Rpc客户端调用库框架搭建

联系QQ：2816010068，加入会员群

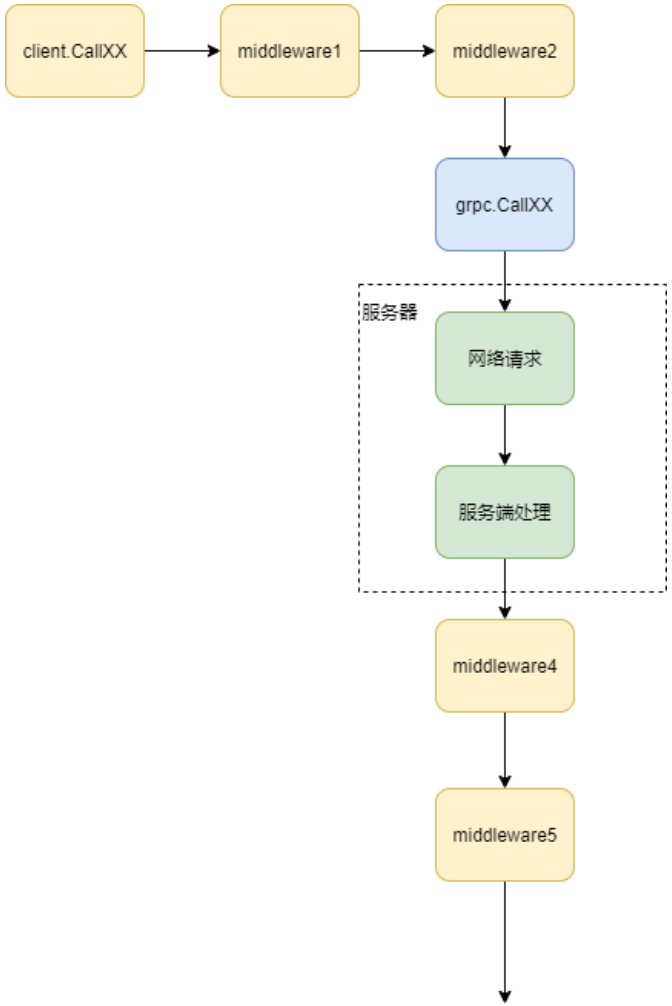
二叉树出品

目录

- 手动封装
- 集成代码自动生成
- Demo测试

手动封装

- 核心流程图


client.CallXX
middleware1
middleware2
grpc.CallXX
服务器
网络请求
服务端处理
middleware4
middleware5

手动封装

- 核心功能
  - 负载均衡
    - 为了服务的高可用，一般一个服务至少部署两个机器。所以需要有一个策略，进行选择机器
  - 序列化&反序列化
    - Protobuf
    - Json
    - Thrift
  - 重试和容错
    - 一台机器挂掉了，怎么处理？
  - 服务发现
    - 微服务架构，所有服务的元信息都是动态的维护在注册中心
  - 过载保护
    - 限流策略
    - 熔断策略

手动封装

- 用户如何调用
  - 通过服务名进行调用
    - 集成服务发现功能
    - 实现负载均衡功能
  - 对用户提供友好的调用接口
    - `client := NewXXXClient(serviceName)`
    - `resp, err := client.CallXXX(ctx, req)`
    - `if err != nil {xxx}`

手动封装

- 第一版开发
  - 封装原生的客户端

```
type HelloClient struct {
    serviceName string
}

func NewHelloClient(serviceName string) *HelloClient {
    return &HelloClient{
        serviceName: serviceName,
    }
}
```

手动封装

- 方法封装
  - 建立连接
  - 生成原生client
  - 调用

```
func (h *HelloClient) SayHello(ctx context.Context, in *hello.HelloRequest, opts ...grpc.CallOption

    conn, err := grpc.Dial(address, grpc.WithInsecure())
    if err != nil {
        logs.Error(ctx, "did not connect: %v", err)
        return nil, err
    }

    defer conn.Close()
    client := hello.NewHelloServiceClient(conn)
    return client.SayHello(ctx, in, opts...)
}
```

手动封装

- 代码测试
  - 记得要先启动server

```go
func myClientExample() {
    client := NewHelloClient("hello")
    ctx := context.Background()
    resp, err := client.SayHello(ctx, &hello.HelloRequest{Name: "test my client"})
    if err != nil {
        logs.Error(ctx, "could not greet: %v", err)
        return
    }

    logs.Info(ctx, "Greeting: %s", resp.Reply)
    return
}
```

第二版封装

- 增加中间件的支持
  - 把原生调用封装成中间件函数的形式
  - Koala库中增加rpc包，用来构建中间件架构模式
  - 整合测试

第二版封装

- 原生调用封装成中间件函数

```
func mwClientSayHello(ctx context.Context, request interface{}) (resp interface{}, err error) {
    conn, err := grpc.Dial(address, grpc.WithInsecure())
    if err != nil {
        logs.Error(ctx, "did not connect: %v", err)
        return nil, err
    }
    req := request.(*hello.HelloRequest)
    defer conn.Close()
    client := hello.NewHelloServiceClient(conn)
    return client.SayHello(ctx, req)
}
```

第二版封装

- Koala库中增加rpc包，用来构建中间件架构模式

```go
func BuildClientMiddleware(handle middleware.MiddlewareFunc) middleware.MiddlewareFunc {
    var mids []middleware.Middleware
    if len(mids) == 0 {
        return handle
    }

    m := middleware.Chain(mids[0], mids...)
    return m(handle)
}
```

第二版封装

- 整合测试核心代码

```
func (h *HelloClient) SayHello(ctx context.Context, in *hello.HelloRequest, opts ...grpc.CallOptio

    middlewareFunc := rpc.BuildClientMiddleware(mwClientSayHello)
    mkResp, err := middlewareFunc(ctx, in)
    if err != nil {
        return nil, err
    }

    resp, ok := mkResp.(*hello.HelloResponse)
    if !ok {
        err = fmt.Errorf("invalid resp, not *hello.HelloResponse")
        return nil, err
    }

    return resp, err
}
```

自动代码生成功能开发

- 模板抽象
- 实现rpc_client_generator.go，渲染模板
- 集合代码

自动代码生成功能开发

- 模板抽象
  - 以newclient.go为原型，把可变参数通过模板变量进行替换

自动代码生成功能开发

- 实现rpc_client_generator.go，渲染模板

自动代码生成功能开发

- 集成代码，进行测试
  - 实现-c命令行配置项
    - 服务端生成器和客户端生成器分别进行注册
    - 通过-c命令行配置项进行判断，分别调用生成客户端代码和服务端代码