[一个简单RPC框架是如何炼成的(IV)——实现RPC消息的编解码](http://blog.csdn.net/crylearner/article/details/46900203)

之前我们制定了一个很简单的RPC消息 的格式，但是还遗留了两个问题，上一篇解决掉了一个，还留下一个

* **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 。

下面我们处理掉这个编解码问题。

实际的RPC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。但如果抽象一下，实际也就是一个编解码，管你编码成什么内容呢，就是不编码也可以。管他黑猫白猫，只要能传过去，就是好猫。

这里我还是简单原则，重点在于晓义嘛。利用python里的两个运算。 str 和eval。

假设 一个字典msg =  { 'a' : 1, 'b' : 2}.  那么str(msg) =   " { 'a' : 1, 'b' : 2}", 注意变成字符串喽。

        然后eval（" { 'a' : 1, 'b' : 2}"）-->msg， 做一个eval运算，又从字符串变成 字典变量了。

       于是编码时，先将RPC消息转换成dict，然后调用str编码成字符串。

         解码时，先调用eval 得到dict对象，然后再转换为具体的RPC消息对象

设计已定，剩下的就只是code filling。

先修改一下原来Request的str方法，返回一个dict的字符串表示。对Response也做类似处理

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46900203) [copy](http://blog.csdn.net/crylearner/article/details/46900203)

1. **class** Request(object):
2. **def** \_\_str\_\_(self):
3. **return** str({'id':self.id,  'command':self.command, 'parameter':self.parameter})

然后引入encode方法

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46900203) [copy](http://blog.csdn.net/crylearner/article/details/46900203)

1. @classmethod
2. **def** encode(cls, message):
3. **if** isinstance(message, Request):
4. **return** str(message)
5. **elif** isinstance(message, Response):
6. **return** str(message)
7. **elif** isinstance(message, Notification):
8. **return** str(message)
9. **else**:
10. **raise** Exception('unknown type when encode')

同样的，引入decode方法，稍微复杂一些。主要的麻烦在于如何区分解码出来的是Response还是Request

我的办法是比较投机的，直接根据字典的内容去判断。有command字段的肯定是request，有result字段的肯定是response

**[python]** [view plain](http://blog.csdn.net/crylearner/article/details/46900203) [copy](http://blog.csdn.net/crylearner/article/details/46900203)

1. @classmethod
2. **def** decode(cls, data):
3. info = eval(data)
4. **if** 'command' **in** info:
5. request = Request()
6. request.id = info.get('id')
7. request.command = info.get('command')
8. request.parameter = info.get('parameter', {})
9. **return** request
10. **elif** 'result' **in** info:
11. response = Response()
12. response.id = info.get('id')
13. response.result = info.get('result')
14. **return** response
15. **elif** 'message' **in** info:
16. note = Notification()
17. note.message = info.get('message')
18. **return** note
19. **else**:
20. **raise** Exception('unknown data when decode')

另外，client和server的代码也要稍作调整，那个很简单，调用一下上面的 方法就可以了，这里不贴了。

over，RPC消息 这一块，我们彻底玩完了。

总结：

* Request和Resonse的定义可以一直不变，
* encode 和decode方法，如孙大圣，是可以千变万化的。如果采用xml或者json或者其他的描述，只要自定义encode和decode方法即可
* 从更高的层次看，RPC消息，其实就是对函数调用的一个描述，所以充其量就是view。既然是view，实际自由度是很大的。