Goland之网络编程-RPC

一、什么是RPC协议

RPC即远程过程调用(Remote Procedure Call)，用于构建计算机之间的通信协议，该协议允许运行于一台计算机的程序调用另一台计算机上的程序，开发人员无需对交互过程进行编程

rpc和rpc/jsonrpc包提供了对RPC的支持

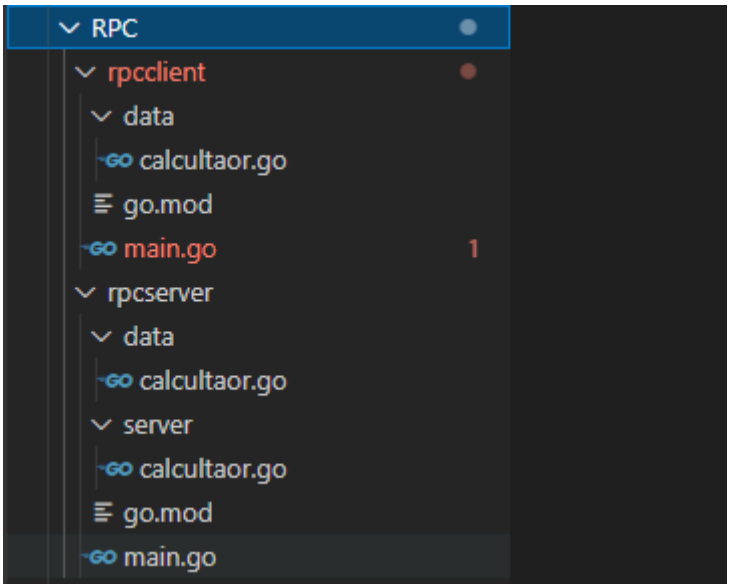
- rpc构建于TCP或HTTP协议之上，底层数据编码使用gob，因gob编码为golang自定义，所以无法支持跨语言调用
- rpc/jsonrpc构建于TCP协议之上，底层数据编码使用json，可支持跨语言调用

实际上就是一个请求，一个响应

二、实现一个简单的JSONRPC

2.1、目录结构

client的data和server的data是一样的，模拟远程仓库的数据



2.2、data--calcultaor.go

```
package data

// RPC必须要有的请求对象
type CalcultaorRequest struct {
    Left  int
    Rigth int
}

// RPC必须要有的响应对象
type CalcultaorReponse struct {
    Result int
}
```

2.3、rpcserver--server

```
package server

import (
    "log"
    "rpcserver/data"
)

// RPC必须要有的结构体 (定义计算服务)
type Calcultaor struct {
}

// RPC必须要有的方法  (Add方法)
func (c *Calcultaor) Add(request *data.CalcultaorRequest, reponse
*data.CalcultaorReponse) error {
    log.Printf("+ call ADD \n")
    reponse.Result = request.Rigth + request.Left
    return nil
}
```

2.4、rpcserver--main.go

```
package main

import (
    "fmt"
    "log"
    "net"
    "net/rpc"
    "net/rpc/jsonrpc"
    "rpcserver/server"
)

func main() {

    // 注册服务（==暴露服务）未指定名称默认使用结构体名
    rpc.Register(&server.Calcultaor{})
    addr := ":9999"
    lister, err := net.Listen("tcp", addr)
    if err != nil {
        log.Fatal(err)
    }
```

```
    defer lister.Close()
    fmt.Printf("server的ip: port: %s \n", addr)

    for {
        // 处理客户端连接
        conn, err := lister.Accept()
        if err != nil {
            log.Printf(err.Error())
            continue
        }
        log.Println(conn.RemoteAddr())

        // 使用例程处理客户端请求
        go jsonrpc.ServeConn(conn)
    }
}
```

2.5、rpcclient-main.go

```
package main

import (
    "fmt"
    "log"
    "net/rpc/jsonrpc"
    "rpcclient/data"
)

func main() {

    addr := ":9999"
    conn, err := jsonrpc.Dial("tcp", addr)
    if err != nil {
        log.Fatal(err)
    }
    defer conn.Close()

    // 定义请求对象
    request := &data.CalcultaorRequest{2, 5}

    // 定义响应对象
    reponse := &data.CalcultaorReponse{}

    // 调用远程方法
    err1 := conn.Call("Calcultaor.Add", request, reponse)

    // 获取调用结果
    fmt.Println(err1, reponse.Result)
}
```