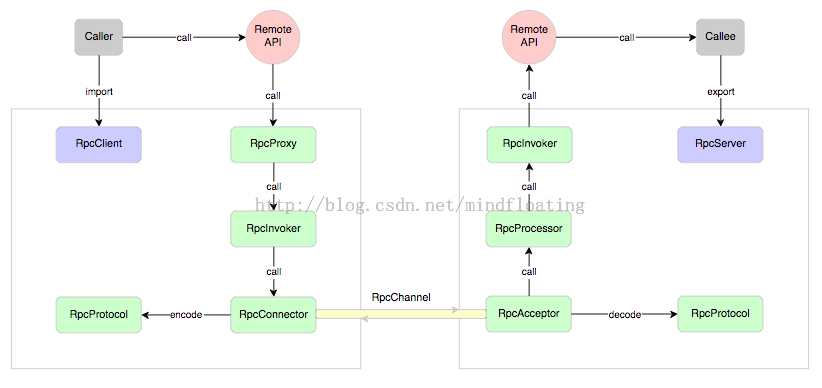
[一个简单RPC框架是如何炼成的(I)——开局篇](http://blog.csdn.net/crylearner/article/details/46898537)

开场白，这是一个关于RPC的相关概念的普及篇系列，主要是通过一步步的调整，提炼出一个相对完整的RPC框架。

RPC（Remote Procedure Call Protocol）——远程过程调用协议，基于C/S模型。网络上有一篇文章写得不错，可以去了解一下相关概念[深入浅出](http://blog.csdn.net/mindfloating/article/details/39474123" \t "_blank)RPC

这里，直接使用一下上面作者的一个示意图



总结下来就是有4块核心内容

1. **RPC数据的传输**。如上面的RPCConnector，RPCChannel。它们主要负责数据传输这一块， 具体客户端与服务器之间的连接是不是socket连接，是原始tcp连接还是使用http，这些RPC协议本身不做任何规定。**那么我们的任务就是抽象出这样一个传输层。**
2. **RPC消息。**如上面的RPCProtocol， 以及encode，decode方法。 因为RPC是远程调用，所以没办法直接函数调用，于是就必须用一套专门的协议，去表示调用以及调用结果。另外，因为实际应用基本都是跨机器连接，所以无法直接传递内存变量，也就是说还需要将消息编码成 诸如字符串一类的可以跨设备传输的内容。具体的RPC消息的封装协议很多，常见的是基于xml，json封装的。**那么我们的任务就是抽象出这样一个协议层。**
3. **RPC服务注册。**如上面Callee -->export。 服务端具体支持哪些调用，收到来自客户端的RPC请求后，怎样去调用真正的需要执行的方法，这些内容也是一个完整的RPC框架必须考虑的。一些稍微高级一点的框架，都是可以服务自动注册的，现在主流的RPC框架，还支持通过 IDL（Interface Definition Language）来定义远程接口，实现跨语言的RPC 。**那么我们的任务就是抽象出一个RPC服务的注册机制**。
4. **RPC消息处理**。如上面的RPCInvoker。这里其实与RPC本身关系不大，一般就是考虑**支持异步/同步调用**。 这一部分，大概我也会做一些说明，但不是这个系列的重点。

这次RPC框架的搭建，庄稼人将采用python作为开发语言，原因嘛，还有点不好意思：代码量少，方便解释（偷懒是码工的天性大笑）。初步的想法是，整个框架是从原始的普通调用开始，然后一步步的演化，最后生成一个完整的RPC框架。希望在这个过程中，能带给各位同学更深刻的RPC入门知识，以及代码修改的知识。说得有点高了，呵呵。