

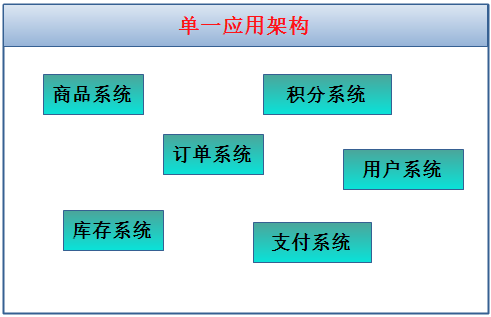
# 第一章、RPC基础知识

## 软件架构

#### 单一应用架构（单机系统）

当网站流量很小时，应用规模小时，只需一个应用，将所有功能都部署在一起，以减少部署服务器数量和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。数据库的处理时间影响应用的性能

ORM：Hibernate Mybatis



这种结构的应用适合小型系统，小型网站，或者企业的内部系统，用户较少，请求量不大，对请求的处理时间没有太高的要求。 将所有功能都部署到一个服务器，简单易用。开发项目的难度低。--低访问量 性能要求不高

缺点：

1、性能扩展比较困难

2、不利于多人同时开发

3、不利于升级维护

4、整个系统的空间占用比较大

\*\*垂直系统：把需求模块进行划分多个系统（独立项目），划分后的系统各自没有联系，相互独立部署（用户系统部署10台服务器，积分系统部署2台）

缺点：

1、性能扩展比较困难

2、不利于多人同时开发

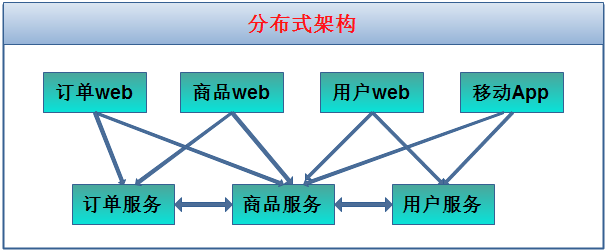
3、不利于升级维护

4、整个系统的空间占用比较大

#### 分布式服务（SOA）架构

当应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的**分布式服务框架(RPC)**是关键。分布式系统将服务作为独立的应用，实现服务共享和重用。

**RPC：**Remote Procedure Call 远程方法调用



## 分布式系统

### 什么是分布式系统

分布式系统是若干独立计算机（服务器）的集合，这些计算机对于用户来说就像单个相关系统，分布式系统（distributed system）是建立在网络之上的服务器端一种结构。

\*\*部署在独立服务器上的各个子系统（项目），相互之间可以调用，形成一个大型分布式系统

\*\*独立部署的服务器没有额外要求，只需要满足子系统需求即可。

分布式系统中的计算机可以使用不同的操作系统，可以运行不同应用程序提供服务，将服务分散部署到多个计算机服务器上。

### RPC ？

RPC 【Remote Procedure Call】是指远程过程调用，是一种进程间通信方式，是一种技术思想，而不是规范。它允许程序调用另一个地址空间（网络的另一台机器上）的过程或函数，而不用开发人员显式编码这个调用的细节。调用本地方法和调用远程方法一样。

\*\*天气信息：各个网站、app可以远程调用气象台服务获取天气信息

RPC的实现方式可以不同。例如java的rmi, hessian, webservice，spring远程调用等。

Rmi：reomote Method invoke Java中RPC底层

RPC概念是在上世纪80年代由 Brue Jay Nelson(布鲁·杰伊·纳尔逊)提出。使用PRC可以将本地的调用扩展到远程调用（分布式系统的其他服务器）。

RPC的特点

1. 简单：使用简单，建立分布式应用更容易。

2. 高效：调用过程看起来十分清晰，效率高。

# 第二章、dubbo框架

## 2.1 dubbo概述

Apache Dubbo (incubating) |ˈdʌbəʊ| 是一款高性能、轻量级的开源Java RPC框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

\*\* Dubbo(读音[ˈdʌbəʊ])是阿里巴巴公司开源的一个高性能优秀的[服务框架](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E6%A1%86%E6%9E%B6)，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 [1]  [Spring](https://baike.baidu.com/item/Spring)框架无缝集成

Dubbo是一个分布式服务框架，致力于提供高性能和透明化的[**RPC**](javascript:if(confirm(%27http://en.wikipedia.org/wiki/Remote_procedure_call%20%20\n\nThis%20file%20was%20not%20retrieved%20by%20Teleport%20Ultra,%20because%20it%20is%20addressed%20on%20a%20domain%20or%20path%20outside%20the%20boundaries%20set%20for%20its%20Starting%20Address.%20%20\n\nDo%20you%20want%20to%20open%20it%20from%20the%20server?%27))window.location=%27http://en.wikipedia.org/wiki/Remote_procedure_call%27)远程服务调用方案、服务治理方案。

**官网：**http://dubbo.apache.org/zh-cn/



**特性:**



面向接口代理：调用接口的方法，在A服务器调用B服务器的方法，由dubbo实现对B的调用，无需关心实现的细节，就像MyBatis访问Dao的接口，可以操作数据库一样。不用关心Dao接口方法的实现。这样开发是方便，舒服的。

## 2.2 基本架构



**服务提供者（Provider）**：暴露服务的服务提供方，服务提供者在启动时，向注册中心注册自己提供的服务。

**服务消费者（Consumer）**: 调用远程服务的服务消费方，服务消费者在启动时，向注册中心订阅自己所需的服务，服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

**注册中心（Registry）**：注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者 –如果信息有变，注册中心提供新的信息给消费者

**监控中心（Monitor）**：服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心 –监控服务提供者、消费者状态，与开发没有直接关系

**调用关系说明:**

* 服务容器spring负责启动，加载，运行服务提供者。
* 服务提供者在启动时，向注册中心注册自己提供的服务。
* 服务消费者在启动时，向注册中心订阅自己所需的服务。
* 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
* 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
* 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

## 2.3 dubbo支持的协议

支持多种协议：dubbo , hessian , rmi , http, webservice , thrift , memcached , redis。

dubbo官方推荐使用dubbo协议。dubbo协议默认端口20880

使用dubbo协议，spring配置文件加入：

<dubbo:protocol name="dubbo" port="20880" />

## 2.4 电商平台需求

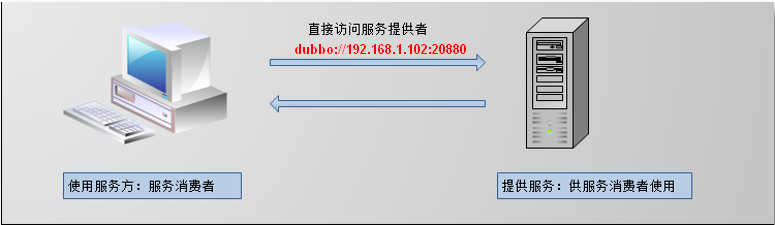
某电商平台系统需求，用户浏览商品；选择商品下订单，订单系统需要获取用户信息中的送货地址；向支付系统请求完成付款。

|  |  |
| --- | --- |
| 服务 | 功能 |
| 网站系统 | 展示商品，修改用户信息 |
| 订单系统 | 生成订单 |
| 用户系统 | 用户信息（地址，收件人，联系方式等） |

## 2.5 直连方式dubbo

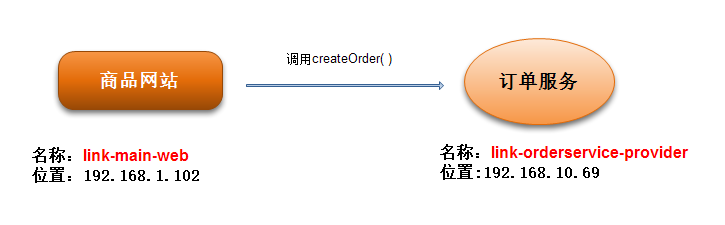
点对点的直连项目:消费者直接访问服务提供者，没有注册中心。消费者必须指定服务提供者的访问地址（url）。

消费者直接通过url地址访问固定的服务提供者。这个url地址是不变的。



### 2.5.1 实现目标

用户访问 ------>【商品网站服务】访问-----> 【订单服务】



### 2.5.2 实现方式

以JavaSE为例，服务提供者，服务消费者都是JavaSE项目

#### 创建服务提供者：订单服务

##### 新建java project

项目名称：**link-orderservice-provider**

**设置version 为 1.0.0**

##### maven pom.xml

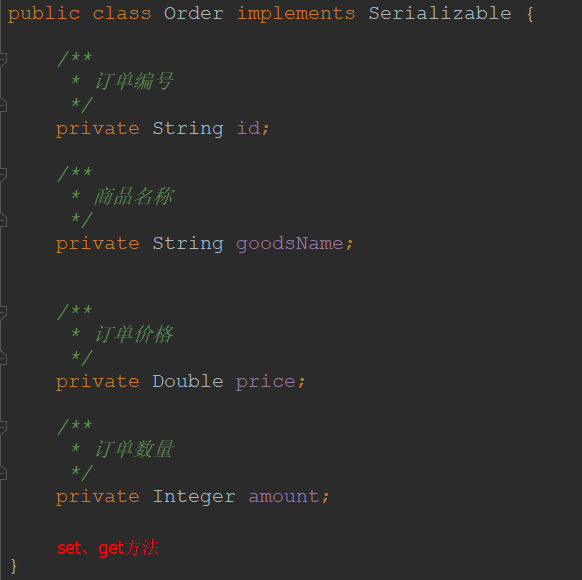
<!--Spring -->  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<!--Dubbo依赖-->  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.6.2</version>  
</dependency>

在<build> 下的 <plugins>标签中加入JDK1.8编译插件

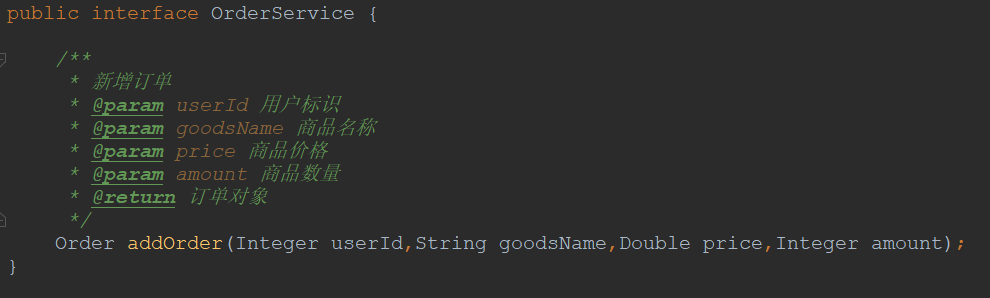
<--JDK1.8编译插件-->

<plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
</plugin>

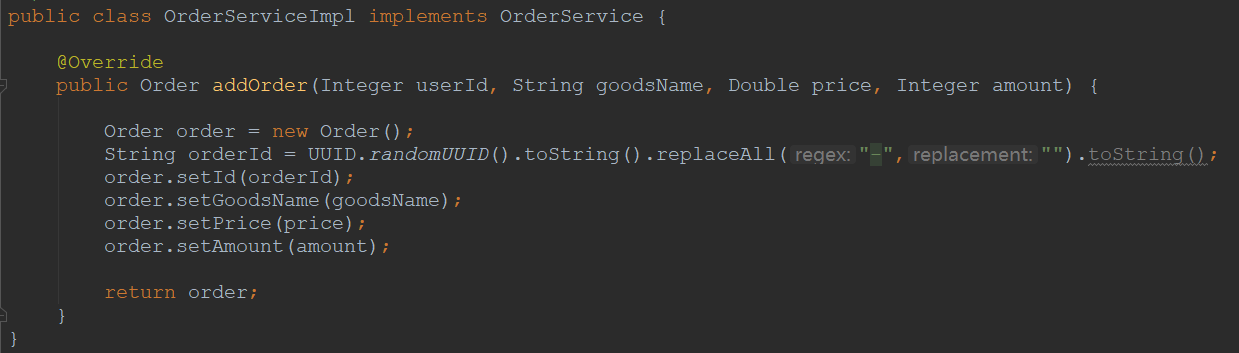
##### 创建订单实体类：Order



##### 新建订单服务接口：OrderService



##### 新建接口的实现类：OrderServiceImpl



##### 创建dubbo配置文件

orderservce-provider.xml



详细配置：

<?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:dubbo="http://dubbo.apache.org/schema/dubbo"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

<!--服务提供者声明名称:必须保证服务名称的唯一性,它的名称是dubbo内部使用的唯一标识-->

<dubbo:application name="001-link-userservice-provider"/>

<!--访问服务协议的名称及端口号,dubbo官方推荐使用的是dubbo协议,端口号默认为20880-->

<!--

name:指定协议的名称

port:指定协议的端口号(默认为20880)

-->

<dubbo:protocol name="dubbo" port="20880"/>

<!--

暴露服务接口->dubbo:service

interface:暴露服务接口的全限定类名

ref:接口引用的实现类在spring容器中的标识

registry:如果不使用注册中心,则值为:N/A

-->

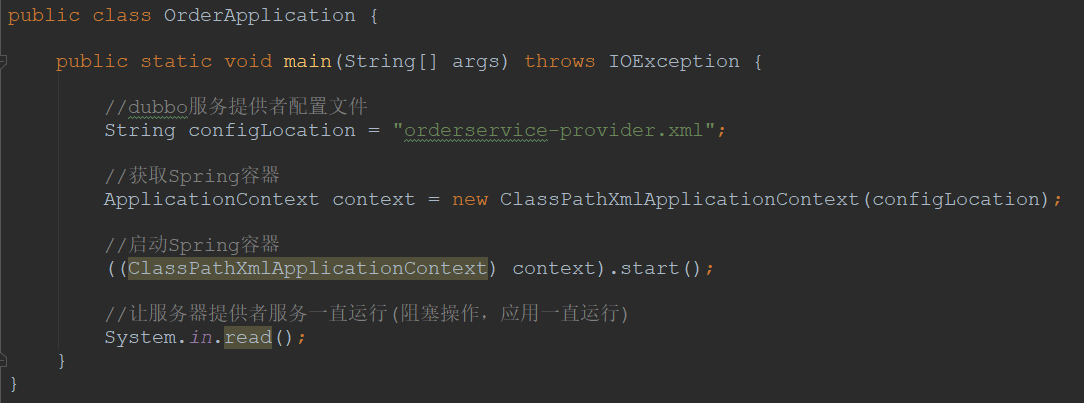
<dubbo:service interface="com.wkcto.dubbo.service.UserService" ref="a" registry="N/A"/>

<!--将接口的实现类加载到spring容器中-->

<bean id="a" class="com.wkcto.dubbo.service.impl.UserServiceImpl"/>

</beans>

##### 测试配置文件



##### 安装本地jar 到maven仓库

服务接口中的方法要给消费者使用，消费者项目需要知道接口名称和接口中的方法名称、参数等。这些信息服务提供者才知道。需要把接口的class文件打包为jar .

服务接口项目的类文件打包为jar， 安装到maven仓库，仓库中的提供者jar可以被消费者使用。

使用IDEA的maven窗口执行 install

#### 创建服务消费者：商品网站

##### 新建java project

项目名称：**link-main-web**

##### maven pom.xml

<!--  
 依赖link-orderservice-provider：获取暴露的服务接口  
-->  
<dependency>  
 <groupId>com.abc</groupId>  
 <artifactId>001-link-orderservice-provider</artifactId>  
 <version>1.0.0</version>  
</dependency>

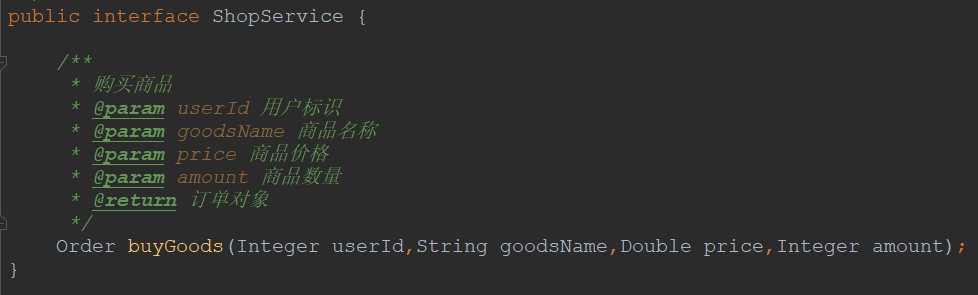
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.6.2</version>  
</dependency>

在<build> 下的 <plugins>标签中加入JDK1.8编译插件

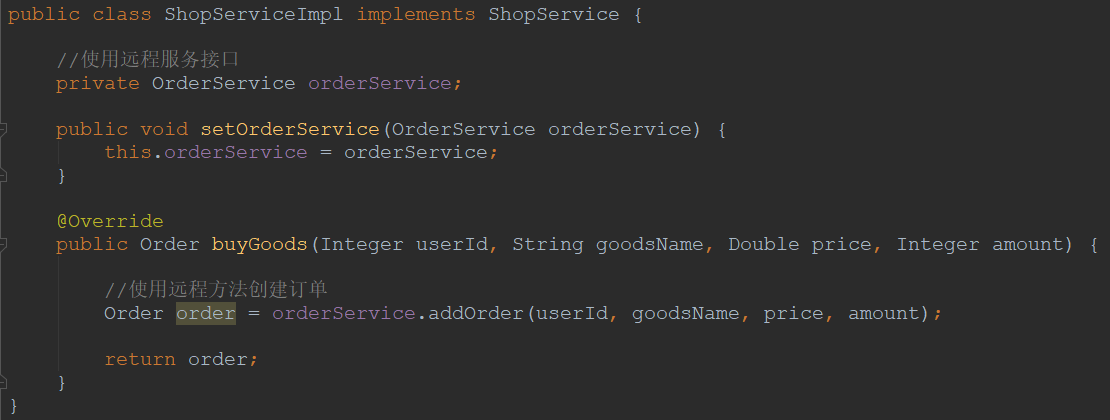
<--JDK1.8编译插件-->

<plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
</plugin>

##### 创建购买商品接口



##### 创建购买接口的实现类



##### 创建dubbo配置文件

shop-consume.xml



##### 执行消费者



## 2.6 dubbo服务化最佳实践

### 2.6.1 分包

建议将服务接口、服务模型、服务异常等均放在公共包中。

\*\*\*服务提供者、消费者都需要使用的类和接口，放入公共包中，实现都能访问

### 2.6.2 粒度

服务接口尽可能大粒度，每个服务方法应代表一个功能，而不是某功能的一个步骤，

服务接口建议以业务场景为单位划分，并对相近业务做抽象，防止接口数量爆炸。

不建议使用过于抽象的通用接口，如：Map query(Map)，这样的接口没有明确语义，会给后期维护带来不便。

### 2.6.3 版本

每个接口都应定义版本号，区分同一接口的不同实现，如： <dubbo:service interface="com.xxx.XxxService" version="1.0" />。

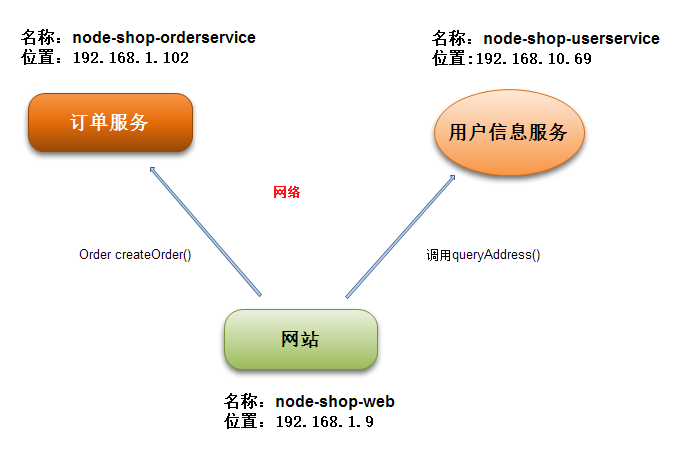
## 2.7 改造dubbo项目

抽象分散在多个项目中的公共接口，实体类，异常，工具类到一个项目中，在其他项目如服务提供者，消费者共用公共的资源。

### 2.7.1 实现目标

用户访问电商网站浏览商品—选择商品购买

用户访问电商网站—查看用户信息（收件人地址）



项目是web应用，需要加入spring web开发jar：



maven依赖

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.16.RELEASE</version>

</dependency>

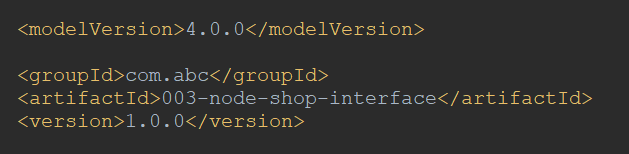
### 2.7.2 创建公共资源项目

服务提供者，消费者，网站等多个服务中共用，重复使用的类单独定义在一个项目。

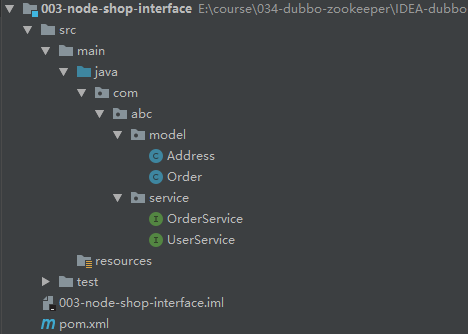
##### A、 创建公共的maven java project

项目名称：node-shop-interface

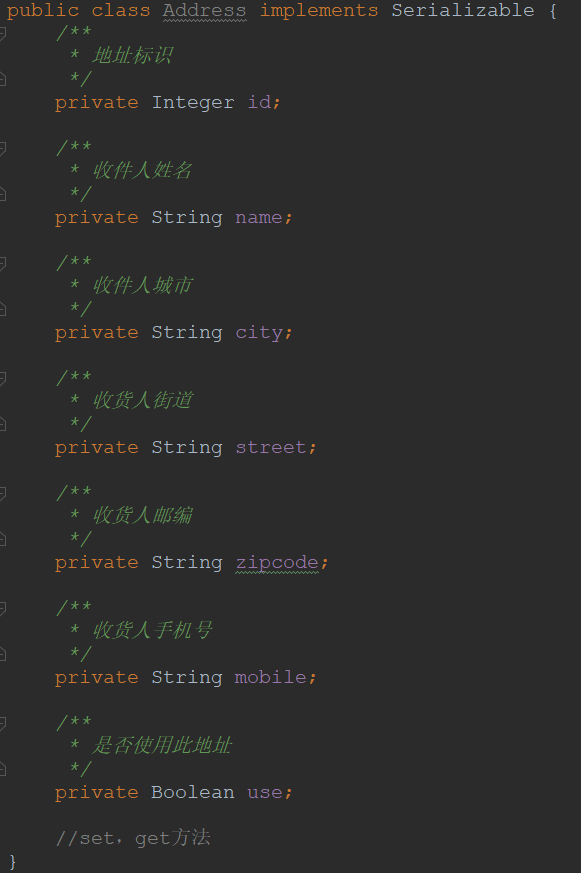
接口工程的Maven坐标



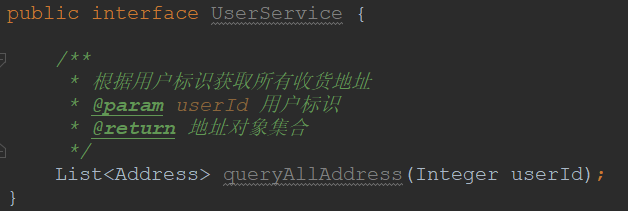
复制之前的Order实体类、OrderService业务接口到node-shop-interface项目，



##### 新建Address实体类



##### 新建UserInfoService接口



##### 安装jar到maven仓库

使用IDEA的maven窗口执行 install

### 2.7.3 创建用户信息服务

##### A、 新建web project

项目名称：node-shop-userservice

##### maven pom.xml

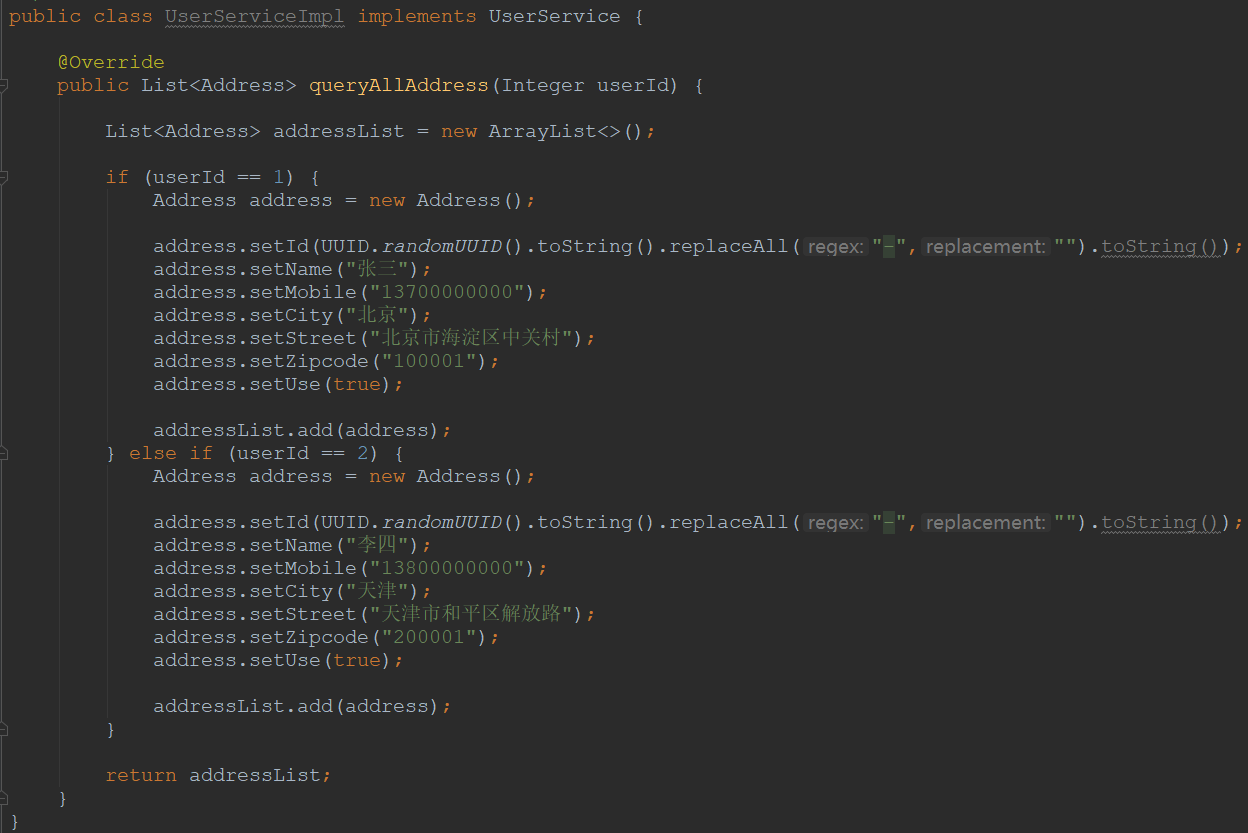
<!--接口工程-->  
<dependency>  
 <groupId>com.abc</groupId>  
 <artifactId>003-node-shop-interface</artifactId>  
 <version>1.0.0</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.6.2</version>  
</dependency>

在<build> 下的 <plugins>标签中加入JDK1.8编译插件

<--JDK1.8编译插件-->

<plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
</plugin>

##### 创建UserInfoServiceImpl实现类

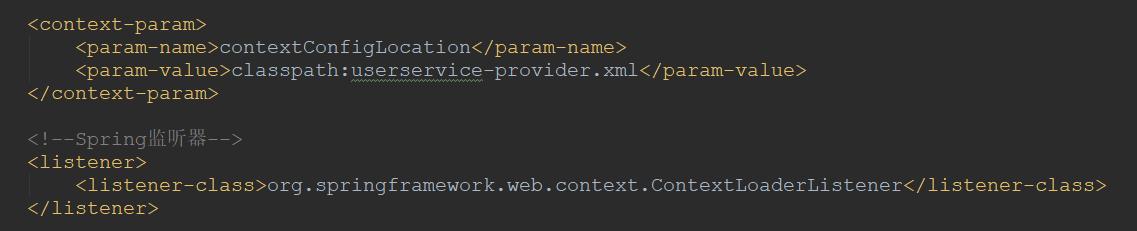


##### dubbo配置文件

文件名称：userservice-provider.xml



##### web.xml注册spring监听器



### 2.7.4 创建订单服务

##### 新建web project

项目名称：node-shop-orderservice

##### maven pom.xml

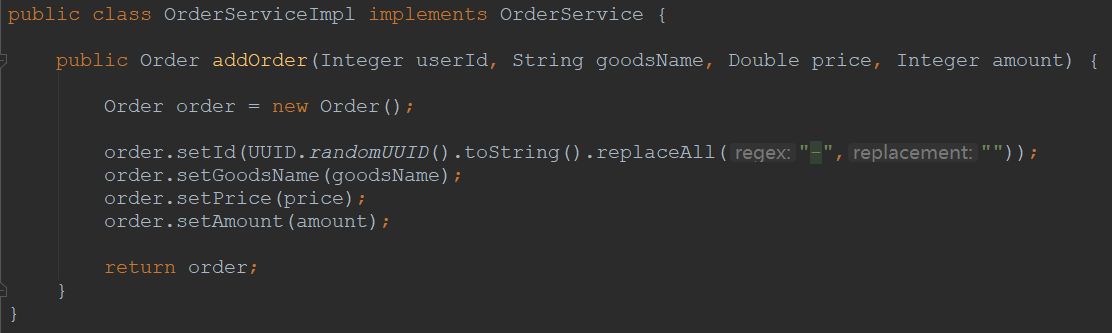
<!--接口工程-->  
<dependency>  
 <groupId>com.abc</groupId>  
 <artifactId>003-node-shop-interface</artifactId>  
 <version>1.0.0</version>  
</dependency>  
  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.6.2</version>  
</dependency>

在<build> 下的 <plugins>标签中加入JDK1.8编译插件

<--JDK1.8编译插件-->

<plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
</plugin>

##### 创建OrderService接口实现类



##### dubbo配置文件

文件名称：dubbo-orderservice-provider.xml



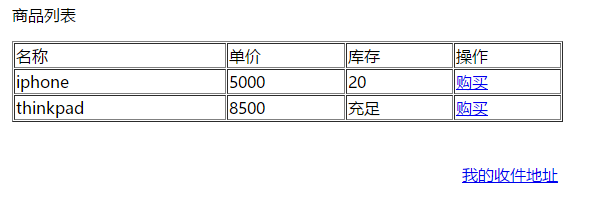
##### web.xml注册spring监听器

### 2.7.5 创建商品网站

##### 新建web project

项目名称：node-shop-web

##### 创建首页index.jsp



页面代码：

<div style="margin-left:400px">  
 <table border="1" cellpadding="1" cellspacing="1" width="60%">  
 <p>商品列表</p>  
 <tr>  
 <td>iphone11</td>  
 <td>5000</td>  
 <td>8</td>  
 <td><a href="javascript:void(0)" onClick="buyGoods(1,'iphone11','5000','8')">购买</td>  
 </tr>  
  
 <tr>  
 <td>电视</td>  
 <td>2000</td>  
 <td>2</td>  
 <td><a href="javascript:void(0)" onClick="buyGoods(2,'电视','2000','2')">购买</td>  
 </tr>  
  
 </table>  
</div>  
  
<div style="margin-left:450px">  
 <a href="javascript:void(0)" onClick="showAddress(2)" >我的收件地址</a>  
</div>

js代码

<script type="text/javascript">  
 **var** URL\_PREFIX="**${**pageContext.request.contextPath**}**";  
 **function** buyGoods(userId,name,price,amount){  
 window.location.href= URL\_PREFIX +"/shop/buy?userId="+userId  
 +"&name="+name  
 +"&price="+price  
 +"&amount="+amount;  
 }  
  
 **function** showAddress(userId){  
 window.location.href= URL\_PREFIX +"/shop/addresses?userId="+userId;  
 }  
</script>

##### maven pom.xml

<!--接口工程-->  
<dependency>  
 <groupId>com.abc</groupId>  
 <artifactId>003-node-shop-interface</artifactId>  
 <version>1.0.0</version>  
</dependency>  
  
<!--Servlet、JSTL依赖-->  
<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>3.1.0</version>  
</dependency>  
  
<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
</dependency>  
  
<dependency>  
 <groupId>taglibs</groupId>  
 <artifactId>standard</artifactId>  
 <version>1.1.2</version>  
</dependency>  
  
<!--Spring依赖-->  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>4.3.16.RELEASE</version>  
</dependency>  
  
<!--Dubbo依赖-->  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.6.2</version>  
</dependency>

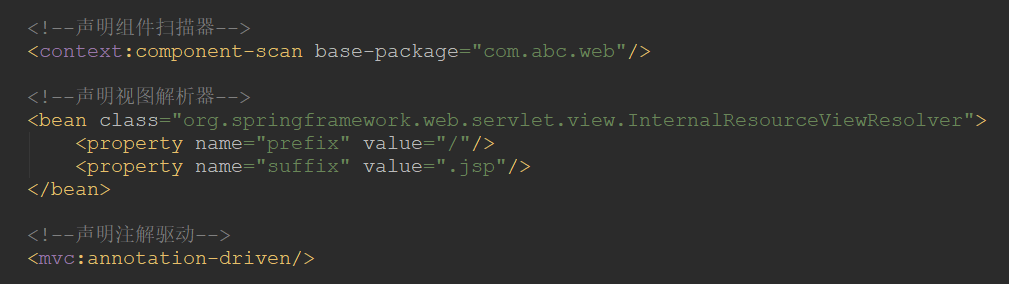
在<build> 下的 <plugins>标签中加入JDK1.8编译插件

<--JDK1.8编译插件-->

<plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
</plugin>

##### 创建Spring配置文件

文件名称：dispatcherServlet.xml



详细配置：

<?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:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!--扫描组件-->

<context:component-scan base-package="com.wkcto.dubbo.web"/>

<!--配置注解驱动-->

<mvc:annotation-driven/>

<!--视图解析器-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/"/>

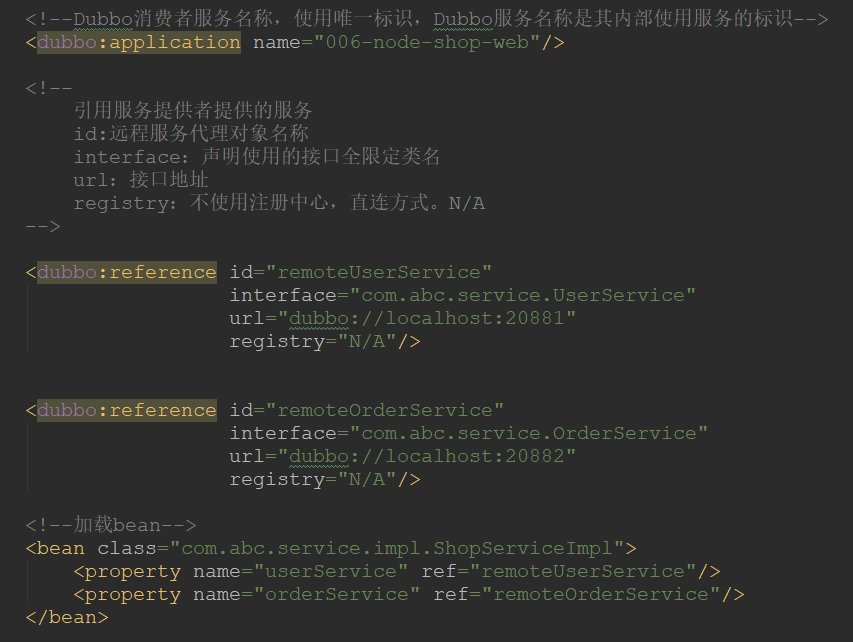
<property name="suffix" value=".jsp"/>

</bean>

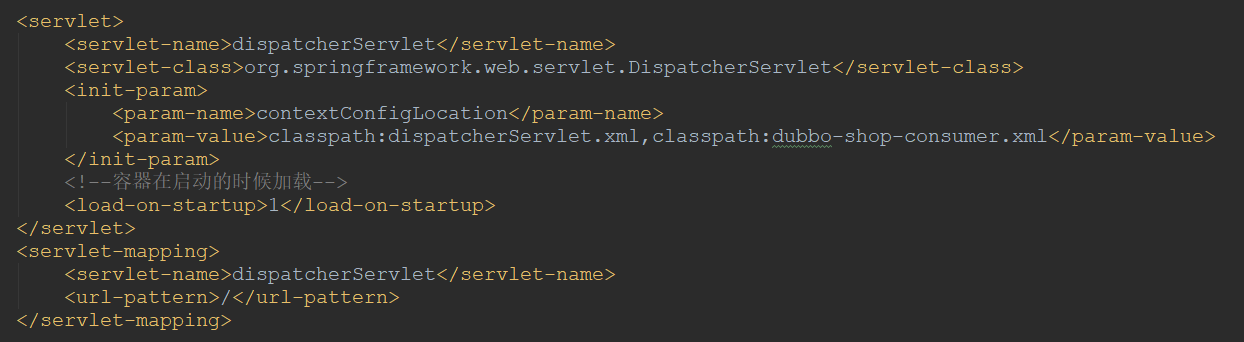
</beans>

##### 创建dubbo配置文件

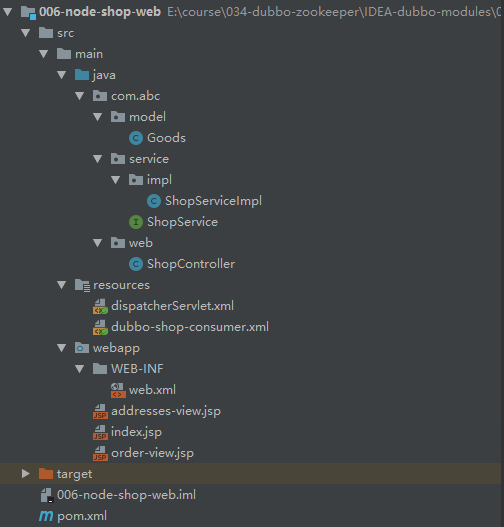
文件名称：dubbo-shop-consumer.xml



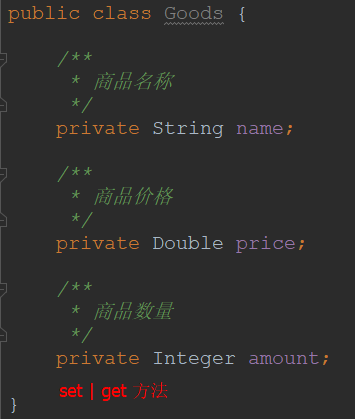
##### web.xml注册中央调度器DispatcherServlet



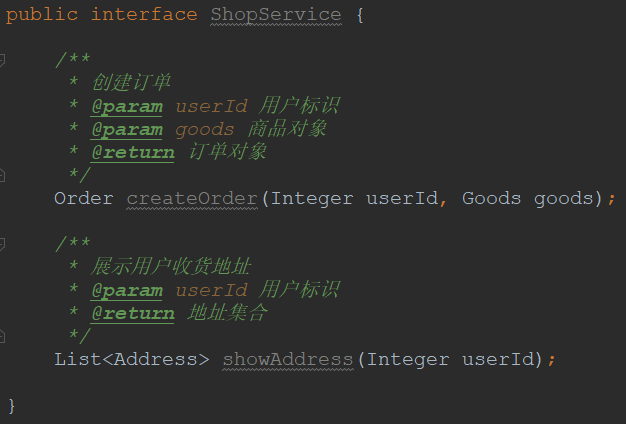
##### 项目结构



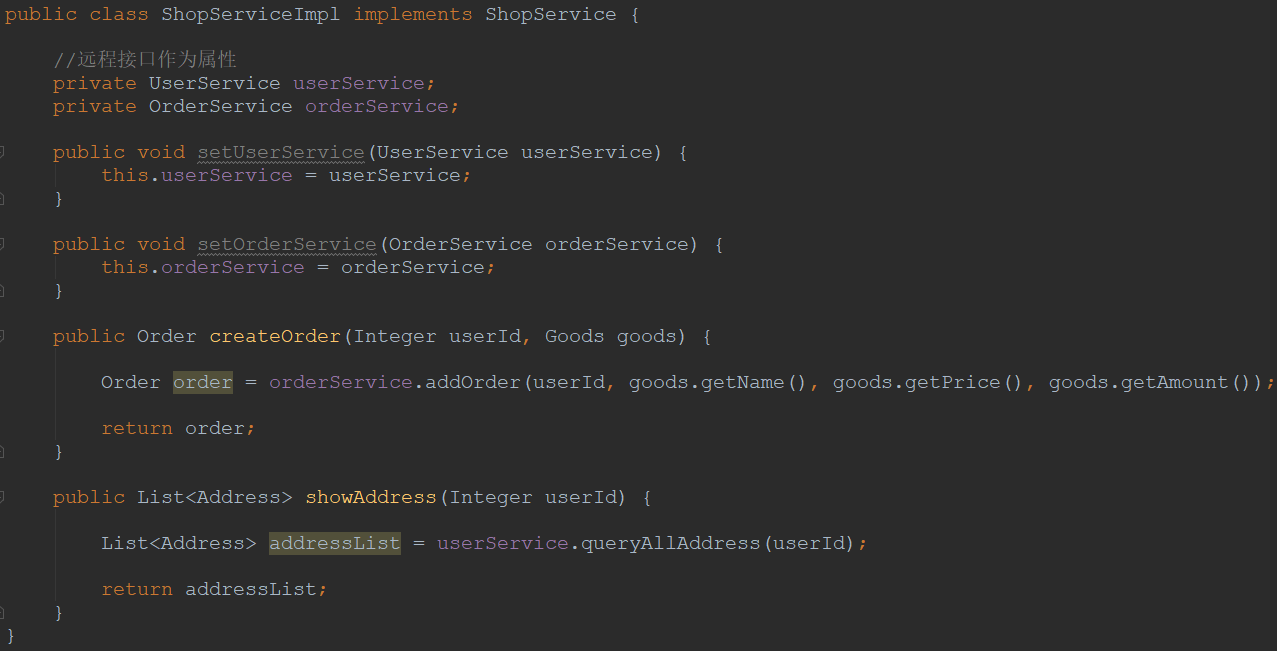
##### 创建商品信息实体类Goods



##### 创建接口ShopService

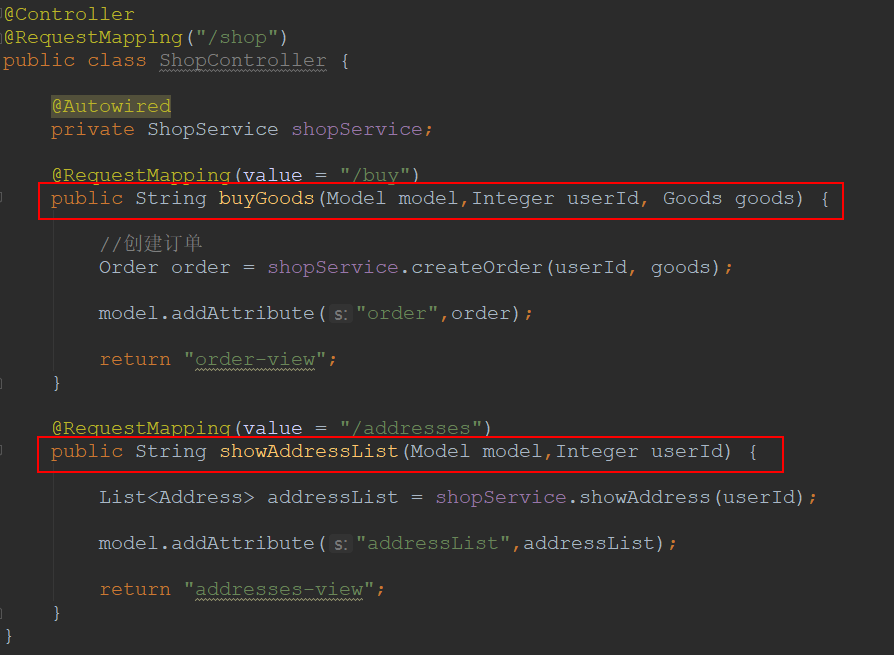


##### 创建接口实现类ShopServiceImpl



##### 创建类ShopController

接收来自页面的请求，处理订单，查询地址信息。



##### view-order.jsp

显示订单信息页面：

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>订单展示</title>  
</head>  
<body>  
<h1>您的订单已经生成：</h1>  
<h3>订单编号：**${**order.id**}**</h3>  
<h3>订单金额：**${**order.price**}**</h3>  
<h3>商品名称：**${**order.goodsName**}**</h3>  
<h3>商品数量：**${**order.amount**}**</h3>  
  
</body>  
</html>

##### view-address.jsp

显示收件人地址信息

页面加入jstl：<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

使用jstl需要加入jar：



<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<html>  
<head>  
 <title>展示地址</title>  
</head>  
<body>  
<h1>收货人地址如下：</h1>  
<**c:forEach** items="**${**addressList**}**" var="address">  
 <h3>收货人姓名：**${**address.name**}**</h3>  
 <h3>收货人手机：**${**address.mobile**}**</h3>  
 <h3>收货人城市：**${**address.city**}**</h3>  
 <h3>收货人街道：**${**address.street**}**</h3>  
 <h3>邮编：**${**address.zipcode**}**</h3>  
 <h3>默认地址：**${**address.use**}**</h3>  
 <div>---------------------------------</div>  
</**c:forEach**>  
</body>  
</html>

## 2.8 dubbo常用标签

Dubbo中常用标签。分为三个类别：公用标签，服务提供者标签，服务消费者标签

### 2.8.1 公用标签

<dubbo:application/> 和 <dubbo:registry/>

##### A、配置应用信息

<dubbo:application name=”服务的名称”/>

##### B、配置注册中心

<dubbo:registry address=”ip:port” protocol=”协议”/>

### 2.8.2服务提供者标签

##### 配置暴露的服务

<dubbo:service interface=”服务接口名” ref=”服务实现对象bean”>

### 2.8.3 服务消费者

##### 配置服务消费者引用远程服务

<dubbo:reference id=”服务引用bean的id” interface=”服务接口名”/>

# 第三章、注册中心-Zookeeper

## 3.1 注册中心概述

对于服务提供方，它需要发布服务，而且由于应用系统的复杂性，服务的数量、类型也不断膨胀；对于服务消费方，它最关心如何获取到它所需要的服务，而面对复杂的应用系统，需要管理大量的服务调用。

而且，对于服务提供方和服务消费方来说，他们还有可能兼具这两种角色，即需要提供服务，有需要消费服务。 通过将服务统一管理起来，可以有效地优化内部应用对服务发布/使用的流程和管理。服务注册中心可以通过特定协议来完成服务对外的统一。Dubbo 提供的注册中心有如下几种类型可供选：

Multicast注册中心：组播方式

Redis注册中心：使用Redis作为注册中心

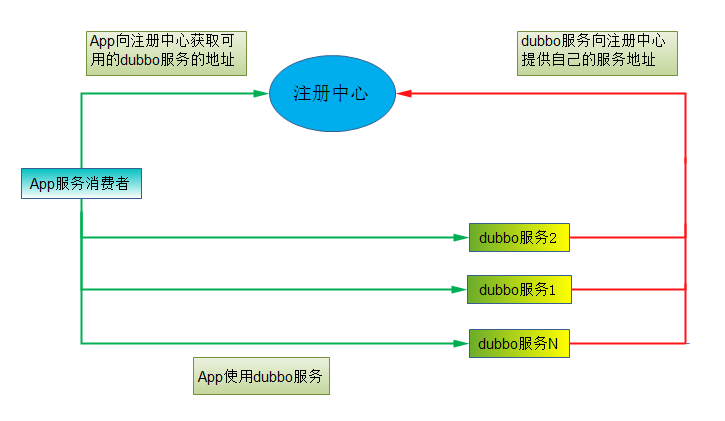
Simple注册中心：就是一个dubbo服务。作为注册中心。提供查找服务的功能。

Zookeeper注册中心：使用Zookeeper作为注册中心

推荐使用Zookeeper注册中心。

\*\*\*注册中心就是统一管理这些服务，提供服务提供者、消费者能够访问！

## 3.2 注册中心工作方式



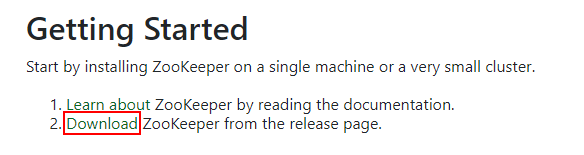
## 3.3 Zookeeper注册中心

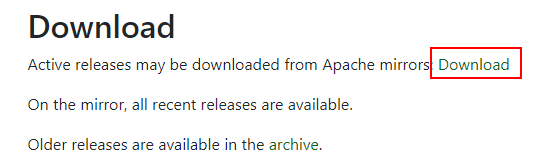
Zookeeper是一个高性能的，[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F)的，开放源码的[分布式应用程序](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)协调服务。简称zk。Zookeeper是翻译过来是动物管理员。可以理解为windows中的资源管理器或者注册表。他是一个树形结构。这种树形结构和标准文件系统相似。ZooKeeper树中的每个节点被称为Znode。和文件系统的目录树一样，ZooKeeper树中的每个节点可以拥有子节点。每个节点表示一个唯一服务资源。Zookeeper运行需要java环境。

### 3.3.1 下载安装文件

官网下载地址: <http://zookeeper.apache.org/>

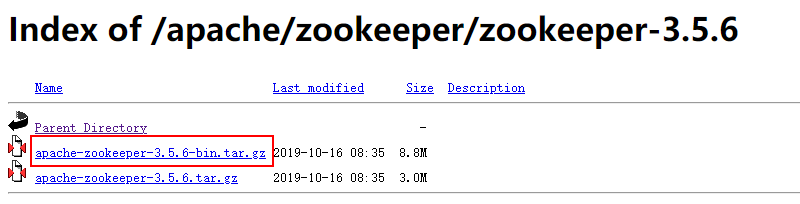
进入官网地址，首页找到下载地址









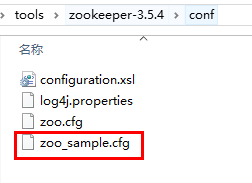


### 3.3.2 安装配置Zookeeper

##### Windows平台Zookeeper安装，配置

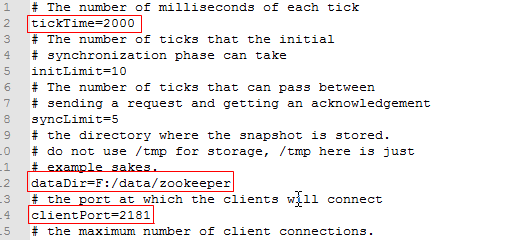
下载的文件zookeeper-3.5.4-beta.tar.gz. 解压后到目录就可以了，例如d:/servers/ zookeeper-3.5.4

修改zookeeper-3.5.4/conf/ 目录下配置文件



复制zoo-sample.cfg改名为zoo.cfg

文件内容：



tickTime: 心跳的时间，单位毫秒. Zookeeper服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime时间就会发送一个心跳。表明存活状态。

dataDir: 数据目录，可以是任意目录。存储zookeeper的快照文件、pid文件，默认为/tmp/zookeeper，建议在zookeeper安装目录下创建data目录，将dataDir配置改为/usr/local/zookeeper-3.4.10/data

clientPort: 客户端连接zookeeper的端口，即zookeeper对外的服务端口，默认为2181

配置内容：

1.dataDir : zookeeper数据的存放目录

2. admin.serverPort=8888

原因：zookeeper 3.5.x 内部默认会启动一个应用服务器，默认占用8080端口

##### Linux平台 Zookeeper安装、配置

Zookeeper的运行需要jdk。使用前Linux系统要安装好jdk.

①：上传zookeeper-3.5.4-beta.tar.gz.并解压

解压文件zookeeper-3.5.4-beta.tar.gz.

执行命令：tar -zxvf zookeeper-3.5.4-beta.tar.gz. -C /usr/local/

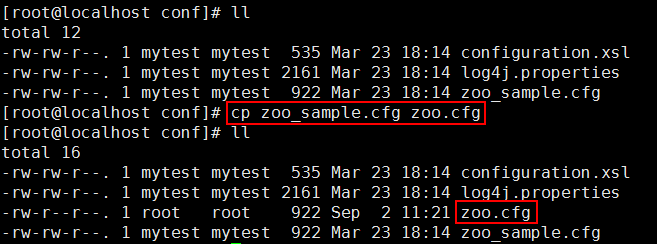
②：配置文件

在zookeeper的conf目录下，将zoo\_sample.cfg改名为zoo.cfg，cp zoo\_sample.cfg zoo.cfg

zookeeper启动时会读取该文件作为默认配置文件。

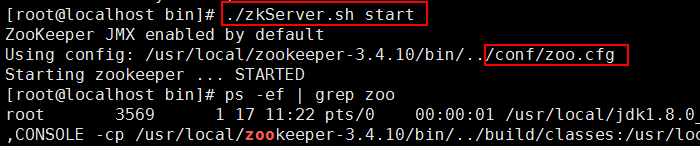
进入zookeeper目录下的conf

拷贝样例文件 zoo-sample.cfg 为 zoo.cfg



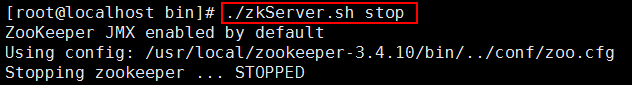
③：启动Zookeeper

启动（切换到安装目录的bin目录下）：./zkServer.sh start



④：关闭Zookeeper

关闭（切换到安装目录的bin目录下）：./zkServer.sh stop



## 3.4 改造dubbo—使用Zookeeper

拷贝项目

|  |  |
| --- | --- |
| 原项目 | 新项目 |
| node-shop-userservice | zk-node-shop-userservice |
| node-shop-orderservice | zk-node-shop-orderservice |
| node-shop-web | zk-node-shop-web |

新项目加入zookeepeer客户端依赖：

<!-- zookeeper客户端依赖 -->

<dependency>  
 <groupId>org.apache.curator</groupId>  
 <artifactId>curator-framework</artifactId>  
 <version>4.1.0</version>  
</dependency>

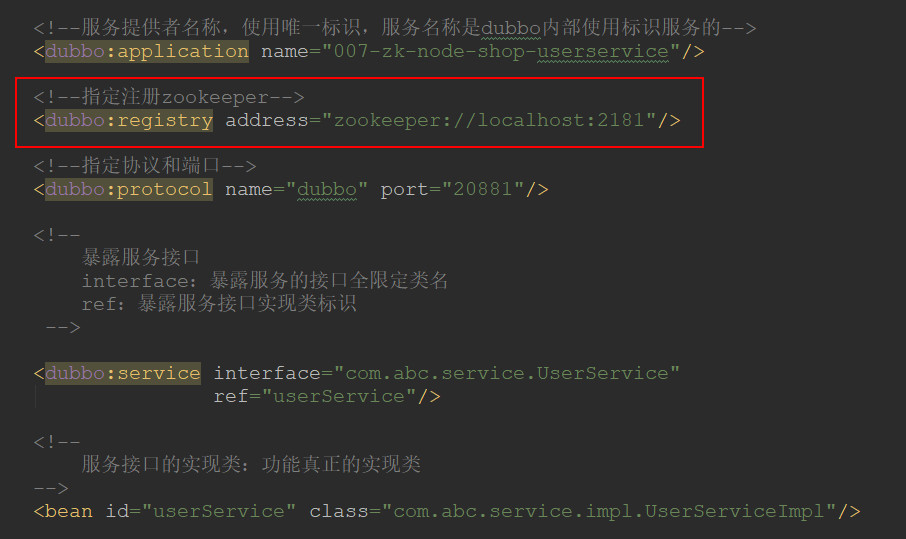
### 3.4.1 zk-node-shop-orderservice服务dubbo配置文件

加入红色方框中的内容，使用zookeeper注册中心



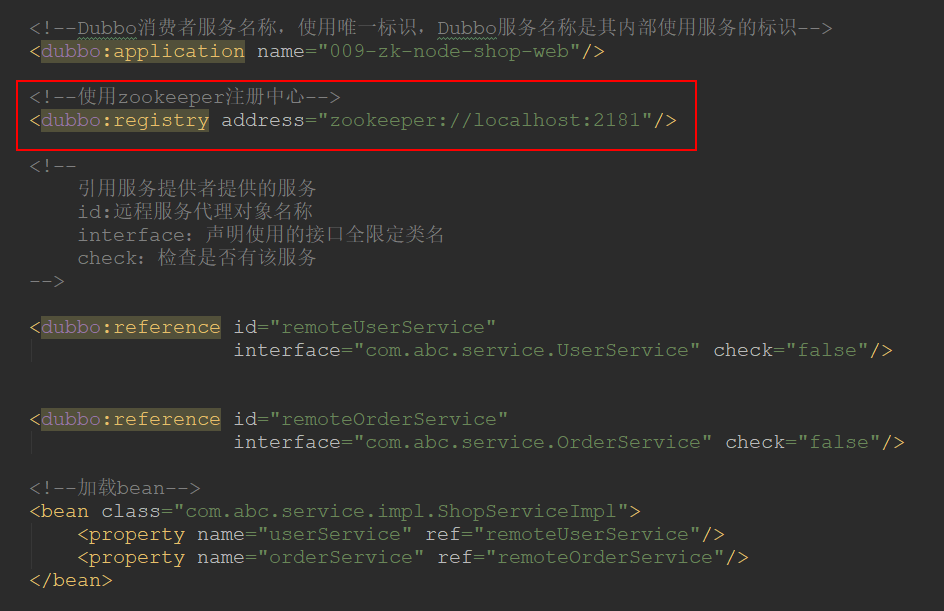
### 3.4.2 zk-node-shop-userservice服务dubbo配置文件

加入红色方框中的内容，使用zookeeper注册中心



### 3.4.3 zk-node-shop-web网站dubbo配置文件

加入红色方框中的内容，使用zookeeper注册中心



### 3.4.4 运行应用

1.先启动注册中心

2.再启动tomcat服务器

3.访问zk-node-shop-web的index.jsp

## 注册中心的高可用

概念：

高可用性（High Availability）：通常来描述一个系统经过专门的设计，从而减少不能提供服务的时间，而保持其服务的高度可用性。

Zookeeper是高可用的，健壮的。Zookeeper宕机，正在运行中的dubbo服务仍然可以正常访问。

健壮性

* 监控中心宕掉不影响使用，只是丢失部分采样数据
* 注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
* 服务提供者无状态，任意一台宕掉后，不影响使用
* 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

演示操作：

1. 先启动zookeeper, dubbo服务提供者，dubbo服务消费者。

2. 测试正常访问

3. 停止zookeeper

4. 测试消费者仍然可以访问提供者

# 第四章、dubbo的配置

## 4.1 配置原则

在服务提供者配置访问参数。因为服务提供者更了解服务的各种参数。

## 4.2 关闭检查

dubbo缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止Spring初始化完成，以便上线时，能及早发现问题，默认check=true。通过check="false"关闭检查，比如，测试时，有些服务不关心，或者出现了循环依赖，必须有一方先启动。

例1：关闭某个服务的启动时检查

<dubbo:reference interface="com.foo.BarService" check="false" />

例2：关闭注册中心启动时检查

<dubbo:registry check="false" />

默认启动服务时检查注册中心存在并已运行。注册中心不启动会报错。

## 4.2 重试次数

消费者访问提供者，如果访问失败，则切换重试访问其它服务器，但重试会带来更长延迟。访问时间变长，用户的体验较差。多次重新访问服务器有可能访问成功。可通过 retries="2" 来设置重试次数(不含第一次)。

|  |
| --- |
| 重试次数配置如下：  <dubbo:service retries="2" />  或  <dubbo:reference retries="2" /> |

## 4.3 超时时间

由于网络或服务端不可靠，会导致调用出现一种不确定的中间状态（超时）。为了避免超时导致客户端资源（线程）挂起耗尽，必须设置超时时间。

timeout：调用远程服务超时时间(毫秒)

### 4.3.1 dubbo消费端

|  |
| --- |
| 指定接口超时配置  <dubbo:reference interface="com.foo.BarService" timeout="2000" /> |

### 4.3.2 dubbo服务端

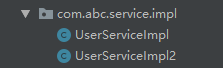
|  |
| --- |
| 指定接口超时配置  <dubbo:server interface="com.foo.BarService" timeout="2000" /> |

## 4.4 版本号

每个接口都应定义版本号，为后续不兼容升级提供可能。当一个接口有不同的实现，项目早期使用的一个实现类， 之后创建接口的新的实现类。区分不同的接口实现使用version。

复制zk-node-shop-userservice为zk-node-shop-multi-userservice

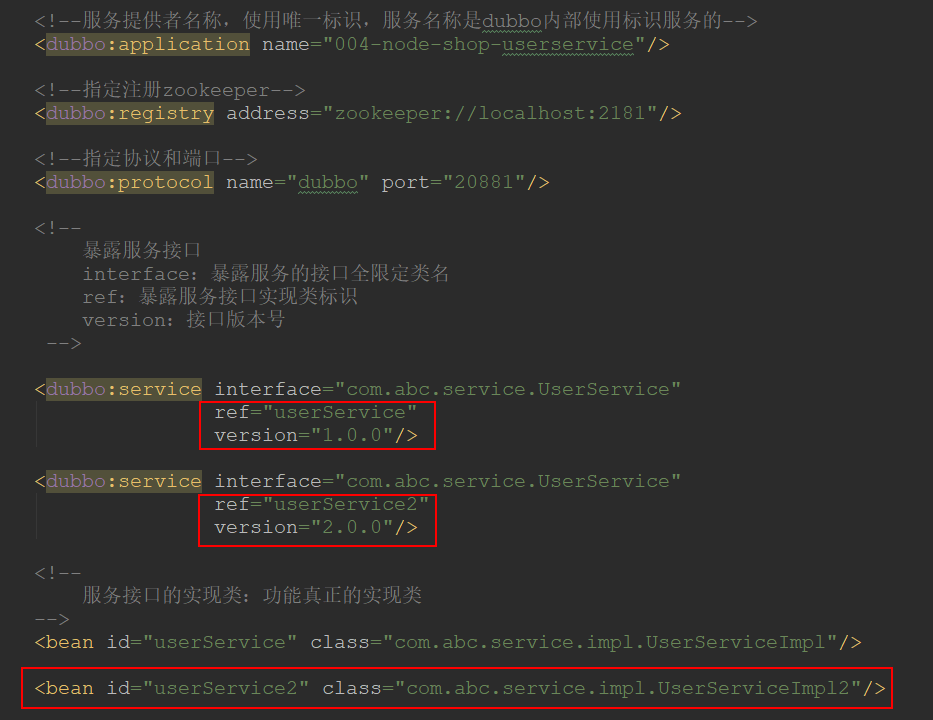
### 4.4.1 复制UserServiceImpl.java



UserServiceImpl2中的地址信息都加入2的内容，用来区别原始的数据。

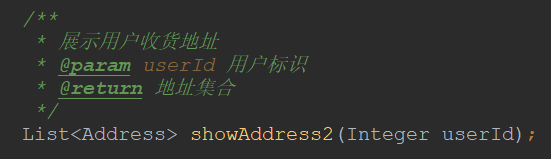
### 4.4.2 dubbo配置文件userservice-provider.xml

增加版本version标志

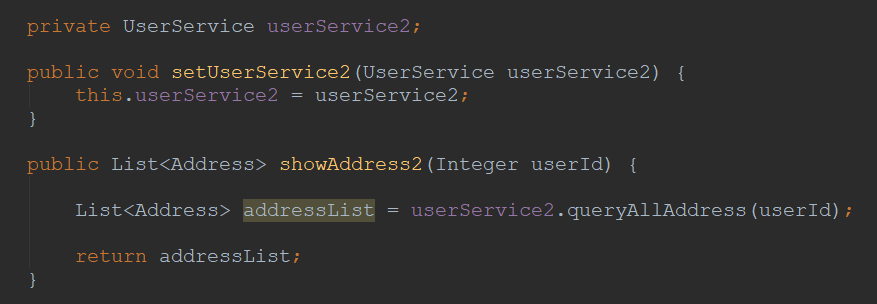


### 4.4.4 复制zk-node-shop-web项目为zk-node-shop-multi-web

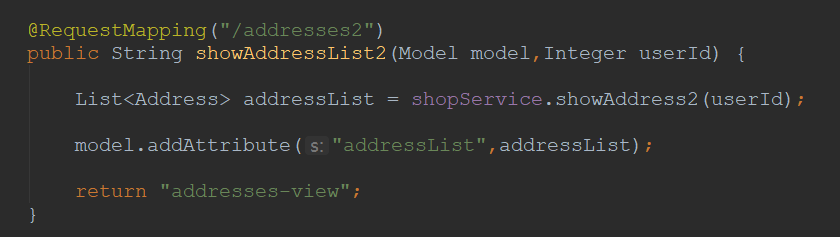
#### ShopService接口



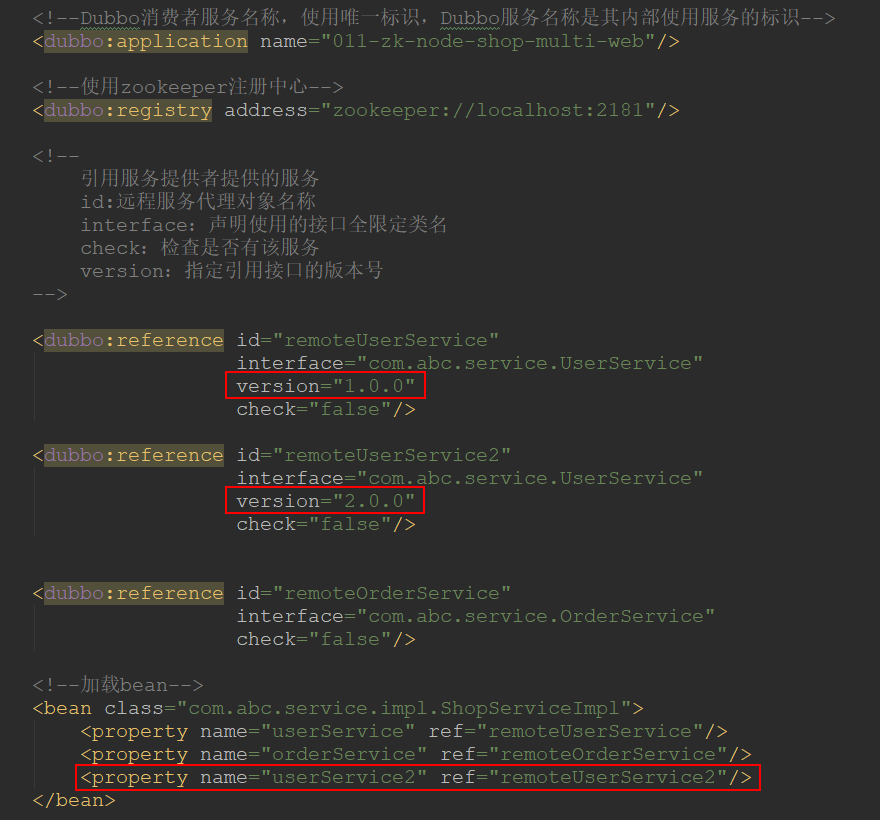
#### ShopServiceImpl接口实现类



#### ShopController类中添加方法



#### 修改消费者配置文件



### 4.4.5 测试应用

1.先启动zookeeper

2.启动tomcat

3.访问服务

比较订单中的地址 ，查看用户信息的地址是不同的内容

# 第五章、监控中心

## 5.1 什么是监控中心

Dubbo 的使用，其实只需要有注册中心，消费者，提供者这三个就可以使用了，但是并不能看到有哪些消费者和提供者，为了更好的调试，发现问题，解决问题，因此引入 dubbo-admin。通过 dubbo-admin 可以对消费者和提供者进行管理。可以在dubbo应用部署做动态的调整，服务的管理。

dubbo-admin

图形化的服务管理页面；安装时需要指定注册中心地址，即可从注册中心中获取到所有的提供者/消费者进行配置管理

dubbo-monitor-simple

简单的监控中心；

## 5.2 发布配置中心

##### 下载监控中心，<https://github.com/apache/incubator-dubbo-ops>

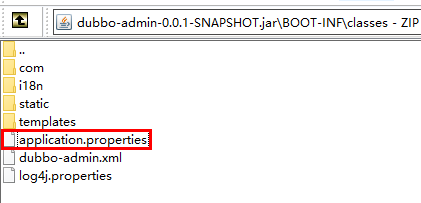
这里下载的是源代码，需要手工编译才能使用。

##### 运行管理后台dubbo-admin

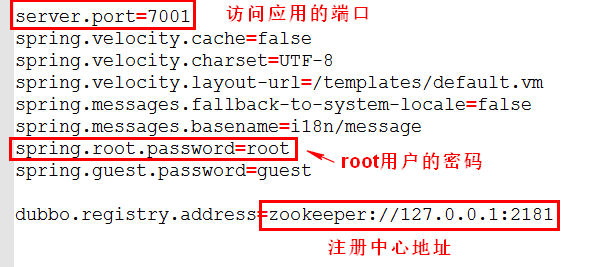
到dubbo-admin-0.0.1-SNAPSHOT.jar 所在的目录。执行下面命令

java -jar dubbo-admin-0.0.1-SNAPSHOT.jar

##### 修改配置dubbo-properties文件



application.properties文件，内容如下：



##### 运行dubbo-admin应用

1. 先启动注册中心
2. 执行提供者项目
3. java -jar dubbo-admin-0.0.1-SNAPSHOT.jar 启动dubbo管理后台
4. 在浏览器地址栏输入 <http://localhost:7001> 访问监控中心-控制台。



## 5.3 监控中心的数据来源

dubbo.registry.address=zookeeper://127.0.0.1:2181

监控中心的数据来自注册中心（Zookeeper）

## 5.4 应用监控中心

通过浏览器，访问监控中心主页。点击菜单访问功能选项。