

# **Web Service 学习手册**

**Christian 2010-11-11**



---

# 目 录

## 目录

文档说明.....	4
1 Web Service 简介 .....	4
1.1    什么是 Web Service? .....	4
1.2    Web Service 用到的技术.....	4
1.3    主流 Web Service 框架.....	5
1.3.1    Axis .....	5
1.3.2    Xfire .....	6
1.3.3    CXF.....	6
1.3.4    主流 Web Service 的比较.....	7
2    开发篇.....	8
2.1    Xfire .....	8
2.1.1    下载 Xfire .....	8
2.1.2    使用 Xfire 开发 Web Service 的基本步骤 .....	9
2.1.3    具体应用.....	13
2.1.4    Xfire 与 spring 集成 .....	42

---

# 文档说明

本文详细介绍了如何用 Xfire,CXF,Axis 开发 Web 服务.Web service 的底层原理并不在本文研究的范围内。所有示例都在 sample 目录里。这个学习手册的目的是只要仔细阅读本文档以及示例代码,任何对 webservice 没有基础的程序员都可以用这三个框架编写 webservice 接口。但前提是需要动手做每一个示例。

## 1 Web Service 简介

### 1.1 什么是 Web Service?

从表面上看,Web service 就是一个应用程序,它向外界暴露出一个能够通过 Web 进行调用的 API。这就是说,你能够用编程的方法通过 Web 来调用这个应用程序。我们把调用这个 Web service 的应用程序叫做客户。

另一种更精确的解释:Web services 是建立可互操作的分布式应用程序的新平台。Web service 平台是一套标准,它定义了应用程序如何在 Web 上实现互操作性。你可以用任何你喜欢的语言,在任何你喜欢的平台上写 Web service,只要我们可以通过 Web service 标准对这些服务进行查询和访问。Web service 平台需要一套协议来实现分布式应用程序的创建。任何平台都有它的数据表示方法和类型系统。要实现互操作性,Web service 平台必须提供一套标准的类型系统,用于沟通不同平台、编程语言和组件模型中的不同类型系统。在传统的分布式系统中,基于界面(interface)的平台提供了一些方法来描述界面、方法和参数(译注:如 COM 和 COBAR 中的 IDL 语言)。同样的,Web service 平台也必须提供一种标准来描述 Web service,让客户可以得到足够的信息来调用这个 Web service。最后,我们还必须有一种方法来对这个 Web service 进行远程调用。这种方法实际是一种远程过程调用协议(RPC)。为了达到互操作性,这种 RPC 协议还必须与平台和编程语言无关。

### 1.2 Web Service 用到的技术

为了实现平台无关,实现独立的访问 Web 服务,业界制定了一系列技术标准,下面是一些重要的技术:

---

## 1. XML

可扩展的标记语言(XML)是 Web service 平台中表示数据的基本格式。它的内容与表示的分离十分理想,除了易于建立和易于分析外,XML 主要的优点在于它既是平台无关的,又是厂商无关的。无关性比技术优越性更重要的:软件厂商是不会选择一个由竞争对手所发明的技术的。

## 2. SOAP

Web service 建好以后,你或者其他人就会去调用它,简单对象访问协议(SOAP)提供了标准的 RPC 方法来调用 Web service,SOAP 规范定义了 SOAP 消息的格式,以及怎样通过 HTTP 协议来使用 SOAP,SOAP 也是基于 XML,XML 是 SOAP 的数据编码方式。

## 3. WSDL

你会怎样向别人介绍你的 Web service 有什么功能,以及每个函数调用时的参数呢?你可能会自己写一套文档,你甚至可能会口头上告诉需要使用你的 Web service 的人。这些非正式的方法至少都有一个严重的问题:当程序员坐到电脑前,想要使用你的 Web service 的时候,他们的工具(如 Visual Studio)无法给他们提供任何帮助,因为这些工具根本就不了解你的 Web service。解决方法是:用机器能阅读的方式提供一个正式的描述文档。Web service 描述语言(WSDL)就是这样一个基于 XML 的语言,用于描述 Web service 及其函数、参数和返回值。因为是基于 XML 的,所以 WS 是机器可阅读的,又是人可阅读的,这将是一个很大的好处。一些最新的开发工具既能根据你的 Web service 生成 WSDL 文档,又能导入 WSDL 文档,生成调用相应 Web service 的代码。

# 1.3 主流 Web Service 框架

## 1.3.1 Axis

Axis (Apache Extensible Interaction System)是一款开源的Web Service 运行引擎,它是SOAP 协议的一个实现,其本身来源于Apache 的另一个项目Apache SOAP。Axis 分为1.x 系列和2 系列,两个系列体系结构和使用上有较大的区别,相对而言,Axis1.x 更加稳定,文档也比较齐全。

---

## 1.3.2 Xfire

XFire 是下一代的java SOAP 框架。XFire 提供了非常方便的API, 使用这些API 可以开发面向服务(SOA)的程序。它支持各种标准, 性能优良(基于低内存的STAX 模型)。

- 支持多个重要的Web Service 标准, 包括SOAP、WSDL、WS-I Basic Profile WSAddressing、WS-Security 等。
- 高性能的SOAP 栈。
- 可选的绑定(binding)方式, 如POJO、XMLBeansJAXB1.1、JAXB2.0、Castor 和JiBX 等。
- 支持JSR181 API。
- 多种传输方式, 如HTTP、JMS、XMPP、In-JVM 等。
- 灵活的接口。
- 支持多个容器, 如Spring、Pico、Plexus、Loom。
- 支持JBI, 参看servicemix 项目(<http://servicemix.org>)。
- 客户端和服务端代码生成。

## 1.3.3 CXF

### 1.3.3.1 CXF 的由来

Apache CXF 项目是由Objectweb Celtix 和Codehaus XFire 合并成立的。Objectweb Celtix 是由IONA 公司赞助、于2005 年成立的开源Java ESB 产品, XFire 则是业界知名的SOAP 堆栈。合并后的Apache CXF 融合该两个开源项目的功能精华, 提供了实现SOA 所需要的核心ESB 功能框架, 包括SOA 服务创建, 服务路由, 及一系列企业级QoS 功能。此次发布代表了Apache CXF 开发人员及社区用户一年的努力结果, 并标志Apache CXF 软件的进一步成熟, 成为实现SOA 的优秀技术解决方案之一。2.1 版本的CXF, 已经是一个正式的Apache 顶级项目。

### 1.3.3.2 CXF 的功能

CXF 提供了一套创建SOA 服务的基础设施框架, 用户由此可以按照自己喜欢的编程模式, 利用Apache CXF 提供的简单易用工具(包括Maven 插件), 创建适合SOA 环境的任何WEB 服务, 包括SOAP/HTTP 服务及REST/HTTP 服务。Apache CXF 可扩展的插拔式架构不但支持XML 消息格式和HTTP 通信协议, 而且还支持基于其他通信协议如IIOP 和非XML 消息格式如CORBA CDL 或JSON。

主要功能列表如下:

- 支持JAX-WS 2.1, 部署JAX-WS 已经更新至JAX-WS 2.1 规范。
- JAX-RS 0.6 REST 的初期部署基于服务框架。
- Javascript 客户端生成和支持, SOAP 基于端点的可以有Javascript 带着JS URL 自动

- 
- 创建。
  - CORBA 的约束力来自Yoko, JAX-WS 客户端/服务器能使IIOP 与CORBA 进程进行交互。
  - 支持Java to WSDL、WSDL to Java、XSD to WSDL、WSDL to XML、WSDL to SOAP、WSDL to Service。
  - 支持XmlBeans 运行库, 允许数据模型使用XmlBeans。
  - 支持SOAP1.1&1.2、WSDL1&2、WS-Addressing、WS-Policy、WS-RM、WS-Security 和WS-I BasicProfile。
  - 支持JAX-WS、JAX-WSA、JSR-181、SAAJ。
  - Apache CXF 提供方便的Spring 整合方法, 可以通过注解、Spring 标签式配置来暴露 WebServices 和消费WebServices。

### 1.3.4 主流 Web Service 的比较



主流WebService框架.doc

---

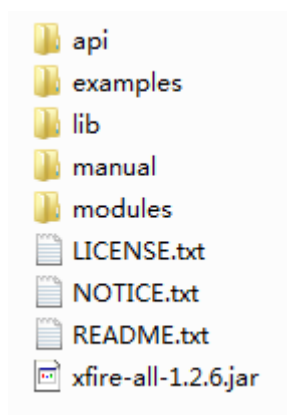
## 2 开发篇

### 2.1 Xfire

#### 2.1.1 下载 Xfire

XFire的官方下载地址: <http://XFire.codehaus.org/>, 目前最新版本是1.2.6,这里我们的例子就是使用这个版本。

从官方网站下载到本地机器后解压, 目录结构如下:



xfire-distribution-1.2.6

|\_\_\_api (javadoc文档)

|\_\_\_examples (XFire例子)

|\_\_\_lib (XFire所需的jars包)

|\_\_\_modules (XFire模块)

|\_\_\_xFire-all-1.2.6.jar

|\_\_\_几个授权和说明TXT文档

XFire 的配置我们以一个简单的 Web 服务的例子来进行说明, 并且在这里你将了解到如何将一个 java 文件转换成 web 服务? services.xml 文件是如何定义的? 如何发布这个 web 服务?



---

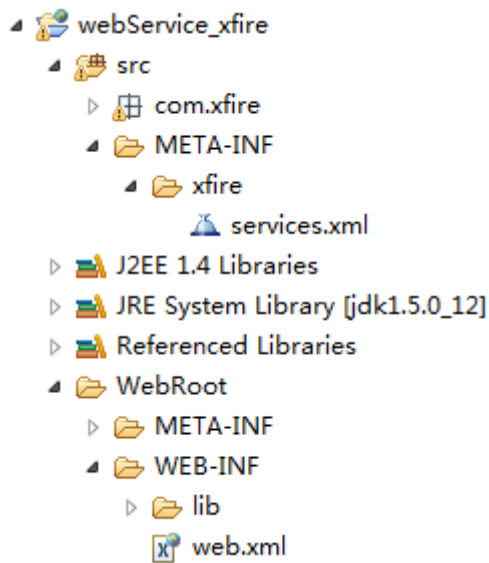
如何获得这个服务的 WSDL?

开发环境: Eclipse6.5, tomcat6, xFire1.2.6, Win7

## 2.1.2 使用 Xfire 开发 Web Service 的基本步骤

- 检验 Java 类的方法和构造函数是公共的 (public)。
- 将 XFire 和其他第三方库添加到你的 Web 应用的 WEB-INF/lib 目录下。
- 将 XFire Servlet 相关的入口添加到 web.xml 中。
- 创建 services.xml 并把它放到 WEB-INF/classes/META-INF/XFire 目录下。
- 编写 Java 接口与实现类。


首先打开 Eclipse, 创建一个 web 工程。目录结构如下图:





### 2.1.2.1 将 XFire 和其他第三方库添加到 WEB-INF/lib 目录下



















将需要的 Jar 包 copy 到 WEB-INF/lib 目录下,需要的 jar 如下图所示: (这些 jar 文件可以从 distribution 和它的 lib 目录下得到)

---

 JRE System Library [jdk1.5.0\_12]

 J2EE 1.4 Libraries

 Referenced Libraries

-  XmlSchema-1.1.jar
-  activation-1.1.jar
-  commons-beanutils-1.7.0.jar
-  commons-codec-1.3.jar
-  commons-httpclient-3.0.jar
-  commons-logging-1.0.4.jar
-  jaxen-1.1-beta-9.jar
-  jdom-1.0.jar
-  mail-1.4.jar
-  stax-api-1.0.1.jar
-  stax-utils-20040917.jar
-  wsdl4j-1.6.1.jar
-  wstx-asl-3.2.0.jar
-  xbean-2.2.0.jar
-  xbean-spring-2.8.jar
-  xfire-all-1.2.6.jar
-  xmlsec-1.3.0.jar
-  spring-2.5.6.jar

## 2.1.2.2 修改 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <servlet>
    <servlet-name>XFireServlet</servlet-name>
    <servlet-class>org.codehaus.xfire.transport.http.XFireConfigurableServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

---

### 2.1.2.3 创建 Java 服务类

```
package com.xfire.simple;

import java.util.Date;
import com.xfire.simple.exception.HelloException;
import com.xfire.simple.exception.HelloExceptionDetail;
/**
 * Xfire服务端, 一个简单的普通java类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class HelloWroldManager {

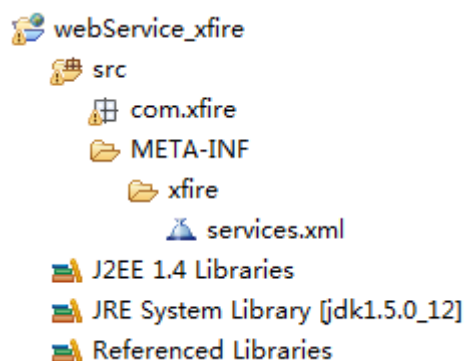
    public void sayHello() {
        System.out.println("Hello Web Service");
    }

    public long add(int a, int b) {
        System.out.println("client call HelloWroldService.add(\"+a\", \"+b\") .....");
        return a + b;
    }

    public String reverse(String str) {
        System.out.println("client call HelloWroldService.reverse(\"+str\") .....");
        StringBuffer sb = new StringBuffer();
        for(int i = str.length() - 1 ; i >= 0 ; i--){
            sb.append(str.charAt(i));
        }
        return sb.toString();
    }
}
```

### 2.1.2.4 新建 services.xml

在 src 下创建 META-INF/xfire/目录, 然后新建 services.xml.



---

内容如下:

```
<beans xmlns="http://xfire.codehaus.org/config/1.0">
  <service>
    <name>HelloWroldService</name>
    <namespace>http://xfire.webservice.com/HelloWroldService</namespace>
    <serviceClass>com.xfire.simple>HelloWroldManager</serviceClass>
  </service>
</beans>
```

这个文档定义了你发布的 Web 服务,定义了一个名为 HelloWroldService 的服务,服务类为 com.xfire.simple>HelloWroldManager。具体解释如下:

对 Web 服务的定义包含在<service>元素内,<service>元素下还有若干子元素。第一个子元素是<name>,你可以提供任何有效的 xml 名字,这个名字会被客户端程序和服务器上的其他组件使用。例如,当服务器起来以后,你可以在浏览器上使用这个名称来查看 WSDL。

下一个子元素是<namespace>,任何有效地 xml 名称都可以,<namespace>将作为你服务器的唯一标识变量使用。

<serviceClass>元素包含 Java 类名用来指明方法的签名。

<implementationClass>元素记录实现接口的 Java 类名。这是一个可选元素,如果前一个元素<serviceClass>填入的是接口,那么此处就要填入相应的实现类名。

service: 包含你要生成的服务。

name: 表示你的服务名字。

namespace: 表示你的命名空间。

serviceClass: 提供服务的类一般是接口,也可以是具体的实现类。

implementationClass: 对应接口的实现。

scope: 存活的范围。

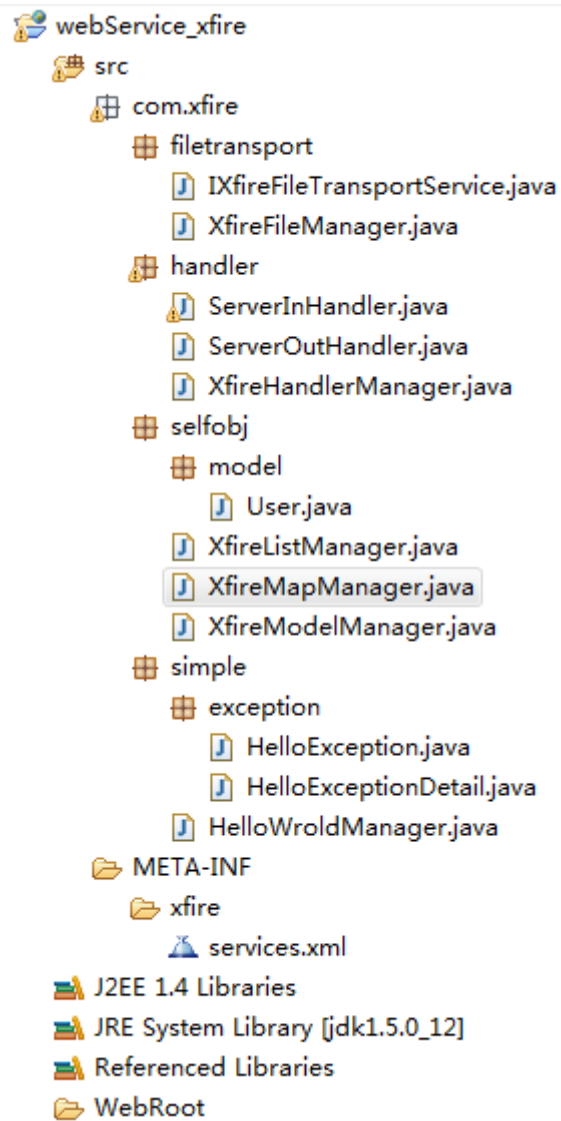
### 2.1.2.5 发布

如何部署略。启动tomcat, 打开浏览器, 在浏览器地址栏中输入

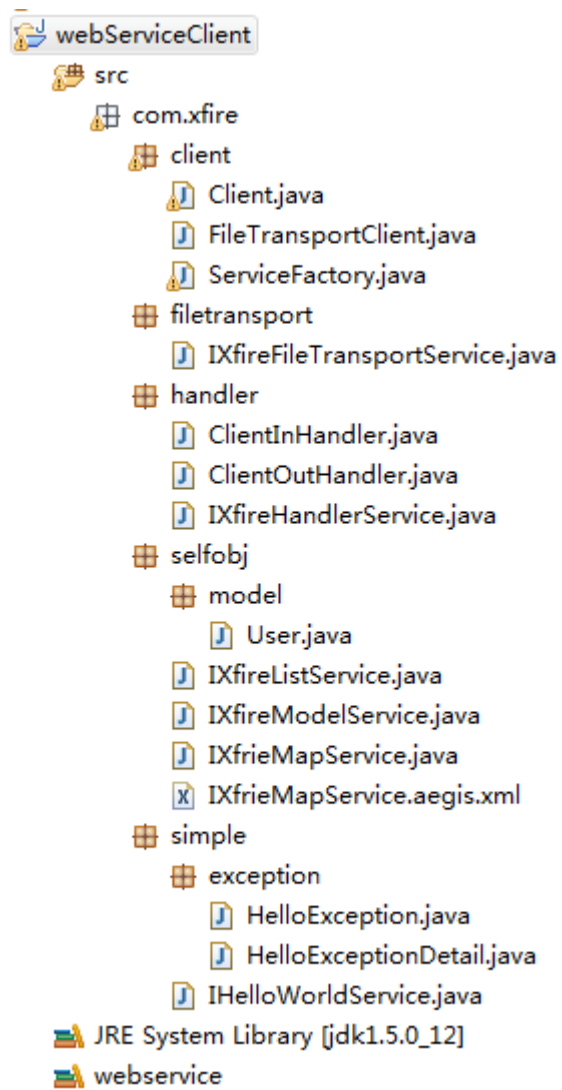
<http://localhost:8080/xfire/services/HelloWroldService?wsdl>

可以查看这个web服务的wsdl文档。





客户端(JavaProject):



Web Service 的接口(客户端建立)

---

```
package com.xfire.simple;

import java.util.Date;
import com.xfire.simple.exception.HelloException;

/**
 * 客户端本地接口，其中包含远程ws的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程ws对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IHelloWorldService {

    public void sayHello();

    public long add(int a, int b);

    public String reverse(String str);

    public Date getDate();

    public String getException(String message) throws HelloException;
}
```

服务端 Service 业务实现:



---

```
package com.xfire.simple;

import java.util.Date;
import com.xfire.simple.exception.HelloException;
import com.xfire.simple.exception.HelloExceptionDetail;
/**
 * webService 服务端, 只是一个简单的类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class HelloWroldManager {

    public void sayHello() {
        System.out.println("Hello Web Service");
    }

    public long add(int a, int b) {
        System.out.println("client call HelloWroldService.add(\"+a+\",\"+b+\") .....");
        return a + b;
    }

    public String reverse(String str) {
        System.out.println("client call HelloWroldService.reverse(\"+str+\") .....");
        StringBuffer sb = new StringBuffer();
        for(int i = str.length() - 1 ; i >= 0 ; i--){
            sb.append(str.charAt(i));
        }
        return sb.toString();
    }

    public Date getDate() {
        return new Date();
    }
}
```

Services.xml 配置

```
<service>
    <name>HelloWroldService</name>
    <namespace>http://xfire.webservice.com/HelloWroldService</namespace>
    <serviceClass>com.xfire.simple.HelloWroldManager</serviceClass>
</service>
```

客户端:

---

```
package com.xfire.client;

import java.util.ArrayList;

/**
 * Xfire客户端,
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class Client {

    public static void helloWorld() {
        System.out.println("===== helloWorld() =====");
        String url = "http://localhost:8080/xfire/services/HelloWroldService";
        IHelloWorldService service = (IHelloWorldService) ServiceFactory.getService(
            url, IHelloWorldService.class);

        service.sayHello();
        long result = service.add(1, 2);
        String str = service.reverse("123456");
        Date date = service.getDate();
        System.out.println(result);
        System.out.println(str);
        System.out.println(date);
    }

    public static void main(String[] args) {
        helloWorld();
    }
}
```

```

package com.xfire.client;

import java.net.MalformedURLException;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

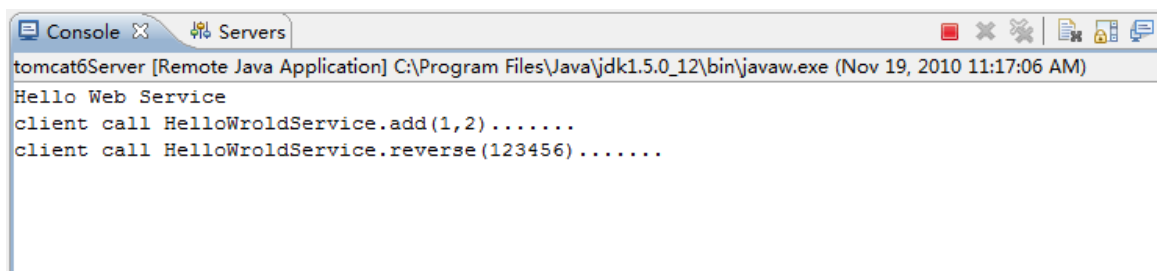
/**
 * Xfire客户端, create a proxy for the deployed service
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public final class ServiceFactory {

    private ServiceFactory() {}

    public static Object getService(String url, Class clazz) {
        Service serviceModel = new ObjectServiceFactory().create(clazz);
        Object service = null;
        try {
            //create a proxy for the deployed service
            service = new XFireProxyFactory().create(serviceModel, url);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        return service;
    }
}

```

服务端结果:

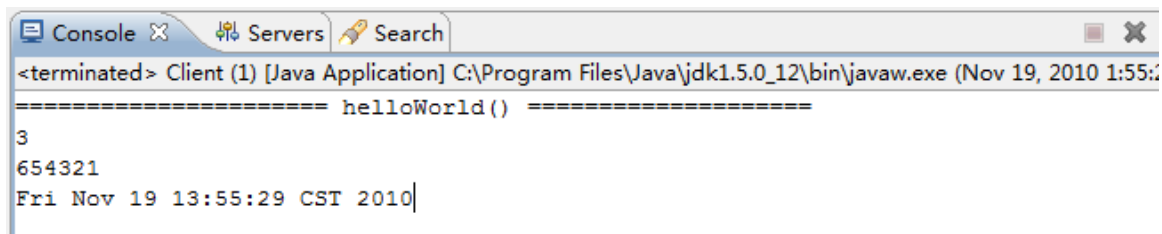


```

tomcat6Server [Remote Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw.exe (Nov 19, 2010 11:17:06 AM)
Hello Web Service
client call HelloWorldService.add(1,2).....
client call HelloWorldService.reverse(123456).....

```

客户端结果:



```

<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw.exe (Nov 19, 2010 1:55:29 PM)
==== helloWorld() =====
3
654321
Fri Nov 19 13:55:29 CST 2010

```

---

## 2.1.3.2 传递复杂对象

### 2.1.3.2.1 传递 List 和数组

Web Service 的接口(客户端建立)

```
package com.xfire.selfobj;

import java.util.List;

import com.xfire.selfobj.model.User;

/**
 * 客户端本地接口，其中包含远程WS的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程WS对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 * 可以传递List,Array,Map,自定义对象
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IXfireListService {

    public List<String> getStringList();
    public List<String> setStringList(List<String> list);
    public List<Object> getObjectList();
    public List<User> getUserList();
    public String[] getStringArray();
}
```

服务端 Service 业务实现:

---

```
package com.xfire.selfobj;

import java.util.ArrayList;

/**
 * 服务端获得List的Service实现类
 * List一定要泛型,否则报错:
 * org.codehaus.xfire.XFireRuntimeException: Cannot create mapping for java.util.List
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class XfireListManager implements IXfireListService{

    public List<String> getStringList(){
        List<String> list = new ArrayList<String>();
        list.add("a");
        list.add("b");
        list.add("c");
        list.add("d");
        return list;
    }

    /**
     * 服务端可以得到客户端传的List对象
     *
     * @param list
     * @return
     */
    public List<String> setStringList(List<String> list){
        list.add("z");
        return list;
    }
}
```

---

```
public List<Object> getObjectList() {
    List<Object> list = new ArrayList<Object>();
    list.add("a");
    list.add(1);
    list.add("c");
    list.add("d");
    return list;
}

public List<User> getUserList() {
    List<User> list = new ArrayList<User>();
    User user = new User();
    user.setName("power");
    user.setAge(12);
    user.setSex("公司");
    user.getAddress().add("北京");
    user.getAddress().add("朝阳");
    user.getAddress().add("北辰");

    list.add(user);

    user = new User();
    user.setName("christian");
    user.setAge(12);
    user.setSex("男");
    user.getAddress().add("北京");
    user.getAddress().add("海淀");
    user.getAddress().add("中关村");
    list.add(user);

    return list;
}
```

Services.xml 配置

```
<service>
    <name>XfireListService</name>
    <namespace>tlservice</namespace>
    <serviceClass>com.xfire.selfobj.XfireListManager</serviceClass>
</service>
```

客户端:

---

```

public static void xfireListService() {
    System.out.println("===== xfireListService() =====");
    String url = "http://localhost:8080/xfire/services/XfireListService";
    IXfireListService service = (IXfireListService) ServiceFactory.getService(
        url, IXfireListService.class);

    List objects = service.getObjectList();
    List results = service.getStringList();
    List<User> users = service.getUserList();
    String[] arrays = service.getStringArray();
    System.out.println("objects :"+objects);
    System.out.println("results :"+results);
    System.out.println("arrays :"+arrays[0]+arrays[1]);
    for(int i = 0 ; i < users.size() ; i++){
        User user = users.get(i);
        System.out.println("用户"+i+":");
        System.out.println("    用户名: "+user.getName());
        System.out.println("    年龄: "+user.getAge());
        System.out.println("    性别: "+user.getSex());
        System.out.print("    地址: ");
        for(String address:user.getAddresss()){
            System.out.print(address+",");
        }
        System.out.println();
    }
    List<String> list = new ArrayList<String>();
    list.add("x");
    list.add("y");
    List returnList = service.setStringList(list);
    System.out.println(returnList);
}

public static void main(String[] args) {
    xfireListService();
}

```

客户端结果:

```
Console X Servers Search
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw.e
===== xfireListService() =====
objects :[a, 1, c, d]
results :[a, b, c, d]
arrays :ultrapower
用户0:
    用户名: power
    年龄: 12
    性别: 公司
    地址: 北京,朝阳,北辰,
用户1:
    用户名: christian
    年龄: 12
    性别: 男
    地址: 北京,海淀,中关村,
[x, y, z]
```

注意: List 一定要泛型 否则报错, 不好使。

org.codehaus.xfire.XFireRuntimeException: Cannot create mapping for java.util.List

### 2.1.3.2.2 传递自定义对象

Web Service 的接口(客户端建立)



---

```
package com.xfire.selfobj;

import java.util.Map;

import com.xfire.selfobj.model.User;

/**
 * 客户端本地接口，其中包含远程ws的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程ws对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IXfireMapService {

    public Map<Integer, String> getMap();

    public Map<Integer, User> getUserMap();

    public Map<Integer, String> setMap(Map<Integer, String> map);
}
```

服务端:

---

```
package com.xfire.selfobj.model;

import java.util.ArrayList;

/**
 * webService 服务端的自定义对象，这个对象里包含了List,Map,以及自定义对象
 * 客户端必须有一个与这个类名和包名一样的自定义对象，要不然会找不着
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class User {

    private String name;
    private int age;
    private String sex;
    private List<String> addresss = new ArrayList<String>();
    private List<User> users = new ArrayList<User>();
    private List<Map<String, String>> maps = new ArrayList<Map<String, String>>();
    private Map<String, String> skill = new HashMap<String, String>();

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

---

```

package com.xfire.selfobj;

import java.util.HashMap;

/**
 * 服务端获得自定义对象的Service实现类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class XfireModelManager {

    public User getUser() {
        User user = new User();
        user.setName("power");
        user.setAge(12);
        user.setSex("公司");
        user.getAddress().add("北京");
        user.getAddress().add("朝阳");
        user.getAddress().add("北辰");

        User user2 = new User();
        user2.setName("power2");

        user.getMaps().add(new HashMap<String, String>());

        user getUsers().add(user2);

        return user;
    }

    /**
     * 服务端可以得到客户端传的自定义复杂类型
     *
     * @param user
     * @return
     */
    public User saveUser(User user) {
        System.out.println(user.getName());
        System.out.println(user.getAddress().get(0));
        user.setName("chinese");
        return user;
    }
}

```

Services.xml 配置

---

```
<service>
    <name>XfireModelService</name>
    <namespace>tlservice</namespace>
    <serviceClass>com.xfire.selfobj.XfireModelManager</serviceClass>
    <scope>application</scope>
</service>
```

客户端:

```
public static void xfireUserService() {
    System.out.println("===== xfireUserService() =====");
    String url = "http://localhost:8080/xfire/services/XfireModelService";
    IXfireModelService service = (IXfireModelService) ServiceFactory.getService(
        url, IXfireModelService.class);
    User user = service.getUser();
    System.out.println("用户名: " + user.getName());
    System.out.println("年龄: " + user.getAge());
    System.out.println("性别: " + user.getSex());
    System.out.print("地址: ");
    for (String address: user.getAddresss()) {
        System.out.print(address + ", ");
    }
    for (Map<String, String> map: user.getMaps()) {
        System.out.print(map + ", ");
    }
    User user2 = new User();
    System.out.println(user2);
    user2.getAddresss().add("beijing");
    user2.setName("china");
    User user3 = service.saveUser(user2);
    System.out.println(user3);
    for (User u: user getUsers()) {
        System.out.print(u.getName() + ", ");
    }
    System.out.println();
    System.out.print("技能: ");
    Map<String, String> map = user.getSkill();
    for (Map.Entry<String, String> entry : map.entrySet()) {
        System.out.println(entry.getKey() + ": " + entry.getValue());
    }
}
```

客户端结果:

```
Console X Servers Search
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw
===== xfireUserService() =====
用户名: power
年龄: 12
性别: 公司
地址: 北京,朝阳,北辰,{},com.xfire.selfobj.model.User@b61fd1
com.xfire.selfobj.model.User@11f2ee1
power2,
技能: java: 5年
xml: 1年
oracle: 3年
Linux: 2年
```

注意：自定义对象 server 和 client 端一定要同包名同类名，不然会不好使。如果你去断点调试，你会很清楚的发现在客户端与服务端的程序关系。

### 2.1.3.2.3 传递 Map

Web Service 的接口(客户端建立)

```
package com.xfire.selfobj;

import java.util.Map;

import com.xfire.selfobj.model.User;

/**
 * 客户端本地接口，其中包含远程ws的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程ws对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IXfrieMapService {

    public Map<Integer, String> getMap();

    public Map<Integer, User> getUserMap();

    public Map<Integer, String> setMap(Map<Integer, String> map);
}
```

除了以上的做法还要做一个事情就是绑定 aegis，这个在这说下 aegis 的绑定有个规则就是必须和你服务端的接口叫一样的名字并且放在同一目录下，例如我们的是

---

IXfrieMapService 接口那么我们的 aegis 就应该叫做 IXfrieMapService.aegis.xml 并且要和 IXfrieMapService 放在同一目录下。命名规范就是：接口名字.aegis.xml

现在给出 IXfrieMapService.aegis.xml

```
<?xml version="1.0" encoding="utf-8"?>
<mappings>
    <mapping>
        <method name="getUserMap">
            <return-type componentType="com.xfire.selfobj.model.User"/>
        </method>
    </mapping>
</mappings>
```

服务端:

```
package com.xfire.selfobj;

import java.util.HashMap;

/**
 * 服务端获得Map的Service实现类
 * 对于map的value是自定义类型的，需要在客户端进行配置
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class XfireMapManager{

    public Map<Integer, String> getMap() {
        Map<Integer, String> map = new HashMap<Integer, String>();
        map.put(1, "haha");
        map.put(2, "hehe");
        return map;
    }

    public Map<Integer, User> getUserMap() {
        Map<Integer, User> map = new HashMap<Integer, User>();
        User user = new User();
        user.setName("first");
        map.put(1, user);
        user = new User();
        user.setName("second");
        map.put(2, user);
        return map;
    }
}
```

Services.xml 配置

---

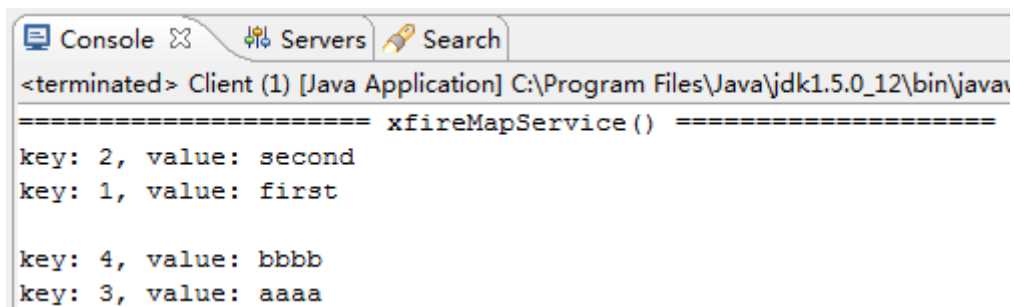
```
<service>
  <name>XfireMapService</name>
  <namespace>http://xfire.webservice.com/MapService</namespace>
  <serviceClass>com.xfire.selfobj.XfireMapManager</serviceClass>
</service>
```

客户端:

```
public static void xfireMapService() {
    System.out.println("===== xfireMapService() =====");
    String url = "http://localhost:8080/xfire/services/XfireMapService";
    String namespace = "http://xfire.webservice.com/MapService";
    IXfireMapService service = (IXfireMapService) ServiceFactory.getService(
        url, namespace, IXfireMapService.class);
    Map<Integer, String> map = service.getMap();
    Map<Integer, User> usermap = service.getUserMap();

    for(Map.Entry<Integer, User> entry : usermap.entrySet()) {
        System.out.println("key: "+entry.getKey()+" , value: "+entry.getValue().getName());
    }
    System.out.println();
    Map<Integer, String> returnMap = service.setMap(map);
    for(Map.Entry<Integer, String> entry : returnMap.entrySet()) {
        System.out.println("key: "+entry.getKey()+" , value: "+entry.getValue());
    }
}
```

客户端结果:



```
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\java
===== xfireMapService() =====
key: 2, value: second
key: 1, value: first

key: 4, value: bbbb
key: 3, value: aaaa
```

### 2.1.3.2.4 文件传输处理

不知道为什么只有在和 spring 集成时文件才能传输, 单独用 xfire 时不成功。我目前还没找到解决办法, 如果哪位读者有解决办法, 请给我发邮件, 不胜感谢。

现在我们先看看与 spring 集成时文件传输如何处理。

服务端与客户端都必须提一个接口, 且要同包同名。且记!

---

```
package com.xfire.filetransport;

import javax.activation.DataHandler;

/**
 * 传送文件的接口，本地（客户端）与服务端必须用同一个接口，
 * 即：客户端与服务端的接口必须同包同名。
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IXfireFileTransportService {

    public DataHandler getFile(String fileName);
    public String uploadFile(DataHandler dataHandler);
}
```

服务端文件传送实现类：

```
package com.xfire.filetransport;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;

/**
 * 服务端传送文件的实现类，服务的提供者
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class XfireFileManager implements IXfireFileTransportService{

    /**
     * 这是文件在服务端的路径，测试时我把文件都放到了这个路径下面。
     */
    static final String PATH = "D:/code/selfCode/source/xfire_spring/WebRoot/files/";

    public DataHandler getFile(String fileName){
        System.out.println("Client get File..." + fileName);
        DataHandler dataHandler = new DataHandler(
            new FileDataSource(PATH + fileName));
        return dataHandler;
    }

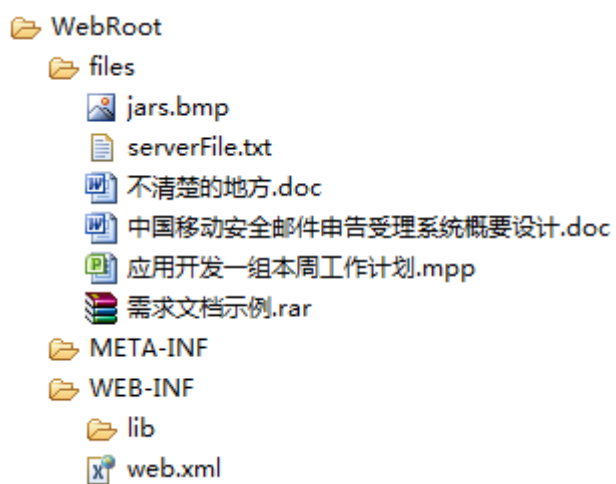
    public String uploadFile(DataHandler dataHandler) {
        return null;
    }
}
```

可以传送任意格式的文件，我测试了如下几种全部通过。我没有写客户端向服务端传文件的例



---

子，但原理都是一样的，有兴趣的同事可以参考着弄弄。



客户端：

---

```
package com.xfire.client;

import java.io.*;
import java.lang.reflect.Proxy;
import javax.activation.DataHandler;
import org.codehaus.xfire.client.XFireProxy;
import org.codehaus.xfire.soap.SoapConstants;
import org.codehaus.xfire.transport.http.HttpTransport;
import com.xfire.filetransport.IXfireFileTransportService;
import org.codehaus.xfire.client.Client;

/**
 * 与服务端进行文件传递的客户端
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class FileTransportClient {

    public static void xfireFileService() {
        System.out.println("===== xfireFileService() =====");
        InputStream input = null;
        OutputStream out = null;
        String url = "http://localhost:8080/xfire/services/XfireFileService";
        String namespace = "http://xfire.webservice.com/XfireFileTransfer";
        IXfireFileTransportService service
            = (IXfireFileTransportService) ServiceFactory.getService(
                url, namespace, IXfireFileTransportService.class);
        Client client = ((XFireProxy) Proxy.getInvocationHandler(service)).getClient();
        client.setProperty(SoapConstants.MTOM_ENABLED, "true");
        client.setProperty(HttpTransport.CHUNKING_ENABLED, "true");

        String fileName = "不清楚的地方.doc";
        DataHandler dataHandler = service.getFile(fileName);
    }
}
```

---

```
System.out.println(FileTransportClient.class.getResource("/").getPath()+fileName);

try {
    input = dataHandler.getInputStream();
    out = new FileOutputStream(
        FileTransportClient.class.getResource("/").getPath()+fileName);
    byte[] buf = new byte[1024];
    while(true){
        int n = input.read(buf); // n 个表示实际读了多少个字节
        out.write(buf,0,n);
        if(n<1024) break;
    }
} catch (IOException e) {
    e.printStackTrace();
} catch (Exception e) {
}finally{
    if(input != null){
        try {
            input.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    if(out != null){
        try {
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
```

### 2.1.3.3 异常处理

Web Service 的接口(客户端建立)

---

```
package com.xfire.simple;

import java.util.Date;
import com.xfire.simple.exception.HelloException;

/**
 * 客户端本地接口，其中包含远程ws的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程ws对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IHelloWorldService {

    public void sayHello();

    public long add(int a, int b);

    public String reverse(String str);

    public Date getDate();

    public String getException(String message) throws HelloException;
}
```

服务端 Service 业务实现:

```
package com.xfire.simple;

import java.util.Date;
/**
 * webService 服务端，只是一个简单的类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class HelloWroldManager {

    public String getException(String message) throws HelloException{
        if("").equals(message)){
            throw new HelloException("参数不能为空",
                new HelloExceptionDetail("异常测试成功!"));
        }
        System.out.println(message);
        return "没有异常";
    }
}
```

客户端:

```

public static void testXfireException() {
    System.out.println("===== testXfireException() =====");
    String url = "http://localhost:8080/xfire/services/HelloWroldService";
    String namespace = "http://xfire.webservice.com/HelloWroldService";
    IHelloWorldService service = (IHelloWorldService) ServiceFactory.getService(
        url, namespace, IHelloWorldService.class);

    try {
        String s = service.getException("");
        System.out.println(s);
    } catch (HelloException e) {
        System.out.println(e.getFaultInfo().getMessage());
        e.printStackTrace();
    }
}

```

当输入参数为“”时，

```

<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw.exe (Nov 19, 2010 3:01:10 PM)
===== testXfireException() =====
异常测试成功!
com.xfire.simple.exception.HelloException: 参数不能为空
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0 (Native Method)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance (NativeConstructorAccessorImpl.java:62)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance (DelegatingConstructorAccessorImpl.java:45)
    at java.lang.reflect.Constructor.newInstance (Constructor.java:494)
    at org.codehaus.xfire.client.ClientFaultConverter.processFaultDetail (ClientFaultConverter.java:100)
    at org.codehaus.xfire.client.ClientFaultConverter.invoke (ClientFaultConverter.java:75)
    at org.codehaus.xfire.handler.HandlerPipeline.invoke (HandlerPipeline.java:100)
    at org.codehaus.xfire.client.Client.onReceive (Client.java:424)
    at org.codehaus.xfire.transport.http.HttpChannel.sendViaClient (HttpChannel.java:100)
    at org.codehaus.xfire.transport.http.HttpChannel.send (HttpChannel.java:48)

```

当参数不为“”时。

```

<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw.exe
===== testXfireException() =====
没有异常

```

注意：在客户端也需要有与服务端同包同名的两个异常类。

## 2.1.3.4 Handler 处理

Web Service 的接口(客户端建立)

---

```
package com.xfire.handler;

/**
 * 客户端本地接口，其中包含远程ws的方法（同名、同返回类型、同参数类型），
 * 则通过Service可以获得一个对远程ws对象的引用。用该引用可以直接像调用本地方法一样调用远程方法。
 * 服务端不用做任何设置和调整。
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public interface IXfireHandlerService {

    public String testHandler(String message);
}
```

服务端 Handler

```
package com.xfire.handler;

import java.util.List;

/**
 * 服务端前置handler类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class ServerInHandler extends AbstractHandler{

    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("调用服务端之前做的一件事");
        ctx.setProperty("key", "abcd");
        System.out.println(ctx.getCurrentMessage().getUri());
        System.out.println(((List) ctx.getCurrentMessage().getBody()).get(0));
        System.out.println("ServerInHandler Exit...");
    }
}
```

---

```
package com.xfire.handler;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 服务端前置handler类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class ServerOutHandler extends AbstractHandler {

    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("调用服务端之后做的一件事");
        System.out.println(ctx.getProperty("key"));
        System.out.println("ServerOutHandler Exit...");
    }
}
```

服务端业务类:

```
package com.xfire.handler;

/**
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class XfireHandlerManager {

    public String testHandler(String message) {
        System.out.println("Receive message: "+message);
        return message+"_";
    }
}
```

Services.xml 配置

---

```
<service>
  <name>XfireHandlerService</name>
  <namespace>http://xfire.webservice.com/XfireHandlerService</namespace>
  <serviceClass>com.xfire.handler.XfireHandlerManager</serviceClass>
  <scope>application</scope>
  <inHandlers>
    <handler handlerClass="com.xfire.handler.ServerInHandler"></handler>
  </inHandlers>
  <outHandlers>
    <handler handlerClass="com.xfire.handler.ServerOutHandler"></handler>
  </outHandlers>
</service>
```

客户端 Handler:

```
package com.xfire.handler;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 客户端前置handler类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class ClientOutHandler extends AbstractHandler {

    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("调用客户端之前做的一件事");
        ctx.setProperty("key", "1234");
        System.out.println("ClientInHandler Exit...");
    }
}
```

---



---

```

package com.xfire.handler;

import org.codehaus.xfire.MessageContext;
import org.codehaus.xfire.handler.AbstractHandler;

/**
 * 客户端后置handler类
 *
 * @author <a href="mailto:tianle@ultrapower.com.cn">tianle</a>
 * @version 1.0
 * @since 2010-11-17
 */
public class ClientInHandler extends AbstractHandler {

    public void invoke(MessageContext ctx) throws Exception {
        System.out.println("调用客户端之后做的一件事");
        System.out.println(ctx.getProperty("key"));
        System.out.println("ClientOutHandler Exit...");
    }
}

```

---

客户端:

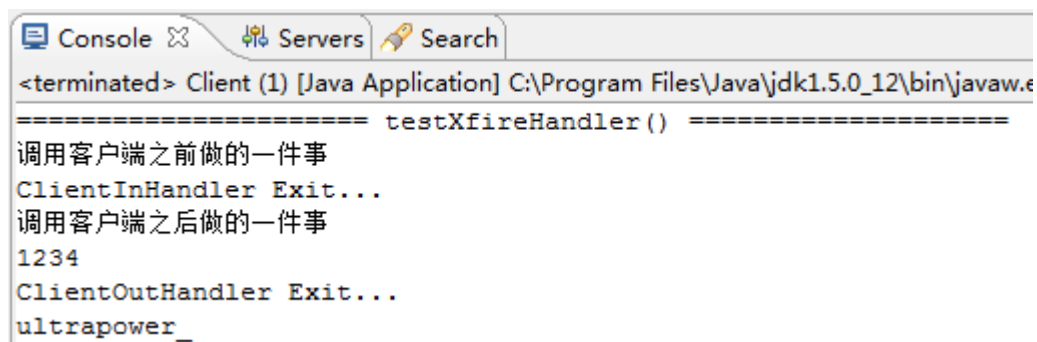
```

public static void testXfireHandler() {
    System.out.println("===== testXfireHandler() =====");
    String url = "http://localhost:8080/xfire/services/XfireHandlerService";
    String namespace = "http://xfire.webservice.com/XfireHandlerService";
    IXfireHandlerService service = (IXfireHandlerService) ServiceFactory.getService(
        url, namespace, IXfireHandlerService.class);

    org.codehaus.xfire.client.Client client
        = org.codehaus.xfire.client.Client.getInstance(service);
    client.addInHandler(new ClientInHandler());
    client.addOutHandler(new ClientOutHandler());
    System.out.println(service.testHandler("ultrapower"));
}

```

客户端结果:



```

Console  Servers  Search
<terminated> Client (1) [Java Application] C:\Program Files\Java\jdk1.5.0_12\bin\javaw.exe
===== testXfireHandler() =====
调用客户端之前做的一件事
ClientInHandler Exit...
调用客户端之后做的一件事
1234
ClientOutHandler Exit...
ultrapower_

```

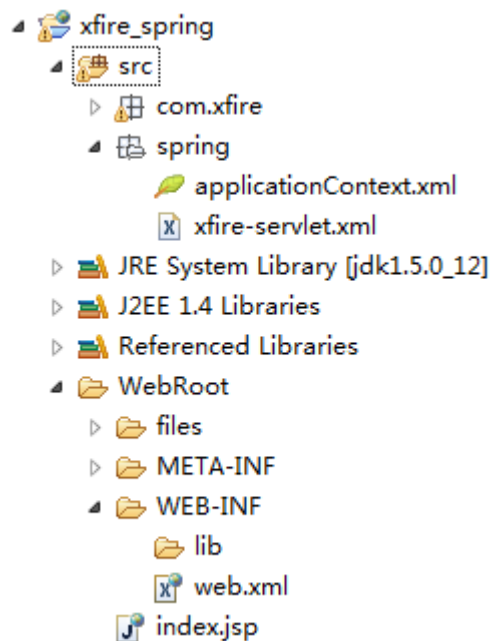
服务端结果:

```
tomcat6Server [Remote Java Application] C:\Pro
ultrapower
ServerInHandler Exit...
Receive message: ultrapower
调用服务端之后做的一件事
abcd
ServerOutHandler Exit...
```

## 2.1.4 Xfire 与 spring 集成

在和Spring 集成时候主要有需要配置Web.xml, applicationContext.xml, xfire-ervlet.xml 这3 个文件，剩下的代码都是上面提供的一样。

新建一个工程，基本上和上面的一样。结构如图：



---

## 2.1.4.1 Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring/applicationContext.xml,
      classpath:org/codehaus/xfire/spring/xfire.xml
    </param-value>
  </context-param>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <listener>
    <listener-class>
      org.springframework.web.util.IntrospectorCleanupListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>XFireServlet</servlet-name>
    <servlet-class>org.codehaus.xfire.spring.XFireSpringServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

---

## 2.1.4.2 applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

    <import resource="xfire-servlet.xml"/>

</beans>
```

## 2.1.4.3 xfire-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

    <bean id="HelloWroldService" class="org.codehaus.xfire.spring.ServiceBean">
        <property name="name" value="HelloWroldService"></property>
        <property name="namespace" value="http://xfire.webservice.com/HelloWroldService"></property>
        <property name="serviceClass" value="com.xfire.simple.HelloWroldManager"></property>
        <property name="serviceBean">
            <ref bean="HelloWroldServiceBean"/>
        </property>
    </bean>

    <bean id="XfireListService" class="org.codehaus.xfire.spring.ServiceBean">
        <property name="name" value="XfireListService"></property>
        <property name="namespace" value="tlservice"></property>
        <property name="serviceClass" value="com.xfire.selfobj.XfireListManager"></property>
        <property name="serviceBean">
            <ref bean="XfireListServiceBean"/>
        </property>
    </bean>

    <bean id="XfireModelService" class="org.codehaus.xfire.spring.ServiceBean">
        <property name="name" value="XfireModelService"></property>
        <property name="namespace" value="tlservice"></property>
        <property name="serviceClass" value="com.xfire.selfobj.XfireModelManager"></property>
        <property name="serviceBean">
            <ref bean="XfireModelServiceBean"/>
        </property>
    </bean>

</beans>
```

---

```

<bean id="XfireMapService" class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name" value="XfireMapService"></property>
    <property name="namespace" value="http://xfire.webservice.com/MapService"></property>
    <property name="serviceClass" value="com.xfire.selfobj.XfireMapManager"></property>
    <property name="serviceBean">
        <ref bean="XfireMapServiceBean"/>
    </property>
</bean>

<bean id="XfireFileService" class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name" value="XfireFileService"></property>
    <property name="namespace" value="http://xfire.webservice.com/XfireFileTransfer"></property>
    <property name="serviceClass" value="com.xfire.filetransport.IXfireFileTransportService"></property>
    <property name="serviceBean">
        <ref bean="XfireFileServiceBean"/>
    </property>
    <property name="properties">
        <map>
            <entry key="mtom-enabled">
                <value>true</value>
            </entry>
        </map>
    </property>
</bean>

<bean id="XfireHandlerService" class="org.codehaus.xfire.spring.ServiceBean">
    <property name="name" value="XfireHandlerService"></property>
    <property name="namespace" value="tlservice"></property>
    <property name="serviceClass" value="com.xfire.handler.XfireHandlerManager"></property>
    <property name="serviceBean">
        <ref bean="XfireHandlerServiceBean"/>
    </property>
    <property name="inHandlers">
        <list>
            <ref bean="ServerInHandler"/>

            </list>
        </property>
        <property name="outHandlers">
            <list>
                <ref bean="ServerOutHandler"/>
            </list>
        </property>
    </bean>

<bean id="HelloWroldServiceBean" class="com.xfire.simple.HelloWroldManager"></bean>
<bean id="XfireListServiceBean" class="com.xfire.selfobj.XfireListManager"></bean>
<bean id="XfireModelServiceBean" class="com.xfire.selfobj.XfireModelManager"></bean>
<bean id="XfireMapServiceBean" class="com.xfire.selfobj.XfireMapManager"></bean>
<bean id="XfireFileServiceBean" class="com.xfire.filetransport.XfireFileManager"></bean>
<bean id="XfireHandlerServiceBean" class="com.xfire.handler.XfireHandlerManager"></bean>
<bean id="ServerInHandler" class="com.xfire.handler.ServerInHandler"></bean>
<bean id="ServerOutHandler" class="com.xfire.handler.ServerOutHandler"></bean>

</beans>

```

---

---

如果这 3 个文件都配置正确了，结果应该和上面的一样。这里一定注意一点：

与 spring 集成时，不能写 `default-lazy-init="true"`，否则配置不起作用。我试了几次都是这样，但我没有去深究其原因，有兴趣的同事可以自己去研究研究。

细心的同事可以会发现客户端得到代理 service 对象的工厂方法有两种重载形式。

其中： 用第一种实现的是

1. 异常处理
2. 文件
3. Map
4. Handler

用第二种实现的是：

1. 简单应用
2. 自定义对象
3. List,Array