第十五章 开发 XFire Web Service 应用

第十五章 开发XFire Web Service应用 1
15.1 介绍 1
15.1.1 Web Service 简介 1
15.1.2 XFire Java SOAP 框架一览 4
15.1.3 MyEclipse 的Web Service 工具简介 5
15.2 基于代码方式创建Web Service项目 6
15.2.1 系统需求 6
15.2.2 创建 HelloWorldService 项目 6
15.2.3 创建Web Service 9
15.2.4 发布运行项目，显示 WSDL 11
15.2.5 用 Web Service Explorer 测试 13
15.2.6 创建Java测试客户端 15
15.2.7 给现有Web项目加入Web Service开发功能 17
15.3 创建单独的客户端项目 17
15.3.1 创建天气预报客户端WeatherWSClient项目 18
15.3.2 创建Web Service Client,从 WSDL生成客户端代码 18
15.3.3 编写运行测试代码 20
15.4 JSR 181 标注方式 Web 服务开发 25
15.4.1 XFire的标注服务开发 25
15.4.2 JBoss下的标注服务开发 27
15.5 可视化创建、修改WSDL 30
15.6 常见问题 33
15.10 小结 35
15.11 参考资料 35

未经许可 禁止打印传播

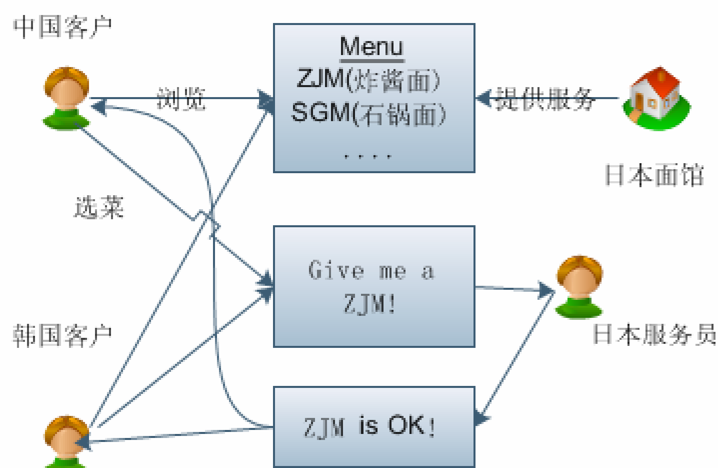
15.1 介绍

15.1.1 Web Service 简介

Web Service（中文一般译作 Web 服务，或者直接就保持原来的英文词汇不变）是什么？它是一项最早主要由微软和 IBM 提出的技术规范。Web Service 的服务描述囊括了与服务交互需要的全部细节，包括消息格式，传输协议等，该接口隐藏了服务实现的细节，允许通过独立于服务实现，独立于硬件或者软件平台，独立于编程语言的方式使用服务。这使得基于 Web Services 的应用程序具备松散耦合,面向组件和跨技术实现的特点。Web Service 可以被设计来履行一项或一组特定的任务，可以单独或与其他 Web Service 一起用于实现复杂的聚集或商业交易，构建复杂的 Web Service 网络。这是官方的定义。那么其

实主要的问题，还是为了解决开发语言过多，互相之间无法方便的进行调用的困难，希望用一种大家都认可的方式实现网络功能的互联互通。如果读者去 Google 搜索这个词，那将会得到很多不同的答案，甚至是模糊不清的定义。而它，却是现在热炒的面向服务的体系结构（SOA，Service Oriented Architecture）的基石，SOA 强调的重点是 Web 服务。那么，到底什么是 Web 服务？答案也许很简单：一种中间技术层（或称中间件）。

我们来举一个简单的例子，大家都知道中国人懂日文的比较少，而日本人懂中文的也比较少，假设现在日本人开了个日本面馆，提供各种面例如炸酱面，石锅面等等，而且他想提供服务给全球各地的客户，让他们都能看懂提供的服务，并自己点面。那么是否这个日本人就需要学习全球各地的语言，然后挂上 50 种语言的面单，之后接待 50 种不同语言的人呢？其实不用那么复杂，大家只需要妥协一下，都采用中间语言：英语来交流就行了。提供的菜单使用英文，而大家点面也使用英文，服务员通告服务结果也用英文，这样这个矛盾就解决了。当然来吃饭的客户私下里如何评价面，或者如何称呼面，而日本服务员是否需要用英文来通知厨房师傅来做饭，这都不是大家所关心的问题，大家所关心的就是：看菜单，点面，吃到面，这样一个结果而已。此过程如图 15.1 所示。再此过程中，英语就扮演了一个中间交互层标注协议的角色，双方都遵守它，就可以实现国际化的互联互通的服务。



15.1 日本面馆用英文解决国际化服务问题

好了，现在引用一些资料来简介 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 调用来实现某个功能的应用程序。例如，可以创建一个 Web Service，它的作用是查询某公司某员工的基本信息。它接受该员工的编号作为查询字符串，返回该员工的具体信息。你可以在浏览器的地址栏中直接输入 HTTP GET 请求来调用罗列该员工基本信息的 ASP 页面，这就可以算作是体验 Web Service 了。

从深层次上看，Web Service 是一种新的 Web 应用程序分支，它们是自包含、自描述、模块化的应用，可以在网络(通常为 Web)中被描述、发布、查找以及通过 Web 来调用。

Web Service 便是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行互操作。它可以使用标准的互联网协议，像超文本传输协议 HTTP 和 XML，将功能体现在互联网和企业内部网上。

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。你可以用你喜欢的任何语言，在你喜欢的任何平台上写 Web Service。”

我想，看到这定义的人都会感到一头雾水，不知所谓。那么好了，看图 15.2 中的 Web 服务结构图吧。在这个图中，列出了 Web Service 的几个关键部分。包括：

- 简单对象访问协议（SOAP，Simple Object Access Protocol）
- Web 服务描述语言（WSDL，Web Service Definition Language）
- 统一描述、发现和集成（UDDI，Universal Description, Discovery, and Integration）

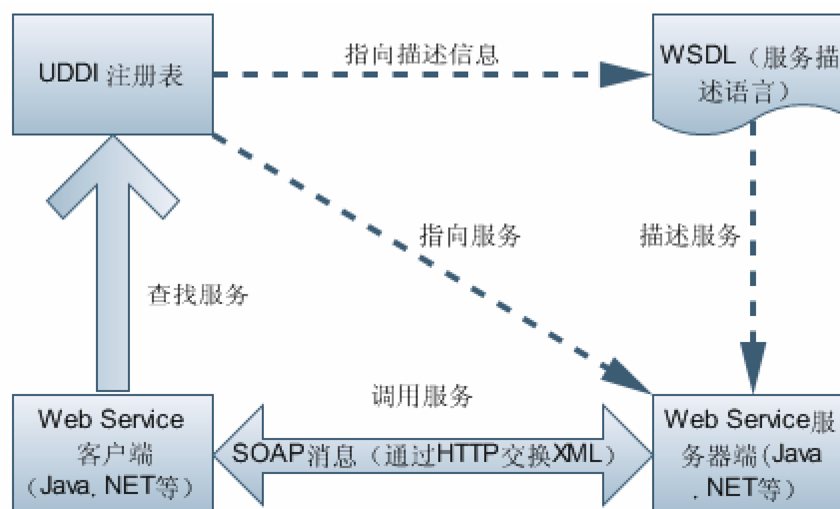


图 15.2 Web Service 结构图

现在，让我们把它和图 15.1 的就餐过程进行类比。假设我们在路上饥肠辘辘，忽然，眼前一亮，前面出现一指示牌：前方 500 米处有就餐服务，详情请去店内咨询。这是什么？这就是 UDDI，它告诉您在什么位置有什么样的 Web 服务，但是很显然服务的细节是不会贴在指示牌上的，需要你亲自去店内查看。到了店内，您最关心的内容就是菜单，也就是 WSDL，它会描述每种菜的特点和定价，您根据自己的情况，来选择打算吃那种菜（当然，这里是国际标准的菜单描述格式）。然后，您用英文，一种大家都听得懂的语言，来向服务员表明您的希望，当然，内容必须是和菜单有关的内容，您不能向服务员说：请问一只轮胎多少钱？这就是 SOAP 所做的事，您必须用 XML 协议来表达您在 WSDL 上看到的服务项目。服务员听到后，翻译成自己能理解的内容后根据实际情况作出反应：很好，某某菜一会就做好；或者不幸的是：对不起，这道菜原料用完了，请换一道。同样，它也用 SOAP 向您返回结果。这就是 Web 服务的过程，不过在计算机的世界里，客户端成了某电脑上的程序（可以用各种语言开发），服务器端则是某台运行 HTTP 并支持 Web Service 服务的服务器程序。因此，问题也潜在存在，那些无法写上菜单的服务项，是无法通过这种途径来传播的，换句话说 Web 服务只能在一定范围内实现互联互通，所以它无法取代现有的其它沟通途径，例如 QQ 和 MSN。最后，UDDI 的出发点很好，不过在企业内部，因为 WSDL 唾手可得，所以没有人愿意再去公司大院里竖起一块牌子列出本公司都有哪些 Web 服务，所以 UDDI 在企业内部使用时，是几乎用不到的。那么，Web 服务能做什么？可以查股票，天气预报等这些类似于一问一答的服务。

现在 Java 的最新版，包括 JDK 6 和 Java EE 5，都对 Web 服务提供了很完整的支持。现在的开发者已经很少再需要去详细的了解 SOAP 和 WSDL 的详细格式，一般来说大家通过可视化的设计器或者开发工具，就可以很快的实现一个 Web 服务，包括生成服务器端代码和客户端代码等等。例如 Java EE 5 使用标注显著改进和简化了 Web 服务支持，而 JDK

直接内置/简化对 **Web Service** 的支持，同时 JAX-WS 2.0 是 Java EE 5 平台中用于 Web 服务的新 API，保留了自然的 RPC 编程模型，同时在以下几个方面进行了改进：数据绑定、协议和传输的独立性、对 Web 服务的 REST 样式的支持以及易开发性。下面是一个 JDK 6 中的 Web 服务端的示例：

```
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
//定义 Web 服务类和方法
@WebService(targetNamespace = "http://jdk.study.hermit.org/client")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class Hello {
    @WebMethod
   public String sayHello(String name) {
       return "hello:" + name;
   }
}

// 启动 Web 服务
 import javax.xml.ws.Endpoint;

  public class StartService {
      public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/HelloService", new Hello());
    }
 }
```

。

当前有很多主流 IDE 对 Web 服务的开发提供了完美的支持，例如商业软件 JBuilder 中有 Web 服务的可视化设计器和生成器，开源免费的 Netbeans 6 中也提供了同样的功能，而 Eclipse 则稍微弱一点，不过在 MyEclipse 6 中则提供了基于 XFire 的 Web 服务的开发支持。Spring 框架则提供了这些支持：使用 JAX-RPC 暴露服务；访问 Web 服务。一般的开发过程是：

1. 开发普通的功能类；
2. 将其包装成 Web 服务，发布于服务器并生成 WSDL 描述；
3. 调用者根据 WSDL 生成客户端代码。

15.1.2 XFire Java SOAP 框架一览

XFire的官方网站地址是 http://xfire.codehaus.org/ ，根据其网站的说法：“

XFire is now CXF

User's looking to use XFire on a new project, should use CXF instead. CXF is a continuation of the XFire project and is considered XFire 2.0. It has many new features, a ton of bug fixes, and is now JAX-WS compliant! XFire will continue to be maintained

未经许可 禁止打印传播

through bug fix releases, but most development will occur on CXF now. For more information see the XFire/Celtix merge FAQ and the CXF website.”可以看出有两处变动，首先是新版本的 XFire 改名成了 CXF，可以认为 CXF 就是 XFire 2.0，是其改进版本。那么老的 XFire 项目只是停留在项目的 Bug 修正上，新功能都转向 CXF。另外，CXF 现在即将成为 Apache 的一个子项目。总之，它们都是开源项目，我们可以免费使用。图 15.3 是 XFire 的标志图。

图 15.3 XFire 标志图

MyEclipse web service 工具基于开源的 XFire Java SOAP 引擎构建。XFire 是一款比较少见的而且非常流行的厂商独立的 SOAP 引擎。Codehaus XFire 是面向服务的开发更加简便，提供容易使用的 API 并支持标准。同时性能也非常高，因为使用了基于消耗内存很少的 StAX 模型进行开发。其功能和目标包括：

- 支持重要的 Web Service 标准 – SOAP，WSDL，WS-I Basic Profile,，WS-Addressing，WS-Security 等等
- 高性能的 SOAP 栈
- 可插入式的绑定机制支持 POJO，XMLBean，JAXB 1.1，JAXB 2.0 和 Castor
- 支持 JSR 181 API ，可以在 Java 5 and 1.4 下配置 Web 服务(Commons 属性，JSR 181 语法)
- 支持多种传输协议- HTTP, JMS, XMPP, In-JVM 等等
- 可嵌入的以及容易掌握的 API
- 支持 Spring, Pico, Plexus,Loom
- 支持 JBI（Java Business Integration，Java 业务集成）
- 客户端和服务器端骨架代码生成
- 支持 JAX-WS 体验版

。笔者在开发的过程中注意到它基本上是利用 Spring 的代理机制进行工作的，也就是先开发 POJO 的普通类，然后通过 Spring 生成代理类，最后通过 XFire 的核心 Servlet 将其暴露成一个 Web 服务。

15.1.3 MyEclipse 的 Web Service 工具简介

Web service 正在逐渐成为未来的跨平台跨系统跨网络，整合内外信息的 IT 解决方案（其实就是现在提出的 SOA）的核心技术。MyEclipse 的 web service 开发功能提供了快速开发测试 web service，以及轻量级的 web service 容器。这些核心的功能包括：

- 从上层向下和从底层开始的 web service 创建工具
- 用来测试 web service 的 Web Service Explorer（Web 服务浏览器）
- WSDL 创建，编辑和验证工具，包括 JSR-181 WS 标注（Web 服务标注）
- 创建 Web service 项目，配置和检验
- 向任何 Java servlet 2.4+ 容器发布 web service

未经许可 禁止打印传播

- 调试 Java web service 实现

15.2 基于代码方式创建 Web Service 项目

从 MyEclipse 5.0 起引入了一种新的项目名为 Web Services Project。这种项目对 MyEclipse Web Project 进行了扩展，来支持更多的概念包括 web services 配置，开发和发布。本节内容将会展示如何使用 MyEclipse Web Services Project 向导来创建和配置新的 web service 项目。我们将会进行下列步骤的操作：

- 创建 Web Services Project
- 在项目的 web.xml 文件中配置 XFire Servlet
- 创建 XFire 配置文件 services.xml
- 将 MyEclipse-XFire 类库添加到项目的构造路径中
- 在项目文件.project 中添加特殊的 MyEclipse web 项目构造器，这样能够发布项目时自动将文件 services.xml 发布到正确的位置，例如：<webroot>/WEB-INF/classes/META-INF/xfire/
- 创建 Web Service 客户端

15.2.1 系统需求

本章内容的练习仅需要 MyEclipse 6 即可，测试发布时需要 Tomcat 服务器。另外必须使用 1.5 或者更高版本的 JDK，并且项目的编译版本需要设置成 Java 5 或者更高。

15.2.2 创建 HelloWorldService 项目

首先要启动 Web Service Project 向导。该向导由三个页面组成，第一页设置 Web 项目配置的详细信息；第二页设置 XFire 的配置详细信息；第三页配置需要添加到项目构造路径中的 XFire 类库。

1. 启动 Web Services Project 向导

首先确保使用的透视图是 **MyEclipse Java Enterprise**，接着选择菜单 **File > New > Web Service Project**，稍后即可启动创建项目的向导对话框。注意不要选成创建 Web Project 的菜单项。

2. 在第一页完成 Web Project 信息的设置然后选择 **Next** 按钮继续

在弹出的对话框的 **Project Name** 中输入 *HelloWorldService*，然后选中 **J2EE Specification Level** 下面的 **Java EE 5.0** 单选钮，接着点击对话框底部的 Next 按钮进入下一页。此页设置如图 15.4 所示。

3. 在向导的第二页可以设置 XFire servlet 和 service.xml 配置文件的详细信息。 可以完全保持默认值然后点击 **Next** 按钮进入最后一页的设置即可。

XFire 这个 Servlet 专门用来根据 service.xml 中的配置来生成 WSDL 并提供 Web Service 服务功能，而且支持多种整合方式，例如在 Servlet class 下面有五种可以选择的：

a) org.codehaus.xfire.transport.http.XFireConfigurableServlet

b) org.codehaus.xfire.transport.http.XFireServlet

c) org.codehaus.xfire.loom.LoomXFireServlet

未经许可 禁止打印传播

d) org.codehaus.xfire.plexus.PlexusXFireServlet

e) org.codehaus.xfire.spring.XFireSpringServlet

。分别支持不同类型的 Web Service 整合策略，不过默认情况下是和 Spring 整合的。此页的设置如图 15.5 所示。同时在这一页还可以修改的有配置文件目录和配置文件名字。

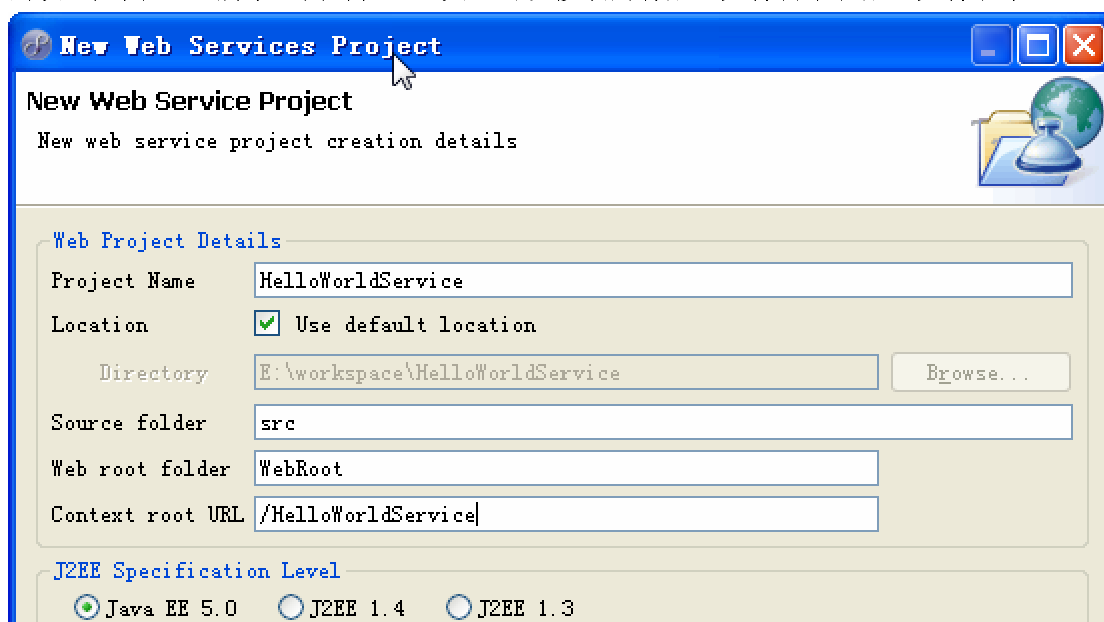


图 15.4 新建 Web Service 项目向导的第一页设置

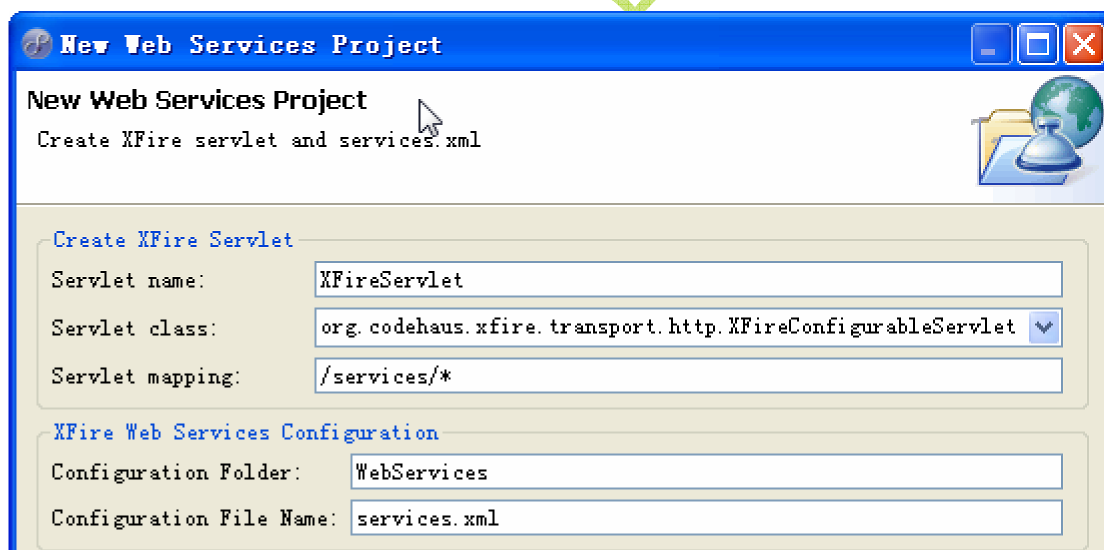


图 15.5 新建 Web Service 项目向导的第二页：XFire Servlet 和 service.xml

4. 向导的第三页是选择添加到项目构造路径中的类库。我们可以根据实际自己的需要来添加类库。不过默认情况下必须选中 XFire Core Library，除非你想自己下载 XFire 类库然后手工加入。如果计划开发一个使用 XFire 的客户端应用，那么还需要选中 XFire HTTP Client Libraries。

向导结束后这些类库将会添加到项目的构造路径中去，不会复制任何 JAR 文件到项目的目录中。按照图 15.6 那样进行设置，最后点击 Finish 按钮结束向导。

稍等片刻后，项目就创建完成了。图 15.7 展示了新创建好的 Web 服务项目的目录结构。

可以注意到这个项目和标准的 MyEclipse Web 项目的结构非常的像，只是额外加了些 XFire 的配置信息。

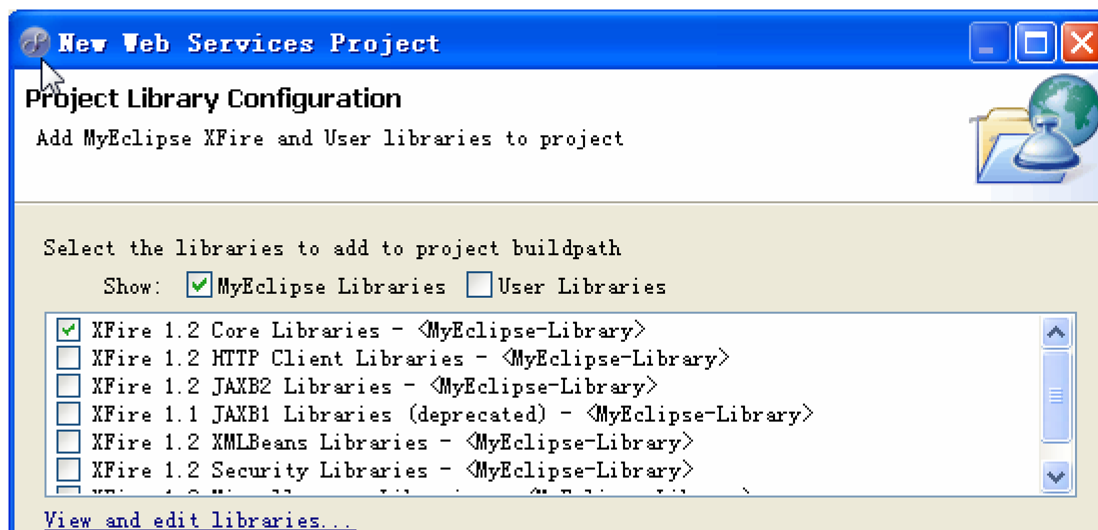


图 15.6 向导最后一页的设置信息

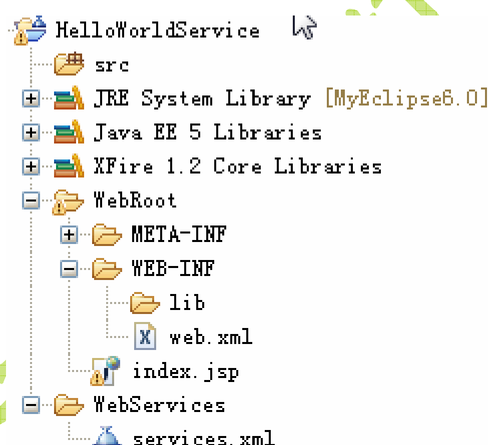


图 15.7 Web Service 项目的目录结构

接下来让我们看看 **web.xml** 的代码清单：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.5"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>XFireServlet</servlet-name>

<servlet-class>org.codehaus.xfire.spring.XFireSpringServlet</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>
```

。此文件定义了一个监听地址为*/services/**的 Web 服务监听地址，将来的 Web 服务都会发布到这个子目录下。再来看看 **services.xml** 的代码清单：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xfire.codehaus.org/config/1.0">

</beans>
```

。其实，这是一个和 Spring 的配置文件很相似的结构。

15.2.3 创建 Web Service

本小节将创建一个简单的 Web Service。首先，需要启动创建 Web Service 的向导。点击工具栏上的新建 Web Service 按钮，即可启动向导。或者选择菜单 **File > New > Other...**，在 **New** 对话框中展开 **MyEclipse > Web Services > Web Service**，之后也可以启动创建向导。启动的向导对话框第一页如图 15.8 所示。在这一页有两种开发模式，一种是从底到上的，也就是从 Java Bean 生成的方式；另一种则是从顶到下的，从 WSDL 文档生成的方式。那么本例中选中单选钮 **Create web service from Java bean** 并选中复选框 **Create new Java bean**，然后点击 **Next** 按钮进入第二页的设置。因为手工编写 WSDL 文件相当的困难，所以对于 Java 开发人员来说使用第一种方式比较简便。

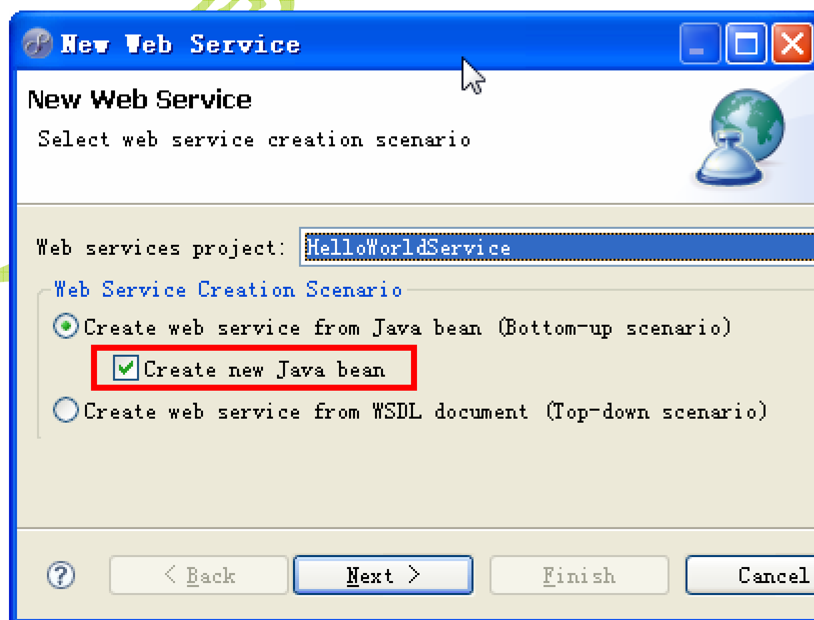


图 15.8 新建 Web Service 向导第一页

在第二页，用来设置 Web Service 的详细信息，这一页需要修改的地方。第一个是在

Web service name:处输入 *HelloWorld*，第二个地方是点击 **Java package** 一行末尾的 **New...**按钮，在弹出的新建包对话框中输入 *hellows*，然后点击 **OK** 按钮返回（也可以直接输入现有的包名）。这一页的其它地方包括设置 Java 源代码目录（Java source folder），以及服务接口名（Service interface，会自动根据输入的服务名生成），服务实现类（Service impl. class），还有 SOAP 的风格（SOAP style/use）以及 Servlet 的作用域（Servlet scope，可用值包括 application，request，session）。当一切设置完毕后，点击 **Finish** 按钮即可完成向导并生成代码。

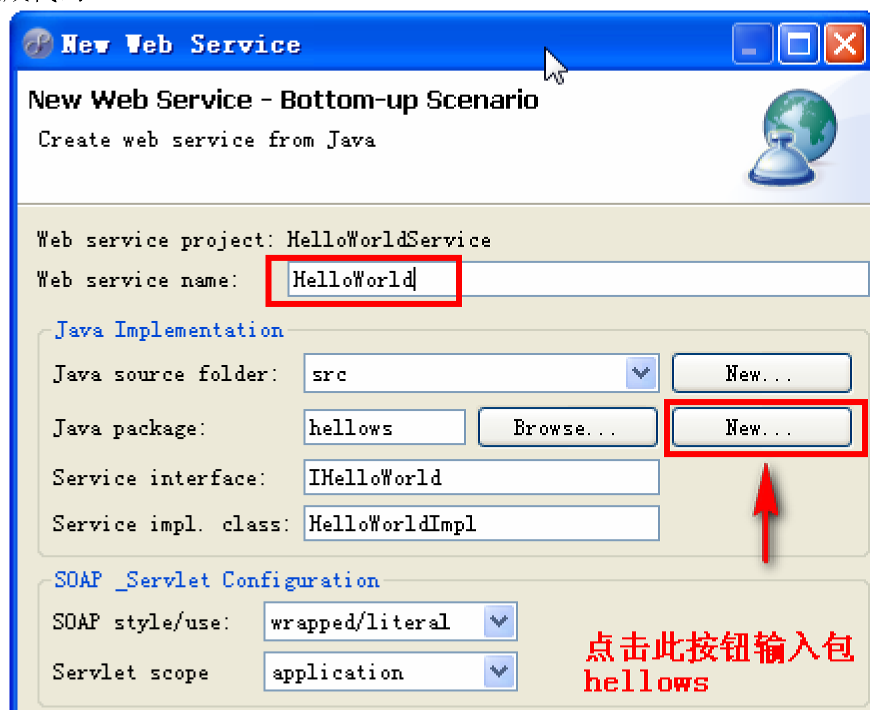


图 15.9 新建 Web 服务向导第二页

此时我们看看所发生的变化，首先是 services.xml 发生了改变：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xfire.codehaus.org/config/1.0">
    <service>
        <name>HelloWorld</name>
        <serviceClass>hellows.IHelloWorld</serviceClass>
        <implementationClass>
            hellows.HelloWorldImpl
        </implementationClass>
        <style>wrapped</style>
        <use>literal</use>
        <scope>application</scope>
    </service>
</beans>
```

。这就给了我们一种提示，那就是我们可以自己手工修改这个文件来加入更多的 Web Service，无非就是需要一个接口和一个实现类，然后在此配置文件中加入更多的 service 标签即可。再来看看在 *src/hellows* 目录下多了两个 Java 类，分别是 *IHelloWorld* 和 *HelloWorld*。源代码非常简单，先看接口 **IHelloWorld**：

```java
package hellows;
//Generated by MyEclipse

public interface IHelloWorld {

    public String example(String message);

}
```

。此接口定义了一个参数为 message，返回值为 String，名为 example 的方法，这个方法就是最终要对外暴露的 Web 服务，可以根据情况修改为其它功能，例如计算加法：
public double add(double a, double b);
也可以是不带参数和返回值的方法。而类 HelloWorld 则定义了其实现：

```java
package hellows;
//Generated by MyEclipse

public class HelloWorldImpl implements IHelloWorld {

    public String example(String message) {
        return message;
    }

}
```

默认的实现很简单，就是把参数返回即可。实际上到这里位置 Web Service 服务端的开发已经完成了，不过，我们还是把它稍微调整一下，修改下方法的实现代码：

```java
    public String example(String message) {
        System.out.println(this);
        return "你好，这是我的第一个 Web Service，你输入的消息是:" +
        message;
    }
```

。这样就有两个目的，第一是能分辨出传入的参数和传出的参数；第二呢，是在控制台上打印一行信息，便于跟踪服务器端所发生的事情。

15.2.4 发布运行项目，显示 WSDL

现在，我们可以将这个应用发布到服务器上，或者在 **Package Explorer** 视图中选中项目节点 *HelloWorldService*，然后选择菜单 **Run > Run As > 3 MyEclipse Server Application**，之后 MyEclipse 可能会显示一个可用的服务器列表，选中其中的服务器之一例如 *MyEclipse Tomcat* 并点击 **OK** 按钮后，项目就会自动发布，对应的服务器会启动。如果读者发现个别时候发布项目后报错，打不开，那可以把项目删除掉再重来一次编写，发布就好了。

那么Web Service是不能直接测试的，我们要用下一小节所提到的Web Service浏览器或者是编写一个客户端来进行测试。这个 Web 服务监听的地址是在：http://localhost:8080/HelloWorldService/services/HelloWorld ，当我们企图直接访问时，将会得到一段报错信息：

未经许可 禁止打印传播

Invalid SOAP request.

即：无效的 SOAP 请求。然而，按照 Web 服务的规范，所有的 Web 服务都必须提供 WSDL 描述，那么这里的 Web 服务也支持，只需要加入?wsdl 参数就可以了。访问如下地址：http://localhost:8080/HelloWorldService/services/HelloWorld?wsdl ，得到的输出如下（可以看出WSDL的确不是给人看的）：

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://hellows" xmlns:tns="http://hellows"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://hellows">
<xsd:element name="example">
<xsd:complexType>
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="in0" nillable="true"
type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="exampleResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
  </wsdl:types>
  <wsdl:message name="exampleRequest">
    <wsdl:part name="parameters" element="tns:example">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="exampleResponse">
    <wsdl:part name="parameters" element="tns:exampleResponse">
    </wsdl:part>
  </wsdl:message>
```

未经许可 禁止打印传播

```
  <wsdl:portType name="HelloWorldPortType">
    <wsdl:operation name="example">
      <wsdl:input name="exampleRequest" message="tns:exampleRequest">
    </wsdl:input>
      <wsdl:output name="exampleResponse" message="tns:exampleResponse">
    </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HelloWorldHttpBinding" type="tns:HelloWorldPortType">
    <wsdlsoap:binding                                    style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="example">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="exampleRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="exampleResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HelloWorld">
    <wsdl:port name="HelloWorldHttpPort" binding="tns:HelloWorldHttpBinding">
      <wsdlsoap:address
location="http://localhost:8080/HelloWorldService/services/HelloWorld"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

。这充分说了目前的项目已经运行成功。

15.2.5 用 Web Service Explorer 测试

MyEclipse 提供了一个名为 Web Services Explorer 的工具，可以专门用来对 Web Service 进行测试。Web Services Explorer 可以操作任何提供了 WSDL 描述文档的 web 服务。点击工具栏上的按钮即可启动它，如图 15.10 所示。稍等片刻，就可以看到启动的 Web Services Explorer，其默认显示的内容是 UDDI Main，必须点击工具栏上的 WSDL Page 按钮才能切换到 WSDL Main 栏目，如图 15.11 所示。

图 15.10 工具栏上的启动 Web Services Explorer 按钮

未经许可 禁止打印传播

此时，可以看到页面（其实这是个JSP的页面）中出现两栏，包括Navigator和Actions，那么在Navigator中点击链接WSDL Main，之后在右侧的Actions栏目中将会显示Open WSDL，此时在文本输入框中输入我们之前所看到的WSDL地址：http://localhost:8080/HelloWorldService/services/HelloWorld?wsdl，然后点击**Go**按钮即可。

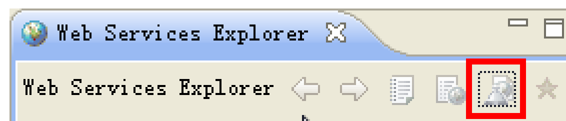


图 15.11 点击 WSDL Page 按钮切换页面

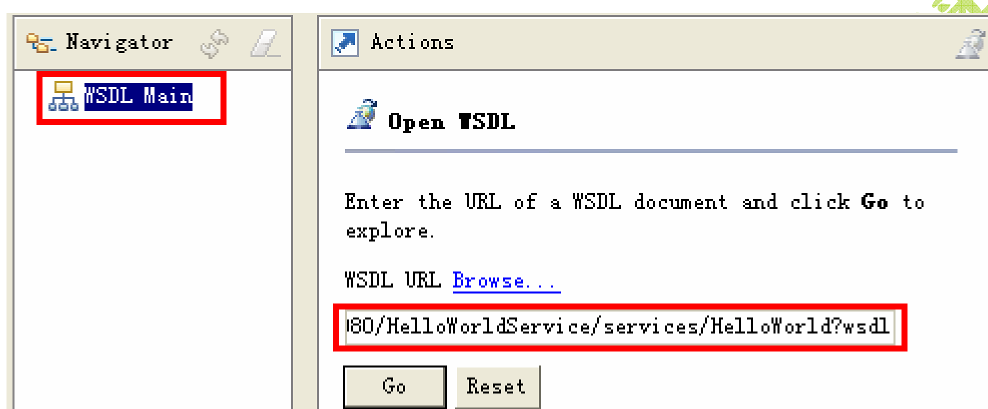


图 15.12 输入要浏览的 WSDL 地址

点击了 Go 按钮之后，Navigator 中将会分析 WSDL 文档的内容，并探测出其中可以被调用的操作（函数，方法），并以树状结构列出。此时我们可以展开左侧节点，选中 example，这时在 Actions 一栏中将会出现 Invoke a WSDL Operation（调用 WSDL 操作）的说明，

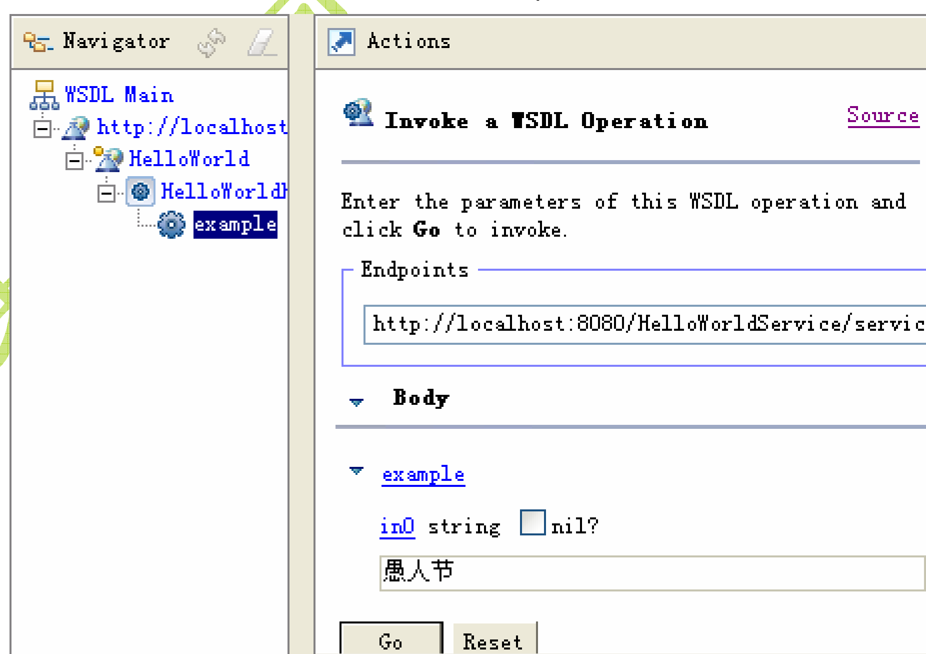


图 15.13 准备调用 example 函数

在输入框中输入参数"愚人节"后，点击 Go 按钮即可完成调用过程。返回值在哪里呢？返回值位于 Actions 下方的 Status 一栏中，可以看到回应的消息，如图 15.14 中上侧图片所示。如果读者对调用的过程中传递的 SOAP 消息内容感兴趣，可以点击链接 Source，即可得到如图 15.14 下方所示的 SOAP 消息内容，上方显示的是 SOAP 请求信封（SOAP Request Envelope），下方显示的则是 SOAP 响应信封（SOAP Response Envelope），其内容都是 XML，如何编写则由 SOAP 协议所规定。这一过程证明了我们的 Web 服务工作是正常的，同时在控制台也会打印出一行信息：hellows.HelloWorldImpl@134263a，这说明调用的的确是我们所编写的类中的方法。那么在我们的测试例子中，Web Service 的执行过程如下所示：

客户端 -> 生成 SOAP 请求信封 ->访问地址 /HelloWorldService/services/HelloWorld 所对应的 Servlet -> 得到 SOAP 响应信封 -> 解析 -> 打印结果。

图 15.14 Web Service 的测试调用结果

15.2.6 创建 Java 测试客户端

本节将会讨论如何使用 XFire 内置的类库来开发一个客户端。XFire 提供了一套动态代理框架，能够读取 WSDL 文档，并根据此创建底层的消息服务，从而能够根据一个 Java 类来执行 web 服务上的功能。本节内容简要介绍如何开发一个 Java 类作为客户端来访问刚刚创建好的 HelloWorld web service。

对于这个项目来说，我们需要在 *HelloWorldService* 这个项目中创建一个类。首先要做的一步就是将类库 *XFire HTTP Client Libraries* 添加到项目的构造路径中去。首先在 **Package Explore** 视图中，右键点击项目根节点，然后选中菜单 **Build Path > Add Library**，接着在弹出的 Add Library 对话框中，选中列表项中的 *MyEclipse Libraries*，然后

点击 **Next** 按钮进入下一页，选中此页列表中的 *XFire 1.2 HTTP Client Libraries*，如图 15.15 所示。最后点击 **Finish** 按钮结束添加类库的过程。

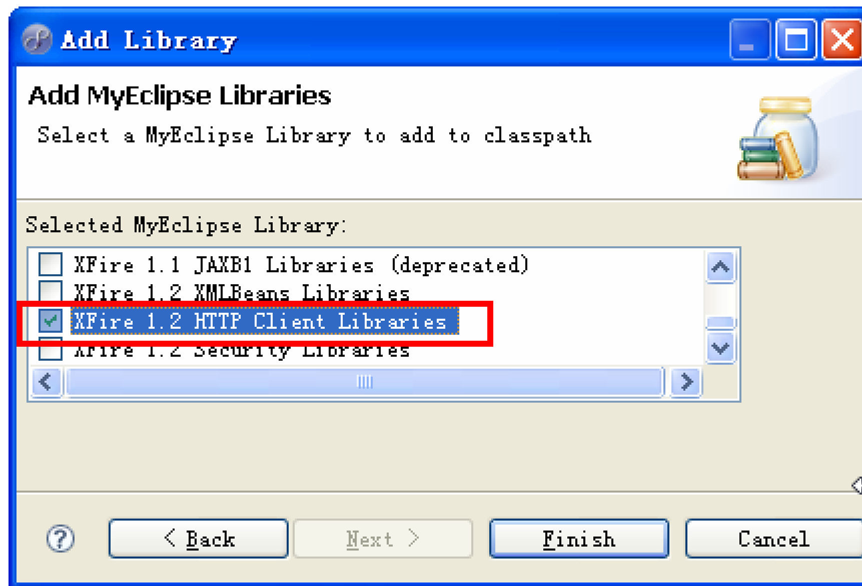


图 15.15 选中 XFire HTTP Client Libraries

随后我们就可以新建客户端类来进行测试了，创建一个类 **client.HelloWSClient**，源代码清单如下：

```
package wsclient;
import java.net.MalformedURLException;

import hellows.IHelloWorld;

import org.codehaus.xfire.XFireFactory;
import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.*;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

public class HelloWSClient {

    public static void main(String[] args) {
        Service srvcModel = new ObjectServiceFactory()
                .create(IHelloWorld.class);
        XFireProxyFactory factory = new XFireProxyFactory(XFireFactory
                .newInstance().getXFire());
        String helloWorldURL =
"http://localhost:8080/HelloWorldService/services/HelloWorld";
        try {
            IHelloWorld srvc = (IHelloWorld) factory.create(
                    srvcModel, helloWorldURL);
            String result = srvc.example("hello world Java 客户端测试");
```


未经许可 禁止打印传播

```
            System.out.print(result);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

}
```

在这段代码中定义了一个 XFire web service 代理，支持 POJO 的接口 IHelloWorldService 。一旦代理创建完成后，并把类型转成 IHelloWorldService 之后，就可以像一个普通的 Java 类那样来调用了，而后台的工作都是代理类自动完成的，也就是说：客户端只需要 XFire 客户端类库，Web Service 地址以及 IHelloWorldService 这个接口就可以完成对 Web Service 的调用过程，当然，这一过程是仅限于 Java 语言作为客户端和服务器端的情况下的。在下一节我们会讨论跨平台的访问其它语言所提供的 Web 服务然后根据 WSDL 生成客户端的开发模式。OK，现在运行这个类，可以得到预期的结果：

你好，这是我的第一个 Web Service，你输入的消息是:hello world Java 客户端测试

。

15.2.7 给现有 Web 项目加入 Web Service 开发功能

除了直接创建 Web Service 项目之外，还有一种方式是给现有的 Web 项目加入 Web Service 开发功能，使之变为一个 Web 服务项目。做法非常简单，先打开 Web 项目，在 **Package Explorer** 视图中选中项目节点，然后选择菜单 **MyEclipse > Project Capabilities > Add Web Service Capabilities ...** 来启动 **Add Web Service Capabilities** 向导，剩下的过程就和以前一样了。不过，也有些许地方需要注意，因为 MyEclipse 带的 XFire 类库实际上包含了 Spring 的类库，如图 15.16 所示的就是项目发布后 WEB-INF/lib 下的文件列表，所以开发的时候一定要避免出现类库冲突的情况。

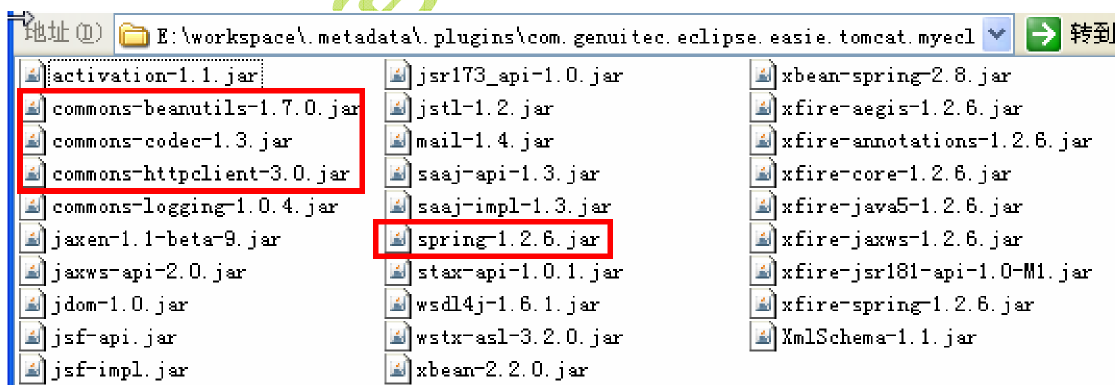


图 15.16 发布后的类库结构

15.3 创建单独的客户端项目

在实际的开发中，Web Service最大的用处也许就是整合异构系统，换句话说就是不同网络，不同语言（例如Delph，.NET，Java，C＋＋）开发的服务系统，而且调用者不需要知道服务器到底是用何种技术实现的，只需要服务地址和WSDL即可，如果再对Web 服务

加上调度，安装，卸载和连接等功能，就成了现在炒的很火的SOA技术的思想了。而MyEclipse支持生成这样的客户端代码，使开发人员能比较方便的完成整合任务。我们今天要调用的，就是一个国人提供的天气预报Web Service（需要提醒的是某些网站的Web Service并不符合标准，无法生成代码，因此试了两个，第二个才成功），地址是 http://www.ws365.net/ws/weather.asmx?WSDL。

15.3.1 创建天气预报客户端 WeatherWSClient 项目

第一步我们来创建一个普通的 Java 项目，名为 *WeatherWSClient*。首先确保已经打开了 **MyEclipse Java Enterprise** 透视图，然后进行下列操作：

1. 从 MyEclipse 菜单栏选择 **File > New > Java Project**，接着会打开 **New Java Project** 向导；
2. 输入 *WeatherWSClient* 到 **Project name**；
3. 在 **Project Layout** 下选中 **Create separate source and output folders** 单选钮；
4. 点击 **Finish** 按钮关闭对话框。

这样 Java 项目就建立完毕了。需要指出的是，我们当然能在普通 Web 项目中生成 Web Service 客户端然后调用，例如用到的天气预报，最好是可以整合进我们自己的站点中去。具体就是在 JSP 页面调用生成的客户端类，然后调用方法，将输出值输出到 JSP 页面中即可。

15.3.2 创建 Web Service Client,从 WSDL 生成客户端代码

选择菜单 **File > New > Other...**，在 **New** 对话框中展开 **MyEclipse > Web Services > Web Service Client**，来启动创建 Web 服务客户端的向导，如图 15.17 所示。

禁止打印传播

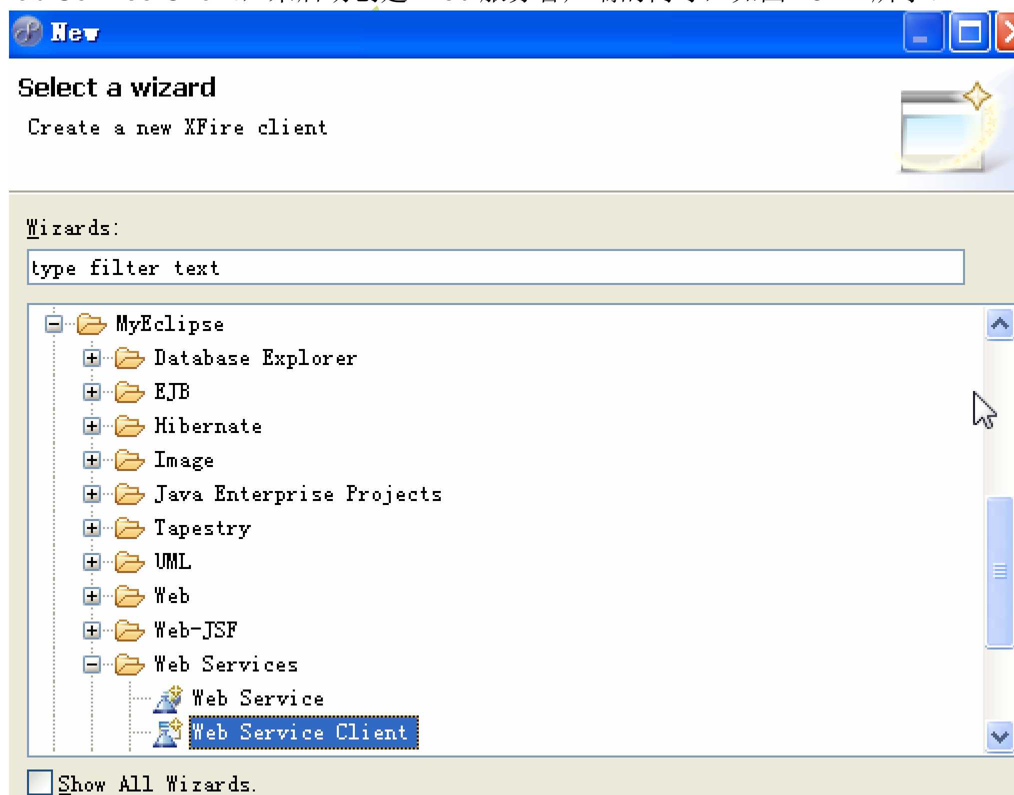


图 15.17 选中新建 Web Service Client 项

接着点击 Next 按钮进入第一页，并按照图 15.18 的内容进行设置。在这一页有三个地方需要进行设置。首先在 **Web service project:**处选择 *WeatherWSClient*(不要不小心选到了别的项目中)；其次要在 **Service Difination**（服务定义）处点击单选钮 *WSDL URL*，并在右侧输入我们希望生成的 Web Service 的 WSDL 的互联网 URL 地址(单选钮 WSDL File 是让我们可以选中本机上的 WSDL 文件），在本例中自然就是 *http://www.ws365.net/ws/weather.asmx?WSDL*，也可以输入我们上一节所开发的 Web Service 项目的 WSDL 地址：*http://localhost:8080/HelloWorldService/services/HelloWorld?wsdl*；最后一个地方是点击 **Java package** 一行末尾的 **New...**按钮，在弹出的新建包对话框中输入 *wsclient*，然后点击 **OK** 按钮返回（也可以直接输入现有的包名），当然也可以保持包为空，不过一半不建议这样做。需要注意的是这个项目需要 Java 5 来支持 JSR-181 所规定的 Web 服务标注。

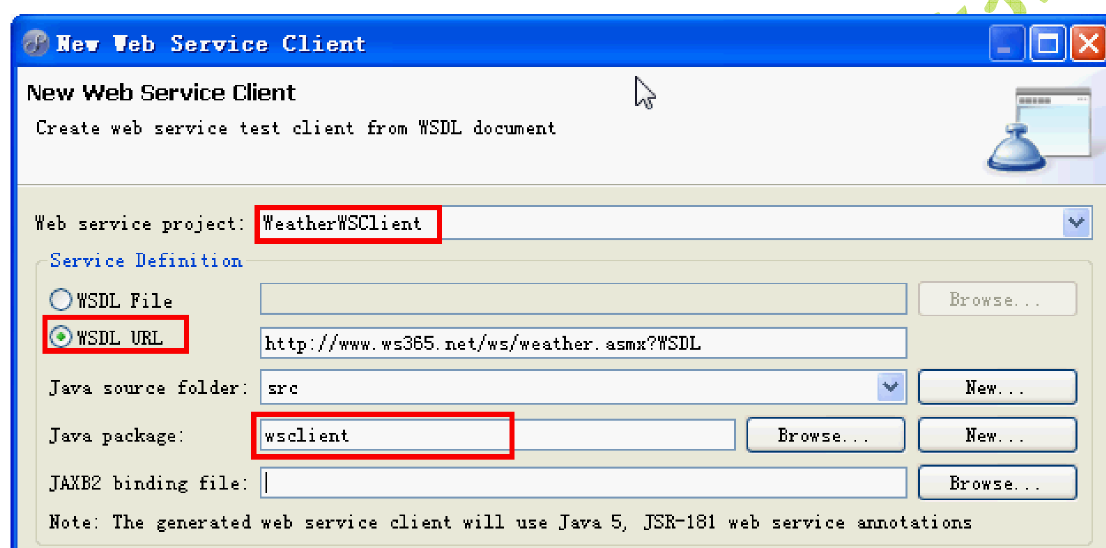


图 15.18 新建 Web 服务客户端设置

设置完毕后点击 Next 按钮进入向导的下一页，同时 MyEclipse 将会尝试对 WSDL 文件的有效性进行验证，如图 15.19 所示。如果 WSDL 文件验证失败，那么生成客户端代码的时候有可能会失败。很不幸，我挑了两个 WSDL 都有错误，不过目前使用的这个虽然有错，缺不影响代码的生成。忽略它然后点击 **Next** 按钮进入最后一页的设置。

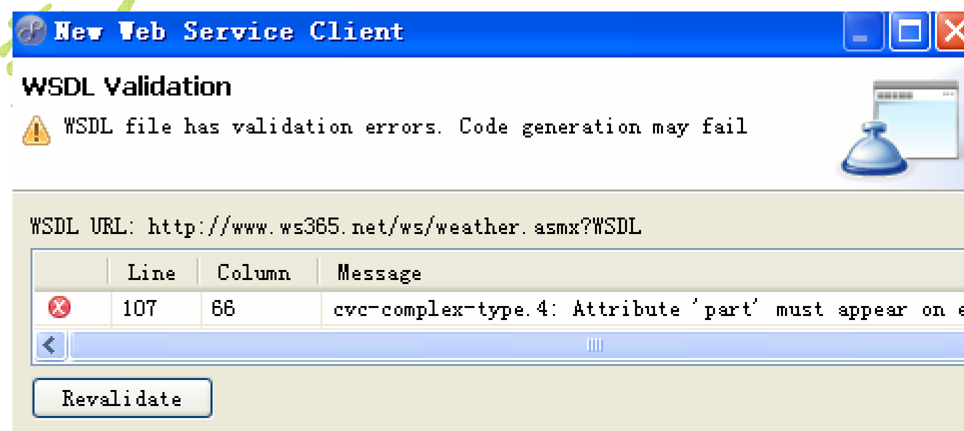


图 15.19 验证 WSDL

在最后一页需要选中所需的 MyEclipse XFire 类库，包括 XFire 核心类库，HTTP Client 库，以及 JAXB2。在本页保持默认选项，最后点击 **Finish** 按钮结束整个向导，开始生成代码过程即可。一般来说没有什么错误的话，代码就会生成，目录结构如图 15.21 所示。此时我们只需要调用 wsclient 包下面的相关类，就可以完成 Web 服务的调用任务。

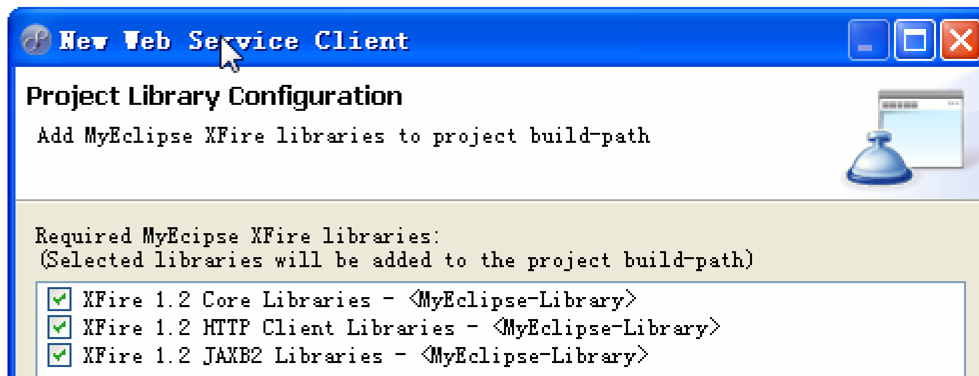


图 15.20 添加相关类库

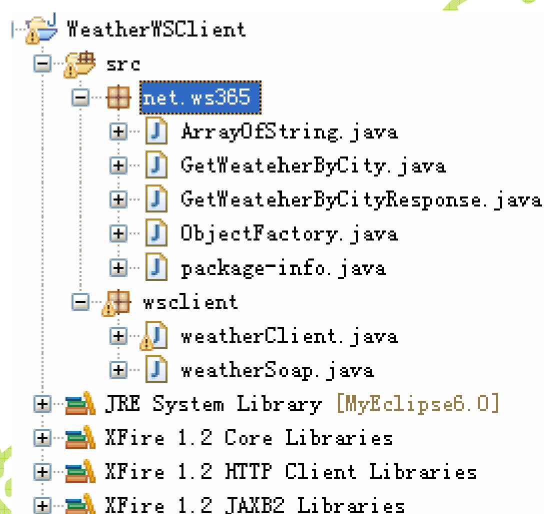


图 15.21 生成代码的目录结构

net.ws365 包下面的代码，是一些和 Web 服务相关的类，其中 ArrayOfString 是一个字符串列表类，这个类是 Web 服务的一个标准类，可以在字符串和 List 之间转换。而真正可以调用的客户端代码则集中在 wsclient 包里面，确切的说只需要调用 *weatherClient* 这个类就可以了。

15.3.3 编写运行测试代码

本节我们就要用生成的类库来调用 Web 服务然后得到北京市的天气。首先让我们来打开 wsclient.weatherClient 类，并在 Outline 视图中定位到方法 main()，代码片段如下所示：

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

```
        weatherClient client = new weatherClient();

        //create a default service endpoint
        weatherSoap service = client.getweatherSoap();

        //TODO: Add custom client code here
            //
            //service.yourServiceOperationHere();

        System.out.println("test client completed");
            System.exit(0);
    }
```

这段代码已经给我们展示了该如何调用，同时还有 TODO 标记指明只要调用 service 变量上的方法即可获得结果，而哪些方法可以调用则需要我们来看 weatherSoap 类的方法定义：

```
package wsclient;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import net.ws365.ArrayOfString;

@WebService(name = "weatherSoap", targetNamespace =
"http://www.ws365.net/")
@SOAPBinding(use = SOAPBinding.Use.LITERAL, parameterStyle =
SOAPBinding.ParameterStyle.WRAPPED)
public interface weatherSoap {

    @WebMethod(operationName = "GetWeateherByCity", action =
"http://www.ws365.net/GetWeateherByCity")
    @WebResult(name = "GetWeateherByCityResult", targetNamespace =
"http://www.ws365.net/")
    public ArrayOfString getWeateherByCity(
            @WebParam(name = "city", targetNamespace =
"http://www.ws365.net/")
            String city);

}
```

可以看到只有一个方法可以调用：public ArrayOfString getWeateherByCity(String city)，这样，我们就可以修改上面的代码片段完成调用过程，为了方便读者对比，这里列出完整的类 **weatherClient** 的代码清单，并以粗斜体显示改动部分：

未经许可 禁止打印传播

```
package wsclient;

import java.net.MalformedURLException;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;

import javax.xml.namespace.QName;

import net.ws365.ArrayOfString;

import org.codehaus.xfire.XFireRuntimeException;
import org.codehaus.xfire.aegis.AegisBindingProvider;
import org.codehaus.xfire.annotations.AnnotationServiceFactory;
import org.codehaus.xfire.annotations.jsr181.Jsr181WebAnnotations;
import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.jaxb2.JaxbTypeRegistry;
import org.codehaus.xfire.service.Endpoint;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.soap.AbstractSoapBinding;
import org.codehaus.xfire.transport.TransportManager;

public class weatherClient {

    private static XFireProxyFactory proxyFactory = new
XFireProxyFactory();
    private HashMap endpoints = new HashMap();
    private Service service1;

    public weatherClient() {
        create1();
        Endpoint weatherSoapLocalEndpointEP = service1 .addEndpoint(new
QName("http://www.ws365.net/", "weatherSoapLocalEndpoint"), new
QName("http://www.ws365.net/", "weatherSoapLocalBinding"),
"xfire.local://weather");
        endpoints.put(new QName("http://www.ws365.net/",
"weatherSoapLocalEndpoint"), weatherSoapLocalEndpointEP);
        Endpoint weatherSoapEP = service1 .addEndpoint(new
QName("http://www.ws365.net/", "weatherSoap"), new
QName("http://www.ws365.net/", "weatherSoap"),
"http://www.ws365.net/ws/weather.asmx");
        endpoints.put(new QName("http://www.ws365.net/",
"weatherSoap"), weatherSoapEP);
```

未经许可 禁止打印传播

```
    }

    public Object getEndpoint(Endpoint endpoint) {
        try {
            return proxyFactory.create((endpoint).getBinding(),
(endpoint).getUrl());
        } catch (MalformedURLException e) {
            throw new XFireRuntimeException("Invalid URL", e);
        }
    }

    public Object getEndpoint(QName name) {
        Endpoint endpoint = ((Endpoint) endpoints.get((name)));
        if ((endpoint) == null) {
            throw new IllegalStateException("No such endpoint!");
        }
        return getEndpoint((endpoint));
    }

    public Collection getEndpoints() {
        return endpoints.values();
    }

    private void create1() {
        TransportManager tm =
(org.codehaus.xfire.XFireFactory.newInstance().getXFire().getTranspor
tManager());
        HashMap props = new HashMap();
        props.put("annotations.allow.interface", true);
        AnnotationServiceFactory asf = new AnnotationServiceFactory(new
Jsr181WebAnnotations(), tm, new AegisBindingProvider(new
JaxbTypeRegistry()));
        asf.setBindingCreationEnabled(false);
        service1 = asf.create((wsclient.weatherSoap.class), props);
        {
            AbstractSoapBinding soapBinding =
asf.createSoap11Binding(service1, new QName("http://www.ws365.net/",
"weatherSoapLocalBinding"), "urn:xfire:transport:local");
        }
        {
            AbstractSoapBinding soapBinding =
asf.createSoap11Binding(service1, new QName("http://www.ws365.net/",
"weatherSoap"), "http://schemas.xmlsoap.org/soap/http");
        }
```

未经许可 禁止打印传播

```
    }

    public weatherSoap getweatherSoapLocalEndpoint() {
        return ((weatherSoap)(this).getEndpoint(new
QName("http://www.ws365.net/", "weatherSoapLocalEndpoint")));
    }

    public weatherSoap getweatherSoapLocalEndpoint(String url) {
        weatherSoap var = getweatherSoapLocalEndpoint();
        org.codehaus.xfire.client.Client.getInstance(var).setUrl(url);
        return var;
    }

    public weatherSoap getweatherSoap() {
        return ((weatherSoap)(this).getEndpoint(new
QName("http://www.ws365.net/", "weatherSoap")));
    }

    public weatherSoap getweatherSoap(String url) {
        weatherSoap var = getweatherSoap();
        org.codehaus.xfire.client.Client.getInstance(var).setUrl(url);
        return var;
    }

    public static void main(String[] args) {


        weatherClient client = new weatherClient();

        //create a default service endpoint
        weatherSoap service = client.getweatherSoap();

        //TODO: Add custom client code here
            //
            //service.yourServiceOperationHere();
        ArrayOfString result = service.getWeateherByCity("北京");
        if(result != null) {
         List<String> weathers = result.getString();

         for(int i = 0; i < weathers.size(); i++) {
             System.out.println(weathers.get(i));
         }
        }
```

未经许可 禁止打印传播

```
        System.out.println("test client completed");
            System.exit(0);
    }

}
```

主要的修改依然集中在 main 方法中，我们传递了参数“北京”，但是返回值却是一个 ArrayOfString，因此判断如果有返回值的话，就将其内容转换为一个 List，每一项都是 String，然后将结果打印出来。好了，执行这个类，将会在 Console 视图中看到输出：

```
本web服务由www.ws365.net免费提供;请访问:http://www.ws365.net 获取更多web服务
4月02日-4月03日
北京
http://weather.tq121.com.cn/images/a0.gif
http://weather.tq121.com.cn/images/00.gif
晴
18℃~3℃
北风3-4级转微风
紫外线：中等
test client completed
```

非常的好，读者可以将城市更改成别的地方，例如上海，然后再次运行，自然可以得到相应的结果。

15.4 JSR 181 标注方式 Web 服务开发

XFire 支持 JSR 181 标注方式的 Web 服务开发，另外，所有支持 Java EE 5 的应用服务器（Tomcat 只是 Web 层的，本身不支持 EJB 和 Web 服务开发），都支持这种开发方式，例如：JBoss，GlassFish，WebLogic10 等等，本节就简要讨论使用 XFire 开发标注式 Web 服务和用 JBoss 开发的过程。

未经许可 禁止打印传播

15.4.1 XFire 的标注服务开发

XFire 支持 JRS181 方式的标注服务开发，这样开发的时候只需要编写一个普通的 Java 类，然后加上标注信息后，加入 services.xml 中即可。我们仍然在第 2 节所开发的 *HelloWorldService* 项目中进行修改。下面是创建的 Web 服务类 **echo.Jsr181EchoService**：

```
package echo;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

@WebService(name = "EchoService", serviceName="EchoServiceTest",
targetNamespace = "http://www.openuri.org/2004/04/HelloWorld")
```

```
        public class Jsr181EchoService
{
    @WebMethod(operationName = "echoString", action = "urn:EchoString")
    @WebResult(name = "echoResult")
    public String echo(@WebParam(name = "echoParam", header = true) String
input)
    {
        return input;
    }
}
```

这个代码中有很多标注，绝大部分都可以在最后生成的 WSDL 文档中找到对应值。

@WebService 这个标注放置在 Java 类之前，注明这个类的部分方法可以被发布为 Web 服务（还记得上一章提到的标注嘛？这个标注最终被 XFire 读取后进行分析后会进一步处理成 Web 服务）。它的属性用于设置 Web 服务被发布时的配置信息，常用的属性包括：

name（可选）：Web 服务的名字，WSDL 中 wsdl:portType 元素的 name 属性和它保持一致，默认是 Java 类或者接口的名字，也可以进行自定义，例如本例中的 *EchoService*。

serviceName（可选）：Web 服务的服务名，WSDL 中 wsdl:service 元素的 name 属性和它保持一致，默认是 Java 类的名字（*Jsr181EchoService*），不过如果设置了 name 属性，则名字改为 name 属性的取值。

targetNamespace（可选）：WSDL 文件所使用的 namespace，该 Web 服务中所产生的其他 XML 文档同样采用这个作为 namespace，一般取值为 Web 服务所在网站的名字，不过看起来任意取值并无出错之处。

@WebMethod（可选）标注放在需要发布成 Web 服务的方法之前，有一些属性可以设置。例如 **openrationName** 指明了 SOAP 调用时所看到的方法名为 echoString，而不是类中的方法名 echo，**action** 则定义了操作的类型。一个类里面可以定义多个@WebMethod。

@WebResult（可选）标注定义了返回值（SOAP Response Envolope）中的 **name**(名字)为 echoResult。

@WebParam（可选）则定义了哪些参数可以作为 Web 服务中的远程可见的参数被调用，**name** 设置了其属性。

乍看之下，这个 Web 服务中所用的标注有点多，实际上，这些标注可以不加任何属性，例如只写下@WebService，@WebMethod 即可，甚至于整个类只需要一个@WebService 标注即可，此时代码如下所示：

```
package echo;

import javax.jws.WebService;

@WebService
public class Jsr181EchoService {
    public String echo(
    String input) {
        return input;
    }
```

未经许可 禁止打印传播

```
}
```

。此时最后所生成的 Web 服务中，所有的操作名，方法名和参数名都和此通 Java 类中的名称一致。在这种情况下，该 Web 服务的访问地址应为：http://localhost:8080/HelloWorldService/services/Jsr181EchoService?wsdl 。

最后一步，乃是在 XFire 中配置并发布此服务了，在 **services.xml** 中加入的 Web 服务配置格式如下：

```
    <service>
        <!-- 如果配置文件中配置了额外的name属性，那么最终的Web Service 名字将会以此处为准，即：
            ServiceName?wsdl；同样的namespace的取值也会覆盖Java类中的标注的值。
            <name>ServiceName</name>
            <namespace>http://www.un.gov/HelloEcho</namespace>
        -->
        <serviceClass>echo.Jsr181EchoService</serviceClass>
        <serviceFactory>
            org.codehaus.xfire.annotations.AnnotationServiceFactory
        </serviceFactory>
    </service>
```

，此配置相当的简单，需要注意的是 serviceClass 就是我们写有 Web 服务标注的 Java 类，而 serviceFactory 则必须是 AnnotationServiceFactory（标注服务工厂），否则此 Web 服务将无法正常发布。

好了，现在按照 15.2.4 一节的内容发布项目并运行后，即可在浏览器中键入下面地址进行测试了：http://localhost:8080/HelloWorldService/services/EchoServiceTest?wsdl 。**EchoServiceTest** 的名字是因为在类中标注了@WebService 的 serviceName 属性。之后就可以在 Web Service Explorer 中对它进行测试，或者是生成客户端代码。

从这里看到这种开发方式相对也简单的多，而且您将在下一节看到如果是 Java EE 5 的容器，开发过程将更加的简单，而且不需要 XFire 类库及其配置文件。

15.4.2 JBoss 下的标注服务开发

任何支持 Java EE 5 的服务器都支持标注方式的 Web 服务开发，其基本过程如下：

1. 编写带有 Web 服务标注的普通 Java 类；
2. 在 web.xml 中将其作为 Servlet 定义；
3. 发布项目（有的服务器也许需要打包为 war 格式后才能发布）。

JBoss 服务器的配置方式参考第六章 管理应用服务器。

首先让我们创建一个普通的 Web 项目，名字为 *JBossWS*，不需要添加额外类库，注意选中 **J2EE Specification Level** 下面的 **Java EE 5.0** 单选钮。然后新建一个带有 Web 服务标注的 Java 类：webservice. HelloWorldService，代码清单如下：

```
package webservice;


import javax.jws.WebMethod;
import javax.jws.WebService;
```

未经许可 禁止打印传播

```java
import javax.jws.soap.SOAPBinding;

@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class HelloWorldService {
	@WebMethod
	public String hello(String name) {
		return "欢迎您访问 JBoss Web 服务: " + name;
	}
}
```

此代码和上一小节的内容没有本质的不同，唯一是多了个**@SOAPBinding** 标注，表示这个服务可以映射到一个 SOAP 消息中(其实默认就是 SOAP 方式)，style 属性用于指定 SOAP 消息请求和回应的编码方式。此类最终会被服务器当作 Servlet 来处理。

接下来修改 web.xml 加入 Servlet 映射定义，代码清单如下所示：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <servlet>
        <servlet-name>HelloWorldService</servlet-name>
        <servlet-class>webservice.HelloWorldService</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorldService</servlet-name>
        <url-pattern>/HelloWorldService/*</url-pattern>
    </servlet-mapping>

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

。此时项目的目录结构如图 15.22 所示，没有任何特殊之处。

未经许可 禁止打印传播

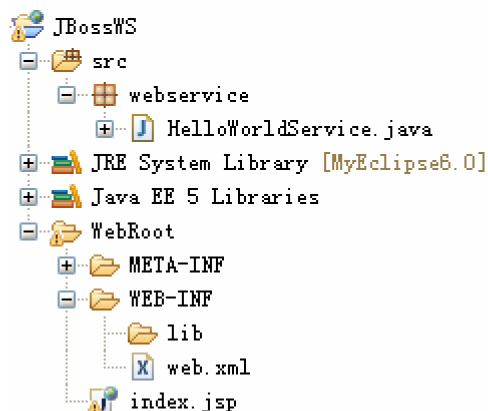


图 15.22 JBoss 下的 Web 服务项目目录

到此为止，此 Web 服务已经开发完毕了，我们可以将这个应用发布到服务器上，或者在 **Package Explorer** 视图中选中项目节点 *JBossWS*，然后选择菜单 **Run > Run As > 3 MyEclipse Server Application**，之后 MyEclipse 可能会显示一个可用的服务器列表，选中其中的 JBoss 服务器：*JBoss 4.x* 并点击 **OK** 按钮后，项目就会自动发布，对应的服务器会启动。

此时我们可以键入地址 http://localhost:8080/JBossWS/HelloWorldService/?wsdl 即可看到此Web服务，或者使用JBoss提供的Web服务控制台来查看。在浏览器键入地址 http://localhost:8080/jbossws/ 即可看到Web服务首页面，点击页面中的View a list of deployed services链接，即可看到发布的Web服务列表，包括了我们刚刚开发的HelloWorldService。此过程如图 15.23 所示。

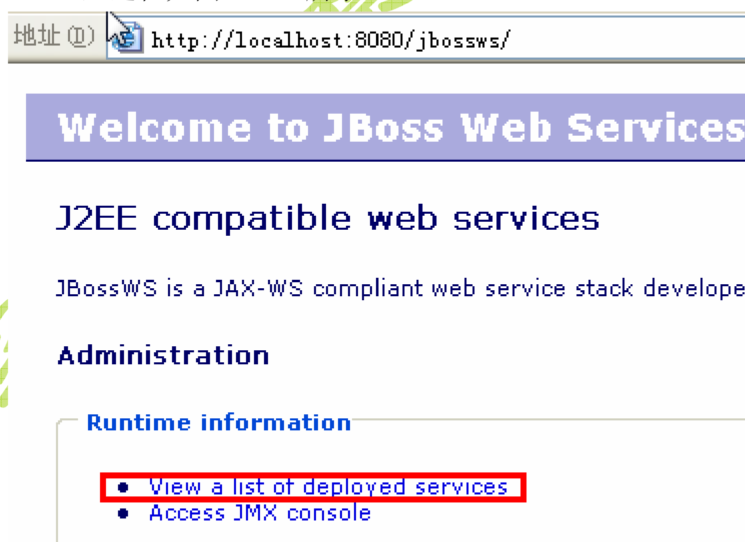


图 15.23 在 JBoss Web Service 控制台中检查 Web 服务

15.5 可视化创建、修改 WSDL

MyEclipse 内置了一个可视化的 WSDL 编辑器，可以用来修改现有的 WSDL 文件，另外还提供了创建 WSDL 的向导。由于现在 WSDL 一般都是使用工具来生成，因此本节内容只做简介，了解即可。

首先，我们看看创建 WSDL 文件，依然在 HelloWorldService 项目中进行。选择菜单 **File > New > Other...**，在 **New** 对话框中展开 **MyEclipse > Web Services > WSDL**，之后即可启动创建 WSDL 的向导。第一页如图 15.24 所示，在 File name:(文件名)处输入文件名后，点击 Next 按钮后可进入第二页。

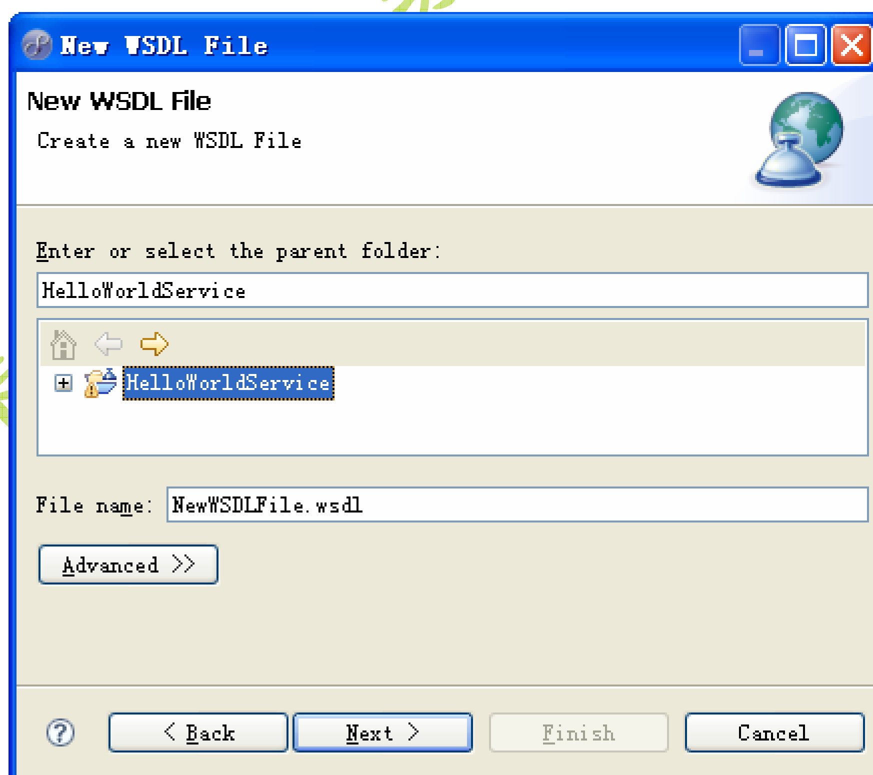


图 15.24 新建 WSDL 文件向导第一页

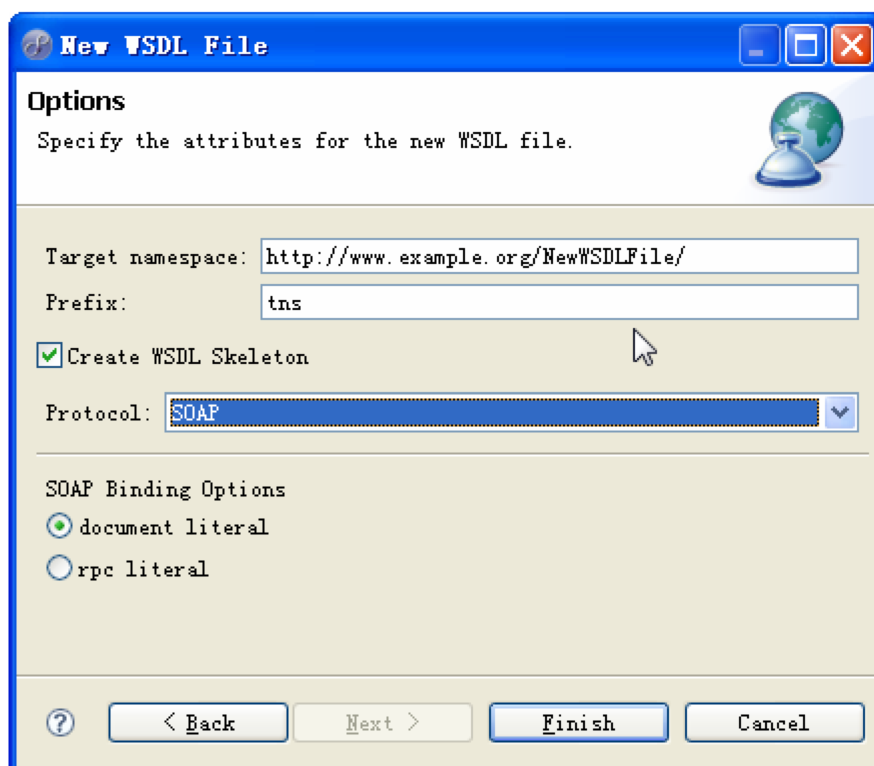


图 15.25 新建 WSDL 文件向导第二页

在第二页，需要对 WSDL 文件的一些属性进行设置，包括 Target namespace（目标命名空间），Prefix（前缀），以及 Create WSDL Skeleton（创建 WSDL 骨架），以及所选的协议（Protocol）和 SOAP 绑定选项（SOAP Binding Options）：document literal（文档字面）和 rpc literal（远程过程调用字面）。除非读者对 WSDL 比较熟悉，一般来说保持默认值即可。设置完毕后，点击 **Finish** 按钮后关闭向导即可创建完成 WSDL 文件。此后 MyEclipse 即会在 WSDL 编辑器中打开刚刚创建的文件 **NewWSDLFile.wsdl**，此时文件的代码清单如下所示：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.example.org/NewWSDLFile/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="NewWSDLFile"
targetNamespace="http://www.example.org/NewWSDLFile/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.example.org/NewWSDLFile/">
      <xsd:element name="NewOperation">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="NewOperationResponse">
```



```
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="out" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="NewOperationRequest">
    <wsdl:part element="tns:NewOperation" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="NewOperationResponse">
    <wsdl:part element="tns:NewOperationResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="NewWSDLFile">
    <wsdl:operation name="NewOperation">
      <wsdl:input message="tns:NewOperationRequest"/>
      <wsdl:output message="tns:NewOperationResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="NewWSDLFileSOAP" type="tns:NewWSDLFile">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="NewOperation">
      <soap:operation
soapAction="http://www.example.org/NewWSDLFile/NewOperation"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="NewWSDLFile">
    <wsdl:port binding="tns:NewWSDLFileSOAP" name="NewWSDLFileSOAP">
      <soap:address location="http://www.example.org/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

。

MyEclipse 的 WSDL 编辑器提供了 WSDL 设计器，以及对应的 Outline 视图，并可以在 Properties 视图中进行属性的修改，并在设计视图中有上下文菜单可以快速的创建 WSDL 元素。

未经许可 禁止打印传播

首先我们来看看主面板，如图 15.26 所示，基本上包括了 Design（设计）和 Source（源码）两个视图；另外 Outline 视图和 Properties 视图也显示在图中了。其实主要的我们要通过这个图看出 WSDL 都有哪些对外的方法可以用，其实它相当于这样的类定义：

public interface NewWSDLFile {

public String NewOperation(String in);

}

。每个 Service 对应一个对外的服务地址，而 PortType 则提供了对外服务的类型名称，Binding 则将 Service 和 PortType 组合到一起。限于篇幅，我们就不对 WSDL 做更多的讨论了。

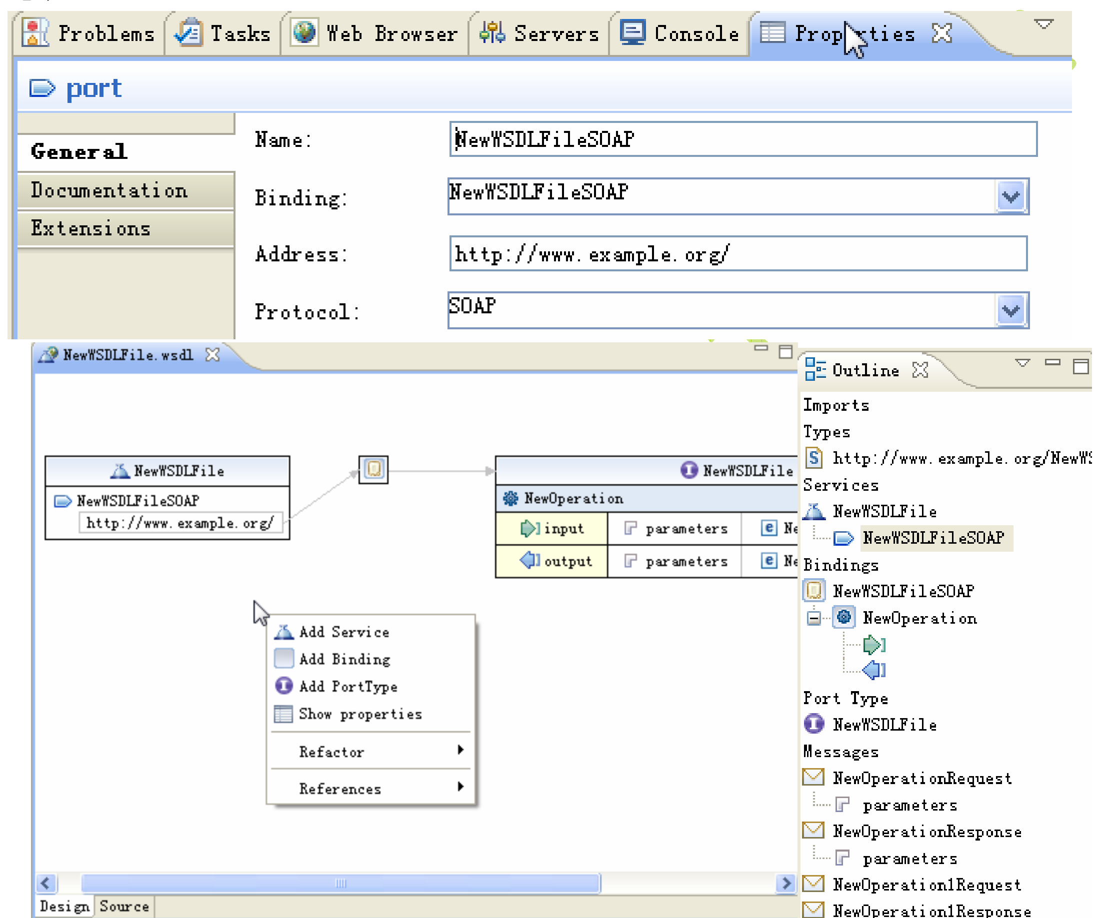


图 15.26 主设计面板及其快捷菜单

15.6 常见问题

1. Web 服务是否是 Servlet？

是的，严格的说 Web Service 的服务端是一个 HTTP 服务，所有的 Web 服务调用都通过 HTTP 来完成，因此无须担心防火墙问题；

2. Web 服务能否传输复杂数据，例如自定义的对象以及下载电影？

听起来很有趣，不过 Web 服务一般用于短暂的交互式服务，不是所有类型的数据都能通过 Web 服务传输，也不能用它来下载电影。

3. Web 服务是否能和 Spring，Hibernate 等框架整合？

当然没有问题，任何服务器端的技术都可以使用。

4. 如何判断 Web 服务是否发布成功？

第一种方式当然是在浏览器中键入服务地址?wsdl 来查看；另外如果能看到服务器的输出的话，可以注意到正常情况下 Console 中的日志是不会出现长串的异常信息的。

5. Web 服务能否在 JDK 1.4 下开发运行？

当然可以，不过此时您不能使用标注方式来开发 Web 服务，而且必须使用 XFire 提供的类库。

6. Web 服务是否支持安全调用？

可以，详细的资料请参考 Web Service Security 方面的资料。

7. Web 服务都有哪些应用？

目前很多服务，例如天气，股票，地图查询，都可以通过 Web 服务来提供。以前笔者工作时，曾经接触过联通公司的某些服务是以 Web 服务的方式进行的。

8. 读者来信：我最近在做一个 web services 的毕业设计，用 myeclipse+xfire 做的（我是个初学者），服务端做好后用 WSDL 浏览器查看的时候出现了错误：

Operations

Name	Documentation
getConnection	--
getPerson	--

Endpoints Add Remove

☐	Endpoints
☐	http://localhost:8080/gzws/services/Person

Status

```
IWAB0380E Errors were encountered while validating XML schemas.
XSD: Type reference 'http://sql.java#Connection' is unresolved
XSD: Type reference 'http://lang.java#IllegalAccessException' is unresolved
XSD: Type reference 'http://sql.java#SQLException' is unresolved
```

图 15.27 Web 服务报错

请问这事怎么回事，现在我用这个服务器端无法生成对应的客户端，请 beansoft 指教下，多谢了！

解答：首先需要一个概念就是 Web Service 不是专门为 Java 定制的，而是一个跨语言的技术，而上面的错误，从字面看是找不到 Connection，Exception 这样的类，根本原因是因为别的语言没有这样的概念和类，生成的时候自然找不到对应类，因此暴露的方法不要带有这样的签名。例如 C++ 中没有 Exception 这样的对应类, Web Service 是跨平台的，所以不要用一些特殊的类。例如：

add(int a, int b)

这很好，但是下面的签名：

add(int a, int b) throws Exception

试问别的语言并无异常处理，那怎办？只需要将接口中暴露出来的 getConnection()方法去

掉，即可正常工作。换句话说，Web Service 中暴露出来的方法和接口以及变量等，必须是简单类型（例如字符串，数组，整数等等）或者是单独定义的类型（一般是 JavaBean，传递查询结果用的），而不能是具有特殊含义的和特定语言相关的类型。

15.10 小结

本章我们就 Web 服务的开发，运行和测试，以及 MyEclipse 中提供的开发工具进行了简要的介绍，我们可以看到目前 Web 服务的开发是大大的简化了。同时我们提醒大家，Web Service 和 Web 服务器不是一个概念，Web 服务器是指提供网页浏览这样的基于 HTTP 的服务内容的软件和硬件的组合。读者看过 Web 服务后，可以根据自己的兴趣对 SOA 做更深入的一些了解。另外，有些读者所在公司可能会使用非商业软件来开发，推荐使用 Netbeans 6，里面对 Web 服务和 JPA 以及 JSF，包括 SOA 都提供了很好的可视化支持。

15.11 参考资料

http://www.ibm.com/developerworks/cn/webservices/understand/index.html 了解 Web 服务规范，基础系列教程，带你全面了解 Web 服务的重要规范 SOAP，WSDL，UDDI...
http://www.blogjava.net/fastzch/archive/2008/01/03/172535.html XFire完整入门教程
http://xfire.codehaus.org/JSR+181+Annotations JSR 181 标注方式的配置开发
http://xfire.codehaus.org/ XFire Java SOAP 框架的主页
http://xfire.codehaus.org/Aegis+Binding XFire Aegis Java-XML绑定信息

未经许可 禁止打印传播