## 一、Dubbo背景及需求介绍

### 1.1背景

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



* **单一应用架构**
  + 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
  + 此时，用于简化增删改查工作量的 **数据访问框架(ORM)** 是关键。
* **垂直应用架构**
  + 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
  + 此时，用于加速前端页面开发的 **Web框架(MVC)** 是关键。
* **分布式服务架构**
  + 当垂直应用越来越多，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。
* **流动计算架构**
  + 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(SOA)** 是关键。

### 1.2需求



在大规模服务化之前，应用可能只是通过RMI或Hessian等工具，简单的暴露和引用远程服务，通过配置服务的URL地址进行调用，通过F5等硬件进行负载均衡。

**(1) 当服务越来越多时，服务URL配置管理变得非常困难，F5硬件负载均衡器的单点压力也越来越大。**

此时需要一个服务注册中心，动态的注册和发现服务，使服务的位置透明。

并通过在消费方获取服务提供方地址列表，实现软负载均衡和Failover，降低对F5硬件负载均衡器的依赖，也能减少部分成本。

**(2) 当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。**

这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。

**(3) 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？**

为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。

其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阀值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

### 1.3架构

([#](http://dubbo.io/User+Guide-zh.htm#UserGuide-zh-%E6%9E%B6%E6%9E%84))



**节点角色说明：**

* **Provider:** 暴露服务的服务提供方。
* **Consumer:** 调用远程服务的服务消费方。
* **Registry:** 服务注册与发现的注册中心。
* **Monitor:** 统计服务的调用次调和调用时间的监控中心。
* **Container:** 服务运行容器。

**调用关系说明：**

* 0. 服务容器负责启动，加载，运行服务提供者。
* 1. 服务提供者在启动时，向注册中心注册自己提供的服务。
* 2. 服务消费者在启动时，向注册中心订阅自己所需的服务。
* 3. 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。
* 4. 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。
* 5. 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。

**(1) 连通性：**

* 注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小
* 监控中心负责统计各服务调用次数，调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示
* 服务提供者向注册中心注册其提供的服务，并汇报调用时间到监控中心，此时间不包含网络开销
* 服务消费者向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，同时汇报调用时间到监控中心，此时间包含网络开销
* 注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外
* 注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者
* 注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表
* 注册中心和监控中心都是可选的，服务消费者可以直连服务提供者

**(2) 健状性：**

* 监控中心宕掉不影响使用，只是丢失部分采样数据
* 数据库宕掉后，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
* 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
* 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
* 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

**(3) 伸缩性：**

* 注册中心为对等集群，可动态增加机器部署实例，所有客户端将自动发现新的注册中心
* 服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者

**(4) 升级性：**

* 当服务集群规模进一步扩大，带动IT治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力：

## 二、Dubbo概念介绍

### ****2.1 Dubbo是什么？****

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，以及SOA服务治理方案。简单的说，dubbo就是个服务框架，如果没有分布式的需求，其实是不需要用的，只有在分布式的时候，才有dubbo这样的分布式服务框架的需求，并且本质上是个服务调用的东东，说白了就是个远程服务调用的分布式框架

**其核心部分包含**:  
1.**通讯**: 提供对多种基于长连接的NIO框架抽象封装，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。  
2.**容错**: 提供基于接口方法的透明远程过程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持。  
3.**自动发现**: 基于注册中心目录服务，使服务消费方能动态的查找服务提供方，使地址透明，使服务提供方可以平滑增加或减少机器。

### ****2.2 Dubbo能做什么？****

1.透明化的远程方法调用，就像调用本地方法一样调用远程方法，只需简单配置，没有任何API侵入。  
2.软负载均衡及容错机制，可在内网替代F5等硬件负载均衡器，降低成本，减少单点。  
3. 服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。

### ****2.3.dubbo的架构****

**dubbo架构图如下所示**：



**节点角色说明**：  
Provider: 暴露服务的服务提供方。  
Consumer: 调用远程服务的服务消费方。  
Registry: 服务注册与发现的注册中心。  
Monitor: 统计服务的调用次调和调用时间的监控中心。

Container:运行容器。

**对于这些角色来说，其他都还好，Monitor可能前期使用会把它忽略，但是后期会发现它的作用十分明显哦，如服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？为了解决这个问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阀值，记录此时的访问量，再以此访问量乘以机器数反推总容量。**

调用关系说明：  
0 服务容器负责启动，加载，运行服务提供者。  
1 服务提供者在启动时，向注册中心注册自己提供的服务。  
2 服务消费者在启动时，向注册中心订阅自己所需的服务。  
3 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。  
4 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。  
5 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。

### ****2.4.dubbo使用方法****

Dubbo采用全Spring配置方式，透明化接入应用，对应用没有任何API侵入，只需用Spring加载Dubbo的配置即可，Dubbo基于Spring的Schema扩展进行加载。如果不想使用Spring配置，而希望通过API的方式进行调用（不推荐）  
Dubbo采用全Spring配置方式，透明化接入应用，对应用没有任何API侵入，只需用Spring加载Dubbo的配置即可，Dubbo基于Spring的Schema扩展进行加载。

## 三、注册中心、消费者、提供者搭建实例

更好的查看目前发布的接口的消费者和提供者，因此一般都会配置dubbo-admin，方便监控。一般企业使用dubbo也会搭建一个dubbo-admin的。

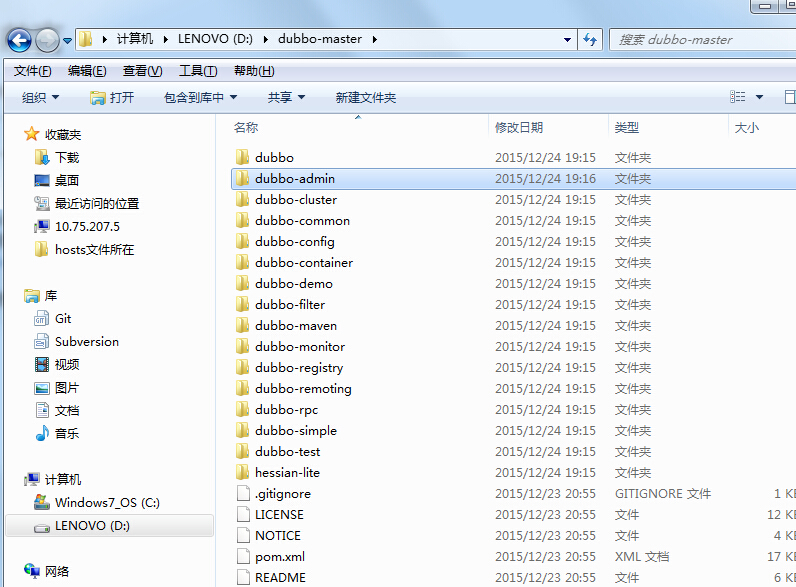
### ****2.1**** Dubbo-admin管理平台搭建

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。

#### 2.1.1配置打包

dubbo-admin的下载，可自行到官网下载：<https://github.com/alibaba/dubbo>   
当然，我也准备好了

一个目录结构：

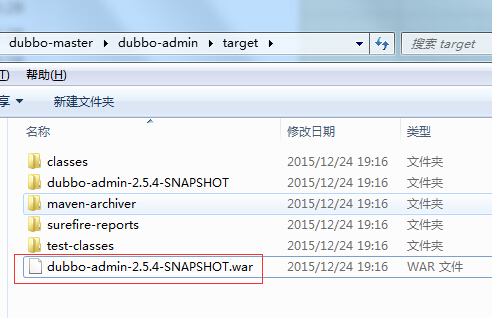


但是这里我们只关心dubbo-admin这个文件夹。

其实下面所谓配置的目的就是为了得到war包，war包网上也有，但是下载了很多下来都会有问题，原因可能是每个人的电脑jdk版本或其他环境不一样，因此我们自己打包一个war就好。

如果一直打包不成功，那么直接用我的war试试吧，注意我的jdk是1.7的

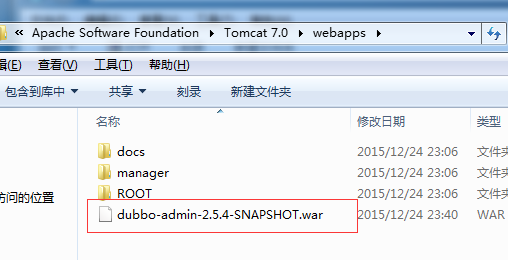
打包成功之后，就会发现dubbo-admin下多了个target文件夹，打开target文件夹，发现里面有个war包：



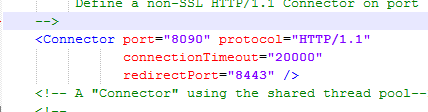
#### 2.1.2安装配置dubbo-admin

第二步我们得到dubbo-admin-2.5.4-SNAPSHOT.war，下面我们将这个war包部署到tomcat上。

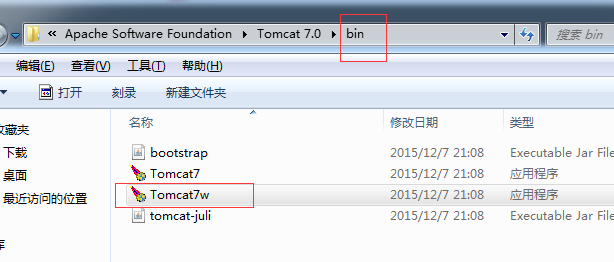
把dubbo-admin-2.5.4-SNAPSHOT.war放到tomcat的webapps目录下：

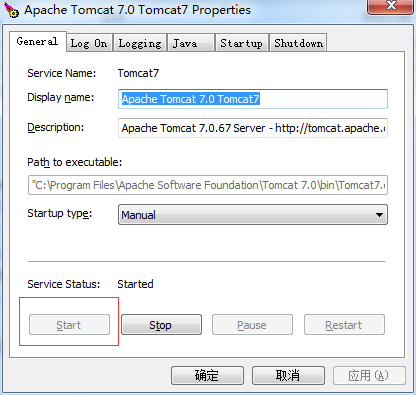


然后打开tomcat的conf目录下的server.xml文件，把启动端口改成8090，因为zookeeper默认使用的是8080，以免冲突：

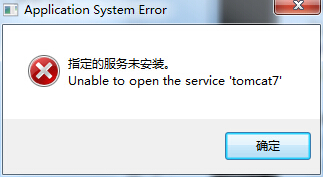


启动tomcat，让它把war解压了

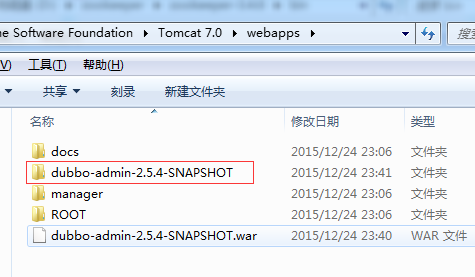




如果一直提示unable to open the service ‘tomcat\*’，那么重新安装一个tomcat（注意一定要使用tomcat安装包，重新安装，不是解压可用那种）

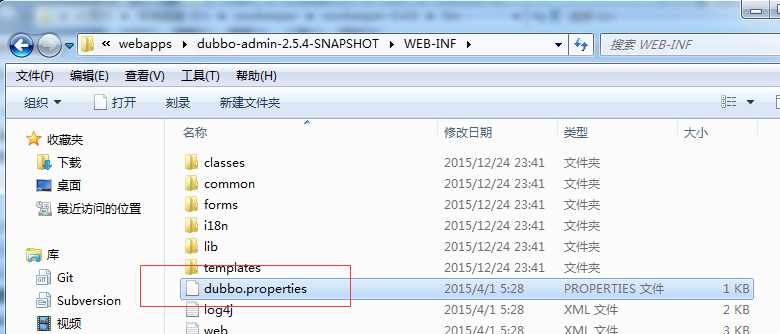


tomcat的webapps里面就会多了这个文件夹：



然后就可以把tomcat stop了

打开刚刚tomcat解压生成的dubbo.properties，我的在Tomcat 7.0\webapps\dubbo-admin-2.5.4-SNAPSHOT\WEB-INF下：



内容是：

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

dubbo.admin.root.password=root

dubbo.admin.guest.password=guest

知道root的密码：root，guest的密码：guest，待会登录用到

### 2.2 Zookeeper的搭建

#### 2.2.1 Zookeeper的介绍

Zookeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是**hadoop**和**hbase**。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、名字服务、分布式同步、组服务等。

之所以把本文归类为Dubbo是因为，关于Dubbo的学习笔记都是以Zookeeper作为注册中心的。

Zookeeper作为Dubbo服务的注册中心，Dubbo原先基于**数据库**中心，没采用Zookeeper，Zookeeper一个分布式的服务框架，是树型的目录服务的数据存储，能做到集群管理数据 ，这里能很好的作为Dubbo服务的注册中心，Dubbo能与Zookeeper做到集群部署，当提供者出现断电等异常停机时，Zookeeper注册中心能自动删除提供者信息，当提供者重启时，能自动恢复注册数据。

关于Zookeeper的部署，但是在生产环境中，你最好部署3，5，7个节点。部署的越多，可靠性就越高。但是，我们作为本地自己熟练使用，部署一个就够了，部署一个Zookeeper是十分简单的。下面是单机部署Zookeeper的实例。

#### 2.2.2 Zookeeper的部署

2.1、 ZooKeeper的下载

官网可下载：<http://www.apache.org/dyn/closer.cgi/zookeeper/>   
当然，也整理了

2.2、配置

下载后解压，Zookeeper 的配置文件在 conf 目录下，有 zoo\_sample.cfg 和 log4j.properties，将zoo\_sample.cfg 重命名成zoo.cfg，因为 Zookeeper 在启动时会找这个文件作为默认配置文件。

下面介绍zoo.cfg文件里面几个配置的意义：

tickTime：这个时间是作为 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。   
dataDir：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。   
clientPort：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。

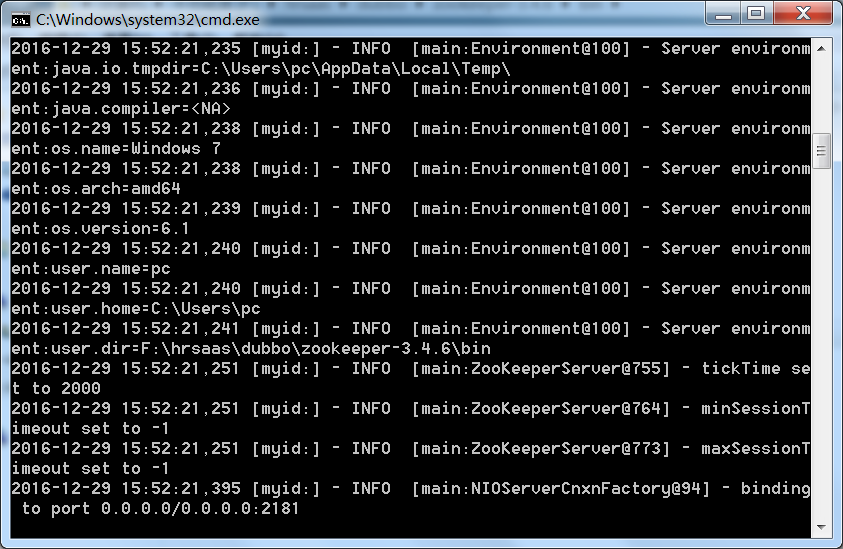
我们需要进行的操作是修改dataDir这个配置的值，我们在Zookeeper的根目录下新建一个文件夹dataTmp，我的对应路径为：D:\Zookeepertest\zookeeper-3.4.6\dataTmp

修改配置文件对应地方为（注意一定是双\哦）：

dataDir=D:\\Zookeepertest\\zookeeper-3.4.6\\dataTmp

2.3、运行

启动Zookeeper目录下bin下的zkServer.cmd，我的是D:\Zookeepertest\zookeeper-3.4.6\bin\zkServer.cmd：



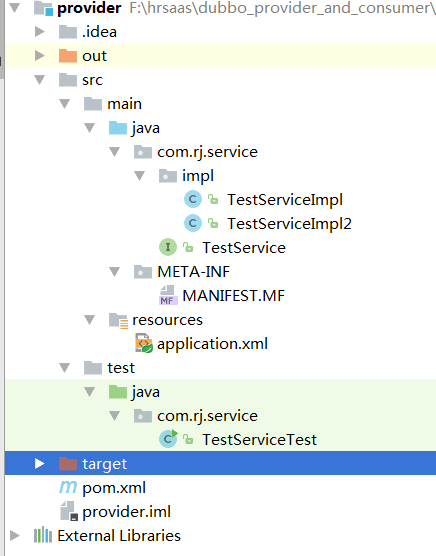
启动之后一直开着，不要关掉这个cmd，记得要在Dubbo消费者和提供者运行之前就开启Zookeeper。

### 2.3 配置提供者

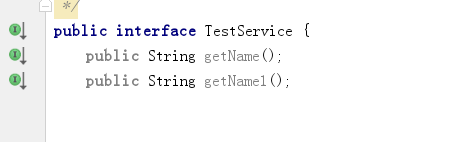
#### 2.3.1 配置pom.xml

<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**groupId**>com.rj</**groupId**>  
 <**artifactId**>provider</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**properties**>  
 <**spring.version**>3.2.8.RELEASE</**spring.version**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>dubbo</**artifactId**>  
 <**version**>2.5.3</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.github.sgroschupf</**groupId**>  
 <**artifactId**>zkclient</**artifactId**>  
 <**version**>0.1</**version**>  
 </**dependency**>  
 *<!-- spring相关 -->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-beans</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jdbc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-web</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-aop</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-tx</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-orm</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context-support</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-test</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jms</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

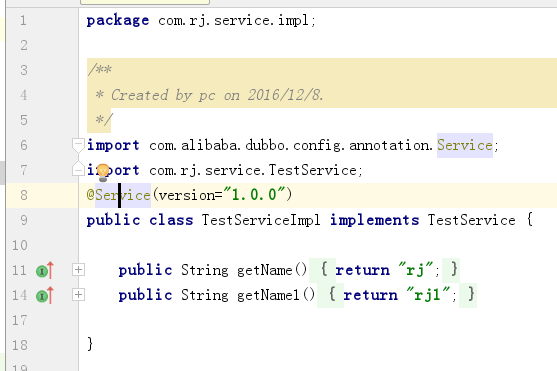
#### 2.3.2添加配置文件，接口和接口实现



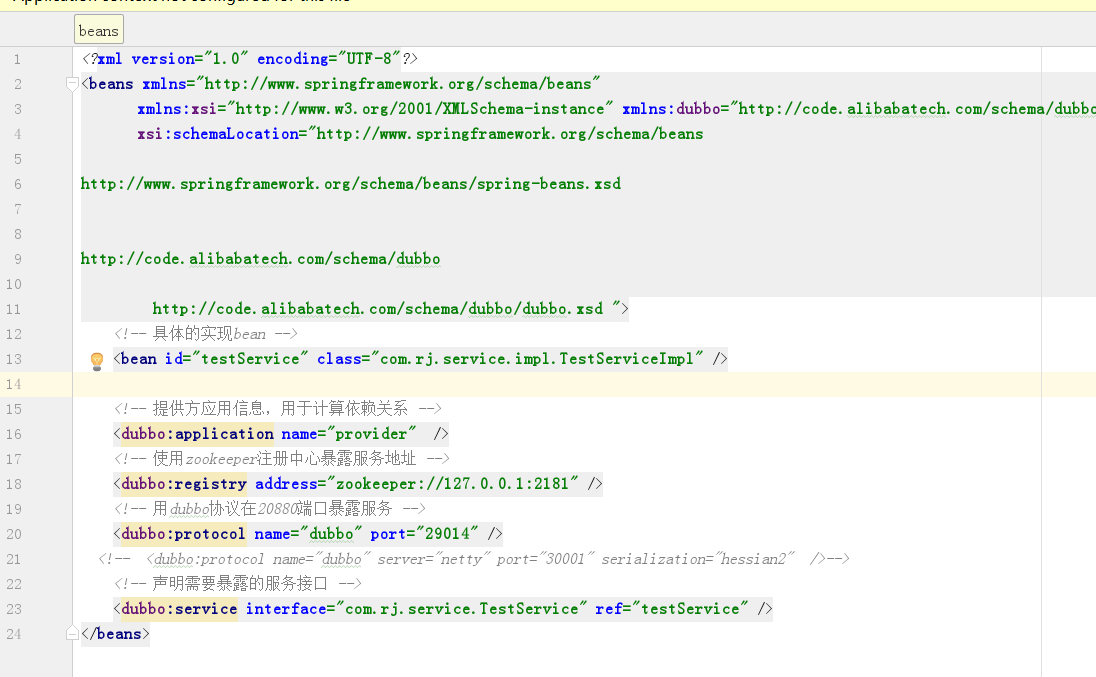
src/main/java下添加包：com.rj.service，然后里面添加接口TestService.java：



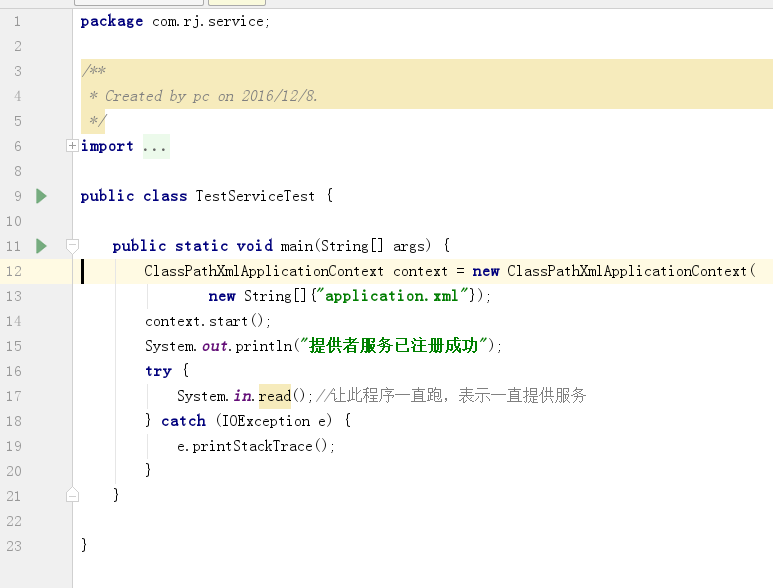
src/main/java下添加包：com.rj.service.impl，然后里面添加接口实现类TestServiceImpl.java：



src/main/resources下添加配置文件application.xml：

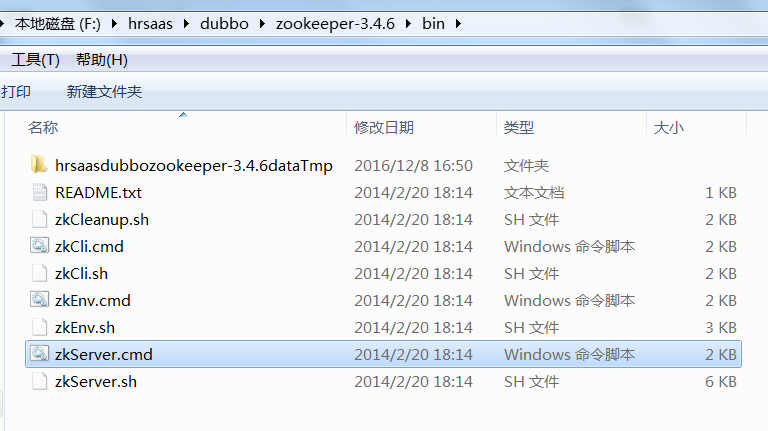


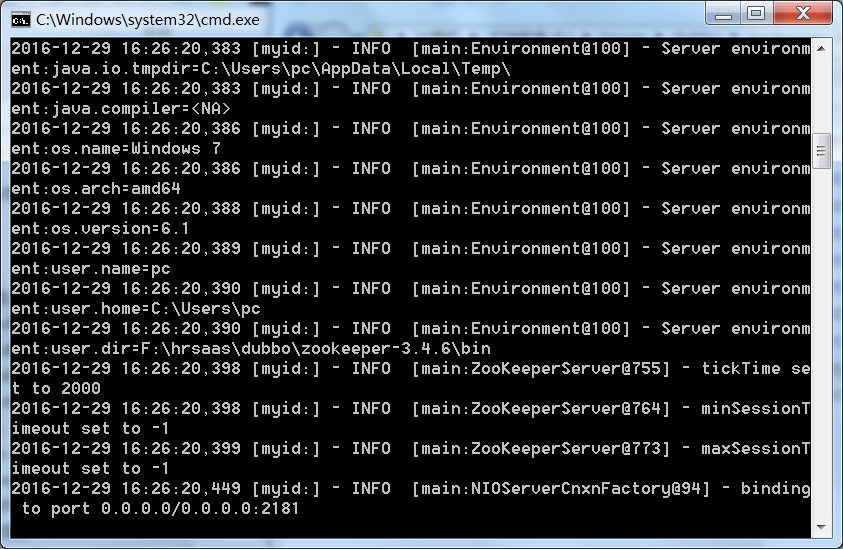
src/test/java下添加包com.rj.service，然后里面添加服务测试类TestServiceTest.java：



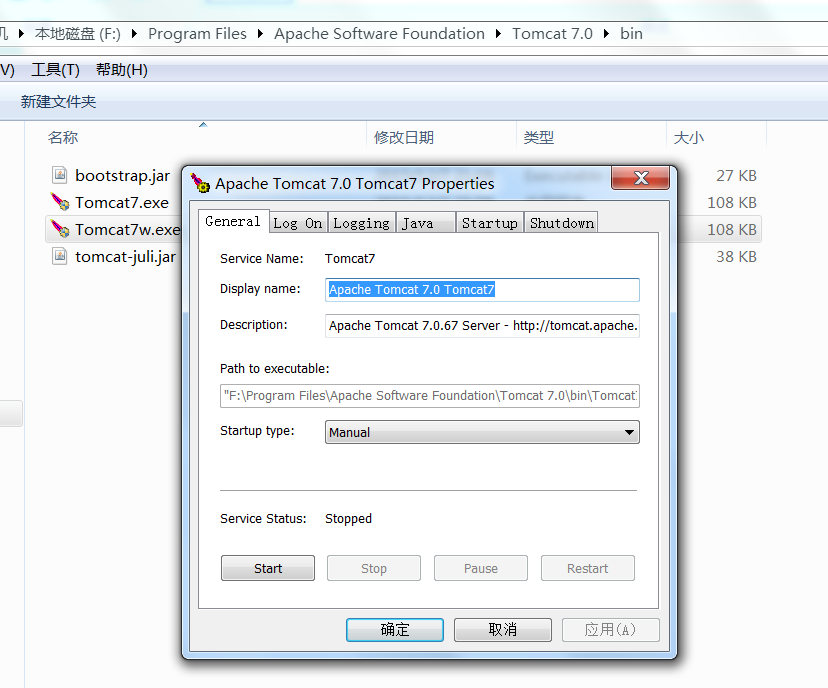
#### 2.3.4运行，注意顺序，一定严格安照下面的顺序

1、运行zookeeper





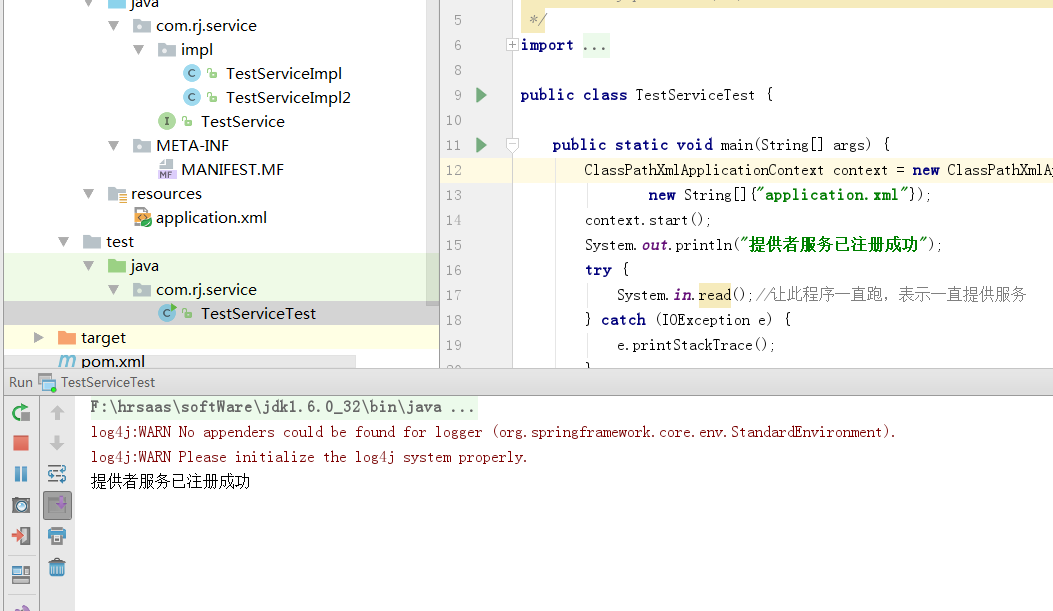
不要关闭黑框，让它打开着，zookeeper 2、运行dubbo-admin，这里我之前已经把它的war包放到tomcat的webapps下了，因此只需要运行tomcat（记得运行tomcat之前一定要把其启动端口改成8090，好启动tomcat：



然后访问localhost:8090/dubbo-admin-2.5.4-SNAPSHOT，结果如下（用户名/密码：root/root），说明启动dubbo-admin成功了



3、运行TestServiceTest，Run As –>Java Application，会看到控制台如下：



保持这个provider一直启动着。 同时在duubo-admin上搜索：com.rj.service.TestService，会看到如下结果：



ok，提供者完成！

### 2.4 配置消费者

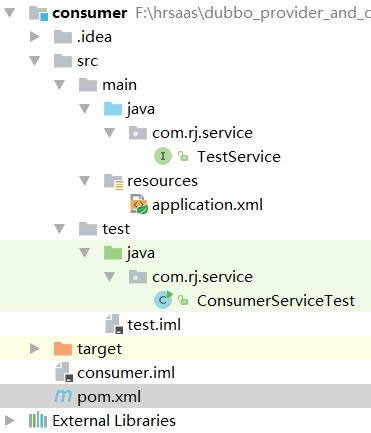
2.4.1创建maven项目，流程跟provider一样，这里我起的项目名为consumer，配置pom.xml文件，如下：

<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**groupId**>com.rj</**groupId**>  
 <**artifactId**>consumer</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
  
 <**properties**>  
 <**spring.version**>3.2.8.RELEASE</**spring.version**>  
 </**properties**>  
  
 <**dependencies**>  
 *<!-- 添加provider的jar包 -->* <**dependency**>  
 <**groupId**>com.rj</**groupId**>  
 <**artifactId**>provider</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 </**dependency**>  
 *<!-- 添加dubbo依赖 -->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>dubbo</**artifactId**>  
 <**version**>2.5.3</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
 *<!-- 添加zk客户端依赖 -->* <**dependency**>  
 <**groupId**>com.github.sgroschupf</**groupId**>  
 <**artifactId**>zkclient</**artifactId**>  
 <**version**>0.1</**version**>  
 </**dependency**>  
 *<!-- spring相关 -->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-beans</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jdbc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-web</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-aop</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-tx</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-orm</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context-support</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-test</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jms</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

其实就是在provider的基础上添加了如下依赖（provider的依赖）：

*<!-- 添加provider的jar包 -->*<**dependency**>  
 <**groupId**>com.rj</**groupId**>  
 <**artifactId**>provider</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
</**dependency**>

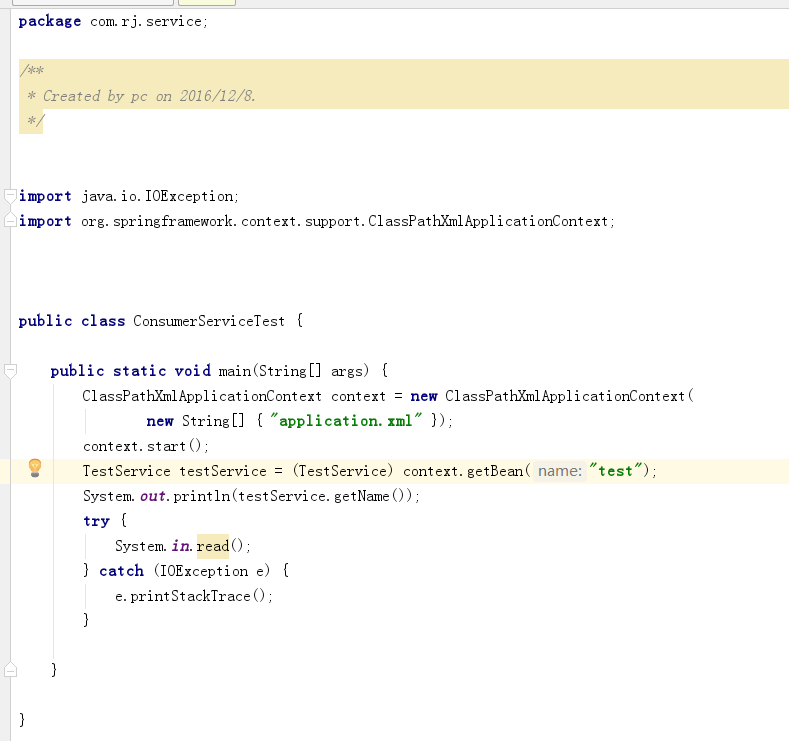
2.3.2、添加如下文件



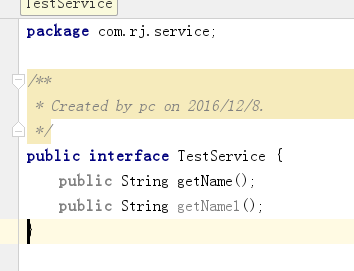
application.xml内容如下：

|  |  |
| --- | --- |
|  |  |

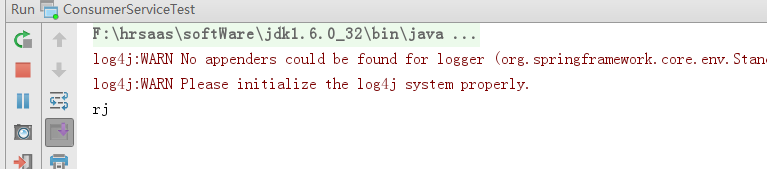
ConsumerServiceTest.java的内容如下：



TestService.java



2.3.3、运行 在保持zookeeper、dubbo-admin和provider运行着的基础上，运行ConsumerServiceTest.java，Run As –> Java Application，会看到控制台如下：



控制台内容rj就是提供者打印出来的 另外也会看到dubbo-admin中com.rj.service.TestService有消费者了：



## 四、基于dubbo框架下的RPC通讯协议性能测试

### 4.1前言

Dubbo RPC服务框架支持丰富的传输协议、序列化方式等通讯相关的配置和扩展。dubbo执行一次RPC请求的过程大致如下：消费者（Consumer）向注册中心（Registry）执行RPC请求，注册中心分配服务URL并路由到具体服务提供方（Provider），消费者和服务提供方建立网络连接，服务提供方在本地创建连接池对象并提供远程服务，对于长连接类型协议（如dubbo协议）将保持连接，减少握手认证，调用过程中可以避免频繁建立和断开连接导致的性能开销，保持长连接需要有心跳包的发送，所以对于非频繁调用的服务保持连接同样会有消耗

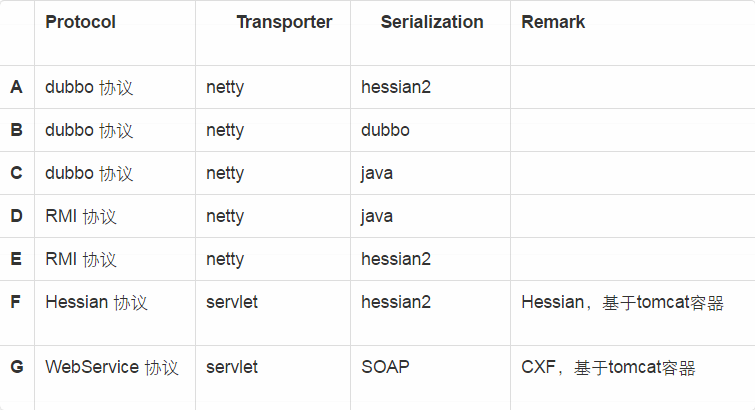
1、支持常见的传输协议：RMI、Dubbo、Hessain、WebService、Http等，其中Dubbo和RMI协议基于TCP实现，Hessian和WebService基于HTTP实现。

2、传输框架：Netty、Mina、以及基于servlet等方式。

3、序列化方式：Hessian2、dubbo、JSON（ [***fastjson***](https://github.com/AlibabaTech/fastjson) 实现）、JAVA、SOAP 等

### 4.2测试方案

基于dubbo 2.5.3框架，使用zookeeper作为dubbo服务注册中心，分别以单线程和多线程的方式测试以下方案：



### 4.3传输测试数据

1、单POJO对象，嵌套复杂集合类型

2、POJO集合，包含100个单POJO对象

3、1K字符串

4、100K字符串

5、1M字符串