

# 1 学习目标

---

- (1) webservice基本概念
- (2) ApacheCXF框架介绍
- (3) JAX-WS规范下webservice开发
- (4) JAX-RS规范下webservice开发

## 2 webservice 基本概念

---

### 2.1 什么是web服务

---

这里列举一些常见的web服务：

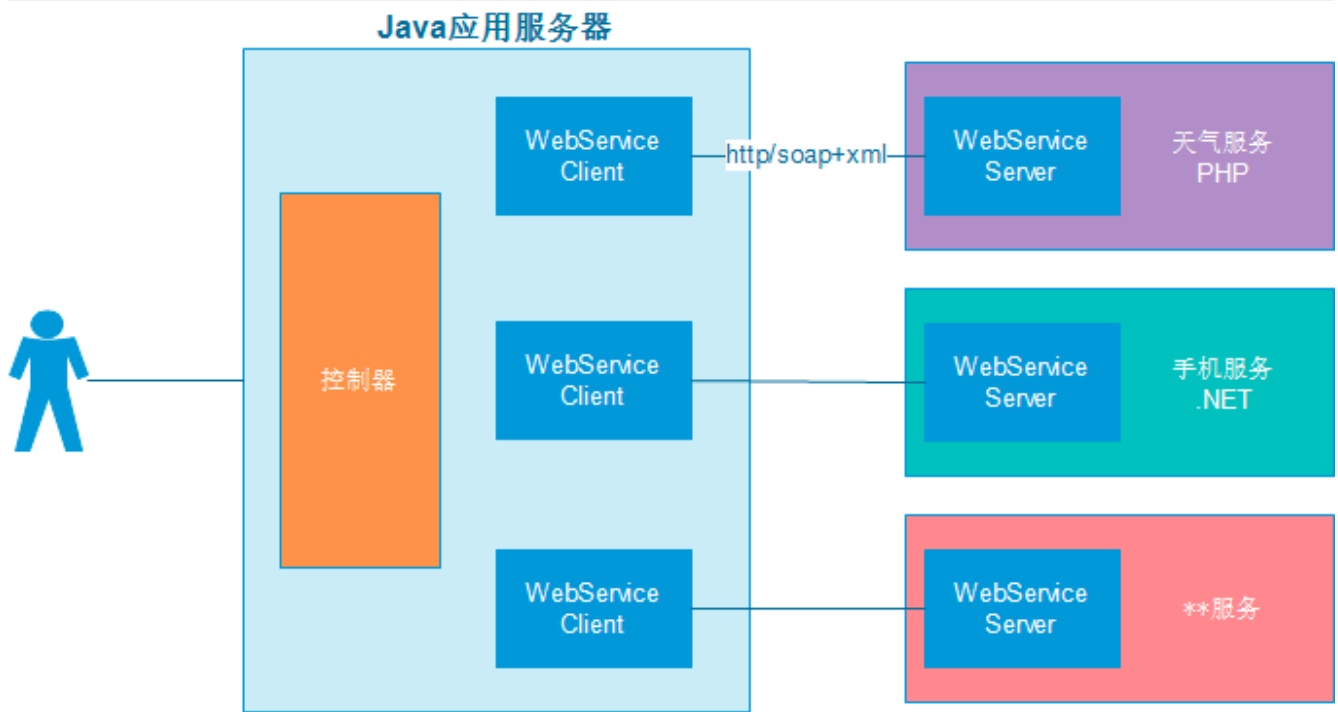
- 1. 手机淘宝、京东....
- 2. 天气预报
- 3. 手机号归属地
- 4. 股票查询
- 5. 发手机短消息
- 6. 手机充值功能
- 7. 中英文翻译
- 8. 银行转账业务
- 9. 公司的“进销存系统”在某商品缺货时自动给供应商下订单

### 2.2 简介

---

(1) webservice 即web服务，它是一种跨编程语言和跨操作系统平台的远程调用技术。

如图：



(2) JAVA 中共有三种WebService 规范，分别是JAX-WS（JAX-RPC）、JAXM&SAAJ、JAX-RS。

(3) webService三要素：soap、wsdl、uddi

注： 这里的一些概念将在下一章节详细讲解。

## 2.3 术语

### 2.3.1 webservice开发规范

JAVA 中共有三种WebService规范，分别是JAXM&SAAJ、JAX-WS（JAX-RPC）、JAX-RS。

下面来分别简要的介绍一下这三个规范。

(1.)JAX-WS:

JAX-WS（Java API For XML-WebService），JDK1.6 自带的版本为JAX-WS2.1，其底层支持为JAXB。JAX-WS（JSR 224）规范的API 位于javax.xml.ws.\*包，其中大部分都是注解，提供API 操作Web 服务（通常在客户端使用的较多，由于客户端可以借助SDK 生成，因此这个包中的API 我们较少会直接使用）。

(2.)JAXM&SAAJ:

JAXM (JAVA API For XML Message) 主要定义了包含了发送和接收消息所需的API，相当

于Web 服务的服务器端，其API 位于javax.messaging.\*包，它是JAVA EE 的可选包，因此

你需要单独下载。

SAAJ (SOAP With Attachment API For Java, JSR 67) 是与JAXM 搭配使用的API，为构建

SOAP 包和解析SOAP 包提供了重要的支持，支持附件传输，它在服务器端、客户端都需要

使用。这里还要提到的是SAAJ 规范，其API 位于javax.xml.soap.\*包。

JAXM&SAAJ 与JAX-WS 都是基于SOAP 的Web 服务，相比之下JAXM&SAAJ暴露了SOAP 更多的底层细节，编码比较麻烦，而JAX-WS 更加抽象，隐藏了更多的细节，更加面向对象，实现起来你基本上不需要关心SOAP 的任何细节。那么如果你想控制SOAP 消息的更多细节，可以使用JAXM&SAAJ，目前版本为1.3。

### (3.)JAX-RS:

JAX-RS 是JAVA 针对REST(RepresentationState Transfer)风格制定的一套Web 服务规范，

由于推出的较晚，该规范 (JSR 311，目前JAX-RS 的版本为1.0) 并未随JDK1.6 一起发行，

你需要到JCP 上单独下载JAX-RS 规范的接口，其API 位于javax.ws.rs.\*包。

这里的JAX-WS 和JAX-RS 规范我们采用Apache CXF 作为实现，CXF 是Objectweb Celtix 和Codehaus XFire 合并而成。CXF 的核心是org.apache.cxf.Bus (总线)，类似于Spring 的

ApplicationContext，Bus 由BusFactory 创建，默认是SpringBusFactory 类，可见默认CXF

是依赖于Spring 的，Bus 都有一个ID，默认的BUS 的ID 是cxf。你要注意的是Apache CXF

2.2 的发行包中的jar 你如果直接全部放到lib 目录，那么你必须使用JDK1.6，否则会报JAX-WS 版本不一致的问题。对于JAXM&SAAJ 规范我们采用JDK 中自带的默认实现。

## 2.3.2 SOAP 协议

(1) SOAP即简单对象访问协议(Simple Object Access Protocol)，它是用于交换XML（标准通用标记语言下的一个子集）编码信息的轻量级协议。它有三个主要方面：XML-envelope为描述信息内容和如何处理内容定义了框架，将程序对象编码成为XML对象的规则，执行远程过程调用(RPC)的约定。SOAP可以运行在任何其他传输协议上。

(2) SOAP作为一个基于XML语言的协议用于有网上传输数据。

(3) SOAP = 在HTTP的基础上+XML数据。

(4) SOAP是基于HTTP的。

(5) SOAP的组成如下

a) Envelope – 必须的部分。以XML的根元素出现。

b) Headers – 可选的。

c) Body – 必须的。在body部分，包含要执行的服务器的方法。和发送到服务器的数据。

## 2.3.3 wsdl说明书

Web Service描述语言WSDL（WebService Definition Language）就是用机器能阅读的方式提供的一个正式描述文档而基于XML（标准通用标记语言下的一个子集）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WSDL既是机器可阅读的，又是人可阅读的。

wsdl说明书，

wsdl说明书，

1) 通过wsdl说明书，就可以描述webservice服务端对外发布的服务；

2) wsdl说明书是一个基于xml文件，通过xml语言描述整个服务；

3) 在wsdl说明中，描述了：

对外发布的服务名称（类）

接口方法名称（方法）

接口参数（方法参数）

服务返回的数据类型（方法返回值）

## 2.3.4 UDDI

Web 服务提供商又如何将自己开发的 Web 服务公布到因特网上，

这就需要使用到 UDDI 了，UDDI的话，是一个跨产业，跨平台的开放性架构，可以帮助 Web 服务提供商在互联网上发布 Web 服务的信息。

UDDI 是一种目录服务，企业可以通过 UDDI 来注册和搜索 Web 服务。

简单来时候话，UDDI 就是一个目录，只不过在这个目录中存放的是一些关于 Web 服务的信息而已。

并且 UDDI 通过 SOAP 进行通讯，构建于 .Net 之上。

UDDI 即 Universal Description, Discovery and Integration，也就是通用的描述，发现以及整合。

UDDI 的目的是为电子商务建立标准；UDDI是一套基于Web的、分布式的、为WebService提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的Web Service注册，以使别的企业能够发现的访问协议的实现标准。

## 2.4 应用场景

Web Service 可以适用于应用程序集成、软件重用、跨防火墙通信等需求。不同的业务要求不同。具体如下：

(1) 跨防火墙通信

## (2) 应用系统集成

## (3) 软件和数据重用

简单来说，如果一个功能，需要被多个系统使用可以使用webservice开发一个服务端接口，供不同的客户端应用。主要应用在企业内部系统之间的接口调用、面向公网的webservice服务。

## 2.5 优缺点

### 2.5.1 优点：

#### a) 异构平台的互通性

理论上，Web Service 最大的优势是提供了异构平台的无缝衔接技术手段。由于不同的用户使用不同的硬件平台，不同的操作平台，不同的操作系统，不同的软件，不同的协议通信，这就产生了互相通信的需求。Web Service 使任何两个应用程序，只要能读写XML，那么就能互相通信。

#### b) 更广泛的软件复用(例如手机淘宝可以复用已有淘宝的业务逻辑.)

软件的复用技术通过组合已有模块来搭建应用程序，能大幅度提高软件的生产效率和质量。用户只要获得了描述 Web Service 的 WSDL 文件，就可以方便地生成客户端代理，并通过代理访问 Web Service 。

#### c) 成本低、可读性强、应用范围广

Web Service 可用基于 XML 的 SOAP 来表示数据和调用请求。并且通过 HTTP 协议传输 XML 格式的数据

#### d) 迅捷的软件发行方式

(每个web Service称为一个生产者.不同的生产者可以相互协同合作完成整个应用)

Web Service 将彻底地改变软件的发行方式。

软件供应商可以把软件分解成若Web Service 模块构成的系统，直接在 Web 上发布。

#### e) 最重要的一点

客户端与服务端可能是用不同的语言开发的，但是，通过webservice提供服务接口，客户端与服务端之前可以传递对象。

## 2.5.2 缺点：

由于soap是基于xml传输，本身使用xml传输会传输一些无关内容从而影响效率，随着soap协议的完善，soap协议增加了许多内容，这样就导致了使用soap去完成简单的数据传输而携带的信息更多效率再受影响；

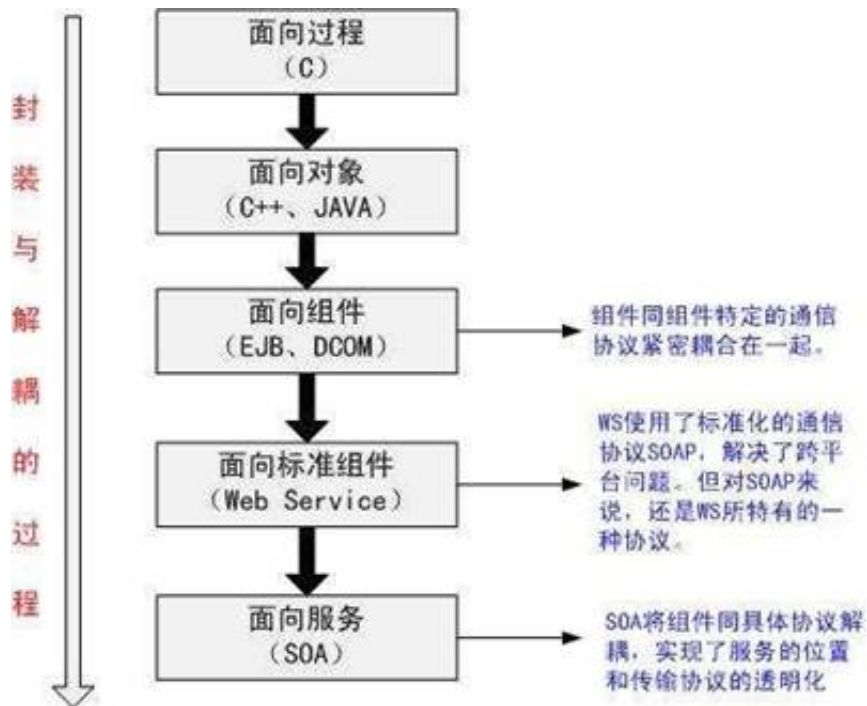
Web Service作为web跨平台访问的标准技术，很多公司都限定要求使用Web Service，但如果是简单的接口可以直接使用http传输自定义数据格式，开发更快捷。

## 2.6 面向服务架构SOA

SOA(Service-OrientedArchitecture)面向服务架构是一种思想，它将应用程序的不同功能单元通过中立的契约（独立于硬件平台、操作系统和编程语言）联系起来，使得各种形式的功能单元更好的集成。目前来说，WebService 是SOA 的一种较好的实现方  
WebService 采用HTTP 作为传输协议，SOAP（Simple Object Access Protocol）作为传输消息的格式。但WebService 并不是完全符合SOA 的概念，因为SOAP 协议是  
WebService 的特有协议，并未符合SOA 的传输协议透明化的要求。SOAP 是一种应用协议，早期应用于RPC 的实现，传输协议可以依赖于HTTP、SMTP 等。



SOA 的产生共经历了如下过程:



通常采用SOA 的系统叫做服务总线 (BUS)，结构如下图所示:





## 3 ApacheCXF 框架介绍

### 3.1 关于 Apache CXF

Apache CXF = Celtix + XFire，ApacheCXF 的前身叫 Apache CeltiXfire，现在已经正式更名为 Apache CXF 了，以下简称为 CXF。CXF 继承了 Celtix 和 XFire 两大开源项目的精华，提供了对 JAX-WS 全面的支持，并且提供了多种 Binding、DataBinding、Transport 以及各种 Format 的支持，并且可以根据实际项目的需要，采用代码优先（Code First）或者 WSDL 优先（WSDL First）来轻松地实现 Web Services 的发布和使用。目前它仍只是 Apache 的一个孵化项目。

Apache CXF 是一个开源的 Services 框架，CXF 帮助您利用 Frontend 编程 API 来构建和开发 Services，像 JAX-WS。这些 Services 可以支持多种协议，比如：SOAP、XML/HTTP、RESTfulHTTP 或者 CORBA，并且可以在多种传输协议上运行，比如：HTTP、JMS 或者 JBI，CXF 大大简化了 Services 的创建，同时它继承了 XFire 传统，一样可以天然地和 Spring 进行无缝集成。

### 3.2 功能特性

CXF 包含了大量的功能特性，但是主要集中在以下几个方面：

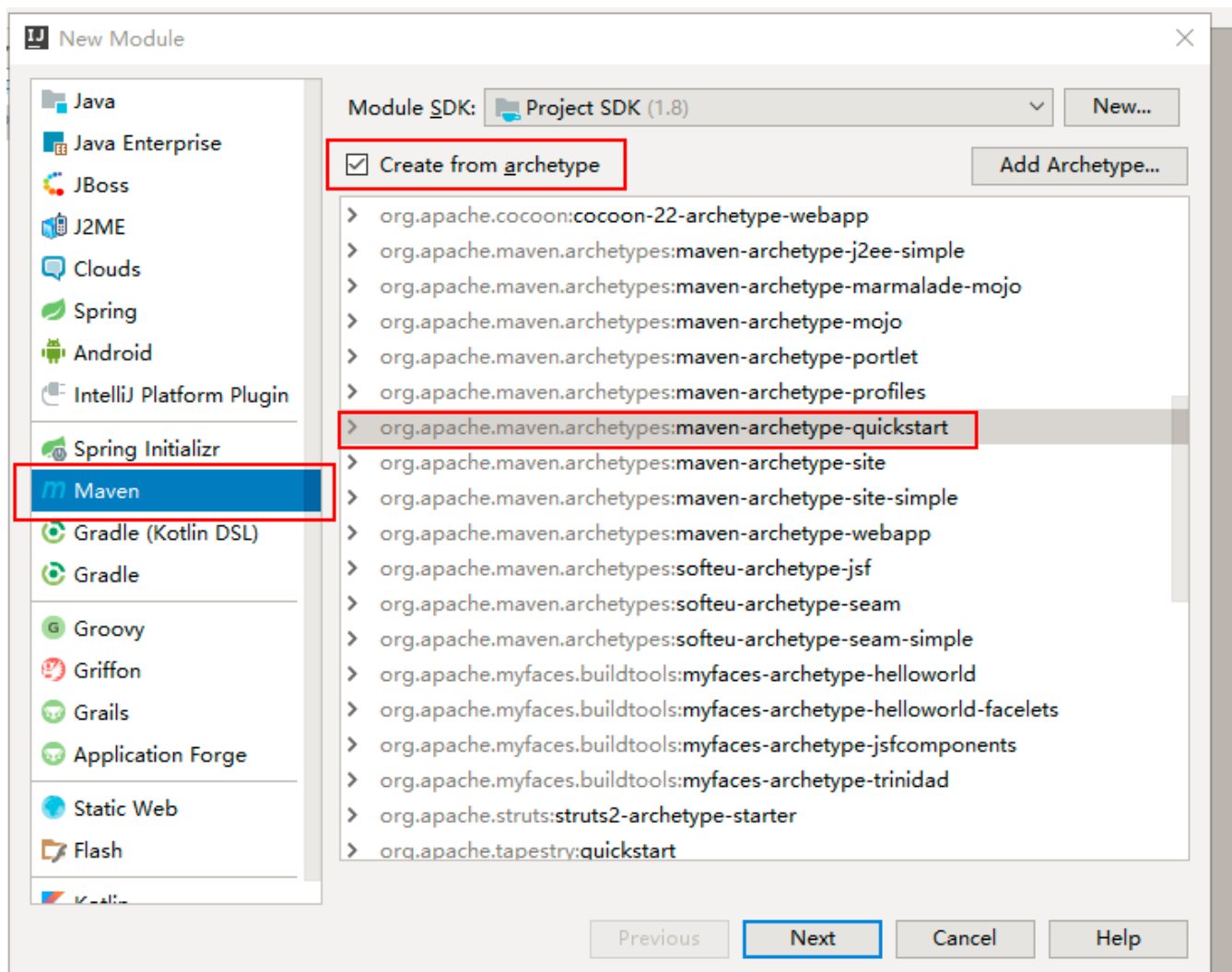
- (1) 支持 Web Services 标准：CXF 支持多种 Web Services 标准，包含 SOAP、Basic Profile、WS-Addressing、WS-Policy、WS-ReliableMessaging 和 WS-Security。
- (2) Frontends：CXF 支持多种“Frontend”编程模型，CXF 实现了 JAX-WS API（遵循 JAX-WS 2.0 TCK 版本），它也包含一个“simple frontend”允许客户端和 EndPoint 的创建，而不需要 Annotation 注解。CXF 既支持 WSDL 优先开发，也支持从 Java 的代码优先开发模式。
- (3) 容易使用：CXF 设计得更加直观与容易使用。有大量简单的 API 用来快速地构建代码优先的 Services，各种 Maven 的插件也使集成更加容易，支持 JAX-WS API，支持 Spring 2.0 更加简化的 XML 配置方式，等等。
- (4) 支持二进制和遗留协议：CXF 的设计是一种可插拨的架构，既可以支持 XML，也可以支持非 XML 的类型绑定，比如：JSON 和 CORBA。

## 4 ApacheCXF 实现 WebService (Jax-ws)

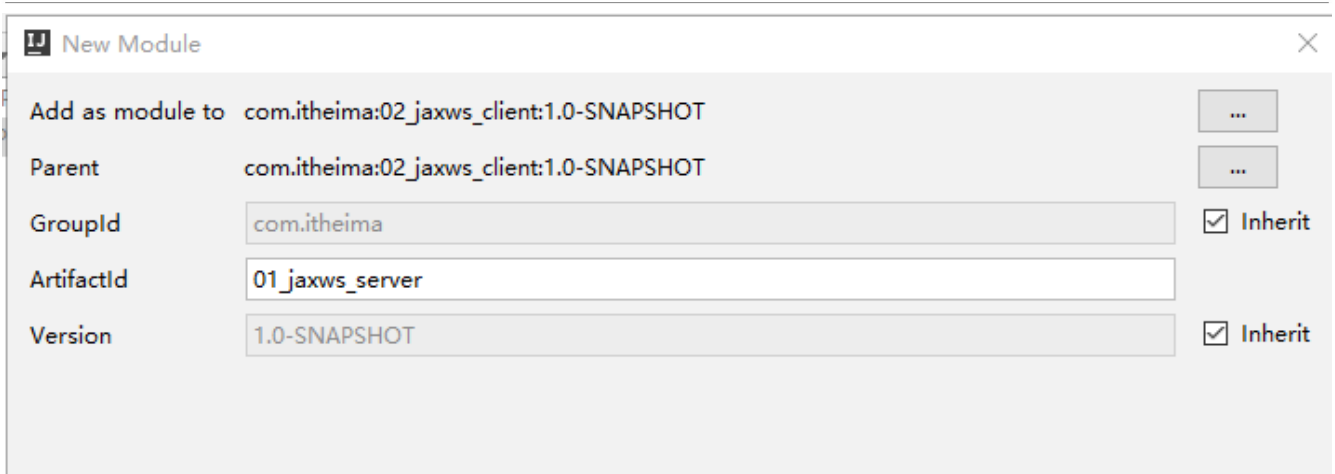
### 4.1 服务端

#### 4.1.1 创建项目

- file--->new --->module...



- next



New Module

Add as module to: com.itheima:02\_jaxws\_client:1.0-SNAPSHOT

Parent: com.itheima:02\_jaxws\_client:1.0-SNAPSHOT

GroupId: com.itheima

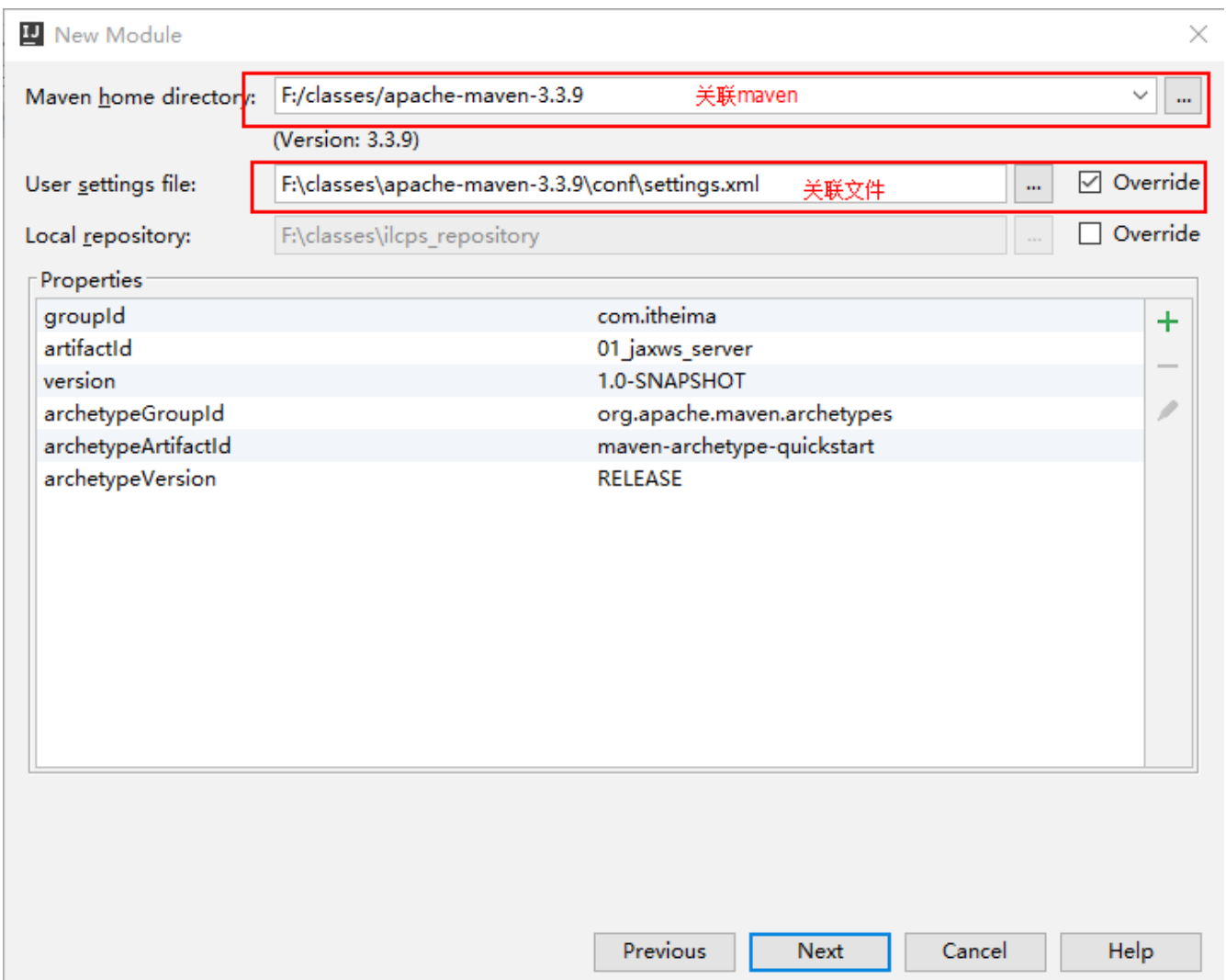
ArtifactId: 01\_jaxws\_server

Version: 1.0-SNAPSHOT

☒ Inherit

☒ Inherit

- next



New Module

Maven home directory: F:/classes/apache-maven-3.3.9 关联maven (Version: 3.3.9)

User settings file: F:\classes\apache-maven-3.3.9\conf\settings.xml 关联文件 ☒ Override

Local repository: F:\classes\ilcps\_repository ☐ Override

Properties

groupId	com.itheima
artifactId	01_jaxws_server
version	1.0-SNAPSHOT
archetypeGroupId	org.apache.maven.archetypes
archetypeArtifactId	maven-archetype-quickstart
archetypeVersion	RELEASE

Previous Next Cancel Help

## 4.1.2 添加cxf依赖

- 项目的pom.xml 添加如下依赖



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>cn.itcast</groupId>
    <artifactId>01_jaxws_server</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>服务端项目</name>

    <dependencies>
        <!-- 要进行jaxws 服务开发 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>3.0.1</version>
        </dependency>

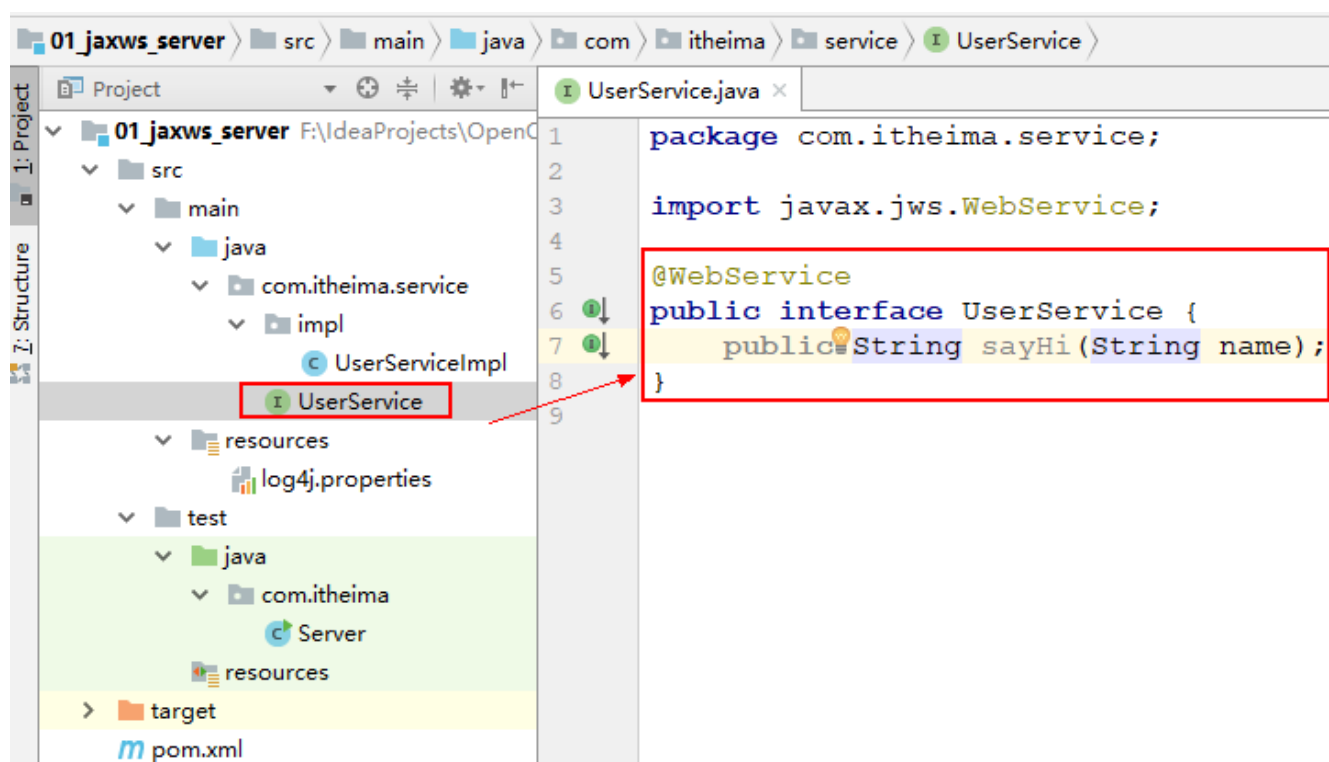
        <!-- 内置jetty web服务器 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>3.0.1</version>
        </dependency>

        <!-- 日志实现 -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.12</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```



```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-
plugin</artifactId>
        <version>3.2</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <encoding>UTF-8</encoding>
          <showWarnings>true</showWarnings>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>
```

### 4.1.3 写服务接口



## 4.1.4 写服务接口实现

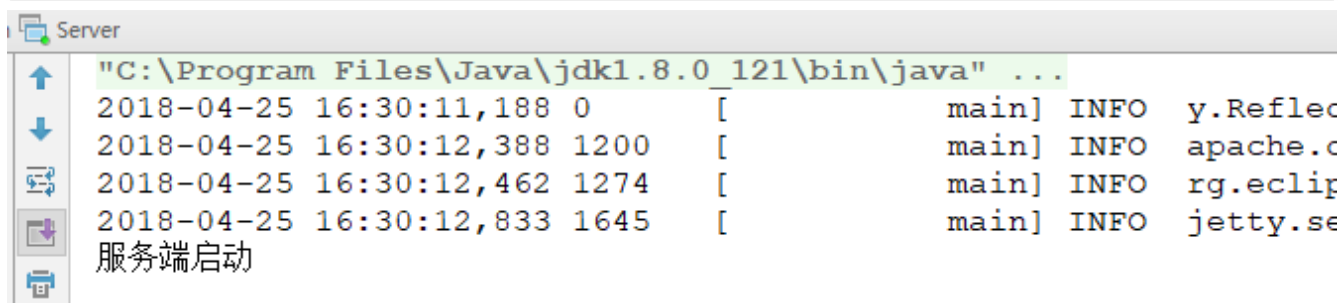


## 4.1.5 发布服务

```
public class Server {
    public static void main(String[] args) {
        // 服务工厂
        JaxWsServerFactoryBean factory = new JaxWsServerFactoryBean();
        // 设置服务地址
        factory.setAddress("http://localhost:8000/user");
        // 设置服务类
        factory.setServiceBean(new UserServiceImpl());
        // 发布服务
        factory.create();
        // 提示
        System.out.println("服务端启动");
    }
}
```

## 4.1.6 访问wsdl说明书

- 服务发布成功:



- 访问wsdl说明书

阅读顺序，从下往上读。



## 4.2 客户端

写客户端之前要先明确：

1. 服务端地址
2. 服务端接口、接口方法（方法参数、返回值）

### 4.2.1 创建项目



创建项目：02\_jaxws\_client

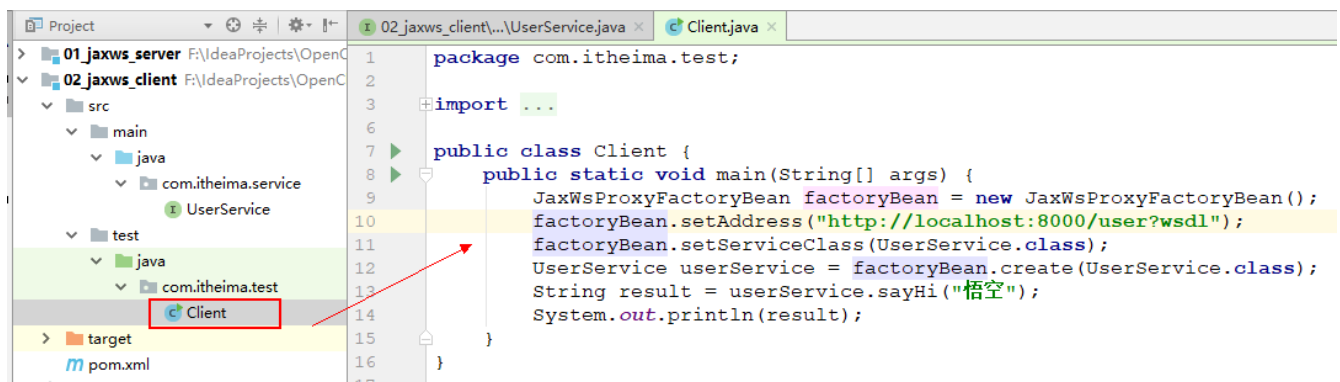
## 4.2.2 添加依赖

与服务端项目依赖一致。

## 4.2.3 服务接口



## 4.2.4 远程访问服务端



## 4.3 添加日志拦截器，观察soap协议内容

### 4.3.1 服务端

```
public class Server {  
    public static void main(String[] args) {  
        // 服务工厂  
        JaxWsServerFactoryBean factory = new JaxWsServerFactoryBean();  
        // 设置服务地址  
        factory.setAddress("http://localhost:8000/user");  
        // 设置服务类  
        factory.setServiceBean(new UserServiceImpl());  
        // 添加日志输入输出拦截器  
        factory.getInInterceptors().add(new LoggingInInterceptor());  
        factory.getOutInterceptors().add(new LoggingOutInterceptor());  
        // 发布服务  
        factory.create();  
        // 提示  
        System.out.println("服务端启动");  
    }  
}
```

### 4.3.2 客户端调用，观察日志

- 查看soap请求、soap响应传输的xml数据：

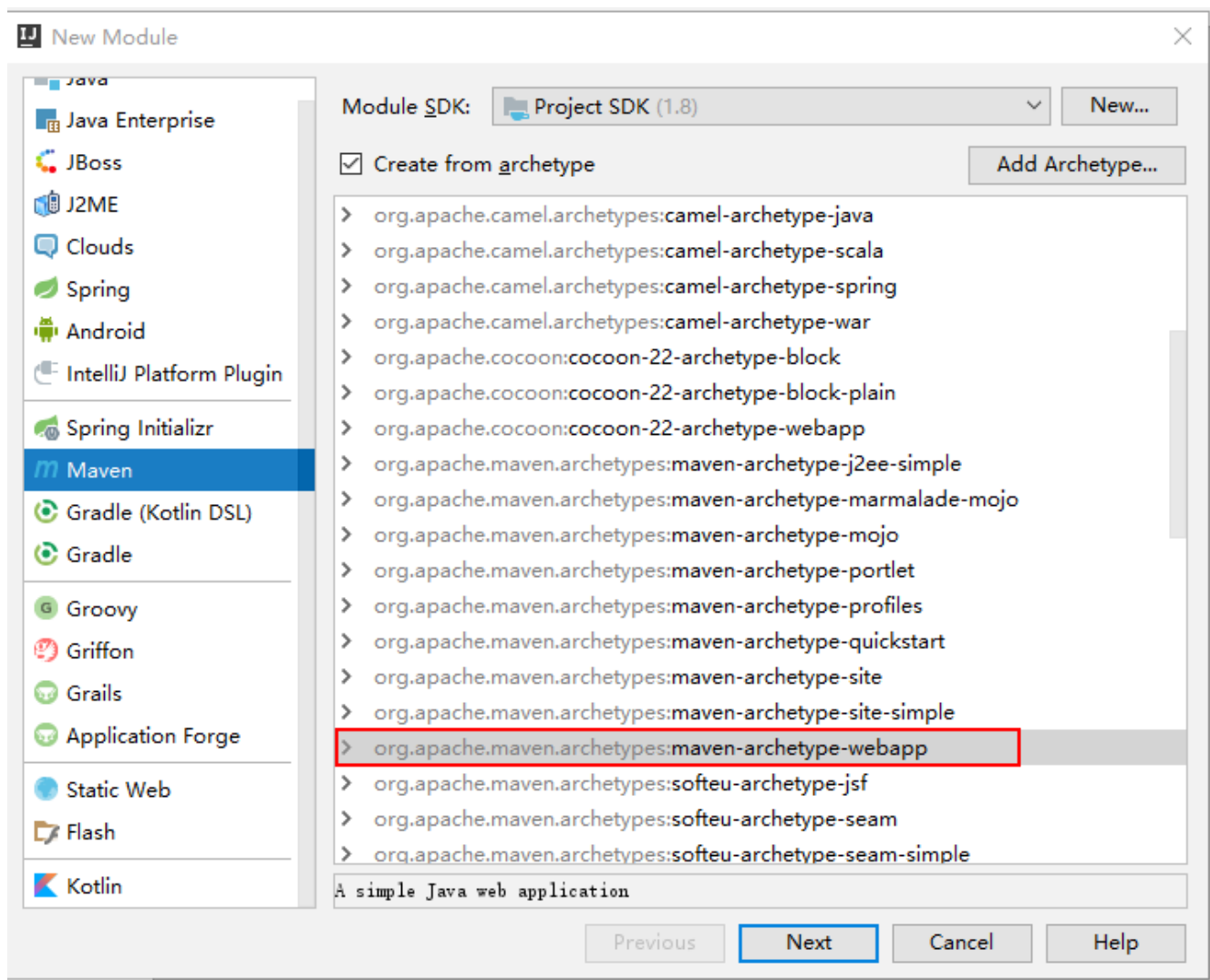
```
2018-04-10 10:52:30,439 72440 [tp2017085051-18] INFO
serServiceImplPort.UserService - Inbound Message
-----
ID: 4
Address: http://192.168.95.62:8000/userService
Encoding: UTF-8
Http-Method: POST      soap请求
Content-Type: text/xml; charset=UTF-8
Headers: {Accept=[*/*], Cache-Control=[no-cache], connection=[keep-
alive], Content-Length=[193], content-type=[text/xml; charset=UTF-8],
Host=[192.168.95.62:8000], Pragma=[no-cache], SOAPAction=[""], User-
Agent=[Apache CXF 3.0.1]}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<ns2:welcome xmlns:ns2="http://service.itcast.cn/"><arg0>请求数据</arg0>
</ns2:welcome></soap:Body></soap:Envelope>
-----
2018-04-10 10:52:30,442 72443 [tp2017085051-18] INFO
serServiceImplPort.UserService - Outbound Message
-----
ID: 4
Response-Code: 200      soap响应
Encoding: UTF-8
Content-Type: text/xml
Headers: {}
Payload: <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<ns2:welcomeResponse xmlns:ns2="http://service.itcast.cn/"><return>响应数
据</return></ns2:welcomeResponse></soap:Body></soap:Envelope>
-----
```

## 5 Spring 整合 ApacheCXF 实现 WebService (Jax-ws)

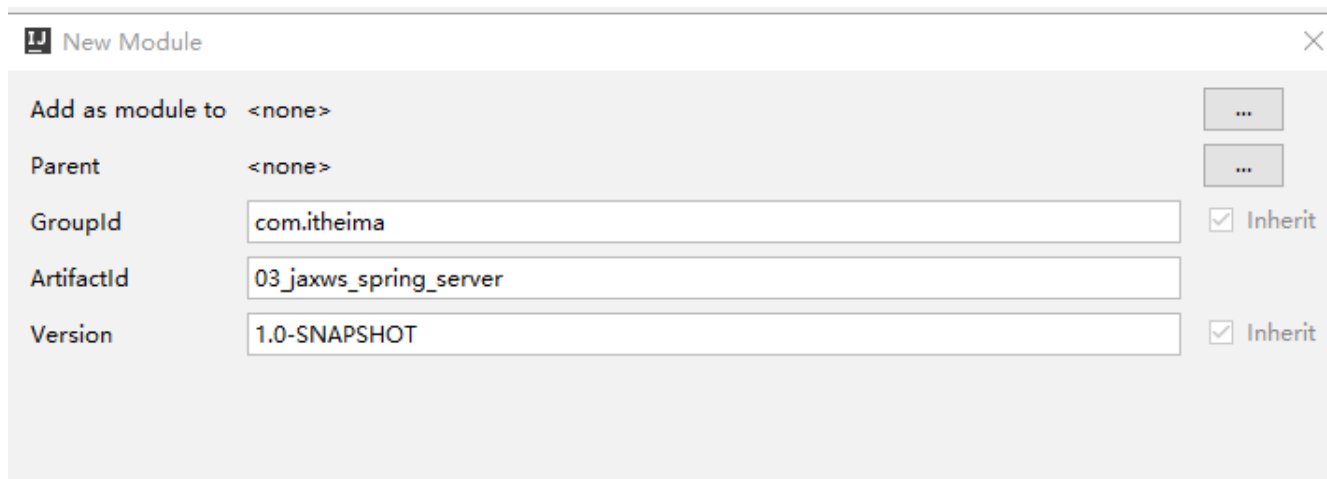
### 5.1 服务端

#### 5.1.1 创建web项目

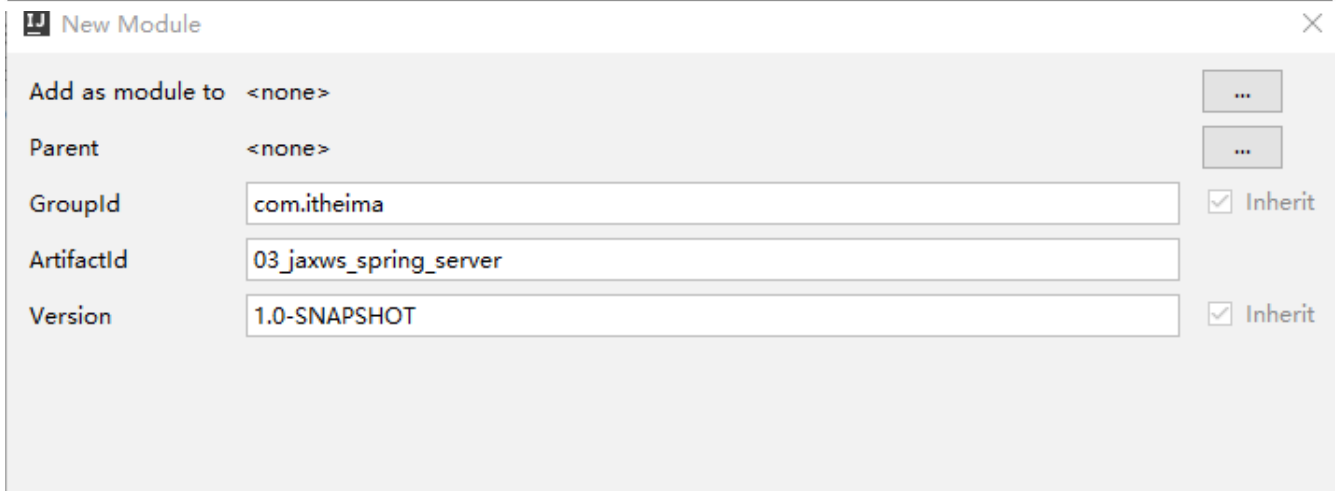
- file--->new --->module...



- next



- next



New Module

Add as module to: <none> ...

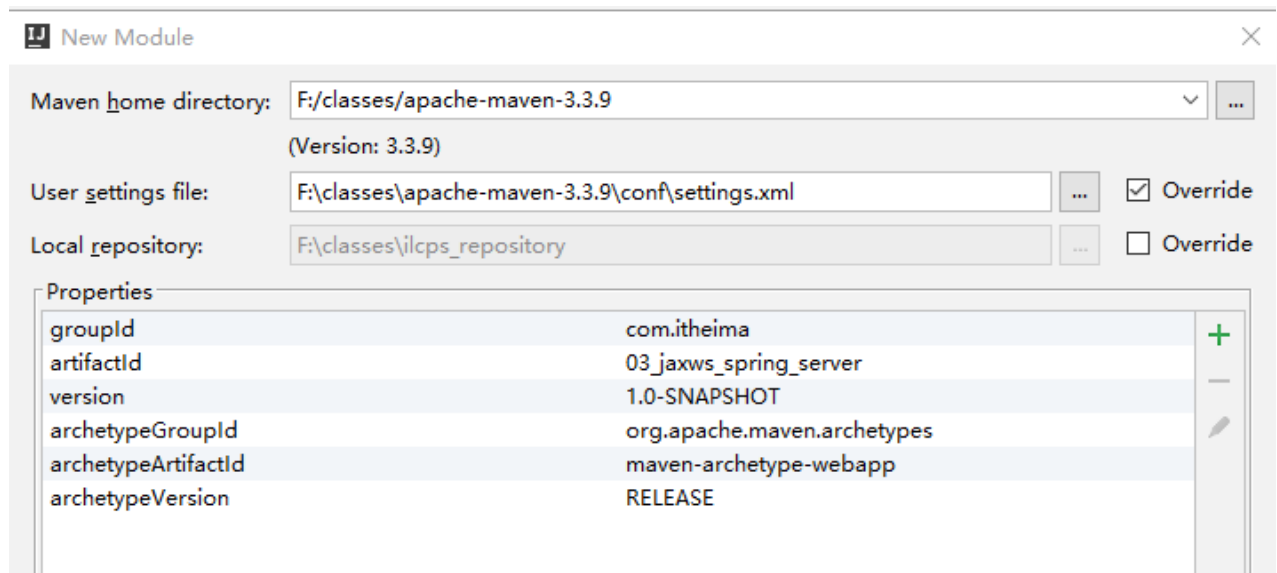
Parent: <none> ...

GroupId: com.itheima ☒ Inherit

ArtifactId: 03\_jaxws\_spring\_server

Version: 1.0-SNAPSHOT ☒ Inherit

- next



New Module

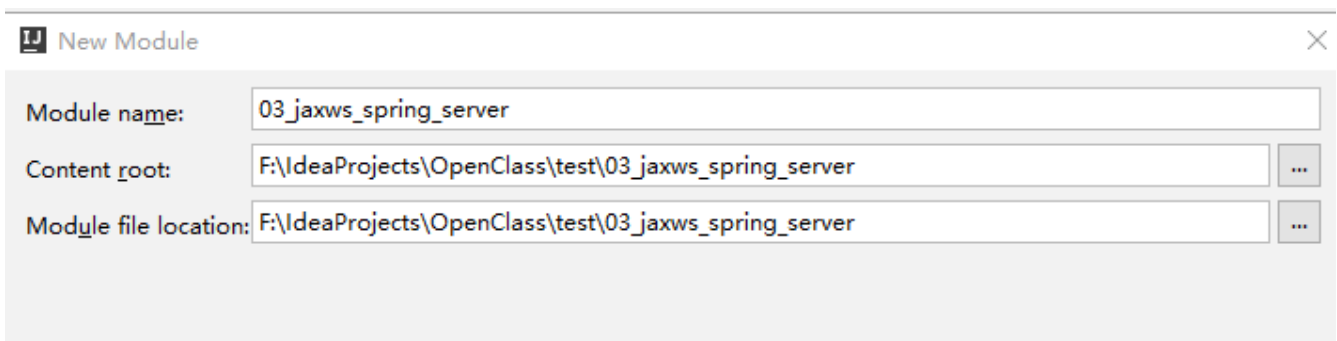
Maven home directory: F:/classes/apache-maven-3.3.9 (Version: 3.3.9) ...

User settings file: F:/classes/apache-maven-3.3.9/conf/settings.xml ☒ Override

Local repository: F:/classes/ilcps\_repository ☐ Override

Properties

groupId	com.itheima	+
artifactId	03_jaxws_spring_server	-
version	1.0-SNAPSHOT	
archetypeGroupId	org.apache.maven.archetypes	
archetypeArtifactId	maven-archetype-webapp	
archetypeVersion	RELEASE	



New Module

Module name: 03\_jaxws\_spring\_server

Content root: F:/IdeaProjects/OpenClass/test/03\_jaxws\_spring\_server ...

Module file location: F:/IdeaProjects/OpenClass/test/03\_jaxws\_spring\_server ...

## 5.1.2 添加依赖

- pom.xml

## 5.1.3 web.xml 配置CXFServlet



```
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd" >

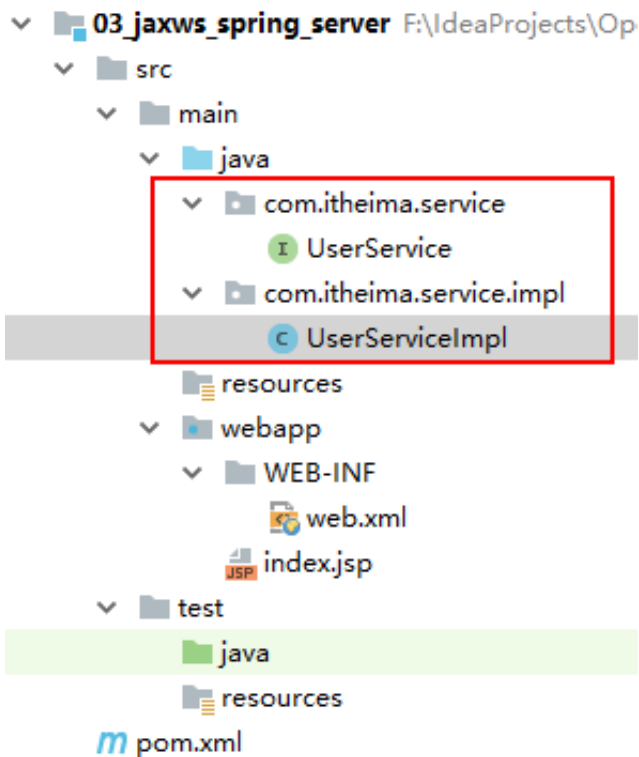
<web-app>
    <display-name>Archetype Created Web Application</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
        </listener>

    <!-- webservice服务端，发布服务需要配置CXFServlet -->
    <!-- 这里配置的servlet路径，最为最终服务路径的一部分： -->
    <!-- 服务访问路径：http://localhost:8080/web.xml配置路径/spring配置的路径 -->
    <servlet>
        <servlet-name>cxfservlet</servlet-name>
        <servlet-
class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxfservlet</servlet-name>
        <url-pattern>/ws/*</url-pattern>
    </servlet-mapping>

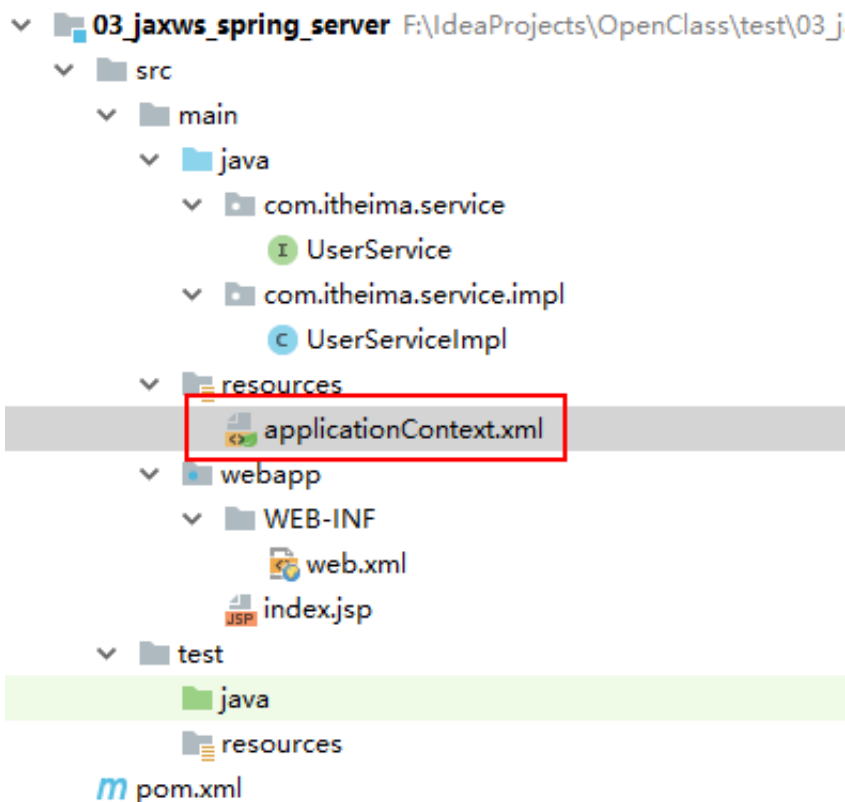
</web-app>
```

## 5.1.4 服务接口、服务实现

项目目录结构：



## 5.1.5 Spring 整合 ApacheCXF





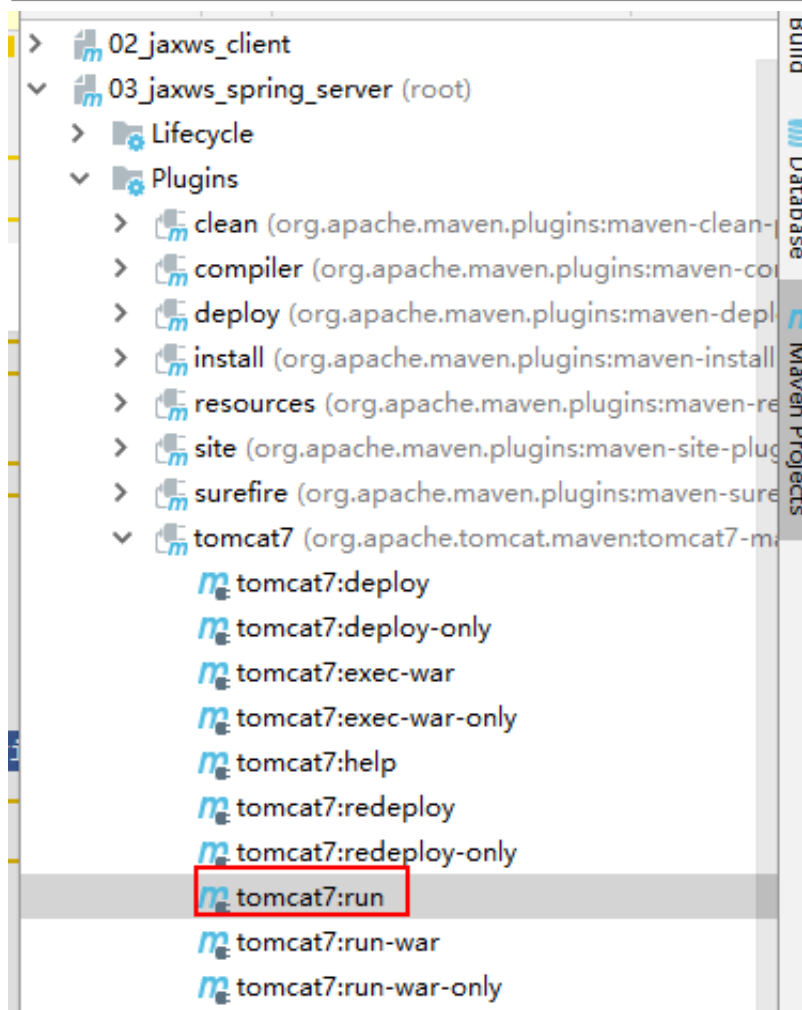


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cxf="http://cxf.apache.org/core"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://cxf.apache.org/core
    http://cxf.apache.org/schemas/core.xsd
    http://cxf.apache.org/jaxws
    http://cxf.apache.org/schemas/jaxws.xsd
    http://cxf.apache.org/jaxrs
    http://cxf.apache.org/schemas/jaxrs.xsd">

  <!--
    Spring整合ApacheCXF，发布jaxws服务：
    1. 服务地址
    2. 服务bean

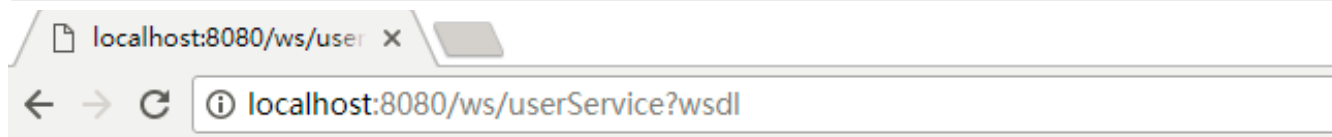
    完整服务地址：
    http://localhost:8080/ws/userService
  -->
  <jaxws:server address="/userService">
    <jaxws:serviceBean>
      <bean class="com.itheima.service.impl.UserServiceImpl">
</bean>
    </jaxws:serviceBean>
  </jaxws:server>
</beans>
```

## 5.1.6 启动服务，发布服务



## 5.1.7 访问wsdl说明书

地址:



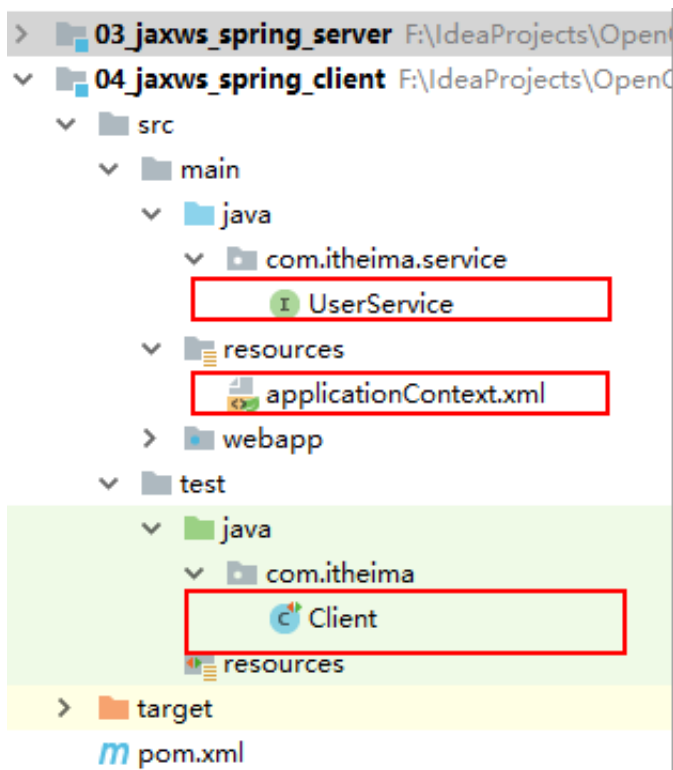
This XML file does not appear to have any style information associated with it. The

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:ns2="http://schemas.xmlsoap.org/soap/http" xmlns:ns1="http://service.itheima.com/" name="UserService" targetNamespace="http://service.itheima.com/">
  <wsdl:import location="http://localhost:8080/ws/userService?wsdl=UserService.wsdl" namespace="http://service.itheima.com/" />
  <wsdl:binding name="UserServiceImplServiceSoapBinding" type="ns1:UserService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="sayHi">
      <soap:operation soapAction="" style="document" />
      <wsdl:input name="sayHi">
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output name="sayHiResponse">
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="UserServiceImplService">
    <wsdl:port binding="tns:UserServiceImplServiceSoapBinding" name="UserServiceImplPort">
      <soap:address location="http://localhost:8080/ws/userService" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

## 5.2 客户端

### 5.2.1 创建项目

复制03项目创建04\_jaxws\_spring\_client项目



web.xml配置内容可以删除。服务实现也删除。

## 5.2.2 添加依赖

复制项目时候已经完成。（与03\_jaxws\_spring\_server中pom.xml配置一样）

## 5.2.3 service接口

复制项目时候已经完成。

## 5.2.4 Spring 整合 ApacheCXF 配置



```
<?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:cxf="http://cxf.apache.org/core"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://cxf.apache.org/core
    http://cxf.apache.org/schemas/core.xsd
    http://cxf.apache.org/jaxws
    http://cxf.apache.org/schemas/jaxws.xsd
    http://cxf.apache.org/jaxrs
    http://cxf.apache.org/schemas/jaxrs.xsd">

  <!--
    Spring整合ApacheCXF，客户端配置
    关键点：
        通过Spring整合ApacheCXF，创建客户端的代理对象，远程访问服务端。
    jaxws:client
        id 应用中注入的接口的代理对象的名称
        address 服务端访问地址
        serviceClass 指定接口路径，会根据该类型生成代理对象
  -->
  <jaxws:client
    id="userService"
    address="http://localhost:8080/ws/userService"
    serviceClass="com.itheima.service.UserService">
  </jaxws:client>
</beans>
```

## 5.2.5 junit测试



```
package com.itheima;

import com.itheima.service.UserService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import javax.annotation.Resource;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class Client {
    // 注入远程访问服务端的接口的代理对象。
    @Resource
    private UserService userService;

    @Test
    public void test() throws Exception {
        // org.apache.cxf.jaxws.JaxWsClientProxy@2826f61
        System.out.println(userService);
        // class com.sun.proxy.$Proxy45
        System.out.println(userService.getClass());

        // 远程调用服务接口
        String content = userService.sayHi("球球");
        System.out.println(content);
    }
}
```

## 6 Restful 风格

一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务端交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

REST（英文：Representational State Transfer，简称REST）描述了一个架构样式的网络系统，比如 web 应用程序。它首次出现在 2000 年 Roy Fielding 的博士论文中，他是 HTTP 规范的主要编写者之一。在目前主流的三种Web服务交互方案中，REST相比于 SOAP（Simple Object Access protocol，简单对象访问协议）以及XML-RPC更加简单明了，无论是对URL的处理还是对Payload的编码，REST都倾向于用更加简单轻量的方法设计和实现。值得注意的是REST并没有一个明确的标准，而更像是一种设计的风格。

所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在。你可以用一个URI指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或独一无二的识别符。所谓"上网"，就是与互联网上一系列的"资源"互动，调用它的URI。

表现层，我们把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。

比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

Restful 风格：<http://bbs.csdn.net/topics/390908212>

非resfull风格：<http://bbs.csdn.net/topics?tid=390908212>

严格地说，有些网址最后的".html"后缀名是不必要的，因为这个后缀名表示格式，属于"表现层"范畴，而URI应该只代表"资源"的位置。它的具体表现形式，应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。

状态转化（State Transfer）

访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和状态的变化。互联网通信协议HTTP协议，是一个无状态协议。这意味着，所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（StateTransfer）。而这种转化是建立在表现层之上的，所以就是"表现层状态转化"。客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：



GET用来获取资源，

POST用来新建资源，

PUT用来更新资源，

DELETE用来删除资源。

访问服务器资源，通过不同的http请求方式，服务器就知道对CRUD的哪个操作！

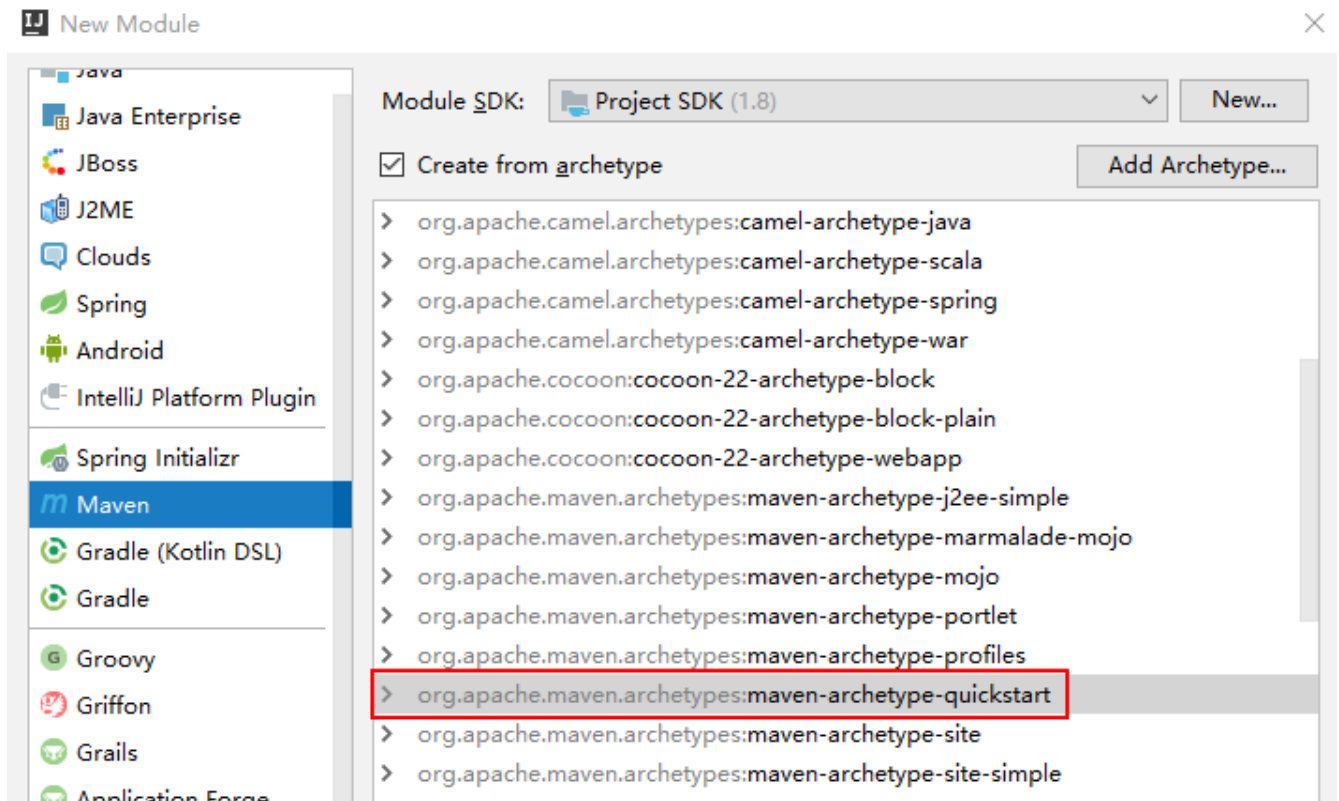
JAX-RS 发布服务就是使用RESTFUL风格。

## 7 ApacheCXF 实现webservice (Jax-rs)

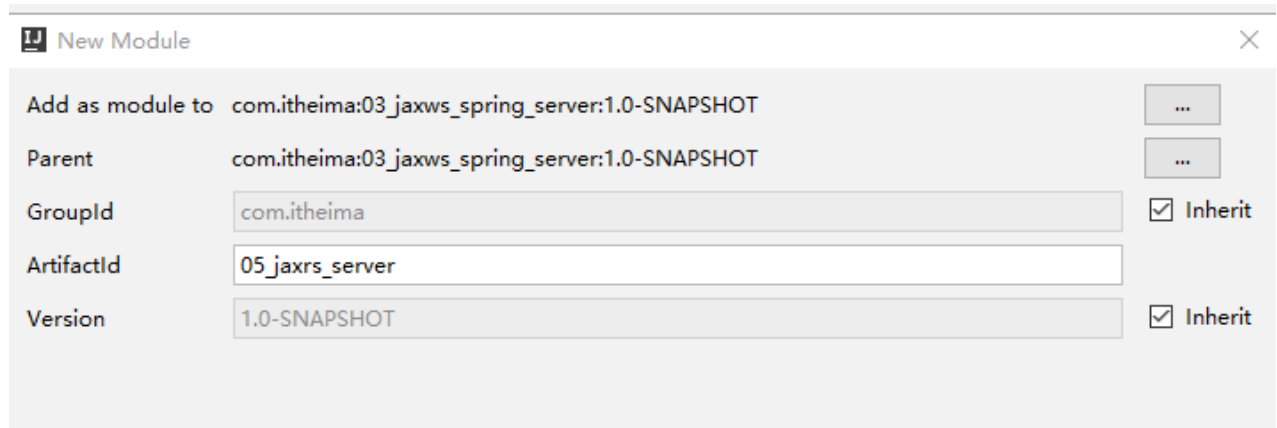
- 基于restful风格的webservice，请求使用的是http协议，可以传递xml/json数据

### 7.1 服务端

#### 7.1.1 创建项目



- next



## 7.1.2 添加依赖

- pom.xml



```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <groupId>com.itheima</groupId>
  <artifactId>05_jaxrs_server</artifactId>
  <version>1.0-SNAPSHOT</version>
  <modelVersion>4.0.0</modelVersion>
  <name>05_jaxrs_server</name>

  <dependencies>
    <!-- jaxrs 支持包 -->
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-frontend-jaxrs</artifactId>
      <version>3.0.1</version>
    </dependency>

    <!-- 内置的jetty服务器 -->
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-transports-http-jetty</artifactId>
      <version>3.0.1</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.7.12</version>
    </dependency>

    <!-- 客户端调用时候使用的包(WebClient工具类调用服务端) -->
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-rs-client</artifactId>
      <version>3.0.1</version>
    </dependency>
  </dependencies>
</project>
```



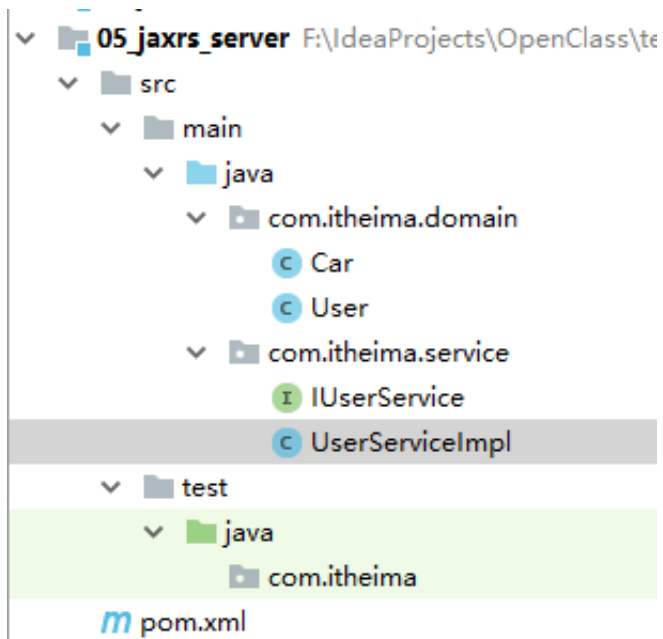
```
<!-- 基于restful风格的webservice，客户端与服务端之间可以传递
json，这个就是json支持相关包 -->
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-rs-extension-providers</artifactId>
    <version>3.0.1</version>
</dependency>
<dependency>
    <groupId>org.codehaus.jettison</groupId>
    <artifactId>jettison</artifactId>
    <version>1.3.7</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
</dependency>
</dependencies>

<build>
    <pluginManagement>
        <plugins>
            <!-- maven的jdk编译插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                    <showWarnings>true</showWarnings>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>

</project>
```

## 7.1.3 服务接口、实现、实体类

### 7.1.3.2 准备代码





```
/**
 * @XmlElement 指定根元素，作用：客户端与服务端传递对象数据时候，序列化为
xml或json的根元素的名称
 * 客户端与服务端传递XML：
 *    <Car1></Car> 这里的根元素Car是由name="Car"指定的
 *
 * 客户端与服务端传递JSON：
 *    {"Car": {"id":100,"name":"","price":100}}
 *
 */
@XmlRootElement(name = "Car")
public class Car {
    private Integer id;
    private String carName;
    private Double price;    省略get、set
}

@XmlRootElement(name = "User")
public class User {
    private Integer id;
    private String username;
    private String city;
    private List<Car> cars = new ArrayList<Car>();    省略get、set
}

@Path("/userService")    // 路径：访问当前服务接口时候的路径。
@Produces("*/*")
public interface IUserService {

    @POST
    @Path("/user")        // 路径：访问当前服务接口的方法路径
    // @Consumes 服务端支持的请求的数据格式(xml、json)
    @Consumes({ "application/xml", "application/json" })
    public void saveUser(User user);

    @PUT
    @Path("/user")
    @Consumes({ "application/xml", "application/json" })
    public void updateUser(User user);

    @GET
```



```
@Path("/user")
// @Produces 服务端支持的响应的数据格式
@Produces({ "application/xml", "application/json" })
public List<User> findAllUsers();

@GET
@Path("/user/{id}")
@Consumes("application/xml")
@Produces({ "application/xml", "application/json" })
public User finUserById(@PathParam("id") Integer id);

@DELETE
@Path("/user/{id}")
@Consumes({ "application/xml", "application/json" })
public void deleteUser(@PathParam("id") Integer id);
}

public class UserServiceImpl implements IUserService {

    public void saveUser(User user) {
        System.out.println("save user:" + user);
    }

    public void updateUser(User user) {
        System.out.println("update user:" + user);
    }

    public List<User> findAllUsers() {
        List<User> users = new ArrayList<User>();

        // 汽车
        List<Car> carList1 = new ArrayList<Car>();
        Car car1 = new Car(101, "保时捷", 1000000d);
        Car car2 = new Car(102, "林肯", 400000d);
        carList1.add(car1);
        carList1.add(car2);

        // 用户
        User user1 = new User(1, "小明", "广州", carList1);
        User user2 = new User(2, "小丽", "深圳", carList1);
    }
}
```



```
// 用户集合
users.add(user1);
users.add(user2);

return users;
}

public User finUserById(Integer id) {
    if (id == 1) {
        return new User(1, "小明", "广州", null);
    }
    return null;
}

public void deleteUser(Integer id) {
    System.out.println("delete user id :" + id);
}

}
```

### 7.1.3.2 使用的注解分析

使用的注解：

@XmlRootElement

指定根元素，

作用：客户端与服务端传递对象数据时候，序列化为xml或json的根元素的名称

@Path("/userService")

路径：访问当前服务接口时候的路径、接口方法的路径。

@POST insert操作

@PUT update操作

@GET select 查询操作

@DELETE delete删除操作

@Consumes 服务端支持的请求的数据格式(xml、json)

@Produces 服务端支持的响应的数据格式

## 7.1.4 发布服务



```
package com.itheima;

import com.itheima.service.UserServiceImpl;
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;

public class Server {

    /**
     * 发布restful风格的webservice的服务
     */
    public static void main(String[] args) {
        //1.创建服务工厂
        JAXRSServerFactoryBean factory = new JAXRSServerFactoryBean();

        //2.设置服务地址、
        factory.setAddress("http://localhost:8001/rs");

        //3.实例化服务类、
        factory.setServiceBean(new UserServiceImpl());

        // 添加日志拦截器
        factory.getInInterceptors().add(new LoggingInInterceptor());
        factory.getOutInterceptors().add(new LoggingOutInterceptor());

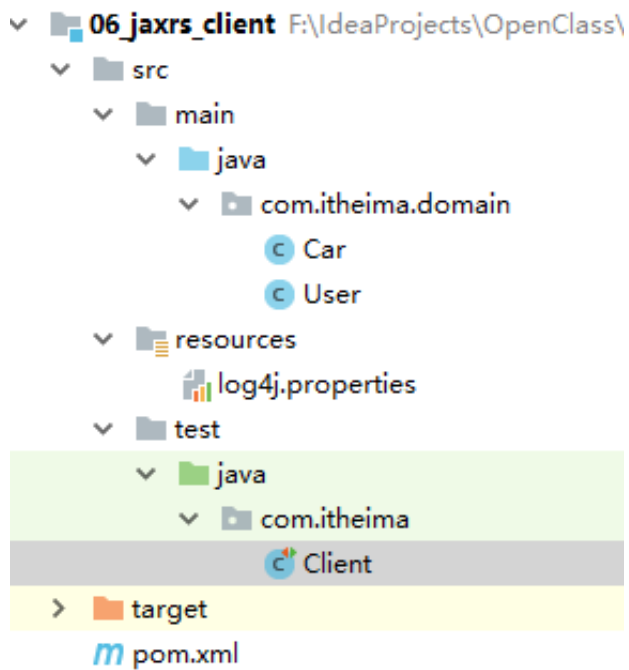
        //4.创建服务
        factory.create();

        System.out.println("发布服务成功..8001");
    }
}
```

## 7.2 客户端

## 7.2.1 创建项目

创建项目



## 7.2.2 添加依赖



```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <groupId>com.itheima</groupId>
    <artifactId>06_jaxrs_client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <modelVersion>4.0.0</modelVersion>
    <name>06_jaxrs_client</name>

    <dependencies>
        <!-- jaxrs 支持包 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>3.0.1</version>
        </dependency>

        <!-- 内置的jetty服务器 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>3.0.1</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.12</version>
        </dependency>

        <!-- 客户端调用时候使用的包(WebClient工具类调用服务端) -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-rs-client</artifactId>
            <version>3.0.1</version>
        </dependency>
    </dependencies>
</project>
```



<!-- 基于restful风格的webservice，客户端与服务端之间可以传递json，这个就是json支持相关包 -->

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-rs-extension-providers</artifactId>
  <version>3.0.1</version>
</dependency>
<dependency>
  <groupId>org.codehaus.jettison</groupId>
  <artifactId>jettison</artifactId>
  <version>1.3.7</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.10</version>
  <scope>test</scope>
</dependency>
</dependencies>
```

```
<build>
  <pluginManagement>
    <plugins>
      <!-- maven的jdk编译插件 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.2</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <encoding>UTF-8</encoding>
          <showWarnings>true</showWarnings>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>
```

## 7.2.3 写junit，远程访问服务端

WebClient

- .create() 指定服务端地址
- .type() 指定请求数据格式（xml、json）
- .accept() 指定响应数据格式
- .post()/put()/delete()/get() 指定请求类型



```
package com.itheima;

import com.itheima.domain.User;
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;
import org.apache.cxf.jaxrs.client.WebClient;
import org.junit.Test;

import javax.ws.rs.core.MediaType;
import java.util.Collection;

public class Client {

    @Test
    public void save() throws Exception {
        // 基于restful风格的webservice开发的客户端调用，直接通过一个类：
        // WebClient类完成
        WebClient
            .create("http://localhost:8001/rs/userService/user") //
            // 地址
            .type(MediaType.APPLICATION_JSON) //
            // 请求数据格式是json
            .post(new User(100, "Kobe", "gz", null));
        // 发送请求的类型
    }

    @Test
    public void update() throws Exception {
        WebClient
            .create("http://localhost:8001/ws/userService/user") //
            // 地址
            .type(MediaType.APPLICATION_JSON) //
            // 请求数据格式是json
            .put(new User(100, "Kobe", "gz", null));
        // 发送请求的类型
    }

    @Test
    public void delete() throws Exception {
```



```
}

@Test
public void findOne() throws Exception {
    User user =
        WebClient

.create("http://localhost:8001/ws/userService/user/1") // 地址
        .accept(MediaType.APPLICATION_JSON) // 响应的数据
        格式
        .get(User.class);
    System.out.println(user);
}

@Test
public void findAll() throws Exception {
    Collection<? extends User> collection =
        WebClient

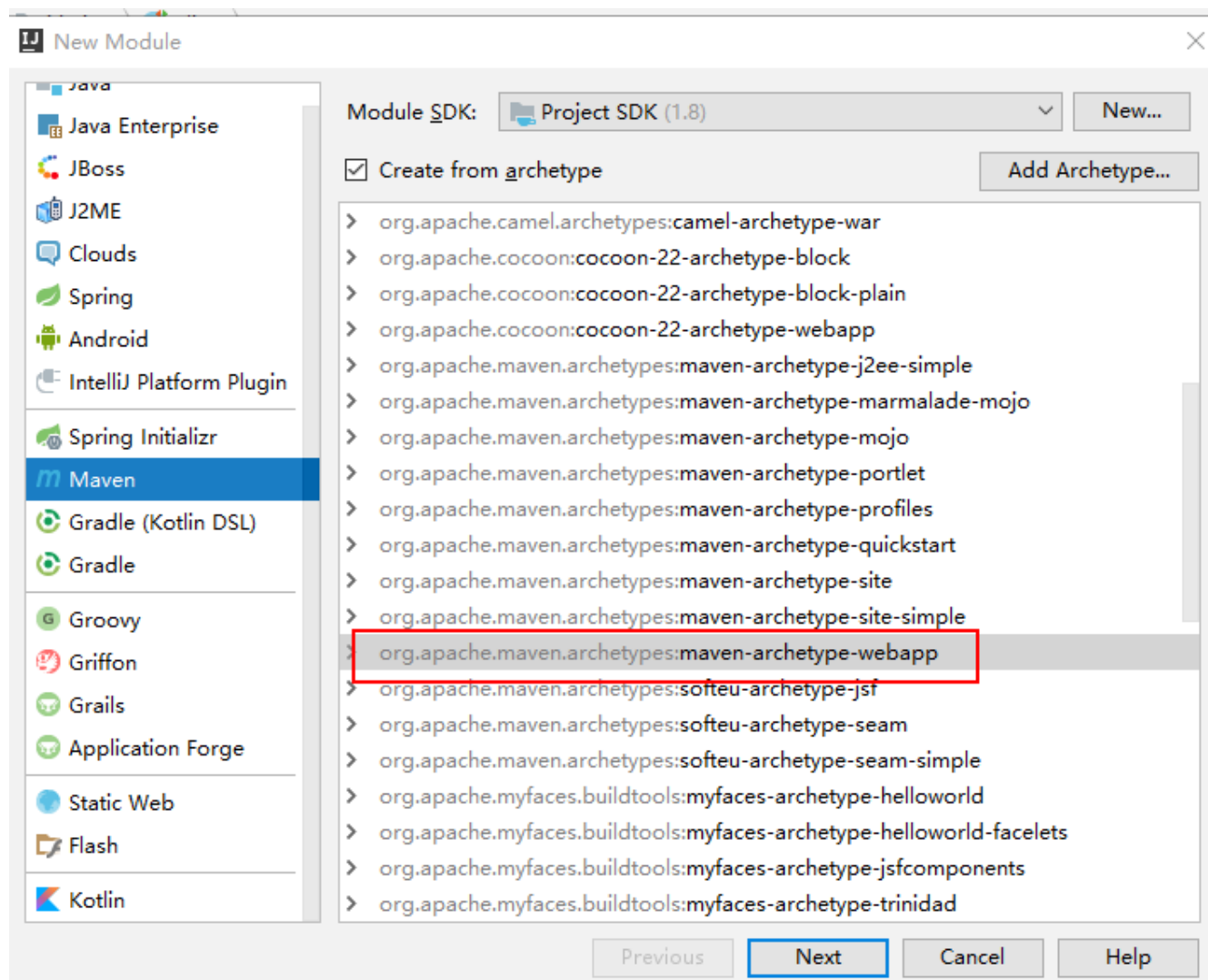
.create("http://localhost:8001/ws/userService/user")
        .accept(MediaType.APPLICATION_JSON)
        .getCollection(User.class);
    System.out.println(collection);
}

}
```

## 8 Spring整合CXF实现基于Restful风格的webservice(jax-rs)

### 8.1 服务端

## 8.1.1 创建web项目



## 8.1.2 添加依赖



```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.itheima</groupId>
    <artifactId>07_jaxrs_spring_server</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>07_jaxrs_spring_server</name>

    <dependencies>
        <!-- cxf 进行rs开发 必须导入 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>3.0.1</version>
        </dependency>
        <!-- 日志引入 -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.12</version>
        </dependency>

        <!-- 客户端 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-rs-client</artifactId>
            <version>3.0.1</version>
        </dependency>

        <!-- 扩展json提供者 -->
        <dependency>
            <groupId>org.apache.cxf</groupId>

            <artifactId>cxf-rt-rs-extension-providers</artifactId>
```



```
<version>3.0.1</version>
</dependency>

<!-- 转换json工具包，被extension providers 依赖 -->
<dependency>
    <groupId>org.codehaus.jettison</groupId>
    <artifactId>jettison</artifactId>
    <version>1.3.7</version>
</dependency>

<!-- spring 核心 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.2.4.RELEASE</version>
</dependency>
<!-- spring web集成 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.2.4.RELEASE</version>
</dependency>
<!-- spring 整合junit -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>4.2.4.RELEASE</version>
</dependency>
<!-- junit 开发包 -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
</dependencies>

<build>
    <pluginManagement>
        <plugins>

            <!-- maven的jdk编译插件 -->
```



```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.2</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <encoding>UTF-8</encoding>
    <showWarnings>true</showWarnings>
  </configuration>
</plugin>
<!-- 运行tomcat7方法: tomcat7:run -->
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <!-- 指定端口 -->
    <port>8080</port>
    <!-- 请求路径 -->
    <path>/</path>
  </configuration>
</plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

## 8.1.3 web.xml



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

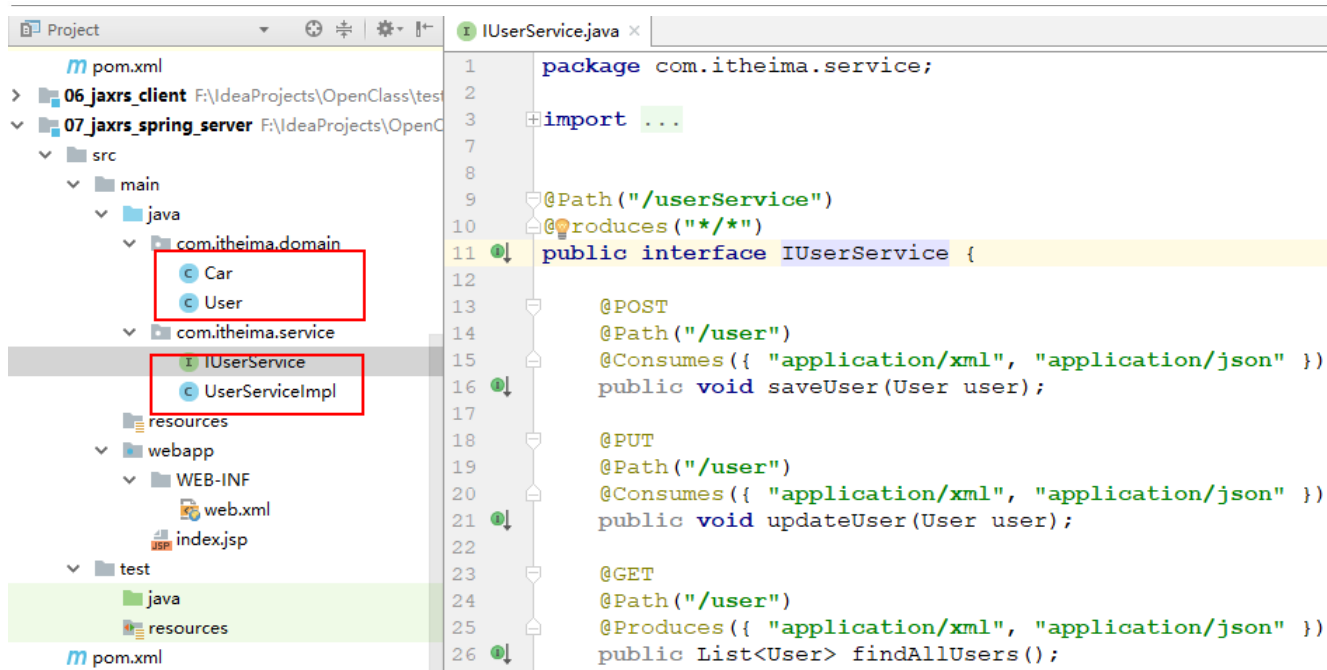
    <servlet>
        <servlet-name>cxfservlet</servlet-name>
        <servlet-
class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxfservlet</servlet-name>
        <url-pattern>/ws/*</url-pattern>
    </servlet-mapping>

    <!-- 1. 配置springioc容器 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>

</web-app>
```

## 8.1.4 服务接口、实现、实体类

与之前案例一样：



## 8.1.5 Spring整合CXF



```
<?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:cxf="http://cxf.apache.org/core"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://cxf.apache.org/core
    http://cxf.apache.org/schemas/core.xsd
    http://cxf.apache.org/jaxws
    http://cxf.apache.org/schemas/jaxws.xsd
    http://cxf.apache.org/jaxrs
    http://cxf.apache.org/schemas/jaxrs.xsd">

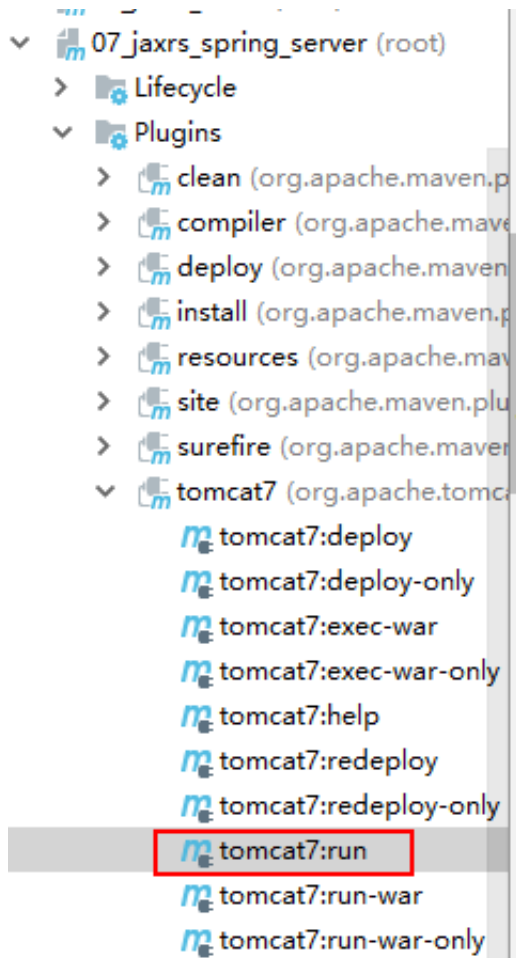
  <!--
    Spring整合ApacheCXF，发布jaxws服务：
    1. 服务地址
    2. 服务bean

    完整服务地址：
    http://localhost:8080/ws/userService
  -->
  <jaxrs:server address="/userService">
    <jaxrs:serviceBeans>
      <bean class="com.itheima.service.UserServiceImpl"></bean>
    </jaxrs:serviceBeans>
    <jaxrs:inInterceptors>
      <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"
/>
    </jaxrs:inInterceptors>
    <jaxrs:outInterceptors>
      <bean
class="org.apache.cxf.interceptor.LoggingOutInterceptor" />
    </jaxrs:outInterceptors>
  </jaxrs:server>

</beans>
```



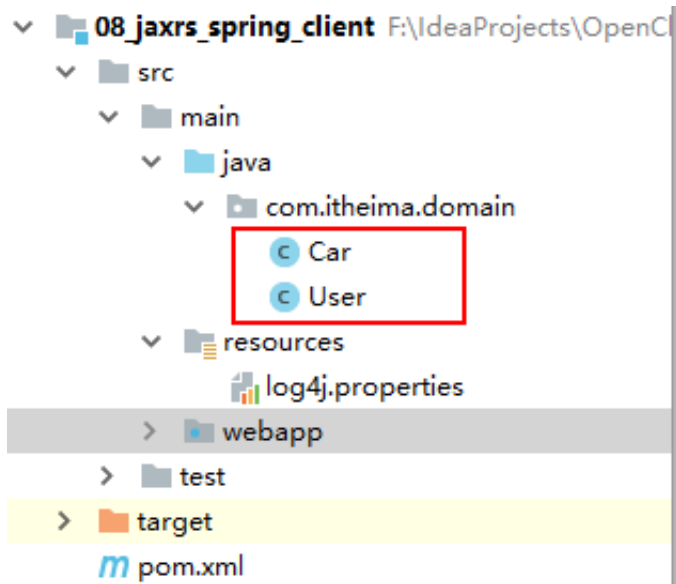
## 8.1.6 发布服务



## 8.2 客户端

### 8.2.1 创建项目

创建服务端



## 8.2.2 添加依赖

上面已经完成

## 8.2.3 实体类

上面已经完成

## 8.2.4 测试

```
package com.itheima;

import com.itheima.domain.User;
import org.apache.cxf.jaxrs.client.WebClient;

public class Client {

    public static void main(String[] args) {
        // 测试1: 保存
        WebClient

        .create("http://localhost:8080/ws/userService/userService/user")
            .post(new User());

        // 测试2: 删除（传入id）
        WebClient

        .create("http://localhost:8080/ws/userService/userService/user/100")
            .delete();
    }
}
```

## 9 小结

所以，以后如果在开发中涉及多个系统之间的交互，webservice是不错的选择。

个人认为使用service最大的好处是客户端与服务端语言的兼容性及交互时候可以传递对象。

今天讲到的内容再次梳理一遍：

### 1) webservice 概念

- 简介
- 术语（开发规范、soap协议、UDDI）

2) 应用场景、优缺点

3) Apache CXF 框架介绍

4) jax-ws 规范下的webservice开发、与spring整合

5) jax-rs 风格下的webservice开发、与spring整合

各位同学，到这里webservice就全部讲解完毕了！

文章所有的代码都是笔者亲测过，分享给大家的！

祝大家学习进步！