Golang Practice

M800 Libs Intro and Guidance

Common Libs

- goctx: wrapped context with Map(), m800 header and log const
- gopkg: code error
- m800log: goctx integrated logrus, m800 log format
- gotrace: goctx integrated jaeger client lib
- gin-prometheus: pre-defined gin metrics middleware
- intercom: internal service interaction, gin and http utils

Database Libs

- kafkautil: worker-pool model consumer and producer practice.
  - producer ack error handle
- mgopool: mgo session pool management, metrics and tracing integration
- redispool: reids-sentinel integration and high level function wrap

goctx

- include native context method
- log key and http key mapping (to reduce log size)
  - i.e., `x-correlation-id` to `cid`
- ctx.GetCID() will generate cid if no cid data

```
ctx := goctx.Background()
ctx := intercom.GetContextFromGin(c)

headerMap := ctx.HeaderKeyMap()
ctx := goctx.GetContextFromHeaderKeyMap(headerMap)

dataMap := ctx.Map()
ctx := goctx.GetContextFromMap(dataMap)
```

gopkg

- m800 defined 7-digit error code

```
type CodeError interface {
        ErrorCode() int
        Error() string
        FullError() string
}

func NewCodeError(code int, message string) CodeError

// i.e.,
err := gopkg.NewCodeError(1234567,"test error msg")
```

m800log

- init: set log level, output and format
- use logf* rather than log(xxx, fmt.Sprintf) to improve performance
- info used in initialization, otherwise, debug and error level

```
m800log.Infof(ctx, "Gin Server is running and listening port: %s", port)

m800log.Debugf(ctx, "[checkAckSeq] room seq: %+v", seq)

m800log.Errorf(ctx, "[crossPub] downstream region:%s grpc error:%v", region, err)
```

gotrace

- it's about distributed request lifecycle
- gotracev2: https://docs.google.com/presentation/d/1n19pOzb-emAgIkjxx5zg5KWvAQfSPm6S1TkjuUavkKE/edit?usp=sharing
- tracing concept intro: https://docs.google.com/presentation/d/1KmRmuamLaQmRxQF_bYX33bMA7EwSW1IbwFPtuWeI9eI/edit?usp=sharing

gotrace example

```
sp := gotrace.CreateChildOfSpan(ctx, spanFinishingTouchEvent)
defer sp.Finish()
-----
sp := gotrace.CreateFollowsFromSpan(ctx, spanDownstreamEvent)
defer sp.Finish()
sp.SetTag(tagNs, ns)
```

gin-prometheus

- provide default metrics in our gin HTTP server
  - request count
  - request durations
  - request and response size

intercom Gin Middleware

- M800Recovery: recover goroutine and return unified M800 response
- AccessMiddleware: log req and resp, set timeout adnd init tracing
- CrossRegionMiddleware: proxy request to service home region by `x-m800-svc-home`
- BanAnonymousMiddleware: return error if `x-m800-usr-anms` header is true

intercom Gin unified http response

```
func GinAllErrorResponse(c *gin.Context, result interface{}, err gopkg.CodeError)
func GinAllResponse(c *gin.Context, result interface{}, err gopkg.CodeError)
func GinError(c *gin.Context, err gopkg.CodeError)
func GinErrorCodeMsg(c *gin.Context, code int, msg string)
func GinOKError(c *gin.Context, err gopkg.CodeError)
func GinOKListResponse(c *gin.Context, result interface{}, total, offset, count int)
func GinOKResponse(c *gin.Context, result interface{})
```

Set HTTP Error code by m800 error code

```
intercom.ErrorHttpStatusMapping.Set(1234567, 400)
```

intercom http client

```
func HTTPDo(ctx goctx.Context, req *http.Request) (resp *http.Response, err gopkg.CodeError)
func HTTPDoGivenBody(ctx goctx.Context, req *http.Request, body []byte) (resp *http.Response, err gopkg.CodeError)
func HTTPNewRequest(ctx goctx.Context, method, url string, body io.Reader) (*http.Request, gopkg.CodeError)
func HTTPPostForm(ctx goctx.Context, url string, data url.Values) (resp *http.Response, err gopkg.CodeError)
func M800Do(ctx goctx.Context, req *http.Request) (result *JsonResponse, err gopkg.CodeError)
```

HTTP Client flexibility

```
// set squid proxy...
func SetHTTPClient(client *http.Client)
func SetHTTPClientTimeout(to time.Duration)
```

intercom utils

```
// json related
func ParseJSON(ctx goctx.Context, data []byte, v interface{}) gopkg.CodeError
func ParseJSONGin(ctx goctx.Context, c *gin.Context, v interface{}) gopkg.CodeError
func ParseJSONReadCloser(ctx goctx.Context, readCloser io.ReadCloser, v interface{}) gopkg.CodeError
func ParseJSONReq(ctx goctx.Context, req *http.Request, v interface{}) gopkg.CodeError
// info
func PrintGinRouteInfo(rs []gin.RouteInfo)
// read closer
func ReadFromReadCloser(readCloser io.ReadCloser) ([]byte, gopkg.CodeError)

// http server unit test
req, _ = http.NewRequest(http.MethodGet, path, nil)
resp = intercom.CreateTestResponseRecorder()
routerTest.ServeHTTP(resp, req)
```

mgopool

Use library default static function (package object) or new an pool object.

```
// static function which uses default object in package
mgopool.Initialize(config)
err = mgopool.QueryAll(ctx, dbPlugin, CollectionCommand, &result, selector, nil, 0, 0)

// new pool object
globalPool, errGlobal := mgopool.NewSessionPool(globalInfo)
localPool, errLocal := mgopool.NewSessionPool(localInfo)

err = localPool.QueryAll(ctx, dbPlugin, CollectionCommand, &result, selector, nil, 0, 0)
```

redispool

```
rPool, err := redispool.NewPool(redisConf)
// ...

rPool.Setex("key", "10", "value")
```

kafkautil init

```
if err := kafkautil.InitKafkaProducer(pCtx, pConf, nil); err != nil {
    return err
}

if err := kafkautil.InitKafkaConsumer(cCtx, cConf, []string{"topic1","topic2"}); err != nil {
    return err
}

defer kafkautil.ConsumerClose()
defer kafkautil.ProducerClose()
```

kafkautil

```
// producer
err := kafkautil.Publish("topic1", []byte("key"), data, headers)

// consumer
// DispatchKafkaJob is handler to consume event
kafkautil.ReadKafkaEvents(kafkaWorkers, kafkaQueueSize, DispatchKafkaJob)

func DispatchKafkaJob(ev *kafka.Message) {
        switch *ev.TopicPartition.Topic {
        case roomEventTopic:
                RoomEventHandle(ev)
        case udomain.UIMS_LOGOUT_KAFKA_TOPIC:
                LogoutEventHandle(ev)
        }
}
```