# 一、远程调用技术概念及原理

在分布式服务框架中，一个最基础的问题就是远程服务是怎么通讯的，在Java领域中有很多可实现远程通讯的技术，例如：RMI、MINA、ESB、Burlap、Hessian、SOAP、EJB和JMS等，这些名词之间到底是些什么关系呢，它们背后到底是基于什么原理实现的呢，了解这些是实现分布式服务框架的基础知识，而如果在性能上有高的要求的话，那深入了解这些技术背后的机制就是必须的了。

## 1.1、在java中实现远程通讯的技术

RMI、MINA、ESB、Burlap、Hessian、SOAP、EJB和JMS

## 1.2、基本原理

要实现网络机器间的通讯，首先得来看看计算机系统网络通信的基本原理，在底层层面去看，网络通信需要做的就是将流从一台计算机传输到另外一台计算机，基于传输协议和网络IO来实现，其中传输协议比较出名的有http、tcp、udp等等，http、tcp、udp都是在为某类应用场景而定义出的传输协议，网络IO，主要有bio、nio、aio三种方式，所有的分布式应用通讯都基于这个原理而实现，只是为了应用的易用，各种语言通常都会提供一些更为贴近应用易用的应用层协议。

## 1.3、应用级协议

远程服务通讯，需要达到的目标是在一台计算机发起请求，另外一台机器在接收到请求后进行相应的处理并将结果返回给请求端，这其中又会有诸如one way request、同步请求、异步请求等等请求方式，按照网络通信原理，需要实现这个需要做的就是将请求转换成流，通过传输协议传输至远端，远端计算机在接收到请求的流后进行处理，处理完毕后将结果转化为流，并通过传输协议返回给调用端。

原理是这样的，但为了应用的方便，业界推出了很多基于此原理之上的应用级的协议，使得大家可以不用去直接操作这么底层的东西，通常应用级的远程通信协议会提供：

（1）、为了避免直接做流操作这么麻烦，提供一种更加易用或贴合语言的标准传输格式；

（2）、网络通信机制的实现，就是替你完成了将传输格式转化为流，通过某种传输协议传输至远端计算机，远端计算机在接收到流后转化为传输格式，并进行存储或以某种方式通知远端计算机。

所以在学习应用级的远程通信协议时，我们可以带着这几个问题进行学习：

1）、传输的标准格式是什么？

2）、怎么样将请求转化为传输的流？

3）、怎么接收和处理流？

4）、传输协议是？

不过应用级的远程通信协议并不会在传输协议上做什么多大的改进，主要是在流操作方面，让应用层生成流和处理流的这个过程更加的贴合所使用的语言或标准，至于传输协议则通常都是可选的，在java领域中知名的有：RMI、XML-RPC、Binary-RPC、SOAP、CORBA、JMS，来具体的看看这些远程通信的应用级协议：

### 1.3.1、RMI

RMI是个典型的为java定制的远程通信协议，我们都知道，在single vm中，我们可以通过直接调用java object instance来实现通信，那么在远程通信时，如果也能按照这种方式当然是最好了，这种远程通信的机制成为RPC（Remote Procedure Call），RMI正是朝着这个目标而诞生的。

基于RMI的一次完整的远程通信过程的原理：

1）、客户端发起请求，请求转交至RMI客户端的stub类；

2）、stub类将请求的接口、方法、参数等信息进行序列化；

3）、基于tcp/ip将序列化后的流传输至服务器端；

4）、服务器端接收到流后转发至相应的skelton类；

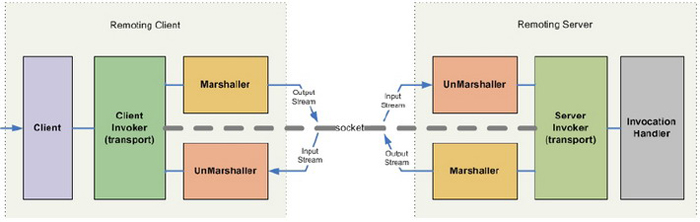
5）、skelton类将请求的信息反序列化后调用实际的处理类；

6）、处理类处理完毕后将结果返回给skelton类；

7）、Skelton类将结果序列化，通过tcp/ip将流传送给客户端的stub；

8）、stub在接收到流后反序列化，将反序列化后的Java Object返回给调用者。

jboss-remoting对于此过程的一个图示：



根据原理来回答下之前应用级协议带着的几个问题：

1）、传输的标准格式是什么？

Java ObjectStream。

2）、怎么样将请求转化为传输的流？

基于Java串行化机制将请求的java object信息转化为流。

3）、怎么接收和处理流？

根据采用的协议启动相应的监听端口，当有流进入后基于Java串行化机制将流进行反序列化，并根据RMI协议获取到相应的处理对象信息，进行调用并处理，处理完毕后的结果同样基于java串行化机制进行返回。

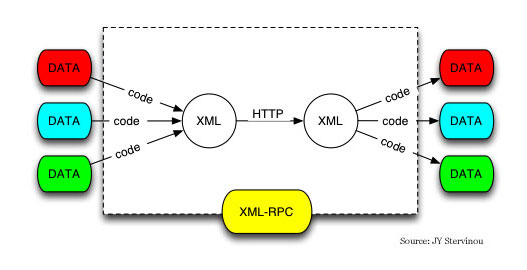
4）、传输协议是？

tcp/ip。

### 1.3.2、XML-RPC

XML-RPC也是一种和RMI类似的远程调用的协议，它和RMI的不同之处在于它以标准的xml格式来定义请求的信息(请求的对象、方法、参数等)，这样的好处是什么呢，就是在跨语言通讯的时候也可以使用。

图示以上过程：



XML-RPC协议的一次远程通信过程：

1）、客户端发起请求，按照XML-RPC协议将请求信息进行填充；

2）、填充完毕后将xml转化为流，通过传输协议进行传输；

3）、接收到在接收到流后转换为xml，按照XML-RPC协议获取请求的信息并进行处理；

4）、处理完毕后将结果按照XML-RPC协议写入xml中并返回。

同样来回答问题：

1）、传输的标准格式是？

标准格式的XML。

2）、怎么样将请求转化为传输的流？

将XML转化为流。

3）、怎么接收和处理流？

通过监听的端口获取到请求的流，转化为XML，并根据协议获取请求的信息，进行处理并将结果写入XML中返回。

4）、传输协议是？

Http。

### 1.3.3、Binary-RPC

Binary-RPC看名字就知道和XML-RPC是差不多的了，不同之处仅在于传输的标准格式由XML转为了二进制的格式。

根据原理来回答下之前应用级协议带着的几个问题：

1）、传输的标准格式是？

标准格式的二进制文件。

2）、怎么样将请求转化为传输的流？

将二进制格式文件转化为流。

3）、怎么接收和处理流？

通过监听的端口获取到请求的流，转化为二进制文件，根据协议获取请求的信息，进行处理并将结果写入XML中返回。

4）、传输协议是？

Http

### 1.3.4、SOAP

SOAP原意为Simple Object Access Protocol，是一个用于分布式环境的、轻量级的、基于XML进行信息交换的通信协议，可以认为SOAP是XML RPC的高级版，两者的原理完全相同，都是http+XML，不同的仅在于两者定义的XML规范不同，SOAP也是Webservice采用的服务调用协议标准，因此在此就不多加阐述了。

### 1.3.5、CORBA

Common Object Request Broker Architecture（公用对象请求代理[调度]程序体系结构），是一组用来定义“分布式对象系统”的标准，由OMG(Object Menagement Group)作为发起和标准制定单位。CORBA的目的是定义一套协议，符合这个协议的对象可以互相交互，不论它们是用什么样的语言写的，不论它们运行于什么样的机器和操作系统。

### 1.3.6、JMS

JMS呢，是实现java领域远程通信的一种手段和方法，基于JMS实现远程通信时和RPC是不同的，虽然可以做到RPC的效果，但因为不是从协议级别定义的，因此我们不认为JMS是个RPC协议，但它确实是个远程通信协议，在其他的语言体系中也存在着类似JMS的东西，可以统一的将这类机制称为消息机制，而消息机制呢，通常是高并发、分布式领域推荐的一种通信机制，这里的主要一个问题是容错（详细见ErLang论文）。

JMS中的一次远程通信的过程：

1）、客户端将请求转化为符合JMS规定的Message；

2）、通过JMS API将Message放入JMS Queue或Topic中；

3）、如为JMS Queue，则发送中相应的目标Queue中，如为Topic，则发送给订阅了此Topic的JMS Queue。

4）、处理端则通过轮训JMS Queue，来获取消息，接收到消息后根据JMS协议来解析Message并处理。

根据原理来回答下之前应用级协议带着的几个问题：

1）、传输的标准格式是？

JMS规定的Message

2）、怎么样将请求转化为传输的流？

将参数信息放入Message中即可

3）、怎么接收和处理流？

轮训JMS Queue来接收Message，接收到后进行处理，处理完毕后仍然是以Message的方式放入Queue中发送或Multicast。

4）、传输协议是？

不限。

基于JMS也是常用的实现远程异步调用的方法之一。在上面的原理中并没有介绍到所有的java领域可选的远程通信协议了，例如还有EJB采用的ORMI、Spring自己定义的一个简单的Http Invoker等等。

# 二、实现远程方法调用

看完原理后我们再来看看目前java领域可用于实现远程通讯的框架或library，知名的有：JBoss-Remoting、Spring-Remoting、Hessian、Burlap、XFire(Axis)、ActiveMQ、Mina、Mule、EJB3等等，来对每种做个简单的介绍和评价，其实呢，要做分布式服务框架，这些东西都是要有非常深刻的了解的，因为分布式服务框架其实是包含了解决分布式领域以及应用层面领域两方面问题的。

当然，你也可以自己根据远程网络通信原理(transport protocol+Net IO)去实现自己的通讯框架或library。

那么在了解这些远程通讯的框架或library时，带着什么问题去学习呢？

1）、基于什么协议实现的？

2）、起请求？

3）、么将请求转化为符合协议的格式的？

4）、使用什么传输协议传输？

5）、响应端基于什么机制来接收请求？

6）、怎么将流还原为传输格式的？

7）、处理完毕后怎么回应？

## 2.1、JBoss-Remoting

Jboss-remoting是由jboss编写的一个java领域的远程通讯框架，基于此框架，可以很简单的实现基于多种传输协议的java对象的RPC。

直接来回答问题：

1）、是基于什么协议实现的？

JBoss-Remoting是个通讯框架，因此它支持多种协议方式的通信，例如tcp/ip+io方式、rmi方式、http+io方式等。

2）、怎么发起请求？

在JBoss-Remoting中，只需将需要发起的请求参数对象传入jboss-remoting的InvocationRequest对象即可，也可根据协议基于InvocationRequest封装符合需求的InvocationRequest对象。

3）、怎么将请求转化为符合协议的格式的？

JBoss-Remoting基于Java串行化机制或JBoss自己的串行化实现来将请求转化为对象字节流。

4）、使用什么传输协议传输？

支持多种传输协议，例如tcp/ip、http等。

5）、响应端基于什么机制来接收请求？

响应端只需将自己的处理对象注册到JBoss-Remoting提供的server端的Connector对象中即可。

6）、怎么将流还原为传输格式的？

JBoss-Remoting基于java串行化机制或jboss自己的串行化实现来将请求信息还原为java对象。

7）、处理完毕后怎么回应？

处理完毕后将结果对象直接返回即可，jboss-remoting会将此对象按照协议进行序列化，返回至调用端。

另外，jboss-remoting支持多种通信方式，例如同步/异步/单向通信等。

## 2.2、Spring-Remoting

Spring-remoting是Spring提供java领域的远程通讯框架，基于此框架，同样也可以很简单的将普通的spring bean以某种远程协议的方式来发布，同样也可以配置spring bean为远程调用的bean。

1）、是基于什么协议实现的？

和JBoss-Remoting一样，作为一个远程通讯的框架，Spring通过集成多种远程通讯的library，从而实现了对多种协议的支持，例如rmi、http+io、xml-rpc、binary-rpc等。

2）、怎么发起请求？

在Spring中，由于其对于远程调用的bean采用的是proxy实现，发起请求完全是通过服务接口调用的方式。

3）、怎么将请求转化为符合协议的格式的？

Spring按照协议方式将请求的对象信息转化为流，例如Spring Http Invoker是基于Spring自己定义的一个协议来实现的，传输协议上采用的为http，请求信息是基于java串行化机制转化为流进行传输。

4）、使用什么传输协议传输？

支持多种传输协议，例如rmi、http等等。

5）、响应端基于什么机制来接收请求？

响应端遵循协议方式来接收请求，对于使用者而言，则只需通过spring的配置方式将普通的spring bean配置为响应端或者说提供服务端。

6）、怎么将流还原为传输格式的？

按照协议方式来进行还原。

7）、处理完毕后怎么回应？

处理完毕后直接返回即可，spring-remoting将根据协议方式来做相应的序列化。

## 2.3、Hessian

Hessian是由caucho提供的一个基于binary-RPC实现的远程通讯library。

1）、是基于什么协议实现的？

基于Binary-RPC协议实现。

2）、怎么发起请求？

需通过Hessian本身提供的API来发起请求。

3）、怎么将请求转化为符合协议的格式的？

Hessian通过其自定义的串行化机制将请求信息进行序列化，产生二进制流。

4）、使用什么传输协议传输？

Hessian基于Http协议进行传输。

5）、响应端基于什么机制来接收请求？

响应端根据Hessian提供的API来接收请求。

6）、怎么将流还原为传输格式的？

Hessian根据其私有的串行化机制来将请求信息进行反序列化，传递给使用者时已是相应的请求信息对象了。

7）、处理完毕后怎么回应？

处理完毕后直接返回，hessian将结果对象进行序列化，传输至调用端。

## 2.4、Burlap

Burlap也是有caucho提供，它和hessian的不同在于，它是基于XML-RPC协议的。

1）、是基于什么协议实现的？

基于XML-RPC协议实现。

2）、怎么发起请求？

根据Burlap提供的API。

3）、怎么将请求转化为符合协议的格式的？

将请求信息转化为符合协议的XML格式，转化为流进行传输。

4）、使用什么传输协议传输？

Http协议。

5）、响应端基于什么机制来接收请求？

监听Http请求。

6）、怎么将流还原为传输格式的？

根据XML-RPC协议进行还原。

7）、处理完毕后怎么回应？

返回结果写入XML中，由Burlap返回至调用端。

## 2.5、XFire、Axis

XFire、Axis是Webservice的实现框架，WebService可算是一个完整的SOA架构实现标准了，因此采用XFire、Axis这些也就意味着是采用webservice方式了。

1）、是基于什么协议实现的？

基于SOAP协议。

2）、怎么发起请求？

获取到远端service的proxy后直接调用。

3）、怎么将请求转化为符合协议的格式的？

将请求信息转化为遵循SOAP协议的XML格式，由框架转化为流进行传输。

4）、使用什么传输协议传输？

Http协议。

5）、响应端基于什么机制来接收请求？

监听Http请求。

6）、怎么将流还原为传输格式的？

根据SOAP协议进行还原。

7）、处理完毕后怎么回应？

返回结果写入XML中，由框架返回至调用端。

## 2.6、ActiveMQ

ActiveMQ是JMS的实现，基于JMS这类消息机制实现远程通讯是一种不错的选择，毕竟消息机制本身的功能使得基于它可以很容易的去实现同步/异步/单向调用等，而且消息机制从容错角度上来说也是个不错的选择，这是Erlang能够做到容错的重要基础。

1）、是基于什么协议实现的？

基于JMS协议。

2）、怎么发起请求？

遵循JMS API发起请求。

3）、怎么将请求转化为符合协议的格式的？

不太清楚，猜想应该是二进制流。

4）、使用什么传输协议传输？

支持多种传输协议，例如tcp/ip、udp、http等等。

5）、响应端基于什么机制来接收请求？

监听符合协议的端口。

6）、怎么将流还原为传输格式的？

同问题3。

7）、处理完毕后怎么回应？

遵循JMS API生成消息，并写入JMS Queue中。

基于JMS此类机制实现远程通讯的例子有Spring-Intergration、Mule、Lingo等等。

## 2.7、Mina

Mina是Apache提供的通讯框架，在之前一直没有提到网络IO这块，之前提及的框架或library基本都是基于BIO的，而Mina是采用NIO的，NIO在并发量增长时对比BIO而言会有明显的性能提升，而java性能的提升，与其NIO这块与OS的紧密结合是有不小的关系的。

1）、是基于什么协议实现的？

可选的传输协议+NIO。

2）、怎么发起请求？

通过Mina提供的Client API。

3）、怎么将请求转化为符合协议的格式的？

Mina遵循java串行化机制对请求对象进行序列化。

4）、使用什么传输协议传输？

支持多种传输协议，例如tcp/ip、http等等。

5）、响应端基于什么机制来接收请求？

以NIO的方式监听协议端口。

6）、怎么将流还原为传输格式的？

遵循java串行化机制对请求对象进行反序列化。

7）、处理完毕后怎么回应？

遵循Mina API进行返回。

MINA是NIO方式的，因此支持异步调用是毫无悬念的。

## 2.8、EJB

EJB最突出的在于其分布式，EJB采用的是ORMI协议，和RMI协议是差不多的，但EJB在分布式通讯的安全控制、transport pool、smart proxy等方面的突出使得其在分布式领域是不可忽视的力量。

1）、是基于什么协议实现的？

基于ORMI协议。

2）、怎么发起请求？

EJB调用。

3）、怎么将请求转化为符合协议的格式的？

遵循java串行化机制对请求对象进行序列化。

4）、使用什么传输协议传输？

tcp/ip。

5）、响应端基于什么机制来接收请求？

监听协议端口。

6）、怎么将流还原为传输格式的？

遵循java串行化机制对请求对象进行反序列化。

7）、处理完毕后怎么回应？

直接返回处理对象即可。

## 2.9、jboss的jndi实现

在将对象实例绑定到jboss jnp server后，当远程端采用context.lookup()方式获取远程对象实例并开始调用时，jboss jndi的实现方法是从jnp server上获取对象实例，将其序列化回本地，然后在本地进行反序列化，之后在本地进行类调用。

通过这个机制，就可以知道了，本地其实是必须有绑定到jboss上的对象实例的class的，否则反序列化的时候肯定就失败了，而远程通讯需要做到的是在远程执行某动作，并获取到相应的结果，可见纯粹基于JNDI是无法实现远程通讯的。

但JNDI也是实现分布式服务框架一个很关键的技术点，因为可以通过它来实现透明化的远端和本地调用，就像ejb，另外它也是个很好的隐藏实际部署机制(就像datasource)等的方案。

由上一系列的分析可知，在远程通讯领域中，涉及的知识点还是相当的多的，例如有：通信协议或远程调用协议(tcp/http/udp/rmi/xml-rpc etc.)、消息机制、网络IO（BIO/NIO/AIO）、MultiThread、本地调用与远程调用的透明化方案（涉及java classloader、Dynamic Proxy、Unit Test etc.）、异步与同步调用、网络通信处理机制（自动重连、广播、异常、池处理等等）、Java Serialization (各种协议的私有序列化机制等)、各种框架的实现原理（传输格式、如何将传输格式转化为流的、如何将请求信息转化为传输格式的、如何接收流的、如何将流还原为传输格式的等等），要精通其中的哪些东西，得根据实际需求来决定了，只有在了解了原理的情况下才能很容易的做出选择，甚至可以根据需求做私有的远程通讯协议，对于从事分布式服务平台或开发较大型的分布式应用的人而言，我觉得至少上面提及的知识点是需要比较了解的。

# 三、Java性能分析：远程调用方法（未验证）

## 3.1、测试项目

①RMI:用Spring3集成发布。

②hessian：用Spring3集成发布到Tomcat容器。

③Java6 WebService：使用Java6原生WebService和注解，直接用Endpoint.publish发布服务。

④CXF：用Spring3集成发布到Tomcat容器。

## 3.2、测试结果：



说明：以上测试虽不是非常的精确，但基本能说明一定的问题。每项案例的服务端方法，都是简单方法，接收客户端传递的一个String类型参数，并打印到console。每轮测试取三次时间的平均值。所有单线程访问测试全部完成并正常处理请求，没有请求拒绝情况发生。而并发访问测试，除hessian中途抛异常无法完成，其余均正常完成测试。

## 3.3、测试结论

RMI的性能最高，这已是公认，RMI底层基于Java远程方法协议(JRMP)和对象序列化技术，而JRMP是直接基于TCP/IP协议的封装，在网络上传输2 byte的有效数据，对于TCP而言，总共有478 byte被额外传输，而对于RMI, 1645byte被额外传输。可见RMI的效率还是相当不错的。JavaEE标准的EJB就是基于RMI的调用。

　　hessian是一个轻量级的remoting on http框架。Hessian没有采用WebService标准的SOAP协议，而是自己实现了一种二进制RPC(Remote Procedure Call Protocol，远程过程调用协议)协议。Hessian的设计理念就是简单高效，使用 Hessian 传输数据量比Soap协议要小很多。这也是为什么Hessian的性能要高于WebService。但是，尽管它再简单高效，它始终是基于Http协议上封装的，所以还是比Java的RMI效率要差一些。我看过其他的一些研究测试报告称Hessian在传输少量对象时，比RMI还要快速高效，但传输数据结构复杂的对象或大量数据对象时，较RMI要慢20%左右。这个结论在我的测试中还真没有发现，也许与测试环境或方法有关系吧。

　　Java6WebService 和 CXF的性能应该说是基本同级别，前者略高于后者。众所周知WebService是基于Soap协议实现的，而Soap协议是在Http协议基础上的XML定义和封装。所有的请求和响应都要被转化成符合SOAP标准的XML格式。显然这直接会导致效率的降低。

　　XML格式的协议是一种易读易理解的协议，但并不高效。解析和组装XML协议数据流都需要耗费系统的处理时间，所以，WebService的性能不如Hessian。这里要说一下的是Java6原生的WebService，开发起来非常方便，而且无需额外引入一大堆的Jar包。性能又强于CXF，至于Axis2和Axis1就更不用说，已经有很多测试表明CXF的性能是Axis2的2倍以上，是Axis1的2-6倍。

　　那么既然RMI性能那么好，为什么我们需要那么多其他的远程调用方式呢?这个问题又引发到了一个原始的真理。越原始越底层的技术效率就越高，但局限性也就越大。RMI是Java的特性，那么它必须基于JVM运行，也就是说RMI无法跨语言运行。而WebService就不同了，Soap是基于标准Http协议的，是一种与语言无关的字符级协议，所以它可以更好的实现异构系统的通信。这在我们现实环境中是非常有用的，相信大家还是WebService用的比较多点吧。

　　不足：这次的测试，还是存在很多不足之处。

Hessian的并发测试时，跑到中途大约300-400次调用时就抛异常中断了，具体原因还没来得及仔细分析。不知道是否与Spring集成或测试方法有关系。

# 四、java几种远程服务调用协议的比较

## 4.1、综述

本文比较了RMI，Hessian，Burlap，Httpinvoker，web service等5种通讯协议的在不同的数据结构和不同数据量时的传输性能。

RMI是java语言本身提供的远程通讯协议，稳定高效，是EJB的基础。但它只能用于JAVA程序之间的通讯。

Hessian和Burlap是caucho公司提供的开源协议，基于HTTP传输，服务端不用开防火墙端口。协议的规范公开，可以用于任意语言。

Httpinvoker是SpringFramework提供的远程通讯协议，只能用于JAVA程序间的通讯，且服务端和客户端必须使用SpringFramework。

Web service是连接异构系统或异构语言的首选协议，它使用SOAP形式通讯，可以用于任何语言，目前的许多开发工具对其的支持也很好。

## 4.2、测试结果显示

几种协议的通讯效率依次为：

RMI > Httpinvoker >= Hessian >> Burlap >> web service

RMI不愧是JAVA的首选远程调用协议，非常高效稳定，特别是在大数据量的情况下，与其他通讯协议的差距尤为明显。

HttpInvoker使用java的序列化技术传输对象，与RMI在本质上是一致的。从效率上看，两者也相差无几，HttpInvoker与RMI的传输时间基本持平。

Hessian在传输少量对象时，比RMI还要快速高效，但传输数据结构复杂的对象或大量数据对象时，较RMI要慢20%左右。

Burlap仅在传输1条数据时速度尚可，通常情况下，它的毫时是RMI的3倍。

Web Service的效率低下是众所周知的，平均来看，Web Service的通讯毫时是RMI的10倍。

## 4.3、结果分析

**1）、直接调用**

直接调用的所有毫时都接近0，这说明程序处理几乎没有花费时间，记录的全部时间都是远程调用耗费的。

**2）、RMI调用**

与设想的一样，RMI理所当然是最快的，在几乎所有的情况下，它的毫时都是最少的。特别是在数据结构复杂，数据量大的情况下，与其他协议的差距尤为明显。

为了充分发挥RMI的性能，另外做了测试类，不使用Spring，用原始的RMI形式（继承UnicastRemoteObject对象）提供服务并远程调用，与Spring对POJO包装成的RMI进行效率比较。结果显示：两者基本持平，Spring提供的服务还稍快些。

初步认为，这是因为Spring的代理和缓存机制比较强大，节省了对象重新获取的时间。

**3）、Hessian调用**

caucho公司的resin服务器号称是最快的服务器，在java领域有一定的知名度。Hessian做为resin的组成部分，其设计也非常精简高效，实际运行情况也证明了这一点。平均来看，Hessian较RMI要慢20%左右，但这只是在数据量特别大，数据结构很复杂的情况下才能体现出来，中等或少量数据时，Hessian并不比RMI慢。

Hessian的好处是精简高效，可以跨语言使用，而且协议规范公开，我们可以针对任意语言开发对其协议的实现。目前已有实现的语言有：java, c++, .net, python, ruby。还没有delphi的实现。

另外，Hessian与WEB服务器结合非常好，借助WEB服务器的成熟功能，在处理大量用户并发访问时会有很大优势，在资源分配，线程排队，异常处理等方面都可以由成熟的WEB服务器保证。而RMI本身并不提供多线程的服务器。而且，RMI需要开防火墙端口，Hessian不用。

**4）、Burlap调用**

Burlap与Hessian都是caucho公司的开源产品，只不过Hessian采用二进制的方式，而Burlap采用xml的格式。

测试结果显示，Burlap在数据结构不复杂，数据量中等的情况下，效率还是可以接受的，但如果数据量大，效率会急剧下降。平均计算，Burlap的调用毫时是RMI的3倍。

我认为，其效率低有两方面的原因，一个是XML数据描述内容太多，同样的数据结构，其传输量要大很多；另一方面，众所周知，对xml的解析是比较费资源的，特别对于大数据量情况下更是如此。

**5）、HttpInvoker调用**

HttpInvoker是SpringFramework提供的JAVA远程调用方法，使用java的序列化机制处理对象的传输。从测试结果看，其效率还是可以的，与RMI基本持平。

不过，它只能用于JAVA语言之间的通讯，而且，要求客户端和服务端都使用SPRING框架。

另外，HttpInvoker 并没有经过实践的检验，目前还没有找到应用该协议的项目。

**6）、****web service调用**

本次测试选用了apache的AXIS组件作为WEB SERVICE的实现，AXIS在WEB SERVICE领域相对成熟老牌。

为了仅测试数据传输和编码、解码的时间，客户端和服务端都使用了缓存，对象只需实例化一次。但是，测试结果显示，web service的效率还是要比其他通讯协议慢10倍。

如果考虑到多个引用指向同一对象的传输情况，web service要落后更多。因为RMI，Hessian等协议都可以传递引用，而web service有多少个引用，就要复制多少份对象实体。

Web service传输的冗余信息过多是其速度慢的原因之一，监控发现，同样的访问请求，描述相同的数据，web service返回的数据量是hessian协议的6.5倍。另外，WEB SERVICE的处理也很耗时，目前的xml解析器效率普遍不高，处理xml <-> bean很毫资源。从测试结果看，异地调用比本地调用要快，也从侧面说明了其毫时主要用在编码和解码xml文件上。这比冗余信息更为严重，冗余信息占用的只是网络带宽，而每次调用的资源耗费直接影响到服务器的负载能力。（MS的工程师曾说过，用WEB SERVICE不能负载100个以上的并发用户。）

测试过程中还发现，web service编码不甚方便，对非基本类型需要逐个注册序列化和反序列化类，很麻烦，生成stub更累，不如spring + RMI/hessian处理那么流畅简洁。而且，web service不支持集合类型，只能用数组，不方便。

# 参考文档（感谢这些文章）

RMI原理及实现：http://www.yesky.com/274/1625274.shtml

Java NIO原理和使用：

http://www.jdon.com/concurrent/nio%D4%AD%C0%ED%D3%A6%D3%C3.htm

XML RPC协议：http://hedong.3322.org/archives/000470.html

http://www.mengyan.org/blog/archives/2005/07/12/30.html

Spring技术应用中的远程服务详解：http://www.builder.com.cn/2007/1027/583384.shtml

JAVA RPC通信机制之SOAP：http://www.java114.com/content16/content3826.html

Java Remoting：Protocol BenchMarks：http://q.sohu.com/forum/5/topic/1148909

Evalution of RMI Alternative：

https://www.jfire.org/modules/phpwiki/index.php/Evaluation%20of%20RMI%20Alternative

Metaprotocol Taxonomy：http://hessian.caucho.com/doc/metaprotocol-taxonomy.xtp

什么是Webservice：http://www.vchome.net/dotnet/webservice/webservice15.htm