山东科技大学

**计算机学院学院**

**毕业设计**

你这篇论文是抄的吧。这个怎么能过完成毕设？所有格式要按照我发的官方文件格式。

所有代码能讲的清楚吗？论文需要查重，目前论文不及格。

**课题javaweb项目间对接方式探究**

**专业计算机科学与技术**

**学历层次本科**

**学生姓名焦晓英**

**学生学号12345678**

**指导教师XXX**

**接受任务日期：2020年 01月10日**

**完成设计（论文）日期：2020年05月28日**

javaweb项目间对接方式探究

**摘要：**随着计算机网络技术的普及，分布式应用迅速发展，迫切要求网络中不同的主机相互协作，互相访问对方的服务和数据，远程调用机制逐步成为解决这种问题的主要手段。本论文主要对比Spring实现远程服务调用的实现方式和不足之处，将服务的接口与实现方法分离，使用“通信代理”来实现对平台差异的封装，根据多个服务进程能提供相同服务的实际情况，实现了客户端和服务进程的动态绑定。

**关键词：**Spring；远程调用机制；通信代理； 异步；

**目录**

[javaweb项目间对接方式探究 1](#_Toc513566284)

[一、javaweb远程接口调用概述 3](#_Toc513566285)

[（一）javaweb远程接口调用的基本原理 3](#_Toc513566286)

[（二）javaweb远程接口调用的现状 4](#_Toc513566287)

[（三）javaweb远程接口调用实现的必要技术 4](#_Toc513566288)

[（四）javaweb几种远程接口调用协议的简单比较 5](#_Toc513566289)

[（五）总结 6](#_Toc513566290)

[二、javaweb远程接口调用在Spring中的集成 7](#_Toc513566291)

[（一）Spring远程调用概览 7](#_Toc513566292)

[（二）Spring远程调用支持模式的比较 7](#_Toc513566293)

[（三）使用Spring远程调用工具类的优点 8](#_Toc513566294)

[三、Spring远程调用应用的实例分析 8](#_Toc513566295)

[（一）RMI远程调用应用的实例分析 10](#_Toc513566296)

[（1）服务端应用实例： 10](#_Toc513566297)

[（2）客户端应用实例： 10](#_Toc513566298)

[（二）httpinvoker远程调用应用的实例分析 11](#_Toc513566299)

[（1）服务端应用实例： 11](#_Toc513566300)

[（2）客户端应用实例： 13](#_Toc513566301)

[（三）webservice（cxf-xml）远程调用的应用实例分析 14](#_Toc513566302)

[（1）服务端应用实例： 14](#_Toc513566303)

[（2）客户端应用实例： 16](#_Toc513566304)

[（四）小结 16](#_Toc513566305)

[结论 17](#_Toc513566306)

[致 谢 18](#_Toc513566307)

[参考文献 19](#_Toc513566308)

# 一、javaweb远程接口调用概述

## （一）javaweb远程接口调用的基本原理

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，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。

javaweb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，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。

## （二）javaweb远程接口调用的现状

远程调用的原理是这样的，但为了应用的方便，业界推出了很对基于此原理之上的应用级的协议，使得大家可以不用直接去操作这么底层的东西，通常应用级的通信协议会提供：为了避免直接作流操作这么麻烦，提供一种更加易用或贴合语言的标准传输格式；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。

不过应用级的远程通讯协议并不会在传输协议上做太大的改进，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、Burlap、dubbo、hessian、Httpinvoker、webservice等。

## （三）javaweb远程接口调用实现的必要技术

远程调用实现的技术总体上分为以下几个方面：

（1）通信技术：远程调用就是通过网络技术将不同系统构成一个整体，因此通信技术是其重点。

（2）序列化和反序列化技术：java的序列化技术是指将对象转换为byte数据，这些数据可以被还原为java对象，这种还原的过程就是反序列化了，该机制可以自动处理不同操作系统之间的差异，例如window下序列化的对象，可以在linux上进行重新构建。Java的jdk里自带了一个序列化和反序列化机制，熟悉hadoop的人知道hadoop设计了一套序列化和反序列化机制，为什么hadoop作者不选择使用java自带的序列化机制，这是因为java序列化机制非常复杂，复杂带来效率低下，java的序列化机制还有一个重要的缺点就是它序列化的二进制数据会非常大，因为java序列化时候会附带太多该对象的相关信息，过大的数据量就会影响网络传输的效率。

（3）压缩技术：做网络编程，最稀缺的资源就是宽带资源，如果传输数据过大，那么对数据的压缩就会显得十分重要，这里我推荐一个压缩技术snappy，它是一种高效的压缩和解压缩包，google公司内部广泛使用的一种压缩技术。

（4）高并发的技术：远程调用技术一定会是多线程，只有这样才能满足多个并发的处理请求，java在1.5的版本里提供了一个Executor框架，它在线程开发里引入了任务的概念，使得多线程的程序开发会更加合理和可控。

（5）非侵入式：这个也可以叫做松耦合，对于java的web开发，最好的解耦方式就是使用spring技术，当我们系统里把远程调用框架引入后，配置好相关的参数，我们可以把用于远程调用的方法定义在spring的配置文件里，那么在程序里调用的时候，利用spring直接获取这个bean，那么对于远程调用的开发就和我们在action里调用server的方法没啥区别了。

（6）负载均衡：分布式系统都离不开负载均衡，好的负载均衡可以充分利用好不同服务器的计算资源，提供系统的并发量和运算能力，对于网站而言少于10台服务器可以使用多种策略：一种是简单轮询；第二种是随机方式，采用随机函数；第三种可以采用HASH技术；第四种可以采用圆周计算算法；等等。

（7）最后提到的是远程调用服务管理组件。

## （四）javaweb几种远程接口调用协议的简单比较

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **协议** | dubbo | rmi | hessian | http | webservice |
| **连接个数** | 单连接 | 多连接 | 多连接 | 多连接 | 多连接 |
| **连接方式** | 长连接 | 短连接 | 短连接 | 短连接 | 短连接 |
| **传输协议** | TCP | TCP | HTTP | HTTP | HTTP |
| **传输方式** | NIO异步传输 | 同步传输 | 同步传输 | 同步传输 | 同步传输 |
| **序列化** | Hessian二进制序列化 | Java标准二进制序列化 | 表单序列化 | 表单序列化 | SOAP文本序列化 |
| **适用范围** | 传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用dubbo协议传输大文件或超大字符串。 | 传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件。 | 传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或URL传入参数，暂不支持传文件。 | 传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或URL传入参数，暂不支持传文件。 |  |
|
| **适用场景** | 常规远程服务方法调用 | 常规远程服务方法调用，与原生RMI服务互操作 | 需同时给应用程序和浏览器JS使用的服务。 | 需同时给应用程序和浏览器JS使用的服务。 | 系统集成，跨语言调用 |

## （五）总结

远程调用是客户端应用和服务端之间的会话，在客户端上所需要的一些功能并不包括在该应用的职能范围内。所以应用向能提供这些功能的其他系统寻求帮助。远程的应用通过远程服务把这些功能公开出来。

# 二、javaweb远程接口调用在Spring中的集成

## （一）Spring远程调用概览

Spring为各种远程访问技术的集成提供了工具类。Spring远程支持是由普通（Spring）POJO实现的，这使得开发具有远程访问功能的服务变得相当容易。

Spring远程调用支持6种不同的RPC模式：远程方法调用（RMI）、Caucho的Hessian和Burlap、Spring自己的HTTP invoker、EJB和使用JAX-RPC 的Web Services（参照图2-1服务端的远程调用实现）。

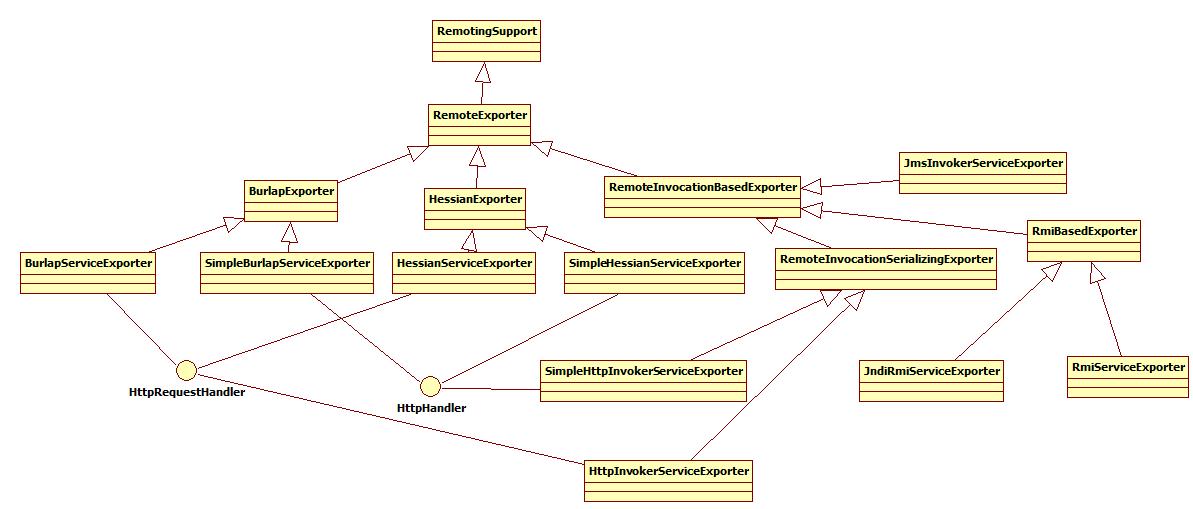


图2-1 名称

## （二）Spring远程调用支持模式的比较

远程方法调用（RMI）：在何种情况下有用，但是Docker不支持。相对应的支持类是RmiProxyFactoryBean和 RmiServiceExporter，Spring同时支持传统的RMI（使用java.rmi.Remote接口和java.rmi.RemoteException）和通过RMI调用器实现的透明远程调用（支持任何Java接口）。

Spring自己的HTTP invoker：考虑网络限制时，通过HTTP访问/公开基于Java的服务。Spring提供了一种特殊的允许通过HTTP进行Java串行化的远程调用策略，支持任意Java接口（就像RMI调用器）。相对应的支持类是 HttpInvokerProxyFactoryBean和 HttpInvokerServiceExporter。

Hessian：不考虑网络限制（如防火墙）时，访问/公开基于Java的服务。通过 HessianProxyFactoryBean和 HessianServiceExporter，可以使用Caucho提供的基于HTTP的轻量级二进制协议来透明地暴露服务。

Burlap：不考虑网络限制（如防火墙）时，访问/公开基于Java的服务。Burlap是Caucho的另外一个子项目，可以作为Hessian基于XML的替代方案。Spring提供了诸如 BurlapProxyFactoryBean和 BurlapServiceExporter的支持类。

JAX RPC：考虑网络限制（如防火墙）时，Spring通过JAX-RPC为远程Web服务提供支持。主要是指webservice的使用，常见的WebService实现有axis、cxf等。

## （三）使用Spring远程调用工具类的优点

（1）易于切换模式：不管选择哪种远程模式，你会发现Spring对每一种模式的支持中贯穿着一个共同的风格。这就意味着你一旦理解了Spring如何配置并使用其中的一种模式，当你决定使用另一种不同的模式的时候，你将非常容易切换到另一种模式。

（2）像本地调用一样易于操作：在所有的模式中，服务可以作为Spring管理的Bean配置到你的应用中。这是用一个代理工厂Bean实现的，这个Bean使你能把远程服务当作本地对象一样置入到其他Bean的属性中。

（3）不用写任何Java代码来支持远程调用：客户端发起对代理的调用，好像是代理提供了这些服务的功能一样。代理代表客户端和远程服务交流。它处理连接的具体情况，并向远程服务发起远程调用。在服务端，你能够把任何Spring管理的Bean的功能公开成为一个远程服务，可使用上述所列的任何模式（除了EJB和JAX-RPC）。不论开发的是使用远程服务的代码，还是实现那些服务的代码，或者二者兼而有之，在Spring中，使用远程服务纯粹是个配置问题。你不用写任何Java代码来支持远程调用。你的服务Bean不必关心它们是否被卷入到RPC里（虽然任何传递给远程调用的Bean或从远程调用返回的Bean可能需要实现java.io.Serializable）。

# 三、Spring远程调用应用的实例分析

在本章我们通过一个具体的例子来简述这些技术的使用以及在实践中如何权衡各种技术的使用场景。作为示例，这里我们建立一个叫springmvc的javaweb项目作为远程服务的服务端，在这个项目中我们建立一个叫做UserSvc的接口：

1. **package** com.springmvc.server.service;
3. **public** **interface** UserSvc {
5. **public** **void** addUser();
7. **public** **void** updateUser ();
9. **public** **void** delUser();
11. **public** String findUser(String username);
13. }

然后建立一个它的实现类UserSvcImpl：

1. **package** com.springmvc.server.service.impl;
2. **import javax.annotation.Resource;**
3. **import** com.springmvc.server.service.UserSvc;
5. @Service("userSvc")
6. **public** **class** UserSvcImpl **implements** UserSvc {
8. **public** **void** addUser() {
9. System.out.println("-------------invoke addUser()---------------");
10. }
12. **public** **void** updateUser() {
13. System.out.println("-------------invoke updateUser()---------------");
14. }
16. **public** **void** delUser() {
17. System.out.println("-------------invoke delUser()---------------");
18. }
20. **public** String findUser(String username) {
21. System.out.println("-------------invoke findUser---------------");
22. **return** "return: " + username;
23. }
25. }

## （一）RMI远程调用应用的实例分析

### （1）服务端应用实例：

在SpringMVC的配置文件中利用RMI来定义我们的远程服务,通过Spring提供的RmiServiceExporter来实现，我们需要在SpringMVC的配置文件中定义一个类型为RmiServiceExporter的bean对象。该bean对象需要接收四个属性，一是serviceName属性，用于客户端调用；二是service属性，用于关联真正的service对象；三是serviceInterface属性，用于指定当前的服务对应的接口；四是registryPort属性，用于指定当前服务对应的端口。当我们请求某一个远程服务的时候实际上请求的就是其对应的RmiServiceExporter对象，RmiServiceExporter会把请求的服务以二进制的方式返回给客户端。这里我们在SpringMVC的配置文件中这样定义：

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"**>**
6. <!-- 声明走动扫描方式管理Bean -->
7. **<context:component-scan base-package="com.springmvc" />**
8. <!-- 开启切面编程功能 -->
9. **<aop:aspectj-autoproxy />**
10. **<bean** id="serviceExporter" class="org.springframework.remoting.rmi.RmiServiceExporter"**>**
11. **<property** name="serviceName" value="userSvc" **/>**
12. **<property** name="service" ref="userSvc" **/>**
13. **<property** name="serviceInterface"
14. value="org.springmvc.server.service.UserSvc" **/>**
15. **<property** name="registryPort" value="8080" **/>**
16. **</bean>**
18. **</beans>**

### （2）客户端应用实例：

对于客户端要使用远程的RMI服务的，我们需要在Spring配置文件中定义对应的org.springframework.remoting.rmi.RmiProxyFactoryBean对象。RmiProxyFactoryBean对象需要指定两个属性，一个是serviceInterface属性，表示当前请求的远程服务对应的接口；另一个是serviceUrl属性，表示当前的远程服务对应的服务端请求地址。这里在客户端为了使用hessianServer定义的UserService服务，我们建立一个对应的hessianClient项目，在hessianClient中我们定义一个对应的UserService接口，这个接口的内容跟hessianServer中的UserService接口的内容是一样的。代码如下所示：

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"**>**
6. **<bean** id="userSvc" class="org.springframework.remoting.rmi.RmiProxyFactoryBean"**>**
7. **<property** name="serviceUrl" value="rmi://192.168.1.103:8080/userSvc" **/>**
8. **<property** name="serviceInterface"
9. value=" org.springmvc.server.service.UserSvc " **/>**
10. **</bean>**
11. **</beans>**

## （二）httpinvoker远程调用应用的实例分析

### （1）服务端应用实例：

首先我们需要在web.xml中配置一个SpringMVC的DispatcherServlet用于接收所有的Web服务请求，这里我们这样配置：

1. **<servlet>**
2. **<servlet-name>**springmvc**</servlet-name>**
3. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
4. **<load-on-startup>**1**</load-on-startup>**
5. <init-param>
6. <param-name>contextConfigLocation</param-name>
7. <param-value>classpath:mvc-config.xml</param-value>
8. </init-param>
9. **</servlet>**
11. **<servlet-mapping>**
12. **<servlet-name>**springmvc**</servlet-name>**
13. **<url-pattern>** /\***</url-pattern>**
14. **</servlet-mapping>**

可以看到我们这个DispatcherServlet会处理url为“/\*”的请求，通配符“\*”就对应着我们的处理器映射。

接下来就是在SpringMVC的配置文件mvc-config.xml中利用httpinvoker来定义我们的远程服务了，这是通过Spring提供的HttpInvokerServiceExporter来实现。我们需要在SpringMVC的配置文件中定义一个类型为HttpInvokerServiceExporter的bean对象。该bean对象需要接收两个属性，一个是service属性，用于关联真正的service对象；另一个是serviceInterface属性，用于指定当前的服务对应的接口。HttpInvokerServiceExporter实现了HttpRequestHandler接口，当我们请求某一个远程服务的时候实际上请求的就是其对应的HttpInvokerServiceExporter对象，HttpInvokerServiceExporter会把请求的服务以二进制的方式返回给客户端。这里我们在SpringMVC的配置文件中这样定义：

1. **<?xml version="1.0" encoding="UTF-8"?>**
2. **<beans xmlns="http://www.springframework.org/schema/beans"**
3. **xmlns:context="http://www.springframework.org/schema/context"**
4. **xmlns:mvc="http://www.springframework.org/schema/mvc"**
5. **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**
6. **xsi:schemaLocation="**
7. **http://www.springframework.org/schema/beans**
8. **http://www.springframework.org/schema/beans/spring-beans-3.0.xsd**
9. **http://www.springframework.org/schema/context**
10. **http://www.springframework.org/schema/context/spring-context-3.0.xsd**
11. **http://www.springframework.org/schema/mvc**
12. **http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">**
13. **<!--引用注解-->**
14. **<mvc:annotation-driven />**
15. **<mvc:default-servlet-handler />**
17. **<bean** id="serviceExporter" name="/userSvc"
18. class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter"**>**
19. **<property** name="service" ref="userSvc" **/>**
20. **<property** name="serviceInterface" value="org.springmvc.server.service.UserSvc" **/>**
21. **</bean>**
22. **<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">**
23. **<property name="mappings">**
24. **<props>**
25. **<prop key="/userSvc">serviceExporter</prop>**
26. **</props>**
27. **</property>**
28. **</bean>**
29. **</beans>**

上述代码中我们的HttpInvokerServiceExporter对应的bean的name是“/userSvc”，在SpringMVC的配置文件中，当一个bean的name是以“/”开始的时候Spring会自动对它进行BeanNameUrlHandlerMapping，在这里我们使用了SimpleUrlHandlerMapping也是同样的效果。所以这里相当于是我们把“/userSvc”映射到了HttpInvokerServiceExporter，因为是根据beanName来进行映射的，所以我们必须要给HttpInvokerServiceExporter bean对象指定name属性，而且其对应的name必须以“/”开头，这样我们的客户端才能访问到对应的服务。

### （2）客户端应用实例：

对于客户端要使用远程的httpinvoker服务的，我们需要在Spring配置文件中定义对应的HttpInvokerProxyFactoryBean对象,配置的方法和RMI服务的配制方法类似。代码如下所示：

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"**>**
6. **<bean** id="userSvc" class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"**>**
7. **<property** name="serviceUrl" value="rmi://192.168.1.103:8080/userSvc" **/>**
8. **<property** name="serviceInterface"
9. value=" org.springmvc.server.service.UserSvc " **/>**
10. **</bean>**
12. **</beans>**

## （三）webservice（cxf-xml）远程调用的应用实例分析

这里主要论述cxf方式在springmvc框架中的应用。

### （1）服务端应用实例：

上述的UserSvc接口想要作为cxf的服务端发布，首先要在UserSvc接口前添加注解，具体的接口如下：

1. **package** com.springmvc.server.webservice;
2. import javax.jws.WebService;
3. @WebService
4. **public** **interface** UserSvc {
6. **public** **void** addUser();
8. **public** **void** updateUser();
10. **public** **void** delUser();
12. **public** String findUser(String username);
14. }

UserSvc的实现类UserSvcImpl也要做相应的改变，具体的接口如下：

1. **package** com.springmvc.server.webservice.impl;
3. **import javax.annotation.Resource;**
4. **import** com.springmvc.server.webservice.UserSvc ;
5. @Service
6. **public** **class** UserSvcImpl **implements** UserSvc {
8. **public** **void** addUser() {
9. System.out.println("-------------invoke addUser()---------------");
10. }
12. **public** **void** updateUser() {
13. System.out.println("-------------invoke updateUser()---------------");
14. }
16. **public** **void** delUser() {
17. System.out.println("-------------invoke delUser()---------------");
18. }
20. **public** String findUser(String username) {
21. System.out.println("-------------invoke findUser---------------");
22. **return** "return: " + username;
23. }
25. }

首先我们需要在web.xml中配置一个SpringMVC的CXFServlet用于接收所有的WebService服务请求，这里我们这样配置：

1. **<servlet>**
2. **<display-name>** jaxws-cxf**</display-name>**
3. **<servlet-name>** jaxws-cxf**</servlet-name>**
4. **<servlet-class>**org.apache.cxf.transport.servlet.CXFServlet**</servlet-class>**
5. **<load-on-startup>**1**</load-on-startup>**
6. **</servlet>**
8. **<servlet-mapping>**
9. **<servlet-name>** jaxws-cxf**</servlet-name>**
10. **<url-pattern>** /ws/\* **</url-pattern>**
11. **</servlet-mapping>**

接下来需要在spring的配置文件中做如下配置：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8. http://www.springframework.org/schema/context
9. http://www.springframework.org/schema/context/spring-context-3.0.xsd
10. http://cxf.apache.org/jaxws
11. http://cxf.apache.org/schemas/jaxws.xsd ">
13. <import resource="classpath\*:META-INF/cxf/cxf.xml" />
14. <import resource="classpath\*:META-INF/cxf/cxf-extension-soap.xml" />
15. <import resource="classpath\*:META-INF/cxf/cxf-servlet.xml" />
16. <jaxws:server address="/userSvc" serviceBean="# userSvcImpl" />
18. </beans>

至此，要想支持代码的正常运行还需要导入一系列的jar包，启动服务端，可以在浏览器中输入“http://localhost:8080/springmvc/ws/userSvc?wsdl”,若访问成功，则说明webservice服务端已配置完毕。

### （2）客户端应用实例：

客户端要想正常调用上述的webservice，调用双方的防火墙必须是贯通的，另外需要通过cxf工具生成可服务的类，可以直接在eclipse中生成，也可以通过下载apache-cxf-2.3.3.zip包，解压zip包，切换到\*\\*\apache-cxf-2.3.3\bin目录下，执行命令wsdl2java http://localhost:8080/springmvc/ws/userSvc?wsdl，在bin目录下找到生成的可服务的类，放到客户端中去，接下来就可以在客户端中正常调用

1. **public final static QName SERVICE = new QName("http://impl.service.webservice.cpic.com/", "UserSvcImplService");**
3. **String wsdlURL = "http://localhost:8080/springmvc/ws/userSvc?wsdl ";**
4. **URL url = null;**
5. **try {**
6. **url = new URL(wsdlURL");**
7. **} catch (MalformedURLException e) {**
8. **System.err.println("Can not initialize the default wsdl from http://localhost:8080/springmvc /ws/userSvc?wsdl ");**
9. **}**
10. **UserSvcImplService userSvcImplService = new UserSvcImplService(url,SERVICE\_NAME);**
11. **UserSvc userSvc = userSvcImplService.getUserSvcImplPort();**
12. **userSvc.** addUser();

## （四）小结

通过上述的应用实例，我们可以看到Spring提供了远程服务的支持，让使用远程服务和使用常规的JavaBean一样简单。在客户端，Spring提供了代理工厂Bean，能让你在Spring应用中配置远程服务。不管是使用RMI、Hessian、Burlap、HTTP invoker、EJB、还是Web service，你都可以把远程服务置入到你的应用里，好像它们是POJO一样。Spring甚至捕获了所有抛出的RemoteException，并在发生异常的地方重新抛出运行时RemoteAccessException，让你的代码从处理可能不可恢复的异常中解放出来。

# 结论

总结下，对企业来讲，Java远程调用采取何种方案没有一个特定的标准。根据我的经验，业务特点以及数据吞吐量决定了技术的选择方向。比如第三方数据接口，重点考虑的是跨平台、跨语言、支持高并发、保证安全；而局域网内的分布式服务，重点考虑的是高性能、稳定性、可伸缩性。如第三方数据接口，重点考虑的是跨平台、跨语言、支持高并发、保证安全；而局域网内的分布式服务，重点考虑的是高性能、稳定性、可伸缩性。现在我正在参与企业的CRM系统，是实施于企业的市场营销、销售、服务与技术支持等与客户相关的领域，通过向企业的销售、市场和客户服务的专业人员提供全面、个性化的客户资料，并强化跟踪服务、信息分析的能力，使他们能够协同建立和维护一系列与客户和生意伙伴之间卓有成效的“一对一关系”，从而使企业得以提供更快捷和周到的优质服务、提高客户满意度、吸引和保持更多的客户，从而增加营业额；另一方面则通过信息共享和优化商业流程来有效地降低企业经营成本。使用的是局域网内的虚拟机集成，CRM系统服务端和客户端都使用的是springmvc的RMI远程接口调用，和其他系统的交互，则使用的是webservice，webservice可以集成在springmvc框架中，但是springframe集成的远程调用，同时只能使用一种，否则会造成访问异常。

# 致 谢

本文之所以能够顺利完成，尤其要感谢XXX老师的鼎力支持。从选题开始到报告结束，每一个环节都在X老师亲切的指导下完成。X老师在工作之余，在休息的时间还与我们一起开研讨会，我的每一个问题，X老师都及时的给予回复，对于我的每一个错误都给予指出，在此由衷的感谢X老师，辛苦了。通过这段时间对虚拟机的了解学习让我对企业虚拟化有了新的认识。

另外，要感谢我的家人，是家人的鼓励和支持，使将近不惑之年的我能够全心投入学习和工作之中，顺利完成学业。最后衷心感谢在百忙之中评阅论文和参加[答辩](http://bylw.yjbys.com/lunwendabian/)的各位专家、教授!

# 参考文献

[1]刘冬.《虚拟机技术在计算机实际教学中的应用》[M].中国科技信息,2010.8

[2]龚京民.计算机组装与维护》[R],中国科学研究院2009.1

[3] 黎玉桥.《服务器虚拟化架构研究开发研究》[R].开发研究院,2011.2

[4] 黄志宝.《基于虚拟化的容灾业务自动部署和自动迁移研究与实现国防科学》[N].

技术大学学报,2012.3

[5] 王保平.《带虚拟化技术支持的内核定制编译》[N].国防科学技术大学学报,2012.5

[6] 王春海.《深入学习VMware vSphere 6》[M].人民邮电出版社，2016.6

[7]Nick Marshall,Grant Orchard,Josh Atwell.《精通VMware vSphere 6》[M].人民邮电出版社，2016.11

[8]何坤源.《VMware vSphere 6虚拟化构架实战指南》[M].人民邮电出版社，2016.9

[9]<http://blog.sina.com.cn/s/blog_c2a66b49010190zy.html>. [J].张洪洋的博客

[10] <http://xdwqs.blog.51cto.com/4758880/1622775>.[J].王庆帅的博客