慕课网慕课教程 11. go 语言的 rpc 之 hello world 涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

Go 语言的 RPC 包的路径为 net/rpc，也就是放在了 net 包目录下面。因此我们可以猜测该 RPC 包是建立在 net 包基础之上的。在第一章 “Hello, World” 革命一节最后，我们基于 http 实现了一个打印例子。下面我们尝试基于 rpc 实现一个类似的例子。

```
package main

import (
  "net"
  "net/rpc"
)

type HelloService struct {}
func (s *HelloService) Hello(request string, reply *string) error {
  *reply = "hello "+ request
  return nil
}

func main(){
  _ = rpc.RegisterName("HelloService", &HelloService{})
  listener, err := net.Listen("tcp", ":1234")
  if err != nil {
    panic("监听端口失败")
  }
  conn, err := listener.Accept()
  if err != nil {
    panic("建立链接失败")
  }
  rpc.ServeConn(conn)

}
```

其中 Hello 方法`必须满足Go语言的RPC规则`：方法`只能有两个可序列化的参数，其中第二个参数是指针类型，并且返回一个error类型，同时必须是公开的方法`。
然后就可以将 HelloService 类型的对象注册为一个 RPC 服务：(TCP RPC 服务)。
其中`rpc.Register`函数调用会将对象类型中所有满足 RPC 规则的对象方法注册为 RPC 函数，所有注册的方法会放在 “HelloService” 服务空间之下。然后我们建立一个唯一的 TCP 链接，并且通过 rpc.ServeConn 函数在该 TCP 链接上为对方提供 RPC 服务。

```
func main() {
    client, err := rpc.Dial("tcp", "localhost:1234")
    if err != nil {
        log.Fatal("dialing:", err)
```

```
    }

    var reply string
    err = client.Call("HelloService.Hello", "hello", &reply)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println(reply)
}
```

首先是通过`rpc.Dial`拨号 RPC 服务，然后通过`client.Call`调用具体的 RPC 方法。在调用`client.Call`时，`第一个参数是用点号链接的RPC服务名字和方法名字，第二和第三个参数分别我们定义RPC方法的两个参数`。