石墨文档GO在K8S上微服务的实践

×

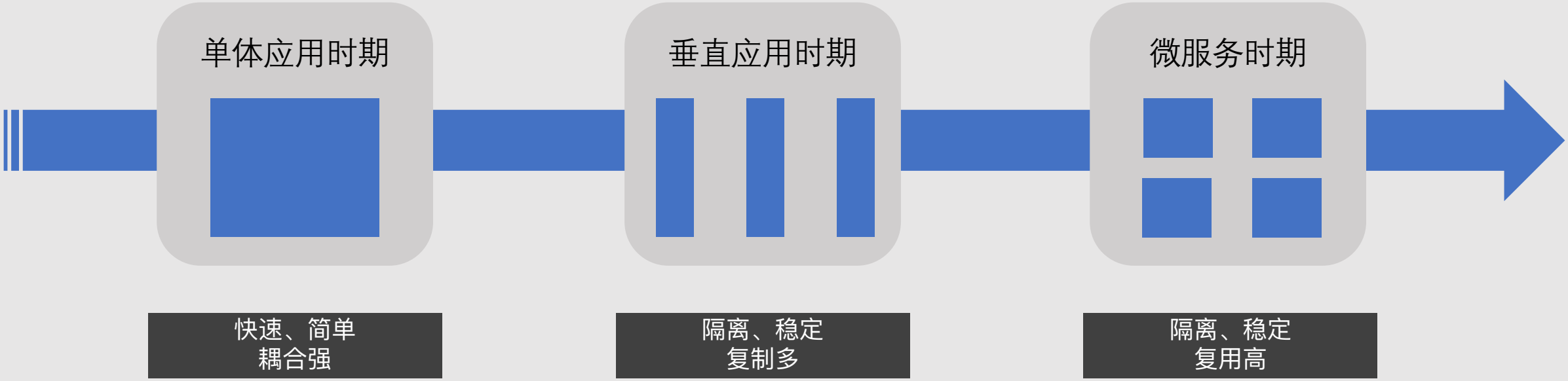
彭友顺

石墨文档
基础设施负责人

目录

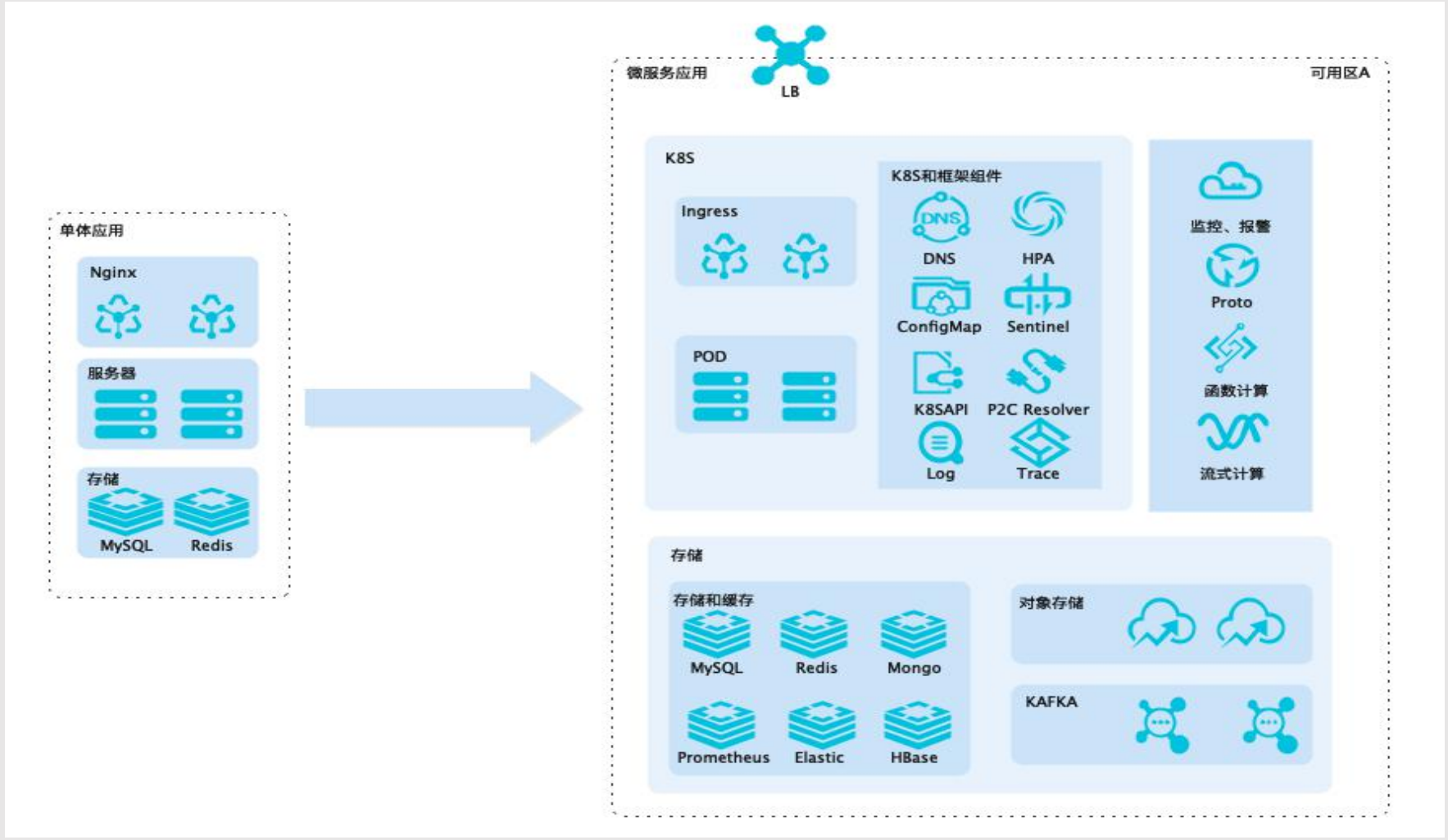
第一部分 架构演进

架构演进



架构演进

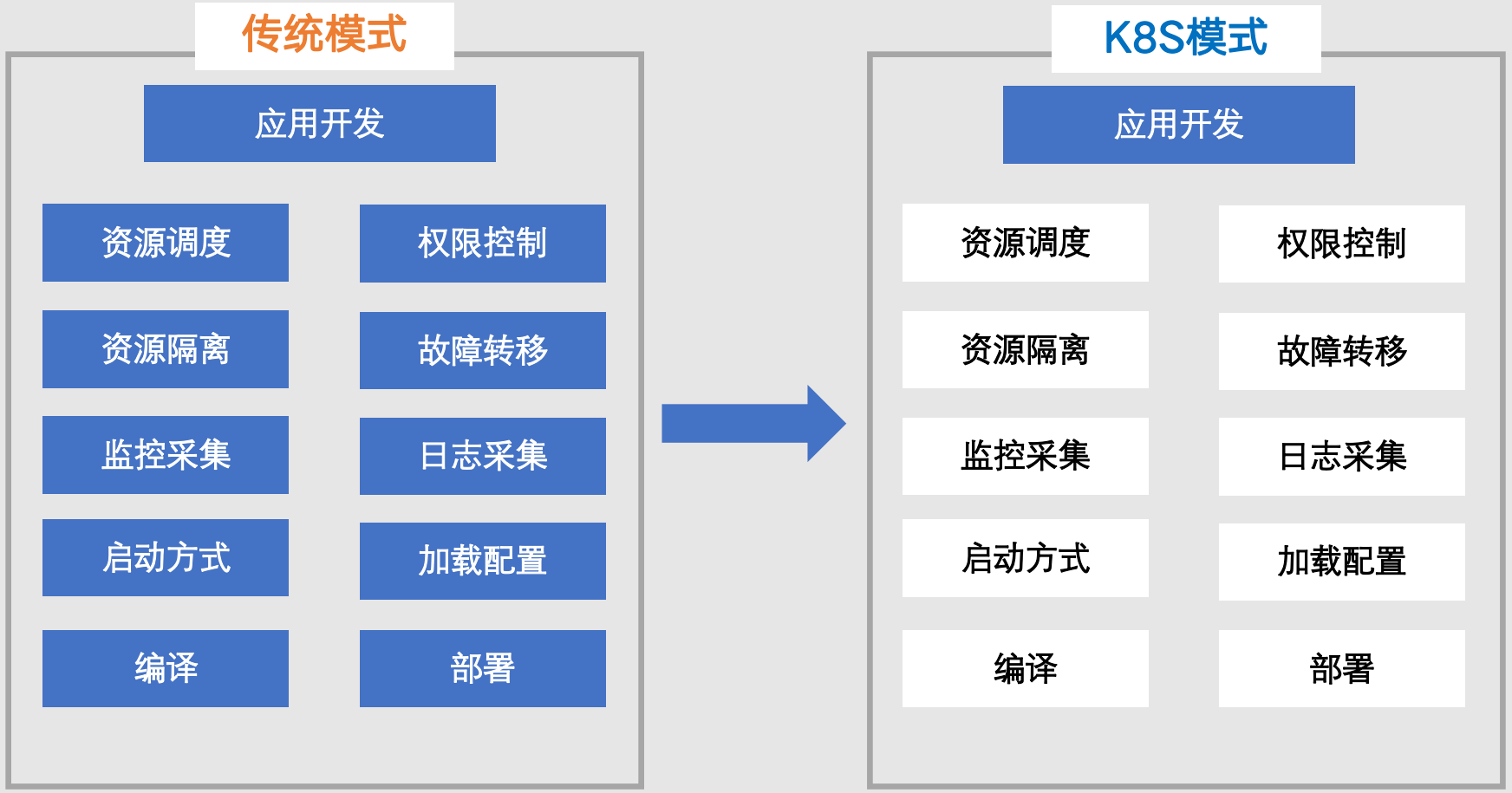
- 组件增多
- 架构复杂
- 维护困难



架构演进

2014年6月K8S开源

- 标准统一
- 运维简单
- 框架简单



第二部分

微服务的生命周期

微服务的生命周期

微服务的开发阶段

配置

- **配置驱动**
- 配置补齐
- 配置工具

对接

- Proto的管理
- 错误码管理
- 调试gRPC

Debug

- 调试信息
- 错误定位

问题：每种开源组件的配置、调用方式、debug方式、记录日志方式都不一样

统一配置、调用用方式，降低开发心智负担

```
[redis]
addr="127.0.0.1:6379"
[mysql]
addr="root:root@tcp(127.0.0.1:3306)/ego?charset=utf8mb4&parseTime=True&loc=Local&readTimeout=1s&timeout=1s&writeTimeout=3s"
[grpc.user]
addr = "k8s:///user-svc:9001"
```

```
ctx := context.Background()
redisClient := eredis.Load( key: "redis").Build()
redisClient.Set(ctx, key: "hello", value: "ego", expire: 0)

grpcUserClient := usersrv.NewUserClient(egrpc.Load( key: "grpc.user").Build().ClientConn)
grpcUserClient.UserInfo(ctx,&usersrv.UserInfoRequest{})

mysqlClient := egorm.Load( key: "mysql").Build()
mysqlClient.WithContext(ctx).Where( query: "id = 1").Find(&user)
```

微服务的开发阶段

配置

- 配置驱动
- **配置补齐**
- 配置工具

对接

- Proto的管理
- 错误码管理
- 调试gRPC

Debug

- 调试信息
- 错误定位

问题：gRPC未设置连接错误，阻塞模式报错不正确
Redis、MySQL连接数配置未设置？超时未设置？

```
// FailOnNonTempDialError only affects the initial dial, and does not do
// anything useful unless you are also using WithBlock().
```

默认补齐配置，给出最佳实践

```
func DefaultConfig() *config {
    return &config{
        Mode:                  StubMode,
        DB:                    0,
        PoolSize:              10,
        MaxRetries:            0,
        MinIdleConns:          20,
        DialTimeout:           xtime.Duration( str: "1s"),
        ReadTimeout:           xtime.Duration( str: "1s"),
        WriteTimeout:          xtime.Duration( str: "1s"),
        IdleTimeout:           xtime.Duration( str: "60s"),
        ReadOnly:              false,
        Debug:                 false,
        EnableMetricInterceptor: true,
        EnableTraceInterceptor:  true,
        SlowLogThreshold:      xtime.Duration( str: "250ms"),
        OnFail:                "panic",
    }
}
```

微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- **配置工具**

对接

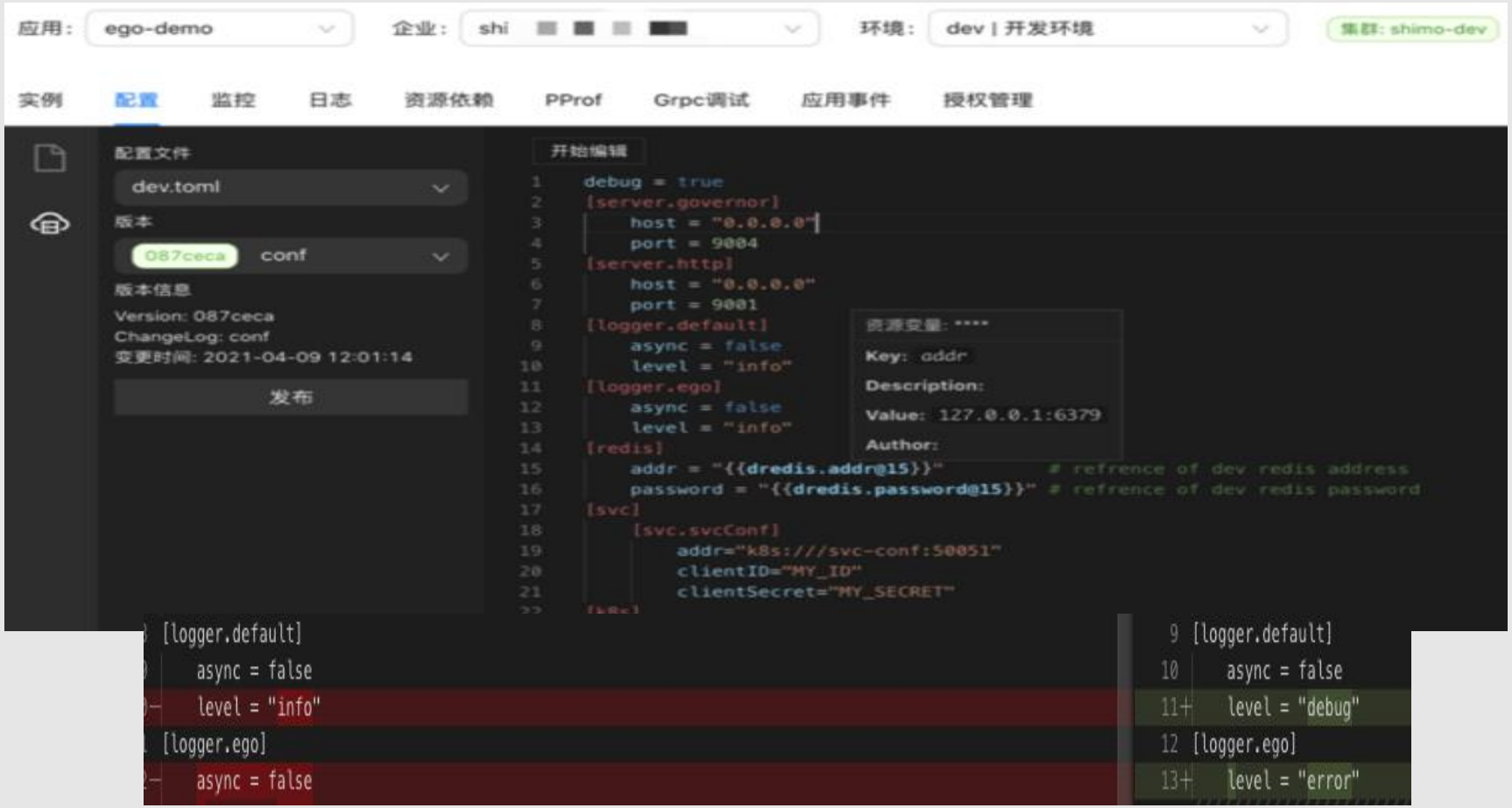
- Proto的管理
- 错误码管理
- 调试gRPC

Debug

- 调试信息
- 错误定位

IDE的体验，右键插入资源引用，悬停查看资源信息

配置版本，发布，回滚，可以更加方便



微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- 配置工具

对接

- **Proto的管理**
- 错误码管理
- 调试gRPC

Debug

- 调试信息
- 错误定位

统一采用gRPC协议和protobuf编解码

CI check 阶段

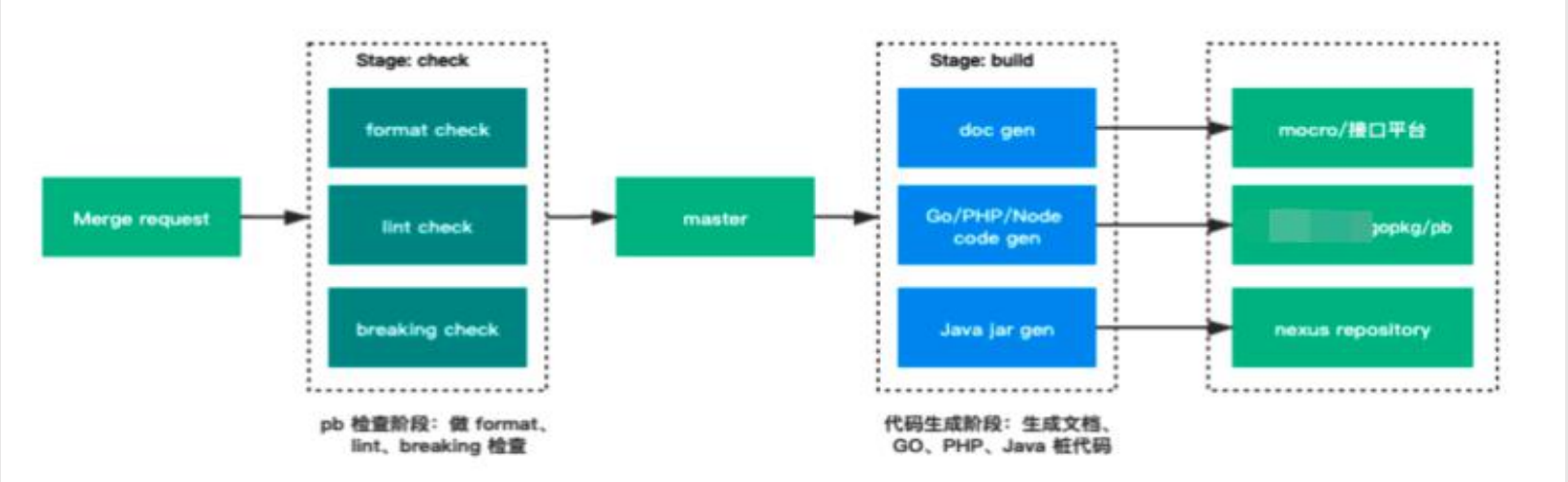
- 主要做 pb 的 format、lint、breaking 检查。

CI build 阶段

- 会基于 pb 的注释自动产生文档，并推送至内部的微服务管理系统接口平台中
- 会生成 Go/PHP/Node/Java 桩代码和错误码，推送到指定的仓库

开发阶段

- go get 客户端、服务端的gRPC和错误码的代码



微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- 配置工具

对接

- Proto的管理
- **错误码管理**
- 调试gRPC

Debug

- 调试信息
- 错误定位

Generate

- protoc -I {error proto file} --go-errors_out={output directory}
  - 实现我们自定义的error类型，方便断言。
  - 根据注解的code信息，在错误码中生成对应的grpc status code
  - 确保错误码唯一，后续在API层响应用户数据确保唯一错误码，例如：下单失败(1008)
  - errors里设置with message，携带更多的错误信息

```
// Error 错误接口
type Error interface {
    error
    WithMetadata(map[string]string) Error
    WithMessage(string) Error
}
```

```
// 错误
enum UserErr {
  // 未知类型
  // @code=UNKNOWN
  USER_ERR_INVALID = 0;
  // 找不到资源
  // @code=NOT_FOUND
  USER_ERR_NOT_FOUND = 1;
  // 密码不匹配
  // @code=INVALID_ARGUMENT
  USER_ERR_PASSWORD_NOT_MATCH = 3;
}
```

微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- 配置工具

对接

- Proto的管理
- **错误码管理**
- 调试gRPC

Debug

- 调试信息
- 错误定位

Check

- gRPC的error可以理解为远程error，他是在另一个服务返回的，所以每次error在客户端是反序列化，new出来的。是无法通过errors.Is判断其根因。
- 将gRPC的错误码注册到一起，然后通过FromError方式，利用map唯一性的判别，转化为本地错误，使用errors.Is来判断根因。
- errors.Is(eerrors.FromError(err)，UserErrNotFound())

```
var resourceErrUnknown *eerrors.EgoError
var resourceErrNotFound *eerrors.EgoError
var resourceErrListMysql *eerrors.EgoError
var resourceErrInfoMysql *eerrors.EgoError

func init() {
    resourceErrUnknown = eerrors.New(int(codes.Unknown), "resource.v1.RESOURCE_ERR_UNKNOWN", Error_RESOURCE_ERR_UNKNOWN.String())
    eerrors.Register(resourceErrUnknown)
    resourceErrNotFound = eerrors.New(int(codes.NotFound), "resource.v1.RESOURCE_ERR_NOT_FOUND", Error_RESOURCE_ERR_NOT_FOUND.String())
    eerrors.Register(resourceErrNotFound)
    resourceErrListMysql = eerrors.New(int(codes.Internal), "resource.v1.RESOURCE_ERR_LIST_MYSQL", Error_RESOURCE_ERR_LIST_MYSQL.String())
    eerrors.Register(resourceErrListMysql)
    resourceErrInfoMysql = eerrors.New(int(codes.Internal), "resource.v1.RESOURCE_ERR_INFO_MYSQL", Error_RESOURCE_ERR_INFO_MYSQL.String())
    eerrors.Register(resourceErrInfoMysql)
}

func ResourceErrUnknown() eerrors.Error {
    return resourceErrUnknown
}
```

```
// FromError try to convert an error to *Error.
// It supports wrapped errors.
func FromError(err error) *EgoError {
    if err == nil {
        return nil
    }
    if se := new(EgoError); errors.As(err, &se) {
        return se
    }
    gs, ok := status.FromError(err)
    if ok {
        for _, detail := range gs.Details() {
            switch d := detail.(type) {
            case *errdetails.ErrorInfo:
                e, ok := errs[errKey(d.Reason)]
                if ok {
                    return e
                }
                return New(
                    int(gs.Code()),
                    d.Reason,
                    gs.Message(),
                ).WithMetadata(d.Metadata).(*EgoError)
            }
        }
    }
    return New(int(codes.Unknown), UnknownReason, err.Error())
}
```

微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- 配置工具

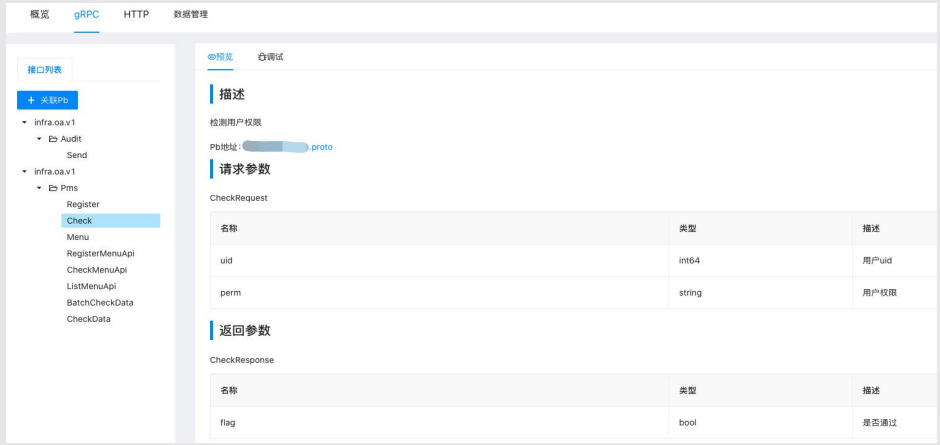
对接

- Proto的管理
- 错误码管理
- **调试gRPC**

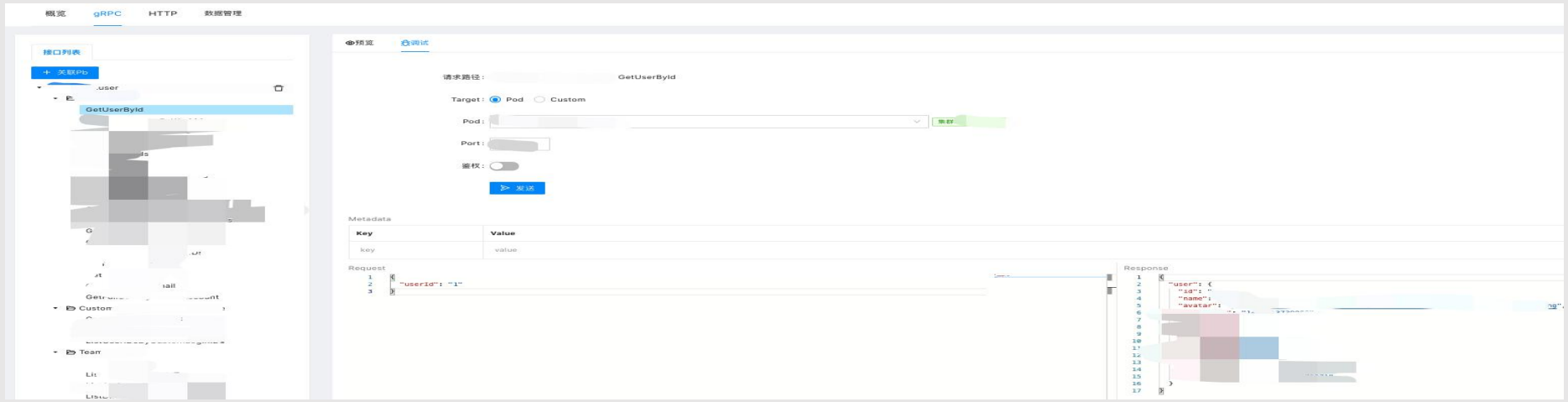
Debug

- 调试信息
- 错误定位

- protobuf lint的注释，利于阅读文档
- 调试gRPC，服务中注入reflection.Register的方法
- 通过K8S API，选择环境、应用、pod，




```
// Register registers the server reflection service on the given gRPC server.
func Register(s GRPCServer) {
    rpb.RegisterServerReflectionServer(s, &serverReflectionServer{
        s: s,
    })
}
```



微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- 配置工具

对接

- Proto的管理
- 错误码管理
- 调试gRPC

Debug

- **调试信息**
- 错误定位

- 展示各种组件gRPC、HTTP、MySQL、Redis、Kafka调试信息
- 六元组（配置名、请求URL、请求参数、响应数据、耗时时间、执行行号）
- 响应数据结构是否正确
- 响应是否有错误

微服务的开发阶段

配置

- 配置驱动
- 配置补齐
- 配置工具

对接

- Proto的管理
- 错误码管理
- 调试gRPC

Debug

- 调试信息
- **错误定位**

- 遵循Fail Fast理念，核心错误尽早panic
- Panic的错误码，组件、配置名、错误信息
- 高亮显示

```
panic:
    msg: parse config error
    loc: /Users/askuy/code/github/gotomicro/ego/examples/grpc/direct/server/main.go:22
    error: server.grpc,err: invalid key, maybe not exist in config
    key: server.grpc
2021-08-26 11:20:33    PANIC    egrpc/container.go:30    parse config error                {"comp": "server.egrpc", "compName": "server.grpc", "error": "server.grpc,err: invalid key, maybe not exist in config", "key": "server.grpc"}
```

微服务的测试阶段

测试类型

工具生成测试用例

简单高效做单元测试

- 单元测试
  - 本地docker—compse
  - 提交代码，触发gitlab ci
- 接口测试
  - 接口平台
- 性能测试
  - benchmark
  - 全链路压测
- 集成测试
  - 以前gitlab ci，docker in docker
  - 目前结合配置中心拓扑图，自动生成jekins编排，ing

微服务的测试阶段

测试类型

工具生成测试用例

简单高效做单元测试

- 业务代码中不要有框架、组件代码，减少单侧用例
- 业务代码做好接口级别单测，简单，快速
- protobuf工具的插件，拿到gRPC服务的描述信息，生成单元测试用例
- 业务人员只需要填写红框内容的断言内容，就可以完成单元测试

```
protoc --proto_path=${ROOT}/examples/helloworld --go-test_out=pkg=main,paths=source_relative:. helloworld.proto
```

```go
// gRPC测试listener
if c.config.Network == "bufnet" {
    listener = bufconn.Listen( sz: 1024 * 1024)
    c.listener = listener
    return nil
}
```

```go
func bufDialer(context.Context, string) (net.Conn, error) {
    return svc.Listener().(*bufconn.Listener).Dial()
}

func TestList(t *testing.T) {
    resourceClient := cegrpc.DefaultContainer().Build(cegrpc.WithDialOption(grpc.WithContextDialer(bufDialer)))
    ctx := context.Background()
    client := resourcev1.NewResourceClient(resourceClient.ClientConn)
    resp, err := client.List(ctx, &resourcev1.ListRequest{})
    want := &resourcev1.ListResponse{
        List: []*resourcev1.Info{
            {
                Id:       1,
                Title:    "测试文章",
                Nickname: "ego",
            },
        },
    }
    assert.NoError(t, err)
    assert.True(t, proto.Equal(want,  resp))
    log.Printf( format: "Response: %+v", resp)
}
```

微服务的测试阶段

测试类型

工具生成测试用例

简单高效做单元测试

单元测试大部分的玩法，都是在做解除依赖

- 面向接口编程
- mock
- 依赖注入

解除依赖很好，但成本很高

基础设施将所有依赖构建起来，就不要让研发用代码去实现

- gitlab.yaml or docker-compose.yaml
- 构建MySQL、Redis
- 创建表 ./app --job=install
- 初始化数据 ./app --job=initialize
- 单元测试 go test ./...

```
listd_tests_1  |
mysql_1        | 2021-09-09T08:51:33.439680Z 3 [Note] Aborted connection 3 to db: 'go-engineering' user: 'root' host: '172.21.0.3' (Got an error reading communication packets)
listd_tests_1  | go: downloading github.com/stretchr/testify v1.7.0
listd_tests_1  | go: downloading github.com/pmezard/go-difflib v1.0.0
listd_tests_1  | ?      go-engineering/resource-svc     [no test files]
listd_tests_1  | ?      go-engineering/resource-svc/pkg/invoker [no test files]
listd_tests_1  | ?      go-engineering/resource-svc/pkg/job     [no test files]
listd_tests_1  | ?      go-engineering/resource-svc/pkg/model/mysql     [no test files]
listd_tests_1  | ok     go-engineering/resource-svc/pkg/router  0.036s
mysql_1        | 2021-09-09T08:51:59.176968Z 4 [Note] Aborted connection 4 to db: 'go-engineering' user: 'root' host: '172.21.0.3' (Got an error reading communication packets)
go-engineering_listd_tests_1 exited with code 0
Aborting on container exit...
Stopping go-engineering_mysql_1      ... done
```

```yaml
services:
  listd_tests:
    build:
      context: .
      dockerfile: Dockerfile.resource-unittest
    depends_on:
      - mysql
    networks:
      - integration-tests-example-test
    links:
      - mysql
  mysql:
    image: mysql:5.7.24
    command: [
        '--character-set-server=utf8mb4',
        '--collation-server=utf8mb4_general_ci',
    ]
    environment:
      MYSQL_USER: root
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: go-engineering
    restart: on-failure
    networks:
      - integration-tests-example-test
```

微服务的部署阶段

注入信息

版本信息

发布版本

- 注入应用名称、应用版本号、编译所在机器、编译时间配置
- 启动应用，获取debug.ReadBuildInfo，注入框架版本号

```
go build -o bin/hello -ldflags -X "github.com/gotomicro/ego/core/eapp.appName=hello
-X
github.com/gotomicro/ego/core/eapp.buildVersion=cbf03b73304d7349d3d681d3abd42a90b8b
a72b0-dirty -X
github.com/gotomicro/ego/core/eapp.buildAppVersion=cbf03b73304d7349d3d681d3abd42a90
b8ba72b0-dirty -X github.com/gotomicro/ego/core/eapp.buildStatus=Modified -X
github.com/gotomicro/ego/core/eapp.buildTag=v0.6.3-2-gcbf03b7 -X
github.com/gotomicro/ego/core/eapp.buildUser=`whoami` -X
github.com/gotomicro/ego/core/eapp.buildHost=`hostname -f` -X
github.com/gotomicro/ego/core/eapp.buildTime=`date +%Y-%m-%d--%T`"
```

https://ego.gocn.vip/micro/chapter1/build.html

```go
egoVersion := "unknown version"
info, ok := debug.ReadBuildInfo()
if ok {
    for _, value := range info.Deps {
        if value.Path == "github.com/gotomicro/ego" {
            egoVersion = value.Version
        }
    }
}
```

微服务的部署阶段

注入信息

版本信息

发布版本

- 执行./bin/hello ——version
- 查看线上使用框架版本

```
./bin/hello --version
EGO           : I am EGO
AppName       : hello
AppHost       : askuydeMacBook-Pro.local
Region        : huabei
Zone          : ali-3
AppVersion    : 925d5b27ff35b4490494ba78ceb897e02cb12d92-dirty
EgoVersion    : 0.1.0
BuildUser     : askuy
BuildHost     : askuydeMacBook-Pro.local
BuildTime     : 2020-12-03 17:26:24
BuildStatus   : Modified
```

https://ego.gocn.vip/micro/chapter1/build.html

Pod	启动时间	应用版本	框架版本	enable	Go版本	Pod IP	Namespace	pod_template_hash	version
-7cdf9d5c58-f69p4	2021-04-11 20:21:07	500	v0.4.5	true	go1.15.2		default	7cdf9d5c58	dev

微服务的部署阶段

注入信息

版本信息

发布版本

- 配置
  - 过去自己实现agent读取etcd，写文件
  - 现在写入config map，挂载到pod
- 应用
  - 一行代码kubectl apply -f deployment.yaml
  - 拉取镜像、启动服务、探活、滚动更新等功能

微服务的启动阶段

- 启动参数
- 加载配置
- 探活
- 滚动更新

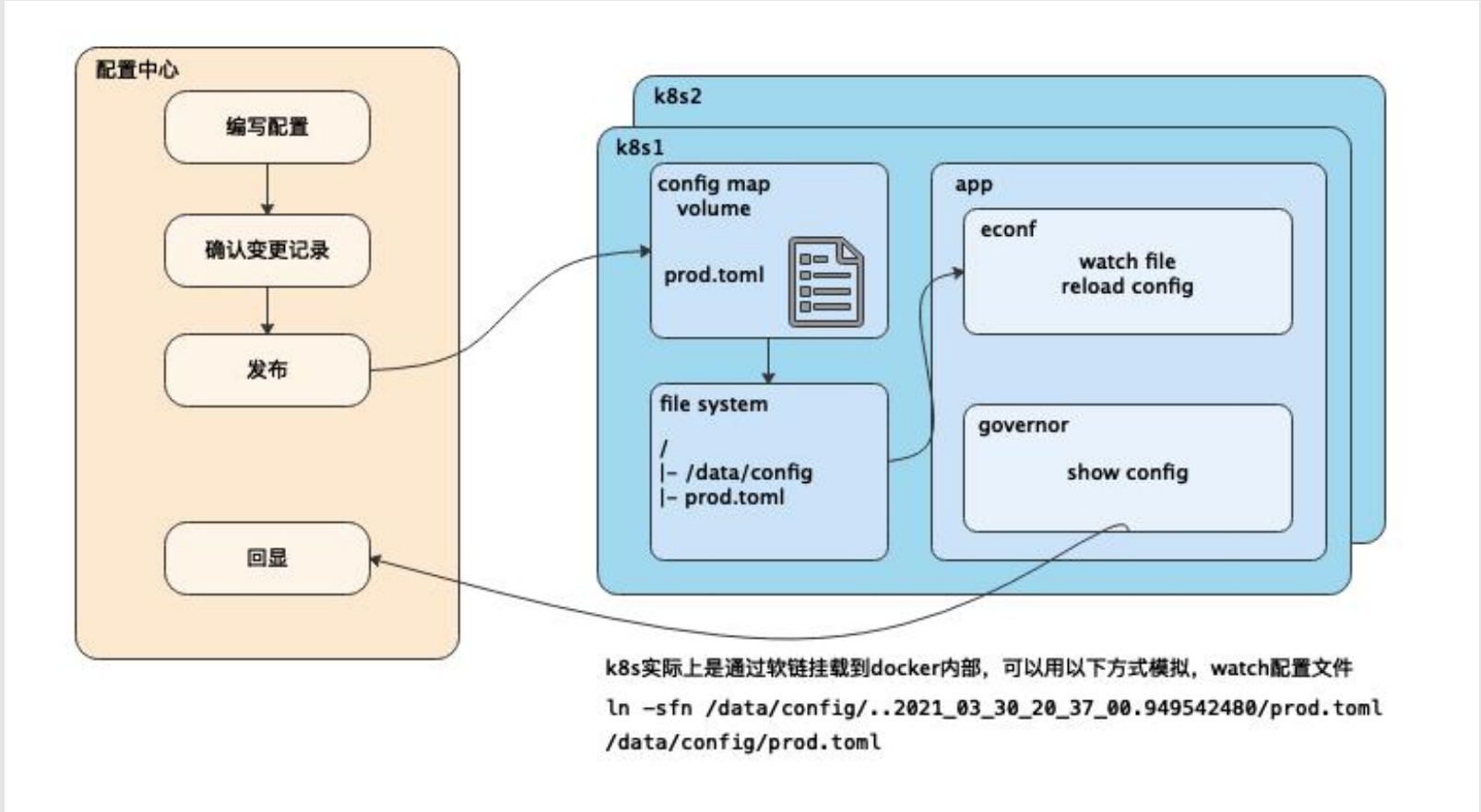
`EGO` 内置很多环境变量，这样可以很方便的通过基础设施将公司内部规范的一些数据预设在 `K8S` 环境变量内，业务方就可以简化很多启动参数，在 `dockerfile` 里启动项变为非常简单的命令行：`CMD ["sh", "-c", "./${APP}"]`

命令行参数	环境变量	默认参数	描述
config	EGO_CONFIG_PATH	config/local.toml	配置路径
host	EGO_HOST	0.0.0.0	启动IP
watch	EGO_WATCH	true	默认监听
debug	EGO_DEBUG	false	是否开启调试
ego_name	EGO_NAME	filepath.Base(os.Args[0])	应用名
ego_log_path	EGO_LOG_PATH	./logs	配置路径
ego_trace_id_name	EGO_TRACE_ID_NAME	x-trace-id	链路名称

微服务的启动阶段

- 启动参数
- 加载配置
- 探活
- 滚动更新

软链接：filepath.EvalSymlinks(fp.path)



微服务的启动阶段

启动参数

加载配置

探活

滚动更新

- livenessProbe：如果检查失败，将杀死容器，根据Pod的restartPolicy来操作
- readinessProbe：　如果检查失败，Kubernetes会把Pod从service endpoints中剔除

- HTTP：server.Use(healthcheck.Default())
- gRPC：healthpb.RegisterHealthServer(newServer，health.NewServer())

```
httpGet:
  path: /
  port: 9002
  scheme: HTTP
  httpHeaders:
  -name:X-Health-Check
   value: 1
```

```
exec:
  command:
  -grpc_health_probe
  --addr=127.0.0.1:9001
```

```
$ curl -iL -XGET -H "X-Health-Check: 1" http://localhost
# HTTP/1.1 200 OK
# Content-Length: 2
# Content-Type: text/plain; charset=utf-8
#
# ok
```

微服务的调用阶段

Resolver

Balancer

Auth

Context

- Kubernetes DNS Resolver VS Kubernetes API Server Resolver
  - DNS resolver is builtin in gRPC framework and its out-of-box for users
  - When connection fail DNS Resolver can resolve name immediately
  - In scale-up scenario, DNS Resolver will not resolve name automatically

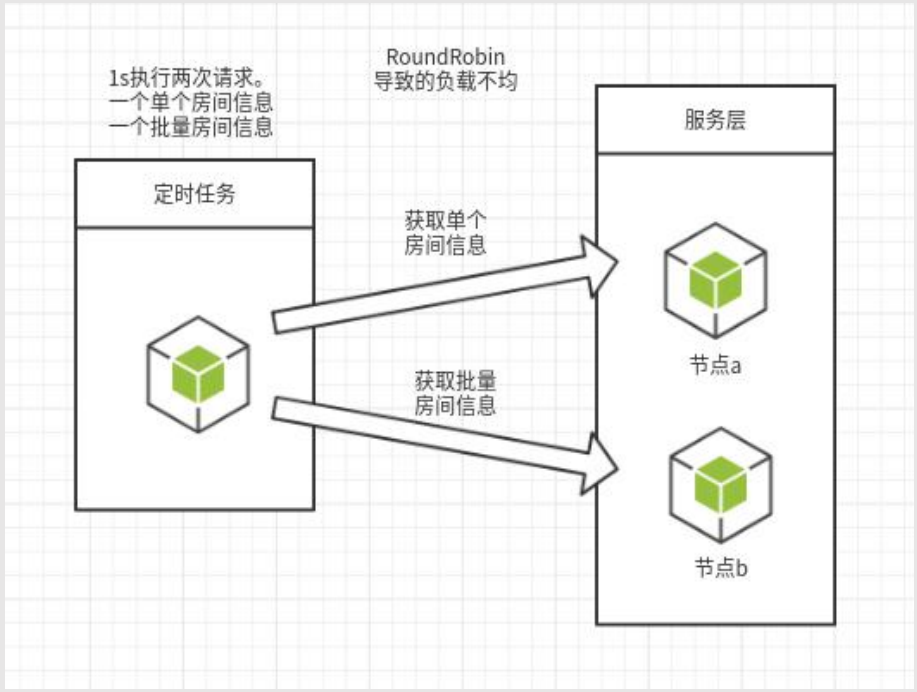
问题：
- 新版本缩容无感知
- 老版本性能问题

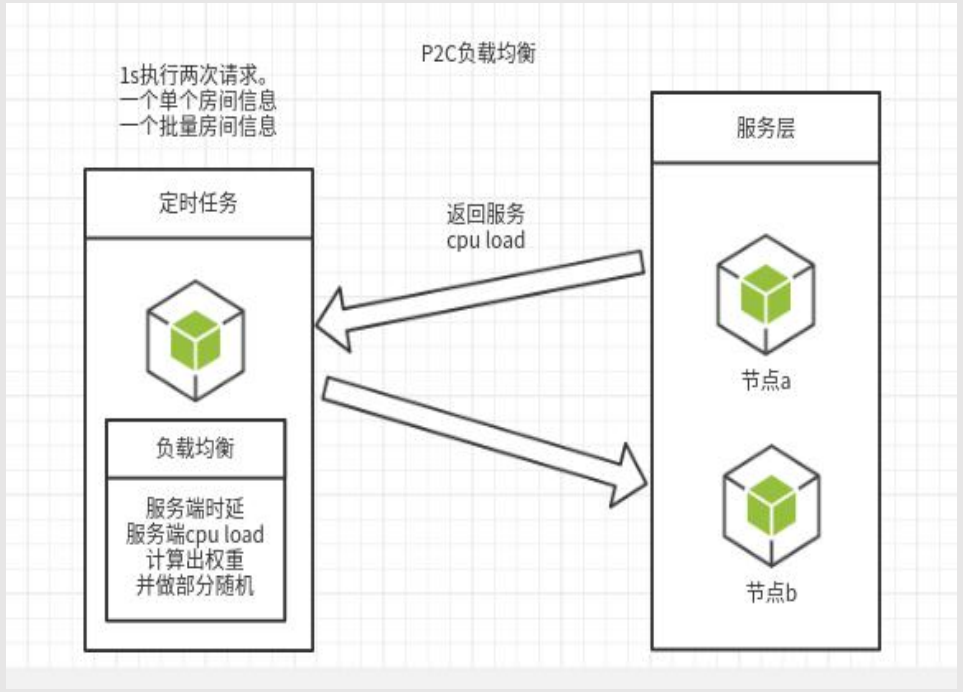
```
…
informer.Informer().AddEventHandler(cache.ResourceE
ventHandlerFuncs{
        AddFunc:    c.addEndpoints,
        UpdateFunc: c.updateEndpoints,
        DeleteFunc: c.deleteEndpoints,
})
…

[grpc.otherService]
    addr = "k8s:///other-service:50051"
```

微服务的调用阶段

- Resolver
- Balancer
- Auth
- Context





微服务的治理阶段

- 监控
- 日志
- 链路
- 限流熔断
- 报警

错误收敛

- 方便报警
- 减少冗余信息

```
func prometheusUnaryServerInterceptor(ctx context.Context, req interface{}, info *grpc.UnaryServerInfo, handler grpc.UnaryHandler) (interface{},
 error) {
    startTime := time.Now()
    resp, err := handler(ctx, req)
    emetric.ServerHandleHistogram.Observe(time.Since(startTime).Seconds(), emetric.TypeGRPCUnary, info.FullMethod, extractApp(ctx))
    emetric.ServerHandleCounter.Inc(emetric.TypeGRPCUnary, info.FullMethod, extractApp(ctx), http.StatusText(ecode.GrpcToHTTPStatusCode(status.Code(
  err))))
    return resp, err
}
```

微服务的治理阶段

监控

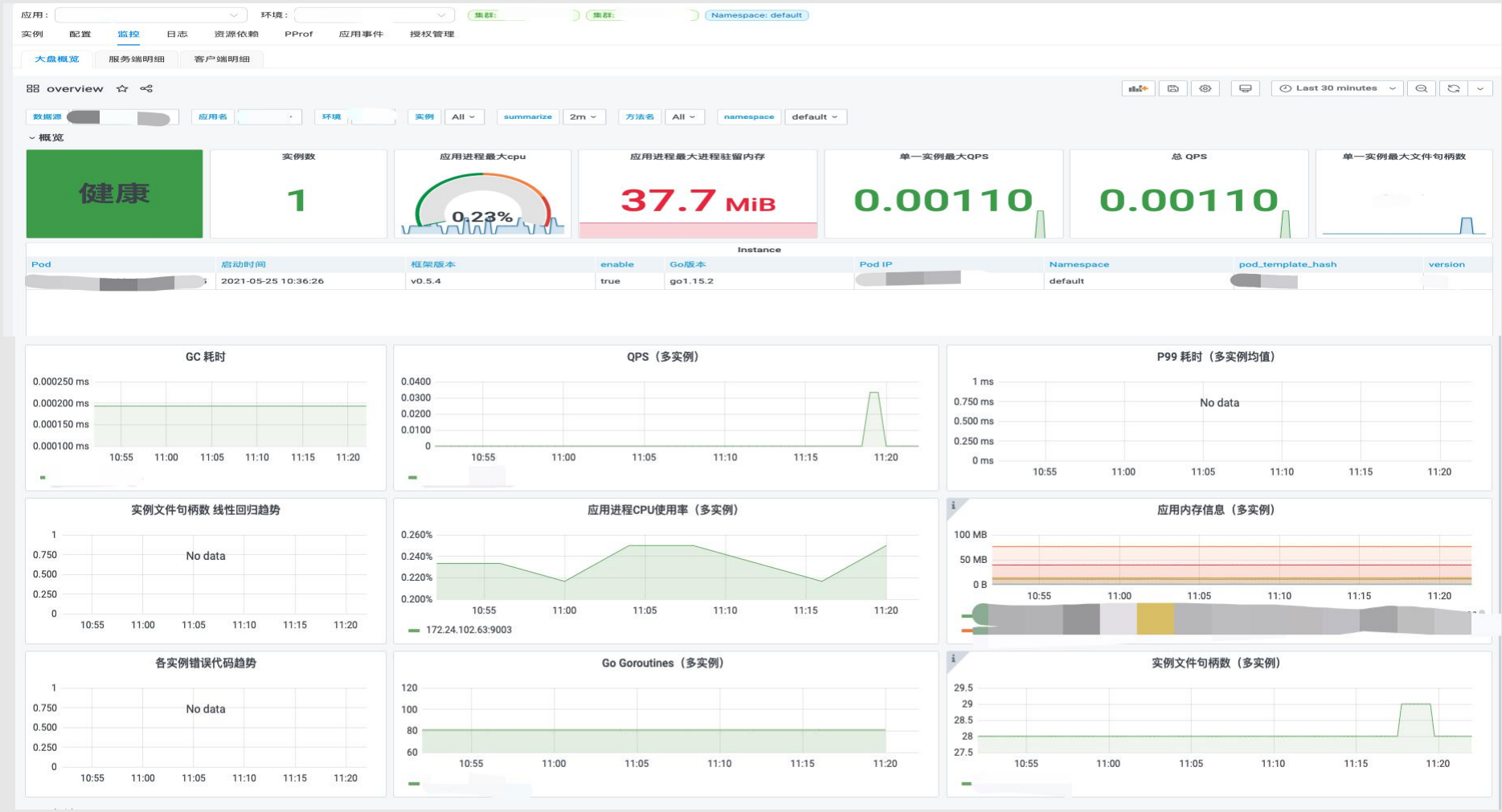
日志

链路

限流熔断

报警

总览、错误、流量、延迟



微服务的治理阶段

监控

日志

链路

限流熔断

报警

https://ego.gocn.vip/frame/core/logger.html

索引收敛

mysql的方法叫sql，redis方法叫cmd，框架这里全部统一用method。降低索引个数

```
interceptor.go  104   elog.FieldMethod(c.Request.Method+"."+c.FullPath()),
interceptor.go  210   elog.FieldMethod(info.FullMethod),
```

```
interceptor.go   70   elog.FieldMethod(op),
interceptor.go  150   elog.FieldMethod(cmd.Name()),
interceptor.go  168   elog.FieldMethod(method),
interceptor.go   40   elog.FieldMethod(fullMethod),
```

日志分类

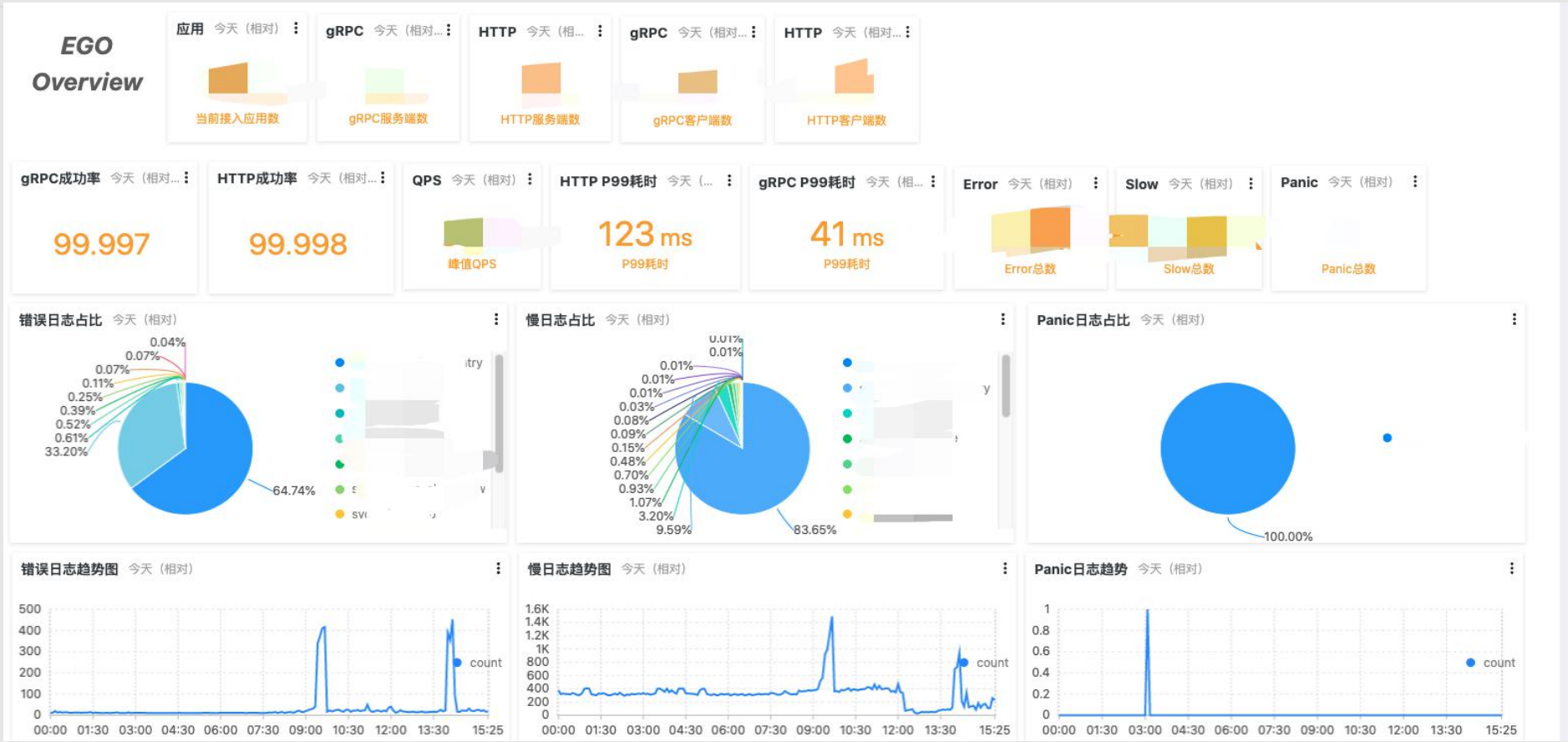
- 框架日志
- 业务日志

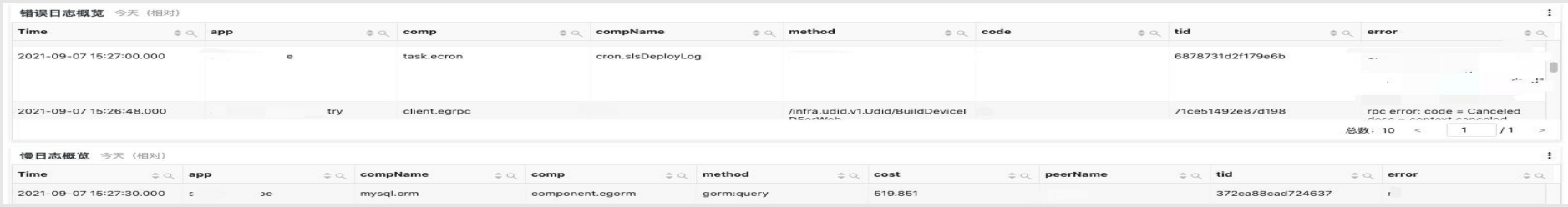
- 慢日志
- Error日志
- Panic日志

微服务的治理阶段

- 监控
- 日志
- 链路
- 限流熔断
- 报警

https://ego.gocn.vip/frame/core/logger.html





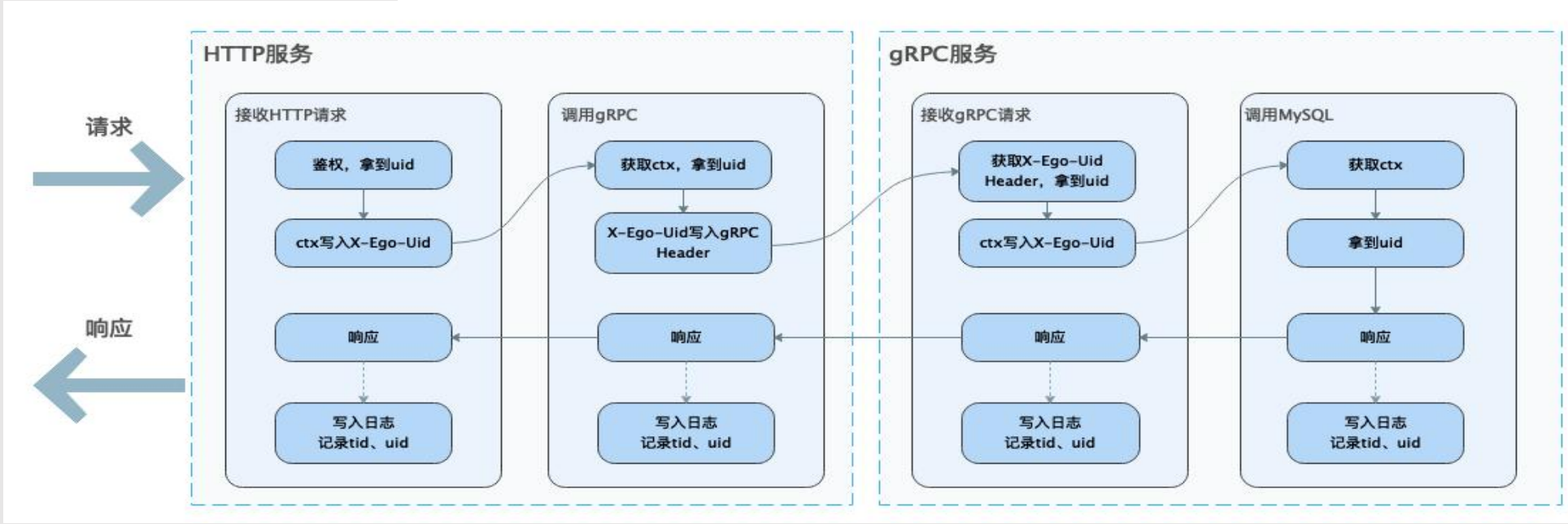
微服务的治理阶段

- 监控
- 日志
- 链路
- 限流熔断
- 报警

https://ego.gocn.vip/micro/chapter2/trace.html

export EGO_LOG_EXTRA_KEYS=X-Ego-Uid,X-Ego-OrderID

方式	采集类型	量级	优势	调整采集	格式
日志	请求、错误、慢日志	中	聚合、错误收敛	调整日志级别	动态
链路	请求日志（可以细粒度到函数级别）	大	traceId，调用关系	调整采样率	固定




```
2021-07-28 22:54:12  WARN  egorm/interceptor.go:101  access  {"comp": "component.egorm", "compName": "mysql.test", "addr": "127.0.0.1:3306", "method": "gorm:query", "name": "test.user",
"cost": 0.493, "req": "SELECT * FROM `user` WHERE id = 100 ORDER BY `user`.`id` LIMIT 1", "res": {"id":0,"name":""}, "tid": "760cfd586623e6a8", "x-ego-uid": "9527", "event": "error", "error": "record not found"}
```

微服务的治理阶段

监控

日志

链路

限流熔断

报警

- 修改sentinel.json
- 框架Watch sentinel.json
- 限流实时生效
- Sentinel监控

```
2021-08-22 16:36:03     INFO    esentinel/component.go:76       read sentinel watch event               {"comp": "core.sentinel", "compName": "sentinel", "event": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config
/sentinel.json", "path": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json", "currentConfigFile": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json", "realConfigFil
e": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json"}
2021-08-22 16:36:03     INFO    esentinel/component.go:76       read sentinel watch event               {"comp": "core.sentinel", "compName": "sentinel", "event": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config
/sentinel.json", "path": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json", "currentConfigFile": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json", "realConfigFil
e": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json"}
2021-08-22 16:36:03     INFO    esentinel/component.go:90       modified sentinel file                  {"comp": "core.sentinel", "compName": "sentinel", "name": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/
sentinel.json", "addr": "/Users/askuy/code/github/gotomicro/ego/examples/sentinel/http/config/sentinel.json"}
```

```
2021-08-22 16:27:14     INFO    egin/interceptor.go:175 access                                  {"comp": "server.egin", "compName": "server.http", "type": "http", "cost": 0.217, "method":
.0.1", "size": 11, "peerIp": "127.0.0.1", "tid": "2ef77f740de7e51f", "req": {"metadata":{"Accept":["*/*"],"User-Agent":["ApacheBench/2.3"]},"payload":""}, "res": {"metadata":{"Content-Typ
ce-Id":["2ef77f740de7e51f"]},"payload":"\"Hello EGO\""}, "event": "normal", "error": "", "code": 200}
2021-08-22 16:27:14     INFO    egin/interceptor.go:175 access                                  {"comp": "server.egin", "compName": "server.http", "type": "http", "cost": 0.082, "method":
.0.1", "size": 11, "peerIp": "127.0.0.1", "tid": "5ca053153476e18b", "req": {"metadata":{"Accept":["*/*"],"User-Agent":["ApacheBench/2.3"]},"payload":""}, "res": {"metadata":{"Content-Typ
ce-Id":["5ca053153476e18b"]},"payload":"\"Hello EGO\""}, "event": "normal", "error": "", "code": 200}
2021-08-22 16:27:14     INFO    egin/interceptor.go:175 access                                  {"comp": "server.egin", "compName": "server.http", "type": "http", "cost": 0.087, "method":
.0.1", "size": 0, "peerIp": "127.0.0.1", "tid": "454e7ebf605ef78", "req": {"metadata":{"Accept":["*/*"],"User-Agent":["ApacheBench/2.3"]},"payload":""}, "res": {"metadata":{"X-Trace-Id":[
normal", "error": "", "code": 429}
2021-08-22 16:27:14     INFO    egin/interceptor.go:175 access                                  {"comp": "server.egin", "compName": "server.http", "type": "http", "cost": 0.08, "method":
0.1", "size": 0, "peerIp": "127.0.0.1", "tid": "1e3928d54057b394", "req": {"metadata":{"Accept":["*/*"],"User-Agent":["ApacheBench/2.3"]},"payload":""}, "res": {"metadata":{"X-Trace-Id":[
"normal", "error": "", "code": 429}
2021-08-22 16:27:14     INFO    egin/interceptor.go:175 access                                  {"comp": "server.egin", "compName": "server.http", "type": "http", "cost": 0.091, "method":
.0.1", "size": 0, "peerIp": "127.0.0.1", "tid": "425f2c8e6c93633e", "req": {"metadata":{"Accept":["*/*"],"User-Agent":["ApacheBench/2.3"]},"payload":""}, "res": {"metadata":{"X-Trace-Id":
 "normal", "error": "", "code": 429}
```

https://ego.gocn.vip/frame/client/sentinel.html

```
# HELP sentinel_process_cpu_ratio current process cpu utilization ratio
# TYPE sentinel_process_cpu_ratio gauge
sentinel_process_cpu_ratio{cpu="pid:20",host="host:askuydeMacBook-Pro.local",process="main",process_cpu_ratio="process_cpu_ratio"} 12.022912858546524
# HELP sentinel_process_memory_size current process memory size in bytes
# TYPE sentinel_process_memory_size gauge
sentinel_process_memory_size{host="host:askuydeMacBook-Pro.local",pid="pid:20",process="main",total_memory_size="total_memory_size"} 1.8976768e+07
# HELP sentinel_resource_flow_threshold resource flow threshold
# TYPE sentinel_resource_flow_threshold gauge
sentinel_resource_flow_threshold{host="host:askuydeMacBook-Pro.local",resource="rs:GET./hello",threshold="threshold"} 2
```

微服务的治理阶段

监控

日志

链路

限流熔断

报警

- K8S自带消息总线
- 入口处最重要
- K8S Ingress统一了所有入口，一条语句，做所有SLA报警

第三部分

如何管理好微服务

如何管理好微服务

责任

管理版本信息

管理拓扑关系

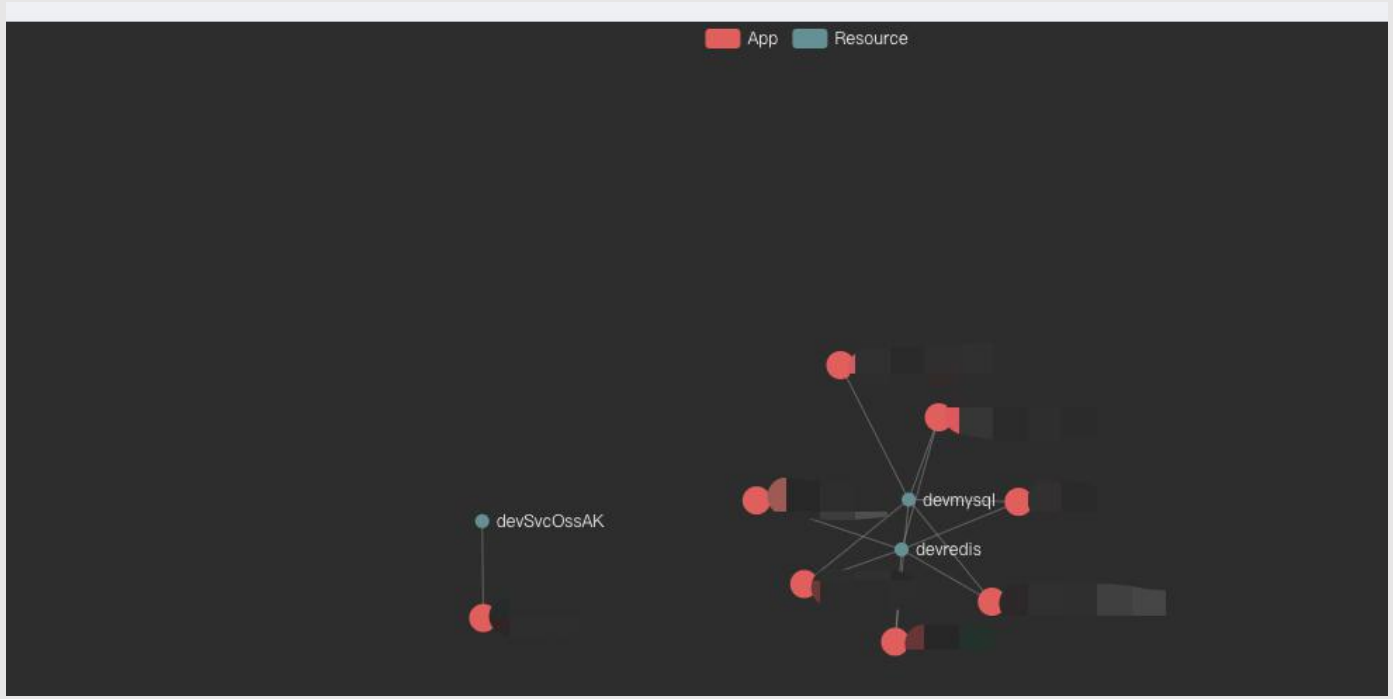
管理成本

- 应用部门、应用负责人

- 升级版本

如何管理好微服务

- 责任
- 管理版本信息
- 管理拓扑关系
- 管理成本



如何管理好微服务

- 责任
- 管理版本信息
- 管理拓扑关系
- 管理成本

资料

框架：https://github.com/gotomicro/ego
编译：https://ego.gocn.vip/micro/chapter1/build.html
链路：https://ego.gocn.vip/micro/chapter2/trace.html
限流：https://ego.gocn.vip/frame/client/sentinel.html
日志：https://ego.gocn.vip/frame/core/logger.html
docker-compose单元测试，protobuf统一错误码：https://github.com/gotomicro/go-engineering
docker测试玩法：https://www.ardanlabs.com/blog/2019/03/integration-testing-in-go-executing-tests-with-docker.html

提问