[一个简单RPC框架是如何炼成的(II)——制定RPC消息](http://blog.csdn.net/crylearner/article/details/46899083)

开局篇我们说了，RPC框架的四个核心内容

1. **RPC数据的传输**。
2. **RPC消息 协议**
3. **RPC服务注册**
4. **RPC消息处理**

下面，我们先看一个普通的过程调用

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46899083) [copy](http://blog.csdn.net/crylearner/article/details/46899083)

1. **class** Client(object):
3. **def** \_\_init\_\_(self):
4. self.remote = None
6. ##
7. # 内部是委托给远程remote对象来获取结果。
8. **def** sayHello(self):
9. **if** self.remote: **return** self.remote.sayHello()
10. **else** : **return** None
12. **class** Server(object):
14. **def** \_\_init\_\_(self):
15. **pass**
17. **def** sayHello(self):
18. **return** 'Hello World'

21. **if** \_\_name\_\_ == '\_\_main\_\_':
22. server = Server()
23. client = Client()
24. client.remote = server
25. **print**(client.sayHello())

这是一个常见的过程调用的例子，client调用sayHello，实际委托给Server的sayHello方法来实现。但他不是RPC调用，因为起码不是远程的，另外，也没有我们提到的四个核心内容。于是我们的任务就是通过一点点的代码修改，为其引入RPC框架。

第一步，订协议。俗话说得好，没有规矩不成方圆。

还是继续能多简单就多简单。

RPC请求：Request， 包含一个请求id 和 一个请求命令，如‘sayHello'

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46899083) [copy](http://blog.csdn.net/crylearner/article/details/46899083)

1. **class** Request(object):
2. '''''
3. @RPC请求，包含命令id和请求内容两部分。这个实现，与具体的RPC协议相关。
4. @这里是简化起见，采用python自身的字典作为其协议的数据结构
5. '''

8. **def** \_\_init\_\_(self):
9. '''''
10. Constructor
11. '''
12. self.id = 0 #id的作用在于将Request和Response建立绑定关系.在异步调用时就有用
13. self.command = None #sayHello
15. **def** \_\_str\_\_(self):
16. **return** ''.join(('id: ', str(self.id),  '   command: ', str(self.command)))

同样的，对RPC Response，也定义如下

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46899083) [copy](http://blog.csdn.net/crylearner/article/details/46899083)

1. **class** Response(object):
2. '''''
3. @RPC回复。 包含答复id和执行结果两部分内容。其中答复id与对应的请求id一致。
4. @简单起见，协议的实现使用python自家的字典
5. '''

8. **def** \_\_init\_\_(self):
9. '''''
10. Constructor
11. '''
12. self.id = 0
13. self.result = None
15. **def** \_\_str\_\_(self):
16. **return** ''.join(('id: ', str(self.id),  '   result: ', str(self.result)))

定义好协议之后，我就对Client稍作修改，将原来直接的接口调用 self.remote.sayHello更改为 send Request （command='sayHello'）

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46899083) [copy](http://blog.csdn.net/crylearner/article/details/46899083)

1. **def** sayHello(self):
2. req = Request() // 构建一个RPC请求消息
3. req.id = 1
4. req.command = 'sayHello'
5. **return** self.request(req)
7. **def** request(self, req):
8. rsp = self.remote.procRequest(req) // 将请求消息发送给远程服务端。但因为传输层这里还没实现，所以先暂时还是直接调用远端接口
9. **return** rsp.result

然后，服务端也要相应修改，需要根据request请求中的command命令，调用具体的方法，并将执行结果封装到Response中，返回给客户端。

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46899083) [copy](http://blog.csdn.net/crylearner/article/details/46899083)

1. **def** procRequest(self,req):
2. rsp = Response()
3. rsp.id = req.id
4. **if** req.command == 'sayHello':
5. rsp.result = self.sayHello()
6. **else**:
7. **raise** Exception("unknown command")
9. **return** rsp

到这里，RPC框架中的RPC消息已经初具雏形，不过

* 我们并没有实现相应的encode和decode方法，没有基于可以跨设备的字符串传输，而是直接的内存变量传递。
* 现在的RPC request不支持带参数的请求命令。如add(a, b), 如何在RPC消息中描述参数a,b 。

这些问题我们在后面还会继续展开处理