Rpc客户端调用库架构设计

联系QQ：2816010068，加入会员群

目录

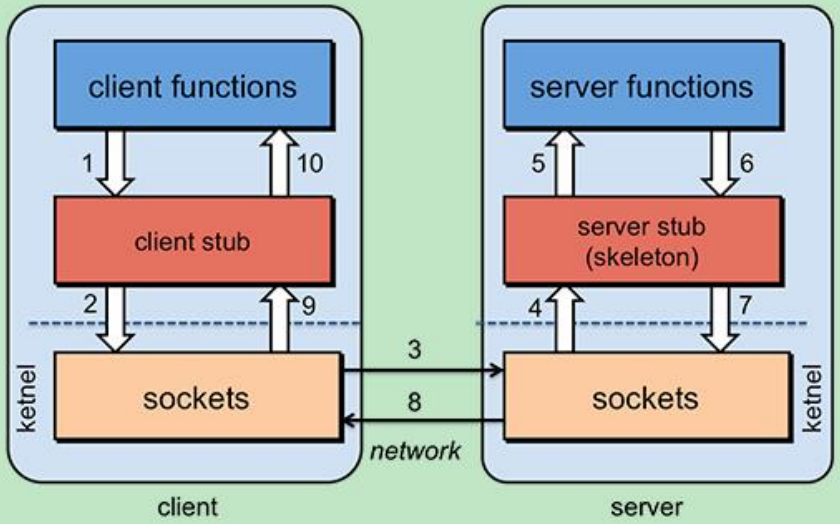
- Rpc简介
- 原生代码分析
- 基于中间件的架构设计

Rpc简介

- 本地过程调用
  - 函数调用
  - 参数和逻辑实现都在本地
  - 没有网络交互
- Rpc,全称Remote Procedure Call, 远程过程调用
  - 调用方,一般称为客户端
  - 功能实现方,一般称为服务端
  - 客户端和服务端可以在同一台机器上,也可以是不同机器上
  - 客户端和服务端通过网络进行通信

Rpc简介

- 调用流程



（图片来源：cs.rutgers.edu/~pxk/417...）

为什么RPC呢？就是无法在一个进程内，甚至一个计算机内通过本地调用的方式完成的需求，比如比如不同的系统间的通讯，甚至不同的组织间的通讯。由于计算能力需要横向扩展，需要在多台机器组成的集群上部署应用，

RPC的协议有很多，比如最早的CORBA，Java RMI，Web Service的RPC风格，Hessian，Thrift，甚至Rest API。

Rpc调用流程

- 总结
  - 调用方(client)准备好调用函数的参数
  - 建立好client到server的连接
  - 把client调用函数的参数、调用函数名字进行序列化，得到网络字节流
  - Client通过第二步建立的连接把网络字节流发送到server
  - Server接收到client发送的网络字节流，一般都把这个过程叫做一个请求
  - Server进行反序列化，拿到调用的函数名以及该函数的参数
  - Server通过函数名、参数调用该函数的具体实现，一般把这个过程叫做请求路由
  - Server调用后拿到函数处理的结果，并进行序列化，得到网络字节流
  - Server通过第二步建立的连接，把结果发送给客户端
  - client拿到结果，并进行反序列化
  - Client拿到该Rpc调用的结果，并进行其他其他业务处理

Rpc简介

- 需要处理的问题
  - 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

原生代码分析

- 建立连接

```
const (
    address     = "localhost:8080"
    defaultName = "world"
)

func main() {
    conn, err := grpc.Dial(address, grpc.WithInsecure())
    if err != nil {
        log.Fatal("did not connect: %v", err)
    }
    defer conn.Close()
    c := pb.NewHelloServiceClient(conn)

    name := defaultName
    if len(os.Args) > 1 {
        name = os.Args[1]
    }
```

原生代码分析

- 初始化client实例

```
c := pb.NewHelloServiceClient(conn)

name := defaultName
if len(os.Args) > 1 {
    name = os.Args[1]
}
```

原生代码分析

- 使用client实现进行调用

```
ctx := context.Background()
ctx = metadata.AppendToOutgoingContext(ctx, "koala_trace_id", "88888888888888888888888888")
r, err := c.SayHello(ctx, &pb.HelloRequest{Name: name})
if err != nil {
    log.Printf("could not greet: %v", err)
    continue
}
_ = r
log.Printf("Greeting: %s", r.Reply)
//time.Sleep(time.Millisecond * 10)
```

原生代码存在的问题

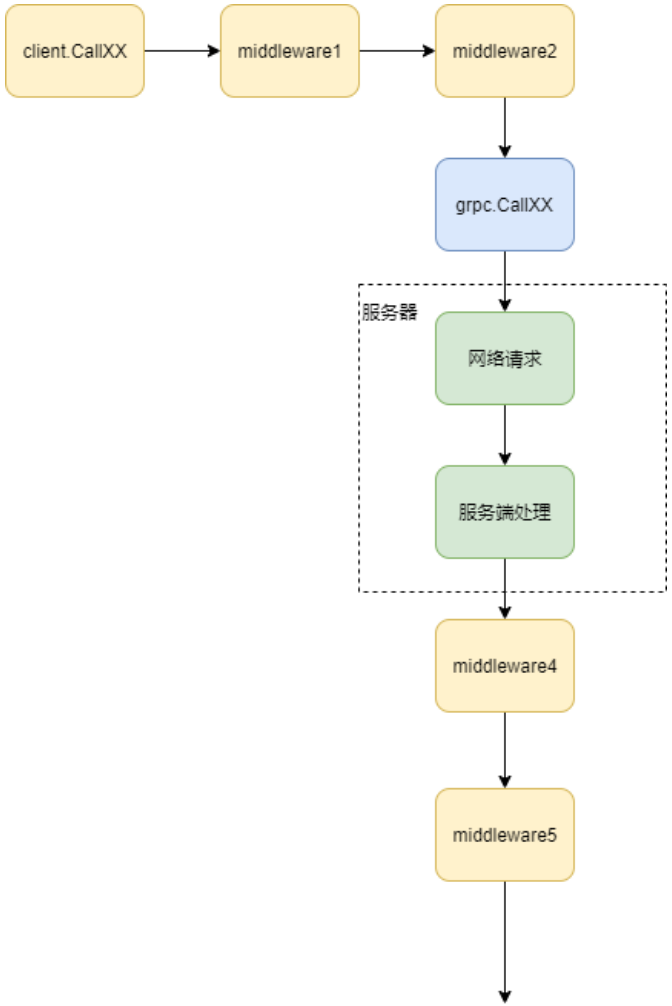
- 不好的地方
  - 不支持负载均衡
  - 不支持服务发现
  - 不支持熔断
  - 不支持重试
  - 不支持超时
  - …
- 好的地方
  - 支持序列化&反序列化
  - 支持网络传输
  - …

Koala rpc库设计

- 设计理念
  - 可扩展性好
  - 可维护性好
  - 高性能
  - 容错性强
- 架构模式
  - 采用中间件的架构方式

Koala rpc库设计

- 基于中间件的设计理念



Koala rpc库设计

- 中间件支持
  - 负载均衡中间件
  - 限流中间件
  - 熔断中间件
  - Rpc日志打印中间件
  - 分布式追踪中间件

client代码生成

- 封装中间件
- 封装grpc原生代码

- 细节
  - 自动生成XXXClient， XXXClient集成中间件和grpc原生代码的功能