# 什么是dubbo？

## 简介

**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。

<https://blog.csdn.net/wrs120/article/details/79980434> 对比

Dubbo是Alibaba开源的分布式服务框架，它最大的特点是按照分层的方式来架构，使用这种方式可以使各个层之间解耦合（或者最大限度地松耦合）。从服务模型的角度来看，Dubbo采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方（Provider）和服务消费方（Consumer）两个角色。关于注册中心、协议支持、服务监控等内容。

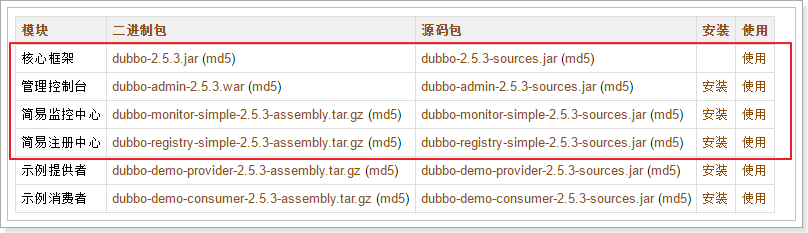
## RPC



本地就是进行接口调用，但是 dubbo 会代理这个调用请求，跟远程机器网络通信，给你处理掉负载均衡、服务实例上下线自动感知、超时重试等等乱七八糟的问题。

# 通过Maven构建dubbo（非必须）

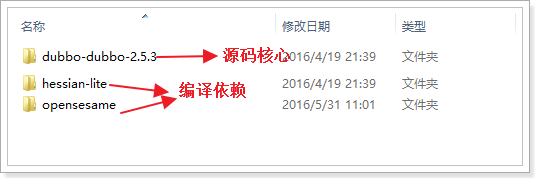
既然可以下载得到源码以及发布包，那么为什么要去构建dubbo呢？，我们先来看下dubbo的主要模块：



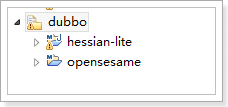
其中，核心框架、管理控制台、简易监控中心、简易注册中心是我们需要的模块，目前，只有核心模块可以下载到，其它的均无法直接下载，所以我们需要构建dubbo。

当然了，在提供的资料中以及有构建好的模块提供，也可以选择不重新构建。

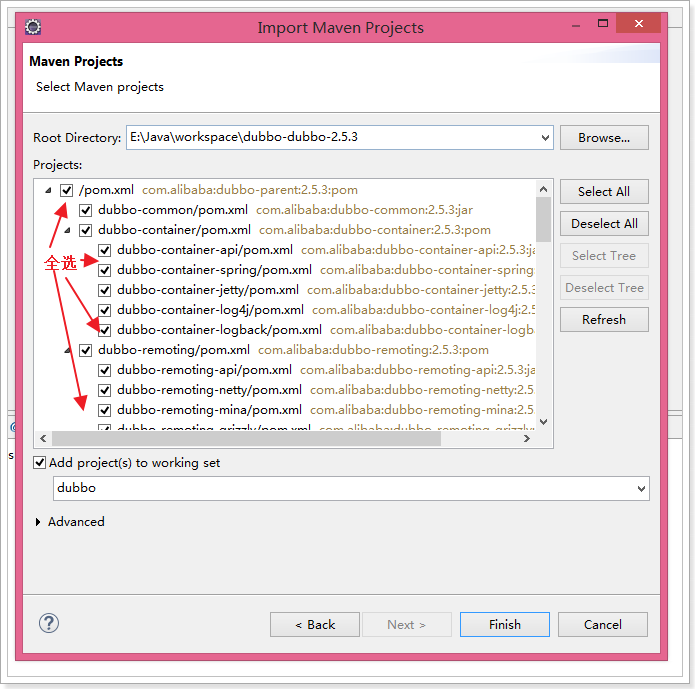
## 导入源码到Eclipse



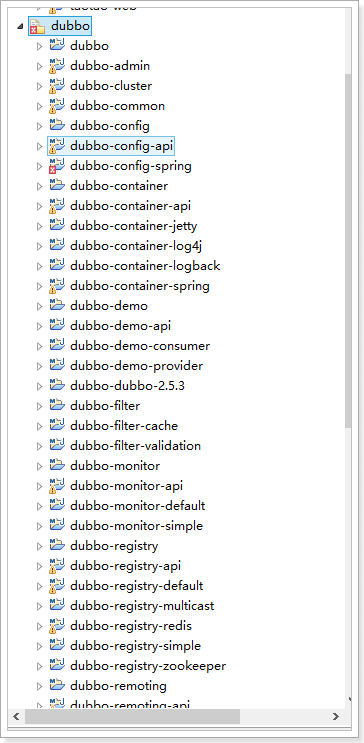
先导入编译依赖到Eclipse：



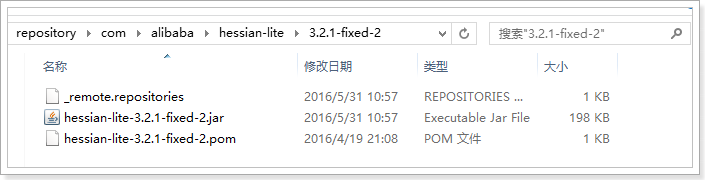
再导入dubbo源码到Eclipse：



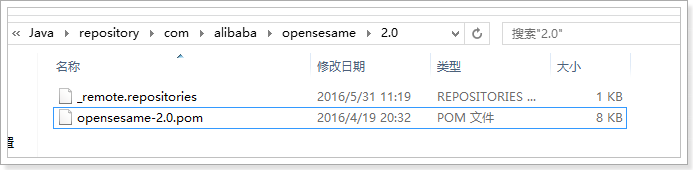
导入完成：



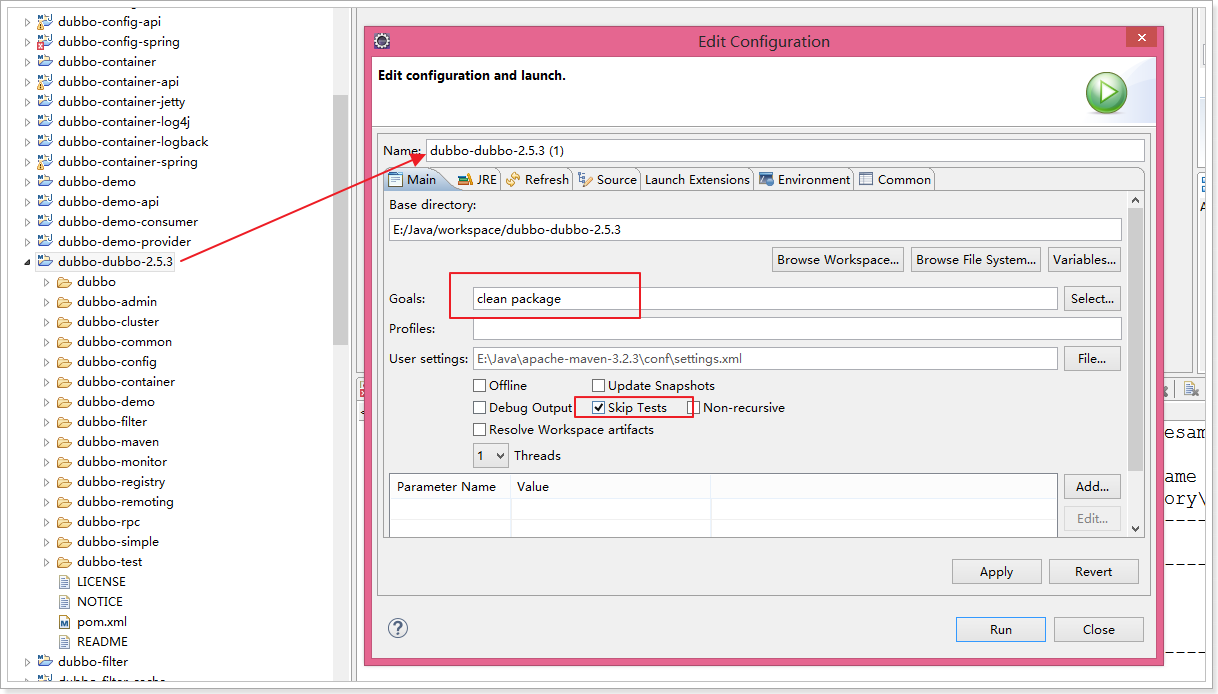
## 安装hessian-lite到本地仓库



## 安装opensesame到本地仓库

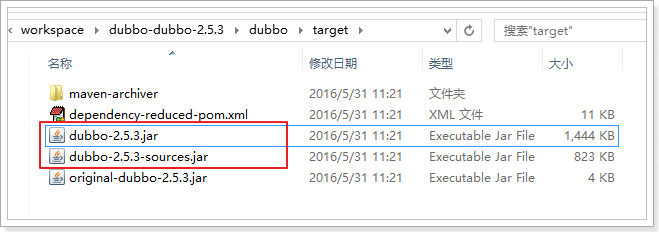


## 构建dubbo

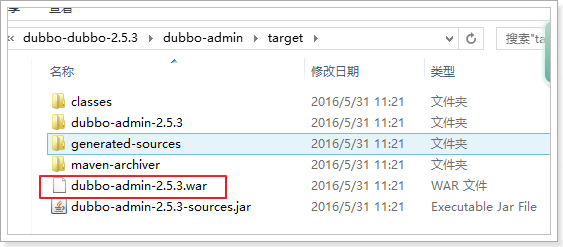


## 找到对应的模块包

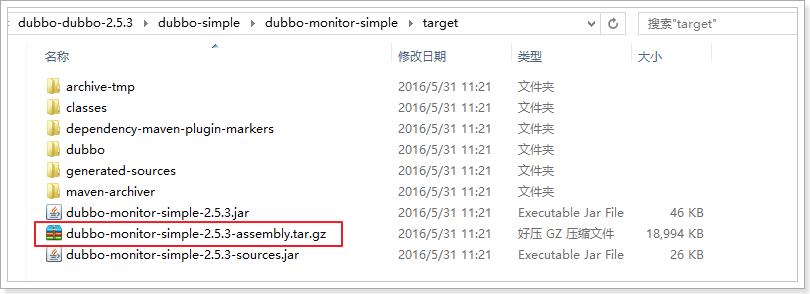
### 核心框架



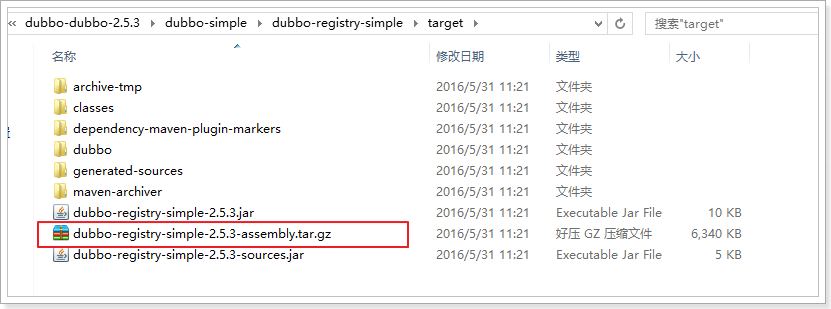
### 管理控制台



### 简易监控中心



### 简易注册中心



# Dubbo框架说明

## 原理

### dubbo 工作原理

* 第一层：service 层，接口层，给服务提供者和消费者来实现的
* 第二层：config 层，配置层，主要是对 dubbo 进行各种配置的
* 第三层：proxy 层，服务代理层，无论是 consumer 还是 provider，dubbo 都会给你生成代理，代理之间进行网络通信
* 第四层：registry 层，服务注册层，负责服务的注册与发现
* 第五层：cluster 层，集群层，封装多个服务提供者的路由以及负载均衡，将多个实例组合成一个服务
* 第六层：monitor 层，监控层，对 rpc 接口的调用次数和调用时间进行监控
* 第七层：protocal 层，远程调用层，封装 rpc 调用
* 第八层：exchange 层，信息交换层，封装请求响应模式，同步转异步
* 第九层：transport 层，网络传输层，抽象 mina 和 netty 为统一接口
* 第十层：serialize 层，数据序列化层

### dubbo 协议

**默认**就是走 dubbo 协议，单一长连接，进行的是 NIO 异步通信，基于 hessian 作为序列化协议。使用的场景是：传输数据量小（每次请求在 100kb 以内），但是并发量很高。

* rmi 协议

走 Java 二进制序列化，多个短连接，适合消费者和提供者数量差不多的情况，适用于文件的传输，一般较少用。

* hessian 协议

走 hessian 序列化协议，多个短连接，适用于提供者数量比消费者数量还多的情况，适用于文件的传输，一般较少用。

* http 协议

走 json 序列化。

* webservice

走 SOAP 文本序列化。

### dubbo 负载均衡策略

1. **random loadbalance**

默认情况下，dubbo 是 random load balance ，即**随机**调用实现负载均衡，可以对 provider 不同实例**设置不同的权重**，会按照权重来负载均衡，权重越大分配流量越高，一般就用这个默认的就可以了。

1. **roundrobin loadbalance**

这个的话默认就是均匀地将流量打到各个机器上去，但是如果各个机器的性能不一样，容易导致性能差的机器负载过高。所以此时需要调整权重，让性能差的机器承载权重小一些，流量少一些。

举个栗子。

1. **leastactive loadbalance**

这个就是自动感知一下，如果某个机器性能越差，那么接收的请求越少，越不活跃，此时就会给**不活跃的性能差的机器更少的请求**。

1. **consistanthash loadbalance**

一致性 Hash 算法，相同参数的请求一定分发到一个 provider 上去，provider 挂掉的时候，会基于虚拟节点均匀分配剩余的流量，抖动不会太大。**如果你需要的不是随机负载均衡**，是要一类请求都到一个节点，那就走这个一致性 Hash 策略。

### dubbo 集群容错策略

**failover cluster 模式**

失败自动切换，自动重试其他机器，**默认**就是这个，常见于读操作。（失败重试其它机器）

可以通过以下几种方式配置重试次数：

<dubbo:service retries="2" />

或者

<dubbo:reference retries="2" />

或者

<dubbo:reference>

<dubbo:method name="findFoo" retries="2" />

</dubbo:reference>

**failfast cluster 模式**

一次调用失败就立即失败，常见于非幂等性的写操作，比如新增一条记录（调用失败就立即失败）

**failsafe cluster 模式**

出现异常时忽略掉，常用于不重要的接口调用，比如记录日志。

配置示例如下：

<dubbo:service cluster="failsafe" />

或者

<dubbo:reference cluster="failsafe" />

**failback cluster 模式**

失败了后台自动记录请求，然后定时重发，比较适合于写消息队列这种。

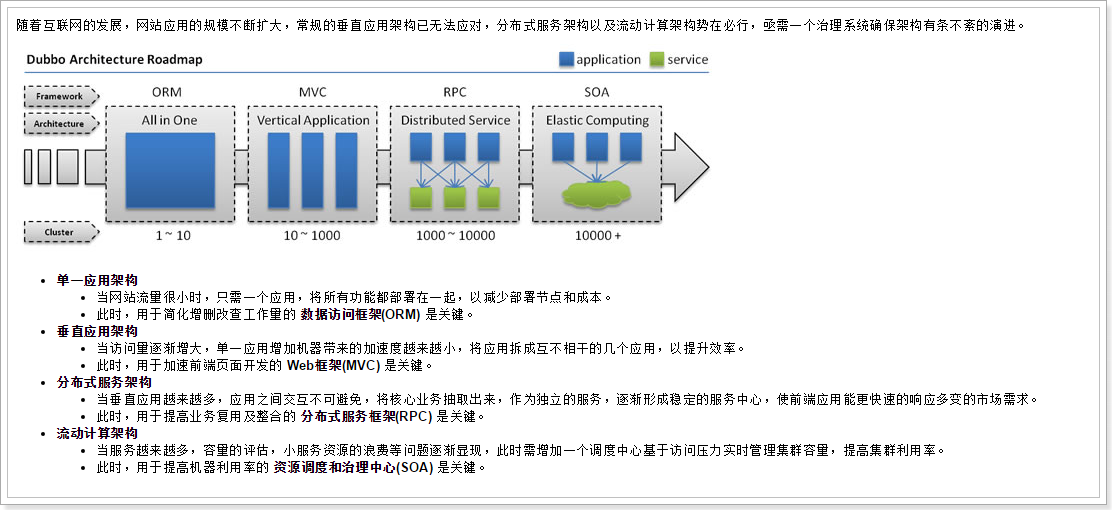
**forking cluster 模式**

**并行调用**多个 provider，只要一个成功就立即返回。常用于实时性要求比较高的读操作，但是会浪费更多的服务资源，可通过 forks="2" 来设置最大并行数。

**broadcacst cluster**

逐个调用所有的 provider。任何一个 provider 出错则报错（从2.1.0 版本开始支持）。通常用于通知所有提供者更新缓存或日志等本地资源信息。

## 背景

**调用关系：**

1. 服务容器负责启动，加载，运行服务提供者。

2. 服务提供者在启动时，向注册中心注册自己提供的服务。

3. 服务消费者在启动时，向注册中心订阅自己所需的服务。

4. 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。

5. 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。

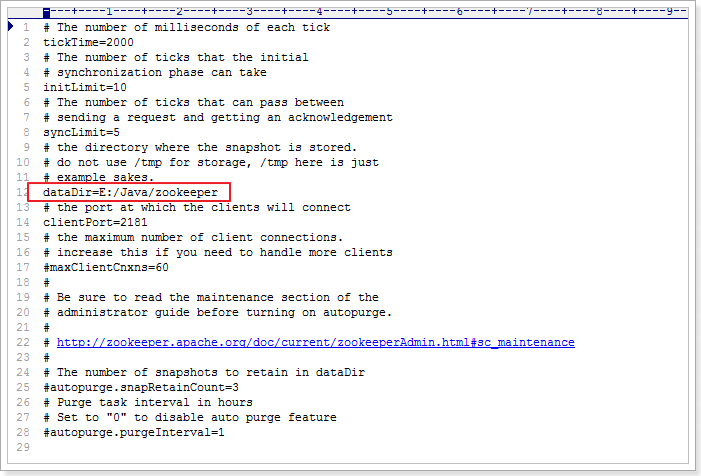
6. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

# 快速入门

## 安装zookeeper

解压得到：

修改配置文件zoo.cfg： 该目录必须存在。



启动zookeeper服务：

## 搭建B系统

### 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>dubbo-b</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</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>

</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>

### 创建User对象

**package** cn.itcast.dubbo.pojo;

// 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;

}

}

### 创建UserService（接口）提供查询服务

**package** cn.itcast.dubbo.service;

**import** java.util.List;

**import** cn.itcast.dubbo.pojo.User;

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

/\*\*

\* 查询所有的用户数据

\*

\* **@return**

\*/

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

}

### 创建UserServiceImpl实现类

**package** cn.itcast.dubbo.service.impl;

**import** java.util.ArrayList;

**import** java.util.List;

**import** cn.itcast.dubbo.pojo.User;

**import** cn.itcast.dubbo.service.UserService;

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

/\*\*

\* 实现查询，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);

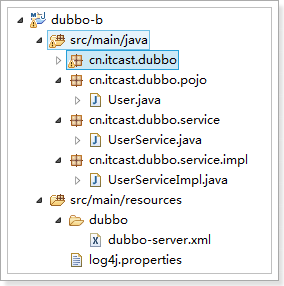
}

**return** list;

}

}

### 编写dubbo配置文件



具体配置：

<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-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>

### 导入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>

### 编写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>dubbo-b</display-name>

<context-param>

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

<param-value>classpath:dubbo/dubbo-\*.xml</param-value>

</context-param>

<!--Spring的ApplicationContext 载入 -->

<listener>

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

</listener>

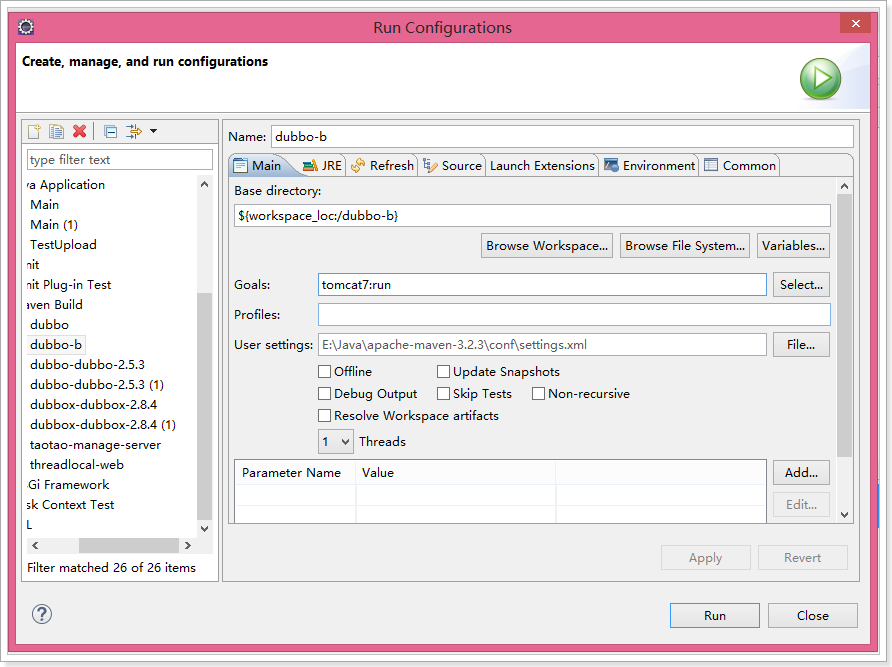
<welcome-file-list>

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

</welcome-file-list>

</web-app>

### 启动tomcat



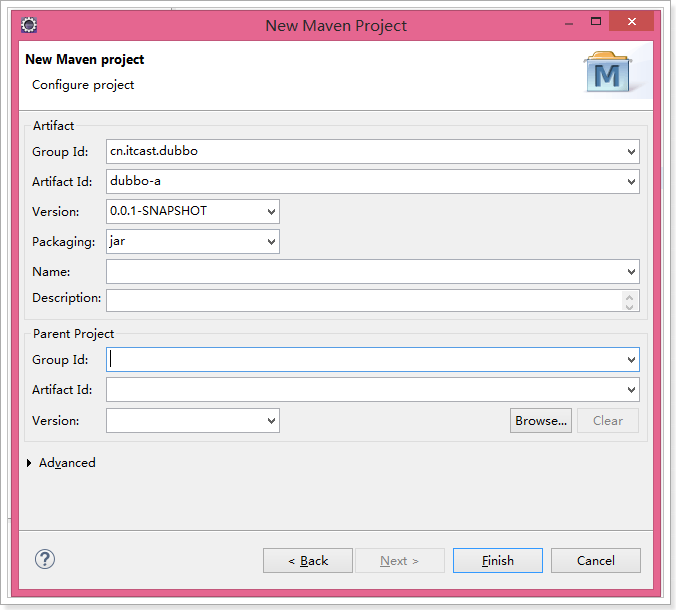
查看tomcat启动日志：

2016-06-03 09:38:35,307 [localhost-startStop-1] [com.alibaba.dubbo.registry.zookeeper.ZookeeperRegistry]-[INFO] [DUBBO] Register: dubbo://192.168.56.1:20880/cn.itcast.dubbo.service.UserService?anyhost=true&application=dubbo-b-server&dubbo=2.5.3&interface=cn.itcast.dubbo.service.UserService&methods=queryAll&pid=5436&side=provider&timestamp=1464917914608, dubbo version: 2.5.3, current host: 127.0.0.1

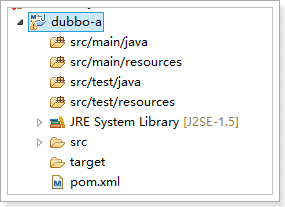
可以看到，已经将UserService服务注册到zookeeper注册中心，协议采用的是dubbo。

## 搭建A系统

### 创建工程



创建完成：



### 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>dubbo-a</artifactId>

<version>0.0.1-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>

<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>

<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>

</project>

### 配置log4j文件

log4j.properties：

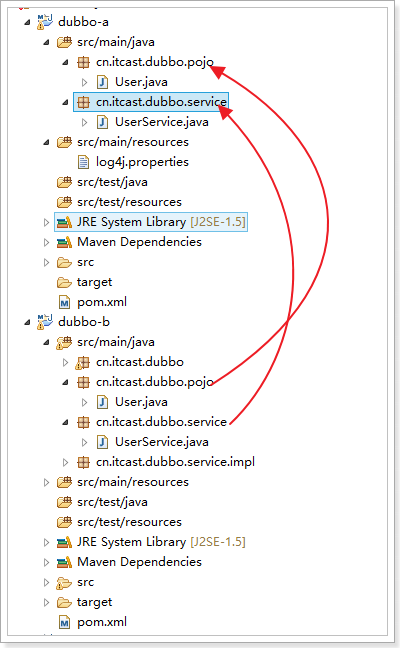
log4j.rootLogger=DEBUG,A1

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

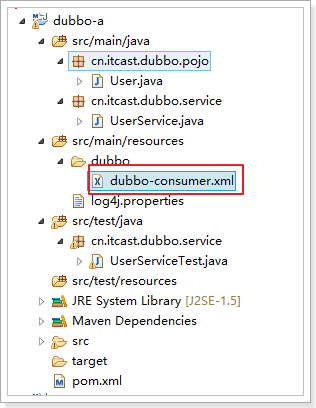
log4j.appender.A1.layout=org.apache.log4j.PatternLayout

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

### 从b系统中拷贝User对象、UserService接口道a系统



### 编写dubbo配置文件



dubbo-consumer.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=*"dubbo-a-consumer"* />

<!-- 这里使用的注册中心是zookeeper -->

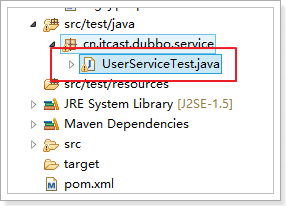
<dubbo:registry address=*"zookeeper://127.0.0.1:2181"* client=*"zkclient"*/>

<!-- 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



**package** cn.itcast.dubbo.service;

**import** **static** org.junit.Assert.\*;

**import** java.util.List;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** cn.itcast.dubbo.pojo.User;

**public** **class** UserServiceTest {

**private** UserService userService;

@Before

**public** **void** setUp() **throws** Exception {

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext(

"classpath:dubbo/\*.xml");

**this**.userService = applicationContext.getBean(UserService.**class**);

}

@Test

**public** **void** testQueryAll() {

List<User> users = **this**.userService.queryAll();

**for** (User user : users) {

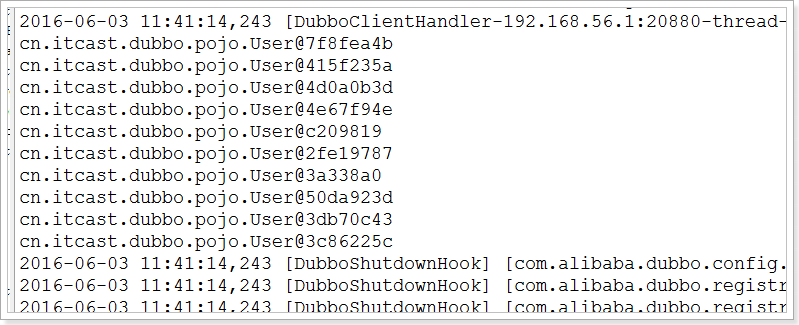
System.***out***.println(user);

}

}

}

### 查看效果



可以看到，已经查询到10条数据，那么，也就是说A系统通过B系统提供的服务获取到了数据。

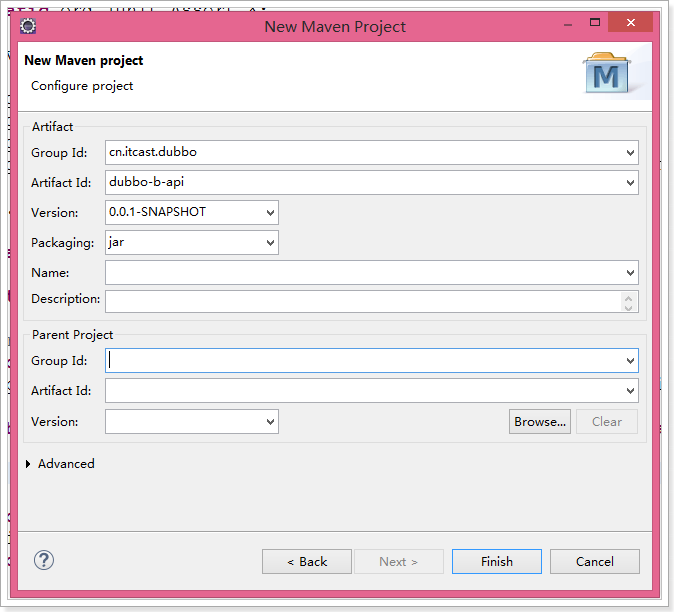
## 解决代码重复问题

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？

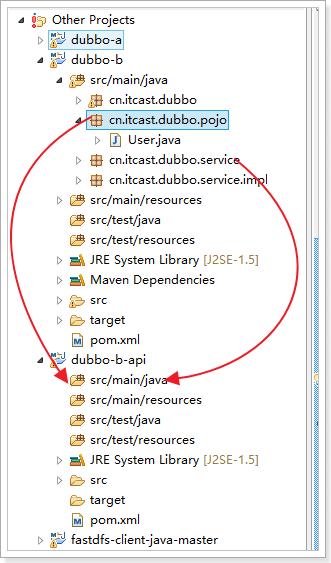
答案是肯定的。

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。

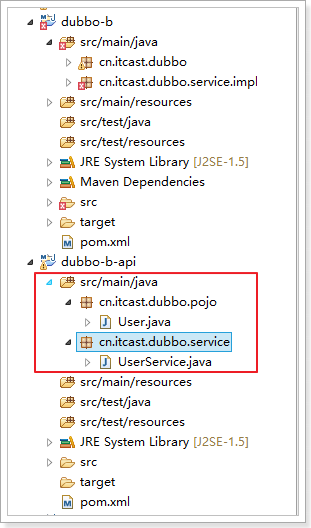
### 创建dubbo-b-api



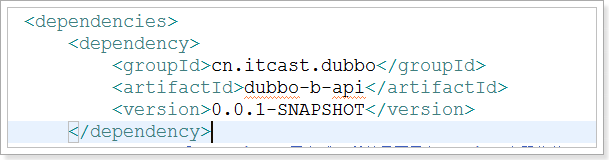
### 将dubbo-b中的pojo和service接口移动到dubbo-b-api中



结果如下：



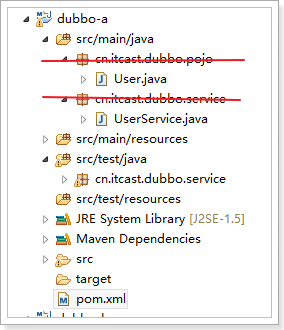
### 在dubbo-b的pom.xml文件中添加dubbo-b-api的依赖



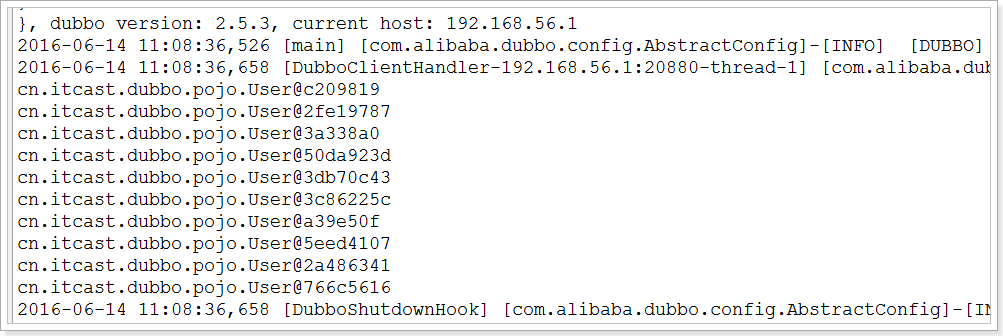
### 在dubbo-a的pom.xml文件中添加dubbo-b-api的依赖，并且将pojo与service接口删除



删除代码：



### 测试

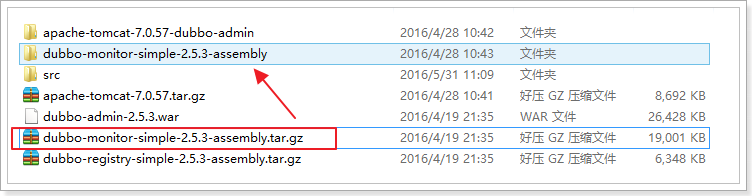


测试，发现，和之前实现一样。

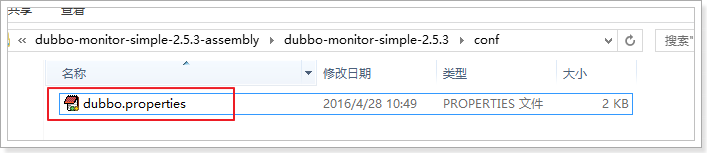
# 监控

原理：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。

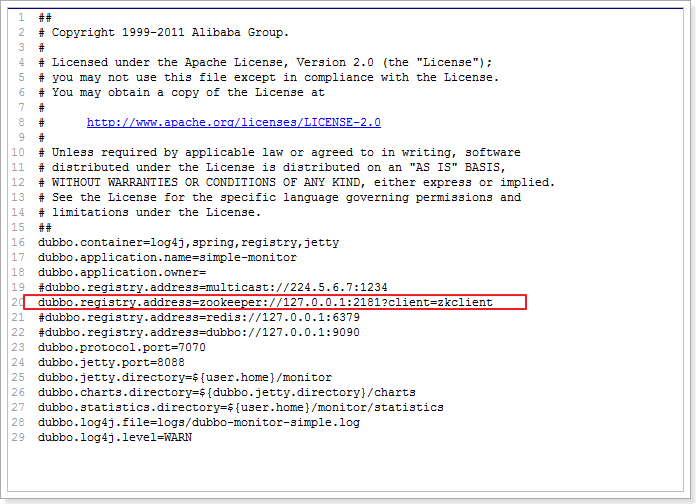
## 搭建监控服务



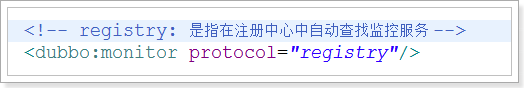
## 修改配置文件



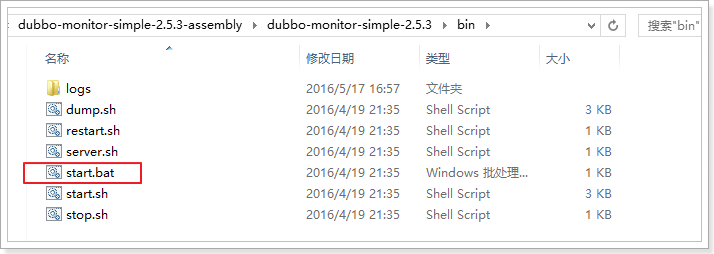
修改注册中心的地址：



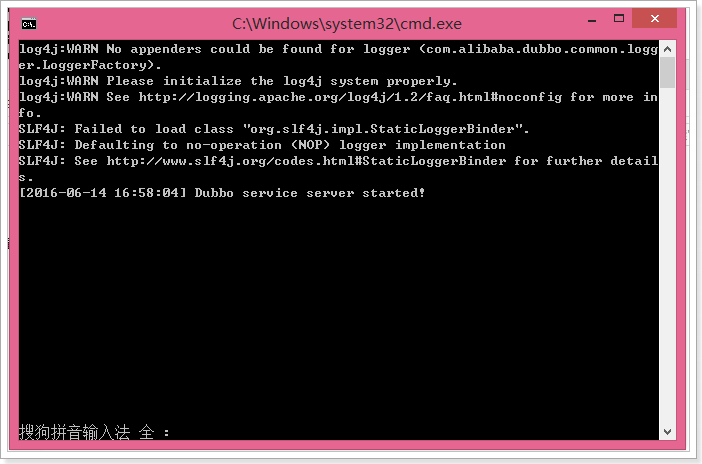
## 在dubbo-b中配置监控



## 启动

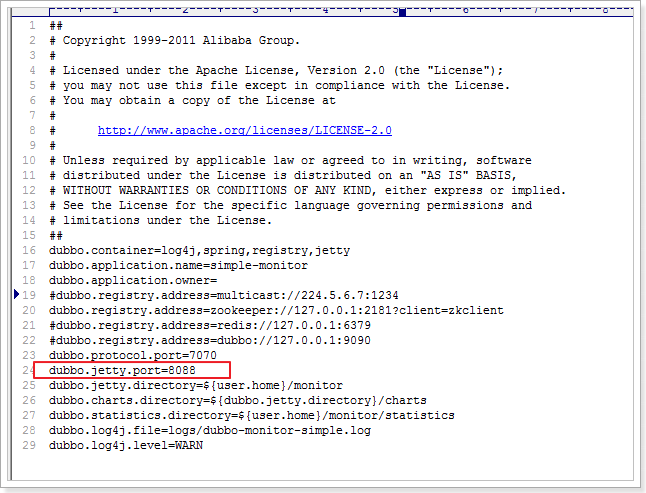


启动成功：



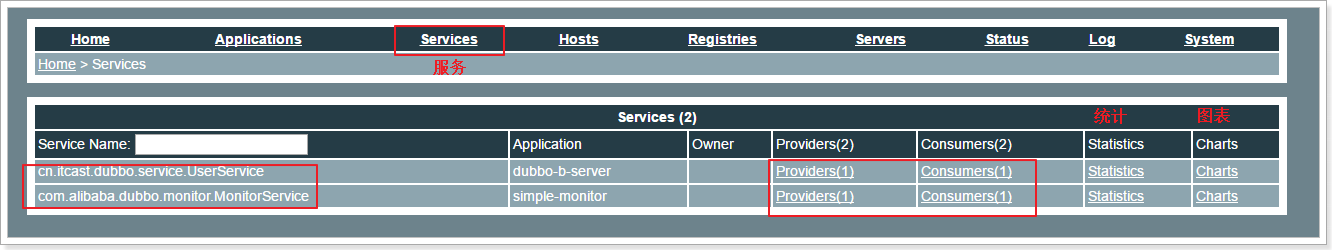
## 查看界面

输入地址：<http://127.0.0.1:8088/>

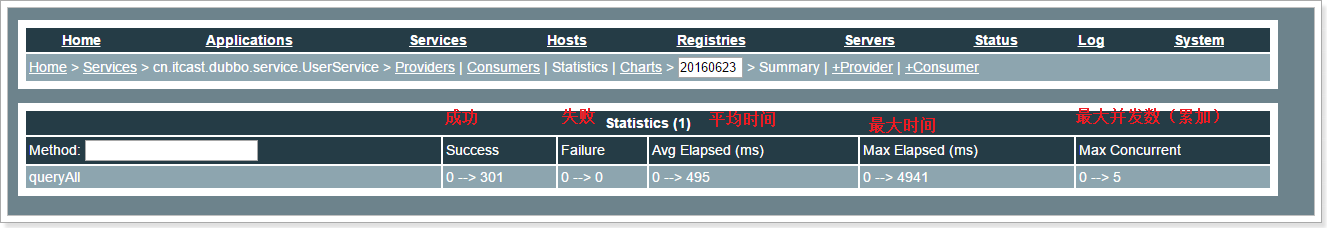


## 功能

查看服务：



统计：

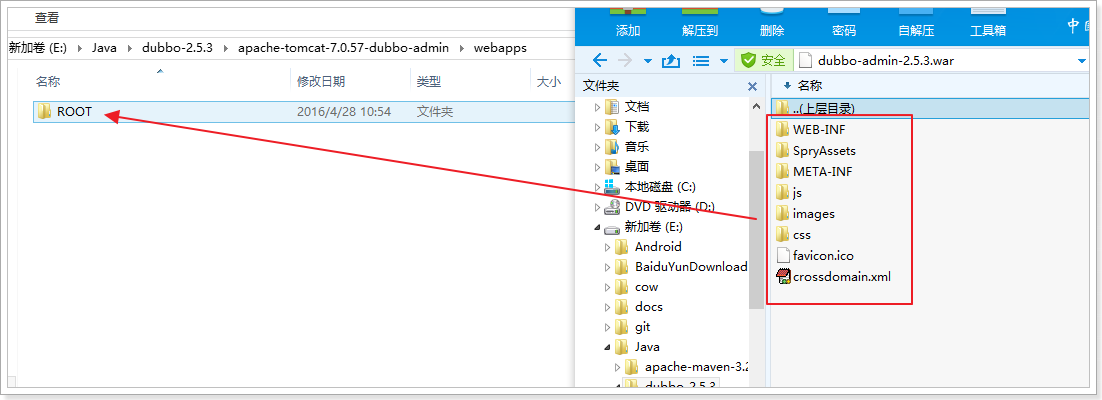


# 管理

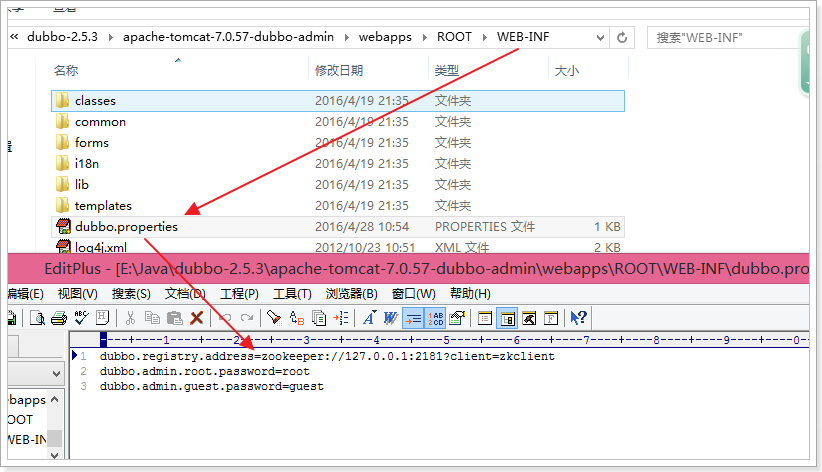
dubbo提供了一套在线管理整个服务的功能，管理控制台为阿里内部裁剪版本，开源部分主要包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。

## 安装

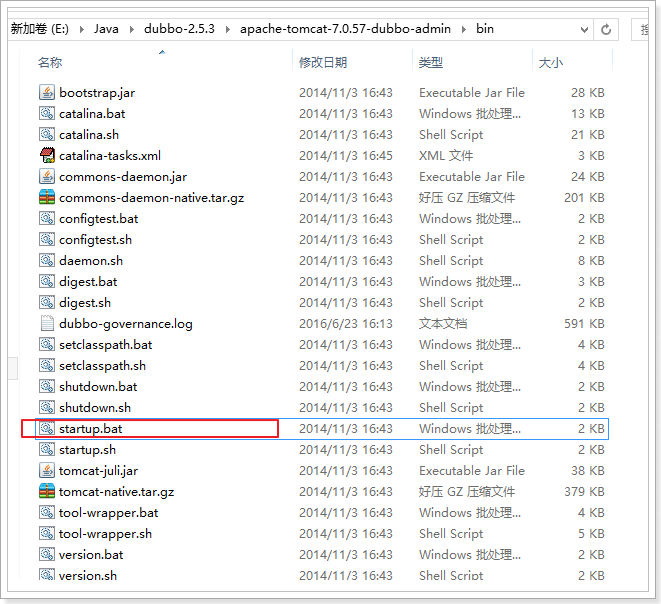
将dubbo-admin-2.5.3.war部署到tomcat的webapps目录下：

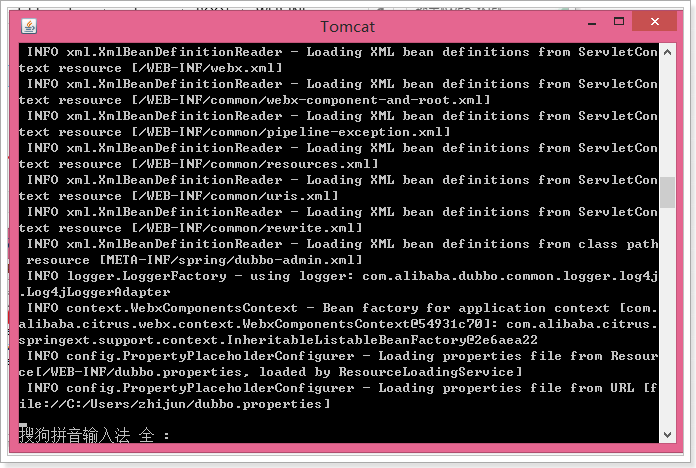


## 修改配置文件



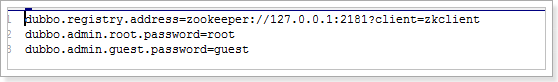
## 启动tomcat





## 查看管理界面

使用用户登录：



<http://127.0.0.1:8080/>

# dubbo的配置详解

## <dubbo:protocol/>

服务提供者协议配置：  
配置类：com.alibaba.dubbo.config.ProtocolConfig  
说明：如果需要支持多协议，可以声明多个<dubbo:protocol>标签，并在<dubbo:service>中通过protocol属性指定使用的协议。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **标签** | **属性** | **对应URL参数** | **类型** | **是否必填** | **缺省值** | **作用** | **描述** | **兼容性** |
| <dubbo:protocol> | id |  | string | 可选 | dubbo | 配置关联 | 协议BeanId，可以在<dubbo:service protocol="">中引用此ID，如果ID不填，缺省和name属性值一样，重复则在name后加序号。 | 2.0.5以上版本 |
| <dubbo:protocol> | name | <protocol> | string | **必填** | dubbo | 性能调优 | 协议名称 | 2.0.5以上版本 |
| <dubbo:protocol> | port | <port> | int | 可选 | dubbo协议缺省端口为20880，rmi协议缺省端口为1099，http和hessian协议缺省端口为80  如果配置为**-1** 或者 **没有**配置port，则会分配一个没有被占用的端口。Dubbo 2.4.0+，分配的端口在协议缺省端口的基础上增长，确保端口段可控。 | 服务发现 | 服务端口 | 2.0.5以上版本 |
| <dubbo:protocol> | host | <host> | string | 可选 | 自动查找本机IP | 服务发现 | -服务主机名，多网卡选择或指定VIP及域名时使用，为空则自动查找本机IP，-建议不要配置，让Dubbo自动获取本机IP | 2.0.5以上版本 |
| <dubbo:protocol> | threadpool | threadpool | string | 可选 | fixed | 性能调优 | 线程池类型，可选：fixed/cached | 2.0.5以上版本 |
| <dubbo:protocol> | threads | threads | int | 可选 | 100 | 性能调优 | 服务线程池大小(固定大小) | 2.0.5以上版本 |
| <dubbo:protocol> | iothreads | threads | int | 可选 | cpu个数+1 | 性能调优 | io线程池大小(固定大小) | 2.0.5以上版本 |
| <dubbo:protocol> | accepts | accepts | int | 可选 | 0 | 性能调优 | 服务提供方最大可接受连接数 | 2.0.5以上版本 |
| <dubbo:protocol> | payload | payload | int | 可选 | 88388608(=8M) | 性能调优 | 请求及响应数据包大小限制，单位：字节 | 2.0.5以上版本 |
| <dubbo:protocol> | codec | codec | string | 可选 | dubbo | 性能调优 | 协议编码方式 | 2.0.5以上版本 |
| <dubbo:protocol> | serialization | serialization | string | 可选 | dubbo协议缺省为hessian2，rmi协议缺省为java，http协议缺省为json | 性能调优 | 协议序列化方式，当协议支持多种序列化方式时使用，比如：dubbo协议的dubbo,hessian2,java,compactedjava，以及http协议的json等 | 2.0.5以上版本 |
| <dubbo:protocol> | accesslog | accesslog | string/boolean | 可选 |  | 服务治理 | 设为true，将向logger中输出访问日志，也可填写访问日志文件路径，直接把访问日志输出到指定文件 | 2.0.5以上版本 |
| <dubbo:protocol> | path | <path> | string | 可选 |  | 服务发现 | 提供者上下文路径，为服务path的前缀 | 2.0.5以上版本 |
| <dubbo:protocol> | transporter | transporter | string | 可选 | dubbo协议缺省为netty | 性能调优 | 协议的服务端和客户端实现类型，比如：dubbo协议的mina,netty等，可以分拆为server和client配置 | 2.0.5以上版本 |
| <dubbo:protocol> | server | server | string | 可选 | dubbo协议缺省为netty，http协议缺省为servlet | 性能调优 | 协议的服务器端实现类型，比如：dubbo协议的mina,netty等，http协议的jetty,servlet等 | 2.0.5以上版本 |
| <dubbo:protocol> | client | client | string | 可选 | dubbo协议缺省为netty | 性能调优 | 协议的客户端实现类型，比如：dubbo协议的mina,netty等 | 2.0.5以上版本 |
| <dubbo:protocol> | dispatcher | dispatcher | string | 可选 | dubbo协议缺省为all | 性能调优 | 协议的消息派发方式，用于指定线程模型，比如：dubbo协议的all, direct, message, execution, connection等 | 2.1.0以上版本 |
| <dubbo:protocol> | queues | queues | int | 可选 | 0 | 性能调优 | 线程池队列大小，当线程池满时，排队等待执行的队列大小，建议不要设置，当线程程池时应立即失败，重试其它服务提供机器，而不是排队，除非有特殊需求。 | 2.0.5以上版本 |
| <dubbo:protocol> | charset | charset | string | 可选 | UTF-8 | 性能调优 | 序列化编码 | 2.0.5以上版本 |
| <dubbo:protocol> | buffer | buffer | int | 可选 | 8192 | 性能调优 | 网络读写缓冲区大小 | 2.0.5以上版本 |
| <dubbo:protocol> | heartbeat | heartbeat | int | 可选 | 0 | 性能调优 | 心跳间隔，对于长连接，当物理层断开时，比如拔网线，TCP的FIN消息来不及发送，对方收不到断开事件，此时需要心跳来帮助检查连接是否已断开 | 2.0.10以上版本 |
| <dubbo:protocol> | telnet | telnet | string | 可选 |  | 服务治理 | 所支持的telnet命令，多个命令用逗号分隔 | 2.0.5以上版本 |
| <dubbo:protocol> | register | register | boolean | 可选 | true | 服务治理 | 该协议的服务是否注册到注册中心 | 2.0.8以上版本 |
| <dubbo:protocol> | contextpath | contextpath | String | 可选 | 缺省为空串 | 服务治理 |  | 2.0.6以上版本 |

## 协议

dubbo提供的协议有：



### dubbo://

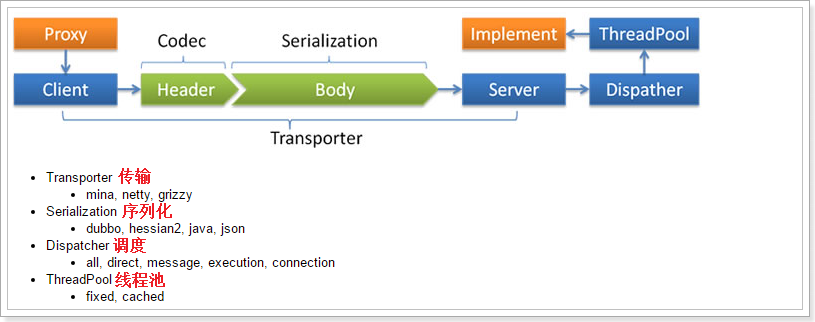
Dubbo缺省协议采用单一长连接和NIO异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。

Dubbo缺省协议不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。

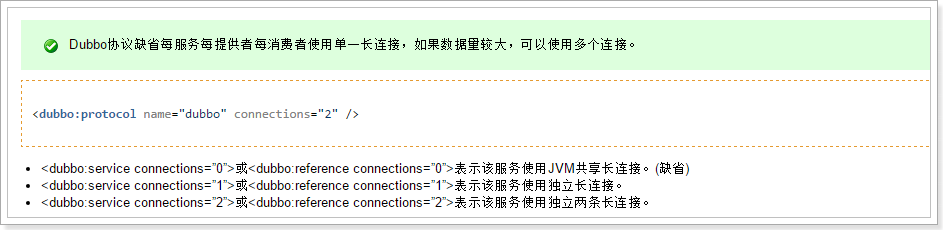
配置：



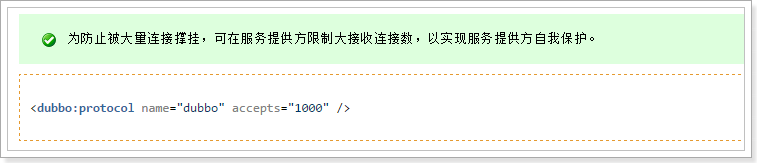
执行过程：



默认连接：



限制最大连接：



## <dubbo:registry/>

注册中心配置：

配置类：com.alibaba.dubbo.config.RegistryConfig

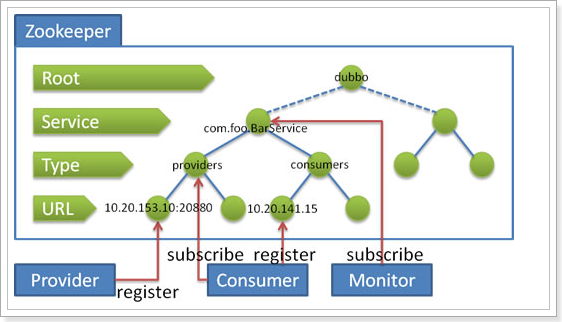
说明：如果有多个不同的注册中心，可以声明多个<dubbo:registry>标签，并在<dubbo:service>或<dubbo:reference>的registry属性指定使用的注册中心。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **标签** | **属性** | **对应URL参数** | **类型** | **是否必填** | **缺省值** | **作用** | **描述** | **兼容性** |
| <dubbo:registry> | id |  | string | 可选 |  | 配置关联 | 注册中心引用BeanId，可以在<dubbo:service registry="">或<dubbo:reference registry="">中引用此ID | 1.0.16以上版本 |
| <dubbo:registry> | address | <host:port> | string | **必填** |  | 服务发现 | 注册中心服务器地址，如果地址没有端口缺省为9090，同一集群内的多个地址用逗号分隔，如：ip:port,ip:port，不同集群的注册中心，请配置多个<dubbo:registry>标签 | 1.0.16以上版本 |
| <dubbo:registry> | protocol | <protocol> | string | 可选 | dubbo | 服务发现 | 注同中心地址协议，支持dubbo, http, local三种协议，分别表示，dubbo地址，http地址，本地注册中心 | 2.0.0以上版本 |
| <dubbo:registry> | port | <port> | int | 可选 | 9090 | 服务发现 | 注册中心缺省端口，当address没有带端口时使用此端口做为缺省值 | 2.0.0以上版本 |
| <dubbo:registry> | username | <username> | string | 可选 |  | 服务治理 | 登录注册中心用户名，如果注册中心不需要验证可不填 | 2.0.0以上版本 |
| <dubbo:registry> | password | <password> | string | 可选 |  | 服务治理 | 登录注册中心密码，如果注册中心不需要验证可不填 | 2.0.0以上版本 |
| <dubbo:registry> | transport | registry.transporter | string | 可选 | netty | 性能调优 | 网络传输方式，可选mina,netty | 2.0.0以上版本 |
| <dubbo:registry> | timeout | registry.timeout | int | 可选 | 5000 | 性能调优 | 注册中心请求超时时间(毫秒) | 2.0.0以上版本 |
| <dubbo:registry> | session | registry.session | int | 可选 | 60000 | 性能调优 | 注册中心会话超时时间(毫秒)，用于检测提供者非正常断线后的脏数据，比如用心跳检测的实现，此时间就是心跳间隔，不同注册中心实现不一样。 | 2.1.0以上版本 |
| <dubbo:registry> | file | registry.file | string | 可选 |  | 服务治理 | 使用文件缓存注册中心地址列表及服务提供者列表，应用重启时将基于此文件恢复，注意：两个注册中心不能使用同一文件存储 | 2.0.0以上版本 |
| <dubbo:registry> | wait | registry.wait | int | 可选 | 0 | 性能调优 | 停止时等待通知完成时间(毫秒) | 2.0.0以上版本 |
| <dubbo:registry> | check | check | boolean | 可选 | true | 服务治理 | 注册中心不存在时，是否报错 | 2.0.0以上版本 |
| <dubbo:registry> | register | register | boolean | 可选 | true | 服务治理 | 是否向此注册中心注册服务，如果设为false，将只订阅，不注册 | 2.0.5以上版本 |
| <dubbo:registry> | subscribe | subscribe | boolean | 可选 | true | 服务治理 | 是否向此注册中心订阅服务，如果设为false，将只注册，不订阅 | 2.0.5以上版本 |
| <dubbo:registry> | dynamic | dynamic | boolean | 可选 | true | 服务治理 | 服务是否动态注册，如果设为false，注册后将显示后disable状态，需人工启用，并且服务提供者停止时，也不会自动取消册，需人工禁用。 | 2.0.5以上版本 |

## 注册中心

### zookeeper注册中心

原理：



流程说明：

* 服务提供者启动时
  + 向/dubbo/com.foo.BarService/providers目录下写入自己的URL地址。
* 服务消费者启动时
  + 订阅/dubbo/com.foo.BarService/providers目录下的提供者URL地址。
  + 并向/dubbo/com.foo.BarService/consumers目录下写入自己的URL地址。
* 监控中心启动时
  + 订阅/dubbo/com.foo.BarService目录下的所有提供者和消费者URL地址。

支持以下功能：

* 当提供者出现断电等异常停机时，注册中心能自动删除提供者信息。
* 当注册中心重启时，能自动恢复注册数据，以及订阅请求。
* 当会话过期时，能自动恢复注册数据，以及订阅请求。
* 当设置<dubbo:registry check="false" />时，记录失败注册和订阅请求，后台定时重试。
* 可通过<dubbo:registry username="admin" password="1234" />设置zookeeper登录信息。
* 可通过<dubbo:registry group="dubbo" />设置zookeeper的根节点，不设置将使用无根树。
* 支持\*号通配符<dubbo:reference group="\*" version="\*" />，可订阅服务的所有分组和所有版本的提供者。

zkClient的使用：

