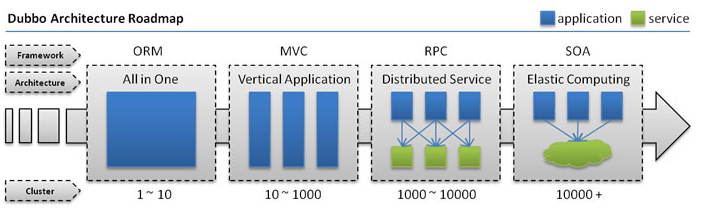
分布式系统所依赖的基础设施包括**服务框架、消息中间件、数据访问中间件、配置中心、分布式缓存系统、持久化存储(关系数据库、nosql数据库)、搜索引擎、CDN网络、负载均衡系统、运维自动化系统、硬件虚拟化及镜像管理系统、分布式文件系统、日志收集系统、监控系统、离线计算、实时计算、数据仓库**等等。随着服务化的进一步发展，服务越来越多，服务之间的调用和依赖关系也越来越复杂，诞生了面向服务的架构体系(SOA)，也因此衍生出了一系列相应的技术，如对服务提供、服务调用、连接处理、通信协议、序列化方式、服务发现、服务路由、日志输出等行为进行封装的服务框架。

    就这样为分布式系统的服务治理框架就出现了，dubbo也就这样产生了。dubbo在整个分布式系统的架构中，按照分层的架构来架构，使得各个层级之间最大限度的松耦合.

1 Dubbo介绍

1.1 dubbox简介

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**](http://lib.csdn.net/base/architecture)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。

**dubbox是dubbo的扩展，主要在dubbo的基础上进行了一下的改进：**

1、支持REST风格远程调用（HTTP + JSON/XML)：基于非常成熟的JBoss RestEasy框架，在dubbo中实现了REST风格（HTTP + JSON/XML）的远程调用，以显著简化企业内部的跨语言交互，同时显著简化企业对外的Open API、无线API甚至AJAX服务端等等的开发。事实上，这个REST调用也使得Dubbo可以对当今特别流行的“微服务”架构提供基础性支持。 另外，REST调用也达到了比较高的性能，在基准测试下，HTTP + JSON与Dubbo 2.x默认的RPC协议（即TCP + Hessian2二进制序列化）之间只有1.5倍左右的差距，详见文档中的基准测试报告。

2、支持基于Kryo和FST的[**Java**](http://lib.csdn.net/base/javaee)高效序列化实现：基于当今比较知名的Kryo和FST高性能序列化库，为Dubbo默认的RPC协议添加新的序列化实现，并优化调整了其序列化体系，比较显著的提高了Dubbo RPC的性能，详见文档中的基准测试报告。

3、支持基于Jackson的JSON序列化：基于业界应用最广泛的Jackson序列化库，为Dubbo默认的RPC协议添加新的JSON序列化实现。

4、支持基于嵌入式Tomcat的HTTP remoting体系：基于嵌入式tomcat实现dubbo的HTTP remoting体系（即dubbo-remoting-http），用以逐步取代Dubbo中旧版本的嵌入式Jetty，可以显著的提高REST等的远程调用性能，并将Servlet API的支持从2.5升级到3.1。（注：除了REST，dubbo中的WebServices、Hessian、HTTP Invoker等协议都基于这个HTTP remoting体系）。

5、升级[**spring**](http://lib.csdn.net/base/javaee)：将dubbo中Spring由2.x升级到目前最常用的3.x版本，减少版本冲突带来的麻烦。

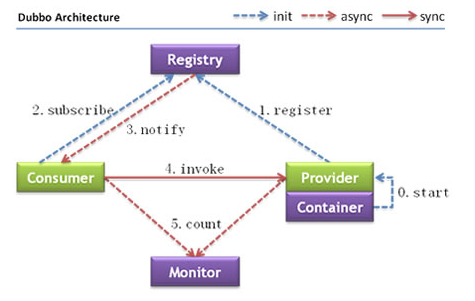
6、升级ZooKeeper客户端：将dubbo中的zookeeper客户端升级到最新的版本，以修正老版本中包含的bug。

7、支持完全基于Java代码的Dubbo配置：基于Spring的[**Java**](http://lib.csdn.net/base/java)Config，实现完全无XML的纯Java代码方式来配置dubbo

8、调整Demo应用：暂时将dubbo的demo应用调整并改写以主要演示REST功能、Dubbo协议的新序列化方式、基于Java代码的Spring配置等等。   
9、修正了dubbo的bug 包括配置、序列化、管理界面等等的bug。

1.2 dubbo架构构成 0

dubbo运行架构如下图示：



* 节点角色说明：

1、Provider:暴露服务的服务提供方。 Consumer: 调用远程服务的服务消费方。

2、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。

3、Container: 服务运行容器。

* 1
* 2
* 3
* 1
* 2
* 3
* 调用关系说明：   
  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。

1.3 dubbo的特性

(1) 连通性：

注册 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**](http://lib.csdn.net/base/datastructure)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：

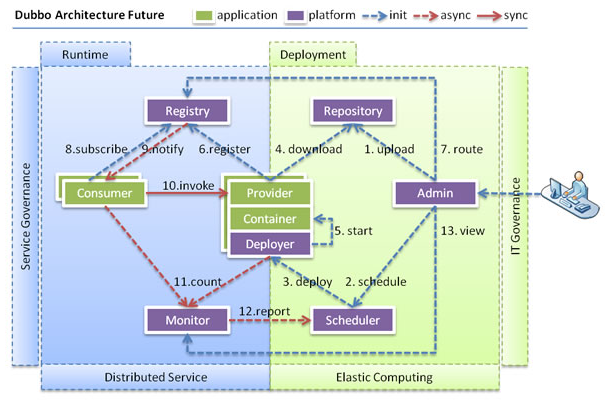
监控中心宕掉不影响使用，只是丢失部分采样数据[**数据库**](http://lib.csdn.net/base/mysql)宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务注册中心对等集群，任意一台宕掉后，将自动切换到另一台注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯服务提供者无状态，任意一台宕掉后，不影响使用服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

(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：

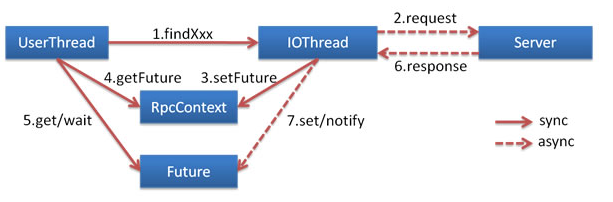
当服务集群规模进一步扩大，带动IT治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力：



1.4 dubbo的调用方式

* 异步调用

基于NIO的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小。



* 本地调用

本地调用，使用了Injvm协议，是一个伪协议，它不开启端口，不发起远程调用，只在JVM内直接关联，但执行Dubbo的Filter链。

Define injvm protocol:

<dubbo:protocol name="injvm" />

* 1
* 1

Set default protocol:

<dubbo:provider protocol="injvm" />

* 1
* 1

Set service protocol:

<dubbo:service protocol="injvm" />

* 1
* 1

Use injvm first:

<dubbo:consumer injvm="true" .../>

<dubbo:provider injvm="true" .../>

或

<dubbo:reference injvm="true" .../>

<dubbo:service injvm="true" .../>

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

注意：服务暴露与服务引用都需要声明injvm=“true”

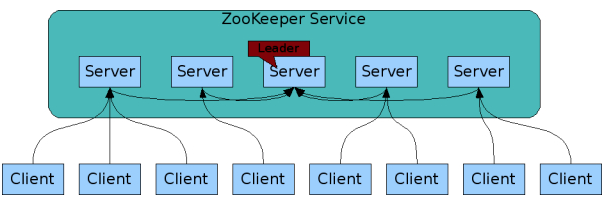
1.5 dubbo支持的注册中心

Dubbo提供的注册中心有如下几种类型可供选择：

* Multicast注册中心
* Zookeeper注册中心
* Redis注册中心
* Simple注册中心

ZooKeeper是一个开源的分布式服务框架，它是Apache [**Hadoop**](http://lib.csdn.net/base/hadoop)项目的一个子项目，主要用来解决分布式应用场景中存在的一些问题，如：**统一命名服务、状态同步服务、集群管理、分布式应用配置管理等**，它支持Standalone模式和分布式模式，在分布式模式下，能够为分布式应用提供高性能和可靠地协调服务，而且使用ZooKeeper可以大大简化分布式协调服务的实现，为开发分布式应用极大地降低了成本。

ZooKeeper总体架构

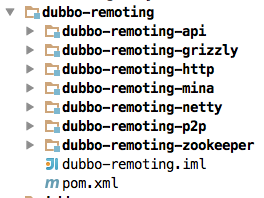


ZooKeeper集群由一组Server节点组成，这一组Server节点中存在一个角色为Leader的节点，其他节点都为Follower。当客户端Client连接到ZooKeeper集群，并且执行写请求时，这些请求会被发送到Leader节点上，然后Leader节点上数据变更会同步到集群中其他的Follower节点。

1.6 dubbo支持的远程通信协议

远程通信需要指定通信双方所约定的协议，在保证通信双方理解协议语义的基础上，还要保证高效、稳定的消息传输。Dubbo继承了当前主流的网络通信框架，主要包括如下几个：

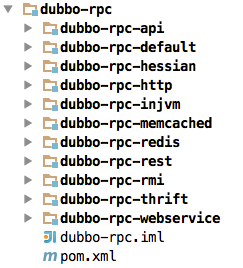
* Mina
* Netty
* Grizzly



1.7 dubbo支持的远程调用协议

Dubbo支持多种协议，如下所示：

* Dubbo协议
* Hessian协议
* HTTP协议
* RMI协议
* WebService协议
* Thrift协议
* Memcached协议
* Redis协议



在通信过程中，不同的服务等级一般对应着不同的服务质量，那么选择合适的协议便是一件非常重要的事情。你可以根据你应用的创建来选择。例如，使用RMI协议，一般会受到防火墙的限制，所以对于外部与内部进行通信的场景，就不要使用RMI协议，而是基于HTTP协议或者Hessian协议。

1.8 dubbo集群容错和负载均衡

**1、集群容错**   
在集群调用失败时，Dubbo提供了多种容错方案，缺省为failover重试。

* Failover Cluster   
  失败自动切换，当出现失败，重试其它服务器。(缺省)   
  通常用于读操作，但重试会带来更长延迟。   
  可通过retries=“2”来设置重试次数(不含第一次)。
* Failfast Cluster   
  快速失败，只发起一次调用，失败立即报错。   
  通常用于非幂等性的写操作，比如新增记录。
* Failsafe Cluster   
  失败安全，出现异常时，直接忽略。   
  通常用于写入审计日志等操作。
* Failback Cluster   
  失败自动恢复，后台记录失败请求，定时重发。   
  通常用于消息通知操作。
* Forking Cluster   
  并行调用多个服务器，只要一个成功即返回。   
  通常用于实时性要求较高的读操作，但需要浪费更多服务资源。   
  可通过forks=“2”来设置最大并行数。
* Broadcast Cluster   
  广播调用所有提供者，逐个调用，任意一台报错则报错。(2.1.0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。

**2、负载均衡**   
- Random LoadBalance随机，按权重设置随机概率。   
在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

* RoundRobin LoadBalance 轮循，按公约后的权重设置轮循比率。   
  存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。
* LeastActive LoadBalance 最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。   
  使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。
* ConsistentHash LoadBalance 一致性Hash，相同参数的请求总是发到同一提供者。   
  当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。

配置如：

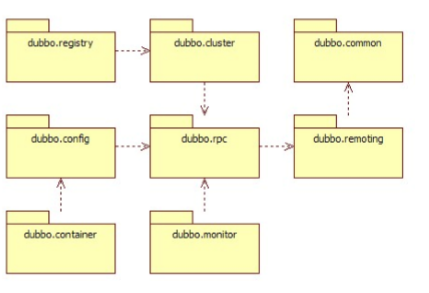
<dubbo:service interface="..." loadbalance="roundrobin" />

* 1
* 1

1.9 dubbo源代码结构



Dubbo以包结构来组织各个模块，各个模块及其关系，如图所示：



* dubbo-common 公共逻辑模块，包括Util类和通用模型。
* dubbo-remoting 远程通讯模块，相当于Dubbo协议的实现，如果RPC用RMI协议则不需要使用此包。
* dubbo-rpc 远程调用模块，抽象各种协议，以及动态代理，只包含一对一的调用，不关心集群的管理。
* dubbo-cluster 集群模块，将多个服务提供方伪装为一个提供方，包括：负载均衡、容错、路由等，集群的地址列表可以是静态配置的，也可以是由注册中心下发。
* dubbo-registry 注册中心模块，基于注册中心下发地址的集群方式，以及对各种注册中心的抽象。
* dubbo-monitor 监控模块，统计服务调用次数，调用时间的，调用链跟踪的服务。
* dubbo-config 配置模块，是Dubbo对外的API，用户通过Config使用Dubbo，隐藏Dubbo所有细节。
* dubbo-[**Container**](http://lib.csdn.net/base/docker) 容器模块，是一个Standalone的容器，以简单的Main加载Spring启动，因为服务通常不需要Tomcat/JBoss等Web容器的特性，没必要用Web容器去加载服务。

1.10 Dubbo内核实现之SPI简单介绍

Dubbo采用微内核+插件体系，使得设计优雅，扩展性强。那所谓的微内核+插件体系是如何实现的呢！即我们定义了服务接口标准，让厂商去实现（如果不了解spi的请谷歌百度下）, jdk通过ServiceLoader类实现spi机制的服务查找功能。

JDK实现spi服务查找: ServiceLoader

首先定义下示例接口

package com.example;

public interface Spi {

booleanisSupport(String name);

String sayHello();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

ServiceLoader会遍历所有jar查找META-INF/services/com.example.Spi文件

A厂商提供实现

package com.a.example;

public class SpiAImpl implements Spi {

publicboolean isSupport(String name) {

return"SPIA".equalsIgnoreCase(name.trim());

}

public String syaHello() {

return “hello 我是厂商A”;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

在A厂商提供的jar包中的META-INF/services/com.example.Spi文件内容为：

com.a.example.SpiAImpl #厂商A的spi实现全路径类名

B厂商提供实现

package com.b.example;

public class SpiBImpl implements Spi {

publicboolean isSupport(String name) {

return"SPIB".equalsIgnoreCase(name.trim());

}

public String syaHello() {

return “hello 我是厂商B”;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

在B厂商提供的jar包中的META-INF/services/com.example.Spi文件内容为：

com.b.example.SpiBImpl #厂商B的spi实现全路径类名

ServiceLoader.load(Spi.class)读取厂商A、B提供jar包中的文件，ServiceLoader实现了Iterable接口可通过while for循环语句遍历出所有实现。

一个接口多种实现，就如策略模式一样提供了策略的实现，但是没有提供策略的选择， 使用方可以根据isSupport方法根据业务传入厂商名来选择具体的厂商。

public class SpiFactory {

//读取配置获取所有实现

privatestatic ServiceLoader spiLoader = ServiceLoader.load(Spi.class);

//根据名字选取对应实现

publicstatic Spi getSpi(String name) {

for(Spi spi : spiLoader) {

if(spi.isSupport(name) ) {

returnspi;

}

}

returnnull;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15

**SPI接口定义**

定义了@SPI注解

public @interface SPI {

Stringvalue() default ""; //指定默认的扩展点

}

* 1
* 2
* 3
* 1
* 2
* 3

只有在接口打了@SPI注解的接口类才会去查找扩展点实现，会依次从这几个文件中读取扩展点

META-INF/dubbo/internal/ //dubbo内部实现的各种扩展都放在了这个目录了

META-INF/dubbo/

META-INF/services/

* 1
* 2
* 3
* 4
* 5
* 1
* 2
* 3
* 4
* 5

我们以Protocol接口为例， 接口上打上SPI注解，默认扩展点名字为dubbo

@SPI("dubbo")

public interface Protocol{

}

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

具体实现的类有:



所以说：Remoting实现是Dubbo协议的实现

2 Dubbo架构部署搭建

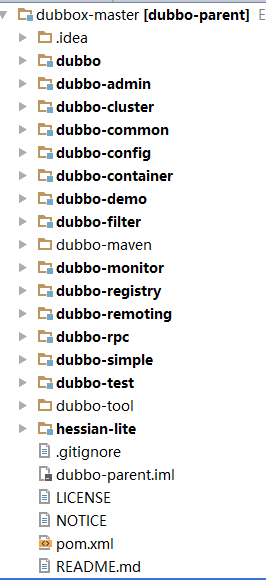
2.1 准备工作

在开始搭建dubbox服务架构前需要完成以下准备工作：

1. 下载安装运行zookeeper 具体安装过程详见zookeeper官网：<http://zookeeper.apache.org/> 使用版本为3.4.7
2. 下载安装tomcat 具体安装过程详见zookeeper官网：<http://tomcat.apache.org/> 使用版本为：7.0.67
3. 下载dubbox代码 github地址：<http://dangdangdotcom.github.io/dubbox>

2.2 build dubbox源码

下载好dubbox的源代码之后，可以看到dubbox的项目架构构成如下：



从上图可以看出dubbox包含的各个组件和功能模块。   
其中dubbo-admin为dubbox的监控管理平台。dubbo-demo中有提供一些dubbox的各种使用实例。我们通过运行dubbo-demo中提供的功能实例来了解dubbox的使用和大致运行原理。

在dubbox中的官方github中提供了如下的说明：

1、[**Git**](http://lib.csdn.net/base/git) clone <https://github.com/dangdangdotcom/dubbox>

2、在checkout出来的dubbox目录执行mvn install -Dmaven.test.skip=true来尝试编译一下dubbo（并将dubbo的jar安装到本地maven库）

3、在checkout出来的dubbox根目录执行mvn idea:idea或者mvn   
eclipse:eclipse，来创建IDE工程文件

4、将项目导入IDE

5、下载解压一个zookeeper，编辑其conf/zoo.cfg后启动zookeeper用作dubbo注册中心：bin/zkServer.sh start

6、用IDE运行/dubbo-demo/dubbo-demo-provider/…/test目录下的DemoProvider启动dubbo服务端，目前他会分别启动dubbo协议（包括用kryo和FST序列化）和REST协议的服务

7、用IDE运行/dubbo-demo/dubbo-demo-consumer/…/test目录下的DemoConsumer来启动dubbo客户端调用上面的服务端，直接看console的输出即可

8、用IDE运行/dubbo-demo/dubbo-demo-consumer/…/test目录下的RestClient来启动rest客户端（模拟非dubbo的rest客户端）调用上面的服务端，直接看console的输出即可

9、可以在浏览器中直接访问<http://localhost:8888/services/users/100.xml>或者<http://localhost:8888/services/users/101.json>之类来测试REST服务

10、了解tomcat和IDE集成的同事，可以直接在IDE中将/dubbo-demo/dubbo-demo-provider/部署到tomcat上，用tomcat的servlet容器来发布REST服务（要同时修改dubbo-demo-provider.xml，请看那个文件中的注释），然后用6、7、8中的方式来访问它。（当然也可以在命令行直接mvn package，然后将生成的war部署到外面的tomcat中做测试）

11、如果想看服务监控效果，或者避免demo抛出找不到监控的异常警告，用IDE运行/dubbo-simple/dubbo-simple-monitor/…/test目录下的SimpleMonitor来启动监控中心即可。

下载代码，执行下边的命令：

mvn install -Dmaven.test.skip=true

* 1
* 1

但是经测试如直接mvn install 的话会出现一系列的build 失败信息。   
需要先修改dubbox根目录下的pom.xml文件，注释：

<!--<plugin>-->

<!--<groupId>org.apache.maven.plugins</groupId>-->

<!--<artifactId>maven-surefire-plugin</artifactId>-->

<!--<configuration>-->

<!--<testFailureIgnore>true</testFailureIgnore>-->

<!--</configuration>-->

<!--</plugin>-->

<!--<plugin>-->

<!--<groupId>org.apache.maven.plugins</groupId>-->

<!--<artifactId>maven-surefire-plugin</artifactId>-->

<!--<version>${maven-surefire-plugin\_version}</version>-->

<!--<configuration>-->

<!--<useSystemClassLoader>true</useSystemClassLoader>-->

<!--<forkMode>once</forkMode>-->

<!--<argLine>${argline}</argLine>-->

<!--<systemProperties>-->

<!--<!– common shared –>-->

<!--<property>-->

<!--<name>transporter</name>-->

<!--<value>${transporter}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>serialization</name>-->

<!--<value>${serialization}</value>-->

<!--</property>-->

<!--<!– server side –>-->

<!--<property>-->

<!--<name>port</name>-->

<!--<value>${port}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>threadpool</name>-->

<!--<value>${threadpool}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>threads</name>-->

<!--<value>${threads}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>iothreads</name>-->

<!--<value>${iothreads}</value>-->

<!--</property>-->

<!--<!– client side –>-->

<!--<property>-->

<!--<name>server</name>-->

<!--<value>${server}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>timeout</name>-->

<!--<value>${timeout}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>length</name>-->

<!--<value>${length}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>connections</name>-->

<!--<value>${connections}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>base</name>-->

<!--<value>${base}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>concurrent</name>-->

<!--<value>${concurrent}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>runs</name>-->

<!--<value>${runs}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>onerror</name>-->

<!--<value>${onerror}</value>-->

<!--</property>-->

<!--</systemProperties>-->

<!--</configuration>-->

<!--</plugin>-->

<!--<plugin>-->

<!--<groupId>org.apache.maven.plugins</groupId>-->

<!--<artifactId>maven-surefire-plugin</artifactId>-->

<!--<version>${maven-surefire-plugin\_version}</version>-->

<!--<configuration>-->

<!--<useSystemClassLoader>true</useSystemClassLoader>-->

<!--<forkMode>once</forkMode>-->

<!--<argLine>${argline}</argLine>-->

<!--<systemProperties>-->

<!--<!– common shared –>-->

<!--<property>-->

<!--<name>transporter</name>-->

<!--<value>${transporter}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>serialization</name>-->

<!--<value>${serialization}</value>-->

<!--</property>-->

<!--<!– server side –>-->

<!--<property>-->

<!--<name>port</name>-->

<!--<value>${port}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>threadpool</name>-->

<!--<value>${threadpool}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>threads</name>-->

<!--<value>${threads}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>iothreads</name>-->

<!--<value>${iothreads}</value>-->

<!--</property>-->

<!--<!– client side –>-->

<!--<property>-->

<!--<name>server</name>-->

<!--<value>${server}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>timeout</name>-->

<!--<value>${timeout}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>length</name>-->

<!--<value>${length}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>connections</name>-->

<!--<value>${connections}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>base</name>-->

<!--<value>${base}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>concurrent</name>-->

<!--<value>${concurrent}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>runs</name>-->

<!--<value>${runs}</value>-->

<!--</property>-->

<!--<property>-->

<!--<name>onerror</name>-->

<!--<value>${onerror}</value>-->

<!--</property>-->

<!--</systemProperties>-->

<!--</configuration>-->

<!--</plugin>-->

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151

等部分。

然后再运行mvn install， 一般情况下有可能还是会build失败。   
尝试不同的网络环境下build，总会成功的。

2.3 运行dubbox-admin

在成功的build好dubbox源代码之后，可以在IDE中运行dubbo-demo项目中的例子。

在运行之前需要：

1.启动zookeeper 执行如下命令启动：

bin/zkServer.sh start

* 1
* 1

2.测试连接zookeeper 执行如下命令测试连接

bin/zkCli.sh -server ip:端口

* 1
* 1

如果发现你确实启动了zookeeper,但是连接不上的情况。请检查防火墙设置。

3.部署dubbo-admin到你的tomcat

首先需要将dubbo-admin.war解压后拷贝所有的文件到 webapp下的/ROOT目录中（首先请删除ROOT目录中的所有文件）。 然后在部署目录（即/ROOT目录）下的WEB-INF目录中找到dubbo.properties文件，打开该文件有如下配置项：

dubbo.registry.address=zookeeper://121.40.97.224:2181

dubbo.admin.root.password=root

dubbo.admin.guest.password=guest

* 1
* 2
* 3
* 1
* 2
* 3

其中dubbo.registry.address项需要配置为注册中心的地址和端口，也就是zookeeper的地址和端口

dubbo.admin.root.password为root管理用户的登录密码。

dubbo.admin.guest.password为guest用户的登录密码。

启动tomcat访问dubbo-admin：   
[http://127.0.0.1:8686](http://127.0.0.1:8686/) 可以看到注册中心的系统环境，系统状态已经供者情况

2.4 设置dubbo-demo中的配置，添加dubbo服务提供者

1.修改配置文件，设置注册中心的地址为安装好的zookeeper

<?xml version="1.0" encoding="UTF-8"?>

<!--

- Copyright 1999-2011 Alibaba Group.

-

- Licensed under the Apache License, Version 2.0 (the "License");

- you may not use this file except in compliance with the License.

- You may obtain a copy of the License at

-

- http://www.apache.org/licenses/LICENSE-2.0

-

- Unless required by applicable law or agreed to in writing, software

- distributed under the License is distributed on an "AS IS" BASIS,

- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

- See the License for the specific language governing permissions and

- limitations under the License.

-->

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

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-2.5.xsd

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

<dubbo:application name="demo-provider" owner="programmer" organization="dubbox"/>

<!--此处修改为安装的注册中心zookeeper地址-->

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

<!--uncomment this if you want to test dubbo's monitor-->

<!--<dubbo:monitor protocol="registry"/>-->

<!-- here we demonstrate both annotation-based and xml-based configs -->

<dubbo:annotation package="com.alibaba.dubbo.demo.user.facade" />

<dubbo:protocol name="dubbo" serialization="kryo" optimizer="com.alibaba.dubbo.demo.SerializationOptimizerImpl"/>

<!--<dubbo:protocol name="dubbo" serialization="fst" optimizer="com.alibaba.dubbo.demo.SerializationOptimizerImpl"/>-->

<!--<dubbo:protocol name="dubbo" serialization="nativejava"/>-->

<!--<dubbo:protocol name="dubbo" serialization="hessian2"/>-->

<!--<dubbo:protocol name="dubbo" serialization="fastjson"/>-->

<!--<dubbo:protocol name="dubbo" serialization="dubbo"/>-->

<!--TODO according to the spring convention, we should use something like keep-alive-->

<!-- use netty server -->

<!--<dubbo:protocol name="rest" port="8888" keepalive="true" server="netty" iothreads="5" threads="100" contextpath="services"/>-->

<!-- use tjws server -->

<!--<dubbo:protocol name="rest" port="8888" server="tjws" contextpath="services"/>-->

<!-- use tomcat server 8888 -->

<dubbo:protocol name="rest" port="8888" threads="500" contextpath="services" server="tomcat" accepts="500"

extension="com.alibaba.dubbo.demo.extension.TraceInterceptor,

com.alibaba.dubbo.demo.extension.TraceFilter,

com.alibaba.dubbo.demo.extension.ClientTraceFilter,

com.alibaba.dubbo.demo.extension.DynamicTraceBinding,

com.alibaba.dubbo.demo.extension.CustomExceptionMapper,

com.alibaba.dubbo.rpc.protocol.rest.support.LoggingFilter"/>

<!-- use the external tomcat or other server with the servlet approach; the port and contextpath must be exactly the same as those in external server -->

<!-- <dubbo:protocol name="rest" port="6080" contextpath="services" server="servlet"/> -->

<!-- <dubbo:protocol name="rest" server="servlet"/> -->

<dubbo:protocol name="http" port="8889"/>

<dubbo:protocol name="hessian" port="8890"/>

<dubbo:protocol name="webservice" port="8892"/>

<dubbo:service interface="com.alibaba.dubbo.demo.bid.BidService" ref="bidService" protocol="dubbo"/>

<!-- we add the group property since there's another annotation-configured service impl: com.alibaba.dubbo.demo.user.facade.AnnotationDrivenUserRestServiceImpl -->

<dubbo:service interface="com.alibaba.dubbo.demo.user.UserService" ref="userService" protocol="dubbo" group="xmlConfig"/>

<dubbo:service interface="com.alibaba.dubbo.demo.user.facade.UserRestService" ref="userRestService" protocol="rest" validation="true"/>

<dubbo:service interface="com.alibaba.dubbo.demo.user.facade.AnotherUserRestService" ref="anotherUserRestService" protocol="rest" timeout="2000" connections="100" validation="true"/>

<bean id="bidService" class="com.alibaba.dubbo.demo.bid.BidServiceImpl" />

<bean id="userService" class="com.alibaba.dubbo.demo.user.UserServiceImpl" />

<bean id="userRestService" class="com.alibaba.dubbo.demo.user.facade.UserRestServiceImpl">

<property name="userService" ref="userService"/>

</bean>

<bean id="anotherUserRestService" class="com.alibaba.dubbo.demo.user.facade.AnotherUserRestServiceImpl">

<property name="userService" ref="userService"/>

</bean>

</beans>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86

2.5 部署运行dubbo-demo-provider,dubbo-demo-consumer到测试服务器

将build之后的dubbo-demo-provider,dubbo-demo-consumer的target目录打包上传至需要部署的服务端。

解压后在target目录编写启动脚本：

#!/bin/bash

java -classpath /root/dubbox/demo/WEB-INF/classes:/root/dubbox/demo/WEB-INF/lib/\*:/root/tomcat/apache-tomcat-7.0.65/lib/\* com.alibaba.dubbo.demo.user.facade.DemoProvider >>dubbox.log 2>&1&

* 1
* 2
* 1
* 2

需要注意将所有依赖的jar包 类文件添加到classpath.

运行该脚本，观察dubbo-admin中的监控情况

3 相关问题及解决方法

3.1 Dubbo-admin无法显示Group分组信息

<http://blog.csdn.net/xlgen157387/article/details/50345545>

3.2 无法访问远程Zookeeper已注册服务的问题

<http://blog.csdn.net/xlgen157387/article/details/50385266>

另外Dubbo项目的案例小Demo可以参考：<https://git.oschina.net/xuliugen/dubbodemo.git>