> 慕课网慕课教程 8. 什么是 rpc 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。

远程过程调用带来的新问题
在远程调用时，我们需要执行的函数体是在远程的机器上的，也就是说，add 是在另一个进程中执行的。这就带来了几个新问题：

1. **Call ID 映射**。我们怎么告诉远程机器我们要调用 add，而不是 sub 或者 Foo 呢？在本地调用中，函数体是直接通过函数指针来指定的，我们调用 add，编译器就自动帮我们调用它相应的函数指针。但是在远程调用中，函数指针是不行的，因为两个进程的地址空间是完全不一样的。所以，在 RPC 中，所有的函数都必须有自己的一个 ID。这个 ID 在所有进程中都是唯一确定的。客户端在做远程过程调用时，必须附上这个 ID。然后我们还需要在客户端和服务端分别维护一个 {函数 <-> Call ID} 的对应表。两者的表不一定需要完全相同，但相同的函数对应的 Call ID 必须相同。当客户端需要进行远程调用时，它就查一下这个表，找出相应的 Call ID，然后把它传给服务端，服务端也通过查表，来确定客户端需要调用的函数，然后执行相应函数的代码。
2. **序列化和反序列化**。客户端怎么把参数值传给远程的函数呢？在本地调用中，我们只需要把参数压到栈里，然后让函数自己去栈里读就行。但是在远程过程调用时，客户端跟服务端是不同的进程，不能通过内存来传递参数。甚至有时候客户端和服务端使用的都不是同一种语言（比如服务端用 C++，客户端用 Java 或者 Python）。这时候就需要客户端把参数先转成一个字节流，传给服务端后，再把字节流转成自己能读取的格式。这个过程叫序列化和反序列化。同理，从服务端返回的值也需要序列化反序列化的过程。
3. **网络传输**。远程调用往往用在网络上，客户端和服务端是通过网络连接的。所有的数据都需要通过网络传输，因此就需要有一个网络传输层。网络传输层需要把 Call ID 和序列化后的参数字节流传给服务端，然后再把序列化后的调用结果传回客户端。只要能完成这两者的，都可以作为传输层使用。因此，它所使用的协议其实是不限的，能完成传输就行。尽管大部分 RPC 框架都使用 TCP 协议，但其实 UDP 也可以，而 gRPC 干脆就用了 HTTP2。Java 的 Netty 也属于这层的东西。

解决了上面三个机制，就能实现 RPC 了，具体过程如下：
client 端解决的问题：

```
1. 将这个调用映射为Call ID。这里假设用最简单的字符串当Call ID的方法
2. 将Call ID，a和b序列化。可以直接将它们的值以二进制形式打包
3. 把2中得到的数据包发送给ServerAddr，这需要使用网络传输层
4. 等待服务器返回结果
4. 如果服务器调用成功，那么就将结果反序列化，并赋给total
```

server 端解决的问题

```
1. 在本地维护一个Call ID到函数指针的映射call_id_map，可以用dict完成
2. 等待请求，包括多线程的并发处理能力
3. 得到一个请求后，将其数据包反序列化，得到Call ID
4. 通过在call_id_map中查找，得到相应的函数指针
5. 将a和rb反序列化后，在本地调用add函数，得到结果
6. 将结果序列化后通过网络返回给Client
```

在上面的整个流程中，估计有部分同学看到了熟悉的计算机网络的流程和 web 服务器的定义。
所以要实现一个 RPC 框架，其实只需要按以上流程实现就基本完成了。

其中：

- Call ID 映射可以直接使用函数字符串，也可以使用整数 ID。映射表一般就是一个哈希表。
- 序列化反序列化可以自己写，也可以使用 Protobuf 或者 FlatBuffers 之类的。
- 网络传输库可以自己写 socket，或者用 asio，ZeroMQ，Netty 之类。

实际上真正的开发过程中，除了上面的基本功能以外还需要更多的细节：网络错误、流量控制、超时和重试等。

最后提一个问题： 如何将远程的这些过程写出本地函数调用的感觉来?