链滴

使用 grpc 开发 RPC 服务 (一)

作者：leoython
原文链接：https://ld246.com/article/1554643590946
来源网站：链滴

前言

笔者最近换了一份工作从一家新零售公司去到一家做电子商务的公司，主要的编程语言也从NodeJs为了Go，因为新公司使用的是grpc做的微服务，所以要重新开始学习新的东西了，正好把这周学习东西做个总结。

整个系列主要涉及Golang、gRPC、go-micros、Docker、Docker-compose、consul，通过本系，你可以了解到如何1、使用grpc/go-mircos构建微服务，2、使用docker进行服务的部署，3、使用onsul进行服务发现

grpc简介

gRPC是谷歌开源的一款跨平台、高性能的RPC框架，笔者目前主要使用它来进行后端微服务的开发。

可能会有的同学对RPC不太熟悉,其实在笔者看来，RPC和HTTP并无多大的区别都是一种调用方式，别则是在于RPC会限制传输协议、传输的参数等，以此换取高效的传输流程，比如grpc就使用的是google开源的protobuf协议，使用TCP的方式进行传输，使得请求比起普通的JSON+HTPP更加快捷。于更多protobuf的信息，可以查看这里

必要的准备

- 了解Golang及其生态
- 安装gRPc及protobuf，　教程
- 安装golang
- 安装protobuf编译器

本期目标

本期目标是使用gRPC实现一个非常小的微服务user-service，服务的功能非常简单，只提供一个获取

户信息的接口

一、首先我们需要定义好整个服务的protobuf文件user.proto

```
syntax = "proto3";  // 指定语法格式，注意 proto3 不再支持 proto2 的 required 和 optional
package  proto;     // 指定生成的 user.pb.go 的包名，防止命名冲突

// service 定义开放调用的服务，即 UserInfoService 微服务
service  UserInfoService {
// rpc 定义服务内的 GetUserInfo 远程调用
rpc  GetUserInfo (UserRequest) returns (UserResponse) {
}
}



// message 对应生成代码的 struct

// 定义客户端请求的数据格式

message  UserRequest {

// [修饰符] 类型 字段名 = 标识符;

string  name = 1;

}

// 定义服务端响应的数据格式

message  UserResponse {

int32  id = 1;

string  name = 2;

int32  age = 3;

repeated  string  title = 4;  // repeated 修饰符表示字段是可变数组，即 slice 类型

}
```

然后我们通过protoc命令编译proto文件，生成对应的go文件

```
protoc -I . --go_out=plugins=grpc:. ./user.proto
```

具体文件太长就不放出来展示了

二、我们来实现server.go

首先我们应该明确实现的步骤：

1、实现GetUserInfo接口

2、使用gRPC建立服务，监听端口

3、将我们实现的服务注册到gRPC中去

话不多说，代码如下

```
package  main

import (
"fmt"
"log"
"net"
// Import the generated protobuf code
pb "go_mirco_service/proto"
"golang.org/x/net/context"
"google.golang.org/grpc"

)
type  UserInfoService  struct{}
var  u  = UserInfoService{}

func (u *UserInfoService) GetUserInfo(ctx context.Context, req *pb.UserRequest) (resp *pb.Use
Response, err error) {

name  := req.Name

if name ==  "leoython" {

resp  =  &pb.UserResponse{

Id: 233,

        Name: name,

Age: 20,

        Title: []string{"Gopher"},

    }

  }

err  =  nil

return

}


func  main() {

port  :=  ":2333"

l, err  := net.Listen("tcp", port)
```

```
if err !=  nil {
        log.Fatalf("listen error: %v\n", err)
    }
    fmt.Printf("listen %s\n", port)
s  := grpc.NewServer()

// 将 UserInfoService 注册到 gRPC
// 注意第二个参数 UserInfoServiceServer 是接口类型的变量
// 需要取地址传参
    pb.RegisterUserInfoServiceServer(s, &u)
    s.Serve(l)
}
```

到此我们就实现了利用gRPC实现了一个非常简单但是五脏俱全的RPC服务，但是却出现了一个问题我们无法直接调用，所以我们还需要实现一个调用server的客户端，代码如下

```
package  main

import (
"fmt"
"log"

pb "go_mirco_service/proto"

"golang.org/x/net/context"
"google.golang.org/grpc"
)

func  main() {
```

```
conn, err  := grpc.Dial(":2333", grpc.WithInsecure())
if err !=  nil {
        log.Fatalf("dial error: %v\n", err)
    }
defer conn.Close()

// 实例化 UserInfoService 微服务的客户端
client  := pb.NewUserInfoServiceClient(conn)

// 调用服务
req  :=  new(pb.UserRequest)
req.Name  =  "leoython"
resp, err  := client.GetUserInfo(context.Background(), req)
if err !=  nil {
        log.Fatalf("resp error: %v\n", err)
    }
    fmt.Printf("Recevied: %v\n", resp)
}
```

结语

至此，我们已经学会使用gRPC进行开发，采用protobuf进行参数的定义，下一篇笔者将会使用grpc-ateway将RPC接口转换为rest接口供客户端调用，而不需要客户端实现RPC，这也是现在主流微服务一种服务提供方式，对外使用REST，对内使用RPC，关于更多微服务的内容，推荐查看nginx的关于服务的文章