUserHasReadSeqMessages Panic", errs.ErrPanic(r))
  }
 }()

 defer och.wg.Done()

 for msg := range och.conversationUserHasReadChan {
  if err := och.msgTransferDatabase.SetHasReadSeqToDB(ctx, msg.conversationID, msg.userHasReadMap); err != nil {
   log.ZWarn(ctx, "set read seq to db error", err, "conversationID", msg.conversationID, "userSeqMap", msg.userHasR
  }
 }

 log.ZInfo(ctx, "Channel closed, exiting handleUserHasReadSeqMessages")
}
func (och *OnlineHistoryRedisConsumerHandler) Close() {
 close(och.conversationUserHasReadChan)
 och.wg.Wait()
}

func (och *OnlineHistoryRedisConsumerHandler) toPushTopic(ctx context.Context, key, conversationID string, msgs []*C
 for _, v := range msgs {
  log.ZDebug(ctx, "push msg to topic", "msg", v.message.String())
  if err := och.msgTransferDatabase.MsgToPushMQ(v.ctx, key, conversationID, v.message); err != nil {
   log.ZError(ctx, "msg to push topic error", err, "msg", v.message.String())
  }
 }
}

func withAggregationCtx(ctx context.Context, values []*ContextMsg) context.Context {
 var allMessageOperationID string
 for i, v := range values {
  if opid := mcontext.GetOperationID(v.ctx); opid != "" {
   if i == 0 {
    allMessageOperationID += opid
   } else {
    allMessageOperationID += "$" + opid
   }
  }
```

```
    }
    return mcontext.SetOperationID(ctx, allMessageOperationID)
}

func (och *OnlineHistoryRedisConsumerHandler) HandlerRedisMessage(msg mq.Message) error { // a instance in the consu
    err := och.redisMessageBatches.Put(msg.Context(), &ConsumerMessage{Ctx: msg.Context(), Key: msg.Key(), Value: msg.V
    if err != nil {
        log.ZWarn(msg.Context(), "put msg to  error", err, "key", msg.Key(), "value", msg.Value())
    }
    return nil
}
```

internal/msgtransfer/online_msg_to_mongo_handler.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msgtransfer

import (
	"github.com/openimsdk/tools/mq"

	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	pbmsg "github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/tools/log"
	"google.golang.org/protobuf/proto"
)

type OnlineHistoryMongoConsumerHandler struct {
	msgTransferDatabase controller.MsgTransferDatabase
	config              *Config
	webhookClient       *webhook.Client
}

func NewOnlineHistoryMongoConsumerHandler(database controller.MsgTransferDatabase, config *Config) *OnlineHistoryMon
	return &OnlineHistoryMongoConsumerHandler{
		msgTransferDatabase: database,
		config:              config,
		webhookClient:       webhook.NewWebhookClient(config.WebhooksConfig.URL),
	}
}

func (mc *OnlineHistoryMongoConsumerHandler) HandleChatWs2Mongo(val mq.Message) {
	ctx := val.Context()
	key := val.Key()
	msg := val.Value()
	msgFromMQ := pbmsg.MsgDataToMongoByMQ{}
	err := proto.Unmarshal(msg, &msgFromMQ)
	if err != nil {
		log.ZError(ctx, "unmarshall failed", err, "key", key, "len", len(msg))
		return
	}
	if len(msgFromMQ.MsgData) == 0 {
		log.ZError(ctx, "msgFromMQ.MsgData is empty", nil, "key", key, "msg", msg)
		return
	}
	log.ZDebug(ctx, "mongo consumer recv msg", "msgs", msgFromMQ.String())
	err = mc.msgTransferDatabase.BatchInsertChat2DB(ctx, msgFromMQ.ConversationID, msgFromMQ.MsgData, msgFromMQ.LastSeq
	if err != nil {
		log.ZError(ctx, "batch data insert to mongo err", err, "msg", msgFromMQ.MsgData, "conversationID", msgFromMQ.Conv
		prommetrics.MsgInsertMongoFailedCounter.Inc()
	} else {
		prommetrics.MsgInsertMongoSuccessCounter.Inc()
		val.Mark()
	}
```

```
    for _, msgData := range msgFromMQ.MsgData {
        mc.webhookAfterMsgSaveDB(ctx, &mc.config.WebhooksConfig.AfterMsgSaveDB, msgData)
    }

    //var seqs []int64
    //for _, msg := range msgFromMQ.MsgData {
    //    seqs = append(seqs, msg.Seq)
    //}
    //if err := mc.msgTransferDatabase.DeleteMessagesFromCache(ctx, msgFromMQ.ConversationID, seqs); err != nil {
    //    log.ZError(ctx, "remove cache msg from redis err", err, "msg",
    //        msgFromMQ.MsgData, "conversationID", msgFromMQ.ConversationID)
    //}
}
```

internal/push

internal/push/callback.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package push

import (
    "context"
    "encoding/json"

    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"

    "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/mcontext"
)

func (c *ConsumerHandler) webhookBeforeOfflinePush(ctx context.Context, before *config.BeforeConfig, userIDs []strin
    return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
        if msg.ContentType == constant.Typing {
            return nil
        }
        req := &callbackstruct.CallbackBeforePushReq{
            UserStatusBatchCallbackReq: callbackstruct.UserStatusBatchCallbackReq{
                UserStatusBaseCallback: callbackstruct.UserStatusBaseCallback{
                    CallbackCommand: callbackstruct.CallbackBeforeOfflinePushCommand,
                    OperationID:     mcontext.GetOperationID(ctx),
                    PlatformID:      int(msg.SenderPlatformID),
                    Platform:        constant.PlatformIDToName(int(msg.SenderPlatformID)),
                },
                UserIDList: userIDs,
            },
            OfflinePushInfo: msg.OfflinePushInfo,
            ClientMsgID:     msg.ClientMsgID,
            SendID:          msg.SendID,
            GroupID:         msg.GroupID,
            ContentType:     msg.ContentType,
            SessionType:     msg.SessionType,
            AtUserIDs:       msg.AtUserIDList,
            Content:         GetContent(msg),
        }

        resp := &callbackstruct.CallbackBeforePushResp{}

        if err := c.webhookClient.SyncPost(ctx, req.GetCallbackCommand(), req, resp, before); err != nil {
            return err
        }

        if len(resp.UserIDs) != 0 {
```

```
        *offlinePushUserIDs = resp.UserIDs
    }
    if resp.OfflinePushInfo != nil {
        msg.OfflinePushInfo = resp.OfflinePushInfo
    }
    return nil
  })
}

func (c *ConsumerHandler) webhookBeforeOnlinePush(ctx context.Context, before *config.BeforeConfig, userIDs []string
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    if msg.ContentType == constant.Typing {
        return nil
    }
    req := callbackstruct.CallbackBeforePushReq{
        UserStatusBatchCallbackReq: callbackstruct.UserStatusBatchCallbackReq{
            UserStatusBaseCallback: callbackstruct.UserStatusBaseCallback{
                CallbackCommand: callbackstruct.CallbackBeforeOnlinePushCommand,
                OperationID:     mcontext.GetOperationID(ctx),
                PlatformID:      int(msg.SenderPlatformID),
                Platform:        constant.PlatformIDToName(int(msg.SenderPlatformID)),
            },
            UserIDList: userIDs,
        },
        ClientMsgID: msg.ClientMsgID,
        SendID:      msg.SendID,
        GroupID:     msg.GroupID,
        ContentType: msg.ContentType,
        SessionType: msg.SessionType,
        AtUserIDs:   msg.AtUserIDList,
        Content:     GetContent(msg),
    }
    resp := &callbackstruct.CallbackBeforePushResp{}
    if err := c.webhookClient.SyncPost(ctx, req.GetCallbackCommand(), req, resp, before); err != nil {
        return err
    }
    return nil
  })
}

func (c *ConsumerHandler) webhookBeforeGroupOnlinePush(
  ctx context.Context,
  before *config.BeforeConfig,
  groupID string,
  msg *sdkws.MsgData,
  pushToUserIDs *[]string,
) error {
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    if msg.ContentType == constant.Typing {
        return nil
    }
    req := callbackstruct.CallbackBeforeSuperGroupOnlinePushReq{
        UserStatusBaseCallback: callbackstruct.UserStatusBaseCallback{
            CallbackCommand: callbackstruct.CallbackBeforeGroupOnlinePushCommand,
            OperationID:     mcontext.GetOperationID(ctx),
            PlatformID:      int(msg.SenderPlatformID),
            Platform:        constant.PlatformIDToName(int(msg.SenderPlatformID)),
        },
        ClientMsgID: msg.ClientMsgID,
        SendID:      msg.SendID,
        GroupID:     groupID,
        ContentType: msg.ContentType,
        SessionType: msg.SessionType,
        AtUserIDs:   msg.AtUserIDList,
        Content:     GetContent(msg),
        Seq:         msg.Seq,
```

```
    }
    resp := &callbackstruct.CallbackBeforeSuperGroupOnlinePushResp{}
    if err := c.webhookClient.SyncPost(ctx, req.GetCallbackCommand(), req, resp, before); err != nil {
        return err
    }
    if len(resp.UserIDs) != 0 {
        *pushToUserIDs = resp.UserIDs
    }
    return nil
  })
}

func GetContent(msg *sdkws.MsgData) string {
  if msg.ContentType >= constant.NotificationBegin && msg.ContentType <= constant.NotificationEnd {
    var notification sdkws.NotificationElem
    if err := json.Unmarshal(msg.Content, &notification); err != nil {
        return ""
    }
    return notification.Detail
  } else {
    return string(msg.Content)
  }
}
```

internal/push/offlinepush_handler.go

```
package push

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/protocol/constant"
	pbpush "github.com/openimsdk/protocol/push"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/jsonutil"
	"google.golang.org/protobuf/proto"
)

type OfflinePushConsumerHandler struct {
	offlinePusher offlinepush.OfflinePusher
}

func NewOfflinePushConsumerHandler(offlinePusher offlinepush.OfflinePusher) *OfflinePushConsumerHandler {
	return &OfflinePushConsumerHandler{
		offlinePusher: offlinePusher,
	}
}

func (o *OfflinePushConsumerHandler) HandleMsg2OfflinePush(ctx context.Context, msg []byte) {
	offlinePushMsg := pbpush.PushMsgReq{}
	if err := proto.Unmarshal(msg, &offlinePushMsg); err != nil {
		log.ZError(ctx, "offline push Unmarshal msg err", err, "msg", string(msg))
		return
	}
	if offlinePushMsg.MsgData == nil || offlinePushMsg.UserIDs == nil {
		log.ZError(ctx, "offline push msg is empty", errs.New("offlinePushMsg is empty"), "userIDs", offlinePushMsg.UserI
		return
	}
	if offlinePushMsg.MsgData.Status == constant.MsgStatusSending {
		offlinePushMsg.MsgData.Status = constant.MsgStatusSendSuccess
	}
	log.ZInfo(ctx, "receive to OfflinePush MQ", "userIDs", offlinePushMsg.UserIDs, "msg", offlinePushMsg.MsgData)

	err := o.offlinePushMsg(ctx, offlinePushMsg.MsgData, offlinePushMsg.UserIDs)
	if err != nil {
		log.ZWarn(ctx, "offline push failed", err, "msg", offlinePushMsg.String())
	}
}

func (o *OfflinePushConsumerHandler) getOfflinePushInfos(msg *sdkws.MsgData) (title, content string, opts *options.O
	type AtTextElem struct {
		Text       string   `json:"text,omitempty"`
		AtUserList []string `json:"atUserList,omitempty"`
		IsAtSelf   bool     `json:"isAtSelf"`
	}

	opts = &options.Opts{Signal: &options.Signal{ClientMsgID: msg.ClientMsgID}}
	if msg.OfflinePushInfo != nil {
		opts.IOSBadgeCount = msg.OfflinePushInfo.IOSBadgeCount
		opts.IOSPushSound = msg.OfflinePushInfo.IOSPushSound
		opts.Ex = msg.OfflinePushInfo.Ex
	}

	if msg.OfflinePushInfo != nil {
		title = msg.OfflinePushInfo.Title
```

```
		content = msg.OfflinePushInfo.Desc
	}
	if title == "" {
		switch msg.ContentType {
		case constant.Text:
			fallthrough
		case constant.Picture:
			fallthrough
		case constant.Voice:
			fallthrough
		case constant.Video:
			fallthrough
		case constant.File:
			title = constant.ContentType2PushContent[int64(msg.ContentType)]
		case constant.AtText:
			ac := AtTextElem{}
			_ = jsonutil.JsonStringToStruct(string(msg.Content), &ac)
		case constant.SignalingNotification:
			title = constant.ContentType2PushContent[constant.SignalMsg]
		default:
			title = constant.ContentType2PushContent[constant.Common]
		}
	}
	if content == "" {
		content = title
	}
	return
}

func (o *OfflinePushConsumerHandler) offlinePushMsg(ctx context.Context, msg *sdkws.MsgData, offlinePushUserIDs []st
	title, content, opts, err := o.getOfflinePushInfos(msg)
	if err != nil {
		return err
	}
	err = o.offlinePusher.Push(ctx, offlinePushUserIDs, title, content, opts)
	if err != nil {
		prommetrics.MsgOfflinePushFailedCounter.Inc()
		return err
	}
	return nil
}
```

internal/push/onlinepusher.go

```
package push

import (
	"context"
	"fmt"
	"sync"

	"github.com/openimsdk/protocol/msggateway"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/runtimeenv"
	"golang.org/x/sync/errgroup"
	"google.golang.org/grpc"

	conf "github.com/openimsdk/open-im-server/v3/pkg/common/config"
)

type OnlinePusher interface {
	GetConnsAndOnlinePush(ctx context.Context, msg *sdkws.MsgData,
		pushToUserIDs []string) (wsResults []*msggateway.SingleMsgToUserResults, err error)
	GetOnlinePushFailedUserIDs(ctx context.Context, msg *sdkws.MsgData, wsResults []*msggateway.SingleMsgToUserResults,
		pushToUserIDs *[]string) []string
}

type emptyOnlinePusher struct{}

func newEmptyOnlinePusher() *emptyOnlinePusher {
	return &emptyOnlinePusher{}
}

func (emptyOnlinePusher) GetConnsAndOnlinePush(ctx context.Context, msg *sdkws.MsgData, pushToUserIDs []string) (wsR
	log.ZInfo(ctx, "emptyOnlinePusher GetConnsAndOnlinePush", nil)
	return nil, nil
}
func (u emptyOnlinePusher) GetOnlinePushFailedUserIDs(ctx context.Context, msg *sdkws.MsgData, wsResults []*msggatew
	log.ZInfo(ctx, "emptyOnlinePusher GetOnlinePushFailedUserIDs", nil)
	return nil
}

func NewOnlinePusher(disCov discovery.Conn, config *Config) (OnlinePusher, error) {
	if conf.Standalone() {
		return NewDefaultAllNode(disCov, config), nil
	}
	if runtimeenv.RuntimeEnvironment() == conf.KUBERNETES {
		return NewDefaultAllNode(disCov, config), nil
	}
	switch config.Discovery.Enable {
	case conf.ETCD:
		return NewDefaultAllNode(disCov, config), nil
	default:
		return nil, errs.New(fmt.Sprintf("unsupported discovery type %s", config.Discovery.Enable))
	}
}

type DefaultAllNode struct {
	disCov discovery.Conn
	config *Config
}

func NewDefaultAllNode(disCov discovery.Conn, config *Config) *DefaultAllNode {
	return &DefaultAllNode{disCov: disCov, config: config}
```

```
}

func (d *DefaultAllNode) GetConnsAndOnlinePush(ctx context.Context, msg *sdkws.MsgData,
	pushToUserIDs []string) (wsResults []*msggateway.SingleMsgToUserResults, err error) {
	conns, err := d.disCov.GetConns(ctx, d.config.Discovery.RpcService.MessageGateway)
	if len(conns) == 0 {
		log.ZWarn(ctx, "get gateway conn 0 ", nil)
	} else {
		log.ZDebug(ctx, "get gateway conn", "conn length", len(conns))
	}

	if err != nil {
		return nil, err
	}

	var (
		mu         sync.Mutex
		wg         = errgroup.Group{}
		input      = &msggateway.OnlineBatchPushOneMsgReq{MsgData: msg, PushToUserIDs: pushToUserIDs}
		maxWorkers = d.config.RpcConfig.MaxConcurrentWorkers
	)

	if maxWorkers < 3 {
		maxWorkers = 3
	}

	wg.SetLimit(maxWorkers)

	// Online push message
	for _, conn := range conns {
		conn := conn // loop var safe
		ctx := ctx
		wg.Go(func() error {
			msgClient := msggateway.NewMsgGatewayClient(conn)
			reply, err := msgClient.SuperGroupOnlineBatchPushOneMsg(ctx, input)
			if err != nil {
				log.ZError(ctx, "SuperGroupOnlineBatchPushOneMsg ", err, "req:", input.String())
				return nil
			}

			log.ZDebug(ctx, "push result", "reply", reply)
			if reply != nil && reply.SinglePushResult != nil {
				mu.Lock()
				wsResults = append(wsResults, reply.SinglePushResult...)
				mu.Unlock()
			}

			return nil
		})
	}

	_ = wg.Wait()

	// always return nil
	return wsResults, nil
}

func (d *DefaultAllNode) GetOnlinePushFailedUserIDs(_ context.Context, msg *sdkws.MsgData,
	wsResults []*msggateway.SingleMsgToUserResults, pushToUserIDs *[]string) []string {

	onlineSuccessUserIDs := []string{msg.SendID}
	for _, v := range wsResults {
		//message sender do not need offline push
		if msg.SendID == v.UserID {
			continue
		}
```

```
    // mobile online push success
    if v.OnlinePush {
        onlineSuccessUserIDs = append(onlineSuccessUserIDs, v.UserID)
    }

}

return datautil.SliceSub(*pushToUserIDs, onlineSuccessUserIDs)
}

type K8sStaticConsistentHash struct {
disCov discovery.SvcDiscoveryRegistry
config *Config
}

func NewK8sStaticConsistentHash(disCov discovery.SvcDiscoveryRegistry, config *Config) *K8sStaticConsistentHash {
return &K8sStaticConsistentHash{disCov: disCov, config: config}
}

func (k *K8sStaticConsistentHash) GetConnsAndOnlinePush(ctx context.Context, msg *sdkws.MsgData,
pushToUserIDs []string) (wsResults []*msggateway.SingleMsgToUserResults, err error) {

var usersHost = make(map[string][]string)
for _, v := range pushToUserIDs {
    tHost, err := k.disCov.GetUserIdHashGatewayHost(ctx, v)
    if err != nil {
        log.ZError(ctx, "get msg gateway hash error", err)
        return nil, err
    }
    tUsers, tbl := usersHost[tHost]
    if tbl {
        tUsers = append(tUsers, v)
        usersHost[tHost] = tUsers
    } else {
        usersHost[tHost] = []string{v}
    }
}
log.ZDebug(ctx, "genUsers send hosts struct:", "usersHost", usersHost)
var usersConns = make(map[grpc.ClientConnInterface][]string)
for host, userIds := range usersHost {
    tconn, _ := k.disCov.GetConn(ctx, host)
    usersConns[tconn] = userIds
}
var (
    mu         sync.Mutex
    wg         = errgroup.Group{}
    maxWorkers = k.config.RpcConfig.MaxConcurrentWorkers
)
if maxWorkers < 3 {
    maxWorkers = 3
}
wg.SetLimit(maxWorkers)
for conn, userIds := range usersConns {
    tcon := conn
    tuserIds := userIds
    wg.Go(func() error {
        input := &msggateway.OnlineBatchPushOneMsgReq{MsgData: msg, PushToUserIDs: tuserIds}
        msgClient := msggateway.NewMsgGatewayClient(tcon)
        reply, err := msgClient.SuperGroupOnlineBatchPushOneMsg(ctx, input)
        if err != nil {
            return nil
        }
        log.ZDebug(ctx, "push result", "reply", reply)
        if reply != nil && reply.SinglePushResult != nil {
            mu.Lock()
            wsResults = append(wsResults, reply.SinglePushResult...)
```

```
        mu.Unlock()
    }
    return nil
  })
 }
 _ = wg.Wait()
 return wsResults, nil
}
func (k *K8sStaticConsistentHash) GetOnlinePushFailedUserIDs(_ context.Context, _ *sdkws.MsgData,
 wsResults []*msggateway.SingleMsgToUserResults, _ *[]string) []string {
 var needOfflinePushUserIDs []string
 for _, v := range wsResults {
  if !v.OnlinePush {
   needOfflinePushUserIDs = append(needOfflinePushUserIDs, v.UserID)
  }
 }
 return needOfflinePushUserIDs
}
```

internal/push/push.go

```
package push

import (
	"context"
	"github.com/openimsdk/tools/mq"
	"math/rand"
	"strconv"

	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
	"github.com/openimsdk/open-im-server/v3/pkg/mqbuild"
	pbpush "github.com/openimsdk/protocol/push"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"google.golang.org/grpc"
)

type pushServer struct {
	pbpush.UnimplementedPushMsgServiceServer
	database      controller.PushDatabase
	disCov        discovery.Conn
	offlinePusher offlinepush.OfflinePusher
}

type Config struct {
	RpcConfig          config.Push
	RedisConfig        config.Redis
	MongoConfig        config.Mongo
	KafkaConfig        config.Kafka
	NotificationConfig config.Notification
	Share              config.Share
	WebhooksConfig     config.Webhooks
	LocalCacheConfig   config.LocalCache
	Discovery          config.Discovery
	FcmConfigPath      config.Path
}

func (p pushServer) DelUserPushToken(ctx context.Context,
	req *pbpush.DelUserPushTokenReq) (resp *pbpush.DelUserPushTokenResp, err error) {
	if err = p.database.DelFcmToken(ctx, req.UserID, int(req.PlatformID)); err != nil {
		return nil, err
	}
	return &pbpush.DelUserPushTokenResp{}, nil
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
	dbb := dbbuild.NewBuilder(&config.MongoConfig, &config.RedisConfig)
	rdb, err := dbb.Redis(ctx)
	if err != nil {
		return err
	}
	var cacheModel cache.ThirdCache
	if rdb == nil {
		mdb, err := dbb.Mongo(ctx)
		if err != nil {
			return err
```

```
    }
    mc, err := mgo.NewCacheMgo(mdb.GetDB())
    if err != nil {
        return err
    }
    cacheModel = mcache.NewThirdCache(mc)
  } else {
    cacheModel = redis.NewThirdCache(rdb)
  }
  offlinePusher, err := offlinepush.NewOfflinePusher(&config.RpcConfig, cacheModel, string(config.FcmConfigPath))
  if err != nil {
    return err
  }
  builder := mqbuild.NewBuilder(&config.KafkaConfig)

  offlinePushProducer, err := builder.GetTopicProducer(ctx, config.KafkaConfig.ToOfflinePushTopic)
  if err != nil {
    return err
  }
  database := controller.NewPushDatabase(cacheModel, offlinePushProducer)

  pushConsumer, err := builder.GetTopicConsumer(ctx, config.KafkaConfig.ToPushTopic)
  if err != nil {
    return err
  }
  offlinePushConsumer, err := builder.GetTopicConsumer(ctx, config.KafkaConfig.ToOfflinePushTopic)
  if err != nil {
    return err
  }

  pushHandler, err := NewConsumerHandler(ctx, config, database, offlinePusher, rdb, client)
  if err != nil {
    return err
  }

  offlineHandler := NewOfflinePushConsumerHandler(offlinePusher)

  pbpush.RegisterPushMsgServiceServer(server, &pushServer{
    database:      database,
    disCov:        client,
    offlinePusher: offlinePusher,
  })

  go func() {
    pushHandler.WaitCache()
    fn := func(msg mq.Message) error {
      pushHandler.HandleMs2PsChat(authverify.WithTempAdmin(msg.Context()), msg.Value())
      return nil
    }
    consumerCtx := mcontext.SetOperationID(context.Background(), "push_"+strconv.Itoa(int(rand.Uint32())))
    log.ZInfo(consumerCtx, "begin consume messages")
    for {
      if err := pushConsumer.Subscribe(consumerCtx, fn); err != nil {
        log.ZError(consumerCtx, "subscribe err", err)
        return
      }
    }
  }()

  go func() {
    fn := func(msg mq.Message) error {
      offlineHandler.HandleMsg2OfflinePush(msg.Context(), msg.Value())
      return nil
    }
    consumerCtx := mcontext.SetOperationID(context.Background(), "push_"+strconv.Itoa(int(rand.Uint32())))
    log.ZInfo(consumerCtx, "begin consume messages")
```

```
    for {
        if err := offlinePushConsumer.Subscribe(consumerCtx, fn); err != nil {
            log.ZError(consumerCtx, "subscribe err", err)
            return
        }
    }
  }()

  return nil
}
```

internal/push/push_handler.go

```
package push

import (
	"context"
	"encoding/json"
	"time"

	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
	"github.com/openimsdk/open-im-server/v3/pkg/rpccache"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/open-im-server/v3/pkg/util/conversationutil"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msggateway"
	pbpush "github.com/openimsdk/protocol/push"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/jsonutil"
	"github.com/openimsdk/tools/utils/timeutil"
	"github.com/redis/go-redis/v9"
	"google.golang.org/protobuf/proto"
)

type ConsumerHandler struct {
	//pushConsumerGroup      mq.Consumer
	offlinePusher          offlinepush.OfflinePusher
	onlinePusher           OnlinePusher
	pushDatabase           controller.PushDatabase
	onlineCache            rpccache.OnlineCache
	groupLocalCache        *rpccache.GroupLocalCache
	conversationLocalCache *rpccache.ConversationLocalCache
	webhookClient          *webhook.Client
	config                 *Config
	userClient             *rpcli.UserClient
	groupClient            *rpcli.GroupClient
	msgClient              *rpcli.MsgClient
	conversationClient     *rpcli.ConversationClient
}

func NewConsumerHandler(ctx context.Context, config *Config, database controller.PushDatabase, offlinePusher offline
	userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
	if err != nil {
		return nil, err
	}
	groupConn, err := client.GetConn(ctx, config.Discovery.RpcService.Group)
	if err != nil {
		return nil, err
	}
	msgConn, err := client.GetConn(ctx, config.Discovery.RpcService.Msg)
	if err != nil {
		return nil, err
	}
	conversationConn, err := client.GetConn(ctx, config.Discovery.RpcService.Conversation)
	if err != nil {
		return nil, err
	}
	onlinePusher, err := NewOnlinePusher(client, config)
```

```
    if err != nil {
        return nil, err
    }
    var consumerHandler ConsumerHandler
    consumerHandler.userClient = rpcli.NewUserClient(userConn)
    consumerHandler.groupClient = rpcli.NewGroupClient(groupConn)
    consumerHandler.msgClient = rpcli.NewMsgClient(msgConn)
    consumerHandler.conversationClient = rpcli.NewConversationClient(conversationConn)

    consumerHandler.offlinePusher = offlinePusher
    consumerHandler.onlinePusher = onlinePusher
    consumerHandler.groupLocalCache = rpccache.NewGroupLocalCache(consumerHandler.groupClient, &config.LocalCacheConfig
    consumerHandler.conversationLocalCache = rpccache.NewConversationLocalCache(consumerHandler.conversationClient, &co
    consumerHandler.webhookClient = webhook.NewWebhookClient(config.WebhooksConfig.URL)
    consumerHandler.config = config
    consumerHandler.pushDatabase = database
    consumerHandler.onlineCache, err = rpccache.NewOnlineCache(consumerHandler.userClient, consumerHandler.groupLocalCa
    if err != nil {
        return nil, err
    }
    return &consumerHandler, nil
}

func (c *ConsumerHandler) HandleMs2PsChat(ctx context.Context, msg []byte) {
    msgFromMQ := pbpush.PushMsgReq{}
    if err := proto.Unmarshal(msg, &msgFromMQ); err != nil {
        log.ZError(ctx, "push Unmarshal msg err", err, "msg", string(msg))
        return
    }

    sec := msgFromMQ.MsgData.SendTime / 1000
    nowSec := timeutil.GetCurrentTimestampBySecond()

    if nowSec-sec > 10 {
        prommetrics.MsgLoneTimePushCounter.Inc()
        log.ZWarn(ctx, "it’s been a while since the message was sent", nil, "msg", msgFromMQ.String(), "sec", sec, "nowSe
    }
    var err error

    switch msgFromMQ.MsgData.SessionType {
    case constant.ReadGroupChatType:
        err = c.Push2Group(ctx, msgFromMQ.MsgData.GroupID, msgFromMQ.MsgData)
    default:
        var pushUserIDList []string
        isSenderSync := datautil.GetSwitchFromOptions(msgFromMQ.MsgData.Options, constant.IsSenderSync)
        if !isSenderSync || msgFromMQ.MsgData.SendID == msgFromMQ.MsgData.RecvID {
            pushUserIDList = append(pushUserIDList, msgFromMQ.MsgData.RecvID)
        } else {
            pushUserIDList = append(pushUserIDList, msgFromMQ.MsgData.RecvID, msgFromMQ.MsgData.SendID)
        }
        err = c.Push2User(ctx, pushUserIDList, msgFromMQ.MsgData)
    }
    if err != nil {
        log.ZWarn(ctx, "push failed", err, "msg", msgFromMQ.String())
    }
}

func (c *ConsumerHandler) WaitCache() {
    c.onlineCache.WaitCache()
}

// Push2User Suitable for two types of conversations, one is SingleChatType and the other is NotificationChatType.
func (c *ConsumerHandler) Push2User(ctx context.Context, userIDs []string, msg *sdkws.MsgData) (err error) {
    log.ZInfo(ctx, "Get msg from msg_transfer And push msg", "userIDs", userIDs, "msg", msg.String())
    defer func(duration time.Time) {
        t := time.Since(duration)
```

```
		log.ZInfo(ctx, "Get msg from msg_transfer And push msg end", "msg", msg.String(), "time cost", t)
	}(time.Now())
	if err := c.webhookBeforeOnlinePush(ctx, &c.config.WebhooksConfig.BeforeOnlinePush, userIDs, msg); err != nil {
		return err
	}

	wsResults, err := c.GetConnsAndOnlinePush(ctx, msg, userIDs)
	if err != nil {
		return err
	}

	log.ZDebug(ctx, "single and notification push result", "result", wsResults, "msg", msg, "push_to_userID", userIDs)
	log.ZInfo(ctx, "single and notification push end")

	if !c.shouldPushOffline(ctx, msg) {
		return nil
	}
	log.ZInfo(ctx, "pushOffline start")

	for _, v := range wsResults {
		//message sender do not need offline push
		if msg.SendID == v.UserID {
			continue
		}
		//receiver online push success
		if v.OnlinePush {
			return nil
		}
	}
	needOfflinePushUserID := []string{msg.RecvID}
	var offlinePushUserID []string

	//receiver offline push
	if err = c.webhookBeforeOfflinePush(ctx, &c.config.WebhooksConfig.BeforeOfflinePush, needOfflinePushUserID, msg, &c
		return err
	}

	if len(offlinePushUserID) > 0 {
		needOfflinePushUserID = offlinePushUserID
	}
	err = c.offlinePushMsg(ctx, msg, needOfflinePushUserID)
	if err != nil {
		log.ZDebug(ctx, "offlinePushMsg failed", err, "needOfflinePushUserID", needOfflinePushUserID, "msg", msg)
		log.ZWarn(ctx, "offlinePushMsg failed", err, "needOfflinePushUserID length", len(needOfflinePushUserID), "msg", ms
		return nil
	}

	return nil
}

func (c *ConsumerHandler) shouldPushOffline(_ context.Context, msg *sdkws.MsgData) bool {
	isOfflinePush := datautil.GetSwitchFromOptions(msg.Options, constant.IsOfflinePush)
	if !isOfflinePush {
		return false
	}
	switch msg.ContentType {
	case constant.RoomParticipantsConnectedNotification:
		return false
	case constant.RoomParticipantsDisconnectedNotification:
		return false
	}
	return true
}

func (c *ConsumerHandler) GetConnsAndOnlinePush(ctx context.Context, msg *sdkws.MsgData, pushToUserIDs []string) ([]
	if msg != nil && msg.Status == constant.MsgStatusSending {
```

```
		msg.Status = constant.MsgStatusSendSuccess
	}
	onlineUserIDs, offlineUserIDs, err := c.onlineCache.GetUsersOnline(ctx, pushToUserIDs)
	if err != nil {
		return nil, err
	}

	log.ZDebug(ctx, "GetConnsAndOnlinePush online cache", "sendID", msg.SendID, "recvID", msg.RecvID, "groupID", msg.Gr
	var result []*msggateway.SingleMsgToUserResults
	if len(onlineUserIDs) > 0 {
		var err error
		result, err = c.onlinePusher.GetConnsAndOnlinePush(ctx, msg, onlineUserIDs)
		if err != nil {
			return nil, err
		}
	}
	for _, userID := range offlineUserIDs {
		result = append(result, &msggateway.SingleMsgToUserResults{
			UserID: userID,
		})
	}
	return result, nil
}

func (c *ConsumerHandler) Push2Group(ctx context.Context, groupID string, msg *sdkws.MsgData) (err error) {
	log.ZInfo(ctx, "Get group msg from msg_transfer and push msg", "msg", msg.String(), "groupID", groupID)
	defer func(duration time.Time) {
		t := time.Since(duration)
		log.ZInfo(ctx, "Get group msg from msg_transfer and push msg end", "msg", msg.String(), "groupID", groupID, "time
	}(time.Now())
	var pushToUserIDs []string
	if err = c.webhookBeforeGroupOnlinePush(ctx, &c.config.WebhooksConfig.BeforeGroupOnlinePush, groupID, msg,
		&pushToUserIDs); err != nil {
		return err
	}

	err = c.groupMessagesHandler(ctx, groupID, &pushToUserIDs, msg)
	if err != nil {
		return err
	}

	wsResults, err := c.GetConnsAndOnlinePush(ctx, msg, pushToUserIDs)
	if err != nil {
		return err
	}

	log.ZDebug(ctx, "group push result", "result", wsResults, "msg", msg)
	log.ZInfo(ctx, "online group push end")

	if !c.shouldPushOffline(ctx, msg) {
		return nil
	}
	needOfflinePushUserIDs := c.onlinePusher.GetOnlinePushFailedUserIDs(ctx, msg, wsResults, &pushToUserIDs)
	//filter some user, like don not disturb or don't need offline push etc.
	needOfflinePushUserIDs, err = c.filterGroupMessageOfflinePush(ctx, groupID, msg, needOfflinePushUserIDs)
	if err != nil {
		return err
	}
	log.ZInfo(ctx, "filterGroupMessageOfflinePush end")

	// Use offline push messaging
	if len(needOfflinePushUserIDs) > 0 {
		c.asyncOfflinePush(ctx, needOfflinePushUserIDs, msg)
	}

	return nil
```

```
}

func (c *ConsumerHandler) asyncOfflinePush(ctx context.Context, needOfflinePushUserIDs []string, msg *sdkws.MsgData)
    var offlinePushUserIDs []string
    err := c.webhookBeforeOfflinePush(ctx, &c.config.WebhooksConfig.BeforeOfflinePush, needOfflinePushUserIDs, msg, &of
    if err != nil {
        log.ZWarn(ctx, "webhookBeforeOfflinePush failed", err, "msg", msg)
        return
    }

    if len(offlinePushUserIDs) > 0 {
        needOfflinePushUserIDs = offlinePushUserIDs
    }
    if err := c.pushDatabase.MsgToOfflinePushMQ(ctx, conversationutil.GenConversationUniqueKeyForSingle(msg.SendID, msg
        log.ZDebug(ctx, "Msg To OfflinePush MQ error", err, "needOfflinePushUserIDs",
            needOfflinePushUserIDs, "msg", msg)
        log.ZWarn(ctx, "Msg To OfflinePush MQ error", err, "needOfflinePushUserIDs length",
            len(needOfflinePushUserIDs), "msg", msg)
        prommetrics.GroupChatMsgProcessFailedCounter.Inc()
        return
    }
}

func (c *ConsumerHandler) groupMessagesHandler(ctx context.Context, groupID string, pushToUserIDs *[]string, msg *sd
    if len(*pushToUserIDs) == 0 {
        *pushToUserIDs, err = c.groupLocalCache.GetGroupMemberIDs(ctx, groupID)
        if err != nil {
            return err
        }
        switch msg.ContentType {
        case constant.MemberQuitNotification:
            var tips sdkws.MemberQuitTips
            if unmarshalNotificationElem(msg.Content, &tips) != nil {
                return err
            }
            if err = c.DeleteMemberAndSetConversationSeq(ctx, groupID, []string{tips.QuitUser.UserID}); err != nil {
                log.ZError(ctx, "MemberQuitNotification DeleteMemberAndSetConversationSeq", err, "groupID", groupID, "userID",
            }
            *pushToUserIDs = append(*pushToUserIDs, tips.QuitUser.UserID)
        case constant.MemberKickedNotification:
            var tips sdkws.MemberKickedTips
            if unmarshalNotificationElem(msg.Content, &tips) != nil {
                return err
            }
            kickedUsers := datautil.Slice(tips.KickedUserList, func(e *sdkws.GroupMemberFullInfo) string { return e.UserID }
            if err = c.DeleteMemberAndSetConversationSeq(ctx, groupID, kickedUsers); err != nil {
                log.ZError(ctx, "MemberKickedNotification DeleteMemberAndSetConversationSeq", err, "groupID", groupID, "userIDs
            }

            *pushToUserIDs = append(*pushToUserIDs, kickedUsers...)
        case constant.GroupDismissedNotification:
            if msgprocessor.IsNotification(msgprocessor.GetConversationIDByMsg(msg)) {
                var tips sdkws.GroupDismissedTips
                if unmarshalNotificationElem(msg.Content, &tips) != nil {
                    return err
                }
                log.ZDebug(ctx, "GroupDismissedNotificationInfo****", "groupID", groupID, "num", len(*pushToUserIDs), "list", p
                if len(c.config.Share.IMAdminUser.UserIDs) > 0 {
                    ctx = mcontext.WithOpUserIDContext(ctx, c.config.Share.IMAdminUser.UserIDs[0])
                }
                defer func(groupID string) {
                    if err := c.groupClient.DismissGroup(ctx, groupID, true); err != nil {
                        log.ZError(ctx, "DismissGroup Notification clear members", err, "groupID", groupID)
                    }
                }(groupID)
            }
```

```
    }
  }
  return err
}

func (c *ConsumerHandler) offlinePushMsg(ctx context.Context, msg *sdkws.MsgData, offlinePushUserIDs []string) error
  title, content, opts, err := c.getOfflinePushInfos(msg)
  if err != nil {
    log.ZError(ctx, "getOfflinePushInfos failed", err, "msg", msg)
    return err
  }
  err = c.offlinePusher.Push(ctx, offlinePushUserIDs, title, content, opts)
  if err != nil {
    prommetrics.MsgOfflinePushFailedCounter.Inc()
    return err
  }
  return nil
}

func (c *ConsumerHandler) filterGroupMessageOfflinePush(ctx context.Context, groupID string, msg *sdkws.MsgData,
  offlinePushUserIDs []string) (userIDs []string, err error) {
  needOfflinePushUserIDs, err := c.conversationClient.GetConversationOfflinePushUserIDs(ctx, conversationutil.GenGrou
  if err != nil {
    return nil, err
  }
  return needOfflinePushUserIDs, nil
}

func (c *ConsumerHandler) getOfflinePushInfos(msg *sdkws.MsgData) (title, content string, opts *options.Opts, err er
  type AtTextElem struct {
    Text       string   `json:"text,omitempty"`
    AtUserList []string `json:"atUserList,omitempty"`
    IsAtSelf   bool     `json:"isAtSelf"`
  }

  opts = &options.Opts{Signal: &options.Signal{ClientMsgID: msg.ClientMsgID}}
  if msg.OfflinePushInfo != nil {
    opts.IOSBadgeCount = msg.OfflinePushInfo.IOSBadgeCount
    opts.IOSPushSound = msg.OfflinePushInfo.IOSPushSound
    opts.Ex = msg.OfflinePushInfo.Ex
  }

  if msg.OfflinePushInfo != nil {
    title = msg.OfflinePushInfo.Title
    content = msg.OfflinePushInfo.Desc
  }
  if title == "" {
    switch msg.ContentType {
    case constant.Text:
      fallthrough
    case constant.Picture:
      fallthrough
    case constant.Voice:
      fallthrough
    case constant.Video:
      fallthrough
    case constant.File:
      title = constant.ContentType2PushContent[int64(msg.ContentType)]
    case constant.AtText:
      ac := AtTextElem{}
      _ = jsonutil.JsonStringToStruct(string(msg.Content), &ac)
    case constant.SignalingNotification:
      title = constant.ContentType2PushContent[constant.SignalMsg]
    default:
      title = constant.ContentType2PushContent[constant.Common]
    }
```

```
	}
	if content == "" {
		content = title
	}
	return
}

func (c *ConsumerHandler) DeleteMemberAndSetConversationSeq(ctx context.Context, groupID string, userIDs []string) e
	conversationID := msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, groupID)
	maxSeq, err := c.msgClient.GetConversationMaxSeq(ctx, conversationID)
	if err != nil {
		return err
	}
	return c.conversationClient.SetConversationMaxSeq(ctx, conversationID, userIDs, maxSeq)
}

func unmarshalNotificationElem(bytes []byte, t any) error {
	var notification sdkws.NotificationElem
	if err := json.Unmarshal(bytes, &notification); err != nil {
		return err
	}
	return json.Unmarshal([]byte(notification.Detail), t)
}
```

internal/push/offlinepush

internal/push/offlinepush/offlinepusher.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package offlinepush

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/dummy"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/fcm"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/getui"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/jpush"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"strings"
)

const (
	geTUI    = "getui"
	firebase = "fcm"
	jPush    = "jpush"
)

// OfflinePusher Offline Pusher.
type OfflinePusher interface {
	Push(ctx context.Context, userIDs []string, title, content string, opts *options.Opts) error
}

func NewOfflinePusher(pushConf *config.Push, cache cache.ThirdCache, fcmConfigPath string) (OfflinePusher, error) {
	var offlinePusher OfflinePusher
	pushConf.Enable = strings.ToLower(pushConf.Enable)
	switch pushConf.Enable {
	case geTUI:
		offlinePusher = getui.NewClient(pushConf, cache)
	case firebase:
		return fcm.NewClient(pushConf, cache, fcmConfigPath)
	case jPush:
		offlinePusher = jpush.NewClient(pushConf)
	default:
		offlinePusher = dummy.NewClient()
	}
	return offlinePusher, nil
}
```

internal/push/offlinepush/fcm

internal/push/offlinepush/fcm/push.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package fcm

import (
	"context"
	"errors"
	"fmt"
	"path/filepath"
	"strings"

	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/tools/utils/httputil"

	firebase "firebase.google.com/go/v4"
	"firebase.google.com/go/v4/messaging"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/errs"
	"github.com/redis/go-redis/v9"
	"google.golang.org/api/option"
)

const SinglePushCountLimit = 400

var Terminal = []int{constant.IOSPlatformID, constant.AndroidPlatformID, constant.WebPlatformID}

type Fcm struct {
	fcmMsgCli *messaging.Client
	cache     cache.ThirdCache
}

// NewClient initializes a new FCM client using the Firebase Admin SDK.
// It requires the FCM service account credentials file located within the project's configuration directory.
func NewClient(pushConf *config.Push, cache cache.ThirdCache, fcmConfigPath string) (*Fcm, error) {
	var opt option.ClientOption
	switch {
	case len(pushConf.FCM.FilePath) != 0:
		// with file path
		credentialsFilePath := filepath.Join(fcmConfigPath, pushConf.FCM.FilePath)
		opt = option.WithCredentialsFile(credentialsFilePath)
	case len(pushConf.FCM.AuthURL) != 0:
		// with authentication URL
		client := httputil.NewHTTPClient(httputil.NewClientConfig())
		resp, err := client.Get(pushConf.FCM.AuthURL)
		if err != nil {
			return nil, err
		}
```

```
    opt = option.WithCredentialsJSON(resp)
  default:
    return nil, errs.New("no FCM config").Wrap()
  }

  fcmApp, err := firebase.NewApp(context.Background(), nil, opt)
  if err != nil {
    return nil, errs.Wrap(err)
  }
  ctx := context.Background()
  fcmMsgClient, err := fcmApp.Messaging(ctx)
  if err != nil {
    return nil, errs.Wrap(err)
  }
  return &Fcm{fcmMsgCli: fcmMsgClient, cache: cache}, nil
}

func (f *Fcm) Push(ctx context.Context, userIDs []string, title, content string, opts *options.Opts) error {
  // accounts->registrationToken
  allTokens := make(map[string][]string, 0)
  for _, account := range userIDs {
    var personTokens []string
    for _, v := range Terminal {
      Token, err := f.cache.GetFcmToken(ctx, account, v)
      if err == nil {
        personTokens = append(personTokens, Token)
      }
    }
    allTokens[account] = personTokens
  }
  Success := 0
  Fail := 0
  notification := &messaging.Notification{}
  notification.Body = content
  notification.Title = title
  var messages []*messaging.Message
  var sendErrBuilder strings.Builder
  var msgErrBuilder strings.Builder
  for userID, personTokens := range allTokens {
    apns := &messaging.APNSConfig{Payload: &messaging.APNSPayload{Aps: &messaging.Aps{Sound: opts.IOSPushSound}}}
    messageCount := len(messages)
    if messageCount >= SinglePushCountLimit {
      response, err := f.fcmMsgCli.SendEach(ctx, messages)
      if err != nil {
        Fail = Fail + messageCount
        // Record push error
        sendErrBuilder.WriteString(err.Error())
        sendErrBuilder.WriteByte('.')
      } else {
        Success = Success + response.SuccessCount
        Fail = Fail + response.FailureCount
        if response.FailureCount != 0 {
          // Record message error
          for i := range response.Responses {
            if !response.Responses[i].Success {
              msgErrBuilder.WriteString(response.Responses[i].Error.Error())
              msgErrBuilder.WriteByte('.')
            }
          }
        }
      }
      messages = messages[0:0]
    }
    if opts.IOSBadgeCount {
      unreadCountSum, err := f.cache.IncrUserBadgeUnreadCountSum(ctx, userID)
      if err == nil {
```

```
        apns.Payload.Aps.Badge = &unreadCountSum
    } else {
        // log.Error(operationID, "IncrUserBadgeUnreadCountSum redis err", err.Error(), uid)
        Fail++
        continue
    }
} else {
    unreadCountSum, err := f.cache.GetUserBadgeUnreadCountSum(ctx, userID)
    if err == nil && unreadCountSum != 0 {
        apns.Payload.Aps.Badge = &unreadCountSum
    } else if errors.Is(err, redis.Nil) || unreadCountSum == 0 {
        zero := 1
        apns.Payload.Aps.Badge = &zero
    } else {
        // log.Error(operationID, "GetUserBadgeUnreadCountSum redis err", err.Error(), uid)
        Fail++
        continue
    }
}
for _, token := range personTokens {
    temp := &messaging.Message{
        Data:         map[string]string{"ex": opts.Ex},
        Token:        token,
        Notification: notification,
        APNS:         apns,
    }
    messages = append(messages, temp)
}
}
messageCount := len(messages)
if messageCount > 0 {
response, err := f.fcmMsgCli.SendEach(ctx, messages)
if err != nil {
    Fail = Fail + messageCount
} else {
    Success = Success + response.SuccessCount
    Fail = Fail + response.FailureCount
}
}
if Fail != 0 {
return errs.New(fmt.Sprintf("%d message send failed;send err:%s;message err:%s",
    Fail, sendErrBuilder.String(), msgErrBuilder.String())).Wrap()
}
return nil
}
```

internal/push/offlinepush/jpush

internal/push/offlinepush/jpush/push.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package jpush

import (
	"context"
	"encoding/base64"
	"fmt"

	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/jpush/body"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/utils/httputil"
)

type JPush struct {
	pushConf   *config.Push
	httpClient *httputil.HTTPClient
}

func NewClient(pushConf *config.Push) *JPush {
	return &JPush{pushConf: pushConf,
		httpClient: httputil.NewHTTPClient(httputil.NewClientConfig()),
	}
}

func (j *JPush) Auth(apiKey, secretKey string, timeStamp int64) (token string, err error) {
	return token, nil
}

func (j *JPush) SetAlias(cid, alias string) (resp string, err error) {
	return resp, nil
}

func (j *JPush) getAuthorization(appKey string, masterSecret string) string {
	str := fmt.Sprintf("%s:%s", appKey, masterSecret)
	buf := []byte(str)
	Authorization := fmt.Sprintf("Basic %s", base64.StdEncoding.EncodeToString(buf))
	return Authorization
}

func (j *JPush) Push(ctx context.Context, userIDs []string, title, content string, opts *options.Opts) error {
	var pf body.Platform
	pf.SetAll()
	var au body.Audience
	au.SetAlias(userIDs)
	var no body.Notification
	extras := make(map[string]string)
	extras["ex"] = opts.Ex
```

```
	if opts.Signal.ClientMsgID != "" {
		extras["ClientMsgID"] = opts.Signal.ClientMsgID
	}
	no.IOSEnableMutableContent()
	no.SetExtras(extras)
	no.SetAlert(content, title, opts)
	no.SetAndroidIntent(j.pushConf)

	var msg body.Message
	msg.SetMsgContent(content)
	msg.SetTitle(title)
	if opts.Signal.ClientMsgID != "" {
		msg.SetExtras("ClientMsgID", opts.Signal.ClientMsgID)
	}
	msg.SetExtras("ex", opts.Ex)
	var opt body.Options
	opt.SetApnsProduction(j.pushConf.IOSPush.Production)
	var pushObj body.PushObj
	pushObj.SetPlatform(&pf)
	pushObj.SetAudience(&au)
	pushObj.SetNotification(&no)
	pushObj.SetMessage(&msg)
	pushObj.SetOptions(&opt)
	var resp map[string]any
	return j.request(ctx, pushObj, &resp, 5)
}

func (j *JPush) request(ctx context.Context, po body.PushObj, resp *map[string]any, timeout int) error {
	err := j.httpClient.PostReturn(
		ctx,
		j.pushConf.JPush.PushURL,
		map[string]string{
			"Authorization": j.getAuthorization(j.pushConf.JPush.AppKey, j.pushConf.JPush.MasterSecret),
		},
		po,
		resp,
		timeout,
	)
	if err != nil {
		return err
	}
	if (*resp)["sendno"] != "0" {
		return fmt.Errorf("jpush push failed %v", resp)
	}
	return nil
}
```

internal/push/offlinepush/jpush/body

internal/push/offlinepush/jpush/body/audience.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package body

const (
	TAG            = "tag"
	TAGAND         = "tag_and"
	TAGNOT         = "tag_not"
	ALIAS          = "alias"
	REGISTRATIONID = "registration_id"
)

type Audience struct {
	Object   any
	audience map[string][]string
}

func (a *Audience) set(key string, v []string) {
	if a.audience == nil {
		a.audience = make(map[string][]string)
		a.Object = a.audience
	}
	// v, ok = this.audience[key]
	// if ok {
	//	return
	//}
	a.audience[key] = v
}

func (a *Audience) SetTag(tags []string) {
	a.set(TAG, tags)
}

func (a *Audience) SetTagAnd(tags []string) {
	a.set(TAGAND, tags)
}

func (a *Audience) SetTagNot(tags []string) {
	a.set(TAGNOT, tags)
}

func (a *Audience) SetAlias(alias []string) {
	a.set(ALIAS, alias)
}

func (a *Audience) SetRegistrationId(ids []string) {
	a.set(REGISTRATIONID, ids)
}
```

```
func (a *Audience) SetAll() {
    a.Object = "all"
}
```

internal/push/offlinepush/jpush/body/message.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package body

type Message struct {
	MsgContent  string         `json:"msg_content"`
	Title       string         `json:"title,omitempty"`
	ContentType string         `json:"content_type,omitempty"`
	Extras      map[string]any `json:"extras,omitempty"`
}

func (m *Message) SetMsgContent(c string) {
	m.MsgContent = c
}

func (m *Message) SetTitle(t string) {
	m.Title = t
}

func (m *Message) SetContentType(c string) {
	m.ContentType = c
}

func (m *Message) SetExtras(key string, value any) {
	if m.Extras == nil {
		m.Extras = make(map[string]any)
	}
	m.Extras[key] = value
}
```

internal/push/offlinepush/jpush/body/notification.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package body

import (
    "github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
)

type Notification struct {
    Alert   string  `json:"alert,omitempty"`
    Android Android `json:"android,omitempty"`
    IOS     Ios     `json:"ios,omitempty"`
}

type Android struct {
    Alert  string `json:"alert,omitempty"`
    Title  string `json:"title,omitempty"`
    Intent struct {
        URL string `json:"url,omitempty"`
    } `json:"intent,omitempty"`
    Extras map[string]string `json:"extras,omitempty"`
}
type Ios struct {
    Alert          IosAlert          `json:"alert,omitempty"`
    Sound          string            `json:"sound,omitempty"`
    Badge          string            `json:"badge,omitempty"`
    Extras         map[string]string `json:"extras,omitempty"`
    MutableContent bool              `json:"mutable-content"`
}

type IosAlert struct {
    Title string `json:"title,omitempty"`
    Body  string `json:"body,omitempty"`
}

func (n *Notification) SetAlert(alert string, title string, opts *options.Opts) {
    n.Alert = alert
    n.Android.Alert = alert
    n.Android.Title = title
    n.IOS.Alert.Body = alert
    n.IOS.Alert.Title = title
    n.IOS.Sound = opts.IOSPushSound
    if opts.IOSBadgeCount {
        n.IOS.Badge = "+1"
    }
}

func (n *Notification) SetExtras(extras map[string]string) {
    n.IOS.Extras = extras
    n.Android.Extras = extras
}
```

```
func (n *Notification) SetAndroidIntent(pushConf *config.Push) {
    n.Android.Intent.URL = pushConf.JPush.PushIntent
}

func (n *Notification) IOSEnableMutableContent() {
    n.IOS.MutableContent = true
}
```

internal/push/offlinepush/jpush/body/options.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package body

type Options struct {
	ApnsProduction bool `json:"apns_production"`
}

func (o *Options) SetApnsProduction(c bool) {
	o.ApnsProduction = c
}
```

internal/push/offlinepush/jpush/body/platform.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package body

import (
	"github.com/openimsdk/tools/errs"

	"github.com/openimsdk/protocol/constant"
)

const (
	ANDROID      = "android"
	IOS          = "ios"
	QUICKAPP     = "quickapp"
	WINDOWSPHONE = "winphone"
	ALL          = "all"
)

type Platform struct {
	Os     any
	osArry []string
}

func (p *Platform) Set(os string) error {
	if p.Os == nil {
		p.osArry = make([]string, 0, 4)
	} else {
		switch p.Os.(type) {
		case string:
			return errs.New("platform is all")
		default:
		}
	}

	for _, value := range p.osArry {
		if os == value {
			return nil
		}
	}

	switch os {
	case IOS:
		fallthrough
	case ANDROID:
		fallthrough
	case QUICKAPP:
		fallthrough
	case WINDOWSPHONE:
		p.osArry = append(p.osArry, os)
		p.Os = p.osArry
	default:
		return errs.New("unknow platform")
```

```
	}

	return nil
}

func (p *Platform) SetPlatform(platform string) error {
	switch platform {
	case constant.AndroidPlatformStr:
		return p.SetAndroid()
	case constant.IOSPlatformStr:
		return p.SetIOS()
	default:
		return errs.New("platform err")
	}
}

func (p *Platform) SetIOS() error {
	return p.Set(IOS)
}

func (p *Platform) SetAndroid() error {
	return p.Set(ANDROID)
}

func (p *Platform) SetQuickApp() error {
	return p.Set(QUICKAPP)
}

func (p *Platform) SetWindowsPhone() error {
	return p.Set(WINDOWSPHONE)
}

func (p *Platform) SetAll() {
	p.Os = ALL
}
```

internal/push/offlinepush/jpush/body/pushobj.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package body

type PushObj struct {
	Platform     any `json:"platform"`
	Audience     any `json:"audience"`
	Notification any `json:"notification,omitempty"`
	Message      any `json:"message,omitempty"`
	Options      any `json:"options,omitempty"`
}

func (p *PushObj) SetPlatform(pf *Platform) {
	p.Platform = pf.Os
}

func (p *PushObj) SetAudience(ad *Audience) {
	p.Audience = ad.Object
}

func (p *PushObj) SetNotification(no *Notification) {
	p.Notification = no
}

func (p *PushObj) SetMessage(m *Message) {
	p.Message = m
}

func (p *PushObj) SetOptions(o *Options) {
	p.Options = o
}
```

internal/push/offlinepush/dummy

internal/push/offlinepush/dummy/push.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package dummy

import (
	"context"
	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/tools/log"
	"sync/atomic"
)

func NewClient() *Dummy {
	return &Dummy{}
}

type Dummy struct {
	v atomic.Bool
}

func (d *Dummy) Push(ctx context.Context, userIDs []string, title, content string, opts *options.Opts) error {
	if d.v.CompareAndSwap(false, true) {
		log.ZWarn(ctx, "dummy push", nil, "ps", "the offline push is not configured. to configure it, please go to config
	}
	return nil
}
```

internal/push/offlinepush/options

internal/push/offlinepush/options/options.go

```go
package options

// Opts opts.
type Opts struct {
	Signal        *Signal
	IOSPushSound  string
	IOSBadgeCount bool
	Ex            string
}

// Signal message id.
type Signal struct {
	ClientMsgID string
}
```

internal/push/offlinepush/getui

internal/push/offlinepush/getui/body.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package getui

import (
	"fmt"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/utils/datautil"
)

var (
	incOne          = datautil.ToPtr("+1")
	addNum          = "1"
	defaultStrategy = strategy{
		Default: 1,
	}
	msgCategory = "CATEGORY_MESSAGE"
)

type Resp struct {
	Code int    `json:"code"`
	Msg  string `json:"msg"`
	Data any    `json:"data"`
}

func (r *Resp) parseError() (err error) {
	switch r.Code {
	case tokenExpireCode:
		err = ErrTokenExpire
	case 0:
		err = nil
	default:
		err = fmt.Errorf("code %d, msg %s", r.Code, r.Msg)
	}
	return err
}

type RespI interface {
	parseError() error
}

type AuthReq struct {
	Sign      string `json:"sign"`
	Timestamp string `json:"timestamp"`
	AppKey    string `json:"appkey"`
}

type AuthResp struct {
```

```
    ExpireTime string `json:"expire_time"`
    Token      string `json:"token"`
}

type TaskResp struct {
    TaskID string `json:"taskID"`
}

type Settings struct {
    TTL      *int64   `json:"ttl"`
    Strategy strategy `json:"strategy"`
}

type strategy struct {
    Default int64 `json:"default"`
    //IOS     int64 `json:"ios"`
    //St      int64 `json:"st"`
    //Hw      int64 `json:"hw"`
    //Ho      int64 `json:"ho"`
    //XM      int64 `json:"xm"`
    //XMG     int64 `json:"xmg"`
    //VV      int64 `json:"vv"`
    //Op      int64 `json:"op"`
    //OpG     int64 `json:"opg"`
    //MZ      int64 `json:"mz"`
    //HosHw   int64 `json:"hoshw"`
    //WX      int64 `json:"wx"`
}

type Audience struct {
    Alias []string `json:"alias"`
}

type PushMessage struct {
    Notification *Notification `json:"notification,omitempty"`
    Transmission *string       `json:"transmission,omitempty"`
}

type PushChannel struct {
    Ios     *Ios     `json:"ios"`
    Android *Android `json:"android"`
}

type PushReq struct {
    RequestID   *string      `json:"request_id"`
    Settings    *Settings    `json:"settings"`
    Audience    *Audience    `json:"audience"`
    PushMessage *PushMessage `json:"push_message"`
    PushChannel *PushChannel `json:"push_channel"`
    IsAsync     *bool        `json:"is_async"`
    TaskID      *string      `json:"taskid"`
}

type Ios struct {
    NotificationType *string `json:"type"`
    AutoBadge        *string `json:"auto_badge"`
    Aps              struct {
        Sound string `json:"sound"`
        Alert Alert  `json:"alert"`
    } `json:"aps"`
}

type Alert struct {
    Title string `json:"title"`
    Body  string `json:"body"`
}
```

```
type Android struct {
	Ups struct {
		Notification Notification `json:"notification"`
		Options      Options      `json:"options"`
	} `json:"ups"`
}

type Notification struct {
	Title       string `json:"title"`
	Body        string `json:"body"`
	ChannelID   string `json:"channelID"`
	ChannelName string `json:"ChannelName"`
	ClickType   string `json:"click_type"`
	BadgeAddNum string `json:"badge_add_num"`
	Category    string `json:"category"`
}

type Options struct {
	HW struct {
		DefaultSound bool   `json:"/message/android/notification/default_sound"`
		ChannelID    string `json:"/message/android/notification/channel_id"`
		Sound        string `json:"/message/android/notification/sound"`
		Importance   string `json:"/message/android/notification/importance"`
		Category     string `json:"/message/android/category"`
	} `json:"HW"`
	XM struct {
		ChannelID string `json:"/extra.channel_id"`
	} `json:"XM"`
	VV struct {
		Classification int `json:"/classification"`
	} `json:"VV"`
}

type Payload struct {
	IsSignal bool `json:"isSignal"`
}

func newPushReq(pushConf *config.Push, title, content string) PushReq {
	pushReq := PushReq{PushMessage: &PushMessage{Notification: &Notification{
		Title:       title,
		Body:        content,
		ClickType:   "startapp",
		ChannelID:   pushConf.GeTui.ChannelID,
		ChannelName: pushConf.GeTui.ChannelName,
		BadgeAddNum: addNum,
		Category:    msgCategory,
	}}}
	return pushReq
}

func newBatchPushReq(userIDs []string, taskID string) PushReq {
	IsAsync := true
	return PushReq{Audience: &Audience{Alias: userIDs}, IsAsync: &IsAsync, TaskID: &taskID}
}

func (pushReq *PushReq) setPushChannel(title string, body string) {
	pushReq.PushChannel = &PushChannel{}
	// autoBadge := "+1"
	pushReq.PushChannel.Ios = &Ios{}
	notify := "notify"
	pushReq.PushChannel.Ios.NotificationType = &notify
	pushReq.PushChannel.Ios.Aps.Sound = "default"
	pushReq.PushChannel.Ios.AutoBadge = incOne
	pushReq.PushChannel.Ios.Aps.Alert = Alert{
		Title: title,
```

```
    Body:  body,
  }
  pushReq.PushChannel.Android = &Android{}
  pushReq.PushChannel.Android.Ups.Notification = Notification{
    Title:     title,
    Body:      body,
    ClickType: "startapp",
  }
  pushReq.PushChannel.Android.Ups.Options = Options{
    HW: struct {
      DefaultSound bool   `json:"/message/android/notification/default_sound"`
      ChannelID    string `json:"/message/android/notification/channel_id"`
      Sound        string `json:"/message/android/notification/sound"`
      Importance   string `json:"/message/android/notification/importance"`
      Category     string `json:"/message/android/category"`
    }{ChannelID: "RingRing4", Sound: "/raw/ring001", Importance: "NORMAL", Category: "IM"},
    XM: struct {
      ChannelID string `json:"/extra.channel_id"`
    }{ChannelID: "high_system"},
    VV: struct {
      Classification int "json:\"/classification\""
    }{
      Classification: 1,
    },
  }
}
```

internal/push/offlinepush/getui/push.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package getui

import (
	"context"
	"crypto/sha256"
	"encoding/hex"
	"errors"
	"strconv"
	"sync"
	"time"

	"github.com/openimsdk/open-im-server/v3/internal/push/offlinepush/options"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/httputil"
	"github.com/openimsdk/tools/utils/splitter"
	"github.com/redis/go-redis/v9"
)

var (
	ErrTokenExpire = errs.New("token expire")
	ErrUserIDEmpty = errs.New("userIDs is empty")
)

const (
	pushURL      = "/push/single/alias"
	authURL      = "/auth"
	taskURL      = "/push/list/message"
	batchPushURL = "/push/list/alias"

	// Codes.
	tokenExpireCode = 10001
	tokenExpireTime = 60 * 60 * 23
	taskIDTTL       = 1000 * 60 * 60 * 24
)

type Client struct {
	cache           cache.ThirdCache
	tokenExpireTime int64
	taskIDTTL       int64
	pushConf        *config.Push
	httpClient      *httputil.HTTPClient
}

func NewClient(pushConf *config.Push, cache cache.ThirdCache) *Client {
	return &Client{cache: cache,
		tokenExpireTime: tokenExpireTime,
```

```
    taskIDTTL:       taskIDTTL,
    pushConf:        pushConf,
    httpClient:      httputil.NewHTTPClient(httputil.NewClientConfig()),
  }
}

func (g *Client) Push(ctx context.Context, userIDs []string, title, content string, opts *options.Opts) error {
  token, err := g.cache.GetGetuiToken(ctx)
  if err != nil {
    if errors.Is(err, redis.Nil) {
      log.ZDebug(ctx, "getui token not exist in redis")
      token, err = g.getTokenAndSave2Redis(ctx)
      if err != nil {
        return err
      }
    } else {
      return err
    }
  }
  pushReq := newPushReq(g.pushConf, title, content)
  pushReq.setPushChannel(title, content)
  if len(userIDs) > 1 {
    maxNum := 999
    if len(userIDs) > maxNum {
      s := splitter.NewSplitter(maxNum, userIDs)
      wg := sync.WaitGroup{}
      wg.Add(len(s.GetSplitResult()))
      for i, v := range s.GetSplitResult() {
        go func(index int, userIDs []string) {
          defer wg.Done()
          for i := 0; i < len(userIDs); i += maxNum {
            end := i + maxNum
            if end > len(userIDs) {
              end = len(userIDs)
            }
            if err = g.batchPush(ctx, token, userIDs[i:end], pushReq); err != nil {
              log.ZError(ctx, "batchPush failed", err, "index", index, "token", token, "req", pushReq)
            }
          }
          if err = g.batchPush(ctx, token, userIDs, pushReq); err != nil {
            log.ZError(ctx, "batchPush failed", err, "index", index, "token", token, "req", pushReq)
          }
        }(i, v.Item)
      }
      wg.Wait()
    } else {
      err = g.batchPush(ctx, token, userIDs, pushReq)
    }
  } else if len(userIDs) == 1 {
    err = g.singlePush(ctx, token, userIDs[0], pushReq)
  } else {
    return ErrUserIDEmpty
  }
  switch err {
  case ErrTokenExpire:
    token, err = g.getTokenAndSave2Redis(ctx)
  }
  return err
}

func (g *Client) Auth(ctx context.Context, timeStamp int64) (token string, expireTime int64, err error) {
  h := sha256.New()
  h.Write(
    []byte(g.pushConf.GeTui.AppKey + strconv.Itoa(int(timeStamp)) + g.pushConf.GeTui.MasterSecret),
  )
  sign := hex.EncodeToString(h.Sum(nil))
```

```
	reqAuth := AuthReq{
		Sign:      sign,
		Timestamp: strconv.Itoa(int(timeStamp)),
		AppKey:    g.pushConf.GeTui.AppKey,
	}
	respAuth := AuthResp{}
	err = g.request(ctx, authURL, reqAuth, "", &respAuth)
	if err != nil {
		return "", 0, err
	}
	expire, err := strconv.Atoi(respAuth.ExpireTime)
	return respAuth.Token, int64(expire), err
}

func (g *Client) GetTaskID(ctx context.Context, token string, pushReq PushReq) (string, error) {
	respTask := TaskResp{}
	ttl := int64(1000 * 60 * 5)
	pushReq.Settings = &Settings{TTL: &ttl, Strategy: defaultStrategy}
	err := g.request(ctx, taskURL, pushReq, token, &respTask)
	if err != nil {
		return "", errs.Wrap(err)
	}
	return respTask.TaskID, nil
}

// max num is 999.
func (g *Client) batchPush(ctx context.Context, token string, userIDs []string, pushReq PushReq) error {
	taskID, err := g.GetTaskID(ctx, token, pushReq)
	if err != nil {
		return err
	}
	pushReq = newBatchPushReq(userIDs, taskID)
	return g.request(ctx, batchPushURL, pushReq, token, nil)
}

func (g *Client) singlePush(ctx context.Context, token, userID string, pushReq PushReq) error {
	operationID := mcontext.GetOperationID(ctx)
	pushReq.RequestID = &operationID
	pushReq.Audience = &Audience{Alias: []string{userID}}
	return g.request(ctx, pushURL, pushReq, token, nil)
}

func (g *Client) request(ctx context.Context, url string, input any, token string, output any) error {
	header := map[string]string{"token": token}
	resp := &Resp{}
	resp.Data = output
	return g.postReturn(ctx, g.pushConf.GeTui.PushUrl+url, header, input, resp, 3)
}

func (g *Client) postReturn(
	ctx context.Context,
	url string,
	header map[string]string,
	input any,
	output RespI,
	timeout int,
) error {
	err := g.httpClient.PostReturn(ctx, url, header, input, output, timeout)
	if err != nil {
		return err
	}
	log.ZDebug(ctx, "postReturn", "url", url, "header", header, "input", input, "timeout", timeout, "output", output)
	return output.parseError()
}

func (g *Client) getTokenAndSave2Redis(ctx context.Context) (token string, err error) {
```

```
	token, _, err = g.Auth(ctx, time.Now().UnixNano()/1e6)
	if err != nil {
		return
	}
	err = g.cache.SetGetuiToken(ctx, token, 60*60*23)
	if err != nil {
		return
	}
	return token, nil
}

func (g *Client) GetTaskIDAndSave2Redis(ctx context.Context, token string, pushReq PushReq) (taskID string, err erro
	pushReq.Settings = &Settings{TTL: &g.taskIDTTL, Strategy: defaultStrategy}
	taskID, err = g.GetTaskID(ctx, token, pushReq)
	if err != nil {
		return
	}
	err = g.cache.SetGetuiTaskID(ctx, taskID, g.tokenExpireTime)
	if err != nil {
		return
	}
	return token, nil
}
```

internal/tools

internal/tools/cron

internal/tools/cron/cron_task.go

```
package cron

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	disetcd "github.com/openimsdk/open-im-server/v3/pkg/common/discovery/etcd"
	pbconversation "github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/discovery/etcd"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/runtimeenv"
	"github.com/robfig/cron/v3"
	"google.golang.org/grpc"
)

type Config struct {
	CronTask  config.CronTask
	Share     config.Share
	Discovery config.Discovery
}

func Start(ctx context.Context, conf *Config, client discovery.SvcDiscoveryRegistry, service grpc.ServiceRegistrar)
	log.CInfo(ctx, "CRON-TASK server is initializing", "runTimeEnv", runtimeenv.RuntimeEnvironment(), "chatRecordsClear
	if conf.CronTask.RetainChatRecords < 1 {
		log.ZInfo(ctx, "disable cron")
		<-ctx.Done()
		return nil
	}
	ctx = mcontext.SetOpUserID(ctx, conf.Share.IMAdminUser.UserIDs[0])

	msgConn, err := client.GetConn(ctx, conf.Discovery.RpcService.Msg)
	if err != nil {
		return err
	}

	thirdConn, err := client.GetConn(ctx, conf.Discovery.RpcService.Third)
	if err != nil {
		return err
	}

	conversationConn, err := client.GetConn(ctx, conf.Discovery.RpcService.Conversation)
	if err != nil {
		return err
	}

	var locker Locker
	if conf.Discovery.Enable == config.ETCD {
		cm := disetcd.NewConfigManager(client.(*etcd.SvcDiscoveryRegistryImpl).GetClient(), []string{
			conf.CronTask.GetConfigFileName(),
			conf.Share.GetConfigFileName(),
			conf.Discovery.GetConfigFileName(),
		})
		cm.Watch(ctx)
```

```
    locker, err = NewEtcdLocker(client.(*etcd.SvcDiscoveryRegistryImpl).GetClient())
    if err != nil {
        return err
    }
}

if locker == nil {
    locker = emptyLocker{}
}

srv := &cronServer{
    ctx:                ctx,
    config:             conf,
    cron:               cron.New(),
    msgClient:          msg.NewMsgClient(msgConn),
    conversationClient: pbconversation.NewConversationClient(conversationConn),
    thirdClient:        third.NewThirdClient(thirdConn),
    locker:             locker,
}

if err := srv.registerClearS3(); err != nil {
    return err
}
if err := srv.registerDeleteMsg(); err != nil {
    return err
}
if err := srv.registerClearUserMsg(); err != nil {
    return err
}
log.ZDebug(ctx, "start cron task", "CronExecuteTime", conf.CronTask.CronExecuteTime)
srv.cron.Start()
log.ZDebug(ctx, "cron task server is running")
<-ctx.Done()
log.ZDebug(ctx, "cron task server is shutting down")
srv.cron.Stop()

return nil
}

type Locker interface {
ExecuteWithLock(ctx context.Context, taskName string, task func())
}

type emptyLocker struct{}

func (emptyLocker) ExecuteWithLock(ctx context.Context, taskName string, task func()) {
task()
}

type cronServer struct {
ctx                context.Context
config             *Config
cron               *cron.Cron
msgClient          msg.MsgClient
conversationClient pbconversation.ConversationClient
thirdClient        third.ThirdClient
locker             Locker
}

func (c *cronServer) registerClearS3() error {
if c.config.CronTask.FileExpireTime <= 0 || len(c.config.CronTask.DeleteObjectType) == 0 {
    log.ZInfo(c.ctx, "disable scheduled cleanup of s3", "fileExpireTime", c.config.CronTask.FileExpireTime, "deleteOb
    return nil
}
_, err := c.cron.AddFunc(c.config.CronTask.CronExecuteTime, func() {
    c.locker.ExecuteWithLock(c.ctx, "clearS3", c.clearS3)
```

```
	})
	return errs.WrapMsg(err, "failed to register clear s3 cron task")
}

func (c *cronServer) registerDeleteMsg() error {
	if c.config.CronTask.RetainChatRecords <= 0 {
		log.ZInfo(c.ctx, "disable scheduled cleanup of chat records", "retainChatRecords", c.config.CronTask.RetainChatRe
		return nil
	}
	_, err := c.cron.AddFunc(c.config.CronTask.CronExecuteTime, func() {
		c.locker.ExecuteWithLock(c.ctx, "deleteMsg", c.deleteMsg)
	})
	return errs.WrapMsg(err, "failed to register delete msg cron task")
}

func (c *cronServer) registerClearUserMsg() error {
	_, err := c.cron.AddFunc(c.config.CronTask.CronExecuteTime, func() {
		c.locker.ExecuteWithLock(c.ctx, "clearUserMsg", c.clearUserMsg)
	})
	return errs.WrapMsg(err, "failed to register clear user msg cron task")
}
```

internal/tools/cron/cron_test.go

```
package cron

import (
	"context"
	"testing"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	kdisc "github.com/openimsdk/open-im-server/v3/pkg/common/discovery"
	pbconversation "github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/mw"
	"github.com/robfig/cron/v3"
	"google.golang.org/grpc"
	"google.golang.org/grpc/credentials/insecure"
)

func TestName(t *testing.T) {
	conf := &config.Discovery{
		Enable: config.ETCD,
		Etcd: config.Etcd{
			RootDirectory: "openim",
			Address:       []string{"localhost:12379"},
		},
	}
	client, err := kdisc.NewDiscoveryRegister(conf, nil)
	if err != nil {
		panic(err)
	}
	client.AddOption(mw.GrpcClient(), grpc.WithTransportCredentials(insecure.NewCredentials()))
	ctx := mcontext.SetOpUserID(context.Background(), "imAdmin")
	msgConn, err := client.GetConn(ctx, "msg-rpc-service")
	if err != nil {
		panic(err)
	}
	thirdConn, err := client.GetConn(ctx, "third-rpc-service")
	if err != nil {
		panic(err)
	}

	conversationConn, err := client.GetConn(ctx, "conversation-rpc-service")
	if err != nil {
		panic(err)
	}

	srv := &cronServer{
		ctx: ctx,
		config: &Config{
			CronTask: config.CronTask{
				RetainChatRecords: 1,
				FileExpireTime:    1,
				DeleteObjectType:  []string{"msg-picture", "msg-file", "msg-voice", "msg-video", "msg-video-snapshot", "sdklog"
			},
		},
		cron:               cron.New(),
		msgClient:          msg.NewMsgClient(msgConn),
		conversationClient: pbconversation.NewConversationClient(conversationConn),
		thirdClient:        third.NewThirdClient(thirdConn),
	}
	srv.deleteMsg()
	//srv.clearS3()
	//srv.clearUserMsg()
}
```

internal/tools/cron/dist_look.go

```
package cron

import (
	"context"
	"fmt"
	"os"
	"time"

	"github.com/openimsdk/tools/log"
	clientv3 "go.etcd.io/etcd/client/v3"
	"go.etcd.io/etcd/client/v3/concurrency"
)

const (
	lockLeaseTTL = 300
)

type EtcdLocker struct {
	client     *clientv3.Client
	instanceID string
}

// NewEtcdLocker creates a new etcd distributed lock
func NewEtcdLocker(client *clientv3.Client) (*EtcdLocker, error) {
	hostname, _ := os.Hostname()
	pid := os.Getpid()
	instanceID := fmt.Sprintf("%s-pid-%d-%d", hostname, pid, time.Now().UnixNano())

	locker := &EtcdLocker{
		client:     client,
		instanceID: instanceID,
	}

	return locker, nil
}

func (e *EtcdLocker) ExecuteWithLock(ctx context.Context, taskName string, task func()) {
	session, err := concurrency.NewSession(e.client, concurrency.WithTTL(lockLeaseTTL))
	if err != nil {
		log.ZWarn(ctx, "Failed to create etcd session", err,
			"taskName", taskName,
			"instanceID", e.instanceID)
		return
	}
	defer session.Close()

	lockKey := fmt.Sprintf("openim/crontask/%s", taskName)
	mutex := concurrency.NewMutex(session, lockKey)

	ctxWithTimeout, cancel := context.WithTimeout(ctx, 100*time.Millisecond)
	defer cancel()

	err = mutex.TryLock(ctxWithTimeout)
	if err != nil {
		// errors.Is(err, concurrency.ErrLocked)
		log.ZDebug(ctx, "Task is being executed by another instance, skipping",
			"taskName", taskName,
			"instanceID", e.instanceID,
			"error", err.Error())

		return
	}

	defer func() {
```

```
    if err := mutex.Unlock(ctx); err != nil {
        log.ZWarn(ctx, "Failed to release task lock", err,
            "taskName", taskName,
            "instanceID", e.instanceID)
    } else {
        log.ZInfo(ctx, "Successfully released task lock",
            "taskName", taskName,
            "instanceID", e.instanceID)
    }
  }()

  log.ZInfo(ctx, "Successfully acquired task lock, starting execution",
    "taskName", taskName,
    "instanceID", e.instanceID,
    "sessionID", session.Lease())

  task()

  log.ZInfo(ctx, "Task execution completed",
    "taskName", taskName,
    "instanceID", e.instanceID)
}
```

internal/tools/cron/msg.go

```
package cron

import (
	"fmt"
	"os"
	"time"

	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
)

func (c *cronServer) deleteMsg() {
	now := time.Now()
	deltime := now.Add(-time.Hour * 24 * time.Duration(c.config.CronTask.RetainChatRecords))
	operationID := fmt.Sprintf("cron_msg_%d_%d", os.Getpid(), deltime.UnixMilli())
	ctx := mcontext.SetOperationID(c.ctx, operationID)
	log.ZDebug(ctx, "Destruct chat records", "deltime", deltime, "timestamp", deltime.UnixMilli())
	const (
		deleteCount = 10000
		deleteLimit = 50
	)
	var count int
	for i := 1; i <= deleteCount; i++ {
		ctx := mcontext.SetOperationID(c.ctx, fmt.Sprintf("%s_%d", operationID, i))
		resp, err := c.msgClient.DestructMsgs(ctx, &msg.DestructMsgsReq{Timestamp: deltime.UnixMilli(), Limit: deleteLimi
		if err != nil {
			log.ZError(ctx, "cron destruct chat records failed", err)
			break
		}
		count += int(resp.Count)
		if resp.Count < deleteLimit {
			break
		}
	}
	log.ZDebug(ctx, "cron destruct chat records end", "deltime", deltime, "cont", time.Since(now), "count", count)
}
```

internal/tools/cron/s3.go

```
package cron

import (
	"fmt"
	"os"
	"time"

	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
)

func (c *cronServer) clearS3() {
	start := time.Now()
	deleteTime := start.Add(-time.Hour * 24 * time.Duration(c.config.CronTask.FileExpireTime))
	operationID := fmt.Sprintf("cron_s3_%d_%d", os.Getpid(), deleteTime.UnixMilli())
	ctx := mcontext.SetOperationID(c.ctx, operationID)
	log.ZDebug(ctx, "deleteoutDatedData", "deletetime", deleteTime, "timestamp", deleteTime.UnixMilli())
	const (
		deleteCount = 10000
		deleteLimit = 100
	)

	var count int
	for i := 1; i <= deleteCount; i++ {
		resp, err := c.thirdClient.DeleteOutdatedData(ctx, &third.DeleteOutdatedDataReq{ExpireTime: deleteTime.UnixMilli(
		if err != nil {
			log.ZError(ctx, "cron deleteoutDatedData failed", err)
			return
		}
		count += int(resp.Count)
		if resp.Count < deleteLimit {
			break
		}
	}
	log.ZDebug(ctx, "cron deleteoutDatedData success", "deltime", deleteTime, "cont", time.Since(start), "count", count
}

//	var req *third.DeleteOutdatedDataReq
//	count1, err := ExtractField(ctx, c.thirdClient.DeleteOutdatedData, req, (*third.DeleteOutdatedDataResp).GetCount)
//
//	c.thirdClient.DeleteOutdatedData(ctx, &third.DeleteOutdatedDataReq{})
//	msggateway.GetUsersOnlineStatusCaller.Invoke(ctx, &msggateway.GetUsersOnlineStatusReq{})
//
//	var cli ThirdClient
//
//	c111, err := cli.DeleteOutdatedData(ctx, 100)
//
//	cli.ThirdClient.DeleteOutdatedData(ctx, &third.DeleteOutdatedDataReq{})
//
//	cli.AuthSign(ctx, &third.AuthSignReq{})
//
//	cli.SetAppBadge()
//
//}
//
//func extractField[A, B, C any](ctx context.Context, fn func(ctx context.Context, req *A, opts ...grpc.CallOption)
//	resp, err := fn(ctx, req)
//	if err != nil {
//		var c C
//		return c, err
//	}
//	return get(resp), nil
//}
```

```
//
//func ignore(_ any, err error) error {
//■return err
//}
//
//type ThirdClient struct {
//■third.ThirdClient
//}
//
//func (c *ThirdClient) DeleteOutdatedData(ctx context.Context, expireTime int64) (int32, error) {
//■return extractField(ctx, c.ThirdClient.DeleteOutdatedData, &third.DeleteOutdatedDataReq{ExpireTime: expireTime},
//}
//
//func (c *ThirdClient) DeleteOutdatedData1(ctx context.Context, expireTime int64) error {
//■return ignore(c.ThirdClient.DeleteOutdatedData(ctx, &third.DeleteOutdatedDataReq{ExpireTime: expireTime}))
//}
```

internal/tools/cron/user_msg.go

```
package cron

import (
	"fmt"
	"os"
	"time"

	pbconversation "github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
)

func (c *cronServer) clearUserMsg() {
	now := time.Now()
	operationID := fmt.Sprintf("cron_user_msg_%d_%d", os.Getpid(), now.UnixMilli())
	ctx := mcontext.SetOperationID(c.ctx, operationID)
	log.ZDebug(ctx, "clear user msg cron start")
	const (
		deleteCount = 10000
		deleteLimit = 100
	)
	var count int
	for i := 1; i <= deleteCount; i++ {
		resp, err := c.conversationClient.ClearUserConversationMsg(ctx, &pbconversation.ClearUserConversationMsgReq{Times
		if err != nil {
			log.ZError(ctx, "ClearUserConversationMsg failed.", err)
			return
		}
		count += int(resp.Count)
		if resp.Count < deleteLimit {
			break
		}
	}
	log.ZDebug(ctx, "clear user msg cron task completed", "cont", time.Since(now), "count", count)
}
```

internal/rpc

internal/rpc/third

internal/rpc/third/log.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package third

import (
	"context"
	"crypto/rand"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
)

func genLogID() string {
	const dataLen = 10
	data := make([]byte, dataLen)
	rand.Read(data)
	chars := []byte("0123456789")
	for i := 0; i < len(data); i++ {
		if i == 0 {
			data[i] = chars[1:][data[i]%9]
		} else {
			data[i] = chars[data[i]%10]
		}
	}
	return string(data)
}

func (t *thirdServer) UploadLogs(ctx context.Context, req *third.UploadLogsReq) (*third.UploadLogsResp, error) {
	var dbLogs []*relationtb.Log
	userID := mcontext.GetOpUserID(ctx)
	platform := constant.PlatformID2Name[int(req.Platform)]
	for _, fileURL := range req.FileURLs {
		log := relationtb.Log{
			Platform:     platform,
			UserID:       userID,
			CreateTime:   time.Now(),
			Url:          fileURL.URL,
			FileName:     fileURL.Filename,
			AppFramework: req.AppFramework,
```

```
        Version:      req.Version,
        Ex:           req.Ex,
    }
    for i := 0; i < 20; i++ {
        id := genLogID()
        logs, err := t.thirdDatabase.GetLogs(ctx, []string{id}, "")
        if err != nil {
            return nil, err
        }
        if len(logs) == 0 {
            log.LogID = id
            break
        }
    }
    if log.LogID == "" {
        return nil, servererrs.ErrData.WrapMsg("Log id gen error")
    }
    dbLogs = append(dbLogs, &log)
  }
  err := t.thirdDatabase.UploadLogs(ctx, dbLogs)
  if err != nil {
    return nil, err
  }
  return &third.UploadLogsResp{}, nil
}

func (t *thirdServer) DeleteLogs(ctx context.Context, req *third.DeleteLogsReq) (*third.DeleteLogsResp, error) {
  if err := authverify.CheckAdmin(ctx); err != nil {
    return nil, err
  }
  userID := ""
  logs, err := t.thirdDatabase.GetLogs(ctx, req.LogIDs, userID)
  if err != nil {
    return nil, err
  }
  var logIDs []string
  for _, log := range logs {
    logIDs = append(logIDs, log.LogID)
  }
  if ids := datautil.Single(req.LogIDs, logIDs); len(ids) > 0 {
    return nil, errs.ErrRecordNotFound.WrapMsg("logIDs not found", "logIDs", ids)
  }
  err = t.thirdDatabase.DeleteLogs(ctx, req.LogIDs, userID)
  if err != nil {
    return nil, err
  }

  return &third.DeleteLogsResp{}, nil
}

func dbToPbLogInfos(logs []*relationtb.Log) []*third.LogInfo {
  db2pbForLogInfo := func(log *relationtb.Log) *third.LogInfo {
    return &third.LogInfo{
        Filename:   log.FileName,
        UserID:     log.UserID,
        Platform:   log.Platform,
        Url:        log.Url,
        CreateTime: log.CreateTime.UnixMilli(),
        LogID:      log.LogID,
        SystemType: log.SystemType,
        Version:    log.Version,
        Ex:         log.Ex,
    }
  }
  return datautil.Slice(logs, db2pbForLogInfo)
}
```

```go
func (t *thirdServer) SearchLogs(ctx context.Context, req *third.SearchLogsReq) (*third.SearchLogsResp, error) {
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	var (
		resp    third.SearchLogsResp
		userIDs []string
	)
	if req.StartTime > req.EndTime {
		return nil, errs.ErrArgs.WrapMsg("startTime>endTime")
	}
	if req.StartTime == 0 && req.EndTime == 0 {
		t := time.Date(2019, time.January, 1, 0, 0, 0, 0, time.UTC)
		timestampMills := t.UnixNano() / int64(time.Millisecond)
		req.StartTime = timestampMills
		req.EndTime = time.Now().UnixNano() / int64(time.Millisecond)
	}

	total, logs, err := t.thirdDatabase.SearchLogs(ctx, req.Keyword, time.UnixMilli(req.StartTime), time.UnixMilli(req.
	if err != nil {
		return nil, err
	}
	pbLogs := dbToPbLogInfos(logs)
	for _, log := range logs {
		userIDs = append(userIDs, log.UserID)
	}
	userMap, err := t.userClient.GetUsersInfoMap(ctx, userIDs)
	if err != nil {
		return nil, err
	}
	for _, pbLog := range pbLogs {
		if user, ok := userMap[pbLog.UserID]; ok {
			pbLog.Nickname = user.Nickname
		}
	}
	resp.LogsInfos = pbLogs
	resp.Total = uint32(total)
	return &resp, nil
}
```

internal/rpc/third/s3.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package third

import (
    "context"
    "encoding/base64"
    "encoding/hex"
    "encoding/json"
    "path"
    "strconv"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"

    "github.com/google/uuid"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/protocol/third"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/s3"
    "github.com/openimsdk/tools/s3/cont"
    "github.com/openimsdk/tools/utils/datautil"
)

func (t *thirdServer) PartLimit(ctx context.Context, req *third.PartLimitReq) (*third.PartLimitResp, error) {
    limit, err := t.s3dataBase.PartLimit()
    if err != nil {
        return nil, err
    }
    return &third.PartLimitResp{
        MinPartSize: limit.MinPartSize,
        MaxPartSize: limit.MaxPartSize,
        MaxNumSize:  int32(limit.MaxNumSize),
    }, nil
}

func (t *thirdServer) PartSize(ctx context.Context, req *third.PartSizeReq) (*third.PartSizeResp, error) {
    size, err := t.s3dataBase.PartSize(ctx, req.Size)
    if err != nil {
        return nil, err
    }
    return &third.PartSizeResp{Size: size}, nil
}

func (t *thirdServer) InitiateMultipartUpload(ctx context.Context, req *third.InitiateMultipartUploadReq) (*third.In
    if err := t.checkUploadName(ctx, req.Name); err != nil {
        return nil, err
    }
    expireTime := time.Now().Add(t.defaultExpire)
```

```
	result, err := t.s3dataBase.InitiateMultipartUpload(ctx, req.Hash, req.Size, t.defaultExpire, int(req.MaxParts), re
	if err != nil {
		if haErr, ok := errs.Unwrap(err).(*cont.HashAlreadyExistsError); ok {
			obj := &model.Object{
				Name:        req.Name,
				UserID:      mcontext.GetOpUserID(ctx),
				Hash:        req.Hash,
				Key:         haErr.Object.Key,
				Size:        haErr.Object.Size,
				ContentType: req.ContentType,
				Group:       req.Cause,
				CreateTime:  time.Now(),
			}
			if err := t.s3dataBase.SetObject(ctx, obj); err != nil {
				return nil, err
			}
			return &third.InitiateMultipartUploadResp{
				Url: t.apiAddress(req.UrlPrefix, obj.Name),
			}, nil
		}
		return nil, err
	}
	var sign *third.AuthSignParts
	if result.Sign != nil && len(result.Sign.Parts) > 0 {
		sign = &third.AuthSignParts{
			Url:    result.Sign.URL,
			Query:  toPbMapArray(result.Sign.Query),
			Header: toPbMapArray(result.Sign.Header),
			Parts:  make([]*third.SignPart, len(result.Sign.Parts)),
		}
		for i, part := range result.Sign.Parts {
			sign.Parts[i] = &third.SignPart{
				PartNumber: int32(part.PartNumber),
				Url:        part.URL,
				Query:      toPbMapArray(part.Query),
				Header:     toPbMapArray(part.Header),
			}
		}
	}
	return &third.InitiateMultipartUploadResp{
		Upload: &third.UploadInfo{
			UploadID:   result.UploadID,
			PartSize:   result.PartSize,
			Sign:       sign,
			ExpireTime: expireTime.UnixMilli(),
		},
	}, nil
}

func (t *thirdServer) AuthSign(ctx context.Context, req *third.AuthSignReq) (*third.AuthSignResp, error) {
	partNumbers := datautil.Slice(req.PartNumbers, func(partNumber int32) int { return int(partNumber) })
	result, err := t.s3dataBase.AuthSign(ctx, req.UploadID, partNumbers)
	if err != nil {
		return nil, err
	}
	resp := &third.AuthSignResp{
		Url:    result.URL,
		Query:  toPbMapArray(result.Query),
		Header: toPbMapArray(result.Header),
		Parts:  make([]*third.SignPart, len(result.Parts)),
	}
	for i, part := range result.Parts {
		resp.Parts[i] = &third.SignPart{
			PartNumber: int32(part.PartNumber),
			Url:        part.URL,
			Query:      toPbMapArray(part.Query),
```

```
		Header:     toPbMapArray(part.Header),
	}
	}
	return resp, nil
}

func (t *thirdServer) CompleteMultipartUpload(ctx context.Context, req *third.CompleteMultipartUploadReq) (*third.Co
	if err := t.checkUploadName(ctx, req.Name); err != nil {
		return nil, err
	}
	result, err := t.s3dataBase.CompleteMultipartUpload(ctx, req.UploadID, req.Parts)
	if err != nil {
		return nil, err
	}
	obj := &model.Object{
		Name:        req.Name,
		UserID:      mcontext.GetOpUserID(ctx),
		Hash:        result.Hash,
		Key:         result.Key,
		Size:        result.Size,
		ContentType: req.ContentType,
		Group:       req.Cause,
		CreateTime:  time.Now(),
	}
	if err := t.s3dataBase.SetObject(ctx, obj); err != nil {
		return nil, err
	}
	return &third.CompleteMultipartUploadResp{
		Url: t.apiAddress(req.UrlPrefix, obj.Name),
	}, nil
}

func (t *thirdServer) AccessURL(ctx context.Context, req *third.AccessURLReq) (*third.AccessURLResp, error) {
	opt := &s3.AccessURLOption{}
	if len(req.Query) > 0 {
		switch req.Query["type"] {
		case "":
		case "image":
			opt.Image = &s3.Image{}
			opt.Image.Format = req.Query["format"]
			opt.Image.Width, _ = strconv.Atoi(req.Query["width"])
			opt.Image.Height, _ = strconv.Atoi(req.Query["height"])
			log.ZDebug(ctx, "AccessURL image", "name", req.Name, "option", opt.Image)
		default:
			return nil, errs.ErrArgs.WrapMsg("invalid query type")
		}
	}
	expireTime, rawURL, err := t.s3dataBase.AccessURL(ctx, req.Name, t.defaultExpire, opt)
	if err != nil {
		return nil, err
	}
	return &third.AccessURLResp{
		Url:        rawURL,
		ExpireTime: expireTime.UnixMilli(),
	}, nil
}

func (t *thirdServer) InitiateFormData(ctx context.Context, req *third.InitiateFormDataReq) (*third.InitiateFormData
	if req.Name == "" {
		return nil, errs.ErrArgs.WrapMsg("name is empty")
	}
	if req.Size <= 0 {
		return nil, errs.ErrArgs.WrapMsg("size must be greater than 0")
	}
	if err := t.checkUploadName(ctx, req.Name); err != nil {
		return nil, err
```

```
	}
	var duration time.Duration
	opUserID := mcontext.GetOpUserID(ctx)
	var key string
	if authverify.CheckUserIsAdmin(ctx, opUserID) {
		if req.Millisecond <= 0 {
			duration = time.Minute * 10
		} else {
			duration = time.Millisecond * time.Duration(req.Millisecond)
		}
		if req.Absolute {
			key = req.Name
		}
	} else {
		duration = time.Minute * 10
	}
	uid, err := uuid.NewRandom()
	if err != nil {
		return nil, errs.WrapMsg(err, "uuid NewRandom failed")
	}
	if key == "" {
		date := time.Now().Format("20060102")
		key = path.Join(cont.DirectPath, date, opUserID, hex.EncodeToString(uid[:])+path.Ext(req.Name))
	}
	mate := FormDataMate{
		Name:        req.Name,
		Size:        req.Size,
		ContentType: req.ContentType,
		Group:       req.Group,
		Key:         key,
	}
	mateData, err := json.Marshal(&mate)
	if err != nil {
		return nil, errs.WrapMsg(err, "marshal failed")
	}
	resp, err := t.s3dataBase.FormData(ctx, key, req.Size, req.ContentType, duration)
	if err != nil {
		return nil, err
	}
	return &third.InitiateFormDataResp{
		Id:       base64.RawStdEncoding.EncodeToString(mateData),
		Url:      resp.URL,
		File:     resp.File,
		Header:   toPbMapArray(resp.Header),
		FormData: resp.FormData,
		Expires:  resp.Expires.UnixMilli(),
		SuccessCodes: datautil.Slice(resp.SuccessCodes, func(code int) int32 {
			return int32(code)
		}),
	}, nil
}

func (t *thirdServer) CompleteFormData(ctx context.Context, req *third.CompleteFormDataReq) (*third.CompleteFormData
	if req.Id == "" {
		return nil, errs.ErrArgs.WrapMsg("id is empty")
	}
	data, err := base64.RawStdEncoding.DecodeString(req.Id)
	if err != nil {
		return nil, errs.ErrArgs.WrapMsg("invalid id " + err.Error())
	}
	var mate FormDataMate
	if err := json.Unmarshal(data, &mate); err != nil {
		return nil, errs.ErrArgs.WrapMsg("invalid id " + err.Error())
	}
	if err := t.checkUploadName(ctx, mate.Name); err != nil {
		return nil, err
```

```
	}
	info, err := t.s3dataBase.StatObject(ctx, mate.Key)
	if err != nil {
		return nil, err
	}
	if info.Size > 0 && info.Size != mate.Size {
		return nil, servererrs.ErrData.WrapMsg("file size mismatch")
	}
	obj := &model.Object{
		Name:        mate.Name,
		UserID:      mcontext.GetOpUserID(ctx),
		Hash:        "etag_" + info.ETag,
		Key:         info.Key,
		Size:        info.Size,
		ContentType: mate.ContentType,
		Group:       mate.Group,
		CreateTime:  time.Now(),
	}
	if err := t.s3dataBase.SetObject(ctx, obj); err != nil {
		return nil, err
	}
	return &third.CompleteFormDataResp{Url: t.apiAddress(req.UrlPrefix, mate.Name)}, nil
}

func (t *thirdServer) apiAddress(prefix, name string) string {
	return prefix + name
}

func (t *thirdServer) DeleteOutdatedData(ctx context.Context, req *third.DeleteOutdatedDataReq) (*third.DeleteOutdat
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	engine := t.config.RpcConfig.Object.Enable
	expireTime := time.UnixMilli(req.ExpireTime)
	// Find all expired data in S3 database
	models, err := t.s3dataBase.FindExpirationObject(ctx, engine, expireTime, req.ObjectGroup, int64(req.Limit))
	if err != nil {
		return nil, err
	}
	for i, obj := range models {
		if err := t.s3dataBase.DeleteSpecifiedData(ctx, engine, []string{obj.Name}); err != nil {
			return nil, errs.Wrap(err)
		}
		if err := t.s3dataBase.DelS3Key(ctx, engine, obj.Name); err != nil {
			return nil, err
		}
		count, err := t.s3dataBase.GetKeyCount(ctx, engine, obj.Key)
		if err != nil {
			return nil, err
		}
		log.ZDebug(ctx, "delete s3 object record", "index", i, "s3", obj, "count", count)
		if count == 0 {
			if err := t.s3.DeleteObject(ctx, obj.Key); err != nil {
				return nil, err
			}
		}
	}
	return &third.DeleteOutdatedDataResp{Count: int32(len(models))}, nil
}

type FormDataMate struct {
	Name        string `json:"name"`
	Size        int64  `json:"size"`
	ContentType string `json:"contentType"`
	Group       string `json:"group"`
	Key         string `json:"key"`
```

```
}
```

internal/rpc/third/third.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package third

import (
    "context"
    "fmt"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "github.com/openimsdk/tools/s3/disable"

    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
    "github.com/openimsdk/open-im-server/v3/pkg/localcache"
    "github.com/openimsdk/tools/s3/aws"
    "github.com/openimsdk/tools/s3/kodo"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/protocol/third"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/s3"
    "github.com/openimsdk/tools/s3/cos"
    "github.com/openimsdk/tools/s3/minio"
    "github.com/openimsdk/tools/s3/oss"
    "google.golang.org/grpc"
)

type thirdServer struct {
    third.UnimplementedThirdServer
    thirdDatabase controller.ThirdDatabase
    s3dataBase    controller.S3Database
    defaultExpire time.Duration
    config        *Config
    s3            s3.Interface
    userClient    *rpcli.UserClient
}

type Config struct {
    RpcConfig          config.Third
    RedisConfig        config.Redis
    MongodbConfig      config.Mongo
    NotificationConfig config.Notification
    Share              config.Share
    MinioConfig        config.Minio
    LocalCacheConfig   config.LocalCache
    Discovery          config.Discovery
```

```
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
    dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
    mgocli, err := dbb.Mongo(ctx)
    if err != nil {
        return err
    }
    rdb, err := dbb.Redis(ctx)
    if err != nil {
        return err
    }

    logdb, err := mgo.NewLogMongo(mgocli.GetDB())
    if err != nil {
        return err
    }
    s3db, err := mgo.NewS3Mongo(mgocli.GetDB())
    if err != nil {
        return err
    }
    var thirdCache cache.ThirdCache
    if rdb == nil {
        tc, err := mgo.NewCacheMgo(mgocli.GetDB())
        if err != nil {
            return err
        }
        thirdCache = mcache.NewThirdCache(tc)
    } else {
        thirdCache = redis.NewThirdCache(rdb)
    }
    // Select the oss method according to the profile policy
    var o s3.Interface
    switch enable := config.RpcConfig.Object.Enable; enable {
    case "minio":
        var minioCache minio.Cache
        if rdb == nil {
            mc, err := mgo.NewCacheMgo(mgocli.GetDB())
            if err != nil {
                return err
            }
            minioCache = mcache.NewMinioCache(mc)
        } else {
            minioCache = redis.NewMinioCache(rdb)
        }
        o, err = minio.NewMinio(ctx, minioCache, *config.MinioConfig.Build())
    case "cos":
        o, err = cos.NewCos(*config.RpcConfig.Object.Cos.Build())
    case "oss":
        o, err = oss.NewOSS(*config.RpcConfig.Object.Oss.Build())
    case "kodo":
        o, err = kodo.NewKodo(*config.RpcConfig.Object.Kodo.Build())
    case "aws":
        o, err = aws.NewAws(*config.RpcConfig.Object.Aws.Build())
    case "":
        o = disable.NewDisable()
    default:
        err = fmt.Errorf("invalid object enable: %s", enable)
    }
    if err != nil {
        return err
    }
    userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
    if err != nil {
        return err
    }
```

```
	localcache.InitLocalCache(&config.LocalCacheConfig)
	third.RegisterThirdServer(server, &thirdServer{
		thirdDatabase: controller.NewThirdDatabase(thirdCache, logdb),
		s3dataBase:    controller.NewS3Database(rdb, o, s3db),
		defaultExpire: time.Hour * 24 * 7,
		config:        config,
		s3:            o,
		userClient:    rpcli.NewUserClient(userConn),
	})
	return nil
}

func (t *thirdServer) FcmUpdateToken(ctx context.Context, req *third.FcmUpdateTokenReq) (resp *third.FcmUpdateTokenR
	err = t.thirdDatabase.FcmUpdateToken(ctx, req.Account, int(req.PlatformID), req.FcmToken, req.ExpireTime)
	if err != nil {
		return nil, err
	}
	return &third.FcmUpdateTokenResp{}, nil
}

func (t *thirdServer) SetAppBadge(ctx context.Context, req *third.SetAppBadgeReq) (resp *third.SetAppBadgeResp, err
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	err = t.thirdDatabase.SetAppBadge(ctx, req.UserID, int(req.AppUnreadCount))
	if err != nil {
		return nil, err
	}
	return &third.SetAppBadgeResp{}, nil
}
```

internal/rpc/third/tool.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package third

import (
	"context"
	"fmt"
	"strings"
	"unicode/utf8"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/mcontext"
)

func toPbMapArray(m map[string][]string) []*third.KeyValues {
	if len(m) == 0 {
		return nil
	}
	res := make([]*third.KeyValues, 0, len(m))
	for key := range m {
		res = append(res, &third.KeyValues{
			Key:    key,
			Values: m[key],
		})
	}
	return res
}

func (t *thirdServer) checkUploadName(ctx context.Context, name string) error {
	if name == "" {
		return errs.ErrArgs.WrapMsg("name is empty")
	}
	if name[0] == '/' {
		return errs.ErrArgs.WrapMsg("name cannot start with `/`")
	}
	if err := checkValidObjectName(name); err != nil {
		return errs.ErrArgs.WrapMsg(err.Error())
	}
	opUserID := mcontext.GetOpUserID(ctx)
	if opUserID == "" {
		return errs.ErrNoPermission.WrapMsg("opUserID is empty")
	}
	if !authverify.CheckUserIsAdmin(ctx, opUserID) {
		if !strings.HasPrefix(name, opUserID+"/") {
			return errs.ErrNoPermission.WrapMsg(fmt.Sprintf("name must start with `%s/`", opUserID))
		}
	}
	return nil
}
```

```
func checkValidObjectNamePrefix(objectName string) error {
    if len(objectName) > 1024 {
        return errs.New("object name cannot be longer than 1024 characters")
    }
    if !utf8.ValidString(objectName) {
        return errs.New("object name with non UTF-8 strings are not supported")
    }
    return nil
}

func checkValidObjectName(objectName string) error {
    if strings.TrimSpace(objectName) == "" {
        return errs.New("object name cannot be empty")
    }
    return checkValidObjectNamePrefix(objectName)
}

func putUpdate[T any](update map[string]any, name string, val interface{ GetValuePtr() *T }) {
    ptrVal := val.GetValuePtr()
    if ptrVal == nil {
        return
    }
    update[name] = *ptrVal
}
```

internal/rpc/user

internal/rpc/user/callback.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package user

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/tools/utils/datautil"

	cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	pbuser "github.com/openimsdk/protocol/user"
)

func (s *userServer) webhookBeforeUpdateUserInfo(ctx context.Context, before *config.BeforeConfig, req *pbuser.Updat
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &cbapi.CallbackBeforeUpdateUserInfoReq{
			CallbackCommand: cbapi.CallbackBeforeUpdateUserInfoCommand,
			UserID:          req.UserInfo.UserID,
			FaceURL:         &req.UserInfo.FaceURL,
			Nickname:        &req.UserInfo.Nickname,
		}
		resp := &cbapi.CallbackBeforeUpdateUserInfoResp{}
		if err := s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
			return err
		}

		datautil.NotNilReplace(&req.UserInfo.FaceURL, resp.FaceURL)
		datautil.NotNilReplace(&req.UserInfo.Ex, resp.Ex)
		datautil.NotNilReplace(&req.UserInfo.Nickname, resp.Nickname)
		return nil
	})
}

func (s *userServer) webhookAfterUpdateUserInfo(ctx context.Context, after *config.AfterConfig, req *pbuser.UpdateUs
	cbReq := &cbapi.CallbackAfterUpdateUserInfoReq{
		CallbackCommand: cbapi.CallbackAfterUpdateUserInfoCommand,
		UserID:          req.UserInfo.UserID,
		FaceURL:         req.UserInfo.FaceURL,
		Nickname:        req.UserInfo.Nickname,
	}
	s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterUpdateUserInfoResp{}, after)
}

func (s *userServer) webhookBeforeUpdateUserInfoEx(ctx context.Context, before *config.BeforeConfig, req *pbuser.Upd
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &cbapi.CallbackBeforeUpdateUserInfoExReq{
			CallbackCommand: cbapi.CallbackBeforeUpdateUserInfoExCommand,
```

```
        UserID:          req.UserInfo.UserID,
        FaceURL:         req.UserInfo.FaceURL,
        Nickname:        req.UserInfo.Nickname,
    }
    resp := &cbapi.CallbackBeforeUpdateUserInfoExResp{}
    if err := s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
        return err
    }

    datautil.NotNilReplace(req.UserInfo.FaceURL, resp.FaceURL)
    datautil.NotNilReplace(req.UserInfo.Ex, resp.Ex)
    datautil.NotNilReplace(req.UserInfo.Nickname, resp.Nickname)
    return nil
  })
}

func (s *userServer) webhookAfterUpdateUserInfoEx(ctx context.Context, after *config.AfterConfig, req *pbuser.Update
  cbReq := &cbapi.CallbackAfterUpdateUserInfoExReq{
    CallbackCommand: cbapi.CallbackAfterUpdateUserInfoExCommand,
    UserID:          req.UserInfo.UserID,
    FaceURL:         req.UserInfo.FaceURL,
    Nickname:        req.UserInfo.Nickname,
  }
  s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterUpdateUserInfoExResp{}, after
}

func (s *userServer) webhookBeforeUserRegister(ctx context.Context, before *config.BeforeConfig, req *pbuser.UserReg
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    cbReq := &cbapi.CallbackBeforeUserRegisterReq{
        CallbackCommand: cbapi.CallbackBeforeUserRegisterCommand,
        Users:           req.Users,
    }

    resp := &cbapi.CallbackBeforeUserRegisterResp{}

    if err := s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
        return err
    }

    if len(resp.Users) != 0 {
        req.Users = resp.Users
    }
    return nil
  })
}

func (s *userServer) webhookAfterUserRegister(ctx context.Context, after *config.AfterConfig, req *pbuser.UserRegist
  cbReq := &cbapi.CallbackAfterUserRegisterReq{
    CallbackCommand: cbapi.CallbackAfterUserRegisterCommand,
    Users:           req.Users,
  }

  s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterUserRegisterResp{}, after)
}
```

internal/rpc/user/config.go

```
package user

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	pbuser "github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/utils/datautil"
)

func (s *userServer) GetUserClientConfig(ctx context.Context, req *pbuser.GetUserClientConfigReq) (*pbuser.GetUserCl
	if req.UserID != "" {
		if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
			return nil, err
		}
		if _, err := s.db.GetUserByID(ctx, req.UserID); err != nil {
			return nil, err
		}
	}
	res, err := s.clientConfig.GetUserConfig(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	return &pbuser.GetUserClientConfigResp{Configs: res}, nil
}

func (s *userServer) SetUserClientConfig(ctx context.Context, req *pbuser.SetUserClientConfigReq) (*pbuser.SetUserCl
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	if req.UserID != "" {
		if _, err := s.db.GetUserByID(ctx, req.UserID); err != nil {
			return nil, err
		}
	}
	if err := s.clientConfig.SetUserConfig(ctx, req.UserID, req.Configs); err != nil {
		return nil, err
	}
	return &pbuser.SetUserClientConfigResp{}, nil
}

func (s *userServer) DelUserClientConfig(ctx context.Context, req *pbuser.DelUserClientConfigReq) (*pbuser.DelUserCl
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	if err := s.clientConfig.DelUserConfig(ctx, req.UserID, req.Keys); err != nil {
		return nil, err
	}
	return &pbuser.DelUserClientConfigResp{}, nil
}

func (s *userServer) PageUserClientConfig(ctx context.Context, req *pbuser.PageUserClientConfigReq) (*pbuser.PageUse
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	total, res, err := s.clientConfig.GetUserConfigPage(ctx, req.UserID, req.Key, req.Pagination)
	if err != nil {
		return nil, err
	}
	return &pbuser.PageUserClientConfigResp{
		Total: total,
		Configs: datautil.Slice(res, func(e *model.ClientConfig) *pbuser.ClientConfig {
			return &pbuser.ClientConfig{
```

```
        UserID: e.UserID,
        Key:    e.Key,
        Value:  e.Value,
    }
  }),
}, nil
}
```

internal/rpc/user/notification.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package user

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/msg"

	relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/notification/common_user"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
	"github.com/openimsdk/open-im-server/v3/pkg/notification"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/sdkws"
)

type UserNotificationSender struct {
	*notification.NotificationSender
	getUsersInfo func(ctx context.Context, userIDs []string) ([]common_user.CommonUser, error)
	// db controller
	db controller.UserDatabase
}

type userNotificationSenderOptions func(*UserNotificationSender)

func WithUserDB(db controller.UserDatabase) userNotificationSenderOptions {
	return func(u *UserNotificationSender) {
		u.db = db
	}
}

func WithUserFunc(
	fn func(ctx context.Context, userIDs []string) (users []*relationtb.User, err error),
) userNotificationSenderOptions {
	return func(u *UserNotificationSender) {
		f := func(ctx context.Context, userIDs []string) (result []common_user.CommonUser, err error) {
			users, err := fn(ctx, userIDs)
			if err != nil {
				return nil, err
			}
			for _, user := range users {
				result = append(result, user)
			}
			return result, nil
		}
		u.getUsersInfo = f
	}
}
```

```
func NewUserNotificationSender(config *Config, msgClient *rpcli.MsgClient, opts ...userNotificationSenderOptions) *U
■f := &UserNotificationSender{
■■NotificationSender: notification.NewNotificationSender(&config.NotificationConfig, notification.WithRpcClient(func
■■■return msgClient.SendMsg(ctx, req)
■■})),
■}
■for _, opt := range opts {
■■opt(f)
■}
■return f
}

/* func (u *UserNotificationSender) getUsersInfoMap(
■ctx context.Context,
■userIDs []string,
) (map[string]*sdkws.UserInfo, error) {
■users, err := u.getUsersInfo(ctx, userIDs)
■if err != nil {
■■return nil, err
■}
■result := make(map[string]*sdkws.UserInfo)
■for _, user := range users {
■■result[user.GetUserID()] = user.(*sdkws.UserInfo)
■}
■return result, nil
} */

/* func (u *UserNotificationSender) getFromToUserNickname(
■ctx context.Context,
■fromUserID, toUserID string,
) (string, string, error) {
■users, err := u.getUsersInfoMap(ctx, []string{fromUserID, toUserID})
■if err != nil {
■■return "", "", nil
■}
■return users[fromUserID].Nickname, users[toUserID].Nickname, nil
} */

func (u *UserNotificationSender) UserStatusChangeNotification(
■ctx context.Context,
■tips *sdkws.UserStatusChangeTips,
) {
■u.Notification(ctx, tips.FromUserID, tips.ToUserID, constant.UserStatusChangeNotification, tips)
}
func (u *UserNotificationSender) UserCommandUpdateNotification(
■ctx context.Context,
■tips *sdkws.UserCommandUpdateTips,
) {
■u.Notification(ctx, tips.FromUserID, tips.ToUserID, constant.UserCommandUpdateNotification, tips)
}
func (u *UserNotificationSender) UserCommandAddNotification(
■ctx context.Context,
■tips *sdkws.UserCommandAddTips,
) {
■u.Notification(ctx, tips.FromUserID, tips.ToUserID, constant.UserCommandAddNotification, tips)
}
func (u *UserNotificationSender) UserCommandDeleteNotification(
■ctx context.Context,
■tips *sdkws.UserCommandDeleteTips,
) {
■u.Notification(ctx, tips.FromUserID, tips.ToUserID, constant.UserCommandDeleteNotification, tips)
}
```

internal/rpc/user/online.go

```
package user

import (
"context"

"github.com/openimsdk/tools/utils/datautil"

"github.com/openimsdk/protocol/constant"
pbuser "github.com/openimsdk/protocol/user"
)

func (s *userServer) getUserOnlineStatus(ctx context.Context, userID string) (*pbuser.OnlineStatus, error) {
platformIDs, err := s.online.GetOnline(ctx, userID)
if err != nil {
return nil, err
}
status := pbuser.OnlineStatus{
UserID:      userID,
PlatformIDs: platformIDs,
}
if len(platformIDs) > 0 {
status.Status = constant.Online
} else {
status.Status = constant.Offline
}
return &status, nil
}

func (s *userServer) getUsersOnlineStatus(ctx context.Context, userIDs []string) ([]*pbuser.OnlineStatus, error) {
res := make([]*pbuser.OnlineStatus, 0, len(userIDs))
for _, userID := range userIDs {
status, err := s.getUserOnlineStatus(ctx, userID)
if err != nil {
return nil, err
}
res = append(res, status)
}
return res, nil
}

// SubscribeOrCancelUsersStatus Subscribe online or cancel online users.
func (s *userServer) SubscribeOrCancelUsersStatus(ctx context.Context, req *pbuser.SubscribeOrCancelUsersStatusReq)
return &pbuser.SubscribeOrCancelUsersStatusResp{}, nil
}

// GetUserStatus Get the online status of the user.
func (s *userServer) GetUserStatus(ctx context.Context, req *pbuser.GetUserStatusReq) (*pbuser.GetUserStatusResp, er
res, err := s.getUsersOnlineStatus(ctx, req.UserIDs)
if err != nil {
return nil, err
}
return &pbuser.GetUserStatusResp{StatusList: res}, nil
}

// SetUserStatus Synchronize user's online status.
func (s *userServer) SetUserStatus(ctx context.Context, req *pbuser.SetUserStatusReq) (*pbuser.SetUserStatusResp, er
var (
online  []int32
offline []int32
)
switch req.Status {
case constant.Online:
online = []int32{req.PlatformID}
case constant.Offline:
```

```
    offline = []int32{req.PlatformID}
  }
  if err := s.online.SetUserOnline(ctx, req.UserID, online, offline); err != nil {
    return nil, err
  }
  return &pbuser.SetUserStatusResp{}, nil
}

// GetSubscribeUsersStatus Get the online status of subscribers.
func (s *userServer) GetSubscribeUsersStatus(ctx context.Context, req *pbuser.GetSubscribeUsersStatusReq) (*pbuser.G
  return &pbuser.GetSubscribeUsersStatusResp{}, nil
}

func (s *userServer) SetUserOnlineStatus(ctx context.Context, req *pbuser.SetUserOnlineStatusReq) (*pbuser.SetUserOn
  for _, status := range req.Status {
    if err := s.online.SetUserOnline(ctx, status.UserID, status.Online, status.Offline); err != nil {
      return nil, err
    }
  }
  return &pbuser.SetUserOnlineStatusResp{}, nil
}

func (s *userServer) GetAllOnlineUsers(ctx context.Context, req *pbuser.GetAllOnlineUsersReq) (*pbuser.GetAllOnlineU
  resMap, nextCursor, err := s.online.GetAllOnlineUsers(ctx, req.Cursor)
  if err != nil {
    return nil, err
  }
  resp := &pbuser.GetAllOnlineUsersResp{
    StatusList: make([]*pbuser.OnlineStatus, 0, len(resMap)),
    NextCursor: nextCursor,
  }
  for userID, plats := range resMap {
    resp.StatusList = append(resp.StatusList, &pbuser.OnlineStatus{
      UserID:      userID,
      Status:      int32(datautil.If(len(plats) > 0, constant.Online, constant.Offline)),
      PlatformIDs: plats,
    })
  }
  return resp, nil
}
```

internal/rpc/user/statistics.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package user

import (
    "context"
    "time"

    pbuser "github.com/openimsdk/protocol/user"
    "github.com/openimsdk/tools/errs"
)

func (s *userServer) UserRegisterCount(ctx context.Context, req *pbuser.UserRegisterCountReq) (*pbuser.UserRegisterC
    if req.Start > req.End {
        return nil, errs.ErrArgs.WrapMsg("start > end")
    }
    total, err := s.db.CountTotal(ctx, nil)
    if err != nil {
        return nil, err
    }
    start := time.UnixMilli(req.Start)
    before, err := s.db.CountTotal(ctx, &start)
    if err != nil {
        return nil, err
    }
    count, err := s.db.CountRangeEverydayTotal(ctx, start, time.UnixMilli(req.End))
    if err != nil {
        return nil, err
    }
    return &pbuser.UserRegisterCountResp{Total: total, Before: before, Count: count}, nil
}
```

internal/rpc/user/user.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package user

import (
    "context"
    "errors"
    "math/rand"
    "strings"
    "sync"
    "time"

    "github.com/openimsdk/open-im-server/v3/internal/rpc/relation"
    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/convert"
    "github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
    tablerelation "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/localcache"
    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/group"
    friendpb "github.com/openimsdk/protocol/relation"
    "github.com/openimsdk/protocol/sdkws"
    pbuser "github.com/openimsdk/protocol/user"
    "github.com/openimsdk/tools/db/pagination"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/utils/datautil"
    "google.golang.org/grpc"
)

const (
    defaultSecret = "openIM123"
)

type userServer struct {
    pbuser.UnimplementedUserServer
    online                   cache.OnlineCache
    db                       controller.UserDatabase
    friendNotificationSender *relation.FriendNotificationSender
    userNotificationSender   *UserNotificationSender
    RegisterCenter           discovery.Conn
    config                   *Config
    webhookClient            *webhook.Client
```

```
    groupClient              *rpcli.GroupClient
    relationClient           *rpcli.RelationClient
    clientConfig             controller.ClientConfigDatabase

    adminUserIDs []string
}

type Config struct {
    RpcConfig          config.User
    RedisConfig        config.Redis
    MongodbConfig      config.Mongo
    KafkaConfig        config.Kafka
    NotificationConfig config.Notification
    Share              config.Share
    WebhooksConfig     config.Webhooks
    LocalCacheConfig   config.LocalCache
    Discovery          config.Discovery
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
    dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
    mgocli, err := dbb.Mongo(ctx)
    if err != nil {
        return err
    }
    rdb, err := dbb.Redis(ctx)
    if err != nil {
        return err
    }

    users := make([]*tablerelation.User, 0)

    for i := range config.Share.IMAdminUser.UserIDs {
        users = append(users, &tablerelation.User{
            UserID:         config.Share.IMAdminUser.UserIDs[i],
            Nickname:       config.Share.IMAdminUser.Nicknames[i],
            AppMangerLevel: constant.AppAdmin,
        })
    }
    userDB, err := mgo.NewUserMongo(mgocli.GetDB())
    if err != nil {
        return err
    }
    clientConfigDB, err := mgo.NewClientConfig(mgocli.GetDB())
    if err != nil {
        return err
    }
    msgConn, err := client.GetConn(ctx, config.Discovery.RpcService.Msg)
    if err != nil {
        return err
    }
    groupConn, err := client.GetConn(ctx, config.Discovery.RpcService.Group)
    if err != nil {
        return err
    }
    friendConn, err := client.GetConn(ctx, config.Discovery.RpcService.Friend)
    if err != nil {
        return err
    }
    msgClient := rpcli.NewMsgClient(msgConn)
    userCache := redis.NewUserCacheRedis(rdb, &config.LocalCacheConfig, userDB, redis.GetRocksCacheOptions())
    database := controller.NewUserDatabase(userDB, userCache, mgocli.GetTx())
    localcache.InitLocalCache(&config.LocalCacheConfig)
    u := &userServer{
        online:                   redis.NewUserOnline(rdb),
        db:                       database,
```

```
    RegisterCenter:          client,
    friendNotificationSender: relation.NewFriendNotificationSender(&config.NotificationConfig, msgClient, relation.Wi
    userNotificationSender:   NewUserNotificationSender(config, msgClient, WithUserFunc(database.FindWithError)),
    config:                   config,
    webhookClient:            webhook.NewWebhookClient(config.WebhooksConfig.URL),
    clientConfig:             controller.NewClientConfigDatabase(clientConfigDB, redis.NewClientConfigCache(rdb, clie
    groupClient:              rpcli.NewGroupClient(groupConn),
    relationClient:           rpcli.NewRelationClient(friendConn),
    adminUserIDs:             config.Share.IMAdminUser.UserIDs,
  }
  pbuser.RegisterUserServer(server, u)
  return u.db.InitOnce(context.Background(), users)
}

func (s *userServer) GetDesignateUsers(ctx context.Context, req *pbuser.GetDesignateUsersReq) (resp *pbuser.GetDesig
  resp = &pbuser.GetDesignateUsersResp{}
  users, err := s.db.Find(ctx, req.UserIDs)
  if err != nil {
    return nil, err
  }

  resp.UsersInfo = convert.UsersDB2Pb(users)
  return resp, nil
}

// deprecated:
// UpdateUserInfo
func (s *userServer) UpdateUserInfo(ctx context.Context, req *pbuser.UpdateUserInfoReq) (resp *pbuser.UpdateUserInfo
  resp = &pbuser.UpdateUserInfoResp{}
  err = authverify.CheckAccess(ctx, req.UserInfo.UserID)
  if err != nil {
    return nil, err
  }

  if err := s.webhookBeforeUpdateUserInfo(ctx, &s.config.WebhooksConfig.BeforeUpdateUserInfo, req); err != nil {
    return nil, err
  }
  data := convert.UserPb2DBMap(req.UserInfo)
  oldUser, err := s.db.GetUserByID(ctx, req.UserInfo.UserID)
  if err != nil {
    return nil, err
  }
  if err := s.db.UpdateByMap(ctx, req.UserInfo.UserID, data); err != nil {
    return nil, err
  }
  s.friendNotificationSender.UserInfoUpdatedNotification(ctx, req.UserInfo.UserID)

  s.webhookAfterUpdateUserInfo(ctx, &s.config.WebhooksConfig.AfterUpdateUserInfo, req)
  if err = s.NotificationUserInfoUpdate(ctx, req.UserInfo.UserID, oldUser); err != nil {
    return nil, err
  }
  return resp, nil
}

func (s *userServer) UpdateUserInfoEx(ctx context.Context, req *pbuser.UpdateUserInfoExReq) (resp *pbuser.UpdateUser
  resp = &pbuser.UpdateUserInfoExResp{}
  err = authverify.CheckAccess(ctx, req.UserInfo.UserID)
  if err != nil {
    return nil, err
  }

  if err = s.webhookBeforeUpdateUserInfoEx(ctx, &s.config.WebhooksConfig.BeforeUpdateUserInfoEx, req); err != nil {
    return nil, err
  }

  oldUser, err := s.db.GetUserByID(ctx, req.UserInfo.UserID)
```

```
	if err != nil {
		return nil, err
	}

	data := convert.UserPb2DBMapEx(req.UserInfo)
	if err = s.db.UpdateByMap(ctx, req.UserInfo.UserID, data); err != nil {
		return nil, err
	}

	s.friendNotificationSender.UserInfoUpdatedNotification(ctx, req.UserInfo.UserID)

	//friends, err := s.friendRpcClient.GetFriendIDs(ctx, req.UserInfo.UserID)
	//if err != nil {
	//	return nil, err
	//}
	//if req.UserInfo.Nickname != nil || req.UserInfo.FaceURL != nil {
	//	if err := s.NotificationUserInfoUpdate(ctx, req.UserInfo.UserID); err != nil {
	//		return nil, err
	//	}
	//}
	//for _, friendID := range friends {
	//	s.friendNotificationSender.FriendInfoUpdatedNotification(ctx, req.UserInfo.UserID, friendID)
	//}

	s.webhookAfterUpdateUserInfoEx(ctx, &s.config.WebhooksConfig.AfterUpdateUserInfoEx, req)
	if err := s.NotificationUserInfoUpdate(ctx, req.UserInfo.UserID, oldUser); err != nil {
		return nil, err
	}

	return resp, nil
}
func (s *userServer) SetGlobalRecvMessageOpt(ctx context.Context, req *pbuser.SetGlobalRecvMessageOptReq) (resp *pbu
	resp = &pbuser.SetGlobalRecvMessageOptResp{}
	if _, err := s.db.FindWithError(ctx, []string{req.UserID}); err != nil {
		return nil, err
	}
	m := make(map[string]any, 1)
	m["global_recv_msg_opt"] = req.GlobalRecvMsgOpt
	if err := s.db.UpdateByMap(ctx, req.UserID, m); err != nil {
		return nil, err
	}
	s.friendNotificationSender.UserInfoUpdatedNotification(ctx, req.UserID)
	return resp, nil
}

func (s *userServer) AccountCheck(ctx context.Context, req *pbuser.AccountCheckReq) (resp *pbuser.AccountCheckResp,
	resp = &pbuser.AccountCheckResp{}
	if datautil.Duplicate(req.CheckUserIDs) {
		return nil, errs.ErrArgs.WrapMsg("userID repeated")
	}
	if err = authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	users, err := s.db.Find(ctx, req.CheckUserIDs)
	if err != nil {
		return nil, err
	}
	userIDs := make(map[string]any, 0)
	for _, v := range users {
		userIDs[v.UserID] = nil
	}
	for _, v := range req.CheckUserIDs {
		temp := &pbuser.AccountCheckRespSingleUserStatus{UserID: v}
		if _, ok := userIDs[v]; ok {
			temp.AccountStatus = constant.Registered
		} else {
```

```
		temp.AccountStatus = constant.UnRegistered
	}
	resp.Results = append(resp.Results, temp)
}
return resp, nil
}

func (s *userServer) GetPaginationUsers(ctx context.Context, req *pbuser.GetPaginationUsersReq) (resp *pbuser.GetPag
if req.UserID == "" && req.NickName == "" {
	total, users, err := s.db.PageFindUser(ctx, constant.IMOrdinaryUser, constant.AppOrdinaryUsers, req.Pagination)
	if err != nil {
		return nil, err
	}
	return &pbuser.GetPaginationUsersResp{Total: int32(total), Users: convert.UsersDB2Pb(users)}, err
} else {
	total, users, err := s.db.PageFindUserWithKeyword(ctx, constant.IMOrdinaryUser, constant.AppOrdinaryUsers, req.Us
	if err != nil {
		return nil, err
	}
	return &pbuser.GetPaginationUsersResp{Total: int32(total), Users: convert.UsersDB2Pb(users)}, err

}

}

func (s *userServer) UserRegister(ctx context.Context, req *pbuser.UserRegisterReq) (resp *pbuser.UserRegisterResp,
resp = &pbuser.UserRegisterResp{}
if len(req.Users) == 0 {
	return nil, errs.ErrArgs.WrapMsg("users is empty")
}
// check if secret is changed
//if s.config.Share.Secret == defaultSecret {
//	return nil, servererrs.ErrSecretNotChanged.Wrap()
//}
if err = authverify.CheckAdmin(ctx); err != nil {
	return nil, err
}
if datautil.DuplicateAny(req.Users, func(e *sdkws.UserInfo) string { return e.UserID }) {
	return nil, errs.ErrArgs.WrapMsg("userID repeated")
}
userIDs := make([]string, 0)
for _, user := range req.Users {
	if user.UserID == "" {
		return nil, errs.ErrArgs.WrapMsg("userID is empty")
	}
	if strings.Contains(user.UserID, ":") {
		return nil, errs.ErrArgs.WrapMsg("userID contains ':' is invalid userID")
	}
	userIDs = append(userIDs, user.UserID)
}
exist, err := s.db.IsExist(ctx, userIDs)
if err != nil {
	return nil, err
}
if exist {
	return nil, servererrs.ErrRegisteredAlready.WrapMsg("userID registered already")
}
if err := s.webhookBeforeUserRegister(ctx, &s.config.WebhooksConfig.BeforeUserRegister, req); err != nil {
	return nil, err
}
now := time.Now()
users := make([]*tablerelation.User, 0, len(req.Users))
for _, user := range req.Users {
	users = append(users, &tablerelation.User{
		UserID:           user.UserID,
		Nickname:         user.Nickname,
```

```
		FaceURL:          user.FaceURL,
		Ex:               user.Ex,
		CreateTime:       now,
		AppMangerLevel:   user.AppMangerLevel,
		GlobalRecvMsgOpt: user.GlobalRecvMsgOpt,
	})
	}
	if err := s.db.Create(ctx, users); err != nil {
		return nil, err
	}

	prommetrics.UserRegisterCounter.Add(float64(len(users)))

	s.webhookAfterUserRegister(ctx, &s.config.WebhooksConfig.AfterUserRegister, req)
	return resp, nil
}

func (s *userServer) GetGlobalRecvMessageOpt(ctx context.Context, req *pbuser.GetGlobalRecvMessageOptReq) (resp *pbu
	user, err := s.db.FindWithError(ctx, []string{req.UserID})
	if err != nil {
		return nil, err
	}
	return &pbuser.GetGlobalRecvMessageOptResp{GlobalRecvMsgOpt: user[0].GlobalRecvMsgOpt}, nil
}

// GetAllUserID Get user account by page.
func (s *userServer) GetAllUserID(ctx context.Context, req *pbuser.GetAllUserIDReq) (resp *pbuser.GetAllUserIDResp,
	total, userIDs, err := s.db.GetAllUserID(ctx, req.Pagination)
	if err != nil {
		return nil, err
	}
	return &pbuser.GetAllUserIDResp{Total: int32(total), UserIDs: userIDs}, nil
}

// ProcessUserCommandAdd user general function add.
func (s *userServer) ProcessUserCommandAdd(ctx context.Context, req *pbuser.ProcessUserCommandAddReq) (*pbuser.Proce
	err := authverify.CheckAccess(ctx, req.UserID)
	if err != nil {
		return nil, err
	}

	var value string
	if req.Value != nil {
		value = req.Value.Value
	}
	var ex string
	if req.Ex != nil {
		value = req.Ex.Value
	}
	// Assuming you have a method in s.storage to add a user command
	err = s.db.AddUserCommand(ctx, req.UserID, req.Type, req.Uuid, value, ex)
	if err != nil {
		return nil, err
	}
	tips := &sdkws.UserCommandAddTips{
		FromUserID: req.UserID,
		ToUserID:   req.UserID,
	}
	s.userNotificationSender.UserCommandAddNotification(ctx, tips)
	return &pbuser.ProcessUserCommandAddResp{}, nil
}

// ProcessUserCommandDelete user general function delete.
func (s *userServer) ProcessUserCommandDelete(ctx context.Context, req *pbuser.ProcessUserCommandDeleteReq) (*pbuser
	err := authverify.CheckAccess(ctx, req.UserID)
	if err != nil {
```

```
    return nil, err
  }

  err = s.db.DeleteUserCommand(ctx, req.UserID, req.Type, req.Uuid)
  if err != nil {
    return nil, err
  }
  tips := &sdkws.UserCommandDeleteTips{
    FromUserID: req.UserID,
    ToUserID:   req.UserID,
  }
  s.userNotificationSender.UserCommandDeleteNotification(ctx, tips)
  return &pbuser.ProcessUserCommandDeleteResp{}, nil
}

// ProcessUserCommandUpdate user general function update.
func (s *userServer) ProcessUserCommandUpdate(ctx context.Context, req *pbuser.ProcessUserCommandUpdateReq) (*pbuser
  err := authverify.CheckAccess(ctx, req.UserID)
  if err != nil {
    return nil, err
  }
  val := make(map[string]any)

  // Map fields from eax to val
  if req.Value != nil {
    val["value"] = req.Value.Value
  }
  if req.Ex != nil {
    val["ex"] = req.Ex.Value
  }

  // Assuming you have a method in s.storage to update a user command
  err = s.db.UpdateUserCommand(ctx, req.UserID, req.Type, req.Uuid, val)
  if err != nil {
    return nil, err
  }
  tips := &sdkws.UserCommandUpdateTips{
    FromUserID: req.UserID,
    ToUserID:   req.UserID,
  }
  s.userNotificationSender.UserCommandUpdateNotification(ctx, tips)
  return &pbuser.ProcessUserCommandUpdateResp{}, nil
}

func (s *userServer) ProcessUserCommandGet(ctx context.Context, req *pbuser.ProcessUserCommandGetReq) (*pbuser.Proce

  err := authverify.CheckAccess(ctx, req.UserID)
  if err != nil {
    return nil, err
  }
  // Fetch user commands from the database
  commands, err := s.db.GetUserCommands(ctx, req.UserID, req.Type)
  if err != nil {
    return nil, err
  }

  // Initialize commandInfoSlice as an empty slice
  commandInfoSlice := make([]*pbuser.CommandInfoResp, 0, len(commands))

  for _, command := range commands {
    // No need to use index since command is already a pointer
    commandInfoSlice = append(commandInfoSlice, &pbuser.CommandInfoResp{
      Type:       command.Type,
      Uuid:       command.Uuid,
      Value:      command.Value,
      CreateTime: command.CreateTime,
```

```
        Ex:         command.Ex,
    })
    }

    // Return the response with the slice
    return &pbuser.ProcessUserCommandGetResp{CommandResp: commandInfoSlice}, nil
}

func (s *userServer) ProcessUserCommandGetAll(ctx context.Context, req *pbuser.ProcessUserCommandGetAllReq) (*pbuser
    err := authverify.CheckAccess(ctx, req.UserID)
    if err != nil {
        return nil, err
    }
    // Fetch user commands from the database
    commands, err := s.db.GetAllUserCommands(ctx, req.UserID)
    if err != nil {
        return nil, err
    }

    // Initialize commandInfoSlice as an empty slice
    commandInfoSlice := make([]*pbuser.AllCommandInfoResp, 0, len(commands))

    for _, command := range commands {
        // No need to use index since command is already a pointer
        commandInfoSlice = append(commandInfoSlice, &pbuser.AllCommandInfoResp{
            Type:       command.Type,
            Uuid:       command.Uuid,
            Value:      command.Value,
            CreateTime: command.CreateTime,
            Ex:         command.Ex,
        })
    }

    // Return the response with the slice
    return &pbuser.ProcessUserCommandGetAllResp{CommandResp: commandInfoSlice}, nil
}

func (s *userServer) AddNotificationAccount(ctx context.Context, req *pbuser.AddNotificationAccountReq) (*pbuser.Add
    if err := authverify.CheckAdmin(ctx); err != nil {
        return nil, err
    }
    if req.AppMangerLevel < constant.AppNotificationAdmin {
        return nil, errs.ErrArgs.WithDetail("app level not supported")
    }
    if req.UserID == "" {
        for i := 0; i < 20; i++ {
            userId := s.genUserID()
            _, err := s.db.FindWithError(ctx, []string{userId})
            if err == nil {
                continue
            }
            req.UserID = userId
            break
        }
        if req.UserID == "" {
            return nil, errs.ErrInternalServer.WrapMsg("gen user id failed")
        }
    } else {
        _, err := s.db.FindWithError(ctx, []string{req.UserID})
        if err == nil {
            return nil, errs.ErrArgs.WrapMsg("userID is used")
        }
    }

    user := &tablerelation.User{
        UserID:         req.UserID,
```

```
    Nickname:       req.NickName,
    FaceURL:        req.FaceURL,
    CreateTime:     time.Now(),
    AppMangerLevel: req.AppMangerLevel,
  }
  if err := s.db.Create(ctx, []*tablerelation.User{user}); err != nil {
    return nil, err
  }

  return &pbuser.AddNotificationAccountResp{
    UserID:         req.UserID,
    NickName:       req.NickName,
    FaceURL:        req.FaceURL,
    AppMangerLevel: req.AppMangerLevel,
  }, nil
}

func (s *userServer) UpdateNotificationAccountInfo(ctx context.Context, req *pbuser.UpdateNotificationAccountInfoReq
  if err := authverify.CheckAdmin(ctx); err != nil {
    return nil, err
  }

  if _, err := s.db.FindWithError(ctx, []string{req.UserID}); err != nil {
    return nil, errs.ErrArgs.Wrap()
  }

  user := map[string]interface{}{}

  if req.NickName != "" {
    user["nickname"] = req.NickName
  }

  if req.FaceURL != "" {
    user["face_url"] = req.FaceURL
  }

  if err := s.db.UpdateByMap(ctx, req.UserID, user); err != nil {
    return nil, err
  }

  return &pbuser.UpdateNotificationAccountInfoResp{}, nil
}

func (s *userServer) SearchNotificationAccount(ctx context.Context, req *pbuser.SearchNotificationAccountReq) (*pbus
  // Check if user is an admin
  if err := authverify.CheckAdmin(ctx); err != nil {
    return nil, err
  }

  var users []*tablerelation.User
  var err error

  // If a keyword is provided in the request
  if req.Keyword != "" {
    // Find users by keyword
    users, err = s.db.Find(ctx, []string{req.Keyword})
    if err != nil {
      return nil, err
    }

    // Convert users to response format
    resp := s.userModelToResp(users, req.Pagination, req.AppManagerLevel)
    if resp.Total != 0 {
      return resp, nil
    }
```

```
    // Find users by nickname if no users found by keyword
    users, err = s.db.FindByNickname(ctx, req.Keyword)
    if err != nil {
        return nil, err
    }
    resp = s.userModelToResp(users, req.Pagination, req.AppManagerLevel)
    return resp, nil
  }

  // If no keyword, find users with notification settings
  if req.AppManagerLevel != nil {
    users, err = s.db.FindNotification(ctx, int64(*req.AppManagerLevel))
    if err != nil {
        return nil, err
    }
  } else {
    users, err = s.db.FindSystemAccount(ctx)
    if err != nil {
        return nil, err
    }
  }

  resp := s.userModelToResp(users, req.Pagination, req.AppManagerLevel)
  return resp, nil
}

func (s *userServer) GetNotificationAccount(ctx context.Context, req *pbuser.GetNotificationAccountReq) (*pbuser.Get
  if req.UserID == "" {
    return nil, errs.ErrArgs.WrapMsg("userID is empty")
  }
  user, err := s.db.GetUserByID(ctx, req.UserID)
  if err != nil {
    return nil, servererrs.ErrUserIDNotFound.Wrap()
  }
  if user.AppMangerLevel >= constant.AppAdmin {
    return &pbuser.GetNotificationAccountResp{Account: &pbuser.NotificationAccountInfo{
        UserID:         user.UserID,
        FaceURL:        user.FaceURL,
        NickName:       user.Nickname,
        AppMangerLevel: user.AppMangerLevel,
    }}, nil
  }

  return nil, errs.ErrNoPermission.WrapMsg("notification messages cannot be sent for this ID")
}

func (s *userServer) genUserID() string {
  const l = 10
  data := make([]byte, l)
  rand.Read(data)
  chars := []byte("0123456789")
  for i := 0; i < len(data); i++ {
    if i == 0 {
        data[i] = chars[1:][data[i]%9]
    } else {
        data[i] = chars[data[i]%10]
    }
  }
  return string(data)
}

func (s *userServer) userModelToResp(users []*tablerelation.User, pagination pagination.Pagination, appManagerLevel
  accounts := make([]*pbuser.NotificationAccountInfo, 0)
  var total int64
  for _, v := range users {
    if v.AppMangerLevel >= constant.AppNotificationAdmin && !datautil.Contain(v.UserID, s.adminUserIDs...) {
```

```
    if appManagerLevel != nil {
        if v.AppMangerLevel != *appManagerLevel {
            continue
        }
    }
    temp := &pbuser.NotificationAccountInfo{
        UserID:         v.UserID,
        FaceURL:        v.FaceURL,
        NickName:       v.Nickname,
        AppMangerLevel: v.AppMangerLevel,
    }
    accounts = append(accounts, temp)
    total += 1
  }
}

notificationAccounts := datautil.Paginate(accounts, int(pagination.GetPageNumber()), int(pagination.GetShowNumber()))

return &pbuser.SearchNotificationAccountResp{Total: total, NotificationAccounts: notificationAccounts}
}

func (s *userServer) NotificationUserInfoUpdate(ctx context.Context, userID string, oldUser *tablerelation.User) err
user, err := s.db.GetUserByID(ctx, userID)
if err != nil {
  return err
}
if user.Nickname == oldUser.Nickname && user.FaceURL == oldUser.FaceURL {
  return nil
}
oldUserInfo := convert.UserDB2Pb(oldUser)
newUserInfo := convert.UserDB2Pb(user)
var wg sync.WaitGroup
var es [2]error
wg.Add(len(es))
go func() {
  defer wg.Done()
  _, es[0] = s.groupClient.NotificationUserInfoUpdate(ctx, &group.NotificationUserInfoUpdateReq{
    UserID:      userID,
    OldUserInfo: oldUserInfo,
    NewUserInfo: newUserInfo,
  })
}()

go func() {
  defer wg.Done()
  _, es[1] = s.relationClient.NotificationUserInfoUpdate(ctx, &friendpb.NotificationUserInfoUpdateReq{
    UserID:      userID,
    OldUserInfo: oldUserInfo,
    NewUserInfo: newUserInfo,
  })
}()
wg.Wait()
return errors.Join(es[:]...)
}

func (s *userServer) SortQuery(ctx context.Context, req *pbuser.SortQueryReq) (*pbuser.SortQueryResp, error) {
users, err := s.db.SortQuery(ctx, req.UserIDName, req.Asc)
if err != nil {
  return nil, err
}
return &pbuser.SortQueryResp{Users: convert.UsersDB2Pb(users)}, nil
}
```

internal/rpc/incrversion

internal/rpc/incrversion/batch_option.go

```go
package incrversion

import (
	"context"
	"fmt"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/errs"
	"go.mongodb.org/mongo-driver/bson/primitive"
)

type BatchOption[A, B any] struct {
	Ctx            context.Context
	TargetKeys     []string
	VersionIDs     []string
	VersionNumbers []uint64
	//SyncLimit       int
	Versions         func(ctx context.Context, dIds []string, versions []uint64, limits []int) (map[string]*model.Versi
	CacheMaxVersions func(ctx context.Context, dIds []string) (map[string]*model.VersionLog, error)
	Find             func(ctx context.Context, dId string, ids []string) (A, error)
	Resp             func(versionsMap map[string]*model.VersionLog, deleteIdsMap map[string][]string, insertListMap, up
}

func (o *BatchOption[A, B]) newError(msg string) error {
	return errs.ErrInternalServer.WrapMsg(msg)
}

func (o *BatchOption[A, B]) check() error {
	if o.Ctx == nil {
		return o.newError("opt ctx is nil")
	}
	if len(o.TargetKeys) == 0 {
		return o.newError("targetKeys is empty")
	}
	if o.Versions == nil {
		return o.newError("func versions is nil")
	}
	if o.Find == nil {
		return o.newError("func find is nil")
	}
	if o.Resp == nil {
		return o.newError("func resp is nil")
	}
	return nil
}

func (o *BatchOption[A, B]) validVersions() []bool {
	valids := make([]bool, len(o.VersionIDs))
	for i, versionID := range o.VersionIDs {
		objID, err := primitive.ObjectIDFromHex(versionID)
		valids[i] = (err == nil && (!objID.IsZero()) && o.VersionNumbers[i] > 0)
	}
	return valids
}

func (o *BatchOption[A, B]) equalIDs(objIDs []primitive.ObjectID) []bool {
	equals := make([]bool, len(o.VersionIDs))
	for i, versionID := range o.VersionIDs {
		equals[i] = versionID == objIDs[i].Hex()
	}
	return equals
```

```
}

func (o *BatchOption[A, B]) getVersions(tags *[]int) (versions map[string]*model.VersionLog, err error) {
	var dIDs []string
	var versionNums []uint64
	var limits []int

	valids := o.validVersions()

	if o.CacheMaxVersions == nil {
		for i, valid := range valids {
			if valid {
				(*tags)[i] = tagQuery
				dIDs = append(dIDs, o.TargetKeys[i])
				versionNums = append(versionNums, o.VersionNumbers[i])
				limits = append(limits, syncLimit)
			} else {
				(*tags)[i] = tagFull
				dIDs = append(dIDs, o.TargetKeys[i])
				versionNums = append(versionNums, 0)
				limits = append(limits, 0)
			}
		}

		versions, err = o.Versions(o.Ctx, dIDs, versionNums, limits)
		if err != nil {
			return nil, errs.Wrap(err)
		}
		return versions, nil

	} else {
		caches, err := o.CacheMaxVersions(o.Ctx, o.TargetKeys)
		if err != nil {
			return nil, errs.Wrap(err)
		}

		objIDs := make([]primitive.ObjectID, len(o.VersionIDs))

		for i, versionID := range o.VersionIDs {
			objID, _ := primitive.ObjectIDFromHex(versionID)
			objIDs[i] = objID
		}

		equals := o.equalIDs(objIDs)
		for i, valid := range valids {
			if !valid {
				(*tags)[i] = tagFull
			} else if !equals[i] {
				(*tags)[i] = tagFull
			} else if o.VersionNumbers[i] == uint64(caches[o.TargetKeys[i]].Version) {
				(*tags)[i] = tagEqual
			} else {
				(*tags)[i] = tagQuery
				dIDs = append(dIDs, o.TargetKeys[i])
				versionNums = append(versionNums, o.VersionNumbers[i])
				limits = append(limits, syncLimit)

				delete(caches, o.TargetKeys[i])
			}
		}

		if dIDs != nil {
			versionMap, err := o.Versions(o.Ctx, dIDs, versionNums, limits)
			if err != nil {
				return nil, errs.Wrap(err)
			}
```

```
        for k, v := range versionMap {
            caches[k] = v
        }
    }

    versions = caches
  }
  return versions, nil
}

func (o *BatchOption[A, B]) Build() (*B, error) {
  if err := o.check(); err != nil {
    return nil, errs.Wrap(err)
  }

  tags := make([]int, len(o.TargetKeys))
  versions, err := o.getVersions(&tags)
  if err != nil {
    return nil, errs.Wrap(err)
  }

  fullMap := make(map[string]bool)
  for i, tag := range tags {
    switch tag {
    case tagQuery:
      vLog := versions[o.TargetKeys[i]]
      fullMap[o.TargetKeys[i]] = vLog.ID.Hex() != o.VersionIDs[i] || uint64(vLog.Version) < o.VersionNumbers[i] || len
    case tagFull:
      fullMap[o.TargetKeys[i]] = true
    case tagEqual:
      fullMap[o.TargetKeys[i]] = false
    default:
      panic(fmt.Errorf("undefined tag %d", tag))
    }
  }

  var (
    insertIdsMap = make(map[string][]string)
    deleteIdsMap = make(map[string][]string)
    updateIdsMap = make(map[string][]string)
  )

  for _, targetKey := range o.TargetKeys {
    if !fullMap[targetKey] {
      version := versions[targetKey]
      insertIds, deleteIds, updateIds := version.DeleteAndChangeIDs()
      insertIdsMap[targetKey] = insertIds
      deleteIdsMap[targetKey] = deleteIds
      updateIdsMap[targetKey] = updateIds
    }
  }

  var (
    insertListMap = make(map[string]A)
    updateListMap = make(map[string]A)
  )

  for targetKey, insertIds := range insertIdsMap {
    if len(insertIds) > 0 {
      insertList, err := o.Find(o.Ctx, targetKey, insertIds)
      if err != nil {
        return nil, errs.Wrap(err)
      }
      insertListMap[targetKey] = insertList
    }
```

```
	}

	for targetKey, updateIds := range updateIdsMap {
		if len(updateIds) > 0 {
			updateList, err := o.Find(o.Ctx, targetKey, updateIds)
			if err != nil {
				return nil, errs.Wrap(err)
			}
			updateListMap[targetKey] = updateList
		}
	}

	return o.Resp(versions, deleteIdsMap, insertListMap, updateListMap, fullMap), nil
}
```

internal/rpc/incrversion/option.go

```
package incrversion

import (
	"context"
	"fmt"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/tools/errs"
	"go.mongodb.org/mongo-driver/bson/primitive"
)

//func Limit(maxSync int, version uint64) int {
//	if version == 0 {
//		return 0
//	}
//	return maxSync
//}

const syncLimit = 200

const (
	tagQuery = iota + 1
	tagFull
	tagEqual
)

type Option[A, B any] struct {
	Ctx           context.Context
	VersionKey    string
	VersionID     string
	VersionNumber uint64
	//SyncLimit       int
	CacheMaxVersion func(ctx context.Context, dId string) (*model.VersionLog, error)
	Version         func(ctx context.Context, dId string, version uint, limit int) (*model.VersionLog, error)
	//SortID          func(ctx context.Context, dId string) ([]string, error)
	Find func(ctx context.Context, ids []string) ([]A, error)
	Resp func(version *model.VersionLog, deleteIds []string, insertList, updateList []A, full bool) *B
}

func (o *Option[A, B]) newError(msg string) error {
	return errs.ErrInternalServer.WrapMsg(msg)
}

func (o *Option[A, B]) check() error {
	if o.Ctx == nil {
		return o.newError("opt ctx is nil")
	}
	if o.VersionKey == "" {
		return o.newError("versionKey is empty")
	}
	//if o.SyncLimit <= 0 {
	//	return o.newError("invalid synchronization quantity")
	//}
	if o.Version == nil {
		return o.newError("func version is nil")
	}
	//if o.SortID == nil {
	//	return o.newError("func allID is nil")
	//}
	if o.Find == nil {
		return o.newError("func find is nil")
	}
	if o.Resp == nil {
		return o.newError("func resp is nil")
```

```
	}
	return nil
}

func (o *Option[A, B]) validVersion() bool {
	objID, err := primitive.ObjectIDFromHex(o.VersionID)
	return err == nil && (!objID.IsZero()) && o.VersionNumber > 0
}

func (o *Option[A, B]) equalID(objID primitive.ObjectID) bool {
	return o.VersionID == objID.Hex()
}

func (o *Option[A, B]) getVersion(tag *int) (*model.VersionLog, error) {
	if o.CacheMaxVersion == nil {
		if o.validVersion() {
			*tag = tagQuery
			return o.Version(o.Ctx, o.VersionKey, uint(o.VersionNumber), syncLimit)
		}
		*tag = tagFull
		return o.Version(o.Ctx, o.VersionKey, 0, 0)
	} else {
		cache, err := o.CacheMaxVersion(o.Ctx, o.VersionKey)
		if err != nil {
			return nil, err
		}
		if !o.validVersion() {
			*tag = tagFull
			return cache, nil
		}
		if !o.equalID(cache.ID) {
			*tag = tagFull
			return cache, nil
		}
		if o.VersionNumber == uint64(cache.Version) {
			*tag = tagEqual
			return cache, nil
		}
		*tag = tagQuery
		return o.Version(o.Ctx, o.VersionKey, uint(o.VersionNumber), syncLimit)
	}
}

func (o *Option[A, B]) Build() (*B, error) {
	if err := o.check(); err != nil {
		return nil, err
	}
	var tag int
	version, err := o.getVersion(&tag)
	if err != nil {
		return nil, err
	}
	var full bool
	switch tag {
	case tagQuery:
		full = version.ID.Hex() != o.VersionID || uint64(version.Version) < o.VersionNumber || len(version.Logs) != versi
	case tagFull:
		full = true
	case tagEqual:
		full = false
	default:
		panic(fmt.Errorf("undefined tag %d", tag))
	}
	var (
		insertIds []string
		deleteIds []string
```

```
  updateIds []string
 )
 if !full {
  insertIds, deleteIds, updateIds = version.DeleteAndChangeIDs()
 }
 var (
  insertList []A
  updateList []A
 )
 if len(insertIds) > 0 {
  insertList, err = o.Find(o.Ctx, insertIds)
  if err != nil {
   return nil, err
  }
 }
 if len(updateIds) > 0 {
  updateList, err = o.Find(o.Ctx, updateIds)
  if err != nil {
   return nil, err
  }
 }
 return o.Resp(version, deleteIds, insertList, updateList, full), nil
}
```

internal/rpc/auth

internal/rpc/auth/auth.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package auth

import (
"context"
"errors"

"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
"github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
"github.com/openimsdk/open-im-server/v3/pkg/localcache"
"github.com/openimsdk/open-im-server/v3/pkg/rpccache"
"github.com/openimsdk/open-im-server/v3/pkg/rpcli"

"github.com/openimsdk/open-im-server/v3/pkg/common/config"
redis2 "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
"github.com/openimsdk/tools/utils/datautil"
"github.com/redis/go-redis/v9"

"github.com/openimsdk/open-im-server/v3/pkg/authverify"
"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
pbauth "github.com/openimsdk/protocol/auth"
"github.com/openimsdk/protocol/constant"
"github.com/openimsdk/protocol/msggateway"
"github.com/openimsdk/tools/discovery"
"github.com/openimsdk/tools/errs"
"github.com/openimsdk/tools/log"
"github.com/openimsdk/tools/tokenverify"
"google.golang.org/grpc"
)

type authServer struct {
pbauth.UnimplementedAuthServer
authDatabase   controller.AuthDatabase
AuthLocalCache *rpccache.AuthLocalCache
RegisterCenter discovery.Conn
config         *Config
userClient     *rpcli.UserClient
adminUserIDs   []string
}

type Config struct {
RpcConfig        config.Auth
RedisConfig      config.Redis
```

```
	MongoConfig      config.Mongo
	Share            config.Share
	LocalCacheConfig config.LocalCache
	Discovery        config.Discovery
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
	dbb := dbbuild.NewBuilder(&config.MongoConfig, &config.RedisConfig)
	rdb, err := dbb.Redis(ctx)
	if err != nil {
		return err
	}
	var token cache.TokenModel
	if rdb == nil {
		mdb, err := dbb.Mongo(ctx)
		if err != nil {
			return err
		}
		mc, err := mgo.NewCacheMgo(mdb.GetDB())
		if err != nil {
			return err
		}
		token = mcache.NewTokenCacheModel(mc, config.RpcConfig.TokenPolicy.Expire)
	} else {
		token = redis2.NewTokenCacheModel(rdb, &config.LocalCacheConfig, config.RpcConfig.TokenPolicy.Expire)
	}
	userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
	if err != nil {
		return err
	}
	authConn, err := client.GetConn(ctx, config.Discovery.RpcService.Auth)
	if err != nil {
		return err
	}

	localcache.InitLocalCache(&config.LocalCacheConfig)

	pbauth.RegisterAuthServer(server, &authServer{
		RegisterCenter: client,
		authDatabase: controller.NewAuthDatabase(
			token,
			config.Share.Secret,
			config.RpcConfig.TokenPolicy.Expire,
			config.Share.MultiLogin,
			config.Share.IMAdminUser.UserIDs,
		),
		AuthLocalCache: rpccache.NewAuthLocalCache(rpcli.NewAuthClient(authConn), &config.LocalCacheConfig, rdb),
		config:         config,
		userClient:     rpcli.NewUserClient(userConn),
		adminUserIDs:   config.Share.IMAdminUser.UserIDs,
	})
	return nil
}

func (s *authServer) GetAdminToken(ctx context.Context, req *pbauth.GetAdminTokenReq) (*pbauth.GetAdminTokenResp, er
	resp := pbauth.GetAdminTokenResp{}
	if req.Secret != s.config.Share.Secret {
		return nil, errs.ErrNoPermission.WrapMsg("secret invalid")
	}

	if !datautil.Contain(req.UserID, s.adminUserIDs...) {
		return nil, errs.ErrArgs.WrapMsg("userID is error.", "userID", req.UserID, "adminUserID", s.adminUserIDs)

	}

	if err := s.userClient.CheckUser(ctx, []string{req.UserID}); err != nil {
```

```
    return nil, err
  }

  token, err := s.authDatabase.CreateToken(ctx, req.UserID, int(constant.AdminPlatformID))
  if err != nil {
    return nil, err
  }

  prommetrics.UserLoginCounter.Inc()

  resp.Token = token
  resp.ExpireTimeSeconds = s.config.RpcConfig.TokenPolicy.Expire * 24 * 60 * 60
  return &resp, nil
}

func (s *authServer) GetUserToken(ctx context.Context, req *pbauth.GetUserTokenReq) (*pbauth.GetUserTokenResp, error
  if err := authverify.CheckAdmin(ctx); err != nil {
    return nil, err
  }

  if req.PlatformID == constant.AdminPlatformID {
    return nil, errs.ErrNoPermission.WrapMsg("platformID invalid. platformID must not be adminPlatformID")
  }

  resp := pbauth.GetUserTokenResp{}

  if authverify.CheckUserIsAdmin(ctx, req.UserID) {
    return nil, errs.ErrNoPermission.WrapMsg("don't get Admin token")
  }
  user, err := s.userClient.GetUserInfo(ctx, req.UserID)
  if err != nil {
    return nil, err
  }
  if user.AppMangerLevel >= constant.AppNotificationAdmin {
    return nil, errs.ErrArgs.WrapMsg("app account can`t get token")
  }
  token, err := s.authDatabase.CreateToken(ctx, req.UserID, int(req.PlatformID))
  if err != nil {
    return nil, err
  }

  resp.Token = token
  resp.ExpireTimeSeconds = s.config.RpcConfig.TokenPolicy.Expire * 24 * 60 * 60
  return &resp, nil
}

func (s *authServer) GetExistingToken(ctx context.Context, req *pbauth.GetExistingTokenReq) (*pbauth.GetExistingToke
  m, err := s.authDatabase.GetTokensWithoutError(ctx, req.UserID, int(req.PlatformID))
  if err != nil {
    return nil, err
  }

  return &pbauth.GetExistingTokenResp{
    TokenStates: convert.TokenMapDB2Pb(m),
  }, nil
}

func (s *authServer) parseToken(ctx context.Context, tokensString string) (claims *tokenverify.Claims, err error) {
  claims, err = tokenverify.GetClaimFromToken(tokensString, authverify.Secret(s.config.Share.Secret))
  if err != nil {
    return nil, err
  }

  m, err := s.AuthLocalCache.GetExistingToken(ctx, claims.UserID, claims.PlatformID)
  if err != nil {
    return nil, err
```

```
	}

	if len(m) == 0 {
		isAdmin := authverify.CheckUserIsAdmin(ctx, claims.UserID)
		if isAdmin {
			if err = s.authDatabase.GetTemporaryTokensWithoutError(ctx, claims.UserID, claims.PlatformID, tokensString); err
				return claims, nil
			}
		}
		return nil, servererrs.ErrTokenNotExist.Wrap()
	}
	if v, ok := m[tokensString]; ok {
		switch v {
		case constant.NormalToken:
			return claims, nil
		case constant.KickedToken:
			return nil, servererrs.ErrTokenKicked.Wrap()
		default:
			return nil, errs.Wrap(errs.ErrTokenUnknown)
		}
	} else {
		isAdmin := authverify.CheckUserIsAdmin(ctx, claims.UserID)
		if isAdmin {
			if err = s.authDatabase.GetTemporaryTokensWithoutError(ctx, claims.UserID, claims.PlatformID, tokensString); err
				return claims, nil
			}
		}
	}
	return nil, servererrs.ErrTokenNotExist.Wrap()
}

func (s *authServer) ParseToken(ctx context.Context, req *pbauth.ParseTokenReq) (resp *pbauth.ParseTokenResp, err er
	resp = &pbauth.ParseTokenResp{}
	claims, err := s.parseToken(ctx, req.Token)
	if err != nil {
		return nil, err
	}
	resp.UserID = claims.UserID
	resp.PlatformID = int32(claims.PlatformID)
	resp.ExpireTimeSeconds = claims.ExpiresAt.Unix()
	return resp, nil
}

func (s *authServer) ForceLogout(ctx context.Context, req *pbauth.ForceLogoutReq) (*pbauth.ForceLogoutResp, error) {
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	if err := s.forceKickOff(ctx, req.UserID, req.PlatformID); err != nil {
		return nil, err
	}
	return &pbauth.ForceLogoutResp{}, nil
}

func (s *authServer) forceKickOff(ctx context.Context, userID string, platformID int32) error {
	conns, err := s.RegisterCenter.GetConns(ctx, s.config.Discovery.RpcService.MessageGateway)
	if err != nil {
		return err
	}
	for _, v := range conns {
		log.ZDebug(ctx, "forceKickOff", "userID", userID, "platformID", platformID)
		client := msggateway.NewMsgGatewayClient(v)
		kickReq := &msggateway.KickUserOfflineReq{KickUserIDList: []string{userID}, PlatformID: platformID}
		_, err := client.KickUserOffline(ctx, kickReq)
		if err != nil {
			log.ZError(ctx, "forceKickOff", err, "kickReq", kickReq)
		}
```

```
	}

	m, err := s.authDatabase.GetTokensWithoutError(ctx, userID, int(platformID))
	if err != nil && !errors.Is(err, redis.Nil) {
		return err
	}
	for k := range m {
		m[k] = constant.KickedToken
		log.ZDebug(ctx, "set token map is ", "token map", m, "userID",
			userID, "token", k)

		err = s.authDatabase.SetTokenMapByUidPid(ctx, userID, int(platformID), m)
		if err != nil {
			return err
		}
	}
	return nil
}

func (s *authServer) InvalidateToken(ctx context.Context, req *pbauth.InvalidateTokenReq) (*pbauth.InvalidateTokenRe
	m, err := s.authDatabase.GetTokensWithoutError(ctx, req.UserID, int(req.PlatformID))
	if err != nil && !errors.Is(err, redis.Nil) {
		return nil, err
	}
	if m == nil {
		return nil, errs.New("token map is empty").Wrap()
	}
	log.ZDebug(ctx, "get token from redis", "userID", req.UserID, "platformID",
		req.PlatformID, "tokenMap", m)

	for k := range m {
		if k != req.GetPreservedToken() {
			m[k] = constant.KickedToken
		}
	}
	log.ZDebug(ctx, "set token map is ", "token map", m, "userID",
		req.UserID, "token", req.GetPreservedToken())
	err = s.authDatabase.SetTokenMapByUidPid(ctx, req.UserID, int(req.PlatformID), m)
	if err != nil {
		return nil, err
	}
	return &pbauth.InvalidateTokenResp{}, nil
}

func (s *authServer) KickTokens(ctx context.Context, req *pbauth.KickTokensReq) (*pbauth.KickTokensResp, error) {
	if err := s.authDatabase.BatchSetTokenMapByUidPid(ctx, req.Tokens); err != nil {
		return nil, err
	}
	return &pbauth.KickTokensResp{}, nil
}
```

internal/rpc/conversation

internal/rpc/conversation/callback.go

```
package conversation

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	dbModel "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/tools/utils/datautil"
)

func (c *conversationServer) webhookBeforeCreateSingleChatConversations(ctx context.Context, before *config.BeforeCo
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &callbackstruct.CallbackBeforeCreateSingleChatConversationsReq{
			CallbackCommand:  callbackstruct.CallbackBeforeCreateSingleChatConversationsCommand,
			OwnerUserID:      req.OwnerUserID,
			ConversationID:   req.ConversationID,
			ConversationType: req.ConversationType,
			UserID:           req.UserID,
			RecvMsgOpt:       req.RecvMsgOpt,
			IsPinned:         req.IsPinned,
			IsPrivateChat:    req.IsPrivateChat,
			BurnDuration:     req.BurnDuration,
			GroupAtType:      req.GroupAtType,
			AttachedInfo:     req.AttachedInfo,
			Ex:               req.Ex,
		}

		resp := &callbackstruct.CallbackBeforeCreateSingleChatConversationsResp{}

		if err := c.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
			return err
		}

		datautil.NotNilReplace(&req.RecvMsgOpt, resp.RecvMsgOpt)
		datautil.NotNilReplace(&req.IsPinned, resp.IsPinned)
		datautil.NotNilReplace(&req.IsPrivateChat, resp.IsPrivateChat)
		datautil.NotNilReplace(&req.BurnDuration, resp.BurnDuration)
		datautil.NotNilReplace(&req.GroupAtType, resp.GroupAtType)
		datautil.NotNilReplace(&req.AttachedInfo, resp.AttachedInfo)
		datautil.NotNilReplace(&req.Ex, resp.Ex)
		return nil
	})
}

func (c *conversationServer) webhookAfterCreateSingleChatConversations(ctx context.Context, after *config.AfterConfi
	cbReq := &callbackstruct.CallbackAfterCreateSingleChatConversationsReq{
		CallbackCommand:  callbackstruct.CallbackAfterCreateSingleChatConversationsCommand,
		OwnerUserID:      req.OwnerUserID,
		ConversationID:   req.ConversationID,
		ConversationType: req.ConversationType,
		UserID:           req.UserID,
		RecvMsgOpt:       req.RecvMsgOpt,
		IsPinned:         req.IsPinned,
		IsPrivateChat:    req.IsPrivateChat,
		BurnDuration:     req.BurnDuration,
		GroupAtType:      req.GroupAtType,
		AttachedInfo:     req.AttachedInfo,
		Ex:               req.Ex,
	}
```

```
    c.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterCreateSingleChatConv
    return nil
}

func (c *conversationServer) webhookBeforeCreateGroupChatConversations(ctx context.Context, before *config.BeforeCon
    return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
        cbReq := &callbackstruct.CallbackBeforeCreateGroupChatConversationsReq{
            CallbackCommand:  callbackstruct.CallbackBeforeCreateGroupChatConversationsCommand,
            ConversationID:   req.ConversationID,
            ConversationType: req.ConversationType,
            GroupID:          req.GroupID,
            RecvMsgOpt:       req.RecvMsgOpt,
            IsPinned:         req.IsPinned,
            IsPrivateChat:    req.IsPrivateChat,
            BurnDuration:     req.BurnDuration,
            GroupAtType:      req.GroupAtType,
            AttachedInfo:     req.AttachedInfo,
            Ex:               req.Ex,
        }

        resp := &callbackstruct.CallbackBeforeCreateGroupChatConversationsResp{}

        if err := c.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
            return err
        }

        datautil.NotNilReplace(&req.RecvMsgOpt, resp.RecvMsgOpt)
        datautil.NotNilReplace(&req.IsPinned, resp.IsPinned)
        datautil.NotNilReplace(&req.IsPrivateChat, resp.IsPrivateChat)
        datautil.NotNilReplace(&req.BurnDuration, resp.BurnDuration)
        datautil.NotNilReplace(&req.GroupAtType, resp.GroupAtType)
        datautil.NotNilReplace(&req.AttachedInfo, resp.AttachedInfo)
        datautil.NotNilReplace(&req.Ex, resp.Ex)
        return nil
    })
}

func (c *conversationServer) webhookAfterCreateGroupChatConversations(ctx context.Context, after *config.AfterConfig
    cbReq := &callbackstruct.CallbackAfterCreateGroupChatConversationsReq{
        CallbackCommand:  callbackstruct.CallbackAfterCreateGroupChatConversationsCommand,
        ConversationID:   req.ConversationID,
        ConversationType: req.ConversationType,
        GroupID:          req.GroupID,
        RecvMsgOpt:       req.RecvMsgOpt,
        IsPinned:         req.IsPinned,
        IsPrivateChat:    req.IsPrivateChat,
        BurnDuration:     req.BurnDuration,
        GroupAtType:      req.GroupAtType,
        AttachedInfo:     req.AttachedInfo,
        Ex:               req.Ex,
    }

    c.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterCreateGroupChatConve
    return nil
}
```

internal/rpc/conversation/conversation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package conversation

import (
    "context"
    "sort"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"

    "google.golang.org/grpc"

    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/convert"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
    dbModel "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
    "github.com/openimsdk/open-im-server/v3/pkg/localcache"
    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/protocol/constant"
    pbconversation "github.com/openimsdk/protocol/conversation"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/datautil"
)

type conversationServer struct {
    pbconversation.UnimplementedConversationServer
    conversationDatabase controller.ConversationDatabase

    conversationNotificationSender *ConversationNotificationSender
    config                         *Config

    webhookClient *webhook.Client
    userClient    *rpcli.UserClient
    msgClient     *rpcli.MsgClient
    groupClient   *rpcli.GroupClient
}

type Config struct {
    RpcConfig          config.Conversation
    RedisConfig        config.Redis
    MongodbConfig      config.Mongo
```

```
	NotificationConfig config.Notification
	Share              config.Share
	WebhooksConfig     config.Webhooks
	LocalCacheConfig   config.LocalCache
	Discovery          config.Discovery
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
	dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
	mgocli, err := dbb.Mongo(ctx)
	if err != nil {
		return err
	}
	rdb, err := dbb.Redis(ctx)
	if err != nil {
		return err
	}
	conversationDB, err := mgo.NewConversationMongo(mgocli.GetDB())
	if err != nil {
		return err
	}
	userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
	if err != nil {
		return err
	}
	groupConn, err := client.GetConn(ctx, config.Discovery.RpcService.Group)
	if err != nil {
		return err
	}
	msgConn, err := client.GetConn(ctx, config.Discovery.RpcService.Msg)
	if err != nil {
		return err
	}

	msgClient := rpcli.NewMsgClient(msgConn)

	cs := conversationServer{
		config:        config,
		webhookClient: webhook.NewWebhookClient(config.WebhooksConfig.URL),
		userClient:    rpcli.NewUserClient(userConn),
		groupClient:   rpcli.NewGroupClient(groupConn),
		msgClient:     msgClient,
	}

	cs.conversationNotificationSender = NewConversationNotificationSender(&config.NotificationConfig, msgClient)
	cs.conversationDatabase = controller.NewConversationDatabase(
		conversationDB,
		redis.NewConversationRedis(rdb, &config.LocalCacheConfig, conversationDB),
		mgocli.GetTx())

	localcache.InitLocalCache(&config.LocalCacheConfig)
	pbconversation.RegisterConversationServer(server, &cs)
	return nil
}

func (c *conversationServer) GetConversation(ctx context.Context, req *pbconversation.GetConversationReq) (*pbconver
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}
	conversations, err := c.conversationDatabase.FindConversations(ctx, req.OwnerUserID, []string{req.ConversationID})
	if err != nil {
		return nil, err
	}
	if len(conversations) < 1 {
		return nil, errs.ErrRecordNotFound.WrapMsg("conversation not found")
	}
```

```
	resp := &pbconversation.GetConversationResp{Conversation: &pbconversation.Conversation{}}
	resp.Conversation = convert.ConversationDB2Pb(conversations[0])
	return resp, nil
}

// Deprecated: Use `GetConversations` instead.
func (c *conversationServer) GetSortedConversationList(ctx context.Context, req *pbconversation.GetSortedConversatio
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	var conversationIDs []string
	if len(req.ConversationIDs) == 0 {
		conversationIDs, err = c.conversationDatabase.GetConversationIDs(ctx, req.UserID)
		if err != nil {
			return nil, err
		}
	} else {
		conversationIDs = req.ConversationIDs
	}

	conversations, err := c.conversationDatabase.FindConversations(ctx, req.UserID, conversationIDs)
	if err != nil {
		return nil, err
	}
	if len(conversations) == 0 {
		return nil, errs.ErrRecordNotFound.Wrap()
	}
	maxSeqs, err := c.msgClient.GetMaxSeqs(ctx, conversationIDs)
	if err != nil {
		return nil, err
	}

	chatLogs, err := c.msgClient.GetMsgByConversationIDs(ctx, conversationIDs, maxSeqs)
	if err != nil {
		return nil, err
	}

	conversationMsg, err := c.getConversationInfo(ctx, chatLogs, req.UserID)
	if err != nil {
		return nil, err
	}

	hasReadSeqs, err := c.msgClient.GetHasReadSeqs(ctx, conversationIDs, req.UserID)
	if err != nil {
		return nil, err
	}

	var unreadTotal int64
	conversation_unreadCount := make(map[string]int64)
	for conversationID, maxSeq := range maxSeqs {
		unreadCount := maxSeq - hasReadSeqs[conversationID]
		conversation_unreadCount[conversationID] = unreadCount
		unreadTotal += unreadCount
	}

	conversation_isPinTime := make(map[int64]string)
	conversation_notPinTime := make(map[int64]string)

	for _, v := range conversations {
		conversationID := v.ConversationID
		var time int64
		if _, ok := conversationMsg[conversationID]; ok {
			time = conversationMsg[conversationID].MsgInfo.LatestMsgRecvTime
		} else {
			conversationMsg[conversationID] = &pbconversation.ConversationElem{
				ConversationID: conversationID,
```

```
        IsPinned:       v.IsPinned,
        MsgInfo:        nil,
    }
    time = v.CreateTime.UnixMilli()
  }

  conversationMsg[conversationID].RecvMsgOpt = v.RecvMsgOpt
  if v.IsPinned {
    conversationMsg[conversationID].IsPinned = v.IsPinned
    conversation_isPinTime[time] = conversationID
    continue
  }
  conversation_notPinTime[time] = conversationID
 }
 resp = &pbconversation.GetSortedConversationListResp{
  ConversationTotal: int64(len(chatLogs)),
  ConversationElems: []*pbconversation.ConversationElem{},
  UnreadTotal:       unreadTotal,
 }

 c.conversationSort(conversation_isPinTime, resp, conversation_unreadCount, conversationMsg)
 c.conversationSort(conversation_notPinTime, resp, conversation_unreadCount, conversationMsg)

 resp.ConversationElems = datautil.Paginate(resp.ConversationElems, int(req.Pagination.GetPageNumber()), int(req.Pag
 return resp, nil
}

func (c *conversationServer) GetAllConversations(ctx context.Context, req *pbconversation.GetAllConversationsReq) (*
 if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
  return nil, err
 }
 conversations, err := c.conversationDatabase.GetUserAllConversation(ctx, req.OwnerUserID)
 if err != nil {
  return nil, err
 }
 resp := &pbconversation.GetAllConversationsResp{Conversations: []*pbconversation.Conversation{}}
 resp.Conversations = convert.ConversationsDB2Pb(conversations)
 return resp, nil
}

func (c *conversationServer) GetConversations(ctx context.Context, req *pbconversation.GetConversationsReq) (*pbconv
 if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
  return nil, err
 }
 conversations, err := c.getConversations(ctx, req.OwnerUserID, req.ConversationIDs)
 if err != nil {
  return nil, err
 }
 return &pbconversation.GetConversationsResp{
  Conversations: conversations,
 }, nil
}

func (c *conversationServer) getConversations(ctx context.Context, ownerUserID string, conversationIDs []string) ([]
 conversations, err := c.conversationDatabase.FindConversations(ctx, ownerUserID, conversationIDs)
 if err != nil {
  return nil, err
 }
 resp := &pbconversation.GetConversationsResp{Conversations: []*pbconversation.Conversation{}}
 resp.Conversations = convert.ConversationsDB2Pb(conversations)
 return convert.ConversationsDB2Pb(conversations), nil
}

// Deprecated
func (c *conversationServer) SetConversation(ctx context.Context, req *pbconversation.SetConversationReq) (*pbconver
 if err := authverify.CheckAccess(ctx, req.GetConversation().GetUserID()); err != nil {
```

```
		return nil, err
	}
	var conversation dbModel.Conversation
	conversation.CreateTime = time.Now()

	if err := datautil.CopyStructFields(&conversation, req.Conversation); err != nil {
		return nil, err
	}
	err := c.conversationDatabase.SetUserConversations(ctx, req.Conversation.OwnerUserID, []*dbModel.Conversation{&conv
	if err != nil {
		return nil, err
	}
	c.conversationNotificationSender.ConversationChangeNotification(ctx, req.Conversation.OwnerUserID, []string{req.Cor
	resp := &pbconversation.SetConversationResp{}
	return resp, nil
}

func (c *conversationServer) SetConversations(ctx context.Context, req *pbconversation.SetConversationsReq) (*pbconv
	for _, userID := range req.UserIDs {
		if err := authverify.CheckAccess(ctx, userID); err != nil {
			return nil, err
		}
	}
	if req.Conversation.ConversationType == constant.WriteGroupChatType {
		groupInfo, err := c.groupClient.GetGroupInfo(ctx, req.Conversation.GroupID)
		if err != nil {
			return nil, err
		}
		if groupInfo == nil {
			return nil, servererrs.ErrGroupIDNotFound.WrapMsg(req.Conversation.GroupID)
		}
		if groupInfo.Status == constant.GroupStatusDismissed {
			return nil, servererrs.ErrDismissedAlready.WrapMsg("group dismissed")
		}
	}

	conversationMap := make(map[string]*dbModel.Conversation)
	var needUpdateUsersList []string

	for _, userID := range req.UserIDs {
		conversationList, err := c.conversationDatabase.FindConversations(ctx, userID, []string{req.Conversation.Conversa
		if err != nil {
			return nil, err
		}
		if len(conversationList) != 0 {
			conversationMap[userID] = conversationList[0]
		} else {
			needUpdateUsersList = append(needUpdateUsersList, userID)
		}
	}

	var conversation dbModel.Conversation
	conversation.ConversationID = req.Conversation.ConversationID
	conversation.ConversationType = req.Conversation.ConversationType
	conversation.UserID = req.Conversation.UserID
	conversation.GroupID = req.Conversation.GroupID
	conversation.CreateTime = time.Now()

	m, conversation, err := UpdateConversationsMap(ctx, req)
	if err != nil {
		return nil, err
	}

	for userID := range conversationMap {
		unequal := UserUpdateCheckMap(ctx, userID, req.Conversation, conversationMap[userID])
```

```
    if unequal {
        needUpdateUsersList = append(needUpdateUsersList, userID)
    }
}

if len(m) != 0 && len(needUpdateUsersList) != 0 {
    if err := c.conversationDatabase.SetUsersConversationFieldTx(ctx, needUpdateUsersList, &conversation, m); err != n
        return nil, err
    }

    for _, userID := range needUpdateUsersList {
        c.conversationNotificationSender.ConversationChangeNotification(ctx, userID, []string{req.Conversation.Conversat
    }
}

if req.Conversation.IsPrivateChat != nil && req.Conversation.ConversationType != constant.ReadGroupChatType {
    var conversations []*dbModel.Conversation
    for _, ownerUserID := range req.UserIDs {
        transConversation := conversation
        transConversation.OwnerUserID = ownerUserID
        transConversation.IsPrivateChat = req.Conversation.IsPrivateChat.Value
        conversations = append(conversations, &transConversation)
    }

    if err := c.conversationDatabase.SyncPeerUserPrivateConversationTx(ctx, conversations); err != nil {
        return nil, err
    }

    for _, userID := range req.UserIDs {
        c.conversationNotificationSender.ConversationSetPrivateNotification(ctx, userID, req.Conversation.UserID,
            req.Conversation.IsPrivateChat.Value, req.Conversation.ConversationID)
    }
}

return &pbconversation.SetConversationsResp{}, nil
}

func (c *conversationServer) UpdateConversationsByUser(ctx context.Context, req *pbconversation.UpdateConversationsB
if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
    return nil, err
}
m := make(map[string]any)
if req.Ex != nil {
    m["ex"] = req.Ex.Value
}
if len(m) > 0 {
    if err := c.conversationDatabase.UpdateUserConversations(ctx, req.UserID, m); err != nil {
        return nil, err
    }
}
return &pbconversation.UpdateConversationsByUserResp{}, nil
}

// create conversation without notification for msg redis transfer.
func (c *conversationServer) CreateSingleChatConversations(ctx context.Context, req *pbconversation.CreateSingleChat
var conversation dbModel.Conversation
conversation.CreateTime = time.Now()

switch req.ConversationType {
case constant.SingleChatType:
    // sendUser create
    conversation.ConversationID = req.ConversationID
    conversation.ConversationType = req.ConversationType
    conversation.OwnerUserID = req.SendID
    conversation.UserID = req.RecvID
```

```
    if err := c.webhookBeforeCreateSingleChatConversations(ctx, &c.config.WebhooksConfig.BeforeCreateSingleChatConvers
        return nil, err
    }

    err := c.conversationDatabase.CreateConversation(ctx, []*dbModel.Conversation{&conversation})
    if err != nil {
        log.ZWarn(ctx, "create conversation failed", err, "conversation", conversation)
    }

    c.webhookAfterCreateSingleChatConversations(ctx, &c.config.WebhooksConfig.AfterCreateSingleChatConversations, &co

    // recvUser create
    conversation2 := conversation
    conversation2.OwnerUserID = req.RecvID
    conversation2.UserID = req.SendID

    if err := c.webhookBeforeCreateSingleChatConversations(ctx, &c.config.WebhooksConfig.BeforeCreateSingleChatConvers
        return nil, err
    }

    err = c.conversationDatabase.CreateConversation(ctx, []*dbModel.Conversation{&conversation2})
    if err != nil {
        log.ZWarn(ctx, "create conversation failed", err, "conversation2", conversation)
    }

    c.webhookAfterCreateSingleChatConversations(ctx, &c.config.WebhooksConfig.AfterCreateSingleChatConversations, &co
case constant.NotificationChatType:
    conversation.ConversationID = req.ConversationID
    conversation.ConversationType = req.ConversationType
    conversation.OwnerUserID = req.RecvID
    conversation.UserID = req.SendID

    if err := c.webhookBeforeCreateSingleChatConversations(ctx, &c.config.WebhooksConfig.BeforeCreateSingleChatConvers
        return nil, err
    }

    err := c.conversationDatabase.CreateConversation(ctx, []*dbModel.Conversation{&conversation})
    if err != nil {
        log.ZWarn(ctx, "create conversation failed", err, "conversation2", conversation)
    }

    c.webhookAfterCreateSingleChatConversations(ctx, &c.config.WebhooksConfig.AfterCreateSingleChatConversations, &co
}

return &pbconversation.CreateSingleChatConversationsResp{}, nil
}

func (c *conversationServer) CreateGroupChatConversations(ctx context.Context, req *pbconversation.CreateGroupChatCo
var conversation dbModel.Conversation

conversation.ConversationID = msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, req.GroupID)
conversation.GroupID = req.GroupID
conversation.ConversationType = constant.ReadGroupChatType
conversation.CreateTime = time.Now()

if err := c.webhookBeforeCreateGroupChatConversations(ctx, &c.config.WebhooksConfig.BeforeCreateGroupChatConversati
    return nil, err
}

err := c.conversationDatabase.CreateGroupChatConversation(ctx, req.GroupID, req.UserIDs, &conversation)
if err != nil {
    return nil, err
}
if err := c.msgClient.SetUserConversationMaxSeq(ctx, conversation.ConversationID, req.UserIDs, 0); err != nil {
    return nil, err
}
```

```
	c.webhookAfterCreateGroupChatConversations(ctx, &c.config.WebhooksConfig.AfterCreateGroupChatConversations, &conver
	return &pbconversation.CreateGroupChatConversationsResp{}, nil
}

func (c *conversationServer) SetConversationMaxSeq(ctx context.Context, req *pbconversation.SetConversationMaxSeqReq
	if err := c.msgClient.SetUserConversationMaxSeq(ctx, req.ConversationID, req.OwnerUserID, req.MaxSeq); err != nil {
		return nil, err
	}
	if err := c.conversationDatabase.UpdateUsersConversationField(ctx, req.OwnerUserID, req.ConversationID,
		map[string]any{"max_seq": req.MaxSeq}); err != nil {
		return nil, err
	}
	for _, userID := range req.OwnerUserID {
		c.conversationNotificationSender.ConversationChangeNotification(ctx, userID, []string{req.ConversationID})
	}
	return &pbconversation.SetConversationMaxSeqResp{}, nil
}

func (c *conversationServer) SetConversationMinSeq(ctx context.Context, req *pbconversation.SetConversationMinSeqReq
	if err := c.msgClient.SetUserConversationMin(ctx, req.ConversationID, req.OwnerUserID, req.MinSeq); err != nil {
		return nil, err
	}
	if err := c.conversationDatabase.UpdateUsersConversationField(ctx, req.OwnerUserID, req.ConversationID,
		map[string]any{"min_seq": req.MinSeq}); err != nil {
		return nil, err
	}
	for _, userID := range req.OwnerUserID {
		c.conversationNotificationSender.ConversationChangeNotification(ctx, userID, []string{req.ConversationID})
	}
	return &pbconversation.SetConversationMinSeqResp{}, nil
}

func (c *conversationServer) GetConversationIDs(ctx context.Context, req *pbconversation.GetConversationIDsReq) (*pb
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	conversationIDs, err := c.conversationDatabase.GetConversationIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	return &pbconversation.GetConversationIDsResp{ConversationIDs: conversationIDs}, nil
}

func (c *conversationServer) GetUserConversationIDsHash(ctx context.Context, req *pbconversation.GetUserConversation
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}
	hash, err := c.conversationDatabase.GetUserConversationIDsHash(ctx, req.OwnerUserID)
	if err != nil {
		return nil, err
	}
	return &pbconversation.GetUserConversationIDsHashResp{Hash: hash}, nil
}

func (c *conversationServer) GetConversationsByConversationID(ctx context.Context, req *pbconversation.GetConversati
	conversations, err := c.conversationDatabase.GetConversationsByConversationID(ctx, req.ConversationIDs)
	if err != nil {
		return nil, err
	}
	return &pbconversation.GetConversationsByConversationIDResp{Conversations: convert.ConversationsDB2Pb(conversations
}

func (c *conversationServer) GetConversationOfflinePushUserIDs(ctx context.Context, req *pbconversation.GetConversat
	if req.ConversationID == "" {
		return nil, errs.ErrArgs.WrapMsg("conversationID is empty")
```

```
	}
	if len(req.UserIDs) == 0 {
		return &pbconversation.GetConversationOfflinePushUserIDsResp{}, nil
	}
	userIDs, err := c.conversationDatabase.GetConversationNotReceiveMessageUserIDs(ctx, req.ConversationID)
	if err != nil {
		return nil, err
	}
	if len(userIDs) == 0 {
		return &pbconversation.GetConversationOfflinePushUserIDsResp{UserIDs: req.UserIDs}, nil
	}
	userIDSet := make(map[string]struct{})
	for _, userID := range req.UserIDs {
		userIDSet[userID] = struct{}{}
	}
	for _, userID := range userIDs {
		delete(userIDSet, userID)
	}
	return &pbconversation.GetConversationOfflinePushUserIDsResp{UserIDs: datautil.Keys(userIDSet)}, nil
}

func (c *conversationServer) conversationSort(conversations map[int64]string, resp *pbconversation.GetSortedConversa
	keys := []int64{}
	for key := range conversations {
		keys = append(keys, key)
	}

	sort.Slice(keys, func(i, j int) bool {
		return keys[i] > keys[j]
	})
	index := 0

	cons := make([]*pbconversation.ConversationElem, len(conversations))
	for _, v := range keys {
		conversationID := conversations[v]
		conversationElem := conversationMsg[conversationID]
		conversationElem.UnreadCount = conversation_unreadCount[conversationID]
		cons[index] = conversationElem
		index++
	}
	resp.ConversationElems = append(resp.ConversationElems, cons...)
}

func (c *conversationServer) getConversationInfo(ctx context.Context, chatLogs map[string]*sdkws.MsgData, userID str
	var (
		sendIDs         []string
		groupIDs        []string
		sendMap         = make(map[string]*sdkws.UserInfo)
		groupMap        = make(map[string]*sdkws.GroupInfo)
		conversationMsg = make(map[string]*pbconversation.ConversationElem)
	)
	for _, chatLog := range chatLogs {
		switch chatLog.SessionType {
		case constant.SingleChatType:
			if chatLog.SendID == userID {
				sendIDs = append(sendIDs, chatLog.RecvID)
			}
			sendIDs = append(sendIDs, chatLog.SendID)
		case constant.WriteGroupChatType, constant.ReadGroupChatType:
			groupIDs = append(groupIDs, chatLog.GroupID)
			sendIDs = append(sendIDs, chatLog.SendID)
		}
	}
	if len(sendIDs) != 0 {
		sendInfos, err := c.userClient.GetUsersInfo(ctx, sendIDs)
		if err != nil {
```

```
        return nil, err
    }
    for _, sendInfo := range sendInfos {
        sendMap[sendInfo.UserID] = sendInfo
    }
}
if len(groupIDs) != 0 {
    groupInfos, err := c.groupClient.GetGroupsInfo(ctx, groupIDs)
    if err != nil {
        return nil, err
    }
    for _, groupInfo := range groupInfos {
        groupMap[groupInfo.GroupID] = groupInfo
    }
}
for conversationID, chatLog := range chatLogs {
    pbchatLog := &pbconversation.ConversationElem{}
    msgInfo := &pbconversation.MsgInfo{}
    if err := datautil.CopyStructFields(msgInfo, chatLog); err != nil {
        return nil, err
    }
    switch chatLog.SessionType {
    case constant.SingleChatType:
        if chatLog.SendID == userID {
            if recv, ok := sendMap[chatLog.RecvID]; ok {
                msgInfo.FaceURL = recv.FaceURL
                msgInfo.SenderName = recv.Nickname
            }
            break
        }
        if send, ok := sendMap[chatLog.SendID]; ok {
            msgInfo.FaceURL = send.FaceURL
            msgInfo.SenderName = send.Nickname
        }
    case constant.WriteGroupChatType, constant.ReadGroupChatType:
        msgInfo.GroupID = chatLog.GroupID
        if group, ok := groupMap[chatLog.GroupID]; ok {
            msgInfo.GroupName = group.GroupName
            msgInfo.GroupFaceURL = group.FaceURL
            msgInfo.GroupMemberCount = group.MemberCount
            msgInfo.GroupType = group.GroupType
        }
        if send, ok := sendMap[chatLog.SendID]; ok {
            msgInfo.SenderName = send.Nickname
        }
    }
    pbchatLog.ConversationID = conversationID
    msgInfo.LatestMsgRecvTime = chatLog.SendTime
    pbchatLog.MsgInfo = msgInfo
    conversationMsg[conversationID] = pbchatLog
}
return conversationMsg, nil
}

func (c *conversationServer) GetConversationNotReceiveMessageUserIDs(ctx context.Context, req *pbconversation.GetCon
userIDs, err := c.conversationDatabase.GetConversationNotReceiveMessageUserIDs(ctx, req.ConversationID)
if err != nil {
    return nil, err
}
return &pbconversation.GetConversationNotReceiveMessageUserIDsResp{UserIDs: userIDs}, nil
}

func (c *conversationServer) UpdateConversation(ctx context.Context, req *pbconversation.UpdateConversationReq) (*pb
for _, userID := range req.UserIDs {
    if err := authverify.CheckAccess(ctx, userID); err != nil {
        return nil, err
```

```
    }
  }
  m := make(map[string]any)
  if req.RecvMsgOpt != nil {
    m["recv_msg_opt"] = req.RecvMsgOpt.Value
  }
  if req.AttachedInfo != nil {
    m["attached_info"] = req.AttachedInfo.Value
  }
  if req.Ex != nil {
    m["ex"] = req.Ex.Value
  }
  if req.IsPinned != nil {
    m["is_pinned"] = req.IsPinned.Value
  }
  if req.GroupAtType != nil {
    m["group_at_type"] = req.GroupAtType.Value
  }
  if req.MsgDestructTime != nil {
    m["msg_destruct_time"] = req.MsgDestructTime.Value
  }
  if req.IsMsgDestruct != nil {
    m["is_msg_destruct"] = req.IsMsgDestruct.Value
  }
  if req.BurnDuration != nil {
    m["burn_duration"] = req.BurnDuration.Value
  }
  if req.IsPrivateChat != nil {
    m["is_private_chat"] = req.IsPrivateChat.Value
  }
  if req.MinSeq != nil {
    m["min_seq"] = req.MinSeq.Value
  }
  if req.MaxSeq != nil {
    m["max_seq"] = req.MaxSeq.Value
  }
  if req.LatestMsgDestructTime != nil {
    m["latest_msg_destruct_time"] = time.UnixMilli(req.LatestMsgDestructTime.Value)
  }
  if len(m) > 0 {
    if err := c.conversationDatabase.UpdateUsersConversationField(ctx, req.UserIDs, req.ConversationID, m); err != ni
      return nil, err
    }
  }
  return &pbconversation.UpdateConversationResp{}, nil
}

func (c *conversationServer) GetOwnerConversation(ctx context.Context, req *pbconversation.GetOwnerConversationReq)
  if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
    return nil, err
  }
  total, conversations, err := c.conversationDatabase.GetOwnerConversation(ctx, req.UserID, req.Pagination)
  if err != nil {
    return nil, err
  }
  return &pbconversation.GetOwnerConversationResp{
    Total:         total,
    Conversations: convert.ConversationsDB2Pb(conversations),
  }, nil
}

func (c *conversationServer) GetConversationsNeedClearMsg(ctx context.Context, _ *pbconversation.GetConversationsNee
  num, err := c.conversationDatabase.GetAllConversationIDsNumber(ctx)
  if err != nil {
    log.ZError(ctx, "GetAllConversationIDsNumber failed", err)
    return nil, err
```

```
	}
	const batchNum = 100

	if num == 0 {
		return nil, errs.New("Need Destruct Msg is nil").Wrap()
	}

	maxPage := (num + batchNum - 1) / batchNum

	temp := make([]*dbModel.Conversation, 0, maxPage*batchNum)

	for pageNumber := 0; pageNumber < int(maxPage); pageNumber++ {
		pagination := &sdkws.RequestPagination{
			PageNumber: int32(pageNumber),
			ShowNumber: batchNum,
		}

		conversationIDs, err := c.conversationDatabase.PageConversationIDs(ctx, pagination)
		if err != nil {
			log.ZError(ctx, "PageConversationIDs failed", err, "pageNumber", pageNumber)
			continue
		}

		// log.ZDebug(ctx, "PageConversationIDs success", "pageNumber", pageNumber, "conversationIDsNum", len(conversatio
		if len(conversationIDs) == 0 {
			continue
		}

		conversations, err := c.conversationDatabase.GetConversationsByConversationID(ctx, conversationIDs)
		if err != nil {
			log.ZError(ctx, "GetConversationsByConversationID failed", err, "conversationIDs", conversationIDs)
			continue
		}

		for _, conversation := range conversations {
			if conversation.IsMsgDestruct && conversation.MsgDestructTime != 0 && ((time.Now().UnixMilli() > (conversation.M
				conversation.LatestMsgDestructTime.IsZero()) {
				temp = append(temp, conversation)
			}
		}
	}

	return &pbconversation.GetConversationsNeedClearMsgResp{Conversations: convert.ConversationsDB2Pb(temp)}, nil
}

func (c *conversationServer) GetNotNotifyConversationIDs(ctx context.Context, req *pbconversation.GetNotNotifyConver
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	conversationIDs, err := c.conversationDatabase.GetNotNotifyConversationIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	return &pbconversation.GetNotNotifyConversationIDsResp{ConversationIDs: conversationIDs}, nil
}

func (c *conversationServer) GetPinnedConversationIDs(ctx context.Context, req *pbconversation.GetPinnedConversation
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	conversationIDs, err := c.conversationDatabase.GetPinnedConversationIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	return &pbconversation.GetPinnedConversationIDsResp{ConversationIDs: conversationIDs}, nil
}
```

```
func (c *conversationServer) ClearUserConversationMsg(ctx context.Context, req *pbconversation.ClearUserConversation
	conversations, err := c.conversationDatabase.FindRandConversation(ctx, req.Timestamp, int(req.Limit))
	if err != nil {
		return nil, err
	}
	latestMsgDestructTime := time.UnixMilli(req.Timestamp)
	for i, conversation := range conversations {
		if !conversation.IsMsgDestruct || conversation.MsgDestructTime == 0 {
			continue
		}
		seq, err := c.msgClient.GetLastMessageSeqByTime(ctx, conversation.ConversationID, req.Timestamp-(conversation.MsgD
		if err != nil {
			return nil, err
		}
		if seq <= 0 {
			log.ZDebug(ctx, "ClearUserConversationMsg GetLastMessageSeqByTime seq <= 0", "index", i, "conversationID", conve
			if err := c.setConversationMinSeqAndLatestMsgDestructTime(ctx, conversation.ConversationID, conversation.OwnerUs
				return nil, err
			}
			continue
		}
		seq++
		if err := c.setConversationMinSeqAndLatestMsgDestructTime(ctx, conversation.ConversationID, conversation.OwnerUse
			return nil, err
		}
		log.ZDebug(ctx, "ClearUserConversationMsg set min seq", "index", i, "conversationID", conversation.ConversationID
	}
	return &pbconversation.ClearUserConversationMsgResp{Count: int32(len(conversations))}, nil
}

func (c *conversationServer) setConversationMinSeqAndLatestMsgDestructTime(ctx context.Context, conversationID strin
	update := map[string]any{
		"latest_msg_destruct_time": latestMsgDestructTime,
	}
	if minSeq >= 0 {
		if err := c.msgClient.SetUserConversationMin(ctx, conversationID, []string{ownerUserID}, minSeq); err != nil {
			return err
		}
		update["min_seq"] = minSeq
	}

	if err := c.conversationDatabase.UpdateUsersConversationField(ctx, []string{ownerUserID}, conversationID, update);
		return err
	}
	c.conversationNotificationSender.ConversationChangeNotification(ctx, ownerUserID, []string{conversationID})
	return nil
}

func (c *conversationServer) DeleteConversations(ctx context.Context, req *pbconversation.DeleteConversationsReq) (r
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}
	if req.NeedDeleteTime == 0 && len(req.ConversationIDs) == 0 {
		return nil, errs.ErrArgs.WrapMsg("need_delete_time or conversationIDs need be set")
	}

	if req.NeedDeleteTime != 0 && len(req.ConversationIDs) != 0 {
		return nil, errs.ErrArgs.WrapMsg("need_delete_time and conversationIDs cannot both be set")
	}

	var needDeleteConversationIDs []string

	if len(req.ConversationIDs) == 0 {
		deleteTimeThreshold := time.Now().AddDate(0, 0, -int(req.NeedDeleteTime)).UnixMilli()
		conversationIDs, err := c.conversationDatabase.GetConversationIDs(ctx, req.OwnerUserID)
```

```
    if err != nil {
        return nil, err
    }
    latestMsgs, err := c.msgClient.GetLastMessage(ctx, &msg.GetLastMessageReq{
        UserID:          req.OwnerUserID,
        ConversationIDs: conversationIDs,
    })
    if err != nil {
        return nil, err
    }

    for conversationID, msg := range latestMsgs.Msgs {
        if msg.SendTime < deleteTimeThreshold {
            needDeleteConversationIDs = append(needDeleteConversationIDs, conversationID)
        }
    }

    if len(needDeleteConversationIDs) == 0 {
        return &pbconversation.DeleteConversationsResp{}, nil
    }
  } else {
    needDeleteConversationIDs = req.ConversationIDs
  }

  if err := c.conversationDatabase.DeleteUsersConversations(ctx, req.OwnerUserID, needDeleteConversationIDs); err !=
    return nil, err
  }

  // c.conversationNotificationSender.ConversationDeleteNotification(ctx, req.OwnerUserID, needDeleteConversationIDs)

  return &pbconversation.DeleteConversationsResp{}, nil
}
```

internal/rpc/conversation/db_map.go

```
package conversation

import (
	"context"

	dbModel "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/conversation"
)

func UpdateConversationsMap(ctx context.Context, req *conversation.SetConversationsReq) (m map[string]any, conversat
	m = make(map[string]any)

	conversation.ConversationID = req.Conversation.ConversationID
	conversation.ConversationType = req.Conversation.ConversationType
	conversation.UserID = req.Conversation.UserID
	conversation.GroupID = req.Conversation.GroupID

	if req.Conversation.RecvMsgOpt != nil {
		conversation.RecvMsgOpt = req.Conversation.RecvMsgOpt.Value
		m["recv_msg_opt"] = req.Conversation.RecvMsgOpt.Value
	}

	if req.Conversation.AttachedInfo != nil {
		conversation.AttachedInfo = req.Conversation.AttachedInfo.Value
		m["attached_info"] = req.Conversation.AttachedInfo.Value
	}

	if req.Conversation.Ex != nil {
		conversation.Ex = req.Conversation.Ex.Value
		m["ex"] = req.Conversation.Ex.Value
	}
	if req.Conversation.IsPinned != nil {
		conversation.IsPinned = req.Conversation.IsPinned.Value
		m["is_pinned"] = req.Conversation.IsPinned.Value
	}
	if req.Conversation.GroupAtType != nil {
		conversation.GroupAtType = req.Conversation.GroupAtType.Value
		m["group_at_type"] = req.Conversation.GroupAtType.Value
	}
	if req.Conversation.MsgDestructTime != nil {
		conversation.MsgDestructTime = req.Conversation.MsgDestructTime.Value
		m["msg_destruct_time"] = req.Conversation.MsgDestructTime.Value
	}
	if req.Conversation.IsMsgDestruct != nil {
		conversation.IsMsgDestruct = req.Conversation.IsMsgDestruct.Value
		m["is_msg_destruct"] = req.Conversation.IsMsgDestruct.Value
	}
	if req.Conversation.BurnDuration != nil {
		conversation.BurnDuration = req.Conversation.BurnDuration.Value
		m["burn_duration"] = req.Conversation.BurnDuration.Value
	}

	return m, conversation, nil
}

func UserUpdateCheckMap(ctx context.Context, userID string, req *conversation.ConversationReq, conversation *dbModel
	unequal = false

	if req.RecvMsgOpt != nil && conversation.RecvMsgOpt != req.RecvMsgOpt.Value {
		unequal = true
	}
	if req.AttachedInfo != nil && conversation.AttachedInfo != req.AttachedInfo.Value {
		unequal = true
	}
```

```
    if req.Ex != nil && conversation.Ex != req.Ex.Value {
        unequal = true
    }
    if req.IsPinned != nil && conversation.IsPinned != req.IsPinned.Value {
        unequal = true
    }
    if req.GroupAtType != nil && conversation.GroupAtType != req.GroupAtType.Value {
        unequal = true
    }
    if req.MsgDestructTime != nil && conversation.MsgDestructTime != req.MsgDestructTime.Value {
        unequal = true
    }
    if req.IsMsgDestruct != nil && conversation.IsMsgDestruct != req.IsMsgDestruct.Value {
        unequal = true
    }
    if req.BurnDuration != nil && conversation.BurnDuration != req.BurnDuration.Value {
        unequal = true
    }

    return unequal
}
```

internal/rpc/conversation/notification.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package conversation

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "github.com/openimsdk/protocol/msg"

    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/notification"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/sdkws"
)

type ConversationNotificationSender struct {
    *notification.NotificationSender
}

func NewConversationNotificationSender(conf *config.Notification, msgClient *rpcli.MsgClient) *ConversationNotificat
    return &ConversationNotificationSender{notification.NewNotificationSender(conf, notification.WithRpcClient(func(ct
        return msgClient.SendMsg(ctx, req)
    }))}
}

// SetPrivate invote.
func (c *ConversationNotificationSender) ConversationSetPrivateNotification(ctx context.Context, sendID, recvID stri
    isPrivateChat bool, conversationID string,
) {
    tips := &sdkws.ConversationSetPrivateTips{
        RecvID:         recvID,
        SendID:         sendID,
        IsPrivate:      isPrivateChat,
        ConversationID: conversationID,
    }

    c.Notification(ctx, sendID, recvID, constant.ConversationPrivateChatNotification, tips)
}

func (c *ConversationNotificationSender) ConversationChangeNotification(ctx context.Context, userID string, conversa
    tips := &sdkws.ConversationUpdateTips{
        UserID:             userID,
        ConversationIDList: conversationIDs,
    }

    c.Notification(ctx, userID, userID, constant.ConversationChangeNotification, tips)
}

func (c *ConversationNotificationSender) ConversationUnreadChangeNotification(
    ctx context.Context,
    userID, conversationID string,
```

```
    unreadCountTime, hasReadSeq int64,
) {
    tips := &sdkws.ConversationHasReadTips{
        UserID:          userID,
        ConversationID:  conversationID,
        HasReadSeq:      hasReadSeq,
        UnreadCountTime: unreadCountTime,
    }

    c.Notification(ctx, userID, userID, constant.ConversationUnreadNotification, tips)
}

func (c *ConversationNotificationSender) ConversationDeleteNotification(ctx context.Context, userID string, conversa
    tips := &sdkws.ConversationDeleteTips{
        UserID:          userID,
        ConversationIDs: conversationIDs,
    }

    c.Notification(ctx, userID, userID, constant.ConversationDeleteNotification, tips)
}
```

internal/rpc/conversation/sync.go

```
package conversation

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/internal/rpc/incrversion"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/util/hashutil"
	"github.com/openimsdk/protocol/conversation"
)

func (c *conversationServer) GetFullOwnerConversationIDs(ctx context.Context, req *conversation.GetFullOwnerConversa
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	vl, err := c.conversationDatabase.FindMaxConversationUserVersionCache(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	conversationIDs, err := c.conversationDatabase.GetConversationIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	idHash := hashutil.IdHash(conversationIDs)
	if req.IdHash == idHash {
		conversationIDs = nil
	}
	return &conversation.GetFullOwnerConversationIDsResp{
		Version:         uint64(vl.Version),
		VersionID:       vl.ID.Hex(),
		Equal:           req.IdHash == idHash,
		ConversationIDs: conversationIDs,
	}, nil
}

func (c *conversationServer) GetIncrementalConversation(ctx context.Context, req *conversation.GetIncrementalConvers
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	opt := incrversion.Option[*conversation.Conversation, conversation.GetIncrementalConversationResp]{
		Ctx:             ctx,
		VersionKey:      req.UserID,
		VersionID:       req.VersionID,
		VersionNumber:   req.Version,
		Version:         c.conversationDatabase.FindConversationUserVersion,
		CacheMaxVersion: c.conversationDatabase.FindMaxConversationUserVersionCache,
		Find: func(ctx context.Context, conversationIDs []string) ([]*conversation.Conversation, error) {
			return c.getConversations(ctx, req.UserID, conversationIDs)
		},
		Resp: func(version *model.VersionLog, delIDs []string, insertList, updateList []*conversation.Conversation, full
			return &conversation.GetIncrementalConversationResp{
				VersionID: version.ID.Hex(),
				Version:   uint64(version.Version),
				Full:      full,
				Delete:    delIDs,
				Insert:    insertList,
				Update:    updateList,
			}
		},
	}
	return opt.Build()
}
```

internal/rpc/group

internal/rpc/group/cache.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	pbgroup "github.com/openimsdk/protocol/group"
)

// GetGroupInfoCache get group info from cache.
func (g *groupServer) GetGroupInfoCache(ctx context.Context, req *pbgroup.GetGroupInfoCacheReq) (*pbgroup.GetGroupIn
	group, err := g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupInfoCacheResp{
		GroupInfo: convert.Db2PbGroupInfo(group, "", 0),
	}, nil
}

func (g *groupServer) GetGroupMemberCache(ctx context.Context, req *pbgroup.GetGroupMemberCacheReq) (*pbgroup.GetGro
	if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
		return nil, err
	}
	members, err := g.db.TakeGroupMember(ctx, req.GroupID, req.GroupMemberID)
	if err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupMemberCacheResp{
		Member: convert.Db2PbGroupMember(members),
	}, nil
}
```

internal/rpc/group/callback.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/apistruct"
    "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/group"
    "github.com/openimsdk/protocol/wrapperspb"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/utils/datautil"
)

// CallbackBeforeCreateGroup callback before create group.
func (g *groupServer) webhookBeforeCreateGroup(ctx context.Context, before *config.BeforeConfig, req *group.CreateGr
    return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
        cbReq := &callbackstruct.CallbackBeforeCreateGroupReq{
            CallbackCommand: callbackstruct.CallbackBeforeCreateGroupCommand,
            OperationID:     mcontext.GetOperationID(ctx),
            GroupInfo:       req.GroupInfo,
        }
        cbReq.InitMemberList = append(cbReq.InitMemberList, &apistruct.GroupAddMemberInfo{
            UserID:    req.OwnerUserID,
            RoleLevel: constant.GroupOwner,
        })
        for _, userID := range req.AdminUserIDs {
            cbReq.InitMemberList = append(cbReq.InitMemberList, &apistruct.GroupAddMemberInfo{
                UserID:    userID,
                RoleLevel: constant.GroupAdmin,
            })
        }
        for _, userID := range req.MemberUserIDs {
            cbReq.InitMemberList = append(cbReq.InitMemberList, &apistruct.GroupAddMemberInfo{
                UserID:    userID,
                RoleLevel: constant.GroupOrdinaryUsers,
            })
        }
        resp := &callbackstruct.CallbackBeforeCreateGroupResp{}

        if err := g.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
            return err
        }

        datautil.NotNilReplace(&req.GroupInfo.GroupID, resp.GroupID)
```

```
    datautil.NotNilReplace(&req.GroupInfo.GroupName, resp.GroupName)
    datautil.NotNilReplace(&req.GroupInfo.Notification, resp.Notification)
    datautil.NotNilReplace(&req.GroupInfo.Introduction, resp.Introduction)
    datautil.NotNilReplace(&req.GroupInfo.FaceURL, resp.FaceURL)
    datautil.NotNilReplace(&req.GroupInfo.OwnerUserID, resp.OwnerUserID)
    datautil.NotNilReplace(&req.GroupInfo.Ex, resp.Ex)
    datautil.NotNilReplace(&req.GroupInfo.Status, resp.Status)
    datautil.NotNilReplace(&req.GroupInfo.CreatorUserID, resp.CreatorUserID)
    datautil.NotNilReplace(&req.GroupInfo.GroupType, resp.GroupType)
    datautil.NotNilReplace(&req.GroupInfo.NeedVerification, resp.NeedVerification)
    datautil.NotNilReplace(&req.GroupInfo.LookMemberInfo, resp.LookMemberInfo)
    return nil
  })
}

func (g *groupServer) webhookAfterCreateGroup(ctx context.Context, after *config.AfterConfig, req *group.CreateGroup
  cbReq := &callbackstruct.CallbackAfterCreateGroupReq{
    CallbackCommand: callbackstruct.CallbackAfterCreateGroupCommand,
    GroupInfo:       req.GroupInfo,
  }
  cbReq.InitMemberList = append(cbReq.InitMemberList, &apistruct.GroupAddMemberInfo{
    UserID:    req.OwnerUserID,
    RoleLevel: constant.GroupOwner,
  })
  for _, userID := range req.AdminUserIDs {
    cbReq.InitMemberList = append(cbReq.InitMemberList, &apistruct.GroupAddMemberInfo{
      UserID:    userID,
      RoleLevel: constant.GroupAdmin,
    })
  }
  for _, userID := range req.MemberUserIDs {
    cbReq.InitMemberList = append(cbReq.InitMemberList, &apistruct.GroupAddMemberInfo{
      UserID:    userID,
      RoleLevel: constant.GroupOrdinaryUsers,
    })
  }
  g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterCreateGroupResp{}, a
}

func (g *groupServer) webhookBeforeMembersJoinGroup(ctx context.Context, before *config.BeforeConfig, groupMembers [
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    groupMembersMap := datautil.SliceToMap(groupMembers, func(e *model.GroupMember) string {
      return e.UserID
    })
    var groupMembersCallback []*callbackstruct.CallbackGroupMember

    for _, member := range groupMembers {
      groupMembersCallback = append(groupMembersCallback, &callbackstruct.CallbackGroupMember{
        UserID: member.UserID,
        Ex:     member.Ex,
      })
    }

    cbReq := &callbackstruct.CallbackBeforeMembersJoinGroupReq{
      CallbackCommand: callbackstruct.CallbackBeforeMembersJoinGroupCommand,
      GroupID:         groupID,
      MembersList:     groupMembersCallback,
      GroupEx:         groupEx,
    }
    resp := &callbackstruct.CallbackBeforeMembersJoinGroupResp{}

    if err := g.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
      return err
    }

    for _, memberCallbackResp := range resp.MemberCallbackList {
```

```
        if _, ok := groupMembersMap[(*memberCallbackResp.UserID)]; ok {
            if memberCallbackResp.MuteEndTime != nil {
                groupMembersMap[(*memberCallbackResp.UserID)].MuteEndTime = time.UnixMilli(*memberCallbackResp.MuteEndTime)
            }

            datautil.NotNilReplace(&groupMembersMap[(*memberCallbackResp.UserID)].FaceURL, memberCallbackResp.FaceURL)
            datautil.NotNilReplace(&groupMembersMap[(*memberCallbackResp.UserID)].Ex, memberCallbackResp.Ex)
            datautil.NotNilReplace(&groupMembersMap[(*memberCallbackResp.UserID)].Nickname, memberCallbackResp.Nickname)
            datautil.NotNilReplace(&groupMembersMap[(*memberCallbackResp.UserID)].RoleLevel, memberCallbackResp.RoleLevel)
        }
    }

    return nil
  })
}

func (g *groupServer) webhookBeforeSetGroupMemberInfo(ctx context.Context, before *config.BeforeConfig, req *group.S
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    cbReq := callbackstruct.CallbackBeforeSetGroupMemberInfoReq{
        CallbackCommand: callbackstruct.CallbackBeforeSetGroupMemberInfoCommand,
        GroupID:         req.GroupID,
        UserID:          req.UserID,
    }
    if req.Nickname != nil {
        cbReq.Nickname = &req.Nickname.Value
    }
    if req.FaceURL != nil {
        cbReq.FaceURL = &req.FaceURL.Value
    }
    if req.RoleLevel != nil {
        cbReq.RoleLevel = &req.RoleLevel.Value
    }
    if req.Ex != nil {
        cbReq.Ex = &req.Ex.Value
    }
    resp := &callbackstruct.CallbackBeforeSetGroupMemberInfoResp{}
    if err := g.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
        return err
    }
    if resp.FaceURL != nil {
        req.FaceURL = wrapperspb.String(*resp.FaceURL)
    }
    if resp.Nickname != nil {
        req.Nickname = wrapperspb.String(*resp.Nickname)
    }
    if resp.RoleLevel != nil {
        req.RoleLevel = wrapperspb.Int32(*resp.RoleLevel)
    }
    if resp.Ex != nil {
        req.Ex = wrapperspb.String(*resp.Ex)
    }
    return nil
  })
}

func (g *groupServer) webhookAfterSetGroupMemberInfo(ctx context.Context, after *config.AfterConfig, req *group.SetG
  cbReq := callbackstruct.CallbackAfterSetGroupMemberInfoReq{
    CallbackCommand: callbackstruct.CallbackAfterSetGroupMemberInfoCommand,
    GroupID:         req.GroupID,
    UserID:          req.UserID,
  }
  if req.Nickname != nil {
    cbReq.Nickname = &req.Nickname.Value
  }
  if req.FaceURL != nil {
    cbReq.FaceURL = &req.FaceURL.Value
```

```
	}
	if req.RoleLevel != nil {
		cbReq.RoleLevel = &req.RoleLevel.Value
	}
	if req.Ex != nil {
		cbReq.Ex = &req.Ex.Value
	}
	g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterSetGroupMemberInfoRe
}

func (g *groupServer) webhookAfterQuitGroup(ctx context.Context, after *config.AfterConfig, req *group.QuitGroupReq)
	cbReq := &callbackstruct.CallbackQuitGroupReq{
		CallbackCommand: callbackstruct.CallbackAfterQuitGroupCommand,
		GroupID:         req.GroupID,
		UserID:          req.UserID,
	}
	g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackQuitGroupResp{}, after)
}

func (g *groupServer) webhookAfterKickGroupMember(ctx context.Context, after *config.AfterConfig, req *group.KickGro
	cbReq := &callbackstruct.CallbackKillGroupMemberReq{
		CallbackCommand: callbackstruct.CallbackAfterKickGroupCommand,
		GroupID:         req.GroupID,
		KickedUserIDs:   req.KickedUserIDs,
		Reason:          req.Reason,
	}
	g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackKillGroupMemberResp{}, af
}

func (g *groupServer) webhookAfterDismissGroup(ctx context.Context, after *config.AfterConfig, req *callbackstruct.C
	req.CallbackCommand = callbackstruct.CallbackAfterDisMissGroupCommand
	g.webhookClient.AsyncPost(ctx, req.GetCallbackCommand(), req, &callbackstruct.CallbackDisMissGroupResp{}, after)
}

func (g *groupServer) webhookBeforeApplyJoinGroup(ctx context.Context, before *config.BeforeConfig, req *callbackstr
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		req.CallbackCommand = callbackstruct.CallbackBeforeJoinGroupCommand
		resp := &callbackstruct.CallbackJoinGroupResp{}
		if err := g.webhookClient.SyncPost(ctx, req.GetCallbackCommand(), req, resp, before); err != nil {
			return err
		}
		return nil
	})
}

func (g *groupServer) webhookAfterTransferGroupOwner(ctx context.Context, after *config.AfterConfig, req *group.Tran
	cbReq := &callbackstruct.CallbackTransferGroupOwnerReq{
		CallbackCommand: callbackstruct.CallbackAfterTransferGroupOwnerCommand,
		GroupID:         req.GroupID,
		OldOwnerUserID:  req.OldOwnerUserID,
		NewOwnerUserID:  req.NewOwnerUserID,
	}
	g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackTransferGroupOwnerResp{},
}

func (g *groupServer) webhookBeforeInviteUserToGroup(ctx context.Context, before *config.BeforeConfig, req *group.In
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &callbackstruct.CallbackBeforeInviteUserToGroupReq{
			CallbackCommand: callbackstruct.CallbackBeforeInviteJoinGroupCommand,
			OperationID:     mcontext.GetOperationID(ctx),
			GroupID:         req.GroupID,
			Reason:          req.Reason,
			InvitedUserIDs:  req.InvitedUserIDs,
		}

		resp := &callbackstruct.CallbackBeforeInviteUserToGroupResp{}
```

```
    if err := g.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
        return err
    }

    // Handle the scenario where certain members are refused
    // You might want to update the req.Members list or handle it as per your business logic

    // if len(resp.RefusedMembersAccount) > 0 {
    // implement members are refused
    // }

    return nil
  })
}

func (g *groupServer) webhookAfterJoinGroup(ctx context.Context, after *config.AfterConfig, req *group.JoinGroupReq)
  cbReq := &callbackstruct.CallbackAfterJoinGroupReq{
    CallbackCommand: callbackstruct.CallbackAfterJoinGroupCommand,
    OperationID:     mcontext.GetOperationID(ctx),
    GroupID:         req.GroupID,
    ReqMessage:      req.ReqMessage,
    JoinSource:      req.JoinSource,
    InviterUserID:   req.InviterUserID,
  }
  g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterJoinGroupResp{}, aft
}

func (g *groupServer) webhookBeforeSetGroupInfo(ctx context.Context, before *config.BeforeConfig, req *group.SetGrou
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    cbReq := &callbackstruct.CallbackBeforeSetGroupInfoReq{
      CallbackCommand: callbackstruct.CallbackBeforeSetGroupInfoCommand,
      GroupID:         req.GroupInfoForSet.GroupID,
      Notification:    req.GroupInfoForSet.Notification,
      Introduction:    req.GroupInfoForSet.Introduction,
      FaceURL:         req.GroupInfoForSet.FaceURL,
      GroupName:       req.GroupInfoForSet.GroupName,
    }
    if req.GroupInfoForSet.Ex != nil {
      cbReq.Ex = req.GroupInfoForSet.Ex.Value
    }
    log.ZDebug(ctx, "debug CallbackBeforeSetGroupInfo", "ex", cbReq.Ex)
    if req.GroupInfoForSet.NeedVerification != nil {
      cbReq.NeedVerification = req.GroupInfoForSet.NeedVerification.Value
    }
    if req.GroupInfoForSet.LookMemberInfo != nil {
      cbReq.LookMemberInfo = req.GroupInfoForSet.LookMemberInfo.Value
    }
    if req.GroupInfoForSet.ApplyMemberFriend != nil {
      cbReq.ApplyMemberFriend = req.GroupInfoForSet.ApplyMemberFriend.Value
    }
    resp := &callbackstruct.CallbackBeforeSetGroupInfoResp{}

    if err := g.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
      return err
    }

    if resp.Ex != nil {
      req.GroupInfoForSet.Ex = wrapperspb.String(*resp.Ex)
    }
    if resp.NeedVerification != nil {
      req.GroupInfoForSet.NeedVerification = wrapperspb.Int32(*resp.NeedVerification)
    }
    if resp.LookMemberInfo != nil {
      req.GroupInfoForSet.LookMemberInfo = wrapperspb.Int32(*resp.LookMemberInfo)
    }
    if resp.ApplyMemberFriend != nil {
```

```
        req.GroupInfoForSet.ApplyMemberFriend = wrapperspb.Int32(*resp.ApplyMemberFriend)
    }
    datautil.NotNilReplace(&req.GroupInfoForSet.GroupID, &resp.GroupID)
    datautil.NotNilReplace(&req.GroupInfoForSet.GroupName, &resp.GroupName)
    datautil.NotNilReplace(&req.GroupInfoForSet.FaceURL, &resp.FaceURL)
    datautil.NotNilReplace(&req.GroupInfoForSet.Introduction, &resp.Introduction)
    return nil
  })
}

func (g *groupServer) webhookAfterSetGroupInfo(ctx context.Context, after *config.AfterConfig, req *group.SetGroupIn
  cbReq := &callbackstruct.CallbackAfterSetGroupInfoReq{
    CallbackCommand: callbackstruct.CallbackAfterSetGroupInfoCommand,
    GroupID:         req.GroupInfoForSet.GroupID,
    Notification:    req.GroupInfoForSet.Notification,
    Introduction:    req.GroupInfoForSet.Introduction,
    FaceURL:         req.GroupInfoForSet.FaceURL,
    GroupName:       req.GroupInfoForSet.GroupName,
  }
  if req.GroupInfoForSet.Ex != nil {
    cbReq.Ex = &req.GroupInfoForSet.Ex.Value
  }
  if req.GroupInfoForSet.NeedVerification != nil {
    cbReq.NeedVerification = &req.GroupInfoForSet.NeedVerification.Value
  }
  if req.GroupInfoForSet.LookMemberInfo != nil {
    cbReq.LookMemberInfo = &req.GroupInfoForSet.LookMemberInfo.Value
  }
  if req.GroupInfoForSet.ApplyMemberFriend != nil {
    cbReq.ApplyMemberFriend = &req.GroupInfoForSet.ApplyMemberFriend.Value
  }
  g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterSetGroupInfoResp{},
}

func (g *groupServer) webhookBeforeSetGroupInfoEx(ctx context.Context, before *config.BeforeConfig, req *group.SetGr
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    cbReq := &callbackstruct.CallbackBeforeSetGroupInfoExReq{
      CallbackCommand: callbackstruct.CallbackBeforeSetGroupInfoExCommand,
      GroupID:         req.GroupID,
      GroupName:       req.GroupName,
      Notification:    req.Notification,
      Introduction:    req.Introduction,
      FaceURL:         req.FaceURL,
    }

    if req.Ex != nil {
      cbReq.Ex = req.Ex
    }
    log.ZDebug(ctx, "debug CallbackBeforeSetGroupInfoEx", "ex", cbReq.Ex)

    if req.NeedVerification != nil {
      cbReq.NeedVerification = req.NeedVerification
    }
    if req.LookMemberInfo != nil {
      cbReq.LookMemberInfo = req.LookMemberInfo
    }
    if req.ApplyMemberFriend != nil {
      cbReq.ApplyMemberFriend = req.ApplyMemberFriend
    }

    resp := &callbackstruct.CallbackBeforeSetGroupInfoExResp{}

    if err := g.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
      return err
    }
```

```
    datautil.NotNilReplace(&req.GroupID, &resp.GroupID)
    datautil.NotNilReplace(&req.GroupName, &resp.GroupName)
    datautil.NotNilReplace(&req.FaceURL, &resp.FaceURL)
    datautil.NotNilReplace(&req.Introduction, &resp.Introduction)
    datautil.NotNilReplace(&req.Ex, &resp.Ex)
    datautil.NotNilReplace(&req.NeedVerification, &resp.NeedVerification)
    datautil.NotNilReplace(&req.LookMemberInfo, &resp.LookMemberInfo)
    datautil.NotNilReplace(&req.ApplyMemberFriend, &resp.ApplyMemberFriend)

    return nil
  })
}

func (g *groupServer) webhookAfterSetGroupInfoEx(ctx context.Context, after *config.AfterConfig, req *group.SetGroup
  cbReq := &callbackstruct.CallbackAfterSetGroupInfoExReq{
    CallbackCommand: callbackstruct.CallbackAfterSetGroupInfoExCommand,
    GroupID:         req.GroupID,
    GroupName:       req.GroupName,
    Notification:    req.Notification,
    Introduction:    req.Introduction,
    FaceURL:         req.FaceURL,
  }

  if req.Ex != nil {
    cbReq.Ex = req.Ex
  }
  if req.NeedVerification != nil {
    cbReq.NeedVerification = req.NeedVerification
  }
  if req.LookMemberInfo != nil {
    cbReq.LookMemberInfo = req.LookMemberInfo
  }
  if req.ApplyMemberFriend != nil {
    cbReq.ApplyMemberFriend = req.ApplyMemberFriend
  }

  g.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &callbackstruct.CallbackAfterSetGroupInfoExResp{}
}
```

internal/rpc/group/convert.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/sdkws"
)

func (g *groupServer) groupDB2PB(group *model.Group, ownerUserID string, memberCount uint32) *sdkws.GroupInfo {
	return &sdkws.GroupInfo{
		GroupID:                group.GroupID,
		GroupName:              group.GroupName,
		Notification:           group.Notification,
		Introduction:           group.Introduction,
		FaceURL:                group.FaceURL,
		OwnerUserID:            ownerUserID,
		CreateTime:             group.CreateTime.UnixMilli(),
		MemberCount:            memberCount,
		Ex:                     group.Ex,
		Status:                 group.Status,
		CreatorUserID:          group.CreatorUserID,
		GroupType:              group.GroupType,
		NeedVerification:       group.NeedVerification,
		LookMemberInfo:         group.LookMemberInfo,
		ApplyMemberFriend:      group.ApplyMemberFriend,
		NotificationUpdateTime: group.NotificationUpdateTime.UnixMilli(),
		NotificationUserID:     group.NotificationUserID,
	}
}

func (g *groupServer) groupMemberDB2PB(member *model.GroupMember, appMangerLevel int32) *sdkws.GroupMemberFullInfo {
	return &sdkws.GroupMemberFullInfo{
		GroupID:        member.GroupID,
		UserID:         member.UserID,
		RoleLevel:      member.RoleLevel,
		JoinTime:       member.JoinTime.UnixMilli(),
		Nickname:       member.Nickname,
		FaceURL:        member.FaceURL,
		AppMangerLevel: appMangerLevel,
		JoinSource:     member.JoinSource,
		OperatorUserID: member.OperatorUserID,
		Ex:             member.Ex,
		MuteEndTime:    member.MuteEndTime.UnixMilli(),
		InviterUserID:  member.InviterUserID,
	}
}

func (g *groupServer) groupMemberDB2PB2(member *model.GroupMember) *sdkws.GroupMemberFullInfo {
	return g.groupMemberDB2PB(member, 0)
}
```

internal/rpc/group/db_map.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
	"context"
	"strings"
	"time"

	pbgroup "github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/mcontext"
)

func UpdateGroupInfoMap(ctx context.Context, group *sdkws.GroupInfoForSet) map[string]any {
	m := make(map[string]any)
	if group.GroupName != "" {
		m["group_name"] = group.GroupName
	}
	if group.Notification != "" {
		m["notification"] = group.Notification
		m["notification_update_time"] = time.Now()
		m["notification_user_id"] = mcontext.GetOpUserID(ctx)
	}
	if group.Introduction != "" {
		m["introduction"] = group.Introduction
	}
	if group.FaceURL != "" {
		m["face_url"] = group.FaceURL
	}
	if group.NeedVerification != nil {
		m["need_verification"] = group.NeedVerification.Value
	}
	if group.LookMemberInfo != nil {
		m["look_member_info"] = group.LookMemberInfo.Value
	}
	if group.ApplyMemberFriend != nil {
		m["apply_member_friend"] = group.ApplyMemberFriend.Value
	}
	if group.Ex != nil {
		m["ex"] = group.Ex.Value
	}
	return m
}

func UpdateGroupInfoExMap(ctx context.Context, group *pbgroup.SetGroupInfoExReq) (m map[string]any, normalFlag, grou
	m = make(map[string]any)

	if group.GroupName != nil {
		if strings.TrimSpace(group.GroupName.Value) != "" {
			m["group_name"] = group.GroupName.Value
```

```
        groupNameFlag = true
    } else {
        return nil, normalFlag, notificationFlag, groupNameFlag, errs.ErrArgs.WrapMsg("group name is empty")
    }
}

if group.Notification != nil {
    notificationFlag = true
    group.Notification.Value = strings.TrimSpace(group.Notification.Value) // if Notification only contains spaces, s

    m["notification"] = group.Notification.Value
    m["notification_user_id"] = mcontext.GetOpUserID(ctx)
    m["notification_update_time"] = time.Now()
}
if group.Introduction != nil {
    m["introduction"] = group.Introduction.Value
    normalFlag = true
}
if group.FaceURL != nil {
    m["face_url"] = group.FaceURL.Value
    normalFlag = true
}
if group.NeedVerification != nil {
    m["need_verification"] = group.NeedVerification.Value
    normalFlag = true
}
if group.LookMemberInfo != nil {
    m["look_member_info"] = group.LookMemberInfo.Value
    normalFlag = true
}
if group.ApplyMemberFriend != nil {
    m["apply_member_friend"] = group.ApplyMemberFriend.Value
    normalFlag = true
}
if group.Ex != nil {
    m["ex"] = group.Ex.Value
    normalFlag = true
}

return m, normalFlag, groupNameFlag, notificationFlag, nil
}

func UpdateGroupStatusMap(status int) map[string]any {
return map[string]any{
    "status": status,
}
}

func UpdateGroupMemberMutedTimeMap(t time.Time) map[string]any {
return map[string]any{
    "mute_end_time": t,
}
}

func UpdateGroupMemberMap(req *pbgroup.SetGroupMemberInfo) map[string]any {
m := make(map[string]any)
if req.Nickname != nil {
    m["nickname"] = req.Nickname.Value
}
if req.FaceURL != nil {
    m["face_url"] = req.FaceURL.Value
}
if req.RoleLevel != nil {
    m["role_level"] = req.RoleLevel.Value
}
if req.Ex != nil {
```

```
        m["ex"] = req.Ex.Value
    }
    return m
}
```

internal/rpc/group/fill.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
    "context"

    relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
)

func (g *groupServer) PopulateGroupMember(ctx context.Context, members ...*relationtb.GroupMember) error {
    return g.notification.PopulateGroupMember(ctx, members...)
}
```

internal/rpc/group/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
    "context"
    "fmt"
    "math/big"
    "math/rand"
    "strconv"
    "strings"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "google.golang.org/grpc"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/convert"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/common"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
    "github.com/openimsdk/open-im-server/v3/pkg/localcache"
    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/open-im-server/v3/pkg/notification/grouphash"
    "github.com/openimsdk/protocol/constant"
    pbconv "github.com/openimsdk/protocol/conversation"
    pbgroup "github.com/openimsdk/protocol/group"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/protocol/wrapperspb"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/mw/specialerror"
    "github.com/openimsdk/tools/utils/datautil"
    "github.com/openimsdk/tools/utils/encrypt"
)

type groupServer struct {
    pbgroup.UnimplementedGroupServer
    db                 controller.GroupDatabase
    notification       *NotificationSender
    config             *Config
    webhookClient      *webhook.Client
    userClient         *rpcli.UserClient
    msgClient          *rpcli.MsgClient
```

```
	conversationClient *rpcli.ConversationClient
	adminUserIDs       []string
}

type Config struct {
	RpcConfig          config.Group
	RedisConfig        config.Redis
	MongodbConfig      config.Mongo
	NotificationConfig config.Notification
	Share              config.Share
	WebhooksConfig     config.Webhooks
	LocalCacheConfig   config.LocalCache
	Discovery          config.Discovery
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
	dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
	mgocli, err := dbb.Mongo(ctx)
	if err != nil {
		return err
	}
	rdb, err := dbb.Redis(ctx)
	if err != nil {
		return err
	}
	groupDB, err := mgo.NewGroupMongo(mgocli.GetDB())
	if err != nil {
		return err
	}
	groupMemberDB, err := mgo.NewGroupMember(mgocli.GetDB())
	if err != nil {
		return err
	}
	groupRequestDB, err := mgo.NewGroupRequestMgo(mgocli.GetDB())
	if err != nil {
		return err
	}
	userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
	if err != nil {
		return err
	}
	msgConn, err := client.GetConn(ctx, config.Discovery.RpcService.Msg)
	if err != nil {
		return err
	}
	conversationConn, err := client.GetConn(ctx, config.Discovery.RpcService.Conversation)
	if err != nil {
		return err
	}
	gs := groupServer{
		config:             config,
		webhookClient:      webhook.NewWebhookClient(config.WebhooksConfig.URL),
		userClient:         rpcli.NewUserClient(userConn),
		msgClient:          rpcli.NewMsgClient(msgConn),
		conversationClient: rpcli.NewConversationClient(conversationConn),
		adminUserIDs:       config.Share.IMAdminUser.UserIDs,
	}
	gs.db = controller.NewGroupDatabase(rdb, &config.LocalCacheConfig, groupDB, groupMemberDB, groupRequestDB, mgocli.C
	gs.notification = NewNotificationSender(gs.db, config, gs.userClient, gs.msgClient, gs.conversationClient)
	localcache.InitLocalCache(&config.LocalCacheConfig)
	pbgroup.RegisterGroupServer(server, &gs)
	return nil
}

func (g *groupServer) NotificationUserInfoUpdate(ctx context.Context, req *pbgroup.NotificationUserInfoUpdateReq) (*
	members, err := g.db.FindGroupMemberUser(ctx, nil, req.UserID)
```

```
	if err != nil {
		return nil, err
	}
	groupIDs := make([]string, 0, len(members))
	for _, member := range members {
		if member.Nickname != "" && member.FaceURL != "" {
			continue
		}
		groupIDs = append(groupIDs, member.GroupID)
	}
	for _, groupID := range groupIDs {
		if err := g.db.MemberGroupIncrVersion(ctx, groupID, []string{req.UserID}, model.VersionStateUpdate); err != nil {
			return nil, err
		}
	}
	for _, groupID := range groupIDs {
		g.notification.GroupMemberInfoSetNotification(ctx, groupID, req.UserID)
	}
	if err = g.db.DeleteGroupMemberHash(ctx, groupIDs); err != nil {
		return nil, err
	}
	return &pbgroup.NotificationUserInfoUpdateResp{}, nil
}

func (g *groupServer) CheckGroupAdmin(ctx context.Context, groupID string) error {
	if !authverify.IsAdmin(ctx) {
		members, err := g.db.FindGroupMembers(ctx, groupID, []string{mcontext.GetOpUserID(ctx)})
		if err != nil {
			return err
		}
		if len(members) == 0 {
			return errs.ErrNoPermission.WrapMsg("op user not in group")
		}
		groupMember := members[0]
		if !(groupMember.RoleLevel == constant.GroupOwner || groupMember.RoleLevel == constant.GroupAdmin) {
			return errs.ErrNoPermission.WrapMsg("no group owner or admin")
		}
	}
	return nil
}

func (g *groupServer) IsNotFound(err error) bool {
	return errs.ErrRecordNotFound.Is(specialerror.ErrCode(errs.Unwrap(err)))
}

func (g *groupServer) GenGroupID(ctx context.Context, groupID *string) error {
	if *groupID != "" {
		_, err := g.db.TakeGroup(ctx, *groupID)
		if err == nil {
			return servererrs.ErrGroupIDExisted.WrapMsg("group id existed " + *groupID)
		} else if g.IsNotFound(err) {
			return nil
		} else {
			return err
		}
	}
	for i := 0; i < 10; i++ {
		id := encrypt.Md5(strings.Join([]string{mcontext.GetOperationID(ctx), strconv.FormatInt(time.Now().UnixNano(), 10
		bi := big.NewInt(0)
		bi.SetString(id[0:8], 16)
		id = bi.String()
		_, err := g.db.TakeGroup(ctx, id)
		if err == nil {
			continue
		} else if g.IsNotFound(err) {
			*groupID = id
```

```
        return nil
    } else {
        return err
    }
}
return servererrs.ErrData.WrapMsg("group id gen error")
}

func (g *groupServer) CreateGroup(ctx context.Context, req *pbgroup.CreateGroupReq) (*pbgroup.CreateGroupResp, error
if req.GroupInfo.GroupType != constant.WorkingGroup {
    return nil, errs.ErrArgs.WrapMsg(fmt.Sprintf("group type only supports %d", constant.WorkingGroup))
}
if req.OwnerUserID == "" {
    return nil, errs.ErrArgs.WrapMsg("no group owner")
}
if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
    return nil, err
}
userIDs := append(append(req.MemberUserIDs, req.AdminUserIDs...), req.OwnerUserID)
opUserID := mcontext.GetOpUserID(ctx)
if !datautil.Contain(opUserID, userIDs...) {
    userIDs = append(userIDs, opUserID)
}

if datautil.Duplicate(userIDs) {
    return nil, errs.ErrArgs.WrapMsg("group member repeated")
}

userMap, err := g.userClient.GetUsersInfoMap(ctx, userIDs)
if err != nil {
    return nil, err
}

if len(userMap) != len(userIDs) {
    return nil, servererrs.ErrUserIDNotFound.WrapMsg("user not found")
}

if err := g.webhookBeforeCreateGroup(ctx, &g.config.WebhooksConfig.BeforeCreateGroup, req); err != nil && err != se
    return nil, err
}

var groupMembers []*model.GroupMember
group := convert.Pb2DBGroupInfo(req.GroupInfo)
if err := g.GenGroupID(ctx, &group.GroupID); err != nil {
    return nil, err
}

joinGroupFunc := func(userID string, roleLevel int32) {
    groupMember := &model.GroupMember{
        GroupID:        group.GroupID,
        UserID:         userID,
        RoleLevel:      roleLevel,
        OperatorUserID: opUserID,
        JoinSource:     constant.JoinByInvitation,
        InviterUserID:  opUserID,
        JoinTime:       time.Now(),
        MuteEndTime:    time.UnixMilli(0),
    }

    groupMembers = append(groupMembers, groupMember)
}

joinGroupFunc(req.OwnerUserID, constant.GroupOwner)

for _, userID := range req.AdminUserIDs {
    joinGroupFunc(userID, constant.GroupAdmin)
```

```
	}

	for _, userID := range req.MemberUserIDs {
		joinGroupFunc(userID, constant.GroupOrdinaryUsers)
	}

	if err := g.webhookBeforeMembersJoinGroup(ctx, &g.config.WebhooksConfig.BeforeMemberJoinGroup, groupMembers, group.
		return nil, err
	}

	if err := g.db.CreateGroup(ctx, []*model.Group{group}, groupMembers); err != nil {
		return nil, err
	}
	resp := &pbgroup.CreateGroupResp{GroupInfo: &sdkws.GroupInfo{}}

	resp.GroupInfo = convert.Db2PbGroupInfo(group, req.OwnerUserID, uint32(len(userIDs)))
	resp.GroupInfo.MemberCount = uint32(len(userIDs))
	tips := &sdkws.GroupCreatedTips{
		Group:          resp.GroupInfo,
		OperationTime:  group.CreateTime.UnixMilli(),
		GroupOwnerUser: g.groupMemberDB2PB(groupMembers[0], userMap[groupMembers[0].UserID].AppMangerLevel),
	}
	for _, member := range groupMembers {
		member.Nickname = userMap[member.UserID].Nickname
		tips.MemberList = append(tips.MemberList, g.groupMemberDB2PB(member, userMap[member.UserID].AppMangerLevel))
		if member.UserID == opUserID {
			tips.OpUser = g.groupMemberDB2PB(member, userMap[member.UserID].AppMangerLevel)
			break
		}
	}
	g.notification.GroupCreatedNotification(ctx, tips, req.SendMessage)

	if req.GroupInfo.Notification != "" {
		notificationFlag := true
		g.notification.GroupInfoSetAnnouncementNotification(ctx, &sdkws.GroupInfoSetAnnouncementTips{
			Group:  tips.Group,
			OpUser: tips.OpUser,
		}, &notificationFlag)
	}

	reqCallBackAfter := &pbgroup.CreateGroupReq{
		MemberUserIDs: userIDs,
		GroupInfo:     resp.GroupInfo,
		OwnerUserID:   req.OwnerUserID,
		AdminUserIDs:  req.AdminUserIDs,
	}

	g.webhookAfterCreateGroup(ctx, &g.config.WebhooksConfig.AfterCreateGroup, reqCallBackAfter)

	return resp, nil
}

func (g *groupServer) GetJoinedGroupList(ctx context.Context, req *pbgroup.GetJoinedGroupListReq) (*pbgroup.GetJoine
	if err := authverify.CheckAccess(ctx, req.FromUserID); err != nil {
		return nil, err
	}
	total, members, err := g.db.PageGetJoinGroup(ctx, req.FromUserID, req.Pagination)
	if err != nil {
		return nil, err
	}
	var resp pbgroup.GetJoinedGroupListResp
	resp.Total = uint32(total)
	if len(members) == 0 {
		return &resp, nil
	}
	groupIDs := datautil.Slice(members, func(e *model.GroupMember) string {
```

```
		return e.GroupID
	})
	groups, err := g.db.FindGroup(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	groupMemberNum, err := g.db.MapGroupMemberNum(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	owners, err := g.db.FindGroupsOwner(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}
	ownerMap := datautil.SliceToMap(owners, func(e *model.GroupMember) string {
		return e.GroupID
	})
	resp.Groups = datautil.Slice(datautil.Order(groupIDs, groups, func(group *model.Group) string {
		return group.GroupID
	}), func(group *model.Group) *sdkws.GroupInfo {
		var userID string
		if user := ownerMap[group.GroupID]; user != nil {
			userID = user.UserID
		}
		return convert.Db2PbGroupInfo(group, userID, groupMemberNum[group.GroupID])
	})
	return &resp, nil
}

func (g *groupServer) InviteUserToGroup(ctx context.Context, req *pbgroup.InviteUserToGroupReq) (*pbgroup.InviteUser
	if len(req.InvitedUserIDs) == 0 {
		return nil, errs.ErrArgs.WrapMsg("user empty")
	}
	if datautil.Duplicate(req.InvitedUserIDs) {
		return nil, errs.ErrArgs.WrapMsg("userID duplicate")
	}
	group, err := g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}

	if group.Status == constant.GroupStatusDismissed {
		return nil, servererrs.ErrDismissedAlready.WrapMsg("group dismissed checking group status found it dismissed")
	}

	if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
		return nil, err
	}

	userMap, err := g.userClient.GetUsersInfoMap(ctx, req.InvitedUserIDs)
	if err != nil {
		return nil, err
	}

	if len(userMap) != len(req.InvitedUserIDs) {
		return nil, errs.ErrRecordNotFound.WrapMsg("user not found")
	}

	var groupMember *model.GroupMember
	opUserID := mcontext.GetOpUserID(ctx)

	if !authverify.IsAdmin(ctx) {
		var err error
```

```
    groupMember, err = g.db.TakeGroupMember(ctx, req.GroupID, opUserID)
    if err != nil {
        return nil, err
    }
    if err := g.PopulateGroupMember(ctx, groupMember); err != nil {
        return nil, err
    }
}

if err := g.webhookBeforeInviteUserToGroup(ctx, &g.config.WebhooksConfig.BeforeInviteUserToGroup, req); err != nil
    return nil, err
}

if group.NeedVerification == constant.AllNeedVerification {
    if !authverify.IsAdmin(ctx) {
        if !(groupMember.RoleLevel == constant.GroupOwner || groupMember.RoleLevel == constant.GroupAdmin) {
            var requests []*model.GroupRequest
            for _, userID := range req.InvitedUserIDs {
                requests = append(requests, &model.GroupRequest{
                    UserID:        userID,
                    GroupID:       req.GroupID,
                    JoinSource:    constant.JoinByInvitation,
                    InviterUserID: opUserID,
                    ReqTime:       time.Now(),
                    HandledTime:   time.Unix(0, 0),
                })
            }
            if err := g.db.CreateGroupRequest(ctx, requests); err != nil {
                return nil, err
            }
            for _, request := range requests {
                g.notification.JoinGroupApplicationNotification(ctx, &pbgroup.JoinGroupReq{
                    GroupID:       request.GroupID,
                    ReqMessage:    request.ReqMsg,
                    JoinSource:    request.JoinSource,
                    InviterUserID: request.InviterUserID,
                }, request)
            }
            return &pbgroup.InviteUserToGroupResp{}, nil
        }
    }
}

var groupMembers []*model.GroupMember
for _, userID := range req.InvitedUserIDs {
    member := &model.GroupMember{
        GroupID:        req.GroupID,
        UserID:         userID,
        RoleLevel:      constant.GroupOrdinaryUsers,
        OperatorUserID: opUserID,
        InviterUserID:  opUserID,
        JoinSource:     constant.JoinByInvitation,
        JoinTime:       time.Now(),
        MuteEndTime:    time.UnixMilli(0),
    }

    groupMembers = append(groupMembers, member)
}

if err := g.webhookBeforeMembersJoinGroup(ctx, &g.config.WebhooksConfig.BeforeMemberJoinGroup, groupMembers, group.
    return nil, err
}

const singleQuantity = 50
for start := 0; start < len(groupMembers); start += singleQuantity {
    end := min(start+singleQuantity, len(groupMembers))
```

```
    currentMembers := groupMembers[start:end]

    if err := g.db.CreateGroup(ctx, nil, currentMembers); err != nil {
        return nil, err
    }

    userIDs := datautil.Slice(currentMembers, func(e *model.GroupMember) string {
        return e.UserID
    })

    if len(userIDs) != 0 {
        g.notification.GroupApplicationAgreeMemberEnterNotification(ctx, req.GroupID, req.SendMessage, opUserID, userIDs
    }
  }
  if err := g.setMemberJoinSeq(ctx, req.GroupID, req.InvitedUserIDs); err != nil {
    return nil, err
  }
  return &pbgroup.InviteUserToGroupResp{}, nil
}

func (g *groupServer) GetGroupAllMember(ctx context.Context, req *pbgroup.GetGroupAllMemberReq) (*pbgroup.GetGroupAl
  members, err := g.db.FindGroupMemberAll(ctx, req.GroupID)
  if err != nil {
    return nil, err
  }
  if !authverify.IsAdmin(ctx) {
    var inGroup bool
    opUserID := mcontext.GetOpUserID(ctx)
    for _, member := range members {
        if member.UserID == opUserID {
            inGroup = true
            break
        }
    }
    if !inGroup {
        return nil, errs.ErrNoPermission.WrapMsg("opuser not in group")
    }
  }
  if err := g.PopulateGroupMember(ctx, members...); err != nil {
    return nil, err
  }
  var resp pbgroup.GetGroupAllMemberResp
  resp.Members = datautil.Slice(members, func(e *model.GroupMember) *sdkws.GroupMemberFullInfo {
    return convert.Db2PbGroupMember(e)
  })
  return &resp, nil
}

func (g *groupServer) checkAdminOrInGroup(ctx context.Context, groupID string) error {
  if authverify.IsAdmin(ctx) {
    return nil
  }
  opUserID := mcontext.GetOpUserID(ctx)
  members, err := g.db.FindGroupMembers(ctx, groupID, []string{opUserID})
  if err != nil {
    return err
  }
  if len(members) == 0 {
    return errs.ErrNoPermission.WrapMsg("op user not in group")
  }
  return nil
}

func (g *groupServer) GetGroupMemberList(ctx context.Context, req *pbgroup.GetGroupMemberListReq) (*pbgroup.GetGroup
  if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
    return nil, err
```

```
}
var (
    total   int64
    members []*model.GroupMember
    err     error
)
if req.Keyword == "" {
    total, members, err = g.db.PageGetGroupMember(ctx, req.GroupID, req.Pagination)
} else {
    total, members, err = g.db.SearchGroupMember(ctx, req.GroupID, req.Keyword, req.Pagination)
}
if err != nil {
    return nil, err
}
if err := g.PopulateGroupMember(ctx, members...); err != nil {
    return nil, err
}
return &pbgroup.GetGroupMemberListResp{
    Total:   uint32(total),
    Members: datautil.Batch(convert.Db2PbGroupMember, members),
}, nil
}

func (g *groupServer) KickGroupMember(ctx context.Context, req *pbgroup.KickGroupMemberReq) (*pbgroup.KickGroupMembe
group, err := g.db.TakeGroup(ctx, req.GroupID)
if err != nil {
    return nil, err
}
if len(req.KickedUserIDs) == 0 {
    return nil, errs.ErrArgs.WrapMsg("KickedUserIDs empty")
}
if datautil.Duplicate(req.KickedUserIDs) {
    return nil, errs.ErrArgs.WrapMsg("KickedUserIDs duplicate")
}
opUserID := mcontext.GetOpUserID(ctx)
if datautil.Contain(opUserID, req.KickedUserIDs...) {
    return nil, errs.ErrArgs.WrapMsg("opUserID in KickedUserIDs")
}
owner, err := g.db.TakeGroupOwner(ctx, req.GroupID)
if err != nil {
    return nil, err
}
if datautil.Contain(owner.UserID, req.KickedUserIDs...) {
    return nil, errs.ErrArgs.WrapMsg("ownerUID can not Kick")
}

members, err := g.db.FindGroupMembers(ctx, req.GroupID, append(req.KickedUserIDs, opUserID))
if err != nil {
    return nil, err
}
if err := g.PopulateGroupMember(ctx, members...); err != nil {
    return nil, err
}
memberMap := make(map[string]*model.GroupMember)
for i, member := range members {
    memberMap[member.UserID] = members[i]
}
isAppManagerUid := authverify.IsAdmin(ctx)
opMember := memberMap[opUserID]
for _, userID := range req.KickedUserIDs {
    member, ok := memberMap[userID]
    if !ok {
        return nil, servererrs.ErrUserIDNotFound.WrapMsg(userID)
    }
    if !isAppManagerUid {
        if opMember == nil {
```

```
        return nil, errs.ErrNoPermission.WrapMsg("opUserID no in group")
    }
    switch opMember.RoleLevel {
    case constant.GroupOwner:
    case constant.GroupAdmin:
        if member.RoleLevel == constant.GroupOwner || member.RoleLevel == constant.GroupAdmin {
            return nil, errs.ErrNoPermission.WrapMsg("group admins cannot remove the group owner and other admins")
        }
    case constant.GroupOrdinaryUsers:
        return nil, errs.ErrNoPermission.WrapMsg("opUserID no permission")
    default:
        return nil, errs.ErrNoPermission.WrapMsg("opUserID roleLevel unknown")
    }
  }
 }
 ownerUserIDs, err := g.db.GetGroupRoleLevelMemberIDs(ctx, req.GroupID, constant.GroupOwner)
 if err != nil {
  return nil, err
 }
 var ownerUserID string
 if len(ownerUserIDs) > 0 {
  ownerUserID = ownerUserIDs[0]
 }
 if err := g.db.DeleteGroupMember(ctx, group.GroupID, req.KickedUserIDs); err != nil {
  return nil, err
 }
 num, err := g.db.FindGroupMemberNum(ctx, req.GroupID)
 if err != nil {
  return nil, err
 }
 tips := &sdkws.MemberKickedTips{
  Group: &sdkws.GroupInfo{
    GroupID:                group.GroupID,
    GroupName:              group.GroupName,
    Notification:           group.Notification,
    Introduction:           group.Introduction,
    FaceURL:                group.FaceURL,
    OwnerUserID:            ownerUserID,
    CreateTime:             group.CreateTime.UnixMilli(),
    MemberCount:            num,
    Ex:                     group.Ex,
    Status:                 group.Status,
    CreatorUserID:          group.CreatorUserID,
    GroupType:              group.GroupType,
    NeedVerification:       group.NeedVerification,
    LookMemberInfo:         group.LookMemberInfo,
    ApplyMemberFriend:      group.ApplyMemberFriend,
    NotificationUpdateTime: group.NotificationUpdateTime.UnixMilli(),
    NotificationUserID:     group.NotificationUserID,
  },
  KickedUserList: []*sdkws.GroupMemberFullInfo{},
 }
 if opMember, ok := memberMap[opUserID]; ok {
  tips.OpUser = convert.Db2PbGroupMember(opMember)
 }
 for _, userID := range req.KickedUserIDs {
  tips.KickedUserList = append(tips.KickedUserList, convert.Db2PbGroupMember(memberMap[userID]))
 }
 g.notification.MemberKickedNotification(ctx, tips, req.SendMessage)
 if err := g.deleteMemberAndSetConversationSeq(ctx, req.GroupID, req.KickedUserIDs); err != nil {
  return nil, err
 }
 g.webhookAfterKickGroupMember(ctx, &g.config.WebhooksConfig.AfterKickGroupMember, req)

 return &pbgroup.KickGroupMemberResp{}, nil
}
```

```go
func (g *groupServer) GetGroupMembersInfo(ctx context.Context, req *pbgroup.GetGroupMembersInfoReq) (*pbgroup.GetGro
	if len(req.UserIDs) == 0 {
		return nil, errs.ErrArgs.WrapMsg("userIDs empty")
	}
	if req.GroupID == "" {
		return nil, errs.ErrArgs.WrapMsg("groupID empty")
	}
	if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
		return nil, err
	}
	members, err := g.getGroupMembersInfo(ctx, req.GroupID, req.UserIDs)
	if err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupMembersInfoResp{
		Members: members,
	}, nil
}

func (g *groupServer) getGroupMembersInfo(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupMemb
	if len(userIDs) == 0 {
		return nil, nil
	}
	members, err := g.db.FindGroupMembers(ctx, groupID, userIDs)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}
	return datautil.Slice(members, func(e *model.GroupMember) *sdkws.GroupMemberFullInfo {
		return convert.Db2PbGroupMember(e)
	}), nil
}

// GetGroupApplicationList handles functions that get a list of group requests.
func (g *groupServer) GetGroupApplicationList(ctx context.Context, req *pbgroup.GetGroupApplicationListReq) (*pbgrou
	if err := authverify.CheckAccess(ctx, req.FromUserID); err != nil {
		return nil, err
	}
	var (
		groupIDs []string
		err      error
	)
	if len(req.GroupIDs) == 0 {
		groupIDs, err = g.db.FindUserManagedGroupID(ctx, req.FromUserID)
		if err != nil {
			return nil, err
		}
	} else {
		req.GroupIDs = datautil.Distinct(req.GroupIDs)
		if !authverify.IsAdmin(ctx) {
			for _, groupID := range req.GroupIDs {
				if err := g.CheckGroupAdmin(ctx, groupID); err != nil {
					return nil, err
				}
			}
		}
		groupIDs = req.GroupIDs
	}
	resp := &pbgroup.GetGroupApplicationListResp{}
	if len(groupIDs) == 0 {
		return resp, nil
	}
	handleResults := datautil.Slice(req.HandleResults, func(e int32) int {
```

```
		return int(e)
	})
	total, groupRequests, err := g.db.PageGroupRequest(ctx, groupIDs, handleResults, req.Pagination)
	if err != nil {
		return nil, err
	}
	resp.Total = uint32(total)
	if len(groupRequests) == 0 {
		return resp, nil
	}
	var userIDs []string

	for _, gr := range groupRequests {
		userIDs = append(userIDs, gr.UserID)
	}
	userIDs = datautil.Distinct(userIDs)
	userMap, err := g.userClient.GetUsersInfoMap(ctx, userIDs)
	if err != nil {
		return nil, err
	}
	groups, err := g.db.FindGroup(ctx, datautil.Distinct(groupIDs))
	if err != nil {
		return nil, err
	}
	groupMap := datautil.SliceToMap(groups, func(e *model.Group) string {
		return e.GroupID
	})
	if ids := datautil.Single(datautil.Keys(groupMap), groupIDs); len(ids) > 0 {
		return nil, servererrs.ErrGroupIDNotFound.WrapMsg(strings.Join(ids, ","))
	}
	groupMemberNumMap, err := g.db.MapGroupMemberNum(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	owners, err := g.db.FindGroupsOwner(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, owners...); err != nil {
		return nil, err
	}
	ownerMap := datautil.SliceToMap(owners, func(e *model.GroupMember) string {
		return e.GroupID
	})
	resp.GroupRequests = datautil.Slice(groupRequests, func(e *model.GroupRequest) *sdkws.GroupRequest {
		var ownerUserID string
		if owner, ok := ownerMap[e.GroupID]; ok {
			ownerUserID = owner.UserID
		}
		return convert.Db2PbGroupRequest(e, userMap[e.UserID], convert.Db2PbGroupInfo(groupMap[e.GroupID], ownerUserID, g
	})
	return resp, nil
}

func (g *groupServer) GetGroupsInfo(ctx context.Context, req *pbgroup.GetGroupsInfoReq) (*pbgroup.GetGroupsInfoResp,
	if len(req.GroupIDs) == 0 {
		return nil, errs.ErrArgs.WrapMsg("groupID is empty")
	}
	groups, err := g.getGroupsInfo(ctx, req.GroupIDs)
	if err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupsInfoResp{
		GroupInfos: groups,
	}, nil
}
```

```
func (g *groupServer) GetGroupApplicationUnhandledCount(ctx context.Context, req *pbgroup.GetGroupApplicationUnhandl
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	groupIDs, err := g.db.FindUserManagedGroupID(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	count, err := g.db.GetGroupApplicationUnhandledCount(ctx, groupIDs, req.Time)
	if err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupApplicationUnhandledCountResp{
		Count: count,
	}, nil
}

func (g *groupServer) getGroupsInfo(ctx context.Context, groupIDs []string) ([]*sdkws.GroupInfo, error) {
	if len(groupIDs) == 0 {
		return nil, nil
	}
	groups, err := g.db.FindGroup(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	groupMemberNumMap, err := g.db.MapGroupMemberNum(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	owners, err := g.db.FindGroupsOwner(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, owners...); err != nil {
		return nil, err
	}
	ownerMap := datautil.SliceToMap(owners, func(e *model.GroupMember) string {
		return e.GroupID
	})
	return datautil.Slice(groups, func(e *model.Group) *sdkws.GroupInfo {
		var ownerUserID string
		if owner, ok := ownerMap[e.GroupID]; ok {
			ownerUserID = owner.UserID
		}
		return convert.Db2PbGroupInfo(e, ownerUserID, groupMemberNumMap[e.GroupID])
	}), nil
}

func (g *groupServer) GroupApplicationResponse(ctx context.Context, req *pbgroup.GroupApplicationResponseReq) (*pbgr
	if !datautil.Contain(req.HandleResult, constant.GroupResponseAgree, constant.GroupResponseRefuse) {
		return nil, errs.ErrArgs.WrapMsg("HandleResult unknown")
	}
	if !authverify.IsAdmin(ctx) {
		groupMember, err := g.db.TakeGroupMember(ctx, req.GroupID, mcontext.GetOpUserID(ctx))
		if err != nil {
			return nil, err
		}
		if !(groupMember.RoleLevel == constant.GroupOwner || groupMember.RoleLevel == constant.GroupAdmin) {
			return nil, errs.ErrNoPermission.WrapMsg("no group owner or admin")
		}
	}
	group, err := g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
```

```
	groupRequest, err := g.db.TakeGroupRequest(ctx, req.GroupID, req.FromUserID)
	if err != nil {
		return nil, err
	}
	if groupRequest.HandleResult != 0 {
		return nil, servererrs.ErrGroupRequestHandled.WrapMsg("group request already processed")
	}
	var inGroup bool
	if _, err := g.db.TakeGroupMember(ctx, req.GroupID, req.FromUserID); err == nil {
		inGroup = true // Already in group
	} else if !g.IsNotFound(err) {
		return nil, err
	}
	if err := g.userClient.CheckUser(ctx, []string{req.FromUserID}); err != nil {
		return nil, err
	}
	var member *model.GroupMember
	if (!inGroup) && req.HandleResult == constant.GroupResponseAgree {
		member = &model.GroupMember{
			GroupID:        req.GroupID,
			UserID:         req.FromUserID,
			Nickname:       "",
			FaceURL:        "",
			RoleLevel:      constant.GroupOrdinaryUsers,
			JoinTime:       time.Now(),
			JoinSource:     groupRequest.JoinSource,
			MuteEndTime:    time.Unix(0, 0),
			InviterUserID:  groupRequest.InviterUserID,
			OperatorUserID: mcontext.GetOpUserID(ctx),
		}

		if err := g.webhookBeforeMembersJoinGroup(ctx, &g.config.WebhooksConfig.BeforeMemberJoinGroup, []*model.GroupMembe
			return nil, err
		}
	}
	log.ZDebug(ctx, "GroupApplicationResponse", "inGroup", inGroup, "HandleResult", req.HandleResult, "member", member)
	if err := g.db.HandlerGroupRequest(ctx, req.GroupID, req.FromUserID, req.HandledMsg, req.HandleResult, member); err
		return nil, err
	}
	switch req.HandleResult {
	case constant.GroupResponseAgree:
		g.notification.GroupApplicationAcceptedNotification(ctx, req)
		if member == nil {
			log.ZDebug(ctx, "GroupApplicationResponse", "member is nil")
		} else {
			if groupRequest.InviterUserID == "" {
				if err = g.notification.MemberEnterNotification(ctx, req.GroupID, req.FromUserID); err != nil {
					return nil, err
				}
			} else {
				if err = g.notification.GroupApplicationAgreeMemberEnterNotification(ctx, req.GroupID, nil, groupRequest.Invite
					return nil, err
				}
			}
			if err := g.setMemberJoinSeq(ctx, req.GroupID, []string{req.FromUserID}); err != nil {
				return nil, err
			}
		}
	case constant.GroupResponseRefuse:
		g.notification.GroupApplicationRejectedNotification(ctx, req)
	}

	return &pbgroup.GroupApplicationResponseResp{}, nil
}

func (g *groupServer) JoinGroup(ctx context.Context, req *pbgroup.JoinGroupReq) (*pbgroup.JoinGroupResp, error) {
```

```
	user, err := g.userClient.GetUserInfo(ctx, req.InviterUserID)
	if err != nil {
		return nil, err
	}
	group, err := g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	if group.Status == constant.GroupStatusDismissed {
		return nil, servererrs.ErrDismissedAlready.Wrap()
	}

	reqCall := &callbackstruct.CallbackJoinGroupReq{
		GroupID:    req.GroupID,
		GroupType:  string(group.GroupType),
		ApplyID:    req.InviterUserID,
		ReqMessage: req.ReqMessage,
		Ex:         req.Ex,
	}

	if err := g.webhookBeforeApplyJoinGroup(ctx, &g.config.WebhooksConfig.BeforeApplyJoinGroup, reqCall); err != nil &&
		return nil, err
	}

	_, err = g.db.TakeGroupMember(ctx, req.GroupID, req.InviterUserID)
	if err == nil {
		return nil, errs.ErrArgs.Wrap()
	} else if !g.IsNotFound(err) && errs.Unwrap(err) != errs.ErrRecordNotFound {
		return nil, err
	}
	log.ZDebug(ctx, "JoinGroup.groupInfo", "group", group, "eq", group.NeedVerification == constant.Directly)
	if group.NeedVerification == constant.Directly {
		groupMember := &model.GroupMember{
			GroupID:        group.GroupID,
			UserID:         user.UserID,
			RoleLevel:      constant.GroupOrdinaryUsers,
			OperatorUserID: mcontext.GetOpUserID(ctx),
			InviterUserID:  req.InviterUserID,
			JoinTime:       time.Now(),
			MuteEndTime:    time.UnixMilli(0),
		}

		if err := g.webhookBeforeMembersJoinGroup(ctx, &g.config.WebhooksConfig.BeforeMemberJoinGroup, []*model.GroupMembe
			return nil, err
		}

		if err := g.db.CreateGroup(ctx, nil, []*model.GroupMember{groupMember}); err != nil {
			return nil, err
		}

		if err = g.notification.MemberEnterNotification(ctx, req.GroupID, req.InviterUserID); err != nil {
			return nil, err
		}
		if err := g.setMemberJoinSeq(ctx, req.GroupID, []string{req.InviterUserID}); err != nil {
			return nil, err
		}
		g.webhookAfterJoinGroup(ctx, &g.config.WebhooksConfig.AfterJoinGroup, req)

		return &pbgroup.JoinGroupResp{}, nil
	}

	groupRequest := model.GroupRequest{
		UserID:      req.InviterUserID,
		ReqMsg:      req.ReqMessage,
		GroupID:     req.GroupID,
		JoinSource:  req.JoinSource,
```

```
		ReqTime:     time.Now(),
		HandledTime: time.Unix(0, 0),
		Ex:          req.Ex,
	}
	if err = g.db.CreateGroupRequest(ctx, []*model.GroupRequest{&groupRequest}); err != nil {
		return nil, err
	}
	g.notification.JoinGroupApplicationNotification(ctx, req, &groupRequest)
	return &pbgroup.JoinGroupResp{}, nil
}

func (g *groupServer) QuitGroup(ctx context.Context, req *pbgroup.QuitGroupReq) (*pbgroup.QuitGroupResp, error) {
	if req.UserID == "" {
		req.UserID = mcontext.GetOpUserID(ctx)
	} else {
		if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
			return nil, err
		}
	}
	member, err := g.db.TakeGroupMember(ctx, req.GroupID, req.UserID)
	if err != nil {
		return nil, err
	}
	if member.RoleLevel == constant.GroupOwner {
		return nil, errs.ErrNoPermission.WrapMsg("group owner can't quit")
	}
	if err := g.PopulateGroupMember(ctx, member); err != nil {
		return nil, err
	}
	err = g.db.DeleteGroupMember(ctx, req.GroupID, []string{req.UserID})
	if err != nil {
		return nil, err
	}
	g.notification.MemberQuitNotification(ctx, g.groupMemberDB2PB(member, 0))
	if err := g.deleteMemberAndSetConversationSeq(ctx, req.GroupID, []string{req.UserID}); err != nil {
		return nil, err
	}
	g.webhookAfterQuitGroup(ctx, &g.config.WebhooksConfig.AfterQuitGroup, req)

	return &pbgroup.QuitGroupResp{}, nil
}

func (g *groupServer) deleteMemberAndSetConversationSeq(ctx context.Context, groupID string, userIDs []string) error
	conversationID := msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, groupID)
	maxSeq, err := g.msgClient.GetConversationMaxSeq(ctx, conversationID)
	if err != nil {
		return err
	}
	return g.conversationClient.SetConversationMaxSeq(ctx, conversationID, userIDs, maxSeq)
}

func (g *groupServer) setMemberJoinSeq(ctx context.Context, groupID string, userIDs []string) error {
	conversationID := msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, groupID)
	return g.conversationClient.SetConversationMaxSeq(ctx, conversationID, userIDs, 0)
}

func (g *groupServer) SetGroupInfo(ctx context.Context, req *pbgroup.SetGroupInfoReq) (*pbgroup.SetGroupInfoResp, er
	var opMember *model.GroupMember
	if !authverify.IsAdmin(ctx) {
		var err error
		opMember, err = g.db.TakeGroupMember(ctx, req.GroupInfoForSet.GroupID, mcontext.GetOpUserID(ctx))
		if err != nil {
			return nil, err
		}
		if !(opMember.RoleLevel == constant.GroupOwner || opMember.RoleLevel == constant.GroupAdmin) {
			return nil, errs.ErrNoPermission.WrapMsg("no group owner or admin")
```

```
    }
    if err := g.PopulateGroupMember(ctx, opMember); err != nil {
        return nil, err
    }
}

if err := g.webhookBeforeSetGroupInfo(ctx, &g.config.WebhooksConfig.BeforeSetGroupInfo, req); err != nil && err !=
    return nil, err
}

group, err := g.db.TakeGroup(ctx, req.GroupInfoForSet.GroupID)
if err != nil {
    return nil, err
}
if group.Status == constant.GroupStatusDismissed {
    return nil, servererrs.ErrDismissedAlready.Wrap()
}

count, err := g.db.FindGroupMemberNum(ctx, group.GroupID)
if err != nil {
    return nil, err
}
owner, err := g.db.TakeGroupOwner(ctx, group.GroupID)
if err != nil {
    return nil, err
}
if err := g.PopulateGroupMember(ctx, owner); err != nil {
    return nil, err
}
update := UpdateGroupInfoMap(ctx, req.GroupInfoForSet)
if len(update) == 0 {
    return &pbgroup.SetGroupInfoResp{}, nil
}
if err := g.db.UpdateGroup(ctx, group.GroupID, update); err != nil {
    return nil, err
}
group, err = g.db.TakeGroup(ctx, req.GroupInfoForSet.GroupID)
if err != nil {
    return nil, err
}
tips := &sdkws.GroupInfoSetTips{
    Group:    g.groupDB2PB(group, owner.UserID, count),
    MuteTime: 0,
    OpUser:   &sdkws.GroupMemberFullInfo{},
}
if opMember != nil {
    tips.OpUser = g.groupMemberDB2PB(opMember, 0)
}
num := len(update)
if req.GroupInfoForSet.Notification != "" {
    num -= 3
    func() {
        conversation := &pbconv.ConversationReq{
            ConversationID:   msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, req.GroupInfoForSet.G
            ConversationType: constant.ReadGroupChatType,
            GroupID:          req.GroupInfoForSet.GroupID,
        }
        resp, err := g.GetGroupMemberUserIDs(ctx, &pbgroup.GetGroupMemberUserIDsReq{GroupID: req.GroupInfoForSet.GroupID
        if err != nil {
            log.ZWarn(ctx, "GetGroupMemberIDs is failed.", err)
            return
        }
        conversation.GroupAtType = &wrapperspb.Int32Value{Value: constant.GroupNotification}
        if err := g.conversationClient.SetConversations(ctx, resp.UserIDs, conversation); err != nil {
            log.ZWarn(ctx, "SetConversations", err, "UserIDs", resp.UserIDs, "conversation", conversation)
        }
```

```
    }()
    notficationFlag := true
    g.notification.GroupInfoSetAnnouncementNotification(ctx, &sdkws.GroupInfoSetAnnouncementTips{Group: tips.Group, Op
  }
  if req.GroupInfoForSet.GroupName != "" {
    num--
    g.notification.GroupInfoSetNameNotification(ctx, &sdkws.GroupInfoSetNameTips{Group: tips.Group, OpUser: tips.OpUse
  }
  if num > 0 {
    g.notification.GroupInfoSetNotification(ctx, tips)
  }

  g.webhookAfterSetGroupInfo(ctx, &g.config.WebhooksConfig.AfterSetGroupInfo, req)

  return &pbgroup.SetGroupInfoResp{}, nil
}

func (g *groupServer) SetGroupInfoEx(ctx context.Context, req *pbgroup.SetGroupInfoExReq) (*pbgroup.SetGroupInfoExRe
  var opMember *model.GroupMember

  if !authverify.IsAdmin(ctx) {
    var err error

    opMember, err = g.db.TakeGroupMember(ctx, req.GroupID, mcontext.GetOpUserID(ctx))
    if err != nil {
      return nil, err
    }

    if !(opMember.RoleLevel == constant.GroupOwner || opMember.RoleLevel == constant.GroupAdmin) {
      return nil, errs.ErrNoPermission.WrapMsg("no group owner or admin")
    }

    if err := g.PopulateGroupMember(ctx, opMember); err != nil {
      return nil, err
    }
  }

  if err := g.webhookBeforeSetGroupInfoEx(ctx, &g.config.WebhooksConfig.BeforeSetGroupInfoEx, req); err != nil && err
    return nil, err
  }

  group, err := g.db.TakeGroup(ctx, req.GroupID)
  if err != nil {
    return nil, err
  }
  if group.Status == constant.GroupStatusDismissed {
    return nil, servererrs.ErrDismissedAlready.Wrap()
  }

  count, err := g.db.FindGroupMemberNum(ctx, group.GroupID)
  if err != nil {
    return nil, err
  }

  owner, err := g.db.TakeGroupOwner(ctx, group.GroupID)
  if err != nil {
    return nil, err
  }

  if err := g.PopulateGroupMember(ctx, owner); err != nil {
    return nil, err
  }

  updatedData, normalFlag, groupNameFlag, notificationFlag, err := UpdateGroupInfoExMap(ctx, req)
  if len(updatedData) == 0 {
    return &pbgroup.SetGroupInfoExResp{}, nil
```

```
	}

	if err != nil {
		return nil, err
	}

	if err := g.db.UpdateGroup(ctx, group.GroupID, updatedData); err != nil {
		return nil, err
	}

	group, err = g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}

	tips := &sdkws.GroupInfoSetTips{
		Group:    g.groupDB2PB(group, owner.UserID, count),
		MuteTime: 0,
		OpUser:   &sdkws.GroupMemberFullInfo{},
	}

	if opMember != nil {
		tips.OpUser = g.groupMemberDB2PB(opMember, 0)
	}

	if notificationFlag {
		if req.Notification.Value != "" {
			conversation := &pbconv.ConversationReq{
				ConversationID:   msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, req.GroupID),
				ConversationType: constant.ReadGroupChatType,
				GroupID:          req.GroupID,
			}

			resp, err := g.GetGroupMemberUserIDs(ctx, &pbgroup.GetGroupMemberUserIDsReq{GroupID: req.GroupID})
			if err != nil {
				log.ZWarn(ctx, "GetGroupMemberIDs is failed.", err)
				return nil, err
			}

			conversation.GroupAtType = &wrapperspb.Int32Value{Value: constant.GroupNotification}
			if err := g.conversationClient.SetConversations(ctx, resp.UserIDs, conversation); err != nil {
				log.ZWarn(ctx, "SetConversations", err, "UserIDs", resp.UserIDs, "conversation", conversation)
			}

			g.notification.GroupInfoSetAnnouncementNotification(ctx, &sdkws.GroupInfoSetAnnouncementTips{Group: tips.Group,
		} else {
			notificationFlag = false
			g.notification.GroupInfoSetAnnouncementNotification(ctx, &sdkws.GroupInfoSetAnnouncementTips{Group: tips.Group,
		}
	}

	if groupNameFlag {
		g.notification.GroupInfoSetNameNotification(ctx, &sdkws.GroupInfoSetNameTips{Group: tips.Group, OpUser: tips.OpUs
	}

	// if updatedData > 0, send the normal notification
	if normalFlag {
		g.notification.GroupInfoSetNotification(ctx, tips)
	}

	g.webhookAfterSetGroupInfoEx(ctx, &g.config.WebhooksConfig.AfterSetGroupInfoEx, req)

	return &pbgroup.SetGroupInfoExResp{}, nil
}

func (g *groupServer) TransferGroupOwner(ctx context.Context, req *pbgroup.TransferGroupOwnerReq) (*pbgroup.Transfer
```

```
	group, err := g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}

	if group.Status == constant.GroupStatusDismissed {
		return nil, servererrs.ErrDismissedAlready.Wrap()
	}

	if req.OldOwnerUserID == req.NewOwnerUserID {
		return nil, errs.ErrArgs.WrapMsg("OldOwnerUserID == NewOwnerUserID")
	}

	members, err := g.db.FindGroupMembers(ctx, req.GroupID, []string{req.OldOwnerUserID, req.NewOwnerUserID})
	if err != nil {
		return nil, err
	}

	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}

	memberMap := datautil.SliceToMap(members, func(e *model.GroupMember) string { return e.UserID })
	if ids := datautil.Single([]string{req.OldOwnerUserID, req.NewOwnerUserID}, datautil.Keys(memberMap)); len(ids) > 0
		return nil, errs.ErrArgs.WrapMsg("user not in group " + strings.Join(ids, ","))
	}

	oldOwner := memberMap[req.OldOwnerUserID]
	if oldOwner == nil {
		return nil, errs.ErrArgs.WrapMsg("OldOwnerUserID not in group " + req.NewOwnerUserID)
	}

	newOwner := memberMap[req.NewOwnerUserID]
	if newOwner == nil {
		return nil, errs.ErrArgs.WrapMsg("NewOwnerUser not in group " + req.NewOwnerUserID)
	}

	if !authverify.IsAdmin(ctx) {
		if !(mcontext.GetOpUserID(ctx) == oldOwner.UserID && oldOwner.RoleLevel == constant.GroupOwner) {
			return nil, errs.ErrNoPermission.WrapMsg("no permission transfer group owner")
		}
	}

	if newOwner.MuteEndTime.After(time.Now()) {
		if _, err := g.CancelMuteGroupMember(ctx, &pbgroup.CancelMuteGroupMemberReq{
			GroupID: group.GroupID,
			UserID:  req.NewOwnerUserID}); err != nil {
			return nil, err
		}
	}

	if err := g.db.TransferGroupOwner(ctx, req.GroupID, req.OldOwnerUserID, req.NewOwnerUserID, newOwner.RoleLevel); er
		return nil, err
	}

	g.webhookAfterTransferGroupOwner(ctx, &g.config.WebhooksConfig.AfterTransferGroupOwner, req)

	g.notification.GroupOwnerTransferredNotification(ctx, req)

	return &pbgroup.TransferGroupOwnerResp{}, nil
}

func (g *groupServer) GetGroups(ctx context.Context, req *pbgroup.GetGroupsReq) (*pbgroup.GetGroupsResp, error) {
	var (
		group []*model.Group
		err   error
```

```
	)
	var resp pbgroup.GetGroupsResp
	if req.GroupID != "" {
		group, err = g.db.FindGroup(ctx, []string{req.GroupID})
		resp.Total = uint32(len(group))
	} else {
		var total int64
		total, group, err = g.db.SearchGroup(ctx, req.GroupName, req.Pagination)
		resp.Total = uint32(total)
	}

	if err != nil {
		return nil, err
	}

	groupIDs := datautil.Slice(group, func(e *model.Group) string {
		return e.GroupID
	})

	ownerMembers, err := g.db.FindGroupsOwner(ctx, groupIDs)
	if err != nil {
		return nil, err
	}

	ownerMemberMap := datautil.SliceToMap(ownerMembers, func(e *model.GroupMember) string {
		return e.GroupID
	})
	groupMemberNumMap, err := g.db.MapGroupMemberNum(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	resp.Groups = datautil.Slice(group, func(group *model.Group) *pbgroup.CMSGroup {
		var (
			userID   string
			username string
		)
		if member, ok := ownerMemberMap[group.GroupID]; ok {
			userID = member.UserID
			username = member.Nickname
		}
		return convert.Db2PbCMSGroup(group, userID, username, groupMemberNumMap[group.GroupID])
	})
	return &resp, nil
}

func (g *groupServer) GetGroupMembersCMS(ctx context.Context, req *pbgroup.GetGroupMembersCMSReq) (*pbgroup.GetGroup
	if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
		return nil, err
	}
	total, members, err := g.db.SearchGroupMember(ctx, req.UserName, req.GroupID, req.Pagination)
	if err != nil {
		return nil, err
	}
	var resp pbgroup.GetGroupMembersCMSResp
	resp.Total = uint32(total)
	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}
	resp.Members = datautil.Slice(members, func(e *model.GroupMember) *sdkws.GroupMemberFullInfo {
		return convert.Db2PbGroupMember(e)
	})
	return &resp, nil
}

func (g *groupServer) GetUserReqApplicationList(ctx context.Context, req *pbgroup.GetUserReqApplicationListReq) (*pb
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
```

```
		return nil, err
	}
	user, err := g.userClient.GetUserInfo(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	handleResults := datautil.Slice(req.HandleResults, func(e int32) int {
		return int(e)
	})
	total, requests, err := g.db.PageGroupRequestUser(ctx, req.UserID, req.GroupIDs, handleResults, req.Pagination)
	if err != nil {
		return nil, err
	}
	if len(requests) == 0 {
		return &pbgroup.GetUserReqApplicationListResp{Total: uint32(total)}, nil
	}
	groupIDs := datautil.Distinct(datautil.Slice(requests, func(e *model.GroupRequest) string {
		return e.GroupID
	}))
	groups, err := g.db.FindGroup(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	groupMap := datautil.SliceToMap(groups, func(e *model.Group) string {
		return e.GroupID
	})
	owners, err := g.db.FindGroupsOwner(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, owners...); err != nil {
		return nil, err
	}
	ownerMap := datautil.SliceToMap(owners, func(e *model.GroupMember) string {
		return e.GroupID
	})
	groupMemberNum, err := g.db.MapGroupMemberNum(ctx, groupIDs)
	if err != nil {
		return nil, err
	}
	return &pbgroup.GetUserReqApplicationListResp{
		Total: uint32(total),
		GroupRequests: datautil.Slice(requests, func(e *model.GroupRequest) *sdkws.GroupRequest {
			var ownerUserID string
			if owner, ok := ownerMap[e.GroupID]; ok {
				ownerUserID = owner.UserID
			}
			return convert.Db2PbGroupRequest(e, user, convert.Db2PbGroupInfo(groupMap[e.GroupID], ownerUserID, groupMemberNu
		}),
	}, nil
}

func (g *groupServer) DismissGroup(ctx context.Context, req *pbgroup.DismissGroupReq) (*pbgroup.DismissGroupResp, er
	owner, err := g.db.TakeGroupOwner(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	if !authverify.IsAdmin(ctx) {
		if owner.UserID != mcontext.GetOpUserID(ctx) {
			return nil, errs.ErrNoPermission.WrapMsg("not group owner")
		}
	}
	if err := g.PopulateGroupMember(ctx, owner); err != nil {
		return nil, err
	}
	group, err := g.db.TakeGroup(ctx, req.GroupID)
```

```
	if err != nil {
		return nil, err
	}
	if !req.DeleteMember && group.Status == constant.GroupStatusDismissed {
		return nil, servererrs.ErrDismissedAlready.WrapMsg("group status is dismissed")
	}
	if err := g.db.DismissGroup(ctx, req.GroupID, req.DeleteMember); err != nil {
		return nil, err
	}
	if !req.DeleteMember {
		num, err := g.db.FindGroupMemberNum(ctx, req.GroupID)
		if err != nil {
			return nil, err
		}
		group.Status = constant.GroupStatusDismissed
		tips := &sdkws.GroupDismissedTips{
			Group:  g.groupDB2PB(group, owner.UserID, num),
			OpUser: &sdkws.GroupMemberFullInfo{},
		}
		if mcontext.GetOpUserID(ctx) == owner.UserID {
			tips.OpUser = g.groupMemberDB2PB(owner, 0)
		}
		g.notification.GroupDismissedNotification(ctx, tips, req.SendMessage)
	}
	membersID, err := g.db.FindGroupMemberUserID(ctx, group.GroupID)
	if err != nil {
		return nil, err
	}
	cbReq := &callbackstruct.CallbackDisMissGroupReq{
		GroupID:   req.GroupID,
		OwnerID:   owner.UserID,
		MembersID: membersID,
		GroupType: string(group.GroupType),
	}

	g.webhookAfterDismissGroup(ctx, &g.config.WebhooksConfig.AfterDismissGroup, cbReq)

	return &pbgroup.DismissGroupResp{}, nil
}

func (g *groupServer) MuteGroupMember(ctx context.Context, req *pbgroup.MuteGroupMemberReq) (*pbgroup.MuteGroupMembe
	member, err := g.db.TakeGroupMember(ctx, req.GroupID, req.UserID)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, member); err != nil {
		return nil, err
	}
	if !authverify.IsAdmin(ctx) {
		opMember, err := g.db.TakeGroupMember(ctx, req.GroupID, mcontext.GetOpUserID(ctx))
		if err != nil {
			return nil, err
		}
		switch member.RoleLevel {
		case constant.GroupOwner:
			return nil, errs.ErrNoPermission.WrapMsg("set group owner mute")
		case constant.GroupAdmin:
			if opMember.RoleLevel != constant.GroupOwner {
				return nil, errs.ErrNoPermission.WrapMsg("set group admin mute")
			}
		case constant.GroupOrdinaryUsers:
			if !(opMember.RoleLevel == constant.GroupAdmin || opMember.RoleLevel == constant.GroupOwner) {
				return nil, errs.ErrNoPermission.WrapMsg("set group ordinary users mute")
			}
		}
	}
```

```
	data := UpdateGroupMemberMutedTimeMap(time.Now().Add(time.Second * time.Duration(req.MutedSeconds)))
	if err := g.db.UpdateGroupMember(ctx, member.GroupID, member.UserID, data); err != nil {
		return nil, err
	}
	g.notification.GroupMemberMutedNotification(ctx, req.GroupID, req.UserID, req.MutedSeconds)
	return &pbgroup.MuteGroupMemberResp{}, nil
}

func (g *groupServer) CancelMuteGroupMember(ctx context.Context, req *pbgroup.CancelMuteGroupMemberReq) (*pbgroup.Ca
	member, err := g.db.TakeGroupMember(ctx, req.GroupID, req.UserID)
	if err != nil {
		return nil, err
	}

	if err := g.PopulateGroupMember(ctx, member); err != nil {
		return nil, err
	}

	if !authverify.IsAdmin(ctx) {
		opMember, err := g.db.TakeGroupMember(ctx, req.GroupID, mcontext.GetOpUserID(ctx))
		if err != nil {
			return nil, err
		}

		switch member.RoleLevel {
		case constant.GroupOwner:
			return nil, errs.ErrNoPermission.WrapMsg("Can not set group owner unmute")
		case constant.GroupAdmin:
			if opMember.RoleLevel != constant.GroupOwner {
				return nil, errs.ErrNoPermission.WrapMsg("Can not set group admin unmute")
			}
		case constant.GroupOrdinaryUsers:
			if !(opMember.RoleLevel == constant.GroupAdmin || opMember.RoleLevel == constant.GroupOwner) {
				return nil, errs.ErrNoPermission.WrapMsg("Can not set group ordinary users unmute")
			}
		}
	}

	data := UpdateGroupMemberMutedTimeMap(time.Unix(0, 0))
	if err := g.db.UpdateGroupMember(ctx, member.GroupID, member.UserID, data); err != nil {
		return nil, err
	}

	g.notification.GroupMemberCancelMutedNotification(ctx, req.GroupID, req.UserID)

	return &pbgroup.CancelMuteGroupMemberResp{}, nil
}

func (g *groupServer) MuteGroup(ctx context.Context, req *pbgroup.MuteGroupReq) (*pbgroup.MuteGroupResp, error) {
	if err := g.CheckGroupAdmin(ctx, req.GroupID); err != nil {
		return nil, err
	}
	if err := g.db.UpdateGroup(ctx, req.GroupID, UpdateGroupStatusMap(constant.GroupStatusMuted)); err != nil {
		return nil, err
	}
	g.notification.GroupMutedNotification(ctx, req.GroupID)
	return &pbgroup.MuteGroupResp{}, nil
}

func (g *groupServer) CancelMuteGroup(ctx context.Context, req *pbgroup.CancelMuteGroupReq) (*pbgroup.CancelMuteGrou
	if err := g.CheckGroupAdmin(ctx, req.GroupID); err != nil {
		return nil, err
	}
	if err := g.db.UpdateGroup(ctx, req.GroupID, UpdateGroupStatusMap(constant.GroupOk)); err != nil {
		return nil, err
	}
```

```
	g.notification.GroupCancelMutedNotification(ctx, req.GroupID)
	return &pbgroup.CancelMuteGroupResp{}, nil
}

func (g *groupServer) SetGroupMemberInfo(ctx context.Context, req *pbgroup.SetGroupMemberInfoReq) (*pbgroup.SetGroup
	if len(req.Members) == 0 {
		return nil, errs.ErrArgs.WrapMsg("members empty")
	}
	opUserID := mcontext.GetOpUserID(ctx)
	if opUserID == "" {
		return nil, errs.ErrNoPermission.WrapMsg("no op user id")
	}
	isAppManagerUid := authverify.IsAdmin(ctx)
	groupMembers := make(map[string][]*pbgroup.SetGroupMemberInfo)
	for i, member := range req.Members {
		if member.RoleLevel != nil {
			switch member.RoleLevel.Value {
			case constant.GroupOwner:
				return nil, errs.ErrNoPermission.WrapMsg("cannot set ungroup owner")
			case constant.GroupAdmin, constant.GroupOrdinaryUsers:
			default:
				return nil, errs.ErrArgs.WrapMsg("invalid role level")
			}
		}
		groupMembers[member.GroupID] = append(groupMembers[member.GroupID], req.Members[i])
	}
	for groupID, members := range groupMembers {
		temp := make(map[string]struct{})
		userIDs := make([]string, 0, len(members)+1)
		for _, member := range members {
			if _, ok := temp[member.UserID]; ok {
				return nil, errs.ErrArgs.WrapMsg(fmt.Sprintf("repeat group %s user %s", member.GroupID, member.UserID))
			}
			temp[member.UserID] = struct{}{}
			userIDs = append(userIDs, member.UserID)
		}
		if _, ok := temp[opUserID]; !ok {
			userIDs = append(userIDs, opUserID)
		}
		dbMembers, err := g.db.FindGroupMembers(ctx, groupID, userIDs)
		if err != nil {
			return nil, err
		}
		opUserIndex := -1
		for i, member := range dbMembers {
			if member.UserID == opUserID {
				opUserIndex = i
				break
			}
		}
		switch len(userIDs) - len(dbMembers) {
		case 0:
			if !isAppManagerUid {
				roleLevel := dbMembers[opUserIndex].RoleLevel
				var (
					dbSelf  = &model.GroupMember{}
					reqSelf *pbgroup.SetGroupMemberInfo
				)
				switch roleLevel {
				case constant.GroupOwner:
					for _, member := range dbMembers {
						if member.UserID == opUserID {
							dbSelf = member
							break
						}
					}
```

```
        case constant.GroupAdmin:
            for _, member := range dbMembers {
                if member.UserID == opUserID {
                    dbSelf = member
                }
                if member.RoleLevel == constant.GroupOwner {
                    return nil, errs.ErrNoPermission.WrapMsg("admin can not change group owner")
                }
                if member.RoleLevel == constant.GroupAdmin && member.UserID != opUserID {
                    return nil, errs.ErrNoPermission.WrapMsg("admin can not change other group admin")
                }
            }
        case constant.GroupOrdinaryUsers:
            for _, member := range dbMembers {
                if member.UserID == opUserID {
                    dbSelf = member
                }
                if !(member.RoleLevel == constant.GroupOrdinaryUsers && member.UserID == opUserID) {
                    return nil, errs.ErrNoPermission.WrapMsg("ordinary users can not change other role level")
                }
            }
        default:
            for _, member := range dbMembers {
                if member.UserID == opUserID {
                    dbSelf = member
                }
                if member.RoleLevel >= roleLevel {
                    return nil, errs.ErrNoPermission.WrapMsg("can not change higher role level")
                }
            }
        }
        for _, member := range req.Members {
            if member.UserID == opUserID {
                reqSelf = member
                break
            }
        }
        if reqSelf != nil && reqSelf.RoleLevel != nil {
            if reqSelf.RoleLevel.GetValue() > dbSelf.RoleLevel {
                return nil, errs.ErrNoPermission.WrapMsg("can not improve role level by self")
            }
            if roleLevel == constant.GroupOwner {
                return nil, errs.ErrArgs.WrapMsg("group owner can not change own role level") // Prevent the absence of a gr
            }
        }
    }
    case 1:
    if opUserIndex >= 0 {
        return nil, errs.ErrArgs.WrapMsg("user not in group")
    }
    if !isAppManagerUid {
        return nil, errs.ErrNoPermission.WrapMsg("user not in group")
    }
    default:
    return nil, errs.ErrArgs.WrapMsg("user not in group")
    }
}

for i := 0; i < len(req.Members); i++ {

    if err := g.webhookBeforeSetGroupMemberInfo(ctx, &g.config.WebhooksConfig.BeforeSetGroupMemberInfo, req.Members[i
    return nil, err
    }

}
if err := g.db.UpdateGroupMembers(ctx, datautil.Slice(req.Members, func(e *pbgroup.SetGroupMemberInfo) *common.Bato
```

```
		return &common.BatchUpdateGroupMember{
			GroupID: e.GroupID,
			UserID:  e.UserID,
			Map:     UpdateGroupMemberMap(e),
		}
	})); err != nil {
		return nil, err
	}
	for _, member := range req.Members {
		if member.RoleLevel != nil {
			switch member.RoleLevel.Value {
			case constant.GroupAdmin:
				g.notification.GroupMemberSetToAdminNotification(ctx, member.GroupID, member.UserID)
			case constant.GroupOrdinaryUsers:
				g.notification.GroupMemberSetToOrdinaryUserNotification(ctx, member.GroupID, member.UserID)
			}
		}
		if member.Nickname != nil || member.FaceURL != nil || member.Ex != nil {
			g.notification.GroupMemberInfoSetNotification(ctx, member.GroupID, member.UserID)
		}
	}
	for i := 0; i < len(req.Members); i++ {
		g.webhookAfterSetGroupMemberInfo(ctx, &g.config.WebhooksConfig.AfterSetGroupMemberInfo, req.Members[i])
	}

	return &pbgroup.SetGroupMemberInfoResp{}, nil
}

func (g *groupServer) GetGroupAbstractInfo(ctx context.Context, req *pbgroup.GetGroupAbstractInfoReq) (*pbgroup.GetG
	if len(req.GroupIDs) == 0 {
		return nil, errs.ErrArgs.WrapMsg("groupIDs empty")
	}
	if datautil.Duplicate(req.GroupIDs) {
		return nil, errs.ErrArgs.WrapMsg("groupIDs duplicate")
	}
	for _, groupID := range req.GroupIDs {
		if err := g.checkAdminOrInGroup(ctx, groupID); err != nil {
			return nil, err
		}
	}
	groups, err := g.db.FindGroup(ctx, req.GroupIDs)
	if err != nil {
		return nil, err
	}
	if ids := datautil.Single(req.GroupIDs, datautil.Slice(groups, func(group *model.Group) string {
		return group.GroupID
	})); len(ids) > 0 {
		return nil, servererrs.ErrGroupIDNotFound.WrapMsg("not found group " + strings.Join(ids, ","))
	}
	groupUserMap, err := g.db.MapGroupMemberUserID(ctx, req.GroupIDs)
	if err != nil {
		return nil, err
	}
	if ids := datautil.Single(req.GroupIDs, datautil.Keys(groupUserMap)); len(ids) > 0 {
		return nil, servererrs.ErrGroupIDNotFound.WrapMsg(fmt.Sprintf("group %s not found member", strings.Join(ids, ",")
	}
	return &pbgroup.GetGroupAbstractInfoResp{
		GroupAbstractInfos: datautil.Slice(groups, func(group *model.Group) *pbgroup.GroupAbstractInfo {
			users := groupUserMap[group.GroupID]
			return convert.Db2PbGroupAbstractInfo(group.GroupID, users.MemberNum, users.Hash)
		}),
	}, nil
}

func (g *groupServer) GetUserInGroupMembers(ctx context.Context, req *pbgroup.GetUserInGroupMembersReq) (*pbgroup.Ge
	if len(req.GroupIDs) == 0 {
```

```
		return nil, errs.ErrArgs.WrapMsg("groupIDs empty")
	}
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	members, err := g.db.FindGroupMemberUser(ctx, req.GroupIDs, req.UserID)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}
	return &pbgroup.GetUserInGroupMembersResp{
		Members: datautil.Slice(members, func(e *model.GroupMember) *sdkws.GroupMemberFullInfo {
			return convert.Db2PbGroupMember(e)
		}),
	}, nil
}

func (g *groupServer) GetGroupMemberUserIDs(ctx context.Context, req *pbgroup.GetGroupMemberUserIDsReq) (*pbgroup.Ge
	userIDs, err := g.db.FindGroupMemberUserID(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	if err := authverify.CheckAccessIn(ctx, userIDs...); err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupMemberUserIDsResp{
		UserIDs: userIDs,
	}, nil
}

func (g *groupServer) GetGroupMemberRoleLevel(ctx context.Context, req *pbgroup.GetGroupMemberRoleLevelReq) (*pbgrou
	if len(req.RoleLevels) == 0 {
		return nil, errs.ErrArgs.WrapMsg("RoleLevels empty")
	}
	if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
		return nil, err
	}
	members, err := g.db.FindGroupMemberRoleLevels(ctx, req.GroupID, req.RoleLevels)
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}
	return &pbgroup.GetGroupMemberRoleLevelResp{
		Members: datautil.Slice(members, func(e *model.GroupMember) *sdkws.GroupMemberFullInfo {
			return convert.Db2PbGroupMember(e)
		}),
	}, nil
}

func (g *groupServer) GetGroupUsersReqApplicationList(ctx context.Context, req *pbgroup.GetGroupUsersReqApplicationL
	if err := g.CheckGroupAdmin(ctx, req.GroupID); err != nil {
		return nil, err
	}
	requests, err := g.db.FindGroupRequests(ctx, req.GroupID, req.UserIDs)
	if err != nil {
		return nil, err
	}

	if len(requests) == 0 {
		return &pbgroup.GetGroupUsersReqApplicationListResp{}, nil
	}
```

```
	groupIDs := datautil.Distinct(datautil.Slice(requests, func(e *model.GroupRequest) string {
		return e.GroupID
	}))

	groups, err := g.db.FindGroup(ctx, groupIDs)
	if err != nil {
		return nil, err
	}

	groupMap := datautil.SliceToMap(groups, func(e *model.Group) string {
		return e.GroupID
	})

	if ids := datautil.Single(groupIDs, datautil.Keys(groupMap)); len(ids) > 0 {
		return nil, servererrs.ErrGroupIDNotFound.WrapMsg(strings.Join(ids, ","))
	}

	userMap, err := g.userClient.GetUsersInfoMap(ctx, req.UserIDs)
	if err != nil {
		return nil, err
	}

	owners, err := g.db.FindGroupsOwner(ctx, groupIDs)
	if err != nil {
		return nil, err
	}

	if err := g.PopulateGroupMember(ctx, owners...); err != nil {
		return nil, err
	}

	ownerMap := datautil.SliceToMap(owners, func(e *model.GroupMember) string {
		return e.GroupID
	})

	groupMemberNum, err := g.db.MapGroupMemberNum(ctx, groupIDs)
	if err != nil {
		return nil, err
	}

	return &pbgroup.GetGroupUsersReqApplicationListResp{
		Total: int64(len(requests)),
		GroupRequests: datautil.Slice(requests, func(e *model.GroupRequest) *sdkws.GroupRequest {
			var ownerUserID string
			if owner, ok := ownerMap[e.GroupID]; ok {
				ownerUserID = owner.UserID
			}

			var userInfo *sdkws.UserInfo
			if user, ok := userMap[e.UserID]; !ok {
				userInfo = user
			}

			return convert.Db2PbGroupRequest(e, userInfo, convert.Db2PbGroupInfo(groupMap[e.GroupID], ownerUserID, groupMemb
		}),
	}, nil
}

func (g *groupServer) GetSpecifiedUserGroupRequestInfo(ctx context.Context, req *pbgroup.GetSpecifiedUserGroupReques
	opUserID := mcontext.GetOpUserID(ctx)

	owners, err := g.db.FindGroupsOwner(ctx, []string{req.GroupID})
	if err != nil {
		return nil, err
	}
```

```
	if req.UserID != opUserID {
		adminIDs, err := g.db.GetGroupRoleLevelMemberIDs(ctx, req.GroupID, constant.GroupAdmin)
		if err != nil {
			return nil, err
		}

		adminIDs = append(adminIDs, owners[0].UserID)
		adminIDs = append(adminIDs, g.adminUserIDs...)

		if !datautil.Contain(opUserID, adminIDs...) {
			return nil, errs.ErrNoPermission.WrapMsg("opUser no permission")
		}
	}

	requests, err := g.db.FindGroupRequests(ctx, req.GroupID, []string{req.UserID})
	if err != nil {
		return nil, err
	}

	if len(requests) == 0 {
		return &pbgroup.GetSpecifiedUserGroupRequestInfoResp{}, nil
	}

	groups, err := g.db.FindGroup(ctx, []string{req.GroupID})
	if err != nil {
		return nil, err
	}

	userInfos, err := g.userClient.GetUsersInfo(ctx, []string{req.UserID})
	if err != nil {
		return nil, err
	}

	groupMemberNum, err := g.db.MapGroupMemberNum(ctx, []string{req.GroupID})
	if err != nil {
		return nil, err
	}

	resp := &pbgroup.GetSpecifiedUserGroupRequestInfoResp{
		GroupRequests: make([]*sdkws.GroupRequest, 0, len(requests)),
	}

	for _, request := range requests {
		resp.GroupRequests = append(resp.GroupRequests, convert.Db2PbGroupRequest(request, userInfos[0], convert.Db2PbGrou
	}

	resp.Total = uint32(len(requests))

	return resp, nil
}
```

internal/rpc/group/notification.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
	"context"
	"errors"
	"fmt"
	"time"

	"github.com/google/uuid"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"

	"go.mongodb.org/mongo-driver/mongo"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/versionctx"
	"github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
	"github.com/openimsdk/open-im-server/v3/pkg/notification"
	"github.com/openimsdk/open-im-server/v3/pkg/notification/common_user"
	"github.com/openimsdk/protocol/constant"
	pbgroup "github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/stringutil"
)

// GroupApplicationReceiver
const (
	applicantReceiver = iota
	adminReceiver
)

func NewNotificationSender(db controller.GroupDatabase, config *Config, userClient *rpcli.UserClient, msgClient *rpc
	return &NotificationSender{
		NotificationSender: notification.NewNotificationSender(&config.NotificationConfig,
			notification.WithRpcClient(func(ctx context.Context, req *msg.SendMsgReq) (*msg.SendMsgResp, error) {
				return msgClient.SendMsg(ctx, req)
			}),
			notification.WithUserRpcClient(userClient.GetUserInfo),
		),
		getUsersInfo: func(ctx context.Context, userIDs []string) ([]common_user.CommonUser, error) {
			users, err := userClient.GetUsersInfo(ctx, userIDs)
```

```
        if err != nil {
            return nil, err
        }
        return datautil.Slice(users, func(e *sdkws.UserInfo) common_user.CommonUser { return e }), nil
    },
    db:                 db,
    config:             config,
    msgClient:          msgClient,
    conversationClient: conversationClient,
  }
}

type NotificationSender struct {
  *notification.NotificationSender
  getUsersInfo       func(ctx context.Context, userIDs []string) ([]common_user.CommonUser, error)
  db                 controller.GroupDatabase
  config             *Config
  msgClient          *rpcli.MsgClient
  conversationClient *rpcli.ConversationClient
}

func (g *NotificationSender) PopulateGroupMember(ctx context.Context, members ...*model.GroupMember) error {
  if len(members) == 0 {
    return nil
  }
  emptyUserIDs := make(map[string]struct{})
  for _, member := range members {
    if member.Nickname == "" || member.FaceURL == "" {
      emptyUserIDs[member.UserID] = struct{}{}
    }
  }
  if len(emptyUserIDs) > 0 {
    users, err := g.getUsersInfo(ctx, datautil.Keys(emptyUserIDs))
    if err != nil {
      return err
    }
    userMap := make(map[string]common_user.CommonUser)
    for i, user := range users {
      userMap[user.GetUserID()] = users[i]
    }
    for i, member := range members {
      user, ok := userMap[member.UserID]
      if !ok {
        continue
      }
      if member.Nickname == "" {
        members[i].Nickname = user.GetNickname()
      }
      if member.FaceURL == "" {
        members[i].FaceURL = user.GetFaceURL()
      }
    }
  }
  return nil
}

func (g *NotificationSender) getUser(ctx context.Context, userID string) (*sdkws.PublicUserInfo, error) {
  users, err := g.getUsersInfo(ctx, []string{userID})
  if err != nil {
    return nil, err
  }
  if len(users) == 0 {
    return nil, servererrs.ErrUserIDNotFound.WrapMsg(fmt.Sprintf("user %s not found", userID))
  }
  return &sdkws.PublicUserInfo{
    UserID:   users[0].GetUserID(),
```

```
    Nickname: users[0].GetNickname(),
    FaceURL:  users[0].GetFaceURL(),
    Ex:       users[0].GetEx(),
  }, nil
}

func (g *NotificationSender) getGroupInfo(ctx context.Context, groupID string) (*sdkws.GroupInfo, error) {
  gm, err := g.db.TakeGroup(ctx, groupID)
  if err != nil {
    return nil, err
  }
  num, err := g.db.FindGroupMemberNum(ctx, groupID)
  if err != nil {
    return nil, err
  }
  ownerUserIDs, err := g.db.GetGroupRoleLevelMemberIDs(ctx, groupID, constant.GroupOwner)
  if err != nil {
    return nil, err
  }
  var ownerUserID string
  if len(ownerUserIDs) > 0 {
    ownerUserID = ownerUserIDs[0]
  }

  return convert.Db2PbGroupInfo(gm, ownerUserID, num), nil
}

func (g *NotificationSender) getGroupMembers(ctx context.Context, groupID string, userIDs []string) ([]*sdkws.GroupM
  members, err := g.db.FindGroupMembers(ctx, groupID, userIDs)
  if err != nil {
    return nil, err
  }
  if err := g.PopulateGroupMember(ctx, members...); err != nil {
    return nil, err
  }
  log.ZDebug(ctx, "getGroupMembers", "members", members)
  res := make([]*sdkws.GroupMemberFullInfo, 0, len(members))
  for _, member := range members {
    res = append(res, g.groupMemberDB2PB(member, 0))
  }
  return res, nil
}

func (g *NotificationSender) getGroupMemberMap(ctx context.Context, groupID string, userIDs []string) (map[string]*s
  members, err := g.getGroupMembers(ctx, groupID, userIDs)
  if err != nil {
    return nil, err
  }
  m := make(map[string]*sdkws.GroupMemberFullInfo)
  for i, member := range members {
    m[member.UserID] = members[i]
  }
  return m, nil
}

func (g *NotificationSender) getGroupMember(ctx context.Context, groupID string, userID string) (*sdkws.GroupMemberF
  members, err := g.getGroupMembers(ctx, groupID, []string{userID})
  if err != nil {
    return nil, err
  }
  if len(members) == 0 {
    return nil, errs.ErrInternalServer.WrapMsg(fmt.Sprintf("group %s member %s not found", groupID, userID))
  }
  return members[0], nil
}
```

```go
func (g *NotificationSender) getGroupOwnerAndAdminUserID(ctx context.Context, groupID string) ([]string, error) {
	members, err := g.db.FindGroupMemberRoleLevels(ctx, groupID, []int32{constant.GroupOwner, constant.GroupAdmin})
	if err != nil {
		return nil, err
	}
	if err := g.PopulateGroupMember(ctx, members...); err != nil {
		return nil, err
	}
	fn := func(e *model.GroupMember) string { return e.UserID }
	return datautil.Slice(members, fn), nil
}

func (g *NotificationSender) groupMemberDB2PB(member *model.GroupMember, appMangerLevel int32) *sdkws.GroupMemberFul
	return &sdkws.GroupMemberFullInfo{
		GroupID:        member.GroupID,
		UserID:         member.UserID,
		RoleLevel:      member.RoleLevel,
		JoinTime:       member.JoinTime.UnixMilli(),
		Nickname:       member.Nickname,
		FaceURL:        member.FaceURL,
		AppMangerLevel: appMangerLevel,
		JoinSource:     member.JoinSource,
		OperatorUserID: member.OperatorUserID,
		Ex:             member.Ex,
		MuteEndTime:    member.MuteEndTime.UnixMilli(),
		InviterUserID:  member.InviterUserID,
	}
}

/* func (g *NotificationSender) getUsersInfoMap(ctx context.Context, userIDs []string) (map[string]*sdkws.UserInfo,
	users, err := g.getUsersInfo(ctx, userIDs)
	if err != nil {
		return nil, err
	}
	result := make(map[string]*sdkws.UserInfo)
	for _, user := range users {
		result[user.GetUserID()] = user.(*sdkws.UserInfo)
	}
	return result, nil
} */

func (g *NotificationSender) fillOpUser(ctx context.Context, targetUser **sdkws.GroupMemberFullInfo, groupID string)
	return g.fillUserByUserID(ctx, mcontext.GetOpUserID(ctx), targetUser, groupID)
}

func (g *NotificationSender) fillUserByUserID(ctx context.Context, userID string, targetUser **sdkws.GroupMemberFull
	if targetUser == nil {
		return errs.ErrInternalServer.WrapMsg("**sdkws.GroupMemberFullInfo is nil")
	}
	if groupID != "" {
		if authverify.CheckUserIsAdmin(ctx, userID) {
			*targetUser = &sdkws.GroupMemberFullInfo{
				GroupID:        groupID,
				UserID:         userID,
				RoleLevel:      constant.GroupAdmin,
				AppMangerLevel: constant.AppAdmin,
			}
		} else {
			member, err := g.db.TakeGroupMember(ctx, groupID, userID)
			if err == nil {
				*targetUser = g.groupMemberDB2PB(member, 0)
			} else if !(errors.Is(err, mongo.ErrNoDocuments) || errs.ErrRecordNotFound.Is(err)) {
				return err
			}
		}
	}
```

```
	user, err := g.getUser(ctx, userID)
	if err != nil {
		return err
	}
	if *targetUser == nil {
		*targetUser = &sdkws.GroupMemberFullInfo{
			GroupID:        groupID,
			UserID:         userID,
			Nickname:       user.Nickname,
			FaceURL:        user.FaceURL,
			OperatorUserID: userID,
		}
	} else {
		if (*targetUser).Nickname == "" {
			(*targetUser).Nickname = user.Nickname
		}
		if (*targetUser).FaceURL == "" {
			(*targetUser).FaceURL = user.FaceURL
		}
	}
	return nil
}

func (g *NotificationSender) setVersion(ctx context.Context, version *uint64, versionID *string, collName string, id
	versions := versionctx.GetVersionLog(ctx).Get()
	for i := len(versions) - 1; i >= 0; i-- {
		coll := versions[i]
		if coll.Name == collName && coll.Doc.DID == id {
			*version = uint64(coll.Doc.Version)
			*versionID = coll.Doc.ID.Hex()
			return
		}
	}
}

func (g *NotificationSender) setSortVersion(ctx context.Context, version *uint64, versionID *string, collName string
	versions := versionctx.GetVersionLog(ctx).Get()
	for _, coll := range versions {
		if coll.Name == collName && coll.Doc.DID == id {
			*version = uint64(coll.Doc.Version)
			*versionID = coll.Doc.ID.Hex()
			for _, elem := range coll.Doc.Logs {
				if elem.EID == model.VersionSortChangeID {
					*sortVersion = uint64(elem.Version)
				}
			}
		}
	}
}

func (g *NotificationSender) GroupCreatedNotification(ctx context.Context, tips *sdkws.GroupCreatedTips, SendMessage
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), tips.Group.GroupID, constant.GroupCreatedNotification, tips, notific
}

func (g *NotificationSender) GroupInfoSetNotification(ctx context.Context, tips *sdkws.GroupInfoSetTips) {
	var err error
```

```
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), tips.Group.GroupID, constant.GroupInfoSetNotification, tips, notific
}

func (g *NotificationSender) GroupInfoSetNameNotification(ctx context.Context, tips *sdkws.GroupInfoSetNameTips) {
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), tips.Group.GroupID, constant.GroupInfoSetNameNotification, tips)
}

func (g *NotificationSender) GroupInfoSetAnnouncementNotification(ctx context.Context, tips *sdkws.GroupInfoSetAnnou
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), tips.Group.GroupID, constant.GroupInfoSetAnnouncementNotification, t
}

func (g *NotificationSender) uuid() string {
	return uuid.New().String()
}

func (g *NotificationSender) getGroupRequest(ctx context.Context, groupID string, userID string) (*sdkws.GroupReques
	request, err := g.db.TakeGroupRequest(ctx, groupID, userID)
	if err != nil {
		return nil, err
	}
	users, err := g.getUsersInfo(ctx, []string{userID})
	if err != nil {
		return nil, err
	}
	if len(users) == 0 {
		return nil, servererrs.ErrUserIDNotFound.WrapMsg(fmt.Sprintf("user %s not found", userID))
	}
	info, ok := users[0].(*sdkws.UserInfo)
	if !ok {
		info = &sdkws.UserInfo{
			UserID:   users[0].GetUserID(),
			Nickname: users[0].GetNickname(),
			FaceURL:  users[0].GetFaceURL(),
			Ex:       users[0].GetEx(),
		}
	}
	return convert.Db2PbGroupRequest(request, info, nil), nil
```

```
}

func (g *NotificationSender) JoinGroupApplicationNotification(ctx context.Context, req *pbgroup.JoinGroupReq, dbReq
    var err error
    defer func() {
        if err != nil {
            log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
        }
    }()
    request, err := g.getGroupRequest(ctx, dbReq.GroupID, dbReq.UserID)
    if err != nil {
        log.ZError(ctx, "JoinGroupApplicationNotification getGroupRequest", err, "dbReq", dbReq)
        return
    }
    var group *sdkws.GroupInfo
    group, err = g.getGroupInfo(ctx, req.GroupID)
    if err != nil {
        return
    }
    var user *sdkws.PublicUserInfo
    user, err = g.getUser(ctx, req.InviterUserID)
    if err != nil {
        return
    }
    userIDs, err := g.getGroupOwnerAndAdminUserID(ctx, req.GroupID)
    if err != nil {
        return
    }
    userIDs = append(userIDs, req.InviterUserID, mcontext.GetOpUserID(ctx))
    tips := &sdkws.JoinGroupApplicationTips{
        Group:     group,
        Applicant: user,
        ReqMsg:    req.ReqMessage,
        Uuid:      g.uuid(),
        Request:   request,
    }
    for _, userID := range datautil.Distinct(userIDs) {
        g.Notification(ctx, mcontext.GetOpUserID(ctx), userID, constant.JoinGroupApplicationNotification, tips)
    }
}

func (g *NotificationSender) MemberQuitNotification(ctx context.Context, member *sdkws.GroupMemberFullInfo) {
    var err error
    defer func() {
        if err != nil {
            log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
        }
    }()
    var group *sdkws.GroupInfo
    group, err = g.getGroupInfo(ctx, member.GroupID)
    if err != nil {
        return
    }
    tips := &sdkws.MemberQuitTips{Group: group, QuitUser: member}
    g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, member.Gro
    g.Notification(ctx, mcontext.GetOpUserID(ctx), member.GroupID, constant.MemberQuitNotification, tips)
}

func (g *NotificationSender) GroupApplicationAcceptedNotification(ctx context.Context, req *pbgroup.GroupApplication
    var err error
    defer func() {
        if err != nil {
            log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
        }
    }()
    request, err := g.getGroupRequest(ctx, req.GroupID, req.FromUserID)
```

```
	if err != nil {
		log.ZError(ctx, "GroupApplicationAcceptedNotification getGroupRequest", err, "req", req)
		return
	}
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, req.GroupID)
	if err != nil {
		return
	}
	var userIDs []string
	userIDs, err = g.getGroupOwnerAndAdminUserID(ctx, req.GroupID)
	if err != nil {
		return
	}

	var opUser *sdkws.GroupMemberFullInfo
	if err = g.fillOpUser(ctx, &opUser, group.GroupID); err != nil {
		return
	}
	tips := &sdkws.GroupApplicationAcceptedTips{
		Group:     group,
		OpUser:    opUser,
		HandleMsg: req.HandledMsg,
		Uuid:      g.uuid(),
		Request:   request,
	}
	for _, userID := range append(userIDs, req.FromUserID) {
		if userID == req.FromUserID {
			tips.ReceiverAs = applicantReceiver
		} else {
			tips.ReceiverAs = adminReceiver
		}
		g.Notification(ctx, mcontext.GetOpUserID(ctx), userID, constant.GroupApplicationAcceptedNotification, tips)
	}
}

func (g *NotificationSender) GroupApplicationRejectedNotification(ctx context.Context, req *pbgroup.GroupApplication
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	request, err := g.getGroupRequest(ctx, req.GroupID, req.FromUserID)
	if err != nil {
		log.ZError(ctx, "GroupApplicationAcceptedNotification getGroupRequest", err, "req", req)
		return
	}
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, req.GroupID)
	if err != nil {
		return
	}
	var userIDs []string
	userIDs, err = g.getGroupOwnerAndAdminUserID(ctx, req.GroupID)
	if err != nil {
		return
	}

	var opUser *sdkws.GroupMemberFullInfo
	if err = g.fillOpUser(ctx, &opUser, group.GroupID); err != nil {
		return
	}
	tips := &sdkws.GroupApplicationRejectedTips{
		Group:     group,
		OpUser:    opUser,
```

```
    HandleMsg: req.HandledMsg,
    Uuid:      g.uuid(),
    Request:   request,
  }
  for _, userID := range append(userIDs, req.FromUserID) {
    if userID == req.FromUserID {
      tips.ReceiverAs = applicantReceiver
    } else {
      tips.ReceiverAs = adminReceiver
    }
    g.Notification(ctx, mcontext.GetOpUserID(ctx), userID, constant.GroupApplicationRejectedNotification, tips)
  }
}

func (g *NotificationSender) GroupOwnerTransferredNotification(ctx context.Context, req *pbgroup.TransferGroupOwnerR
  var err error
  defer func() {
    if err != nil {
      log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
    }
  }()
  var group *sdkws.GroupInfo
  group, err = g.getGroupInfo(ctx, req.GroupID)
  if err != nil {
    return
  }
  opUserID := mcontext.GetOpUserID(ctx)
  var member map[string]*sdkws.GroupMemberFullInfo
  member, err = g.getGroupMemberMap(ctx, req.GroupID, []string{opUserID, req.NewOwnerUserID, req.OldOwnerUserID})
  if err != nil {
    return
  }
  tips := &sdkws.GroupOwnerTransferredTips{
    Group:             group,
    OpUser:            member[opUserID],
    NewGroupOwner:     member[req.NewOwnerUserID],
    OldGroupOwnerInfo: member[req.OldOwnerUserID],
  }
  if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
    return
  }
  g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, req.GroupI
  g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupOwnerTransferredNotification, tips)
}

func (g *NotificationSender) MemberKickedNotification(ctx context.Context, tips *sdkws.MemberKickedTips, SendMessage
  var err error
  defer func() {
    if err != nil {
      log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
    }
  }()
  if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
    return
  }
  g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
  g.Notification(ctx, mcontext.GetOpUserID(ctx), tips.Group.GroupID, constant.MemberKickedNotification, tips, notific
}

func (g *NotificationSender) GroupApplicationAgreeMemberEnterNotification(ctx context.Context, groupID string, SendM
  return g.groupApplicationAgreeMemberEnterNotification(ctx, groupID, SendMessage, invitedOpUserID, entrantUserID...)
}

func (g *NotificationSender) groupApplicationAgreeMemberEnterNotification(ctx context.Context, groupID string, SendM
  var err error
  defer func() {
```

```
    if err != nil {
        log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
    }
}()

if !g.config.RpcConfig.EnableHistoryForNewMembers {
    conversationID := msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, groupID)
    maxSeq, err := g.msgClient.GetConversationMaxSeq(ctx, conversationID)
    if err != nil {
        return err
    }
    if err := g.msgClient.SetUserConversationsMinSeq(ctx, conversationID, entrantUserID, maxSeq+1); err != nil {
        return err
    }
}
if err := g.conversationClient.CreateGroupChatConversations(ctx, groupID, entrantUserID); err != nil {
    return err
}

var group *sdkws.GroupInfo
group, err = g.getGroupInfo(ctx, groupID)
if err != nil {
    return err
}
users, err := g.getGroupMembers(ctx, groupID, entrantUserID)
if err != nil {
    return err
}

tips := &sdkws.MemberInvitedTips{
    Group:           group,
    InvitedUserList: users,
}
opUserID := mcontext.GetOpUserID(ctx)
if err = g.fillUserByUserID(ctx, opUserID, &tips.OpUser, tips.Group.GroupID); err != nil {
    return nil
}
if invitedOpUserID == opUserID {
    tips.InviterUser = tips.OpUser
} else {
    if err = g.fillUserByUserID(ctx, invitedOpUserID, &tips.InviterUser, tips.Group.GroupID); err != nil {
        return err
    }
}
g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.MemberInvitedNotification, tips, notificatio
return nil
}

func (g *NotificationSender) MemberEnterNotification(ctx context.Context, groupID string, entrantUserID string) erro
var err error
defer func() {
    if err != nil {
        log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
    }
}()

if !g.config.RpcConfig.EnableHistoryForNewMembers {
    conversationID := msgprocessor.GetConversationIDBySessionType(constant.ReadGroupChatType, groupID)
    maxSeq, err := g.msgClient.GetConversationMaxSeq(ctx, conversationID)
    if err != nil {
        return err
    }
    if err := g.msgClient.SetUserConversationsMinSeq(ctx, conversationID, []string{entrantUserID}, maxSeq+1); err !=
        return err
    }
```

```
	}
	if err := g.conversationClient.CreateGroupChatConversations(ctx, groupID, []string{entrantUserID}); err != nil {
		return err
	}
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return err
	}
	user, err := g.getGroupMember(ctx, groupID, entrantUserID)
	if err != nil {
		return err
	}

	tips := &sdkws.MemberEnterTips{
		Group:         group,
		EntrantUser:   user,
		OperationTime: time.Now().UnixMilli(),
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.MemberEnterNotification, tips)
	return nil
}

func (g *NotificationSender) GroupDismissedNotification(ctx context.Context, tips *sdkws.GroupDismissedTips, SendMes
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.Notification(ctx, mcontext.GetOpUserID(ctx), tips.Group.GroupID, constant.GroupDismissedNotification, tips, notif
}

func (g *NotificationSender) GroupMemberMutedNotification(ctx context.Context, groupID, groupMemberUserID string, mu
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return
	}
	var user map[string]*sdkws.GroupMemberFullInfo
	user, err = g.getGroupMemberMap(ctx, groupID, []string{mcontext.GetOpUserID(ctx), groupMemberUserID})
	if err != nil {
		return
	}
	tips := &sdkws.GroupMemberMutedTips{
		Group: group, MutedSeconds: mutedSeconds,
		OpUser: user[mcontext.GetOpUserID(ctx)], MutedUser: user[groupMemberUserID],
	}
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupMemberMutedNotification, tips)
}

func (g *NotificationSender) GroupMemberCancelMutedNotification(ctx context.Context, groupID, groupMemberUserID stri
```

```
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return
	}
	var user map[string]*sdkws.GroupMemberFullInfo
	user, err = g.getGroupMemberMap(ctx, groupID, []string{mcontext.GetOpUserID(ctx), groupMemberUserID})
	if err != nil {
		return
	}
	tips := &sdkws.GroupMemberCancelMutedTips{Group: group, OpUser: user[mcontext.GetOpUserID(ctx)], MutedUser: user[gr
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.Group
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupMemberCancelMutedNotification, tips)
}

func (g *NotificationSender) GroupMutedNotification(ctx context.Context, groupID string) {
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return
	}
	var users []*sdkws.GroupMemberFullInfo
	users, err = g.getGroupMembers(ctx, groupID, []string{mcontext.GetOpUserID(ctx)})
	if err != nil {
		return
	}
	tips := &sdkws.GroupMutedTips{Group: group}
	if len(users) > 0 {
		tips.OpUser = users[0]
	}
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, groupID)
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupMutedNotification, tips)
}

func (g *NotificationSender) GroupCancelMutedNotification(ctx context.Context, groupID string) {
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return
	}
	var users []*sdkws.GroupMemberFullInfo
	users, err = g.getGroupMembers(ctx, groupID, []string{mcontext.GetOpUserID(ctx)})
```

```
	if err != nil {
		return
	}
	tips := &sdkws.GroupCancelMutedTips{Group: group}
	if len(users) > 0 {
		tips.OpUser = users[0]
	}
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, groupID)
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupCancelMutedNotification, tips)
}

func (g *NotificationSender) GroupMemberInfoSetNotification(ctx context.Context, groupID, groupMemberUserID string)
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return
	}
	var user map[string]*sdkws.GroupMemberFullInfo
	user, err = g.getGroupMemberMap(ctx, groupID, []string{groupMemberUserID})
	if err != nil {
		return
	}
	tips := &sdkws.GroupMemberInfoSetTips{Group: group, OpUser: user[mcontext.GetOpUserID(ctx)], ChangedUser: user[grou
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setSortVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.C
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupMemberInfoSetNotification, tips)
}

func (g *NotificationSender) GroupMemberSetToAdminNotification(ctx context.Context, groupID, groupMemberUserID strin
	var err error
	defer func() {
		if err != nil {
			log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
		}
	}()
	var group *sdkws.GroupInfo
	group, err = g.getGroupInfo(ctx, groupID)
	if err != nil {
		return
	}
	user, err := g.getGroupMemberMap(ctx, groupID, []string{mcontext.GetOpUserID(ctx), groupMemberUserID})
	if err != nil {
		return
	}
	tips := &sdkws.GroupMemberInfoSetTips{Group: group, OpUser: user[mcontext.GetOpUserID(ctx)], ChangedUser: user[grou
	if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
		return
	}
	g.setSortVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.C
	g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupMemberSetToAdminNotification, tips)
}

func (g *NotificationSender) GroupMemberSetToOrdinaryUserNotification(ctx context.Context, groupID, groupMemberUserI
	var err error
	defer func() {
```

```
    if err != nil {
        log.ZError(ctx, stringutil.GetFuncName(1)+" failed", err)
    }
}()
var group *sdkws.GroupInfo
group, err = g.getGroupInfo(ctx, groupID)
if err != nil {
    return
}
var user map[string]*sdkws.GroupMemberFullInfo
user, err = g.getGroupMemberMap(ctx, groupID, []string{mcontext.GetOpUserID(ctx), groupMemberUserID})
if err != nil {
    return
}
tips := &sdkws.GroupMemberInfoSetTips{Group: group, OpUser: user[mcontext.GetOpUserID(ctx)], ChangedUser: user[grou
if err = g.fillOpUser(ctx, &tips.OpUser, tips.Group.GroupID); err != nil {
    return
}
g.setSortVersion(ctx, &tips.GroupMemberVersion, &tips.GroupMemberVersionID, database.GroupMemberVersionName, tips.G
g.Notification(ctx, mcontext.GetOpUserID(ctx), group.GroupID, constant.GroupMemberSetToOrdinaryUserNotification, ti
}
```

internal/rpc/group/statistics.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package group

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/tools/errs"
)

func (g *groupServer) GroupCreateCount(ctx context.Context, req *group.GroupCreateCountReq) (*group.GroupCreateCount
	if req.Start > req.End {
		return nil, errs.ErrArgs.WrapMsg("start > end: %d > %d", req.Start, req.End)
	}
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	total, err := g.db.CountTotal(ctx, nil)
	if err != nil {
		return nil, err
	}
	start := time.UnixMilli(req.Start)
	before, err := g.db.CountTotal(ctx, &start)
	if err != nil {
		return nil, err
	}
	count, err := g.db.CountRangeEverydayTotal(ctx, start, time.UnixMilli(req.End))
	if err != nil {
		return nil, err
	}
	return &group.GroupCreateCountResp{Total: total, Before: before, Count: count}, nil
}
```

internal/rpc/group/sync.go

```
package group

import (
	"context"
	"errors"

	"github.com/openimsdk/open-im-server/v3/internal/rpc/incrversion"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/util/hashutil"
	"github.com/openimsdk/protocol/constant"
	pbgroup "github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/log"
)

const versionSyncLimit = 500

func (g *groupServer) GetFullGroupMemberUserIDs(ctx context.Context, req *pbgroup.GetFullGroupMemberUserIDsReq) (*pb
	userIDs, err := g.db.FindGroupMemberUserID(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	if err := authverify.CheckAccessIn(ctx, userIDs...); err != nil {
		return nil, err
	}
	vl, err := g.db.FindMaxGroupMemberVersionCache(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	idHash := hashutil.IdHash(userIDs)
	if req.IdHash == idHash {
		userIDs = nil
	}
	return &pbgroup.GetFullGroupMemberUserIDsResp{
		Version:   uint64(vl.Version),
		VersionID: vl.ID.Hex(),
		Equal:     req.IdHash == idHash,
		UserIDs:   userIDs,
	}, nil
}

func (g *groupServer) GetFullJoinGroupIDs(ctx context.Context, req *pbgroup.GetFullJoinGroupIDsReq) (*pbgroup.GetFul
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	vl, err := g.db.FindMaxJoinGroupVersionCache(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	groupIDs, err := g.db.FindJoinGroupID(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	idHash := hashutil.IdHash(groupIDs)
	if req.IdHash == idHash {
		groupIDs = nil
	}
	return &pbgroup.GetFullJoinGroupIDsResp{
		Version:   uint64(vl.Version),
		VersionID: vl.ID.Hex(),
		Equal:     req.IdHash == idHash,
		GroupIDs:  groupIDs,
```

```
	}, nil
}

func (g *groupServer) GetIncrementalGroupMember(ctx context.Context, req *pbgroup.GetIncrementalGroupMemberReq) (*pb
	if err := g.checkAdminOrInGroup(ctx, req.GroupID); err != nil {
		return nil, err
	}
	group, err := g.db.TakeGroup(ctx, req.GroupID)
	if err != nil {
		return nil, err
	}
	if group.Status == constant.GroupStatusDismissed {
		return nil, servererrs.ErrDismissedAlready.Wrap()
	}
	var (
		hasGroupUpdate bool
		sortVersion    uint64
	)
	opt := incrversion.Option[*sdkws.GroupMemberFullInfo, pbgroup.GetIncrementalGroupMemberResp]{
		Ctx:           ctx,
		VersionKey:    req.GroupID,
		VersionID:     req.VersionID,
		VersionNumber: req.Version,
		Version: func(ctx context.Context, groupID string, version uint, limit int) (*model.VersionLog, error) {
			vl, err := g.db.FindMemberIncrVersion(ctx, groupID, version, limit)
			if err != nil {
				return nil, err
			}
			logs := make([]model.VersionLogElem, 0, len(vl.Logs))
			for i, log := range vl.Logs {
				switch log.EID {
				case model.VersionGroupChangeID:
					vl.LogLen--
					hasGroupUpdate = true
				case model.VersionSortChangeID:
					vl.LogLen--
					sortVersion = uint64(log.Version)
				default:
					logs = append(logs, vl.Logs[i])
				}
			}
			vl.Logs = logs
			if vl.LogLen > 0 {
				hasGroupUpdate = true
			}
			return vl, nil
		},
		CacheMaxVersion: g.db.FindMaxGroupMemberVersionCache,
		Find: func(ctx context.Context, ids []string) ([]*sdkws.GroupMemberFullInfo, error) {
			return g.getGroupMembersInfo(ctx, req.GroupID, ids)
		},
		Resp: func(version *model.VersionLog, delIDs []string, insertList, updateList []*sdkws.GroupMemberFullInfo, full
			return &pbgroup.GetIncrementalGroupMemberResp{
				VersionID:   version.ID.Hex(),
				Version:     uint64(version.Version),
				Full:        full,
				Delete:      delIDs,
				Insert:      insertList,
				Update:      updateList,
				SortVersion: sortVersion,
			}
		},
	}
	resp, err := opt.Build()
	if err != nil {
		return nil, err
```

```
	}
	if resp.Full || hasGroupUpdate {
		count, err := g.db.FindGroupMemberNum(ctx, group.GroupID)
		if err != nil {
			return nil, err
		}
		owner, err := g.db.TakeGroupOwner(ctx, group.GroupID)
		if err != nil {
			return nil, err
		}
		resp.Group = g.groupDB2PB(group, owner.UserID, count)
	}
	return resp, nil
}

func (g *groupServer) GetIncrementalJoinGroup(ctx context.Context, req *pbgroup.GetIncrementalJoinGroupReq) (*pbgrou
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	opt := incrversion.Option[*sdkws.GroupInfo, pbgroup.GetIncrementalJoinGroupResp]{
		Ctx:             ctx,
		VersionKey:      req.UserID,
		VersionID:       req.VersionID,
		VersionNumber:   req.Version,
		Version:         g.db.FindJoinIncrVersion,
		CacheMaxVersion: g.db.FindMaxJoinGroupVersionCache,
		Find:            g.getGroupsInfo,
		Resp: func(version *model.VersionLog, delIDs []string, insertList, updateList []*sdkws.GroupInfo, full bool) *pbg
			return &pbgroup.GetIncrementalJoinGroupResp{
				VersionID: version.ID.Hex(),
				Version:   uint64(version.Version),
				Full:      full,
				Delete:    delIDs,
				Insert:    insertList,
				Update:    updateList,
			}
		},
	}
	return opt.Build()
}

func (g *groupServer) BatchGetIncrementalGroupMember(ctx context.Context, req *pbgroup.BatchGetIncrementalGroupMembe
	var num int
	resp := make(map[string]*pbgroup.GetIncrementalGroupMemberResp)

	for _, memberReq := range req.ReqList {
		if _, ok := resp[memberReq.GroupID]; ok {
			continue
		}
		memberResp, err := g.GetIncrementalGroupMember(ctx, memberReq)
		if err != nil {
			if errors.Is(err, servererrs.ErrDismissedAlready) {
				log.ZWarn(ctx, "Failed to get incremental group member", err, "groupID", memberReq.GroupID, "request", memberRe
				continue
			}
			return nil, err
		}

		resp[memberReq.GroupID] = memberResp
		num += len(memberResp.Insert) + len(memberResp.Update) + len(memberResp.Delete)
		if num >= versionSyncLimit {
			break
		}
	}

	return &pbgroup.BatchGetIncrementalGroupMemberResp{RespList: resp}, nil
```

```
}
```

internal/rpc/relation

internal/rpc/relation/black.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package relation

import (
	"context"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/relation"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
)

func (s *friendServer) GetPaginationBlacks(ctx context.Context, req *relation.GetPaginationBlacksReq) (resp *relatio
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	total, blacks, err := s.blackDatabase.FindOwnerBlacks(ctx, req.UserID, req.Pagination)
	if err != nil {
		return nil, err
	}
	resp = &relation.GetPaginationBlacksResp{}
	resp.Blacks, err = convert.BlackDB2Pb(ctx, blacks, s.userClient.GetUsersInfoMap)
	if err != nil {
		return nil, err
	}
	resp.Total = int32(total)
	return resp, nil
}

func (s *friendServer) IsBlack(ctx context.Context, req *relation.IsBlackReq) (*relation.IsBlackResp, error) {
	if err := authverify.CheckAccessIn(ctx, req.UserID1, req.UserID2); err != nil {
		return nil, err
	}
	in1, in2, err := s.blackDatabase.CheckIn(ctx, req.UserID1, req.UserID2)
	if err != nil {
		return nil, err
	}
	resp := &relation.IsBlackResp{}
	resp.InUser1Blacks = in1
	resp.InUser2Blacks = in2
	return resp, nil
}
```

```
func (s *friendServer) RemoveBlack(ctx context.Context, req *relation.RemoveBlackReq) (*relation.RemoveBlackResp, er
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}

	if err := s.blackDatabase.Delete(ctx, []*model.Black{{OwnerUserID: req.OwnerUserID, BlockUserID: req.BlackUserID}})
		return nil, err
	}

	s.notificationSender.BlackDeletedNotification(ctx, req)
	s.webhookAfterRemoveBlack(ctx, &s.config.WebhooksConfig.AfterRemoveBlack, req)

	return &relation.RemoveBlackResp{}, nil
}

func (s *friendServer) AddBlack(ctx context.Context, req *relation.AddBlackReq) (*relation.AddBlackResp, error) {
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}

	if err := s.webhookBeforeAddBlack(ctx, &s.config.WebhooksConfig.BeforeAddBlack, req); err != nil {
		return nil, err
	}
	if err := s.userClient.CheckUser(ctx, []string{req.OwnerUserID, req.BlackUserID}); err != nil {
		return nil, err
	}
	black := model.Black{
		OwnerUserID:    req.OwnerUserID,
		BlockUserID:    req.BlackUserID,
		OperatorUserID: mcontext.GetOpUserID(ctx),
		CreateTime:     time.Now(),
		Ex:             req.Ex,
	}

	if err := s.blackDatabase.Create(ctx, []*model.Black{&black}); err != nil {
		return nil, err
	}
	s.notificationSender.BlackAddedNotification(ctx, req)
	return &relation.AddBlackResp{}, nil
}

func (s *friendServer) GetSpecifiedBlacks(ctx context.Context, req *relation.GetSpecifiedBlacksReq) (*relation.GetSp
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}

	if len(req.UserIDList) == 0 {
		return nil, errs.ErrArgs.WrapMsg("userIDList is empty")
	}

	if datautil.Duplicate(req.UserIDList) {
		return nil, errs.ErrArgs.WrapMsg("userIDList repeated")
	}

	userMap, err := s.userClient.GetUsersInfoMap(ctx, req.UserIDList)
	if err != nil {
		return nil, err
	}

	blacks, err := s.blackDatabase.FindBlackInfos(ctx, req.OwnerUserID, req.UserIDList)
	if err != nil {
		return nil, err
	}

	blackMap := datautil.SliceToMap(blacks, func(e *model.Black) string {
		return e.BlockUserID
```

```
	})

	resp := &relation.GetSpecifiedBlacksResp{
		Blacks: make([]*sdkws.BlackInfo, 0, len(req.UserIDList)),
	}

	toPublcUser := func(userID string) *sdkws.PublicUserInfo {
		v, ok := userMap[userID]
		if !ok {
			return nil
		}
		return &sdkws.PublicUserInfo{
			UserID:   v.UserID,
			Nickname: v.Nickname,
			FaceURL:  v.FaceURL,
			Ex:       v.Ex,
		}
	}

	for _, userID := range req.UserIDList {
		if black := blackMap[userID]; black != nil {
			resp.Blacks = append(resp.Blacks,
				&sdkws.BlackInfo{
					OwnerUserID:    black.OwnerUserID,
					CreateTime:     black.CreateTime.UnixMilli(),
					BlackUserInfo:  toPublcUser(userID),
					AddSource:      black.AddSource,
					OperatorUserID: black.OperatorUserID,
					Ex:             black.Ex,
				})
		}
	}

	resp.Total = int32(len(resp.Blacks))

	return resp, nil
}
```

internal/rpc/relation/callback.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package relation

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"

    cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/protocol/relation"
)

func (s *friendServer) webhookAfterDeleteFriend(ctx context.Context, after *config.AfterConfig, req *relation.Delete
    cbReq := &cbapi.CallbackAfterDeleteFriendReq{
        CallbackCommand: cbapi.CallbackAfterDeleteFriendCommand,
        OwnerUserID:     req.OwnerUserID,
        FriendUserID:    req.FriendUserID,
    }
    s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterDeleteFriendResp{}, after)
}

func (s *friendServer) webhookBeforeAddFriend(ctx context.Context, before *config.BeforeConfig, req *relation.ApplyT
    return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
        cbReq := &cbapi.CallbackBeforeAddFriendReq{
            CallbackCommand: cbapi.CallbackBeforeAddFriendCommand,
            FromUserID:      req.FromUserID,
            ToUserID:        req.ToUserID,
            ReqMsg:          req.ReqMsg,
            Ex:              req.Ex,
        }
        resp := &cbapi.CallbackBeforeAddFriendResp{}

        if err := s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
            return err
        }
        return nil
    })
}

func (s *friendServer) webhookAfterAddFriend(ctx context.Context, after *config.AfterConfig, req *relation.ApplyToAd
    cbReq := &cbapi.CallbackAfterAddFriendReq{
        CallbackCommand: cbapi.CallbackAfterAddFriendCommand,
        FromUserID:      req.FromUserID,
        ToUserID:        req.ToUserID,
        ReqMsg:          req.ReqMsg,
    }
    resp := &cbapi.CallbackAfterAddFriendResp{}
    s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, after)
}
```

```
func (s *friendServer) webhookAfterSetFriendRemark(ctx context.Context, after *config.AfterConfig, req *relation.Set
	cbReq := &cbapi.CallbackAfterSetFriendRemarkReq{
		CallbackCommand: cbapi.CallbackAfterSetFriendRemarkCommand,
		OwnerUserID:     req.OwnerUserID,
		FriendUserID:    req.FriendUserID,
		Remark:          req.Remark,
	}
	resp := &cbapi.CallbackAfterSetFriendRemarkResp{}
	s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, after)
}

func (s *friendServer) webhookAfterImportFriends(ctx context.Context, after *config.AfterConfig, req *relation.Impor
	cbReq := &cbapi.CallbackAfterImportFriendsReq{
		CallbackCommand: cbapi.CallbackAfterImportFriendsCommand,
		OwnerUserID:     req.OwnerUserID,
		FriendUserIDs:   req.FriendUserIDs,
	}
	resp := &cbapi.CallbackAfterImportFriendsResp{}
	s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, after)
}

func (s *friendServer) webhookAfterRemoveBlack(ctx context.Context, after *config.AfterConfig, req *relation.RemoveB
	cbReq := &cbapi.CallbackAfterRemoveBlackReq{
		CallbackCommand: cbapi.CallbackAfterRemoveBlackCommand,
		OwnerUserID:     req.OwnerUserID,
		BlackUserID:     req.BlackUserID,
	}
	resp := &cbapi.CallbackAfterRemoveBlackResp{}
	s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, after)
}

func (s *friendServer) webhookBeforeSetFriendRemark(ctx context.Context, before *config.BeforeConfig, req *relation.
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &cbapi.CallbackBeforeSetFriendRemarkReq{
			CallbackCommand: cbapi.CallbackBeforeSetFriendRemarkCommand,
			OwnerUserID:     req.OwnerUserID,
			FriendUserID:    req.FriendUserID,
			Remark:          req.Remark,
		}
		resp := &cbapi.CallbackBeforeSetFriendRemarkResp{}
		if err := s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
			return err
		}
		if resp.Remark != "" {
			req.Remark = resp.Remark
		}
		return nil
	})
}

func (s *friendServer) webhookBeforeAddBlack(ctx context.Context, before *config.BeforeConfig, req *relation.AddBlac
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &cbapi.CallbackBeforeAddBlackReq{
			CallbackCommand: cbapi.CallbackBeforeAddBlackCommand,
			OwnerUserID:     req.OwnerUserID,
			BlackUserID:     req.BlackUserID,
		}
		resp := &cbapi.CallbackBeforeAddBlackResp{}
		return s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before)
	})
}

func (s *friendServer) webhookBeforeAddFriendAgree(ctx context.Context, before *config.BeforeConfig, req *relation.R
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		cbReq := &cbapi.CallbackBeforeAddFriendAgreeReq{
			CallbackCommand: cbapi.CallbackBeforeAddFriendAgreeCommand,
```

```
        FromUserID:      req.FromUserID,
        ToUserID:        req.ToUserID,
        HandleMsg:       req.HandleMsg,
        HandleResult:    req.HandleResult,
    }
    resp := &cbapi.CallbackBeforeAddFriendAgreeResp{}
    return s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before)
  })
}

func (s *friendServer) webhookAfterAddFriendAgree(ctx context.Context, after *config.AfterConfig, req *relation.Resp
  cbReq := &cbapi.CallbackAfterAddFriendAgreeReq{
    CallbackCommand: cbapi.CallbackAfterAddFriendAgreeCommand,
    FromUserID:      req.FromUserID,
    ToUserID:        req.ToUserID,
    HandleMsg:       req.HandleMsg,
    HandleResult:    req.HandleResult,
  }
  resp := &cbapi.CallbackAfterAddFriendAgreeResp{}
  s.webhookClient.AsyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, after)
}

func (s *friendServer) webhookBeforeImportFriends(ctx context.Context, before *config.BeforeConfig, req *relation.Im
  return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
    cbReq := &cbapi.CallbackBeforeImportFriendsReq{
        CallbackCommand: cbapi.CallbackBeforeImportFriendsCommand,
        OwnerUserID:     req.OwnerUserID,
        FriendUserIDs:   req.FriendUserIDs,
    }
    resp := &cbapi.CallbackBeforeImportFriendsResp{}
    if err := s.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
        return err
    }
    if len(resp.FriendUserIDs) > 0 {
        req.FriendUserIDs = resp.FriendUserIDs
    }
    return nil
  })
}
```

internal/rpc/relation/friend.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package relation

import (
	"context"

	"github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
	"github.com/openimsdk/open-im-server/v3/pkg/notification/common_user"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"

	"github.com/openimsdk/tools/mq/memamq"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/convert"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/open-im-server/v3/pkg/localcache"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/relation"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/utils/datautil"
	"google.golang.org/grpc"
)

type friendServer struct {
	relation.UnimplementedFriendServer
	db                 controller.FriendDatabase
	blackDatabase      controller.BlackDatabase
	notificationSender *FriendNotificationSender
	RegisterCenter     discovery.Conn
	config             *Config
	webhookClient      *webhook.Client
	queue              *memamq.MemoryQueue
	userClient         *rpcli.UserClient
}

type Config struct {
	RpcConfig     config.Friend
	RedisConfig   config.Redis
	MongodbConfig config.Mongo
	// ZookeeperConfig    config.ZooKeeper
	NotificationConfig config.Notification
	Share              config.Share
	WebhooksConfig     config.Webhooks
```

```
    LocalCacheConfig   config.LocalCache
    Discovery          config.Discovery
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
    dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
    mgocli, err := dbb.Mongo(ctx)
    if err != nil {
        return err
    }
    rdb, err := dbb.Redis(ctx)
    if err != nil {
        return err
    }

    friendMongoDB, err := mgo.NewFriendMongo(mgocli.GetDB())
    if err != nil {
        return err
    }

    friendRequestMongoDB, err := mgo.NewFriendRequestMongo(mgocli.GetDB())
    if err != nil {
        return err
    }

    blackMongoDB, err := mgo.NewBlackMongo(mgocli.GetDB())
    if err != nil {
        return err
    }

    userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
    if err != nil {
        return err
    }
    msgConn, err := client.GetConn(ctx, config.Discovery.RpcService.Msg)
    if err != nil {
        return err
    }
    userClient := rpcli.NewUserClient(userConn)
    database := controller.NewFriendDatabase(
        friendMongoDB,
        friendRequestMongoDB,
        redis.NewFriendCacheRedis(rdb, &config.LocalCacheConfig, friendMongoDB),
        mgocli.GetTx(),
    )
    // Initialize notification sender
    notificationSender := NewFriendNotificationSender(
        &config.NotificationConfig,
        rpcli.NewMsgClient(msgConn),
        WithRpcFunc(userClient.GetUsersInfo),
        WithFriendDB(database),
    )
    localcache.InitLocalCache(&config.LocalCacheConfig)

    // Register Friend server with refactored MongoDB and Redis integrations
    relation.RegisterFriendServer(server, &friendServer{
        db: database,
        blackDatabase: controller.NewBlackDatabase(
            blackMongoDB,
            redis.NewBlackCacheRedis(rdb, &config.LocalCacheConfig, blackMongoDB),
        ),
        notificationSender: notificationSender,
        RegisterCenter:     client,
        config:             config,
        webhookClient:      webhook.NewWebhookClient(config.WebhooksConfig.URL),
        queue:              memamq.NewMemoryQueue(16, 1024*1024),
```

```
    userClient:         userClient,
  })
  return nil
}

// ok.
func (s *friendServer) ApplyToAddFriend(ctx context.Context, req *relation.ApplyToAddFriendReq) (resp *relation.Appl
  resp = &relation.ApplyToAddFriendResp{}
  if err := authverify.CheckAccess(ctx, req.FromUserID); err != nil {
    return nil, err
  }
  if req.ToUserID == req.FromUserID {
    return nil, servererrs.ErrCanNotAddYourself.WrapMsg("req.ToUserID", req.ToUserID)
  }
  if err = s.webhookBeforeAddFriend(ctx, &s.config.WebhooksConfig.BeforeAddFriend, req); err != nil && err != servere
    return nil, err
  }
  if err := s.userClient.CheckUser(ctx, []string{req.ToUserID, req.FromUserID}); err != nil {
    return nil, err
  }

  in1, in2, err := s.db.CheckIn(ctx, req.FromUserID, req.ToUserID)
  if err != nil {
    return nil, err
  }
  if in1 && in2 {
    return nil, servererrs.ErrRelationshipAlready.WrapMsg("already friends has f")
  }
  if err = s.db.AddFriendRequest(ctx, req.FromUserID, req.ToUserID, req.ReqMsg, req.Ex); err != nil {
    return nil, err
  }
  s.notificationSender.FriendApplicationAddNotification(ctx, req)
  s.webhookAfterAddFriend(ctx, &s.config.WebhooksConfig.AfterAddFriend, req)
  return resp, nil
}

// ok.
func (s *friendServer) ImportFriends(ctx context.Context, req *relation.ImportFriendReq) (resp *relation.ImportFrien
  if err := authverify.CheckAdmin(ctx); err != nil {
    return nil, err
  }

  if err := s.userClient.CheckUser(ctx, append([]string{req.OwnerUserID}, req.FriendUserIDs...)); err != nil {
    return nil, err
  }
  if datautil.Contain(req.OwnerUserID, req.FriendUserIDs...) {
    return nil, servererrs.ErrCanNotAddYourself.WrapMsg("can not add yourself")
  }
  if datautil.Duplicate(req.FriendUserIDs) {
    return nil, errs.ErrArgs.WrapMsg("friend userID repeated")
  }

  if err := s.webhookBeforeImportFriends(ctx, &s.config.WebhooksConfig.BeforeImportFriends, req); err != nil && err !
    return nil, err
  }

  if err := s.db.BecomeFriends(ctx, req.OwnerUserID, req.FriendUserIDs, constant.BecomeFriendByImport); err != nil {
    return nil, err
  }
  for _, userID := range req.FriendUserIDs {
    s.notificationSender.FriendApplicationAgreedNotification(ctx, &relation.RespondFriendApplyReq{
      FromUserID:   req.OwnerUserID,
      ToUserID:     userID,
      HandleResult: constant.FriendResponseAgree,
    }, false)
  }
```

```
	s.webhookAfterImportFriends(ctx, &s.config.WebhooksConfig.AfterImportFriends, req)
	return &relation.ImportFriendResp{}, nil
}

// ok.
func (s *friendServer) RespondFriendApply(ctx context.Context, req *relation.RespondFriendApplyReq) (resp *relation.
	resp = &relation.RespondFriendApplyResp{}
	if err := authverify.CheckAccess(ctx, req.ToUserID); err != nil {
		return nil, err
	}

	friendRequest := model.FriendRequest{
		FromUserID:   req.FromUserID,
		ToUserID:     req.ToUserID,
		HandleMsg:    req.HandleMsg,
		HandleResult: req.HandleResult,
	}
	if req.HandleResult == constant.FriendResponseAgree {
		if err := s.webhookBeforeAddFriendAgree(ctx, &s.config.WebhooksConfig.BeforeAddFriendAgree, req); err != nil && e
			return nil, err
		}
		err := s.db.AgreeFriendRequest(ctx, &friendRequest)
		if err != nil {
			return nil, err
		}
		s.webhookAfterAddFriendAgree(ctx, &s.config.WebhooksConfig.AfterAddFriendAgree, req)
		s.notificationSender.FriendApplicationAgreedNotification(ctx, req, true)
		return resp, nil
	}
	if req.HandleResult == constant.FriendResponseRefuse {
		err := s.db.RefuseFriendRequest(ctx, &friendRequest)
		if err != nil {
			return nil, err
		}
		s.notificationSender.FriendApplicationRefusedNotification(ctx, req)
		return resp, nil
	}
	return nil, errs.ErrArgs.WrapMsg("req.HandleResult != -1/1")
}

// ok.
func (s *friendServer) DeleteFriend(ctx context.Context, req *relation.DeleteFriendReq) (resp *relation.DeleteFriend
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}

	_, err = s.db.FindFriendsWithError(ctx, req.OwnerUserID, []string{req.FriendUserID})
	if err != nil {
		return nil, err
	}

	if err := s.db.Delete(ctx, req.OwnerUserID, []string{req.FriendUserID}); err != nil {
		return nil, err
	}

	s.notificationSender.FriendDeletedNotification(ctx, req)
	s.webhookAfterDeleteFriend(ctx, &s.config.WebhooksConfig.AfterDeleteFriend, req)

	return &relation.DeleteFriendResp{}, nil
}

// ok.
func (s *friendServer) SetFriendRemark(ctx context.Context, req *relation.SetFriendRemarkReq) (resp *relation.SetFri
	if err = s.webhookBeforeSetFriendRemark(ctx, &s.config.WebhooksConfig.BeforeSetFriendRemark, req); err != nil && er
		return nil, err
```

```
	}

	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}

	_, err = s.db.FindFriendsWithError(ctx, req.OwnerUserID, []string{req.FriendUserID})
	if err != nil {
		return nil, err
	}

	if err := s.db.UpdateRemark(ctx, req.OwnerUserID, req.FriendUserID, req.Remark); err != nil {
		return nil, err
	}

	s.webhookAfterSetFriendRemark(ctx, &s.config.WebhooksConfig.AfterSetFriendRemark, req)
	s.notificationSender.FriendRemarkSetNotification(ctx, req.OwnerUserID, req.FriendUserID)

	return &relation.SetFriendRemarkResp{}, nil
}

func (s *friendServer) GetFriendInfo(ctx context.Context, req *relation.GetFriendInfoReq) (*relation.GetFriendInfoRe
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}
	friends, err := s.db.FindFriendsWithError(ctx, req.OwnerUserID, req.FriendUserIDs)
	if err != nil {
		return nil, err
	}
	return &relation.GetFriendInfoResp{FriendInfos: convert.FriendOnlyDB2PbOnly(friends)}, nil
}

func (s *friendServer) GetDesignatedFriends(ctx context.Context, req *relation.GetDesignatedFriendsReq) (resp *relat
	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}
	resp = &relation.GetDesignatedFriendsResp{}
	if datautil.Duplicate(req.FriendUserIDs) {
		return nil, errs.ErrArgs.WrapMsg("friend userID repeated")
	}
	friends, err := s.getFriend(ctx, req.OwnerUserID, req.FriendUserIDs)
	if err != nil {
		return nil, err
	}
	return &relation.GetDesignatedFriendsResp{
		FriendsInfo: friends,
	}, nil
}

func (s *friendServer) getFriend(ctx context.Context, ownerUserID string, friendUserIDs []string) ([]*sdkws.FriendIn
	if len(friendUserIDs) == 0 {
		return nil, nil
	}
	friends, err := s.db.FindFriendsWithError(ctx, ownerUserID, friendUserIDs)
	if err != nil {
		return nil, err
	}
	return convert.FriendsDB2Pb(ctx, friends, s.userClient.GetUsersInfoMap)
}

// Get the list of friend requests sent out proactively.
func (s *friendServer) GetDesignatedFriendsApply(ctx context.Context, req *relation.GetDesignatedFriendsApplyReq) (r
	if err := authverify.CheckAccessIn(ctx, req.FromUserID, req.ToUserID); err != nil {
		return nil, err
	}
	friendRequests, err := s.db.FindBothFriendRequests(ctx, req.FromUserID, req.ToUserID)
```

```
	if err != nil {
		return nil, err
	}
	resp = &relation.GetDesignatedFriendsApplyResp{}
	resp.FriendRequests, err = convert.FriendRequestDB2Pb(ctx, friendRequests, s.getCommonUserMap)
	if err != nil {
		return nil, err
	}
	return resp, nil
}

// Get received friend requests (i.e., those initiated by others).
func (s *friendServer) GetPaginationFriendsApplyTo(ctx context.Context, req *relation.GetPaginationFriendsApplyToReq
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}

	handleResults := datautil.Slice(req.HandleResults, func(e int32) int {
		return int(e)
	})
	total, friendRequests, err := s.db.PageFriendRequestToMe(ctx, req.UserID, handleResults, req.Pagination)
	if err != nil {
		return nil, err
	}

	resp = &relation.GetPaginationFriendsApplyToResp{}
	resp.FriendRequests, err = convert.FriendRequestDB2Pb(ctx, friendRequests, s.getCommonUserMap)
	if err != nil {
		return nil, err
	}

	resp.Total = int32(total)

	return resp, nil
}

func (s *friendServer) GetPaginationFriendsApplyFrom(ctx context.Context, req *relation.GetPaginationFriendsApplyFro
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}

	handleResults := datautil.Slice(req.HandleResults, func(e int32) int {
		return int(e)
	})
	total, friendRequests, err := s.db.PageFriendRequestFromMe(ctx, req.UserID, handleResults, req.Pagination)
	if err != nil {
		return nil, err
	}

	resp = &relation.GetPaginationFriendsApplyFromResp{}
	resp.FriendRequests, err = convert.FriendRequestDB2Pb(ctx, friendRequests, s.getCommonUserMap)
	if err != nil {
		return nil, err
	}

	resp.Total = int32(total)

	return resp, nil
}

// ok.
func (s *friendServer) IsFriend(ctx context.Context, req *relation.IsFriendReq) (resp *relation.IsFriendResp, err er
	if err := authverify.CheckAccessIn(ctx, req.UserID1, req.UserID2); err != nil {
		return nil, err
	}
	resp = &relation.IsFriendResp{}
```

```
	resp.InUser1Friends, resp.InUser2Friends, err = s.db.CheckIn(ctx, req.UserID1, req.UserID2)
	if err != nil {
		return nil, err
	}
	return resp, nil
}

func (s *friendServer) GetPaginationFriends(ctx context.Context, req *relation.GetPaginationFriendsReq) (resp *relat
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}

	total, friends, err := s.db.PageOwnerFriends(ctx, req.UserID, req.Pagination)
	if err != nil {
		return nil, err
	}

	resp = &relation.GetPaginationFriendsResp{}
	resp.FriendsInfo, err = convert.FriendsDB2Pb(ctx, friends, s.userClient.GetUsersInfoMap)
	if err != nil {
		return nil, err
	}

	resp.Total = int32(total)

	return resp, nil
}

func (s *friendServer) GetFriendIDs(ctx context.Context, req *relation.GetFriendIDsReq) (resp *relation.GetFriendIDs
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}

	resp = &relation.GetFriendIDsResp{}
	resp.FriendIDs, err = s.db.FindFriendUserIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}

	return resp, nil
}

func (s *friendServer) GetSpecifiedFriendsInfo(ctx context.Context, req *relation.GetSpecifiedFriendsInfoReq) (*rela
	if len(req.UserIDList) == 0 {
		return nil, errs.ErrArgs.WrapMsg("userIDList is empty")
	}

	if datautil.Duplicate(req.UserIDList) {
		return nil, errs.ErrArgs.WrapMsg("userIDList repeated")
	}

	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
	}
	userMap, err := s.userClient.GetUsersInfoMap(ctx, req.UserIDList)
	if err != nil {
		return nil, err
	}

	friends, err := s.db.FindFriendsWithError(ctx, req.OwnerUserID, req.UserIDList)
	if err != nil {
		return nil, err
	}

	blacks, err := s.blackDatabase.FindBlackInfos(ctx, req.OwnerUserID, req.UserIDList)
	if err != nil {
```

```
		return nil, err
	}

	friendMap := datautil.SliceToMap(friends, func(e *model.Friend) string {
		return e.FriendUserID
	})

	blackMap := datautil.SliceToMap(blacks, func(e *model.Black) string {
		return e.BlockUserID
	})

	resp := &relation.GetSpecifiedFriendsInfoResp{
		Infos: make([]*relation.GetSpecifiedFriendsInfoInfo, 0, len(req.UserIDList)),
	}

	for _, userID := range req.UserIDList {
		user := userMap[userID]

		if user == nil {
			continue
		}

		var friendInfo *sdkws.FriendInfo
		if friend := friendMap[userID]; friend != nil {
			friendInfo = &sdkws.FriendInfo{
				OwnerUserID:    friend.OwnerUserID,
				Remark:         friend.Remark,
				CreateTime:     friend.CreateTime.UnixMilli(),
				AddSource:      friend.AddSource,
				OperatorUserID: friend.OperatorUserID,
				Ex:             friend.Ex,
				IsPinned:       friend.IsPinned,
			}
		}

		var blackInfo *sdkws.BlackInfo
		if black := blackMap[userID]; black != nil {
			blackInfo = &sdkws.BlackInfo{
				OwnerUserID:    black.OwnerUserID,
				CreateTime:     black.CreateTime.UnixMilli(),
				AddSource:      black.AddSource,
				OperatorUserID: black.OperatorUserID,
				Ex:             black.Ex,
			}
		}

		resp.Infos = append(resp.Infos, &relation.GetSpecifiedFriendsInfoInfo{
			UserInfo:   user,
			FriendInfo: friendInfo,
			BlackInfo:  blackInfo,
		})
	}

	return resp, nil
}

func (s *friendServer) UpdateFriends(ctx context.Context, req *relation.UpdateFriendsReq) (*relation.UpdateFriendsRe
	if len(req.FriendUserIDs) == 0 {
		return nil, errs.ErrArgs.WrapMsg("friendIDList is empty")
	}
	if datautil.Duplicate(req.FriendUserIDs) {
		return nil, errs.ErrArgs.WrapMsg("friendIDList repeated")
	}

	if err := authverify.CheckAccess(ctx, req.OwnerUserID); err != nil {
		return nil, err
```

```
	}

	_, err := s.db.FindFriendsWithError(ctx, req.OwnerUserID, req.FriendUserIDs)
	if err != nil {
		return nil, err
	}

	val := make(map[string]any)

	if req.IsPinned != nil {
		val["is_pinned"] = req.IsPinned.Value
	}
	if req.Remark != nil {
		val["remark"] = req.Remark.Value
	}
	if req.Ex != nil {
		val["ex"] = req.Ex.Value
	}
	if err = s.db.UpdateFriends(ctx, req.OwnerUserID, req.FriendUserIDs, val); err != nil {
		return nil, err
	}

	resp := &relation.UpdateFriendsResp{}

	s.notificationSender.FriendsInfoUpdateNotification(ctx, req.OwnerUserID, req.FriendUserIDs)
	return resp, nil
}

func (s *friendServer) GetSelfUnhandledApplyCount(ctx context.Context, req *relation.GetSelfUnhandledApplyCountReq)
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}

	count, err := s.db.GetUnhandledCount(ctx, req.UserID, req.Time)
	if err != nil {
		return nil, err
	}

	return &relation.GetSelfUnhandledApplyCountResp{
		Count: count,
	}, nil
}

func (s *friendServer) getCommonUserMap(ctx context.Context, userIDs []string) (map[string]common_user.CommonUser, e
	users, err := s.userClient.GetUsersInfo(ctx, userIDs)
	if err != nil {
		return nil, err
	}
	return datautil.SliceToMapAny(users, func(e *sdkws.UserInfo) (string, common_user.CommonUser) {
		return e.UserID, e
	}), nil
}
```

internal/rpc/relation/notification.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package relation

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/datautil"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/versionctx"

    relationtb "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/convert"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/open-im-server/v3/pkg/notification"
    "github.com/openimsdk/open-im-server/v3/pkg/notification/common_user"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/relation"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/mcontext"
)

type FriendNotificationSender struct {
    *notification.NotificationSender
    // Target not found err
    getUsersInfo func(ctx context.Context, userIDs []string) ([]common_user.CommonUser, error)
    // db controller
    db controller.FriendDatabase
}

type friendNotificationSenderOptions func(*FriendNotificationSender)

func WithFriendDB(db controller.FriendDatabase) friendNotificationSenderOptions {
    return func(s *FriendNotificationSender) {
        s.db = db
    }
}

func WithDBFunc(fn func(ctx context.Context, userIDs []string) (users []*relationtb.User, err error)) friendNotifica
    return func(s *FriendNotificationSender) {
        f := func(ctx context.Context, userIDs []string) (result []common_user.CommonUser, err error) {
            users, err := fn(ctx, userIDs)
            if err != nil {
                return nil, err
            }
```

```
        for _, user := range users {
            result = append(result, user)
        }
        return result, nil
    }
    s.getUsersInfo = f
}
}

func WithRpcFunc(fn func(ctx context.Context, userIDs []string) ([]*sdkws.UserInfo, error)) friendNotificationSender
return func(s *FriendNotificationSender) {
    f := func(ctx context.Context, userIDs []string) (result []common_user.CommonUser, err error) {
        users, err := fn(ctx, userIDs)
        if err != nil {
            return nil, err
        }
        for _, user := range users {
            result = append(result, user)
        }
        return result, err
    }
    s.getUsersInfo = f
}
}

func NewFriendNotificationSender(conf *config.Notification, msgClient *rpcli.MsgClient, opts ...friendNotificationSe
f := &FriendNotificationSender{
    NotificationSender: notification.NewNotificationSender(conf, notification.WithRpcClient(func(ctx context.Context,
        return msgClient.SendMsg(ctx, req)
    })),
}
for _, opt := range opts {
    opt(f)
}
return f
}

func (f *FriendNotificationSender) getUsersInfoMap(ctx context.Context, userIDs []string) (map[string]*sdkws.UserInf
users, err := f.getUsersInfo(ctx, userIDs)
if err != nil {
    return nil, err
}
result := make(map[string]*sdkws.UserInfo)
for _, user := range users {
    result[user.GetUserID()] = user.(*sdkws.UserInfo)
}
return result, nil
}

//nolint:unused
func (f *FriendNotificationSender) getFromToUserNickname(ctx context.Context, fromUserID, toUserID string) (string,
users, err := f.getUsersInfoMap(ctx, []string{fromUserID, toUserID})
if err != nil {
    return "", "", nil
}
return users[fromUserID].Nickname, users[toUserID].Nickname, nil
}

func (f *FriendNotificationSender) UserInfoUpdatedNotification(ctx context.Context, changedUserID string) {
tips := sdkws.UserInfoUpdatedTips{UserID: changedUserID}
f.Notification(ctx, mcontext.GetOpUserID(ctx), changedUserID, constant.UserInfoUpdatedNotification, &tips)
}

func (f *FriendNotificationSender) getCommonUserMap(ctx context.Context, userIDs []string) (map[string]common_user.C
users, err := f.getUsersInfo(ctx, userIDs)
if err != nil {
```

```
		return nil, err
	}
	return datautil.SliceToMap(users, func(e common_user.CommonUser) string {
		return e.GetUserID()
	}), nil
}

func (f *FriendNotificationSender) getFriendRequests(ctx context.Context, fromUserID, toUserID string) (*sdkws.Frien
	if f.db == nil {
		return nil, errs.ErrInternalServer.WithDetail("db is nil")
	}
	friendRequests, err := f.db.FindBothFriendRequests(ctx, fromUserID, toUserID)
	if err != nil {
		return nil, err
	}
	requests, err := convert.FriendRequestDB2Pb(ctx, friendRequests, f.getCommonUserMap)
	if err != nil {
		return nil, err
	}
	for _, request := range requests {
		if request.FromUserID == fromUserID && request.ToUserID == toUserID {
			return request, nil
		}
	}
	return nil, errs.ErrRecordNotFound.WrapMsg("friend request not found", "fromUserID", fromUserID, "toUserID", toUser
}

func (f *FriendNotificationSender) FriendApplicationAddNotification(ctx context.Context, req *relation.ApplyToAddFri
	request, err := f.getFriendRequests(ctx, req.FromUserID, req.ToUserID)
	if err != nil {
		log.ZError(ctx, "FriendApplicationAddNotification get friend request", err, "fromUserID", req.FromUserID, "toUser
		return
	}
	tips := sdkws.FriendApplicationTips{
		FromToUserID: &sdkws.FromToUserID{
			FromUserID: req.FromUserID,
			ToUserID:   req.ToUserID,
		},
		Request: request,
	}
	f.Notification(ctx, req.FromUserID, req.ToUserID, constant.FriendApplicationNotification, &tips)
}

func (f *FriendNotificationSender) FriendApplicationAgreedNotification(ctx context.Context, req *relation.RespondFri
	var (
		request *sdkws.FriendRequest
		err     error
	)
	if checkReq {
		request, err = f.getFriendRequests(ctx, req.FromUserID, req.ToUserID)
		if err != nil {
			log.ZError(ctx, "FriendApplicationAgreedNotification get friend request", err, "fromUserID", req.FromUserID, "to
			return
		}
	}
	tips := sdkws.FriendApplicationApprovedTips{
		FromToUserID: &sdkws.FromToUserID{
			FromUserID: req.FromUserID,
			ToUserID:   req.ToUserID,
		},
		HandleMsg: req.HandleMsg,
		Request:   request,
	}
	f.Notification(ctx, req.ToUserID, req.FromUserID, constant.FriendApplicationApprovedNotification, &tips)
}
```

```
func (f *FriendNotificationSender) FriendApplicationRefusedNotification(ctx context.Context, req *relation.RespondFr
    request, err := f.getFriendRequests(ctx, req.FromUserID, req.ToUserID)
    if err != nil {
        log.ZError(ctx, "FriendApplicationRefusedNotification get friend request", err, "fromUserID", req.FromUserID, "toU
        return
    }
    tips := sdkws.FriendApplicationRejectedTips{
        FromToUserID: &sdkws.FromToUserID{
            FromUserID: req.FromUserID,
            ToUserID:   req.ToUserID,
        },
        HandleMsg: req.HandleMsg,
        Request:   request,
    }
    f.Notification(ctx, req.ToUserID, req.FromUserID, constant.FriendApplicationRejectedNotification, &tips)
}

//func (f *FriendNotificationSender) FriendAddedNotification(ctx context.Context, operationID, opUserID, fromUserID,
//    tips := sdkws.FriendAddedTips{Friend: &sdkws.FriendInfo{}, OpUser: &sdkws.PublicUserInfo{}}
//    user, err := f.getUsersInfo(ctx, []string{opUserID})
//    if err != nil {
//        return err
//    }
//    tips.OpUser.UserID = user[0].GetUserID()
//    tips.OpUser.Ex = user[0].GetEx()
//    tips.OpUser.Nickname = user[0].GetNickname()
//    tips.OpUser.FaceURL = user[0].GetFaceURL()
//    friends, err := f.db.FindFriendsWithError(ctx, fromUserID, []string{toUserID})
//    if err != nil {
//        return err
//    }
//    tips.Friend, err = convert.FriendDB2Pb(ctx, friends[0], f.getUsersInfoMap)
//    if err != nil {
//        return err
//    }
//    f.Notification(ctx, fromUserID, toUserID, constant.FriendAddedNotification, &tips)
//    return nil
//}

func (f *FriendNotificationSender) FriendDeletedNotification(ctx context.Context, req *relation.DeleteFriendReq) {
    tips := sdkws.FriendDeletedTips{FromToUserID: &sdkws.FromToUserID{
        FromUserID: req.OwnerUserID,
        ToUserID:   req.FriendUserID,
    }}
    f.Notification(ctx, req.OwnerUserID, req.FriendUserID, constant.FriendDeletedNotification, &tips)
}

func (f *FriendNotificationSender) setVersion(ctx context.Context, version *uint64, versionID *string, collName stri
    versions := versionctx.GetVersionLog(ctx).Get()
    for _, coll := range versions {
        if coll.Name == collName && coll.Doc.DID == id {
            *version = uint64(coll.Doc.Version)
            *versionID = coll.Doc.ID.Hex()
            return
        }
    }
}

func (f *FriendNotificationSender) setSortVersion(ctx context.Context, version *uint64, versionID *string, collName
    versions := versionctx.GetVersionLog(ctx).Get()
    for _, coll := range versions {
        if coll.Name == collName && coll.Doc.DID == id {
            *version = uint64(coll.Doc.Version)
            *versionID = coll.Doc.ID.Hex()
            for _, elem := range coll.Doc.Logs {
                if elem.EID == relationtb.VersionSortChangeID {
```

```
            *sortVersion = uint64(elem.Version)
        }
    }
  }
}

func (f *FriendNotificationSender) FriendRemarkSetNotification(ctx context.Context, fromUserID, toUserID string) {
  tips := sdkws.FriendInfoChangedTips{FromToUserID: &sdkws.FromToUserID{}}
  tips.FromToUserID.FromUserID = fromUserID
  tips.FromToUserID.ToUserID = toUserID
  f.setSortVersion(ctx, &tips.FriendVersion, &tips.FriendVersionID, database.FriendVersionName, toUserID, &tips.Frier
  f.Notification(ctx, fromUserID, toUserID, constant.FriendRemarkSetNotification, &tips)
}

func (f *FriendNotificationSender) FriendsInfoUpdateNotification(ctx context.Context, toUserID string, friendIDs []s
  tips := sdkws.FriendsInfoUpdateTips{FromToUserID: &sdkws.FromToUserID{}}
  tips.FromToUserID.ToUserID = toUserID
  tips.FriendIDs = friendIDs
  f.Notification(ctx, toUserID, toUserID, constant.FriendsInfoUpdateNotification, &tips)
}

func (f *FriendNotificationSender) BlackAddedNotification(ctx context.Context, req *relation.AddBlackReq) {
  tips := sdkws.BlackAddedTips{FromToUserID: &sdkws.FromToUserID{}}
  tips.FromToUserID.FromUserID = req.OwnerUserID
  tips.FromToUserID.ToUserID = req.BlackUserID
  f.Notification(ctx, req.OwnerUserID, req.BlackUserID, constant.BlackAddedNotification, &tips)
}

func (f *FriendNotificationSender) BlackDeletedNotification(ctx context.Context, req *relation.RemoveBlackReq) {
  blackDeletedTips := sdkws.BlackDeletedTips{FromToUserID: &sdkws.FromToUserID{
    FromUserID: req.OwnerUserID,
    ToUserID:   req.BlackUserID,
  }}
  f.Notification(ctx, req.OwnerUserID, req.BlackUserID, constant.BlackDeletedNotification, &blackDeletedTips)
}

func (f *FriendNotificationSender) FriendInfoUpdatedNotification(ctx context.Context, changedUserID string, needNoti
  tips := sdkws.UserInfoUpdatedTips{UserID: changedUserID}
  f.Notification(ctx, mcontext.GetOpUserID(ctx), needNotifiedUserID, constant.FriendInfoUpdatedNotification, &tips)
}
```

internal/rpc/relation/sync.go

```
package relation

import (
	"context"
	"slices"

	"github.com/openimsdk/open-im-server/v3/pkg/util/hashutil"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/log"

	"github.com/openimsdk/open-im-server/v3/internal/rpc/incrversion"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"
	"github.com/openimsdk/protocol/relation"
)

func (s *friendServer) NotificationUserInfoUpdate(ctx context.Context, req *relation.NotificationUserInfoUpdateReq)
	userIDs, err := s.db.FindFriendUserIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	if len(userIDs) > 0 {
		friendUserIDs := []string{req.UserID}
		noCancelCtx := context.WithoutCancel(ctx)
		err := s.queue.PushCtx(ctx, func() {
			for _, userID := range userIDs {
				if err := s.db.OwnerIncrVersion(noCancelCtx, userID, friendUserIDs, model.VersionStateUpdate); err != nil {
					log.ZError(ctx, "OwnerIncrVersion", err, "userID", userID, "friendUserIDs", friendUserIDs)
				}
			}
			for _, userID := range userIDs {
				s.notificationSender.FriendInfoUpdatedNotification(noCancelCtx, req.UserID, userID)
			}
		})
		if err != nil {
			log.ZError(ctx, "NotificationUserInfoUpdate timeout", err, "userID", req.UserID)
		}
	}
	return &relation.NotificationUserInfoUpdateResp{}, nil
}

func (s *friendServer) GetFullFriendUserIDs(ctx context.Context, req *relation.GetFullFriendUserIDsReq) (*relation.G
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	vl, err := s.db.FindMaxFriendVersionCache(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	userIDs, err := s.db.FindFriendUserIDs(ctx, req.UserID)
	if err != nil {
		return nil, err
	}
	idHash := hashutil.IdHash(userIDs)
	if req.IdHash == idHash {
		userIDs = nil
	}
	return &relation.GetFullFriendUserIDsResp{
		Version:   uint64(vl.Version),
		VersionID: vl.ID.Hex(),
		Equal:     req.IdHash == idHash,
		UserIDs:   userIDs,
	}, nil
}
```

```go
func (s *friendServer) GetIncrementalFriends(ctx context.Context, req *relation.GetIncrementalFriendsReq) (*relation
  if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
    return nil, err
  }
  var sortVersion uint64
  opt := incrversion.Option[*sdkws.FriendInfo, relation.GetIncrementalFriendsResp]{
    Ctx:           ctx,
    VersionKey:    req.UserID,
    VersionID:     req.VersionID,
    VersionNumber: req.Version,
    Version: func(ctx context.Context, ownerUserID string, version uint, limit int) (*model.VersionLog, error) {
      vl, err := s.db.FindFriendIncrVersion(ctx, ownerUserID, version, limit)
      if err != nil {
        return nil, err
      }
      vl.Logs = slices.DeleteFunc(vl.Logs, func(elem model.VersionLogElem) bool {
        if elem.EID == model.VersionSortChangeID {
          vl.LogLen--
          sortVersion = uint64(elem.Version)
          return true
        }
        return false
      })
      return vl, nil
    },
    CacheMaxVersion: s.db.FindMaxFriendVersionCache,
    Find: func(ctx context.Context, ids []string) ([]*sdkws.FriendInfo, error) {
      return s.getFriend(ctx, req.UserID, ids)
    },
    Resp: func(version *model.VersionLog, deleteIds []string, insertList, updateList []*sdkws.FriendInfo, full bool)
      return &relation.GetIncrementalFriendsResp{
        VersionID:   version.ID.Hex(),
        Version:     uint64(version.Version),
        Full:        full,
        Delete:      deleteIds,
        Insert:      insertList,
        Update:      updateList,
        SortVersion: sortVersion,
      }
    },
  }
  return opt.Build()
}
```

internal/rpc/msg

internal/rpc/msg/as_read.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
	"context"
	"errors"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/redis/go-redis/v9"
)

func (m *msgServer) GetConversationsHasReadAndMaxSeq(ctx context.Context, req *msg.GetConversationsHasReadAndMaxSeqR
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	var conversationIDs []string
	if len(req.ConversationIDs) == 0 {
		var err error
		conversationIDs, err = m.ConversationLocalCache.GetConversationIDs(ctx, req.UserID)
		if err != nil {
			return nil, err
		}
	} else {
		conversationIDs = req.ConversationIDs
	}

	hasReadSeqs, err := m.MsgDatabase.GetHasReadSeqs(ctx, req.UserID, conversationIDs)
	if err != nil {
		return nil, err
	}

	conversations, err := m.ConversationLocalCache.GetConversations(ctx, req.UserID, conversationIDs)
	if err != nil {
		return nil, err
	}

	conversationMaxSeqMap := make(map[string]int64)
	for _, conversation := range conversations {
		if conversation.MaxSeq != 0 {
			conversationMaxSeqMap[conversation.ConversationID] = conversation.MaxSeq
		}
```

```
	}
	maxSeqs, err := m.MsgDatabase.GetMaxSeqsWithTime(ctx, conversationIDs)
	if err != nil {
		return nil, err
	}
	resp := &msg.GetConversationsHasReadAndMaxSeqResp{Seqs: make(map[string]*msg.Seqs)}
	if req.ReturnPinned {
		pinnedConversationIDs, err := m.ConversationLocalCache.GetPinnedConversationIDs(ctx, req.UserID)
		if err != nil {
			return nil, err
		}
		resp.PinnedConversationIDs = pinnedConversationIDs
	}
	for conversationID, maxSeq := range maxSeqs {
		resp.Seqs[conversationID] = &msg.Seqs{
			HasReadSeq: hasReadSeqs[conversationID],
			MaxSeq:     maxSeq.Seq,
			MaxSeqTime: maxSeq.Time,
		}
		if v, ok := conversationMaxSeqMap[conversationID]; ok {
			resp.Seqs[conversationID].MaxSeq = v
		}
	}
	return resp, nil
}

func (m *msgServer) SetConversationHasReadSeq(ctx context.Context, req *msg.SetConversationHasReadSeqReq) (*msg.SetC
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	maxSeq, err := m.MsgDatabase.GetMaxSeq(ctx, req.ConversationID)
	if err != nil {
		return nil, err
	}
	if req.HasReadSeq > maxSeq {
		return nil, errs.ErrArgs.WrapMsg("hasReadSeq must not be bigger than maxSeq")
	}
	if err := m.MsgDatabase.SetHasReadSeq(ctx, req.UserID, req.ConversationID, req.HasReadSeq); err != nil {
		return nil, err
	}
	m.sendMarkAsReadNotification(ctx, req.ConversationID, constant.SingleChatType, req.UserID, req.UserID, nil, req.Has
	return &msg.SetConversationHasReadSeqResp{}, nil
}

func (m *msgServer) MarkMsgsAsRead(ctx context.Context, req *msg.MarkMsgsAsReadReq) (*msg.MarkMsgsAsReadResp, error)
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	maxSeq, err := m.MsgDatabase.GetMaxSeq(ctx, req.ConversationID)
	if err != nil {
		return nil, err
	}
	hasReadSeq := req.Seqs[len(req.Seqs)-1]
	if hasReadSeq > maxSeq {
		return nil, errs.ErrArgs.WrapMsg("hasReadSeq must not be bigger than maxSeq")
	}
	conversation, err := m.ConversationLocalCache.GetConversation(ctx, req.UserID, req.ConversationID)
	if err != nil {
		return nil, err
	}
	if err := m.MsgDatabase.MarkSingleChatMsgsAsRead(ctx, req.UserID, req.ConversationID, req.Seqs); err != nil {
		return nil, err
	}
	currentHasReadSeq, err := m.MsgDatabase.GetHasReadSeq(ctx, req.UserID, req.ConversationID)
	if err != nil && !errors.Is(err, redis.Nil) {
		return nil, err
```

```
	}
	if hasReadSeq > currentHasReadSeq {
		err = m.MsgDatabase.SetHasReadSeq(ctx, req.UserID, req.ConversationID, hasReadSeq)
		if err != nil {
			return nil, err
		}
	}

	reqCallback := &cbapi.CallbackSingleMsgReadReq{
		ConversationID: conversation.ConversationID,
		UserID:         req.UserID,
		Seqs:           req.Seqs,
		ContentType:    conversation.ConversationType,
	}
	m.webhookAfterSingleMsgRead(ctx, &m.config.WebhooksConfig.AfterSingleMsgRead, reqCallback)
	m.sendMarkAsReadNotification(ctx, req.ConversationID, conversation.ConversationType, req.UserID,
		m.conversationAndGetRecvID(conversation, req.UserID), req.Seqs, hasReadSeq)
	return &msg.MarkMsgsAsReadResp{}, nil
}

func (m *msgServer) MarkConversationAsRead(ctx context.Context, req *msg.MarkConversationAsReadReq) (*msg.MarkConver
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	conversation, err := m.ConversationLocalCache.GetConversation(ctx, req.UserID, req.ConversationID)
	if err != nil {
		return nil, err
	}
	hasReadSeq, err := m.MsgDatabase.GetHasReadSeq(ctx, req.UserID, req.ConversationID)
	if err != nil && !errors.Is(err, redis.Nil) {
		return nil, err
	}
	var seqs []int64

	log.ZDebug(ctx, "MarkConversationAsRead", "hasReadSeq", hasReadSeq, "req.HasReadSeq", req.HasReadSeq)
	if conversation.ConversationType == constant.SingleChatType {
		for i := hasReadSeq + 1; i <= req.HasReadSeq; i++ {
			seqs = append(seqs, i)
		}
		// avoid client missed call MarkConversationMessageAsRead by order
		for _, val := range req.Seqs {
			if !datautil.Contain(val, seqs...) {
				seqs = append(seqs, val)
			}
		}
		if len(seqs) > 0 {
			log.ZDebug(ctx, "MarkConversationAsRead", "seqs", seqs, "conversationID", req.ConversationID)
			if err = m.MsgDatabase.MarkSingleChatMsgsAsRead(ctx, req.UserID, req.ConversationID, seqs); err != nil {
				return nil, err
			}
		}
		if req.HasReadSeq > hasReadSeq {
			err = m.MsgDatabase.SetHasReadSeq(ctx, req.UserID, req.ConversationID, req.HasReadSeq)
			if err != nil {
				return nil, err
			}
			hasReadSeq = req.HasReadSeq
		}
		m.sendMarkAsReadNotification(ctx, req.ConversationID, conversation.ConversationType, req.UserID,
			m.conversationAndGetRecvID(conversation, req.UserID), seqs, hasReadSeq)
	} else if conversation.ConversationType == constant.ReadGroupChatType ||
		conversation.ConversationType == constant.NotificationChatType {
		if req.HasReadSeq > hasReadSeq {
			err = m.MsgDatabase.SetHasReadSeq(ctx, req.UserID, req.ConversationID, req.HasReadSeq)
			if err != nil {
				return nil, err
```

```
        }
        hasReadSeq = req.HasReadSeq
    }
    m.sendMarkAsReadNotification(ctx, req.ConversationID, constant.SingleChatType, req.UserID,
        req.UserID, seqs, hasReadSeq)
}

if conversation.ConversationType == constant.SingleChatType {
    reqCall := &cbapi.CallbackSingleMsgReadReq{
        ConversationID: conversation.ConversationID,
        UserID:         conversation.OwnerUserID,
        Seqs:           req.Seqs,
        ContentType:    conversation.ConversationType,
    }
    m.webhookAfterSingleMsgRead(ctx, &m.config.WebhooksConfig.AfterSingleMsgRead, reqCall)
} else if conversation.ConversationType == constant.ReadGroupChatType {
    reqCall := &cbapi.CallbackGroupMsgReadReq{
        SendID:       conversation.OwnerUserID,
        ReceiveID:    req.UserID,
        UnreadMsgNum: req.HasReadSeq,
        ContentType:  int64(conversation.ConversationType),
    }
    m.webhookAfterGroupMsgRead(ctx, &m.config.WebhooksConfig.AfterGroupMsgRead, reqCall)
}
return &msg.MarkConversationAsReadResp{}, nil
}

func (m *msgServer) sendMarkAsReadNotification(ctx context.Context, conversationID string, sessionType int32, sendID
    tips := &sdkws.MarkAsReadTips{
        MarkAsReadUserID: sendID,
        ConversationID:   conversationID,
        Seqs:             seqs,
        HasReadSeq:       hasReadSeq,
    }
    m.notificationSender.NotificationWithSessionType(ctx, sendID, recvID, constant.HasReadReceipt, sessionType, tips)
}
```

internal/rpc/msg/callback.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
	"context"
	"encoding/base64"
	"encoding/json"

	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/tools/errs"

	cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/constant"
	pbchat "github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/stringutil"
	"google.golang.org/protobuf/proto"
)

func toCommonCallback(ctx context.Context, msg *pbchat.SendMsgReq, command string) cbapi.CommonCallbackReq {
	return cbapi.CommonCallbackReq{
		SendID:           msg.MsgData.SendID,
		ServerMsgID:      msg.MsgData.ServerMsgID,
		CallbackCommand:  command,
		ClientMsgID:      msg.MsgData.ClientMsgID,
		OperationID:      mcontext.GetOperationID(ctx),
		SenderPlatformID: msg.MsgData.SenderPlatformID,
		SenderNickname:   msg.MsgData.SenderNickname,
		SessionType:      msg.MsgData.SessionType,
		MsgFrom:          msg.MsgData.MsgFrom,
		ContentType:      msg.MsgData.ContentType,
		Status:           msg.MsgData.Status,
		SendTime:         msg.MsgData.SendTime,
		CreateTime:       msg.MsgData.CreateTime,
		AtUserIDList:     msg.MsgData.AtUserIDList,
		SenderFaceURL:    msg.MsgData.SenderFaceURL,
		Content:          GetContent(msg.MsgData),
		Seq:              uint32(msg.MsgData.Seq),
		Ex:               msg.MsgData.Ex,
	}
}

func GetContent(msg *sdkws.MsgData) string {
	if msg.ContentType >= constant.NotificationBegin && msg.ContentType <= constant.NotificationEnd {
		var tips sdkws.TipsComm
		_ = proto.Unmarshal(msg.Content, &tips)
		content := tips.JsonDetail
```

```
        return content
    } else {
        return string(msg.Content)
    }
}

func (m *msgServer) webhookBeforeSendSingleMsg(ctx context.Context, before *config.BeforeConfig, msg *pbchat.SendMsg
    return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
        if msg.MsgData.ContentType == constant.Typing {
            return nil
        }
        if !filterBeforeMsg(msg, before) {
            return nil
        }
        cbReq := &cbapi.CallbackBeforeSendSingleMsgReq{
            CommonCallbackReq: toCommonCallback(ctx, msg, cbapi.CallbackBeforeSendSingleMsgCommand),
            RecvID:            msg.MsgData.RecvID,
        }
        resp := &cbapi.CallbackBeforeSendSingleMsgResp{}
        if err := m.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
            return err
        }

        return nil
    })
}

// Move to msgtransfer
func (m *msgServer) webhookAfterSendSingleMsg(ctx context.Context, after *config.AfterConfig, msg *pbchat.SendMsgReq
    if msg.MsgData.ContentType == constant.Typing {
        return
    }
    if !filterAfterMsg(msg, after) {
        return
    }
    cbReq := &cbapi.CallbackAfterSendSingleMsgReq{
        CommonCallbackReq: toCommonCallback(ctx, msg, cbapi.CallbackAfterSendSingleMsgCommand),
        RecvID:            msg.MsgData.RecvID,
    }
    m.webhookClient.AsyncPostWithQuery(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterSendSingleMsgResp{},
}

func (m *msgServer) webhookBeforeSendGroupMsg(ctx context.Context, before *config.BeforeConfig, msg *pbchat.SendMsgR
    return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
        if !filterBeforeMsg(msg, before) {
            return nil
        }
        if msg.MsgData.ContentType == constant.Typing {
            return nil
        }
        cbReq := &cbapi.CallbackBeforeSendGroupMsgReq{
            CommonCallbackReq: toCommonCallback(ctx, msg, cbapi.CallbackBeforeSendGroupMsgCommand),
            GroupID:           msg.MsgData.GroupID,
        }
        resp := &cbapi.CallbackBeforeSendGroupMsgResp{}
        if err := m.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
            return err
        }
        return nil
    })
}

func (m *msgServer) webhookAfterSendGroupMsg(ctx context.Context, after *config.AfterConfig, msg *pbchat.SendMsgReq)
    if msg.MsgData.ContentType == constant.Typing {
        return
    }
```

```
	if !filterAfterMsg(msg, after) {
		return
	}
	cbReq := &cbapi.CallbackAfterSendGroupMsgReq{
		CommonCallbackReq: toCommonCallback(ctx, msg, cbapi.CallbackAfterSendGroupMsgCommand),
		GroupID:           msg.MsgData.GroupID,
	}

	m.webhookClient.AsyncPostWithQuery(ctx, cbReq.GetCallbackCommand(), cbReq, &cbapi.CallbackAfterSendGroupMsgResp{},
}

func (m *msgServer) webhookBeforeMsgModify(ctx context.Context, before *config.BeforeConfig, msg *pbchat.SendMsgReq,
	return webhook.WithCondition(ctx, before, func(ctx context.Context) error {
		//if msg.MsgData.ContentType != constant.Text {
		//	return nil
		//}
		if !filterBeforeMsg(msg, before) {
			return nil
		}
		cbReq := &cbapi.CallbackMsgModifyCommandReq{
			CommonCallbackReq: toCommonCallback(ctx, msg, cbapi.CallbackBeforeMsgModifyCommand),
		}
		resp := &cbapi.CallbackMsgModifyCommandResp{}
		if err := m.webhookClient.SyncPost(ctx, cbReq.GetCallbackCommand(), cbReq, resp, before); err != nil {
			return err
		}
		if beforeMsgData != nil {
			*beforeMsgData = proto.Clone(msg.MsgData).(*sdkws.MsgData)
		}
		if resp.Content != nil {
			msg.MsgData.Content = []byte(*resp.Content)
			if err := json.Unmarshal(msg.MsgData.Content, &struct{}{}); err != nil {
				return errs.ErrArgs.WrapMsg("webhook msg modify content is not json", "content", string(msg.MsgData.Content))
			}
		}
		datautil.NotNilReplace(msg.MsgData.OfflinePushInfo, resp.OfflinePushInfo)
		datautil.NotNilReplace(&msg.MsgData.RecvID, resp.RecvID)
		datautil.NotNilReplace(&msg.MsgData.GroupID, resp.GroupID)
		datautil.NotNilReplace(&msg.MsgData.ClientMsgID, resp.ClientMsgID)
		datautil.NotNilReplace(&msg.MsgData.ServerMsgID, resp.ServerMsgID)
		datautil.NotNilReplace(&msg.MsgData.SenderPlatformID, resp.SenderPlatformID)
		datautil.NotNilReplace(&msg.MsgData.SenderNickname, resp.SenderNickname)
		datautil.NotNilReplace(&msg.MsgData.SenderFaceURL, resp.SenderFaceURL)
		datautil.NotNilReplace(&msg.MsgData.SessionType, resp.SessionType)
		datautil.NotNilReplace(&msg.MsgData.MsgFrom, resp.MsgFrom)
		datautil.NotNilReplace(&msg.MsgData.ContentType, resp.ContentType)
		datautil.NotNilReplace(&msg.MsgData.Status, resp.Status)
		datautil.NotNilReplace(&msg.MsgData.Options, resp.Options)
		datautil.NotNilReplace(&msg.MsgData.AtUserIDList, resp.AtUserIDList)
		datautil.NotNilReplace(&msg.MsgData.AttachedInfo, resp.AttachedInfo)
		datautil.NotNilReplace(&msg.MsgData.Ex, resp.Ex)
		return nil
	})
}

func (m *msgServer) webhookAfterGroupMsgRead(ctx context.Context, after *config.AfterConfig, req *cbapi.CallbackGrou
	req.CallbackCommand = cbapi.CallbackAfterGroupMsgReadCommand
	m.webhookClient.AsyncPost(ctx, req.GetCallbackCommand(), req, &cbapi.CallbackGroupMsgReadResp{}, after)
}

func (m *msgServer) webhookAfterSingleMsgRead(ctx context.Context, after *config.AfterConfig, req *cbapi.CallbackSin

	req.CallbackCommand = cbapi.CallbackAfterSingleMsgReadCommand

	m.webhookClient.AsyncPost(ctx, req.GetCallbackCommand(), req, &cbapi.CallbackSingleMsgReadResp{}, after)
```

```
}

func (m *msgServer) webhookAfterRevokeMsg(ctx context.Context, after *config.AfterConfig, req *pbchat.RevokeMsgReq)
	callbackReq := &cbapi.CallbackAfterRevokeMsgReq{
		CallbackCommand: cbapi.CallbackAfterRevokeMsgCommand,
		ConversationID:  req.ConversationID,
		Seq:             req.Seq,
		UserID:          req.UserID,
	}
	m.webhookClient.AsyncPost(ctx, callbackReq.GetCallbackCommand(), callbackReq, &cbapi.CallbackAfterRevokeMsgResp{},
}

func buildKeyMsgDataQuery(msg *sdkws.MsgData) map[string]string {
	keyMsgData := apistruct.KeyMsgData{
		SendID:  msg.SendID,
		RecvID:  msg.RecvID,
		GroupID: msg.GroupID,
	}

	return map[string]string{
		webhook.Key: base64.StdEncoding.EncodeToString(stringutil.StructToJsonBytes(keyMsgData)),
	}
}
```

internal/rpc/msg/clear.go

```
package msg

import (
	"context"
	"strings"

	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/tools/log"
)

// DestructMsgs hard delete in Database.
func (m *msgServer) DestructMsgs(ctx context.Context, req *msg.DestructMsgsReq) (*msg.DestructMsgsResp, error) {
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	docs, err := m.MsgDatabase.GetRandBeforeMsg(ctx, req.Timestamp, int(req.Limit))
	if err != nil {
		return nil, err
	}
	for i, doc := range docs {
		if err := m.MsgDatabase.DeleteDoc(ctx, doc.DocID); err != nil {
			return nil, err
		}
		log.ZDebug(ctx, "DestructMsgs delete doc", "index", i, "docID", doc.DocID)
		index := strings.LastIndex(doc.DocID, ":")
		if index < 0 {
			continue
		}
		var minSeq int64
		for _, model := range doc.Msg {
			if model.Msg == nil {
				continue
			}
			if model.Msg.Seq > minSeq {
				minSeq = model.Msg.Seq
			}
		}
		if minSeq <= 0 {
			continue
		}
		conversationID := doc.DocID[:index]
		if conversationID == "" {
			continue
		}
		minSeq++
		if err := m.MsgDatabase.SetMinSeq(ctx, conversationID, minSeq); err != nil {
			return nil, err
		}
		log.ZDebug(ctx, "DestructMsgs delete doc set min seq", "index", i, "docID", doc.DocID, "conversationID", conversa
	}
	return &msg.DestructMsgsResp{Count: int32(len(docs))}, nil
}

func (m *msgServer) GetLastMessageSeqByTime(ctx context.Context, req *msg.GetLastMessageSeqByTimeReq) (*msg.GetLastM
	seq, err := m.MsgDatabase.GetLastMessageSeqByTime(ctx, req.ConversationID, req.Time)
	if err != nil {
		return nil, err
	}
	return &msg.GetLastMessageSeqByTimeResp{Seq: seq}, nil
}
```

internal/rpc/msg/delete.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/conversation"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/timeutil"
)

func (m *msgServer) getMinSeqs(maxSeqs map[string]int64) map[string]int64 {
    minSeqs := make(map[string]int64)
    for k, v := range maxSeqs {
        minSeqs[k] = v + 1
    }
    return minSeqs
}

func (m *msgServer) validateDeleteSyncOpt(opt *msg.DeleteSyncOpt) (isSyncSelf, isSyncOther bool) {
    if opt == nil {
        return
    }
    return opt.IsSyncSelf, opt.IsSyncOther
}

func (m *msgServer) ClearConversationsMsg(ctx context.Context, req *msg.ClearConversationsMsgReq) (*msg.ClearConvers
    if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
        return nil, err
    }
    if err := m.clearConversation(ctx, req.ConversationIDs, req.UserID, req.DeleteSyncOpt); err != nil {
        return nil, err
    }
    return &msg.ClearConversationsMsgResp{}, nil
}

func (m *msgServer) UserClearAllMsg(ctx context.Context, req *msg.UserClearAllMsgReq) (*msg.UserClearAllMsgResp, err
    if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
        return nil, err
    }
    conversationIDs, err := m.ConversationLocalCache.GetConversationIDs(ctx, req.UserID)
    if err != nil {
        return nil, err
    }
    if err := m.clearConversation(ctx, conversationIDs, req.UserID, req.DeleteSyncOpt); err != nil {
        return nil, err
    }
```

```
	return &msg.UserClearAllMsgResp{}, nil
}

func (m *msgServer) DeleteMsgs(ctx context.Context, req *msg.DeleteMsgsReq) (*msg.DeleteMsgsResp, error) {
	if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
		return nil, err
	}
	isSyncSelf, isSyncOther := m.validateDeleteSyncOpt(req.DeleteSyncOpt)
	if isSyncOther {
		if err := m.MsgDatabase.DeleteMsgsPhysicalBySeqs(ctx, req.ConversationID, req.Seqs); err != nil {
			return nil, err
		}
		conv, err := m.conversationClient.GetConversationsByConversationID(ctx, req.ConversationID)
		if err != nil {
			return nil, err
		}
		tips := &sdkws.DeleteMsgsTips{UserID: req.UserID, ConversationID: req.ConversationID, Seqs: req.Seqs}
		m.notificationSender.NotificationWithSessionType(ctx, req.UserID, m.conversationAndGetRecvID(conv, req.UserID),
			constant.DeleteMsgsNotification, conv.ConversationType, tips)
	} else {
		if err := m.MsgDatabase.DeleteUserMsgsBySeqs(ctx, req.UserID, req.ConversationID, req.Seqs); err != nil {
			return nil, err
		}
		if isSyncSelf {
			tips := &sdkws.DeleteMsgsTips{UserID: req.UserID, ConversationID: req.ConversationID, Seqs: req.Seqs}
			m.notificationSender.NotificationWithSessionType(ctx, req.UserID, req.UserID, constant.DeleteMsgsNotification, c
		}
	}
	return &msg.DeleteMsgsResp{}, nil
}

func (m *msgServer) DeleteMsgPhysicalBySeq(ctx context.Context, req *msg.DeleteMsgPhysicalBySeqReq) (*msg.DeleteMsgP
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	err := m.MsgDatabase.DeleteMsgsPhysicalBySeqs(ctx, req.ConversationID, req.Seqs)
	if err != nil {
		return nil, err
	}
	return &msg.DeleteMsgPhysicalBySeqResp{}, nil
}

func (m *msgServer) DeleteMsgPhysical(ctx context.Context, req *msg.DeleteMsgPhysicalReq) (*msg.DeleteMsgPhysicalRes
	if err := authverify.CheckAdmin(ctx); err != nil {
		return nil, err
	}
	remainTime := timeutil.GetCurrentTimestampBySecond() - req.Timestamp
	if _, err := m.DestructMsgs(ctx, &msg.DestructMsgsReq{Timestamp: remainTime, Limit: 9999}); err != nil {
		return nil, err
	}
	return &msg.DeleteMsgPhysicalResp{}, nil
}

func (m *msgServer) clearConversation(ctx context.Context, conversationIDs []string, userID string, deleteSyncOpt *m
	conversations, err := m.conversationClient.GetConversationsByConversationIDs(ctx, conversationIDs)
	if err != nil {
		return err
	}
	var existConversations []*conversation.Conversation
	var existConversationIDs []string
	for _, conversation := range conversations {
		existConversations = append(existConversations, conversation)
		existConversationIDs = append(existConversationIDs, conversation.ConversationID)
	}
	log.ZDebug(ctx, "ClearConversationsMsg", "existConversationIDs", existConversationIDs)
	maxSeqs, err := m.MsgDatabase.GetMaxSeqs(ctx, existConversationIDs)
```

```
	if err != nil {
		return err
	}
	isSyncSelf, isSyncOther := m.validateDeleteSyncOpt(deleteSyncOpt)
	if !isSyncOther {
		setSeqs := m.getMinSeqs(maxSeqs)
		if err := m.MsgDatabase.SetUserConversationsMinSeqs(ctx, userID, setSeqs); err != nil {
			return err
		}
		ownerUserIDs := []string{userID}
		for conversationID, seq := range setSeqs {
			if err := m.conversationClient.SetConversationMinSeq(ctx, conversationID, ownerUserIDs, seq); err != nil {
				return err
			}
		}
		// notification 2 self
		if isSyncSelf {
			tips := &sdkws.ClearConversationTips{UserID: userID, ConversationIDs: existConversationIDs}
			m.notificationSender.NotificationWithSessionType(ctx, userID, userID, constant.ClearConversationNotification, co
		}
	} else {
		if err := m.MsgDatabase.SetMinSeqs(ctx, m.getMinSeqs(maxSeqs)); err != nil {
			return err
		}
		for _, conversation := range existConversations {
			tips := &sdkws.ClearConversationTips{UserID: userID, ConversationIDs: []string{conversation.ConversationID}}
			m.notificationSender.NotificationWithSessionType(ctx, userID, m.conversationAndGetRecvID(conversation, userID),
		}
	}
	if err := m.MsgDatabase.UserSetHasReadSeqs(ctx, userID, maxSeqs); err != nil {
		return err
	}
	return nil
}
```

internal/rpc/msg/filter.go

```
package msg

import (
	"strconv"
	"strings"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/constant"
	pbchat "github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/tools/utils/datautil"
)

const (
	separator = "-"
)

func filterAfterMsg(msg *pbchat.SendMsgReq, after *config.AfterConfig) bool {
	return filterMsg(msg, after.AttentionIds, after.DeniedTypes)
}

func filterBeforeMsg(msg *pbchat.SendMsgReq, before *config.BeforeConfig) bool {
	return filterMsg(msg, nil, before.DeniedTypes)
}

func filterMsg(msg *pbchat.SendMsgReq, attentionIds []string, deniedTypes []int32) bool {
	// According to the attentionIds configuration, only some users are sent
	if len(attentionIds) != 0 && !datautil.Contain(msg.MsgData.RecvID, attentionIds...) {
		return false
	}

	if defaultDeniedTypes(msg.MsgData.ContentType) {
		return false
	}

	if len(deniedTypes) != 0 && datautil.Contain(msg.MsgData.ContentType, deniedTypes...) {
		return false
	}
	//if len(allowedTypes) != 0 && !isInInterval(msg.MsgData.ContentType, allowedTypes) {
	//	return false
	//}
	//if len(deniedTypes) != 0 && isInInterval(msg.MsgData.ContentType, deniedTypes) {
	//	return false
	//}
	return true
}

func defaultDeniedTypes(contentType int32) bool {
	if contentType >= constant.NotificationBegin && contentType <= constant.NotificationEnd {
		return true
	}
	if contentType == constant.Typing {
		return true
	}
	return false
}

// isInInterval if data is in interval
// Supports two formats: a single type or a range. The range is defined by the lower and upper bounds connected with
// e.g. [1, 100, 200-500, 600-700] means that only data within the range
// {1, 100} ∪ [200, 500] ∪ [600, 700] will return true.
func isInInterval(data int32, interval []string) bool {
	for _, v := range interval {
		if strings.Contains(v, separator) {
			// is interval
```

```
        bounds := strings.Split(v, separator)
        if len(bounds) != 2 {
            continue
        }
        bottom, err := strconv.Atoi(bounds[0])
        if err != nil {
            continue
        }
        top, err := strconv.Atoi(bounds[1])
        if err != nil {
            continue
        }
        if datautil.BetweenEq(int(data), bottom, top) {
            return true
        }
    } else {
        iv, err := strconv.Atoi(v)
        if err != nil {
            continue
        }
        if int(data) == iv {
            return true
        }
    }
  }
  return false
}
```

internal/rpc/msg/msg_status.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"

    "github.com/openimsdk/protocol/constant"
    pbmsg "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/tools/mcontext"
)

func (m *msgServer) SetSendMsgStatus(ctx context.Context, req *pbmsg.SetSendMsgStatusReq) (*pbmsg.SetSendMsgStatusRe
    resp := &pbmsg.SetSendMsgStatusResp{}
    if err := m.MsgDatabase.SetSendMsgStatus(ctx, mcontext.GetOperationID(ctx), req.Status); err != nil {
        return nil, err
    }
    return resp, nil
}

func (m *msgServer) GetSendMsgStatus(ctx context.Context, req *pbmsg.GetSendMsgStatusReq) (*pbmsg.GetSendMsgStatusRe
    resp := &pbmsg.GetSendMsgStatusResp{}
    status, err := m.MsgDatabase.GetSendMsgStatus(ctx, mcontext.GetOperationID(ctx))
    if IsNotFound(err) {
        resp.Status = constant.MsgStatusNotExist
        return resp, nil
    } else if err != nil {
        return nil, err
    }
    resp.Status = status
    return resp, nil
}
```

internal/rpc/msg/notification.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
"context"

"github.com/openimsdk/open-im-server/v3/pkg/notification"
"github.com/openimsdk/protocol/constant"
"github.com/openimsdk/protocol/sdkws"
)

type MsgNotificationSender struct {
*notification.NotificationSender
}

func NewMsgNotificationSender(config *Config, opts ...notification.NotificationSenderOptions) *MsgNotificationSender
return &MsgNotificationSender{notification.NewNotificationSender(&config.NotificationConfig, opts...)}
}

func (m *MsgNotificationSender) UserDeleteMsgsNotification(ctx context.Context, userID, conversationID string, seqs
tips := sdkws.DeleteMsgsTips{
UserID:         userID,
ConversationID: conversationID,
Seqs:           seqs,
}
m.Notification(ctx, userID, userID, constant.DeleteMsgsNotification, &tips)
}

func (m *MsgNotificationSender) MarkAsReadNotification(ctx context.Context, conversationID string, sessionType int32
tips := &sdkws.MarkAsReadTips{
MarkAsReadUserID: sendID,
ConversationID:   conversationID,
Seqs:             seqs,
HasReadSeq:       hasReadSeq,
}
m.NotificationWithSessionType(ctx, sendID, recvID, constant.HasReadReceipt, sessionType, tips)
}
```

internal/rpc/msg/revoke.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"
    "encoding/json"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/utils/datautil"
)

func (m *msgServer) RevokeMsg(ctx context.Context, req *msg.RevokeMsgReq) (*msg.RevokeMsgResp, error) {
    if req.UserID == "" {
        return nil, errs.ErrArgs.WrapMsg("user_id is empty")
    }
    if req.ConversationID == "" {
        return nil, errs.ErrArgs.WrapMsg("conversation_id is empty")
    }
    if req.Seq < 0 {
        return nil, errs.ErrArgs.WrapMsg("seq is invalid")
    }
    if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
        return nil, err
    }
    user, err := m.UserLocalCache.GetUserInfo(ctx, req.UserID)
    if err != nil {
        return nil, err
    }
    _, _, msgs, err := m.MsgDatabase.GetMsgBySeqs(ctx, req.UserID, req.ConversationID, []int64{req.Seq})
    if err != nil {
        return nil, err
    }
    if len(msgs) == 0 || msgs[0] == nil {
        return nil, errs.ErrRecordNotFound.WrapMsg("msg not found")
    }
    if msgs[0].ContentType == constant.MsgRevokeNotification {
        return nil, servererrs.ErrMsgAlreadyRevoke.WrapMsg("msg already revoke")
    }

    data, _ := json.Marshal(msgs[0])
    log.ZDebug(ctx, "GetMsgBySeqs", "conversationID", req.ConversationID, "seq", req.Seq, "msg", string(data))
```

```
	var role int32
	if !authverify.IsAdmin(ctx) {
		sessionType := msgs[0].SessionType
		switch sessionType {
		case constant.SingleChatType:
			if err := authverify.CheckAccess(ctx, msgs[0].SendID); err != nil {
				return nil, err
			}
			role = user.AppMangerLevel
		case constant.ReadGroupChatType:
			members, err := m.GroupLocalCache.GetGroupMemberInfoMap(ctx, msgs[0].GroupID, datautil.Distinct([]string{req.Use
			if err != nil {
				return nil, err
			}
			if req.UserID != msgs[0].SendID {
				switch members[req.UserID].RoleLevel {
				case constant.GroupOwner:
				case constant.GroupAdmin:
					if sendMember, ok := members[msgs[0].SendID]; ok {
						if sendMember.RoleLevel != constant.GroupOrdinaryUsers {
							return nil, errs.ErrNoPermission.WrapMsg("no permission")
						}
					}
				default:
					return nil, errs.ErrNoPermission.WrapMsg("no permission")
				}
			}
			if member := members[req.UserID]; member != nil {
				role = member.RoleLevel
			}
		default:
			return nil, errs.ErrInternalServer.WrapMsg("msg sessionType not supported", "sessionType", sessionType)
		}
	}
	now := time.Now().UnixMilli()
	err = m.MsgDatabase.RevokeMsg(ctx, req.ConversationID, req.Seq, &model.RevokeModel{
		Role:     role,
		UserID:   req.UserID,
		Nickname: user.Nickname,
		Time:     now,
	})
	if err != nil {
		return nil, err
	}
	revokerUserID := mcontext.GetOpUserID(ctx)
	var flag bool

	if len(m.config.Share.IMAdminUser.UserIDs) > 0 {
		flag = datautil.Contain(revokerUserID, m.adminUserIDs...)
	}
	tips := sdkws.RevokeMsgTips{
		RevokerUserID:  revokerUserID,
		ClientMsgID:    msgs[0].ClientMsgID,
		RevokeTime:     now,
		Seq:            req.Seq,
		SesstionType:   msgs[0].SessionType,
		ConversationID: req.ConversationID,
		IsAdminRevoke:  flag,
	}
	var recvID string
	if msgs[0].SessionType == constant.ReadGroupChatType {
		recvID = msgs[0].GroupID
	} else {
		recvID = msgs[0].RecvID
	}
	m.notificationSender.NotificationWithSessionType(ctx, req.UserID, recvID, constant.MsgRevokeNotification, msgs[0].S
```

```
    m.webhookAfterRevokeMsg(ctx, &m.config.WebhooksConfig.AfterRevokeMsg, req)
    return &msg.RevokeMsgResp{}, nil
}
```

internal/rpc/msg/send.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "google.golang.org/protobuf/proto"

    "github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/open-im-server/v3/pkg/util/conversationutil"
    "github.com/openimsdk/protocol/constant"
    pbconv "github.com/openimsdk/protocol/conversation"
    pbmsg "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/protocol/wrapperspb"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/utils/datautil"
)

func (m *msgServer) SendMsg(ctx context.Context, req *pbmsg.SendMsgReq) (*pbmsg.SendMsgResp, error) {
    if req.MsgData == nil {
        return nil, errs.ErrArgs.WrapMsg("msgData is nil")
    }
    if err := authverify.CheckAccess(ctx, req.MsgData.SendID); err != nil {
        return nil, err
    }
    before := new(sdkws.MsgData)
    resp, err := m.sendMsg(ctx, req, before)
    if err != nil {
        return nil, err
    }
    if *before != nil && proto.Equal(*before, req.MsgData) == false {
        resp.Modify = req.MsgData
    }
    return resp, nil
}

func (m *msgServer) sendMsg(ctx context.Context, req *pbmsg.SendMsgReq, before **sdkws.MsgData) (*pbmsg.SendMsgResp,
    m.encapsulateMsgData(req.MsgData)
    switch req.MsgData.SessionType {
    case constant.SingleChatType:
        return m.sendMsgSingleChat(ctx, req, before)
    case constant.NotificationChatType:
        return m.sendMsgNotification(ctx, req, before)
    case constant.ReadGroupChatType:
        return m.sendMsgGroupChat(ctx, req, before)
    default:
```

```
		return nil, errs.ErrArgs.WrapMsg("unknown sessionType")
	}
}

func (m *msgServer) sendMsgGroupChat(ctx context.Context, req *pbmsg.SendMsgReq, before **sdkws.MsgData) (resp *pbms
	if err = m.messageVerification(ctx, req); err != nil {
		prommetrics.GroupChatMsgProcessFailedCounter.Inc()
		return nil, err
	}

	if err = m.webhookBeforeSendGroupMsg(ctx, &m.config.WebhooksConfig.BeforeSendGroupMsg, req); err != nil {
		return nil, err
	}
	if err := m.webhookBeforeMsgModify(ctx, &m.config.WebhooksConfig.BeforeMsgModify, req, before); err != nil {
		return nil, err
	}
	err = m.MsgDatabase.MsgToMQ(ctx, conversationutil.GenConversationUniqueKeyForGroup(req.MsgData.GroupID), req.MsgDat
	if err != nil {
		return nil, err
	}
	if req.MsgData.ContentType == constant.AtText {
		go m.setConversationAtInfo(ctx, req.MsgData)
	}

	m.webhookAfterSendGroupMsg(ctx, &m.config.WebhooksConfig.AfterSendGroupMsg, req)

	prommetrics.GroupChatMsgProcessSuccessCounter.Inc()
	resp = &pbmsg.SendMsgResp{}
	resp.SendTime = req.MsgData.SendTime
	resp.ServerMsgID = req.MsgData.ServerMsgID
	resp.ClientMsgID = req.MsgData.ClientMsgID
	return resp, nil
}

func (m *msgServer) setConversationAtInfo(nctx context.Context, msg *sdkws.MsgData) {

	log.ZDebug(nctx, "setConversationAtInfo", "msg", msg)

	defer func() {
		if r := recover(); r != nil {
			log.ZPanic(nctx, "setConversationAtInfo Panic", errs.ErrPanic(r))
		}
	}()

	ctx := mcontext.NewCtx("@@@" + mcontext.GetOperationID(nctx))

	var atUserID []string

	conversation := &pbconv.ConversationReq{
		ConversationID:   msgprocessor.GetConversationIDByMsg(msg),
		ConversationType: msg.SessionType,
		GroupID:          msg.GroupID,
	}
	memberUserIDList, err := m.GroupLocalCache.GetGroupMemberIDs(ctx, msg.GroupID)
	if err != nil {
		log.ZWarn(ctx, "GetGroupMemberIDs", err)
		return
	}

	tagAll := datautil.Contain(constant.AtAllString, msg.AtUserIDList...)
	if tagAll {

		memberUserIDList = datautil.DeleteElems(memberUserIDList, msg.SendID)

		atUserID = datautil.Single([]string{constant.AtAllString}, msg.AtUserIDList)
```

```
    if len(atUserID) == 0 { // just @everyone
        conversation.GroupAtType = &wrapperspb.Int32Value{Value: constant.AtAll}
    } else { // @Everyone and @other people
        conversation.GroupAtType = &wrapperspb.Int32Value{Value: constant.AtAllAtMe}
        atUserID = datautil.SliceIntersectFuncs(atUserID, memberUserIDList, func(a string) string { return a }, func(b s
            return b
        })
        if err := m.conversationClient.SetConversations(ctx, atUserID, conversation); err != nil {
            log.ZWarn(ctx, "SetConversations", err, "userID", atUserID, "conversation", conversation)
        }
        memberUserIDList = datautil.Single(atUserID, memberUserIDList)
    }

    conversation.GroupAtType = &wrapperspb.Int32Value{Value: constant.AtAll}
    if err := m.conversationClient.SetConversations(ctx, memberUserIDList, conversation); err != nil {
        log.ZWarn(ctx, "SetConversations", err, "userID", memberUserIDList, "conversation", conversation)
    }

    return
  }
  atUserID = datautil.SliceIntersectFuncs(msg.AtUserIDList, memberUserIDList, func(a string) string { return a }, fun
    return b
  })
  conversation.GroupAtType = &wrapperspb.Int32Value{Value: constant.AtMe}

  if err := m.conversationClient.SetConversations(ctx, atUserID, conversation); err != nil {
    log.ZWarn(ctx, "SetConversations", err, atUserID, conversation)
  }
}

func (m *msgServer) sendMsgNotification(ctx context.Context, req *pbmsg.SendMsgReq, before **sdkws.MsgData) (resp *p
  if err := m.MsgDatabase.MsgToMQ(ctx, conversationutil.GenConversationUniqueKeyForSingle(req.MsgData.SendID, req.Msg
    return nil, err
  }
  resp = &pbmsg.SendMsgResp{
    ServerMsgID: req.MsgData.ServerMsgID,
    ClientMsgID: req.MsgData.ClientMsgID,
    SendTime:    req.MsgData.SendTime,
  }
  return resp, nil
}

func (m *msgServer) sendMsgSingleChat(ctx context.Context, req *pbmsg.SendMsgReq, before **sdkws.MsgData) (resp *pbm
  if err := m.messageVerification(ctx, req); err != nil {
    return nil, err
  }
  isSend := true
  isNotification := msgprocessor.IsNotificationByMsg(req.MsgData)
  if !isNotification {
    isSend, err = m.modifyMessageByUserMessageReceiveOpt(authverify.WithTempAdmin(ctx), req.MsgData.RecvID, conversati
    if err != nil {
        return nil, err
    }
  }
  if !isSend {
    prommetrics.SingleChatMsgProcessFailedCounter.Inc()
    return nil, errs.ErrArgs.WrapMsg("message is not sent")
  } else {
    if err := m.webhookBeforeMsgModify(ctx, &m.config.WebhooksConfig.BeforeMsgModify, req, before); err != nil {
        return nil, err
    }
    if err := m.MsgDatabase.MsgToMQ(ctx, conversationutil.GenConversationUniqueKeyForSingle(req.MsgData.SendID, req.Ms
        prommetrics.SingleChatMsgProcessFailedCounter.Inc()
        return nil, err
    }
```

```
    m.webhookAfterSendSingleMsg(ctx, &m.config.WebhooksConfig.AfterSendSingleMsg, req)
    prommetrics.SingleChatMsgProcessSuccessCounter.Inc()
    return &pbmsg.SendMsgResp{
        ServerMsgID: req.MsgData.ServerMsgID,
        ClientMsgID: req.MsgData.ClientMsgID,
        SendTime:    req.MsgData.SendTime,
    }, nil
  }
}

func (m *msgServer) SendSimpleMsg(ctx context.Context, req *pbmsg.SendSimpleMsgReq) (*pbmsg.SendSimpleMsgResp, error
  if req.MsgData == nil {
    return nil, errs.ErrArgs.WrapMsg("msg data is nil")
  }
  sender, err := m.UserLocalCache.GetUserInfo(ctx, req.MsgData.SendID)
  if err != nil {
    return nil, err
  }
  req.MsgData.SenderFaceURL = sender.FaceURL
  req.MsgData.SenderNickname = sender.Nickname
  resp, err := m.SendMsg(ctx, &pbmsg.SendMsgReq{MsgData: req.MsgData})
  if err != nil {
    return nil, err
  }
  return &pbmsg.SendSimpleMsgResp{
    ServerMsgID: resp.ServerMsgID,
    ClientMsgID: resp.ClientMsgID,
    SendTime:    resp.SendTime,
    Modify:      resp.Modify,
  }, nil
}
```

internal/rpc/msg/seq.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"
    "errors"
    "sort"

    pbmsg "github.com/openimsdk/protocol/msg"
    "github.com/redis/go-redis/v9"
)

func (m *msgServer) GetConversationMaxSeq(ctx context.Context, req *pbmsg.GetConversationMaxSeqReq) (*pbmsg.GetConve
    maxSeq, err := m.MsgDatabase.GetMaxSeq(ctx, req.ConversationID)
    if err != nil && !errors.Is(err, redis.Nil) {
        return nil, err
    }
    return &pbmsg.GetConversationMaxSeqResp{MaxSeq: maxSeq}, nil
}

func (m *msgServer) GetMaxSeqs(ctx context.Context, req *pbmsg.GetMaxSeqsReq) (*pbmsg.SeqsInfoResp, error) {
    maxSeqs, err := m.MsgDatabase.GetMaxSeqs(ctx, req.ConversationIDs)
    if err != nil {
        return nil, err
    }
    return &pbmsg.SeqsInfoResp{MaxSeqs: maxSeqs}, nil
}

func (m *msgServer) GetHasReadSeqs(ctx context.Context, req *pbmsg.GetHasReadSeqsReq) (*pbmsg.SeqsInfoResp, error) {
    hasReadSeqs, err := m.MsgDatabase.GetHasReadSeqs(ctx, req.UserID, req.ConversationIDs)
    if err != nil {
        return nil, err
    }
    return &pbmsg.SeqsInfoResp{MaxSeqs: hasReadSeqs}, nil
}

func (m *msgServer) GetMsgByConversationIDs(ctx context.Context, req *pbmsg.GetMsgByConversationIDsReq) (*pbmsg.GetM
    Msgs, err := m.MsgDatabase.FindOneByDocIDs(ctx, req.ConversationIDs, req.MaxSeqs)
    if err != nil {
        return nil, err
    }
    return &pbmsg.GetMsgByConversationIDsResp{MsgDatas: Msgs}, nil
}

func (m *msgServer) SetUserConversationsMinSeq(ctx context.Context, req *pbmsg.SetUserConversationsMinSeqReq) (*pbms
    for _, userID := range req.UserIDs {
        if err := m.MsgDatabase.SetUserConversationsMinSeqs(ctx, userID, map[string]int64{req.ConversationID: req.Seq}); 
            return nil, err
        }
    }
    return &pbmsg.SetUserConversationsMinSeqResp{}, nil
```

```
}

func (m *msgServer) GetActiveConversation(ctx context.Context, req *pbmsg.GetActiveConversationReq) (*pbmsg.GetActiv
	res, err := m.MsgDatabase.GetCacheMaxSeqWithTime(ctx, req.ConversationIDs)
	if err != nil {
		return nil, err
	}
	conversations := make([]*pbmsg.ActiveConversation, 0, len(res))
	for conversationID, val := range res {
		conversations = append(conversations, &pbmsg.ActiveConversation{
			MaxSeq:         val.Seq,
			LastTime:       val.Time,
			ConversationID: conversationID,
		})
	}
	if req.Limit > 0 {
		sort.Sort(activeConversations(conversations))
		if len(conversations) > int(req.Limit) {
			conversations = conversations[:req.Limit]
		}
	}
	return &pbmsg.GetActiveConversationResp{Conversations: conversations}, nil
}

func (m *msgServer) SetUserConversationMaxSeq(ctx context.Context, req *pbmsg.SetUserConversationMaxSeqReq) (*pbmsg.
	for _, userID := range req.OwnerUserID {
		if err := m.MsgDatabase.SetUserConversationsMaxSeq(ctx, req.ConversationID, userID, req.MaxSeq); err != nil {
			return nil, err
		}
	}
	return &pbmsg.SetUserConversationMaxSeqResp{}, nil
}

func (m *msgServer) SetUserConversationMinSeq(ctx context.Context, req *pbmsg.SetUserConversationMinSeqReq) (*pbmsg.
	for _, userID := range req.OwnerUserID {
		if err := m.MsgDatabase.SetUserConversationsMinSeq(ctx, req.ConversationID, userID, req.MinSeq); err != nil {
			return nil, err
		}
	}
	return &pbmsg.SetUserConversationMinSeqResp{}, nil
}
```

internal/rpc/msg/server.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/mcache"
    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/mqbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "google.golang.org/grpc"

    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/controller"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
    "github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
    "github.com/openimsdk/open-im-server/v3/pkg/notification"
    "github.com/openimsdk/open-im-server/v3/pkg/rpccache"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/conversation"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/discovery"
)

type MessageInterceptorFunc func(ctx context.Context, globalConfig *Config, req *msg.SendMsgReq) (*sdkws.MsgData, er

// MessageInterceptorChain defines a chain of message interceptor functions.
type MessageInterceptorChain []MessageInterceptorFunc

type Config struct {
    RpcConfig          config.Msg
    RedisConfig        config.Redis
    MongodbConfig      config.Mongo
    KafkaConfig        config.Kafka
    NotificationConfig config.Notification
    Share              config.Share
    WebhooksConfig     config.Webhooks
    LocalCacheConfig   config.LocalCache
    Discovery          config.Discovery
}

// MsgServer encapsulates dependencies required for message handling.
type msgServer struct {
    msg.UnimplementedMsgServer
    RegisterCenter         discovery.Conn                  // Service discovery registry for service registration.
    MsgDatabase            controller.CommonMsgDatabase    // Interface for message database operations.
    UserLocalCache         *rpccache.UserLocalCache        // Local cache for user data.
    FriendLocalCache       *rpccache.FriendLocalCache      // Local cache for friend data.
```

```
	GroupLocalCache        *rpccache.GroupLocalCache        // Local cache for group data.
	ConversationLocalCache *rpccache.ConversationLocalCache // Local cache for conversation data.
	Handlers               MessageInterceptorChain          // Chain of handlers for processing messages.
	notificationSender     *notification.NotificationSender // RPC client for sending notifications.
	msgNotificationSender  *MsgNotificationSender           // RPC client for sending msg notifications.
	config                 *Config                          // Global configuration settings.
	webhookClient          *webhook.Client
	conversationClient     *rpcli.ConversationClient

	adminUserIDs []string
}

func (m *msgServer) addInterceptorHandler(interceptorFunc ...MessageInterceptorFunc) {
	m.Handlers = append(m.Handlers, interceptorFunc...)

}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar)
	builder := mqbuild.NewBuilder(&config.KafkaConfig)
	redisProducer, err := builder.GetTopicProducer(ctx, config.KafkaConfig.ToRedisTopic)
	if err != nil {
		return err
	}
	dbb := dbbuild.NewBuilder(&config.MongodbConfig, &config.RedisConfig)
	mgocli, err := dbb.Mongo(ctx)
	if err != nil {
		return err
	}
	rdb, err := dbb.Redis(ctx)
	if err != nil {
		return err
	}
	msgDocModel, err := mgo.NewMsgMongo(mgocli.GetDB())
	if err != nil {
		return err
	}
	var msgModel cache.MsgCache
	if rdb == nil {
		cm, err := mgo.NewCacheMgo(mgocli.GetDB())
		if err != nil {
			return err
		}
		msgModel = mcache.NewMsgCache(cm, msgDocModel)
	} else {
		msgModel = redis.NewMsgCache(rdb, msgDocModel)
	}
	seqConversation, err := mgo.NewSeqConversationMongo(mgocli.GetDB())
	if err != nil {
		return err
	}
	seqConversationCache := redis.NewSeqConversationCacheRedis(rdb, seqConversation)
	seqUser, err := mgo.NewSeqUserMongo(mgocli.GetDB())
	if err != nil {
		return err
	}
	seqUserCache := redis.NewSeqUserCacheRedis(rdb, seqUser)
	userConn, err := client.GetConn(ctx, config.Discovery.RpcService.User)
	if err != nil {
		return err
	}
	groupConn, err := client.GetConn(ctx, config.Discovery.RpcService.Group)
	if err != nil {
		return err
	}
	friendConn, err := client.GetConn(ctx, config.Discovery.RpcService.Friend)
	if err != nil {
```

```
    return err
  }
  conversationConn, err := client.GetConn(ctx, config.Discovery.RpcService.Conversation)
  if err != nil {
    return err
  }
  conversationClient := rpcli.NewConversationClient(conversationConn)
  msgDatabase := controller.NewCommonMsgDatabase(msgDocModel, msgModel, seqUserCache, seqConversationCache, redisProd
  s := &msgServer{
    MsgDatabase:            msgDatabase,
    RegisterCenter:         client,
    UserLocalCache:         rpccache.NewUserLocalCache(rpcli.NewUserClient(userConn), &config.LocalCacheConfig, rdb),
    GroupLocalCache:        rpccache.NewGroupLocalCache(rpcli.NewGroupClient(groupConn), &config.LocalCacheConfig, rd
    ConversationLocalCache: rpccache.NewConversationLocalCache(conversationClient, &config.LocalCacheConfig, rdb),
    FriendLocalCache:       rpccache.NewFriendLocalCache(rpcli.NewRelationClient(friendConn), &config.LocalCacheConfi
    config:                 config,
    webhookClient:          webhook.NewWebhookClient(config.WebhooksConfig.URL),
    conversationClient:     conversationClient,
    adminUserIDs:           config.Share.IMAdminUser.UserIDs,
  }

  s.notificationSender = notification.NewNotificationSender(&config.NotificationConfig, notification.WithLocalSendMsg
  s.msgNotificationSender = NewMsgNotificationSender(config, notification.WithLocalSendMsg(s.SendMsg))

  msg.RegisterMsgServer(server, s)

  return nil
}

func (m *msgServer) conversationAndGetRecvID(conversation *conversation.Conversation, userID string) string {
  if conversation.ConversationType == constant.SingleChatType ||
    conversation.ConversationType == constant.NotificationChatType {
    if userID == conversation.OwnerUserID {
      return conversation.UserID
    } else {
      return conversation.OwnerUserID
    }
  } else if conversation.ConversationType == constant.ReadGroupChatType {
    return conversation.GroupID
  }
  return ""
}
```

internal/rpc/msg/statistics.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/common/storage/model"

    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/utils/datautil"
)

func (m *msgServer) GetActiveUser(ctx context.Context, req *msg.GetActiveUserReq) (*msg.GetActiveUserResp, error) {
    if err := authverify.CheckAdmin(ctx); err != nil {
        return nil, err
    }
    msgCount, userCount, users, dateCount, err := m.MsgDatabase.RangeUserSendCount(ctx, time.UnixMilli(req.Start), time
    if err != nil {
        return nil, err
    }
    var pbUsers []*msg.ActiveUser
    if len(users) > 0 {
        userIDs := datautil.Slice(users, func(e *model.UserCount) string { return e.UserID })
        userMap, err := m.UserLocalCache.GetUsersInfoMap(ctx, userIDs)
        if err != nil {
            return nil, err
        }
        pbUsers = make([]*msg.ActiveUser, 0, len(users))
        for _, user := range users {
            pbUser := userMap[user.UserID]
            if pbUser == nil {
                pbUser = &sdkws.UserInfo{
                    UserID:   user.UserID,
                    Nickname: user.UserID,
                }
            }
            pbUsers = append(pbUsers, &msg.ActiveUser{
                User:  pbUser,
                Count: user.Count,
            })
        }
    }
    return &msg.GetActiveUserResp{
        MsgCount:  msgCount,
        UserCount: userCount,
        DateCount: dateCount,
        Users:     pbUsers,
    }, nil
```

```
}

func (m *msgServer) GetActiveGroup(ctx context.Context, req *msg.GetActiveGroupReq) (*msg.GetActiveGroupResp, error)
  if err := authverify.CheckAdmin(ctx); err != nil {
    return nil, err
  }
  msgCount, groupCount, groups, dateCount, err := m.MsgDatabase.RangeGroupSendCount(ctx, time.UnixMilli(req.Start), t
  if err != nil {
    return nil, err
  }
  var pbgroups []*msg.ActiveGroup
  if len(groups) > 0 {
    groupIDs := datautil.Slice(groups, func(e *model.GroupCount) string { return e.GroupID })
    resp, err := m.GroupLocalCache.GetGroupInfos(ctx, groupIDs)
    if err != nil {
      return nil, err
    }
    groupMap := make(map[string]*sdkws.GroupInfo, len(groups))
    for i, group := range groups {
      groupMap[group.GroupID] = resp[i]
    }
    pbgroups = make([]*msg.ActiveGroup, 0, len(groups))
    for _, group := range groups {
      pbgroup := groupMap[group.GroupID]
      if pbgroup == nil {
        pbgroup = &sdkws.GroupInfo{
          GroupID:   group.GroupID,
          GroupName: group.GroupID,
        }
      }
      pbgroups = append(pbgroups, &msg.ActiveGroup{
        Group: pbgroup,
        Count: group.Count,
      })
    }
  }
  return &msg.GetActiveGroupResp{
    MsgCount:   msgCount,
    GroupCount: groupCount,
    DateCount:  dateCount,
    Groups:     pbgroups,
  }, nil
}
```

internal/rpc/msg/sync_msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
    "github.com/openimsdk/open-im-server/v3/pkg/util/conversationutil"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/datautil"
    "github.com/openimsdk/tools/utils/timeutil"
)

func (m *msgServer) PullMessageBySeqs(ctx context.Context, req *sdkws.PullMessageBySeqsReq) (*sdkws.PullMessageBySeq
    if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
        return nil, err
    }
    resp := &sdkws.PullMessageBySeqsResp{}
    resp.Msgs = make(map[string]*sdkws.PullMsgs)
    resp.NotificationMsgs = make(map[string]*sdkws.PullMsgs)
    for _, seq := range req.SeqRanges {
        if !msgprocessor.IsNotification(seq.ConversationID) {
            conversation, err := m.ConversationLocalCache.GetConversation(ctx, req.UserID, seq.ConversationID)
            if err != nil {
                log.ZError(ctx, "GetConversation error", err, "conversationID", seq.ConversationID)
                continue
            }
            minSeq, maxSeq, msgs, err := m.MsgDatabase.GetMsgBySeqsRange(ctx, req.UserID, seq.ConversationID,
                seq.Begin, seq.End, seq.Num, conversation.MaxSeq)
            if err != nil {
                log.ZWarn(ctx, "GetMsgBySeqsRange error", err, "conversationID", seq.ConversationID, "seq", seq)
                continue
            }
            var isEnd bool
            switch req.Order {
            case sdkws.PullOrder_PullOrderAsc:
                isEnd = maxSeq <= seq.End
            case sdkws.PullOrder_PullOrderDesc:
                isEnd = seq.Begin <= minSeq
            }
            if len(msgs) == 0 {
                log.ZWarn(ctx, "not have msgs", nil, "conversationID", seq.ConversationID, "seq", seq)
                continue
            }
            resp.Msgs[seq.ConversationID] = &sdkws.PullMsgs{Msgs: msgs, IsEnd: isEnd}
        } else {
            var seqs []int64
```

```
        for i := seq.Begin; i <= seq.End; i++ {
            seqs = append(seqs, i)
        }
        minSeq, maxSeq, notificationMsgs, err := m.MsgDatabase.GetMsgBySeqs(ctx, req.UserID, seq.ConversationID, seqs)
        if err != nil {
            log.ZWarn(ctx, "GetMsgBySeqs error", err, "conversationID", seq.ConversationID, "seq", seq)

            continue
        }
        var isEnd bool
        switch req.Order {
        case sdkws.PullOrder_PullOrderAsc:
            isEnd = maxSeq <= seq.End
        case sdkws.PullOrder_PullOrderDesc:
            isEnd = seq.Begin <= minSeq
        }
        if len(notificationMsgs) == 0 {
            log.ZWarn(ctx, "not have notificationMsgs", nil, "conversationID", seq.ConversationID, "seq", seq)

            continue
        }
        resp.NotificationMsgs[seq.ConversationID] = &sdkws.PullMsgs{Msgs: notificationMsgs, IsEnd: isEnd}
    }
    return resp, nil
}

func (m *msgServer) GetSeqMessage(ctx context.Context, req *msg.GetSeqMessageReq) (*msg.GetSeqMessageResp, error) {
    resp := &msg.GetSeqMessageResp{
        Msgs:             make(map[string]*sdkws.PullMsgs),
        NotificationMsgs: make(map[string]*sdkws.PullMsgs),
    }
    for _, conv := range req.Conversations {
        isEnd, endSeq, msgs, err := m.MsgDatabase.GetMessagesBySeqWithBounds(ctx, req.UserID, conv.ConversationID, conv.S
        if err != nil {
            return nil, err
        }
        var pullMsgs *sdkws.PullMsgs
        if ok := false; conversationutil.IsNotificationConversationID(conv.ConversationID) {
            pullMsgs, ok = resp.NotificationMsgs[conv.ConversationID]
            if !ok {
                pullMsgs = &sdkws.PullMsgs{}
                resp.NotificationMsgs[conv.ConversationID] = pullMsgs
            }
        } else {
            pullMsgs, ok = resp.Msgs[conv.ConversationID]
            if !ok {
                pullMsgs = &sdkws.PullMsgs{}
                resp.Msgs[conv.ConversationID] = pullMsgs
            }
        }
        pullMsgs.Msgs = append(pullMsgs.Msgs, msgs...)
        pullMsgs.IsEnd = isEnd
        pullMsgs.EndSeq = endSeq
    }
    return resp, nil
}

func (m *msgServer) GetMaxSeq(ctx context.Context, req *sdkws.GetMaxSeqReq) (*sdkws.GetMaxSeqResp, error) {
    if err := authverify.CheckAccess(ctx, req.UserID); err != nil {
        return nil, err
    }
    conversationIDs, err := m.ConversationLocalCache.GetConversationIDs(ctx, req.UserID)
    if err != nil {
        return nil, err
    }
```

```
	for _, conversationID := range conversationIDs {
		conversationIDs = append(conversationIDs, conversationutil.GetNotificationConversationIDByConversationID(conversa
	}
	conversationIDs = append(conversationIDs, conversationutil.GetSelfNotificationConversationID(req.UserID))
	log.ZDebug(ctx, "GetMaxSeq", "conversationIDs", conversationIDs)
	maxSeqs, err := m.MsgDatabase.GetMaxSeqs(ctx, conversationIDs)
	if err != nil {
		log.ZWarn(ctx, "GetMaxSeqs error", err, "conversationIDs", conversationIDs, "maxSeqs", maxSeqs)
		return nil, err
	}
	// avoid pulling messages from sessions with a large number of max seq values of 0
	for conversationID, seq := range maxSeqs {
		if seq == 0 {
			delete(maxSeqs, conversationID)
		}
	}
	resp := new(sdkws.GetMaxSeqResp)
	resp.MaxSeqs = maxSeqs
	return resp, nil
}

func (m *msgServer) SearchMessage(ctx context.Context, req *msg.SearchMessageReq) (resp *msg.SearchMessageResp, err
	// var chatLogs []*sdkws.MsgData
	var chatLogs []*msg.SearchedMsgData
	var total int64
	resp = &msg.SearchMessageResp{}
	if total, chatLogs, err = m.MsgDatabase.SearchMessage(ctx, req); err != nil {
		return nil, err
	}

	var (
		sendIDs  []string
		recvIDs  []string
		groupIDs []string
		sendMap  = make(map[string]string)
		recvMap  = make(map[string]string)
		groupMap = make(map[string]*sdkws.GroupInfo)
	)

	for _, chatLog := range chatLogs {
		if chatLog.MsgData.SenderNickname == "" {
			sendIDs = append(sendIDs, chatLog.MsgData.SendID)
		}
		switch chatLog.MsgData.SessionType {
		case constant.SingleChatType, constant.NotificationChatType:
			recvIDs = append(recvIDs, chatLog.MsgData.RecvID)
		case constant.WriteGroupChatType, constant.ReadGroupChatType:
			groupIDs = append(groupIDs, chatLog.MsgData.GroupID)
		}
	}

	// Retrieve sender and receiver information
	if len(sendIDs) != 0 {
		sendInfos, err := m.UserLocalCache.GetUsersInfo(ctx, sendIDs)
		if err != nil {
			return nil, err
		}
		for _, sendInfo := range sendInfos {
			sendMap[sendInfo.UserID] = sendInfo.Nickname
		}
	}

	if len(recvIDs) != 0 {
		recvInfos, err := m.UserLocalCache.GetUsersInfo(ctx, recvIDs)
		if err != nil {
			return nil, err
```

```
    }
    for _, recvInfo := range recvInfos {
        recvMap[recvInfo.UserID] = recvInfo.Nickname
    }
}

// Retrieve group information including member counts
if len(groupIDs) != 0 {
    groupInfos, err := m.GroupLocalCache.GetGroupInfos(ctx, groupIDs)
    if err != nil {
        return nil, err
    }
    for _, groupInfo := range groupInfos {
        groupMap[groupInfo.GroupID] = groupInfo
        // Get actual member count
        memberIDs, err := m.GroupLocalCache.GetGroupMemberIDs(ctx, groupInfo.GroupID)
        if err == nil {
            groupInfo.MemberCount = uint32(len(memberIDs)) // Update the member count with actual number
        }
    }
}

// Construct response with updated information
for _, chatLog := range chatLogs {
    pbchatLog := &msg.ChatLog{}
    datautil.CopyStructFields(pbchatLog, chatLog.MsgData)
    pbchatLog.SendTime = chatLog.MsgData.SendTime
    pbchatLog.CreateTime = chatLog.MsgData.CreateTime
    if chatLog.MsgData.SenderNickname == "" {
        pbchatLog.SenderNickname = sendMap[chatLog.MsgData.SendID]
    }
    switch chatLog.MsgData.SessionType {
    case constant.SingleChatType, constant.NotificationChatType:
        pbchatLog.RecvNickname = recvMap[chatLog.MsgData.RecvID]
    case constant.ReadGroupChatType:
        groupInfo := groupMap[chatLog.MsgData.GroupID]
        pbchatLog.SenderFaceURL = groupInfo.FaceURL
        pbchatLog.GroupMemberCount = groupInfo.MemberCount // Reflects actual member count
        pbchatLog.RecvID = groupInfo.GroupID
        pbchatLog.GroupName = groupInfo.GroupName
        pbchatLog.GroupOwner = groupInfo.OwnerUserID
        pbchatLog.GroupType = groupInfo.GroupType
    }
    searchChatLog := &msg.SearchChatLog{ChatLog: pbchatLog, IsRevoked: chatLog.IsRevoked}

    resp.ChatLogs = append(resp.ChatLogs, searchChatLog)
}
resp.ChatLogsNum = int32(total)
return resp, nil
}

func (m *msgServer) GetServerTime(ctx context.Context, _ *msg.GetServerTimeReq) (*msg.GetServerTimeResp, error) {
    return &msg.GetServerTimeResp{ServerTime: timeutil.GetCurrentTimestampByMill()}, nil
}

func (m *msgServer) GetLastMessage(ctx context.Context, req *msg.GetLastMessageReq) (*msg.GetLastMessageResp, error)
    msgs, err := m.MsgDatabase.GetLastMessage(ctx, req.ConversationIDs, req.UserID)
    if err != nil {
        return nil, err
    }
    return &msg.GetLastMessageResp{Msgs: msgs}, nil
}
```

internal/rpc/msg/utils.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/tools/errs"
    "github.com/redis/go-redis/v9"
    "go.mongodb.org/mongo-driver/mongo"
)

func IsNotFound(err error) bool {
    switch errs.Unwrap(err) {
    case redis.Nil, mongo.ErrNoDocuments:
        return true
    default:
        return false
    }
}

type activeConversations []*msg.ActiveConversation

func (s activeConversations) Len() int {
    return len(s)
}

func (s activeConversations) Less(i, j int) bool {
    return s[i].LastTime > s[j].LastTime
}

func (s activeConversations) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}

//type seqTime struct {
//    ConversationID string
//    Seq            int64
//    Time           int64
//    Unread         int64
//    Pinned         bool
//}
//
//func (s seqTime) String() string {
//    return fmt.Sprintf("<Time_%d,Unread_%d,Pinned_%t>", s.Time, s.Unread, s.Pinned)
//}
//
//type seqTimes []seqTime
//
//func (s seqTimes) Len() int {
//    return len(s)
//}
//
```

```
//// Less sticky priority, unread priority, time descending
//func (s seqTimes) Less(i, j int) bool {
//■iv, jv := s[i], s[j]
//■if iv.Pinned && (!jv.Pinned) {
//■■return true
//■}
//■if jv.Pinned && (!iv.Pinned) {
//■■return false
//■}
//■if iv.Unread > 0 && jv.Unread == 0 {
//■■return true
//■}
//■if jv.Unread > 0 && iv.Unread == 0 {
//■■return false
//■}
//■return iv.Time > jv.Time
//}
//
//func (s seqTimes) Swap(i, j int) {
//■s[i], s[j] = s[j], s[i]
//}
//
//type conversationStatus struct {
//■ConversationID string
//■Pinned         bool
//■Recv           bool
//}
```

internal/rpc/msg/verify.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msg

import (
    "context"
    "math/rand"
    "strconv"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/tools/utils/datautil"
    "github.com/openimsdk/tools/utils/encrypt"
    "github.com/openimsdk/tools/utils/timeutil"

    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/errs"
)

var ExcludeContentType = []int{constant.HasReadReceipt}

type Validator interface {
    validate(pb *msg.SendMsgReq) (bool, int32, string)
}

type MessageRevoked struct {
    RevokerID                   string `json:"revokerID"`
    RevokerRole                 int32  `json:"revokerRole"`
    ClientMsgID                 string `json:"clientMsgID"`
    RevokerNickname             string `json:"revokerNickname"`
    RevokeTime                  int64  `json:"revokeTime"`
    SourceMessageSendTime       int64  `json:"sourceMessageSendTime"`
    SourceMessageSendID         string `json:"sourceMessageSendID"`
    SourceMessageSenderNickname string `json:"sourceMessageSenderNickname"`
    SessionType                 int32  `json:"sessionType"`
    Seq                         uint32 `json:"seq"`
}

func (m *msgServer) messageVerification(ctx context.Context, data *msg.SendMsgReq) error {
    switch data.MsgData.SessionType {
    case constant.SingleChatType:
        if datautil.Contain(data.MsgData.SendID, m.adminUserIDs...) {
            return nil
        }
        if data.MsgData.ContentType <= constant.NotificationEnd &&
            data.MsgData.ContentType >= constant.NotificationBegin {
            return nil
        }
        if err := m.webhookBeforeSendSingleMsg(ctx, &m.config.WebhooksConfig.BeforeSendSingleMsg, data); err != nil {
```

```
        return err
    }
    u, err := m.UserLocalCache.GetUserInfo(ctx, data.MsgData.SendID)
    if err != nil {
        return err
    }
    if authverify.CheckSystemAccount(ctx, u.AppMangerLevel) {
        return nil
    }
    black, err := m.FriendLocalCache.IsBlack(ctx, data.MsgData.SendID, data.MsgData.RecvID)
    if err != nil {
        return err
    }
    if black {
        return servererrs.ErrBlockedByPeer.Wrap()
    }
    if m.config.RpcConfig.FriendVerify {
        friend, err := m.FriendLocalCache.IsFriend(ctx, data.MsgData.SendID, data.MsgData.RecvID)
        if err != nil {
            return err
        }
        if !friend {
            return servererrs.ErrNotPeersFriend.Wrap()
        }
        return nil
    }
    return nil
case constant.ReadGroupChatType:
    groupInfo, err := m.GroupLocalCache.GetGroupInfo(ctx, data.MsgData.GroupID)
    if err != nil {
        return err
    }
    if groupInfo.Status == constant.GroupStatusDismissed &&
        data.MsgData.ContentType != constant.GroupDismissedNotification {
        return servererrs.ErrDismissedAlready.Wrap()
    }
    if groupInfo.GroupType == constant.SuperGroup {
        return nil
    }

    if datautil.Contain(data.MsgData.SendID, m.adminUserIDs...) {
        return nil
    }
    if data.MsgData.ContentType <= constant.NotificationEnd &&
        data.MsgData.ContentType >= constant.NotificationBegin {
        return nil
    }
    memberIDs, err := m.GroupLocalCache.GetGroupMemberIDMap(ctx, data.MsgData.GroupID)
    if err != nil {
        return err
    }
    if _, ok := memberIDs[data.MsgData.SendID]; !ok {
        return servererrs.ErrNotInGroupYet.Wrap()
    }

    groupMemberInfo, err := m.GroupLocalCache.GetGroupMember(ctx, data.MsgData.GroupID, data.MsgData.SendID)
    if err != nil {
        if errs.ErrRecordNotFound.Is(err) {
            return servererrs.ErrNotInGroupYet.WrapMsg(err.Error())
        }
        return err
    }
    if groupMemberInfo.RoleLevel == constant.GroupOwner {
        return nil
    } else {
        if groupMemberInfo.MuteEndTime >= time.Now().UnixMilli() {
```

```
        return servererrs.ErrMutedInGroup.Wrap()
    }
    if groupInfo.Status == constant.GroupStatusMuted && groupMemberInfo.RoleLevel != constant.GroupAdmin {
        return servererrs.ErrMutedGroup.Wrap()
    }
  }
  return nil
 default:
  return nil
 }
}

func (m *msgServer) encapsulateMsgData(msg *sdkws.MsgData) {
 msg.ServerMsgID = GetMsgID(msg.SendID)
 if msg.SendTime == 0 {
  msg.SendTime = timeutil.GetCurrentTimestampByMill()
 }
 switch msg.ContentType {
 case constant.Text, constant.Picture, constant.Voice, constant.Video,
  constant.File, constant.AtText, constant.Merger, constant.Card,
  constant.Location, constant.Custom, constant.Quote, constant.AdvancedText, constant.MarkdownText:
 case constant.Revoke:
  datautil.SetSwitchFromOptions(msg.Options, constant.IsUnreadCount, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsOfflinePush, false)
 case constant.HasReadReceipt:
  datautil.SetSwitchFromOptions(msg.Options, constant.IsConversationUpdate, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsSenderConversationUpdate, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsUnreadCount, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsOfflinePush, false)
 case constant.Typing:
  datautil.SetSwitchFromOptions(msg.Options, constant.IsHistory, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsPersistent, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsSenderSync, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsConversationUpdate, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsSenderConversationUpdate, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsUnreadCount, false)
  datautil.SetSwitchFromOptions(msg.Options, constant.IsOfflinePush, false)
 }
}

func GetMsgID(sendID string) string {
 t := timeutil.GetCurrentTimeFormatted()
 return encrypt.Md5(t + "-" + sendID + "-" + strconv.Itoa(rand.Int()))
}

func (m *msgServer) modifyMessageByUserMessageReceiveOpt(ctx context.Context, userID, conversationID string, session
 opt, err := m.UserLocalCache.GetUserGlobalMsgRecvOpt(ctx, userID)
 if err != nil {
  return false, err
 }
 switch opt {
 case constant.ReceiveMessage:
 case constant.NotReceiveMessage:
  return false, nil
 case constant.ReceiveNotNotifyMessage:
  if pb.MsgData.Options == nil {
   pb.MsgData.Options = make(map[string]bool, 10)
  }
  datautil.SetSwitchFromOptions(pb.MsgData.Options, constant.IsOfflinePush, false)
  return true, nil
 }
 singleOpt, err := m.ConversationLocalCache.GetSingleConversationRecvMsgOpt(ctx, userID, conversationID)
 if errs.ErrRecordNotFound.Is(err) {
  return true, nil
 } else if err != nil {
  return false, err
```

```
	}
	switch singleOpt {
	case constant.ReceiveMessage:
		return true, nil
	case constant.NotReceiveMessage:
		if datautil.Contain(int(pb.MsgData.ContentType), ExcludeContentType...) {
			return true, nil
		}
		return false, nil
	case constant.ReceiveNotNotifyMessage:
		if pb.MsgData.Options == nil {
			pb.MsgData.Options = make(map[string]bool, 10)
		}
		datautil.SetSwitchFromOptions(pb.MsgData.Options, constant.IsOfflinePush, false)
		return true, nil
	}
	return true, nil
}
```

internal/api

internal/api/auth.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
	"github.com/gin-gonic/gin"
	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/tools/a2r"
)

type AuthApi struct {
	Client auth.AuthClient
}

func NewAuthApi(client auth.AuthClient) AuthApi {
	return AuthApi{client}
}

func (o *AuthApi) GetAdminToken(c *gin.Context) {
	a2r.Call(c, auth.AuthClient.GetAdminToken, o.Client)
}

func (o *AuthApi) GetUserToken(c *gin.Context) {
	a2r.Call(c, auth.AuthClient.GetUserToken, o.Client)
}

func (o *AuthApi) ParseToken(c *gin.Context) {
	a2r.Call(c, auth.AuthClient.ParseToken, o.Client)
}

func (o *AuthApi) ForceLogout(c *gin.Context) {
	a2r.Call(c, auth.AuthClient.ForceLogout, o.Client)
}
```

internal/api/config_manager.go

```
package api

import (
	"encoding/json"
	"reflect"
	"strconv"
	"time"

	"github.com/gin-gonic/gin"
	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/discovery/etcd"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/runtimeenv"
	clientv3 "go.etcd.io/etcd/client/v3"
)

const (
	// wait for Restart http call return
	waitHttp = time.Millisecond * 200
)

type ConfigManager struct {
	imAdminUserID []string
	config        *config.AllConfig
	client        *clientv3.Client

	configPath string
}

func NewConfigManager(IMAdminUserID []string, cfg *config.AllConfig, client *clientv3.Client, configPath string) *Co
	cm := &ConfigManager{
		imAdminUserID: IMAdminUserID,
		config:        cfg,
		client:        client,
		configPath:    configPath,
	}
	return cm
}

func (cm *ConfigManager) CheckAdmin(c *gin.Context) {
	if err := authverify.CheckAdmin(c); err != nil {
		apiresp.GinError(c, err)
		c.Abort()
	}
}

func (cm *ConfigManager) GetConfig(c *gin.Context) {
	var req apistruct.GetConfigReq
	if err := c.BindJSON(&req); err != nil {
		apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
		return
	}
	conf := cm.config.Name2Config(req.ConfigName)
	if conf == nil {
		apiresp.GinError(c, errs.ErrArgs.WithDetail("config name not found").Wrap())
		return
	}
	b, err := json.Marshal(conf)
```

```
	if err != nil {
		apiresp.GinError(c, err)
		return
	}
	apiresp.GinSuccess(c, string(b))
}

func (cm *ConfigManager) GetConfigList(c *gin.Context) {
	var resp apistruct.GetConfigListResp
	resp.ConfigNames = cm.config.GetConfigNames()
	resp.Environment = runtimeenv.RuntimeEnvironment()
	resp.Version = version.Version

	apiresp.GinSuccess(c, resp)
}

func (cm *ConfigManager) SetConfig(c *gin.Context) {
	if cm.config.Discovery.Enable != config.ETCD {
		apiresp.GinError(c, errs.New("only etcd support set config").Wrap())
		return
	}
	var req apistruct.SetConfigReq
	if err := c.BindJSON(&req); err != nil {
		apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
		return
	}
	var err error
	switch req.ConfigName {
	case cm.config.Discovery.GetConfigFileName():
		err = compareAndSave[config.Discovery](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Kafka.GetConfigFileName():
		err = compareAndSave[config.Kafka](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.LocalCache.GetConfigFileName():
		err = compareAndSave[config.LocalCache](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Log.GetConfigFileName():
		err = compareAndSave[config.Log](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Minio.GetConfigFileName():
		err = compareAndSave[config.Minio](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Mongo.GetConfigFileName():
		err = compareAndSave[config.Mongo](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Notification.GetConfigFileName():
		err = compareAndSave[config.Notification](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.API.GetConfigFileName():
		err = compareAndSave[config.API](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.CronTask.GetConfigFileName():
		err = compareAndSave[config.CronTask](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.MsgGateway.GetConfigFileName():
		err = compareAndSave[config.MsgGateway](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.MsgTransfer.GetConfigFileName():
		err = compareAndSave[config.MsgTransfer](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Push.GetConfigFileName():
		err = compareAndSave[config.Push](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Auth.GetConfigFileName():
		err = compareAndSave[config.Auth](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Conversation.GetConfigFileName():
		err = compareAndSave[config.Conversation](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Friend.GetConfigFileName():
		err = compareAndSave[config.Friend](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Group.GetConfigFileName():
		err = compareAndSave[config.Group](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Msg.GetConfigFileName():
		err = compareAndSave[config.Msg](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Third.GetConfigFileName():
		err = compareAndSave[config.Third](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.User.GetConfigFileName():
		err = compareAndSave[config.User](c, cm.config.Name2Config(req.ConfigName), &req, cm)
```

```
	case cm.config.Redis.GetConfigFileName():
		err = compareAndSave[config.Redis](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Share.GetConfigFileName():
		err = compareAndSave[config.Share](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	case cm.config.Webhooks.GetConfigFileName():
		err = compareAndSave[config.Webhooks](c, cm.config.Name2Config(req.ConfigName), &req, cm)
	default:
		apiresp.GinError(c, errs.ErrArgs.Wrap())
		return
	}
	if err != nil {
		apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
		return
	}
	apiresp.GinSuccess(c, nil)
}

func (cm *ConfigManager) SetConfigs(c *gin.Context) {
	if cm.config.Discovery.Enable != config.ETCD {
		apiresp.GinError(c, errs.New("only etcd support set config").Wrap())
		return
	}
	var req apistruct.SetConfigsReq
	if err := c.BindJSON(&req); err != nil {
		apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
		return
	}
	var (
		err error
		ops []*clientv3.Op
	)

	for _, cf := range req.Configs {
		var op *clientv3.Op
		switch cf.ConfigName {
		case cm.config.Discovery.GetConfigFileName():
			op, err = compareAndOp[config.Discovery](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Kafka.GetConfigFileName():
			op, err = compareAndOp[config.Kafka](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.LocalCache.GetConfigFileName():
			op, err = compareAndOp[config.LocalCache](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Log.GetConfigFileName():
			op, err = compareAndOp[config.Log](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Minio.GetConfigFileName():
			op, err = compareAndOp[config.Minio](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Mongo.GetConfigFileName():
			op, err = compareAndOp[config.Mongo](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Notification.GetConfigFileName():
			op, err = compareAndOp[config.Notification](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.API.GetConfigFileName():
			op, err = compareAndOp[config.API](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.CronTask.GetConfigFileName():
			op, err = compareAndOp[config.CronTask](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.MsgGateway.GetConfigFileName():
			op, err = compareAndOp[config.MsgGateway](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.MsgTransfer.GetConfigFileName():
			op, err = compareAndOp[config.MsgTransfer](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Push.GetConfigFileName():
			op, err = compareAndOp[config.Push](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Auth.GetConfigFileName():
			op, err = compareAndOp[config.Auth](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Conversation.GetConfigFileName():
			op, err = compareAndOp[config.Conversation](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Friend.GetConfigFileName():
			op, err = compareAndOp[config.Friend](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
		case cm.config.Group.GetConfigFileName():
```

```
        op, err = compareAndOp[config.Group](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    case cm.config.Msg.GetConfigFileName():
        op, err = compareAndOp[config.Msg](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    case cm.config.Third.GetConfigFileName():
        op, err = compareAndOp[config.Third](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    case cm.config.User.GetConfigFileName():
        op, err = compareAndOp[config.User](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    case cm.config.Redis.GetConfigFileName():
        op, err = compareAndOp[config.Redis](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    case cm.config.Share.GetConfigFileName():
        op, err = compareAndOp[config.Share](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    case cm.config.Webhooks.GetConfigFileName():
        op, err = compareAndOp[config.Webhooks](c, cm.config.Name2Config(cf.ConfigName), &cf, cm)
    default:
        apiresp.GinError(c, errs.ErrArgs.Wrap())
        return
    }
    if err != nil {
        apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
        return
    }
    if op != nil {
        ops = append(ops, op)
    }
  }
  if len(ops) > 0 {
    tx := cm.client.Txn(c)
    if _, err = tx.Then(datautil.Batch(func(op *clientv3.Op) clientv3.Op { return *op }, ops)...).Commit(); err != ni
        apiresp.GinError(c, errs.WrapMsg(err, "save to etcd failed"))
        return
    }

  }

  apiresp.GinSuccess(c, nil)
}

func compareAndOp[T any](c *gin.Context, old any, req *apistruct.SetConfigReq, cm *ConfigManager) (*clientv3.Op, err
  conf := new(T)
  err := json.Unmarshal([]byte(req.Data), &conf)
  if err != nil {
    return nil, errs.ErrArgs.WithDetail(err.Error()).Wrap()
  }
  eq := reflect.DeepEqual(old, conf)
  if eq {
    return nil, nil
  }
  data, err := json.Marshal(conf)
  if err != nil {
    return nil, errs.ErrArgs.WithDetail(err.Error()).Wrap()
  }
  op := clientv3.OpPut(etcd.BuildKey(req.ConfigName), string(data))
  return &op, nil
}

func compareAndSave[T any](c *gin.Context, old any, req *apistruct.SetConfigReq, cm *ConfigManager) error {
  conf := new(T)
  err := json.Unmarshal([]byte(req.Data), &conf)
  if err != nil {
    return errs.ErrArgs.WithDetail(err.Error()).Wrap()
  }
  eq := reflect.DeepEqual(old, conf)
  if eq {
    return nil
  }
  data, err := json.Marshal(conf)
```

```
	if err != nil {
		return errs.ErrArgs.WithDetail(err.Error()).Wrap()
	}
	_, err = cm.client.Put(c, etcd.BuildKey(req.ConfigName), string(data))
	if err != nil {
		return errs.WrapMsg(err, "save to etcd failed")
	}
	return nil
}

func (cm *ConfigManager) ResetConfig(c *gin.Context) {
	go func() {
		if err := cm.resetConfig(c, true); err != nil {
			log.ZError(c, "reset config err", err)
		}
	}()
	apiresp.GinSuccess(c, nil)
}

func (cm *ConfigManager) resetConfig(c *gin.Context, checkChange bool, ops ...clientv3.Op) error {
	txn := cm.client.Txn(c)
	type initConf struct {
		old any
		new any
	}
	configMap := map[string]*initConf{
		cm.config.Discovery.GetConfigFileName():    {old: &cm.config.Discovery, new: new(config.Discovery)},
		cm.config.Kafka.GetConfigFileName():        {old: &cm.config.Kafka, new: new(config.Kafka)},
		cm.config.LocalCache.GetConfigFileName():   {old: &cm.config.LocalCache, new: new(config.LocalCache)},
		cm.config.Log.GetConfigFileName():          {old: &cm.config.Log, new: new(config.Log)},
		cm.config.Minio.GetConfigFileName():        {old: &cm.config.Minio, new: new(config.Minio)},
		cm.config.Mongo.GetConfigFileName():        {old: &cm.config.Mongo, new: new(config.Mongo)},
		cm.config.Notification.GetConfigFileName(): {old: &cm.config.Notification, new: new(config.Notification)},
		cm.config.API.GetConfigFileName():          {old: &cm.config.API, new: new(config.API)},
		cm.config.CronTask.GetConfigFileName():     {old: &cm.config.CronTask, new: new(config.CronTask)},
		cm.config.MsgGateway.GetConfigFileName():   {old: &cm.config.MsgGateway, new: new(config.MsgGateway)},
		cm.config.MsgTransfer.GetConfigFileName():  {old: &cm.config.MsgTransfer, new: new(config.MsgTransfer)},
		cm.config.Push.GetConfigFileName():         {old: &cm.config.Push, new: new(config.Push)},
		cm.config.Auth.GetConfigFileName():         {old: &cm.config.Auth, new: new(config.Auth)},
		cm.config.Conversation.GetConfigFileName(): {old: &cm.config.Conversation, new: new(config.Conversation)},
		cm.config.Friend.GetConfigFileName():       {old: &cm.config.Friend, new: new(config.Friend)},
		cm.config.Group.GetConfigFileName():        {old: &cm.config.Group, new: new(config.Group)},
		cm.config.Msg.GetConfigFileName():          {old: &cm.config.Msg, new: new(config.Msg)},
		cm.config.Third.GetConfigFileName():        {old: &cm.config.Third, new: new(config.Third)},
		cm.config.User.GetConfigFileName():         {old: &cm.config.User, new: new(config.User)},
		cm.config.Redis.GetConfigFileName():        {old: &cm.config.Redis, new: new(config.Redis)},
		cm.config.Share.GetConfigFileName():        {old: &cm.config.Share, new: new(config.Share)},
		cm.config.Webhooks.GetConfigFileName():     {old: &cm.config.Webhooks, new: new(config.Webhooks)},
	}

	changedKeys := make([]string, 0, len(configMap))
	for k, v := range configMap {
		err := config.Load(cm.configPath, k, config.EnvPrefixMap[k], v.new)
		if err != nil {
			log.ZError(c, "load config failed", err)
			continue
		}
		equal := reflect.DeepEqual(v.old, v.new)
		if !checkChange || !equal {
			changedKeys = append(changedKeys, k)
		}
	}

	for _, k := range changedKeys {
		data, err := json.Marshal(configMap[k].new)
		if err != nil {
```

```
        log.ZError(c, "marshal config failed", err)
        continue
    }
    ops = append(ops, clientv3.OpPut(etcd.BuildKey(k), string(data)))
  }
  if len(ops) > 0 {
    txn.Then(ops...)
    _, err := txn.Commit()
    if err != nil {
        return errs.WrapMsg(err, "commit etcd txn failed")
    }
  }
  return nil
}

func (cm *ConfigManager) Restart(c *gin.Context) {
  go cm.restart(c)
  apiresp.GinSuccess(c, nil)
}

func (cm *ConfigManager) restart(c *gin.Context) {
  time.Sleep(waitHttp) // wait for Restart http call return
  t := time.Now().Unix()
  _, err := cm.client.Put(c, etcd.BuildKey(etcd.RestartKey), strconv.Itoa(int(t)))
  if err != nil {
    log.ZError(c, "restart etcd put key failed", err)
  }
}

func (cm *ConfigManager) SetEnableConfigManager(c *gin.Context) {
  if cm.config.Discovery.Enable != config.ETCD {
    apiresp.GinError(c, errs.New("only etcd support config manager").Wrap())
    return
  }
  var req apistruct.SetEnableConfigManagerReq
  if err := c.BindJSON(&req); err != nil {
    apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
    return
  }
  var enableStr string
  if req.Enable {
    enableStr = etcd.Enable
  } else {
    enableStr = etcd.Disable
  }
  resp, err := cm.client.Get(c, etcd.BuildKey(etcd.EnableConfigCenterKey))
  if err != nil {
    apiresp.GinError(c, errs.WrapMsg(err, "getEnableConfigManager failed"))
    return
  }
  if !(resp.Count > 0 && string(resp.Kvs[0].Value) == etcd.Enable) && req.Enable {
    go func() {
        time.Sleep(waitHttp) // wait for Restart http call return
        err := cm.resetConfig(c, false, clientv3.OpPut(etcd.BuildKey(etcd.EnableConfigCenterKey), enableStr))
        if err != nil {
            log.ZError(c, "resetConfig failed", err)
        }
    }()
  } else {
    _, err = cm.client.Put(c, etcd.BuildKey(etcd.EnableConfigCenterKey), enableStr)
    if err != nil {
        apiresp.GinError(c, errs.WrapMsg(err, "setEnableConfigManager failed"))
        return
    }
  }
```

```
    apiresp.GinSuccess(c, nil)
}

func (cm *ConfigManager) GetEnableConfigManager(c *gin.Context) {
    resp, err := cm.client.Get(c, etcd.BuildKey(etcd.EnableConfigCenterKey))
    if err != nil {
        apiresp.GinError(c, errs.WrapMsg(err, "getEnableConfigManager failed"))
        return
    }
    var enable bool
    if resp.Count > 0 && string(resp.Kvs[0].Value) == etcd.Enable {
        enable = true
    }
    apiresp.GinSuccess(c, &apistruct.GetEnableConfigManagerResp{Enable: enable})
}
```

internal/api/conversation.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
    "github.com/gin-gonic/gin"

    "github.com/openimsdk/protocol/conversation"
    "github.com/openimsdk/tools/a2r"
)

type ConversationApi struct {
    Client conversation.ConversationClient
}

func NewConversationApi(client conversation.ConversationClient) ConversationApi {
    return ConversationApi{client}
}

func (o *ConversationApi) GetAllConversations(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetAllConversations, o.Client)
}

func (o *ConversationApi) GetSortedConversationList(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetSortedConversationList, o.Client)
}

func (o *ConversationApi) GetConversation(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetConversation, o.Client)
}

func (o *ConversationApi) GetConversations(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetConversations, o.Client)
}

func (o *ConversationApi) SetConversations(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.SetConversations, o.Client)
}

//func (o *ConversationApi) GetConversationOfflinePushUserIDs(c *gin.Context) {
//    a2r.Call(c, conversation.ConversationClient.GetConversationOfflinePushUserIDs, o.Client)
//}

func (o *ConversationApi) GetFullOwnerConversationIDs(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetFullOwnerConversationIDs, o.Client)
}

func (o *ConversationApi) GetIncrementalConversation(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetIncrementalConversation, o.Client)
}

func (o *ConversationApi) GetOwnerConversation(c *gin.Context) {
```

```
    a2r.Call(c, conversation.ConversationClient.GetOwnerConversation, o.Client)
}

func (o *ConversationApi) GetNotNotifyConversationIDs(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetNotNotifyConversationIDs, o.Client)
}

func (o *ConversationApi) GetPinnedConversationIDs(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.GetPinnedConversationIDs, o.Client)
}

func (o *ConversationApi) UpdateConversationsByUser(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.UpdateConversationsByUser, o.Client)
}

func (o *ConversationApi) DeleteConversations(c *gin.Context) {
    a2r.Call(c, conversation.ConversationClient.DeleteConversations, o.Client)
}
```

internal/api/custom_validator.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
    "github.com/go-playground/validator/v10"
    "github.com/openimsdk/protocol/constant"
)

// RequiredIf validates if the specified field is required based on the session type.
func RequiredIf(fl validator.FieldLevel) bool {
    sessionType := fl.Parent().FieldByName("SessionType").Int()

    switch sessionType {
    case constant.SingleChatType, constant.NotificationChatType:
        return fl.FieldName() != "RecvID" || fl.Field().String() != ""
    case constant.WriteGroupChatType, constant.ReadGroupChatType:
        return fl.FieldName() != "GroupID" || fl.Field().String() != ""
    default:
        return true
    }
}
```

internal/api/friend.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
    "github.com/gin-gonic/gin"

    "github.com/openimsdk/protocol/relation"
    "github.com/openimsdk/tools/a2r"
)

type FriendApi struct {
    Client relation.FriendClient
}

func NewFriendApi(client relation.FriendClient) FriendApi {
    return FriendApi{client}
}

func (o *FriendApi) ApplyToAddFriend(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.ApplyToAddFriend, o.Client)
}

func (o *FriendApi) RespondFriendApply(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.RespondFriendApply, o.Client)
}

func (o *FriendApi) DeleteFriend(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.DeleteFriend, o.Client)
}

func (o *FriendApi) GetFriendApplyList(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetPaginationFriendsApplyTo, o.Client)
}

func (o *FriendApi) GetDesignatedFriendsApply(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetDesignatedFriendsApply, o.Client)
}

func (o *FriendApi) GetSelfApplyList(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetPaginationFriendsApplyFrom, o.Client)
}

func (o *FriendApi) GetFriendList(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetPaginationFriends, o.Client)
}

func (o *FriendApi) GetDesignatedFriends(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetDesignatedFriends, o.Client)
}

func (o *FriendApi) SetFriendRemark(c *gin.Context) {
```

```
    a2r.Call(c, relation.FriendClient.SetFriendRemark, o.Client)
}

func (o *FriendApi) AddBlack(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.AddBlack, o.Client)
}

func (o *FriendApi) GetPaginationBlacks(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetPaginationBlacks, o.Client)
}

func (o *FriendApi) GetSpecifiedBlacks(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetSpecifiedBlacks, o.Client)
}

func (o *FriendApi) RemoveBlack(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.RemoveBlack, o.Client)
}

func (o *FriendApi) ImportFriends(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.ImportFriends, o.Client)
}

func (o *FriendApi) IsFriend(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.IsFriend, o.Client)
}

func (o *FriendApi) GetFriendIDs(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetFriendIDs, o.Client)
}

func (o *FriendApi) GetSpecifiedFriendsInfo(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetSpecifiedFriendsInfo, o.Client)
}

func (o *FriendApi) UpdateFriends(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.UpdateFriends, o.Client)
}

func (o *FriendApi) GetIncrementalFriends(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetIncrementalFriends, o.Client)
}

// GetIncrementalBlacks is temporarily unused.
// Deprecated: This function is currently unused and may be removed in future versions.
func (o *FriendApi) GetIncrementalBlacks(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetIncrementalBlacks, o.Client)
}

func (o *FriendApi) GetFullFriendUserIDs(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetFullFriendUserIDs, o.Client)
}

func (o *FriendApi) GetSelfUnhandledApplyCount(c *gin.Context) {
    a2r.Call(c, relation.FriendClient.GetSelfUnhandledApplyCount, o.Client)
}
```

internal/api/group.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
    "github.com/gin-gonic/gin"
    "github.com/openimsdk/protocol/group"
    "github.com/openimsdk/tools/a2r"
)

type GroupApi struct {
    Client group.GroupClient
}

func NewGroupApi(client group.GroupClient) GroupApi {
    return GroupApi{client}
}

func (o *GroupApi) CreateGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.CreateGroup, o.Client)
}

func (o *GroupApi) SetGroupInfo(c *gin.Context) {
    a2r.Call(c, group.GroupClient.SetGroupInfo, o.Client)
}

func (o *GroupApi) SetGroupInfoEx(c *gin.Context) {
    a2r.Call(c, group.GroupClient.SetGroupInfoEx, o.Client)
}

func (o *GroupApi) JoinGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.JoinGroup, o.Client)
}

func (o *GroupApi) QuitGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.QuitGroup, o.Client)
}

func (o *GroupApi) ApplicationGroupResponse(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GroupApplicationResponse, o.Client)
}

func (o *GroupApi) TransferGroupOwner(c *gin.Context) {
    a2r.Call(c, group.GroupClient.TransferGroupOwner, o.Client)
}

func (o *GroupApi) GetRecvGroupApplicationList(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetGroupApplicationList, o.Client)
}

func (o *GroupApi) GetUserReqGroupApplicationList(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetUserReqApplicationList, o.Client)
```

```
}

func (o *GroupApi) GetGroupUsersReqApplicationList(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetGroupUsersReqApplicationList, o.Client)
}

func (o *GroupApi) GetSpecifiedUserGroupRequestInfo(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetSpecifiedUserGroupRequestInfo, o.Client)
}

func (o *GroupApi) GetGroupsInfo(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetGroupsInfo, o.Client)
    //a2r.Call(c, group.GroupClient.GetGroupsInfo, o.Client, c, a2r.NewNilReplaceOption(group.GroupClient.GetGroupsInfo
}

func (o *GroupApi) KickGroupMember(c *gin.Context) {
    a2r.Call(c, group.GroupClient.KickGroupMember, o.Client)
}

func (o *GroupApi) GetGroupMembersInfo(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetGroupMembersInfo, o.Client)
    //a2r.Call(c, group.GroupClient.GetGroupMembersInfo, o.Client, c, a2r.NewNilReplaceOption(group.GroupClient.GetGrou
}

func (o *GroupApi) GetGroupMemberList(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetGroupMemberList, o.Client)
}

func (o *GroupApi) InviteUserToGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.InviteUserToGroup, o.Client)
}

func (o *GroupApi) GetJoinedGroupList(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetJoinedGroupList, o.Client)
}

func (o *GroupApi) DismissGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.DismissGroup, o.Client)
}

func (o *GroupApi) MuteGroupMember(c *gin.Context) {
    a2r.Call(c, group.GroupClient.MuteGroupMember, o.Client)
}

func (o *GroupApi) CancelMuteGroupMember(c *gin.Context) {
    a2r.Call(c, group.GroupClient.CancelMuteGroupMember, o.Client)
}

func (o *GroupApi) MuteGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.MuteGroup, o.Client)
}

func (o *GroupApi) CancelMuteGroup(c *gin.Context) {
    a2r.Call(c, group.GroupClient.CancelMuteGroup, o.Client)
}

func (o *GroupApi) SetGroupMemberInfo(c *gin.Context) {
    a2r.Call(c, group.GroupClient.SetGroupMemberInfo, o.Client)
}

func (o *GroupApi) GetGroupAbstractInfo(c *gin.Context) {
    a2r.Call(c, group.GroupClient.GetGroupAbstractInfo, o.Client)
}

// func (g *Group) SetGroupMemberNickname(c *gin.Context) {
//    a2r.Call(c, group.GroupClient.SetGroupMemberNickname, g.userClient)
```

```
//}
//
// func (g *Group) GetGroupAllMemberList(c *gin.Context) {
//■a2r.Call(c, group.GroupClient.GetGroupAllMember, g.userClient)
//}

func (o *GroupApi) GroupCreateCount(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GroupCreateCount, o.Client)
}

func (o *GroupApi) GetGroups(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetGroups, o.Client)
}

func (o *GroupApi) GetGroupMemberUserIDs(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetGroupMemberUserIDs, o.Client)
}

func (o *GroupApi) GetIncrementalJoinGroup(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetIncrementalJoinGroup, o.Client)
}

func (o *GroupApi) GetIncrementalGroupMember(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetIncrementalGroupMember, o.Client)
}

func (o *GroupApi) GetIncrementalGroupMemberBatch(c *gin.Context) {
■a2r.Call(c, group.GroupClient.BatchGetIncrementalGroupMember, o.Client)
}

func (o *GroupApi) GetFullGroupMemberUserIDs(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetFullGroupMemberUserIDs, o.Client)
}

func (o *GroupApi) GetFullJoinGroupIDs(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetFullJoinGroupIDs, o.Client)
}

func (o *GroupApi) GetGroupApplicationUnhandledCount(c *gin.Context) {
■a2r.Call(c, group.GroupClient.GetGroupApplicationUnhandledCount, o.Client)
}
```

internal/api/init.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
    "context"
    "errors"
    "fmt"
    "net"
    "net/http"
    "strconv"
    "time"

    conf "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/utils/datautil"
    "github.com/openimsdk/tools/utils/network"
    "github.com/openimsdk/tools/utils/runtimeenv"
    "google.golang.org/grpc"
)

type Config struct {
    conf.AllConfig

    ConfigPath conf.Path
    Index      conf.Index
}

func Start(ctx context.Context, config *Config, client discovery.SvcDiscoveryRegistry, service grpc.ServiceRegistrar
    apiPort, err := datautil.GetElemByIndex(config.API.Api.Ports, int(config.Index))
    if err != nil {
        return err
    }

    router, err := newGinRouter(ctx, client, config)
    if err != nil {
        return err
    }

    apiCtx, apiCancel := context.WithCancelCause(context.Background())
    done := make(chan struct{})
    go func() {
        httpServer := &http.Server{
            Handler: router,
            Addr:    net.JoinHostPort(network.GetListenIP(config.API.Api.ListenIP), strconv.Itoa(apiPort)),
        }
        go func() {
            defer close(done)
            select {
            case <-ctx.Done():
                apiCancel(fmt.Errorf("recv ctx %w", context.Cause(ctx)))
```

```
    case <-apiCtx.Done():
    }
    log.ZDebug(ctx, "api server is shutting down")
    if err := httpServer.Shutdown(context.Background()); err != nil {
        log.ZWarn(ctx, "api server shutdown err", err)
    }
  }()
  log.CInfo(ctx, "api server is init", "runtimeEnv", runtimeenv.RuntimeEnvironment(), "address", httpServer.Addr, "a
  err := httpServer.ListenAndServe()
  if err == nil {
    err = errors.New("api done")
  }
  apiCancel(err)
 }()

 //if config.Discovery.Enable == conf.ETCD {
 //  cm := disetcd.NewConfigManager(client.(*etcd.SvcDiscoveryRegistryImpl).GetClient(), config.GetConfigNames())
 //  cm.Watch(ctx)
 //}
 //sigs := make(chan os.Signal, 1)
 //signal.Notify(sigs, syscall.SIGTERM)
 //select {
 //case val := <-sigs:
 //  log.ZDebug(ctx, "recv exit", "signal", val.String())
 //  cancel(fmt.Errorf("signal %s", val.String()))
 //case <-ctx.Done():
 //}
 <-apiCtx.Done()
 exitCause := context.Cause(apiCtx)
 log.ZWarn(ctx, "api server exit", exitCause)
 timer := time.NewTimer(time.Second * 15)
 defer timer.Stop()
 select {
 case <-timer.C:
  log.ZWarn(ctx, "api server graceful stop timeout", nil)
 case <-done:
  log.ZDebug(ctx, "api server graceful stop done")
 }
 return exitCause
}
```

internal/api/msg.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
	"encoding/base64"
	"encoding/json"
	"sync"

	"github.com/gin-gonic/gin"
	"github.com/go-playground/validator/v10"
	"github.com/mitchellh/mapstructure"
	"google.golang.org/protobuf/reflect/protoreflect"

	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/a2r"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/openimsdk/tools/utils/idutil"
	"github.com/openimsdk/tools/utils/jsonutil"
	"github.com/openimsdk/tools/utils/timeutil"
)

var (
	msgDataDescriptor     []protoreflect.FieldDescriptor
	msgDataDescriptorOnce sync.Once
)

func getMsgDataDescriptor() []protoreflect.FieldDescriptor {
	msgDataDescriptorOnce.Do(func() {
		skip := make(map[string]struct{})
		respFields := new(msg.SendMsgResp).ProtoReflect().Descriptor().Fields()
		for i := 0; i < respFields.Len(); i++ {
			field := respFields.Get(i)
			if !field.HasJSONName() {
				continue
			}
			skip[field.JSONName()] = struct{}{}
		}
		fields := new(sdkws.MsgData).ProtoReflect().Descriptor().Fields()
		num := fields.Len()
		msgDataDescriptor = make([]protoreflect.FieldDescriptor, 0, num)
```

```
    for i := 0; i < num; i++ {
        field := fields.Get(i)
        if !field.HasJSONName() {
            continue
        }
        if _, ok := skip[field.JSONName()]; ok {
            continue
        }
        msgDataDescriptor = append(msgDataDescriptor, fields.Get(i))
    }
    })
    return msgDataDescriptor
}

type MessageApi struct {
    Client        msg.MsgClient
    userClient    *rpcli.UserClient
    imAdminUserID []string
    validate      *validator.Validate
}

func NewMessageApi(client msg.MsgClient, userClient *rpcli.UserClient, imAdminUserID []string) MessageApi {
    return MessageApi{Client: client, userClient: userClient, imAdminUserID: imAdminUserID, validate: validator.New()}
}

func (*MessageApi) SetOptions(options map[string]bool, value bool) {
    datautil.SetSwitchFromOptions(options, constant.IsHistory, value)
    datautil.SetSwitchFromOptions(options, constant.IsPersistent, value)
    datautil.SetSwitchFromOptions(options, constant.IsSenderSync, value)
    datautil.SetSwitchFromOptions(options, constant.IsConversationUpdate, value)
}

func (m *MessageApi) newUserSendMsgReq(_ *gin.Context, params *apistruct.SendMsg, data any) *msg.SendMsgReq {
    msgData := &sdkws.MsgData{
        SendID:           params.SendID,
        GroupID:          params.GroupID,
        ClientMsgID:      idutil.GetMsgIDByMD5(params.SendID),
        SenderPlatformID: params.SenderPlatformID,
        SenderNickname:   params.SenderNickname,
        SenderFaceURL:    params.SenderFaceURL,
        SessionType:      params.SessionType,
        MsgFrom:          constant.SysMsgType,
        ContentType:      params.ContentType,
        CreateTime:       timeutil.GetCurrentTimestampByMill(),
        SendTime:         params.SendTime,
        OfflinePushInfo:  params.OfflinePushInfo,
        Ex:               params.Ex,
    }
    var newContent string
    options := make(map[string]bool, 5)
    switch params.ContentType {
    case constant.OANotification:
        notification := sdkws.NotificationElem{}
        notification.Detail = jsonutil.StructToJsonString(params.Content)
        newContent = jsonutil.StructToJsonString(&notification)
    case constant.Text:
        fallthrough
    case constant.AtText:
        if atElem, ok := data.(*apistruct.AtElem); ok {
            msgData.AtUserIDList = atElem.AtUserList
        }
        fallthrough
    case constant.Picture:
        fallthrough
    case constant.Custom:
        fallthrough
```

```
	case constant.Voice:
		fallthrough
	case constant.Video:
		fallthrough
	case constant.File:
		fallthrough
	default:
		newContent = jsonutil.StructToJsonString(params.Content)
	}
	if params.IsOnlineOnly {
		m.SetOptions(options, false)
	}
	if params.NotOfflinePush {
		datautil.SetSwitchFromOptions(options, constant.IsOfflinePush, false)
	}
	msgData.Content = []byte(newContent)
	msgData.Options = options
	pbData := msg.SendMsgReq{
		MsgData: msgData,
	}
	return &pbData
}

func (m *MessageApi) GetSeq(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetMaxSeq, m.Client)
}

func (m *MessageApi) PullMsgBySeqs(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.PullMessageBySeqs, m.Client)
}

func (m *MessageApi) RevokeMsg(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.RevokeMsg, m.Client)
}

func (m *MessageApi) MarkMsgsAsRead(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.MarkMsgsAsRead, m.Client)
}

func (m *MessageApi) MarkConversationAsRead(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.MarkConversationAsRead, m.Client)
}

func (m *MessageApi) GetConversationsHasReadAndMaxSeq(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetConversationsHasReadAndMaxSeq, m.Client)
}

func (m *MessageApi) SetConversationHasReadSeq(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.SetConversationHasReadSeq, m.Client)
}

func (m *MessageApi) ClearConversationsMsg(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.ClearConversationsMsg, m.Client)
}

func (m *MessageApi) UserClearAllMsg(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.UserClearAllMsg, m.Client)
}

func (m *MessageApi) DeleteMsgs(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.DeleteMsgs, m.Client)
}

func (m *MessageApi) DeleteMsgPhysicalBySeq(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.DeleteMsgPhysicalBySeq, m.Client)
}
```

```go
func (m *MessageApi) DeleteMsgPhysical(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.DeleteMsgPhysical, m.Client)
}

func (m *MessageApi) getSendMsgReq(c *gin.Context, req apistruct.SendMsg) (sendMsgReq *msg.SendMsgReq, err error) {
	var data any
	log.ZDebug(c, "getSendMsgReq", "req", req.Content)
	switch req.ContentType {
	case constant.Text:
		data = &apistruct.TextElem{}
	case constant.Picture:
		data = &apistruct.PictureElem{}
	case constant.Voice:
		data = &apistruct.SoundElem{}
	case constant.Video:
		data = &apistruct.VideoElem{}
	case constant.File:
		data = &apistruct.FileElem{}
	case constant.AtText:
		data = &apistruct.AtElem{}
	case constant.Custom:
		data = &apistruct.CustomElem{}
	case constant.MarkdownText:
		data = &apistruct.MarkdownTextElem{}
	case constant.Quote:
		data = &apistruct.QuoteElem{}
	case constant.OANotification:
		data = &apistruct.OANotificationElem{}
		req.SessionType = constant.NotificationChatType
		if err = m.userClient.GetNotificationByID(c, req.SendID); err != nil {
			return nil, err
		}
	default:
		return nil, errs.WrapMsg(errs.ErrArgs, "unsupported content type", "contentType", req.ContentType)
	}
	if err := mapstructure.WeakDecode(req.Content, data); err != nil {
		return nil, errs.WrapMsg(err, "failed to decode message content")
	}
	log.ZDebug(c, "getSendMsgReq", "decodedContent", data)
	if err := m.validate.Struct(data); err != nil {
		return nil, errs.WrapMsg(err, "validation error")
	}
	return m.newUserSendMsgReq(c, &req, data), nil
}

func (m *MessageApi) getModifyFields(req, respModify *sdkws.MsgData) map[string]any {
	if req == nil || respModify == nil {
		return nil
	}
	fields := make(map[string]any)
	reqProtoReflect := req.ProtoReflect()
	respProtoReflect := respModify.ProtoReflect()
	for _, descriptor := range getMsgDataDescriptor() {
		reqValue := reqProtoReflect.Get(descriptor)
		respValue := respProtoReflect.Get(descriptor)
		if !reqValue.Equal(respValue) {
			val := respValue.Interface()
			name := descriptor.JSONName()
			if name == "content" {
				if bs, ok := val.([]byte); ok {
					val = string(bs)
				}
			}
			fields[name] = val
		}
```

```
	}
	if len(fields) == 0 {
		fields = nil
	}
	return fields
}

func (m *MessageApi) ginRespSendMsg(c *gin.Context, req *msg.SendMsgReq, resp *msg.SendMsgResp) {
	res := m.getModifyFields(req.MsgData, resp.Modify)
	resp.Modify = nil
	apiresp.GinSuccess(c, &apistruct.SendMsgResp{
		SendMsgResp: resp,
		Modify:      res,
	})
}

// SendMessage handles the sending of a message. It's an HTTP handler function to be used with Gin framework.
func (m *MessageApi) SendMessage(c *gin.Context) {
	// Initialize a request struct for sending a message.
	req := apistruct.SendMsgReq{}

	// Bind the JSON request body to the request struct.
	if err := c.BindJSON(&req); err != nil {
		// Respond with an error if request body binding fails.
		apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
		return
	}

	// Check if the user has the app manager role.
	if !authverify.IsAdmin(c) {
		// Respond with a permission error if the user is not an app manager.
		apiresp.GinError(c, errs.ErrNoPermission.WrapMsg("only app manager can send message"))
		return
	}

	// Prepare the message request with additional required data.
	sendMsgReq, err := m.getSendMsgReq(c, req.SendMsg)
	if err != nil {
		// Log and respond with an error if preparation fails.
		apiresp.GinError(c, err)
		return
	}

	// Set the receiver ID in the message data.
	sendMsgReq.MsgData.RecvID = req.RecvID

	// Attempt to send the message using the client.
	respPb, err := m.Client.SendMsg(c, sendMsgReq)
	if err != nil {
		// Set the status to failed and respond with an error if sending fails.
		apiresp.GinError(c, err)
		return
	}

	// Set the status to successful if the message is sent.
	var status = constant.MsgSendSuccessed

	// Attempt to update the message sending status in the system.
	_, err = m.Client.SetSendMsgStatus(c, &msg.SetSendMsgStatusReq{
		Status: int32(status),
	})

	if err != nil {
		// Log the error if updating the status fails.
		apiresp.GinError(c, err)
		return
```

```
}

// Respond with a success message and the response payload.
m.ginRespSendMsg(c, sendMsgReq, respPb)
}

func (m *MessageApi) SendBusinessNotification(c *gin.Context) {
req := struct {
Key              string `json:"key"`
Data             string `json:"data"`
SendUserID       string `json:"sendUserID" binding:"required"`
RecvUserID       string `json:"recvUserID"`
RecvGroupID      string `json:"recvGroupID"`
SendMsg          bool   `json:"sendMsg"`
ReliabilityLevel *int   `json:"reliabilityLevel"`
}{}
if err := c.BindJSON(&req); err != nil {
apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
return
}
if req.RecvUserID == "" && req.RecvGroupID == "" {
apiresp.GinError(c, errs.ErrArgs.WrapMsg("recvUserID and recvGroupID cannot be empty at the same time"))
return
}
if req.RecvUserID != "" && req.RecvGroupID != "" {
apiresp.GinError(c, errs.ErrArgs.WrapMsg("recvUserID and recvGroupID cannot be set at the same time"))
return
}
var sessionType int32
if req.RecvUserID != "" {
sessionType = constant.SingleChatType
} else {
sessionType = constant.ReadGroupChatType
}
if req.ReliabilityLevel == nil {
req.ReliabilityLevel = datautil.ToPtr(1)
}
if !authverify.IsAdmin(c) {
apiresp.GinError(c, errs.ErrNoPermission.WrapMsg("only app manager can send message"))
return
}
sendMsgReq := msg.SendMsgReq{
MsgData: &sdkws.MsgData{
SendID:  req.SendUserID,
RecvID:  req.RecvUserID,
GroupID: req.RecvGroupID,
Content: []byte(jsonutil.StructToJsonString(&sdkws.NotificationElem{
Detail: jsonutil.StructToJsonString(&struct {
Key  string `json:"key"`
Data string `json:"data"`
}{Key: req.Key, Data: req.Data}),
})),
MsgFrom:     constant.SysMsgType,
ContentType: constant.BusinessNotification,
SessionType: sessionType,
CreateTime:  timeutil.GetCurrentTimestampByMill(),
ClientMsgID: idutil.GetMsgIDByMD5(mcontext.GetOpUserID(c)),
Options: config.GetOptionsByNotification(config.NotificationConfig{
IsSendMsg:        req.SendMsg,
ReliabilityLevel: *req.ReliabilityLevel,
UnreadCount:      false,
}, nil),
},
}
respPb, err := m.Client.SendMsg(c, &sendMsgReq)
if err != nil {
```

```
		apiresp.GinError(c, err)
		return
	}
	m.ginRespSendMsg(c, &sendMsgReq, respPb)
}

func (m *MessageApi) BatchSendMsg(c *gin.Context) {
	var (
		req  apistruct.BatchSendMsgReq
		resp apistruct.BatchSendMsgResp
	)
	if err := c.BindJSON(&req); err != nil {
		apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
		return
	}
	if err := authverify.CheckAdmin(c); err != nil {
		apiresp.GinError(c, errs.ErrNoPermission.WrapMsg("only app manager can send message"))
		return
	}

	var recvIDs []string
	if req.IsSendAll {
		var pageNumber int32 = 1
		const showNumber = 500
		for {
			recvIDsPart, err := m.userClient.GetAllUserIDs(c, pageNumber, showNumber)
			if err != nil {
				apiresp.GinError(c, err)
				return
			}
			recvIDs = append(recvIDs, recvIDsPart...)
			if len(recvIDsPart) < showNumber {
				break
			}
			pageNumber++
		}
	} else {
		recvIDs = req.RecvIDs
	}
	log.ZDebug(c, "BatchSendMsg nums", "nums ", len(recvIDs))
	sendMsgReq, err := m.getSendMsgReq(c, req.SendMsg)
	if err != nil {
		apiresp.GinError(c, err)
		return
	}
	for _, recvID := range recvIDs {
		sendMsgReq.MsgData.RecvID = recvID
		rpcResp, err := m.Client.SendMsg(c, sendMsgReq)
		if err != nil {
			resp.FailedIDs = append(resp.FailedIDs, recvID)
			continue
		}
		resp.Results = append(resp.Results, &apistruct.SingleReturnResult{
			ServerMsgID: rpcResp.ServerMsgID,
			ClientMsgID: rpcResp.ClientMsgID,
			SendTime:    rpcResp.SendTime,
			RecvID:      recvID,
			Modify:      m.getModifyFields(sendMsgReq.MsgData, rpcResp.Modify),
		})
	}
	apiresp.GinSuccess(c, resp)
}

func (m *MessageApi) SendSimpleMessage(c *gin.Context) {
	encodedKey, ok := c.GetQuery(webhook.Key)
	if !ok {
```

```
    apiresp.GinError(c, errs.ErrArgs.WithDetail("missing key in query").Wrap())
    return
  }

  decodedData, err := base64.StdEncoding.DecodeString(encodedKey)
  if err != nil {
    apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
    return
  }
  var (
    req        apistruct.SendSingleMsgReq
    keyMsgData apistruct.KeyMsgData

    sendID      string
    sessionType int32
    recvID      string
  )
  if err = c.BindJSON(&req); err != nil {
    apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
    return
  }
  err = json.Unmarshal(decodedData, &keyMsgData)
  if err != nil {
    apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
    return
  }
  if keyMsgData.GroupID != "" {
    sessionType = constant.ReadGroupChatType
    sendID = req.SendID
  } else {
    sessionType = constant.SingleChatType
    sendID = keyMsgData.RecvID
    recvID = keyMsgData.SendID
  }
  // check param
  if keyMsgData.SendID == "" {
    apiresp.GinError(c, errs.ErrArgs.WithDetail("missing recvID or GroupID").Wrap())
    return
  }
  if sendID == "" {
    apiresp.GinError(c, errs.ErrArgs.WithDetail("missing sendID").Wrap())
    return
  }

  content, err := jsonutil.JsonMarshal(apistruct.MarkdownTextElem{Content: req.Content})
  if err != nil {
    apiresp.GinError(c, errs.Wrap(err))
    return
  }
  msgData := &sdkws.MsgData{
    SendID:           sendID,
    RecvID:           recvID,
    GroupID:          keyMsgData.GroupID,
    ClientMsgID:      idutil.GetMsgIDByMD5(sendID),
    SenderPlatformID: constant.AdminPlatformID,
    SessionType:      sessionType,
    MsgFrom:          constant.UserMsgType,
    ContentType:      constant.MarkdownText,
    Content:          content,
    OfflinePushInfo:  req.OfflinePushInfo,
    Ex:               req.Ex,
  }

  sendReq := &msg.SendSimpleMsgReq{
    MsgData: msgData,
  }
```

```
	respPb, err := m.Client.SendSimpleMsg(c, sendReq)
	if err != nil {
		apiresp.GinError(c, err)
		return
	}

	var status = constant.MsgSendSuccessed

	_, err = m.Client.SetSendMsgStatus(c, &msg.SetSendMsgStatusReq{
		Status: int32(status),
	})

	if err != nil {
		apiresp.GinError(c, err)
		return
	}

	m.ginRespSendMsg(c, &msg.SendMsgReq{MsgData: sendReq.MsgData}, &msg.SendMsgResp{
		ServerMsgID: respPb.ServerMsgID,
		ClientMsgID: respPb.ClientMsgID,
		SendTime:    respPb.SendTime,
		Modify:      respPb.Modify,
	})
}

func (m *MessageApi) CheckMsgIsSendSuccess(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetSendMsgStatus, m.Client)
}

func (m *MessageApi) GetUsersOnlineStatus(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetSendMsgStatus, m.Client)
}

func (m *MessageApi) GetActiveUser(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetActiveUser, m.Client)
}

func (m *MessageApi) GetActiveGroup(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetActiveGroup, m.Client)
}

func (m *MessageApi) SearchMsg(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.SearchMessage, m.Client)
}

func (m *MessageApi) GetServerTime(c *gin.Context) {
	a2r.Call(c, msg.MsgClient.GetServerTime, m.Client)
}
```

internal/api/prometheus_discovery.go

```
package api

import (
	"encoding/json"
	"errors"
	"net/http"

	"github.com/gin-gonic/gin"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/errs"
)

type PrometheusDiscoveryApi struct {
	config *Config
	kv     discovery.KeyValue
}

func NewPrometheusDiscoveryApi(config *Config, client discovery.SvcDiscoveryRegistry) *PrometheusDiscoveryApi {
	api := &PrometheusDiscoveryApi{
		config: config,
		kv:     client,
	}
	return api
}

func (p *PrometheusDiscoveryApi) discovery(c *gin.Context, key string) {
	value, err := p.kv.GetKeyWithPrefix(c, prommetrics.BuildDiscoveryKeyPrefix(key))
	if err != nil {
		if errors.Is(err, discovery.ErrNotSupported) {
			c.JSON(http.StatusOK, []struct{}{})
			return
		}
		apiresp.GinError(c, errs.WrapMsg(err, "get key value"))
		return
	}
	if len(value) == 0 {
		c.JSON(http.StatusOK, []*prommetrics.RespTarget{})
		return
	}
	var resp prommetrics.RespTarget
	for i := range value {
		var tmp prommetrics.Target
		if err = json.Unmarshal(value[i], &tmp); err != nil {
			apiresp.GinError(c, errs.WrapMsg(err, "json unmarshal err"))
			return
		}

		resp.Targets = append(resp.Targets, tmp.Target)
		resp.Labels = tmp.Labels // default label is fixed. See prommetrics.BuildDefaultTarget
	}

	c.JSON(http.StatusOK, []*prommetrics.RespTarget{&resp})
}

func (p *PrometheusDiscoveryApi) Api(c *gin.Context) {
	p.discovery(c, prommetrics.APIKeyName)
}

func (p *PrometheusDiscoveryApi) User(c *gin.Context) {
	p.discovery(c, p.config.Discovery.RpcService.User)
}
```

```
func (p *PrometheusDiscoveryApi) Group(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Group)
}

func (p *PrometheusDiscoveryApi) Msg(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Msg)
}

func (p *PrometheusDiscoveryApi) Friend(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Friend)
}

func (p *PrometheusDiscoveryApi) Conversation(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Conversation)
}

func (p *PrometheusDiscoveryApi) Third(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Third)
}

func (p *PrometheusDiscoveryApi) Auth(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Auth)
}

func (p *PrometheusDiscoveryApi) Push(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.Push)
}

func (p *PrometheusDiscoveryApi) MessageGateway(c *gin.Context) {
    p.discovery(c, p.config.Discovery.RpcService.MessageGateway)
}

func (p *PrometheusDiscoveryApi) MessageTransfer(c *gin.Context) {
    p.discovery(c, prommetrics.MessageTransferKeyName)
}
```

internal/api/ratelimit.go

```
package api

import (
	"fmt"
	"math"
	"net/http"
	"strconv"
	"time"

	"github.com/gin-gonic/gin"

	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/stability/ratelimit"
	"github.com/openimsdk/tools/stability/ratelimit/bbr"
)

type RateLimiter struct {
	Enable       bool          `yaml:"enable"`
	Window       time.Duration `yaml:"window"`       // time duration per window
	Bucket       int           `yaml:"bucket"`       // bucket number for each window
	CPUThreshold int64         `yaml:"cpuThreshold"` // CPU threshold; valid range 0–1000 (1000 = 100%)
}

func RateLimitMiddleware(config *RateLimiter) gin.HandlerFunc {
	if !config.Enable {
		return func(c *gin.Context) {
			c.Next()
		}
	}

	limiter := bbr.NewBBRLimiter(
		bbr.WithWindow(config.Window),
		bbr.WithBucket(config.Bucket),
		bbr.WithCPUThreshold(config.CPUThreshold),
	)

	return func(c *gin.Context) {
		status := limiter.Stat()

		c.Header("X-BBR-CPU", strconv.FormatInt(status.CPU, 10))
		c.Header("X-BBR-MinRT", strconv.FormatInt(status.MinRt, 10))
		c.Header("X-BBR-MaxPass", strconv.FormatInt(status.MaxPass, 10))
		c.Header("X-BBR-MaxInFlight", strconv.FormatInt(status.MaxInFlight, 10))
		c.Header("X-BBR-InFlight", strconv.FormatInt(status.InFlight, 10))

		done, err := limiter.Allow()
		if err != nil {

			c.Header("X-RateLimit-Policy", "BBR")
			c.Header("Retry-After", calculateBBRRetryAfter(status))
			c.Header("X-RateLimit-Limit", strconv.FormatInt(status.MaxInFlight, 10))
			c.Header("X-RateLimit-Remaining", "0") // There is no concept of remaining quota in BBR.

			fmt.Println("rate limited:", err, "path:", c.Request.URL.Path)
			log.ZWarn(c, "rate limited", err, "path", c.Request.URL.Path)
			c.AbortWithStatus(http.StatusTooManyRequests)
			apiresp.GinError(c, errs.NewCodeError(http.StatusTooManyRequests, "too many requests, please try again later"))
			return
		}

		c.Next()
		done(ratelimit.DoneInfo{})
```

```
	}
}

func calculateBBRRetryAfter(status bbr.Stat) string {
	loadRatio := float64(status.CPU) / float64(status.CPU)

	if loadRatio < 0.8 {
		return "1"
	}
	if loadRatio < 0.95 {
		return "2"
	}

	backoff := 1 + int64(math.Pow(loadRatio-0.95, 2)*50)
	if backoff > 5 {
		backoff = 5
	}
	return strconv.FormatInt(backoff, 10)
}
```

internal/api/router.go

```
package api

import (
	"context"
	"net/http"
	"strings"

	"github.com/gin-contrib/gzip"
	"github.com/gin-gonic/gin"
	"github.com/gin-gonic/gin/binding"
	"github.com/go-playground/validator/v10"
	"github.com/openimsdk/open-im-server/v3/internal/api/jssdk"
	"github.com/openimsdk/open-im-server/v3/pkg/authverify"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	pbAuth "github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/relation"
	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/discovery/etcd"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mw"
	"github.com/openimsdk/tools/mw/api"
	clientv3 "go.etcd.io/etcd/client/v3"
)

const (
	NoCompression      = -1
	DefaultCompression = 0
	BestCompression    = 1
	BestSpeed          = 2
)

func prommetricsGin() gin.HandlerFunc {
	return func(c *gin.Context) {
		c.Next()
		path := c.FullPath()
		if c.Writer.Status() == http.StatusNotFound {
			prommetrics.HttpCall("<404>", c.Request.Method, c.Writer.Status())
		} else {
			prommetrics.HttpCall(path, c.Request.Method, c.Writer.Status())
		}
		if resp := apiresp.GetGinApiResponse(c); resp != nil {
			prommetrics.APICall(path, c.Request.Method, resp.ErrCode)
		}
	}
}

func newGinRouter(ctx context.Context, client discovery.SvcDiscoveryRegistry, cfg *Config) (*gin.Engine, error) {
	authConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.Auth)
	if err != nil {
		return nil, err
	}
	userConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.User)
	if err != nil {
		return nil, err
```

```
}
groupConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.Group)
if err != nil {
    return nil, err
}
friendConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.Friend)
if err != nil {
    return nil, err
}
conversationConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.Conversation)
if err != nil {
    return nil, err
}
thirdConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.Third)
if err != nil {
    return nil, err
}
msgConn, err := client.GetConn(ctx, cfg.Discovery.RpcService.Msg)
if err != nil {
    return nil, err
}
gin.SetMode(gin.ReleaseMode)
r := gin.New()
if v, ok := binding.Validator.Engine().(*validator.Validate); ok {
    _ = v.RegisterValidation("required_if", RequiredIf)
}
switch cfg.API.Api.CompressionLevel {
case NoCompression:
case DefaultCompression:
    r.Use(gzip.Gzip(gzip.DefaultCompression))
case BestCompression:
    r.Use(gzip.Gzip(gzip.BestCompression))
case BestSpeed:
    r.Use(gzip.Gzip(gzip.BestSpeed))
}

// Use rate limiter middleware
if cfg.API.RateLimiter.Enable {
    rl := &RateLimiter{
        Enable:       cfg.API.RateLimiter.Enable,
        Window:       cfg.API.RateLimiter.Window,
        Bucket:       cfg.API.RateLimiter.Bucket,
        CPUThreshold: cfg.API.RateLimiter.CPUThreshold,
    }
    r.Use(RateLimitMiddleware(rl))
}

if config.Standalone() {
    r.Use(func(c *gin.Context) {
        c.Set(authverify.CtxAdminUserIDsKey, cfg.Share.IMAdminUser.UserIDs)
    })
}
r.Use(api.GinLogger(), prommetricsGin(), gin.RecoveryWithWriter(gin.DefaultErrorWriter, mw.GinPanicErr), mw.CorsHar
    mw.GinParseOperationID(), GinParseToken(rpcli.NewAuthClient(authConn)), setGinIsAdmin(cfg.Share.IMAdminUser.UserI

u := NewUserApi(user.NewUserClient(userConn), client, cfg.Discovery.RpcService)
{
    userRouterGroup := r.Group("/user")
    userRouterGroup.POST("/user_register", u.UserRegister)
    userRouterGroup.POST("/update_user_info", u.UpdateUserInfo)
    userRouterGroup.POST("/update_user_info_ex", u.UpdateUserInfoEx)
    userRouterGroup.POST("/set_global_msg_recv_opt", u.SetGlobalRecvMessageOpt)
    userRouterGroup.POST("/get_users_info", u.GetUsersPublicInfo)
    userRouterGroup.POST("/get_all_users_uid", u.GetAllUsersID)
    userRouterGroup.POST("/account_check", u.AccountCheck)
    userRouterGroup.POST("/get_users", u.GetUsers)
```

```
    userRouterGroup.POST("/get_users_online_status", u.GetUsersOnlineStatus)
    userRouterGroup.POST("/get_users_online_token_detail", u.GetUsersOnlineTokenDetail)
    userRouterGroup.POST("/subscribe_users_status", u.SubscriberStatus)
    userRouterGroup.POST("/get_users_status", u.GetUserStatus)
    userRouterGroup.POST("/get_subscribe_users_status", u.GetSubscribeUsersStatus)

    userRouterGroup.POST("/process_user_command_add", u.ProcessUserCommandAdd)
    userRouterGroup.POST("/process_user_command_delete", u.ProcessUserCommandDelete)
    userRouterGroup.POST("/process_user_command_update", u.ProcessUserCommandUpdate)
    userRouterGroup.POST("/process_user_command_get", u.ProcessUserCommandGet)
    userRouterGroup.POST("/process_user_command_get_all", u.ProcessUserCommandGetAll)

    userRouterGroup.POST("/add_notification_account", u.AddNotificationAccount)
    userRouterGroup.POST("/update_notification_account", u.UpdateNotificationAccountInfo)
    userRouterGroup.POST("/search_notification_account", u.SearchNotificationAccount)

    userRouterGroup.POST("/get_user_client_config", u.GetUserClientConfig)
    userRouterGroup.POST("/set_user_client_config", u.SetUserClientConfig)
    userRouterGroup.POST("/del_user_client_config", u.DelUserClientConfig)
    userRouterGroup.POST("/page_user_client_config", u.PageUserClientConfig)
  }
  // friend routing group
  {
    f := NewFriendApi(relation.NewFriendClient(friendConn))
    friendRouterGroup := r.Group("/friend")
    friendRouterGroup.POST("/delete_friend", f.DeleteFriend)
    friendRouterGroup.POST("/get_friend_apply_list", f.GetFriendApplyList)
    friendRouterGroup.POST("/get_designated_friend_apply", f.GetDesignatedFriendsApply)
    friendRouterGroup.POST("/get_self_friend_apply_list", f.GetSelfApplyList)
    friendRouterGroup.POST("/get_friend_list", f.GetFriendList)
    friendRouterGroup.POST("/get_designated_friends", f.GetDesignatedFriends)
    friendRouterGroup.POST("/add_friend", f.ApplyToAddFriend)
    friendRouterGroup.POST("/add_friend_response", f.RespondFriendApply)
    friendRouterGroup.POST("/set_friend_remark", f.SetFriendRemark)
    friendRouterGroup.POST("/add_black", f.AddBlack)
    friendRouterGroup.POST("/get_black_list", f.GetPaginationBlacks)
    friendRouterGroup.POST("/get_specified_blacks", f.GetSpecifiedBlacks)
    friendRouterGroup.POST("/remove_black", f.RemoveBlack)
    friendRouterGroup.POST("/get_incremental_blacks", f.GetIncrementalBlacks)
    friendRouterGroup.POST("/import_friend", f.ImportFriends)
    friendRouterGroup.POST("/is_friend", f.IsFriend)
    friendRouterGroup.POST("/get_friend_id", f.GetFriendIDs)
    friendRouterGroup.POST("/get_specified_friends_info", f.GetSpecifiedFriendsInfo)
    friendRouterGroup.POST("/update_friends", f.UpdateFriends)
    friendRouterGroup.POST("/get_incremental_friends", f.GetIncrementalFriends)
    friendRouterGroup.POST("/get_full_friend_user_ids", f.GetFullFriendUserIDs)
    friendRouterGroup.POST("/get_self_unhandled_apply_count", f.GetSelfUnhandledApplyCount)
  }

  g := NewGroupApi(group.NewGroupClient(groupConn))
  {
    groupRouterGroup := r.Group("/group")
    groupRouterGroup.POST("/create_group", g.CreateGroup)
    groupRouterGroup.POST("/set_group_info", g.SetGroupInfo)
    groupRouterGroup.POST("/set_group_info_ex", g.SetGroupInfoEx)
    groupRouterGroup.POST("/join_group", g.JoinGroup)
    groupRouterGroup.POST("/quit_group", g.QuitGroup)
    groupRouterGroup.POST("/group_application_response", g.ApplicationGroupResponse)
    groupRouterGroup.POST("/transfer_group", g.TransferGroupOwner)
    groupRouterGroup.POST("/get_recv_group_applicationList", g.GetRecvGroupApplicationList)
    groupRouterGroup.POST("/get_user_req_group_applicationList", g.GetUserReqGroupApplicationList)
    groupRouterGroup.POST("/get_group_users_req_application_list", g.GetGroupUsersReqApplicationList)
    groupRouterGroup.POST("/get_specified_user_group_request_info", g.GetSpecifiedUserGroupRequestInfo)
    groupRouterGroup.POST("/get_groups_info", g.GetGroupsInfo)
    groupRouterGroup.POST("/kick_group", g.KickGroupMember)
    groupRouterGroup.POST("/get_group_members_info", g.GetGroupMembersInfo)
```

```
    groupRouterGroup.POST("/get_group_member_list", g.GetGroupMemberList)
    groupRouterGroup.POST("/invite_user_to_group", g.InviteUserToGroup)
    groupRouterGroup.POST("/get_joined_group_list", g.GetJoinedGroupList)
    groupRouterGroup.POST("/dismiss_group", g.DismissGroup) //
    groupRouterGroup.POST("/mute_group_member", g.MuteGroupMember)
    groupRouterGroup.POST("/cancel_mute_group_member", g.CancelMuteGroupMember)
    groupRouterGroup.POST("/mute_group", g.MuteGroup)
    groupRouterGroup.POST("/cancel_mute_group", g.CancelMuteGroup)
    groupRouterGroup.POST("/set_group_member_info", g.SetGroupMemberInfo)
    groupRouterGroup.POST("/get_group_abstract_info", g.GetGroupAbstractInfo)
    groupRouterGroup.POST("/get_groups", g.GetGroups)
    groupRouterGroup.POST("/get_group_member_user_id", g.GetGroupMemberUserIDs)
    groupRouterGroup.POST("/get_incremental_join_groups", g.GetIncrementalJoinGroup)
    groupRouterGroup.POST("/get_incremental_group_members", g.GetIncrementalGroupMember)
    groupRouterGroup.POST("/get_incremental_group_members_batch", g.GetIncrementalGroupMemberBatch)
    groupRouterGroup.POST("/get_full_group_member_user_ids", g.GetFullGroupMemberUserIDs)
    groupRouterGroup.POST("/get_full_join_group_ids", g.GetFullJoinGroupIDs)
    groupRouterGroup.POST("/get_group_application_unhandled_count", g.GetGroupApplicationUnhandledCount)
  }
  // certificate
  {
    a := NewAuthApi(pbAuth.NewAuthClient(authConn))
    authRouterGroup := r.Group("/auth")
    authRouterGroup.POST("/get_admin_token", a.GetAdminToken)
    authRouterGroup.POST("/get_user_token", a.GetUserToken)
    authRouterGroup.POST("/parse_token", a.ParseToken)
    authRouterGroup.POST("/force_logout", a.ForceLogout)

  }
  // Third service
  {
    t := NewThirdApi(third.NewThirdClient(thirdConn), cfg.API.Prometheus.GrafanaURL)
    thirdGroup := r.Group("/third")
    thirdGroup.GET("/prometheus", t.GetPrometheus)
    thirdGroup.POST("/fcm_update_token", t.FcmUpdateToken)
    thirdGroup.POST("/set_app_badge", t.SetAppBadge)

    logs := thirdGroup.Group("/logs")
    logs.POST("/upload", t.UploadLogs)
    logs.POST("/delete", t.DeleteLogs)
    logs.POST("/search", t.SearchLogs)

    objectGroup := r.Group("/object")

    objectGroup.POST("/part_limit", t.PartLimit)
    objectGroup.POST("/part_size", t.PartSize)
    objectGroup.POST("/initiate_multipart_upload", t.InitiateMultipartUpload)
    objectGroup.POST("/auth_sign", t.AuthSign)
    objectGroup.POST("/complete_multipart_upload", t.CompleteMultipartUpload)
    objectGroup.POST("/access_url", t.AccessURL)
    objectGroup.POST("/initiate_form_data", t.InitiateFormData)
    objectGroup.POST("/complete_form_data", t.CompleteFormData)
    objectGroup.GET("/*name", t.ObjectRedirect)
  }
  // Message
  m := NewMessageApi(msg.NewMsgClient(msgConn), rpcli.NewUserClient(userConn), cfg.Share.IMAdminUser.UserIDs)
  {
    msgGroup := r.Group("/msg")
    msgGroup.POST("/newest_seq", m.GetSeq)
    msgGroup.POST("/search_msg", m.SearchMsg)
    msgGroup.POST("/send_msg", m.SendMessage)
    msgGroup.POST("/send_business_notification", m.SendBusinessNotification)
    msgGroup.POST("/pull_msg_by_seq", m.PullMsgBySeqs)
    msgGroup.POST("/revoke_msg", m.RevokeMsg)
    msgGroup.POST("/mark_msgs_as_read", m.MarkMsgsAsRead)
    msgGroup.POST("/mark_conversation_as_read", m.MarkConversationAsRead)
```

```
    msgGroup.POST("/get_conversations_has_read_and_max_seq", m.GetConversationsHasReadAndMaxSeq)
    msgGroup.POST("/set_conversation_has_read_seq", m.SetConversationHasReadSeq)

    msgGroup.POST("/clear_conversation_msg", m.ClearConversationsMsg)
    msgGroup.POST("/user_clear_all_msg", m.UserClearAllMsg)
    msgGroup.POST("/delete_msgs", m.DeleteMsgs)
    msgGroup.POST("/delete_msg_phsical_by_seq", m.DeleteMsgPhysicalBySeq)
    msgGroup.POST("/delete_msg_physical", m.DeleteMsgPhysical)

    msgGroup.POST("/batch_send_msg", m.BatchSendMsg)
    msgGroup.POST("/send_simple_msg", m.SendSimpleMessage)
    msgGroup.POST("/check_msg_is_send_success", m.CheckMsgIsSendSuccess)
    msgGroup.POST("/get_server_time", m.GetServerTime)
  }
  // Conversation
  {
    c := NewConversationApi(conversation.NewConversationClient(conversationConn))
    conversationGroup := r.Group("/conversation")
    conversationGroup.POST("/get_sorted_conversation_list", c.GetSortedConversationList)
    conversationGroup.POST("/get_all_conversations", c.GetAllConversations)
    conversationGroup.POST("/get_conversation", c.GetConversation)
    conversationGroup.POST("/get_conversations", c.GetConversations)
    conversationGroup.POST("/set_conversations", c.SetConversations)
    //conversationGroup.POST("/get_conversation_offline_push_user_ids", c.GetConversationOfflinePushUserIDs)
    conversationGroup.POST("/get_full_conversation_ids", c.GetFullOwnerConversationIDs)
    conversationGroup.POST("/get_incremental_conversations", c.GetIncrementalConversation)
    conversationGroup.POST("/get_owner_conversation", c.GetOwnerConversation)
    conversationGroup.POST("/get_not_notify_conversation_ids", c.GetNotNotifyConversationIDs)
    conversationGroup.POST("/get_pinned_conversation_ids", c.GetPinnedConversationIDs)
    conversationGroup.POST("/delete_conversations", c.DeleteConversations)
    conversationGroup.POST("/update_conversations_by_user", c.UpdateConversationsByUser)
  }

  {
    statisticsGroup := r.Group("/statistics")
    statisticsGroup.POST("/user/register", u.UserRegisterCount)
    statisticsGroup.POST("/user/active", m.GetActiveUser)
    statisticsGroup.POST("/group/create", g.GroupCreateCount)
    statisticsGroup.POST("/group/active", m.GetActiveGroup)
  }

  {
    j := jssdk.NewJSSdkApi(rpcli.NewUserClient(userConn), rpcli.NewRelationClient(friendConn),
      rpcli.NewGroupClient(groupConn), rpcli.NewConversationClient(conversationConn), rpcli.NewMsgClient(msgConn))
    jssdk := r.Group("/jssdk")
    jssdk.POST("/get_conversations", j.GetConversations)
    jssdk.POST("/get_active_conversations", j.GetActiveConversations)
  }
  {
    pd := NewPrometheusDiscoveryApi(cfg, client)
    proDiscoveryGroup := r.Group("/prometheus_discovery")
    proDiscoveryGroup.GET("/api", pd.Api)
    proDiscoveryGroup.GET("/user", pd.User)
    proDiscoveryGroup.GET("/group", pd.Group)
    proDiscoveryGroup.GET("/msg", pd.Msg)
    proDiscoveryGroup.GET("/friend", pd.Friend)
    proDiscoveryGroup.GET("/conversation", pd.Conversation)
    proDiscoveryGroup.GET("/third", pd.Third)
    proDiscoveryGroup.GET("/auth", pd.Auth)
    proDiscoveryGroup.GET("/push", pd.Push)
    proDiscoveryGroup.GET("/msg_gateway", pd.MessageGateway)
    proDiscoveryGroup.GET("/msg_transfer", pd.MessageTransfer)
  }

  var etcdClient *clientv3.Client
  if cfg.Discovery.Enable == config.ETCD {
```

```
    etcdClient = client.(*etcd.SvcDiscoveryRegistryImpl).GetClient()
  }
  cm := NewConfigManager(cfg.Share.IMAdminUser.UserIDs, &cfg.AllConfig, etcdClient, string(cfg.ConfigPath))
  {
    configGroup := r.Group("/config", cm.CheckAdmin)
    configGroup.POST("/get_config_list", cm.GetConfigList)
    configGroup.POST("/get_config", cm.GetConfig)
    configGroup.POST("/set_config", cm.SetConfig)
    configGroup.POST("/reset_config", cm.ResetConfig)
    configGroup.POST("/set_enable_config_manager", cm.SetEnableConfigManager)
    configGroup.POST("/get_enable_config_manager", cm.GetEnableConfigManager)
  }
  {
    r.POST("/restart", cm.CheckAdmin, cm.Restart)
  }
  return r, nil
}

func GinParseToken(authClient *rpcli.AuthClient) gin.HandlerFunc {
  return func(c *gin.Context) {
    switch c.Request.Method {
    case http.MethodPost:
      for _, wApi := range Whitelist {
        if strings.HasPrefix(c.Request.URL.Path, wApi) {
          c.Next()
          return
        }
      }

      token := c.Request.Header.Get(constant.Token)
      if token == "" {
        log.ZWarn(c, "header get token error", servererrs.ErrArgs.WrapMsg("header must have token"))
        apiresp.GinError(c, servererrs.ErrArgs.WrapMsg("header must have token"))
        c.Abort()
        return
      }
      resp, err := authClient.ParseToken(c, token)
      if err != nil {
        apiresp.GinError(c, err)
        c.Abort()
        return
      }
      c.Set(constant.OpUserPlatform, constant.PlatformIDToName(int(resp.PlatformID)))
      c.Set(constant.OpUserID, resp.UserID)
      c.Next()
    }
  }
}

func setGinIsAdmin(imAdminUserID []string) gin.HandlerFunc {
  return func(c *gin.Context) {
    c.Set(authverify.CtxAdminUserIDsKey, imAdminUserID)
  }
}

// Whitelist api not parse token
var Whitelist = []string{
  "/auth/get_admin_token",
  "/auth/parse_token",
}
```

internal/api/third.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
    "context"
    "google.golang.org/grpc"
    "math/rand"
    "net/http"
    "net/url"
    "strconv"
    "strings"

    "github.com/gin-gonic/gin"
    "github.com/openimsdk/protocol/third"
    "github.com/openimsdk/tools/a2r"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/mcontext"
)

type ThirdApi struct {
    GrafanaUrl string
    Client     third.ThirdClient
}

func NewThirdApi(client third.ThirdClient, grafanaUrl string) ThirdApi {
    return ThirdApi{Client: client, GrafanaUrl: grafanaUrl}
}

func (o *ThirdApi) FcmUpdateToken(c *gin.Context) {
    a2r.Call(c, third.ThirdClient.FcmUpdateToken, o.Client)
}

func (o *ThirdApi) SetAppBadge(c *gin.Context) {
    a2r.Call(c, third.ThirdClient.SetAppBadge, o.Client)
}

// ###################### s3 ######################

func setURLPrefixOption[A, B, C any](_ func(client C, ctx context.Context, req *A, options ...grpc.CallOption) (*B,
    return &a2r.Option[A, B]{
        BindAfter: fn,
    }
}

func setURLPrefix(c *gin.Context, urlPrefix *string) error {
    host := c.GetHeader("X-Request-Api")
    if host != "" {
        if strings.HasSuffix(host, "/") {
            *urlPrefix = host + "object/"
            return nil
        } else {
```

```
        *urlPrefix = host + "/object/"
        return nil
    }
}
u := url.URL{
    Scheme: "http",
    Host:   c.Request.Host,
    Path:   "/object/",
}
if c.Request.TLS != nil {
    u.Scheme = "https"
}
*urlPrefix = u.String()
return nil
}

func (o *ThirdApi) PartLimit(c *gin.Context) {
a2r.Call(c, third.ThirdClient.PartLimit, o.Client)
}

func (o *ThirdApi) PartSize(c *gin.Context) {
a2r.Call(c, third.ThirdClient.PartSize, o.Client)
}

func (o *ThirdApi) InitiateMultipartUpload(c *gin.Context) {
opt := setURLPrefixOption(third.ThirdClient.InitiateMultipartUpload, func(req *third.InitiateMultipartUploadReq) er
    return setURLPrefix(c, &req.UrlPrefix)
})
a2r.Call(c, third.ThirdClient.InitiateMultipartUpload, o.Client, opt)
}

func (o *ThirdApi) AuthSign(c *gin.Context) {
a2r.Call(c, third.ThirdClient.AuthSign, o.Client)
}

func (o *ThirdApi) CompleteMultipartUpload(c *gin.Context) {
opt := setURLPrefixOption(third.ThirdClient.CompleteMultipartUpload, func(req *third.CompleteMultipartUploadReq) er
    return setURLPrefix(c, &req.UrlPrefix)
})
a2r.Call(c, third.ThirdClient.CompleteMultipartUpload, o.Client, opt)
}

func (o *ThirdApi) AccessURL(c *gin.Context) {
a2r.Call(c, third.ThirdClient.AccessURL, o.Client)
}

func (o *ThirdApi) InitiateFormData(c *gin.Context) {
a2r.Call(c, third.ThirdClient.InitiateFormData, o.Client)
}

func (o *ThirdApi) CompleteFormData(c *gin.Context) {
opt := setURLPrefixOption(third.ThirdClient.CompleteFormData, func(req *third.CompleteFormDataReq) error {
    return setURLPrefix(c, &req.UrlPrefix)
})
a2r.Call(c, third.ThirdClient.CompleteFormData, o.Client, opt)
}

func (o *ThirdApi) ObjectRedirect(c *gin.Context) {
name := c.Param("name")
if name == "" {
    c.String(http.StatusBadRequest, "name is empty")
    return
}
if name[0] == '/' {
    name = name[1:]
}
```

```
	operationID := c.Query("operationID")
	if operationID == "" {
		operationID = strconv.Itoa(rand.Int())
	}
	ctx := mcontext.SetOperationID(c, operationID)
	query := make(map[string]string)
	for key, values := range c.Request.URL.Query() {
		if len(values) == 0 {
			continue
		}
		query[key] = values[0]
	}
	resp, err := o.Client.AccessURL(ctx, &third.AccessURLReq{Name: name, Query: query})
	if err != nil {
		if errs.ErrArgs.Is(err) {
			c.String(http.StatusBadRequest, err.Error())
			return
		}
		if errs.ErrRecordNotFound.Is(err) {
			c.String(http.StatusNotFound, err.Error())
			return
		}
		c.String(http.StatusInternalServerError, err.Error())
		return
	}
	c.Redirect(http.StatusFound, resp.Url)
}

// ###################### logs ######################.
func (o *ThirdApi) UploadLogs(c *gin.Context) {
	a2r.Call(c, third.ThirdClient.UploadLogs, o.Client)
}

func (o *ThirdApi) DeleteLogs(c *gin.Context) {
	a2r.Call(c, third.ThirdClient.DeleteLogs, o.Client)
}

func (o *ThirdApi) SearchLogs(c *gin.Context) {
	a2r.Call(c, third.ThirdClient.SearchLogs, o.Client)
}

func (o *ThirdApi) GetPrometheus(c *gin.Context) {
	c.Redirect(http.StatusFound, o.GrafanaUrl)
}
```

internal/api/user.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package api

import (
	"github.com/gin-gonic/gin"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msggateway"
	"github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/a2r"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
)

type UserApi struct {
	Client user.UserClient
	discov discovery.Conn
	config config.RpcService
}

func NewUserApi(client user.UserClient, discov discovery.Conn, config config.RpcService) UserApi {
	return UserApi{Client: client, discov: discov, config: config}
}

func (u *UserApi) UserRegister(c *gin.Context) {
	a2r.Call(c, user.UserClient.UserRegister, u.Client)
}

// UpdateUserInfo is deprecated. Use UpdateUserInfoEx
func (u *UserApi) UpdateUserInfo(c *gin.Context) {
	a2r.Call(c, user.UserClient.UpdateUserInfo, u.Client)
}

func (u *UserApi) UpdateUserInfoEx(c *gin.Context) {
	a2r.Call(c, user.UserClient.UpdateUserInfoEx, u.Client)
}
func (u *UserApi) SetGlobalRecvMessageOpt(c *gin.Context) {
	a2r.Call(c, user.UserClient.SetGlobalRecvMessageOpt, u.Client)
}

func (u *UserApi) GetUsersPublicInfo(c *gin.Context) {
	a2r.Call(c, user.UserClient.GetDesignateUsers, u.Client)
}

func (u *UserApi) GetAllUsersID(c *gin.Context) {
	a2r.Call(c, user.UserClient.GetAllUserID, u.Client)
}

func (u *UserApi) AccountCheck(c *gin.Context) {
```

```
	a2r.Call(c, user.UserClient.AccountCheck, u.Client)
}

func (u *UserApi) GetUsers(c *gin.Context) {
	a2r.Call(c, user.UserClient.GetPaginationUsers, u.Client)
}

// GetUsersOnlineStatus Get user online status.
func (u *UserApi) GetUsersOnlineStatus(c *gin.Context) {
	var req msggateway.GetUsersOnlineStatusReq
	if err := c.BindJSON(&req); err != nil {
		apiresp.GinError(c, err)
		return
	}
	conns, err := u.discov.GetConns(c, u.config.MessageGateway)
	if err != nil {
		apiresp.GinError(c, err)
		return
	}

	var wsResult []*msggateway.GetUsersOnlineStatusResp_SuccessResult
	var respResult []*msggateway.GetUsersOnlineStatusResp_SuccessResult
	flag := false

	// Online push message
	for _, v := range conns {
		msgClient := msggateway.NewMsgGatewayClient(v)
		reply, err := msgClient.GetUsersOnlineStatus(c, &req)
		if err != nil {
			log.ZDebug(c, "GetUsersOnlineStatus rpc error", err)

			parseError := apiresp.ParseError(err)
			if parseError.ErrCode == errs.NoPermissionError {
				apiresp.GinError(c, err)
				return
			}
		} else {
			wsResult = append(wsResult, reply.SuccessResult...)
		}
	}
	// Traversing the userIDs in the api request body
	for _, v1 := range req.UserIDs {
		flag = false
		res := new(msggateway.GetUsersOnlineStatusResp_SuccessResult)
		// Iterate through the online results fetched from various gateways
		for _, v2 := range wsResult {
			// If matches the above description on the line, and vice versa
			if v2.UserID == v1 {
				flag = true
				res.UserID = v1
				res.Status = constant.Online
				res.DetailPlatformStatus = append(res.DetailPlatformStatus, v2.DetailPlatformStatus...)
				break
			}
		}
		if !flag {
			res.UserID = v1
			res.Status = constant.Offline
		}
		respResult = append(respResult, res)
	}
	apiresp.GinSuccess(c, respResult)
}

func (u *UserApi) UserRegisterCount(c *gin.Context) {
	a2r.Call(c, user.UserClient.UserRegisterCount, u.Client)
```

```
}

// GetUsersOnlineTokenDetail Get user online token details.
func (u *UserApi) GetUsersOnlineTokenDetail(c *gin.Context) {
    var wsResult []*msggateway.GetUsersOnlineStatusResp_SuccessResult
    var respResult []*msggateway.SingleDetail
    flag := false
    var req msggateway.GetUsersOnlineStatusReq
    if err := c.BindJSON(&req); err != nil {
        apiresp.GinError(c, errs.ErrArgs.WithDetail(err.Error()).Wrap())
        return
    }
    conns, err := u.discov.GetConns(c, u.config.MessageGateway)
    if err != nil {
        apiresp.GinError(c, err)
        return
    }
    // Online push message
    for _, v := range conns {
        msgClient := msggateway.NewMsgGatewayClient(v)
        reply, err := msgClient.GetUsersOnlineStatus(c, &req)
        if err != nil {
            log.ZWarn(c, "GetUsersOnlineStatus rpc err", err)
            continue
        } else {
            wsResult = append(wsResult, reply.SuccessResult...)
        }
    }

    for _, v1 := range req.UserIDs {
        m := make(map[int32][]string, 10)
        flag = false
        temp := new(msggateway.SingleDetail)
        for _, v2 := range wsResult {
            if v2.UserID == v1 {
                flag = true
                temp.UserID = v1
                temp.Status = constant.Online
                for _, status := range v2.DetailPlatformStatus {
                    if v, ok := m[status.PlatformID]; ok {
                        m[status.PlatformID] = append(v, status.Token)
                    } else {
                        m[status.PlatformID] = []string{status.Token}
                    }
                }
            }
        }
        for p, tokens := range m {
            t := new(msggateway.SinglePlatformToken)
            t.PlatformID = p
            t.Token = tokens
            t.Total = int32(len(tokens))
            temp.SinglePlatformToken = append(temp.SinglePlatformToken, t)
        }

        if flag {
            respResult = append(respResult, temp)
        }
    }

    apiresp.GinSuccess(c, respResult)
}

// SubscriberStatus Presence status of subscribed users.
func (u *UserApi) SubscriberStatus(c *gin.Context) {
    a2r.Call(c, user.UserClient.SubscribeOrCancelUsersStatus, u.Client)
```

```
}

// GetUserStatus Get the online status of the user.
func (u *UserApi) GetUserStatus(c *gin.Context) {
    a2r.Call(c, user.UserClient.GetUserStatus, u.Client)
}

// GetSubscribeUsersStatus Get the online status of subscribers.
func (u *UserApi) GetSubscribeUsersStatus(c *gin.Context) {
    a2r.Call(c, user.UserClient.GetSubscribeUsersStatus, u.Client)
}

// ProcessUserCommandAdd user general function add.
func (u *UserApi) ProcessUserCommandAdd(c *gin.Context) {
    a2r.Call(c, user.UserClient.ProcessUserCommandAdd, u.Client)
}

// ProcessUserCommandDelete user general function delete.
func (u *UserApi) ProcessUserCommandDelete(c *gin.Context) {
    a2r.Call(c, user.UserClient.ProcessUserCommandDelete, u.Client)
}

// ProcessUserCommandUpdate  user general function update.
func (u *UserApi) ProcessUserCommandUpdate(c *gin.Context) {
    a2r.Call(c, user.UserClient.ProcessUserCommandUpdate, u.Client)
}

// ProcessUserCommandGet user general function get.
func (u *UserApi) ProcessUserCommandGet(c *gin.Context) {
    a2r.Call(c, user.UserClient.ProcessUserCommandGet, u.Client)
}

// ProcessUserCommandGet user general function get all.
func (u *UserApi) ProcessUserCommandGetAll(c *gin.Context) {
    a2r.Call(c, user.UserClient.ProcessUserCommandGetAll, u.Client)
}

func (u *UserApi) AddNotificationAccount(c *gin.Context) {
    a2r.Call(c, user.UserClient.AddNotificationAccount, u.Client)
}

func (u *UserApi) UpdateNotificationAccountInfo(c *gin.Context) {
    a2r.Call(c, user.UserClient.UpdateNotificationAccountInfo, u.Client)
}

func (u *UserApi) SearchNotificationAccount(c *gin.Context) {
    a2r.Call(c, user.UserClient.SearchNotificationAccount, u.Client)
}

func (u *UserApi) GetUserClientConfig(c *gin.Context) {
    a2r.Call(c, user.UserClient.GetUserClientConfig, u.Client)
}

func (u *UserApi) SetUserClientConfig(c *gin.Context) {
    a2r.Call(c, user.UserClient.SetUserClientConfig, u.Client)
}

func (u *UserApi) DelUserClientConfig(c *gin.Context) {
    a2r.Call(c, user.UserClient.DelUserClientConfig, u.Client)
}

func (u *UserApi) PageUserClientConfig(c *gin.Context) {
    a2r.Call(c, user.UserClient.PageUserClientConfig, u.Client)
}
```

internal/api/jssdk

internal/api/jssdk/jssdk.go

```
package jssdk

import (
	"context"
	"sort"

	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/log"

	"github.com/gin-gonic/gin"

	"github.com/openimsdk/protocol/conversation"
	"github.com/openimsdk/protocol/jssdk"
	"github.com/openimsdk/protocol/msg"
	"github.com/openimsdk/protocol/relation"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
)

const (
	maxGetActiveConversation     = 500
	defaultGetActiveConversation = 100
)

func NewJSSdkApi(userClient *rpcli.UserClient, relationClient *rpcli.RelationClient, groupClient *rpcli.GroupClient,
	conversationClient *rpcli.ConversationClient, msgClient *rpcli.MsgClient) *JSSdk {
	return &JSSdk{
		userClient:         userClient,
		relationClient:     relationClient,
		groupClient:        groupClient,
		conversationClient: conversationClient,
		msgClient:          msgClient,
	}
}

type JSSdk struct {
	userClient         *rpcli.UserClient
	relationClient     *rpcli.RelationClient
	groupClient        *rpcli.GroupClient
	conversationClient *rpcli.ConversationClient
	msgClient          *rpcli.MsgClient
}

func (x *JSSdk) GetActiveConversations(c *gin.Context) {
	call(c, x.getActiveConversations)
}

func (x *JSSdk) GetConversations(c *gin.Context) {
	call(c, x.getConversations)
}

func (x *JSSdk) fillConversations(ctx context.Context, conversations []*jssdk.ConversationMsg) error {
	if len(conversations) == 0 {
		return nil
	}
	var (
		userIDs  []string
		groupIDs []string
	)
```

```
	for _, c := range conversations {
		if c.Conversation.GroupID == "" {
			userIDs = append(userIDs, c.Conversation.UserID)
		} else {
			groupIDs = append(groupIDs, c.Conversation.GroupID)
		}
	}
	var (
		userMap   map[string]*sdkws.UserInfo
		friendMap map[string]*relation.FriendInfoOnly
		groupMap  map[string]*sdkws.GroupInfo
	)
	if len(userIDs) > 0 {
		users, err := x.userClient.GetUsersInfo(ctx, userIDs)
		if err != nil {
			return err
		}
		friends, err := x.relationClient.GetFriendsInfo(ctx, conversations[0].Conversation.OwnerUserID, userIDs)
		if err != nil {
			return err
		}
		userMap = datautil.SliceToMap(users, (*sdkws.UserInfo).GetUserID)
		friendMap = datautil.SliceToMap(friends, (*relation.FriendInfoOnly).GetFriendUserID)
	}
	if len(groupIDs) > 0 {
		groups, err := x.groupClient.GetGroupsInfo(ctx, groupIDs)
		if err != nil {
			return err
		}
		groupMap = datautil.SliceToMap(groups, (*sdkws.GroupInfo).GetGroupID)
	}
	for _, c := range conversations {
		if c.Conversation.GroupID == "" {
			c.User = userMap[c.Conversation.UserID]
			c.Friend = friendMap[c.Conversation.UserID]
		} else {
			c.Group = groupMap[c.Conversation.GroupID]
		}
	}
	return nil
}

func (x *JSSdk) getActiveConversations(ctx context.Context, req *jssdk.GetActiveConversationsReq) (*jssdk.GetActiveC
	if req.Count <= 0 || req.Count > maxGetActiveConversation {
		req.Count = defaultGetActiveConversation
	}
	req.OwnerUserID = mcontext.GetOpUserID(ctx)
	conversationIDs, err := x.conversationClient.GetConversationIDs(ctx, req.OwnerUserID)
	if err != nil {
		return nil, err
	}
	if len(conversationIDs) == 0 {
		return &jssdk.GetActiveConversationsResp{}, nil
	}

	activeConversation, err := x.msgClient.GetActiveConversation(ctx, conversationIDs)
	if err != nil {
		return nil, err
	}
	if len(activeConversation) == 0 {
		return &jssdk.GetActiveConversationsResp{}, nil
	}
	readSeq, err := x.msgClient.GetHasReadSeqs(ctx, conversationIDs, req.OwnerUserID)
	if err != nil {
		return nil, err
	}
```

```
	sortConversations := sortActiveConversations{
		Conversation: activeConversation,
	}
	if len(activeConversation) > 1 {
		pinnedConversationIDs, err := x.conversationClient.GetPinnedConversationIDs(ctx, req.OwnerUserID)
		if err != nil {
			return nil, err
		}
		sortConversations.PinnedConversationIDs = datautil.SliceSet(pinnedConversationIDs)
	}
	sort.Sort(&sortConversations)
	sortList := sortConversations.Top(int(req.Count))
	conversations, err := x.conversationClient.GetConversations(ctx, datautil.Slice(sortList, func(c *msg.ActiveConvers
		return c.ConversationID
	}), req.OwnerUserID)
	if err != nil {
		return nil, err
	}
	msgs, err := x.msgClient.GetSeqMessage(ctx, req.OwnerUserID, datautil.Slice(sortList, func(c *msg.ActiveConversatio
		return &msg.ConversationSeqs{
			ConversationID: c.ConversationID,
			Seqs:           []int64{c.MaxSeq},
		}
	}))
	if err != nil {
		return nil, err
	}
	x.checkMessagesAndGetLastMessage(ctx, req.OwnerUserID, msgs)
	conversationMap := datautil.SliceToMap(conversations, func(c *conversation.Conversation) string {
		return c.ConversationID
	})
	resp := make([]*jssdk.ConversationMsg, 0, len(sortList))
	for _, c := range sortList {
		conv, ok := conversationMap[c.ConversationID]
		if !ok {
			continue
		}
		if msgList, ok := msgs[c.ConversationID]; ok && len(msgList.Msgs) > 0 {
			resp = append(resp, &jssdk.ConversationMsg{
				Conversation: conv,
				LastMsg:      msgList.Msgs[0],
				MaxSeq:       c.MaxSeq,
				ReadSeq:      readSeq[c.ConversationID],
			})
		}

	}
	if err := x.fillConversations(ctx, resp); err != nil {
		return nil, err
	}
	var unreadCount int64
	for _, c := range activeConversation {
		count := c.MaxSeq - readSeq[c.ConversationID]
		if count > 0 {
			unreadCount += count
		}
	}
	return &jssdk.GetActiveConversationsResp{
		Conversations: resp,
		UnreadCount:   unreadCount,
	}, nil
}

func (x *JSSdk) getConversations(ctx context.Context, req *jssdk.GetConversationsReq) (*jssdk.GetConversationsResp,
	req.OwnerUserID = mcontext.GetOpUserID(ctx)
	conversations, err := x.conversationClient.GetConversations(ctx, req.ConversationIDs, req.OwnerUserID)
```

```
	if err != nil {
		return nil, err
	}
	if len(conversations) == 0 {
		return &jssdk.GetConversationsResp{}, nil
	}
	req.ConversationIDs = datautil.Slice(conversations, func(c *conversation.Conversation) string {
		return c.ConversationID
	})
	maxSeqs, err := x.msgClient.GetMaxSeqs(ctx, req.ConversationIDs)
	if err != nil {
		return nil, err
	}
	readSeqs, err := x.msgClient.GetHasReadSeqs(ctx, req.ConversationIDs, req.OwnerUserID)
	if err != nil {
		return nil, err
	}
	conversationSeqs := make([]*msg.ConversationSeqs, 0, len(conversations))
	for _, c := range conversations {
		if seq := maxSeqs[c.ConversationID]; seq > 0 {
			conversationSeqs = append(conversationSeqs, &msg.ConversationSeqs{
				ConversationID: c.ConversationID,
				Seqs:           []int64{seq},
			})
		}
	}
	var msgs map[string]*sdkws.PullMsgs
	if len(conversationSeqs) > 0 {
		msgs, err = x.msgClient.GetSeqMessage(ctx, req.OwnerUserID, conversationSeqs)
		if err != nil {
			return nil, err
		}
	}
	x.checkMessagesAndGetLastMessage(ctx, req.OwnerUserID, msgs)
	resp := make([]*jssdk.ConversationMsg, 0, len(conversations))
	for _, c := range conversations {
		if msgList, ok := msgs[c.ConversationID]; ok && len(msgList.Msgs) > 0 {
			resp = append(resp, &jssdk.ConversationMsg{
				Conversation: c,
				LastMsg:      msgList.Msgs[0],
				MaxSeq:       maxSeqs[c.ConversationID],
				ReadSeq:      readSeqs[c.ConversationID],
			})
		}

	}
	if err := x.fillConversations(ctx, resp); err != nil {
		return nil, err
	}
	var unreadCount int64
	for conversationID, maxSeq := range maxSeqs {
		count := maxSeq - readSeqs[conversationID]
		if count > 0 {
			unreadCount += count
		}
	}
	return &jssdk.GetConversationsResp{
		Conversations: resp,
		UnreadCount:   unreadCount,
	}, nil
}

// This function checks whether the latest MaxSeq message is valid.
// If not, it needs to fetch a valid message again.
func (x *JSSdk) checkMessagesAndGetLastMessage(ctx context.Context, userID string, messages map[string]*sdkws.PullMs
	var conversationIDs []string
```

```
	for conversationID, message := range messages {
		allInValid := true
		for _, data := range message.Msgs {
			if data.Status < constant.MsgStatusHasDeleted {
				allInValid = false
				break
			}
		}
		if allInValid {
			conversationIDs = append(conversationIDs, conversationID)
		}
	}
	if len(conversationIDs) > 0 {
		resp, err := x.msgClient.GetLastMessage(ctx, &msg.GetLastMessageReq{
			UserID:          userID,
			ConversationIDs: conversationIDs,
		})
		if err != nil {
			log.ZError(ctx, "fetchLatestValidMessages", err, "conversationIDs", conversationIDs)
			return
		}
		for conversationID, message := range resp.Msgs {
			messages[conversationID] = &sdkws.PullMsgs{Msgs: []*sdkws.MsgData{message}}
		}
	}

}
```

internal/api/jssdk/sort.go

```
package jssdk

import "github.com/openimsdk/protocol/msg"

type sortActiveConversations struct {
	Conversation          []*msg.ActiveConversation
	PinnedConversationIDs map[string]struct{}
}

func (s sortActiveConversations) Top(limit int) []*msg.ActiveConversation {
	if limit > 0 && len(s.Conversation) > limit {
		return s.Conversation[:limit]
	}
	return s.Conversation
}

func (s sortActiveConversations) Len() int {
	return len(s.Conversation)
}

func (s sortActiveConversations) Less(i, j int) bool {
	iv, jv := s.Conversation[i], s.Conversation[j]
	_, ip := s.PinnedConversationIDs[iv.ConversationID]
	_, jp := s.PinnedConversationIDs[jv.ConversationID]
	if ip != jp {
		return ip
	}
	return iv.LastTime > jv.LastTime
}

func (s sortActiveConversations) Swap(i, j int) {
	s.Conversation[i], s.Conversation[j] = s.Conversation[j], s.Conversation[i]
}
```

internal/api/jssdk/tools.go

```
package jssdk

import (
	"context"
	"github.com/gin-gonic/gin"
	"github.com/openimsdk/tools/a2r"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/checker"
	"github.com/openimsdk/tools/errs"
	"google.golang.org/grpc"
	"google.golang.org/protobuf/proto"
	"io"
	"strings"
)

func field[A, B, C any](ctx context.Context, fn func(ctx context.Context, req *A, opts ...grpc.CallOption) (*B, erro
	resp, err := fn(ctx, req)
	if err != nil {
		var c C
		return c, err
	}
	return get(resp), nil
}

func call[A, B any](c *gin.Context, fn func(ctx context.Context, req *A) (*B, error)) {
	var isJSON bool
	switch contentType := c.GetHeader("Content-Type"); {
	case contentType == "":
		isJSON = true
	case strings.Contains(contentType, "application/json"):
		isJSON = true
	case strings.Contains(contentType, "application/protobuf"):
	case strings.Contains(contentType, "application/x-protobuf"):
	default:
		apiresp.GinError(c, errs.ErrArgs.WrapMsg("unsupported content type"))
		return
	}
	var req *A
	if isJSON {
		var err error
		req, err = a2r.ParseRequest[A](c)
		if err != nil {
			apiresp.GinError(c, err)
			return
		}
	} else {
		body, err := io.ReadAll(c.Request.Body)
		if err != nil {
			apiresp.GinError(c, err)
			return
		}
		req = new(A)
		if err := proto.Unmarshal(body, any(req).(proto.Message)); err != nil {
			apiresp.GinError(c, err)
			return
		}
		if err := checker.Validate(&req); err != nil {
			apiresp.GinError(c, err)
			return
		}
	}
	resp, err := fn(c, req)
	if err != nil {
		apiresp.GinError(c, err)
```

```
    return
  }
  if isJSON {
    apiresp.GinSuccess(c, resp)
    return
  }
  body, err := proto.Marshal(any(resp).(proto.Message))
  if err != nil {
    apiresp.GinError(c, err)
    return
  }
  apiresp.GinSuccess(c, body)
}
```

internal/msggateway

internal/msggateway/callback.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
	"context"
	"time"

	cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/mcontext"
)

func (ws *WsServer) webhookAfterUserOnline(ctx context.Context, after *config.AfterConfig, userID string, platformID
	req := cbapi.CallbackUserOnlineReq{
		UserStatusCallbackReq: cbapi.UserStatusCallbackReq{
			UserStatusBaseCallback: cbapi.UserStatusBaseCallback{
				CallbackCommand: cbapi.CallbackAfterUserOnlineCommand,
				OperationID:     mcontext.GetOperationID(ctx),
				PlatformID:      platformID,
				Platform:        constant.PlatformIDToName(platformID),
			},
			UserID: userID,
		},
		Seq:              time.Now().UnixMilli(),
		IsAppBackground: isAppBackground,
		ConnID:           connID,
	}
	ws.webhookClient.AsyncPost(ctx, req.GetCallbackCommand(), req, &cbapi.CommonCallbackResp{}, after)
}

func (ws *WsServer) webhookAfterUserOffline(ctx context.Context, after *config.AfterConfig, userID string, platformI
	req := &cbapi.CallbackUserOfflineReq{
		UserStatusCallbackReq: cbapi.UserStatusCallbackReq{
			UserStatusBaseCallback: cbapi.UserStatusBaseCallback{
				CallbackCommand: cbapi.CallbackAfterUserOfflineCommand,
				OperationID:     mcontext.GetOperationID(ctx),
				PlatformID:      platformID,
				Platform:        constant.PlatformIDToName(platformID),
			},
			UserID: userID,
		},
		Seq:    time.Now().UnixMilli(),
		ConnID: connID,
	}
	ws.webhookClient.AsyncPost(ctx, req.GetCallbackCommand(), req, &cbapi.CallbackUserOfflineResp{}, after)
}
```

```
func (ws *WsServer) webhookAfterUserKickOff(ctx context.Context, after *config.AfterConfig, userID string, platformI
    req := &cbapi.CallbackUserKickOffReq{
        UserStatusCallbackReq: cbapi.UserStatusCallbackReq{
            UserStatusBaseCallback: cbapi.UserStatusBaseCallback{
                CallbackCommand: cbapi.CallbackAfterUserKickOffCommand,
                OperationID:     mcontext.GetOperationID(ctx),
                PlatformID:      platformID,
                Platform:        constant.PlatformIDToName(platformID),
            },
            UserID: userID,
        },
        Seq: time.Now().UnixMilli(),
    }
    ws.webhookClient.AsyncPost(ctx, req.GetCallbackCommand(), req, &cbapi.CommonCallbackResp{}, after)
}
```

internal/msggateway/client.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
	"context"
	"fmt"
	"sync"
	"sync/atomic"
	"time"

	"google.golang.org/protobuf/proto"

	"github.com/openimsdk/open-im-server/v3/pkg/msgprocessor"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/errs"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
)

var (
	ErrConnClosed                = errs.New("conn has closed")
	ErrNotSupportMessageProtocol = errs.New("not support message protocol")
	ErrClientClosed              = errs.New("client actively close the connection")
	ErrPanic                     = errs.New("panic error")
)

const (
	// MessageText is for UTF-8 encoded text messages like JSON.
	MessageText = iota + 1
	// MessageBinary is for binary messages like protobufs.
	MessageBinary
	// CloseMessage denotes a close control message. The optional message
	// payload contains a numeric code and text. Use the FormatCloseMessage
	// function to format a close message payload.
	CloseMessage = 8

	// PingMessage denotes a ping control message. The optional message payload
	// is UTF-8 encoded text.
	PingMessage = 9

	// PongMessage denotes a pong control message. The optional message payload
	// is UTF-8 encoded text.
	PongMessage = 10
)

type PingPongHandler func(string) error

type Client struct {
	w              *sync.Mutex
```

```
    conn           ClientConn
    PlatformID     int    `json:"platformID"`
    IsCompress     bool   `json:"isCompress"`
    UserID         string `json:"userID"`
    IsBackground   bool   `json:"isBackground"`
    SDKType        string `json:"sdkType"`
    SDKVersion     string `json:"sdkVersion"`
    Encoder        Encoder
    ctx            *UserConnContext
    longConnServer LongConnServer
    closed         atomic.Bool
    closedErr      error
    token          string
    hbCtx          context.Context
    hbCancel       context.CancelFunc
    subLock        *sync.Mutex
    subUserIDs     map[string]struct{} // client conn subscription list
}

// ResetClient updates the client's state with new connection and context information.
func (c *Client) ResetClient(ctx *UserConnContext, conn ClientConn, longConnServer LongConnServer) {
    c.w = new(sync.Mutex)
    c.conn = conn
    c.PlatformID = ctx.GetPlatformID()
    c.IsCompress = ctx.GetCompression()
    c.IsBackground = ctx.GetBackground()
    c.UserID = ctx.GetUserID()
    c.ctx = ctx
    c.longConnServer = longConnServer
    c.IsBackground = false
    c.closed.Store(false)
    c.closedErr = nil
    c.token = ctx.GetToken()
    c.SDKType = ctx.GetSDKType()
    c.SDKVersion = ctx.GetSDKVersion()
    c.hbCtx, c.hbCancel = context.WithCancel(c.ctx)
    c.subLock = new(sync.Mutex)
    if c.subUserIDs != nil {
        clear(c.subUserIDs)
    }
    if c.SDKType == GoSDK {
        c.Encoder = NewGobEncoder()
    } else {
        c.Encoder = NewJsonEncoder()
    }
    c.subUserIDs = make(map[string]struct{})
}

// readMessage continuously reads messages from the connection.
func (c *Client) readMessage() {
    defer func() {
        if r := recover(); r != nil {
            c.closedErr = ErrPanic
            log.ZPanic(c.ctx, "socket have panic err:", errs.ErrPanic(r))
        }
        c.close()
    }()

    for {
        log.ZDebug(c.ctx, "readMessage")
        message, returnErr := c.conn.ReadMessage()
        if returnErr != nil {
            log.ZWarn(c.ctx, "readMessage", returnErr)
            c.closedErr = returnErr
            return
        }
```

```
        if c.closed.Load() {
            // The scenario where the connection has just been closed, but the coroutine has not exited
            c.closedErr = ErrConnClosed
            return
        }

        parseDataErr := c.handleMessage(message)
        if parseDataErr != nil {
            c.closedErr = parseDataErr
            return
        }
    }
}

// handleMessage processes a single message received by the client.
func (c *Client) handleMessage(message []byte) error {
    if c.IsCompress {
        var err error
        message, err = c.longConnServer.DecompressWithPool(message)
        if err != nil {
            return errs.Wrap(err)
        }
    }

    var binaryReq = getReq()
    defer freeReq(binaryReq)

    err := c.Encoder.Decode(message, binaryReq)
    if err != nil {
        return err
    }

    if err := c.longConnServer.Validate(binaryReq); err != nil {
        return err
    }

    if binaryReq.SendID != c.UserID {
        return errs.New("exception conn userID not same to req userID", "binaryReq", binaryReq.String())
    }

    ctx := mcontext.WithMustInfoCtx(
        []string{binaryReq.OperationID, binaryReq.SendID, constant.PlatformIDToName(c.PlatformID), c.ctx.GetConnID()},
    )

    log.ZDebug(ctx, "gateway req message", "req", binaryReq.String())

    var (
        resp       []byte
        messageErr error
    )

    switch binaryReq.ReqIdentifier {
    case WSGetNewestSeq:
        resp, messageErr = c.longConnServer.GetSeq(ctx, binaryReq)
    case WSSendMsg:
        resp, messageErr = c.longConnServer.SendMessage(ctx, binaryReq)
    case WSSendSignalMsg:
        resp, messageErr = c.longConnServer.SendSignalMessage(ctx, binaryReq)
    case WSPullMsgBySeqList:
        resp, messageErr = c.longConnServer.PullMessageBySeqList(ctx, binaryReq)
    case WSPullMsg:
        resp, messageErr = c.longConnServer.GetSeqMessage(ctx, binaryReq)
    case WSGetConvMaxReadSeq:
        resp, messageErr = c.longConnServer.GetConversationsHasReadAndMaxSeq(ctx, binaryReq)
    case WsPullConvLastMessage:
```

```
	resp, messageErr = c.longConnServer.GetLastMessage(ctx, binaryReq)
	case WsLogoutMsg:
	resp, messageErr = c.longConnServer.UserLogout(ctx, binaryReq)
	case WsSetBackgroundStatus:
	resp, messageErr = c.setAppBackgroundStatus(ctx, binaryReq)
	case WsSubUserOnlineStatus:
	resp, messageErr = c.longConnServer.SubUserOnlineStatus(ctx, c, binaryReq)
	default:
	return fmt.Errorf(
	"ReqIdentifier failed,sendID:%s,msgIncr:%s,reqIdentifier:%d",
	binaryReq.SendID,
	binaryReq.MsgIncr,
	binaryReq.ReqIdentifier,
	)
	}

	return c.replyMessage(ctx, binaryReq, messageErr, resp)
}

func (c *Client) setAppBackgroundStatus(ctx context.Context, req *Req) ([]byte, error) {
	resp, isBackground, messageErr := c.longConnServer.SetUserDeviceBackground(ctx, req)
	if messageErr != nil {
	return nil, messageErr
	}

	c.IsBackground = isBackground
	// TODO: callback
	return resp, nil
}

func (c *Client) close() {
	c.w.Lock()
	defer c.w.Unlock()
	if c.closed.Load() {
	return
	}
	c.closed.Store(true)
	c.conn.Close()
	c.hbCancel() // Close server-initiated heartbeat.
	c.longConnServer.UnRegister(c)
}

func (c *Client) replyMessage(ctx context.Context, binaryReq *Req, err error, resp []byte) error {
	errResp := apiresp.ParseError(err)
	mReply := Resp{
	ReqIdentifier: binaryReq.ReqIdentifier,
	MsgIncr:       binaryReq.MsgIncr,
	OperationID:   binaryReq.OperationID,
	ErrCode:       errResp.ErrCode,
	ErrMsg:        errResp.ErrMsg,
	Data:          resp,
	}
	t := time.Now()
	log.ZDebug(ctx, "gateway reply message", "resp", mReply.String())
	err = c.writeBinaryMsg(mReply)
	if err != nil {
	log.ZWarn(ctx, "wireBinaryMsg replyMessage", err, "resp", mReply.String())
	}
	log.ZDebug(ctx, "wireBinaryMsg end", "time cost", time.Since(t))

	if binaryReq.ReqIdentifier == WsLogoutMsg {
	return errs.New("user logout", "operationID", binaryReq.OperationID).Wrap()
	}
	return nil
}
```

```
func (c *Client) PushMessage(ctx context.Context, msgData *sdkws.MsgData) error {
	var msg sdkws.PushMessages
	conversationID := msgprocessor.GetConversationIDByMsg(msgData)
	m := map[string]*sdkws.PullMsgs{conversationID: {Msgs: []*sdkws.MsgData{msgData}}}
	if msgprocessor.IsNotification(conversationID) {
		msg.NotificationMsgs = m
	} else {
		msg.Msgs = m
	}
	log.ZDebug(ctx, "PushMessage", "msg", &msg)
	data, err := proto.Marshal(&msg)
	if err != nil {
		return err
	}
	resp := Resp{
		ReqIdentifier: WSPushMsg,
		OperationID:   mcontext.GetOperationID(ctx),
		Data:          data,
	}
	return c.writeBinaryMsg(resp)
}

func (c *Client) KickOnlineMessage() error {
	resp := Resp{
		ReqIdentifier: WSKickOnlineMsg,
	}
	log.ZDebug(c.ctx, "KickOnlineMessage debug ")
	err := c.writeBinaryMsg(resp)
	c.close()
	return err
}

func (c *Client) PushUserOnlineStatus(data []byte) error {
	resp := Resp{
		ReqIdentifier: WsSubUserOnlineStatus,
		Data:          data,
	}
	return c.writeBinaryMsg(resp)
}

func (c *Client) writeBinaryMsg(resp Resp) error {
	if c.closed.Load() {
		return nil
	}

	encodedBuf, err := c.Encoder.Encode(resp)
	if err != nil {
		return err
	}

	c.w.Lock()
	defer c.w.Unlock()

	if c.IsCompress {
		resultBuf, compressErr := c.longConnServer.CompressWithPool(encodedBuf)
		if compressErr != nil {
			return compressErr
		}
		return c.conn.WriteMessage(resultBuf)
	}

	return c.conn.WriteMessage(encodedBuf)
}
```

internal/msggateway/client_conn.go

```
package msggateway

import (
	"context"
	"encoding/json"
	"errors"
	"fmt"
	"sync/atomic"
	"time"

	"github.com/gorilla/websocket"

	"github.com/openimsdk/tools/log"
)

var ErrWriteFull = fmt.Errorf("websocket write buffer full,close connection")

type ClientConn interface {
	ReadMessage() ([]byte, error)
	WriteMessage(message []byte) error
	Close() error
}

type websocketMessage struct {
	MessageType int
	Data        []byte
}

func NewWebSocketClientConn(conn *websocket.Conn, readLimit int64, readTimeout time.Duration, pingInterval time.Dura
	c := &websocketClientConn{
		readTimeout: readTimeout,
		conn:        conn,
		writer:      make(chan *websocketMessage, 256),
		done:        make(chan struct{}),
	}
	if readLimit > 0 {
		c.conn.SetReadLimit(readLimit)
	}
	c.conn.SetPingHandler(c.pingHandler)
	c.conn.SetPongHandler(c.pongHandler)

	go c.loopSend()
	if pingInterval > 0 {
		go c.doPing(pingInterval)
	}
	return c
}

type websocketClientConn struct {
	readTimeout time.Duration
	conn        *websocket.Conn
	writer      chan *websocketMessage
	done        chan struct{}
	err         atomic.Pointer[error]
}

func (c *websocketClientConn) ReadMessage() ([]byte, error) {
	buf, err := c.readMessage()
	if err != nil {
		return nil, c.closeBy(fmt.Errorf("read message %w", err))
	}
	return buf, nil
}
```

```
func (c *websocketClientConn) WriteMessage(message []byte) error {
	return c.writeMessage(websocket.BinaryMessage, message)
}

func (c *websocketClientConn) Close() error {
	return c.closeBy(fmt.Errorf("websocket connection closed"))
}

func (c *websocketClientConn) closeBy(err error) error {
	if !c.err.CompareAndSwap(nil, &err) {
		return *c.err.Load()
	}
	close(c.done)
	log.ZWarn(context.Background(), "websocket connection closed", err, "remoteAddr", c.conn.RemoteAddr(),
		"chan length", len(c.writer))
	return err
}

func (c *websocketClientConn) writeMessage(messageType int, data []byte) error {
	if errPtr := c.err.Load(); errPtr != nil {
		return *errPtr
	}
	select {
	case c.writer <- &websocketMessage{MessageType: messageType, Data: data}:
		return nil
	default:
		return c.closeBy(ErrWriteFull)
	}
}

func (c *websocketClientConn) loopSend() {
	defer func() {
		_ = c.conn.Close()
	}()
	var err error
	for {
		select {
		case <-c.done:
			for {
				select {
				case msg := <-c.writer:
					switch msg.MessageType {
					case websocket.TextMessage, websocket.BinaryMessage:
						err = c.conn.WriteMessage(msg.MessageType, msg.Data)
					default:
						err = c.conn.WriteControl(msg.MessageType, msg.Data, time.Time{})
					}
					if err != nil {
						_ = c.closeBy(err)
						return
					}
				default:
					return
				}
			}
		case msg := <-c.writer:
			switch msg.MessageType {
			case websocket.TextMessage, websocket.BinaryMessage:
				err = c.conn.WriteMessage(msg.MessageType, msg.Data)
			default:
				err = c.conn.WriteControl(msg.MessageType, msg.Data, time.Time{})
			}
			if err != nil {
				_ = c.closeBy(err)
				return
			}
```

```
    }
  }
}

func (c *websocketClientConn) setReadDeadline() error {
  deadline := time.Now().Add(c.readTimeout)
  return c.conn.SetReadDeadline(deadline)
}

func (c *websocketClientConn) readMessage() ([]byte, error) {
  for {
    if err := c.setReadDeadline(); err != nil {
      return nil, err
    }
    messageType, buf, err := c.conn.ReadMessage()
    if err != nil {
      return nil, err
    }
    switch messageType {
    case websocket.BinaryMessage:
      return buf, nil
    case websocket.TextMessage:
      if err := c.onReadTextMessage(buf); err != nil {
        return nil, err
      }
    case websocket.PingMessage:
      if err := c.pingHandler(string(buf)); err != nil {
        return nil, err
      }
    case websocket.PongMessage:
      if err := c.pongHandler(string(buf)); err != nil {
        return nil, err
      }
    case websocket.CloseMessage:
      if len(buf) == 0 {
        return nil, errors.New("websocket connection closed by peer")
      }
      return nil, fmt.Errorf("websocket connection closed by peer, data %s", string(buf))
    default:
      return nil, fmt.Errorf("unknown websocket message type %d", messageType)
    }
  }
}

func (c *websocketClientConn) onReadTextMessage(buf []byte) error {
  var msg struct {
    Type string          `json:"type"`
    Body json.RawMessage `json:"body"`
  }
  if err := json.Unmarshal(buf, &msg); err != nil {
    return err
  }
  switch msg.Type {
  case TextPong:
    return nil
  case TextPing:
    msg.Type = TextPong
    msgData, err := json.Marshal(msg)
    if err != nil {
      return err
    }
    return c.writeMessage(websocket.TextMessage, msgData)
  default:
    return fmt.Errorf("not support text message type %s", msg.Type)
  }
}
```

```
func (c *websocketClientConn) pingHandler(appData string) error {
	log.ZDebug(context.Background(), "ping handler recv ping", "remoteAddr", c.conn.RemoteAddr(), "appData", appData)
	if err := c.setReadDeadline(); err != nil {
		return err
	}
	err := c.conn.WriteControl(websocket.PongMessage, []byte(appData), time.Now().Add(time.Second*1))
	if err != nil {
		log.ZWarn(context.Background(), "ping handler write pong error", err, "remoteAddr", c.conn.RemoteAddr(), "appData
	}
	log.ZDebug(context.Background(), "ping handler write pong success", "remoteAddr", c.conn.RemoteAddr(), "appData", a
	return nil
}

func (c *websocketClientConn) pongHandler(string) error {
	return nil
}

func (c *websocketClientConn) doPing(d time.Duration) {
	ticker := time.NewTicker(d)
	defer ticker.Stop()
	for {
		select {
		case <-c.done:
			return
		case <-ticker.C:
			if err := c.writeMessage(websocket.PingMessage, nil); err != nil {
				_ = c.closeBy(fmt.Errorf("send ping %w", err))
				return
			}
		}
	}
}
```

internal/msggateway/compressor.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
    "bytes"
    "compress/gzip"
    "io"
    "sync"

    "github.com/openimsdk/tools/errs"
)

var (
    gzipWriterPool = sync.Pool{New: func() any { return gzip.NewWriter(nil) }}
    gzipReaderPool = sync.Pool{New: func() any { return new(gzip.Reader) }}
)

type Compressor interface {
    Compress(rawData []byte) ([]byte, error)
    CompressWithPool(rawData []byte) ([]byte, error)
    DeCompress(compressedData []byte) ([]byte, error)
    DecompressWithPool(compressedData []byte) ([]byte, error)
}

type GzipCompressor struct {
    compressProtocol string
}

func NewGzipCompressor() *GzipCompressor {
    return &GzipCompressor{compressProtocol: "gzip"}
}

func (g *GzipCompressor) Compress(rawData []byte) ([]byte, error) {
    gzipBuffer := bytes.Buffer{}
    gz := gzip.NewWriter(&gzipBuffer)

    if _, err := gz.Write(rawData); err != nil {
        return nil, errs.WrapMsg(err, "GzipCompressor.Compress: writing to gzip writer failed")
    }

    if err := gz.Close(); err != nil {
        return nil, errs.WrapMsg(err, "GzipCompressor.Compress: closing gzip writer failed")
    }

    return gzipBuffer.Bytes(), nil
}

func (g *GzipCompressor) CompressWithPool(rawData []byte) ([]byte, error) {
    gz := gzipWriterPool.Get().(*gzip.Writer)
    defer gzipWriterPool.Put(gz)
```

```
	gzipBuffer := bytes.Buffer{}
	gz.Reset(&gzipBuffer)

	if _, err := gz.Write(rawData); err != nil {
		return nil, errs.WrapMsg(err, "GzipCompressor.CompressWithPool: error writing data")
	}
	if err := gz.Close(); err != nil {
		return nil, errs.WrapMsg(err, "GzipCompressor.CompressWithPool: error closing gzip writer")
	}
	return gzipBuffer.Bytes(), nil
}

func (g *GzipCompressor) DeCompress(compressedData []byte) ([]byte, error) {
	buff := bytes.NewBuffer(compressedData)
	reader, err := gzip.NewReader(buff)
	if err != nil {
		return nil, errs.WrapMsg(err, "GzipCompressor.DeCompress: NewReader creation failed")
	}
	decompressedData, err := io.ReadAll(reader)
	if err != nil {
		return nil, errs.WrapMsg(err, "GzipCompressor.DeCompress: reading from gzip reader failed")
	}
	if err = reader.Close(); err != nil {
		// Even if closing the reader fails, we've successfully read the data,
		// so we return the decompressed data and an error indicating the close failure.
		return decompressedData, errs.WrapMsg(err, "GzipCompressor.DeCompress: closing gzip reader failed")
	}
	return decompressedData, nil
}

func (g *GzipCompressor) DecompressWithPool(compressedData []byte) ([]byte, error) {
	reader := gzipReaderPool.Get().(*gzip.Reader)
	defer gzipReaderPool.Put(reader)

	err := reader.Reset(bytes.NewReader(compressedData))
	if err != nil {
		return nil, errs.WrapMsg(err, "GzipCompressor.DecompressWithPool: resetting gzip reader failed")
	}

	decompressedData, err := io.ReadAll(reader)
	if err != nil {
		return nil, errs.WrapMsg(err, "GzipCompressor.DecompressWithPool: reading from pooled gzip reader failed")
	}
	if err = reader.Close(); err != nil {
		// Similar to DeCompress, return the data and error for close failure.
		return decompressedData, errs.WrapMsg(err, "GzipCompressor.DecompressWithPool: closing pooled gzip reader failed"
	}
	return decompressedData, nil
}
```

internal/msggateway/compressor_test.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
	"crypto/rand"
	"github.com/stretchr/testify/assert"
	"sync"
	"testing"
	"unsafe"
)

func mockRandom() []byte {
	bs := make([]byte, 50)
	rand.Read(bs)
	return bs
}

func TestCompressDecompress(t *testing.T) {

	compressor := NewGzipCompressor()

	for i := 0; i < 2000; i++ {
		src := mockRandom()

		// compress
		dest, err := compressor.CompressWithPool(src)
		if err != nil {
			t.Log(err)
		}
		assert.Equal(t, nil, err)

		// decompress
		res, err := compressor.DecompressWithPool(dest)
		if err != nil {
			t.Log(err)
		}
		assert.Equal(t, nil, err)

		// check
		assert.EqualValues(t, src, res)
	}
}

func TestCompressDecompressWithConcurrency(t *testing.T) {
	wg := sync.WaitGroup{}
	compressor := NewGzipCompressor()

	for i := 0; i < 200; i++ {
		wg.Add(1)
		go func() {
			defer wg.Done()
```

```
        src := mockRandom()

        // compress
        dest, err := compressor.CompressWithPool(src)
        if err != nil {
            t.Log(err)
        }
        assert.Equal(t, nil, err)

        // decompress
        res, err := compressor.DecompressWithPool(dest)
        if err != nil {
            t.Log(err)
        }
        assert.Equal(t, nil, err)

        // check
        assert.EqualValues(t, src, res)

    }()
  }
  wg.Wait()
}

func BenchmarkCompress(b *testing.B) {
  src := mockRandom()
  compressor := NewGzipCompressor()

  for i := 0; i < b.N; i++ {
    _, err := compressor.Compress(src)
    assert.Equal(b, nil, err)
  }
}

func BenchmarkCompressWithSyncPool(b *testing.B) {
  src := mockRandom()

  compressor := NewGzipCompressor()
  for i := 0; i < b.N; i++ {
    _, err := compressor.CompressWithPool(src)
    assert.Equal(b, nil, err)
  }
}

func BenchmarkDecompress(b *testing.B) {
  src := mockRandom()

  compressor := NewGzipCompressor()
  comdata, err := compressor.Compress(src)

  assert.Equal(b, nil, err)

  for i := 0; i < b.N; i++ {
    _, err := compressor.DeCompress(comdata)
    assert.Equal(b, nil, err)
  }
}

func BenchmarkDecompressWithSyncPool(b *testing.B) {
  src := mockRandom()

  compressor := NewGzipCompressor()
  comdata, err := compressor.Compress(src)
  assert.Equal(b, nil, err)

  for i := 0; i < b.N; i++ {
```

```
    _, err := compressor.DecompressWithPool(comdata)
    assert.Equal(b, nil, err)
  }
}

func TestName(t *testing.T) {
  t.Log(unsafe.Sizeof(Client{}))

}
```

internal/msggateway/constant.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import "time"

const (
    WsUserID                = "sendID"
    CommonUserID            = "userID"
    PlatformID              = "platformID"
    ConnID                  = "connID"
    Token                   = "token"
    OperationID             = "operationID"
    Compression             = "compression"
    GzipCompressionProtocol = "gzip"
    BackgroundStatus        = "isBackground"
    SendResponse            = "isMsgResp"
    SDKType                 = "sdkType"
    SDKVersion              = "sdkVersion"
)

const (
    GoSDK = "go"
    JsSDK = "js"
)

const (
    WebSocket = iota + 1
)

const (
    // Websocket Protocol.
    WSGetNewestSeq        = 1001
    WSPullMsgBySeqList    = 1002
    WSSendMsg             = 1003
    WSSendSignalMsg       = 1004
    WSPullMsg             = 1005
    WSGetConvMaxReadSeq   = 1006
    WsPullConvLastMessage = 1007
    WSPushMsg             = 2001
    WSKickOnlineMsg       = 2002
    WsLogoutMsg           = 2003
    WsSetBackgroundStatus = 2004
    WsSubUserOnlineStatus = 2005
    WSDataError           = 3001
)

const (
    // Time allowed to write a message to the peer.
    writeWait = 10 * time.Second

    // Time allowed to read the next pong message from the peer.
```

```
	pongWait = 30 * time.Second

	// Send pings to peer with this period. Must be less than pongWait.
	pingPeriod = (pongWait * 9) / 10

	// Maximum message size allowed from peer.
	maxMessageSize = 51200
)
```

internal/msggateway/context.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
	"encoding/base64"
	"encoding/json"
	"net/http"
	"net/url"
	"strconv"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"

	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/tools/utils/encrypt"
	"github.com/openimsdk/tools/utils/timeutil"
)

type UserConnContextInfo struct {
	Token        string `json:"token"`
	UserID       string `json:"userID"`
	PlatformID   int    `json:"platformID"`
	OperationID  string `json:"operationID"`
	Compression  string `json:"compression"`
	SDKType      string `json:"sdkType"`
	SendResponse bool   `json:"sendResponse"`
	Background   bool   `json:"background"`
	SDKVersion   string `json:"sdkVersion"`
}

type UserConnContext struct {
	RespWriter http.ResponseWriter
	Req        *http.Request
	Path       string
	Method     string
	RemoteAddr string
	ConnID     string
	info       *UserConnContextInfo
}

func (c *UserConnContext) Deadline() (deadline time.Time, ok bool) {
	return
}

func (c *UserConnContext) Done() <-chan struct{} {
	return nil
}

func (c *UserConnContext) Err() error {
	return nil
}
```

```
func (c *UserConnContext) Value(key any) any {
	switch key {
	case constant.OpUserID:
		return c.GetUserID()
	case constant.OperationID:
		return c.GetOperationID()
	case constant.ConnID:
		return c.GetConnID()
	case constant.OpUserPlatform:
		return c.GetPlatformID()
	case constant.RemoteAddr:
		return c.RemoteAddr
	case SDKVersion:
		return c.info.SDKVersion
	default:
		return ""
	}
}

func newContext(respWriter http.ResponseWriter, req *http.Request) *UserConnContext {
	remoteAddr := req.RemoteAddr
	if forwarded := req.Header.Get("X-Forwarded-For"); forwarded != "" {
		remoteAddr += "_" + forwarded
	}
	return &UserConnContext{
		RespWriter: respWriter,
		Req:        req,
		Path:       req.URL.Path,
		Method:     req.Method,
		RemoteAddr: remoteAddr,
		ConnID:     encrypt.Md5(req.RemoteAddr + "_" + strconv.Itoa(int(timeutil.GetCurrentTimestampByMill()))),
	}
}

func newTempContext() *UserConnContext {
	return &UserConnContext{
		Req:  &http.Request{URL: &url.URL{}},
		info: &UserConnContextInfo{},
	}
}

func (c *UserConnContext) ParseEssentialArgs() error {
	query := c.Req.URL.Query()
	if data := query.Get("v"); data != "" {
		return c.parseByJson(data)
	} else {
		return c.parseByQuery(query, c.Req.Header)
	}
}

func (c *UserConnContext) parseByQuery(query url.Values, header http.Header) error {
	info := UserConnContextInfo{
		Token:       query.Get(Token),
		UserID:      query.Get(WsUserID),
		OperationID: query.Get(OperationID),
		Compression: query.Get(Compression),
		SDKType:     query.Get(SDKType),
		SDKVersion:  query.Get(SDKVersion),
	}
	platformID, err := strconv.Atoi(query.Get(PlatformID))
	if err != nil {
		return servererrs.ErrConnArgsErr.WrapMsg("platformID is not int")
	}
	info.PlatformID = platformID
	if val := query.Get(SendResponse); val != "" {
```

```
		ok, err := strconv.ParseBool(val)
		if err != nil {
			return servererrs.ErrConnArgsErr.WrapMsg("isMsgResp is not bool")
		}
		info.SendResponse = ok
	}
	if info.Compression == "" {
		info.Compression = header.Get(Compression)
	}
	background, err := strconv.ParseBool(query.Get(BackgroundStatus))
	if err != nil {
		return err
	}
	info.Background = background
	return c.checkInfo(&info)
}

func (c *UserConnContext) parseByJson(data string) error {
	reqInfo, err := base64.RawURLEncoding.DecodeString(data)
	if err != nil {
		return servererrs.ErrConnArgsErr.WrapMsg("data is not base64")
	}
	var info UserConnContextInfo
	if err := json.Unmarshal(reqInfo, &info); err != nil {
		return servererrs.ErrConnArgsErr.WrapMsg("data is not json", "info", err.Error())
	}
	return c.checkInfo(&info)
}

func (c *UserConnContext) checkInfo(info *UserConnContextInfo) error {
	if info.OperationID == "" {
		return servererrs.ErrConnArgsErr.WrapMsg("operationID is empty")
	}
	if info.Token == "" {
		return servererrs.ErrConnArgsErr.WrapMsg("token is empty")
	}
	if info.UserID == "" {
		return servererrs.ErrConnArgsErr.WrapMsg("sendID is empty")
	}
	if _, ok := constant.PlatformID2Name[info.PlatformID]; !ok {
		return servererrs.ErrConnArgsErr.WrapMsg("platformID is invalid")
	}
	switch info.SDKType {
	case "":
		info.SDKType = GoSDK
	case GoSDK, JsSDK:
	default:
		return servererrs.ErrConnArgsErr.WrapMsg("sdkType is invalid")
	}
	c.info = info
	return nil
}

func (c *UserConnContext) GetRemoteAddr() string {
	return c.RemoteAddr
}

func (c *UserConnContext) SetHeader(key, value string) {
	c.RespWriter.Header().Set(key, value)
}

func (c *UserConnContext) ErrReturn(error string, code int) {
	http.Error(c.RespWriter, error, code)
}

func (c *UserConnContext) GetConnID() string {
```

```
	return c.ConnID
}

func (c *UserConnContext) GetUserID() string {
	if c == nil || c.info == nil {
		return ""
	}
	return c.info.UserID
}

func (c *UserConnContext) GetPlatformID() int {
	if c == nil || c.info == nil {
		return 0
	}
	return c.info.PlatformID
}

func (c *UserConnContext) GetOperationID() string {
	if c == nil || c.info == nil {
		return ""
	}
	return c.info.OperationID
}

func (c *UserConnContext) SetOperationID(operationID string) {
	if c.info == nil {
		c.info = &UserConnContextInfo{}
	}
	c.info.OperationID = operationID
}

func (c *UserConnContext) GetToken() string {
	if c == nil || c.info == nil {
		return ""
	}
	return c.info.Token
}

func (c *UserConnContext) GetCompression() bool {
	return c != nil && c.info != nil && c.info.Compression == GzipCompressionProtocol
}

func (c *UserConnContext) GetSDKType() string {
	if c == nil || c.info == nil {
		return GoSDK
	}
	switch c.info.SDKType {
	case "", GoSDK:
		return GoSDK
	case JsSDK:
		return JsSDK
	default:
		return ""
	}
}

func (c *UserConnContext) GetSDKVersion() string {
	if c == nil || c.info == nil {
		return ""
	}
	return c.info.SDKVersion
}

func (c *UserConnContext) ShouldSendResp() bool {
	return c != nil && c.info != nil && c.info.SendResponse
}
```

```
func (c *UserConnContext) SetToken(token string) {
    if c.info == nil {
        c.info = &UserConnContextInfo{}
    }
    c.info.Token = token
}

func (c *UserConnContext) GetBackground() bool {
    return c != nil && c.info != nil && c.info.Background
}
```

internal/msggateway/encoder.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
	"bytes"
	"encoding/gob"
	"encoding/json"

	"github.com/openimsdk/tools/errs"
)

type Encoder interface {
	Encode(data any) ([]byte, error)
	Decode(encodeData []byte, decodeData any) error
}

type GobEncoder struct{}

func NewGobEncoder() Encoder {
	return GobEncoder{}
}

func (g GobEncoder) Encode(data any) ([]byte, error) {
	var buff bytes.Buffer
	enc := gob.NewEncoder(&buff)
	if err := enc.Encode(data); err != nil {
		return nil, errs.WrapMsg(err, "GobEncoder.Encode failed", "action", "encode")
	}
	return buff.Bytes(), nil
}

func (g GobEncoder) Decode(encodeData []byte, decodeData any) error {
	buff := bytes.NewBuffer(encodeData)
	dec := gob.NewDecoder(buff)
	if err := dec.Decode(decodeData); err != nil {
		return errs.WrapMsg(err, "GobEncoder.Decode failed", "action", "decode")
	}
	return nil
}

type JsonEncoder struct{}

func NewJsonEncoder() Encoder {
	return JsonEncoder{}
}

func (g JsonEncoder) Encode(data any) ([]byte, error) {
	b, err := json.Marshal(data)
	if err != nil {
		return nil, errs.New("JsonEncoder.Encode failed", "action", "encode")
	}
```

```
    return b, nil
}

func (g JsonEncoder) Decode(encodeData []byte, decodeData any) error {
    err := json.Unmarshal(encodeData, decodeData)
    if err != nil {
        return errs.New("JsonEncoder.Decode failed", "action", "decode")
    }
    return nil
}
```

internal/msggateway/http_error.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
	"github.com/openimsdk/tools/apiresp"
	"github.com/openimsdk/tools/log"
)

func httpError(ctx *UserConnContext, err error) {
	log.ZWarn(ctx, "ws connection error", err)
	apiresp.HttpError(ctx.RespWriter, err)
}
```

internal/msggateway/hub_server.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
    "context"
    "sync/atomic"

    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"

    "github.com/openimsdk/open-im-server/v3/pkg/authverify"
    "github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
    "github.com/openimsdk/protocol/constant"
    "github.com/openimsdk/protocol/msggateway"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/log"
    "github.com/openimsdk/tools/mcontext"
    "github.com/openimsdk/tools/mq/memamq"
    "github.com/openimsdk/tools/utils/datautil"
    "google.golang.org/grpc"
)

func (s *Server) InitServer(ctx context.Context, config *Config, disCov discovery.Conn, server grpc.ServiceRegistrar
    userConn, err := disCov.GetConn(ctx, config.Discovery.RpcService.User)
    if err != nil {
        return err
    }
    s.userClient = rpcli.NewUserClient(userConn)
    if err := s.LongConnServer.SetDiscoveryRegistry(ctx, disCov, config); err != nil {
        return err
    }
    msggateway.RegisterMsgGatewayServer(server, s)
    if s.ready != nil {
        return s.ready(s)
    }
    return nil
}

//func (s *Server) Start(ctx context.Context, index int, conf *Config) error {
//    return startrpc.Start(ctx, &conf.Discovery, &conf.MsgGateway.Prometheus, conf.MsgGateway.ListenIP,
//        conf.MsgGateway.RPC.RegisterIP,
//        conf.MsgGateway.RPC.AutoSetPorts, conf.MsgGateway.RPC.Ports, index,
//        conf.Discovery.RpcService.MessageGateway,
//        nil,
//        conf,
//        []string{
//            conf.Share.GetConfigFileName(),
//            conf.Discovery.GetConfigFileName(),
//            conf.MsgGateway.GetConfigFileName(),
//            conf.WebhooksConfig.GetConfigFileName(),
```

```
//■■■conf.RedisConfig.GetConfigFileName(),
//■■},
//■■[]string{
//■■■conf.Discovery.RpcService.MessageGateway,
//■■},
//■■s.InitServer,
//■)
//}

type Server struct {
■msggateway.UnimplementedMsgGatewayServer

■LongConnServer LongConnServer
■config         *Config
■pushTerminal   map[int]struct{}
■ready          func(srv *Server) error
■queue          *memamq.MemoryQueue
■userClient     *rpcli.UserClient
}

func (s *Server) SetLongConnServer(LongConnServer LongConnServer) {
■s.LongConnServer = LongConnServer
}

func NewServer(longConnServer LongConnServer, conf *Config, ready func(srv *Server) error) *Server {
■s := &Server{
■■LongConnServer: longConnServer,
■■pushTerminal:   make(map[int]struct{}),
■■config:         conf,
■■ready:          ready,
■■queue:          memamq.NewMemoryQueue(512, 1024*16),
■}
■s.pushTerminal[constant.IOSPlatformID] = struct{}{}
■s.pushTerminal[constant.AndroidPlatformID] = struct{}{}
■return s
}

func (s *Server) GetUsersOnlineStatus(ctx context.Context, req *msggateway.GetUsersOnlineStatusReq) (*msggateway.Get
■if !authverify.IsAdmin(ctx) {
■■return nil, errs.ErrNoPermission.WrapMsg("only app manager")
■}
■var resp msggateway.GetUsersOnlineStatusResp
■for _, userID := range req.UserIDs {
■■clients, ok := s.LongConnServer.GetUserAllCons(userID)
■■if !ok {
■■■continue
■■}

■■uresp := new(msggateway.GetUsersOnlineStatusResp_SuccessResult)
■■uresp.UserID = userID
■■for _, client := range clients {
■■■if client == nil {
■■■■continue
■■■}

■■■ps := new(msggateway.GetUsersOnlineStatusResp_SuccessDetail)
■■■ps.PlatformID = int32(client.PlatformID)
■■■ps.ConnID = client.ctx.GetConnID()
■■■ps.Token = client.token
■■■ps.IsBackground = client.IsBackground
■■■uresp.Status = constant.Online
■■■uresp.DetailPlatformStatus = append(uresp.DetailPlatformStatus, ps)
■■}
■■if uresp.Status == constant.Online {
■■■resp.SuccessResult = append(resp.SuccessResult, uresp)
■■}
```

```
	}
	return &resp, nil
}

func (s *Server) pushToUser(ctx context.Context, userID string, msgData *sdkws.MsgData) *msggateway.SingleMsgToUserR
	clients, ok := s.LongConnServer.GetUserAllCons(userID)
	if !ok {
		log.ZDebug(ctx, "push user not online", "userID", userID)
		return &msggateway.SingleMsgToUserResults{
			UserID: userID,
		}
	}
	log.ZDebug(ctx, "push user online", "clients", clients, "userID", userID)
	result := &msggateway.SingleMsgToUserResults{
		UserID: userID,
		Resp:   make([]*msggateway.SingleMsgToUserPlatform, 0, len(clients)),
	}
	for _, client := range clients {
		if client == nil {
			continue
		}
		userPlatform := &msggateway.SingleMsgToUserPlatform{
			RecvPlatFormID: int32(client.PlatformID),
		}
		if client.IsBackground && client.PlatformID == constant.IOSPlatformID {
			userPlatform.ResultCode = int64(servererrs.ErrIOSBackgroundPushErr.Code())
			result.Resp = append(result.Resp, userPlatform)
			continue
		}
		if err := client.PushMessage(ctx, msgData); err != nil {
			log.ZWarn(ctx, "online push msg failed", err, "userID", userID, "platformID", client.PlatformID)
			userPlatform.ResultCode = int64(servererrs.ErrPushMsgErr.Code())
		} else if _, ok := s.pushTerminal[client.PlatformID]; ok {
			result.OnlinePush = true
		}
		result.Resp = append(result.Resp, userPlatform)
	}
	return result
}

func (s *Server) SuperGroupOnlineBatchPushOneMsg(ctx context.Context, req *msggateway.OnlineBatchPushOneMsgReq) (*ms
	if len(req.PushToUserIDs) == 0 {
		return &msggateway.OnlineBatchPushOneMsgResp{}, nil
	}
	ch := make(chan *msggateway.SingleMsgToUserResults, len(req.PushToUserIDs))
	var count atomic.Int64
	count.Add(int64(len(req.PushToUserIDs)))
	for i := range req.PushToUserIDs {
		userID := req.PushToUserIDs[i]
		err := s.queue.PushCtx(ctx, func() {
			ch <- s.pushToUser(ctx, userID, req.MsgData)
			if count.Add(-1) == 0 {
				close(ch)
			}
		})
		if err != nil {
			if count.Add(-1) == 0 {
				close(ch)
			}
			log.ZError(ctx, "pushToUser MemoryQueue failed", err, "userID", userID)
			ch <- &msggateway.SingleMsgToUserResults{
				UserID: userID,
			}
		}
	}
	resp := &msggateway.OnlineBatchPushOneMsgResp{
```

```
    SinglePushResult: make([]*msggateway.SingleMsgToUserResults, 0, len(req.PushToUserIDs)),
  }
  for {
    select {
    case <-ctx.Done():
      log.ZError(ctx, "SuperGroupOnlineBatchPushOneMsg ctx done", context.Cause(ctx))
      userIDSet := datautil.SliceSet(req.PushToUserIDs)
      for _, results := range resp.SinglePushResult {
        delete(userIDSet, results.UserID)
      }
      for userID := range userIDSet {
        resp.SinglePushResult = append(resp.SinglePushResult, &msggateway.SingleMsgToUserResults{
          UserID: userID,
        })
      }
      return resp, nil
    case res, ok := <-ch:
      if !ok {
        return resp, nil
      }
      resp.SinglePushResult = append(resp.SinglePushResult, res)
    }
  }
}

func (s *Server) KickUserOffline(ctx context.Context, req *msggateway.KickUserOfflineReq) (*msggateway.KickUserOffli
  for _, v := range req.KickUserIDList {
    clients, _, ok := s.LongConnServer.GetUserPlatformCons(v, int(req.PlatformID))
    if !ok {
      log.ZDebug(ctx, "conn not exist", "userID", v, "platformID", req.PlatformID)
      continue
    }

    for _, client := range clients {
      log.ZDebug(ctx, "kick user offline", "userID", v, "platformID", req.PlatformID, "client", client)
      if err := client.longConnServer.KickUserConn(client); err != nil {
        log.ZWarn(ctx, "kick user offline failed", err, "userID", v, "platformID", req.PlatformID)
      }
    }
    continue
  }

  return &msggateway.KickUserOfflineResp{}, nil
}

func (s *Server) MultiTerminalLoginCheck(ctx context.Context, req *msggateway.MultiTerminalLoginCheckReq) (*msggatew
  if oldClients, userOK, clientOK := s.LongConnServer.GetUserPlatformCons(req.UserID, int(req.PlatformID)); userOK {
    tempUserCtx := newTempContext()
    tempUserCtx.SetToken(req.Token)
    tempUserCtx.SetOperationID(mcontext.GetOperationID(ctx))
    client := &Client{}
    client.ctx = tempUserCtx
    client.token = req.Token
    client.UserID = req.UserID
    client.PlatformID = int(req.PlatformID)
    i := &kickHandler{
      clientOK:   clientOK,
      oldClients: oldClients,
      newClient:  client,
    }
    s.LongConnServer.SetKickHandlerInfo(i)
  }
  return &msggateway.MultiTerminalLoginCheckResp{}, nil
}
```

internal/msggateway/init.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
    "context"
    "time"

    "github.com/openimsdk/open-im-server/v3/pkg/common/config"
    "github.com/openimsdk/open-im-server/v3/pkg/dbbuild"
    "github.com/openimsdk/open-im-server/v3/pkg/rpccache"
    "github.com/openimsdk/tools/discovery"
    "github.com/openimsdk/tools/utils/datautil"
    "github.com/openimsdk/tools/utils/runtimeenv"
    "google.golang.org/grpc"

    "github.com/openimsdk/tools/log"
)

type Config struct {
    MsgGateway     config.MsgGateway
    Share          config.Share
    RedisConfig    config.Redis
    WebhooksConfig config.Webhooks
    Discovery      config.Discovery
    Index          config.Index
}

// Start run ws server.
func Start(ctx context.Context, conf *Config, client discovery.SvcDiscoveryRegistry, server grpc.ServiceRegistrar) e
    log.CInfo(ctx, "MSG-GATEWAY server is initializing", "runtimeEnv", runtimeenv.RuntimeEnvironment(),
        "rpcPorts", conf.MsgGateway.RPC.Ports,
        "wsPort", conf.MsgGateway.LongConnSvr.Ports, "prometheusPorts", conf.MsgGateway.Prometheus.Ports)
    wsPort, err := datautil.GetElemByIndex(conf.MsgGateway.LongConnSvr.Ports, int(conf.Index))
    if err != nil {
        return err
    }

    dbb := dbbuild.NewBuilder(nil, &conf.RedisConfig)
    rdb, err := dbb.Redis(ctx)
    if err != nil {
        return err
    }

    longServer := NewWsServer(
        conf,
        WithPort(wsPort),
        WithMaxConnNum(int64(conf.MsgGateway.LongConnSvr.WebsocketMaxConnNum)),
        WithHandshakeTimeout(time.Duration(conf.MsgGateway.LongConnSvr.WebsocketTimeout)*time.Second),
        WithMessageMaxMsgLength(conf.MsgGateway.LongConnSvr.WebsocketMaxMsgLen),
    )
```

```
    hubServer := NewServer(longServer, conf, func(srv *Server) error {
        var err error
        longServer.online, err = rpccache.NewOnlineCache(srv.userClient, nil, rdb, false, longServer.subscriberUserOnline
        return err
    })

    if err := hubServer.InitServer(ctx, conf, client, server); err != nil {
        return err
    }

    go longServer.ChangeOnlineStatus(4)

    return hubServer.LongConnServer.Run(ctx)
}

//
//// Start run ws server.
//func Start(ctx context.Context, index int, conf *Config) error {
//    log.CInfo(ctx, "MSG-GATEWAY server is initializing", "runtimeEnv", runtimeenv.RuntimeEnvironment(),
//        "rpcPorts", conf.MsgGateway.RPC.Ports,
//        "wsPort", conf.MsgGateway.LongConnSvr.Ports, "prometheusPorts", conf.MsgGateway.Prometheus.Ports)
//    wsPort, err := datautil.GetElemByIndex(conf.MsgGateway.LongConnSvr.Ports, index)
//    if err != nil {
//        return err
//    }
//
//    rdb, err := redisutil.NewRedisClient(ctx, conf.RedisConfig.Build())
//    if err != nil {
//        return err
//    }
//    longServer := NewWsServer(
//        conf,
//        WithPort(wsPort),
//        WithMaxConnNum(int64(conf.MsgGateway.LongConnSvr.WebsocketMaxConnNum)),
//        WithHandshakeTimeout(time.Duration(conf.MsgGateway.LongConnSvr.WebsocketTimeout)*time.Second),
//        WithMessageMaxMsgLength(conf.MsgGateway.LongConnSvr.WebsocketMaxMsgLen),
//    )
//
//    hubServer := NewServer(longServer, conf, func(srv *Server) error {
//        var err error
//        longServer.online, err = rpccache.NewOnlineCache(srv.userClient, nil, rdb, false, longServer.subscriberUserOnli
//        return err
//    })
//
//    go longServer.ChangeOnlineStatus(4)
//
//    netDone := make(chan error)
//    go func() {
//        err = hubServer.Start(ctx, index, conf)
//        netDone <- err
//    }()
//    return hubServer.LongConnServer.Run(netDone)
//}
```

internal/msggateway/message_handler.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import (
    "context"
    "encoding/json"
    "github.com/openimsdk/open-im-server/v3/pkg/rpcli"
    "sync"

    "github.com/go-playground/validator/v10"
    "google.golang.org/protobuf/proto"

    "github.com/openimsdk/protocol/msg"
    "github.com/openimsdk/protocol/push"
    "github.com/openimsdk/protocol/sdkws"
    "github.com/openimsdk/tools/errs"
    "github.com/openimsdk/tools/utils/jsonutil"
)

const (
    TextPing = "ping"
    TextPong = "pong"
)

type TextMessage struct {
    Type string          `json:"type"`
    Body json.RawMessage `json:"body"`
}

type Req struct {
    ReqIdentifier int32  `json:"reqIdentifier" validate:"required"`
    Token         string `json:"token"`
    SendID        string `json:"sendID"        validate:"required"`
    OperationID   string `json:"operationID"   validate:"required"`
    MsgIncr       string `json:"msgIncr"       validate:"required"`
    Data          []byte `json:"data"`
}

func (r *Req) String() string {
    var tReq Req
    tReq.ReqIdentifier = r.ReqIdentifier
    tReq.Token = r.Token
    tReq.SendID = r.SendID
    tReq.OperationID = r.OperationID
    tReq.MsgIncr = r.MsgIncr
    return jsonutil.StructToJsonString(tReq)
}

var reqPool = sync.Pool{
    New: func() any {
        return new(Req)
```

```
	},
}

func getReq() *Req {
	req := reqPool.Get().(*Req)
	req.Data = nil
	req.MsgIncr = ""
	req.OperationID = ""
	req.ReqIdentifier = 0
	req.SendID = ""
	req.Token = ""
	return req
}

func freeReq(req *Req) {
	reqPool.Put(req)
}

type Resp struct {
	ReqIdentifier int32  `json:"reqIdentifier"`
	MsgIncr       string `json:"msgIncr"`
	OperationID   string `json:"operationID"`
	ErrCode       int    `json:"errCode"`
	ErrMsg        string `json:"errMsg"`
	Data          []byte `json:"data"`
}

func (r *Resp) String() string {
	var tResp Resp
	tResp.ReqIdentifier = r.ReqIdentifier
	tResp.MsgIncr = r.MsgIncr
	tResp.OperationID = r.OperationID
	tResp.ErrCode = r.ErrCode
	tResp.ErrMsg = r.ErrMsg
	return jsonutil.StructToJsonString(tResp)
}

type MessageHandler interface {
	GetSeq(ctx context.Context, data *Req) ([]byte, error)
	SendMessage(ctx context.Context, data *Req) ([]byte, error)
	SendSignalMessage(ctx context.Context, data *Req) ([]byte, error)
	PullMessageBySeqList(ctx context.Context, data *Req) ([]byte, error)
	GetConversationsHasReadAndMaxSeq(ctx context.Context, data *Req) ([]byte, error)
	GetSeqMessage(ctx context.Context, data *Req) ([]byte, error)
	UserLogout(ctx context.Context, data *Req) ([]byte, error)
	SetUserDeviceBackground(ctx context.Context, data *Req) ([]byte, bool, error)
	GetLastMessage(ctx context.Context, data *Req) ([]byte, error)
}

var _ MessageHandler = (*GrpcHandler)(nil)

type GrpcHandler struct {
	validate   *validator.Validate
	msgClient  *rpcli.MsgClient
	pushClient *rpcli.PushMsgServiceClient
}

func NewGrpcHandler(validate *validator.Validate, msgClient *rpcli.MsgClient, pushClient *rpcli.PushMsgServiceClient
	return &GrpcHandler{
		validate:   validate,
		msgClient:  msgClient,
		pushClient: pushClient,
	}
}

func (g *GrpcHandler) GetSeq(ctx context.Context, data *Req) ([]byte, error) {
```

```
	req := sdkws.GetMaxSeqReq{}
	if err := proto.Unmarshal(data.Data, &req); err != nil {
		return nil, errs.WrapMsg(err, "GetSeq: error unmarshaling request", "action", "unmarshal", "dataType", "GetMaxSeq
	}
	if err := g.validate.Struct(&req); err != nil {
		return nil, errs.WrapMsg(err, "GetSeq: validation failed", "action", "validate", "dataType", "GetMaxSeqReq")
	}
	resp, err := g.msgClient.MsgClient.GetMaxSeq(ctx, &req)
	if err != nil {
		return nil, err
	}
	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "GetSeq: error marshaling response", "action", "marshal", "dataType", "GetMaxSeqRes
	}
	return c, nil
}

// SendMessage handles the sending of messages through gRPC. It unmarshals the request data,
// validates the message, and then sends it using the message RPC client.
func (g *GrpcHandler) SendMessage(ctx context.Context, data *Req) ([]byte, error) {
	var msgData sdkws.MsgData
	if err := proto.Unmarshal(data.Data, &msgData); err != nil {
		return nil, errs.WrapMsg(err, "SendMessage: error unmarshaling message data", "action", "unmarshal", "dataType",
	}

	if err := g.validate.Struct(&msgData); err != nil {
		return nil, errs.WrapMsg(err, "SendMessage: message data validation failed", "action", "validate", "dataType", "M
	}

	req := msg.SendMsgReq{MsgData: &msgData}
	resp, err := g.msgClient.MsgClient.SendMsg(ctx, &req)
	if err != nil {
		return nil, err
	}

	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "SendMessage: error marshaling response", "action", "marshal", "dataType", "SendMsg
	}

	return c, nil
}

func (g *GrpcHandler) SendSignalMessage(ctx context.Context, data *Req) ([]byte, error) {
	resp, err := g.msgClient.MsgClient.SendMsg(ctx, nil)
	if err != nil {
		return nil, err
	}
	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "error marshaling response", "action", "marshal", "dataType", "SendMsgResp")
	}
	return c, nil
}

func (g *GrpcHandler) PullMessageBySeqList(ctx context.Context, data *Req) ([]byte, error) {
	req := sdkws.PullMessageBySeqsReq{}
	if err := proto.Unmarshal(data.Data, &req); err != nil {
		return nil, errs.WrapMsg(err, "err proto unmarshal", "action", "unmarshal", "dataType", "PullMessageBySeqsReq")
	}
	if err := g.validate.Struct(data); err != nil {
		return nil, errs.WrapMsg(err, "validation failed", "action", "validate", "dataType", "PullMessageBySeqsReq")
	}
	resp, err := g.msgClient.MsgClient.PullMessageBySeqs(ctx, &req)
	if err != nil {
```

```
		return nil, err
	}
	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "error marshaling response", "action", "marshal", "dataType", "PullMessageBySeqsResp
	}
	return c, nil
}

func (g *GrpcHandler) GetConversationsHasReadAndMaxSeq(ctx context.Context, data *Req) ([]byte, error) {
	req := msg.GetConversationsHasReadAndMaxSeqReq{}
	if err := proto.Unmarshal(data.Data, &req); err != nil {
		return nil, errs.WrapMsg(err, "err proto unmarshal", "action", "unmarshal", "dataType", "GetConversationsHasReadAn
	}
	if err := g.validate.Struct(data); err != nil {
		return nil, errs.WrapMsg(err, "validation failed", "action", "validate", "dataType", "GetConversationsHasReadAndMa
	}
	resp, err := g.msgClient.MsgClient.GetConversationsHasReadAndMaxSeq(ctx, &req)
	if err != nil {
		return nil, err
	}
	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "error marshaling response", "action", "marshal", "dataType", "GetConversationsHasR
	}
	return c, nil
}

func (g *GrpcHandler) GetSeqMessage(ctx context.Context, data *Req) ([]byte, error) {
	req := msg.GetSeqMessageReq{}
	if err := proto.Unmarshal(data.Data, &req); err != nil {
		return nil, errs.WrapMsg(err, "error unmarshaling request", "action", "unmarshal", "dataType", "GetSeqMessage")
	}
	if err := g.validate.Struct(data); err != nil {
		return nil, errs.WrapMsg(err, "validation failed", "action", "validate", "dataType", "GetSeqMessage")
	}
	resp, err := g.msgClient.MsgClient.GetSeqMessage(ctx, &req)
	if err != nil {
		return nil, err
	}
	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "error marshaling response", "action", "marshal", "dataType", "GetSeqMessage")
	}
	return c, nil
}

func (g *GrpcHandler) UserLogout(ctx context.Context, data *Req) ([]byte, error) {
	req := push.DelUserPushTokenReq{}
	if err := proto.Unmarshal(data.Data, &req); err != nil {
		return nil, errs.WrapMsg(err, "error unmarshaling request", "action", "unmarshal", "dataType", "DelUserPushTokenR
	}
	resp, err := g.pushClient.PushMsgServiceClient.DelUserPushToken(ctx, &req)
	if err != nil {
		return nil, err
	}
	c, err := proto.Marshal(resp)
	if err != nil {
		return nil, errs.WrapMsg(err, "error marshaling response", "action", "marshal", "dataType", "DelUserPushTokenResp
	}
	return c, nil
}

func (g *GrpcHandler) SetUserDeviceBackground(ctx context.Context, data *Req) ([]byte, bool, error) {
	req := sdkws.SetAppBackgroundStatusReq{}
	if err := proto.Unmarshal(data.Data, &req); err != nil {
```

```
		return nil, false, errs.WrapMsg(err, "error unmarshaling request", "action", "unmarshal", "dataType", "SetAppBack
	}
	if err := g.validate.Struct(data); err != nil {
		return nil, false, errs.WrapMsg(err, "validation failed", "action", "validate", "dataType", "SetAppBackgroundStat
	}
	return nil, req.IsBackground, nil
}

func (g *GrpcHandler) GetLastMessage(ctx context.Context, data *Req) ([]byte, error) {
	var req msg.GetLastMessageReq
	if err := proto.Unmarshal(data.Data, &req); err != nil {
		return nil, err
	}
	resp, err := g.msgClient.GetLastMessage(ctx, &req)
	if err != nil {
		return nil, err
	}
	return proto.Marshal(resp)
}
```

internal/msggateway/online.go

```
package msggateway

import (
	"context"
	"crypto/md5"
	"encoding/binary"
	"fmt"
	"math/rand"
	"os"
	"strconv"
	"sync/atomic"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/cachekey"
	pbuser "github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/mcontext"
	"github.com/openimsdk/tools/utils/datautil"
)

func (ws *WsServer) ChangeOnlineStatus(concurrent int) {
	if concurrent < 1 {
		concurrent = 1
	}
	const renewalTime = cachekey.OnlineExpire / 3
	//const renewalTime = time.Second * 10
	renewalTicker := time.NewTicker(renewalTime)

	requestChs := make([]chan *pbuser.SetUserOnlineStatusReq, concurrent)
	changeStatus := make([][]UserState, concurrent)

	for i := 0; i < concurrent; i++ {
		requestChs[i] = make(chan *pbuser.SetUserOnlineStatusReq, 64)
		changeStatus[i] = make([]UserState, 0, 100)
	}

	mergeTicker := time.NewTicker(time.Second)

	local2pb := func(u UserState) *pbuser.UserOnlineStatus {
		return &pbuser.UserOnlineStatus{
			UserID:  u.UserID,
			Online:  u.Online,
			Offline: u.Offline,
		}
	}

	rNum := rand.Uint64()
	pushUserState := func(us ...UserState) {
		for _, u := range us {
			sum := md5.Sum([]byte(u.UserID))
			i := (binary.BigEndian.Uint64(sum[:]) + rNum) % uint64(concurrent)
			changeStatus[i] = append(changeStatus[i], u)
			status := changeStatus[i]
			if len(status) == cap(status) {
				req := &pbuser.SetUserOnlineStatusReq{
					Status: datautil.Slice(status, local2pb),
				}
				changeStatus[i] = status[:0]
				select {
				case requestChs[i] <- req:
				default:
					log.ZError(context.Background(), "user online processing is too slow", nil)
				}
			}
		}
```

```
    }
  }

  pushAllUserState := func() {
    for i, status := range changeStatus {
      if len(status) == 0 {
        continue
      }
      req := &pbuser.SetUserOnlineStatusReq{
        Status: datautil.Slice(status, local2pb),
      }
      changeStatus[i] = status[:0]
      select {
      case requestChs[i] <- req:
      default:
        log.ZError(context.Background(), "user online processing is too slow", nil)
      }
    }
  }

  var count atomic.Int64
  operationIDPrefix := fmt.Sprintf("p_%d_", os.Getpid())
  doRequest := func(req *pbuser.SetUserOnlineStatusReq) {
    opIdCtx := mcontext.SetOperationID(context.Background(), operationIDPrefix+strconv.FormatInt(count.Add(1), 10))
    ctx, cancel := context.WithTimeout(opIdCtx, time.Second*5)
    defer cancel()
    if err := ws.userClient.SetUserOnlineStatus(ctx, req); err != nil {
      log.ZError(ctx, "update user online status", err)
    }
    for _, ss := range req.Status {
      for _, online := range ss.Online {
        client, _, _ := ws.clients.Get(ss.UserID, int(online))
        back := false
        if len(client) > 0 {
          back = client[0].IsBackground
        }
        ws.webhookAfterUserOnline(ctx, &ws.msgGatewayConfig.WebhooksConfig.AfterUserOnline, ss.UserID, int(online), bac
      }
      for _, offline := range ss.Offline {
        ws.webhookAfterUserOffline(ctx, &ws.msgGatewayConfig.WebhooksConfig.AfterUserOffline, ss.UserID, int(offline),
      }
    }
  }

  for i := 0; i < concurrent; i++ {
    go func(ch <-chan *pbuser.SetUserOnlineStatusReq) {
      for req := range ch {
        doRequest(req)
      }
    }(requestChs[i])
  }

  for {
    select {
    case <-mergeTicker.C:
      pushAllUserState()
    case now := <-renewalTicker.C:
      deadline := now.Add(-cachekey.OnlineExpire / 3)
      users := ws.clients.GetAllUserStatus(deadline, now)
      log.ZDebug(context.Background(), "renewal ticker", "deadline", deadline, "nowtime", now, "num", len(users), "use
      pushUserState(users...)
    case state := <-ws.clients.UserState():
      log.ZDebug(context.Background(), "OnlineCache user online change", "userID", state.UserID, "online", state.Onlin
      pushUserState(state)
    }
  }
```

```
}
```

internal/msggateway/options.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package msggateway

import "time"

type (
	Option  func(opt *configs)
	configs struct {
		// Long connection listening port
		port int
		// Maximum number of connections allowed for long connection
		maxConnNum int64
		// Connection handshake timeout
		handshakeTimeout time.Duration
		// Maximum length allowed for messages
		messageMaxMsgLength int
		// Websocket write buffer, default: 4096, 4kb.
		writeBufferSize int
	}
)

func WithPort(port int) Option {
	return func(opt *configs) {
		opt.port = port
	}
}

func WithMaxConnNum(num int64) Option {
	return func(opt *configs) {
		opt.maxConnNum = num
	}
}

func WithHandshakeTimeout(t time.Duration) Option {
	return func(opt *configs) {
		opt.handshakeTimeout = t
	}
}

func WithMessageMaxMsgLength(length int) Option {
	return func(opt *configs) {
		opt.messageMaxMsgLength = length
	}
}

func WithWriteBufferSize(size int) Option {
	return func(opt *configs) {
		opt.writeBufferSize = size
	}
}
```

internal/msggateway/subscription.go

```
package msggateway

import (
	"context"
	"github.com/openimsdk/protocol/sdkws"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/utils/datautil"
	"google.golang.org/protobuf/proto"
	"sync"
)

func (ws *WsServer) subscriberUserOnlineStatusChanges(ctx context.Context, userID string, platformIDs []int32) {
	if ws.clients.RecvSubChange(userID, platformIDs) {
		log.ZDebug(ctx, "gateway receive subscription message and go back online", "userID", userID, "platformIDs", platf
	} else {
		log.ZDebug(ctx, "gateway ignore user online status changes", "userID", userID, "platformIDs", platformIDs)
	}
	ws.pushUserIDOnlineStatus(ctx, userID, platformIDs)
}

func (ws *WsServer) SubUserOnlineStatus(ctx context.Context, client *Client, data *Req) ([]byte, error) {
	var sub sdkws.SubUserOnlineStatus
	if err := proto.Unmarshal(data.Data, &sub); err != nil {
		return nil, err
	}
	ws.subscription.Sub(client, sub.SubscribeUserID, sub.UnsubscribeUserID)
	var resp sdkws.SubUserOnlineStatusTips
	if len(sub.SubscribeUserID) > 0 {
		resp.Subscribers = make([]*sdkws.SubUserOnlineStatusElem, 0, len(sub.SubscribeUserID))
		for _, userID := range sub.SubscribeUserID {
			platformIDs, err := ws.online.GetUserOnlinePlatform(ctx, userID)
			if err != nil {
				return nil, err
			}
			resp.Subscribers = append(resp.Subscribers, &sdkws.SubUserOnlineStatusElem{
				UserID:            userID,
				OnlinePlatformIDs: platformIDs,
			})
		}
	}
	return proto.Marshal(&resp)
}

func newSubscription() *Subscription {
	return &Subscription{
		userIDs: make(map[string]*subClient),
	}
}

type subClient struct {
	clients map[string]*Client
}

type Subscription struct {
	lock    sync.RWMutex
	userIDs map[string]*subClient // subscribe to the user's client connection
}

func (s *Subscription) DelClient(client *Client) {
	client.subLock.Lock()
	userIDs := datautil.Keys(client.subUserIDs)
	for _, userID := range userIDs {
		delete(client.subUserIDs, userID)
	}
```

```
	client.subLock.Unlock()
	if len(userIDs) == 0 {
		return
	}
	addr := client.ctx.GetRemoteAddr()
	s.lock.Lock()
	defer s.lock.Unlock()
	for _, userID := range userIDs {
		sub, ok := s.userIDs[userID]
		if !ok {
			continue
		}
		delete(sub.clients, addr)
		if len(sub.clients) == 0 {
			delete(s.userIDs, userID)
		}
	}
}

func (s *Subscription) GetClient(userID string) []*Client {
	s.lock.RLock()
	defer s.lock.RUnlock()
	cs, ok := s.userIDs[userID]
	if !ok {
		return nil
	}
	clients := make([]*Client, 0, len(cs.clients))
	for _, client := range cs.clients {
		clients = append(clients, client)
	}
	return clients
}

func (s *Subscription) Sub(client *Client, addUserIDs, delUserIDs []string) {
	if len(addUserIDs)+len(delUserIDs) == 0 {
		return
	}
	var (
		del = make(map[string]struct{})
		add = make(map[string]struct{})
	)
	client.subLock.Lock()
	for _, userID := range delUserIDs {
		if _, ok := client.subUserIDs[userID]; !ok {
			continue
		}
		del[userID] = struct{}{}
		delete(client.subUserIDs, userID)
	}
	for _, userID := range addUserIDs {
		delete(del, userID)
		if _, ok := client.subUserIDs[userID]; ok {
			continue
		}
		client.subUserIDs[userID] = struct{}{}
		add[userID] = struct{}{}
	}
	client.subLock.Unlock()
	if len(del)+len(add) == 0 {
		return
	}
	addr := client.ctx.GetRemoteAddr()
	s.lock.Lock()
	defer s.lock.Unlock()
	for userID := range del {
		sub, ok := s.userIDs[userID]
```

```
		if !ok {
			continue
		}
		delete(sub.clients, addr)
		if len(sub.clients) == 0 {
			delete(s.userIDs, userID)
		}
	}
	for userID := range add {
		sub, ok := s.userIDs[userID]
		if !ok {
			sub = &subClient{clients: make(map[string]*Client)}
			s.userIDs[userID] = sub
		}
		sub.clients[addr] = client
	}
}

func (ws *WsServer) pushUserIDOnlineStatus(ctx context.Context, userID string, platformIDs []int32) {
	clients := ws.subscription.GetClient(userID)
	if len(clients) == 0 {
		return
	}
	onlineStatus, err := proto.Marshal(&sdkws.SubUserOnlineStatusTips{
		Subscribers: []*sdkws.SubUserOnlineStatusElem{{UserID: userID, OnlinePlatformIDs: platformIDs}},
	})
	if err != nil {
		log.ZError(ctx, "pushUserIDOnlineStatus json.Marshal", err)
		return
	}
	for _, client := range clients {
		if err := client.PushUserOnlineStatus(onlineStatus); err != nil {
			log.ZError(ctx, "UserSubscribeOnlineStatusNotification push failed", err, "userID", client.UserID, "platformID",
		}
	}
}
```

internal/msggateway/user_map.go

```
package msggateway

import (
	"github.com/openimsdk/tools/utils/datautil"
	"sync"
	"time"
)

type UserMap interface {
	GetAll(userID string) ([]*Client, bool)
	Get(userID string, platformID int) ([]*Client, bool, bool)
	Set(userID string, v *Client)
	DeleteClients(userID string, clients []*Client) (isDeleteUser bool)
	UserState() <-chan UserState
	GetAllUserStatus(deadline time.Time, nowtime time.Time) []UserState
	RecvSubChange(userID string, platformIDs []int32) bool
}

type UserState struct {
	UserID  string
	Online  []int32
	Offline []int32
}

type UserPlatform struct {
	Time    time.Time
	Clients []*Client
}

func (u *UserPlatform) PlatformIDs() []int32 {
	if len(u.Clients) == 0 {
		return nil
	}
	platformIDs := make([]int32, 0, len(u.Clients))
	for _, client := range u.Clients {
		platformIDs = append(platformIDs, int32(client.PlatformID))
	}
	return platformIDs
}

func (u *UserPlatform) PlatformIDSet() map[int32]struct{} {
	if len(u.Clients) == 0 {
		return nil
	}
	platformIDs := make(map[int32]struct{})
	for _, client := range u.Clients {
		platformIDs[int32(client.PlatformID)] = struct{}{}
	}
	return platformIDs
}

func newUserMap() UserMap {
	return &userMap{
		data: make(map[string]*UserPlatform),
		ch:   make(chan UserState, 10000),
	}
}

type userMap struct {
	lock sync.RWMutex
	data map[string]*UserPlatform
	ch   chan UserState
}
```

```
func (u *userMap) RecvSubChange(userID string, platformIDs []int32) bool {
	u.lock.RLock()
	defer u.lock.RUnlock()
	result, ok := u.data[userID]
	if !ok {
		return false
	}
	localPlatformIDs := result.PlatformIDSet()
	for _, platformID := range platformIDs {
		delete(localPlatformIDs, platformID)
	}
	if len(localPlatformIDs) == 0 {
		return false
	}
	u.push(userID, result, nil)
	return true
}

func (u *userMap) push(userID string, userPlatform *UserPlatform, offline []int32) bool {
	select {
	case u.ch <- UserState{UserID: userID, Online: userPlatform.PlatformIDs(), Offline: offline}:
		userPlatform.Time = time.Now()
		return true
	default:
		return false
	}
}

func (u *userMap) GetAll(userID string) ([]*Client, bool) {
	u.lock.RLock()
	defer u.lock.RUnlock()
	result, ok := u.data[userID]
	if !ok {
		return nil, false
	}
	return result.Clients, true
}

func (u *userMap) Get(userID string, platformID int) ([]*Client, bool, bool) {
	u.lock.RLock()
	defer u.lock.RUnlock()
	result, ok := u.data[userID]
	if !ok {
		return nil, false, false
	}
	var clients []*Client
	for _, client := range result.Clients {
		if client.PlatformID == platformID {
			clients = append(clients, client)
		}
	}
	return clients, true, len(clients) > 0
}

func (u *userMap) Set(userID string, client *Client) {
	u.lock.Lock()
	defer u.lock.Unlock()
	result, ok := u.data[userID]
	if ok {
		result.Clients = append(result.Clients, client)
	} else {
		result = &UserPlatform{
			Clients: []*Client{client},
		}
		u.data[userID] = result
	}
```

```
	u.push(client.UserID, result, nil)
}

func (u *userMap) DeleteClients(userID string, clients []*Client) (isDeleteUser bool) {
	if len(clients) == 0 {
		return false
	}
	u.lock.Lock()
	defer u.lock.Unlock()
	result, ok := u.data[userID]
	if !ok {
		return false
	}
	offline := make([]int32, 0, len(clients))
	deleteAddr := datautil.SliceSetAny(clients, func(client *Client) string {
		return client.ctx.GetRemoteAddr()
	})
	tmp := result.Clients
	result.Clients = result.Clients[:0]
	for _, client := range tmp {
		if _, delCli := deleteAddr[client.ctx.GetRemoteAddr()]; delCli {
			offline = append(offline, int32(client.PlatformID))
		} else {
			result.Clients = append(result.Clients, client)
		}
	}
	defer u.push(userID, result, offline)
	if len(result.Clients) > 0 {
		return false
	}
	delete(u.data, userID)
	return true
}

func (u *userMap) GetAllUserStatus(deadline time.Time, nowtime time.Time) (result []UserState) {
	u.lock.RLock()
	defer u.lock.RUnlock()
	result = make([]UserState, 0, len(u.data))
	for userID, userPlatform := range u.data {
		if deadline.Before(userPlatform.Time) {
			continue
		}
		userPlatform.Time = nowtime
		online := make([]int32, 0, len(userPlatform.Clients))
		for _, client := range userPlatform.Clients {
			online = append(online, int32(client.PlatformID))
		}
		result = append(result, UserState{UserID: userID, Online: online})
	}
	return result
}

func (u *userMap) UserState() <-chan UserState {
	return u.ch
}
```

internal/msggateway/ws_server.go

```
package msggateway

import (
	"context"
	"encoding/json"
	"fmt"
	"net/http"
	"sync"
	"sync/atomic"
	"time"

	"github.com/gorilla/websocket"
	"github.com/openimsdk/open-im-server/v3/pkg/rpcli"
	"github.com/openimsdk/tools/apiresp"

	"github.com/openimsdk/open-im-server/v3/pkg/common/webhook"
	"github.com/openimsdk/open-im-server/v3/pkg/rpccache"
	pbAuth "github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/tools/mcontext"

	"github.com/go-playground/validator/v10"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/open-im-server/v3/pkg/common/servererrs"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/msggateway"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/log"
	"golang.org/x/sync/errgroup"
)

var wsSuccessResponse, _ = json.Marshal(&apiresp.ApiResponse{})

type LongConnServer interface {
	Run(ctx context.Context) error
	wsHandler(w http.ResponseWriter, r *http.Request)
	GetUserAllCons(userID string) ([]*Client, bool)
	GetUserPlatformCons(userID string, platform int) ([]*Client, bool, bool)
	Validate(s any) error
	SetDiscoveryRegistry(ctx context.Context, client discovery.Conn, config *Config) error
	KickUserConn(client *Client) error
	UnRegister(c *Client)
	SetKickHandlerInfo(i *kickHandler)
	SubUserOnlineStatus(ctx context.Context, client *Client, data *Req) ([]byte, error)
	Compressor
	MessageHandler
}

type WsServer struct {
	websocket         *websocket.Upgrader
	msgGatewayConfig  *Config
	port              int
	wsMaxConnNum      int64
	registerChan      chan *Client
	unregisterChan    chan *Client
	kickHandlerChan   chan *kickHandler
	clients           UserMap
	online            rpccache.OnlineCache
	subscription      *Subscription
	clientPool        sync.Pool
	onlineUserNum     atomic.Int64
	onlineUserConnNum atomic.Int64
	handshakeTimeout  time.Duration
	writeBufferSize   int
	validate          *validator.Validate
```

```
	disCov            discovery.Conn
	Compressor
	//Encoder
	MessageHandler
	webhookClient *webhook.Client
	userClient    *rpcli.UserClient
	authClient    *rpcli.AuthClient

	ready atomic.Bool
}

type kickHandler struct {
	clientOK   bool
	oldClients []*Client
	newClient  *Client
}

func (ws *WsServer) SetDiscoveryRegistry(ctx context.Context, disCov discovery.Conn, config *Config) error {
	userConn, err := disCov.GetConn(ctx, config.Discovery.RpcService.User)
	if err != nil {
		return err
	}
	pushConn, err := disCov.GetConn(ctx, config.Discovery.RpcService.Push)
	if err != nil {
		return err
	}
	authConn, err := disCov.GetConn(ctx, config.Discovery.RpcService.Auth)
	if err != nil {
		return err
	}
	msgConn, err := disCov.GetConn(ctx, config.Discovery.RpcService.Msg)
	if err != nil {
		return err
	}
	ws.userClient = rpcli.NewUserClient(userConn)
	ws.authClient = rpcli.NewAuthClient(authConn)
	ws.MessageHandler = NewGrpcHandler(ws.validate, rpcli.NewMsgClient(msgConn), rpcli.NewPushMsgServiceClient(pushConn
	ws.disCov = disCov

	ws.ready.Store(true)
	return nil
}

//func (ws *WsServer) SetUserOnlineStatus(ctx context.Context, client *Client, status int32) {
//	err := ws.userClient.SetUserStatus(ctx, client.UserID, status, client.PlatformID)
//	if err != nil {
//		log.ZWarn(ctx, "SetUserStatus err", err)
//	}
//	switch status {
//	case constant.Online:
//		ws.webhookAfterUserOnline(ctx, &ws.msgGatewayConfig.WebhooksConfig.AfterUserOnline, client.UserID, client.Platf
//	case constant.Offline:
//		ws.webhookAfterUserOffline(ctx, &ws.msgGatewayConfig.WebhooksConfig.AfterUserOffline, client.UserID, client.Pla
//	}
//}

func (ws *WsServer) UnRegister(c *Client) {
	ws.unregisterChan <- c
}

func (ws *WsServer) Validate(_ any) error {
	return nil
}

func (ws *WsServer) GetUserAllCons(userID string) ([]*Client, bool) {
	return ws.clients.GetAll(userID)
```

```
}

func (ws *WsServer) GetUserPlatformCons(userID string, platform int) ([]*Client, bool, bool) {
	return ws.clients.Get(userID, platform)
}

func NewWsServer(msgGatewayConfig *Config, opts ...Option) *WsServer {
	var config configs
	for _, o := range opts {
		o(&config)
	}
	//userRpcClient := rpcclient.NewUserRpcClient(client, config.Discovery.RpcService.User, config.Share.IMAdminUser)
	upgrader := &websocket.Upgrader{
		HandshakeTimeout: config.handshakeTimeout,
		CheckOrigin:      func(r *http.Request) bool { return true },
	}
	v := validator.New()
	return &WsServer{
		websocket:        upgrader,
		msgGatewayConfig: msgGatewayConfig,
		port:             config.port,
		wsMaxConnNum:     config.maxConnNum,
		writeBufferSize:  config.writeBufferSize,
		handshakeTimeout: config.handshakeTimeout,
		clientPool: sync.Pool{
			New: func() any {
				return new(Client)
			},
		},
		registerChan:    make(chan *Client, 1000),
		unregisterChan:  make(chan *Client, 1000),
		kickHandlerChan: make(chan *kickHandler, 1000),
		validate:        v,
		clients:         newUserMap(),
		subscription:    newSubscription(),
		Compressor:      NewGzipCompressor(),
		webhookClient:   webhook.NewWebhookClient(msgGatewayConfig.WebhooksConfig.URL),
	}
}

func (ws *WsServer) Run(ctx context.Context) error {
	var client *Client

	ctx, cancel := context.WithCancelCause(ctx)
	go func() {
		for {
			select {
			case <-ctx.Done():
				return
			case client = <-ws.registerChan:
				ws.registerClient(client)
			case client = <-ws.unregisterChan:
				ws.unregisterClient(client)
			case onlineInfo := <-ws.kickHandlerChan:
				ws.multiTerminalLoginChecker(onlineInfo.clientOK, onlineInfo.oldClients, onlineInfo.newClient)
			}
		}
	}()

	done := make(chan struct{})
	go func() {
		wsServer := http.Server{Addr: fmt.Sprintf(":%d", ws.port), Handler: nil}
		http.HandleFunc("/", ws.wsHandler)
		go func() {
			defer close(done)
			<-ctx.Done()
```

```
        _ = wsServer.Shutdown(context.Background())
    }()
    err := wsServer.ListenAndServe()
    if err == nil {
        err = fmt.Errorf("http server closed")
    }
    cancel(fmt.Errorf("msg gateway %w", err))
}()

<-ctx.Done()

timeout := time.NewTimer(time.Second * 15)
defer timeout.Stop()
select {
case <-timeout.C:
    log.ZWarn(ctx, "msg gateway graceful stop timeout", nil)
case <-done:
    log.ZDebug(ctx, "msg gateway graceful stop done")
}
return context.Cause(ctx)
}

const concurrentRequest = 3

func (ws *WsServer) sendUserOnlineInfoToOtherNode(ctx context.Context, client *Client) error {
conns, err := ws.disCov.GetConns(ctx, ws.msgGatewayConfig.Discovery.RpcService.MessageGateway)
if err != nil {
    return err
}
if len(conns) == 0 || (len(conns) == 1 && ws.disCov.IsSelfNode(conns[0])) {
    return nil
}

wg := errgroup.Group{}
wg.SetLimit(concurrentRequest)

// Online push user online message to other node
for _, v := range conns {
    v := v
    log.ZDebug(ctx, "sendUserOnlineInfoToOtherNode conn")
    if ws.disCov.IsSelfNode(v) {
        log.ZDebug(ctx, "Filter out this node")
        continue
    }

    wg.Go(func() error {
        msgClient := msggateway.NewMsgGatewayClient(v)
        _, err := msgClient.MultiTerminalLoginCheck(ctx, &msggateway.MultiTerminalLoginCheckReq{
            UserID:     client.UserID,
            PlatformID: int32(client.PlatformID), Token: client.token,
        })
        if err != nil {
            log.ZWarn(ctx, "MultiTerminalLoginCheck err", err)
        }
        return nil
    })
}

_ = wg.Wait()
return nil
}

func (ws *WsServer) SetKickHandlerInfo(i *kickHandler) {
ws.kickHandlerChan <- i
}
```

```
func (ws *WsServer) registerClient(client *Client) {
    var (
        userOK     bool
        clientOK   bool
        oldClients []*Client
    )
    oldClients, userOK, clientOK = ws.clients.Get(client.UserID, client.PlatformID)

    log.ZInfo(client.ctx, "registerClient", "userID", client.UserID, "platformID", client.PlatformID)

    if !userOK {
        ws.clients.Set(client.UserID, client)
        log.ZDebug(client.ctx, "user not exist", "userID", client.UserID, "platformID", client.PlatformID)
        prommetrics.OnlineUserGauge.Add(1)
        ws.onlineUserNum.Add(1)
        ws.onlineUserConnNum.Add(1)
    } else {
        ws.multiTerminalLoginChecker(clientOK, oldClients, client)
        log.ZDebug(client.ctx, "user exist", "userID", client.UserID, "platformID", client.PlatformID)
        if clientOK {
            ws.clients.Set(client.UserID, client)
            // There is already a connection to the platform
            log.ZDebug(client.ctx, "repeat login", "userID", client.UserID, "platformID",
                client.PlatformID, "old remote addr", getRemoteAdders(oldClients))
            ws.onlineUserConnNum.Add(1)
        } else {
            ws.clients.Set(client.UserID, client)
            ws.onlineUserConnNum.Add(1)
        }
    }

    wg := sync.WaitGroup{}
    log.ZDebug(client.ctx, "ws.msgGatewayConfig.Discovery.Enable", "discoveryEnable", ws.msgGatewayConfig.Discovery.Ena

    if ws.msgGatewayConfig.Discovery.Enable != "k8s" {
        wg.Add(1)
        go func() {
            defer wg.Done()
            _ = ws.sendUserOnlineInfoToOtherNode(client.ctx, client)
        }()
    }

    //wg.Add(1)
    //go func() {
    //    defer wg.Done()
    //    ws.SetUserOnlineStatus(client.ctx, client, constant.Online)
    //}()

    wg.Wait()

    log.ZDebug(client.ctx, "user online", "online user Num", ws.onlineUserNum.Load(), "online user conn Num", ws.online
}

func getRemoteAdders(client []*Client) string {
    var ret string
    for i, c := range client {
        if i == 0 {
            ret = c.ctx.GetRemoteAddr()
        } else {
            ret += "@" + c.ctx.GetRemoteAddr()
        }
    }
    return ret
}

func (ws *WsServer) KickUserConn(client *Client) error {
```

```
	ws.clients.DeleteClients(client.UserID, []*Client{client})
	return client.KickOnlineMessage()
}

func (ws *WsServer) multiTerminalLoginChecker(clientOK bool, oldClients []*Client, newClient *Client) {
	kickTokenFunc := func(kickClients []*Client) {
		var kickTokens []string
		ws.clients.DeleteClients(newClient.UserID, kickClients)
		for _, c := range kickClients {
			kickTokens = append(kickTokens, c.token)
			err := c.KickOnlineMessage()
			if err != nil {
				log.ZWarn(c.ctx, "KickOnlineMessage", err)
			}
		}
		ctx := mcontext.WithMustInfoCtx(
			[]string{newClient.ctx.GetOperationID(), newClient.ctx.GetUserID(),
				constant.PlatformIDToName(newClient.PlatformID), newClient.ctx.GetConnID()},
		)
		if err := ws.authClient.KickTokens(ctx, kickTokens); err != nil {
			log.ZWarn(newClient.ctx, "kickTokens err", err)
		}
	}

	// If reconnect: When multiple msgGateway instances are deployed, a client may disconnect from instance A and recon
	// During this process, instance A might still be executing, resulting in two clients with the same token existing
	// This situation needs to be filtered to prevent duplicate clients.
	checkSameTokenFunc := func(oldClients []*Client) []*Client {
		var clientsNeedToKick []*Client

		for _, c := range oldClients {
			if c.token == newClient.token {
				log.ZDebug(newClient.ctx, "token is same, not kick",
					"userID", newClient.UserID,
					"platformID", newClient.PlatformID,
					"token", newClient.token)
				continue
			}

			clientsNeedToKick = append(clientsNeedToKick, c)
		}

		return clientsNeedToKick
	}

	switch ws.msgGatewayConfig.Share.MultiLogin.Policy {
	case constant.DefalutNotKick:
	case constant.PCAndOther:
		if constant.PlatformIDToClass(newClient.PlatformID) == constant.TerminalPC {
			return
		}
		clients, ok := ws.clients.GetAll(newClient.UserID)
		clientOK = ok
		oldClients = make([]*Client, 0, len(clients))
		for _, c := range clients {
			if constant.PlatformIDToClass(c.PlatformID) == constant.TerminalPC {
				continue
			}
			oldClients = append(oldClients, c)
		}

		fallthrough
	case constant.AllLoginButSameTermKick:
		if !clientOK {
			return
		}
```

```
    oldClients = checkSameTokenFunc(oldClients)

    ws.clients.DeleteClients(newClient.UserID, oldClients)
    for _, c := range oldClients {
        err := c.KickOnlineMessage()
        if err != nil {
            log.ZWarn(c.ctx, "KickOnlineMessage", err)
        }
    }

    ctx := mcontext.WithMustInfoCtx(
        []string{newClient.ctx.GetOperationID(), newClient.ctx.GetUserID(),
            constant.PlatformIDToName(newClient.PlatformID), newClient.ctx.GetConnID()},
    )
    req := &pbAuth.InvalidateTokenReq{
        PreservedToken: newClient.token,
        UserID:         newClient.UserID,
        PlatformID:     int32(newClient.PlatformID),
    }
    if err := ws.authClient.InvalidateToken(ctx, req); err != nil {
        log.ZWarn(newClient.ctx, "InvalidateToken err", err, "userID", newClient.UserID,
            "platformID", newClient.PlatformID)
    }
  case constant.AllLoginButSameClassKick:
    clients, ok := ws.clients.GetAll(newClient.UserID)
    if !ok {
        return
    }

    var kickClients []*Client
    for _, client := range clients {
        if constant.PlatformIDToClass(client.PlatformID) == constant.PlatformIDToClass(newClient.PlatformID) {
            {
                kickClients = append(kickClients, client)
            }
        }
    }
    kickClients = checkSameTokenFunc(kickClients)

    kickTokenFunc(kickClients)
  }
}

func (ws *WsServer) unregisterClient(client *Client) {
  defer ws.clientPool.Put(client)
  isDeleteUser := ws.clients.DeleteClients(client.UserID, []*Client{client})
  if isDeleteUser {
    ws.onlineUserNum.Add(-1)
    prommetrics.OnlineUserGauge.Dec()
  }
  ws.onlineUserConnNum.Add(-1)
  ws.subscription.DelClient(client)
  //ws.SetUserOnlineStatus(client.ctx, client, constant.Offline)
  log.ZDebug(client.ctx, "user offline", "close reason", client.closedErr, "online user Num",
    ws.onlineUserNum.Load(), "online user conn Num",
    ws.onlineUserConnNum.Load(),
  )
}

// validateRespWithRequest checks if the response matches the expected userID and platformID.
func (ws *WsServer) validateRespWithRequest(ctx *UserConnContext, resp *pbAuth.ParseTokenResp) error {
  userID := ctx.GetUserID()
  platformID := int32(ctx.GetPlatformID())
  if resp.UserID != userID {
    return servererrs.ErrTokenInvalid.WrapMsg(fmt.Sprintf("token uid %s != userID %s", resp.UserID, userID))
```

```
	}
	if resp.PlatformID != platformID {
		return servererrs.ErrTokenInvalid.WrapMsg(fmt.Sprintf("token platform %d != platformID %d", resp.PlatformID, plat
	}
	return nil
}

func (ws *WsServer) handlerError(ctx *UserConnContext, w http.ResponseWriter, r *http.Request, err error) {
	if !ctx.ShouldSendResp() {
		httpError(ctx, err)
		return
	}
	// the browser cannot get the response of upgrade failure
	data, err := json.Marshal(apiresp.ParseError(err))
	if err != nil {
		log.ZError(ctx, "json marshal failed", err)
		return
	}
	conn, upgradeErr := ws.websocket.Upgrade(w, r, nil)
	if upgradeErr != nil {
		log.ZWarn(ctx, "websocket upgrade failed", upgradeErr, "respErr", err, "resp", string(data))
		return
	}
	defer conn.Close()
	if err := conn.WriteMessage(websocket.TextMessage, data); err != nil {
		log.ZWarn(ctx, "WriteMessage failed", err, "respErr", err, "resp", string(data))
		return
	}
}

func (ws *WsServer) wsHandler(w http.ResponseWriter, r *http.Request) {
	// Create a new connection context
	connContext := newContext(w, r)

	// Check if the current number of online user connections exceeds the maximum limit
	if ws.onlineUserConnNum.Load() >= ws.wsMaxConnNum {
		// If it exceeds the maximum connection number, return an error via HTTP and stop processing
		ws.handlerError(connContext, w, r, servererrs.ErrConnOverMaxNumLimit.WrapMsg("over max conn num limit"))
		return
	}

	// Parse essential arguments (e.g., user ID, Token)
	err := connContext.ParseEssentialArgs()
	if err != nil {
		// If there's an error during parsing, return an error via HTTP and stop processing
		ws.handlerError(connContext, w, r, err)
		return
	}

	// Call the authentication client to parse the Token obtained from the context
	resp, err := ws.authClient.ParseToken(connContext, connContext.GetToken())
	if err != nil {
		ws.handlerError(connContext, w, r, err)
		return
	}

	// Validate the authentication response matches the request (e.g., user ID and platform ID)
	err = ws.validateRespWithRequest(connContext, resp)
	if err != nil {
		// If validation fails, return an error via HTTP and stop processing
		ws.handlerError(connContext, w, r, err)
		return
	}
	conn, err := ws.websocket.Upgrade(w, r, nil)
	if err != nil {
		log.ZWarn(connContext, "websocket upgrade failed", err)
```

```
		return
	}
	if connContext.ShouldSendResp() {
		if err := conn.WriteMessage(websocket.TextMessage, wsSuccessResponse); err != nil {
			log.ZWarn(connContext, "WriteMessage first response", err)
			return
		}
	}

	log.ZDebug(connContext, "new conn", "token", connContext.GetToken())

	var pingInterval time.Duration
	if connContext.GetPlatformID() == constant.WebPlatformID {
		pingInterval = pingPeriod
	}

	client := new(Client)
	client.ResetClient(connContext, NewWebSocketClientConn(conn, maxMessageSize, pongWait, pingInterval), ws)

	// Register the client with the server and start message processing
	ws.registerChan <- client
	go client.readMessage()
}
```

tools

tools/imctl

tools/imctl/main.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import "fmt"

func main() {

	fmt.Println("imctl")
}
```

tools/versionchecker

tools/versionchecker/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"bytes"
	"fmt"
	"os/exec"
	"runtime"

	"github.com/fatih/color"
	"github.com/openimsdk/tools/utils/timeutil"
)

func ExecuteCommand(cmdName string, args ...string) (string, error) {
	cmd := exec.Command(cmdName, args...)
	var out bytes.Buffer
	var stderr bytes.Buffer
	cmd.Stdout = &out
	cmd.Stderr = &stderr

	err := cmd.Run()
	if err != nil {
		return "", fmt.Errorf("error executing %s: %v, stderr: %s", cmdName, err, stderr.String())
	}
	return out.String(), nil
}

func printTime() string {
	formattedTime := timeutil.GetCurrentTimeFormatted()
	return fmt.Sprintf("Current Date & Time: %s", formattedTime)
}

func getGoVersion() string {
	version := runtime.Version()
	goos := runtime.GOOS
	goarch := runtime.GOARCH
	return fmt.Sprintf("Go Version: %s\nOS: %s\nArchitecture: %s", version, goos, goarch)
}

func getDockerVersion() string {
	version, err := ExecuteCommand("docker", "--version")
	if err != nil {
		return "Docker is not installed. Please install it to get the version."
	}
	return version
}

func getKubernetesVersion() string {
```

```
    version, err := ExecuteCommand("kubectl", "version", "--client", "--short")
    if err != nil {
        return "Kubernetes is not installed. Please install it to get the version."
    }
    return version
}

func getGitVersion() string {
    version, err := ExecuteCommand("git", "branch", "--show-current")
    if err != nil {
        return "Git is not installed. Please install it to get the version."
    }
    return version
}

// // NOTE: You'll need to provide appropriate commands for OpenIM versions.
// func getOpenIMServerVersion() string {
// 	// Placeholder
// 	openimVersion := version.GetSingleVersion()
// 	return "OpenIM Server: " + openimVersion + "\n"
// }

// func getOpenIMClientVersion() (string, error) {
// 	openIMClientVersion, err := version.GetClientVersion()
// 	if err != nil {
// 		return "", err
// 	}
// 	return "OpenIM Client: " + openIMClientVersion.ClientVersion + "\n", nil
// }

func main() {
    // red := color.New(color.FgRed).SprintFunc()
    //green := color.New(color.FgGreen).SprintFunc()
    blue := color.New(color.FgBlue).SprintFunc()
    //yellow := color.New(color.FgYellow).SprintFunc()
    fmt.Println(blue("## Go Version"))
    fmt.Println(getGoVersion())
    fmt.Println(blue("## Branch Type"))
    fmt.Println(getGitVersion())
    fmt.Println(blue("## Docker Version"))
    fmt.Println(getDockerVersion())
    fmt.Println(blue("## Kubernetes Version"))
    fmt.Println(getKubernetesVersion())
    // fmt.Println(blue("## OpenIM Versions"))
    // fmt.Println(getOpenIMServerVersion())
    // clientVersion, err := getOpenIMClientVersion()
    // if err != nil {
    // 	fmt.Println(red("Error getting OpenIM Client Version: "), err)
    // } else {
    // 	fmt.Println(clientVersion)
    // }
}
```

tools/seq

tools/seq/main.go

```
package main

import (
	"flag"
	"fmt"
	"os"
	"time"

	"github.com/openimsdk/open-im-server/v3/tools/seq/internal"
)

func main() {
	var (
		config string
		second int
	)
	flag.StringVar(&config, "c", "", "config directory")
	flag.IntVar(&second, "sec", 3600*24, "delayed deletion of the original seq key after conversion")
	flag.Parse()
	if err := internal.Main(config, time.Duration(second)*time.Second); err != nil {
		fmt.Println("seq task", err)
		os.Exit(1)
		return
	}
	fmt.Println("seq task success!")
}
```

tools/seq/internal

tools/seq/internal/seq.go

```
package internal

import (
	"bytes"
	"context"
	"errors"
	"fmt"
	"os"
	"os/signal"
	"path/filepath"
	"strconv"
	"strings"
	"sync"
	"sync/atomic"
	"syscall"
	"time"

	"github.com/mitchellh/mapstructure"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"github.com/openimsdk/tools/utils/runtimeenv"
	"github.com/redis/go-redis/v9"
	"github.com/spf13/viper"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
)

const StructTagName = "yaml"

const (
	MaxSeq                 = "MAX_SEQ:"
	MinSeq                 = "MIN_SEQ:"
	ConversationUserMinSeq = "CON_USER_MIN_SEQ:"
	HasReadSeq             = "HAS_READ_SEQ:"
)

const (
	batchSize             = 100
	dataVersionCollection = "data_version"
	seqKey                = "seq"
	seqVersion            = 38
)

func readConfig[T any](dir string, name string) (*T, error) {
	if runtimeenv.RuntimeEnvironment() == config.KUBERNETES {
		dir = os.Getenv(config.MountConfigFilePath)
	}
	v := viper.New()
	v.SetEnvPrefix(config.EnvPrefixMap[name])
	v.AutomaticEnv()
	v.SetEnvKeyReplacer(strings.NewReplacer(".", "_"))
	v.SetConfigFile(filepath.Join(dir, name))
	if err := v.ReadInConfig(); err != nil {
		return nil, err
	}

	var conf T
	if err := v.Unmarshal(&conf, func(config *mapstructure.DecoderConfig) {
```

```
    config.TagName = StructTagName
  }); err != nil {
    return nil, err
  }

  return &conf, nil
}

func Main(conf string, del time.Duration) error {
  redisConfig, err := readConfig[config.Redis](conf, config.RedisConfigFileName)
  if err != nil {
    return err
  }

  mongodbConfig, err := readConfig[config.Mongo](conf, config.MongodbConfigFileName)
  if err != nil {
    return err
  }
  ctx, cancel := context.WithTimeout(context.Background(), time.Second*10)
  defer cancel()
  rdb, err := redisutil.NewRedisClient(ctx, redisConfig.Build())
  if err != nil {
    return err
  }
  mgocli, err := mongoutil.NewMongoDB(ctx, mongodbConfig.Build())
  if err != nil {
    return err
  }
  versionColl := mgocli.GetDB().Collection(dataVersionCollection)
  converted, err := CheckVersion(versionColl, seqKey, seqVersion)
  if err != nil {
    return err
  }
  if converted {
    fmt.Println("[seq] seq data has been converted")
    return nil
  }
  if _, err := mgo.NewSeqConversationMongo(mgocli.GetDB()); err != nil {
    return err
  }
  cSeq, err := mgo.NewSeqConversationMongo(mgocli.GetDB())
  if err != nil {
    return err
  }
  uSeq, err := mgo.NewSeqUserMongo(mgocli.GetDB())
  if err != nil {
    return err
  }
  uSpitHasReadSeq := func(id string) (conversationID string, userID string, err error) {
    // HasReadSeq + userID + ":" + conversationID
    arr := strings.Split(id, ":")
    if len(arr) != 2 || arr[0] == "" || arr[1] == "" {
      return "", "", fmt.Errorf("invalid has read seq id %s", id)
    }
    userID = arr[0]
    conversationID = arr[1]
    return
  }
  uSpitConversationUserMinSeq := func(id string) (conversationID string, userID string, err error) {
    // ConversationUserMinSeq + conversationID + "u:" + userID
    arr := strings.Split(id, "u:")
    if len(arr) != 2 || arr[0] == "" || arr[1] == "" {
      return "", "", fmt.Errorf("invalid has read seq id %s", id)
    }
    conversationID = arr[0]
    userID = arr[1]
```

```
        return
    }

    ts := []*taskSeq{
        {
            Prefix: MaxSeq,
            GetSeq: cSeq.GetMaxSeq,
            SetSeq: cSeq.SetMaxSeq,
        },
        {
            Prefix: MinSeq,
            GetSeq: cSeq.GetMinSeq,
            SetSeq: cSeq.SetMinSeq,
        },
        {
            Prefix: HasReadSeq,
            GetSeq: func(ctx context.Context, id string) (int64, error) {
                conversationID, userID, err := uSpitHasReadSeq(id)
                if err != nil {
                    return 0, err
                }
                return uSeq.GetUserReadSeq(ctx, conversationID, userID)
            },
            SetSeq: func(ctx context.Context, id string, seq int64) error {
                conversationID, userID, err := uSpitHasReadSeq(id)
                if err != nil {
                    return err
                }
                return uSeq.SetUserReadSeq(ctx, conversationID, userID, seq)
            },
        },
        {
            Prefix: ConversationUserMinSeq,
            GetSeq: func(ctx context.Context, id string) (int64, error) {
                conversationID, userID, err := uSpitConversationUserMinSeq(id)
                if err != nil {
                    return 0, err
                }
                return uSeq.GetUserMinSeq(ctx, conversationID, userID)
            },
            SetSeq: func(ctx context.Context, id string, seq int64) error {
                conversationID, userID, err := uSpitConversationUserMinSeq(id)
                if err != nil {
                    return err
                }
                return uSeq.SetUserMinSeq(ctx, conversationID, userID, seq)
            },
        },
    }

    cancel()
    ctx = context.Background()

    var wg sync.WaitGroup
    wg.Add(len(ts))

    for i := range ts {
        go func(task *taskSeq) {
            defer wg.Done()
            err := seqRedisToMongo(ctx, rdb, task.GetSeq, task.SetSeq, task.Prefix, del, &task.Count)
            task.End = time.Now()
            task.Error = err
        }(ts[i])
    }
    start := time.Now()
    done := make(chan struct{})
```

```
	go func() {
		wg.Wait()
		close(done)
	}()

	sigs := make(chan os.Signal, 1)
	signal.Notify(sigs, syscall.SIGTERM)

	ticker := time.NewTicker(time.Second)
	defer ticker.Stop()
	var buf bytes.Buffer

	printTaskInfo := func(now time.Time) {
		buf.Reset()
		buf.WriteString(now.Format(time.DateTime))
		buf.WriteString(" \n")
		for i := range ts {
			task := ts[i]
			if task.Error == nil {
				if task.End.IsZero() {
					buf.WriteString(fmt.Sprintf("[%s] converting %s* count %d", now.Sub(start), task.Prefix, atomic.LoadInt64(&tas
				} else {
					buf.WriteString(fmt.Sprintf("[%s] success %s* count %d", task.End.Sub(start), task.Prefix, atomic.LoadInt64(&t
				}
			} else {
				buf.WriteString(fmt.Sprintf("[%s] failed %s* count %d error %s", task.End.Sub(start), task.Prefix, atomic.LoadI
			}
			buf.WriteString("\n")
		}
		fmt.Println(buf.String())
	}

	for {
		select {
		case <-ctx.Done():
			return ctx.Err()
		case s := <-sigs:
			return fmt.Errorf("exit by signal %s", s)
		case <-done:
			errs := make([]error, 0, len(ts))
			for i := range ts {
				task := ts[i]
				if task.Error != nil {
					errs = append(errs, fmt.Errorf("seq %s failed %w", task.Prefix, task.Error))
				}
			}
			if len(errs) > 0 {
				return errors.Join(errs...)
			}
			printTaskInfo(time.Now())
			if err := SetVersion(versionColl, seqKey, seqVersion); err != nil {
				return fmt.Errorf("set mongodb seq version %w", err)
			}
			return nil
		case now := <-ticker.C:
			printTaskInfo(now)
		}
	}
}

type taskSeq struct {
	Prefix string
	Count  int64
	Error  error
	End    time.Time
	GetSeq func(ctx context.Context, id string) (int64, error)
```

```
	SetSeq func(ctx context.Context, id string, seq int64) error
}

func seqRedisToMongo(ctx context.Context, rdb redis.UniversalClient, getSeq func(ctx context.Context, id string) (in
	var (
		cursor uint64
		keys   []string
		err    error
	)
	for {
		keys, cursor, err = rdb.Scan(ctx, cursor, prefix+"*", batchSize).Result()
		if err != nil {
			return err
		}
		if len(keys) > 0 {
			for _, key := range keys {
				seqStr, err := rdb.Get(ctx, key).Result()
				if err != nil {
					return fmt.Errorf("redis get %s failed %w", key, err)
				}
				seq, err := strconv.Atoi(seqStr)
				if err != nil {
					return fmt.Errorf("invalid %s seq %s", key, seqStr)
				}
				if seq < 0 {
					return fmt.Errorf("invalid %s seq %s", key, seqStr)
				}
				id := strings.TrimPrefix(key, prefix)
				redisSeq := int64(seq)
				mongoSeq, err := getSeq(ctx, id)
				if err != nil {
					return fmt.Errorf("get mongo seq %s failed %w", key, err)
				}
				if mongoSeq < redisSeq {
					if err := setSeq(ctx, id, redisSeq); err != nil {
						return fmt.Errorf("set mongo seq %s failed %w", key, err)
					}
				}
				if delAfter > 0 {
					if err := rdb.Expire(ctx, key, delAfter).Err(); err != nil {
						return fmt.Errorf("redis expire key %s failed %w", key, err)
					}
				} else {
					if err := rdb.Del(ctx, key).Err(); err != nil {
						return fmt.Errorf("redis del key %s failed %w", key, err)
					}
				}
				atomic.AddInt64(count, 1)
			}
		}
		if cursor == 0 {
			return nil
		}
	}
}

func CheckVersion(coll *mongo.Collection, key string, currentVersion int) (converted bool, err error) {
	type VersionTable struct {
		Key   string `bson:"key"`
		Value string `bson:"value"`
	}
	ctx, cancel := context.WithTimeout(context.Background(), time.Second*5)
	defer cancel()
	res, err := mongoutil.FindOne[VersionTable](ctx, coll, bson.M{"key": key})
	if err == nil {
		ver, err := strconv.Atoi(res.Value)
```

```
    if err != nil {
        return false, fmt.Errorf("version %s parse error %w", res.Value, err)
    }
    if ver >= currentVersion {
        return true, nil
    }
    return false, nil
  } else if errors.Is(err, mongo.ErrNoDocuments) {
    return false, nil
  } else {
    return false, err
  }
}

func SetVersion(coll *mongo.Collection, key string, version int) error {
  ctx, cancel := context.WithTimeout(context.Background(), time.Second*5)
  defer cancel()
  option := options.Update().SetUpsert(true)
  filter := bson.M{"key": key}
  update := bson.M{"$set": bson.M{"key": key, "value": strconv.Itoa(version)}}
  return mongoutil.UpdateOne(ctx, coll, filter, update, false, option)
}
```

tools/ncpu

tools/ncpu/main.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"fmt"
	"runtime"

	"go.uber.org/automaxprocs/maxprocs"
)

func main() {
	// Set maxprocs with a custom logger that does nothing to ignore logs.
	maxprocs.Set(maxprocs.Logger(func(string, ...interface{}) {
		// Intentionally left blank to suppress all log output from automaxprocs.
	}))

	// Now this will print the GOMAXPROCS value without printing the automaxprocs log message.
	fmt.Println(runtime.GOMAXPROCS(0))
}
```

tools/ncpu/main_test.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import "testing"

func Test_main(t *testing.T) {
    tests := []struct {
        name string
    }{
        {
            name: "Test_main",
        },
        {
            name: "Test_main2",
        },
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            main()
        })
    }
}
```

tools/check-component

tools/check-component/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"context"
	"flag"
	"fmt"
	"io"
	"log"
	"os"
	"path/filepath"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"github.com/openimsdk/tools/discovery/etcd"
	"github.com/openimsdk/tools/discovery/zookeeper"
	"github.com/openimsdk/tools/mq/kafka"
	"github.com/openimsdk/tools/s3/minio"
	"github.com/openimsdk/tools/system/program"
)

const maxRetry = 180

const (
	MountConfigFilePath = "CONFIG_PATH"
	DeploymentType      = "DEPLOYMENT_TYPE"
	KUBERNETES          = "kubernetes"
)

func CheckZookeeper(ctx context.Context, config *config.ZooKeeper) error {
	// Temporary disable logging
	originalLogger := log.Default().Writer()
	log.SetOutput(io.Discard)
	defer log.SetOutput(originalLogger) // Ensure logging is restored
	return zookeeper.Check(ctx, config.Address, config.Schema, zookeeper.WithUserNameAndPassword(config.Username, confi
}

func CheckEtcd(ctx context.Context, config *config.Etcd) error {
	return etcd.Check(ctx, config.Address, "/check_openim_component",
		true,
		etcd.WithDialTimeout(10*time.Second),
		etcd.WithMaxCallSendMsgSize(20*1024*1024),
		etcd.WithUsernameAndPassword(config.Username, config.Password))
}

func CheckMongo(ctx context.Context, config *config.Mongo) error {
```

```
    return mongoutil.Check(ctx, config.Build())
}

func CheckRedis(ctx context.Context, config *config.Redis) error {
    return redisutil.Check(ctx, config.Build())
}

func CheckMinIO(ctx context.Context, config *config.Minio) error {
    return minio.Check(ctx, config.Build())
}

func CheckKafka(ctx context.Context, conf *config.Kafka) error {
    return kafka.CheckHealth(ctx, conf.Build())
}

func initConfig(configDir string) (*config.Mongo, *config.Redis, *config.Kafka, *config.Minio, *config.Discovery, er
    var (
        mongoConfig = &config.Mongo{}
        redisConfig = &config.Redis{}
        kafkaConfig = &config.Kafka{}
        minioConfig = &config.Minio{}
        discovery   = &config.Discovery{}
        thirdConfig = &config.Third{}
    )

    err := config.Load(configDir, config.MongodbConfigFileName, config.EnvPrefixMap[config.MongodbConfigFileName], mong
    if err != nil {
        return nil, nil, nil, nil, nil, err
    }

    err = config.Load(configDir, config.RedisConfigFileName, config.EnvPrefixMap[config.RedisConfigFileName], redisConf
    if err != nil {
        return nil, nil, nil, nil, nil, err
    }

    err = config.Load(configDir, config.KafkaConfigFileName, config.EnvPrefixMap[config.KafkaConfigFileName], kafkaConf
    if err != nil {
        return nil, nil, nil, nil, nil, err
    }

    err = config.Load(configDir, config.OpenIMRPCThirdCfgFileName, config.EnvPrefixMap[config.OpenIMRPCThirdCfgFileName
    if err != nil {
        return nil, nil, nil, nil, nil, err
    }

    if thirdConfig.Object.Enable == "minio" {
        err = config.Load(configDir, config.MinioConfigFileName, config.EnvPrefixMap[config.MinioConfigFileName], minioCo
        if err != nil {
            return nil, nil, nil, nil, nil, err
        }
    } else {
        minioConfig = nil
    }
    err = config.Load(configDir, config.DiscoveryConfigFilename, config.EnvPrefixMap[config.DiscoveryConfigFilename], c
    if err != nil {
        return nil, nil, nil, nil, nil, err
    }
    return mongoConfig, redisConfig, kafkaConfig, minioConfig, discovery, nil
}

func main() {
    var index int
    var configDir string
    flag.IntVar(&index, "i", 0, "Index number")
    defaultConfigDir := filepath.Join("..", "..", "..", "..", "..", "config")
    flag.StringVar(&configDir, "c", defaultConfigDir, "Configuration dir")
```

```
    flag.Parse()

    fmt.Printf("%s Index: %d, Config Path: %s\n", filepath.Base(os.Args[0]), index, configDir)

    mongoConfig, redisConfig, kafkaConfig, minioConfig, zookeeperConfig, err := initConfig(configDir)
    if err != nil {
        program.ExitWithError(err)
    }

    ctx := context.Background()
    err = performChecks(ctx, mongoConfig, redisConfig, kafkaConfig, minioConfig, zookeeperConfig, maxRetry)
    if err != nil {
        // Assume program.ExitWithError logs the error and exits.
        // Replace with your error handling logic as necessary.
        program.ExitWithError(err)
    }
}

func performChecks(ctx context.Context, mongoConfig *config.Mongo, redisConfig *config.Redis, kafkaConfig *config.Ka
    checksDone := make(map[string]bool)

    checks := map[string]func(ctx context.Context) error{
        "Mongo": func(ctx context.Context) error {
            return CheckMongo(ctx, mongoConfig)
        },
        "Redis": func(ctx context.Context) error {
            return CheckRedis(ctx, redisConfig)
        },
        "Kafka": func(ctx context.Context) error {
            return CheckKafka(ctx, kafkaConfig)
        },
    }
    if minioConfig != nil {
        checks["MinIO"] = func(ctx context.Context) error {
            return CheckMinIO(ctx, minioConfig)
        }
    }
    if discovery.Enable == "etcd" {
        checks["Etcd"] = func(ctx context.Context) error {
            return CheckEtcd(ctx, &discovery.Etcd)
        }
    }

    for i := 0; i < maxRetry; i++ {
        allSuccess := true
        for name, check := range checks {
            if !checksDone[name] {
                if err := check(ctx); err != nil {
                    fmt.Printf("%s check failed: %v\n", name, err)
                    allSuccess = false
                } else {
                    fmt.Printf("%s check succeeded.\n", name)
                    checksDone[name] = true
                }
            }
        }

        if allSuccess {
            fmt.Println("All components checks passed successfully.")
            return nil
        }

        time.Sleep(1 * time.Second)
    }

    return fmt.Errorf("not all components checks passed successfully after %d attempts", maxRetry)
```

```
}
```

tools/changelog

tools/changelog/changelog.go

```go
package main

import (
	"encoding/json"
	"fmt"
	"io"
	"net/http"
	"os"
	"regexp"
	"strings"
)

// You can specify a tag as a command line argument to generate the changelog for a specific version.
// Example: go run tools/changelog/changelog.go v0.0.33
// If no tag is provided, the latest release will be used.

// Setting repo owner and repo name by generate changelog
const (
	repoOwner = "openimsdk"
	repoName  = "open-im-server"
)

// GitHubRepo struct represents the repo details.
type GitHubRepo struct {
	Owner         string
	Repo          string
	FullChangelog string
}

// ReleaseData represents the JSON structure for release data.
type ReleaseData struct {
	TagName   string `json:"tag_name"`
	Body      string `json:"body"`
	HtmlUrl   string `json:"html_url"`
	Published string `json:"published_at"`
}

// Method to classify and format release notes.
func (g *GitHubRepo) classifyReleaseNotes(body string) map[string][]string {
	result := map[string][]string{
		"feat":     {},
		"fix":      {},
		"chore":    {},
		"refactor": {},
		"build":    {},
		"other":    {},
	}

	// Regular expression to extract PR number and URL (case insensitive)
	rePR := regexp.MustCompile(`(?i)in (https://github\.com/[^\s]+/pull/(\d+))`)

	// Split the body into individual lines.
	lines := strings.Split(body, "\n")

	for _, line := range lines {
		// Skip lines that contain "deps: Merge"
		if strings.Contains(strings.ToLower(line), "deps: merge #") {
			continue
		}

		// Use a regular expression to extract Full Changelog link and its title (case insensitive).
```

```
    if strings.Contains(strings.ToLower(line), "**full changelog**") {
        matches := regexp.MustCompile(`(?i)\*\*full changelog\*\*: (https://github\.com/[^\s]+/compare/([^\s]+))`).FindS
        if len(matches) > 2 {
            // Format the Full Changelog link with title
            g.FullChangelog = fmt.Sprintf("[%s](%s)", matches[2], matches[1])
        }
        continue // Skip further processing for this line.
    }

    if strings.HasPrefix(line, "*") {
        var category string

        // Use strings.ToLower to make the matching case insensitive
        lowerLine := strings.ToLower(line)

        // Determine the category based on the prefix (case insensitive).
        if strings.HasPrefix(lowerLine, "* feat") {
            category = "feat"
        } else if strings.HasPrefix(lowerLine, "* fix") {
            category = "fix"
        } else if strings.HasPrefix(lowerLine, "* chore") {
            category = "chore"
        } else if strings.HasPrefix(lowerLine, "* refactor") {
            category = "refactor"
        } else if strings.HasPrefix(lowerLine, "* build") {
            category = "build"
        } else {
            category = "other"
        }

        // Extract PR number and URL (case insensitive)
        matches := rePR.FindStringSubmatch(line)
        if len(matches) == 3 {
            prURL := matches[1]
            prNumber := matches[2]
            // Format the line with the PR link and use original content for the final result
            formattedLine := fmt.Sprintf("* %s [#%s](%s)", strings.Split(line, " by ")[0][2:], prNumber, prURL)
            result[category] = append(result[category], formattedLine)
        } else {
            // If no PR link is found, just add the line as is
            result[category] = append(result[category], line)
        }
    }
}

return result
}

// Method to generate the final changelog.
func (g *GitHubRepo) generateChangelog(tag, date, htmlURL, body string) string {
    sections := g.classifyReleaseNotes(body)

    // Convert ISO 8601 date to simpler format (YYYY-MM-DD)
    formattedDate := date[:10]

    // Changelog header with tag, date, and links.
    changelog := fmt.Sprintf("## [%s](%s) \t(%s)\n\n", tag, htmlURL, formattedDate)

    if len(sections["feat"]) > 0 {
        changelog += "### New Features\n" + strings.Join(sections["feat"], "\n") + "\n\n"
    }
    if len(sections["fix"]) > 0 {
        changelog += "### Bug Fixes\n" + strings.Join(sections["fix"], "\n") + "\n\n"
    }
    if len(sections["chore"]) > 0 {
        changelog += "### Chores\n" + strings.Join(sections["chore"], "\n") + "\n\n"
```

```
    }
    if len(sections["refactor"]) > 0 {
        changelog += "### Refactors\n" + strings.Join(sections["refactor"], "\n") + "\n\n"
    }
    if len(sections["build"]) > 0 {
        changelog += "### Builds\n" + strings.Join(sections["build"], "\n") + "\n\n"
    }
    if len(sections["other"]) > 0 {
        changelog += "### Others\n" + strings.Join(sections["other"], "\n") + "\n\n"
    }

    if g.FullChangelog != "" {
        changelog += fmt.Sprintf("**Full Changelog**: %s\n", g.FullChangelog)
    }

    return changelog
}

// Method to fetch release data from GitHub API.
func (g *GitHubRepo) fetchReleaseData(version string) (*ReleaseData, error) {
    var apiURL string

    if version == "" {
        // Fetch the latest release.
        apiURL = fmt.Sprintf("https://api.github.com/repos/%s/%s/releases/latest", g.Owner, g.Repo)
    } else {
        // Fetch a specific version.
        apiURL = fmt.Sprintf("https://api.github.com/repos/%s/%s/releases/tags/%s", g.Owner, g.Repo, version)
    }

    resp, err := http.Get(apiURL)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return nil, err
    }

    var releaseData ReleaseData
    err = json.Unmarshal(body, &releaseData)
    if err != nil {
        return nil, err
    }

    return &releaseData, nil
}

func main() {
    repo := &GitHubRepo{Owner: repoOwner, Repo: repoName}

    // Get the version from command line arguments, if provided
    var version string // Default is use latest

    if len(os.Args) > 1 {
        version = os.Args[1] // Use the provided version
    }

    // Fetch release data (either for latest or specific version)
    releaseData, err := repo.fetchReleaseData(version)
    if err != nil {
        fmt.Println("Error fetching release data:", err)
        return
    }
```

```
    // Generate and print the formatted changelog
    changelog := repo.generateChangelog(releaseData.TagName, releaseData.Published, releaseData.HtmlUrl, releaseData.Bo
    fmt.Println(changelog)
}
```

tools/stress-test-v2

tools/stress-test-v2/main.go

```
package main

import (
	"bytes"
	"context"
	"encoding/json"
	"flag"
	"fmt"
	"io"
	"net/http"
	"os"
	"os/signal"
	"sync"
	"syscall"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/sdkws"
	pbuser "github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/system/program"
)

// 1. Create 100K New Users
// 2. Create 100 100K Groups
// 3. Create 1000 999 Groups
// 4. Send message to 100K Groups every second
// 5. Send message to 999 Groups every minute

var (
	//  Use default userIDs List for testing, need to be created.
	TestTargetUserList = []string{
		// "<need-update-it>",
	}
	// DefaultGroupID = "<need-update-it>" // Use default group ID for testing, need to be created.
)

var (
	ApiAddress string

	// API method
	GetAdminToken      = "/auth/get_admin_token"
	UserCheck          = "/user/account_check"
	CreateUser         = "/user/user_register"
	ImportFriend       = "/friend/import_friend"
	InviteToGroup      = "/group/invite_user_to_group"
	GetGroupMemberInfo = "/group/get_group_members_info"
	SendMsg            = "/msg/send_msg"
	CreateGroup        = "/group/create_group"
	GetUserToken       = "/auth/user_token"
)

const (
	MaxUser            = 100000
	Max100KGroup       = 100
	Max999Group        = 1000
	MaxInviteUserLimit = 999
```

```
	CreateUserTicker         = 1 * time.Second
	CreateGroupTicker        = 1 * time.Second
	Create100KGroupTicker    = 1 * time.Second
	Create999GroupTicker     = 1 * time.Second
	SendMsgTo100KGroupTicker = 1 * time.Second
	SendMsgTo999GroupTicker  = 1 * time.Minute
)

type BaseResp struct {
	ErrCode int             `json:"errCode"`
	ErrMsg  string          `json:"errMsg"`
	Data    json.RawMessage `json:"data"`
}

type StressTest struct {
	Conf                  *conf
	AdminUserID           string
	AdminToken            string
	DefaultGroupID        string
	DefaultUserID         string
	UserCounter           int
	CreateUserCounter     int
	Create100kGroupCounter int
	Create999GroupCounter  int
	MsgCounter            int
	CreatedUsers          []string
	CreatedGroups         []string
	Mutex                 sync.Mutex
	Ctx                   context.Context
	Cancel                context.CancelFunc
	HttpClient            *http.Client
	Wg                    sync.WaitGroup
	Once                  sync.Once
}

type conf struct {
	Share config.Share
	Api   config.API
}

func initConfig(configDir string) (*config.Share, *config.API, error) {
	var (
		share     = &config.Share{}
		apiConfig = &config.API{}
	)

	err := config.Load(configDir, config.ShareFileName, config.EnvPrefixMap[config.ShareFileName], share)
	if err != nil {
		return nil, nil, err
	}

	err = config.Load(configDir, config.OpenIMAPICfgFileName, config.EnvPrefixMap[config.OpenIMAPICfgFileName], apiConf
	if err != nil {
		return nil, nil, err
	}

	return share, apiConfig, nil
}

// Post Request
func (st *StressTest) PostRequest(ctx context.Context, url string, reqbody any) ([]byte, error) {
	// Marshal body
	jsonBody, err := json.Marshal(reqbody)
	if err != nil {
		log.ZError(ctx, "Failed to marshal request body", err, "url", url, "reqbody", reqbody)
```

```
    return nil, err
  }

  req, err := http.NewRequest(http.MethodPost, url, bytes.NewReader(jsonBody))
  if err != nil {
    return nil, err
  }
  req.Header.Set("Content-Type", "application/json")
  req.Header.Set("operationID", st.AdminUserID)
  if st.AdminToken != "" {
    req.Header.Set("token", st.AdminToken)
  }

  // log.ZInfo(ctx, "Header info is ", "Content-Type", "application/json", "operationID", st.AdminUserID, "token", st

  resp, err := st.HttpClient.Do(req)
  if err != nil {
    log.ZError(ctx, "Failed to send request", err, "url", url, "reqbody", reqbody)
    return nil, err
  }
  defer resp.Body.Close()

  respBody, err := io.ReadAll(resp.Body)
  if err != nil {
    log.ZError(ctx, "Failed to read response body", err, "url", url)
    return nil, err
  }

  var baseResp BaseResp
  if err := json.Unmarshal(respBody, &baseResp); err != nil {
    log.ZError(ctx, "Failed to unmarshal response body", err, "url", url, "respBody", string(respBody))
    return nil, err
  }

  if baseResp.ErrCode != 0 {
    err = fmt.Errorf(baseResp.ErrMsg)
    log.ZError(ctx, "Failed to send request", err, "url", url, "reqbody", reqbody, "resp", baseResp)
    return nil, err
  }

  return baseResp.Data, nil
}

func (st *StressTest) GetAdminToken(ctx context.Context) (string, error) {
  req := auth.GetAdminTokenReq{
    Secret: st.Conf.Share.Secret,
    UserID: st.AdminUserID,
  }

  resp, err := st.PostRequest(ctx, ApiAddress+GetAdminToken, &req)
  if err != nil {
    return "", err
  }

  data := &auth.GetAdminTokenResp{}
  if err := json.Unmarshal(resp, &data); err != nil {
    return "", err
  }

  return data.Token, nil
}

func (st *StressTest) CheckUser(ctx context.Context, userIDs []string) ([]string, error) {
  req := pbuser.AccountCheckReq{
    CheckUserIDs: userIDs,
  }
```

```
	resp, err := st.PostRequest(ctx, ApiAddress+UserCheck, &req)
	if err != nil {
		return nil, err
	}

	data := &pbuser.AccountCheckResp{}
	if err := json.Unmarshal(resp, &data); err != nil {
		return nil, err
	}

	unRegisteredUserIDs := make([]string, 0)

	for _, res := range data.Results {
		if res.AccountStatus == constant.UnRegistered {
			unRegisteredUserIDs = append(unRegisteredUserIDs, res.UserID)
		}
	}

	return unRegisteredUserIDs, nil
}

func (st *StressTest) CreateUser(ctx context.Context, userID string) (string, error) {
	user := &sdkws.UserInfo{
		UserID:   userID,
		Nickname: userID,
	}

	req := pbuser.UserRegisterReq{
		Users: []*sdkws.UserInfo{user},
	}

	_, err := st.PostRequest(ctx, ApiAddress+CreateUser, &req)
	if err != nil {
		return "", err
	}

	st.UserCounter++
	return userID, nil
}

func (st *StressTest) CreateUserBatch(ctx context.Context, userIDs []string) error {
	// The method can import a large number of users at once.
	var userList []*sdkws.UserInfo

	defer st.Once.Do(
		func() {
			st.DefaultUserID = userIDs[0]
			fmt.Println("Default Send User Created ID:", st.DefaultUserID)
		})

	needUserIDs, err := st.CheckUser(ctx, userIDs)
	if err != nil {
		return err
	}

	for _, userID := range needUserIDs {
		user := &sdkws.UserInfo{
			UserID:   userID,
			Nickname: userID,
		}
		userList = append(userList, user)
	}

	req := pbuser.UserRegisterReq{
		Users: userList,
```

```
	}

	_, err = st.PostRequest(ctx, ApiAddress+CreateUser, &req)
	if err != nil {
		return err
	}

	st.UserCounter += len(userList)
	return nil
}

func (st *StressTest) GetGroupMembersInfo(ctx context.Context, groupID string, userIDs []string) ([]string, error) {
	needInviteUserIDs := make([]string, 0)

	const maxBatchSize = 500
	if len(userIDs) > maxBatchSize {
		for i := 0; i < len(userIDs); i += maxBatchSize {
			end := min(i+maxBatchSize, len(userIDs))
			batchUserIDs := userIDs[i:end]

			// log.ZInfo(ctx, "Processing group members batch", "groupID", groupID, "batch", i/maxBatchSize+1,
			// 	"batchUserCount", len(batchUserIDs))

			// Process a single batch
			batchReq := group.GetGroupMembersInfoReq{
				GroupID: groupID,
				UserIDs: batchUserIDs,
			}

			resp, err := st.PostRequest(ctx, ApiAddress+GetGroupMemberInfo, &batchReq)
			if err != nil {
				log.ZError(ctx, "Batch query failed", err, "batch", i/maxBatchSize+1)
				continue
			}

			data := &group.GetGroupMembersInfoResp{}
			if err := json.Unmarshal(resp, &data); err != nil {
				log.ZError(ctx, "Failed to parse batch response", err, "batch", i/maxBatchSize+1)
				continue
			}

			// Process the batch results
			existingMembers := make(map[string]bool)
			for _, member := range data.Members {
				existingMembers[member.UserID] = true
			}

			for _, userID := range batchUserIDs {
				if !existingMembers[userID] {
					needInviteUserIDs = append(needInviteUserIDs, userID)
				}
			}
		}

		return needInviteUserIDs, nil
	}

	req := group.GetGroupMembersInfoReq{
		GroupID: groupID,
		UserIDs: userIDs,
	}

	resp, err := st.PostRequest(ctx, ApiAddress+GetGroupMemberInfo, &req)
	if err != nil {
		return nil, err
	}
```

```
	data := &group.GetGroupMembersInfoResp{}
	if err := json.Unmarshal(resp, &data); err != nil {
		return nil, err
	}

	existingMembers := make(map[string]bool)
	for _, member := range data.Members {
		existingMembers[member.UserID] = true
	}

	for _, userID := range userIDs {
		if !existingMembers[userID] {
			needInviteUserIDs = append(needInviteUserIDs, userID)
		}
	}

	return needInviteUserIDs, nil
}

func (st *StressTest) InviteToGroup(ctx context.Context, groupID string, userIDs []string) error {
	req := group.InviteUserToGroupReq{
		GroupID:        groupID,
		InvitedUserIDs: userIDs,
	}
	_, err := st.PostRequest(ctx, ApiAddress+InviteToGroup, &req)
	if err != nil {
		return err
	}

	return nil
}

func (st *StressTest) SendMsg(ctx context.Context, userID string, groupID string) error {
	contentObj := map[string]any{
		// "content": fmt.Sprintf("index %d. The current time is %s", st.MsgCounter, time.Now().Format("2006-01-02 15:04:0
		"content": fmt.Sprintf("The current time is %s", time.Now().Format("2006-01-02 15:04:05.000")),
	}

	req := &apistruct.SendMsgReq{
		SendMsg: apistruct.SendMsg{
			SendID:         userID,
			SenderNickname: userID,
			GroupID:        groupID,
			ContentType:    constant.Text,
			SessionType:    constant.ReadGroupChatType,
			Content:        contentObj,
		},
	}

	_, err := st.PostRequest(ctx, ApiAddress+SendMsg, &req)
	if err != nil {
		log.ZError(ctx, "Failed to send message", err, "userID", userID, "req", &req)
		return err
	}

	st.MsgCounter++

	return nil
}

// Max userIDs number is 1000
func (st *StressTest) CreateGroup(ctx context.Context, groupID string, userID string, userIDsList []string) (string,
	groupInfo := &sdkws.GroupInfo{
		GroupID:   groupID,
		GroupName: groupID,
```

```
    GroupType: constant.WorkingGroup,
  }

  req := group.CreateGroupReq{
    OwnerUserID:   userID,
    MemberUserIDs: userIDsList,
    GroupInfo:     groupInfo,
  }

  resp := group.CreateGroupResp{}

  response, err := st.PostRequest(ctx, ApiAddress+CreateGroup, &req)
  if err != nil {
    return "", err
  }

  if err := json.Unmarshal(response, &resp); err != nil {
    return "", err
  }

  // st.GroupCounter++

  return resp.GroupInfo.GroupID, nil
}

func main() {
  var configPath string
  // defaultConfigDir := filepath.Join("..", "..", "..", "..", "..", "config")
  // flag.StringVar(&configPath, "c", defaultConfigDir, "config path")
  flag.StringVar(&configPath, "c", "", "config path")
  flag.Parse()

  if configPath == "" {
    _, _ = fmt.Fprintln(os.Stderr, "config path is empty")
    os.Exit(1)
    return
  }

  fmt.Printf(" Config Path: %s\n", configPath)

  share, apiConfig, err := initConfig(configPath)
  if err != nil {
    program.ExitWithError(err)
    return
  }

  ApiAddress = fmt.Sprintf("http://%s:%s", "127.0.0.1", fmt.Sprint(apiConfig.Api.Ports[0]))

  ctx, cancel := context.WithCancel(context.Background())
  // ch := make(chan struct{})

  st := &StressTest{
    Conf: &conf{
      Share: *share,
      Api:   *apiConfig,
    },
    AdminUserID: share.IMAdminUser.UserIDs[0],
    Ctx:         ctx,
    Cancel:      cancel,
    HttpClient: &http.Client{
      Timeout: 50 * time.Second,
    },
  }

  c := make(chan os.Signal, 1)
  signal.Notify(c, os.Interrupt, syscall.SIGTERM)
```

```
	go func() {
		<-c
		fmt.Println("\nReceived stop signal, stopping...")

		go func() {
			// time.Sleep(5 * time.Second)
			fmt.Println("Force exit")
			os.Exit(0)
		}()

		st.Cancel()
	}()

	token, err := st.GetAdminToken(st.Ctx)
	if err != nil {
		log.ZError(ctx, "Get Admin Token failed.", err, "AdminUserID", st.AdminUserID)
	}

	st.AdminToken = token
	fmt.Println("Admin Token:", st.AdminToken)
	fmt.Println("ApiAddress:", ApiAddress)
	for i := 0; i < MaxUser; i++ {
		userID := fmt.Sprintf("v2_StressTest_User_%d", i)
		st.CreatedUsers = append(st.CreatedUsers, userID)
		st.CreateUserCounter++
	}

	// err = st.CreateUserBatch(st.Ctx, st.CreatedUsers)
	// if err != nil {
	// 	log.ZError(ctx, "Create user failed.", err)
	// }

	const batchSize = 1000
	totalUsers := len(st.CreatedUsers)
	successCount := 0

	if st.DefaultUserID == "" && len(st.CreatedUsers) > 0 {
		st.DefaultUserID = st.CreatedUsers[0]
	}

	for i := 0; i < totalUsers; i += batchSize {
		end := min(i+batchSize, totalUsers)

		userBatch := st.CreatedUsers[i:end]
		log.ZInfo(st.Ctx, "Creating user batch", "batch", i/batchSize+1, "count", len(userBatch))

		err = st.CreateUserBatch(st.Ctx, userBatch)
		if err != nil {
			log.ZError(st.Ctx, "Batch user creation failed", err, "batch", i/batchSize+1)
		} else {
			successCount += len(userBatch)
			log.ZInfo(st.Ctx, "Batch user creation succeeded", "batch", i/batchSize+1,
				"progress", fmt.Sprintf("%d/%d", successCount, totalUsers))
		}
	}

	// Execute create 100k group
	st.Wg.Add(1)
	go func() {
		defer st.Wg.Done()

		create100kGroupTicker := time.NewTicker(Create100KGroupTicker)
		defer create100kGroupTicker.Stop()

		for i := 0; i < Max100KGroup; i++ {
			select {
```

```
    case <-st.Ctx.Done():
        log.ZInfo(st.Ctx, "Stop Create 100K Group")
        return

    case <-create100kGroupTicker.C:
        // Create 100K groups
        st.Wg.Add(1)
        go func(idx int) {
            defer st.Wg.Done()
            defer func() {
                st.Create100kGroupCounter++
            }()

            groupID := fmt.Sprintf("v2_StressTest_Group_100K_%d", idx)

            if _, err = st.CreateGroup(st.Ctx, groupID, st.DefaultUserID, TestTargetUserList); err != nil {
                log.ZError(st.Ctx, "Create group failed.", err)
                // continue
            }

            for i := 0; i < MaxUser/MaxInviteUserLimit; i++ {
                InviteUserIDs := make([]string, 0)
                // ensure TargetUserList is in group
                InviteUserIDs = append(InviteUserIDs, TestTargetUserList...)

                startIdx := max(i*MaxInviteUserLimit, 1)
                endIdx := min((i+1)*MaxInviteUserLimit, MaxUser)

                for j := startIdx; j < endIdx; j++ {
                    userCreatedID := fmt.Sprintf("v2_StressTest_User_%d", j)
                    InviteUserIDs = append(InviteUserIDs, userCreatedID)
                }

                if len(InviteUserIDs) == 0 {
                    log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
                    continue
                }

                InviteUserIDs, err := st.GetGroupMembersInfo(ctx, groupID, InviteUserIDs)
                if err != nil {
                    log.ZError(st.Ctx, "GetGroupMembersInfo failed.", err, "groupID", groupID)
                    continue
                }

                if len(InviteUserIDs) == 0 {
                    log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
                    continue
                }

                // Invite To Group
                if err = st.InviteToGroup(st.Ctx, groupID, InviteUserIDs); err != nil {
                    log.ZError(st.Ctx, "Invite To Group failed.", err, "UserID", InviteUserIDs)
                    continue
                    // os.Exit(1)
                    // return
                }
            }
        }(i)
    }
  }
}()

// create 999 groups
st.Wg.Add(1)
go func() {
  defer st.Wg.Done()
```

```
    create999GroupTicker := time.NewTicker(Create999GroupTicker)
    defer create999GroupTicker.Stop()

    for i := 0; i < Max999Group; i++ {
        select {
        case <-st.Ctx.Done():
            log.ZInfo(st.Ctx, "Stop Create 999 Group")
            return

        case <-create999GroupTicker.C:
            // Create 999 groups
            st.Wg.Add(1)
            go func(idx int) {
                defer st.Wg.Done()
                defer func() {
                    st.Create999GroupCounter++
                }()

                groupID := fmt.Sprintf("v2_StressTest_Group_1K_%d", idx)

                if _, err = st.CreateGroup(st.Ctx, groupID, st.DefaultUserID, TestTargetUserList); err != nil {
                    log.ZError(st.Ctx, "Create group failed.", err)
                    // continue
                }
                for i := 0; i < MaxUser/MaxInviteUserLimit; i++ {
                    InviteUserIDs := make([]string, 0)
                    // ensure TargetUserList is in group
                    InviteUserIDs = append(InviteUserIDs, TestTargetUserList...)

                    startIdx := max(i*MaxInviteUserLimit, 1)
                    endIdx := min((i+1)*MaxInviteUserLimit, MaxUser)

                    for j := startIdx; j < endIdx; j++ {
                        userCreatedID := fmt.Sprintf("v2_StressTest_User_%d", j)
                        InviteUserIDs = append(InviteUserIDs, userCreatedID)
                    }

                    if len(InviteUserIDs) == 0 {
                        log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
                        continue
                    }

                    InviteUserIDs, err := st.GetGroupMembersInfo(ctx, groupID, InviteUserIDs)
                    if err != nil {
                        log.ZError(st.Ctx, "GetGroupMembersInfo failed.", err, "groupID", groupID)
                        continue
                    }

                    if len(InviteUserIDs) == 0 {
                        log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
                        continue
                    }

                    // Invite To Group
                    if err = st.InviteToGroup(st.Ctx, groupID, InviteUserIDs); err != nil {
                        log.ZError(st.Ctx, "Invite To Group failed.", err, "UserID", InviteUserIDs)
                        continue
                        // os.Exit(1)
                        // return
                    }
                }
            }(i)
        }
    }
}()
```

```
// Send message to 100K groups
st.Wg.Wait()
fmt.Println("All groups created successfully, starting to send messages...")
log.ZInfo(ctx, "All groups created successfully, starting to send messages...")

var groups100K []string
var groups999 []string

for i := 0; i < Max100KGroup; i++ {
groupID := fmt.Sprintf("v2_StressTest_Group_100K_%d", i)
groups100K = append(groups100K, groupID)
}

for i := 0; i < Max999Group; i++ {
groupID := fmt.Sprintf("v2_StressTest_Group_1K_%d", i)
groups999 = append(groups999, groupID)
}

send100kGroupLimiter := make(chan struct{}, 20)
send999GroupLimiter := make(chan struct{}, 100)

// execute Send message to 100K groups
go func() {
ticker := time.NewTicker(SendMsgTo100KGroupTicker)
defer ticker.Stop()

for {
select {
case <-st.Ctx.Done():
log.ZInfo(st.Ctx, "Stop Send Message to 100K Group")
return

case <-ticker.C:
// Send message to 100K groups
for _, groupID := range groups100K {
send100kGroupLimiter <- struct{}{}
go func(groupID string) {
defer func() { <-send100kGroupLimiter }()
if err := st.SendMsg(st.Ctx, st.DefaultUserID, groupID); err != nil {
log.ZError(st.Ctx, "Send message to 100K group failed.", err)
}
}(groupID)
}
// log.ZInfo(st.Ctx, "Send message to 100K groups successfully.")
}
}
}()

// execute Send message to 999 groups
go func() {
ticker := time.NewTicker(SendMsgTo999GroupTicker)
defer ticker.Stop()

for {
select {
case <-st.Ctx.Done():
log.ZInfo(st.Ctx, "Stop Send Message to 999 Group")
return

case <-ticker.C:
// Send message to 999 groups
for _, groupID := range groups999 {
send999GroupLimiter <- struct{}{}
go func(groupID string) {
defer func() { <-send999GroupLimiter }()
```

```
                if err := st.SendMsg(st.Ctx, st.DefaultUserID, groupID); err != nil {
                    log.ZError(st.Ctx, "Send message to 999 group failed.", err)
                }
            }(groupID)
        }
        // log.ZInfo(st.Ctx, "Send message to 999 groups successfully.")
    }
}()

<-st.Ctx.Done()
fmt.Println("Received signal to exit, shutting down...")
}
```

tools/yamlfmt

tools/yamlfmt/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

// OPENIM plan on prow tools
package main

import (
	"flag"
	"fmt"
	"io"
	"os"

	"gopkg.in/yaml.v3"
)

func main() {
	// Prow OWNERs file defines the default indent as 2 spaces.
	indent := flag.Int("indent", 2, "default indent")
	flag.Parse()
	for _, path := range flag.Args() {
		sourceYaml, err := os.ReadFile(path)
		if err != nil {
			fmt.Fprintf(os.Stderr, "%s: %v\n", path, err)
			continue
		}
		rootNode, err := fetchYaml(sourceYaml)
		if err != nil {
			fmt.Fprintf(os.Stderr, "%s: %v\n", path, err)
			continue
		}
		writer, err := os.OpenFile(path, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0o666)
		if err != nil {
			fmt.Fprintf(os.Stderr, "%s: %v\n", path, err)
			continue
		}
		err = streamYaml(writer, indent, rootNode)
		if err != nil {
			fmt.Fprintf(os.Stderr, "%s: %v\n", path, err)
			continue
		}
	}
}

func fetchYaml(sourceYaml []byte) (*yaml.Node, error) {
	rootNode := yaml.Node{}
	err := yaml.Unmarshal(sourceYaml, &rootNode)
	if err != nil {
		return nil, err
	}
	return &rootNode, nil
```

```
}

func streamYaml(writer io.Writer, indent *int, in *yaml.Node) error {
	encoder := yaml.NewEncoder(writer)
	encoder.SetIndent(*indent)
	err := encoder.Encode(in)
	if err != nil {
		return err
	}
	return encoder.Close()
}
```

tools/yamlfmt/main_test.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"bufio"
	"bytes"
	"reflect"
	"testing"

	"github.com/likexian/gokit/assert"
	"gopkg.in/yaml.v3"
)

func Test_main(t *testing.T) {
	sourceYaml := ` # See the OWNERS docs at https://go.k8s.io/owners
approvers:
- dep-approvers
- thockin         # Network
- liggitt

labels:
- sig/architecture
`

	outputYaml := `# See the OWNERS docs at https://go.k8s.io/owners
approvers:
  - dep-approvers
  - thockin # Network
  - liggitt
labels:
  - sig/architecture
`
	node, _ := fetchYaml([]byte(sourceYaml))
	var output bytes.Buffer
	indent := 2
	writer := bufio.NewWriter(&output)
	_ = streamYaml(writer, &indent, node)
	_ = writer.Flush()
	assert.Equal(t, outputYaml, string(output.Bytes()), "yaml was not formatted correctly")
}

func Test_fetchYaml(t *testing.T) {
	type args struct {
		sourceYaml []byte
	}
	tests := []struct {
		name    string
		args    args
		want    *yaml.Node
		wantErr bool
	}{
```

```
  {
    name: "Valid YAML",
    args: args{sourceYaml: []byte("key: value")},
    want: &yaml.Node{
      Kind:  yaml.MappingNode,
      Tag:   "!!map",
      Value: "",
      Content: []*yaml.Node{
        {
          Kind:  yaml.ScalarNode,
          Tag:   "!!str",
          Value: "key",
        },
        {
          Kind:  yaml.ScalarNode,
          Tag:   "!!str",
          Value: "value",
        },
      },
    },
    wantErr: false,
  },
  {
    name:    "Invalid YAML",
    args:    args{sourceYaml: []byte("key:")},
    want:    nil,
    wantErr: true,
  },
 }
 for _, tt := range tests {
  t.Run(tt.name, func(t *testing.T) {
   got, err := fetchYaml(tt.args.sourceYaml)
   if (err != nil) != tt.wantErr {
    t.Errorf("fetchYaml() error = %v, wantErr %v", err, tt.wantErr)
    return
   }
   if !reflect.DeepEqual(got, tt.want) {
    t.Errorf("fetchYaml() = %v, want %v", got, tt.want)
   }
  })
 }
}

func Test_streamYaml(t *testing.T) {
 type args struct {
  indent *int
  in     *yaml.Node
 }
 defaultIndent := 2
 tests := []struct {
  name       string
  args       args
  wantWriter string
  wantErr    bool
 }{
  {
   name: "Valid YAML node with default indent",
   args: args{
    indent: &defaultIndent,
    in: &yaml.Node{
     Kind:  yaml.MappingNode,
     Tag:   "!!map",
     Value: "",
     Content: []*yaml.Node{
      {
       Kind:  yaml.ScalarNode,
```

```
                Tag:   "!!str",
                Value: "key",
            },
            {
                Kind:  yaml.ScalarNode,
                Tag:   "!!str",
                Value: "value",
            },
        },
    },
    wantWriter: "key: value\n",
    wantErr:    false,
  },
 }
 for _, tt := range tests {
  t.Run(tt.name, func(t *testing.T) {
   writer := &bytes.Buffer{}
   if err := streamYaml(writer, tt.args.indent, tt.args.in); (err != nil) != tt.wantErr {
    t.Errorf("streamYaml() error = %v, wantErr %v", err, tt.wantErr)
    return
   }
   if gotWriter := writer.String(); gotWriter != tt.wantWriter {
    t.Errorf("streamYaml() = %v, want %v", gotWriter, tt.wantWriter)
   }
  })
 }
}
```

tools/url2im

tools/url2im/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"flag"
	"log"
	"os"
	"path/filepath"
	"time"

	"github.com/openimsdk/open-im-server/v3/tools/url2im/pkg"
)

/*take.txt
{"url":"http://xxx/xxxx","name":"xxxx","contentType":"image/jpeg"}
{"url":"http://xxx/xxxx","name":"xxxx","contentType":"image/jpeg"}
{"url":"http://xxx/xxxx","name":"xxxx","contentType":"image/jpeg"}
*/

func main() {
	var conf pkg.Config // Configuration object, '*' denotes required fields

	// *Required*: Path for the task log file
	flag.StringVar(&conf.TaskPath, "task", "take.txt", "Path for the task log file")

	// Optional: Path for the progress log file
	flag.StringVar(&conf.ProgressPath, "progress", "", "Path for the progress log file")

	// Number of concurrent operations
	flag.IntVar(&conf.Concurrency, "concurrency", 1, "Number of concurrent operations")

	// Number of retry attempts
	flag.IntVar(&conf.Retry, "retry", 1, "Number of retry attempts")

	// Optional: Path for the temporary directory
	flag.StringVar(&conf.TempDir, "temp", "", "Path for the temporary directory")

	// Cache size in bytes (downloads move to disk when exceeded)
	flag.Int64Var(&conf.CacheSize, "cache", 1024*1024*100, "Cache size in bytes")

	// Request timeout in milliseconds
	flag.Int64Var((*int64)(&conf.Timeout), "timeout", 5000, "Request timeout in milliseconds")

	// *Required*: API endpoint for the IM service
	flag.StringVar(&conf.Api, "api", "http://127.0.0.1:10002", "API endpoint for the IM service")

	// IM administrator's user ID
	flag.StringVar(&conf.UserID, "userID", "openIM123456", "IM administrator's user ID")
```

```
	// Secret for the IM configuration
	flag.StringVar(&conf.Secret, "secret", "openIM123", "Secret for the IM configuration")

	flag.Parse()
	if !filepath.IsAbs(conf.TaskPath) {
		var err error
		conf.TaskPath, err = filepath.Abs(conf.TaskPath)
		if err != nil {
			log.Println("get abs path err:", err)
			return
		}
	}
	if conf.ProgressPath == "" {
		conf.ProgressPath = conf.TaskPath + ".progress.txt"
	} else if !filepath.IsAbs(conf.ProgressPath) {
		var err error
		conf.ProgressPath, err = filepath.Abs(conf.ProgressPath)
		if err != nil {
			log.Println("get abs path err:", err)
			return
		}
	}
	if conf.TempDir == "" {
		conf.TempDir = conf.TaskPath + ".temp"
	}
	if info, err := os.Stat(conf.TempDir); err == nil {
		if !info.IsDir() {
			log.Printf("temp dir %s is not dir\n", err)
			return
		}
	} else if os.IsNotExist(err) {
		if err := os.MkdirAll(conf.TempDir, os.ModePerm); err != nil {
			log.Printf("mkdir temp dir %s err %+v\n", conf.TempDir, err)
			return
		}
		defer os.RemoveAll(conf.TempDir)
	} else {
		log.Println("get temp dir err:", err)
		return
	}
	if conf.Concurrency <= 0 {
		conf.Concurrency = 1
	}
	if conf.Retry <= 0 {
		conf.Retry = 1
	}
	if conf.CacheSize <= 0 {
		conf.CacheSize = 1024 * 1024 * 100 // 100M
	}
	if conf.Timeout <= 0 {
		conf.Timeout = 5000
	}
	conf.Timeout = conf.Timeout * time.Millisecond
	if err := pkg.Run(conf); err != nil {
		log.Println("main err:", err)
	}
}
```

tools/url2im/pkg

tools/url2im/pkg/api.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import (
	"bytes"
	"context"
	"encoding/json"
	"fmt"
	"io"
	"net/http"

	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/protocol/third"
	"github.com/openimsdk/tools/errs"
)

type Api struct {
	Api    string
	UserID string
	Secret string
	Token  string
	Client *http.Client
}

func (a *Api) apiPost(ctx context.Context, path string, req any, resp any) error {
	operationID, _ := ctx.Value("operationID").(string)
	if operationID == "" {
		return errs.New("call api operationID is empty")
	}
	reqBody, err := json.Marshal(req)
	if err != nil {
		return err
	}
	request, err := http.NewRequestWithContext(ctx, http.MethodPost, a.Api+path, bytes.NewReader(reqBody))
	if err != nil {
		return err
	}
	DefaultRequestHeader(request.Header)
	request.ContentLength = int64(len(reqBody))
	request.Header.Set("Content-Type", "application/json")
	request.Header.Set("operationID", operationID)
	if a.Token != "" {
		request.Header.Set("token", a.Token)
	}
	response, err := a.Client.Do(request)
	if err != nil {
		return err
	}
```

```
	defer response.Body.Close()
	body, err := io.ReadAll(response.Body)
	if err != nil {
		return err
	}
	if response.StatusCode != http.StatusOK {
		return fmt.Errorf("api %s status %s body %s", path, response.Status, body)
	}
	var baseResponse struct {
		ErrCode int             `json:"errCode"`
		ErrMsg  string          `json:"errMsg"`
		ErrDlt  string          `json:"errDlt"`
		Data    json.RawMessage `json:"data"`
	}
	if err := json.Unmarshal(body, &baseResponse); err != nil {
		return err
	}
	if baseResponse.ErrCode != 0 {
		return fmt.Errorf("api %s errCode %d errMsg %s errDlt %s", path, baseResponse.ErrCode, baseResponse.ErrMsg, baseR
	}
	if resp != nil {
		if err := json.Unmarshal(baseResponse.Data, resp); err != nil {
			return err
		}
	}
	return nil
}

func (a *Api) GetAdminToken(ctx context.Context) (string, error) {
	req := auth.GetAdminTokenReq{
		UserID: a.UserID,
		Secret: a.Secret,
	}
	var resp auth.GetAdminTokenResp
	if err := a.apiPost(ctx, "/auth/get_admin_token", &req, &resp); err != nil {
		return "", err
	}
	return resp.Token, nil
}

func (a *Api) GetPartLimit(ctx context.Context) (*third.PartLimitResp, error) {
	var resp third.PartLimitResp
	if err := a.apiPost(ctx, "/object/part_limit", &third.PartLimitReq{}, &resp); err != nil {
		return nil, err
	}
	return &resp, nil
}

func (a *Api) InitiateMultipartUpload(ctx context.Context, req *third.InitiateMultipartUploadReq) (*third.InitiateMu
	var resp third.InitiateMultipartUploadResp
	if err := a.apiPost(ctx, "/object/initiate_multipart_upload", req, &resp); err != nil {
		return nil, err
	}
	return &resp, nil
}

func (a *Api) CompleteMultipartUpload(ctx context.Context, req *third.CompleteMultipartUploadReq) (string, error) {
	var resp third.CompleteMultipartUploadResp
	if err := a.apiPost(ctx, "/object/complete_multipart_upload", req, &resp); err != nil {
		return "", err
	}
	return resp.Url, nil
}
```

tools/url2im/pkg/buffer.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import (
	"bytes"
	"io"
	"os"
)

type ReadSeekSizeCloser interface {
	io.ReadSeekCloser
	Size() int64
}

func NewReader(r io.Reader, max int64, path string) (ReadSeekSizeCloser, error) {
	buf := make([]byte, max+1)
	n, err := io.ReadFull(r, buf)
	if err == nil {
		f, err := os.OpenFile(path, os.O_CREATE|os.O_RDWR, 0o666)
		if err != nil {
			return nil, err
		}
		var ok bool
		defer func() {
			if !ok {
				_ = f.Close()
				_ = os.Remove(path)
			}
		}()
		if _, err := f.Write(buf[:n]); err != nil {
			return nil, err
		}
		cn, err := io.Copy(f, r)
		if err != nil {
			return nil, err
		}
		if _, err := f.Seek(0, io.SeekStart); err != nil {
			return nil, err
		}
		ok = true
		return &fileBuffer{
			f: f,
			n: cn + int64(n),
		}, nil
	} else if err == io.EOF || err == io.ErrUnexpectedEOF {
		return &memoryBuffer{
			r: bytes.NewReader(buf[:n]),
		}, nil
	} else {
		return nil, err
	}
```

```
}

type fileBuffer struct {
	n int64
	f *os.File
}

func (r *fileBuffer) Read(p []byte) (n int, err error) {
	return r.f.Read(p)
}

func (r *fileBuffer) Seek(offset int64, whence int) (int64, error) {
	return r.f.Seek(offset, whence)
}

func (r *fileBuffer) Size() int64 {
	return r.n
}

func (r *fileBuffer) Close() error {
	name := r.f.Name()
	if err := r.f.Close(); err != nil {
		return err
	}
	return os.Remove(name)
}

type memoryBuffer struct {
	r *bytes.Reader
}

func (r *memoryBuffer) Read(p []byte) (n int, err error) {
	return r.r.Read(p)
}

func (r *memoryBuffer) Seek(offset int64, whence int) (int64, error) {
	return r.r.Seek(offset, whence)
}

func (r *memoryBuffer) Close() error {
	return nil
}

func (r *memoryBuffer) Size() int64 {
	return r.r.Size()
}
```

tools/url2im/pkg/config.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import "time"

type Config struct {
    TaskPath     string
    ProgressPath string
    Concurrency  int
    Retry        int
    Timeout      time.Duration
    Api          string
    UserID       string
    Secret       string
    TempDir      string
    CacheSize    int64
}
```

tools/url2im/pkg/http.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import "net/http"

func DefaultRequestHeader(header http.Header) {
    header.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/1
}
```

tools/url2im/pkg/manage.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import (
	"bufio"
	"context"
	"crypto/md5"
	"encoding/hex"
	"encoding/json"
	"fmt"
	"io"
	"log"
	"net/http"
	"net/url"
	"os"
	"path/filepath"
	"strconv"
	"strings"
	"sync"
	"sync/atomic"
	"time"

	"github.com/openimsdk/tools/errs"

	"github.com/openimsdk/protocol/third"
)

type Upload struct {
	URL         string `json:"url"`
	Name        string `json:"name"`
	ContentType string `json:"contentType"`
}

type Task struct {
	Index  int
	Upload Upload
}

type PartInfo struct {
	ContentType string
	PartSize    int64
	PartNum     int
	FileMd5     string
	PartMd5     string
	PartSizes   []int64
	PartMd5s    []string
}

func Run(conf Config) error {
	m := &Manage{
		prefix: time.Now().Format("20060102150405"),
```

```
    conf:   &conf,
    ctx:    context.Background(),
  }
  return m.Run()
}

type Manage struct {
  conf      *Config
  ctx       context.Context
  api       *Api
  partLimit *third.PartLimitResp
  prefix    string
  tasks     chan Task
  id        uint64
  success   int64
  failed    int64
}

func (m *Manage) tempFilePath() string {
  return filepath.Join(m.conf.TempDir, fmt.Sprintf("%s_%d", m.prefix, atomic.AddUint64(&m.id, 1)))
}

func (m *Manage) Run() error {
  defer func(start time.Time) {
    log.Printf("run time %s\n", time.Since(start))
  }(time.Now())
  m.api = &Api{
    Api:    m.conf.Api,
    UserID: m.conf.UserID,
    Secret: m.conf.Secret,
    Client: &http.Client{Timeout: m.conf.Timeout},
  }
  var err error
  ctx := context.WithValue(m.ctx, "operationID", fmt.Sprintf("%s_init", m.prefix))
  m.api.Token, err = m.api.GetAdminToken(ctx)
  if err != nil {
    return err
  }
  m.partLimit, err = m.api.GetPartLimit(ctx)
  if err != nil {
    return err
  }
  progress, err := ReadProgress(m.conf.ProgressPath)
  if err != nil {
    return err
  }
  progressFile, err := os.OpenFile(m.conf.ProgressPath, os.O_CREATE|os.O_RDWR|os.O_APPEND, 0666)
  if err != nil {
    return err
  }
  var mutex sync.Mutex
  writeSuccessIndex := func(index int) {
    mutex.Lock()
    defer mutex.Unlock()
    if _, err := progressFile.Write([]byte(strconv.Itoa(index) + "\n")); err != nil {
      log.Printf("write progress err: %v\n", err)
    }
  }
  file, err := os.Open(m.conf.TaskPath)
  if err != nil {
    return err
  }
  m.tasks = make(chan Task, m.conf.Concurrency*2)
  go func() {
    defer file.Close()
    defer close(m.tasks)
```

```
  scanner := bufio.NewScanner(file)
  var (
    index int
    num   int
  )
  for scanner.Scan() {
    line := strings.TrimSpace(scanner.Text())
    if line == "" {
      continue
    }
    index++
    if progress.IsUploaded(index) {
      log.Printf("index: %d already uploaded %s\n", index, line)
      continue
    }
    var upload Upload
    if err := json.Unmarshal([]byte(line), &upload); err != nil {
      log.Printf("index: %d json.Unmarshal(%s) err: %v", index, line, err)
      continue
    }
    num++
    m.tasks <- Task{
      Index:  index,
      Upload: upload,
    }
  }
  if num == 0 {
    log.Println("mark all completed")
  }
 }()
 var wg sync.WaitGroup
 wg.Add(m.conf.Concurrency)
 for i := 0; i < m.conf.Concurrency; i++ {
  go func(tid int) {
    defer wg.Done()
    for task := range m.tasks {
      var success bool
      for n := 0; n < m.conf.Retry; n++ {
        ctx := context.WithValue(m.ctx, "operationID", fmt.Sprintf("%s_%d_%d_%d", m.prefix, tid, task.Index, n+1))
        if urlRaw, err := m.RunTask(ctx, task); err == nil {
          writeSuccessIndex(task.Index)
          log.Println("index:", task.Index, "upload success", "urlRaw", urlRaw)
          success = true
          break
        } else {
          log.Printf("index: %d upload: %+v err: %v", task.Index, task.Upload, err)
        }
      }
      if success {
        atomic.AddInt64(&m.success, 1)
      } else {
        atomic.AddInt64(&m.failed, 1)
        log.Printf("index: %d upload: %+v failed", task.Index, task.Upload)
      }
    }
  }(i + 1)
 }
 wg.Wait()
 log.Printf("execution completed success %d failed %d\n", m.success, m.failed)
 return nil
}

func (m *Manage) RunTask(ctx context.Context, task Task) (string, error) {
 resp, err := m.HttpGet(ctx, task.Upload.URL)
 if err != nil {
  return "", err
```

```
	}
	defer resp.Body.Close()
	reader, err := NewReader(resp.Body, m.conf.CacheSize, m.tempFilePath())
	if err != nil {
		return "", err
	}
	defer reader.Close()
	part, err := m.getPartInfo(ctx, reader, reader.Size())
	if err != nil {
		return "", err
	}
	var contentType string
	if task.Upload.ContentType == "" {
		contentType = part.ContentType
	} else {
		contentType = task.Upload.ContentType
	}
	initiateMultipartUploadResp, err := m.api.InitiateMultipartUpload(ctx, &third.InitiateMultipartUploadReq{
		Hash:        part.PartMd5,
		Size:        reader.Size(),
		PartSize:    part.PartSize,
		MaxParts:    -1,
		Cause:       "batch-import",
		Name:        task.Upload.Name,
		ContentType: contentType,
	})
	if err != nil {
		return "", err
	}
	if initiateMultipartUploadResp.Upload == nil {
		return initiateMultipartUploadResp.Url, nil
	}
	if _, err := reader.Seek(0, io.SeekStart); err != nil {
		return "", err
	}
	uploadParts := make([]*third.SignPart, part.PartNum)
	for _, part := range initiateMultipartUploadResp.Upload.Sign.Parts {
		uploadParts[part.PartNumber-1] = part
	}
	for i, currentPartSize := range part.PartSizes {
		md5Reader := NewMd5Reader(io.LimitReader(reader, currentPartSize))
		if err := m.doPut(ctx, m.api.Client, initiateMultipartUploadResp.Upload.Sign, uploadParts[i], md5Reader, currentPa
			return "", err
		}
		if md5val := md5Reader.Md5(); md5val != part.PartMd5s[i] {
			return "", fmt.Errorf("upload part %d failed, md5 not match, expect %s, got %s", i, part.PartMd5s[i], md5val)
		}
	}
	urlRaw, err := m.api.CompleteMultipartUpload(ctx, &third.CompleteMultipartUploadReq{
		UploadID:    initiateMultipartUploadResp.Upload.UploadID,
		Parts:       part.PartMd5s,
		Name:        task.Upload.Name,
		ContentType: contentType,
		Cause:       "batch-import",
	})
	if err != nil {
		return "", err
	}
	return urlRaw, nil
}

func (m *Manage) partSize(size int64) (int64, error) {
	if size <= 0 {
		return 0, errs.New("size must be greater than 0")
	}
	if size > m.partLimit.MaxPartSize*int64(m.partLimit.MaxNumSize) {
```

```
    return 0, errs.New("size must be less than", "size", m.partLimit.MaxPartSize*int64(m.partLimit.MaxNumSize))
  }
  if size <= m.partLimit.MinPartSize*int64(m.partLimit.MaxNumSize) {
    return m.partLimit.MinPartSize, nil
  }
  partSize := size / int64(m.partLimit.MaxNumSize)
  if size%int64(m.partLimit.MaxNumSize) != 0 {
    partSize++
  }
  return partSize, nil
}

func (m *Manage) partMD5(parts []string) string {
  s := strings.Join(parts, ",")
  md5Sum := md5.Sum([]byte(s))
  return hex.EncodeToString(md5Sum[:])
}

func (m *Manage) getPartInfo(ctx context.Context, r io.Reader, fileSize int64) (*PartInfo, error) {
  partSize, err := m.partSize(fileSize)
  if err != nil {
    return nil, err
  }
  partNum := int(fileSize / partSize)
  if fileSize%partSize != 0 {
    partNum++
  }
  partSizes := make([]int64, partNum)
  for i := 0; i < partNum; i++ {
    partSizes[i] = partSize
  }
  partSizes[partNum-1] = fileSize - partSize*(int64(partNum)-1)
  partMd5s := make([]string, partNum)
  buf := make([]byte, 1024*8)
  fileMd5 := md5.New()
  var contentType string
  for i := 0; i < partNum; i++ {
    h := md5.New()
    r := io.LimitReader(r, partSize)
    for {
      if n, err := r.Read(buf); err == nil {
        if contentType == "" {
          contentType = http.DetectContentType(buf[:n])
        }
        h.Write(buf[:n])
        fileMd5.Write(buf[:n])
      } else if err == io.EOF {
        break
      } else {
        return nil, err
      }
    }
    partMd5s[i] = hex.EncodeToString(h.Sum(nil))
  }
  partMd5Val := m.partMD5(partMd5s)
  fileMd5val := hex.EncodeToString(fileMd5.Sum(nil))
  return &PartInfo{
    ContentType: contentType,
    PartSize:    partSize,
    PartNum:     partNum,
    FileMd5:     fileMd5val,
    PartMd5:     partMd5Val,
    PartSizes:   partSizes,
    PartMd5s:    partMd5s,
  }, nil
}
```

```
func (m *Manage) doPut(ctx context.Context, client *http.Client, sign *third.AuthSignParts, part *third.SignPart, re
    rawURL := part.Url
    if rawURL == "" {
        rawURL = sign.Url
    }
    if len(sign.Query)+len(part.Query) > 0 {
        u, err := url.Parse(rawURL)
        if err != nil {
            return err
        }
        query := u.Query()
        for i := range sign.Query {
            v := sign.Query[i]
            query[v.Key] = v.Values
        }
        for i := range part.Query {
            v := part.Query[i]
            query[v.Key] = v.Values
        }
        u.RawQuery = query.Encode()
        rawURL = u.String()
    }
    req, err := http.NewRequestWithContext(ctx, http.MethodPut, rawURL, reader)
    if err != nil {
        return err
    }
    for i := range sign.Header {
        v := sign.Header[i]
        req.Header[v.Key] = v.Values
    }
    for i := range part.Header {
        v := part.Header[i]
        req.Header[v.Key] = v.Values
    }
    req.ContentLength = size
    resp, err := client.Do(req)
    if err != nil {
        return err
    }
    defer func() {
        _ = resp.Body.Close()
    }()
    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return err
    }
    if resp.StatusCode/200 != 1 {
        return fmt.Errorf("PUT %s part %d failed, status code %d, body %s", rawURL, part.PartNumber, resp.StatusCode, str
    }
    return nil
}

func (m *Manage) HttpGet(ctx context.Context, url string) (*http.Response, error) {
    reqUrl := url
    for {
        request, err := http.NewRequestWithContext(ctx, http.MethodGet, reqUrl, nil)
        if err != nil {
            return nil, err
        }
        DefaultRequestHeader(request.Header)
        response, err := m.api.Client.Do(request)
        if err != nil {
            return nil, err
        }
        if response.StatusCode != http.StatusOK {
```

```
    _ = response.Body.Close()
    return nil, fmt.Errorf("webhook get %s status %s", url, response.Status)
  }
  return response, nil
 }
}
```

tools/url2im/pkg/md5.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import (
	"crypto/md5"
	"encoding/hex"
	"hash"
	"io"
)

func NewMd5Reader(r io.Reader) *Md5Reader {
	return &Md5Reader{h: md5.New(), r: r}
}

type Md5Reader struct {
	h hash.Hash
	r io.Reader
}

func (r *Md5Reader) Read(p []byte) (n int, err error) {
	n, err = r.r.Read(p)
	if err == nil && n > 0 {
		r.h.Write(p[:n])
	}
	return
}

func (r *Md5Reader) Md5() string {
	return hex.EncodeToString(r.h.Sum(nil))
}
```

tools/url2im/pkg/progress.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package pkg

import (
	"bufio"
	"os"
	"strconv"

	"github.com/kelindar/bitmap"
)

func ReadProgress(path string) (*Progress, error) {
	file, err := os.Open(path)
	if err != nil {
		if os.IsNotExist(err) {
			return &Progress{}, nil
		}
		return nil, err
	}
	defer file.Close()
	scanner := bufio.NewScanner(file)
	var upload bitmap.Bitmap
	for scanner.Scan() {
		index, err := strconv.Atoi(scanner.Text())
		if err != nil || index < 0 {
			continue
		}
		upload.Set(uint32(index))
	}
	return &Progress{upload: upload}, nil
}

type Progress struct {
	upload bitmap.Bitmap
}

func (p *Progress) IsUploaded(index int) bool {
	if p == nil {
		return false
	}
	return p.upload.Contains(uint32(index))
}
```

tools/s3

tools/s3/main.go

```
package main

import (
	"flag"
	"fmt"
	"github.com/openimsdk/open-im-server/v3/tools/s3/internal"
	"os"
)

func main() {
	var (
		name   string
		config string
	)
	flag.StringVar(&name, "name", "", "old previous storage name")
	flag.StringVar(&config, "config", "", "config directory")
	flag.Parse()
	if err := internal.Main(config, name); err != nil {
		fmt.Fprintln(os.Stderr, err)
		os.Exit(1)
	}
	fmt.Fprintln(os.Stdout, "success")
}
```

tools/s3/internal

tools/s3/internal/conversion.go

```
package internal

import (
	"context"
	"errors"
	"fmt"
	"log"
	"net/http"
	"path/filepath"
	"time"

	"github.com/mitchellh/mapstructure"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/cache/redis"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database"
	"github.com/openimsdk/open-im-server/v3/pkg/common/storage/database/mgo"
	"github.com/openimsdk/tools/db/mongoutil"
	"github.com/openimsdk/tools/db/redisutil"
	"github.com/openimsdk/tools/s3"
	"github.com/openimsdk/tools/s3/aws"
	"github.com/openimsdk/tools/s3/cos"
	"github.com/openimsdk/tools/s3/kodo"
	"github.com/openimsdk/tools/s3/minio"
	"github.com/openimsdk/tools/s3/oss"
	"github.com/spf13/viper"
	"go.mongodb.org/mongo-driver/mongo"
)

const defaultTimeout = time.Second * 10

func readConf(path string, val any) error {
	v := viper.New()
	v.SetConfigFile(path)
	if err := v.ReadInConfig(); err != nil {
		return err
	}
	fn := func(config *mapstructure.DecoderConfig) {
		config.TagName = "mapstructure"
	}
	return v.Unmarshal(val, fn)
}

func getS3(path string, name string, thirdConf *config.Third) (s3.Interface, error) {
	switch name {
	case "minio":
		ctx, cancel := context.WithTimeout(context.Background(), defaultTimeout)
		defer cancel()
		var minioConf config.Minio
		if err := readConf(filepath.Join(path, minioConf.GetConfigFileName()), &minioConf); err != nil {
			return nil, err
		}
		var redisConf config.Redis
		if err := readConf(filepath.Join(path, redisConf.GetConfigFileName()), &redisConf); err != nil {
			return nil, err
		}
		rdb, err := redisutil.NewRedisClient(ctx, redisConf.Build())
		if err != nil {
			return nil, err
		}
		return minio.NewMinio(ctx, redis.NewMinioCache(rdb), *minioConf.Build())
	case "cos":
```

```
    return cos.NewCos(*thirdConf.Object.Cos.Build())
  case "oss":
    return oss.NewOSS(*thirdConf.Object.Oss.Build())
  case "kodo":
    return kodo.NewKodo(*thirdConf.Object.Kodo.Build())
  case "aws":
    return aws.NewAws(*thirdConf.Object.Aws.Build())
  default:
    return nil, fmt.Errorf("invalid object enable: %s", name)
  }
}

func getMongo(path string) (database.ObjectInfo, error) {
  var mongoConf config.Mongo
  if err := readConf(filepath.Join(path, mongoConf.GetConfigFileName()), &mongoConf); err != nil {
    return nil, err
  }
  ctx, cancel := context.WithTimeout(context.Background(), defaultTimeout)
  defer cancel()
  mgocli, err := mongoutil.NewMongoDB(ctx, mongoConf.Build())
  if err != nil {
    return nil, err
  }
  return mgo.NewS3Mongo(mgocli.GetDB())
}

func Main(path string, engine string) error {
  var thirdConf config.Third
  if err := readConf(filepath.Join(path, thirdConf.GetConfigFileName()), &thirdConf); err != nil {
    return err
  }
  if thirdConf.Object.Enable == engine {
    return errors.New("same s3 storage")
  }
  s3db, err := getMongo(path)
  if err != nil {
    return err
  }
  oldS3, err := getS3(path, engine, &thirdConf)
  if err != nil {
    return err
  }
  newS3, err := getS3(path, thirdConf.Object.Enable, &thirdConf)
  if err != nil {
    return err
  }
  count, err := getEngineCount(s3db, oldS3.Engine())
  if err != nil {
    return err
  }
  log.Printf("engine %s count: %d", oldS3.Engine(), count)
  var skip int
  for i := 1; i <= count+1; i++ {
    log.Printf("start %d/%d", i, count)
    start := time.Now()
    res, err := doObject(s3db, newS3, oldS3, skip)
    if err != nil {
      log.Printf("end [%s] %d/%d error %s", time.Since(start), i, count, err)
      return err
    }
    log.Printf("end [%s] %d/%d result %+v", time.Since(start), i, count, *res)
    if res.Skip {
      skip++
    }
    if res.End {
      break
```

```
		}
	}
	return nil
}

func getEngineCount(db database.ObjectInfo, name string) (int, error) {
	ctx, cancel := context.WithTimeout(context.Background(), defaultTimeout)
	defer cancel()
	count, err := db.GetEngineCount(ctx, name)
	if err != nil {
		return 0, err
	}
	return int(count), nil
}

func doObject(db database.ObjectInfo, newS3, oldS3 s3.Interface, skip int) (*Result, error) {
	ctx, cancel := context.WithTimeout(context.Background(), defaultTimeout)
	defer cancel()
	infos, err := db.GetEngineInfo(ctx, oldS3.Engine(), 1, skip)
	if err != nil {
		return nil, err
	}
	if len(infos) == 0 {
		return &Result{End: true}, nil
	}
	obj := infos[0]
	if _, err := db.Take(ctx, newS3.Engine(), obj.Name); err == nil {
		return &Result{Skip: true}, nil
	} else if !errors.Is(err, mongo.ErrNoDocuments) {
		return nil, err
	}
	downloadURL, err := oldS3.AccessURL(ctx, obj.Key, time.Hour, &s3.AccessURLOption{})
	if err != nil {
		return nil, err
	}
	putURL, err := newS3.PresignedPutObject(ctx, obj.Key, time.Hour, &s3.PutOption{ContentType: obj.ContentType})
	if err != nil {
		return nil, err
	}
	downloadResp, err := http.Get(downloadURL)
	if err != nil {
		return nil, err
	}
	defer downloadResp.Body.Close()
	switch downloadResp.StatusCode {
	case http.StatusNotFound:
		return &Result{Skip: true}, nil
	case http.StatusOK:
	default:
		return nil, fmt.Errorf("download object failed %s", downloadResp.Status)
	}
	log.Printf("file size %d", obj.Size)
	request, err := http.NewRequest(http.MethodPut, putURL.URL, downloadResp.Body)
	if err != nil {
		return nil, err
	}
	putResp, err := http.DefaultClient.Do(request)
	if err != nil {
		return nil, err
	}
	defer putResp.Body.Close()
	if putResp.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("put object failed %s", putResp.Status)
	}
	ctx, cancel = context.WithTimeout(context.Background(), defaultTimeout)
	defer cancel()
```

```
	if err := db.UpdateEngine(ctx, obj.Engine, obj.Name, newS3.Engine()); err != nil {
		return nil, err
	}
	return &Result{}, nil
}

type Result struct {
	Skip bool
	End  bool
}
```

tools/infra

tools/infra/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"fmt"

	"github.com/fatih/color"
)

// Define a function to print important link information
func printLinks() {
	blue := color.New(color.FgBlue).SprintFunc()
	fmt.Printf("OpenIM Github: %s\n", blue("https://github.com/OpenIMSDK/Open-IM-Server"))
	fmt.Printf("Slack Invitation: %s\n", blue("https://openimsdk.slack.com"))
	fmt.Printf("Follow Twitter: %s\n", blue("https://twitter.com/founder_im63606"))
}

func main() {
	yellow := color.New(color.FgYellow)
	blue := color.New(color.FgBlue, color.Bold)

	yellow.Println("Please use the release branch or tag for production environments!")

	message := `
____                    _____  __  __
/ __ \                  |_   _||  \/  |
| |  | | _ __    ___  _ __    | |  | \  / |
| |  | || '_ \  / _ \| '_ \   | |  | |\/| |
| |__| || |_) ||  __/| | | | _| |_ | |  | |
\____/ | .__/  \___||_| |_||_____||_|  |_|
       | |
       |_|

Keep checking for updates!
`

	blue.Println(message)
	printLinks() // Call the function to print the link information
}
```

tools/check-free-memory

tools/check-free-memory/main.go

```
package main

import (
	"fmt"
	"os"

	"github.com/shirou/gopsutil/mem"
)

func main() {
	vMem, err := mem.VirtualMemory()
	if err != nil {
		fmt.Fprintf(os.Stderr, "Failed to get virtual memory info: %v\n", err)
		os.Exit(1)
	}

	// Use the Available field to get the available memory
	availableMemoryGB := float64(vMem.Available) / float64(1024*1024*1024)

	if availableMemoryGB < 1.0 {
		fmt.Fprintf(os.Stderr, "System available memory is less than 1GB: %.2fGB\n", availableMemoryGB)
		os.Exit(1)
	} else {
		fmt.Printf("System available memory is sufficient: %.2fGB\n", availableMemoryGB)
	}
}
```

cmd

cmd/main.go

```
package main

import (
	"bytes"
	"context"
	"flag"
	"fmt"
	"net"
	"os"
	"os/signal"
	"path"
	"path/filepath"
	"reflect"
	"runtime"
	"strings"
	"sync"
	"syscall"
	"time"

	"github.com/mitchellh/mapstructure"
	"github.com/openimsdk/open-im-server/v3/internal/api"
	"github.com/openimsdk/open-im-server/v3/internal/msggateway"
	"github.com/openimsdk/open-im-server/v3/internal/msgtransfer"
	"github.com/openimsdk/open-im-server/v3/internal/push"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/auth"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/conversation"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/group"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/msg"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/relation"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/third"
	"github.com/openimsdk/open-im-server/v3/internal/rpc/user"
	"github.com/openimsdk/open-im-server/v3/internal/tools/cron"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/open-im-server/v3/pkg/common/prommetrics"
	"github.com/openimsdk/open-im-server/v3/version"
	"github.com/openimsdk/tools/discovery"
	"github.com/openimsdk/tools/discovery/standalone"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/system/program"
	"github.com/openimsdk/tools/utils/datautil"
	"github.com/spf13/viper"
	"google.golang.org/grpc"
)

func init() {
	config.SetStandalone()
	prommetrics.RegistryAll()
}

func main() {
	var configPath string
	flag.StringVar(&configPath, "c", "", "config path")
	flag.Parse()
	if configPath == "" {
		_, _ = fmt.Fprintln(os.Stderr, "config path is empty")
		os.Exit(1)
		return
	}
	cmd := newCmds(configPath)
	putCmd(cmd, false, auth.Start)
```

```
	putCmd(cmd, false, conversation.Start)
	putCmd(cmd, false, relation.Start)
	putCmd(cmd, false, group.Start)
	putCmd(cmd, false, msg.Start)
	putCmd(cmd, false, third.Start)
	putCmd(cmd, false, user.Start)
	putCmd(cmd, false, push.Start)
	putCmd(cmd, true, msggateway.Start)
	putCmd(cmd, true, msgtransfer.Start)
	putCmd(cmd, true, api.Start)
	putCmd(cmd, true, cron.Start)
	ctx := context.Background()
	if err := cmd.run(ctx); err != nil {
		_, _ = fmt.Fprintf(os.Stderr, "server exit %s", err)
		os.Exit(1)
		return
	}
}

func newCmds(confPath string) *cmds {
	return &cmds{confPath: confPath}
}

type cmdName struct {
	Name  string
	Func  func(ctx context.Context) error
	Block bool
}
type cmds struct {
	confPath string
	cmds     []cmdName
	config   config.AllConfig
	conf     map[string]reflect.Value
}

func (x *cmds) getTypePath(typ reflect.Type) string {
	return path.Join(typ.PkgPath(), typ.Name())
}

func (x *cmds) initDiscovery() {
	x.config.Discovery.Enable = "standalone"
	vof := reflect.ValueOf(&x.config.Discovery.RpcService).Elem()
	tof := reflect.TypeOf(&x.config.Discovery.RpcService).Elem()
	num := tof.NumField()
	for i := 0; i < num; i++ {
		field := tof.Field(i)
		if !field.IsExported() {
			continue
		}
		if field.Type.Kind() != reflect.String {
			continue
		}
		vof.Field(i).SetString(field.Name)
	}
}

func (x *cmds) initAllConfig() error {
	x.conf = make(map[string]reflect.Value)
	vof := reflect.ValueOf(&x.config).Elem()
	num := vof.NumField()
	for i := 0; i < num; i++ {
		field := vof.Field(i)
		for ptr := true; ptr; {
			if field.Kind() == reflect.Ptr {
				field = field.Elem()
			} else {
```

```
        ptr = false
    }
  }
  x.conf[x.getTypePath(field.Type())] = field
  val := field.Addr().Interface()
  name := val.(interface{ GetConfigFileName() string }).GetConfigFileName()
  confData, err := os.ReadFile(filepath.Join(x.confPath, name))
  if err != nil {
    if os.IsNotExist(err) {
      continue
    }
    return err
  }
  v := viper.New()
  v.SetConfigType("yaml")
  if err := v.ReadConfig(bytes.NewReader(confData)); err != nil {
    return err
  }
  opt := func(conf *mapstructure.DecoderConfig) {
    conf.TagName = config.StructTagName
  }
  if err := v.Unmarshal(val, opt); err != nil {
    return err
  }
 }
 x.initDiscovery()
 x.config.Redis.Disable = false
 x.config.LocalCache = config.LocalCache{}
 config.InitNotification(&x.config.Notification)
 return nil
}

func (x *cmds) parseConf(conf any) error {
 vof := reflect.ValueOf(conf)
 for {
  if vof.Kind() == reflect.Ptr {
    vof = vof.Elem()
  } else {
    break
  }
 }
 tof := vof.Type()
 numField := vof.NumField()
 for i := 0; i < numField; i++ {
  typeField := tof.Field(i)
  if !typeField.IsExported() {
    continue
  }
  field := vof.Field(i)
  pkt := x.getTypePath(field.Type())
  val, ok := x.conf[pkt]
  if !ok {
    switch field.Interface().(type) {
    case config.Index:
    case config.Path:
      field.SetString(x.confPath)
    case config.AllConfig:
      field.Set(reflect.ValueOf(x.config))
    case *config.AllConfig:
      field.Set(reflect.ValueOf(&x.config))
    default:
      return fmt.Errorf("config field %s %s not found", vof.Type().Name(), typeField.Name)
    }
    continue
  }
  field.Set(val)
```

```
	}
	return nil
}

func (x *cmds) add(name string, block bool, fn func(ctx context.Context) error) {
	x.cmds = append(x.cmds, cmdName{Name: name, Block: block, Func: fn})
}

func (x *cmds) initLog() error {
	conf := x.config.Log
	if err := log.InitLoggerFromConfig(
		"openim-server",
		program.GetProcessName(),
		"", "",
		conf.RemainLogLevel,
		conf.IsStdout,
		conf.IsJson,
		conf.StorageLocation,
		conf.RemainRotationCount,
		conf.RotationTime,
		strings.TrimSpace(version.Version),
		conf.IsSimplify,
	); err != nil {
		return err
	}
	return nil

}

func (x *cmds) run(ctx context.Context) error {
	if len(x.cmds) == 0 {
		return fmt.Errorf("no command to run")
	}
	if err := x.initAllConfig(); err != nil {
		return err
	}
	if err := x.initLog(); err != nil {
		return err
	}

	ctx, cancel := context.WithCancelCause(ctx)

	go func() {
		<-ctx.Done()
		log.ZError(ctx, "context server exit cause", context.Cause(ctx))
	}()

	if prometheus := x.config.API.Prometheus; prometheus.Enable {
		var (
			port int
			err  error
		)
		if !prometheus.AutoSetPorts {
			port, err = datautil.GetElemByIndex(prometheus.Ports, 0)
			if err != nil {
				return err
			}
		}
		listener, err := net.Listen("tcp", fmt.Sprintf(":%d", port))
		if err != nil {
			return fmt.Errorf("prometheus listen %d error %w", port, err)
		}
		defer listener.Close()
		log.ZDebug(ctx, "prometheus start", "addr", listener.Addr())
		go func() {
			err := prommetrics.Start(listener)
```

```
        if err == nil {
            err = fmt.Errorf("http done")
        }
        cancel(fmt.Errorf("prometheus %w", err))
    }()
}

go func() {
    sigs := make(chan os.Signal, 1)
    signal.Notify(sigs, syscall.SIGTERM, syscall.SIGINT, syscall.SIGKILL)
    select {
    case <-ctx.Done():
        return
    case val := <-sigs:
        log.ZDebug(ctx, "recv signal", "signal", val.String())
        cancel(fmt.Errorf("signal %s", val.String()))
    }
}()

for i := range x.cmds {
    cmd := x.cmds[i]
    if cmd.Block {
        continue
    }
    if err := cmd.Func(ctx); err != nil {
        cancel(fmt.Errorf("server %s exit %w", cmd.Name, err))
        return err
    }
    go func() {
        if cmd.Block {
            cancel(fmt.Errorf("server %s exit", cmd.Name))
        }
    }()
}

var wait cmdManger
for i := range x.cmds {
    cmd := x.cmds[i]
    if !cmd.Block {
        continue
    }
    wait.Start(cmd.Name)
    go func() {
        defer wait.Shutdown(cmd.Name)
        if err := cmd.Func(ctx); err != nil {
            cancel(fmt.Errorf("server %s exit %w", cmd.Name, err))
            return
        }
        cancel(fmt.Errorf("server %s exit", cmd.Name))
    }()
}
<-ctx.Done()
exitCause := context.Cause(ctx)
log.ZWarn(ctx, "notification of service closure", exitCause)
done := wait.Wait()
timeout := time.NewTimer(time.Second * 10)
defer timeout.Stop()
for {
    select {
    case <-timeout.C:
        log.ZWarn(ctx, "server exit timeout", nil, "running", wait.Running())
        return exitCause
    case _, ok := <-done:
        if ok {
            log.ZWarn(ctx, "waiting for the service to exit", nil, "running", wait.Running())
        } else {
```

```
        log.ZInfo(ctx, "all server exit done")
        return exitCause
    }
  }
 }
}

func putCmd[C any](cmd *cmds, block bool, fn func(ctx context.Context, config *C, client discovery.SvcDiscoveryRegis
 name := path.Base(runtime.FuncForPC(reflect.ValueOf(fn).Pointer()).Name())
 if index := strings.Index(name, "."); index >= 0 {
  name = name[:index]
 }
 cmd.add(name, block, func(ctx context.Context) error {
  var conf C
  if err := cmd.parseConf(&conf); err != nil {
   return err
  }
  return fn(ctx, &conf, standalone.GetSvcDiscoveryRegistry(), standalone.GetServiceRegistrar())
 })
}

type cmdManger struct {
 lock  sync.Mutex
 done  chan struct{}
 count int
 names map[string]struct{}
}

func (x *cmdManger) Start(name string) {
 x.lock.Lock()
 defer x.lock.Unlock()
 if x.names == nil {
  x.names = make(map[string]struct{})
 }
 if x.done == nil {
  x.done = make(chan struct{}, 1)
 }
 if _, ok := x.names[name]; ok {
  panic(fmt.Errorf("cmd %s already exists", name))
 }
 x.count++
 x.names[name] = struct{}{}
}

func (x *cmdManger) Shutdown(name string) {
 x.lock.Lock()
 defer x.lock.Unlock()
 if _, ok := x.names[name]; !ok {
  panic(fmt.Errorf("cmd %s not exists", name))
 }
 delete(x.names, name)
 x.count--
 if x.count == 0 {
  close(x.done)
 } else {
  select {
  case x.done <- struct{}{}:
  default:
  }
 }
}

func (x *cmdManger) Wait() <-chan struct{} {
 x.lock.Lock()
 defer x.lock.Unlock()
 if x.count == 0 || x.done == nil {
```

```
		tmp := make(chan struct{})
		close(tmp)
		return tmp
	}
	return x.done
}

func (x *cmdManger) Running() []string {
	x.lock.Lock()
	defer x.lock.Unlock()
	names := make([]string, 0, len(x.names))
	for name := range x.names {
		names = append(names, name)
	}
	return names
}
```

cmd/openim-msgtransfer

cmd/openim-msgtransfer/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	if err := cmd.NewMsgTransferCmd().Exec(); err != nil {
		program.ExitWithError(err)
	}
}
```

cmd/openim-cmdutils

cmd/openim-cmdutils/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	msgUtilsCmd := cmd.NewMsgUtilsCmd("openIMCmdUtils", "openIM cmd utils", nil)
	getCmd := cmd.NewGetCmd()
	fixCmd := cmd.NewFixCmd()
	clearCmd := cmd.NewClearCmd()
	seqCmd := cmd.NewSeqCmd()
	msgCmd := cmd.NewMsgCmd()
	getCmd.AddCommand(seqCmd.GetSeqCmd(), msgCmd.GetMsgCmd())
	getCmd.AddSuperGroupIDFlag()
	getCmd.AddUserIDFlag()
	getCmd.AddConfigDirFlag()
	getCmd.AddIndexFlag()
	getCmd.AddBeginSeqFlag()
	getCmd.AddLimitFlag()
	// openIM get seq --userID=xxx
	// openIM get seq --superGroupID=xxx
	// openIM get msg --userID=xxx --beginSeq=100 --limit=10
	// openIM get msg --superGroupID=xxx --beginSeq=100 --limit=10

	fixCmd.AddCommand(seqCmd.FixSeqCmd())
	fixCmd.AddSuperGroupIDFlag()
	fixCmd.AddUserIDFlag()
	fixCmd.AddConfigDirFlag()
	fixCmd.AddIndexFlag()
	fixCmd.AddFixAllFlag()
	// openIM fix seq --userID=xxx
	// openIM fix seq --superGroupID=xxx
	// openIM fix seq --fixAll

	clearCmd.AddCommand(msgCmd.ClearMsgCmd())
	clearCmd.AddSuperGroupIDFlag()
	clearCmd.AddUserIDFlag()
	clearCmd.AddConfigDirFlag()
	clearCmd.AddIndexFlag()
	clearCmd.AddClearAllFlag()
	clearCmd.AddBeginSeqFlag()
	clearCmd.AddLimitFlag()
	// openIM clear msg --userID=xxx --beginSeq=100 --limit=10
	// openIM clear msg --superGroupID=xxx --beginSeq=100 --limit=10
	// openIM clear msg --clearAll
```

```
    msgUtilsCmd.AddCommand(&getCmd.Command, &fixCmd.Command, &clearCmd.Command)
    if err := msgUtilsCmd.Execute(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-msggateway

cmd/openim-msggateway/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
    "github.com/openimsdk/tools/system/program"
)

func main() {
    if err := cmd.NewMsgGatewayCmd().Exec(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-push

cmd/openim-push/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
    "github.com/openimsdk/tools/system/program"
)

func main() {
    if err := cmd.NewPushRpcCmd().Exec(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-api

cmd/openim-api/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
    "github.com/openimsdk/tools/system/program"
    _ "net/http/pprof"
)

func main() {
    if err := cmd.NewApiCmd().Exec(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-rpc

cmd/openim-rpc/openim-rpc-auth

cmd/openim-rpc/openim-rpc-auth/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	if err := cmd.NewAuthRpcCmd().Exec(); err != nil {
		program.ExitWithError(err)
	}
}
```

cmd/openim-rpc/openim-rpc-third

cmd/openim-rpc/openim-rpc-third/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	if err := cmd.NewThirdRpcCmd().Exec(); err != nil {
		program.ExitWithError(err)
	}
}
```

cmd/openim-rpc/openim-rpc-group

cmd/openim-rpc/openim-rpc-group/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	if err := cmd.NewGroupRpcCmd().Exec(); err != nil {
		program.ExitWithError(err)
	}
}
```

cmd/openim-rpc/openim-rpc-user

cmd/openim-rpc/openim-rpc-user/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	if err := cmd.NewUserRpcCmd().Exec(); err != nil {
		program.ExitWithError(err)
	}
}
```

cmd/openim-rpc/openim-rpc-conversation

cmd/openim-rpc/openim-rpc-conversation/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
    "github.com/openimsdk/tools/system/program"
)

func main() {
    if err := cmd.NewConversationRpcCmd().Exec(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-rpc/openim-rpc-msg

cmd/openim-rpc/openim-rpc-msg/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
    "github.com/openimsdk/tools/system/program"
)

func main() {
    if err := cmd.NewMsgRpcCmd().Exec(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-rpc/openim-rpc-friend

cmd/openim-rpc/openim-rpc-friend/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
    "github.com/openimsdk/tools/system/program"
)

func main() {
    if err := cmd.NewFriendRpcCmd().Exec(); err != nil {
        program.ExitWithError(err)
    }
}
```

cmd/openim-crontask

cmd/openim-crontask/main.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"github.com/openimsdk/open-im-server/v3/pkg/common/cmd"
	"github.com/openimsdk/tools/system/program"
)

func main() {
	if err := cmd.NewCronTaskCmd().Exec(); err != nil {
		program.ExitWithError(err)
	}
}
```

test

test/stress-test

test/stress-test/main.go

```go
package main

import (
	"bytes"
	"context"
	"encoding/json"
	"flag"
	"fmt"
	"io"
	"net/http"
	"os"
	"os/signal"
	"sync"
	"syscall"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/relation"
	"github.com/openimsdk/protocol/sdkws"
	pbuser "github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/system/program"
)

/*
 1. Create one user every minute
 2. Import target users as friends
 3. Add users to the default group
 4. Send a message to the default group every second, containing index and current timestamp
 5. Create a new group every minute and invite target users to join
*/

// !!! ATTENTION: This variable is must be added!
var (
	//  Use default userIDs List for testing, need to be created.
	TestTargetUserList = []string{
		"<need-update-it>",
	}
	DefaultGroupID = "<need-update-it>" // Use default group ID for testing, need to be created.
)

var (
	ApiAddress string

	// API method
	GetAdminToken = "/auth/get_admin_token"
	CreateUser    = "/user/user_register"
	ImportFriend  = "/friend/import_friend"
	InviteToGroup = "/group/invite_user_to_group"
	SendMsg       = "/msg/send_msg"
	CreateGroup   = "/group/create_group"
	GetUserToken  = "/auth/user_token"
)
```

```
const (
	MaxUser  = 10000
	MaxGroup = 1000

	CreateUserTicker  = 1 * time.Minute // Ticker is 1min in create user
	SendMessageTicker = 1 * time.Second // Ticker is 1s in send message
	CreateGroupTicker = 1 * time.Minute
)

type BaseResp struct {
	ErrCode int             `json:"errCode"`
	ErrMsg  string          `json:"errMsg"`
	Data    json.RawMessage `json:"data"`
}

type StressTest struct {
	Conf           *conf
	AdminUserID    string
	AdminToken     string
	DefaultGroupID string
	DefaultUserID  string
	UserCounter    int
	GroupCounter   int
	MsgCounter     int
	CreatedUsers   []string
	CreatedGroups  []string
	Mutex          sync.Mutex
	Ctx            context.Context
	Cancel         context.CancelFunc
	HttpClient     *http.Client
	Wg             sync.WaitGroup
	Once           sync.Once
}

type conf struct {
	Share config.Share
	Api   config.API
}

func initConfig(configDir string) (*config.Share, *config.API, error) {
	var (
		share     = &config.Share{}
		apiConfig = &config.API{}
	)

	err := config.Load(configDir, config.ShareFileName, config.EnvPrefixMap[config.ShareFileName], share)
	if err != nil {
		return nil, nil, err
	}

	err = config.Load(configDir, config.OpenIMAPICfgFileName, config.EnvPrefixMap[config.OpenIMAPICfgFileName], apiCon
	if err != nil {
		return nil, nil, err
	}

	return share, apiConfig, nil
}

// Post Request
func (st *StressTest) PostRequest(ctx context.Context, url string, reqbody any) ([]byte, error) {
	// Marshal body
	jsonBody, err := json.Marshal(reqbody)
	if err != nil {
		log.ZError(ctx, "Failed to marshal request body", err, "url", url, "reqbody", reqbody)
		return nil, err
	}
```

```
	req, err := http.NewRequest(http.MethodPost, url, bytes.NewReader(jsonBody))
	if err != nil {
		return nil, err
	}
	req.Header.Set("Content-Type", "application/json")
	req.Header.Set("operationID", st.AdminUserID)
	if st.AdminToken != "" {
		req.Header.Set("token", st.AdminToken)
	}

	// log.ZInfo(ctx, "Header info is ", "Content-Type", "application/json", "operationID", st.AdminUserID, "token", st

	resp, err := st.HttpClient.Do(req)
	if err != nil {
		log.ZError(ctx, "Failed to send request", err, "url", url, "reqbody", reqbody)
		return nil, err
	}
	defer resp.Body.Close()

	respBody, err := io.ReadAll(resp.Body)
	if err != nil {
		log.ZError(ctx, "Failed to read response body", err, "url", url)
		return nil, err
	}

	var baseResp BaseResp
	if err := json.Unmarshal(respBody, &baseResp); err != nil {
		log.ZError(ctx, "Failed to unmarshal response body", err, "url", url, "respBody", string(respBody))
		return nil, err
	}

	if baseResp.ErrCode != 0 {
		err = fmt.Errorf(baseResp.ErrMsg)
		log.ZError(ctx, "Failed to send request", err, "url", url, "reqbody", reqbody, "resp", baseResp)
		return nil, err
	}

	return baseResp.Data, nil
}

func (st *StressTest) GetAdminToken(ctx context.Context) (string, error) {
	req := auth.GetAdminTokenReq{
		Secret: st.Conf.Share.Secret,
		UserID: st.AdminUserID,
	}

	resp, err := st.PostRequest(ctx, ApiAddress+GetAdminToken, &req)
	if err != nil {
		return "", err
	}

	data := &auth.GetAdminTokenResp{}
	if err := json.Unmarshal(resp, &data); err != nil {
		return "", err
	}

	return data.Token, nil
}

func (st *StressTest) CreateUser(ctx context.Context, userID string) (string, error) {
	user := &sdkws.UserInfo{
		UserID:   userID,
		Nickname: userID,
	}
```

```
	req := pbuser.UserRegisterReq{
		Users: []*sdkws.UserInfo{user},
	}

	_, err := st.PostRequest(ctx, ApiAddress+CreateUser, &req)
	if err != nil {
		return "", err
	}

	st.UserCounter++
	return userID, nil
}

func (st *StressTest) ImportFriend(ctx context.Context, userID string) error {
	req := relation.ImportFriendReq{
		OwnerUserID:   userID,
		FriendUserIDs: TestTargetUserList,
	}

	_, err := st.PostRequest(ctx, ApiAddress+ImportFriend, &req)
	if err != nil {
		return err
	}

	return nil
}

func (st *StressTest) InviteToGroup(ctx context.Context, userID string) error {
	req := group.InviteUserToGroupReq{
		GroupID:        st.DefaultGroupID,
		InvitedUserIDs: []string{userID},
	}
	_, err := st.PostRequest(ctx, ApiAddress+InviteToGroup, &req)
	if err != nil {
		return err
	}

	return nil
}

func (st *StressTest) SendMsg(ctx context.Context, userID string) error {
	contentObj := map[string]any{
		"content": fmt.Sprintf("index %d. The current time is %s", st.MsgCounter, time.Now().Format("2006-01-02 15:04:05.
	}

	req := &apistruct.SendMsgReq{
		SendMsg: apistruct.SendMsg{
			SendID:         userID,
			SenderNickname: userID,
			GroupID:        st.DefaultGroupID,
			ContentType:    constant.Text,
			SessionType:    constant.ReadGroupChatType,
			Content:        contentObj,
		},
	}

	_, err := st.PostRequest(ctx, ApiAddress+SendMsg, &req)
	if err != nil {
		log.ZError(ctx, "Failed to send message", err, "userID", userID, "req", &req)
		return err
	}

	st.MsgCounter++

	return nil
}
```

```
func (st *StressTest) CreateGroup(ctx context.Context, userID string) (string, error) {
    groupID := fmt.Sprintf("StressTestGroup_%d_%s", st.GroupCounter, time.Now().Format("20060102150405"))

    groupInfo := &sdkws.GroupInfo{
        GroupID:   groupID,
        GroupName: groupID,
        GroupType: constant.WorkingGroup,
    }

    req := group.CreateGroupReq{
        OwnerUserID:   userID,
        MemberUserIDs: TestTargetUserList,
        GroupInfo:     groupInfo,
    }

    resp := group.CreateGroupResp{}

    response, err := st.PostRequest(ctx, ApiAddress+CreateGroup, &req)
    if err != nil {
        return "", err
    }

    if err := json.Unmarshal(response, &resp); err != nil {
        return "", err
    }

    st.GroupCounter++

    return resp.GroupInfo.GroupID, nil
}

func main() {
    var configPath string
    // defaultConfigDir := filepath.Join("..", "..", "..", "..", "..", "config")
    // flag.StringVar(&configPath, "c", defaultConfigDir, "config path")
    flag.StringVar(&configPath, "c", "", "config path")
    flag.Parse()

    if configPath == "" {
        _, _ = fmt.Fprintln(os.Stderr, "config path is empty")
        os.Exit(1)
        return
    }

    fmt.Printf(" Config Path: %s\n", configPath)

    share, apiConfig, err := initConfig(configPath)
    if err != nil {
        program.ExitWithError(err)
        return
    }

    ApiAddress = fmt.Sprintf("http://%s:%s", "127.0.0.1", fmt.Sprint(apiConfig.Api.Ports[0]))

    ctx, cancel := context.WithCancel(context.Background())
    ch := make(chan struct{})

    defer cancel()

    st := &StressTest{
        Conf: &conf{
            Share: *share,
            Api:   *apiConfig,
        },
        AdminUserID: share.IMAdminUser.UserIDs[0],
```

```
    Ctx:         ctx,
    Cancel:      cancel,
    HttpClient: &http.Client{
        Timeout: 50 * time.Second,
    },
}

c := make(chan os.Signal, 1)
signal.Notify(c, os.Interrupt, syscall.SIGTERM)
go func() {
    <-c
    fmt.Println("\nReceived stop signal, stopping...")

    select {
    case <-ch:
    default:
        close(ch)
    }

    st.Cancel()
}()

token, err := st.GetAdminToken(st.Ctx)
if err != nil {
    log.ZError(ctx, "Get Admin Token failed.", err, "AdminUserID", st.AdminUserID)
}

st.AdminToken = token
fmt.Println("Admin Token:", st.AdminToken)
fmt.Println("ApiAddress:", ApiAddress)

st.DefaultGroupID = DefaultGroupID

st.Wg.Add(1)
go func() {
    defer st.Wg.Done()

    ticker := time.NewTicker(CreateUserTicker)
    defer ticker.Stop()

    for st.UserCounter < MaxUser {
        select {
        case <-st.Ctx.Done():
            log.ZInfo(st.Ctx, "Stop Create user", "reason", "context done")
            return

        case <-ticker.C:
            // Create User
            userID := fmt.Sprintf("%d_Stresstest_%s", st.UserCounter, time.Now().Format("0102150405"))

            userCreatedID, err := st.CreateUser(st.Ctx, userID)
            if err != nil {
                log.ZError(st.Ctx, "Create User failed.", err, "UserID", userID)
                os.Exit(1)
                return
            }
            // fmt.Println("User Created ID:", userCreatedID)

            // Import Friend
            if err = st.ImportFriend(st.Ctx, userCreatedID); err != nil {
                log.ZError(st.Ctx, "Import Friend failed.", err, "UserID", userCreatedID)
                os.Exit(1)
                return
            }
            // Invite To Group
            if err = st.InviteToGroup(st.Ctx, userCreatedID); err != nil {
```

```
            log.ZError(st.Ctx, "Invite To Group failed.", err, "UserID", userCreatedID)
            os.Exit(1)
            return
        }

        st.Once.Do(func() {
            st.DefaultUserID = userCreatedID
            fmt.Println("Default Send User Created ID:", userCreatedID)
            close(ch)
        })
    }
}()

st.Wg.Add(1)
go func() {
    defer st.Wg.Done()

    ticker := time.NewTicker(SendMessageTicker)
    defer ticker.Stop()
    <-ch

    for {
        select {
        case <-st.Ctx.Done():
            log.ZInfo(st.Ctx, "Stop Send message", "reason", "context done")
            return

        case <-ticker.C:
            // Send Message
            if err = st.SendMsg(st.Ctx, st.DefaultUserID); err != nil {
                log.ZError(st.Ctx, "Send Message failed.", err, "UserID", st.DefaultUserID)
                continue
            }
        }
    }
}()

st.Wg.Add(1)
go func() {
    defer st.Wg.Done()

    ticker := time.NewTicker(CreateGroupTicker)
    defer ticker.Stop()
    <-ch

    for st.GroupCounter < MaxGroup {

        select {
        case <-st.Ctx.Done():
            log.ZInfo(st.Ctx, "Stop Create Group", "reason", "context done")
            return

        case <-ticker.C:

            // Create Group
            _, err := st.CreateGroup(st.Ctx, st.DefaultUserID)
            if err != nil {
                log.ZError(st.Ctx, "Create Group failed.", err, "UserID", st.DefaultUserID)
                os.Exit(1)
                return
            }

            // fmt.Println("Group Created ID:", groupID)
        }
    }
```

```
    }()

    st.Wg.Wait()
}
```

test/jwt

test/jwt/main.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"fmt"

	"github.com/golang-jwt/jwt/v4"
)

func main() {
	rawJWT := `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySUQiOiI4MjkzODEzMTgzIiwiUGxhdGZvcm1JRCI6NSwiZXhwIjoxNzA2NTk

	// Verify the token
	claims := &jwt.MapClaims{}
	parsedT, err := jwt.ParseWithClaims(rawJWT, claims, func(token *jwt.Token) (any, error) {
		// Validate the alg is HMAC signature
		if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
			return nil, fmt.Errorf("unexpected signing method: %v", token.Header["alg"])
		}

		if kid, ok := token.Header["kid"].(string); ok {
			fmt.Println("kid", kid)
		}

		return []byte("key1"), nil
	})

	if err != nil || !parsedT.Valid {
		fmt.Println("token valid failed", err)

		return
	}

	fmt.Println("ok")
}
```

test/webhook

test/webhook/msgmodify

test/webhook/msgmodify/main.go

```go
package main

import (
	"encoding/json"
	"fmt"
	"io"
	"net/http"
	"strings"

	"github.com/gin-gonic/gin"
	cbapi "github.com/openimsdk/open-im-server/v3/pkg/callbackstruct"
	"github.com/openimsdk/protocol/constant"
)

func main() {
	g := gin.Default()
	g.POST("/callbackExample/callbackBeforeMsgModifyCommand", toGin(handlerMsg))
	if err := g.Run(":10006"); err != nil {
		panic(err)
	}
}

func toGin[R any](fn func(c *gin.Context, req *R)) gin.HandlerFunc {
	return func(c *gin.Context) {
		body, err := io.ReadAll(c.Request.Body)
		if err != nil {
			c.String(http.StatusInternalServerError, err.Error())
			return
		}
		fmt.Printf("HTTP %s %s %s\n", c.Request.Method, c.Request.URL, body)
		var req R
		if err := json.Unmarshal(body, &req); err != nil {
			c.String(http.StatusInternalServerError, err.Error())
			return
		}
		fn(c, &req)
	}
}

func handlerMsg(c *gin.Context, req *cbapi.CallbackMsgModifyCommandReq) {
	var resp cbapi.CallbackMsgModifyCommandResp
	if req.ContentType != constant.Text {
		c.JSON(http.StatusOK, &resp)
		return
	}
	var textElem struct {
		Content string `json:"content"`
	}
	if err := json.Unmarshal([]byte(req.Content), &textElem); err != nil {
		c.String(http.StatusInternalServerError, err.Error())
		return
	}
	const word = "xxx"
	if strings.Contains(textElem.Content, word) {
		textElem.Content = strings.ReplaceAll(textElem.Content, word, strings.Repeat("*", len(word)))
		content, err := json.Marshal(&textElem)
		if err != nil {
			c.String(http.StatusInternalServerError, err.Error())
```

```
            return
        }
        tmp := string(content)
        resp.Content = &tmp
    }
    c.JSON(http.StatusOK, &resp)
}
```

test/stress-test-v2

test/stress-test-v2/main.go

```
package main

import (
	"bytes"
	"context"
	"encoding/json"
	"flag"
	"fmt"
	"io"
	"net/http"
	"os"
	"os/signal"
	"sync"
	"syscall"
	"time"

	"github.com/openimsdk/open-im-server/v3/pkg/apistruct"
	"github.com/openimsdk/open-im-server/v3/pkg/common/config"
	"github.com/openimsdk/protocol/auth"
	"github.com/openimsdk/protocol/constant"
	"github.com/openimsdk/protocol/group"
	"github.com/openimsdk/protocol/sdkws"
	pbuser "github.com/openimsdk/protocol/user"
	"github.com/openimsdk/tools/log"
	"github.com/openimsdk/tools/system/program"
)

// 1. Create 100K New Users
// 2. Create 100 100K Groups
// 3. Create 1000 999 Groups
// 4. Send message to 100K Groups every second
// 5. Send message to 999 Groups every minute

var (
	//  Use default userIDs List for testing, need to be created.
	TestTargetUserList = []string{
		// "<need-update-it>",
	}
	// DefaultGroupID = "<need-update-it>" // Use default group ID for testing, need to be created.
)

var (
	ApiAddress string

	// API method
	GetAdminToken      = "/auth/get_admin_token"
	UserCheck          = "/user/account_check"
	CreateUser         = "/user/user_register"
	ImportFriend       = "/friend/import_friend"
	InviteToGroup      = "/group/invite_user_to_group"
	GetGroupMemberInfo = "/group/get_group_members_info"
	SendMsg            = "/msg/send_msg"
	CreateGroup        = "/group/create_group"
	GetUserToken       = "/auth/user_token"
)

const (
	MaxUser            = 100000
	Max1kUser          = 1000
	Max100KGroup       = 100
	Max999Group        = 1000
```

```
	MaxInviteUserLimit = 999

	CreateUserTicker         = 1 * time.Second
	CreateGroupTicker        = 1 * time.Second
	Create100KGroupTicker    = 1 * time.Second
	Create999GroupTicker     = 1 * time.Second
	SendMsgTo100KGroupTicker = 1 * time.Second
	SendMsgTo999GroupTicker  = 1 * time.Minute
)

type BaseResp struct {
	ErrCode int             `json:"errCode"`
	ErrMsg  string          `json:"errMsg"`
	Data    json.RawMessage `json:"data"`
}

type StressTest struct {
	Conf                  *conf
	AdminUserID           string
	AdminToken            string
	DefaultGroupID        string
	DefaultUserID         string
	UserCounter           int
	CreateUserCounter     int
	Create100kGroupCounter int
	Create999GroupCounter  int
	MsgCounter            int
	CreatedUsers          []string
	CreatedGroups         []string
	Mutex                 sync.Mutex
	Ctx                   context.Context
	Cancel                context.CancelFunc
	HttpClient            *http.Client
	Wg                    sync.WaitGroup
	Once                  sync.Once
}

type conf struct {
	Share config.Share
	Api   config.API
}

func initConfig(configDir string) (*config.Share, *config.API, error) {
	var (
		share     = &config.Share{}
		apiConfig = &config.API{}
	)

	err := config.Load(configDir, config.ShareFileName, config.EnvPrefixMap[config.ShareFileName], share)
	if err != nil {
		return nil, nil, err
	}

	err = config.Load(configDir, config.OpenIMAPICfgFileName, config.EnvPrefixMap[config.OpenIMAPICfgFileName], apiCon
	if err != nil {
		return nil, nil, err
	}

	return share, apiConfig, nil
}

// Post Request
func (st *StressTest) PostRequest(ctx context.Context, url string, reqbody any) ([]byte, error) {
	// Marshal body
	jsonBody, err := json.Marshal(reqbody)
	if err != nil {
```

```
		log.ZError(ctx, "Failed to marshal request body", err, "url", url, "reqbody", reqbody)
		return nil, err
	}

	req, err := http.NewRequest(http.MethodPost, url, bytes.NewReader(jsonBody))
	if err != nil {
		return nil, err
	}
	req.Header.Set("Content-Type", "application/json")
	req.Header.Set("operationID", st.AdminUserID)
	if st.AdminToken != "" {
		req.Header.Set("token", st.AdminToken)
	}

	// log.ZInfo(ctx, "Header info is ", "Content-Type", "application/json", "operationID", st.AdminUserID, "token", st

	resp, err := st.HttpClient.Do(req)
	if err != nil {
		log.ZError(ctx, "Failed to send request", err, "url", url, "reqbody", reqbody)
		return nil, err
	}
	defer resp.Body.Close()

	respBody, err := io.ReadAll(resp.Body)
	if err != nil {
		log.ZError(ctx, "Failed to read response body", err, "url", url)
		return nil, err
	}

	var baseResp BaseResp
	if err := json.Unmarshal(respBody, &baseResp); err != nil {
		log.ZError(ctx, "Failed to unmarshal response body", err, "url", url, "respBody", string(respBody))
		return nil, err
	}

	if baseResp.ErrCode != 0 {
		err = fmt.Errorf(baseResp.ErrMsg)
		// log.ZError(ctx, "Failed to send request", err, "url", url, "reqbody", reqbody, "resp", baseResp)
		return nil, err
	}

	return baseResp.Data, nil
}

func (st *StressTest) GetAdminToken(ctx context.Context) (string, error) {
	req := auth.GetAdminTokenReq{
		Secret: st.Conf.Share.Secret,
		UserID: st.AdminUserID,
	}

	resp, err := st.PostRequest(ctx, ApiAddress+GetAdminToken, &req)
	if err != nil {
		return "", err
	}

	data := &auth.GetAdminTokenResp{}
	if err := json.Unmarshal(resp, &data); err != nil {
		return "", err
	}

	return data.Token, nil
}

func (st *StressTest) CheckUser(ctx context.Context, userIDs []string) ([]string, error) {
	req := pbuser.AccountCheckReq{
		CheckUserIDs: userIDs,
```

```
    }

    resp, err := st.PostRequest(ctx, ApiAddress+UserCheck, &req)
    if err != nil {
        return nil, err
    }

    data := &pbuser.AccountCheckResp{}
    if err := json.Unmarshal(resp, &data); err != nil {
        return nil, err
    }

    unRegisteredUserIDs := make([]string, 0)

    for _, res := range data.Results {
        if res.AccountStatus == constant.UnRegistered {
            unRegisteredUserIDs = append(unRegisteredUserIDs, res.UserID)
        }
    }

    return unRegisteredUserIDs, nil
}

func (st *StressTest) CreateUser(ctx context.Context, userID string) (string, error) {
    user := &sdkws.UserInfo{
        UserID:   userID,
        Nickname: userID,
    }

    req := pbuser.UserRegisterReq{
        Users: []*sdkws.UserInfo{user},
    }

    _, err := st.PostRequest(ctx, ApiAddress+CreateUser, &req)
    if err != nil {
        return "", err
    }

    st.UserCounter++
    return userID, nil
}

func (st *StressTest) CreateUserBatch(ctx context.Context, userIDs []string) error {
    // The method can import a large number of users at once.
    var userList []*sdkws.UserInfo

    defer st.Once.Do(
        func() {
            st.DefaultUserID = userIDs[0]
            fmt.Println("Default Send User Created ID:", st.DefaultUserID)
        })

    needUserIDs, err := st.CheckUser(ctx, userIDs)
    if err != nil {
        return err
    }

    for _, userID := range needUserIDs {
        user := &sdkws.UserInfo{
            UserID:   userID,
            Nickname: userID,
        }
        userList = append(userList, user)
    }

    req := pbuser.UserRegisterReq{
```

```
    Users: userList,
  }

  _, err = st.PostRequest(ctx, ApiAddress+CreateUser, &req)
  if err != nil {
    return err
  }

  st.UserCounter += len(userList)
  return nil
}

func (st *StressTest) GetGroupMembersInfo(ctx context.Context, groupID string, userIDs []string) ([]string, error) {
  needInviteUserIDs := make([]string, 0)

  const maxBatchSize = 500
  if len(userIDs) > maxBatchSize {
    for i := 0; i < len(userIDs); i += maxBatchSize {
      end := min(i+maxBatchSize, len(userIDs))
      batchUserIDs := userIDs[i:end]

      // log.ZInfo(ctx, "Processing group members batch", "groupID", groupID, "batch", i/maxBatchSize+1,
      // "batchUserCount", len(batchUserIDs))

      // Process a single batch
      batchReq := group.GetGroupMembersInfoReq{
        GroupID: groupID,
        UserIDs: batchUserIDs,
      }

      resp, err := st.PostRequest(ctx, ApiAddress+GetGroupMemberInfo, &batchReq)
      if err != nil {
        log.ZError(ctx, "Batch query failed", err, "batch", i/maxBatchSize+1)
        continue
      }

      data := &group.GetGroupMembersInfoResp{}
      if err := json.Unmarshal(resp, &data); err != nil {
        log.ZError(ctx, "Failed to parse batch response", err, "batch", i/maxBatchSize+1)
        continue
      }

      // Process the batch results
      existingMembers := make(map[string]bool)
      for _, member := range data.Members {
        existingMembers[member.UserID] = true
      }

      for _, userID := range batchUserIDs {
        if !existingMembers[userID] {
          needInviteUserIDs = append(needInviteUserIDs, userID)
        }
      }
    }
  }

  return needInviteUserIDs, nil
}

req := group.GetGroupMembersInfoReq{
  GroupID: groupID,
  UserIDs: userIDs,
}

resp, err := st.PostRequest(ctx, ApiAddress+GetGroupMemberInfo, &req)
if err != nil {
  return nil, err
```

```
	}

	data := &group.GetGroupMembersInfoResp{}
	if err := json.Unmarshal(resp, &data); err != nil {
		return nil, err
	}

	existingMembers := make(map[string]bool)
	for _, member := range data.Members {
		existingMembers[member.UserID] = true
	}

	for _, userID := range userIDs {
		if !existingMembers[userID] {
			needInviteUserIDs = append(needInviteUserIDs, userID)
		}
	}

	return needInviteUserIDs, nil
}

func (st *StressTest) InviteToGroup(ctx context.Context, groupID string, userIDs []string) error {
	req := group.InviteUserToGroupReq{
		GroupID:        groupID,
		InvitedUserIDs: userIDs,
	}
	_, err := st.PostRequest(ctx, ApiAddress+InviteToGroup, &req)
	if err != nil {
		return err
	}

	return nil
}

func (st *StressTest) SendMsg(ctx context.Context, userID string, groupID string) error {
	contentObj := map[string]any{
		// "content": fmt.Sprintf("index %d. The current time is %s", st.MsgCounter, time.Now().Format("2006-01-02 15:04:0
		"content": fmt.Sprintf("The current time is %s", time.Now().Format("2006-01-02 15:04:05.000")),
	}

	req := &apistruct.SendMsgReq{
		SendMsg: apistruct.SendMsg{
			SendID:         userID,
			SenderNickname: userID,
			GroupID:        groupID,
			ContentType:    constant.Text,
			SessionType:    constant.ReadGroupChatType,
			Content:        contentObj,
		},
	}

	_, err := st.PostRequest(ctx, ApiAddress+SendMsg, &req)
	if err != nil {
		log.ZError(ctx, "Failed to send message", err, "userID", userID, "req", &req)
		return err
	}

	st.MsgCounter++

	return nil
}

// Max userIDs number is 1000
func (st *StressTest) CreateGroup(ctx context.Context, groupID string, userID string, userIDsList []string) (string,
	groupInfo := &sdkws.GroupInfo{
		GroupID:   groupID,
```

```
    GroupName: groupID,
    GroupType: constant.WorkingGroup,
  }

  req := group.CreateGroupReq{
    OwnerUserID:   userID,
    MemberUserIDs: userIDsList,
    GroupInfo:     groupInfo,
  }

  resp := group.CreateGroupResp{}

  response, err := st.PostRequest(ctx, ApiAddress+CreateGroup, &req)
  if err != nil {
    return "", err
  }

  if err := json.Unmarshal(response, &resp); err != nil {
    return "", err
  }

  // st.GroupCounter++

  return resp.GroupInfo.GroupID, nil
}

func main() {
  var configPath string
  // defaultConfigDir := filepath.Join("..", "..", "..", "..", "..", "config")
  // flag.StringVar(&configPath, "c", defaultConfigDir, "config path")
  flag.StringVar(&configPath, "c", "", "config path")
  flag.Parse()

  if configPath == "" {
    _, _ = fmt.Fprintln(os.Stderr, "config path is empty")
    os.Exit(1)
    return
  }

  fmt.Printf(" Config Path: %s\n", configPath)

  share, apiConfig, err := initConfig(configPath)
  if err != nil {
    program.ExitWithError(err)
    return
  }

  ApiAddress = fmt.Sprintf("http://%s:%s", "127.0.0.1", fmt.Sprint(apiConfig.Api.Ports[0]))

  ctx, cancel := context.WithCancel(context.Background())
  // ch := make(chan struct{})

  st := &StressTest{
    Conf: &conf{
      Share: *share,
      Api:   *apiConfig,
    },
    AdminUserID: share.IMAdminUser.UserIDs[0],
    Ctx:         ctx,
    Cancel:      cancel,
    HttpClient: &http.Client{
      Timeout: 50 * time.Second,
    },
  }

  c := make(chan os.Signal, 1)
```

```
signal.Notify(c, os.Interrupt, syscall.SIGTERM)
go func() {
<-c
fmt.Println("\nReceived stop signal, stopping...")

go func() {
// time.Sleep(5 * time.Second)
fmt.Println("Force exit")
os.Exit(0)
}()

st.Cancel()
}()

token, err := st.GetAdminToken(st.Ctx)
if err != nil {
log.ZError(ctx, "Get Admin Token failed.", err, "AdminUserID", st.AdminUserID)
}

st.AdminToken = token
fmt.Println("Admin Token:", st.AdminToken)
fmt.Println("ApiAddress:", ApiAddress)

for i := range MaxUser {
userID := fmt.Sprintf("v2_StressTest_User_%d", i)
st.CreatedUsers = append(st.CreatedUsers, userID)
st.CreateUserCounter++
}

// err = st.CreateUserBatch(st.Ctx, st.CreatedUsers)
// if err != nil {
// log.ZError(ctx, "Create user failed.", err)
// }

const batchSize = 1000
totalUsers := len(st.CreatedUsers)
successCount := 0

if st.DefaultUserID == "" && len(st.CreatedUsers) > 0 {
st.DefaultUserID = st.CreatedUsers[0]
}

for i := 0; i < totalUsers; i += batchSize {
end := min(i+batchSize, totalUsers)

userBatch := st.CreatedUsers[i:end]
log.ZInfo(st.Ctx, "Creating user batch", "batch", i/batchSize+1, "count", len(userBatch))

err = st.CreateUserBatch(st.Ctx, userBatch)
if err != nil {
log.ZError(st.Ctx, "Batch user creation failed", err, "batch", i/batchSize+1)
} else {
successCount += len(userBatch)
log.ZInfo(st.Ctx, "Batch user creation succeeded", "batch", i/batchSize+1,
"progress", fmt.Sprintf("%d/%d", successCount, totalUsers))
}
}

// Execute create 100k group
st.Wg.Add(1)
go func() {
defer st.Wg.Done()

create100kGroupTicker := time.NewTicker(Create100KGroupTicker)
defer create100kGroupTicker.Stop()
```

```
    for i := range Max100KGroup {
        select {
        case <-st.Ctx.Done():
            log.ZInfo(st.Ctx, "Stop Create 100K Group")
            return

        case <-create100kGroupTicker.C:
            // Create 100K groups
            st.Wg.Add(1)
            go func(idx int) {
                startTime := time.Now()
                defer func() {
                    elapsedTime := time.Since(startTime)
                    log.ZInfo(st.Ctx, "100K group creation completed",
                        "groupID", fmt.Sprintf("v2_StressTest_Group_100K_%d", idx),
                        "index", idx,
                        "duration", elapsedTime.String())
                }()

                defer st.Wg.Done()
                defer func() {
                    st.Mutex.Lock()
                    st.Create100kGroupCounter++
                    st.Mutex.Unlock()
                }()

                groupID := fmt.Sprintf("v2_StressTest_Group_100K_%d", idx)

                if _, err = st.CreateGroup(st.Ctx, groupID, st.DefaultUserID, TestTargetUserList); err != nil {
                    log.ZError(st.Ctx, "Create group failed.", err)
                    // continue
                }

                for i := 0; i <= MaxUser/MaxInviteUserLimit; i++ {
                    InviteUserIDs := make([]string, 0)
                    // ensure TargetUserList is in group
                    InviteUserIDs = append(InviteUserIDs, TestTargetUserList...)

                    startIdx := max(i*MaxInviteUserLimit, 1)
                    endIdx := min((i+1)*MaxInviteUserLimit, MaxUser)

                    for j := startIdx; j < endIdx; j++ {
                        userCreatedID := fmt.Sprintf("v2_StressTest_User_%d", j)
                        InviteUserIDs = append(InviteUserIDs, userCreatedID)
                    }

                    if len(InviteUserIDs) == 0 {
                        // log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
                        continue
                    }

                    InviteUserIDs, err := st.GetGroupMembersInfo(ctx, groupID, InviteUserIDs)
                    if err != nil {
                        log.ZError(st.Ctx, "GetGroupMembersInfo failed.", err, "groupID", groupID)
                        continue
                    }

                    if len(InviteUserIDs) == 0 {
                        // log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
                        continue
                    }

                    // Invite To Group
                    if err = st.InviteToGroup(st.Ctx, groupID, InviteUserIDs); err != nil {
                        log.ZError(st.Ctx, "Invite To Group failed.", err, "UserID", InviteUserIDs)
                        continue
```

```
        // os.Exit(1)
        // return
    }
  }
 }(i)
 }
 }()

 // create 999 groups
 st.Wg.Add(1)
 go func() {
  defer st.Wg.Done()

  create999GroupTicker := time.NewTicker(Create999GroupTicker)
  defer create999GroupTicker.Stop()

  for i := range Max999Group {
   select {
   case <-st.Ctx.Done():
    log.ZInfo(st.Ctx, "Stop Create 999 Group")
    return

   case <-create999GroupTicker.C:
    // Create 999 groups
    st.Wg.Add(1)
    go func(idx int) {
     startTime := time.Now()
     defer func() {
      elapsedTime := time.Since(startTime)
      log.ZInfo(st.Ctx, "999 group creation completed",
       "groupID", fmt.Sprintf("v2_StressTest_Group_1K_%d", idx),
       "index", idx,
       "duration", elapsedTime.String())
     }()

     defer st.Wg.Done()
     defer func() {
      st.Mutex.Lock()
      st.Create999GroupCounter++
      st.Mutex.Unlock()
     }()

     groupID := fmt.Sprintf("v2_StressTest_Group_1K_%d", idx)

     if _, err = st.CreateGroup(st.Ctx, groupID, st.DefaultUserID, TestTargetUserList); err != nil {
      log.ZError(st.Ctx, "Create group failed.", err)
      // continue
     }
     for i := 0; i <= Max1kUser/MaxInviteUserLimit; i++ {
      InviteUserIDs := make([]string, 0)
      // ensure TargetUserList is in group
      InviteUserIDs = append(InviteUserIDs, TestTargetUserList...)

      startIdx := max(i*MaxInviteUserLimit, 1)
      endIdx := min((i+1)*MaxInviteUserLimit, Max1kUser)

      for j := startIdx; j < endIdx; j++ {
       userCreatedID := fmt.Sprintf("v2_StressTest_User_%d", j)
       InviteUserIDs = append(InviteUserIDs, userCreatedID)
      }

      if len(InviteUserIDs) == 0 {
       // log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
       continue
      }
```

```
        InviteUserIDs, err := st.GetGroupMembersInfo(ctx, groupID, InviteUserIDs)
        if err != nil {
            log.ZError(st.Ctx, "GetGroupMembersInfo failed.", err, "groupID", groupID)
            continue
        }

        if len(InviteUserIDs) == 0 {
            // log.ZWarn(st.Ctx, "InviteUserIDs is empty", nil, "groupID", groupID)
            continue
        }

        // Invite To Group
        if err = st.InviteToGroup(st.Ctx, groupID, InviteUserIDs); err != nil {
            log.ZError(st.Ctx, "Invite To Group failed.", err, "UserID", InviteUserIDs)
            continue
            // os.Exit(1)
            // return
        }
    }
  }(i)
 }
}()

// Send message to 100K groups
st.Wg.Wait()
fmt.Println("All groups created successfully, starting to send messages...")
log.ZInfo(ctx, "All groups created successfully, starting to send messages...")

var groups100K []string
var groups999 []string

for i := range Max100KGroup {
 groupID := fmt.Sprintf("v2_StressTest_Group_100K_%d", i)
 groups100K = append(groups100K, groupID)
}

for i := range Max999Group {
 groupID := fmt.Sprintf("v2_StressTest_Group_1K_%d", i)
 groups999 = append(groups999, groupID)
}

send100kGroupLimiter := make(chan struct{}, 20)
send999GroupLimiter := make(chan struct{}, 100)

// execute Send message to 100K groups
go func() {
 ticker := time.NewTicker(SendMsgTo100KGroupTicker)
 defer ticker.Stop()

 for {
  select {
  case <-st.Ctx.Done():
   log.ZInfo(st.Ctx, "Stop Send Message to 100K Group")
   return

  case <-ticker.C:
   // Send message to 100K groups
   for _, groupID := range groups100K {
    send100kGroupLimiter <- struct{}{}
    go func(groupID string) {
     defer func() { <-send100kGroupLimiter }()
     if err := st.SendMsg(st.Ctx, st.DefaultUserID, groupID); err != nil {
      log.ZError(st.Ctx, "Send message to 100K group failed.", err)
     }
```

```
            }(groupID)
        }
        // log.ZInfo(st.Ctx, "Send message to 100K groups successfully.")
    }
}
}()

// execute Send message to 999 groups
go func() {
    ticker := time.NewTicker(SendMsgTo999GroupTicker)
    defer ticker.Stop()

    for {
        select {
        case <-st.Ctx.Done():
            log.ZInfo(st.Ctx, "Stop Send Message to 999 Group")
            return

        case <-ticker.C:
            // Send message to 999 groups
            for _, groupID := range groups999 {
                send999GroupLimiter <- struct{}{}
                go func(groupID string) {
                    defer func() { <-send999GroupLimiter }()

                    if err := st.SendMsg(st.Ctx, st.DefaultUserID, groupID); err != nil {
                        log.ZError(st.Ctx, "Send message to 999 group failed.", err)
                    }
                }(groupID)
            }
            // log.ZInfo(st.Ctx, "Send message to 999 groups successfully.")
        }
    }
}()

<-st.Ctx.Done()
fmt.Println("Received signal to exit, shutting down...")
}
```

test/testdata

test/testdata/responses

test/testdata/responses/login.json

test/testdata/responses/register.json

test/testdata/responses/sendMessage.json

test/testdata/db

test/testdata/db/messages.json

test/testdata/db/users.json

test/testdata/requests

test/testdata/requests/login.json

test/testdata/requests/register.json

test/testdata/requests/send-message.json

test/e2e

test/e2e/e2e.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package e2e

import (
	"testing"

	gettoken "github.com/openimsdk/open-im-server/v3/test/e2e/api/token"
	"github.com/openimsdk/open-im-server/v3/test/e2e/api/user"
)

// RunE2ETests checks configuration parameters (specified through flags) and then runs
// E2E tests using the Ginkgo runner.
// If a "report directory" is specified, one or more JUnit test reports will be
// generated in this directory, and cluster logs will also be saved.
// This function is called on each Ginkgo node in parallel mode.
func RunE2ETests(t *testing.T) {

	// Example usage of new functions
	token, _ := gettoken.GetUserToken("openIM123456")

	// Example of getting user info
	_ = user.GetUsersInfo(token, []string{"user1", "user2"})

	// Example of updating user info
	_ = user.UpdateUserInfo(token, "user1", "NewNickname", "https://github.com/openimsdk/open-im-server/blob/main/asset

	// Example of getting users' online status
	_ = user.GetUsersOnlineStatus(token, []string{"user1", "user2"})

	// Example of forcing a logout
	_ = user.ForceLogout(token, "4950983283", 2)

	// Example of checking user account
	_ = user.CheckUserAccount(token, []string{"openIM123456", "anotherUserID"})

	// Example of getting users
	_ = user.GetUsers(token, 1, 100)
}
```

test/e2e/e2e_test.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package e2e

import (
	"flag"
	"testing"

	"github.com/openimsdk/open-im-server/v3/test/e2e/framework/config"
)

// handleFlags sets up all flags and parses the command line.
func handleFlags() {
	config.CopyFlags(config.Flags, flag.CommandLine)
	flag.Parse()
}

func TestMain(m *testing.M) {
	handleFlags()
	m.Run()
}

func TestE2E(t *testing.T) {
	RunE2ETests(t)
}
```

test/e2e/page

test/e2e/page/chat_page.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package page
```

test/e2e/page/login_page.go

```go
// Copyright © 2024 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package page
```

test/e2e/framework

test/e2e/framework/ginkgowrapper

test/e2e/framework/ginkgowrapper/ginkgowrapper.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package ginkgowrapper
```

test/e2e/framework/ginkgowrapper/ginkgowrapper_test.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package ginkgowrapper
```

test/e2e/framework/config

test/e2e/framework/config/config.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package config

import (
	"flag"
	"os"
)

// Flags is the flag set that AddOptions adds to. Test authors should
// also use it instead of directly adding to the global command line.
var Flags = flag.NewFlagSet("", flag.ContinueOnError)

// CopyFlags ensures that all flags that are defined in the source flag
// set appear in the target flag set as if they had been defined there
// directly. From the flag package it inherits the behavior that there
// is a panic if the target already contains a flag from the source.
func CopyFlags(source *flag.FlagSet, target *flag.FlagSet) {
	source.VisitAll(func(flag *flag.Flag) {
		// We don't need to copy flag.DefValue. The original
		// default (from, say, flag.String) was stored in
		// the value and gets extracted by Var for the help
		// message.
		target.Var(flag.Value, flag.Name, flag.Usage)
	})
}

// Config defines the configuration structure for the OpenIM components.
type Config struct {
	APIHost             string
	APIPort             string
	MsgGatewayHost      string
	MsgTransferHost     string
	PushHost            string
	RPCAuthHost         string
	RPCConversationHost string
	RPCFriendHost       string
	RPCGroupHost        string
	RPCMsgHost          string
	RPCThirdHost        string
	RPCUserHost         string
	// Add other configuration fields as needed
}

// LoadConfig loads the configurations from environment variables or default values.
func LoadConfig() *Config {
	return &Config{
		APIHost: getEnv("OPENIM_API_HOST", "127.0.0.1"),
		APIPort: getEnv("API_OPENIM_PORT", "10002"),
```

```
    // TODO: Set default variable
    MsgGatewayHost:      getEnv("OPENIM_MSGGATEWAY_HOST", "default-msggateway-host"),
    MsgTransferHost:     getEnv("OPENIM_MSGTRANSFER_HOST", "default-msgtransfer-host"),
    PushHost:            getEnv("OPENIM_PUSH_HOST", "default-push-host"),
    RPCAuthHost:         getEnv("OPENIM_RPC_AUTH_HOST", "default-rpc-auth-host"),
    RPCConversationHost: getEnv("OPENIM_RPC_CONVERSATION_HOST", "default-rpc-conversation-host"),
    RPCFriendHost:       getEnv("OPENIM_RPC_FRIEND_HOST", "default-rpc-friend-host"),
    RPCGroupHost:        getEnv("OPENIM_RPC_GROUP_HOST", "default-rpc-group-host"),
    RPCMsgHost:          getEnv("OPENIM_RPC_MSG_HOST", "default-rpc-msg-host"),
    RPCThirdHost:        getEnv("OPENIM_RPC_THIRD_HOST", "default-rpc-third-host"),
    RPCUserHost:         getEnv("OPENIM_RPC_USER_HOST", "default-rpc-user-host"),
  }
}

// getEnv is a helper function to read an environment variable or return a default value.
func getEnv(key, defaultValue string) string {
  value, exists := os.LookupEnv(key)
  if !exists {
    return defaultValue
  }
  return value
}
```

test/e2e/framework/config/config_test.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package config

import (
	"flag"
	"reflect"
	"testing"
)

func TestCopyFlags(t *testing.T) {
	type args struct {
		source *flag.FlagSet
		target *flag.FlagSet
	}
	tests := []struct {
		name    string
		args    args
		wantErr bool
	}{{
		{
			name: "Copy empty source to empty target",
			args: args{
				source: flag.NewFlagSet("source", flag.ContinueOnError),
				target: flag.NewFlagSet("target", flag.ContinueOnError),
			},
			wantErr: false,
		},
		{
			name: "Copy non-empty source to empty target",
			args: args{
				source: func() *flag.FlagSet {
					fs := flag.NewFlagSet("source", flag.ContinueOnError)
					fs.String("test-flag", "default", "test usage")
					return fs
				}(),
				target: flag.NewFlagSet("target", flag.ContinueOnError),
			},
			wantErr: false,
		},
		{
			name: "Copy source to target with existing flag",
			args: args{
				source: func() *flag.FlagSet {
					fs := flag.NewFlagSet("source", flag.ContinueOnError)
					fs.String("test-flag", "default", "test usage")
					return fs
				}(),
				target: func() *flag.FlagSet {
					fs := flag.NewFlagSet("target", flag.ContinueOnError)
					fs.String("test-flag", "default", "test usage")
					return fs
```

```
        }(),
    },
    wantErr: true,
  },
 }
 for _, tt := range tests {
  t.Run(tt.name, func(t *testing.T) {
   defer func() {
    if r := recover(); (r != nil) != tt.wantErr {
     t.Errorf("CopyFlags() panic = %v, wantErr %v", r, tt.wantErr)
    }
   }()
   CopyFlags(tt.args.source, tt.args.target)

   // Verify the replicated tag
   if !tt.wantErr {
    tt.args.source.VisitAll(func(f *flag.Flag) {
     if gotFlag := tt.args.target.Lookup(f.Name); gotFlag == nil || !reflect.DeepEqual(gotFlag, f) {
      t.Errorf("CopyFlags() failed to copy flag %s", f.Name)
     }
    })
   }
  })
 }
}
```

test/e2e/framework/helpers

test/e2e/framework/helpers/chat

test/e2e/framework/helpers/chat/chat.go

```
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
	"fmt"
	"io"
	"net/http"
	"os"
	"os/exec"
	"path/filepath"
)

var (
	// The default template version.
	defaultTemplateVersion = "v1.3.0"
)

func main() {
	// Define the URL to get the latest version
	// latestVersionURL := "https://github.com/openimsdk/chat/releases/latest"
	// latestVersion, err := getLatestVersion(latestVersionURL)
	// if err != nil {
	// 	fmt.Printf("Failed to get the latest version: %v\n", err)
	// 	return
	// }
	latestVersion := defaultTemplateVersion

	// getLatestVersion

	// Construct the download URL
	downloadURL := fmt.Sprintf("https://github.com/openimsdk/chat/releases/download/%s/chat_Linux_x86_64.tar.gz", lates

	// Set the installation directory
	installDir := "/tmp/chat"

	// Clear the installation directory before proceeding
	err := os.RemoveAll(installDir)
	if err != nil {
		fmt.Printf("Failed to clear installation directory: %v\n", err)
		return
	}

	// Create the installation directory
	err = os.MkdirAll(installDir, 0755)
	if err != nil {
```

```
		fmt.Printf("Failed to create installation directory: %v\n", err)
		return
	}

	// Download and extract OpenIM Chat to the installation directory
	err = downloadAndExtract(downloadURL, installDir)
	if err != nil {
		fmt.Printf("Failed to download and extract OpenIM Chat: %v\n", err)
		return
	}

	// Create configuration file directory
	configDir := filepath.Join(installDir, "config")
	err = os.MkdirAll(configDir, 0755)
	if err != nil {
		fmt.Printf("Failed to create configuration directory: %v\n", err)
		return
	}

	// Download configuration files
	configURL := "https://raw.githubusercontent.com/openimsdk/chat/main/config/config.yaml"
	err = downloadAndExtract(configURL, configDir)
	if err != nil {
		fmt.Printf("Failed to download and extract configuration files: %v\n", err)
		return
	}

	// Define the processes to be started
	cmds := []string{
		"admin-api",
		"admin-rpc",
		"chat-api",
		"chat-rpc",
	}

	// Start each process in a new goroutine
	for _, cmd := range cmds {
		go startProcess(filepath.Join(installDir, cmd))
	}

	// Block the main thread indefinitely
	select {}
}

/* func getLatestVersion(url string) (string, error) {
	resp, err := webhook.Get(url)
	if err != nil {
		return "", err
	}
	defer resp.Body.Close()

	// 	location := resp.Header.Get("Location")
	// 	if location == "" {
	// 		return defaultTemplateVersion, nil
	// 	}

	// Extract the version number from the URL
	latestVersion := filepath.Base(location)
	return latestVersion, nil
} */

// downloadAndExtract downloads a file from a URL and extracts it to a destination directory.
func downloadAndExtract(url, destDir string) error {
	resp, err := http.Get(url)
	if err != nil {
		return err
```

```
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        return fmt.Errorf("error downloading file, HTTP status code: %d", resp.StatusCode)
    }

    // Create the destination directory
    err = os.MkdirAll(destDir, 0755)
    if err != nil {
        return err
    }

    // Define the path for the downloaded file
    filePath := filepath.Join(destDir, "downloaded_file.tar.gz")
    file, err := os.Create(filePath)
    if err != nil {
        return err
    }
    defer file.Close()

    // Copy the downloaded file
    _, err = io.Copy(file, resp.Body)
    if err != nil {
        return err
    }

    // Extract the file
    cmd := exec.Command("tar", "xzvf", filePath, "-C", destDir)
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr
    return cmd.Run()
}

// startProcess starts a process and prints any errors encountered.
func startProcess(cmdPath string) {
    cmd := exec.Command(cmdPath)
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr
    if err := cmd.Run(); err != nil {
        fmt.Printf("Failed to start process %s: %v\n", cmdPath, err)
    }
}
```

test/e2e/api

test/e2e/api/user

test/e2e/api/user/curd.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package user

import (
	"fmt"

	gettoken "github.com/openimsdk/open-im-server/v3/test/e2e/api/token"
	"github.com/openimsdk/open-im-server/v3/test/e2e/framework/config"
)

// UserInfoRequest represents a request to get or update user information.
type UserInfoRequest struct {
	UserIDs  []string       `json:"userIDs,omitempty"`
	UserInfo *gettoken.User `json:"userInfo,omitempty"`
}

// GetUsersOnlineStatusRequest represents a request to get users' online status.
type GetUsersOnlineStatusRequest struct {
	UserIDs []string `json:"userIDs"`
}

// GetUsersInfo retrieves detailed information for a list of user IDs.
func GetUsersInfo(token string, userIDs []string) error {

	url := fmt.Sprintf("http://%s:%s/user/get_users_info", config.LoadConfig().APIHost, config.LoadConfig().APIPort)

	requestBody := UserInfoRequest{
		UserIDs: userIDs,
	}
	return sendPostRequestWithToken(url, token, requestBody)
}

// UpdateUserInfo updates the information for a user.
func UpdateUserInfo(token, userID, nickname, faceURL string) error {

	url := fmt.Sprintf("http://%s:%s/user/update_user_info", config.LoadConfig().APIHost, config.LoadConfig().APIPort)

	requestBody := UserInfoRequest{
		UserInfo: &gettoken.User{
			UserID:   userID,
			Nickname: nickname,
			FaceURL:  faceURL,
		},
	}
	return sendPostRequestWithToken(url, token, requestBody)
```

```go
}

// GetUsersOnlineStatus retrieves the online status for a list of user IDs.
func GetUsersOnlineStatus(token string, userIDs []string) error {

    url := fmt.Sprintf("http://%s:%s/user/get_users_online_status", config.LoadConfig().APIHost, config.LoadConfig().AP

    requestBody := GetUsersOnlineStatusRequest{
        UserIDs: userIDs,
    }

    return sendPostRequestWithToken(url, token, requestBody)
}
```

test/e2e/api/user/user.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package user

import (
	"bytes"
	"encoding/json"
	"fmt"
	"io"
	"net/http"

	gettoken "github.com/openimsdk/open-im-server/v3/test/e2e/api/token"
	"github.com/openimsdk/open-im-server/v3/test/e2e/framework/config"
)

// ForceLogoutRequest represents a request to force a user logout.
type ForceLogoutRequest struct {
	PlatformID int    `json:"platformID"`
	UserID     string `json:"userID"`
}

// CheckUserAccountRequest represents a request to check a user account.
type CheckUserAccountRequest struct {
	CheckUserIDs []string `json:"checkUserIDs"`
}

// GetUsersRequest represents a request to get a list of users.
type GetUsersRequest struct {
	Pagination Pagination `json:"pagination"`
}

// Pagination specifies the page number and number of items per page.
type Pagination struct {
	PageNumber int `json:"pageNumber"`
	ShowNumber int `json:"showNumber"`
}

// ForceLogout forces a user to log out.
func ForceLogout(token, userID string, platformID int) error {

	url := fmt.Sprintf("http://%s:%s/auth/force_logout", config.LoadConfig().APIHost, config.LoadConfig().APIPort)

	requestBody := ForceLogoutRequest{
		PlatformID: platformID,
		UserID:     userID,
	}
	return sendPostRequestWithToken(url, token, requestBody)
}

// CheckUserAccount checks if the user accounts exist.
func CheckUserAccount(token string, userIDs []string) error {
```

```
    url := fmt.Sprintf("http://%s:%s/user/account_check", config.LoadConfig().APIHost, config.LoadConfig().APIPort)

    requestBody := CheckUserAccountRequest{
        CheckUserIDs: userIDs,
    }
    return sendPostRequestWithToken(url, token, requestBody)
}

// GetUsers retrieves a list of users with pagination.
func GetUsers(token string, pageNumber, showNumber int) error {

    url := fmt.Sprintf("http://%s:%s/user/account_check", config.LoadConfig().APIHost, config.LoadConfig().APIPort)

    requestBody := GetUsersRequest{
        Pagination: Pagination{
            PageNumber: pageNumber,
            ShowNumber: showNumber,
        },
    }
    return sendPostRequestWithToken(url, token, requestBody)
}

// sendPostRequestWithToken sends a POST request with a token in the header.
func sendPostRequestWithToken(url, token string, body any) error {
    reqBytes, err := json.Marshal(body)
    if err != nil {
        return err
    }

    client := &http.Client{}
    req, err := http.NewRequest("POST", url, bytes.NewBuffer(reqBytes))
    if err != nil {
        return err
    }

    req.Header.Add("Content-Type", "application/json")
    req.Header.Add("operationID", gettoken.OperationID)
    req.Header.Add("token", token)

    resp, err := client.Do(req)
    if err != nil {
        return err
    }
    defer resp.Body.Close()

    respBody, err := io.ReadAll(resp.Body)
    if err != nil {
        return err
    }

    var respData map[string]any
    if err := json.Unmarshal(respBody, &respData); err != nil {
        return err
    }

    if errCode, ok := respData["errCode"].(float64); ok && errCode != 0 {
        return fmt.Errorf("error in response: %v", respData)
    }

    return nil
}
```

test/e2e/api/token

test/e2e/api/token/token.go

```go
// Copyright © 2023 OpenIM. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package token

import (
"bytes"
"encoding/json"
"fmt"
"io"
"net/http"
)

// API endpoints and other constants.
const (
APIHost         = "http://127.0.0.1:10002"
UserTokenURL    = APIHost + "/auth/user_token"
UserRegisterURL = APIHost + "/user/user_register"
SecretKey       = "openIM123"
OperationID     = "1646445464564"
)

// UserTokenRequest represents a request to get a user token.
type UserTokenRequest struct {
Secret     string `json:"secret"`
PlatformID int    `json:"platformID"`
UserID     string `json:"userID"`
}

// UserTokenResponse represents a response containing a user token.
type UserTokenResponse struct {
Token   string `json:"token"`
ErrCode int    `json:"errCode"`
}

// User represents user data for registration.
type User struct {
UserID   string `json:"userID"`
Nickname string `json:"nickname"`
FaceURL  string `json:"faceURL"`
}

// UserRegisterRequest represents a request to register a user.
type UserRegisterRequest struct {
Users []User `json:"users"`
}

/* func main() {
// Example usage of functions
token, err := GetUserToken("openIM123456")
```

```
    if err != nil {
        log.Fatalf("Error getting user token: %v", err)
    }
    fmt.Println("Token:", token)

    err = RegisterUser(token, "testUserID", "TestNickname", "https://example.com/image.jpg")
    if err != nil {
        log.Fatalf("Error registering user: %v", err)
    }
    } */

    // GetUserToken requests a user token from the API.
    func GetUserToken(userID string) (string, error) {
        reqBody := UserTokenRequest{
            Secret:     SecretKey,
            PlatformID: 1,
            UserID:     userID,
        }
        reqBytes, err := json.Marshal(reqBody)
        if err != nil {
            return "", err
        }

        resp, err := http.Post(UserTokenURL, "application/json", bytes.NewBuffer(reqBytes))
        if err != nil {
            return "", err
        }
        defer resp.Body.Close()

        var tokenResp UserTokenResponse
        if err := json.NewDecoder(resp.Body).Decode(&tokenResp); err != nil {
            return "", err
        }

        if tokenResp.ErrCode != 0 {
            return "", fmt.Errorf("error in token response: %v", tokenResp.ErrCode)
        }

        return tokenResp.Token, nil
    }

    // RegisterUser registers a new user using the API.
    func RegisterUser(token, userID, nickname, faceURL string) error {
        user := User{
            UserID:   userID,
            Nickname: nickname,
            FaceURL:  faceURL,
        }
        reqBody := UserRegisterRequest{
            Users: []User{user},
        }
        reqBytes, err := json.Marshal(reqBody)
        if err != nil {
            return err
        }

        client := &http.Client{}
        req, err := http.NewRequest("POST", UserRegisterURL, bytes.NewBuffer(reqBytes))
        if err != nil {
            return err
        }

        req.Header.Add("Content-Type", "application/json")
        req.Header.Add("operationID", OperationID)
        req.Header.Add("token", token)
```

```
	resp, err := client.Do(req)
	if err != nil {
		return err
	}
	defer resp.Body.Close()

	respBody, err := io.ReadAll(resp.Body)
	if err != nil {
		return err
	}

	var respData map[string]any
	if err := json.Unmarshal(respBody, &respData); err != nil {
		return err
	}

	if errCode, ok := respData["errCode"].(float64); ok && errCode != 0 {
		return fmt.Errorf("error in user registration response: %v", respData)
	}

	return nil
}
```

deployments

deployments/deploy

deployments/deploy/clusterRole.yml

```yaml
# ClusterRole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: service-reader
rules:
  - apiGroups: [""]
    resources: ["services", "endpoints"]
    verbs: ["get", "list", "watch"]

---
# ClusterRoleBinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: default-service-reader-binding
subjects:
  - kind: ServiceAccount
    name: default
    namespace: default
roleRef:
  kind: ClusterRole
  name: service-reader
  apiGroup: rbac.authorization.k8s.io
```

deployments/deploy/ingress.yml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: openim-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: openim-nginx
  rules:
    - http:
        paths:
          - path: /openim-api
            pathType: Prefix
            backend:
              service:
                name: openim-api-service
                port:
                  number: 10002
          - path: /openim-msggateway
            pathType: Prefix
            backend:
              service:
                name: openim-msggateway-service
                port:
                  number: 10001
```

deployments/deploy/kafka-secret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: openim-kafka-secret
type: Opaque
data:
  kafka-password: ""
```

deployments/deploy/kafka-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: kafka-service
  labels:
    app: kafka
spec:
  ports:
    - name: plaintext
      port: 9092
      targetPort: 9092
    - name: controller
      port: 9093
      targetPort: 9093
    - name: external
      port: 19094
      targetPort: 9094
  selector:
    app: kafka
  type: ClusterIP
```

deployments/deploy/kafka-statefulset.yml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: kafka-statefulset
  labels:
    app: kafka
spec:
  replicas: 2
  selector:
    matchLabels:
      app: kafka
  serviceName: "kafka-service"
  template:
    metadata:
      labels:
        app: kafka
    spec:
      containers:
        - name: kafka
          image: bitnami/kafka:3.5.1
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              memory: "2Gi"
              cpu: "1000m"
            requests:
              memory: "1Gi"
              cpu: "500m"
          ports:
            - containerPort: 9092 # PLAINTEXT
            - containerPort: 9093 # CONTROLLER
            - containerPort: 9094 # EXTERNAL
          env:
            - name: TZ
              value: "Asia/Shanghai"
            - name: KAFKA_CFG_NODE_ID
              value: "0"
            - name: KAFKA_CFG_PROCESS_ROLES
              value: "controller,broker"
            - name: KAFKA_CFG_CONTROLLER_QUORUM_VOTERS
              value: "0@kafka-service:9093"
            - name: KAFKA_CFG_LISTENERS
              value: "PLAINTEXT://:9092,CONTROLLER://:9093,EXTERNAL://:9094"
            - name: KAFKA_CFG_ADVERTISED_LISTENERS
              value: "PLAINTEXT://kafka-service:9092,EXTERNAL://kafka-service:19094"
            - name: KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP
              value: "CONTROLLER:PLAINTEXT,EXTERNAL:PLAINTEXT,PLAINTEXT:PLAINTEXT"
            - name: KAFKA_CFG_CONTROLLER_LISTENER_NAMES
              value: "CONTROLLER"
            - name: KAFKA_CFG_AUTO_CREATE_TOPICS_ENABLE
              value: "true"
          volumeMounts:
            - name: kafka-data
              mountPath: /bitnami/kafka

      volumes:
        - name: kafka-data
          persistentVolumeClaim:
            claimName: kafka-pvc

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name: kafka-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

deployments/deploy/minio-secret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: openim-minio-secret
type: Opaque
data:
  minio-root-user: cm9vdA== # Base64 encoded "root"
  minio-root-password: b3BlbklNMTIz # Base64 encoded "openIM123"
```

deployments/deploy/minio-service.yml

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: minio-service
spec:
  selector:
    app: minio
  ports:
    - name: minio
      protocol: TCP
      port: 10005 # External port for accessing MinIO service
      targetPort: 9000 # Container port for MinIO service
    - name: minio-console
      protocol: TCP
      port: 19090 # External port for accessing MinIO console
      targetPort: 9090 # Container port for MinIO console
  type: NodePort
```

deployments/deploy/minio-statefulset.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: minio
  labels:
    app: minio
spec:
  replicas: 2
  selector:
    matchLabels:
      app: minio
  template:
    metadata:
      labels:
        app: minio
    spec:
      containers:
        - name: minio
          image: minio/minio:RELEASE.2024-01-11T07-46-16Z
          ports:
            - containerPort: 9000 # MinIO service port
            - containerPort: 9090 # MinIO console port
          volumeMounts:
            - name: minio-data
              mountPath: /data
            - name: minio-config
              mountPath: /root/.minio
          env:
            - name: TZ
              value: "Asia/Shanghai"
            - name: MINIO_ROOT_USER
              valueFrom:
                secretKeyRef:
                  name: openim-minio-secret
                  key: minio-root-user
            - name: MINIO_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-minio-secret
                  key: minio-root-password
          command:
            - "/bin/sh"
            - "-c"
            - |
              mkdir -p /data && \
              minio server /data --console-address ":9090"
      volumes:
        - name: minio-data
          persistentVolumeClaim:
            claimName: minio-pvc
        - name: minio-config
          persistentVolumeClaim:
            claimName: minio-config-pvc

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

```
      storage: 10Gi

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-config-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

deployments/deploy/mongo-secret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: openim-mongo-secret
type: Opaque
data:
  mongo_openim_username: b3BlbklN # base64 for "openIM", this user credentials need in authSource database.
  mongo_openim_password: b3BlbklNMTIz # base64 for "openIM123"
```

deployments/deploy/mongo-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
spec:
  selector:
    app: mongo
  ports:
    - name: mongodb-port
      protocol: TCP
      port: 37017
      targetPort: 27017
  type: NodePort
```

deployments/deploy/mongo-statefulset.yml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongo-statefulset
spec:
  serviceName: "mongo"
  replicas: 2
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
        - name: mongo
          image: mongo:7.0
          command: ["/bin/bash", "-c"]
          args:
            - >
              docker-entrypoint.sh mongod --wiredTigerCacheSizeGB ${wiredTigerCacheSizeGB} --auth &
              until mongosh -u ${MONGO_INITDB_ROOT_USERNAME} -p ${MONGO_INITDB_ROOT_PASSWORD} --authenticationDataba
                echo "Waiting for MongoDB to start...";
                sleep 1;
              done &&
              mongosh -u ${MONGO_INITDB_ROOT_USERNAME} -p ${MONGO_INITDB_ROOT_PASSWORD} --authenticationDatabase adm
              db = db.getSiblingDB(\"${MONGO_INITDB_DATABASE}\");
              if (!db.getUser(\"${MONGO_OPENIM_USERNAME}\")) {
                db.createUser({
                  user: \"${MONGO_OPENIM_USERNAME}\",
                  pwd: \"${MONGO_OPENIM_PASSWORD}\",
                  roles: [{role: \"readWrite\", db: \"${MONGO_INITDB_DATABASE}\"}]
                });
                print(\"User created successfully: \");
                print(\"Username: ${MONGO_OPENIM_USERNAME}\");
                print(\"Password: ${MONGO_OPENIM_PASSWORD}\");
                print(\"Database: ${MONGO_INITDB_DATABASE}\");
              } else {
                print(\"User already exists in database: ${MONGO_INITDB_DATABASE}, Username: ${MONGO_OPENIM_USERNAME
              }
              " &&
              tail -f /dev/null
          ports:
            - containerPort: 27017
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-init-secret
                  key: mongo_initdb_root_username
            - name: MONGO_INITDB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-init-secret
                  key: mongo_initdb_root_password
            - name: MONGO_INITDB_DATABASE
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-init-secret
                  key: mongo_initdb_database
            - name: MONGO_OPENIM_USERNAME
              valueFrom:
                secretKeyRef:
```

```
                name: openim-mongo-init-secret
                key: mongo_openim_username
            - name: MONGO_OPENIM_PASSWORD
              valueFrom:
                secretKeyRef:
                name: openim-mongo-init-secret
                key: mongo_openim_password
            - name: TZ
              value: "Asia/Shanghai"
            - name: wiredTigerCacheSizeGB
              value: "1"
          volumeMounts:
            - name: mongo-storage
              mountPath: /data/db

      volumes:
        - name: mongo-storage
          persistentVolumeClaim:
            claimName: mongo-pvc

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi

---
apiVersion: v1
kind: Secret
metadata:
  name: openim-mongo-init-secret
type: Opaque
data:
  mongo_initdb_root_username: cm9vdA== # base64 for "root"
  mongo_initdb_root_password: b3BlbklNMTIz # base64 for "openIM123"
  mongo_initdb_database: b3BlbmltX3Yz # base64 for "openim_v3"
  mongo_openim_username: b3BlbklN # base64 for "openIM"
  mongo_openim_password: b3BlbklNMTIz # base64 for "openIM123"
```

deployments/deploy/openim-api-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: openim-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: openim-api
  template:
    metadata:
      labels:
        app: openim-api
    spec:
      containers:
        - name: openim-api-container
          image: openim/openim-api:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password

          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10002
            - containerPort: 12002
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-api-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: openim-api-service
spec:
  selector:
    app: openim-api
  ports:
    - name: http-10002
      protocol: TCP
      port: 10002
      targetPort: 10002
    - name: prometheus-12002
      protocol: TCP
      port: 12002
      targetPort: 12002
  type: NodePort
```

deployments/deploy/openim-config.yml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: openim-config
data:
  discovery.yml: |
    enable: "kubernetes" # "kubernetes" or "etcd"
    kubernetes:
      namespace: default
    etcd:
      rootDirectory: openim
      address: [ localhost:12379 ]
      username: ''
      password: ''

    rpcService:
      user: user-rpc-service
      friend: friend-rpc-service
      msg: msg-rpc-service
      push: push-rpc-service
      messageGateway: messagegateway-rpc-service
      group: group-rpc-service
      auth: auth-rpc-service
      conversation: conversation-rpc-service
      third: third-rpc-service

  log.yml: |
    # Log storage path, default is acceptable, change to a full path if modification is needed
    storageLocation: ./logs/
    # Log rotation period (in hours), default is acceptable
    rotationTime: 24
    # Number of log files to retain, default is acceptable
    remainRotationCount: 2
    # Log level settings: 3 for production environment; 6 for more verbose logging in debugging environments
    remainLogLevel: 6
    # Whether to output to standard output, default is acceptable
    isStdout: true
    # Whether to log in JSON format, default is acceptable
    isJson: false
    # output simplify log when KeyAndValues's value len is bigger than 50 in rpc method log
    isSimplify: true

  mongodb.yml: |
    # URI for database connection, leave empty if using address and credential settings directly
    uri: ''
    # List of MongoDB server addresses
    address: [ mongo-service:37017 ]
    # Name of the database
    database: openim_v3
    # Username for database authentication
    username: ''  # openIM
    # Password for database authentication
    password: '' # openIM123
    # Authentication source for database authentication, if use root user, set it to admin
    authSource: openim_v3
    # Maximum number of connections in the connection pool
    maxPoolSize: 100
    # Maximum number of retry attempts for a failed database connection
    maxRetry: 10

  local-cache.yml: |
    user:
      topic: DELETE_CACHE_USER
      slotNum: 100
```

```
    slotSize: 2000
    successExpire: 300
    failedExpire: 5
  group:
    topic: DELETE_CACHE_GROUP
    slotNum: 100
    slotSize: 2000
    successExpire: 300
    failedExpire: 5
  friend:
    topic: DELETE_CACHE_FRIEND
    slotNum: 100
    slotSize: 2000
    successExpire: 300
    failedExpire: 5
  conversation:
    topic: DELETE_CACHE_CONVERSATION
    slotNum: 100
    slotSize: 2000
    successExpire: 300
    failedExpire: 5

openim-api.yml: |
  api:
    # Listening IP; 0.0.0.0 means both internal and external IPs are listened to, default is recommended
    listenIP: 0.0.0.0
    # Listening ports; if multiple are configured, multiple instances will be launched, must be consistent with th
    ports: [ 10002 ]
    # API compression level; 0: default compression, 1: best compression, 2: best speed, -1: no compression
    compressionLevel: 0

  prometheus:
    # Whether to enable prometheus
    enable: true
    # Prometheus listening ports, must match the number of api.ports
    ports: [ 12002 ]
    # This address can be accessed via a browser
    grafanaURL: http://127.0.0.1:13000/

openim-rpc-user.yml: |
  rpc:
    # API or other RPCs can access this RPC through this IP; if left blank, the internal network IP is obtained by
    registerIP:
    # Listening IP; 0.0.0.0 means both internal and external IPs are listened to, if blank, the internal network I
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10320 ]
  prometheus:
    # Whether to enable prometheus
    enable: true
    # Prometheus listening ports, must be consistent with the number of rpc.ports
    ports: [ 12320 ]

openim-crontask.yml: |
  cronExecuteTime: 0 2 * * *
  retainChatRecords: 365
  fileExpireTime: 180
  deleteObjectType: ["msg-picture","msg-file", "msg-voice","msg-video","msg-video-snapshot","sdklog"]

openim-msggateway.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
```

```
    registerIP:
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10140 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12140 ]

  # IP address that the RPC/WebSocket service listens on; setting to 0.0.0.0 listens on both internal and external
  listenIP: 0.0.0.0

  longConnSvr:
    # WebSocket listening ports, must match the number of rpc.ports
    ports: [ 10001 ]
    # Maximum number of WebSocket connections
    websocketMaxConnNum: 100000
    # Maximum length of the entire WebSocket message packet
    websocketMaxMsgLen: 4096
    # WebSocket connection handshake timeout in seconds
    websocketTimeout: 10

openim-msgtransfer.yml: |
  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; each port corresponds to an instance of monitoring. Ensure these a
    # Because four instances have been launched, four ports need to be specified
    ports: [ 12020 ]

openim-push.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10170 ]


  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12170 ]

  maxConcurrentWorkers: 3
  #Use geTui for offline push notifications, or choose fcm or jpns; corresponding configuration settings must be s
  enable:
  geTui:
    pushUrl: https://restapi.getui.com/v2/$appId
    masterSecret:
    appKey:
    intent:
    channelID:
    channelName:
  fcm:
```

```
    # Prioritize using file paths. If the file path is empty, use URL
    filePath:   # File path is concatenated with the parameters passed in through - c(`mage` default pass in `conf
    authURL:   #  Must start with https or http.
  jpush:
    appKey:
    masterSecret:
    pushURL:
    pushIntent:

  # iOS system push sound and badge count
  iosPush:
    pushSound: xxx
    badgeCount: true
    production: false

  fullUserCache: true

openim-rpc-auth.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10200 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [12200]

  tokenPolicy:
    # Token validity period, in days
    expire: 90

openim-rpc-conversation.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10220 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12200 ]

  tokenPolicy:
    # Token validity period, in days
    expire: 90

openim-rpc-friend.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
```

```
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10240 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12240 ]

openim-rpc-group.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10260 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12260 ]

  enableHistoryForNewMembers: true

openim-rpc-msg.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    ports: [ 10280 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12280 ]


  # Does sending messages require friend verification
  friendVerify: false

openim-rpc-third.yml: |
  rpc:
    # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network I
    registerIP:
    # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If
    listenIP: 0.0.0.0
    # autoSetPorts indicates whether to automatically set the ports
```

```
    # if you use in kubernetes, set it to false
    autoSetPorts: false
    # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. Th
    # It will only take effect when autoSetPorts is set to false.
    ports: [ 10300 ]

  prometheus:
    # Enable or disable Prometheus monitoring
    enable: true
    # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitor
    ports: [ 12300 ]


  object:
    # Use MinIO as object storage, or set to "cos", "oss", "kodo", "aws", while also configuring the corresponding
    enable: minio
    cos:
      bucketURL: https://temp-1252357374.cos.ap-chengdu.myqcloud.com
      secretID:
      secretKey:
      sessionToken:
      publicRead: false
    oss:
      endpoint: https://oss-cn-chengdu.aliyuncs.com
      bucket: demo-9999999
      bucketURL: https://demo-9999999.oss-cn-chengdu.aliyuncs.com
      accessKeyID:
      accessKeySecret:
      sessionToken:
      publicRead: false
    kodo:
      endpoint: http://s3.cn-south-1.qiniucs.com
      bucket: kodo-bucket-test
      bucketURL: http://kodo-bucket-test-oetobfb.qiniudns.com
      accessKeyID:
      accessKeySecret:
      sessionToken:
      publicRead: false
    aws:
      region: ap-southeast-2
      bucket: testdemo832234
      accessKeyID:
      secretAccessKey:
      sessionToken:
      publicRead: false

share.yml: |
  secret: openIM123

  imAdminUserID: ["imAdmin"]

  # 1: For Android, iOS, Windows, Mac, and web platforms, only one instance can be online at a time
  multiLogin:
    policy: 1
    maxNumOneEnd: 30

kafka.yml: |
  # Username for authentication
  username: ''
  # Password for authentication
  password: ''
  # Producer acknowledgment settings
  producerAck:
  # Compression type to use (e.g., none, gzip, snappy)
  compressType: none
  # List of Kafka broker addresses
```

```
    address: [ "kafka-service:19094" ]
    # Kafka topic for Redis integration
    toRedisTopic: toRedis
    # Kafka topic for MongoDB integration
    toMongoTopic: toMongo
    # Kafka topic for push notifications
    toPushTopic: toPush
    # Kafka topic for offline push notifications
    toOfflinePushTopic: toOfflinePush
    # Consumer group ID for Redis topic
    toRedisGroupID: redis
    # Consumer group ID for MongoDB topic
    toMongoGroupID: mongo
    # Consumer group ID for push notifications topic
    toPushGroupID: push
    # Consumer group ID for offline push notifications topic
    toOfflinePushGroupID: offlinePush
    # TLS (Transport Layer Security) configuration
    tls:
      # Enable or disable TLS
      enableTLS: false
      # CA certificate file path
      caCrt:
      # Client certificate file path
      clientCrt:
      # Client key file path
      clientKey:
      # Client key password
      clientKeyPwd:
      # Whether to skip TLS verification (not recommended for production)
      insecureSkipVerify: false

  redis.yml: |
    address: [ "redis-service:16379" ]
    username:
    password:  # openIM123
    clusterMode: false
    db: 0
    maxRetry: 10
    poolSize: 100

  minio.yml: |
    # Name of the bucket in MinIO
    bucket: openim
    # Access key ID for MinIO authentication
    accessKeyID: root
    # Secret access key for MinIO authentication
    secretAccessKey:    # openIM123
    # Session token for MinIO authentication (optional)
    sessionToken:
    # Internal address of the MinIO server
    internalAddress: minio-service:10005
    # External address of the MinIO server, accessible from outside. Supports both HTTP and HTTPS using a domain nam
    externalAddress: http://minio-service:10005
    # Flag to enable or disable public read access to the bucket
    publicRead: "false"

  notification.yml: |
    groupCreated:
      isSendMsg: true
    # Reliability level of the message sending.
    # Set to 1 to send only when online, 2 for guaranteed delivery.
      reliabilityLevel: 1
    # This setting is effective only when 'isSendMsg' is true.
    # It controls whether to count unread messages.
      unreadCount: false
```

```yaml
# Configuration for offline push notifications.
  offlinePush:
    # Enables or disables offline push notifications.
    enable: false
    # Title for the notification when a group is created.
    title: create group title
    # Description for the notification.
    desc: create group desc
    # Additional information for the notification.
    ext: create group ext

groupInfoSet:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupInfoSet title
    desc: groupInfoSet desc
    ext: groupInfoSet ext

joinGroupApplication:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: joinGroupApplication title
    desc: joinGroupApplication desc
    ext: joinGroupApplication ext

memberQuit:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberQuit title
    desc: memberQuit desc
    ext: memberQuit ext

groupApplicationAccepted:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupApplicationAccepted title
    desc: groupApplicationAccepted desc
    ext: groupApplicationAccepted ext

groupApplicationRejected:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupApplicationRejected title
    desc: groupApplicationRejected desc
    ext: groupApplicationRejected ext

groupOwnerTransferred:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
```

```
    enable: false
    title: groupOwnerTransferred title
    desc: groupOwnerTransferred desc
    ext: groupOwnerTransferred ext

memberKicked:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberKicked title
    desc: memberKicked desc
    ext: memberKicked ext

memberInvited:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberInvited title
    desc: memberInvited desc
    ext: memberInvited ext

memberEnter:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberEnter title
    desc: memberEnter desc
    ext: memberEnter ext

groupDismissed:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupDismissed title
    desc: groupDismissed desc
    ext: groupDismissed ext

groupMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMuted title
    desc: groupMuted desc
    ext: groupMuted ext

groupCancelMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupCancelMuted title
    desc: groupCancelMuted desc
    ext: groupCancelMuted ext
  defaultTips:
    tips: group Cancel Muted
```

```
groupMemberMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMemberMuted title
    desc: groupMemberMuted desc
    ext: groupMemberMuted ext

groupMemberCancelMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMemberCancelMuted title
    desc: groupMemberCancelMuted desc
    ext: groupMemberCancelMuted ext

groupMemberInfoSet:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMemberInfoSet title
    desc: groupMemberInfoSet desc
    ext: groupMemberInfoSet ext

groupInfoSetAnnouncement:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupInfoSetAnnouncement title
    desc: groupInfoSetAnnouncement desc
    ext: groupInfoSetAnnouncement ext

groupInfoSetName:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupInfoSetName title
    desc: groupInfoSetName desc
    ext: groupInfoSetName ext

##############################friend###################################
friendApplicationAdded:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: Somebody applies to add you as a friend
    desc: Somebody applies to add you as a friend
    ext: Somebody applies to add you as a friend

friendApplicationApproved:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
```

```
  offlinePush:
    enable: true
    title: Someone applies to add your friend application
    desc: Someone applies to add your friend application
    ext: Someone applies to add your friend application

friendApplicationRejected:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: Someone rejected your friend application
    desc: Someone rejected your friend application
    ext: Someone rejected your friend application

friendAdded:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: We have become friends
    desc: We have become friends
    ext: We have become friends

friendDeleted:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: deleted a friend
    desc: deleted a friend
    ext: deleted a friend

friendRemarkSet:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: Your friend's profile has been changed
    desc: Your friend's profile has been changed
    ext: Your friend's profile has been changed

blackAdded:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: blocked a user
    desc: blocked a user
    ext: blocked a user

blackDeleted:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: Remove a blocked user
    desc: Remove a blocked user
    ext: Remove a blocked user
```

```
  friendInfoUpdated:
    isSendMsg: false
    reliabilityLevel: 1
    unreadCount: false
    offlinePush:
      enable: true
      title: friend info updated
      desc: friend info updated
      ext: friend info updated

  ######################user##########################
  userInfoUpdated:
    isSendMsg: false
    reliabilityLevel: 1
    unreadCount: false
    offlinePush:
      enable: true
      title: userInfo updated
      desc: userInfo updated
      ext: userInfo updated

  userStatusChanged:
    isSendMsg: false
    reliabilityLevel: 1
    unreadCount: false
    offlinePush:
      enable: false
      title: user status changed
      desc: user status changed
      ext: user status changed

  ######################conversation##########################
  conversationChanged:
    isSendMsg: false
    reliabilityLevel: 1
    unreadCount: false
    offlinePush:
      enable: true
      title: conversation changed
      desc: conversation changed
      ext: conversation changed

  conversationSetPrivate:
    isSendMsg: true
    reliabilityLevel: 1
    unreadCount: false
    offlinePush:
      enable: true
      title: burn after reading
      desc: burn after reading
      ext: burn after reading

webhooks.yml: |
  url: http://127.0.0.1:10006/callbackExample
  beforeSendSingleMsg:
    enable: false
    timeout: 5
    failedContinue: true
    # Only the contentType in allowedTypes will send the callback.
    # Supports two formats: a single type or a range. The range is defined by the lower and upper bounds connected
    # e.g. allowedTypes: [1, 100, 200-500, 600-700] means that only contentType within the range
    # {1, 100} ∪ [200, 500] ∪ [600, 700] will be allowed through the filter.
    # If not set, all contentType messages will through this filter.
    allowedTypes: []
    # Only the contentType not in deniedTypes will send the callback.
    # Supports two formats, same as allowedTypes.
```

```
  # If not set, all contentType messages will through this filter.
  deniedTypes: []
beforeUpdateUserInfoEx:
  enable:  false
  timeout: 5
  failedContinue: true
afterUpdateUserInfoEx:
  enable:  false
  timeout: 5
afterSendSingleMsg:
  enable: false
  timeout: 5
  # Only the senID/recvID specified in attentionIds will send the callback
  # if not set, all user messages will be callback
  attentionIds: []
  # See beforeSendSingleMsg comment.
  allowedTypes: []
  deniedTypes: []
beforeSendGroupMsg:
  enable: false
  timeout: 5
  failedContinue: true
  # See beforeSendSingleMsg comment.
  allowedTypes: []
  deniedTypes: []
beforeMsgModify:
  enable: false
  timeout: 5
  failedContinue: true
  # See beforeSendSingleMsg comment.
  allowedTypes: []
  deniedTypes: []
afterSendGroupMsg:
  enable: false
  timeout: 5
  # See beforeSendSingleMsg comment.
  allowedTypes: []
  deniedTypes: []
afterUserOnline:
  enable: false
  timeout: 5
afterUserOffline:
  enable: false
  timeout: 5
afterUserKickOff:
  enable: false
  timeout: 5
beforeOfflinePush:
  enable: false
  timeout: 5
  failedContinue: true
beforeOnlinePush:
  enable: false
  timeout: 5
  failedContinue: true
beforeGroupOnlinePush:
  enable: false
  timeout: 5
  failedContinue: true
beforeAddFriend:
  enable: false
  timeout: 5
  failedContinue: true
beforeUpdateUserInfo:
  enable: false
  timeout: 5
```

```
  failedContinue: true
afterUpdateUserInfo:
  enable: false
  timeout: 5
beforeCreateGroup:
  enable: false
  timeout: 5
  failedContinue: true
afterCreateGroup:
  enable: false
  timeout: 5
beforeMemberJoinGroup:
  enable: false
  timeout: 5
  failedContinue: true
beforeSetGroupMemberInfo:
  enable: false
  timeout: 5
  failedContinue: true
afterSetGroupMemberInfo:
  enable: false
  timeout: 5
afterQuitGroup:
  enable: false
  timeout: 5
afterKickGroupMember:
  enable: false
  timeout: 5
afterDismissGroup:
  enable: false
  timeout: 5
beforeApplyJoinGroup:
  enable: false
  timeout: 5
  failedContinue: true
afterGroupMsgRead:
  enable: false
  timeout: 5
afterSingleMsgRead:
  enable: false
  timeout: 5
beforeUserRegister:
  enable: false
  timeout: 5
  failedContinue: true
afterUserRegister:
  enable: false
  timeout: 5
afterTransferGroupOwner:
  enable: false
  timeout: 5
beforeSetFriendRemark:
  enable: false
  timeout: 5
  failedContinue: true
afterSetFriendRemark:
  enable: false
  timeout: 5
afterGroupMsgRevoke:
  enable: false
  timeout: 5
afterJoinGroup:
  enable: false
  timeout: 5
beforeInviteUserToGroup:
  enable: false
```

```
    timeout: 5
    failedContinue: true
  afterSetGroupInfo:
    enable: false
    timeout: 5
  beforeSetGroupInfo:
    enable: false
    timeout: 5
    failedContinue: true
  afterSetGroupInfoEx:
    enable: false
    timeout: 5
  beforeSetGroupInfoEx:
    enable: false
    timeout: 5
    failedContinue: true
  afterRevokeMsg:
    enable: false
    timeout: 5
  beforeAddBlack:
    enable: false
    timeout: 5
    failedContinue:
  afterAddFriend:
    enable: false
    timeout: 5
  beforeAddFriendAgree:
    enable: false
    timeout: 5
    failedContinue: true
  afterAddFriendAgree:
    enable: false
    timeout: 5
  afterDeleteFriend:
    enable: false
    timeout: 5
  beforeImportFriends:
    enable: false
    timeout: 5
    failedContinue: true
  afterImportFriends:
    enable: false
    timeout: 5
  afterRemoveBlack:
    enable: false
    timeout: 5

prometheus.yml: |
  # my global config
  global:
    scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
    evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
    # scrape_timeout is set to the global default (10s).

  # Alertmanager configuration
  alerting:
    alertmanagers:
      - static_configs:
          - targets: [internal_ip:19093]

  # Load rules once and periodically evaluate them according to the global evaluation_interval.
  rule_files:
    - instance-down-rules.yml
  # - first_rules.yml
  # - second_rules.yml
```

```
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label "job=job_name" to any timeseries scraped from this config.
  # Monitored information captured by prometheus

  # prometheus fetches application services
  - job_name: node_exporter
    static_configs:
      - targets: [ internal_ip:20500 ]
  - job_name: openimserver-openim-api
    static_configs:
      - targets: [ internal_ip:12002 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-msggateway
    static_configs:
      - targets: [ internal_ip:12140 ]
#      - targets: [ internal_ip:12140, internal_ip:12141, internal_ip:12142, internal_ip:12143, internal_ip:1214
        labels:
          namespace: default
  - job_name: openimserver-openim-msgtransfer
    static_configs:
      - targets: [ internal_ip:12020, internal_ip:12021, internal_ip:12022, internal_ip:12023, internal_ip:12024
#      - targets: [ internal_ip:12020, internal_ip:12021, internal_ip:12022, internal_ip:12023, internal_ip:1202
        labels:
          namespace: default
  - job_name: openimserver-openim-push
    static_configs:
      - targets: [ internal_ip:12170, internal_ip:12171, internal_ip:12172, internal_ip:12173, internal_ip:12174
#      - targets: [ internal_ip:12170, internal_ip:12171, internal_ip:12172, internal_ip:12173, internal_ip:1217
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-auth
    static_configs:
      - targets: [ internal_ip:12200 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-conversation
    static_configs:
      - targets: [ internal_ip:12220 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-friend
    static_configs:
      - targets: [ internal_ip:12240 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-group
    static_configs:
      - targets: [ internal_ip:12260 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-msg
    static_configs:
      - targets: [ internal_ip:12280 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-third
    static_configs:
      - targets: [ internal_ip:12300 ]
        labels:
          namespace: default
  - job_name: openimserver-openim-rpc-user
    static_configs:
      - targets: [ internal_ip:12320 ]
```

```
labels:
  namespace: default
```

deployments/deploy/openim-crontask-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: openim-crontask
spec:
  replicas: 2
  selector:
    matchLabels:
      app: crontask
  template:
    metadata:
      labels:
        app: crontask
    spec:
      containers:
        - name: crontask-container
          image: openim/openim-crontask:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-msggateway-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: messagegateway-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: messagegateway-rpc-server
  template:
    metadata:
      labels:
        app: messagegateway-rpc-server
    spec:
      containers:
        - name: openim-msggateway-container
          image: openim/openim-msggateway:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10140
            - containerPort: 12001
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-msggateway-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: messagegateway-rpc-service
spec:
  selector:
    app: messagegateway-rpc-server
  ports:
    - name: longConnServer-10001
      protocol: TCP
      port: 10001
      targetPort: 10001
    - name: grpc-10140
      protocol: TCP
      port: 10140
      targetPort: 10140
    - name: prometheus-12001
      protocol: TCP
      port: 12001
      targetPort: 12001
  type: NodePort
```

deployments/deploy/openim-msgtransfer-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: openim-msgtransfer-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: openim-msgtransfer-server
  template:
    metadata:
      labels:
        app: openim-msgtransfer-server
    spec:
      containers:
        - name: openim-msgtransfer-container
          image: openim/openim-msgtransfer:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
            - name: IMENV_KAFKA_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-kafka-secret
                  key: kafka-password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 12020
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-msgtransfer-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: openim-msgtransfer-service
spec:
  selector:
    app: openim-msgtransfer-server
  ports:
    - name: prometheus-12020
      protocol: TCP
      port: 12020
      targetPort: 12020
  type: ClusterIP
```

deployments/deploy/openim-push-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: push-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: push-rpc-server
  template:
    metadata:
      labels:
        app: push-rpc-server
    spec:
      containers:
        - name: push-rpc-server-container
          image: openim/openim-push:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_KAFKA_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-kafka-secret
                  key: kafka-password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10170
            - containerPort: 12170
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-push-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: push-rpc-service
spec:
  selector:
    app: push-rpc-server
  ports:
    - name: http-10170
      protocol: TCP
      port: 10170
      targetPort: 10170
    - name: prometheus-12170
      protocol: TCP
      port: 12170
      targetPort: 12170
  type: ClusterIP
```

deployments/deploy/openim-rpc-auth-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: auth-rpc-server
  template:
    metadata:
      labels:
        app: auth-rpc-server
    spec:
      containers:
        - name: auth-rpc-server-container
          image: openim/openim-rpc-auth:v3.8.3
          imagePullPolicy: Never
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10200
            - containerPort: 12200
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-auth-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: auth-rpc-service
spec:
  selector:
    app: auth-rpc-server
  ports:
    - name: http-10200
      protocol: TCP
      port: 10200
      targetPort: 10200
    - name: prometheus-12200
      protocol: TCP
      port: 12200
      targetPort: 12200
  type: ClusterIP
```

deployments/deploy/openim-rpc-conversation-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: conversation-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: conversation-rpc-server
  template:
    metadata:
      labels:
        app: conversation-rpc-server
    spec:
      containers:
        - name: conversation-rpc-server-container
          image: openim/openim-rpc-conversation:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10220
            - containerPort: 12220
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-conversation-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: conversation-rpc-service
spec:
  selector:
    app: conversation-rpc-server
  ports:
    - name: http-10220
      protocol: TCP
      port: 10220
      targetPort: 10220
    - name: prometheus-12220
      protocol: TCP
      port: 12220
      targetPort: 12220
  type: ClusterIP
```

deployments/deploy/openim-rpc-friend-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: friend-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: friend-rpc-server
  template:
    metadata:
      labels:
        app: friend-rpc-server
    spec:
      containers:
        - name: friend-rpc-server-container
          image: openim/openim-rpc-friend:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10240
            - containerPort: 12240
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-friend-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: friend-rpc-service
spec:
  selector:
    app: friend-rpc-server
  ports:
    - name: http-10240
      protocol: TCP
      port: 10240
      targetPort: 10240
    - name: prometheus-12240
      protocol: TCP
      port: 12240
      targetPort: 12240
  type: ClusterIP
```

deployments/deploy/openim-rpc-group-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: group-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: group-rpc-server
  template:
    metadata:
      labels:
        app: group-rpc-server
    spec:
      containers:
        - name: group-rpc-server-container
          image: openim/openim-rpc-group:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10260
            - containerPort: 12260
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-group-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: group-rpc-service
spec:
  selector:
    app: group-rpc-server
  ports:
    - name: http-10260
      protocol: TCP
      port: 10260
      targetPort: 10260
    - name: prometheus-12260
      protocol: TCP
      port: 12260
      targetPort: 12260
  type: ClusterIP
```

deployments/deploy/openim-rpc-msg-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: msg-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: msg-rpc-server
  template:
    metadata:
      labels:
        app: msg-rpc-server
    spec:
      containers:
        - name: msg-rpc-server-container
          image: openim/openim-rpc-msg:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
            - name: IMENV_KAFKA_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-kafka-secret
                  key: kafka-password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10280
            - containerPort: 12280
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-msg-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: msg-rpc-service
spec:
  selector:
    app: msg-rpc-server
  ports:
    - name: http-10280
      protocol: TCP
      port: 10280
      targetPort: 10280
    - name: prometheus-12280
      protocol: TCP
      port: 12280
      targetPort: 12280
  type: ClusterIP
```

deployments/deploy/openim-rpc-third-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: third-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: third-rpc-server
  template:
    metadata:
      labels:
        app: third-rpc-server
    spec:
      containers:
        - name: third-rpc-server-container
          image: openim/openim-rpc-third:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_MINIO_ACCESSKEYID
              valueFrom:
                secretKeyRef:
                  name: openim-minio-secret
                  key: minio-root-user
            - name: IMENV_MINIO_SECRETACCESSKEY
              valueFrom:
                secretKeyRef:
                  name: openim-minio-secret
                  key: minio-root-password
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10300
            - containerPort: 12300
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-third-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: third-rpc-service
spec:
  selector:
    app: third-rpc-server
  ports:
    - name: http-10300
      protocol: TCP
      port: 10300
      targetPort: 10300
    - name: prometheus-12300
      protocol: TCP
      port: 12300
      targetPort: 12300
  type: ClusterIP
```

deployments/deploy/openim-rpc-user-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-rpc-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: user-rpc-server
  template:
    metadata:
      labels:
        app: user-rpc-server
    spec:
      containers:
        - name: user-rpc-server-container
          image: openim/openim-rpc-user:v3.8.3
          env:
            - name: CONFIG_PATH
              value: "/config"
            - name: IMENV_REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-redis-secret
                  key: redis-password
            - name: IMENV_MONGODB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_username
            - name: IMENV_MONGODB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-mongo-secret
                  key: mongo_openim_password
            - name: IMENV_KAFKA_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: openim-kafka-secret
                  key: kafka-password
          volumeMounts:
            - name: openim-config
              mountPath: "/config"
              readOnly: true
          ports:
            - containerPort: 10320
            - containerPort: 12320
      volumes:
        - name: openim-config
          configMap:
            name: openim-config
```

deployments/deploy/openim-rpc-user-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: user-rpc-service
spec:
  selector:
    app: user-rpc-server
  ports:
    - name: http-10320
      protocol: TCP
      port: 10320
      targetPort: 10320
    - name: prometheus-12320
      protocol: TCP
      port: 12320
      targetPort: 12320
  type: ClusterIP
```

deployments/deploy/redis-secret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: openim-redis-secret
type: Opaque
data:
  redis-password: b3BlbklNMTIz # "openIM123" in base64
```

deployments/deploy/redis-service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: redis-service
  labels:
    app: redis
spec:
  type: ClusterIP
  selector:
    app: redis
  ports:
    - name: redis-port
      protocol: TCP
      port: 16379
      targetPort: 6379
```

deployments/deploy/redis-statefulset.yml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis-statefulset
spec:
  serviceName: "redis"
  replicas: 2
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis:7.0.0
          ports:
            - containerPort: 6379
          env:
            - name: TZ
              value: "Asia/Shanghai"
            - name: REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: redis-secret
                  key: redis-password
          volumeMounts:
            - name: redis-data
              mountPath: /data
          command:
            [
              "/bin/sh",
              "-c",
              'redis-server  --requirepass "$REDIS_PASSWORD" --appendonly yes',
            ]
      volumes:
        - name: redis-config-volume
          configMap:
            name: openim-config
        - name: redis-data
          persistentVolumeClaim:
            claimName: redis-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: redis-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

.github

.github/.codecov.yml

```yaml
coverage:
  status:
    project:
      default: false # disable the default status that measures entire project
      pkg: # declare a new status context "pkg"
        paths:
          - pkg/* # only include coverage in "pkg/" folder
        informational: true # Always pass check
      tools: # declare a new status context "tools"
        paths:
          - tools/* # only include coverage in "tools/" folder
        informational: true # Always pass check
      test: # declare a new status context "test"
        paths:
          - test/* # only include coverage in "test/" folder
        informational: true # Always pass check

      # internal: # declare a new status context "internal"
      #   paths:
      #     - internal/* # only include coverage in "internal/" folder
      #   informational: true # Always pass check
      # cmd: # declare a new status context "cmd"
      #   paths:
      #     - cmd/* # only include coverage in "cmd/" folder
      #   informational: true # Always pass check
    patch: off # disable the commit only checks
```

.github/sync-release.yml

```yaml
openimsdk/openim-docker:
  - source: ./config
    dest: ./openim-server/release/config
    replace: true
  - source: ./docs
    dest: ./openim-server/release/docs
    replace: true
  - source: ./scripts
    dest: ./openim-server/release/scripts
    replace: true
  - source: ./scripts
    dest: ./scripts
    replace: false
  - source: ./Makefile
    dest: ./Makefile
    replace: false
```

.github/workflows

.github/workflows/auto-assign-issue.yml

```yaml
name: Assign issue to comment author
on:
  issue_comment:
    types: [created]
jobs:
  assign-issue:
    if: |
      contains(github.event.comment.body, '/assign') || contains(github.event.comment.body, '/accept') &&
      !contains(github.event.comment.user.login, 'openim-robot')
    runs-on: ubuntu-latest
    permissions:
      issues: write
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Assign the issue
        run: |
          export LETASE_MILESTONES=$(curl 'https://api.github.com/repos/$OWNER/$PEPO/milestones' | jq -r 'last(.[]).
          gh issue edit ${{ github.event.issue.number }} --add-assignee "${{ github.event.comment.user.login }}"
          gh issue edit ${{ github.event.issue.number }} --add-label "accepted"
          gh issue comment $ISSUE --body "@${{ github.event.comment.user.login }} Glad to see you accepted this issu

        # gh issue edit ${{ github.event.issue.number }} --milestone "$LETASE_MILESTONES"
        env:
          GH_TOKEN: ${{ secrets.BOT_TOKEN }}
          ISSUE: ${{ github.event.issue.html_url }}
          OWNER: ${{ github.repository_owner }}
          REPO: ${{ github.event.repository.name }}
```

.github/workflows/auto-invite-comment.yml

```yaml
name: Invite users to join OpenIM Community.
on:
  issue_comment:
    types:
      - created
jobs:
  issue_comment:
    name: Invite users to join OpenIM Community
    if: ${{ github.event.comment.body == '/invite' || github.event.comment.body == '/close' || github.event.comment.
    runs-on: ubuntu-latest
    permissions:
      issues: write
    steps:
      - name: Invite user to join OpenIM Community
        uses: peter-evans/create-or-update-comment@v4
        with:
          token: ${{ secrets.BOT_GITHUB_TOKEN }}
          issue-number: ${{ github.event.issue.number }}
          body: |
            We value close connections with our users, developers, and contributors here at Open-IM-Server. With a l

            Our most recommended way to get in touch is through [Slack](https://join.slack.com/t/openimsdk/shared_in

            In addition to Slack, we also offer the following ways to get in touch:

            + <a href="https://join.slack.com/t/openimsdk/shared_invite/zt-2ijy1ys1f-O0aEDCr7ExRZ7mwsHAVg9A" target=
            + <a href="https://mail.google.com/mail/u/0/?fs=1&tf=cm&to=info@openim.io" target="_blank"><img src="htt
            + <a href="https://doc.rentsoft.cn/" target="_blank"><img src="https://img.shields.io/badge/%E5%8D%9A%E5
            + <a href="https://github.com/OpenIMSDK/OpenIM-Docs/blob/main/docs/images/WechatIMG20.jpeg" target="_bla

      # - name: Close Issue
      #   uses: peter-evans/close-issue@v3
      #   with:
      #     token: ${{ secrets.BOT_GITHUB_TOKEN }}
      #     issue-number: ${{ github.event.issue.number }}
      #     comment: ■ Auto-closing issue, if you still need help please reopen the issue or ask for help in the com
      #     labels: |
      #       accepted
```

.github/workflows/changelog.yml

```yaml
name: Release Changelog

on:
  release:
    types: [released]

permissions:
  contents: write
  pull-requests: write

jobs:
  update-changelog:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Run Go Changelog Generator
      run: |
        # Run the Go changelog generator, passing the release tag if available
        if [ "${{ github.event.release.tag_name }}" = "latest" ]; then
          go run tools/changelog/changelog.go > "${{ github.event.release.tag_name }}-changelog.md"
        else
          go run tools/changelog/changelog.go "${{ github.event.release.tag_name }}" > "${{ github.event.release.tag
        fi

    - name: Handle changelog files
      run: |
        # Ensure that the CHANGELOG directory exists
        mkdir -p CHANGELOG

        # Extract Major.Minor version by removing the 'v' prefix from the tag name
        TAG_NAME=${{ github.event.release.tag_name }}
        CHANGELOG_VERSION_NUMBER=$(echo "$TAG_NAME" | sed 's/^v//' | grep -oP '^\d+\.\d+')

        # Define the new changelog file path
        CHANGELOG_FILENAME="CHANGELOG-$CHANGELOG_VERSION_NUMBER.md"
        CHANGELOG_PATH="CHANGELOG/$CHANGELOG_FILENAME"

        # Check if the changelog file for the current release already exists
        if [ -f "$CHANGELOG_PATH" ]; then
          # If the file exists, append the new changelog to the existing one
          cat "$CHANGELOG_PATH" >> "${TAG_NAME}-changelog.md"
          # Overwrite the existing changelog with the updated content
          mv "${TAG_NAME}-changelog.md" "$CHANGELOG_PATH"
        else
          # If the changelog file doesn't exist, rename the temp changelog file to the new changelog file
          mv "${TAG_NAME}-changelog.md" "$CHANGELOG_PATH"

          # Ensure that README.md exists
          if [ ! -f "CHANGELOG/README.md" ]; then
            echo -e "# CHANGELOGs\n\n" > CHANGELOG/README.md
          fi

            # Add the new changelog entry at the top of the README.md
            if ! grep -q "\[$CHANGELOG_FILENAME\]" CHANGELOG/README.md; then
            sed -i "3i- [$CHANGELOG_FILENAME](./$CHANGELOG_FILENAME)" CHANGELOG/README.md
            # Remove the extra newline character added by sed
            # sed -i '4d' CHANGELOG/README.md
            fi
        fi

    - name: Clean up
      run: |
```

```
    # Remove any temporary files that were created during the process
    rm -f "${{ github.event.release.tag_name }}-changelog.md"

- name: Create Pull Request
  uses: peter-evans/create-pull-request@v7.0.5
  with:
    token: ${{ secrets.GITHUB_TOKEN }}
    commit-message: "Update CHANGELOG for release ${{ github.event.release.tag_name }}"
    title: "Update CHANGELOG for release ${{ github.event.release.tag_name }}"
    body: "This PR updates the CHANGELOG files for release ${{ github.event.release.tag_name }}"
    branch: changelog-${{ github.event.release.tag_name }}
    base: main
    delete-branch: true
    labels: changelog
```

.github/workflows/cla-assistant.yml

```yaml
name: CLA Assistant
on:
  issue_comment:
    types: [created]
  pull_request_target:
    types: [opened,closed,synchronize]

# explicitly configure permissions, in case your GITHUB_TOKEN workflow permissions are set to read-only in repositor
permissions:
  actions: write
  contents: write # this can be 'read' if the signatures are in remote repository
  pull-requests: write
  statuses: write

jobs:
  CLA-Assistant:
    runs-on: ubuntu-latest
    steps:
      - name: "CLA Assistant"
        if: (github.event.comment.body == 'recheck' || github.event.comment.body == 'I have read the CLA Document an
        uses: contributor-assistant/github-action@v2.4.0
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          PERSONAL_ACCESS_TOKEN: ${{ secrets.BOT_TOKEN }}
        with:
          path-to-signatures: 'signatures/cla.json'
          path-to-document: 'https://github.com/OpenIM-Robot/cla/blob/main/README.md' # e.g. a CLA or a DCO document
          branch: 'main'
          allowlist: 'bot*,*bot,OpenIM-Robot'

         # the followings are the optional inputs - If the optional inputs are not given, then default values will b
          remote-organization-name: OpenIM-Robot
          remote-repository-name: cla
          create-file-commit-message: 'Creating file for storing CLA Signatures'
          # signed-commit-message: '$contributorName has signed the CLA in $owner/$repo#$pullRequestNo'
          custom-notsigned-prcomment: '■ Thank you for your contribution and please kindly read and sign our CLA. [C
          custom-pr-sign-comment: 'I have read the CLA Document and I hereby sign the CLA'
          custom-allsigned-prcomment: '■ All Contributors have signed the [CLA](https://github.com/OpenIM-Robot/cla/
          #lock-pullrequest-aftermerge: false - if you don't want this bot to automatically lock the pull request af
          #use-dco-flag: true - If you are using DCO instead of CLA
```

.github/workflows/cleanup-after-milestone-prs-merged.yml

```yaml
name: Cleanup After Milestone PRs Merged

on:
  pull_request:
    types:
      - closed

jobs:
  handle_pr:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v4.2.0

    - name: Get the PR title and extract PR numbers
      id: extract_pr_numbers
      run: |
        # Get the PR title
        PR_TITLE="${{ github.event.pull_request.title }}"

        echo "PR Title: $PR_TITLE"

        # Extract PR numbers from the title
        PR_NUMBERS=$(echo "$PR_TITLE" | grep -oE "#[0-9]+" | tr -d '#' | tr '\n' ' ')
        echo "Extracted PR Numbers: $PR_NUMBERS"

        # Save PR numbers to a file
        echo "$PR_NUMBERS" > pr_numbers.txt
        echo "Saved PR Numbers to pr_numbers.txt"

        # Check if the title matches a specific pattern
        if echo "$PR_TITLE" | grep -qE "^deps: Merge( #[0-9]+)+ PRs into .+"; then
          echo "proceed=true" >> $GITHUB_OUTPUT
        else
          echo "proceed=false" >> $GITHUB_OUTPUT
        fi

    - name: Use extracted PR numbers and label PRs
      if: (steps.extract_pr_numbers.outputs.proceed == 'true' || contains(github.event.pull_request.labels.*.name, '
      run: |
        # Read the previously saved PR numbers
        PR_NUMBERS=$(cat pr_numbers.txt)
        echo "Using extracted PR Numbers: $PR_NUMBERS"

        # Loop through each PR number and add label
        for PR_NUMBER in $PR_NUMBERS; do
          echo "Adding 'cherry-picked' label to PR #$PR_NUMBER"
          curl -X POST \
            -H "Authorization: token ${{ secrets.GITHUB_TOKEN }}" \
            -H "Accept: application/vnd.github+json" \
            https://api.github.com/repos/${{ github.repository }}/issues/$PR_NUMBER/labels \
            -d '{"labels":["cherry-picked"]}'
        done
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

    - name: Delete branch after PR close
      if: steps.extract_pr_numbers.outputs.proceed == 'true' || contains(github.event.pull_request.labels.*.name, 'm
      run: |
        BRANCH_NAME="${{ github.event.pull_request.head.ref }}"
        echo "Branch to delete: $BRANCH_NAME"
        git push origin --delete "$BRANCH_NAME"
      env:
```

```
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

.github/workflows/codeql-analysis.yml

```yaml
# For most projects, this workflow file will not need changing; you simply need
# to commit it to your repository.
#
# You may wish to alter this file to override the set of languages analyzed,
# or to provide custom queries or build logic.
#
# ******** NOTE ********
# We have attempted to detect the languages in your repository. Please check
# the `language` matrix defined below to confirm you have the correct set of
# supported CodeQL languages.

name: "CodeQL"

on:
  push:
    branches: [ main ]
  pull_request:
    # The branches below must be a subset of the branches above
    branches: [ main ]
  schedule:
    - cron: '18 19 * * 6'

jobs:
  analyze:
    name: Analyze
    runs-on: ubuntu-latest

    strategy:
      fail-fast: false
      matrix:
        language: [ 'go' ]
        # CodeQL supports [ 'cpp', 'csharp', 'go', 'java', 'javascript', 'python' ]
        # Learn more:
        # https://docs.github.com/en/free-pro-team@latest/github/finding-security-vulnerabilities-and-errors-in-your

    steps:
    - name: Checkout repository
      uses: actions/checkout@v4

    # Initializes the CodeQL tools for scanning.
    - name: Initialize CodeQL
      uses: github/codeql-action/init@v3
      with:
        languages: ${{ matrix.language }}
        # If you wish to specify custom queries, you can do so here or in a config file.
        # By default, queries listed here will override any specified in a config file.
        # Prefix the list here with "+" to use these queries and those in the config file.
        # queries: ./path/to/local/query, your-org/your-repo/queries@main

    # Autobuild attempts to build any compiled languages  (C/C++, C#, or Java).
    # If this step fails, then you should remove it and run the build manually (see below)
    - name: Autobuild
      uses: github/codeql-action/autobuild@v3

    # ■■ Command-line programs to run using the OS shell.
    # ■ https://git.io/JvXDl

    # ═■ If the Autobuild fails above, remove it and uncomment the following three lines
    #    and modify them (or add more) to build your code if your project
    #    uses a compiled language

    #- run: |
    #   make bootstrap
    #   make release
```

```
- name: Perform CodeQL Analysis
  uses: github/codeql-action/analyze@v3
```

.github/workflows/comment-check.yml

```yaml
name: Non-English Comments Check

on:
  pull_request:
    branches:
      - main
  workflow_dispatch:

jobs:
  non-english-comments-check:
    runs-on: ubuntu-latest

    env:
      # need ignore Dirs
      EXCLUDE_DIRS: ".git docs tests scripts assets node_modules build"
      # need ignore Files
      EXCLUDE_FILES: "*.md *.txt *.html *.css *.min.js *.mdx"

    steps:
      - uses: actions/checkout@v4

      - name: Search for Non-English comments
        run: |
          set -e
          # Define the regex pattern to match Chinese characters
          pattern='[\p{Han}]'

          # Process the directories to be excluded
          exclude_dirs=""
          for dir in $EXCLUDE_DIRS; do
            exclude_dirs="$exclude_dirs --exclude-dir=$dir"
          done

          # Process the file types to be excluded
          exclude_files=""
          for file in $EXCLUDE_FILES; do
            exclude_files="$exclude_files --exclude=$file"
          done

          # Use grep to find all comments containing Non-English characters and save to file
          grep -Pnr "$pattern" . $exclude_dirs $exclude_files > non_english_comments.txt || true

      - name: Output non-English comments are found
        run: |
          if [ -s non_english_comments.txt ]; then
            echo "Non-English comments found in the following locations:"
            cat non_english_comments.txt
            exit 1  # terminate the workflow
          else
            echo "No Non_English comments found."
          fi
```

.github/workflows/docker-build-and-release-services-images.yml

```yaml
name: Build and release services Docker Images

on:
  push:
    branches:
      - release-*
  release:
    types: [published]
  workflow_dispatch:
    inputs:
      tag:
        description: "Tag version to be used for Docker image"
        required: true
        default: "v3.8.3"

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3.8.0

      - name: Log in to Docker Hub
        uses: docker/login-action@v3.3.0
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v3.3.0
        with:
          registry: ghcr.io
          username: ${{ github.repository_owner }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Log in to Aliyun Container Registry
        uses: docker/login-action@v3.3.0
        with:
          registry: registry.cn-hangzhou.aliyuncs.com
          username: ${{ secrets.ALIREGISTRY_USERNAME }}
          password: ${{ secrets.ALIREGISTRY_TOKEN }}

      - name: Extract metadata for Docker (tags, labels)
        id: meta
        uses: docker/metadata-action@v5.6.0
        with:
          tags: |
            type=ref,event=tag
            type=schedule
            type=ref,event=branch
            type=semver,pattern={{version}}
            type=semver,pattern=v{{version}}
            type=semver,pattern=release-{{raw}}
            type=sha
            type=raw,value=${{ github.event.inputs.tag }}

      - name: Build and push Docker images
        run: |
          IMG_DIR="build/images"
          for dir in "$IMG_DIR"/*/; do
```

```
    # Find Dockerfile or *.dockerfile in a case-insensitive manner
    dockerfile=$(find "$dir" -maxdepth 1 -type f \( -iname 'dockerfile' -o -iname '*.dockerfile' \) | head

    if [ -n "$dockerfile" ] && [ -f "$dockerfile" ]; then
        IMAGE_NAME=$(basename "$dir")
        echo "Building Docker image for $IMAGE_NAME with tags:"

        # Initialize tag arguments
        tag_args=()

        # Read each tag and append --tag arguments
        while IFS= read -r tag; do
            tag_args+=(--tag "${{ secrets.DOCKER_USERNAME }}/$IMAGE_NAME:$tag")
            tag_args+=(--tag "ghcr.io/${{ github.repository_owner }}/$IMAGE_NAME:$tag")
            tag_args+=(--tag "registry.cn-hangzhou.aliyuncs.com/openimsdk/$IMAGE_NAME:$tag")
        done <<< "${{ steps.meta.outputs.tags }}"

        # Build and push the Docker image with all tags
        docker buildx build --platform linux/amd64,linux/arm64 \
          --file "$dockerfile" \
          "${tag_args[@]}" \
          --push \
          "."
    else
        echo "No valid Dockerfile found in $dir"
    fi
done
```

.github/workflows/go-build-test.yml

```yaml
name: Go Build Test

on:
  push:
  pull_request:
    paths-ignore:
      - "**/*.md"

  workflow_dispatch:

jobs:
  go-build:
    name: Test with go ${{ matrix.go_version }} on ${{ matrix.os }}
    runs-on: ${{ matrix.os }}

    env:
      SHARE_CONFIG_PATH: config/share.yml

    permissions:
      contents: write
      pull-requests: write
    strategy:
      matrix:
        os: [ubuntu-latest]
        go_version: ["1.22.x"]

    steps:
      - name: Checkout Server repository
        uses: actions/checkout@v4

      - name: Set up Go ${{ matrix.go_version }}
        uses: actions/setup-go@v5
        with:
          go-version: ${{ matrix.go_version }}

      - name: Get Server dependencies
        run: |
          go install github.com/magefile/mage@latest
          go mod tidy
          go mod download

      - name: Set up infra services
        uses: hoverkraft-tech/compose-action@v2.0.1
        with:
          compose-file: "./docker-compose.yml"

      - name: Modify Server Configuration
        run: |
          yq e '.secret = 123456' -i ${{ env.SHARE_CONFIG_PATH }}

      # - name: Get Internal IP Address
      #   id: get-ip
      #   run: |
      #     IP=$(hostname -I | awk '{print $1}')
      #     echo "The IP Address is: $IP"
      #     echo "::set-output name=ip::$IP"

      # - name: Update .env
      #   run: |
      #     sed -i 's|externalAddress:.*|externalAddress: "http://${{ steps.get-ip.outputs.ip }}:10005"|' config/min
      #     cat config/minio.yml

      - name: Build and test Server Services
        run: |
```

```
    mage build
    mage start
    mage check

- name: Checkout Chat repository
  uses: actions/checkout@v4
  with:
    repository: "openimsdk/chat"
    path: "chat-repo"

- name: Get Chat dependencies
  run: |
    cd ${{ github.workspace }}/chat-repo
    go mod tidy
    go mod download
    go install github.com/magefile/mage@latest

- name: Modify Chat Configuration
  run: |
    cd ${{ github.workspace }}/chat-repo
    yq e '.openIM.secret = 123456' -i ${{ env.SHARE_CONFIG_PATH }}

- name: Build and test Chat Services
  run: |
    cd ${{ github.workspace }}/chat-repo
    mage build
    mage start
    mage check

- name: Test Server and Chat
  run: |
    check_error() {
      echo "Response: $1"
      errCode=$(echo $1 | jq -r '.errCode')
      if [ "$errCode" != "0" ]; then
        errMsg=$(echo $1 | jq -r '.errMsg')
        echo "Error: $errMsg"
        exit 1
      fi
    }

    # Test register
    response1=$(curl -X POST -H "Content-Type: application/json" -H "operationID: imAdmin" -d '{
      "verifyCode": "666666",
      "platform": 3,
      "autoLogin": true,
      "user":{
      "nickname": "test12312",
      "areaCode":"+86",
      "phoneNumber": "12345678190",
      "password":"test123456"
      }
    }' http://127.0.0.1:10008/account/register)
    check_error "$response1"
    userID1=$(echo $response1 | jq -r '.data.userID')
    echo "userID1: $userID1"

    response2=$(curl -X POST -H "Content-Type: application/json" -H "operationID: imAdmin" -d '{
      "verifyCode": "666666",
      "platform": 3,
      "autoLogin": true,
      "user":{
      "nickname": "test22312",
      "areaCode":"+86",
      "phoneNumber": "12345678290",
      "password":"test123456"
```

```
          }
        }' http://127.0.0.1:10008/account/register)
        check_error "$response2"
        userID2=$(echo $response2 | jq -r '.data.userID')
        echo "userID2: $userID2"

        # Test login
        login_response=$(curl -X POST -H "Content-Type: application/json" -H "operationID: imAdmin"  -d '{
          "platform": 3,
          "areaCode":"+86",
          "phoneNumber": "12345678190",
          "password":"test123456"
        }' http://localhost:10008/account/login)
        check_error "$login_response"

        # Test get admin token
        get_admin_token_response=$(curl -X POST -H "Content-Type: application/json" -H "operationID: imAdmin" -d '
          "secret": "123456",
          "platformID": 2,
          "userID": "imAdmin"
        }' http://127.0.0.1:10002/auth/get_admin_token)
        check_error "$get_admin_token_response"
        adminToken=$(echo $get_admin_token_response | jq -r '.data.token')
        echo "adminToken: $adminToken"

        # Test send message
        send_msg_response=$(curl -X POST -H "Content-Type: application/json" -H "operationID: imAdmin" -H "token:
          "sendID": "'$userID1'",
          "recvID": "'$userID2'",
          "senderPlatformID": 3,
          "content": {
            "content": "hello!!"
          },
          "contentType": 101,
          "sessionType": 1
         }' http://127.0.0.1:10002/msg/send_msg)
         check_error "$send_msg_response"

        # Test get users
        get_users_response=$(curl -X POST -H "Content-Type: application/json" -H "operationID: imAdmin" -H "token:
          "pagination": {
             "pageNumber": 1,
             "showNumber": 100
           }
          }' http://127.0.0.1:10002/user/get_users)
        check_error "$get_users_response"

  go-test:
    name: Benchmark Test with go ${{ matrix.go_version }} on ${{ matrix.os }}
    runs-on: ${{ matrix.os }}
    permissions:
      contents: write
    env:
      SDK_DIR: openim-sdk-core
      NOTIFICATION_CONFIG_PATH: config/notification.yml
      SHARE_CONFIG_PATH: config/share.yml

    strategy:
      matrix:
        os: [ubuntu-latest]
        go_version: ["1.22.x"]

    steps:
      - name: Checkout Server repository
        uses: actions/checkout@v4
```

```yaml
    - name: Checkout SDK repository
      uses: actions/checkout@v4
      with:
        repository: "openimsdk/openim-sdk-core"
        ref: "main"
        path: ${{ env.SDK_DIR }}

    - name: Set up Go ${{ matrix.go_version }}
      uses: actions/setup-go@v5
      with:
        go-version: ${{ matrix.go_version }}

    - name: Get Server dependencies
      run: |
        go install github.com/magefile/mage@latest
        go mod download

    - name: Modify Server Configuration
      run: |
        yq e '.groupCreated.isSendMsg = true' -i ${{ env.NOTIFICATION_CONFIG_PATH }}
        yq e '.friendApplicationApproved.isSendMsg = true' -i ${{ env.NOTIFICATION_CONFIG_PATH }}
        yq e '.secret = 123456' -i ${{ env.SHARE_CONFIG_PATH }}

    - name: Start Server Services
      run: |
        docker compose up -d
        mage build
        mage start
        mage check

    - name: Build test SDK core
      run: |
        cd ${{ env.SDK_DIR }}
        go mod tidy
        cd integration_test
        mkdir data
        go run main.go -lgr 0.8 -imf -crg -ckgn -ckcon -sem -ckmsn -u 20 -su 5 -lg 2 -cg 2 -cgm 3 -sm 10 -gm 10 -r

dockerfile-test:
  name: Build and Test Dockerfile
  runs-on: ubuntu-latest
  strategy:
    matrix:
      go_version: ["1.22"]

  steps:
    - name: Checkout Repository
      uses: actions/checkout@v4

    - name: Set up Go ${{ matrix.go_version }}
      uses: actions/setup-go@v5
      with:
        go-version: ${{ matrix.go_version }}

    - name: Get dependencies
      run: |
        go mod tidy
        go mod download
        go install github.com/magefile/mage@latest

    - name: Build Docker Image
      run: |
        IMAGE_NAME="${{ github.event.repository.name }}-test"
        CONTAINER_NAME="${{ github.event.repository.name }}-container"
        docker build -t $IMAGE_NAME .
```

```
- name: Run Docker Container
  run: |
    IMAGE_NAME="${{ github.event.repository.name }}-test"
    CONTAINER_NAME="${{ github.event.repository.name }}-container"
    docker run --name $CONTAINER_NAME -d $IMAGE_NAME
    docker ps -a

- name: Test Docker Container Logs
  run: |
    CONTAINER_NAME="${{ github.event.repository.name }}-container"
    docker logs $CONTAINER_NAME
```

.github/workflows/help-comment-issue.yml

```yaml
# Copyright © 2023 OpenIM. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

name: Good frist issue add comment
on:
  issues:
    types:
      - labeled

jobs:
  add-comment:
    if: github.event.label.name == 'help wanted' || github.event.label.name == 'good first issue'
    runs-on: ubuntu-latest
    permissions:
      issues: write
    steps:
      - name: Add comment
        uses: peter-evans/create-or-update-comment@v4
        with:
          issue-number: ${{ github.event.issue.number }}
          token: ${{ secrets.BOT_TOKEN }}
          body: |
            This issue is available for anyone to work on. **Make sure to reference this issue in your pull request.
            [Join slack ■](https://join.slack.com/t/openimsdk/shared_invite/zt-2ijy1ys1f-O0aEDCr7ExRZ7mwsHAVg9A) to
            If you wish to accept this assignment, please leave a comment in the comments section: `/accept`.■
```

.github/workflows/issue-translator.yml

```yaml
name: 'issue-translator'
on:
  issue_comment:
    types: [created]
  issues:
    types: [opened]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: usthe/issues-translate-action@v2.7
        with:
          BOT_GITHUB_TOKEN: ${{ secrets.BOT_TOKEN }}
          IS_MODIFY_TITLE: true
          # not require, default false, . Decide whether to modify the issue title
          # if true, the robot account @Issues-translate-bot must have modification permissions, invite @Issues-tran
          CUSTOM_BOT_NOTE: Bot detected the issue body's language is not English, translate it automatically. ■■■■■
          # not require. Customize the translation robot prefix message.
```

.github/workflows/merge-from-milestone.yml

```yaml
name: Create Individual PRs from Milestone

permissions:
  contents: write
  pull-requests: write
  issues: write

on:
  workflow_dispatch:
    inputs:
      milestone_name:
        description: "Milestone name to collect closed PRs from"
        required: true
        default: "v3.8.4"
      target_branch:
        description: "Target branch to merge the consolidated PR"
        required: true
        default: "pre-release-v3.8.4"

env:
  MILESTONE_NAME: ${{ github.event.inputs.milestone_name || 'v3.8.4' }}
  TARGET_BRANCH: ${{ github.event.inputs.target_branch || 'pre-release-v3.8.4' }}
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
  BOT_TOKEN: ${{ secrets.BOT_TOKEN }}
  LABEL_NAME: cherry-picked
  TEMP_DIR: /tmp

jobs:
  merge_milestone_prs:
    runs-on: ubuntu-latest
    steps:
      - name: Setup temp directory
        run: |
          # Create the temporary directory and initialize necessary files
          mkdir -p ${{ env.TEMP_DIR }}
          touch ${{ env.TEMP_DIR }}/pr_numbers.txt
          touch ${{ env.TEMP_DIR }}/commit_hashes.txt
          touch ${{ env.TEMP_DIR }}/pr_title.txt
          touch ${{ env.TEMP_DIR }}/pr_body.txt
          touch ${{ env.TEMP_DIR }}/created_pr_number.txt

      - name: Checkout repository
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
          token: ${{ secrets.BOT_TOKEN }}

      - name: Setup Git User for OpenIM-Robot
        run: |
          git config --global user.email "OpenIM-Robot@users.noreply.github.com"
          git config --global user.name "OpenIM-Robot"

      - name: Fetch Milestone ID and Filter PR Numbers
        env:
          MILESTONE_NAME: ${{ env.MILESTONE_NAME }}
        run: |
          # Fetch milestone details and extract milestone ID
          milestones=$(curl -s -H "Authorization: token $BOT_TOKEN" \
            -H "Accept: application/vnd.github+json" \
            "https://api.github.com/repos/${{ github.repository }}/milestones")
          milestone_id=$(echo "$milestones" | grep -B3 "\"title\": \"$MILESTONE_NAME\"" | grep '"number":' | head -n
          if [ -z "$milestone_id" ]; then
            echo "Milestone '$MILESTONE_NAME' not found. Exiting."
            exit 1
```

```
    fi
    echo "Milestone ID: $milestone_id"
    echo "MILESTONE_ID=$milestone_id" >> $GITHUB_ENV

    # Fetch issues for the milestone
    issues=$(curl -s -H "Authorization: token $BOT_TOKEN" \
          -H "Accept: application/vnd.github+json" \
          "https://api.github.com/repos/${{ github.repository }}/issues?milestone=$milestone_id&state=closed&p

    > ${{ env.TEMP_DIR }}/pr_numbers.txt

    # Filter PRs that do not have the 'cherry-picked' label
    for pr_number in $(echo "$issues" | jq -r '.[] | select(.pull_request != null) | .number'); do
      labels=$(curl -s -H "Authorization: token $BOT_TOKEN" \
        -H "Accept: application/vnd.github+json" \
        "https://api.github.com/repos/${{ github.repository }}/issues/$pr_number/labels" | jq -r '.[].name')

      if ! echo "$labels" | grep -q "${LABEL_NAME}"; then
        echo "PR #$pr_number does not have the 'cherry-picked' label. Adding to the list."
        echo "$pr_number" >> ${{ env.TEMP_DIR }}/pr_numbers.txt
      fi
    done

    sort -n ${{ env.TEMP_DIR }}/pr_numbers.txt -o ${{ env.TEMP_DIR }}/pr_numbers.txt

- name: Create Individual PRs
  run: |
    for pr_number in $(cat ${{ env.TEMP_DIR }}/pr_numbers.txt); do
      pr_details=$(curl -s -H "Authorization: token $BOT_TOKEN" \
        -H "Accept: application/vnd.github+json" \
        "https://api.github.com/repos/${{ github.repository }}/pulls/$pr_number")
      pr_title=$(echo "$pr_details" | jq -r '.title')
      pr_body=$(echo "$pr_details" | jq -r '.body')
      pr_creator=$(echo "$pr_details" | jq -r '.user.login')
      merge_commit=$(echo "$pr_details" | jq -r '.merge_commit_sha')
      short_commit_hash=$(echo "$merge_commit" | cut -c 1-7)

      if [ "$merge_commit" != "null" ]; then
        git fetch origin

        echo "Checking out target branch: $TARGET_BRANCH"
        git checkout $TARGET_BRANCH

        echo "Pulling latest changes from target branch: $TARGET_BRANCH"
        git pull origin $TARGET_BRANCH

        cherry_pick_branch="cherry-pick-${short_commit_hash}"
        git checkout -b $cherry_pick_branch

        echo "Cherry-picking commit: $merge_commit"
        if ! git cherry-pick "$merge_commit" --strategy=recursive -X theirs; then
          echo "Conflict detected for $merge_commit. Resolving with incoming changes."
          conflict_files=$(git diff --name-only --diff-filter=U)
          echo "Conflicting files:"
          echo "$conflict_files"

          for file in $conflict_files; do
            if [ -f "$file" ]; then
              echo "Resolving conflict for $file"
              git add "$file"
            else
              echo "File $file has been deleted. Skipping."
              git rm "$file"
            fi
          done
```

```
      echo "Conflicts resolved. Continuing cherry-pick."
      git cherry-pick --continue || { echo "Cherry-pick failed, but continuing to create PR."; }
    else
      echo "Cherry-pick successful for commit $merge_commit."
    fi

    git remote set-url origin "https://${BOT_TOKEN}@github.com/${{ github.repository }}.git"

    echo "Pushing branch: $cherry_pick_branch"
    if ! git push origin $cherry_pick_branch --force; then
      echo "Push failed, but continuing to create PR..."
    fi

    new_pr_title="$pr_title [Created by @$pr_creator from #$pr_number]"
    new_pr_body="$pr_body
    > This PR is created from original PR #$pr_number."

    response=$(curl -s -X POST -H "Authorization: token $BOT_TOKEN" \
      -H "Accept: application/vnd.github+json" \
      https://api.github.com/repos/${{ github.repository }}/pulls \
      -d "$(jq -n --arg title "$new_pr_title" \
        --arg head "$cherry_pick_branch" \
        --arg base "$TARGET_BRANCH" \
        --arg body "$new_pr_body" \
        '{title: $title, head: $head, base: $base, body: $body}')")

    new_pr_number=$(echo "$response" | jq -r '.number')

    if [[ "$new_pr_number" == "null" || -z "$new_pr_number" ]]; then
      echo "Failed to create PR. Response: $response"

      git checkout $TARGET_BRANCH

      git branch -D $cherry_pick_branch

      echo "Deleted branch: $cherry_pick_branch"
      git push origin --delete $cherry_pick_branch
    else
      echo "Created PR #$new_pr_number"

      curl -s -X POST -H "Authorization: token $GITHUB_TOKEN" \
      -H "Accept: application/vnd.github+json" \
      -d '{"labels": ["milestone-merge"]}' \
      "https://api.github.com/repos/${{ github.repository }}/issues/$new_pr_number/labels"
    fi

    echo ""
    echo "----------------------------------------"
    echo ""
  fi
done
```

.github/workflows/publish-docker-image.yml

```yaml
name: Publish Docker image to registries

on:
  push:
    branches:
      - release-*
  release:
    types: [published]
  workflow_dispatch:
    inputs:
      tag:
        description: "Tag version to be used for Docker image"
        required: true
        default: "v3.8.3"

env:
  GO_VERSION: "1.22"
  IMAGE_NAME: "openim-server"
  # IMAGE_NAME: ${{ github.event.repository.name }}
  DOCKER_BUILDKIT: 1

jobs:
  publish-docker-images:
    runs-on: ubuntu-latest
    if: ${{ !(github.event_name == 'pull_request' && github.event.pull_request.merged == false) }}
    steps:
      - name: Checkout main repository
        uses: actions/checkout@v4
        with:
          path: main-repo

      - name: Set up QEMU
        uses: docker/setup-qemu-action@v3.3.0

      - name: Set up Docker Buildx
        id: buildx
        uses: docker/setup-buildx-action@v3
        with:
          driver-opts: network=host

      - name: Extract metadata for Docker
        id: meta
        uses: docker/metadata-action@v5.6.0
        with:
          images: |
            ${{ secrets.DOCKER_USERNAME }}/${{ env.IMAGE_NAME }}
            ghcr.io/${{ github.repository_owner }}/${{ env.IMAGE_NAME }}
            registry.cn-hangzhou.aliyuncs.com/openimsdk/${{ env.IMAGE_NAME }}
          tags: |
            type=ref,event=tag
            type=schedule
            type=ref,event=branch
            type=ref,event=pr
            type=semver,pattern={{version}}
            type=semver,pattern=v{{version}}
            type=semver,pattern={{major}}.{{minor}}
            type=semver,pattern={{major}}
            type=sha

      - name: Install skopeo
        run: |
          sudo apt-get update && sudo apt-get install -y skopeo

      - name: Build multi-arch images as OCI
```

```
  run: |
    mkdir -p /tmp/oci-image /tmp/docker-cache

    # Build multi-architecture image and save in OCI format
    docker buildx build \
      --platform linux/amd64,linux/arm64 \
      --output type=oci,dest=/tmp/oci-image/multi-arch.tar \
      --cache-to type=local,dest=/tmp/docker-cache \
      --cache-from type=gha \
      ./main-repo

      # Use skopeo to convert the amd64 image from OCI format to Docker format and load it
    skopeo copy --override-arch amd64 oci-archive:/tmp/oci-image/multi-arch.tar docker-daemon:${{ secrets.DOCK

    # check image
    docker image ls | grep openim

- name: Checkout compose repository
  uses: actions/checkout@v4
  with:
    repository: "openimsdk/openim-docker"
    path: "compose-repo"

- name: Get Internal IP Address
  id: get-ip
  run: |
    IP=$(hostname -I | awk '{print $1}')
    echo "The IP Address is: $IP"
    echo "ip=$IP" >> $GITHUB_OUTPUT

- name: Update .env to use the local image
  run: |
    sed -i 's|OPENIM_SERVER_IMAGE=.*|OPENIM_SERVER_IMAGE=${{ secrets.DOCKER_USERNAME }}/${{ env.IMAGE_NAME }}:
    sed -i 's|MINIO_EXTERNAL_ADDRESS=.*|MINIO_EXTERNAL_ADDRESS=http://${{ steps.get-ip.outputs.ip }}:10005|' $

- name: Start services using Docker Compose
  run: |
    cd ${{ github.workspace }}/compose-repo
    docker compose up -d

    docker compose ps

# - name: Check openim-server health
#   run: |
#     timeout=300
#     interval=30
#     elapsed=0
#     while [[ $elapsed -le $timeout ]]; do
#       if ! docker exec openim-server mage check; then
#         echo "openim-server is not ready, waiting..."
#         sleep $interval
#         elapsed=$(($elapsed + $interval))
#       else
#         echo "Health check successful"
#         exit 0
#       fi
#     done
#     echo "Health check failed after 5 minutes"
#     exit 1

# - name: Check openim-chat health
#   if: success()
#   run: |
#     if ! docker exec openim-chat mage check; then
#         echo "openim-chat check failed"
#         exit 1
```

```
#       else
#         echo "Health check successful"
#         exit 0
#       fi

- name: Log in to Docker Hub
  uses: docker/login-action@v3.3.0
  with:
    username: ${{ secrets.DOCKER_USERNAME }}
    password: ${{ secrets.DOCKER_PASSWORD }}

- name: Log in to GitHub Container Registry
  uses: docker/login-action@v3.3.0
  with:
    registry: ghcr.io
    username: ${{ github.repository_owner }}
    password: ${{ secrets.GITHUB_TOKEN }}

- name: Log in to Aliyun Container Registry
  uses: docker/login-action@v3.3.0
  with:
    registry: registry.cn-hangzhou.aliyuncs.com
    username: ${{ secrets.ALIREGISTRY_USERNAME }}
    password: ${{ secrets.ALIREGISTRY_TOKEN }}

- name: Push multi-architecture images
  if: success()
  run: |
    docker buildx build \
      --platform linux/amd64,linux/arm64 \
      $(echo "${{ steps.meta.outputs.tags }}" | sed 's/,/ --tag /g' | sed 's/^/--tag /') \
      --cache-from type=local,src=/tmp/docker-cache \
      --push \
      ./main-repo

- name: Verify multi-platform support
  run: |
    images=(
      "${{ secrets.DOCKER_USERNAME }}/${{ env.IMAGE_NAME }}"
      "ghcr.io/${{ github.repository_owner }}/${{ env.IMAGE_NAME }}"
      "registry.cn-hangzhou.aliyuncs.com/openimsdk/${{ env.IMAGE_NAME }}"
    )

    for image in "${images[@]}"; do
        for tag in $(echo "${{ steps.meta.outputs.tags }}" | tr ',' '\n' | cut -d':' -f2); do
            echo "Verifying multi-arch support for $image:$tag"
            manifest=$(docker manifest inspect "$image:$tag" || echo "error")
            if [[ "$manifest" == "error" ]]; then
                echo "Manifest not found for $image:$tag"
                exit 1
            fi
            amd64_found=$(echo "$manifest" | jq '.manifests[] | select(.platform.architecture == "amd64")')
            arm64_found=$(echo "$manifest" | jq '.manifests[] | select(.platform.architecture == "arm64")')
            if [[ -z "$amd64_found" ]]; then
                echo "Multi-platform support check failed for $image:$tag - missing amd64"
                exit 1
            fi
            if [[ -z "$arm64_found" ]]; then
                echo "Multi-platform support check failed for $image:$tag - missing arm64"
                exit 1
            fi
            echo "■ $image:$tag supports both amd64 and arm64 architectures"
        done
    done
```

.github/workflows/remove-unused-labels.yml

```yaml
name: Remove Unused Labels
on:
  workflow_dispatch:

jobs:
  cleanup:
    runs-on: ubuntu-latest
    permissions:
      issues: write
      pull-requests: write
      contents: read
    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Fetch All Issues and PRs
        id: fetch_issues_prs
        uses: actions/github-script@v7.0.1
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            const issues = await github.paginate(github.rest.issues.listForRepo, {
              owner: context.repo.owner,
              repo: context.repo.repo,
              state: 'all',
              per_page: 100
            });

            const labelsInUse = new Set();
            issues.forEach(issue => {
              issue.labels.forEach(label => {
                labelsInUse.add(label.name);
              });
            });

            return JSON.stringify(Array.from(labelsInUse));
          result-encoding: string

      - name: Fetch All Labels
        id: fetch_labels
        uses: actions/github-script@v7.0.1
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            const labels = await github.paginate(github.rest.issues.listLabelsForRepo, {
              owner: context.repo.owner,
              repo: context.repo.repo,
              per_page: 100
            });

            return JSON.stringify(labels.map(label => label.name));
          result-encoding: string

      - name: Remove Unused Labels
        uses: actions/github-script@v7.0.1
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            const labelsInUse = new Set(JSON.parse(process.env.LABELS_IN_USE));
            const allLabels = JSON.parse(process.env.ALL_LABELS);

            const unusedLabels = allLabels.filter(label => !labelsInUse.has(label));

            for (const label of unusedLabels) {
```

```
      await github.rest.issues.deleteLabel({
        owner: context.repo.owner,
        repo: context.repo.repo,
        name: label
      });
      console.log(`Deleted label: ${label}`);
    }
env:
  LABELS_IN_USE: ${{ steps.fetch_issues_prs.outputs.result }}
  ALL_LABELS: ${{ steps.fetch_labels.outputs.result }}
```

.github/workflows/reopen-issue.yml

```yaml
name: Reopen and Update Stale Issues

on:
  workflow_dispatch:

jobs:
  reopen_stale_issues:
    runs-on: ubuntu-latest
    permissions:
      issues: write
      contents: read

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Fetch Closed Issues with lifecycle/stale Label
        id: fetch_issues
        uses: actions/github-script@v7
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            const issues = await github.paginate(github.rest.issues.listForRepo, {
              owner: context.repo.owner,
              repo: context.repo.repo,
              state: 'closed',
              labels: 'lifecycle/stale',
              per_page: 100
            });
            const issueNumbers = issues
              .filter(issue => !issue.pull_request) // exclude PR
              .map(issue => issue.number);
            console.log(`Fetched issues: ${issueNumbers}`);
            return issueNumbers;

      - name: Set issue numbers
        id: set_issue_numbers
        run: |
          echo "ISSUE_NUMBERS=${{ steps.fetch_issues.outputs.result }}" >> $GITHUB_ENV
          echo "Issue numbers: ${{ steps.fetch_issues.outputs.result }}"

      - name: Reopen Issues
        uses: actions/github-script@v7
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            const issueNumbers = JSON.parse(process.env.ISSUE_NUMBERS);
            console.log(`Reopening issues: ${issueNumbers}`);

            for (const issue_number of issueNumbers) {
              // Reopen the issue
              await github.rest.issues.update({
                owner: context.repo.owner,
                repo: context.repo.repo,
                issue_number: issue_number,
                state: 'open'
              });
              console.log(`Reopened issue #${issue_number}`);
            }

      - name: Remove lifecycle/stale Label
        uses: actions/github-script@v7
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
```

```
script: |
  const issueNumbers = JSON.parse(process.env.ISSUE_NUMBERS);
  console.log(`Removing 'lifecycle/stale' label from issues: ${issueNumbers}`);

  for (const issue_number of issueNumbers) {
    // Remove the lifecycle/stale label
    await github.rest.issues.removeLabel({
      owner: context.repo.owner,
      repo: context.repo.repo,
      issue_number: issue_number,
      name: 'lifecycle/stale'
    });
    console.log(`Removed label 'lifecycle/stale' from issue #${issue_number}`);
  }
```

.github/workflows/update-version-file-on-release.yml

```yaml
name: Update Version File on Release

on:
  release:
    types: [created]

jobs:
  update-version:
    runs-on: ubuntu-latest
    env:
      TAG_VERSION: ${{ github.event.release.tag_name }}
    steps:
      # Step 1: Checkout the original repository's code
      - name: Checkout code
        uses: actions/checkout@v4
        with:
          fetch-depth: 0
          # submodules: "recursive"

      - name: Safe submodule initialization
        run: |
          echo "Checking for submodules..."
          if [ -f .gitmodules ]; then
            if [ -s .gitmodules ]; then
              echo "Initializing submodules..."
              if git submodule sync --recursive 2>/dev/null; then
                git submodule update --init --force --recursive || {
                  echo "Warning: Some submodules failed to initialize, continuing anyway..."
                }
              else
                echo "Warning: Submodule sync failed, continuing without submodules..."
              fi
            else
              echo ".gitmodules exists but is empty, skipping submodule initialization"
            fi
          else
            echo "No .gitmodules file found, no submodules to initialize"
          fi

      # Step 2: Set up Git with official account
      - name: Set up Git
        run: |
          git config --global user.name "github-actions[bot]"
          git config --global user.email "github-actions[bot]@users.noreply.github.com"

      # Step 3: Check and delete existing tag
      - name: Check and delete existing tag
        run: |
          if git rev-parse ${{ env.TAG_VERSION }} >/dev/null 2>&1; then
            git tag -d ${{ env.TAG_VERSION }}
            git push --delete origin ${{ env.TAG_VERSION }}
          fi

      # Step 4: Update version file
      - name: Update version file
        run: |
          mkdir -p version
          echo -n "${{ env.TAG_VERSION }}" > version/version

      # Step 5: Commit and push changes
      - name: Commit and push changes
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
```

```
    git add version/version
    git commit -m "Update version to ${{ env.TAG_VERSION }}"

# Step 6: Update tag
- name: Update tag
  run: |
    git tag -fa ${{ env.TAG_VERSION }} -m "Update version to ${{ env.TAG_VERSION }}"
    git push origin ${{ env.TAG_VERSION }} --force

# Step 7: Find and Publish Draft Release
- name: Find and Publish Draft Release
  uses: actions/github-script@v7
  with:
    github-token: ${{ secrets.GITHUB_TOKEN }}
    script: |
      const { owner, repo } = context.repo;
      const tagName = process.env.TAG_VERSION;

      try {
        let release;
        try {
          const response = await github.rest.repos.getReleaseByTag({
            owner,
            repo,
            tag: tagName
          });
          release = response.data;
        } catch (tagError) {
          core.info(`Release not found by tag, searching all releases...`);
          const releases = await github.rest.repos.listReleases({
            owner,
            repo,
            per_page: 100
          });

          release = releases.data.find(r => r.draft && r.tag_name === tagName);
          if (!release) {
            throw new Error(`No release found with tag ${tagName}`);
          }
        }

        await github.rest.repos.updateRelease({
          owner,
          repo,
          release_id: release.id,
          draft: false,
          prerelease: release.prerelease
        });

        const status = release.draft ? "was draft" : "was already published";
        core.info(`Release ${tagName} ensured to be published (${status}).`);

      } catch (error) {
        core.warning(`Could not find or update release for tag ${tagName}: ${error.message}`);
      }
```

.github/workflows/user-first-interaction.yml

```yaml
name: User First Interaction

on:
  issues:
    types: [opened]
  pull_request:
    branches: [main]
    types: [opened]

jobs:
  check_for_first_interaction:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/first-interaction@v1.3.0
        with:
          repo-token: ${{ secrets.BOT_TOKEN }}
          pr-message: |
            Hello! Thank you for your contribution.

            If you are fixing a bug, please reference the issue number in the description.

            If you are implementing a feature request, please check with the maintainers that the feature will be ac

            [Join slack ■](https://join.slack.com/t/openimsdk/shared_invite/zt-2ijy1ys1f-O0aEDCr7ExRZ7mwsHAVg9A) to

            Please leave your information in the [■ discussions](https://github.com/orgs/OpenIMSDK/discussions/426),

          issue-message: |
            Hello! Thank you for filing an issue.

            If this is a bug report, please include relevant logs to help us debug the problem.

            [Join slack ■](https://join.slack.com/t/openimsdk/shared_invite/zt-2ijy1ys1f-O0aEDCr7ExRZ7mwsHAVg9A) to
        continue-on-error: true
```

.github/ISSUE_TEMPLATE

.github/ISSUE_TEMPLATE/bug-report.yml

```yaml
name: Bug Report
title: "[BUG] "
labels: ["bug"]
description: "Create a detailed report to help us identify and resolve issues."
# assignees: []

body:
  - type: markdown
    attributes:
      value: "Thank you for taking the time to fill out the bug report. Please provide as much information as possib

  - type: input
    id: openim-server-version
    attributes:
      label: OpenIM Server Version
      description: "Please provide the version number of OpenIM Server you are using."
      placeholder: "e.g., 3.8.0"
    validations:
      required: true

  - type: dropdown
    id: operating-system
    attributes:
      label: Operating System and CPU Architecture
      description: "Please select the operating system and describe the CPU architecture."
      options:
        - Linux (AMD)
        - Linux (ARM)
        - Windows (AMD)
        - Windows (ARM)
        - macOS (AMD)
        - macOS (ARM)
    validations:
      required: true

  - type: dropdown
    id: deployment-method
    attributes:
      label: Deployment Method
      description: "Please specify how OpenIM Server was deployed."
      options:
        - Source Code Deployment
        - Docker Deployment
    validations:
      required: true

  - type: textarea
    id: bug-description-reproduction
    attributes:
      label: Bug Description and Steps to Reproduce
      description: "Provide a detailed description of the bug and a step-by-step guide on how to reproduce it."
      placeholder: "Describe the bug in detail here...\n\nSteps to reproduce the bug on the server:\n1. Start the se
    validations:
      required: true

  - type: markdown
    attributes:
      value: "If possible, please add screenshots to help explain your problem."

  - type: textarea
    id: screenshots-link
```

```
attributes:
  label: Screenshots Link
  description: "If applicable, please provide any links to screenshots here."
  placeholder: "Paste your screenshot URL here, e.g., http://imgur.com/example"
```

.github/ISSUE_TEMPLATE/config.yml

```yaml
blank_issues_enabled: false
contact_links:
  # - name: "Bug Report"
  #   description: "Report a bug in the project"
  #   file: "bug-report.yml"
  - name: ■ Connect on slack
    url: https://join.slack.com/t/openimsdk/shared_invite/zt-2ijy1ys1f-O0aEDCr7ExRZ7mwsHAVg9A
    about: Support OpenIM-related requests or issues, get in touch with developers and help on slack
  - name: ■ OpenIM Blog
    url: https://www.openim.io/
    about: Open the OpenIM community blog
```

.github/ISSUE_TEMPLATE/deployment.yml

```yaml
name: Deployment issue
title: "[Deployment] "
labels: ["deployment"]
description: "Create a detailed report to help us identify and resolve deployment issues."
# assignees: []

body:
  - type: markdown
    attributes:
      value: "Thank you for taking the time to fill out the deployment issue report. Please provide as much informat

  - type: input
    id: openim-server-version
    attributes:
      label: OpenIM Server Version
      description: "Please provide the version number of OpenIM Server you are using."
      placeholder: "e.g., 3.8.0"
    validations:
      required: true

  - type: dropdown
    id: operating-system
    attributes:
      label: Operating System and CPU Architecture
      description: "Please select the operating system and describe the CPU architecture."
      options:
        - Linux (AMD)
        - Linux (ARM)
        - Windows (AMD)
        - Windows (ARM)
        - macOS (AMD)
        - macOS (ARM)
    validations:
      required: true

  - type: dropdown
    id: deployment-method
    attributes:
      label: Deployment Method
      description: "Please specify how OpenIM Server was deployed."
      options:
        - Source Code Deployment
        - Docker Deployment
    validations:
      required: true

  - type: textarea
    id: issue-description-reproduction
    attributes:
      label: Issue Description and Steps to Reproduce
      description: "Provide a detailed description of the issue and a step-by-step guide on how to reproduce it."
      placeholder: "Describe the issue in detail here...\n\nSteps to reproduce the issue on the server:\n1. Start th
    validations:
      required: true

  - type: markdown
    attributes:
      value: "If possible, please add screenshots to help explain your problem."

  - type: textarea
    id: screenshots-link
    attributes:
      label: Screenshots Link
      description: "If applicable, please provide any links to screenshots here."
```

```
placeholder: "Paste your screenshot URL here, e.g., http://imgur.com/example"
```

.github/ISSUE_TEMPLATE/feature-request.yml

```yaml
name: Feature Request
title: "[FEATURE REQUEST] "
labels: ["feature request","enhancement"]
description: "Propose a new feature or improvement that you believe will help enhance the project."
# assignees: []

body:
  - type: markdown
    attributes:
      value: "Thank you for taking the time to propose a feature request. Please fill in as much detail as possible

  - type: textarea
    id: feature-reason
    attributes:
      label: Why this feature?
      description: "Explain why this feature is needed. What problem does it solve? How does it benefit the project
      placeholder: "Describe the need for this feature..."
    validations:
      required: true

  - type: textarea
    id: solution-proposal
    attributes:
      label: Suggested Solution
      description: "Describe your proposed solution for this feature. How do you envision it working?"
      placeholder: "Detail your solution here..."
    validations:
      required: true

  - type: markdown
    attributes:
      value: "Please provide any other relevant information or screenshots that could help illustrate your idea."

  - type: textarea
    id: additional-info
    attributes:
      label: Additional Information
      description: "Include any additional information, links, or screenshots that might be relevant to your feature
      placeholder: "Add more context or links to relevant resources..."

  - type: markdown
    attributes:
      value: "Thank you for contributing to the project! We appreciate your input and will review your suggestion as
```

.github/ISSUE_TEMPLATE/other.yml

```yaml
name: ■ Other
description: Use this for any other issues. Please do NOT create blank issues
title: "[Other]: <give this problem a name>"
labels: ["other"]
# assignees: []

body:
  - type: markdown
    attributes:
      value: "# Other issue"
  - type: textarea
    id: issuedescription
    attributes:
      label: What would you like to share?
      description: Provide a clear and concise explanation of your issue.
    validations:
      required: true
  - type: textarea
    id: extrainfo
    attributes:
      label: Additional information
      description: Is there anything else we should know about this issue?
    validations:
      required: false
  - type: markdown
    attributes:
      value: |
        You can also join our Discord community [here](https://join.slack.com/t/openimsdk/shared_invite/zt-2ijy1ys1f
        Feel free to check out other cool repositories of the openim Community [here](https://github.com/openimsdk)
```

config

config/alertmanager.yml

```yaml
global:
  resolve_timeout: 5m
  smtp_from: alert@openim.io
  smtp_smarthost: smtp.163.com:465
  smtp_auth_username: alert@openim.io
  smtp_auth_password: YOURAUTHPASSWORD
  smtp_require_tls: false
  smtp_hello: xxx

templates:
  - /etc/alertmanager/email.tmpl

route:
  group_by: [ 'alertname' ]
  group_wait: 5s
  group_interval: 5s
  repeat_interval: 5m
  receiver: email
  routes:
    - matchers:
        - alertname = "XXX"
      group_by: [ 'instance' ]
      group_wait: 5s
      group_interval: 5s
      repeat_interval: 5m
      receiver: email

receivers:
  - name: email
    email_configs:
      - to: 'alert@example.com'
        html: '{{ template "email.to.html" . }}'
        headers: { Subject: "[OPENIM-SERVER]Alarm" }
        send_resolved: true
```

config/discovery.yml

```yaml
enable: etcd
etcd:
  rootDirectory: openim
  address: [localhost:12379]
  ## Attention: If you set auth in etcd
  ## you must also update the username and password in Chat project.
  username:
  password:

kubernetes:
  namespace: default

rpcService:
  user: user-rpc-service
  friend: friend-rpc-service
  msg: msg-rpc-service
  push: push-rpc-service
  messageGateway: messagegateway-rpc-service
  group: group-rpc-service
  auth: auth-rpc-service
  conversation: conversation-rpc-service
  third: third-rpc-service
```

config/instance-down-rules.yml

```yaml
groups:
  - name: instance_down
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Instance {{ $labels.instance }} down"
          description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minutes."

  - name: database_insert_failure_alerts
    rules:
      - alert: DatabaseInsertFailed
        expr: (increase(msg_insert_redis_failed_total[5m]) > 0) or (increase(msg_insert_mongo_failed_total[5m]) > 0)
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Increase in MsgInsertRedisFailedCounter or MsgInsertMongoFailedCounter detected"
          description: "Either MsgInsertRedisFailedCounter or MsgInsertMongoFailedCounter has increased in the last

  - name: registrations_few
    rules:
      - alert: RegistrationsFew
        expr: increase(user_login_total[1h]) == 0
        for: 1m
        labels:
          severity: info
        annotations:
          summary: "Too few registrations within the time frame"
          description: "The number of registrations in the last hour is 0. There might be some issues."

  - name: messages_few
    rules:
      - alert: MessagesFew
        expr: (increase(single_chat_msg_process_success_total[1h])+increase(group_chat_msg_process_success_total[1h]
        for: 1m
        labels:
          severity: info
        annotations:
          summary: "Too few messages within the time frame"
          description: "The number of messages sent in the last hour is 0. There might be some issues."
```

config/kafka.yml

```yaml
## Kafka authentication
username:
password:

# Producer acknowledgment settings
producerAck:
# Compression type to use (e.g., none, gzip, snappy)
compressType: none
# List of Kafka broker addresses
address: [localhost:19094]
# Kafka topic for Redis integration
toRedisTopic: toRedis
# Kafka topic for MongoDB integration
toMongoTopic: toMongo
# Kafka topic for push notifications
toPushTopic: toPush
# Kafka topic for offline push notifications
toOfflinePushTopic: toOfflinePush
# Consumer group ID for Redis topic
toRedisGroupID: redis
# Consumer group ID for MongoDB topic
toMongoGroupID: mongo
# Consumer group ID for push notifications topic
toPushGroupID: push
# Consumer group ID for offline push notifications topic
toOfflinePushGroupID: offlinePush
# TLS (Transport Layer Security) configuration
tls:
  # Enable or disable TLS
  enableTLS: false
  # CA certificate file path
  caCrt:
  # Client certificate file path
  clientCrt:
  # Client key file path
  clientKey:
  # Client key password
  clientKeyPwd:
  # Whether to skip TLS verification (not recommended for production)
  insecureSkipVerify: false
```

config/local-cache.yml

```yaml
auth:
  topic: DELETE_CACHE_AUTH
  slotNum: 100
  slotSize: 2000
  successExpire: 300
  failedExpire: 5

user:
  topic: DELETE_CACHE_USER
  slotNum: 100
  slotSize: 2000
  successExpire: 300
  failedExpire: 5

group:
  topic: DELETE_CACHE_GROUP
  slotNum: 100
  slotSize: 2000
  successExpire: 300
  failedExpire: 5

friend:
  topic: DELETE_CACHE_FRIEND
  slotNum: 100
  slotSize: 2000
  successExpire: 300
  failedExpire: 5

conversation:
  topic: DELETE_CACHE_CONVERSATION
  slotNum: 100
  slotSize: 2000
  successExpire: 300
  failedExpire: 5
```

config/log.yml

```yaml
# Log storage path, default is acceptable, change to a full path if modification is needed
storageLocation: ../../../../logs/
# Log rotation period (in hours), default is acceptable
rotationTime: 24
# Number of log files to retain, default is acceptable
remainRotationCount: 2
# Log level settings: 3 for production environment; 6 for more verbose logging in debugging environments
remainLogLevel: 6
# Whether to output to standard output, default is acceptable
isStdout: false
# Whether to log in JSON format, default is acceptable
isJson: false
# output simplify log when KeyAndValues's value len is bigger than 50 in rpc method log
isSimplify: true
```

config/minio.yml

```yaml
# Name of the bucket in MinIO
bucket: openim
# Access key ID for MinIO authentication
accessKeyID: root
# Secret access key for MinIO authentication
secretAccessKey: openIM123
# Session token for MinIO authentication (optional)
sessionToken:
# Internal address of the MinIO server
internalAddress: localhost:10005
# External address of the MinIO server, accessible from outside. Supports both HTTP and HTTPS using a domain name
externalAddress: http://external_ip:10005
# Flag to enable or disable public read access to the bucket
publicRead: false
```

config/mongodb.yml

```yaml
# URI for database connection, leave empty if using address and credential settings directly
uri:
# List of MongoDB server addresses
address: [localhost:37017]
# Name of the database
database: openim_v3
# Username for database authentication
username: openIM
# Password for database authentication
password: openIM123
# Authentication source for database authentication, if use root user, set it to admin
authSource: openim_v3
# Maximum number of connections in the connection pool
maxPoolSize: 100
# Maximum number of retry attempts for a failed database connection
maxRetry: 10
# MongoDB Mode, including "standalone", "replicaSet"
mongoMode: "standalone"

# The following configurations only take effect when mongoMode is set to "replicaSet"
replicaSet:
  name: rs0
  hosts: [127.0.0.1:37017, 127.0.0.1:37018, 127.0.0.1:37019]
  # Read concern level: "local", "available", "majority", "linearizable", "snapshot"
  readConcern: majority
  # maximum staleness of data in seconds
  maxStaleness: 90s

# The following configurations only take effect when mongoMode is set to "replicaSet"
readPreference:
  # Read preference mode, can be "primary", "primaryPreferred", "secondary", "secondaryPreferred", "nearest"
  mode: primary
  maxStaleness: 90s
  # TagSets is an array of maps with priority based on order, empty map must be placed last for fallback tagSets
  tagSets:
    - datacenter: "cn-east"
      rack: "1"
      storage: "ssd"
    - datacenter: "cn-east"
      storage: "ssd"
    - datacenter: "cn-east"
    - {} # Empty map, indicates any node

# The following configurations only take effect when mongoMode is set to "replicaSet"
writeConcern:
  # Write node count or tag (int, "majority", or custom tag)
  w: majority
  # Whether to wait for journal confirmation
  j: true
  # Write timeout duration
  wtimeout: 30s
```

config/notification.yml

```yaml
groupCreated:
  isSendMsg: true
# Deprecated. Fixed as 1.
  reliabilityLevel: 1
# Deprecated. Fixed as false.
  unreadCount: false
# Configuration for offline push notifications.
  offlinePush:
    # Enables or disables offline push notifications.
    enable: false
    # Title for the notification when a group is created.
    title: create group title
    # Description for the notification.
    desc: create group desc
    # Additional information for the notification.
    ext: create group ext

groupInfoSet:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupInfoSet title
    desc: groupInfoSet desc
    ext: groupInfoSet ext


joinGroupApplication:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: joinGroupApplication title
    desc: joinGroupApplication desc
    ext: joinGroupApplication ext

memberQuit:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberQuit title
    desc: memberQuit desc
    ext: memberQuit ext

groupApplicationAccepted:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: groupApplicationAccepted title
    desc: groupApplicationAccepted desc
    ext: groupApplicationAccepted ext

groupApplicationRejected:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
```

```
    title: groupApplicationRejected title
    desc: groupApplicationRejected desc
    ext: groupApplicationRejected ext


groupOwnerTransferred:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupOwnerTransferred title
    desc: groupOwnerTransferred desc
    ext: groupOwnerTransferred ext

memberKicked:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberKicked title
    desc: memberKicked desc
    ext: memberKicked ext

memberInvited:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberInvited title
    desc: memberInvited desc
    ext: memberInvited ext

memberEnter:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: memberEnter title
    desc: memberEnter desc
    ext: memberEnter ext

groupDismissed:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupDismissed title
    desc: groupDismissed desc
    ext: groupDismissed ext

groupMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMuted title
    desc: groupMuted desc
    ext: groupMuted ext

groupCancelMuted:
```

```
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupCancelMuted title
    desc: groupCancelMuted desc
    ext: groupCancelMuted ext
  defaultTips:
    tips: group Cancel Muted


groupMemberMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMemberMuted title
    desc: groupMemberMuted desc
    ext: groupMemberMuted ext

groupMemberCancelMuted:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMemberCancelMuted title
    desc: groupMemberCancelMuted desc
    ext: groupMemberCancelMuted ext

groupMemberInfoSet:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupMemberInfoSet title
    desc: groupMemberInfoSet desc
    ext: groupMemberInfoSet ext

groupInfoSetAnnouncement:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupInfoSetAnnouncement title
    desc: groupInfoSetAnnouncement desc
    ext: groupInfoSetAnnouncement ext


groupInfoSetName:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: groupInfoSetName title
    desc: groupInfoSetName desc
    ext: groupInfoSetName ext


##############################friend###################################
friendApplicationAdded:
```

```
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: Somebody applies to add you as a friend
    desc: Somebody applies to add you as a friend
    ext: Somebody applies to add you as a friend

friendApplicationApproved:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: Someone applies to add your friend application
    desc: Someone applies to add your friend application
    ext: Someone applies to add your friend application

friendApplicationRejected:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: true
    title: Someone rejected your friend application
    desc: Someone rejected your friend application
    ext: Someone rejected your friend application

friendAdded:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: We have become friends
    desc: We have become friends
    ext: We have become friends

friendDeleted:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: deleted a friend
    desc: deleted a friend
    ext: deleted a friend

friendRemarkSet:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: Your friend's profile has been changed
    desc: Your friend's profile has been changed
    ext: Your friend's profile has been changed

blackAdded:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: blocked a user
```

```
    desc: blocked a user
    ext: blocked a user

blackDeleted:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: Remove a blocked user
    desc: Remove a blocked user
    ext: Remove a blocked user

friendInfoUpdated:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: friend info updated
    desc: friend info updated
    ext: friend info updated

######################user#########################
userInfoUpdated:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: userInfo updated
    desc: userInfo updated
    ext: userInfo updated

userStatusChanged:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: user status changed
    desc: user status changed
    ext: user status changed

######################conversation#########################
conversationChanged:
  isSendMsg: false
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: conversation changed
    desc: conversation changed
    ext: conversation changed

conversationSetPrivate:
  isSendMsg: true
  reliabilityLevel: 1
  unreadCount: false
  offlinePush:
    enable: false
    title: burn after reading
    desc: burn after reading
    ext: burn after reading
```

config/openim-api.yml

```yaml
api:
  # Listening IP; 0.0.0.0 means both internal and external IPs are listened to, default is recommended
  listenIP: 0.0.0.0
  # Listening ports; if multiple are configured, multiple instances will be launched, must be consistent with the nu
  ports: [ 10002 ]
  # API compression level; 0: default compression, 1: best compression, 2: best speed, -1: no compression
  compressionLevel: 0


prometheus:
  # Whether to enable prometheus
  enable: true
  # autoSetPorts indicates whether to automatically set the ports
  autoSetPorts: true
  # Prometheus listening ports, must match the number of api.ports
  # It will only take effect when autoSetPorts is set to false.
  ports:
  # This address can be accessed via a browser
  grafanaURL:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850
```

config/openim-crontask.yml

```
cronExecuteTime: 0 2 * * *
retainChatRecords: 365
fileExpireTime: 180
deleteObjectType: ["msg-picture","msg-file", "msg-voice","msg-video","msg-video-snapshot","sdklog"]
```

config/openim-msggateway.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:
# IP address that the RPC/WebSocket service listens on; setting to 0.0.0.0 listens on both internal and external IPs
listenIP: 0.0.0.0

longConnSvr:
  # WebSocket listening ports, must match the number of rpc.ports
  ports: [ 10001 ]
  # Maximum number of WebSocket connections
  websocketMaxConnNum: 100000
  # Maximum length of the entire WebSocket message packet
  websocketMaxMsgLen: 4096
  # WebSocket connection handshake timeout in seconds
  websocketTimeout: 10

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-msgtransfer.yml

```yaml
prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # autoSetPorts indicates whether to automatically set the ports
  autoSetPorts: true
  # List of ports that Prometheus listens on; each port corresponds to an instance of monitoring. Ensure these are m
  # It will only take effect when autoSetPorts is set to false.
  ports:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-push.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached

prometheus:
  # Enable or disable Prometheus monitoring
  enable: false
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:

maxConcurrentWorkers: 3
#Use geTui for offline push notifications, or choose fcm or jpns; corresponding configuration settings must be speci
enable:
getui:
  pushUrl: https://restapi.getui.com/v2/$appId
  masterSecret:
  appKey:
  intent:
  channelID:
  channelName:
fcm:
  # Prioritize using file paths. If the file path is empty, use URL
  filePath:   # File path is concatenated with the parameters passed in through - c(`mage` default pass in `config/`
  authURL:   #  Must start with https or http.
jpush:
  appKey:
  masterSecret:
  pushURL:
  pushIntent:

# iOS system push sound and badge count
iosPush:
  pushSound: xxx
  badgeCount: true
  production: false

fullUserCache: true
```

config/openim-rpc-auth.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:

tokenPolicy:
  # Token validity period, in days
  expire: 90

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-rpc-conversation.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-rpc-friend.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-rpc-group.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:


enableHistoryForNewMembers: true

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-rpc-msg.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:


# Does sending messages require friend verification
friendVerify: false

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/openim-rpc-third.yml

```yaml
rpc:
  # The IP address where this RPC service registers itself; if left blank, it defaults to the internal network IP
  registerIP:
  # IP address that the RPC service listens on; setting to 0.0.0.0 listens on both internal and external IPs. If lef
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Enable or disable Prometheus monitoring
  enable: true
  # List of ports that Prometheus listens on; these must match the number of rpc.ports to ensure correct monitoring
  # It will only take effect when autoSetPorts is set to false.
  ports:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached

object:
  # Use MinIO as object storage, or set to "cos", "oss", "kodo", "aws", while also configuring the corresponding set
  enable: minio
  cos:
    bucketURL: https://temp-1252357374.cos.ap-chengdu.myqcloud.com
    secretID:
    secretKey:
    sessionToken:
    publicRead: false
  oss:
    endpoint: https://oss-cn-chengdu.aliyuncs.com
    bucket: demo-9999999
    bucketURL: https://demo-9999999.oss-cn-chengdu.aliyuncs.com
    accessKeyID:
    accessKeySecret:
    sessionToken:
    publicRead: false
  kodo:
    endpoint: https://s3.cn-south-1.qiniucs.com
    bucket: testdemo12313
    bucketURL: http://so2at6d05.hn-bkt.clouddn.com
    accessKeyID:
    accessKeySecret:
    sessionToken:
    publicRead: false
  aws:
    region: ap-southeast-2
    bucket: testdemo832234
```

```
accessKeyID:
secretAccessKey:
sessionToken:
publicRead: false
```

config/openim-rpc-user.yml

```yaml
rpc:
  # API or other RPCs can access this RPC through this IP; if left blank, the internal network IP is obtained by def
  registerIP:
  # Listening IP; 0.0.0.0 means both internal and external IPs are listened to, if blank, the internal network IP is
  listenIP: 0.0.0.0
  # autoSetPorts indicates whether to automatically set the ports
  # if you use in kubernetes, set it to false
  autoSetPorts: true
  # List of ports that the RPC service listens on; configuring multiple ports will launch multiple instances. These
  # It will only take effect when autoSetPorts is set to false.
  ports:

prometheus:
  # Whether to enable prometheus
  enable: true
  # Prometheus listening ports, must be consistent with the number of rpc.ports
  # It will only take effect when autoSetPorts is set to false.
  ports:

ratelimiter:
  # Whether to enable rate limiting
  enable: false
  # WindowSize defines time duration per window
  window: 20s
  # BucketNum defines bucket number for each window
  bucket: 500
  # CPU threshold; valid range 0–1000 (1000 = 100%)
  cpuThreshold: 850

circuitBreaker:
  enable: false
  window: 5s            # Time window size (seconds)
  bucket: 100            # Number of buckets
  success: 0.6          # Success rate threshold (0.6 means 60%)
  request: 500 # Request threshold; circuit breaker evaluation occurs when reached
```

config/prometheus.yml

```yaml
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets: [127.0.0.1:19093]

# Load rules once and periodically evaluate them according to the global evaluation_interval.
rule_files:
  - instance-down-rules.yml
# - first_rules.yml
# - second_rules.yml

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label "job=job_name" to any timeseries scraped from this config.
  # Monitored information captured by prometheus

  # prometheus fetches application services
  - job_name: node_exporter
    static_configs:
      - targets: [ 127.0.0.1:19100 ]

  - job_name: openimserver-openim-api
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/api"
#    static_configs:
#      - targets: [ 127.0.0.1:12002 ]
#        labels:
#          namespace: default

  - job_name: openimserver-openim-msggateway
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/msg_gateway"
#    static_configs:
#      - targets: [ 127.0.0.1:12140 ]
#        #      - targets: [ 127.0.0.1:12140, 127.0.0.1:12141, 127.0.0.1:12142, 127.0.0.1:12143, 127.0.0.1:12144, 12
#        labels:
#          namespace: default

  - job_name: openimserver-openim-msgtransfer
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/msg_transfer"
#    static_configs:
#      - targets: [ 127.0.0.1:12020, 127.0.0.1:12021, 127.0.0.1:12022, 127.0.0.1:12023, 127.0.0.1:12024, 127.0.0.1:1
#        #      - targets: [ 127.0.0.1:12020, 127.0.0.1:12021, 127.0.0.1:12022, 127.0.0.1:12023, 127.0.0.1:12024, 12
#        labels:
#          namespace: default

  - job_name: openimserver-openim-push
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/push"
#    static_configs:
#      - targets: [ 127.0.0.1:12170, 127.0.0.1:12171, 127.0.0.1:12172, 127.0.0.1:12173, 127.0.0.1:12174, 127.0.0.1:1
##      - targets: [ 127.0.0.1:12170, 127.0.0.1:12171, 127.0.0.1:12172, 127.0.0.1:12173, 127.0.0.1:12174, 127.0.0.1:
#        labels:
#          namespace: default
```

```
  - job_name: openimserver-openim-rpc-auth
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/auth"
#    static_configs:
#      - targets: [ 127.0.0.1:12200 ]
#        labels:
#          namespace: default

  - job_name: openimserver-openim-rpc-conversation
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/conversation"
#    static_configs:
#      - targets: [ 127.0.0.1:12220 ]
#        labels:
#          namespace: default

  - job_name: openimserver-openim-rpc-friend
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/friend"
#    static_configs:
#      - targets: [ 127.0.0.1:12240 ]
#        labels:
#          namespace: default

  - job_name: openimserver-openim-rpc-group
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/group"
#    static_configs:
#      - targets: [ 127.0.0.1:12260 ]
#        labels:
#          namespace: default.

  - job_name: openimserver-openim-rpc-msg
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/msg"
#    static_configs:
#      - targets: [ 127.0.0.1:12280 ]
#        labels:
#          namespace: default

  - job_name: openimserver-openim-rpc-third
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/third"
#    static_configs:
#      - targets: [ 127.0.0.1:12300 ]
#        labels:
#          namespace: default

  - job_name: openimserver-openim-rpc-user
    http_sd_configs:
      - url: "http://127.0.0.1:10002/prometheus_discovery/user"
#    static_configs:
#      - targets: [ 127.0.0.1:12320 ]
#        labels:
#          namespace: default
```

config/redis.yml

```yaml
address: [localhost:16379]
username:
password: openIM123
# redis Mode, including "standalone","cluster","sentinel"
redisMode: "standalone"
db: 0
maxRetry: 10
poolSize: 100
# Sentinel configuration (only used when redisMode is "sentinel")
sentinelMode:
  masterName: "redis-master"
  sentinelsAddrs: ["127.0.0.1:26379", "127.0.0.1:26380", "127.0.0.1:26381"]
  routeByLatency: true
  routeRandomly: true
```

config/share.yml

```yaml
secret: openIM123

# imAdminUser: Configuration for instant messaging system administrators
imAdminUser:
  # userIDs: List of administrator user IDs.
  # Each entry here corresponds by index to the matching entry in the nicknames list below.
  userIDs: [imAdmin]
  # nicknames: List of administrator display names.
  # Each entry here corresponds by index to the matching entry in the userIDs list above.
  nicknames: [superAdmin]

# 1: For Android, iOS, Windows, Mac, and web platforms, only one instance can be online at a time
multiLogin:
  policy: 1
  # max num of tokens in one end
  maxNumOneEnd: 30

rpcMaxBodySize:
  requestMaxBodySize:  8388608
  responseMaxBodySize: 8388608
```

config/webhooks.yml

```yaml
url: http://127.0.0.1:10006/callbackExample
beforeSendSingleMsg:
  enable: false
  timeout: 5
  failedContinue: true
  # Only the contentType not in deniedTypes will send the callback.
  # If not set, all contentType messages will through this filter.
  deniedTypes: []
beforeUpdateUserInfoEx:
  enable: false
  timeout: 5
  failedContinue: true
afterUpdateUserInfoEx:
  enable: false
  timeout: 5
afterSendSingleMsg:
  enable: false
  timeout: 5
  # Only the recvIDs specified in attentionIds will send the callback
  # if not set, all user messages will be callback
  attentionIds: []
  # See beforeSendSingleMsg comment.
  deniedTypes: []
beforeSendGroupMsg:
  enable: false
  timeout: 5
  failedContinue: true
  # See beforeSendSingleMsg comment.
  deniedTypes: []
beforeMsgModify:
  enable: false
  timeout: 5
  failedContinue: true
  # See beforeSendSingleMsg comment.
  deniedTypes: []
afterSendGroupMsg:
  enable: false
  timeout: 5
  # Only the GroupIDs specified in attentionIds will send the callback
  # if not set, all user messages will be callback
  attentionIds: []
  # See beforeSendSingleMsg comment.
  deniedTypes: []
afterMsgSaveDB:
  enable: false
  timeout: 5
afterUserOnline:
  enable: false
  timeout: 5
afterUserOffline:
  enable: false
  timeout: 5
afterUserKickOff:
  enable: false
  timeout: 5
beforeOfflinePush:
  enable: false
  timeout: 5
  failedContinue: true
beforeOnlinePush:
  enable: false
  timeout: 5
  failedContinue: true
beforeGroupOnlinePush:
```

```
  enable: false
  timeout: 5
  failedContinue: true
beforeAddFriend:
  enable: false
  timeout: 5
  failedContinue: true
beforeUpdateUserInfo:
  enable: false
  timeout: 5
  failedContinue: true
afterUpdateUserInfo:
  enable: false
  timeout: 5
beforeCreateGroup:
  enable: false
  timeout: 5
  failedContinue: true
afterCreateGroup:
  enable: false
  timeout: 5
beforeMemberJoinGroup:
  enable: false
  timeout: 5
  failedContinue: true
beforeSetGroupMemberInfo:
  enable: false
  timeout: 5
  failedContinue: true
afterSetGroupMemberInfo:
  enable: false
  timeout: 5
afterQuitGroup:
  enable: false
  timeout: 5
afterKickGroupMember:
  enable: false
  timeout: 5
afterDismissGroup:
  enable: false
  timeout: 5
beforeApplyJoinGroup:
  enable: false
  timeout: 5
  failedContinue: true
afterGroupMsgRead:
  enable: false
  timeout: 5
afterSingleMsgRead:
  enable: false
  timeout: 5
beforeUserRegister:
  enable: false
  timeout: 5
  failedContinue: true
afterUserRegister:
  enable: false
  timeout: 5
afterTransferGroupOwner:
  enable: false
  timeout: 5
beforeSetFriendRemark:
  enable: false
  timeout: 5
  failedContinue: true
afterSetFriendRemark:
```

```
  enable: false
  timeout: 5
afterGroupMsgRevoke:
  enable: false
  timeout: 5
afterJoinGroup:
  enable: false
  timeout: 5
beforeInviteUserToGroup:
  enable: false
  timeout: 5
  failedContinue: true
afterSetGroupInfo:
  enable: false
  timeout: 5
beforeSetGroupInfo:
  enable: false
  timeout: 5
  failedContinue: true
afterSetGroupInfoEx:
  enable: false
  timeout: 5
beforeSetGroupInfoEx:
  enable: false
  timeout: 5
  failedContinue: true
afterRevokeMsg:
  enable: false
  timeout: 5
beforeAddBlack:
  enable: false
  timeout: 5
  failedContinue:
afterAddFriend:
  enable: false
  timeout: 5
beforeAddFriendAgree:
  enable: false
  timeout: 5
  failedContinue: true
afterAddFriendAgree:
  enable: false
  timeout: 5
afterDeleteFriend:
  enable: false
  timeout: 5
beforeImportFriends:
  enable: false
  timeout: 5
  failedContinue: true
afterImportFriends:
  enable: false
  timeout: 5
afterRemoveBlack:
  enable: false
  timeout: 5
beforeCreateSingleChatConversations:
  enable: false
  timeout: 5
  failedContinue: false
afterCreateSingleChatConversations:
  enable: false
  timeout: 5
  failedContinue: false
beforeCreateGroupChatConversations:
  enable: false
```

```
  timeout: 5
  failedContinue: false
afterCreateGroupChatConversations:
  enable: false
  timeout: 5
  failedContinue: false
```

config/grafana-template

config/grafana-template/Demo.json

```
{
  "__inputs": [
    {
      "name": "DS_PROMETHEUS",
      "label": "prometheus",
      "description": "",
      "type": "datasource",
      "pluginId": "prometheus",
      "pluginName": "Prometheus"
    }
  ],
  "__elements": {},
  "__requires": [
    {
      "type": "grafana",
      "id": "grafana",
      "name": "Grafana",
      "version": "11.0.1"
    },
    {
      "type": "datasource",
      "id": "prometheus",
      "name": "Prometheus",
      "version": "1.0.0"
    },
    {
      "type": "panel",
      "id": "timeseries",
      "name": "Time series",
      "version": ""
    }
  ],
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": {
          "type": "grafana",
          "uid": "-- Grafana --"
        },
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "fiscalYearStartMonth": 0,
  "graphTooltip": 0,
  "id": null,
  "links": [],
  "liveNow": false,
  "panels": [
    {
      "collapsed": false,
      "gridPos": {
        "h": 1,
        "w": 24,
        "x": 0,
```

```
    "y": 0
  },
  "id": 35,
  "panels": [],
  "title": "Server",
  "type": "row"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "Is the service up.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "stepBefore",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 2,
        "pointSize": 9,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
```

```
          ]
        },
        "unit": "bool_on_off"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 11,
      "w": 12,
      "x": 6,
      "y": 1
    },
    "id": 1,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "up",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "$legendName",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "UP",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of online users and login users within the time frame.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
```

```
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": [
      {
        "matcher": {
          "id": "byName",
          "options": "online users"
        },
        "properties": [
          {
            "id": "color",
            "value": {
              "fixedColor": "#37bbff",
              "mode": "fixed",
              "seriesBy": "last"
            }
          }
        ]
      }
    ]
  },
  "gridPos": {
    "h": 11,
    "w": 12,
```

```
      "x": 0,
      "y": 12
    },
    "id": 37,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "online_user_num",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "online users",
        "range": true,
        "refId": "A"
      },
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "expr": "increase(user_login_total[$time])",
        "hide": false,
        "instant": false,
        "legendFormat": "login num",
        "range": true,
        "refId": "B"
      }
    ],
    "title": "Login Information",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of register users within the time frame.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
```

```
      "axisLabel": "",
      "axisPlacement": "auto",
      "barAlignment": 0,
      "drawStyle": "line",
      "fillOpacity": 0,
      "gradientMode": "none",
      "hideFrom": {
        "legend": false,
        "tooltip": false,
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green",
          "value": null
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": [
    {
      "matcher": {
        "id": "byName",
        "options": "register users"
      },
      "properties": [
        {
          "id": "color",
          "value": {
            "fixedColor": "#7437ff",
            "mode": "fixed",
            "seriesBy": "last"
          }
        }
      ]
    }
  ]
```

```
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 12,
    "y": 12
  },
  "id": 59,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "user_register_total",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "register users",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Register num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of chat msg success.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
```

```
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green",
          "value": null
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 10,
  "w": 12,
  "x": 0,
  "y": 23
},
"id": 38,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
```

```
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "increase(single_chat_msg_process_success_total[$time])",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "single msgs",
      "range": true,
      "refId": "A"
    },
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "expr": "increase(group_chat_msg_process_success_total[$time])",
      "hide": false,
      "instant": false,
      "legendFormat": "group msgs",
      "range": true,
      "refId": "B"
    }
  ],
  "title": "Chat Msg Success Num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of chat msg failed .",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
```

```
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": [
      {
        "matcher": {
          "id": "byName",
          "options": "single msgs"
        },
        "properties": [
          {
            "id": "color",
            "value": {
              "fixedColor": "#ff00dc",
              "mode": "fixed",
              "seriesBy": "last"
            }
          }
        ]
      },
      {
        "matcher": {
          "id": "byName",
          "options": "group msgs"
        },
        "properties": [
          {
            "id": "color",
            "value": {
              "fixedColor": "#0cffef",
              "mode": "fixed"
            }
          }
        ]
      }
    ]
  },
  "gridPos": {
    "h": 10,
    "w": 12,
    "x": 12,
    "y": 23
  },
```

```
    "id": 39,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "increase(single_chat_msg_process_failed_total[$time])",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "single msgs",
        "range": true,
        "refId": "A"
      },
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "expr": "increase(group_chat_msg_process_failed_total[$time])",
        "hide": false,
        "instant": false,
        "legendFormat": "group msgs",
        "range": true,
        "refId": "B"
      }
    ],
    "title": "Chat Msg Failed Num",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of msg failed offline pushed.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
```

```
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": [
      {
        "matcher": {
          "id": "byName",
          "options": "failed msgs"
        },
        "properties": [
          {
            "id": "color",
            "value": {
              "fixedColor": "dark-red",
              "mode": "fixed",
              "seriesBy": "last"
            }
          }
        ]
      }
    ]
  },
  "gridPos": {
    "h": 11,
```

```
      "w": 8,
      "x": 0,
      "y": 33
    },
    "id": 42,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "increase(msg_offline_push_failed_total[$time])",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "addr:{{instance}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Msg Offline Push Failed Num",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of failed set seq.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
```

```
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": [
      {
        "matcher": {
          "id": "byName",
          "options": "failed msgs"
        },
        "properties": [
          {
            "id": "color",
            "value": {
              "fixedColor": "semi-dark-green",
              "mode": "fixed",
              "seriesBy": "last"
            }
          }
        ]
      }
    ]
  },
  "gridPos": {
    "h": 11,
    "w": 8,
    "x": 8,
    "y": 33
  },
  "id": 43,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
```

```
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "increase(seq_set_failed_total[$time])",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "addr: {{instance}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Seq Set Failed Num",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of messages that take a long time to send.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineStyle": {
            "fill": "solid"
          },
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
```

```
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green",
          "value": null
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": [
    {
      "matcher": {
        "id": "byName",
        "options": "failed msgs"
      },
      "properties": [
        {
          "id": "color",
          "value": {
            "fixedColor": "dark-red",
            "mode": "fixed",
            "seriesBy": "last"
          }
        }
      ]
    }
  ]
},
"gridPos": {
  "h": 11,
  "w": 8,
  "x": 16,
  "y": 33
},
"id": 60,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
```

```
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "msg_long_time_push_total",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "addr:{{instance}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Long Time Send Msg Total",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of successfully inserted messages.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
```

```
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 10,
    "w": 12,
    "x": 0,
    "y": 44
  },
  "id": 44,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "increase(msg_insert_redis_success_total[$time])",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "redis: {{instance}}",
      "range": true,
      "refId": "A"
    },
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "expr": "increase(msg_insert_mongo_success_total[$time])",
      "hide": false,
      "instant": false,
```

```
      "legendFormat": "mongo: {{instance}}",
      "range": true,
      "refId": "B"
    }
  ],
  "title": "Msg Success Insert Num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of failed insertion messages.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
```

```
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 10,
    "w": 12,
    "x": 12,
    "y": 44
  },
  "id": 45,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "increase(msg_insert_redis_failed_total[$time])",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "redis: {{instance}}",
      "range": true,
      "refId": "A"
    },
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "expr": "increase(msg_insert_mongo_failed_total[$time])",
      "hide": false,
      "instant": false,
      "legendFormat": "mongo: {{instance}}",
      "range": true,
      "refId": "B"
    }
  ],
  "title": "Msg Failed Insert Num",
  "type": "timeseries"
},
{
  "collapsed": true,
  "gridPos": {
    "h": 1,
    "w": 24,
```

```
    "x": 0,
    "y": 54
  },
  "id": 22,
  "panels": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "description": "This metric represents the number of call of all API.",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisBorderShow": false,
            "axisCenteredZero": false,
            "axisColorMode": "text",
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "insertNulls": false,
            "lineInterpolation": "linear",
            "lineStyle": {
              "fill": "solid"
            },
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "none"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "fieldMinMax": false,
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green"
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "none"
```

```
      },
      "overrides": []
    },
    "gridPos": {
      "h": 9,
      "w": 12,
      "x": 0,
      "y": 13
    },
    "id": 29,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "right",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum by (path) (api_count)",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "__auto",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "API Requests Total",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of call of all API within the time frame.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
```

```
      "legend": false,
      "tooltip": false,
      "viz": false
    },
    "insertNulls": false,
    "lineInterpolation": "linear",
    "lineStyle": {
      "fill": "solid"
    },
    "lineWidth": 1,
    "pointSize": 5,
    "scaleDistribution": {
      "type": "linear"
    },
    "showPoints": "auto",
    "spanNulls": false,
    "stacking": {
      "group": "A",
      "mode": "none"
    },
    "thresholdsStyle": {
      "mode": "off"
    }
  },
  "fieldMinMax": false,
  "mappings": [],
  "thresholds": {
    "mode": "absolute",
    "steps": [
      {
        "color": "green"
      },
      {
        "color": "red",
        "value": 80
      }
    ]
  },
  "unit": "none"
},
"overrides": [
  {
    "__systemRef": "hideSeriesFrom",
    "matcher": {
      "id": "byNames",
      "options": {
        "mode": "exclude",
        "names": [
          "/friend/get_friend_list"
        ],
        "prefix": "All except:",
        "readOnly": true
      }
    },
    "properties": [
      {
        "id": "custom.hideFrom",
        "value": {
          "legend": false,
          "tooltip": false,
          "viz": true
        }
      }
    ]
  }
]
```

```
  },
  "gridPos": {
    "h": 9,
    "w": 12,
    "x": 12,
    "y": 13
  },
  "id": 48,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "right",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (path) (increase(api_count[$time]))",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "API Requests Num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of err return of API.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
```

```
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 14,
  "w": 12,
  "x": 0,
  "y": 22
},
"id": 24,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "right",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
```

```
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (path) (api_count{code != \"0\"})",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "API Error Total",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of err return of API with err code.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
```

```
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 14,
    "w": 12,
    "x": 12,
    "y": 22
  },
  "id": 23,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "right",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (path, code) (api_count{code != \"0\"})",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "{{path}}: code={{code}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "API Error Total With Code",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the qps of API.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
```

```
      "axisBorderShow": false,
      "axisCenteredZero": false,
      "axisColorMode": "text",
      "axisLabel": "",
      "axisPlacement": "auto",
      "barAlignment": 0,
      "drawStyle": "line",
      "fillOpacity": 0,
      "gradientMode": "none",
      "hideFrom": {
        "legend": false,
        "tooltip": false,
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "reqps"
  },
  "overrides": [
    {
      "matcher": {
        "id": "byName",
        "options": "Value"
      },
      "properties": [
        {
          "id": "color",
          "value": {
            "fixedColor": "#1ed9d4",
            "mode": "fixed"
          }
        }
      ]
    }
```

```
      ]
    },
    "gridPos": {
      "h": 9,
      "w": 24,
      "x": 0,
      "y": 36
    },
    "id": 51,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum(rate(api_count[1m]))",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "qps",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "API QPS",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of err return of API within the time frame.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
```

```
        "tooltip": false,
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 12,
  "w": 12,
  "x": 0,
  "y": 45
},
"id": 49,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "right",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
```

```
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (path) (increase(api_count{code != \"0\"}[$time]))",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "API Error Num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of err return of API with err code within the time frame
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
```

```
            "steps": [
              {
                "color": "green"
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "none"
        },
        "overrides": []
      },
      "gridPos": {
        "h": 12,
        "w": 12,
        "x": 12,
        "y": 45
      },
      "id": 50,
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "right",
          "showLegend": true
        },
        "tooltip": {
          "maxHeight": 600,
          "mode": "single",
          "sort": "none"
        }
      },
      "targets": [
        {
          "datasource": {
            "type": "prometheus",
            "uid": "${DS_PROMETHEUS}"
          },
          "editorMode": "code",
          "exemplar": false,
          "expr": "sum by (path, code) (increase(api_count{code != \"0\"}[$time]))",
          "format": "time_series",
          "hide": false,
          "instant": false,
          "interval": "",
          "legendFormat": "{{path}}: code={{code}}",
          "range": true,
          "refId": "A"
        }
      ],
      "title": "API Error Num With Code",
      "type": "timeseries"
    }
  ],
  "title": "API",
  "type": "row"
},
{
  "collapsed": true,
  "gridPos": {
    "h": 1,
    "w": 24,
    "x": 0,
    "y": 55
```

```
},
"id": 28,
"panels": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of call of all RPC.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineStyle": {
            "fill": "solid"
          },
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": false,
          "stacking": {
            "group": "A",
            "mode": "none"
          },
          "thresholdsStyle": {
            "mode": "off"
          }
        },
        "fieldMinMax": false,
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green"
            },
            {
              "color": "red",
              "value": 80
            }
          ]
        },
        "unit": "none"
      },
      "overrides": []
```

```
  },
  "gridPos": {
    "h": 10,
    "w": 24,
    "x": 0,
    "y": 14
  },
  "id": 21,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "right",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (path) (rpc_count)",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "RPC Total Count",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the error return of RPC.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
```

```
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 10,
  "w": 12,
  "x": 0,
  "y": 24
},
"id": 31,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "right",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
```

```
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (path) (rpc_count{code!=\"0\"})",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "RPC Error Count",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the error return of RPC with code.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
```

```
            {
              "color": "green"
            },
            {
              "color": "red",
              "value": 80
            }
          ]
        },
        "unit": "none"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 10,
      "w": 12,
      "x": 12,
      "y": 24
    },
    "id": 33,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "right",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum by (path, code) (rpc_count{code!=\"0\"})",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{path}}: code={{code}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "RPC Error Count With Code",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of call of all RPC within the time frame.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
```

```
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 9,
    "w": 24,
    "x": 0,
    "y": 34
  },
  "id": 52,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "right",
      "showLegend": true
    },
```

```
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum by (path) (increase(rpc_count[$time]))",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "__auto",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "RPC Total Num",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of RPC calls within the time frame, aggregated by name."
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineStyle": {
            "fill": "solid"
          },
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": false,
          "stacking": {
```

```
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 13,
    "w": 12,
    "x": 0,
    "y": 43
  },
  "id": 30,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (name) (increase(rpc_count[$time]))",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "RPC Num by Name",
  "type": "timeseries"
},
```

```
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of call of RPC within the time frame, aggregated by addr
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 13,
```

```
      "w": 12,
      "x": 12,
      "y": 43
    },
    "id": 32,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum by (instance) (increase(rpc_count[$time]))",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "__auto",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "RPC Num by Address",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the error return of RPC within the time frame within the time frame
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
```

```
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 10,
  "w": 12,
  "x": 0,
  "y": 56
},
"id": 54,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "right",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "editorMode": "code",
    "exemplar": false,
    "expr": "sum by (path) (increase(rpc_count{code!=\"0\"}[$time]))",
```

```
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "__auto",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "RPC Error Num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the error return of RPC with code within the time frame within the
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
```

```
              {
                "color": "red",
                "value": 80
              }
            ]
          },
          "unit": "none"
        },
        "overrides": []
      },
      "gridPos": {
        "h": 10,
        "w": 12,
        "x": 12,
        "y": 56
      },
      "id": 53,
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "right",
          "showLegend": true
        },
        "tooltip": {
          "maxHeight": 600,
          "mode": "single",
          "sort": "none"
        }
      },
      "targets": [
        {
          "datasource": {
            "type": "prometheus",
            "uid": "${DS_PROMETHEUS}"
          },
          "editorMode": "code",
          "exemplar": false,
          "expr": "sum by (path, code) (increase(rpc_count{code!=\"0\"}[$time]))",
          "format": "time_series",
          "hide": false,
          "instant": false,
          "interval": "",
          "legendFormat": "{{path}}: code={{code}}",
          "range": true,
          "refId": "A"
        }
      ],
      "title": "RPC Error Num With Code",
      "type": "timeseries"
    }
  ],
  "title": "RPC",
  "type": "row"
},
{
  "collapsed": true,
  "gridPos": {
    "h": 1,
    "w": 24,
    "x": 0,
    "y": 56
  },
  "id": 25,
  "panels": [
    {
```

```
"datasource": {
  "type": "prometheus",
  "uid": "${DS_PROMETHEUS}"
},
"description": "This metric represents the number of HTTP requests.",
"fieldConfig": {
  "defaults": {
    "color": {
      "mode": "palette-classic"
    },
    "custom": {
      "axisBorderShow": false,
      "axisCenteredZero": false,
      "axisColorMode": "text",
      "axisLabel": "",
      "axisPlacement": "auto",
      "barAlignment": 0,
      "drawStyle": "line",
      "fillOpacity": 0,
      "gradientMode": "none",
      "hideFrom": {
        "legend": false,
        "tooltip": false,
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
```

```
      "x": 0,
      "y": 15
    },
    "id": 27,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "right",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum by (method, path) (http_count)",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{method}}: {{path}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "HTTP Total Count",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of HTTP requests with status.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
```

```
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
  "x": 12,
  "y": 15
},
"id": 26,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "right",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "editorMode": "code",
    "exemplar": false,
    "expr": "sum by (method, path, status) (http_count)",
    "format": "time_series",
```

```
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "{{method}}: {{path}}: {{status}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "HTTP Total Count With Status",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of HTTP requests within the time frame.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
```

```
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 0,
    "y": 26
  },
  "id": 55,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "right",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "sum by (method, path) (increase(http_count[$time]))",
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "{{method}}: {{path}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "HTTP Total Num",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of HTTP requests with status within the time frame.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
```

```
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 12,
    "y": 26
  },
  "id": 56,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "right",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
```

```
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "sum by (method, path, status) (increase(http_count[$time]))",
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{method}}: {{path}}: {{status}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "HTTP Total Num With Status",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the qps of HTTP.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineStyle": {
            "fill": "solid"
          },
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": false,
          "stacking": {
            "group": "A",
            "mode": "none"
          },
          "thresholdsStyle": {
```

```
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "reqps"
  },
  "overrides": [
    {
      "matcher": {
        "id": "byName",
        "options": "Value"
      },
      "properties": [
        {
          "id": "color",
          "value": {
            "fixedColor": "#1ed9d4",
            "mode": "fixed"
          }
        }
      ]
    }
  ]
},
"gridPos": {
  "h": 9,
  "w": 24,
  "x": 0,
  "y": 37
},
"id": 57,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "editorMode": "code",
    "exemplar": false,
    "expr": "sum(rate(http_count[1m]))",
```

```
          "format": "time_series",
          "hide": false,
          "instant": false,
          "interval": "",
          "legendFormat": "qps",
          "range": true,
          "refId": "A"
        }
      ],
      "title": "HTTP QPS",
      "type": "timeseries"
    }
  ],
  "title": "HTTP",
  "type": "row"
},
{
  "collapsed": true,
  "gridPos": {
    "h": 1,
    "w": 24,
    "x": 0,
    "y": 57
  },
  "id": 6,
  "panels": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "description": "This metric represents the proportion of CPU runtime within 1 second. It is calculated as
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisBorderShow": false,
            "axisCenteredZero": false,
            "axisColorMode": "text",
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "insertNulls": false,
            "lineInterpolation": "linear",
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "none"
            },
            "thresholdsStyle": {
```

```
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "percent"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 0,
    "y": 5
  },
  "id": 5,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "pluginVersion": "10.3.7",
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "label_replace(\r\n  rate(process_cpu_seconds_total{job=~\"$rpcNameFilter\"}[1m])*100,\r\n  \"
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "{{job}}: {{instance}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "CPU Usage Percentage",
  "type": "timeseries"
},
{
  "datasource": {
```

```
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the proportion of CPU runtime within 1 second. It is calculated as
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "percent"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 12,
    "y": 5
  },
```

```
    "id": 4,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "pluginVersion": "10.3.7",
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  rate(process_cpu_seconds_total{job!~\"$rpcNameFilter\"}[1m])*100,\r\n  \"
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{job}}: {{instance}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "CPU Usage Percentage",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of open file descriptors.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineWidth": 1,
          "pointSize": 5,
```

```
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green",
          "value": null
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "none"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
  "x": 0,
  "y": 16
},
"id": 7,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"pluginVersion": "10.3.7",
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "editorMode": "code",
    "exemplar": false,
    "expr": "label_replace(\r\n  process_open_fds{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r\n  \"$1\",\r\n
    "format": "time_series",
    "hide": false,
    "instant": false,
    "interval": "",
```

```
      "legendFormat": "{{job}}: {{instance}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Open File Descriptors",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of open file descriptors.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
```

```
        "unit": "none"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 11,
      "w": 12,
      "x": 12,
      "y": 16
    },
    "id": 8,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "pluginVersion": "10.3.7",
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  process_open_fds{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r\n  \"$1\",\r\n
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{job}}: {{instance}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Open File Descriptors",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of process virtual memory bytes.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
```

```
      "gradientMode": "none",
      "hideFrom": {
        "legend": false,
        "tooltip": false,
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green",
          "value": null
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "bytes"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
  "x": 0,
  "y": 27
},
"id": 9,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"pluginVersion": "10.3.7",
"targets": [
  {
    "datasource": {
```

```
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "label_replace(\r\n  process_virtual_memory_bytes{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r\n
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "{{job}}: {{instance}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Virtual Memory bytes",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of process virtual memory bytes.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
```

```
          {
            "color": "green",
            "value": null
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "bytes"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 12,
    "y": 27
  },
  "id": 10,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "pluginVersion": "10.3.7",
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "label_replace(\r\n  process_virtual_memory_bytes{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r\n
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "{{job}}: {{instance}}",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Virtual Memory bytes",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of process resident memory bytes.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
```

```
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "bytes"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 0,
    "y": 38
  },
  "id": 11,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
```

```
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "pluginVersion": "10.3.7",
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  process_resident_memory_bytes{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r\n
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{job}}: {{instance}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Resident Memory bytes",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of process resident memory bytes.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": false,
          "stacking": {
            "group": "A",
            "mode": "none"
          },
```

```
          "thresholdsStyle": {
            "mode": "off"
          }
        },
        "fieldMinMax": false,
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green"
            },
            {
              "color": "red",
              "value": 80
            }
          ]
        },
        "unit": "bytes"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 11,
      "w": 12,
      "x": 12,
      "y": 38
    },
    "id": 12,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "pluginVersion": "10.3.7",
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  process_resident_memory_bytes{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r\n
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "{{job}}: {{instance}}",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Resident Memory bytes",
    "type": "timeseries"
  }
],
"title": "Process",
```

```
    "type": "row"
  },
  {
    "collapsed": true,
    "gridPos": {
      "h": 1,
      "w": 24,
      "x": 0,
      "y": 58
    },
    "id": 3,
    "panels": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "description": "Measures the frequency of garbage collection operations in the Go environment, averaged ov
        "fieldConfig": {
          "defaults": {
            "color": {
              "mode": "palette-classic"
            },
            "custom": {
              "axisBorderShow": false,
              "axisCenteredZero": false,
              "axisColorMode": "text",
              "axisLabel": "",
              "axisPlacement": "auto",
              "barAlignment": 0,
              "drawStyle": "line",
              "fillOpacity": 0,
              "gradientMode": "none",
              "hideFrom": {
                "legend": false,
                "tooltip": false,
                "viz": false
              },
              "insertNulls": false,
              "lineInterpolation": "linear",
              "lineStyle": {
                "fill": "solid"
              },
              "lineWidth": 1,
              "pointSize": 5,
              "scaleDistribution": {
                "type": "linear"
              },
              "showPoints": "auto",
              "spanNulls": false,
              "stacking": {
                "group": "A",
                "mode": "none"
              },
              "thresholdsStyle": {
                "mode": "off"
              }
            },
            "fieldMinMax": false,
            "mappings": [],
            "thresholds": {
              "mode": "absolute",
              "steps": [
                {
                  "color": "green"
                },
```

```
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "s"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 0,
    "y": 6
  },
  "id": 58,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "label_replace(\r\n  rate(go_gc_duration_seconds_count{job=~\"$rpcNameFilter\"}[5m]),\r\n  \"j
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "$legendName",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "GC Rate Per Second",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "Measures the frequency of garbage collection operations in the Go environment, averaged ov
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
```

```
      "axisLabel": "",
      "axisPlacement": "auto",
      "barAlignment": 0,
      "drawStyle": "line",
      "fillOpacity": 0,
      "gradientMode": "none",
      "hideFrom": {
        "legend": false,
        "tooltip": false,
        "viz": false
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "s"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
  "x": 12,
  "y": 6
},
"id": 2,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
```

```
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "expr": "label_replace(\r\n  rate(go_gc_duration_seconds_count{job!~\"$rpcNameFilter\"}[5m]),\r\n  \"j
      "hide": false,
      "instant": false,
      "legendFormat": "$legendName",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "GC Rate Per Second",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of goroutines.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
```

```
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 0,
    "y": 17
  },
  "id": 13,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "label_replace(\r\n  go_goroutines{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r\n  \"$1\",\r\n  \
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "$legendName",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Goroutines",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of goroutines.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
```

```
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "none"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 12,
    "y": 17
  },
  "id": 14,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
```

```
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  go_goroutines{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r\n  \"$1\",\r\n  \
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "$legendName",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Goroutines",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of bytes allocated and still in use.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineStyle": {
            "fill": "solid"
          },
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
```

```
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "bytes"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
  "x": 0,
  "y": 28
},
"id": 15,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "editorMode": "code",
    "exemplar": false,
    "expr": "label_replace(\r\n  go_memstats_alloc_bytes{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r\n  \"$1
    "format": "time_series",
    "hide": false,
    "instant": false,
    "interval": "",
    "legendFormat": "$legendName",
    "range": true,
    "refId": "A"
  }
],
"title": "Go Alloc Bytes ",
```

```
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of bytes allocated and still in use.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "bytes"
    },
    "overrides": []
  },
```

```
    "gridPos": {
      "h": 11,
      "w": 12,
      "x": 12,
      "y": 28
    },
    "id": 16,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  go_memstats_alloc_bytes{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r\n  \"$1
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "$legendName",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Go Alloc Bytes ",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of bytes used by the profiling bucket hash table.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
```

```
      },
      "insertNulls": false,
      "lineInterpolation": "linear",
      "lineStyle": {
        "fill": "solid"
      },
      "lineWidth": 1,
      "pointSize": 5,
      "scaleDistribution": {
        "type": "linear"
      },
      "showPoints": "auto",
      "spanNulls": false,
      "stacking": {
        "group": "A",
        "mode": "none"
      },
      "thresholdsStyle": {
        "mode": "off"
      }
    },
    "fieldMinMax": false,
    "mappings": [],
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {
          "color": "green"
        },
        {
          "color": "red",
          "value": 80
        }
      ]
    },
    "unit": "bytes"
  },
  "overrides": []
},
"gridPos": {
  "h": 11,
  "w": 12,
  "x": 0,
  "y": 39
},
"id": 17,
"options": {
  "legend": {
    "calcs": [],
    "displayMode": "list",
    "placement": "bottom",
    "showLegend": true
  },
  "tooltip": {
    "maxHeight": 600,
    "mode": "single",
    "sort": "none"
  }
},
"targets": [
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "editorMode": "code",
```

```
      "exemplar": false,
      "expr": "label_replace(\r\n  go_memstats_buck_hash_sys_bytes{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "$legendName",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Go Buck Hash Sys Bytes ",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of bytes used by the profiling bucket hash table.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
```

```
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "bytes"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 12,
    "y": 39
  },
  "id": 18,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
      "maxHeight": 600,
      "mode": "single",
      "sort": "none"
    }
  },
  "targets": [
    {
      "datasource": {
        "type": "prometheus",
        "uid": "${DS_PROMETHEUS}"
      },
      "editorMode": "code",
      "exemplar": false,
      "expr": "label_replace(\r\n  go_memstats_buck_hash_sys_bytes{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r
      "format": "time_series",
      "hide": false,
      "instant": false,
      "interval": "",
      "legendFormat": "$legendName",
      "range": true,
      "refId": "A"
    }
  ],
  "title": "Go Buck Hash Sys Bytes ",
  "type": "timeseries"
},
{
  "datasource": {
    "type": "prometheus",
    "uid": "${DS_PROMETHEUS}"
  },
  "description": "This metric represents the number of bytes in use by mcache structures.",
  "fieldConfig": {
    "defaults": {
      "color": {
        "mode": "palette-classic"
      },
      "custom": {
        "axisBorderShow": false,
```

```
        "axisCenteredZero": false,
        "axisColorMode": "text",
        "axisLabel": "",
        "axisPlacement": "auto",
        "barAlignment": 0,
        "drawStyle": "line",
        "fillOpacity": 0,
        "gradientMode": "none",
        "hideFrom": {
          "legend": false,
          "tooltip": false,
          "viz": false
        },
        "insertNulls": false,
        "lineInterpolation": "linear",
        "lineStyle": {
          "fill": "solid"
        },
        "lineWidth": 1,
        "pointSize": 5,
        "scaleDistribution": {
          "type": "linear"
        },
        "showPoints": "auto",
        "spanNulls": false,
        "stacking": {
          "group": "A",
          "mode": "none"
        },
        "thresholdsStyle": {
          "mode": "off"
        }
      },
      "fieldMinMax": false,
      "mappings": [],
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {
            "color": "green"
          },
          {
            "color": "red",
            "value": 80
          }
        ]
      },
      "unit": "bytes"
    },
    "overrides": []
  },
  "gridPos": {
    "h": 11,
    "w": 12,
    "x": 0,
    "y": 50
  },
  "id": 19,
  "options": {
    "legend": {
      "calcs": [],
      "displayMode": "list",
      "placement": "bottom",
      "showLegend": true
    },
    "tooltip": {
```

```
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  go_memstats_mcache_inuse_bytes{job=~\"$rpcNameFilter\"},\r\n  \"job\",\r\
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "$legendName",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Go Mcache Bytes",
    "type": "timeseries"
  },
  {
    "datasource": {
      "type": "prometheus",
      "uid": "${DS_PROMETHEUS}"
    },
    "description": "This metric represents the number of bytes in use by mcache structures.",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "palette-classic"
        },
        "custom": {
          "axisBorderShow": false,
          "axisCenteredZero": false,
          "axisColorMode": "text",
          "axisLabel": "",
          "axisPlacement": "auto",
          "barAlignment": 0,
          "drawStyle": "line",
          "fillOpacity": 0,
          "gradientMode": "none",
          "hideFrom": {
            "legend": false,
            "tooltip": false,
            "viz": false
          },
          "insertNulls": false,
          "lineInterpolation": "linear",
          "lineStyle": {
            "fill": "solid"
          },
          "lineWidth": 1,
          "pointSize": 5,
          "scaleDistribution": {
            "type": "linear"
          },
          "showPoints": "auto",
          "spanNulls": false,
          "stacking": {
            "group": "A",
```

```
            "mode": "none"
          },
          "thresholdsStyle": {
            "mode": "off"
          }
        },
        "fieldMinMax": false,
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green"
            },
            {
              "color": "red",
              "value": 80
            }
          ]
        },
        "unit": "bytes"
      },
      "overrides": []
    },
    "gridPos": {
      "h": 11,
      "w": 12,
      "x": 12,
      "y": 50
    },
    "id": 20,
    "options": {
      "legend": {
        "calcs": [],
        "displayMode": "list",
        "placement": "bottom",
        "showLegend": true
      },
      "tooltip": {
        "maxHeight": 600,
        "mode": "single",
        "sort": "none"
      }
    },
    "targets": [
      {
        "datasource": {
          "type": "prometheus",
          "uid": "${DS_PROMETHEUS}"
        },
        "editorMode": "code",
        "exemplar": false,
        "expr": "label_replace(\r\n  go_memstats_mcache_inuse_bytes{job!~\"$rpcNameFilter\"},\r\n  \"job\",\r\
        "format": "time_series",
        "hide": false,
        "instant": false,
        "interval": "",
        "legendFormat": "$legendName",
        "range": true,
        "refId": "A"
      }
    ],
    "title": "Go Mcache Bytes",
    "type": "timeseries"
  }
],
```

```
    "title": "GO infomation",
    "type": "row"
  }
],
"refresh": "5s",
"schemaVersion": 39,
"tags": [],
"templating": {
  "list": [
    {
      "current": {
        "selected": false,
        "text": "openimserver-openim-rpc.*",
        "value": "openimserver-openim-rpc.*"
      },
      "hide": 0,
      "includeAll": false,
      "label": "filter",
      "multi": false,
      "name": "rpcNameFilter",
      "options": [
        {
          "selected": true,
          "text": "openimserver-openim-rpc.*",
          "value": "openimserver-openim-rpc.*"
        }
      ],
      "query": "openimserver-openim-rpc.*",
      "queryValue": "",
      "skipUrlSync": false,
      "type": "custom"
    },
    {
      "current": {
        "selected": false,
        "text": "{{job}}: {{instance}}",
        "value": "{{job}}: {{instance}}"
      },
      "description": "common legend name",
      "hide": 0,
      "includeAll": false,
      "label": "legend",
      "multi": false,
      "name": "legendName",
      "options": [
        {
          "selected": true,
          "text": "{{job}}: {{instance}}",
          "value": "{{job}}: {{instance}}"
        }
      ],
      "query": "{{job}}: {{instance}}",
      "queryValue": "",
      "skipUrlSync": false,
      "type": "custom"
    },
    {
      "current": {
        "selected": false,
        "text": "5m",
        "value": "5m"
      },
      "description": "Global promQL time range.",
      "hide": 0,
      "includeAll": false,
      "label": "time",
```

```
"multi": false,
"name": "time",
"options": [
  {
    "selected": false,
    "text": "1m",
    "value": "1m"
  },
  {
    "selected": true,
    "text": "5m",
    "value": "5m"
  },
  {
    "selected": false,
    "text": "30m",
    "value": "30m"
  },
  {
    "selected": false,
    "text": "1h",
    "value": "1h"
  },
  {
    "selected": false,
    "text": "3h",
    "value": "3h"
  },
  {
    "selected": false,
    "text": "6h",
    "value": "6h"
  },
  {
    "selected": false,
    "text": "12h",
    "value": "12h"
  },
  {
    "selected": false,
    "text": "24h",
    "value": "24h"
  },
  {
    "selected": false,
    "text": "1w",
    "value": "1w"
  },
  {
    "selected": false,
    "text": "4w",
    "value": "4w"
  },
  {
    "selected": false,
    "text": "12w",
    "value": "12w"
  },
  {
    "selected": false,
    "text": "24w",
    "value": "24w"
  },
  {
    "selected": false,
    "text": "1y",
```

```
            "value": "1y"
          },
          {
            "selected": false,
            "text": "2y",
            "value": "2y"
          },
          {
            "selected": false,
            "text": "4y",
            "value": "4y"
          },
          {
            "selected": false,
            "text": "10y",
            "value": "10y"
          }
        ],
        "query": "1m,5m,30m,1h,3h,6h,12h,24h,1w,4w,12w,24w,1y,2y,4y,10y",
        "queryValue": "",
        "skipUrlSync": false,
        "type": "custom"
      }
    ]
  },
  "time": {
    "from": "now-15m",
    "to": "now"
  },
  "timeRangeUpdatedDuringEditOrView": false,
  "timepicker": {},
  "timezone": "",
  "title": "Demo",
  "uid": "a506d250-b606-4702-86a7-ac6aa1d069a1",
  "version": 2,
  "weekStart": ""
}
```