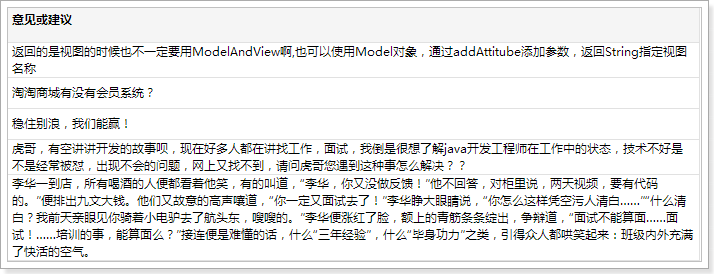
**day15-淘淘商城**

# 今日大纲

* Dubbo入门学习
* 使用dubbo优化单点登录系统

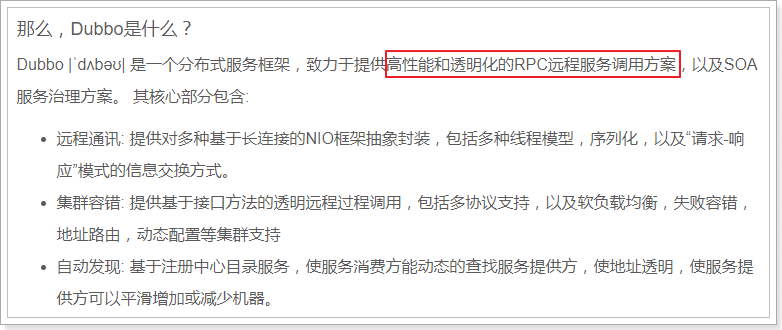


# 课堂笔记

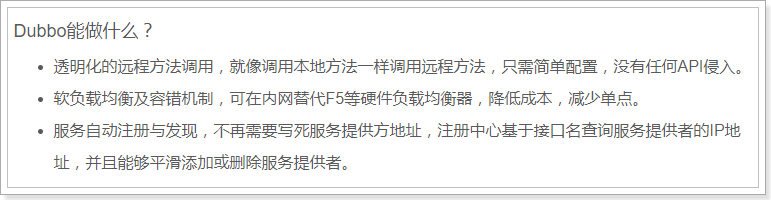
# dubbo概述

## 什么是dubbo

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，是阿里巴巴SOA服务化治理方案的核心框架，每天为\*\*2,000+个服务提供3,000,000,000+\*\*次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点。



## 有什么用？



## 什么是RPC



RPC，即远程过程调用，是一种服务间调用的方式。用大白话来说，就是让你的程序像调用自己项目中方法一样去远程调用其它程序中的方法。

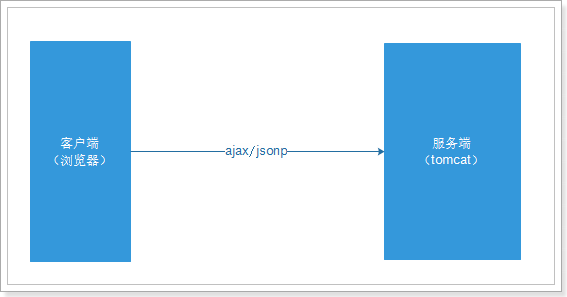
RMI：

## 其它远程通信方式

除了RPC以外，我们以前还学过哪些远程调用方式？

### 浏览器直接访问

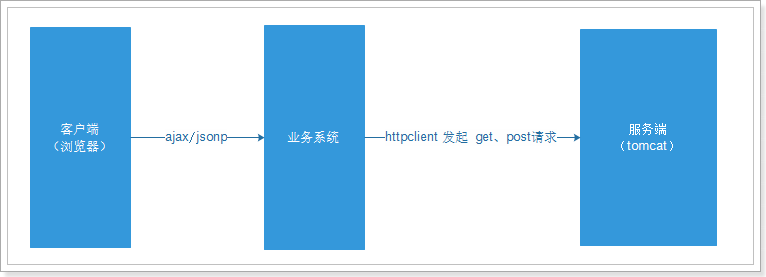
浏览器发起请求，通过ajax或jsonp方式请求：



发起的是Http请求的，使用Http协议。

### Httpclient方式

系统与系统之间通过Httpclient发起http请求来请求数据：



发起的是Http请求的，使用Http协议。

我们发现，无论是Ajax请求，还是HttpClient都是通过 Http请求来访问服务端，使用的都是Http协议，而http协议是短连接。

RPC的底层协议，大部分都是使用长连接，减少了建立和销毁连接的次数，大大提高了程序的访问效率，但是会导致连接长期被占用。

## 为什么要用dubbo

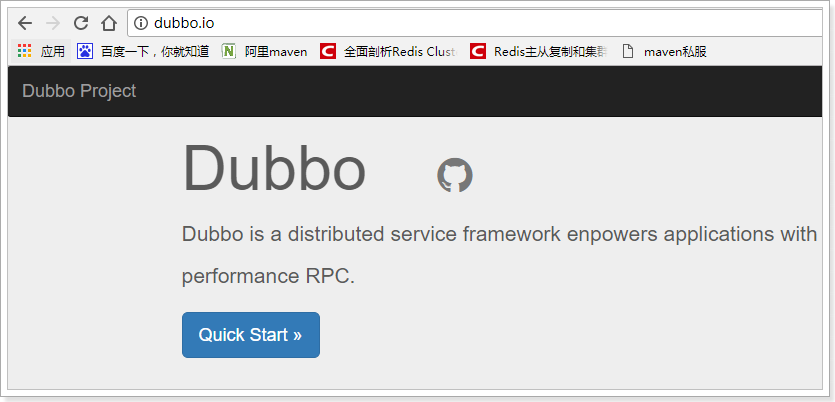
在用户量很大，请求频繁的场景下，例如用户访问我们的网站服务，请求数量很大，请求连接数有限，使用短连接比较合适。此时使用长连接就不太合适。

当请求连接数不多，但是请求非常频繁的时候，例如系统间访问，此时使用长连接，在一次连接中完成尽可能多的数据交互，效率更高。

而系统的调用在分布式系统中，十分的常见。这种场景下，使用RPC会非常合适。而Dubbo就是一款非常优秀的RPC服务架构

## 官网

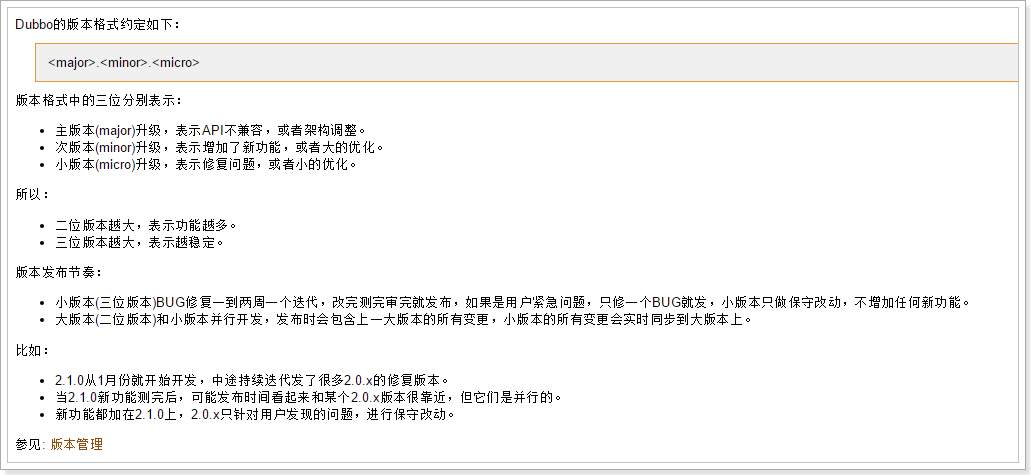
官网：<http://dubbo.io>

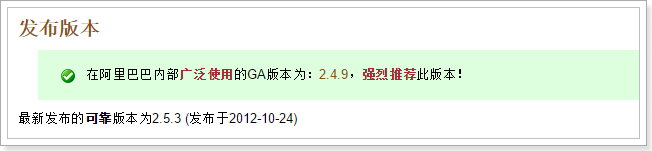


官方文档：<http://dubbo.io/user-guide/>



## 版本及说明

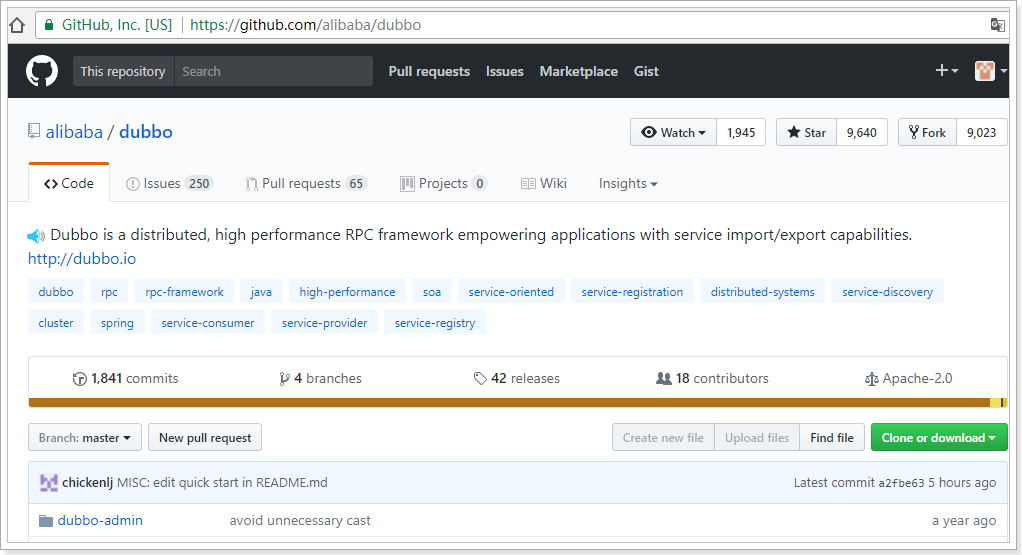




我们使用2.5.3版本。

## 下载

dubbo官网并没有提供下载，如果要下载可以到其github页面去下载：<https://github.com/alibaba/dubbo>



maven仓库下载地址：<http://repo1.maven.org/maven2/com/alibaba/dubbo/>

阿里云maven仓库：<http://maven.aliyun.com/nexus/content/groups/public/com/alibaba/dubbo/>

# 构建dubbo（可选）

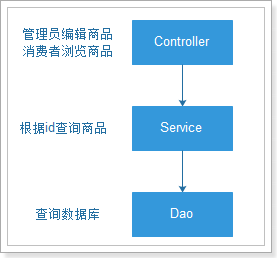
参考课前资料：《Maven构建Dubbo模块》

# Dubbo的架构

## 系统架构的演变

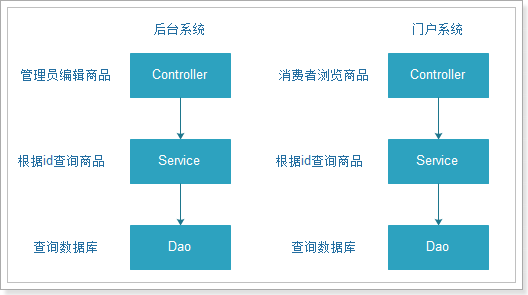
* **阶段1：**集中式架构：

有耦合性高，无法水平扩展（集群）的问题，并发能力差



* **阶段2：**功能拆分

按功能进行垂直拆分：

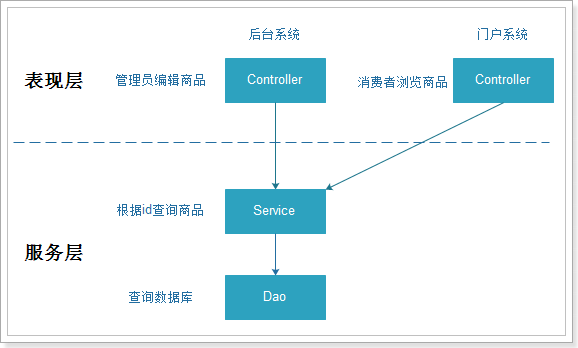


虽然解决集中式架构存在的问题，但是多个系统间，会有一些相同的功能逻辑，重复编码。

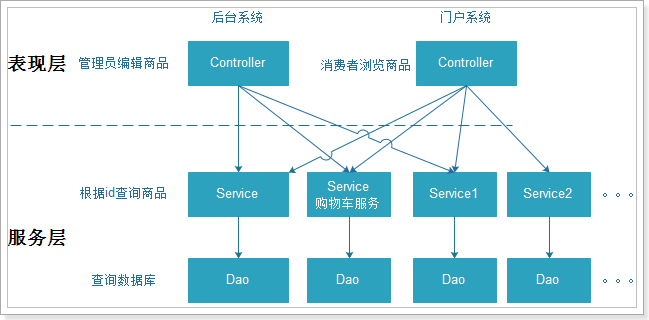
如上图，两个系统都需要实现根据商品id查询商品的功能

* **阶段3：**分布式架构服务

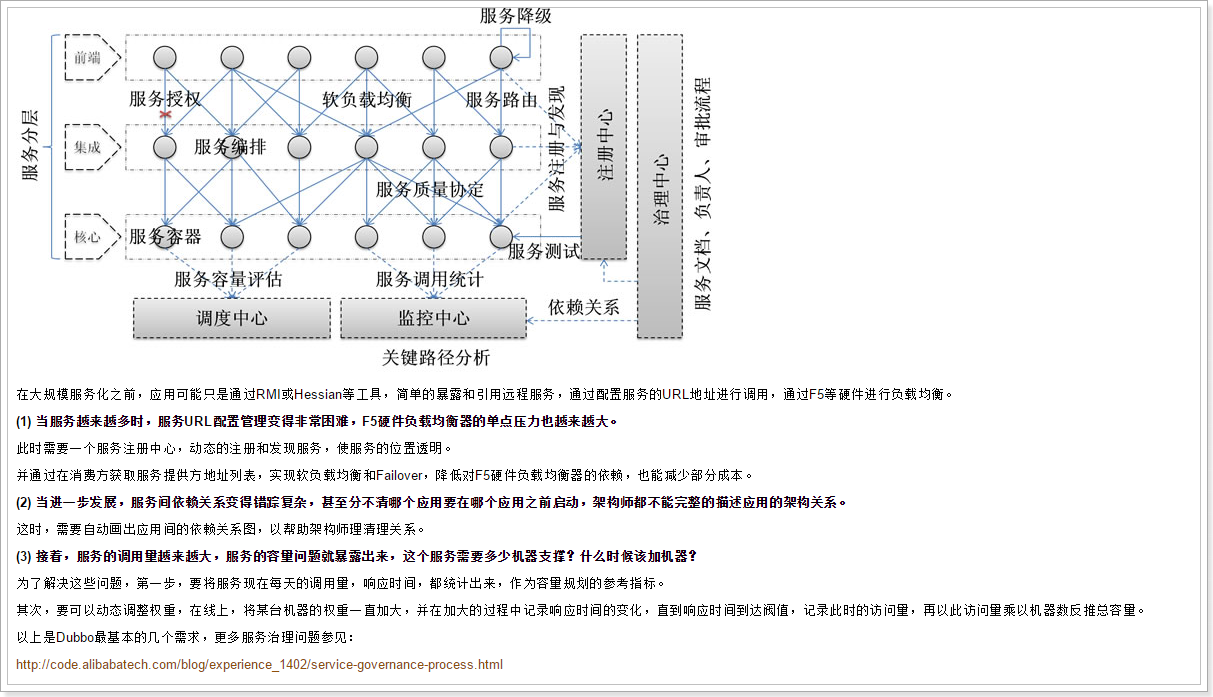
形成分布式架构，把 核心业务 抽取成服务，形成服务中心



存在的问题：当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。 此时，用于提高机器利用率的资源调度和治理中心



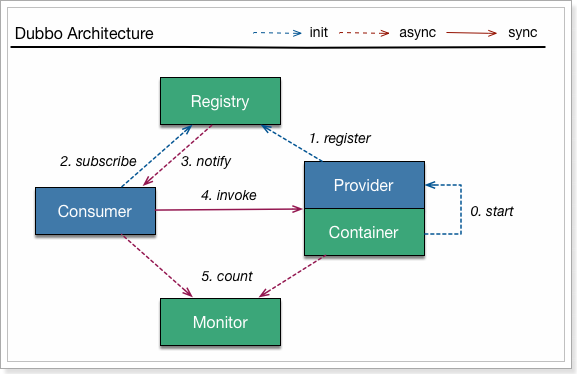
* **阶段4：**服务治理



Dubbo就是资源调度和治理中心，用来解决这些问题的

## Dubbo的架构

* **架构图：**



container：服务提供方的容器，启动服务提供者。

Provider：服务的提供方，提供接口服务被别人调用。

Registry：注册中心，实现服务的自动注册和发现，并且推送服务信息给消费者

Consumer：服务的消费方，远程调用服务的提供者。

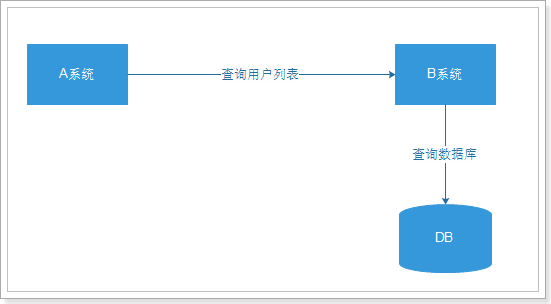
Monitor：服务的监控者，监控服务的调用情况，并且形成统计。

* **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。

# Dubbo的入门案例

## 功能需求：

存在2个系统，A系统和B系统，A系统调用B系统的接口获取数据，用于查询用户列表。



需求中重要角色：

consumer ：服务的消费者，本例中就是系统A

provider：服务的提供者，本例中就是系统B

registry：dubbo支持很多种注册中心。我们这里计划采用zookeeper作为注册中心，也是比较流行的用法。

monitor：默认

container：B系统的容器可以有各种各样，此处我们不使用Tomcat作为启动容器

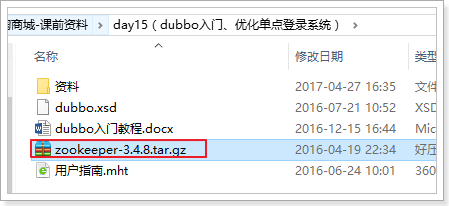
原因，官方不推荐使用Tomcat容器：



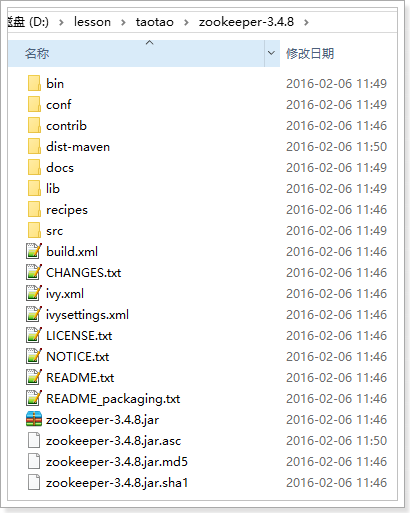
## 安装zookeeper

### 解压

课前资料中：

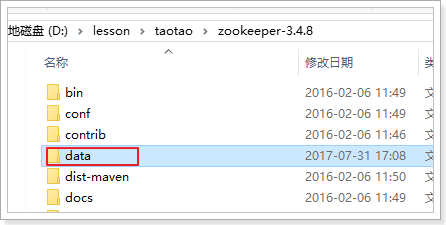


解压后：

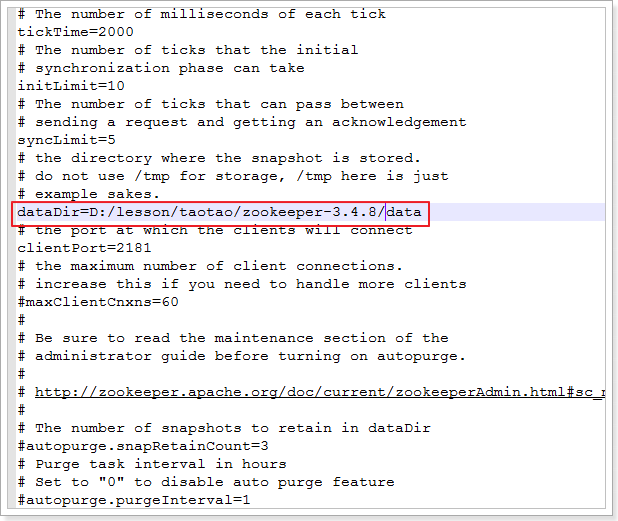


### 修改配置

先在解压的zookeeper目录下创建新目录：data



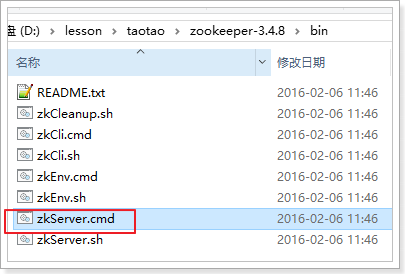
进入conf目录，修改zoo.sample.cfg为zoo.cfg，然后打开并编辑：

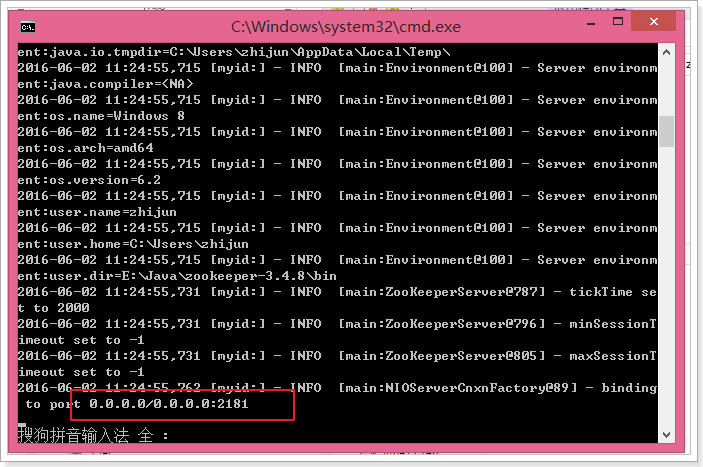


**注意：**这个目录修改为你刚刚创建的data目录。

### 运行

进入 zookeeper-3.4.8/bin目录：运行zkServer.cmd





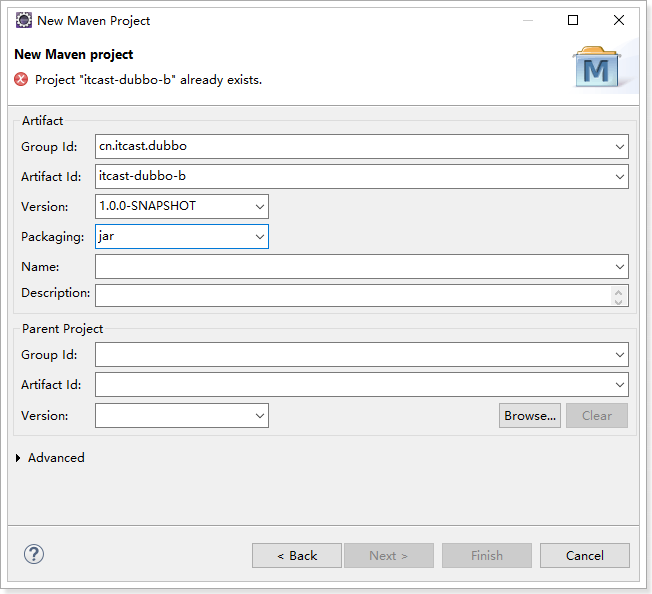
## 搭建B系统（provider）

### 创建工程

因为我们使用Tomcat作为容器，也可以直接使用Dubbo的内置容器。Dubbo推荐是使用内置的容器：



因此我们这里创建jar工程，使用默认的main函数启动：



### 导入依赖

<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.dubbo</groupId>

<artifactId>itcast-dubbo-b</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>jar</packaging>

<dependencies>

<!-- dubbo采用spring配置方式，所以需要导入spring容器依赖 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.6.4</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.3</version>

<exclusions>

<exclusion>

<!-- 排除传递spring依赖 -->

<artifactId>spring</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<!-- zookeeper依赖 -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.3.3</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<port>8081</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

### 配置文件

#### log4j

log4j.properties

log4j.rootLogger=DEBUG,A1

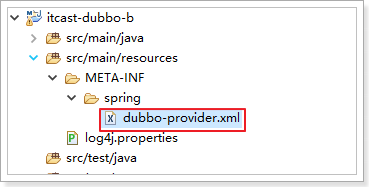
log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n

#### dubbo配置文件

这个配置文件，按照约定需要放到classpath下的 META-INF/spring目录：



<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"dubbo-b-server"* />

<!-- 这里使用的注册中心是zookeeper -->

<dubbo:registry address=*"zookeeper://127.0.0.1:2181"* client=*"zkclient"*/>

<!-- 用dubbo协议在20880端口暴露服务 -->

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

<!-- 将该接口暴露到dubbo中 -->

<dubbo:service interface=*"cn.itcast.dubbo.service.UserService"* ref=*"userServiceImpl"* />

<!-- 将具体的实现类加入到Spring容器中 -->

<bean id=*"userServiceImpl"* class=*"cn.itcast.dubbo.service.impl.UserServiceImpl"* />

</beans>

### 查询用户列表的接口

#### 实体类

// 使用dubbo要求传输的对象必须实现序列化接口

**public** **class** User **implements** java.io.Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = -2668999087589887337L;

**private** Long id;

**private** String username;

**private** String password;

**private** Integer age;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

}

#### UserService接口

/\*\*

\* UserService接口

\* **@author** 虎哥

\*

\*/

**public** **interface** UserService {

/\*\*

\* 查询所有的用户数据

\*

\* **@return**

\*/

**public** List<User> queryAll();

}

#### UserService实现类

/\*\*

\* UserService实现类

\*

\* **@author** 虎哥

\*

\*/

**public** **class** UserServiceImpl **implements** UserService {

/\*\*

\* 实现查询用户列表，这里做模拟实现，不做具体的数据库查询

\*/

**public** List<User> queryAll() {

List<User> list = **new** ArrayList<User>();

**for** (**int** i = 0; i < 10; i++) {

User user = **new** User();

user.setAge(10 + i);

user.setId(Long.*valueOf*(i + 1));

user.setPassword("123456");

user.setUsername("username\_" + i);

list.add(user);

}

**return** list;

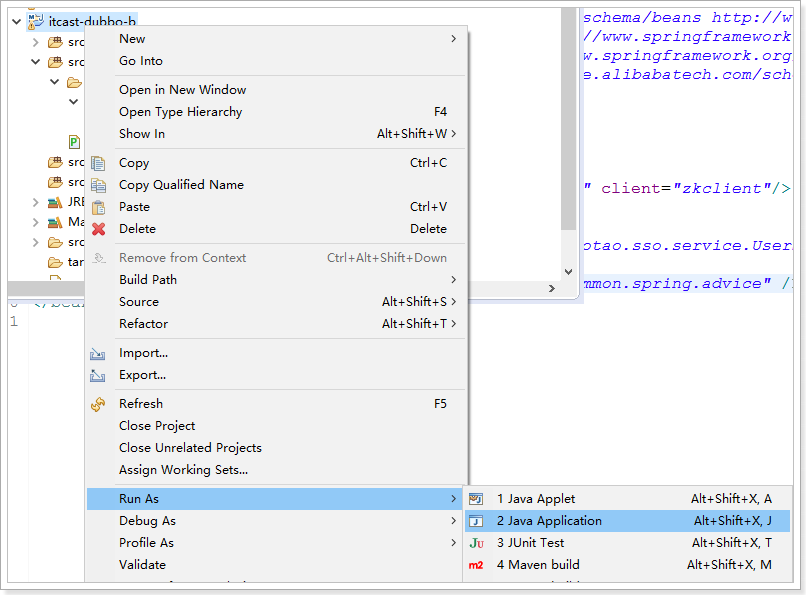
}

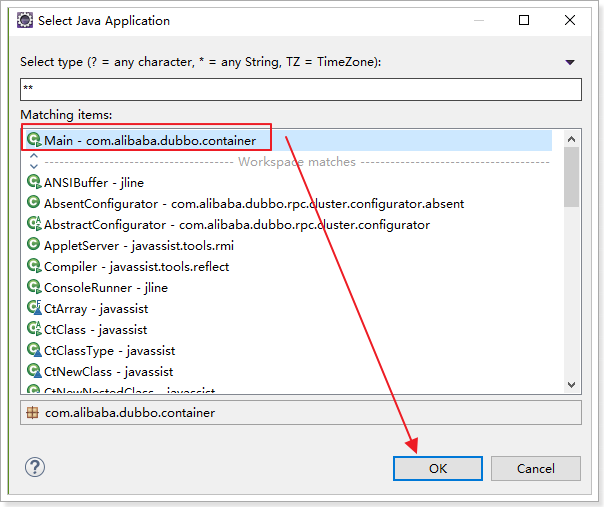
}

### 启动服务

启动一个带有main函数的项目

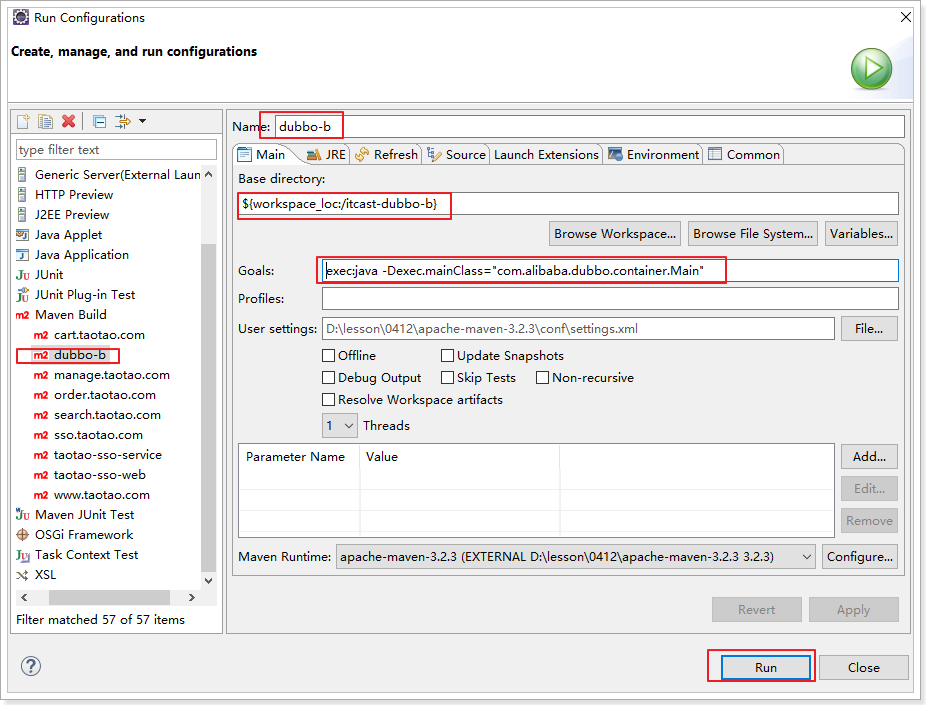
* **方式1：**





* **方式2：maven运行**

exec:java -Dexec.mainClass="com.alibaba.dubbo.container.Main"

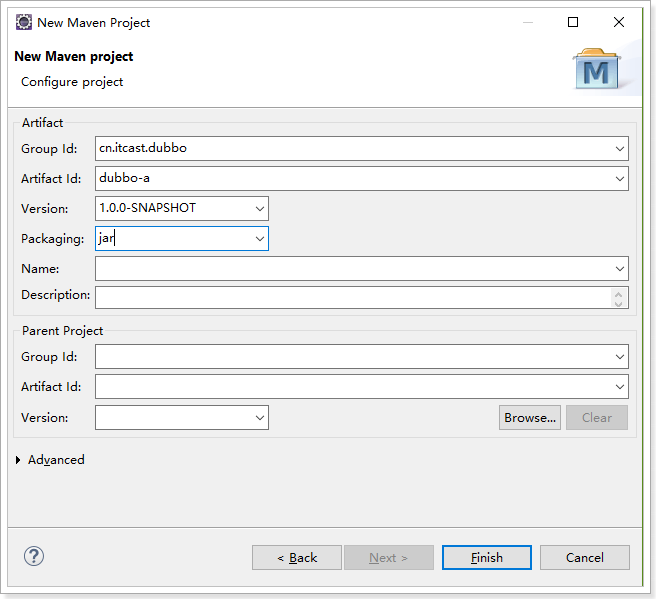


查看日志，服务已经被注册了：

2017-08-01 08:54:38,917 [localhost-startStop-1] [com.alibaba.dubbo.config.AbstractConfig]-[INFO] [DUBBO] Register dubbo service cn.itcast.dubbo.service.UserService url dubbo://172.16.1.62:20880/cn.itcast.dubbo.service.UserService?anyhost=true&application=dubbo-b-server&dubbo=2.5.3&interface=cn.itcast.dubbo.service.UserService&methods=queryAll&pid=8816&side=provider&timestamp=1501548878841 to registry registry://127.0.0.1:2181/com.alibaba.dubbo.registry.RegistryService?application=dubbo-b-server&client=zkclient&dubbo=2.5.3&pid=8816&registry=zookeeper&timestamp=1501548878832, dubbo version: 2.5.3, current host: 127.0.0.1

## 搭建A系统（consumer）

### 创建工程



### 导入依赖

<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.dubbo</groupId>

<artifactId>itcast-dubbo-a</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<!-- dubbo采用spring配置方式，所以需要导入spring容器依赖 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.6.4</version>

</dependency>

<!-- dubbo依赖，排除dubbo中的spring依赖，版本较低 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.3</version>

<exclusions>

<exclusion>

<!-- 排除传递spring依赖 -->

<artifactId>spring</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<!-- zookeeper -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.3.3</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<!-- spring测试 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.1.3.RELEASE</version>

<scope>test</scope>

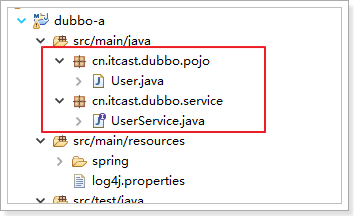
</dependency>

</dependencies>

</project>

### 拷贝接口

从B系统拷贝User实体类，和UserService接口。无序实现类



### 编写配置：

#### log4j配置

略。。直接从B系统拷贝

#### dubbo配置

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"dubbo-a-consumer"* />

<!-- 这里使用的注册中心是zookeeper -->

<dubbo:registry address=*"zookeeper://127.0.0.1:2181"* client=*"zkclient"*/>

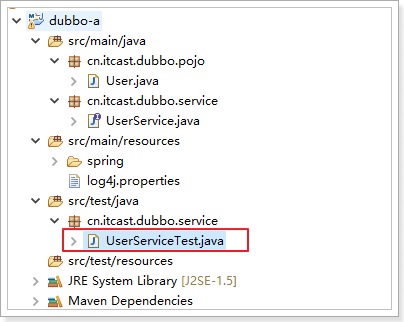
<!-- 从注册中心中查找服务 -->

<dubbo:reference id=*"userService"* interface=*"cn.itcast.dubbo.service.UserService"*/>

</beans>

### 编写UserService测试用例

使用Junit对UserService接口生成测试类：

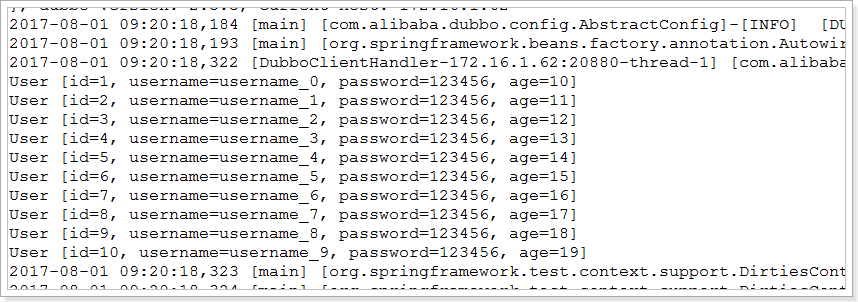


代码：



本项目中虽然没有UserService接口的实现类，但是我们只管注入，dubbo会结合spring实现注入，并且在不知不觉间实现远程调用，与调用本地方法看不出差别。

结果：



## 解决代码重复

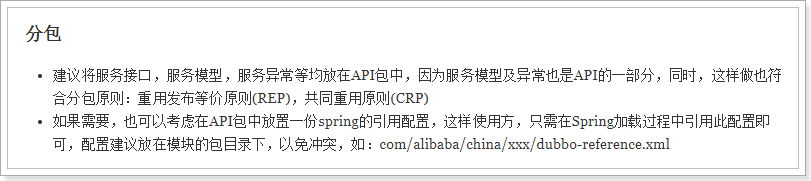
### 问题分析及最佳实践

通过刚刚的示例我们可以发现，其中User对象和UserService在A系统和B系统中都使用，那么，我们是否应该讲该代码复用呢？

答案是肯定的。

在使用dubbo时，provider需要将提供服务所需要的java代码（bean、interface等）单独打包成jar提供给consumer使用。

dubbo官方也是这样推荐的：



最佳实践方案：

以下内容需要抽取到单独的项目，并打包发布：

1. 服务接口
2. 接口中使用的模型类（JavaBean）
3. dubbo消费者的配置文件（可选。这个有配置文件名冲突风险，所以我们不实现这一点）

传统的WEB工程，如果使用Dubbo格式，一般应该提供服务的是除了WEB层以外的所有。

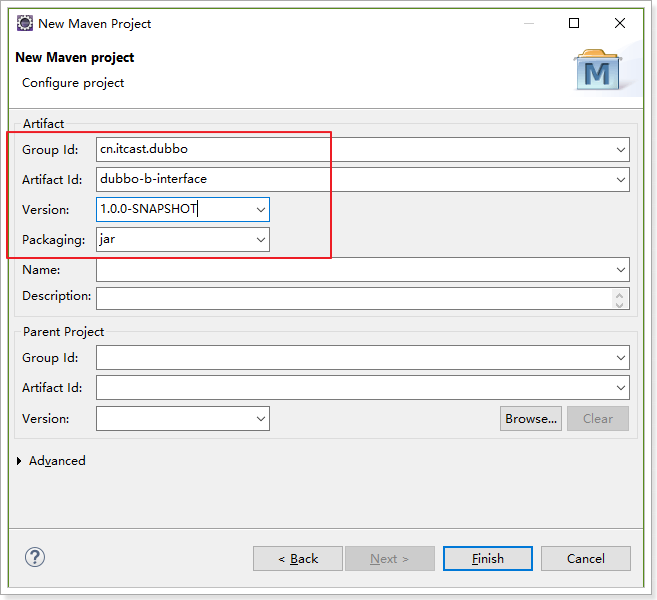
然后，我们一般创建成一个聚合工程，包含这几个自工程：

mapper：与数据库CRUD

service：业务层，提供服务。

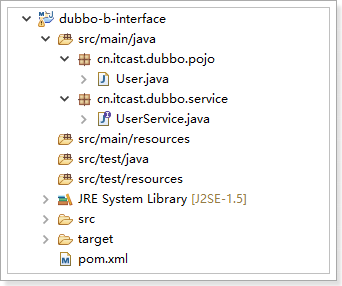
interface：里面包含服务的接口、实体类、自定义异常

### 创建dubbo-b-interface



### 复制dubbo-b中的服务模型和服务接口

我们把 服务模型：User，服务接口：UserService ，复制到新的项目中：



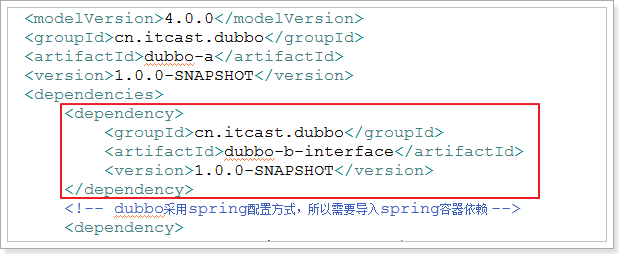
### 再次测试

我们把dubbo-b-interface 安装的maven仓库，然后在A系统和B系统中引用

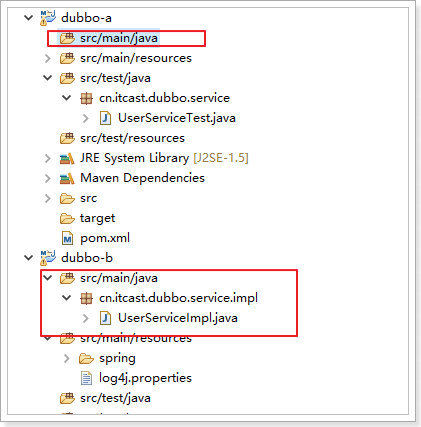
B系统中引用duubo-b-interface：



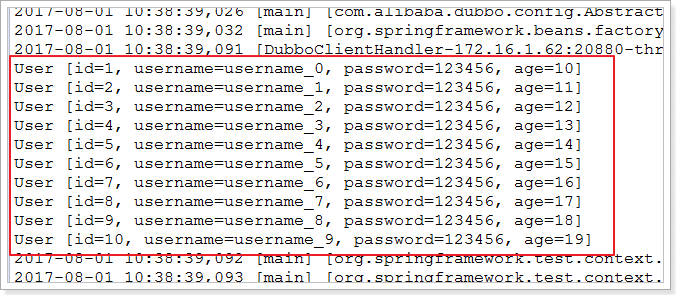
A 系统中引用duubo-b-interface：



AB系统中，删除重复的User和UserService

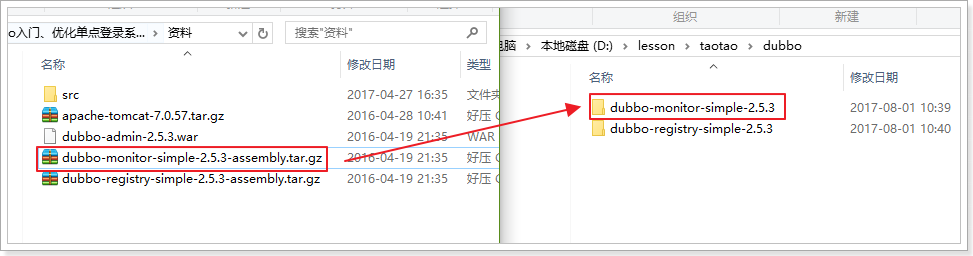


再次运行：



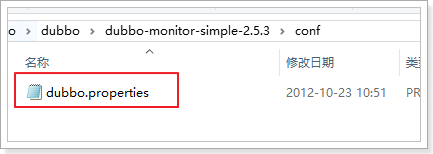
## 监控中心

### 解压监控服务文件

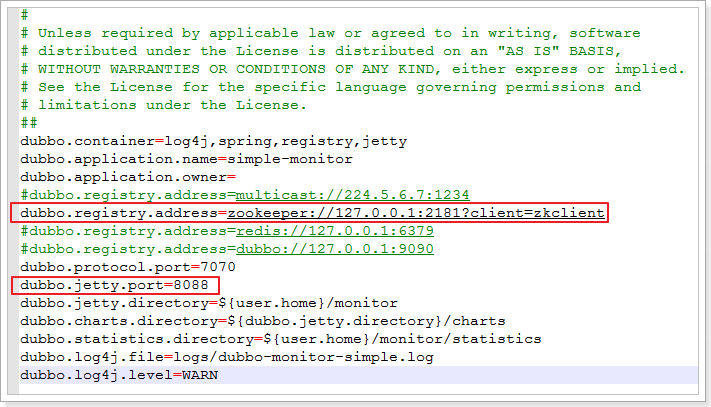


### 修改配置文件：

修改：dubbo-monitor-simple-2.5.3\conf\dubbo.properties文件



修改注册中心的地址：



监控系统启动后，会把自己也注册到注册中心，成为一个服务。所有的provider都可以看到并访问监控中心

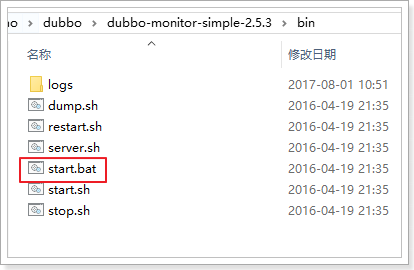
### 在dubbo-b中配置监控系统

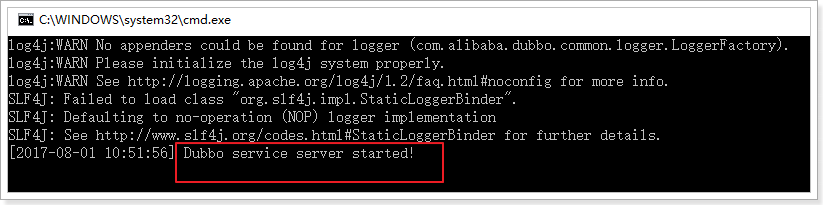
服务的提供者每隔1分钟都会向监控系统报告自己的运行情况，我们需要在provider（提供者）中配置监控：

<!-- 配置监控中心，不是直接写地址，而是去注册中心中寻找 -->

<dubbo:monitor protocol=*"registry"* />

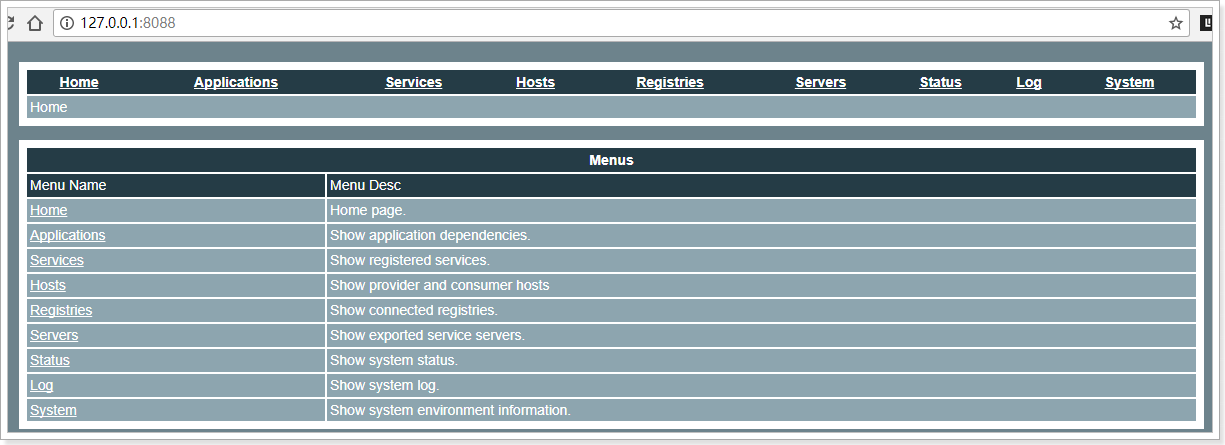
### 启动





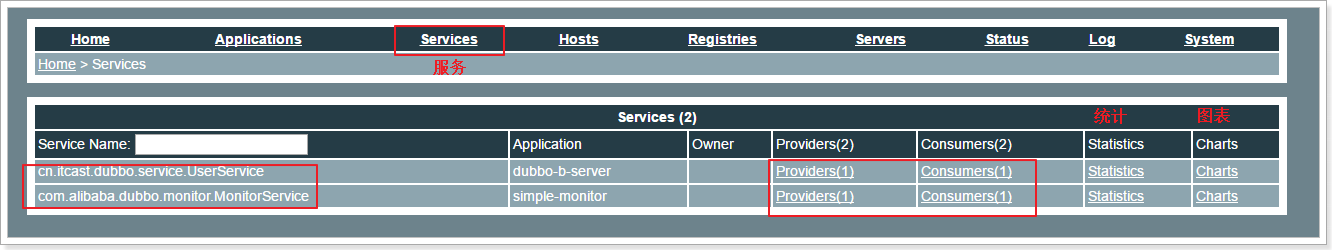
### 访问

打开浏览器，访问地址：127.0.0.1:8088

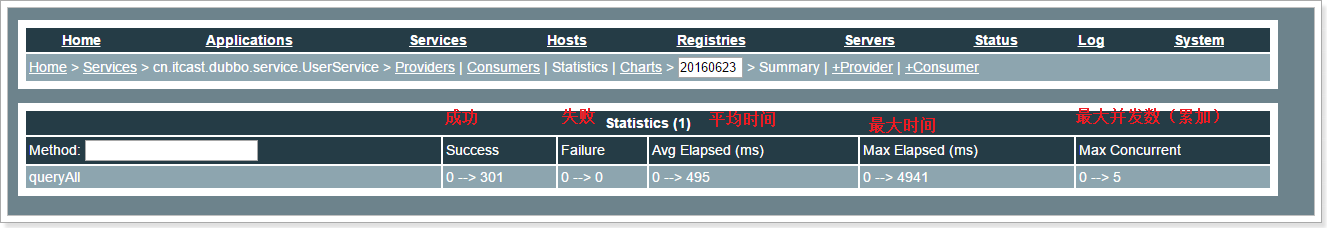


### 界面介绍

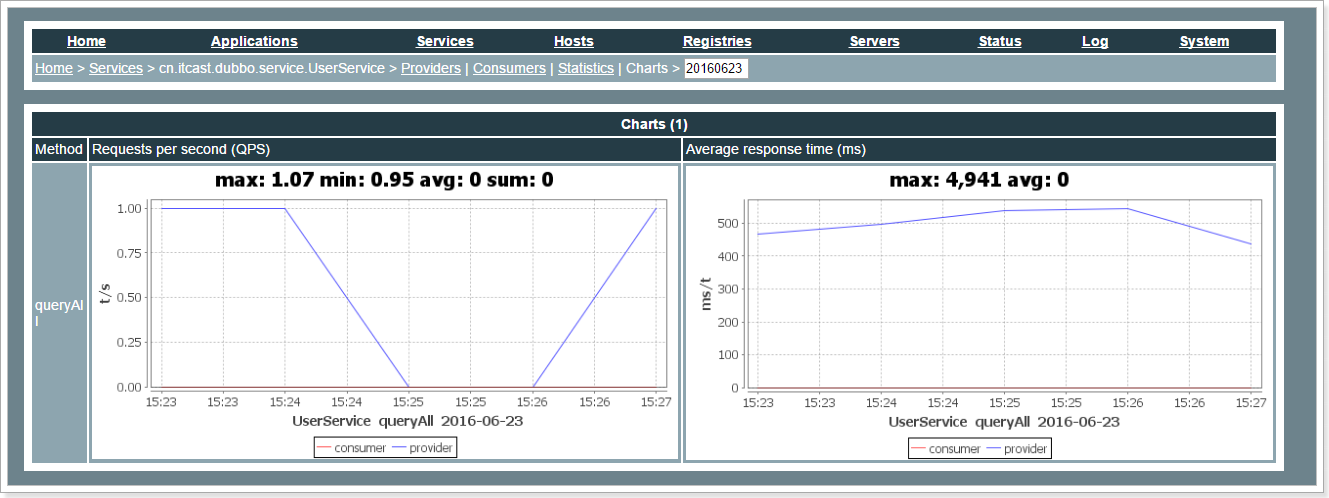
查看服务：



统计：



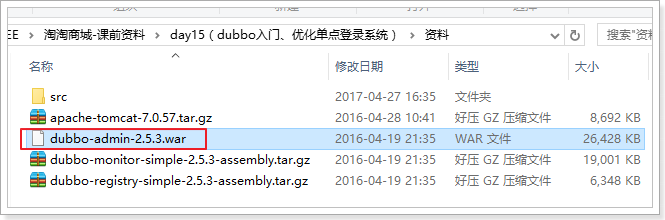
图表：



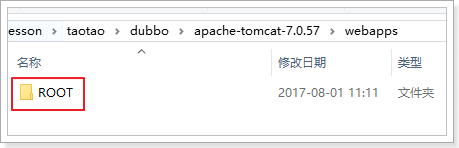
## 治理中心

### 部署

把课前资料中的war包部署到Tomcat中：

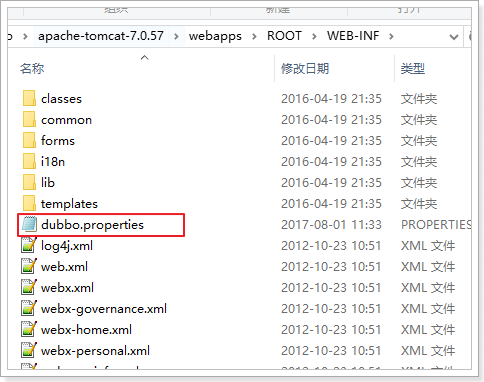


并且以ROOT目录部署：

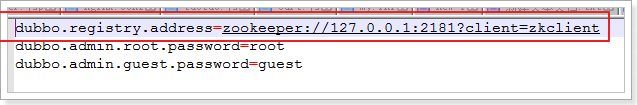


### 修改配置文件

到 ROOT/WEB-INF下，修改dubbo.properties:



修改其中的注册地址：

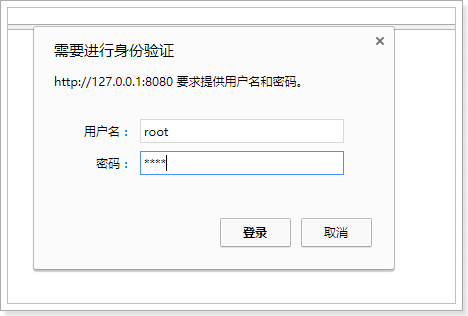


### 启动Tomcat并访问

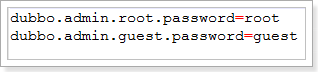
启动Tomcat，访问:

<http://127.0.0.1:8080>

效果：



账户和密码在刚才的配置文件中有写：



登录后：



功能：



# 使用Dubbo优化单点登录系统

## 分析

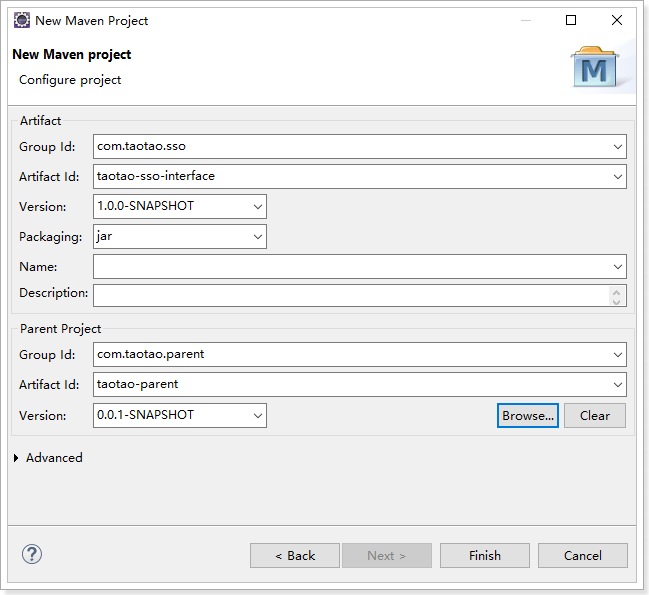
现在的单点登录系统是一个集中式结构，web表现层、业务层、数据层作为一个整体。我们现在要做的事情，是让业务层变为一个Dubbo服务。所以，我们需要对系统进行拆分：

拆分后，应该有3部分组成：

1. taotao-sso-interface：作为接口及模型的项目
   1. 里面包含POJO和Service接口
2. taotao-sso-service：包含业务层及DAO层，实现业务逻辑及数据查询。对外提供Dubbo的服务。是Provider
   1. 包含Service的具体实现及Mapper
3. taotao-sso-web：主要包含页面及Controller，是表现层。需要的业务去taotao-sso-service中获取。是Consumer
   1. 包含Controller层，及页面

## 创建taotao-sso-interface工程

### 创建工程



### 导入依赖

<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>

<parent>

<groupId>com.taotao.parent</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.1.3.Final</version>

</dependency>

<dependency>

<groupId>javax.persistence</groupId>

<artifactId>persistence-api</artifactId>

<version>1.0</version>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

</dependencies>

</project>

### 复制taotao-sso中的User 和 UserService

直接存原来的taotao-sso复制



但是把UserService改名字，去掉多余的接口方法：

/\*\*

\* 用户操作接口

\*

\* **@author** huge

\*/

**public** **interface** UserService {

/\*\*

\* 校验数据可用

\*/

**public** Boolean checkData(String param, Integer type);

/\*\*

\* 注册用户

\*/

**public** Boolean register(User user);

/\*\*

\* 登录

\*/

**public** String login(String username, String password) **throws** Exception;

/\*\*

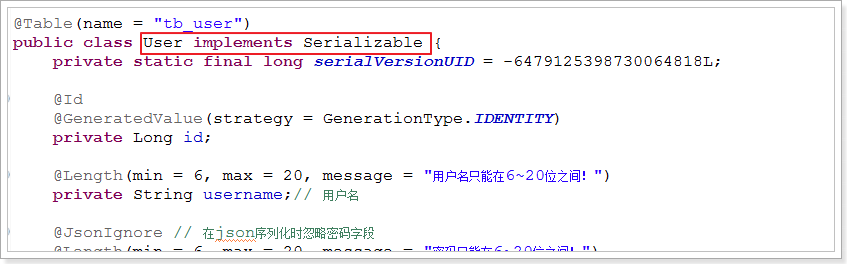
\* 根据token查询用户

\*/

**public** User queryUserByToken(String token);

}

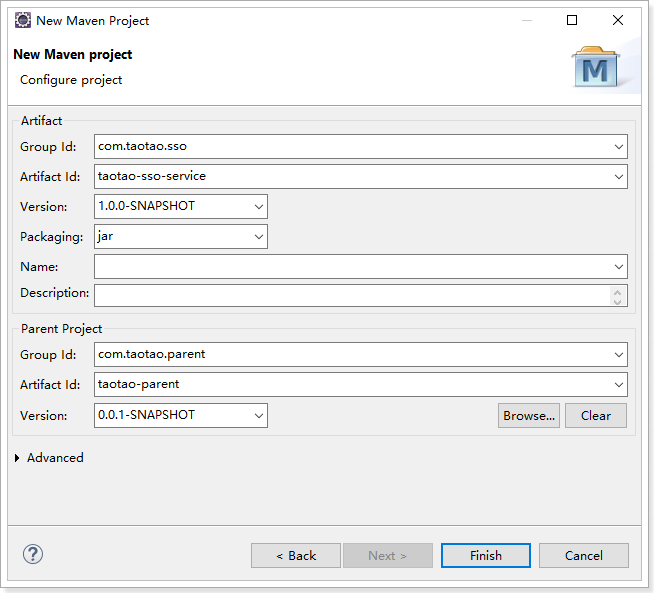
User类一定要实现Serializable接口：



**不要忘了Install**

## 创建taotao-sso-service提供服务

### 创建工程



### 导入依赖

<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>

<parent>

<groupId>com.taotao.parent</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-service</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>com.taotao.common</groupId>

<artifactId>taotao-common</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<!-- Mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

</dependency>

<!-- 通用Mapper -->

<dependency>

<groupId>com.github.abel533</groupId>

<artifactId>mapper</artifactId>

</dependency>

<!-- MySql -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- 连接池 -->

<dependency>

<groupId>com.jolbox</groupId>

<artifactId>bonecp-spring</artifactId>

</dependency>

<!-- Apache工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.6</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.3</version>

<exclusions>

<exclusion>

<!-- 排除传递spring依赖 -->

<artifactId>spring</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<!-- zookeeper依赖 -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.3.3</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

<dependency>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

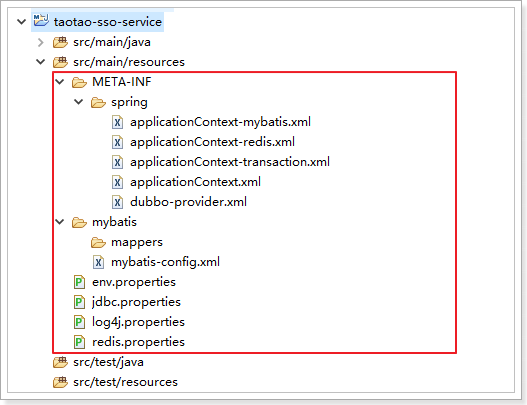
</project>

### 编写配置

#### 其它配置

我们从taotao-sso拷贝除了SpringMVC以外的所有配置：

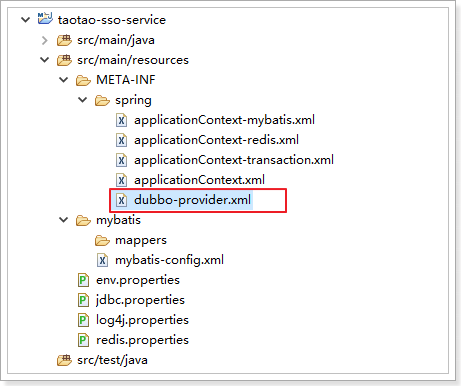
注意放到META-INF下：



具体内容 略。。

#### dubbo配置

从前面的itcast-dubbo-b复制其配置文件dubbo-provider.xml：



<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"taotao-sso-provider"* />

<!-- 这里使用的注册中心是zookeeper -->

<dubbo:registry address=*"zookeeper://127.0.0.1:2181"* client=*"zkclient"*/>

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"20881"* />

<!-- 将该接口暴露到dubbo中 -->

<dubbo:service interface=*"com.taotao.sso.service.UserService"* ref=*"ssoUserServiceImpl"* />

<!-- 将具体的实现类加入到Spring容器中 -->

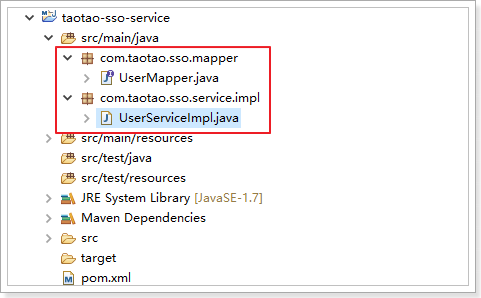
<bean id=*"ssoUserServiceImpl"* class=*"com.taotao.sso.service.impl.UserServiceImpl"* />

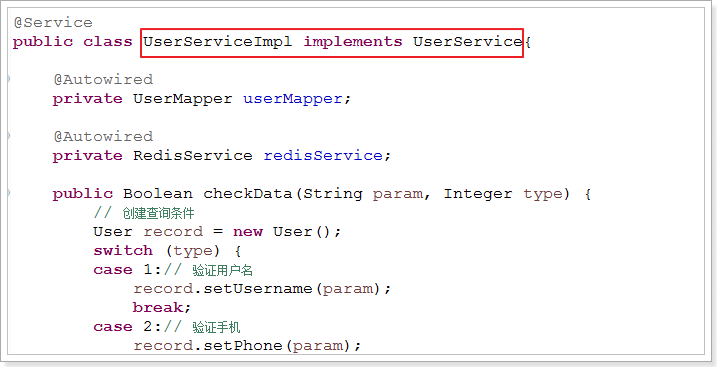
</beans>

### 编写UserQueryServiceImpl

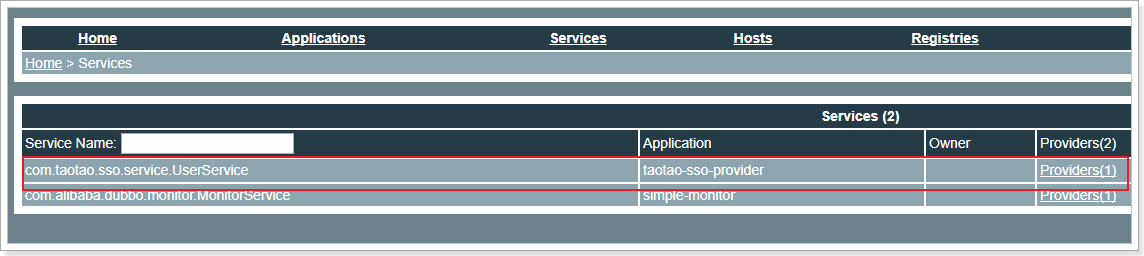
从taotao-sso复制UserService和UserMapper。

注意：一定要把UserService改成UserServiceImple，并且实现刚才定义的UserService接口：



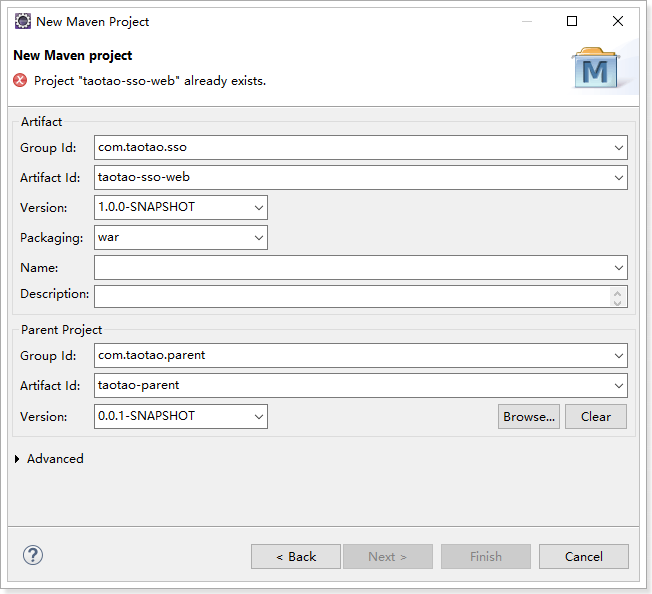


### 查看监控中心：



## 创建表现层taotao-sso-web工程

### 创建工程



### 导入依赖

注意，这里只需要导入SpringMVC相关依赖即可~因为这里只有WEB层的内容

<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>

<parent>

<groupId>com.taotao.parent</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-web</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>com.taotao.common</groupId>

<artifactId>taotao-common</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- JSP相关 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- Apache工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.6</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.3</version>

<exclusions>

<exclusion>

<!-- 排除传递spring依赖 -->

<artifactId>spring</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<!-- zookeeper依赖 -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.3.3</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

<dependency>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8083</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

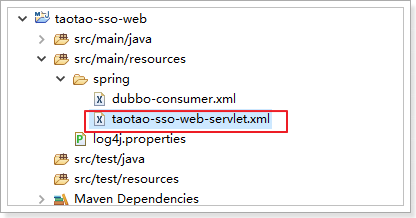
</build>

</project>

### 配置文件

#### springMVC配置

从原来的SSO复制，修改即可：



内容：

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

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-4.0.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd"*>

<!-- 扫描包 -->

<context:component-scan base-package=*"com.taotao.sso.web.controller"*/>

<!-- 配置内容协调管理器，解决406问题 -->

<bean id=*"contentNegotiationManager"* class=*"org.springframework.web.accept.ContentNegotiationManagerFactoryBean"*>

<property name=*"favorPathExtension"* value=*"false"* />

</bean>

<!-- 注解驱动 -->

<mvc:annotation-driven content-negotiation-manager=*"contentNegotiationManager"* />

<!-- 配置视图解析器 -->

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

<property name=*"prefix"* value=*"/WEB-INF/views/"*/>

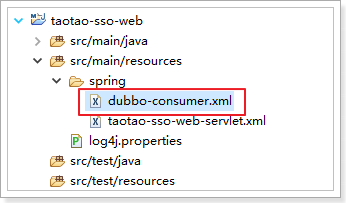
<property name=*"suffix"* value=*".jsp"*/>

</bean>

</beans>

#### dubbo配置

从测试案例的itcast-dubbo-a中拷贝：



内容：

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"*>

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"taotao-sso-consumer"* />

<!-- 这里使用的注册中心是zookeeper -->

<dubbo:registry address=*"zookeeper://127.0.0.1:2181"* client=*"zkclient"*/>

<!-- 从注册中心中查找服务 -->

<dubbo:reference id=*"userService"* interface=*"com.taotao.sso.service.UserService"*/>

</beans>

#### 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"*

id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>taotao-sso-web</display-name>

<context-param>

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

<param-value>classpath:spring/dubbo-consumer.xml</param-value>

</context-param>

<context-param>

<param-name>staticPath</param-name>

<param-value>http://static.taotao.com</param-value>

</context-param>

<!--Spring的ApplicationContext 载入 -->

<listener>

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

</listener>

<!-- 编码过滤器，以UTF8编码 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置SpringMVC框架入口 -->

<servlet>

<servlet-name>taotao-sso-web</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

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

<param-value>classpath:spring/taotao-sso-web-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>taotao-sso-web</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

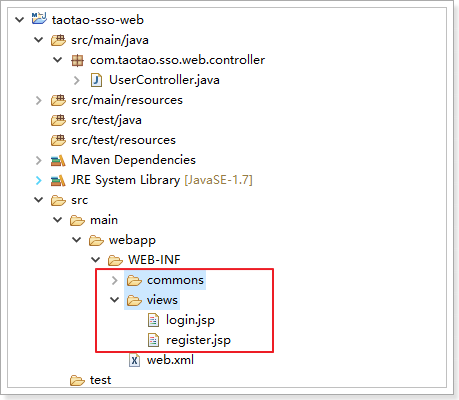
</web-app>

### 复制controller

从taotao-sso复制即可



### 复制静态资源：



### 测试并解决跨域

登录没有任何问题，但是在首页无法获取到登录状态，查看返回的结果：



发现没有实现跨域！！

回忆我们的跨域是如何实现的？

是有一个通用的ResponseAdvice，放在taotao-common中。以前可以自动扫描，因为我们配置了扫描包，扫描com.taotao包

但是在这里，我们并没有配置spring，所以无法扫描到。怎么办？

其实dubbo配置文件，就是spring的配置文件，我们在里面将那个advice添加即可：



重启后，再次测试：OK~



# 改造前台系统

## 引入依赖

我们需要引入 taotao-sso-interface和Dubbo的一系列依赖

<dependency>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- Dubbo -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.3</version>

<exclusions>

<exclusion>

<!-- 排除传递spring依赖 -->

<artifactId>spring</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<!-- zookeeper依赖 -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.3.3</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

## 修改配置

### dubbo配置

从taotao-sso-web复制dubbo配置并修改：

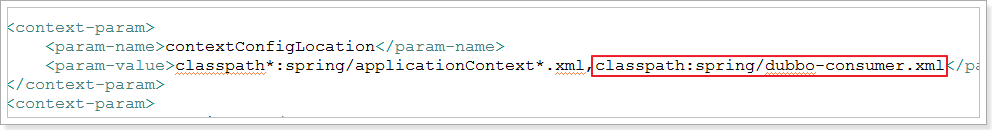
特别需要注意的是：

1. 应用的名称；



### web.xml配置

我们需要修改web.xml配置，读取dubbo配置文件

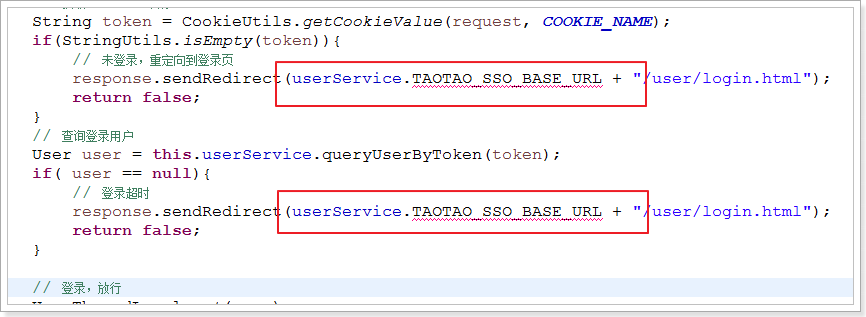


## 修改代码

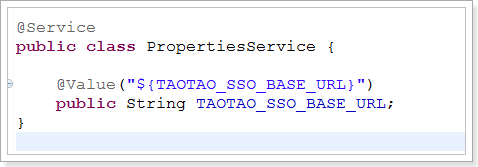
首先删除taotao-cart中的User对象和UserService对象。

然后把需要的地方都修改为taotao-sso-interface中的类。

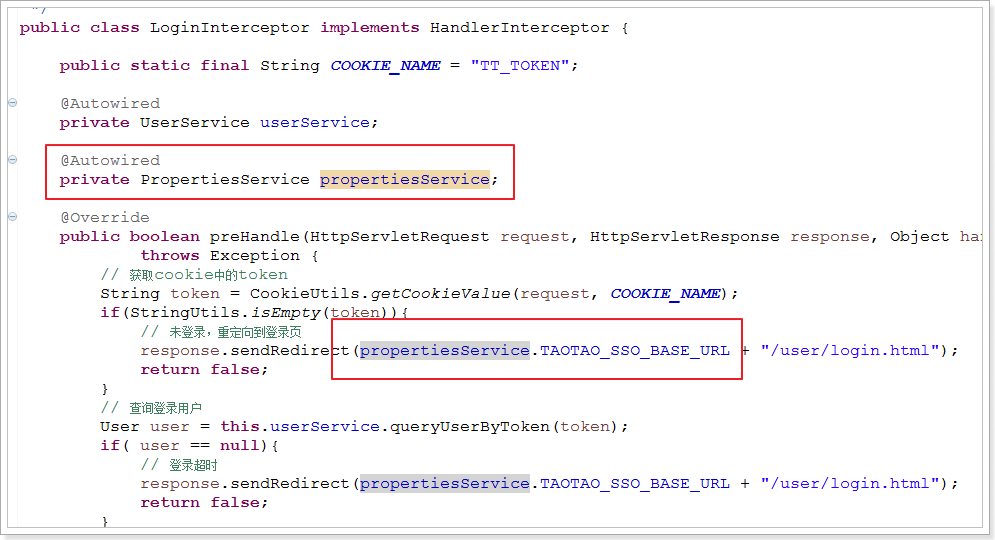
此时还有一个问题：在LoginInterceptor中，引用了UserService中的地址：TAOTAO\_SSO\_BASE\_URL，现在已经没有了。



我们需要新建一个PropertieService



然后在LoginInterceptor中使用：



## 测试



# 改造购物车系统

## 引入依赖

我们需要引入 taotao-sso-interface和Dubbo的一系列依赖

<dependency>

<groupId>com.taotao.sso</groupId>

<artifactId>taotao-sso-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- Dubbo -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<version>2.5.3</version>

<exclusions>

<exclusion>

<!-- 排除传递spring依赖 -->

<artifactId>spring</artifactId>

<groupId>org.springframework</groupId>

</exclusion>

</exclusions>

</dependency>

<!-- zookeeper依赖 -->

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.3.3</version>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

<version>0.1</version>

</dependency>

## 修改配置

### dubbo配置

从taotao-sso-web复制dubbo配置并修改：

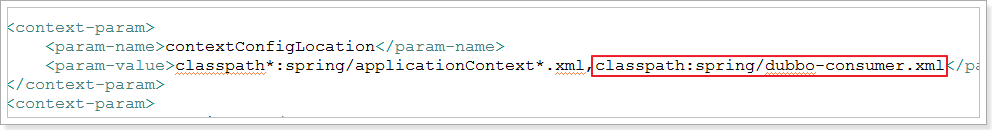
特别需要注意的是：

1. 应用的名称；



### web.xml配置

我们需要修改web.xml配置，读取dubbo配置文件



## 修改代码

首先删除taotao-cart中的User对象和UserService对象。

然后把需要的地方都修改为taotao-sso-interface中的类

## 测试：