# app-common组件简单使用实现

## 参考文章及本文说明：

本文只是简单介绍一下RMI技术的使用和公司内app-common组件的使用（不对原理和底层做过多的解释），关于公司app-common组件设计的具体情况参见嘉耀大佬的相关文章：<http://wiki.info/pages/viewpage.action?pageId=6488263>

由于项目的原因，我需要完成一个将原本完全岛内的区域聊天模式改为全服共有的聊天室聊天模式的功能。功能开始之初，我就被告知如果需要进程间的通信可以使用同步和异步两种方式。

我一开始想着进程间同步通信可以使用RPC，然后通过json来传递参数...，对于具体的实现我没有进行尝试，但是我看了看项目已有功能的代码发现，**在项目中进程间的通信是通过对象来实现的**，即找不RPC也找不到json的影子，**并且直接返回对象**。我想是不是传输过来的只是json、protocolBuf这样的东西，然后项目中自己又封装了一层去转换为所需的对象。可是看过代码之后，发现并不是这样。带着问题我上RTX喊了一句，然后被大神指点了一下去看看RMI（在此之前，我一直把它划分为RPC一类的东西来着），继此我才慢慢发现rmi的本来面目。

上面的附件是本文的源码，其中一共分为四个部分（服务端、客户端、app-common组件和app-rmi组件）app-rmi和app-common是服务端和客户端的共用组件。项目实现了一个**简易投票系统**（客户端的每个玩家有一张票，可以投给自己喜欢的游戏。并且玩家可以实时获取最新的投票结果和自己的投票信息）。为了突出app-common组件的优势，项目中我也给出了一个用Java原生rmi实现的例子（业务上功能一致），区分标准为是否有**“native”**前后缀。

## 原生RMI实现:

首先介绍一下使用java原生rmi实现远程对象调用的过程。实现这样一个demo分为两个部分，服务端部分（负责创建远程对象）和客户端部分（调用服务端的远程对象中的方法），具体步骤为：

### 服务器端：

**1、IVoteNative：定义一个远程接口（这个接口是服务器端和客户端共用的接口，并且该接口要继承java.rmi.Remote接口）注意接口中的方法都要抛出java.rmi.RemoteException异常**

**2、VoteRemoteNative：在服务器端实现上面所说的远程接口，并继承java.rmi.server.UnicastRemoteObject类**

**3、VoteServerNativeMain：在服务器端创建一张注册表，创建远程对象并将它绑定到该注册表中。**

### 客户端：

**4、VoteClientProxyNative：在客户端定义一个远程接口对象，通过服务端的ip和远程对象名，在客户端lookup到远程对象，然后赋值给客户端的远程接口对象。这样就可以实现一个完整的原生rmi demo。**

大致的实现过程就是这样，至于客户端底层是通过什么方式调用服务端的远程对象，我这里不做讨论。上面总结成一句话就是——**定义公共接口，服务端注册对象给客户端调。**

这个原生rmi demo中，有很多功能是没有实现的。比如：没有断线重连机制、没有监听服务端和客户端调用和和客户端对服务端报错的处理等。接下来我会使用app-common组件解决以上的问题。

## app-common组件实现：

使用app-common实现远程对象调用的过程，在这个简单的场景下面是会复杂了，但如果面对一个大一点项目这个实现就相当友好了（抽象出不变的部分作为共用组件，而变化的部分通过扩展实现）。

### 服务器端：

**1、VoteRemote，VoteServerMonitor：定义远程对象接口，实现服务器端监听器；**

**2、使用app-common中的AppServer创建注册表；**

**3、新建远程对象和服务端监听器，然后新建一个CServer对象绑定到注册表中；**

### 客户端：

**1、RemoteServiceProvider<T>：该类是客户端为了复用实现的一个泛型类（当系统中有多个远程对象时，就可以复用代码），它就是将CClient包多一层。并且该类通过初始化的参数获得远程对象。**

**2、VoteRmiHelper：标明服务器远程对象坐标（RemoteNode），然后新建RemoteServiceProvider对象调用对象。并且还注册了客户端的监听器。**

**3、VoteClientMain：客户端直接调用VoteRmiHelper的getService方法就可以调用远程对象的方法调用了。**

## 文章写后感：

app-common组件最颠覆我想法的是断线重连的处理。我一直以为要实现五秒一次重连是一件很困难的事（毕竟我一直以为是app-common主动起个线程然后定时重连来着），然而看了源码之后才发现我被骗了，但是也给了我很好的经验。

app-common里面CClient的defaultService这种模式也给了我很大的启发，虽然一开始并没有真正理解，但是在后来项目的运用中逐步的理解和运用（其实就是一种备用方案，我也只是用来返回错误码不让程序直接跑崩报错而已）。这种算是对业务和系统的保护吧。

还有很多类似的地方，像java.lang.reflect.Proxy实现监听器的功能；AppAdmin提供了shutdownServer(int port)方法，用在关服的时候。等等，这些都是我以前没有考虑过但又很必要的功能。