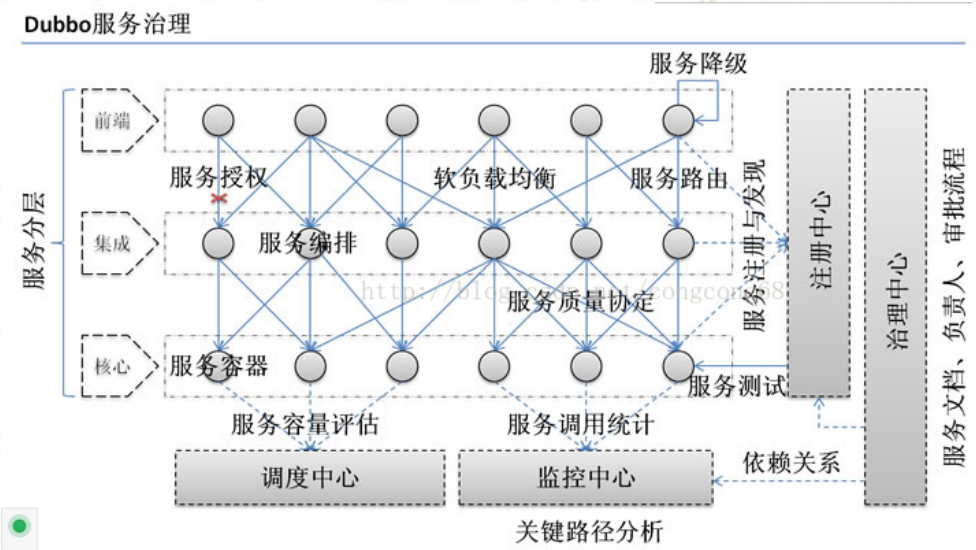
http://blog.csdn.net/congcong68/article/details/41113239

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，Dubbo是一个分布式服务框架，在这种情况下诞生的。现在核心业务抽取出来，作为独立的服务，使前端应用能更快速和稳定的响应。

第一：介绍Dubbo背景



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，实现容错和软负载均衡，降低对F5硬件负载均衡器的依赖，也能减少部分成本。

=======以下转自http://dubbo.io官方文档==========

集群容错模式(consumer的优先级大于provider, 方法级的配置优先级大于类级的配置):

1. Failover Cluster(缺省) <dubbo:service retries="2" /> (重试2次)
   - 失败自动切换，当出现失败，重试其它服务器。
   - 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)。
2. Failfast Cluster <dubbo:service/reference cluster="failsafe"/>
   - 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。
3. Failsafe Cluster <dubbo:service/reference cluster="failsafe"/>
   - 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。
4. Failback Cluster <dubbo:service/reference cluster="failback"/>
   - 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。
5. Forking Cluster <dubbo:service/reference cluster="forking"/>
   - 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。
6. Broadcast Cluster <dubbo:service/reference cluster="broadcast"/>
   - 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。

软负载均衡策略(consumer的优先级大于provider, 方法级的配置优先级大于类级的配置):

Random LoadBalance(缺省) <dubbo:service/reference interface="..." loadbalance="random" />
   - 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 <dubbo:service/reference interface="..." loadbalance="roundrobin" />
   - 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。

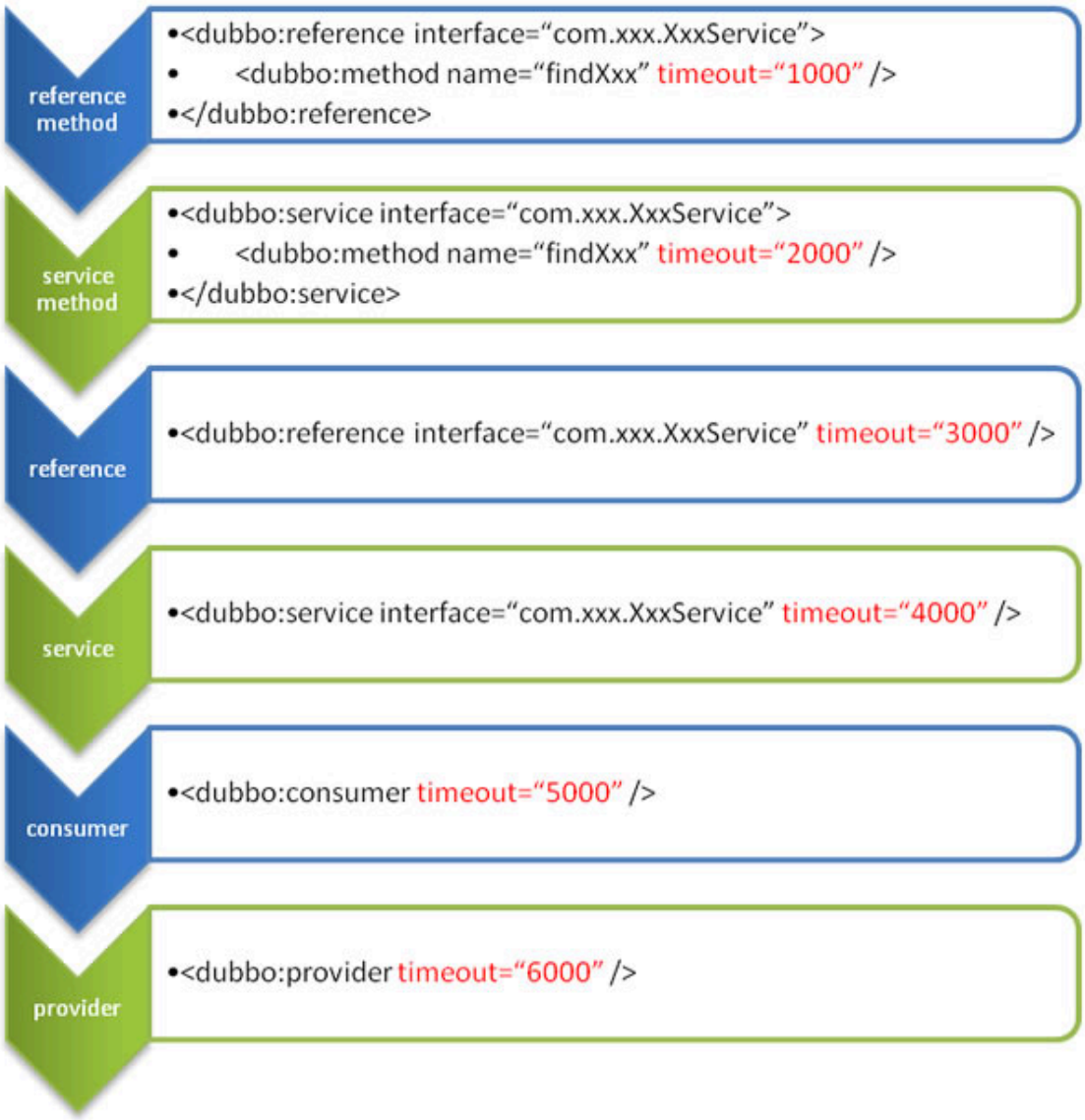
LeastActive LoadBalance <dubbo:service/reference interface="..." loadbalance="leastactive" />
   - 最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。

- 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 `<dubbo:service/reference interface="..." loadbalance="consistenthash" />`

- 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。
- 算法参见：http://en.wikipedia.org/wiki/Consistent_hashing。
- 缺省只对第一个参数Hash，如果要修改，请配置<dubbo:parameter key="hash.arguments" value="0,1" />
- 缺省用160份虚拟节点，如果要修改，请配置<dubbo:parameter key="hash.nodes" value="320" />

以下为配置的优先级关系:



- 上图中以timeout为例，显示了配置的查找顺序，其它retries, loadbalance, actives等类似。
    - 方法级优先，接口级次之，全局配置再次之。
    - 如果级别一样，则消费方优先，提供方次之。

==========

==========

(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。

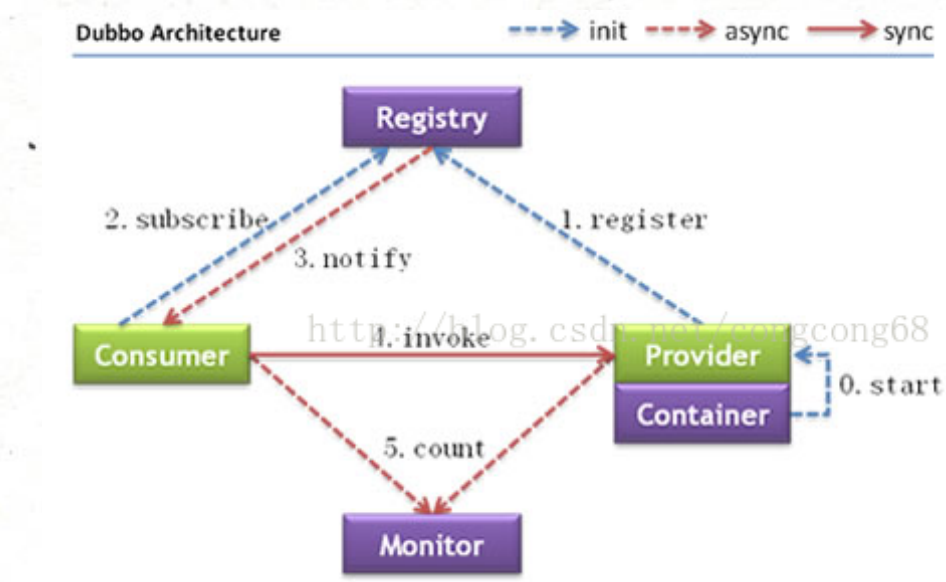
(3) 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？

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。

第二：Dubbo的简介

Dubbo是一个分布式服务框架，解决了上面的所面对的问题，Dubbo的架构如图所示：



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(zookeeper注册中心[推荐],Multicast注册中心[不推荐],Redis注册中心[不推荐])。

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(用main()方法加载Spring容器,用于暴露服务,非Tomcat/JBoss等Web容器)。

调用关系说明(图中的0,1,2,3,4,5详细解释)：

0. 服务容器负责启动，加载，运行服务提供者。

1. Provider在启动时，向注册中心注册自己提供的服务。

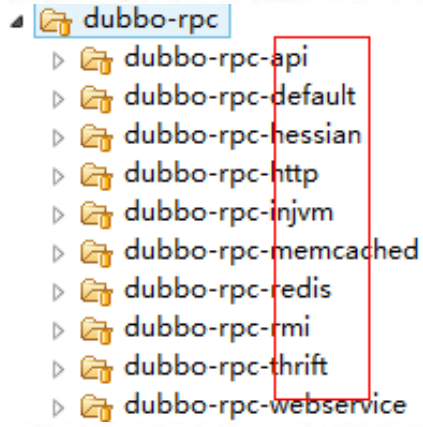
2. Consumer在启动时，向注册中心订阅自己所需的服务。

3. 注册中心返回服务提供者地址列表给Consumer，如果有变更，注册中心将基于长连接推送变更数据给Consumer。

4. Consumer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。

Dubbo提供了很多协议，Dubbo协议、RMI协议、Hessian协议，我们查看Dubbo源代码，有各种协议的实现，如图所示：



我们之前没用Dubbo之前时，大部分都使用Hessian来使用我们服务的暴露和调用，利用HessianProxyFactory调用远程接口。

上面是参考了Dubbo官方网介绍，接下来我们来介绍SpringMVC、Dubbo、Zookeeper整合使用。

第三：Dubbo与Zookeeper、SpringMVC整合使用

第一步：在Linux上安装Zookeeper

Zookeeper作为Dubbo服务的注册中心，Dubbo原先基于**数据库**的注册中心，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，以及订阅请求。我们先在linux上安装Zookeeper，我们安装最简单的单点，集群比较麻烦。

（1）下载Zookeeper-3.4.6.tar.gz 地

址http://www.apache.org/dist/zookeeper/

（2）我们放到Linux下的一个文件夹，然后解压：

#tar zxvf zookeeper-3.4.6.tar.gz

（3）然后在对应的zookeeper-3.4.6/conf 下有一个文件zoo_sample.cfg的这个文件里面配置了监听客户端连接的端口等一些信息，Zookeeper 在启动时会找zoo.cfg这个文件作为默认配置文件,所以我们复制一个名称为zoo.cfg的文件，如图所示：

```
[root@zhengcy opt]# cd zookeeper-3.4.6
[root@zhengcy zookeeper-3.4.6]# ll
总用量 1552
drwxr-xr-x.  2 1000 1000    4096 2月   20 2014 bin
-rw-rw-r--.  1 1000 1000   82446 2月   20 2014 build.xml
-rw-rw-r--.  1 1000 1000   80776 2月   20 2014 CHANGES.txt
drwxr-xr-x.  2 1000 1000    4096 2月   20 2014 conf
drwxr-xr-x. 10 1000 1000    4096 2月   20 2014 contrib
drwxr-xr-x.  2 1000 1000    4096 2月   20 2014 dist-maven
drwxr-xr-x.  6 1000 1000    4096 2月   20 2014 docs
-rw-rw-r--.  1 1000 1000    1953 2月   20 2014 ivysettings.xml
-rw-rw-r--.  1 1000 1000    3375 2月   20 2014 ivy.xml
drwxr-xr-x.  4 1000 1000    4096 2月   20 2014 lib
-rw-rw-r--.  1 1000 1000   11358 2月   20 2014 LICENSE.txt
-rw-rw-r--.  1 1000 1000     170 2月   20 2014 NOTICE.txt
-rw-rw-r--.  1 1000 1000    1770 2月   20 2014 README_packaging.txt
-rw-rw-r--.  1 1000 1000    1585 2月   20 2014 README.txt
drwxr-xr-x.  5 1000 1000    4096 2月   20 2014 recipes
drwxr-xr-x.  8 1000 1000    4096 2月   20 2014 src
-rw-rw-r--.  1 1000 1000 1340305 2月   20 2014 zookeeper-3.4.6.jar
-rw-rw-r--.  1 1000 1000     836 2月   20 2014 zookeeper-3.4.6.jar.asc
-rw-rw-r--.  1 1000 1000      33 2月   20 2014 zookeeper-3.4.6.jar.md5
-rw-rw-r--.  1 1000 1000      41 2月   20 2014 zookeeper-3.4.6.jar.sha1
[root@zhengcy zookeeper-3.4.6]# cd conf
[root@zhengcy conf]# ll
总用量 12
-rw-rw-r--. 1 1000 1000  535 2月   20 2014 configuration.xsl
-rw-rw-r--. 1 1000 1000 2161 2月   20 2014 log4j.properties
-rw-rw-r--. 1 1000 1000  922 2月   20 2014 zoo_sample.cfg
[root@zhengcy conf]# cp zoo_sample.cfg  zoo.cfg
```

我们查看一下这个文件的里面的一些配置信息，如图所示：

```
[root@zhengcy conf]# cat zoo.cfg
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
```


说明：

clientPort：监听客户端连接的端口。

tickTime：基本事件单元，以毫秒为单位。它用来控制心跳和超时，默认情况下最小的会话超时时间为两倍的 tickTime。

我们可以对配置文件的端口等或者进行高级配置和集群配置例如：maxClientCnxns：限制连接到 ZooKeeper 的客户端的数量等

(4)启动Zookeeper 的服务，如图所示：

```
[root@zhengcy zookeeper-3.4.6]# ll
总用量 1552
drwxr-xr-x.   2 1000 1000    4096 2月   20 2014 bin
-rw-rw-r--.   1 1000 1000   82446 2月   20 2014 build.xml
-rw-rw-r--.   1 1000 1000   80776 2月   20 2014 CHANGES.txt
drwxr-xr-x.   2 1000 1000    4096 11月 13 00:38 conf
drwxr-xr-x.  10 1000 1000    4096 2月   20 2014 contrib
drwxr-xr-x.   2 1000 1000    4096 2月   20 2014 dist-maven
drwxr-xr-x.   6 1000 1000    4096 2月   20 2014 docs
-rw-rw-r--.   1 1000 1000    1953 2月   20 2014 ivysettings.xml
-rw-rw-r--.   1 1000 1000    3375 2月   20 2014 ivy.xml
drwxr-xr-x.   4 1000 1000    4096 2月   20 2014 lib
-rw-rw-r--.   1 1000 1000   11358 2月   20 2014 LICENSE.txt
-rw-rw-r--.   1 1000 1000     170 2月   20 2014 NOTICE.txt
-rw-rw-r--.   1 1000 1000    1770 2月   20 2014 README_packaging.txt
-rw-rw-r--.   1 1000 1000    1585 2月   20 2014 README.txt
drwxr-xr-x.   5 1000 1000    4096 2月   20 2014 recipes
drwxr-xr-x.   8 1000 1000    4096 2月   20 2014 src
-rw-rw-r--.   1 1000 1000 1340305 2月   20 2014 zookeeper-3.4.6.jar
-rw-rw-r--.   1 1000 1000     836 2月   20 2014 zookeeper-3.4.6.jar.asc
-rw-rw-r--.   1 1000 1000      33 2月   20 2014 zookeeper-3.4.6.jar.md5
-rw-rw-r--.   1 1000 1000      41 2月   20 2014 zookeeper-3.4.6.jar.sha1
[root@zhengcy zookeeper-3.4.6]# ./bin/zkServer.sh start
JMX enabled by default
Using config: /opt/zookeeper-3.4.6/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

到这边Zookeeper的安装和配置完成

第二步：配置dubbo-admin的管理页面，方便我们管理页面

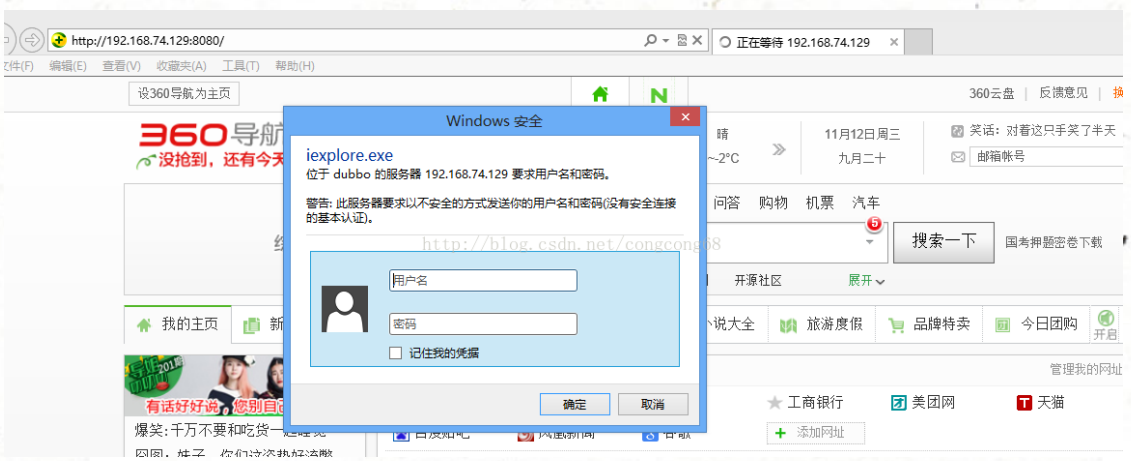
(1)下载dubbo-admin-2.4.1.war包，在Linux的tomcat部署，先把dubbo-admin-2.4.1放在tomcat的webapps/ROOT下，然后进行解压：

#jar -xvf dubbo-admin-2.4.1.war

(2)然后到webapps/ROOT/WEB-INF下，有一个dubbo.properties文件，里面指向Zookeeper ，使用的是Zookeeper 的注册中心，如图所示：

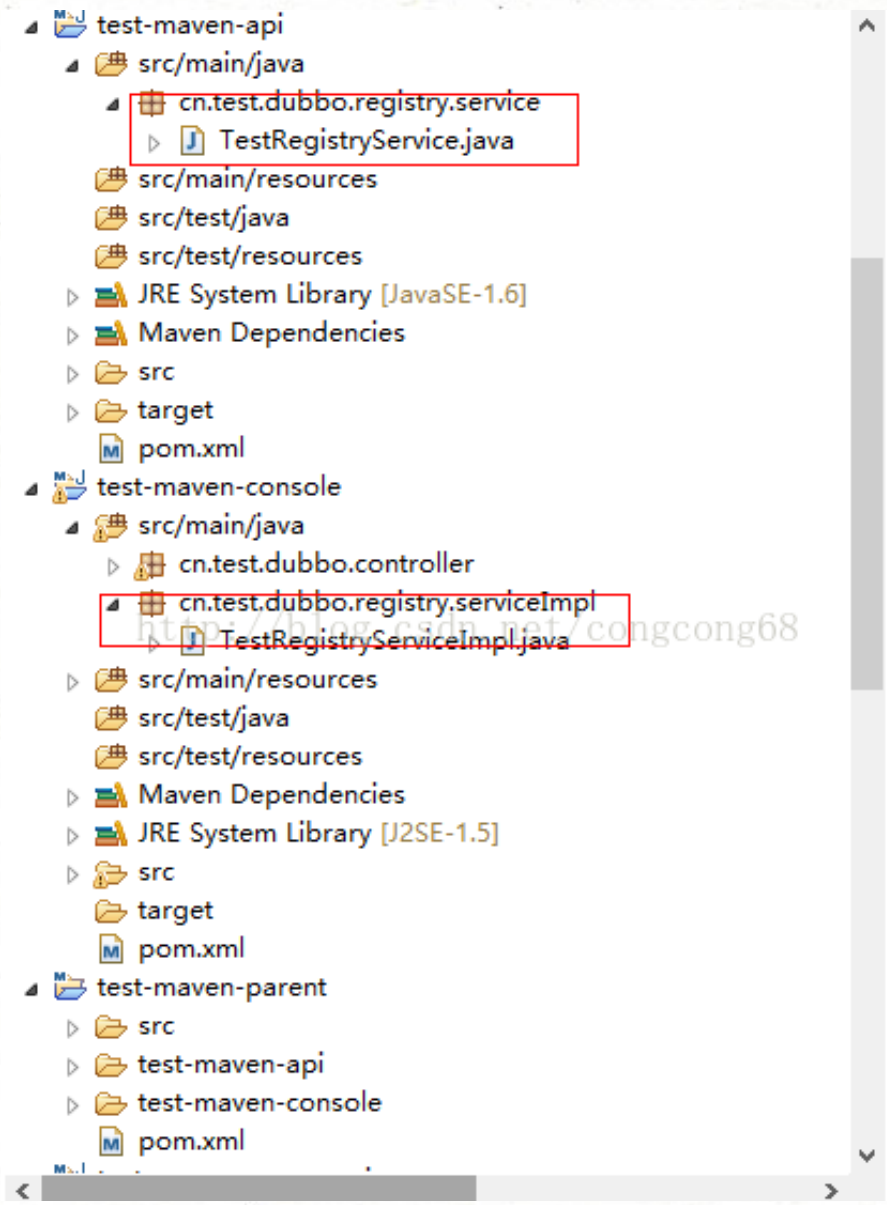
```
[root@zhengcy WEB-INF]# vi  dubbo.properties

dubbo.registry.address=zookeeper://127.0.0.1:2181
dubbo.admin.root.password=root
dubbo.admin.guest.password=guest
~
~
~
~
```

(3)然后启动tomcat服务，用户名和密码：root,并访问服务，显示登陆页面，说明dubbo-admin部署成功，如图所示：



第三步：SpringMVC与Dubbo的整合，这边使用的Maven的管理项目

第一：我们先开发服务注册的，就是提供服务，项目结构如图所示：



（1）test-maven-api项目加入了一个服务接口，代码如下：

[java] view plain copy

```
1. public interface TestRegistryService {
2.     public String hello(String name);
3. }
```

（2）test-maven-console在pom.xml加入Dubbo和Zookeeper的jar包、引用test-maven-api的jar包，代码如下：

[java] view plain copy

```
1.     <dependency>
2.      <groupId>cn.test</groupId>
3.      <artifactId>test-maven-api</artifactId>
```

```
    <version>0.0.1-SNAPSHOT</version>
</dependency>

  <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
         <version>2.5.3</version>
    </dependency>

     <dependency>
       <groupId>org.apache.zookeeper</groupId>
<artifactId>zookeeper</artifactId>
<version>3.4.6</version>
    </dependency>

  <dependency>
    <groupId>com.github.sgroschupf</groupId>
<artifactId>zkclient</artifactId>
<version>0.1</version>
  </dependency>
```

(3)test-maven-console实现具体的服务，代码如下：

[java] view plain copy

```
@Service("testRegistryService")
ublic class TestRegistryServiceImpl implements
TestRegistryService {
public String hello(String name) {
    return "hello"+name;
}
```

(4)我们服务以及实现好了，这时要暴露服务，代码如下：

[java] view plain copy

```
<?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:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    <span
```

```
style="color:#cc0000;">xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"</span>
7.      xmlns:context="http://www.springframework.org/schema/context"
8.
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
9.      http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
10.      http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
11.      <span
style="color:#990000;">http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd</span>
12.      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.1.xsd"
13.      default-lazy-init="false" >
14.     <!-- 提供方应用名称信息，这个相当于起一个名字，我们dubbo管理页面比较清晰是哪个应用暴露出来的 -->
15.     <dubbo:application name="dubbo_provider"></dubbo:application>
16.     <!-- 使用zookeeper注册中心暴露服务地址 -->
17.     <dubbo:registry address="zookeeper://127.0.0.1:2181"
check="false" subscribe="false" register=""></dubbo:registry>
18.    <!-- 要暴露的服务接口 -->
19.    <dubbo:service
interface="cn.test.dubbo.registry.service.TestRegistryService"
ref="testRegistryService" />
20. </beans>
```

说明：

dubbo:registry 标签一些属性的说明：

1）register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）check注册中心不存在时，是否报错。

3）subscribe是否向此注册中心订阅服务，如果设为false，将只注册，不订阅。

4）timeout注册中心请求超时时间(毫秒)。

5）address可以Zookeeper集群配置，地址可以多个以逗号

隔开等。

dubbo:service标签的一些属性说明：

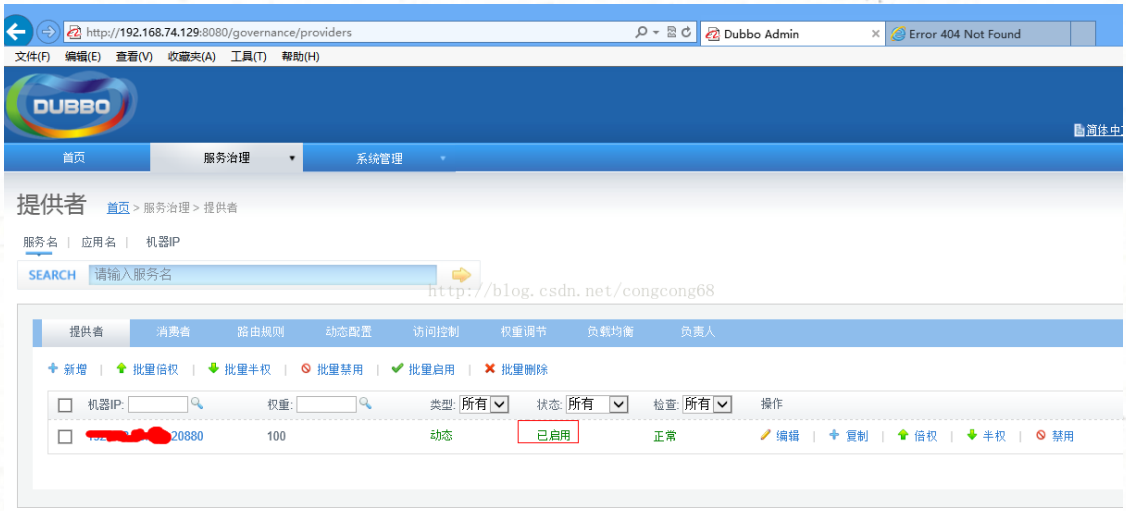
1）interface服务接口的路径

2）ref引用对应的实现类的Bean的ID

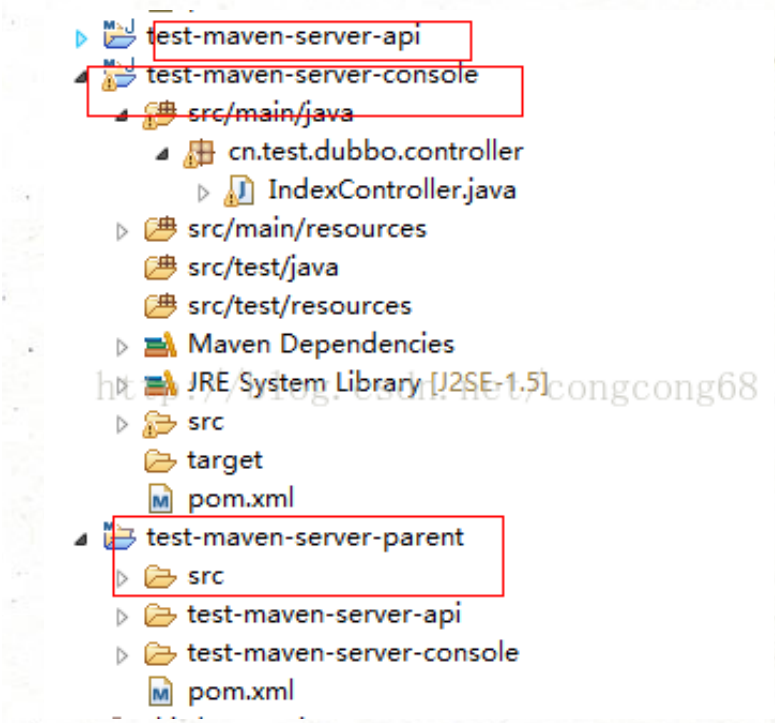
3）registry向指定注册中心注册，在多个注册中心时使用，值为<dubbo:registry>的id属性，多个注册中心ID用逗号分隔，如果不想将该服务注册到任何registry，可将值设为N/A

4）register 默认true ，该协议的服务是否注册到注册中心。

(5)启动项目，然后我们在Dubbo管理页面上显示，已经暴露的服务，但显示还没有消费者，因为我们还没实现消费者服务，如图所示：



第二：我们在开发服务消费者，就是调用服务，我们在新建一个新的消费者项目结构如图所示：



（1）test-maven-server-console的pom.xml引入Dubbo和Zookeeper的jar包、test-maven-api的jar包，因为引入test-maven-api的jar包，我们在项目中调用像在本地调用一样。代码如下：

[java] view plain copy

```
1.    <dependency>
2.     <groupId>cn.test</groupId>
3.     <artifactId>test-maven-api</artifactId>
4.     <version>0.0.1-SNAPSHOT</version>
5. </dependency>
6.
7.    <dependency>
8.          <groupId>com.alibaba</groupId>
9.          <artifactId>dubbo</artifactId>
10.          <version>2.5.3</version>
11.       </dependency>
12.
13.        <dependency>
14.          <groupId>org.apache.zookeeper</groupId>
15. <artifactId>zookeeper</artifactId>
16. <version>3.4.6</version>
```

```
17.       </dependency>
18.
19.     <dependency>
20.       <groupId>com.github.sgroschupf</groupId>
21. <artifactId>zkclient</artifactId>
22. <version>0.1</version>
23.     </dependency>
```

（2）test-maven-server-console项目的具体实现，代码如下：

[java] view plain copy

```
1. @Controller
2. public class IndexController {
3.
4.     @Autowired
5.     private TestRegistryService testRegistryService;
6.
7.     @RequestMapping("/hello")
8.     public String index(Model model){
9.           String name=testRegistryService.hello("zz");
10.           System.out.println("xx=="+name);
11.           return "";
12.     }
13.
14. }
```

(3)我们要引用的地址，代码如下：

[java] view plain copy

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:jee="http://www.springframework.org/schema/jee"
5.     xmlns:tx="http://www.springframework.org/schema/tx"
6.     <span style="background-color: rgb(255, 255, 255);"><span
style="color:#990000;">xmlns:dubbo="http://code.alibabatech.com/sche
ma/dubbo"</span></span>
7.     xmlns:context="http://www.springframework.org/schema/context"
8.
```

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
    http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
    <span
style="color:#990000;">http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd</span>
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.1.xsd"
    default-lazy-init="false" >

    <dubbo:application name="dubbo_consumer"></dubbo:application>
    <!-- 使用zookeeper注册中心暴露服务地址 -->
    <dubbo:registry address="zookeeper://192.168.74.129:2181"
check="false"></dubbo:registry>
     <!-- 要引用的服务 -->
    <dubbo:reference
interface="cn.test.dubbo.registry.service.TestRegistryService"
id="testRegistryService"
                    loadbalance="roundrobin"></dubbo:reference>
</beans>
```

说明：

dubbo:reference 的一些属性的说明：

1）interface调用的服务接口

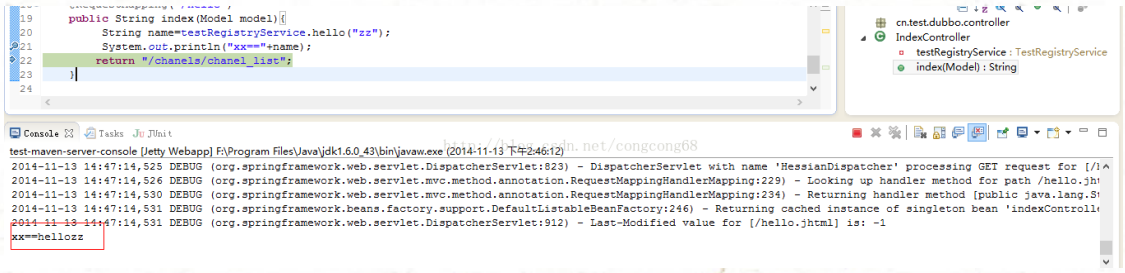
2）check 启动时检查提供者是否存在，true报错，false忽略

3）registry 从指定注册中心注册获取服务列表，在多个注册中心时使用，值为<dubbo:registry>的id属性，多个注册中心ID用逗号分隔

4）loadbalance 负载均衡策略：1.random(随机,按权重设置随机概率)(默认的),2.roundrobin(轮循),3.leastactive(最少活跃调用),4.ConsistentHash(一致性Hash，相同参数的请求总是发到同一提供者)

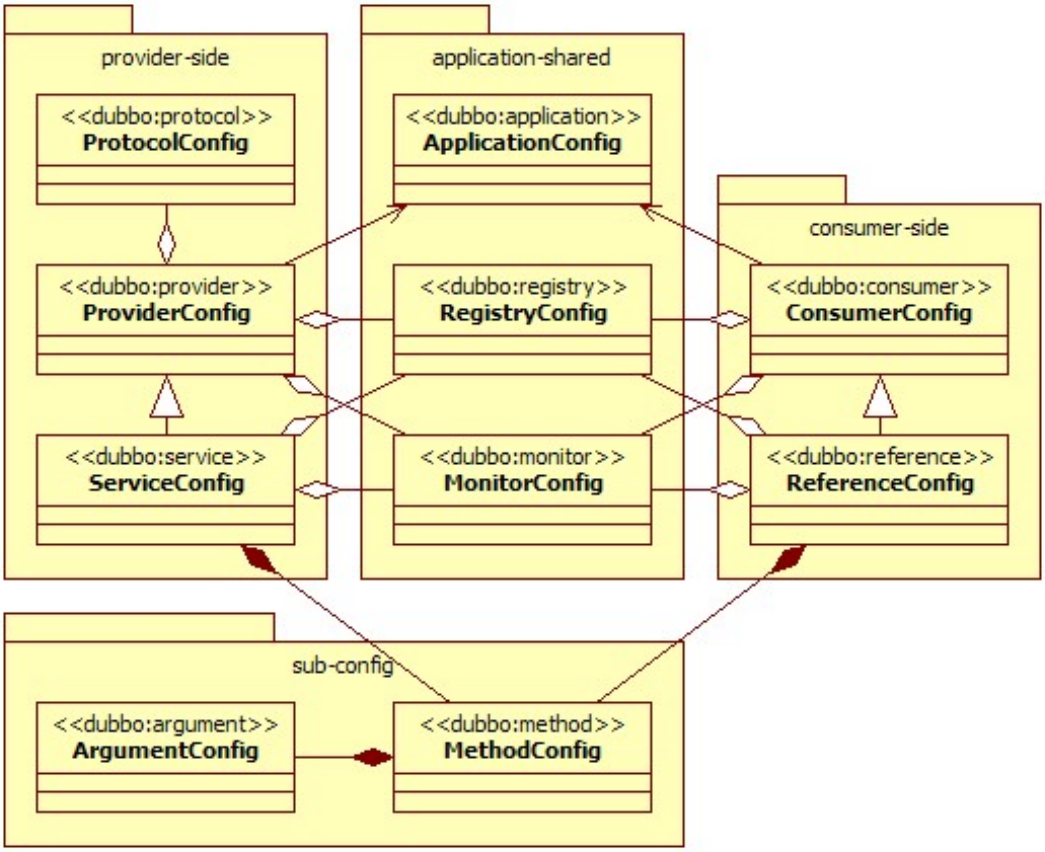
(4)项目启动，Dubbo管理页面，能看到消费者，如图所示：

(5)然后访问消费者项目，Controller层能像调用本地一样调用服务的具体实现，如图所示：



Dubbo提供了多种容错方案，包括负载均衡这些，如图所示：

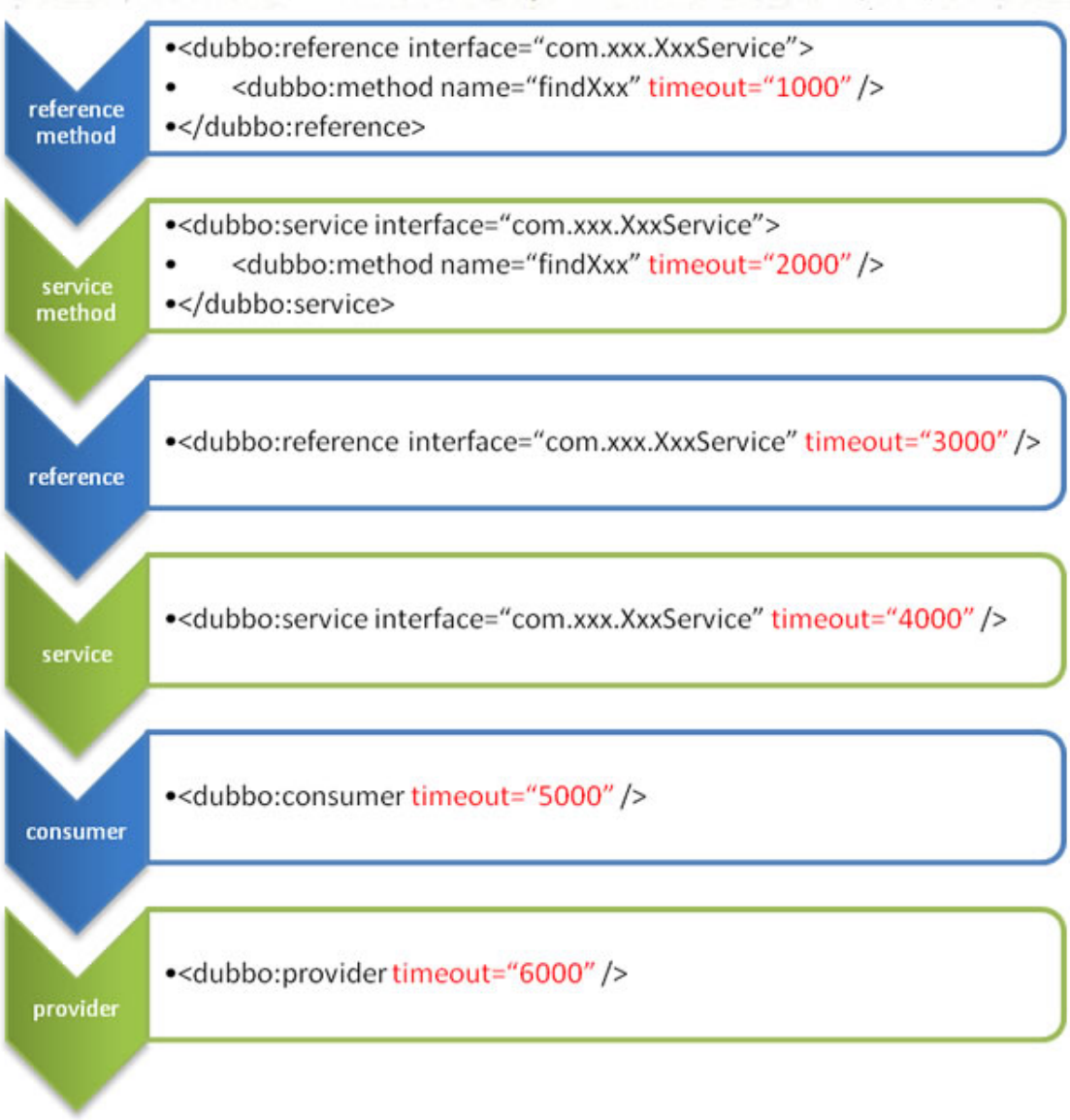
Configuration Relation:



- <dubbo:service/> 服务配置，用于暴露一个服务，定义服务的元信息，一个服务可以用多个协议暴露，一个服务也可以注册到多个注册中心。
- <dubbo:reference/> 引用配置，用于创建一个远程服务代理，一个引用可以指向多个注册中心。

- <dubbo:protocol/> 协议配置，用于配置提供服务的协议信息，协议由提供方指定，消费方被动接受。
- <dubbo:application/> 应用配置，用于配置当前应用信息，不管该应用是提供者还是消费者。
- <dubbo:module/> 模块配置，用于配置当前模块信息，可选。
- <dubbo:registry/> 注册中心配置，用于配置连接注册中心相关信息。
- <dubbo:monitor/> 监控中心配置，用于配置连接监控中心相关信息，可选。
- <dubbo:provider/> 提供方的缺省值，当ProtocolConfig和ServiceConfig某属性没有配置时，采用此缺省值，可选。
- <dubbo:consumer/> 消费方缺省配置，当ReferenceConfig某属性没有配置时，采用此缺省值，可选。
- <dubbo:method/> 方法配置，用于ServiceConfig和ReferenceConfig指定方法级的配置信息。
- <dubbo:argument/> 用于指定方法参数配置。

Configuration Override:



- 上图中以timeout为例，显示了配置的查找顺序，其它retries, loadbalance, actives等类似。
  - 方法级优先，接口级次之，全局配置再次之。
  - 如果级别一样，则消费方优先，提供方次之。
- 其中，服务提供方配置，通过URL经由注册中心传递给消费方。
- 建议由服务提供方设置超时，因为一个方法需要执行多长时间，服务提供方更清楚，如果一个消费方同时引用多个服务，就不需要关心每个服务的超时设置。
- 理论上ReferenceConfig的非服务标识配置，在ConsumerConfig，ServiceConfig, ProviderConfig均可以缺省配置。