Java Web Services开发基础

刘涛 2011.9

Web Services（Web服务）是主流的分布式软件技术之一，本文列举了Web服务的基本概念、Java开发Web Services的主流框架、Hello World实例，有助于Java Web Services开发入门。

1. Web Services简单理解

Web Services是发布在网络上（包括Internet、Intranet和局域网)，由众多Client（不论平台、语言）通过网络调用的软件功能/服务，是一种特殊的Web应用。从最终使用Web Services角度来看，程序员可以像调用本地jar包或dll文件里的方法一样，调用远程Web Services里的操作。（附录有Web Services简介）

2. Java Web Services两个规范：JAX-RPC 1.1、JAX-WS 2.0

JAX-RPC 1.1、JAX-WS 2.0是Java Web Services的两个主要规范，后者是前者的延续，兼容前者，又比前者支持更高版本的SOAP、WSDL等规范，增加异步调用等新内容。更多的差异，可参考《Web 服务提示与技巧: JAX-RPC 与 JAX-WS 的比较》一文。

3. 直接查看/调用现有Web Services

直接在浏览器中输入wsdl地址，形如<http://localhost:9080/jdk6WsExample?wsdl>，可查看到wsdl。

打开Eclipse Java EE perspective中Run/Launch the Web Services Explorer，通过wsdl url找到Web Services，然后给WSDL operation输入参数值即可调用服务、返回结果了，具体可参考Eclipse Help Web Application Development User Guide > Developing Web service applications > Testing and validating Web services > Testing WSDL documents and Web services using the WSDL Explorer。

在Eclipse中也可配置TCP/IP Monitor监视Web Services客户端和服务端之间的HTTP通信，详见Eclipse Help Web Application Development User Guide > Developing Web service applications > Testing and validating Web services > Using the TCP/IP Monitor to test Web services。

axis1带有一个tcpmon，可独立运行，详见axis1 User's Guide/Appendix : Using the Axis TCP Monitor (tcpmon)。

4. Java Web Services的开发模式

自上而下，契约优先，WSDL优先，先创建WSDL，再由WSDL生成代码。

自下而上，代码优先，先写代码，再由代码生成WSDL。

5. Web Services开发注意点

**安全性**。Web Services以soap消息通过http等传输，天生不具备安全机制。实现服务安全的方法：https传输层加密，WS-Security消息加密/签名，内容自行加密。

**复杂数据类型**。Java数据类型要和WSDL/XSD数据类型互相映射，字符串等简单数据类型映射方便，对于List、Map之类复杂数据类型一般需要给框架配置序列化/反序列化方式。可通过输入参数、输出参数均是xml串、自己解析数据来绕过复杂数据类型映射问题。

**互操作性**。不同版本规范、不同语言、不同框架生成的web服务互用性不是100%。

6. Java Web Services的开发框架

Java中Web Services开发框架广为人知的主要有：axis1，xfire，axis2，cxf，metro/JAX-WS RI，spring-ws。

axis1实现了JAX-RPC 1.1，最高版本是1.4（2006.4.22）。

xfire实现了JAX-WS 2.0部分，最高版本是1.2.6（2007.5.3）。

axis2是axis1的下一代框架，实现了JAX-WS 2.0，目前版本是1.6.1（2011.8.30）。

cxf是在xfire和Celtix合并基础上演进的，实现了JAX-WS 2.0，目前版本是2.4.2（2011.8.15）。

metro来自应用服务器glassfish，是以JAX-WS RI为核心的Web Services栈。JAX-WS RI实现了JAX-WS各版本，目前版本是2.2.5。

spring-ws最大的特点是只支持WSDL优先开发模式，入门困难，目前版本是2.0.2（2011.5.20）。

全面、中肯地比较各个框架优劣是困难的，需要对多种规范、框架有深刻理解，但我们开发者可简单从主流IDE对这些框架的支持来获得直观印象，比如最新的Eclipse 3.7同时支持axis1、axis2、cxf。

7. J2SE 6的Hello World服务实例

7.1 用注解编写JAX-WS服务

**package** example.ws.jdk6.server;

**import** javax.jws.WebService;

@WebService

**public** **class** SayHello {

**private** **static** **final** String *SALUTATION* = "Hello";

**public** String getGreeting(String name) {

**return** *SALUTATION* + " " + name;

}

}

7.2 通过ant用wsgen命令行或ant task生成服务发布所需的类

<target name="wsgen">

<exec executable="wsgen">

<arg line="-cp ${cp.dir} -keep -s ${src.gen.dir} -d ${output.dir}

example.ws.jdk6.server.SayHello" />

</exec>

</target>

7.3 用JDK 提供的轻量HTTP server部署web services

**package** example.ws.jdk6.server;

**import** javax.xml.ws.Endpoint;

**public** **class** RunWSWithJdk6HttpServer {

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

Endpoint.*publish*("http://localhost:9080/jdk6WsExample", **new** SayHello());

}

}

7.4 启动web services，用Web Services Explorer测试

7.5 通过ant用wsimport命令行或ant task生成客户端所需的类

<target name="wsimport">

<exec executable="wsimport">

<arg line="-keep -s ${src.gen.dir} -p example.ws.jdk6.client.jaxws

-d ${output.dir} ${wsdl.url}" />

</exec>

</target>

7.6 编写客户端

**package** example.ws.jdk6.client;

**import** javax.xml.ws.BindingProvider;

**import** org.apache.log4j.Logger;

**import** example.ws.jdk6.client.jaxws.SayHello;

**import** example.ws.jdk6.client.jaxws.SayHelloService;

**public** **class** SayHelloClient {

**private** **static** **final** Logger *LOGGER* = Logger.*getLogger*(SayHelloClient.**class**);

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

String orignWsdlUrl = "http://localhost:9080/jdk6WsExample";

SayHelloService shs = **new** SayHelloService();

SayHello sh = (SayHello) shs.getSayHelloPort();

((BindingProvider) sh).getRequestContext().put(

BindingProvider.*ENDPOINT\_ADDRESS\_PROPERTY*, orignWsdlUrl);

*LOGGER*.info(((BindingProvider) sh).toString());

*LOGGER*.info(sh.getGreeting("liutao"));

}

}

8. axis1的Hello World服务实例

8.1 编写JAX-RPC服务

**package** example.ws.axis.server;

**public** **class** Welcome {

**private** **static** **final** String *SALUTATION* = "Hello";

**public** String greet(String name) {

**return** *SALUTATION* + " " + name;

}

}

8.2 选中服务类，右键弹出Web Services向导，向导拷贝axis及其依赖jar、生成axis部署文件和WSDL

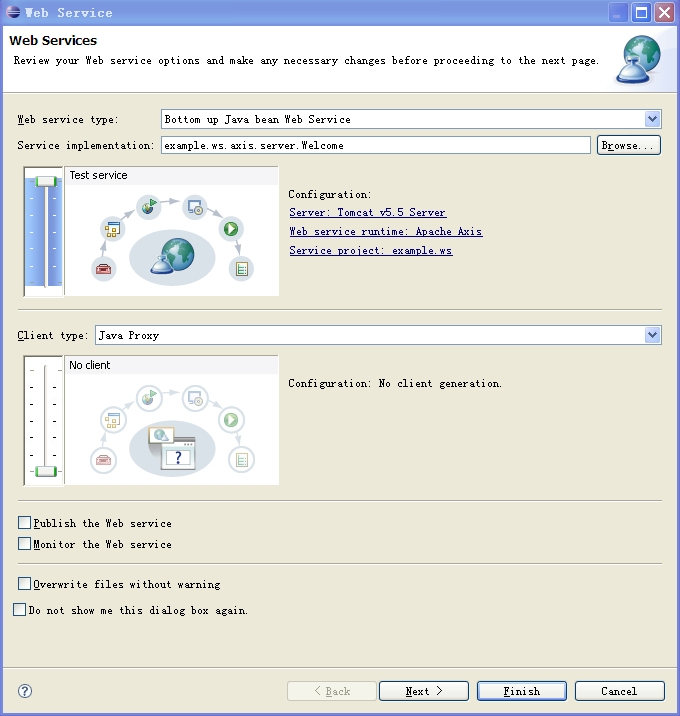


图 8-1 axis服务向导1

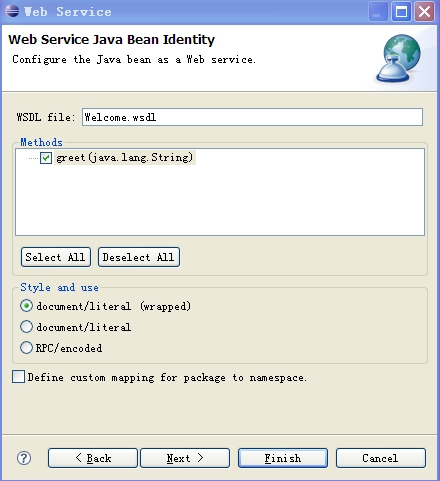


图 8-2 axis服务向导2



图 8-3 axis服务向导3

单击“Start server”后，成功启动Server后再Next



图 8-4 axis服务向导4

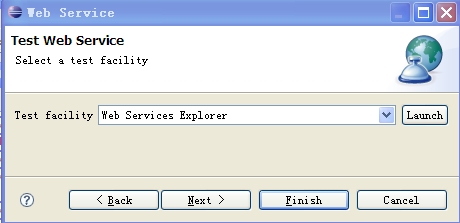


图 8-5 axis服务向导5



图 8-6 axis服务向导6

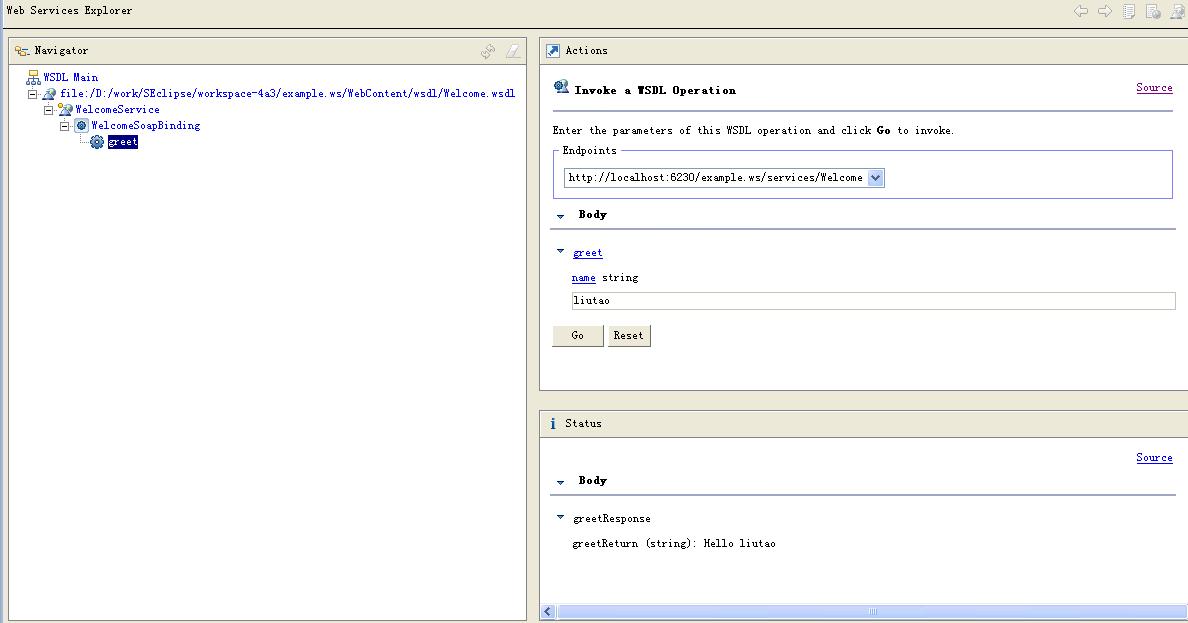


图 8-7 axis服务向导7

8.3 生成JAX-RPC客户端

8.3.1 用Web Services向导生成Proxy客户端

选中工程，单击File>New>Other>Web Services>Web Services Client：

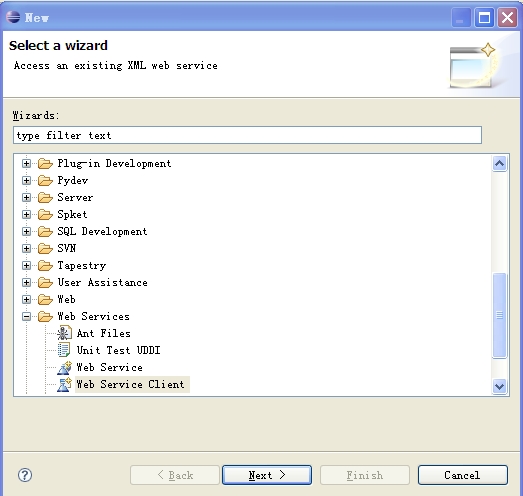


图 8-8 axis客户端向导1

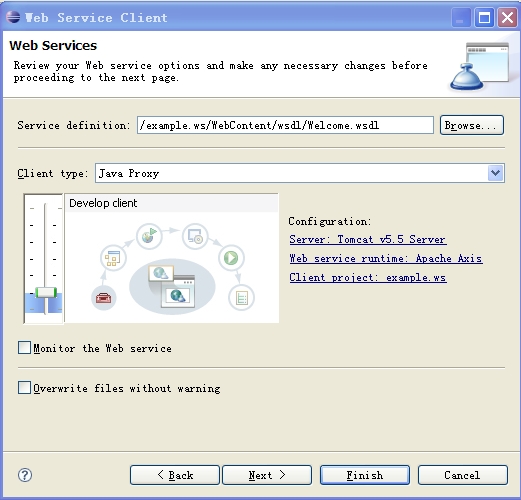


图 8-9 axis客户端向导2

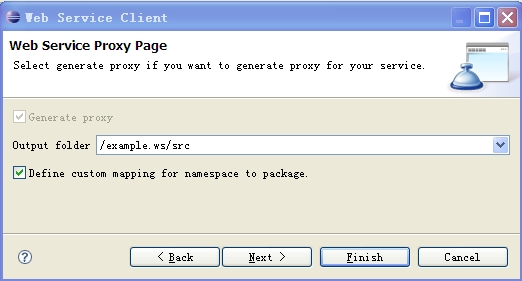


图 8-10 axis客户端向导3

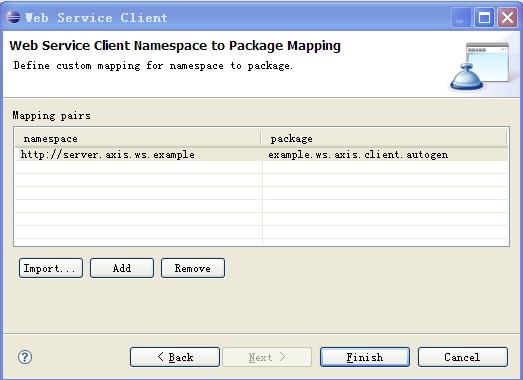


图 8-11 axis客户端向导4

调用生成的Proxy客户端：

**package** example.ws.axis.client;

**import** java.rmi.RemoteException;

**import** org.apache.log4j.Logger;

**import** example.ws.axis.client.autogen.WelcomeProxy;

**public** **class** WelcomeProxyClient {

**private** **static** **final** Logger *LOGGER* = Logger.*getLogger*(WelcomeProxyClient.**class**);

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

WelcomeProxy welcomeProxy = **new** WelcomeProxy();

String greeting = **null**;

**try** {

greeting = welcomeProxy.greet("liutao");

} **catch** (RemoteException e) {

*LOGGER*.error(e.getMessage(), e);

**return**;

}

*LOGGER*.info(greeting);

}

}

8.3.2 直接调用Web Services向导生成的服务stub

**package** example.ws.axis.client;

**import** java.net.URL;

**import** org.apache.log4j.Logger;

**import** example.ws.axis.client.autogen.WelcomeSoapBindingStub;

**public** **class** WelcomeStubClient {

**private** **static** **final** Logger *LOGGER* = Logger.*getLogger*(WelcomeStubClient.**class**);

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

String greeting = **null**;

**try** {

URL endpointURL = **new** URL("http://localhost:8080/example.ws/services/Welcome");

WelcomeSoapBindingStub welcomeSoapBindingStub = **new** WelcomeSoapBindingStub(endpointURL, **null**);

greeting = welcomeSoapBindingStub.greet("liutao");

} **catch** (Exception e) {

*LOGGER*.error(e.getMessage(), e);

**return**;

}

*LOGGER*.info(greeting);

}

}

8.3.3 编写动态客户端

**package** example.ws.axis.client;

**import** java.net.URL;

**import** javax.xml.namespace.QName;

**import** org.apache.axis.client.Call;

**import** org.apache.axis.client.Service;

**import** org.apache.log4j.Logger;

**public** **class** WelcomeCallClient {

**private** **static** **final** Logger *LOGGER* = Logger.*getLogger*(WelcomeCallClient.**class**);

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

String greeting = **null**;

**try** {

URL endpointURL = **new** URL("http://localhost:8080/example.ws/services/Welcome");

Service service = **new** Service();

Call call = (Call)service.createCall();

call.setTargetEndpointAddress(endpointURL);

call.setOperationName(**new** QName("http://server.axis.ws.example", "greet"));

greeting = (String)call.invoke(**new** Object[]{"liutao"});

} **catch** (Exception e) {

*LOGGER*.error(e.getMessage(), e);

**return**;

}

*LOGGER*.info(greeting);

}

}

9. axis1 + spring的Hello World服务实例

9.1 编写服务接口和实现类

**package** example.ws.axisspring.server;

**public** **interface** Hello {

String greet(String name);

}

**package** example.ws.axisspring.server;

**public** **class** HelloImpl **implements** Hello {

**public** String greet(String name){

**return** "Hello " + name;

}

}

9.2 服务集成Spring

**package** example.ws.axisspring.server;

**import** org.springframework.remoting.jaxrpc.ServletEndpointSupport;

**public** **class** HelloWs **extends** ServletEndpointSupport **implements** Hello{

**private** Hello hello;

**protected** **void** onInit() {

**this**.hello = (Hello) getWebApplicationContext().getBean("hello");

}

**public** **void** setHello(Hello hello){

**this**.hello = hello;

}

**public** String greet(String name){

**return** hello.greet(name);

}

}

类路径上放置spring配置文件spring-axis-server.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:context="http://www.springframework.org/schema/context"

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/context> http://www.springframework.org/schema/context/spring-context-2.5.xsd

<http://www.springframework.org/schema/tx> http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<bean id="hello" class="example.ws.axisspring.server.HelloImpl">

</bean>

</beans>

web.xml中增加spring配置：

<listener> <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/classes/spring-axis-server.xml

</param-value>

</context-param>

9.3 选中集成Spring的服务类，右键弹出Web Services向导，向导拷贝axis及其依赖jar、生成axis部署文件和WSDL

首先将servlet-api.jar加入到类路径上。

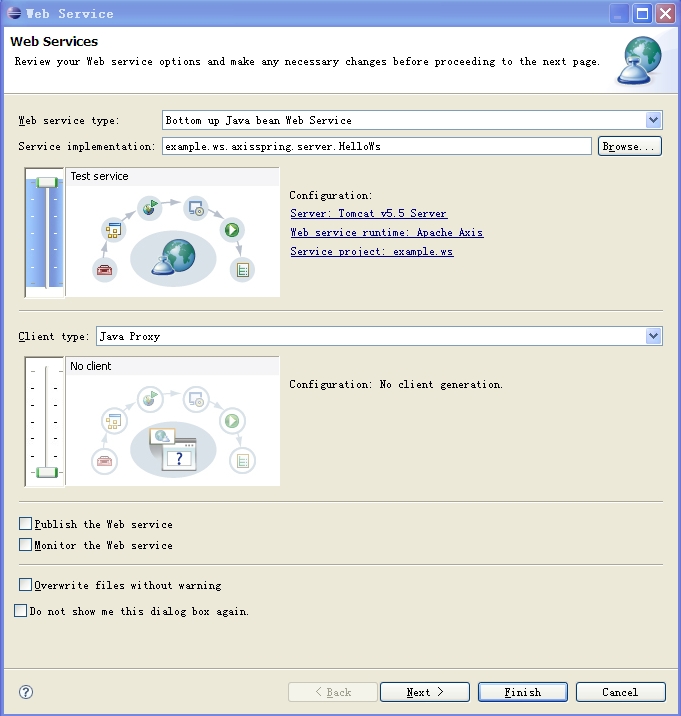


图 9-1 axis服务向导1

Eclipse会弹出关于服务互操作性的告警，忽略之。

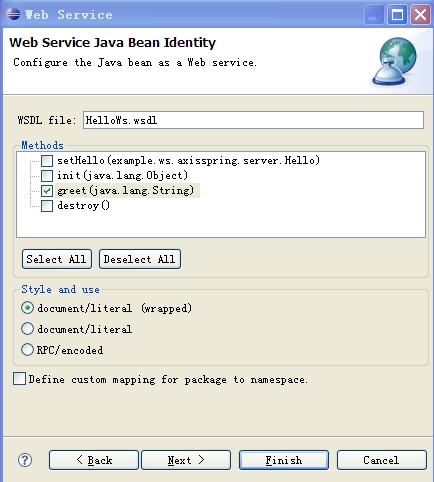


图 9-2 axis服务向导2

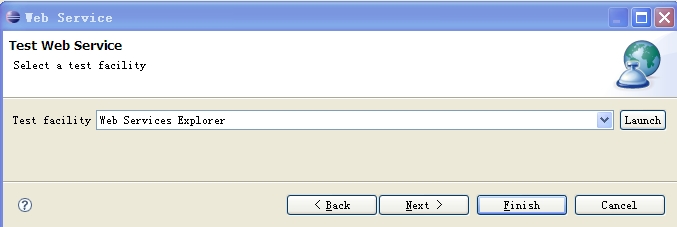


图 9-3 axis服务向导3

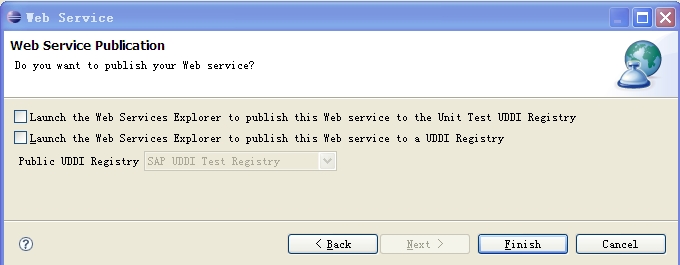


图 9-4 axis服务向导4

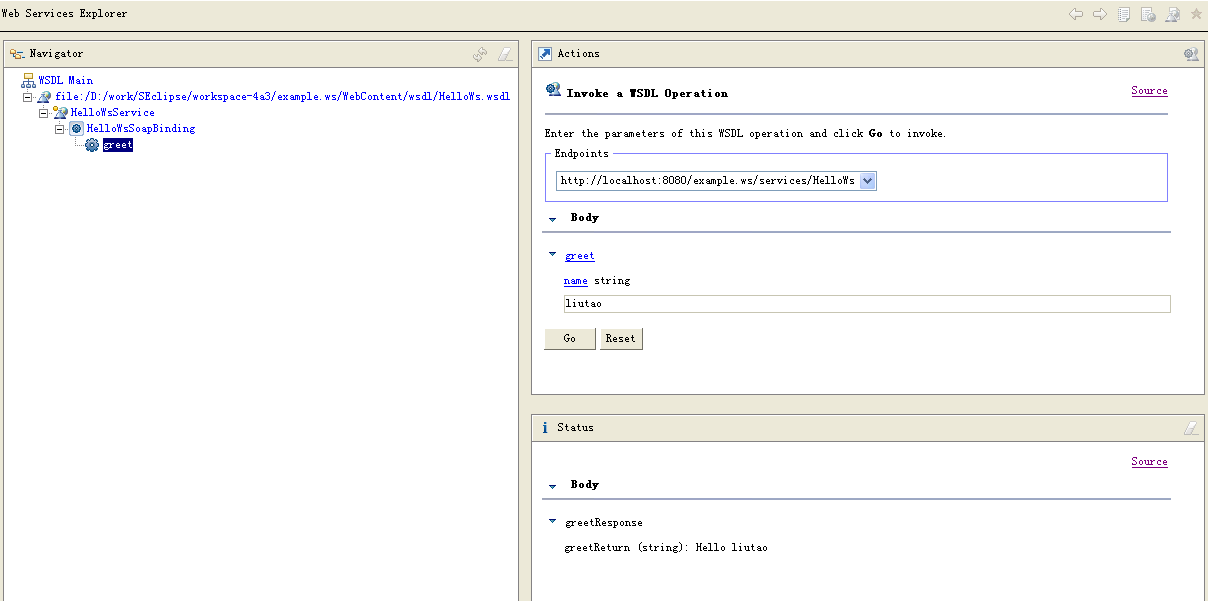


图 9-5 axis服务向导5

9.4 编写集成Spring的客户端

**package** example.ws.axisspring.client;

**import** org.apache.axis.client.ServiceFactory;

**import** org.apache.log4j.Logger;

**import** org.springframework.beans.MutablePropertyValues;

**import** org.springframework.context.support.StaticApplicationContext;

**import** org.springframework.remoting.jaxrpc.JaxRpcPortProxyFactoryBean;

**import** example.ws.axisspring.server.Hello;

**public** **class** HelloClient {

**private** **static** **final** Logger *LOGGER* = Logger.*getLogger*(HelloClient.**class**);

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

String wsdlUrl = "http://localhost:8080/example.ws/services/HelloWs?WSDL";

MutablePropertyValues mutPropVal = **new** MutablePropertyValues();

mutPropVal.addPropertyValue("serviceFactoryClass", ServiceFactory.**class**);

mutPropVal.addPropertyValue("serviceInterface", Hello.**class**);

mutPropVal.addPropertyValue("wsdlDocumentUrl", wsdlUrl);

mutPropVal.addPropertyValue("namespaceUri", "http://server.axisspring.ws.example");

mutPropVal.addPropertyValue("serviceName", "HelloWsService");

mutPropVal.addPropertyValue("portName", "HelloWs");

StaticApplicationContext staticAppContext = **new** StaticApplicationContext();

staticAppContext.registerSingleton("hello", JaxRpcPortProxyFactoryBean.**class**,

mutPropVal);

Hello hello = (Hello)staticAppContext.getBean("hello");

*LOGGER*.info(hello.greet("liutao"));

}

}

10. xfire的Hello World动态客户端实例

**package** example.ws.xfire.client;

**import** java.net.URL;

**import** org.codehaus.xfire.client.Client;

**import** org.apache.log4j.Logger;

**public** **class** HelloWsCallClient {

**private** **static** **final** Logger *LOGGER* =

Logger.*getLogger*(HelloWsCallClient.**class**);

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

String greeting = **null**;

**try** {

URL endpointURL = **new**

URL("http://localhost:8080/example.ws/services/HelloWs?wsdl");

Client client = **new** Client(endpointURL);

greeting = (String)client.invoke("greet", **new**

Object[]{"liutao"})[0];

} **catch** (Exception e) {

*LOGGER*.error(e.getMessage(), e);

**return**;

}

*LOGGER*.info(greeting);

}

}

11. 4A工程服务实例

cvs 192.168.4.144

xfire服务：/E/securer/4A-R08009/01工作产品管理/99代码/faimserver

xfire动态客户端、简单认证：/E/securer/4A-R08009/03工程项目管理/陕西移动/基线区/99代码/app-ws-agent

axis1服务：/E/securer/4A-R08009/03工程项目管理/陕西移动/基线区/99代码/SSOServer

（最新版3.1 SSOServer在/E/securer/4A-Securer/01工作产品管理/99代码/com.linkage.fa.ssoserver，在线）

axis1 Proxy客户端：/E/securer/4A-R08009/03工程项目管理/陕西移动/基线区/99代码/CommonAuthClient

（最新版单点登录客户端在/E/securer/4A-R08009/01工作产品管理/FA3/ CommonAuthClient）

axis1服务、动态客户端、wss4j：/E/securer/4A-R08009/03工程项目管理/天津4A/99代码/appacctwebservices

12. 进阶内容

(1) 读懂、维护4A Web Services工程

(2) 研究Axis2、CXF，储备技术

(3) 熟悉各主流框架下的复杂数据类型处理

(4) 熟悉各主流框架下的服务安全

(5) 研究各主流框架所开发服务的互操作性

参考资料

1. Eclipse Help Web Application Development User Guide > Developing Web service applications

2. IBM developerWorks技术主题SOA and web services

<http://www.ibm.com/developerworks/cn/webservices/>

3. axis1 <http://axis.apache.org/axis/>

4. axis2 <http://axis.apache.org/>

5. xfire <http://xfire.codehaus.org/>

6. cxf <http://cxf.apache.org/>

7. metro/JAX\_WS RI <http://jax-ws.java.net/>

8. spring-ws <http://static.springsource.org/spring-ws/site/index.html>

9. Web 服务提示与技巧: JAX-RPC 与 JAX-WS 的比较

<http://www.ibm.com/developerworks/cn/webservices/ws-tip-jaxwsrpc.html>

10. 使用 Eclipse 和 Java SE 6 创建独立 Web Services 应用程序，第 1 部分: Web Services 服务端应用程序

<https://www.ibm.com/developerworks/cn/education/webservices/ws-eclipse-javase1/>

使用 Eclipse 和 Java SE 6 创建独立 Web Services 应用程序，第 2 部分: Web 服务客户端应用程序

<http://www.ibm.com/developerworks/cn/webservices/tutorials/ws-jse/>

11. Service Oriented Architecture with Java Chapter2和Chapter3

附录

**Web Services 简介**

Web services是自包含、自描述的模块化应用，能在网络上发布、定位和调用。

自包含。客户端，编程语言只要支持XML和HTTP客户端即可，无需额外软件。服务端，需要一个Web服务器和servlet引擎。客户端和服务端能在不同的环境中实现。有可能不需要写代码就可以将一个已存在应用转化为Web服务。

自描述。客户端和服务端仅需识别请求/响应消息的格式和内容。消息本身携带消息格式定义；不需要外部元数据或代码生成工具。

模块化。简单的Web服务可通过工作流技术或调用更低层Web服务聚合成更复杂的Web服务。

平台无关性。Web服务基于一系列开放的、基于XML的标准，这些标准可为一个Web服务和各种计算平台上、编程语言的客户端提供互用性。

Web Services是一种分布式软件体系结构，有各种规范、协议，有不同的实现和开发工具支持，例如在JEE和.NET上都有实现。

满足一定的规范如WS-I后，Web Services间可实现互操作，例如一个Web Service可调用另一个Web Service。

图1表示了Web Services的架构。

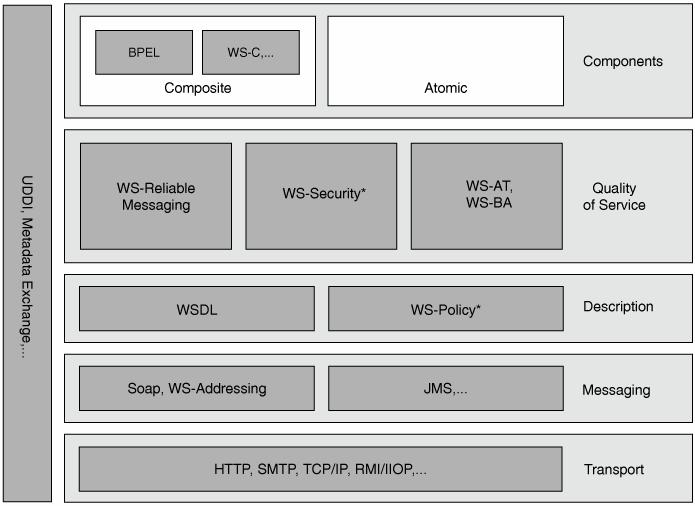


图 1 Web Services架构

Web Services的基本协议有：

SOAP：即Simple Object Access Protocol，简单对象访问协议。SOAP属于消息传递协议，基于XML，它对消息进行了编码，这样就可以通过传输协议（如 HTTP、IIOP、SMTP 或其他协议）在网络上传递它们。

WSDL：即Web Services Description Language，Web 服务描述语言。WSDL属于接口描述协议，基于XML，描述了网络上Web服务的接口和实例。WSDL文档可通过UDDI、WSIL获得，或者通过e-mail、网站等公开其URL。

WS-Security：Web 服务安全性规范，为安全通信定义了基于令牌的体系结构。

UDDI：即Universal Description, Discovery and Integration，统一描述、发现和集成协议。UDDI为查找和访问服务定义了注册中心和相关协议。

WSIL：即Web Services Inspection Language，Web服务检查语言。WSIL基于XML，定义了分布式的服务发现方法，是对UDDI的补充，可在Web站点上发现服务。

图2表示了SOAP, UDDI, WSIL 和WSDL间的关系。

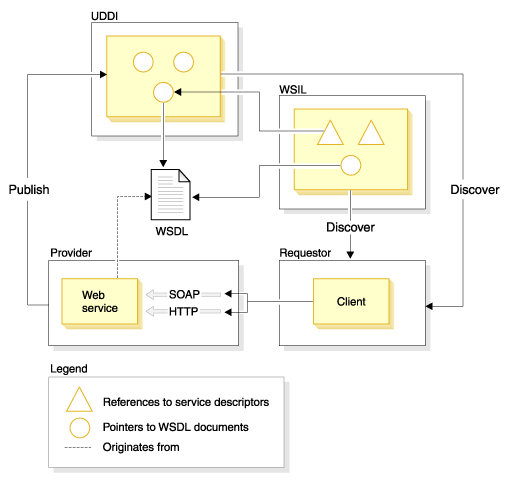


图 2 SOAP, UDDI, WSIL和WSDL间的关系

Web Services可分为三类：

1. 业务信息（business information）。企业与消费者或企业共享信息，例如天气预报、股票行情。

2. 业务集成（business integration）。企业为消费者提供事务性的收费服务，例如订票系统、电子商务。

3. 业务过程整合（business process externalization）。企业与其合作伙伴如制造商、组装商等整合业务。

Web Services中有三种角色：

1. 服务提供者（service provider），创建、部署和发布Web服务。

2. 服务代理（service broker），注册、分类被发布的Web服务，并提供服务查询。例如UDDI可作为WSDL描述的Web服务的服务代理。

3. 服务客户端（service client），使用代理服务发现、绑定和调用服务提供者。

三种角色的交互方式如图3所示：

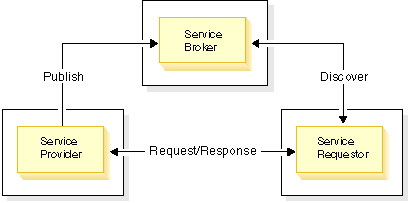


图 3 Web Services角色交互方式