leilei1334

leilei1334
浏览: 526 次
性别:
来自: 北京
我现在离线
详细资料 留言簿

搜索本博客
搜索

最近访客 >>更多访客

huang_sm advenplayer

C032026

博客分类
- 全部博客 (2)

其他分类
- 我的收藏 (0)
- 我的论坛主题贴 (0)
- 我的所有论坛贴 (0)
- 我的精华良好贴 (0)

最近加入圈子

存档
- 2008-11 (2)
- 更多存档...

最新评论

评论排行榜

使用Apache CXF历程

2008-11-14

设计与开发 JAX-WS 2.0 Web 服务

关键字: java webservice

实验环境:
Java SE Development Kit (JDK) 6 Update 10

原文:
Naveen Balani, 开发经理, IBM
2007 年 11 月 29 日
通过使用 Java™ API for XML Web Services (JAX-WS) 技术设计和开发 Web 服务，可以带来很多好处，能简化 Web 服务的开发和部署，并能加速 Web 服务的开发。通过此教程，可以了解如何开发将其功能作为 Web 服务公开的示例订单处理程序，从而进行所有这些工作以及其他任务。完成了此教程后，您将能够应用这些概念和新获得的知识，来使用 JAX-WS 技术为应用程序开发 Web 服务。

JAX-WS 简介
为何使用 JAX-WS？
JAX-WS 是用于简化使用 Java 构造 Web 服务和 Web 服务客户机的工作的技术。该技术提供了完整的 Web 服务堆栈，可减少开发和部署 Web 服务的任务。JAX-WS 支持 WS-I Basic Profile 1.1，后者可确保使用 JAX-WS 堆栈开发的 Web 服务能够供采用 WS-I Basic Profile 标准使用任意语言开发的任意客户机使用。
JAX-WS 还包括了 Java Architecture for XML Binding (JAXB) 和 SOAP with Attachments API for Java (SAAJ)。
注: Simple Object Access Protocol (SOAP)

JAXB 提供了一种非常方便的方法来将 XML 模式映射到 Java 代码的表示形式，从而支持数据绑定功能。JAXB 消除了将 SOAP 消息中的 XML 模式消息转换为 Java 代码的工作，因而不必全面了解 XML 和 SOAP 解析。JAXB 规范定义 Java 和 XML 模式之间的绑定。SAAJ 提供了标准的方法来处理 SOAP 消息中包含的 XML 附件。

而且，JAX-WS 提供了用于将传统 Java 对象（Plain Old Java Object，POJO）类转换为 Web 服务的 Annotation 库，从而加速了 Web 服务的开发工作。另外，它还指定了从采用 Web 服务描述语言（Web Services Description Language，WSDL）定义的服务到实现该服务的 Java 类之间的详细映射。采用 WSDL 定义的任意复杂类型都通过遵循 JAXB 规范定义的映射来映射为 Java 类。
JAX-WS 之前与 Java Platform, Enterprise Edition (Java EE) 5 绑定。而 JAX-WS 2.0 规范是作为 Java Community Process (JCP) 的 Java Specification Requests(JSR) 224 开发的。

开发 Web 服务

契约优先方法与代码优先方法
进入 JAX-WS 时代的最好方法莫过于首先开发一个 Web 服务。可以采用以下两种方法之一开发 Web 服务:
契约优先：从 WSDL 契约着手，生成 Java 类来实现服务。
代码优先：从 Java 类着手，使用 Annotation 来生成 WSDL 文件和 Java 接口。

契约优先 WSDL 方法需要对用于定义消息格式的 WSDL 和 XML 模式定义（XML Schema Definition，XSD）有良好的理解。如果您对 Web 服务相当陌生，最好从代码优先方法着手，本教程中将使用此方法开发 Web 服务。

代码优先 Web 服务开发
使用代码优先方法时，将从实现希望作为服务公开的功能的 Java 类或类入手。在已经提供了 Java 实现且需要将实现作为服务公开的情况下，代码优先方法尤为有用。

开发订单处理 Web 服务
让我们首先创建一个订单处理 Web 服务，用于接受订单信息、配送信息和订购物品并最终生成确认 ID 作为响应。订单处理服务的代码如清单 1 中所示。这是一个虚拟实现，将在控制台输出客户 ID 和物品数量，然后输出虚拟订单 ID A1234。

清单 1. 订单处理 Web 服务实现

Java代码

```java
package com.ibm.jaxws.tutorial.service;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import com.ibm.jaxws.tutorial.service.bean.OrderBean;
//JWS annotation that specifies that the portType name of the
//Web service is "OrderProcessPort," the service name
//is "OrderProcess," and the targetNamespace used in the generated
//WSDL is "http://jawxs.ibm.tutorial/jaxws/orderprocess."

@WebService(serviceName = "OrderProcess",
  portName = "OrderProcessPort",
  targetNamespace = "http://jawxs.ibm.tutorial/jaxws/orderprocess")

//JWS annotation that specifies the mapping of the service onto the
// SOAP message protocol. In particular, it specifies that the SOAP messages
//are document literal.

@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,use=SOAPBinding.Use.LITERAL,
   parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)

public class OrderProcessService {

 @WebMethod
 public OrderBean processOrder(OrderBean orderBean) {

  // Do processing...
  System.out.println("processOrder called for customer"
    + orderBean.getCustomer().getCustomerId());

  // Items ordered are
  if (orderBean.getOrderItems() != null) {
   System.out.println("Number of items is "
     + orderBean.getOrderItems().length);
  }

  //Process order.

  //Set the order ID.
  orderBean.setOrderId("A1234");

  return orderBean;
 }
}
```

OrderBean 中包含订单信息，如清单 2 中所示。具体来说，其中包含对客户、订单项和配送地址对象的引用。

清单 2. 包含订单信息的 OrderBean 类

Java代码

```java
package com.ibm.jaxws.tutorial.service.bean;

public class OrderBean {

 private Customer customer;

 private Address shippingAddress;

 private OrderItem[] orderItems;

 private String orderId;

 public Customer getCustomer() {
  return customer;
```

```
 }

 public void setCustomer(Customer customer) {
  this.customer = customer;
 }

 public String getOrderId() {
  return orderId;
 }

 public void setOrderId(String orderId) {
  this.orderId = orderId;
 }

 public Address getShippingAddress() {
  return shippingAddress;
 }

 public void setShippingAddress(Address shippingAddress) {
  this.shippingAddress = shippingAddress;
 }

 public OrderItem[] getOrderItems() {
  return orderItems;
 }

 public void setOrderItems(OrderItem[] orderItems) {
  this.orderItems = orderItems;
 }

}
public class Customer {
private String customerId;
private String firstName;
private String lastName;
private String phoneNumber;
private String faxNumber;
private Address address;
public Address getAddress() {
   return address;
}
public void setAddress(Address address) {
   this.address = address;
}
public String getCustomerId() {
   return customerId;
}
public void setCustomerId(String customerId) {
   this.customerId = customerId;
}
public String getFaxNumber() {
   return faxNumber;
}
public void setFaxNumber(String faxNumber) {
   this.faxNumber = faxNumber;
}
public String getFirstName() {
   return firstName;
}
public void setFirstName(String firstName) {
   this.firstName = firstName;
}
public String getLastName() {
   return lastName;
}
```

```
public void setLastName(String lastName) {
   this.lastName = lastName;
}
public String getPhoneNumber() {
   return phoneNumber;
}
public void setPhoneNumber(String phoneNumber) {
   this.phoneNumber = phoneNumber;
}
}
public class Address {
private String addressLine1;
private String addressLine2;
private String city;
private String state;
private String zipCode;
private String country;
public String getAddressLine1() {
   return addressLine1;
}
public void setAddressLine1(String addressLine1) {
   this.addressLine1 = addressLine1;
}
public String getAddressLine2() {
   return addressLine2;
}
public void setAddressLine2(String addressLine2) {
   this.addressLine2 = addressLine2;
}
public String getCity() {
   return city;
}
public void setCity(String city) {
   this.city = city;
}
public String getCountry() {
   return country;
}
public void setCountry(String country) {
   this.country = country;
}
public String getState() {
   return state;
}
public void setState(String state) {
   this.state = state;
}
public String getZipCode() {
   return zipCode;
}
public void setZipCode(String zipCode) {
   this.zipCode = zipCode;
}
}
public class OrderItem {
private String itemId;
private int qty;
public String getItemId() {
   return itemId;
}
public void setItemId(String itemId) {
   this.itemId = itemId;
}
public int getQty() {
   return qty;
```

```
}
public void setQty(int qty) {
   this.qty = qty;
}
}
```

开发 JAX-WS Web 服务的起点是一个使用 javax.jws.WebService Annotation 进行了标注的 Java 类。所使用的 JAX-WS Annotation 属于 Web Services Metadata for the Java Platform 规范 (JSR-181) 的一部分。您可能已经注意到了，OrderProcessService 使用 WebService Annotation 进行了标注，而后者将类定义为了 Web 服务端点。

OrderProcessService 类（带有 @javax.jws.WebService Annotation 的类）隐式地定义了服务端点接口（Service Endpoint Interface，SEI），用于声明客户机可以对服务调用的方法。除了使用@WebMethod Annotation 标注且 exclude 元素设置为 true 的方法外，类中定义的所有公共方法都会映射到 WSDL 操作。@WebMethod Annotation 是可选的，用于对 Web 服务操作进行自定义。除了 exclude 元素外，javax.jws.WebMethod Annotation 还提供 operation name 和 action 元素，用于在 WSDL 文档中自定义操作的 name 属性和 SOAP action 元素。这些属性是可选的；如果未定义，会从类名称派生缺省值。

实现 Web 服务后，需要生成部署服务所需的所有构件，然后将 Web 服务打包为部署构件（通常为 WAR 文件），并将 WAR 文件部署到任何支持 JAX-WS 2.0 规范的兼容服务器上。通常生成的构件是提供基于服务接口将 Java 对象转换为 XML、WSDL 文件和 XSD 模式的功能的类。

出于测试目的，Java 6 绑定了一个轻量级 Web 服务器，可以通过调用简单的 API 调用将 Web 服务发布到该服务器上。接下来我们将了解如何使用此方法测试 Web 服务。

生成 JAX-WS 构件
运行 wsgen 工具，以生成订单处理 Web 服务的 JAX-WS 可移植构件。此工具将读取 Web SEI 类，并生成 Web 服务部署和调用所需的所有构件。wsgen 工具生成需要发布的 Web 服务的 WSDL 文件和 XSD 模式。
为了生成 JAX-WS 构件，首先需要编译服务和 Bean 源文件：
运行以下命令，以编译 Java 文件，并将类文件放入其各自文件夹中：

javac com\ibm\jaxws\tutorial\service*.java com\ibm\jaxws\tutorial\service\bean*.java

运行以下命令，以生成 JAX-WS 构件：
wsgen -cp . com.ibm.jaxws.tutorial.service.OrderProcessService -s ../../src -d . -wsdl
wsgen 工具提供了大量的选项，
例如，其中提供了 -wsdl 选项，用于生成服务的 WSDL 和模式构件。
-cp 代表classpath
-s 代表你将把生成的stub类的源代码放置到的目录
-d 代表你将把生成的stub编译好的class放置到的目录
运行此命令后，应该看到生成的 OrderProcess.wsdl 和 OrderProcess_schema1.xsd，而且会看到在 com\ibm\jaxws\tutorial\service\jaxws 文件夹中创建了 JAX-WS 构件。
生成了构件后，运行以下 Web 服务发布器客户机，以发布订单处理 Web 服务。
运行以下命令，以编译 OrderWebServicePublisher：

javac com\ibm\jaxws\tutorial\service\publish\OrderWebServicePublisher.java
源代码:

Java代码

```
public class OrderWebServicePublisher {
public static void main(String[] args) {
   String address = "http://localhost:8080/OrderProcessWeb/orderprocess" ;
   OrderProcessService implementor = new OrderProcessService() ;
   Endpoint.publish(address, implementor) ;
   System.out.println("The web service is published at http://localhost:8080/OrderProcess
Web/orderprocess");
   System.out.println("To stop running the web service , terminate the java process");
}
}
```

5 然后运行以下命令：

java com.ibm.jaxws.tutorial.service.publish.OrderWebServicePublisher

运行 Java 程序后，应该看到以下消息：
The Web service is published at http://localhost:8080/OrderProcessWeb/orderprocess. To stop running the Web service, terminate this Java process.
这会将订单 Web 服务发布到 http://localhost:8080/OrderProcessWeb/orderprocess。可以通过显示订单处理 Web 服务生成的 WSDL 来验证 Web 服务是否在运行：
打开浏览器，并导航到 http://localhost:8080/OrderProcessWeb/orderprocess?wsdl。

分析 OrderWebServicePublisher
在分析 WSDL 和模式构件前，让我们分析一下 OrderWebServicePublisher 的代码。

通过 Endpoint.publish() 方法，可以方便地发布和测试 JAX-WS Web 服务。publish() 接受两个参数：Web 服务的位置和 JAX-WS Web 服务实现类。publish() 方法在指定的 URL（本例中为本地主机，端口为 8080）创建轻量级 Web 服务器，并将 Web 服务部署到该位置。此轻量级 Web 服务器在 Java 虚拟机（Java Virtual Machine，JVM）中运行，可通过调用 endpoint.stop() 方法以有条件的方式终止，或终止 OrderWebServicePublisher 客户机。
分析生成的 WSDL

要查看生成的订单处理 Web 服务 WSDL，在浏览器中键入以下 URL 位置： http://localhost:8080/OrderProcessWeb/orderprocess?wsdl.

让我们分析 WSDL 一些重要方面的内容，并了解如何基于 JAX-WS 元数据生成 WSDL 和模式构件，首先要分析的是生成的 XSD。此内容使用 xsd:import 标记导入到 WSDL 文件中（请参见清单 4）；schemaLocation 指定 XSD 的位置。

清单 4. 包含订单处理模式定义的 WSDL 文件

Xml代码

```xml
<types>
    <xsd:schema>
        <xsd:import namespace="http://jawxs.ibm.tutorial/jaxws/orderprocess" schemaLocation="http://localhost:8080/OrderProcessWeb/orderprocess?xsd=1" />
    </xsd:schema>
</types>
```

在浏览器中打开 schemaLocation (http://localhost:8080/OrderProcessWeb/orderprocess?xsd=1)，以查看模式定义在浏览器中呈现的情况。让我们分析一下其中的情况：模式定义最开始是targetNamspace 和 tns 声明，映射到在 OrderProcessService 的 @WebService Annotation 中定义的 targetNamespace http://jawxs.ibm.tutorial/jaxws/orderprocess。
清单 5 给出了对应的代码。

清单 5. 模式（Schema）命名空间声明

Xml代码

```xml

<xs:schema version="1.0"
    targetNamespace="http://jawxs.ibm.tutorial/jaxws/orderprocess"
    xmlns:tns="http://jawxs.ibm.tutorial/jaxws/orderprocess"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

前面指定的 wsgen 工具命令生成两个包装 Bean 类，ProcessOrder 和 ProcessOrderResponse，分别包含订单处理 Web 服务的输入和输出消息。将基于这些包装 Bean 类生成以下模式元素：

processOrder 属于 processOrder 类型，表示其中包含一个元素，且此元素的名称为 arg0，类型为 orderBean。可以看到，在 ProcessOrder 类和 processOrder 复杂类型之间存在一对一映射。
processOrderResponse 与 processOrderResponse 类型类似，后者的定义映射到 ProcessOrderResponse 类。
让我们仔细分析一下清单 6 中的代码。

清单 6. processOrder 的模式声明

Xml代码

```xml
<xs:element name="processOrder" type="tns:processOrder"/>
<xs:element name="processOrderResponse" type="tns:processOrderResponse"/>
<xs:complexType name="processOrder">
    <xs:sequence>
      <xs:element name="arg0" type="tns:orderBean" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

清单 7 中所示的 orderBean 类型定义映射到 OrderBean 类。orderBean 类型定义包括：

一个 customer 元素，其类型为 customer。
一个 orderId，其类型为 string。
orderItems（它为数组类型，因为其将 maxOccurs 属性指定为 unbounded），其类型为 orderItem。
shippingAddress，其类型为 address。

清单 7. processOrder 的模式声明

Xml代码

```xml
<xs:complexType name="orderBean">
<xs:sequence>
<xs:element name="customer" type="tns:customer" minOccurs="0" />
          <xs:element name="orderId" type="xs:string" minOccurs="0" />
          <xs:element nillable="true" maxOccurs="unbounded" name="orderItems"
                        type="tns:orderItem" minOccurs="0" />
          <xs:element name="shippingAddress" type="tns:address"
                       minOccurs="0" />
</xs:sequence>
</xs:complexType>
```

类似地，模式的其余部分 customer、orderItems 和 address 分别映射到 Customer、OrderItem 和 Address Java Bean。

分析了模式定义后，接下来让我们回头来看看 WSDL 中的消息定义，如清单 8 中所示。WSDL 指定消息 processOrder 和 processOrderResponse，其所属的元素为 processOrder 和processOrderResponse（我们已经讨论了其模式定义）。portType 指定操作 processOrder，其输入消息为 processOrder，而输出消息为 processOrderResponse。

清单 8. WSDL 文档中的 processOrder 消息元素

Xml代码

```xml
<message name="processOrder">
   <part element="tns:processOrder" name="parameters" />
</message>
<message name="processOrderResponse">
   <part element="tns:processOrderResponse" name="parameters" />
</message>
<portType name="OrderProcessService">
<operation name="processOrder">
<input message="tns:processOrder" />
<output message="tns:processOrderResponse" />
</operation>
</portType>
```

接下来定义了 WSDL 绑定。此绑定将 soap:binding 样式定义为 document，soap:body 使用 literal 标记指定操作 processOrder 的输入和输出消息格式。生成的 WSDL 定义映射到 @SOAPBindingAnnotation（已在 OrderProcessService 类上定义，请参见清单 9）。

清单 9. WSDL 文档的绑定信息

Xml代码

```xml
<binding name="OrderProcessPortBinding" type="tns:OrderProcessService">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="processOrder">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
</binding>
```

接下来定义 WSDL 服务。这将指定端口和对应的绑定类型，以及服务的实际位置。此位置通常为 HTTP 位置，在本例中为 http://localhost:8080/OrderProcessWeb/orderprocess。可以在清单 10 中了解到具体的情况。

清单 10. WSDL 文档的服务信息

Xml代码

```xml
<service name="OrderProcess">
<port name="OrderProcessPort" binding="tns:OrderProcessPortBinding">
<soap:address location="http://localhost:8080/OrderProcessWeb/orderprocess" />
</port>
```

我们已经对生成的 WSDL 和模式构件进行了分析。清单 11 给出了一个示例 SOAP 请求消息，此消息是在 Web 服务客户机调用 processOrder 操作时发送的。

清单 11. processOrder 操作的示例 SOAP 消息

Xml代码

```xml
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1=" http://jawxs.ibm.tutorial/jaxws/orderprocess"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soapenv:Body>
<ns1:processOrder>
<arg0>
<customer><customerId>A123</customerId>
<firstName>John</firstName><lastName>Smith</lastName></customer>
<orderItems><itemId>11</itemId><qty>11</qty></orderItems>
</arg0>
</ns1:processOrder>
</soapenv:Body>
</soapenv:Envelope>
```

创建 Web 服务客户机

从 WSDL 创建 Web 服务客户机

在本部分，我们将了解如何从 WSDL 创建 Web 服务客户机。JAX-WS 提供了名为 wsimport 的工具，用于从 WSDL 生成 JAX-WS 可移

植构件。生成的可移植构件通常包括以下内容：

SEI
服务（需要实现的服务实现类）
从模式类型生成的 JAXB 生成类
从 wsdl:fault 映射的异常类（如果有）

客户机使用生成的构件调用 Web 服务。Web 服务客户机并不需要处理任何 SOAP(Simple Object Access Protocol) 格式（如创建或解析 SOAP 消息）。这将由 JAX-WS 运行时予以处理，此运行时将使用生成的构件代码（JAXB 生成类）。Web 服务将处理 Java 代码（JAXB 生成类），从而减少了开发 Web 服务客户机和对 Web 服务调用操作的工作。

先使用 wsimport 工具从 OrderProcess WSDL 生成 JAX-WS 构件。然后要创建 Web 服务客户机，后者使用生成的构件代码调用订单处理 Web 服务。运行清单 12 中所示的 wsimport 命令。不过，进行操作前，请确保已经按照生成 JAX-WS 构件部分中的步骤 5 所述的方法，通过运行 OrderWebServicePublisher 发布了 Web 服务。

清单 12. 用于生成供 Web 服务客户机使用的 JAX-WS 构件的 wsimport 命令

Java代码

```
wsimport -keep -p com.ibm.jaxws.tutorial.service.client
http://localhost:8080/OrderProcessWeb/orderprocess?wsdl
```

-keep 选项指示保留生成的文件，-p 选项指定需要在其中生成构件的包名称。http://localhost:8080/OrderProcessWeb/orderprocess?wsdl 指定 WSDL 文件的位置。以下构件是从 OrderProcessServiceWSDL 生成的：

JAXB 类（Address、Customer, OrderBean 和 OrderItem）：通过读取 OrderProcessService WSDL 中定义的模式定义生成
RequestWrapper 和 ResponseWrapper 类（ProcessOrder 和 ProcessOrderResponse）：包装 document literal-wrapped 样式类型的输入和输出
服务类 (OrderProcess)：客户机用于请求 Web 服务的类
服务接口 (OrderProcessService)：包含着用于服务实现接口的类
接下来了解一下如何使用上面生成的构件创建 Web 服务客户机。Web 服务客户机的代码如清单 13 中所示。

清单 13. 订单处理 Web 服务客户机的代码清单

Java代码

```
package com.ibm.jaxws.tutorial.service.client;

import java.net.MalformedURLException;
import java.net.URL;

import javax.xml.namespace.QName;

public class OrderClient {

final QName qName = new QName(
    "http://jawxs.ibm.tutorial/jaxws/orderprocess", "OrderProcess");

public static void main(String[] args) {
   if (args.length != 1) {
    System.out
      .println("Specify the URL of the OrderProcess Web Service");
|-------10--------20--------30--------40--------50--------60--------70--------80--------9|
|-------- XML error: The previous line is longer than the max of 90 characters ---------|
    System.exit(-1);
   }
   URL url = getWSDLURL(args[0]);
   OrderClient client = new OrderClient();
   client.processOrder(url);
}

private static URL getWSDLURL(String urlStr) {
```

```
   URL url = null;
   try {
    url = new URL(urlStr);
   } catch (MalformedURLException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
   }
   return url;
}

public void processOrder(URL url) {

   OrderProcess orderProcessingService = new OrderProcess(url, qName);

   System.out.println("Service is" + orderProcessingService);

   OrderBean order = populateOrder();

   OrderProcessService port = orderProcessingService.getOrderProcessPort();
   OrderBean orderResponse = port.processOrder(order);

   System.out.println("Order id is " + orderResponse.getOrderId());

}

private OrderBean populateOrder() {

   OrderBean order = new OrderBean();
   Customer customer = new Customer();
   customer.setCustomerId("A123");
   customer.setFirstName("John");
   customer.setLastName("Smith");
   order.setCustomer(customer);

   // Populate Order Item.
   OrderItem item = new OrderItem();
   item.setItemId("11");
   item.setQty(11);

   order.getOrderItems().add(item);
   return order;
}
}
```

运行 Web 服务客户机
上面列出的 Web 服务客户机代码执行以下任务：

通过传入 OrderProcess Web 服务的 WSDL URL 和服务的 QName 创建 OrderProcess 类的实例。
创建 OrderBean 的实例，并使用 populateOrder() 方法填充订单信息。
对服务调用 getOrderProcessPort()，以检索到服务的代理（也称为端口）。端口实现服务所定义的接口。
调用端口的 processOrder 方法，并同时传入在上面的第二个列表项目中创建的 OrderBean 实例。
从服务获得 OrderBean 响应并输出订单 ID。

运行 Web 服务客户机

javac com\ibm\jaxws\tutorial\service\client\OrderClient.java

通过使用以下命令提供订单处理 Web 服务的 WSDL URL 来运行 Web 服务客户机：

java com.ibm.jaxws.tutorial.service.client.OrderClient http://localhost:8080/OrderProcessWeb/orderprocess?wsdl

运行 Web 服务客户机时，会在服务端控制台看到以下输出（OrderWebServicePublisher 在控制台中运行）：

processOrder called for customer A123
Number of items is 1

在运行 Web 服务客户机的控制台中，会得到以下输出：

Order id is A1234

如上面的客户机代码中所示，并不会处理调用 Web 服务操作时使用的任何基于 SOAP 或 XML 的格式；相反，需要处理的是输入和输出消息的 JAXB 生成类，并使用服务接口和服务类对象（充当 Web 服务调用的存根）。存根负责从 JAXB Annotation 创建 SOAP 请求，并将 SOAP 响应转换回 Java 对象。

您现在已经成功地创建和发布了 Web 服务，并通过 Web 服务客户机成功地执行了此服务。

总结

在本教程中，我们了解了如何使用代码优先的开发方法和 JAX-WS 技术设计和开发 Web 服务。JAX-WS 是一个非常不错的选择，因为其中提供了完整的 Web 服务堆栈，以简化 Web 服务的开发和部署。

本教程中开发的订单处理 Web 服务使用 Document 样式的 Web 服务，可确保服务使用者和服务提供者使用 XML 文档进行通信。XML 文档遵循定义良好的契约，而此类契约通常都是使用 XML 模式定义创建的。XML 模式格式指定服务使用者能够调用和遵循的业务消息的契约。Document 样式的 Web 服务应该是开发企业 Web 服务的首选方法。

使用Apache CXF历程

09:36 | 浏览 (460) | 评论 (0) | 相关推荐

评论

发表评论

表情图标

B | I | U | Quote | Code | List | Img | URL | Flash | Table

字体颜色: 标准 字体大小: 标准 对齐: 标准

提示：选择您需要装饰的文字, 按上列按钮即可添加上相应的标签

您还没有登录，请登录后发表评论(快捷键 Alt+S / Ctrl+Enter) 提交