作者：洪春涛  
链接：https://www.zhihu.com/question/25536695/answer/221638079  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

**本地过程调用**

RPC就是要像调用本地的函数一样去调远程函数。在研究RPC前，我们先看看本地调用是怎么调的。假设我们要调用函数Multiply来计算lvalue \* rvalue的结果:

1 int Multiply(int l, int r) {

2 int y = l \* r;

3 return y;

4 }

5

6 int lvalue = 10;

7 int rvalue = 20;

8 int l\_times\_r = Multiply(lvalue, rvalue);

那么在第8行时，我们实际上执行了以下操作：

1. 将 lvalue 和 rvalue 的值压栈
2. 进入Multiply函数，取出栈中的值10 和 20，将其赋予 l 和 r
3. 执行第2行代码，计算 l \* r ，并将结果存在 y
4. 将 y 的值压栈，然后从Multiply返回
5. 第8行，从栈中取出返回值 200 ，并赋值给 l\_times\_r

以上5步就是执行本地调用的过程。

**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，Multiply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Multiply，而不是Add或者FooBar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Multiply，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。

所以，要实现一个RPC框架，其实只需要把以上三点实现了就基本完成了。

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。

最后，有兴趣的可以看我们自己写的一个小而精的RPC库 tinyrpc（[hjk41/tinyrpc](https://link.zhihu.com/?target=https%3A//github.com/hjk41/tinyrpc)），对于理解RPC如何工作很有好处。