

Go 微服务框架工程化设计

陈志辉

哔哩哔哩 资深开发工程师

自我介绍

中间件开发者沙龙

陈志辉，哔哩哔哩，基础架构团队

主要擅长微服务高可用架构，服务治理，Go语言

2016年加入b站，经历过完整的微服务转型，早期深度参与业务研发工作

同时也参与了内部很多微服务中间件的研发设计

开源比较影响力的项目：

- 微服务框架：<https://github.com/go-kratos/kratos>
- API网关：<https://github.com/go-kratos/gateway>
- 长连接网关：<https://github.com/Terry-Mao/goim>



目录

- 面向包的设计理念
- 配置规范思考
- IDL & API Definition
- API Error Reason
- Transport HTTP/gRPC
- Middleware 插件化使用
- Kratos 启动管理器

What is Go ?

中间件开发者沙龙

Go is :

- open source
- concurrent
- garbage-collected
- efficient
- scalable
- simple
- fun
- ...

<https://golang.org>

```
for {  
    // do something  
}  
for n < 5 {  
    // do something  
}  
for i := 0; i < 5; i++ {  
    // do something  
}  
for i, s := range strings {  
    // do something  
}
```

loop、while、do..while、for、range

Go 是一个面向包名设计的语言。

Package 在 Go 程序中主要起到功能隔离的作用：

- 程序的各个部分可以分开
- 在大型团队中处理复杂的项目
- 包名可改善团队成员之间的沟通

Go 对于包支持其实很棒的，标准库就是很好的设计。

但是，我们在开发过程中，如果没有采用很好的代码组织方式，可能会让项目非常地难以理解。

在 Go 标准库，也提供很多常用的 Packages：

- `fmt`
- `strings`
- `bytes`
- `io`
- `errors`
- `encoding`
- `sync`
- `time`
- `net/http`
- `net/rpc`
- `database/sql`

为了有目的，包必须提供，而不是包含。

包的命名必须旨在描述它提供的内容：

- 包的目的是为特定问题域而提供的。
- 每个包的目的越集中，就越清楚知道包提供了什么。
- 包的名称必须描述它提供的内容，如果包的名称不能立即暗示这一点，则它可能包含一组零散的功能。
- 这些零散的功能，可能包含一些 `util`、`common`、`helpers` 相关。

包不能成为不同问题域的聚合地，随着时间的推移，它将影响项目的简洁和重构、适应、扩展和分离的能力。

在 Kratos 框架中，我们主要参考了标准库的包设计理念。

包名按功能进行划分，每个包具有唯一的职责。

当 用户不可见 或者 不稳定 的接口需要放到 /internal 目录中。

然而，我们也主要分为这几层的设计思想：

- API Layer
- Transport Layer
- Middleware Layer
- Service Layer
- Data Layer

Kratos 每个包功能特性：

- `/cmd`，可以通过 `go install` 一键安装生成工具使用户更加方便地使用框架。
- `/errors`，统一的业务错误封装，方便返回错误码和业务原因。
- `/config`，支持多数据源方式，进行配置合并铺平，支持配置热更。
- `/transport`，传输层（HTTP/gRPC）的抽象封装，可以方便获取对应的接口信息。
- `/middleware`，中间件抽象接口，主要跟 `transport` 和 `service` 之间的桥梁适配器。
- `/metadata`，跨服务间的元信息传递和使用。
- `/registry`，注册中心适配接口，可以实现各种服务发现，例如：`etcd/consul/zookeeper`。

github.com/go-kratos/kratos

```
|— cmd  
|— docs  
|— internal  
|— examples  
|— api  
|— errors  
|— config  
|— encoding  
|— log  
|— metrics  
|— metadata  
|— middleware  
|— transport  
|— registry  
|— third_party  
|— app.go  
|— options.go  
|— go.mod  
|— go.sum
```

Kratos 主要的框架工具：

- `cmd/kratos`
 - `kratos new project_name` , 创建项目模板 , 默认通过 `kratos-layout` 仓库下载 , 也可以通过 `-r` 或者 环境变量 `KRATOS_LAYOUT_REPO` 指定自定义模板仓库。
 - `kratos proto add` , 添加一个 proto 模板文件 (CURD) 。
 - `kratos proto client` , 生成 proto 源码文件。
 - `kratos proto server` , 生成 proto service 实现代码文件。
 - `kratos upgrade` , 更新 kratos 工具到最新版本。
- `cmd/protoc-gen-go-errors` , 为 errors 生成对应 `IsXxx`、`ErrorXxx` 辅助代码。
- `cmd/protoc-gen-go-http` , 为 HTTP 生成对应的接口定义 , 根据 `google.api.http` 规范实现。

```
go install github.com/go-kratos/kratos/cmd/kratos/v2@latestkratos new helloworld
```


在 Application 项目结构中，其实是包含一起部署的程序集。

程序集可以包括 server、cli工具 和 task 等应用，通常会放到 /cmd 目录中。

如果在一个单体大型项目中，可以按包名类似划分为：

- /api
- /cmd
 - xxxcli
 - xxxd
- /internal
 - order
 - payment
 - platform
 - mysql
 - redis

如果在微服务中，通常我们也可以通过统一仓库（mono-repo）进行管理项目。
把不同的业务域划分出对应的微服务，再通过 HTTP/gRPC 进行进程之间的通信。
微服务结构划分，跟单体项目有所不同，可以把一个个项目放到 app 中：

- api
- app
 - user
 - order
 - payment
- pkg
 - backoff
 - pagination

在 Kratos 项目中，我们主要是以微服务类型为标准的项目布局。

通过 Kratos 工具生成的项目结构为：

- api
- cmd
 - myapp
- configs
 - application.yaml
- go.mod
- go.sum
- internal
 - service // implements protobuf
 - biz
 - data

在 Kratos 项目中，配置源可以指定多个，并且 config 会进行合并成 key/value。然后用户通过 Scan 或者 Value 获取对应键值内容；

主要功能特性：

- 默认实现了本地文件数据源。
- 用户可以自定义数据源实现。
- 支持配置热更新，通过 Atomic 方式变更已有 Value。
- 支持自定义数据源解码实现。

配置扩展：

- 增加对 flags、环境变量 占位符替换。
- 通过铺平的 key/value，进行二次赋值替换。
- 例如，a.b.c = {{ xx.xxx }}

```
// KeyValue is config key value.
type KeyValue struct {
    Key    string
    Value  []byte
    Format string
}

// Source is config source.
type Source interface {
    Load() ([]*KeyValue, error)
    Watch() (Watcher, error)
}

// Watcher watches a source for changes.
type Watcher interface {
    Next() ([]*KeyValue, error)
    Stop() error
}
```


在 Kratos 项目中，我们默认通过 proto 进行定义配置文件。

主要的以下几点好处：

- 可以定义统一的模板配置
- 添加对应的配置校验
- 更好地管理配置
- 跨语言支持

```
server:
  http:
    addr: 0.0.0.0:8000
    timeout: 1s
  grpc:
    addr: 0.0.0.0:9000
    timeout: 1s
data:
  database:
    driver: mysql
    source: root:root@tcp(127.0.0.1:3306)/test
  redis:
    addr: 127.0.0.1:6379
    read_timeout: 0.2s
    write_timeout: 0.2s
```

```
message Bootstrap {
  Server server = 1;
  Data data = 2;
}

message Server {
  message HTTP {
    string network = 1;
    string addr = 2;
    google.protobuf.Duration timeout = 3;
  }
  message GRPC {
    string network = 1;
    string addr = 2;
    google.protobuf.Duration timeout = 3;
  }
  HTTP http = 1;
  GRPC grpc = 2;
}

message Data {
  message Database {
    string driver = 1;
    string source = 2;
  }
  message Redis {
    string network = 1;
    string addr = 2;
    google.protobuf.Duration read_timeout = 3;
    google.protobuf.Duration write_timeout = 4;
  }
  Database database = 1;
  Redis redis = 2;
}
```

IDL & API Definition

中间件开发者沙龙

为了更好地管理 API 接口代码，我们可以在 Proto 中进行统一定义 API 接口，可以通过 HTTPRule 定义 HTTP 接口。

将会有以下几个好处：

- 统一 元信息 和 研发流程
- 统一 生成 client/server 代码；
- IDL 即定义，IDL 即代码，IDL 即文档；
- 生成 openapi.yaml，同时可以导入到一个测试工具

```
service Messaging {  
  rpc UpdateMessage(UpdateMessageRequest) returns (Message) {  
    option (google.api.http) = {  
      patch: "/v1/messages/{message_id}"  
      body: "message" // "*" mapped to the request  
    };  
  }  
}  
  
message UpdateMessageRequest {  
  string message_id = 1; // mapped to the URL  
  Message message = 2; // mapped to the body  
}
```

MiddlewareService 插件服务

GET /v1/admin/gateway/middlewares

POST /v1/admin/gateway/middlewares

PATCH /v1/admin/gateway/middlewares

GET /v1/admin/gateway/middlewares/{id}

IDL & Project Layout

中间件开发者沙龙

在统一仓库中管理 Proto , 以仓库为包名根目录:

目录结构和 package 对齐 ;

- /<业务域>.<业务>.<应用>.<版本>
- /account.passport.login.v1/<resource>

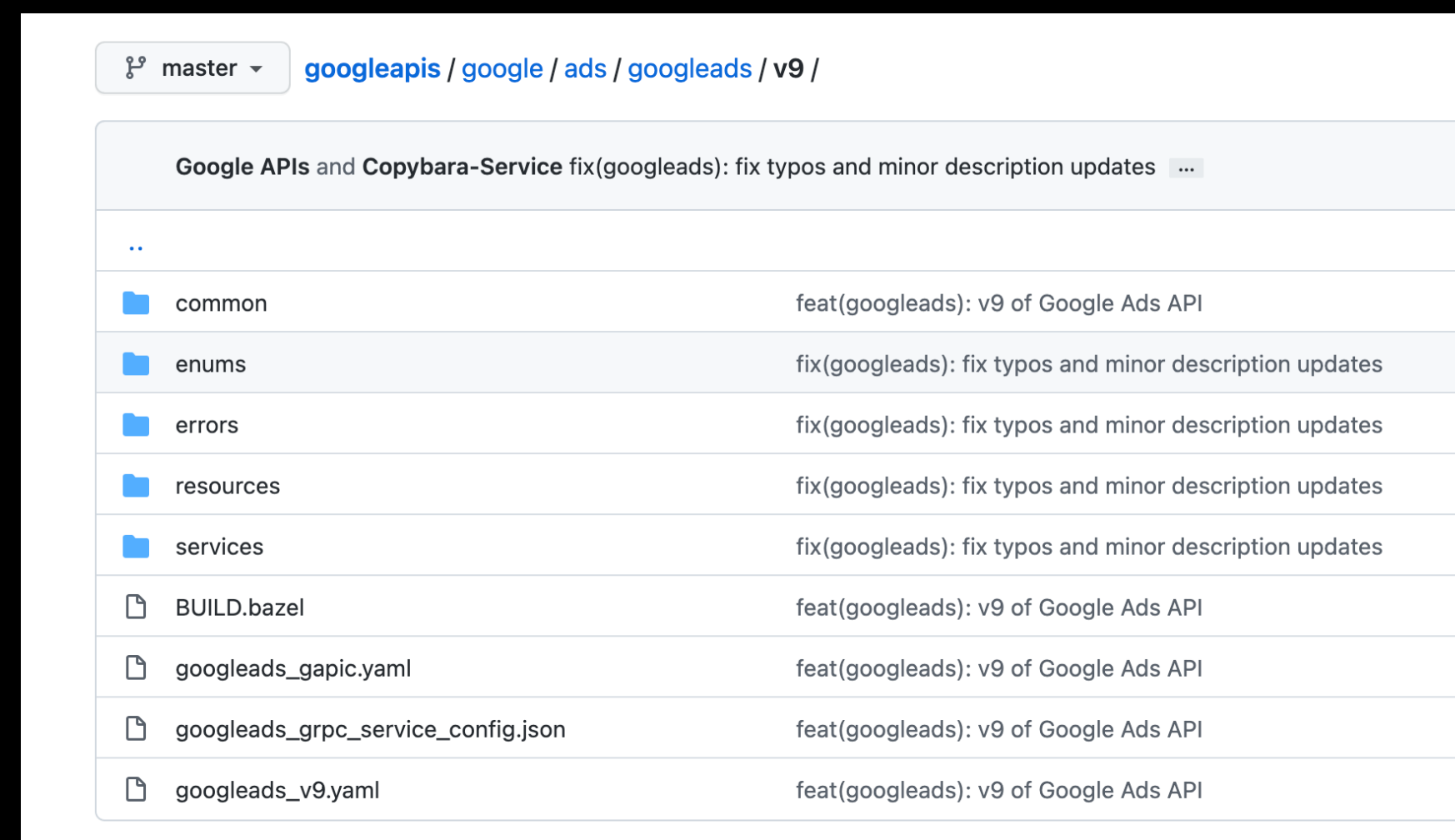
bapis

-account // DCDN 转发到数据中心

--passport // ELB 转发到 APIGW

---login // APIGW 转发到 Service

----v1



master googleapis / google / ads / googleads / v9 /	
Google APIs and Copybara-Service fix(googleads): fix typos and minor description updates ...	
..	
common	feat(googleads): v9 of Google Ads API
enums	fix(googleads): fix typos and minor description updates
errors	fix(googleads): fix typos and minor description updates
resources	fix(googleads): fix typos and minor description updates
services	fix(googleads): fix typos and minor description updates
BUILD.bazel	feat(googleads): v9 of Google Ads API
googleads_gapic.yaml	feat(googleads): v9 of Google Ads API
googleads_grpc_service_config.json	feat(googleads): v9 of Google Ads API
googleads_v9.yaml	feat(googleads): v9 of Google Ads API

<https://github.com/googleapis/googleapis>

API Error Reason

中间件开发者沙龙

错误传播

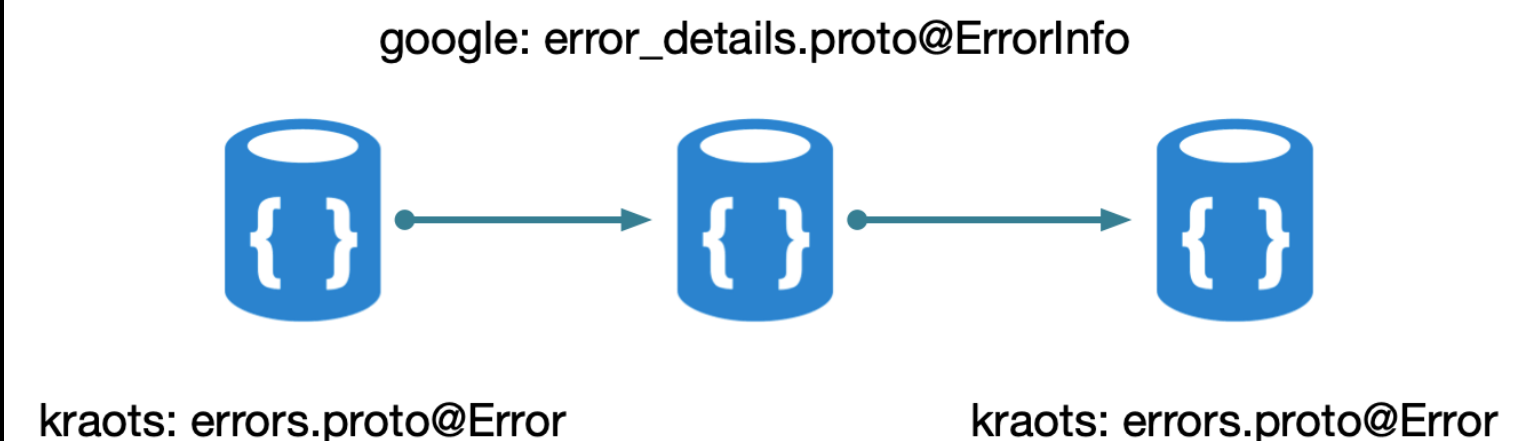
如果您的 API 服务依赖于其他服务，则不应盲目地将这些服务的错误传播到您的客户端。在翻译错误时，我们建议执行以下操作：

- 隐藏实现详细信息和机密信息。
- 调整负责该错误的一方。例如，从另一个服务接收

INVALID_ARGUMENT 错误的服务器应该将
INTERNAL 传播给它自己的调用者。

```
message Error {  
    int32 code = 1;  
    string reason = 2;  
    string message = 3;  
    map<string, string> metadata = 4;  
};
```

```
{  
    // 错误码, 跟 http-status 一致, 并且在 grpc 中可以转换成 grpc-status  
    "code": 500,  
    // 错误原因, 定义为业务判定错误码  
    "reason": "USER_NOT_FOUND",  
    // 错误信息, 为用户可读的信息, 可作为用户提示内容  
    "message": "invalid argument error",  
    // 错误元信息, 为错误添加附加可扩展信息  
    "metadata": {}  
}
```



Transport HTTP & gRPC

中间件开发者沙龙

Kratos 框架对传输层进行了抽象，用户也可以实现自己的传输层，框架默认实现了 gRPC 和 HTTP 两种通信协议传输层。

Transport 主要的接口：

- Server
 - 服务的启动和停止，用于管理服务生命周期。
- Transporter
 - Kind，代表实现的通讯协议的类型。
 - Endpoint，提供的服务终端地址。
 - Operation，用于标识服务的方法路径
 - Header，请求头的元数据
- Endpointer
 - 用于实现注册到注册中心的终端地址
 - 如果不实现这个方法则不会注册到注册中心

```
// Server is transport server.
type Server interface {
    Start(context.Context) error
    Stop(context.Context) error
}

// Endpointer is registry endpoint.
type Endpointer interface {
    Endpoint() (*url.URL, error)
}

// Header is the storage medium used by a Header.
type Header interface {
    Get(key string) string
    Set(key string, value string)
    Keys() []string
}

// Transporter is transport context value interface.
type Transporter interface {
    // grpc
    // http
    Kind() Kind
    // Server Transport: grpc://127.0.0.1:9000
    // Client Transport: discovery:///provider-demo
    Endpoint() string
    // Service full method selector generated by protobuf
    // example: /helloworld.Greeter/SayHello
    Operation() string
    // http: http.Header
    // grpc: metadata.MD
    Header() Header
}
```

Middleware 插件化使用

中间件开发者沙龙

Kratos 内置了一系列的中间件用于处理日志、指标、跟踪链等通用场景。用户也可以通过实现 Middleware 接口，开发自定义 middleware，进行通用的业务处理，比如用户鉴权等。

主要的内置中间件：

- recovery，用于 recovery panic
- tracing，用于启用 trace
- logging，用于请求日志的记录
- metrics，用于启用 metrics
- validate，用于处理参数校验
- metadata，用于启用元信息传递

```
// Handler defines the handler invoked by Middleware.
type Handler func(ctx context.Context, req interface{}) (interface{}, error)

// Middleware is HTTP/gRPC transport middleware.
type Middleware func(Handler) Handler

// Chain returns a Middleware that specifies the chained handler for endpoint.
func Chain(m ...Middleware) Middleware {
    return func(next Handler) Handler {
        for i := len(m) - 1; i >= 0; i-- {
            next = m[i](next)
        }
        return next
    }
}
```


Kratos 启动管理器

中间件开发者沙龙

在 Kratos 中，可以通过实现 `transprt.Server` 接口，然后通过 `kratos.New` 启动器进行管理服务生命周期。

启动器主要处理：

- server 生命周期管理
- registry 注册中心管理

```
// Server is transport server.
type Server interface {
    Start(context.Context) error
    Stop(context.Context) error
}

// Endpointer is registry endpoint.
type Endpointer interface {
    Endpoint() (*url.URL, error)
}
```

```
func main() {
    s := &server{}
    grpcSrv := grpc.NewServer(
        grpc.Address(":9000"),
        grpc.Middleware(
            recovery.Recovery(),
        ),
    )
    httpSrv := http.NewServer(
        http.Address(":8000"),
        http.Middleware(
            recovery.Recovery(),
        ),
    )
    helloworld.RegisterGreeterServer(grpcSrv, s)
    helloworld.RegisterGreeterHTTPServer(httpSrv, s)

    app := kratos.New(
        kratos.Name(Name),
        kratos.Server(
            httpSrv,
            grpcSrv,
        ),
    )

    if err := app.Run(); err != nil {
        log.Fatal(err)
    }
}
```

公众号:
哔哩哔哩技术



微服务框架：<https://github.com/go-kratos/kratos>

中间件开发者沙龙