链滴

二、go-kit 与 grpc 结合开发微服务

作者：450370050

原文链接：https://ld246.com/article/1524822608375

来源网站：链滴

介绍

go-kit 是一个微服务的开发工具集，微服务系统中的大多数常见问题，因此，使用者可以将精力集中业务逻辑上。

grpc缺乏服务治理的功能，我们可以通过go-kit结合grpc来实现我们的完整需求。go-kit抽象的endpoint设计让我们可以很容易包装其它微服务框架使用的协议。

go-kit提供以下功能：

1、Circuit breaker（熔断器）

2、Rate limiter（限流器）

3、Logging（日志）

4、Metrics（Prometheus统计）

5、Request tracing（请求跟踪）

6、Service discovery and load balancing（服务发现和负载均衡）

安装go包

git clone https://github.com/go-kit/kit.git

封装grpc服务

之前已经搭建过grpc的一个微服务实例BookServer,我们在此服务的基础上包装go-kit

go-kit [TransportServer]

一个Transport的Server 必须要拥有endPoint、decodeRequestFunc、encodeResponseFunc

1、 endPoint一个端点代表一个RPC，也就是我们服务接口中的一个函数

2、 decodeRequestFunc 请求参数解码函数

3、 encodeResponseFunc 返回参数编码函数

请求流程：

请求->decodeRequestFunc -> endPoint -> encodeResponseFunc -> 返回输出

```
/**
bookListHandler := grpc_transport.NewServer(
  makeGetBookListEndpoint(),
  decodeRequest,
  encodeResponse,
)
```

```
*/

//创建一个endPoint
func makeGetBookInfoEndpoint() endpoint.Endpoint {
   return func(ctx context.Context, request interface{}) (interface{}, error) {
      req := request.(*book.BookInfoParams)
      b := new(book.BookInfo)
      b.BookId = req.BookId
      b.BookName = "21天精通php"
      return b,nil
   }
}
//请求数据解码函数
func decodeRequest(_ context.Context, req interface{}) (interface{}, error) {
   return req, nil
}
//返回数据编码函数
func encodeResponse(_ context.Context, rsp interface{}) (interface{}, error) {
   return rsp, nil
}
```

完整代码

```
package main

import (
   "grpc-test/pb"
   "context"
   grpc_transport "github.com/go-kit/kit/transport/grpc"
   "github.com/go-kit/kit/endpoint"
   "google.golang.org/grpc"
   "net"
)

type BookServer struct {
   bookListHandler  grpc_transport.Handler
   bookInfoHandler  grpc_transport.Handler
}

//通过grpc调用GetBookInfo时,GetBookInfo只做数据透传, 调用BookServer中对应Handler.ServeGPC转交给go-kit处理
func (s *BookServer) GetBookInfo(ctx context.Context, in *book.BookInfoParams) (*book.Booknfo, error) {
   _, rsp, err := s.bookInfoHandler.ServeGRPC(ctx, in)
   if err != nil {
      return nil, err
   }
   return rsp.(*book.BookInfo),err
}
//通过grpc调用GetBookList时,GetBookList只做数据透传, 调用BookServer中对应Handler.ServeGRC转交给go-kit处理
func (s *BookServer) GetBookList(ctx context.Context, in *book.BookListParams) (*book.BookLst, error) {
   _, rsp, err := s.bookListHandler.ServeGRPC(ctx, in)
```

```
    if err != nil {
        return nil, err
    }
    return rsp.(*book.BookList),err
}

//创建bookList的EndPoint
func makeGetBookListEndpoint() endpoint.Endpoint {
    return func(ctx context.Context, request interface{}) (interface{}, error) {
        //请求列表时返回 书籍列表
        bl := new(book.BookList)
        bl.BookList = append(bl.BookList, &book.BookInfo{BookId:1,BookName:"21天精通php"})
        bl.BookList = append(bl.BookList, &book.BookInfo{BookId:2,BookName:"21天精通java"})
        return bl,nil
    }
}
//创建bookInfo的EndPoint
func makeGetBookInfoEndpoint() endpoint.Endpoint {
    return func(ctx context.Context, request interface{}) (interface{}, error) {
        //请求详情时返回 书籍信息
        req := request.(*book.BookInfoParams)
        b := new(book.BookInfo)
        b.BookId = req.BookId
        b.BookName = "21天精通php"
        return b,nil
    }
}

func decodeRequest(_ context.Context, req interface{}) (interface{}, error) {
    return req, nil
}

func encodeResponse(_ context.Context, req interface{}) (interface{}, error) {
    return req, nil
}


func main() {
    //包装BookServer

    bookServer := new(BookServer)
    //创建bookList的Handler
    bookListHandler := grpc_transport.NewServer(
        makeGetBookListEndpoint(),
        decodeRequest,
        encodeResponse,
    )
    //bookServer 增加 go-kit流程的 bookList处理逻辑
    bookServer.bookListHandler = bookListHandler

    //创建bookInfo的Handler
    bookInfoHandler := grpc_transport.NewServer(
        makeGetBookInfoEndpoint(),
        decodeRequest,
```

```
        encodeResponse,
    )
    //bookServer 增加 go-kit流程的 bookInfo处理逻辑
    bookServer.bookInfoHandler = bookInfoHandler

    //启动grpc服务
    serviceAddress := ":50052"
    ls, _ := net.Listen("tcp", serviceAddress)
    gs := grpc.NewServer()
    book.RegisterBookServiceServer(gs, bookServer)
    gs.Serve(ls)
}
```

测试

启动服务端，客户端代码不做改变还以我们之前的grpc客户端实例代码启动。

```
GOROOT=D:\Go #gosetup

GOPATH=D:\go\gopath #gosetup
D:\Go\bin\go.exe build -i -o C:\Users\Administrator\AppData\Local\Temp\___8373go_build_
ain_go.exe D:/go/gopath/src/grpc-test/client/main.go #gosetup
D:\软件\GoLand\bin\runnerw.exe C:\Users\Administrator\AppData\Local\Temp\___8373go_bui
d_main_go.exe #gosetup
获取书籍详情
bookId: 1  =>  bookName: 21天精通php
获取书籍列表
bookId: 1  =>  bookName: 21天精通php
bookId: 2  =>  bookName: 21天精通java

Process finished with exit code 0
```