



北京动力节点教育科技有限公司

动力节点课程讲义

DONGLIJIEDIANKECHENGJIANGYI

www.bjpowernode.com

第一章、RPC 基础知识

1.1 软件架构

(1) 单一应用架构

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



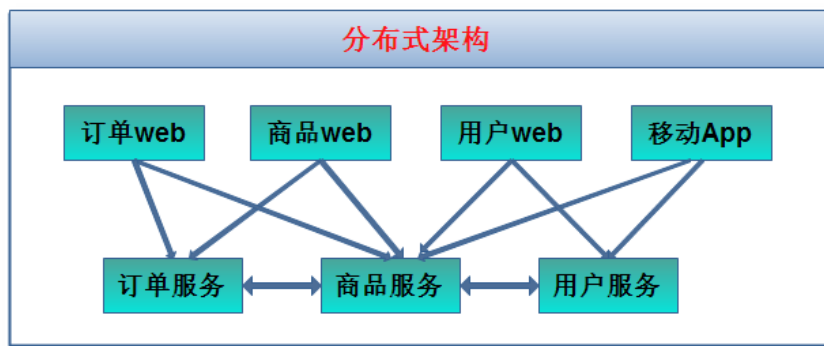
这种结构的应用适合小型系统，小型网站，或者企业的内部系统，用户较少，请求量不大，对请求的处理时间没有太高的要求。 将所有功能都部署到一个服务器，简单易用。开发项目的难度低。

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

(2) 分布式服务架构

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。



1.2 分布式系统

1.2.1 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。

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。

1.2.2 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。

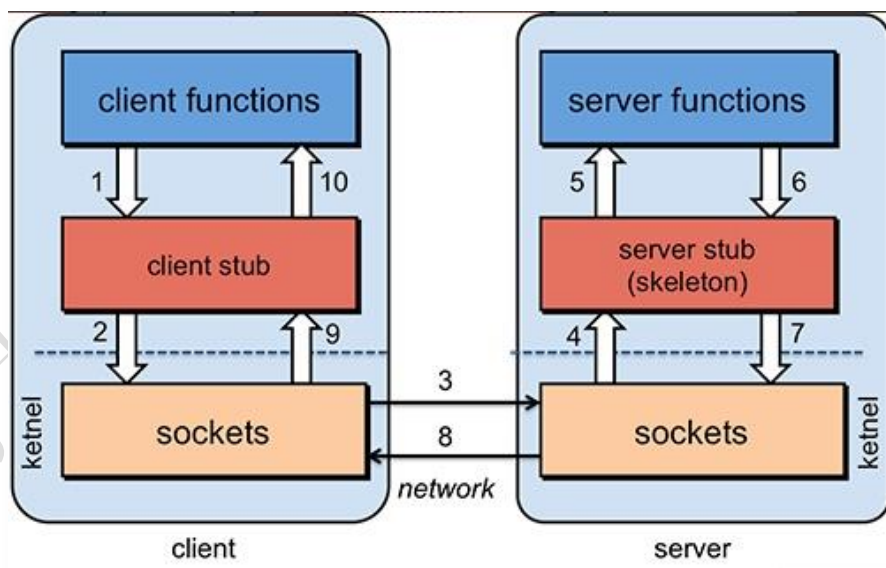
RPC 的实现方式可以不同。例如 java 的 rmi, spring 远程调用等。

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。
3. 通用：进程间通讯的方式，有通用的规则。

1.2.3 RPC 基本原理



PRC 调用过程:

- 1.调用方 client 要使用右侧 server 的功能（方法），发起对方法的调用
- 2.client stub 是 PRC 中定义的存根，看做是 client 的助手。stub 把要调用的方法参数进行序列化，方法名称和其他数据包装起来。
- 3.通过网络 socket(网络通信的技术)，把方法调用的细节内容发送给右侧的 server
- 4.server 端通过 socket 接收请求的方法名称，参数等数据，传给 stub。
- 5.server 端接到的数据由 server stub(server 的助手)处理，调用 server 的真正方法，处理业务
- 6.server 方法处理完业务，把处理的结果对象（Object）交给了助手，助手把 Object 进行序列化，对象转为二进制数据。
7. server 助手二进制数据交给网络处理程序
8. 通过网络将二进制数据，发送给 client。
- 9.client 接数据，交给 client 助手。
- 10.client 助手，接收数据通过反序列化为 java 对象（Object），作为远程方法调用结果。

其他:

rpc 通讯是基于 tcp 或 udp 议
序列化方式（xml/json/二进制）

第二章、dubbo 框架

2.1 dubbo 概述

Apache Dubbo (incubating) ['dʌbəʊ] 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 是一个分布式服务框架，致力于提供高性能和透明化的 **RPC** 远程服务调用方案、服务治理方案。

官网：<https://dubbo.apache.org/zh/>

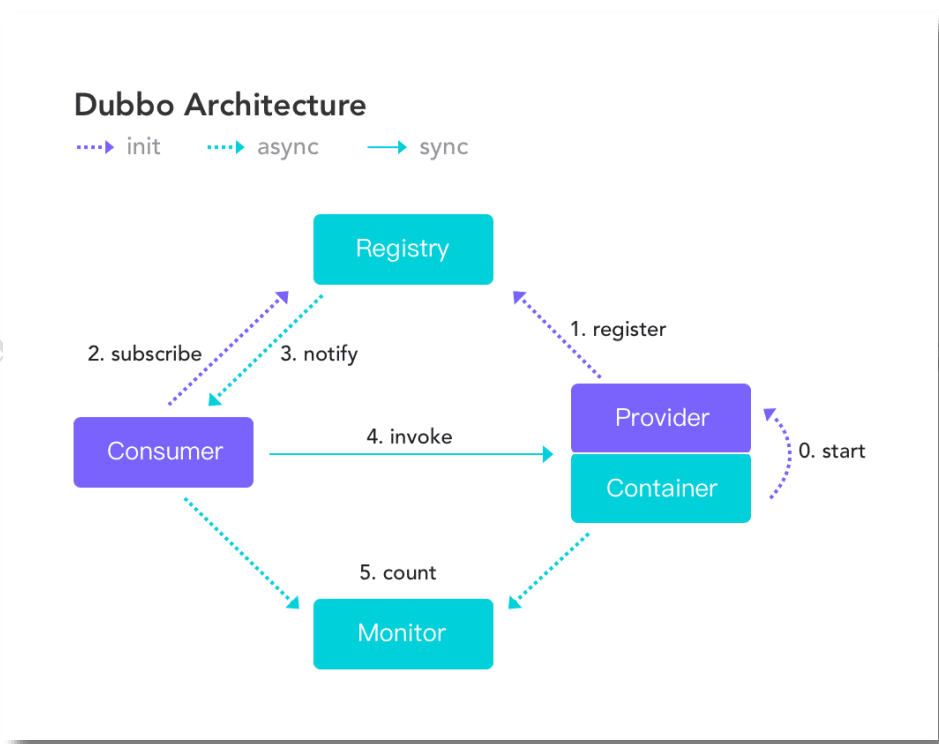


特性:



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

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

调用关系说明:

- 服务容器负责启动，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，获取用户地址
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

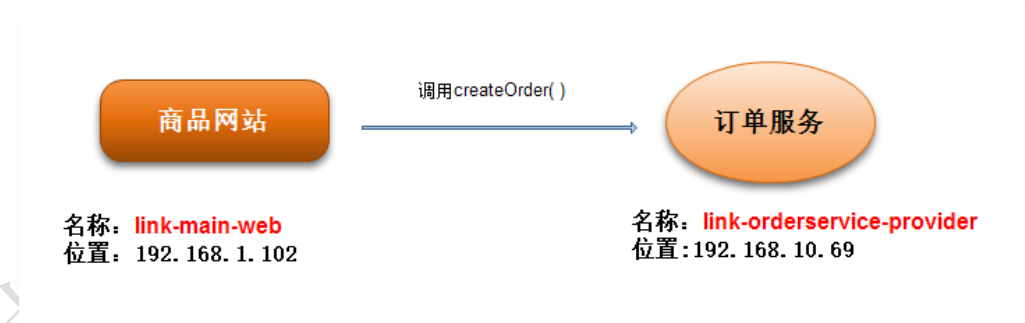
点对点的直连项目:消费者直接访问服务提供者，没有注册中心。消费者必须指定服务提供者的访问地址（url）。

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

(1) 创建服务提供者：订单服务

A、新建 java project

项目名称: **link-orderservice-provider**

设置 version 为 1.0.0

B、maven pom.xml

```
<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>中加入 plugin

```
<plugins>
```



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

C、创建订单实体类：Order

```
public class Order implements Serializable{
    private static final long serialVersionUID = 1L;
    //订单id
    private String id;
    //商品名称
    private String goodName;
    //商品单价
    private float price;
    //购买数量
    private Integer amount;
    //set, get方法
```

D、新建订单服务接口：OrderService

```
public interface OrderService {
    //用户的id, 商品名称, 单价, 数量
    public Order createOrder(Integer userId,
        String goodName, float price, Integer amount);
}
```

E、新建接口的实现类：OrderServiceImpl

```
public class OrderServiceImpl implements OrderService {

    @Override
    public Order createOrder(Integer userId,String goodName,
                             float price, Integer amount) {
        Order order = new Order();
        String orderId = UUID.randomUUID().toString().replaceAll("-", "");
        order.setId(orderId);
        order.setGoodName(goodName);
        order.setPrice(price);
        order.setAmount(amount);
        return order;
    }
}
```

F、创建 dubbo 配置文件

orderservice-provider.xml

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="link-orderservice-provider" />
<!-- 访问服务的协议名称，端口-->
<dubbo:protocol name="dubbo" port="20880"/>
<!--
    interface: 接口的全限定名称
    ref: 接口的实现bean的id
    registry: 直连方式，不使用注册中心。N/A:不使用注册中心
-->
<dubbo:service interface="com.bjpowernode.service.OrderService"
               ref="orderService" registry="N/A" />
<!-- 声明订单实现bean -->
<bean id="orderService" class="com.bjpowernode.service.impl.OrderServiceImpl" />
```

G、测试配置文件

```
public class OrderApplication {

    public static void main(String[] args) throws IOException {
        String configLocation="orderservice-provider.xml";
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(configLocation);
        ((ClassPathXmlApplicationContext)ctx).start();
        //阻塞操作，应用一直运行
        System.in.read();
    }
}
```

H、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

(2) 创建服务消费者：商品网站

A、新建 java project

项目名称：link-main-web

B、maven pom.xml

```
<dependency>
  <groupId>com.bjpowernode</groupId>
  <artifactId>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>
```

<build>加入 maven 编译插件

```
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
```

```
<configuration>
  <source>1.8</source>
  <target>1.8</target>
</configuration>
</plugin>
</plugins>
```

C、创建购买商品接口

```
public interface ShopService {

    //用户的id, 商品名称, 单价, 数量
    public Order buyGoods(Integer userId,
        String goodName,float price,Integer amount);
}
```

D、创建购买接口的实现类

```
public class ShopServiceImpl implements ShopService {
    //使用远程服务接口
    private OrderService orderService;

    public void setOrderService(OrderService orderService) {
        this.orderService = orderService;
    }

    @Override
    public Order buyGoods(Integer userId,String goodName,
        float price, Integer amount) {
        //调用远程方法创建订单
        Order order = orderService.createOrder(userId, goodName, price, amount);
        return order;
    }
}
```

E、创建 dubbo 配置文件

shop-consume.xml

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="Link-main-web" />

<!-- 引用远程服务接口
    id: 远程接口的代理对象名称
    interface: 远程接口的全限定名称
    url: 访问服务提供者的地址
-->
<dubbo:reference id="remoteOrderService"
    interface="com.bjpowernode.service.OrderService"
    url="dubbo://localhost:20880" registry="N/A" />

<bean id="shopService" class="com.bjpowernode.service.impl.ShopServiceImpl">
    <property name="orderService" ref="remoteOrderService" />
</bean>
```

F、执行消费者

```
public class ConsumeApplication {

    public static void main(String[] args) throws IOException {
        String configLocation="shop-consume.xml";
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(configLocation);
        ((ClassPathXmlApplicationContext)ctx).start();

        ShopService service = (ShopService) ctx.getBean("shopService");
        Order order = service.buyGoods(1, "手机", 2000, 2);
        System.out.println("order:"+order);
    }
}
```

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。

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，否则将面临分布式事务问题，Dubbo 暂未提供分布式事务支持。

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，为后续不兼容升级提供可能，如：<dubbo:service interface="com.xxx.XxxService" version="1.0" />。

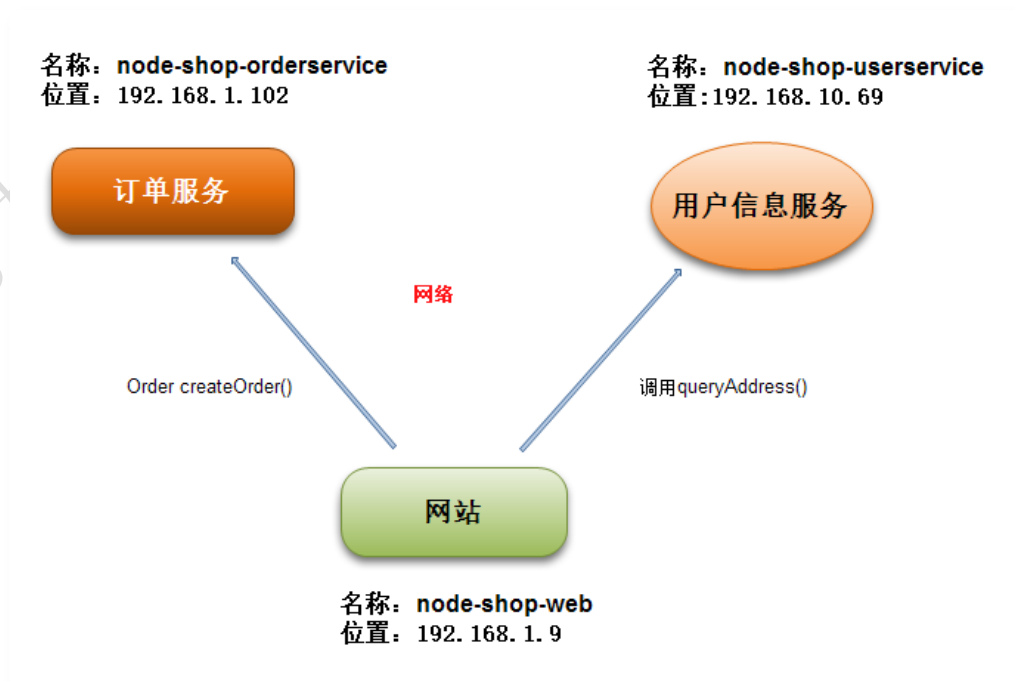
建议使用两位版本号，要变更服务版本。先升级一半提供者为新版本，再将消费者全部升为新版本，然后将剩下的一半提供者升为新版本。

2.7 改造 dubbo 项目

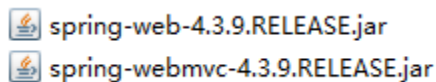
抽象分散在多个项目中的公共接口，实体类，异常，工具类到一个项目中，在其他项目如服务提供者，消费者共用公共的资源。

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。

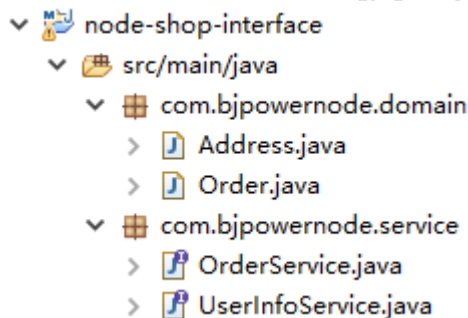
A、创建公共的 maven java project

项目名称: node-shop-interface

修改 version

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.bjpowernode</groupId>
  <artifactId>node-shop-interface</artifactId>
  <version>1.0.0</version>
</project>
```

复制之前 link-orderservice-provider 项目的接口文件，实体类文件



C、新建 Address 实体类

```
public class Address implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
    //收件人姓名  
    private String name;  
    //城市  
    private String city;  
    //街道  
    private String street;  
    //邮编  
    private String zipcode;  
    //手机号  
    private String mobile;  
    //是否使用此地址  
    private Boolean use;  
    //set , get 方法
```

D、新建 UserInfoService 接口

```
public interface UserInfoService {  
  
    List<Address> queryAddress(Integer userId);  
}
```

E、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

B、maven pom.xml

```
<dependency>  
    <groupId>com.bjpowernode</groupId>  
    <artifactId>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>

<build>加入 maven 编译插件
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
    <configuration>
      <source>1.8</source>
      <target>1.8</target>
    </configuration>
  </plugin>
</plugins>
```

C、创建 UserInfoServiceImpl 实现类

```
public class UserInfoServiceImpl implements UserInfoService {  
    @Override  
    public List<Address> queryAddress(Integer userId) {  
        List<Address> adds = new ArrayList<>();  
        if( userId == 1) {  
            Address address = new Address();  
            address.setName("张三");  
            address.setCity("北京");  
            address.setStreet("大兴区凉水河二街");  
            //其他属性赋值  
            address.setUse(true);  
            adds.add(address);  
            //其他Address对象的创建  
        } else if(userId == 2) {  
            Address address = new Address();  
            address.setName("赵明");  
            address.setCity("北京");  
            address.setStreet("西城区西单10号");  
            address.setUse(true);  
            //其他属性赋值  
            adds.add(address);  
            //其他Address对象的创建  
        }  
        return adds;  
    }  
}
```

D、dubbo 配置文件

文件名称: userservice-provider.xml

```
<!-- 服务的名称,使用唯一值, 服务名称是dubbo内部使用标识服务的 -->  
<dubbo:application name="node-shop-userservice" />  
  
<!-- 访问服务的协议名称, 端口-->  
<dubbo:protocol name="dubbo" port="20881"/>  
  
<!--  
    声明暴露的服务, 消费调用此接口的方法  
    interface:接口的全限定名称  
    ref: 接口的实现类对象id  
-->  
<dubbo:service interface="com.bjpowernode.service.UserInfoService"  
    ref="userService" registry="N/A"/>  
  
<!-- 服务接口的实现对象: 功能的真正实现者 -->  
<bean id="userService" class="com.bjpowernode.service.impl.UserInfoServiceImpl" />
```

E、web.xml 注册 spring 监听器

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:userservice-provider.xml</param-value>
</context-param>
<!-- spring监听器 -->
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

2.7.4 创建订单服务

A、新建 web project

项目名称: node-shop-orderservice

B、maven pom.xml

```
<dependency>
    <groupId>com.bjpowernode</groupId>
    <artifactId>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>
```

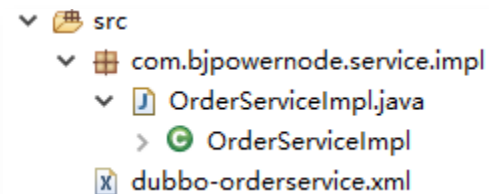
```
<artifactId>dubbo</artifactId>
<version>2.6.2</version>
</dependency>
```

<build>加入 maven 编译插件

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

C、创建 OrderService 接口实现类

拷贝 link-order-service-provider 的实现类



D、dubbo 配置文件

文件名称: dubbo-orderservice.xml

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="node-shop-orderservice" />
<!-- 访问服务的协议名称，端口-->
<dubbo:protocol name="dubbo" port="20882"/>
<!--
  interface:接口的全限定名称
  ref: 接口的实现bean的id
  registry: 直连方式，不使用注册中心。N/A:不使用注册中心
-->
<dubbo:service interface="com.bjpowernode.service.OrderService"
  ref="orderService" registry="N/A" />

<!-- 声明订单实现bean -->
<bean id="orderService" class="com.bjpowernode.service.impl.OrderServiceImpl" />
```

E、web.xml 注册 spring 监听器

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:dubbo-orderservice.xml</param-value>
</context-param>
<!-- spring监听器 -->
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

2.7.5 创建商品网站

A、新建 web project

项目名称: node-shop-web

B、创建首页 index.jsp

商品列表

名称	单价	库存	操作
iphone	5000	20	购买
thinkpad	8500	充足	购买

[我的收件地址](#)

部分代码:

```
<div style="margin-left:400px">
  <table border="1" cellpadding="1" cellspacing="1" width="60%">
    <p>商品列表</p>
  </table>
</div>
```

购买按钮

```
<tr>
  <td>iphone</td>
```

```
<td>5000</td>
<td>20</td>
<td><a href="javascript:void(0)" onClick="buyGoods(1,'iphone','5000','1')">购买</td>
</tr>
```

我的收件地址

```
<div style="margin-left:450px">
  <a href="javascript:void(0)" onClick="showAddress(2)" >我的收件地址</a>
</div>
```

js方法

```
<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>
```

C、maven pom.xml

```
<dependency>
  <groupId>com.bjpowernode</groupId>
  <artifactId>nod-shop-interface</artifactId>
  <version>1.0.0</version>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</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>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>4.3.16.RELEASE</version>
</dependency>

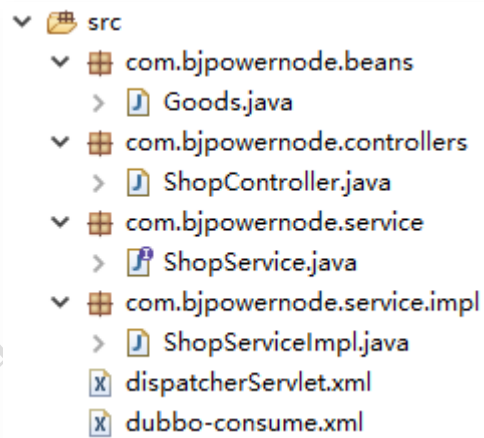
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.3.16.RELEASE</version>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>dubbo</artifactId>
  <version>2.6.2</version>
</dependency>
```

<build>加入 maven 编译插件

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

D、项目结构



```
src
├── com.bjpowernode.beans
│   └── Goods.java
├── com.bjpowernode.controllers
│   └── ShopController.java
├── com.bjpowernode.service
│   └── ShopService.java
├── com.bjpowernode.service.impl
│   └── ShopServiceImpl.java
├── dispatcherServlet.xml
└── dubbo-consume.xml
```

E、创建商品信息实体类 Goods

```
public class Goods {

    private String name;
    private Float price;
    private Integer amount;
    // set | get 方法
}
```

F、创建接口 ShopService

```
public interface ShopService {
    public Order createOrder(Integer userId, Goods goods);
    public List<Address> showAddress(Integer userId);
}
```

G、创建接口实现类 ShopServiceImpl

```
public class ShopServiceImpl implements ShopService {  
    //远程接口作为属性  
    private OrderService orderService;  
    private UserInfoService userInfoService;  
  
    public void setOrderService(OrderService orderService) {  
        this.orderService = orderService;  
    }  
    public void setUserInfoService(UserInfoService userInfoService) {  
        this.userInfoService = userInfoService;  
    }  
    @Override  
    public Order createOrder(Integer userId, Goods goods) {  
        Order order = orderService.createOrder(userId, goods.getName(),  
            goods.getPrice().floatValue(), goods.getAmount());  
        return order;  
    }  
    @Override  
    public List<Address> showAddress(Integer userId) {  
        List<Address> addresses = userInfoService.queryAddress(userId);  
        return addresses;  
    }  
}
```

H、创建类 ShopController

接收来自页面的请求，处理订单，查询地址信息。

```
@Controller
@RequestMapping("/shop")
public class ShopController {
    @Autowired
    private ShopService shopService;
    @RequestMapping("/buy")
    public ModelAndView buyGoods(Integer userId, Goods goods) {
        ModelAndView mv = new ModelAndView();
        Order order = shopService.createOrder(userId, goods);
        mv.addObject("order", order);
        mv.setViewName("view-order");
        return mv;
    }
    @RequestMapping("/addresses")
    public ModelAndView showListAddress(Integer userId){
        ModelAndView mv = new ModelAndView();
        List<Address> addresses = shopService.showAddress(userId);
        mv.addObject("addresses", addresses);
        mv.setViewName("view-address");
        return mv;
    }
}
```

I、 view-order.jsp

显示订单信息:



```
<body>
<br/>
<div align="align" style="margin-left:400px">
    view-order.jsp <br>
    <h5>您的订单已经生成: </h5>
    <h5>订单号: ${order.id }</h5>
    <h5>商品: ${order.goodName }</h5>
</div>
</body>
```

J、 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:

 jstl-1.2.jar
 standard.jar

```
<body>
<br/>
<div style="margin-left:400px">
  view-address.jsp <br>
  <c:forEach items="${addresses}" var="address" >
    <h5>收件人: ${address.name }</h5>
    <h5>收件地址:${address.city }${address.street }</h5>
    <h5>联系电话: ${address.mobile }</h5>
    <h5>邮编: ${address.zipcode }</h5>
    <h5>默认地址: ${address.use }</h5>
    <br>
  </c:forEach>
</div>
</body>
```

K、新建 spring 配置文件

文件名称: dispatcherServlet.xml

```
<!-- 声明组件扫描器 -->
<context:component-scan base-package="com.bjpowernode.controllers" />

<!-- 声明视图解析器 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/" />
  <property name="suffix" value=".jsp" />
</bean>

<!-- 声明注解驱动 -->
<mvc:annotation-driven />
```

L、新建 dubbo 配置文件

文件名称: dubbo-consume.xml

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="node-shop-web" />

<dubbo:reference id="remoteUserService"
    interface="com.bjpowernode.service.UserInfoService"
    url="dubbo://localhost:20881" registry="N/A" check="false"/>

<dubbo:reference id="remoteOrderService"
    interface="com.bjpowernode.service.OrderService"
    url="dubbo://localhost:20882" registry="N/A" check="false"/>

<bean id="shopService" class="com.bjpowernode.service.impl.ShopServiceImpl">
    <property name="orderService" ref="remoteOrderService" />
    <property name="userInfoService" ref="remoteUserService" />
</bean>
```

M、 web.xml 注册 DispatcherServlet

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:dispatcherServlet.xml,classpath:dubbo-consume.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

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

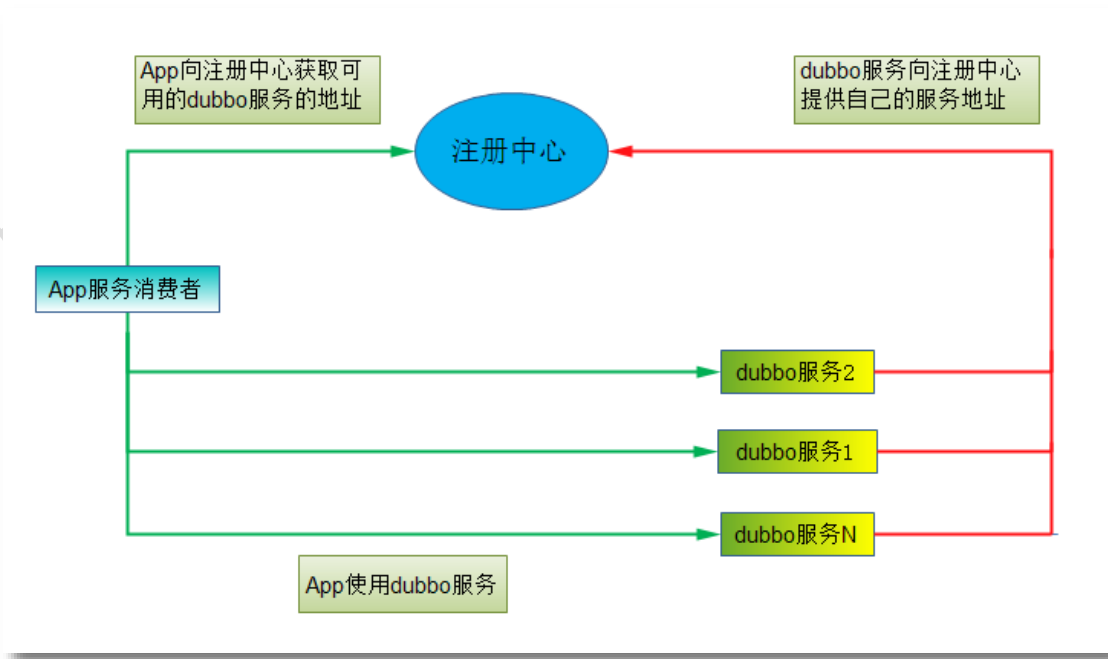
Redis 注册中心：使用 Redis 作为注册中心

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。

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，分布式的，开放源码的分布式应用程序协调服务。简称 zk。Zookeeper 是翻译管理是动物管理员。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/>

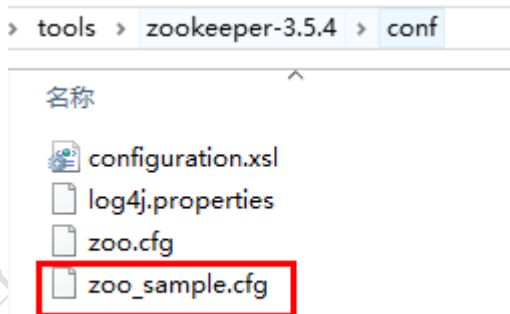
进入官网地址，首页找到下载地址，最新版本 3.5.4

3.3.2 安装配置 Zookeeper

A、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：

```
1 # The number of milliseconds of each tick
2 tickTime=2000
3 # The number of ticks that the initial
4 # synchronization phase can take
5 initLimit=10
6 # The number of ticks that can pass between
7 # sending a request and getting an acknowledgement
8 syncLimit=5
9 # the directory where the snapshot is stored.
10 # do not use /tmp for storage, /tmp here is just
11 # example sake.
12 dataDir=F:/data/zookeeper
13 # the port at which the clients will connect
14 clientPort=2181
15 # the maximum number of client connections.
```

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 占用 8080

B、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

```
[root@localhost conf]# ll
total 12
-rw-rw-r--. 1 mytest mytest 535 Mar 23 18:14 configuration.xml
-rw-rw-r--. 1 mytest mytest 2161 Mar 23 18:14 log4j.properties
-rw-rw-r--. 1 mytest mytest 922 Mar 23 18:14 zoo_sample.cfg
[root@localhost conf]# cp zoo_sample.cfg zoo.cfg
[root@localhost conf]# ll
total 16
-rw-rw-r--. 1 mytest mytest 535 Mar 23 18:14 configuration.xml
-rw-rw-r--. 1 mytest mytest 2161 Mar 23 18:14 log4j.properties
-rw-r--r--. 1 root root 922 Sep 2 11:21 zoo.cfg
-rw-rw-r--. 1 mytest mytest 922 Mar 23 18:14 zoo_sample.cfg
```

③：启动 Zookeeper

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

```
[root@localhost bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper-3.4.10/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost bin]# ps -ef | grep zoo
root      3569      1  17 11:22 pts/0    00:00:01 /usr/local/jdk1.8.0_
,CONSOLE -cp /usr/local/zookeeper-3.4.10/bin/../build/classes:/usr/loc
```

④：关闭 Zookeeper

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

```
[root@localhost bin]# ./zkServer.sh stop
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper-3.4.10/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
```

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 zookeeper 相关 jar:

```
<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：

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="zk-node-shop-orderservice" />
<!-- 注册中心zookeeper -->
<dubbo:registry address="zookeeper://localhost:2181" />
<!-- 访问服务的协议名称，端口-->
<dubbo:protocol name="dubbo" port="20882"/>
<!--
    interface:接口的全限定名称
    ref: 接口的实现bean的id
-->
<dubbo:service interface="com.bjpowernode.service.OrderService"
    ref="orderService" />

<!-- 声明订单实现bean -->
<bean id="orderService" class="com.bjpowernode.service.impl.OrderServiceImpl" />
```

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

加入红色方框中的内容：

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="zk-node-shop-userservice" />
<!-- 注册中心zookeeper -->
<dubbo:registry address="zookeeper://localhost:2181" />

<!-- 访问服务的协议名称，端口-->
<dubbo:protocol name="dubbo" port="20881"/>
<!--
    interface:接口的全限定名称
    ref: 接口的实现bean的id
-->
<dubbo:service interface="com.bjpowernode.service.UserInfoService"
    ref="userService" />

<!-- 服务接口的实现对象：功能的真正实现者 -->
<bean id="userService" class="com.bjpowernode.service.impl.UserInfoServiceImpl" />
```

3.4.3 zk-node-shop-web 网站 dubbo 配置文件

加入红色方框内文字：

```
<!-- 服务的名称, 使用唯一值, 服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="zk-node-shop-web" />

<!-- 注册中心zookeeper -->
<dubbo:registry address="zookeeper://localhost:2181" />

<dubbo:reference id="remoteUserService"
    interface="com.bjpowernode.service.UserInfoService" check="false"/>

<dubbo:reference id="remoteOrderService"
    interface="com.bjpowernode.service.OrderService" check="false"/>

<bean id="shopService" class="com.bjpowernode.service.impl.ShopServiceImpl">
    <property name="orderService" ref="remoteOrderService" />
    <property name="userInfoService" ref="remoteUserService" />
</bean>
```

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

1.3 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**。特别是项目需要把早期接口的实现全部换位新的实现类，也需要使用 **version**。

可以用版本号从早期的接口实现过渡到新的接口实现，版本号不同的服务相互间不引用。

可以按照以下的步骤进行版本迁移：

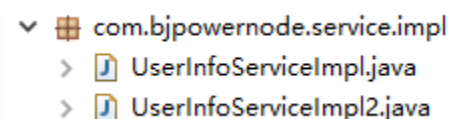
在低压力时间段，先升级一半提供者为新版本

再将所有消费者升级为新版本

然后将剩下的一半提供者升级为新版本

例：以 zk-node-shop-userservice 为例。

4.4.1 复制 UserInfoServiceImpl.java



UserInfoServiceImpl2 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

```
<!-- 服务的名称，使用唯一值，服务名称是dubbo内部使用标识服务的 -->
<dubbo:application name="zk-node-shop-userservice" />

<!-- 注册中心zookeeper -->
<dubbo:registry address="zookeeper://localhost:2181" />

<!-- 访问服务的协议名称，端口-->
<dubbo:protocol name="dubbo" port="20881"/>
<!--
    interface: 接口的全限定名称
    ref: 接口的实现bean的id
-->
<dubbo:service interface="com.bjpowernode.service.UserInfoService"
    ref="userService-v1" version="1.0"/>
<dubbo:service interface="com.bjpowernode.service.UserInfoService"
    ref="userService-v2" version="2.0"/>

<!-- 服务接口的实现对象：功能的真正实现者 -->
<bean id="userService-v1" class="com.bjpowernode.service.impl.UserInfoServiceImpl" />
<bean id="userService-v2" class="com.bjpowernode.service.impl.UserInfoServiceImpl2" />
```

4.4.4 zk-node-shop-web 服务的 dubbo 配置文件

增加访问的 version=2.0

```
<dubbo:reference id="remoteUserService"
    interface="com.bjpowernode.service.UserInfoService"
    check="false" version="2.0"/>
```

4.4.5 测试应用

1. 先启动 zookeeper
2. 启动 tomcat
3. 访问 zk-node-shop-web

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

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

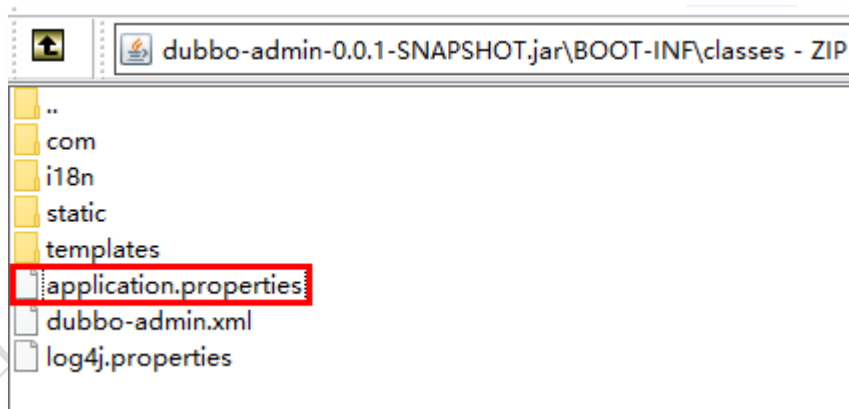
这里下载的是源代码，需要手工编译才能使用。

B、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
```

C、修改配置 dubbo-properties 文件



application.properties 文件，内容如下：

```
server.port=7001      访问应用的端口
spring.velocity.cache=false
spring.velocity.charset=UTF-8
spring.velocity.layout-url=/templates/default.vm
spring.messages.fallback-to-system-locale=false
spring.messages.basename=i18n/message
spring.root.password=root  ← root用户的密码
spring.guest.password=guest

dubbo.registry.address=zookeeper://127.0.0.1:2181
                        注册中心地址
```

D、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。