# 简介

Dubbo[http://code.alibabatech.com/wiki/download/attachments/6951075/dubbo-pronounced.jpg?version=1&modificationDate=1325057267000]是一个分布式服务框架，致力于提供高性能和透明化的[RPC](http://en.wikipedia.org/wiki/Remote_procedure_call)远程服务调用方案，以及[SOA](http://en.wikipedia.org/wiki/Service-oriented_architecture)服务治理方案。

DUBBO的核心部分包含:

* 远程通讯: 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。
* 集群容错: 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。
* 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

# 功能特性

DUBBO的产品特性，包括：

(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：

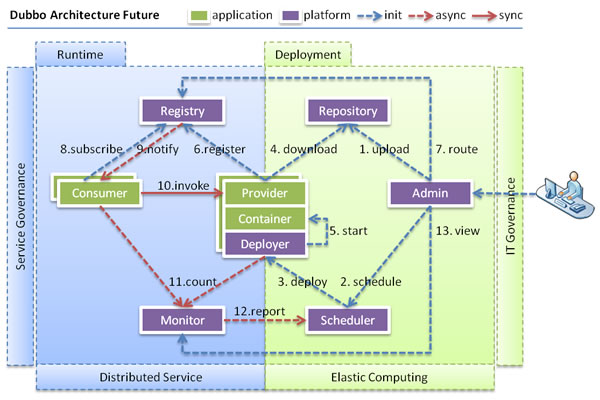
* 监控中心宕掉不影响使用，只是丢失部分采样数据
* 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
* 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
* 注册中心全部宕掉后，服务消费者通过本地缓存的服务提供者列表，与服务提供者通讯
* 服务提供者无状态，任意一台宕掉后，不影响使用
* 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

(3) 伸缩性：

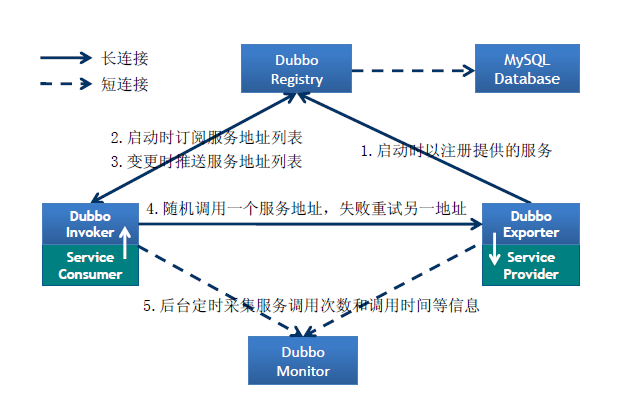
* 注册中心为对等集群，可动态增加机器部署实例，注册中心发送新注册中心集群列表到所有客户端（即自动发现新的注册中心）
* 服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者

(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：



# 原理逻辑示意图



**节点角色说明：**

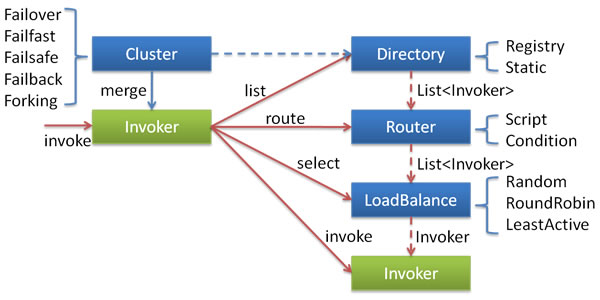
* Provider: 暴露服务的服务提供方。
* Consumer: 调用远程服务的服务消费方。
* Registry: 服务注册与发现的注册中心。
* Monitor: 统计服务的调用次调和调用时间的监控中心。
* Container: 服务运行容器。

**调用关系说明：**

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。

# 应用框架（说明该产品的应用模式）

## 集群容错



各节点关系：

* 这里的Invoker是Provider的一个可调用Service的抽象，Invoker封装了Provider地址及Service接口信息。
* Directory代表多个Invoker，可以把它看成List<Invoker>，但与List不同的是，它的值可能是动态变化的，比如注册中心推送变更。
* Cluster将Directory中的多个Invoker伪装成一个Invoker，对上层透明，伪装过程包含了容错逻辑，调用失败后，重试另一个。
* Router负责从多个Invoker中按路由规则选出子集，比如读写分离，应用隔离等。
* LoadBalance负责从多个Invoker中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选

## 负载均衡

在集群负载均衡时，Dubbo提供了多种均衡策略，缺省为random随机调用。

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。

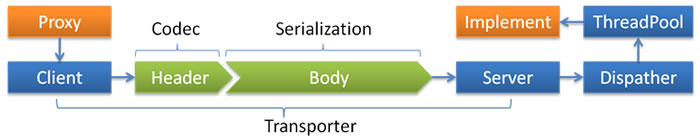
LeastActive LoadBalance

* 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。
* 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" />

## 线程模型



|  |  |
| --- | --- |
| http://code.alibabatech.com/wiki/images/icons/emoticons/check.gif | **事件处理线程说明**   * 如果事件处理的逻辑能迅速完成，并且不会发起新的IO请求，比如只是在内存中记个标识，则直接在IO线程上处理更快，因为减少了线程池调度。 * 但如果事件处理逻辑较慢，或者需要发起新的IO请求，比如需要查询数据库，则必须派发到线程池，否则IO线程阻塞，将导致不能接收其它请求。 * 如果用IO线程处理事件，又在事件处理过程中发起新的IO请求，比如在连接事件中发起登录请求，会报“可能引发死锁”异常，但不会真死锁。 |

# 使用限制

Dubbo运行JDK1.5之上。

# 部署依赖关系

## 依赖系统

缺省依赖javassist、netty、spring等包，但不是必须依赖，通过配置Dubbo可不依赖任何三方库运行。

DUBBO缺省依赖：

* log4j.jar和commons-logging.jar日志输出包。
  + 可以直接去掉，dubbo本身的日志会自动切换为JDK的java.util.logging输出。
  + 但如果其它三方库比如spring.jar间接依赖commons-logging，则不能去掉。
* javassist.jar 字节码生成。
  + 如果<dubbo:provider proxy="jdk" />或<dubbo:consumer proxy="jdk" />，以及<dubbo:application compiler="jdk" />，则不需要。
* spring.jar 配置解析。
  + 如果用ServiceConfig和ReferenceConfig的API调用，则不需要。
* netty.jar 网络传输。
  + 如果<dubbo:protocol server="mina"/>或<dubbo:protocol server="grizzly"/>，则换成mina.jar或grizzly.jar。
  + 如果<protocol name="rmi"/>，则不需要。

DUBBO可选依赖：

以下依赖，在主动配置使用相应实现策略时用到，需自行加入依赖。

* mina: 1.1.7
* grizzly: 2.1.4
* httpclient: 4.1.2
* hessian\_lite: 3.2.1-fixed
* xstream: 1.4.1
* fastjson: 1.1.8
* zookeeper: 3.3.3
* jedis: 2.0.0
* xmemcached: 1.3.6
* jfreechart: 1.0.13
* hessian: 4.0.7
* jetty: 6.1.26
* hibernate-validator: 4.2.0.Final
* zkclient: 0.1
* curator: 1.1.10
* cxf: 2.6.1
* thrift: 0.8.0

项目的Maven的POM文件增加DUBBO的依赖

如果想将此发布包用于你的[Maven](http://maven.apache.org)项目, 请在你项目的Maven的POM文件中加上以下依赖：

如果: Dubbo已发布到Maven中央仓库中: <http://central.maven.org/maven2/com/alibaba/dubbo>

Pom.xml文件如下：

<project>

    <dependencies>

        <dependency>

            <groupId>com.alibaba</groupId>

            <artifactId>dubbo</artifactId>

            <version>2.5.3</version>

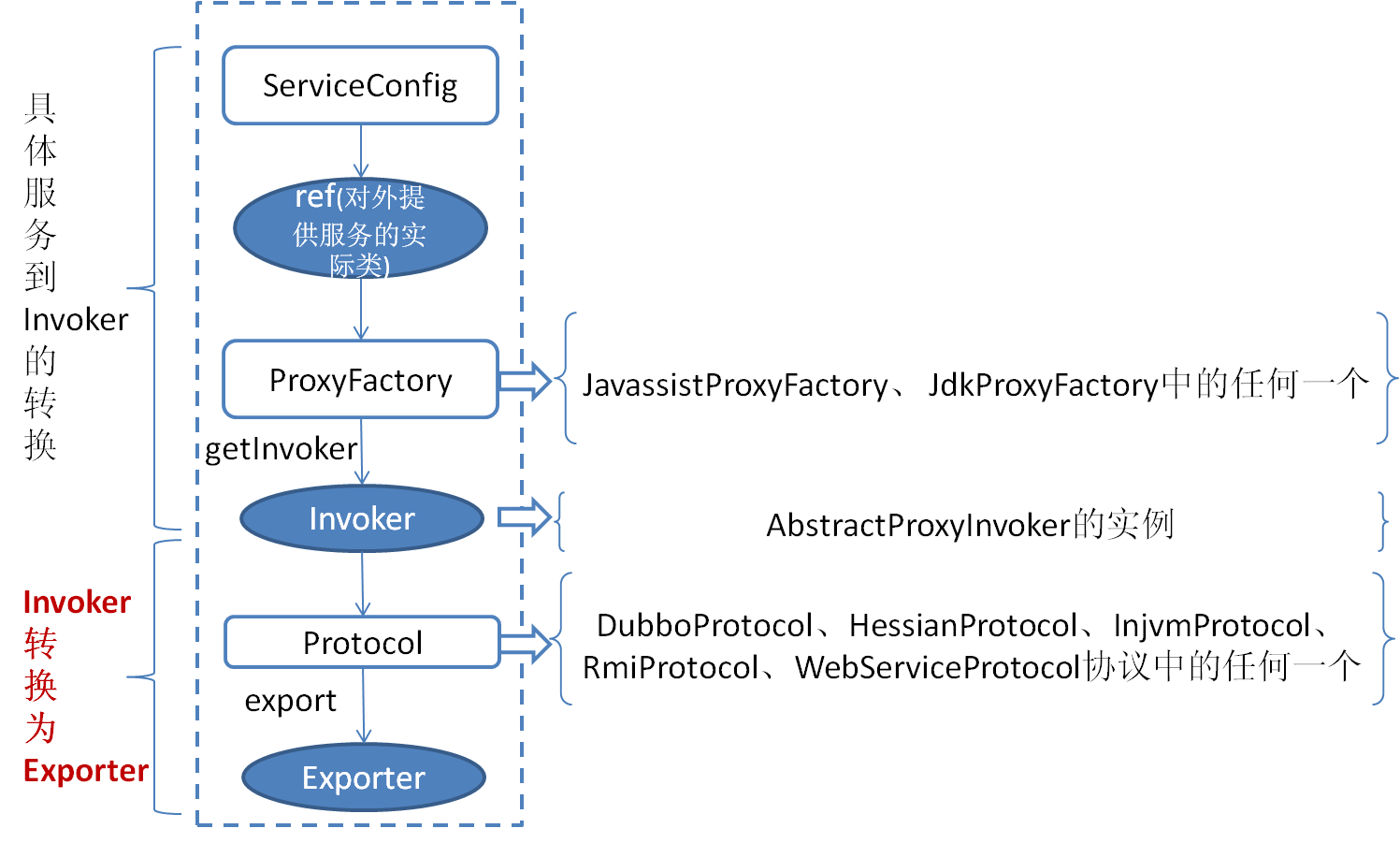
        </dependency>

    </dependencies>

</project>

## 系统间数据流关系

### 服务提供者暴露一个服务的详细过程



图是服务提供者暴露服务的主过程：

首先ServiceConfig类拿到对外提供服务的实际类ref(如：HelloWorldImpl),然后通过ProxyFactory类的getInvoker方法使用ref生成一个AbstractProxyInvoker实例，到这一步就完成具体服务到Invoker的转化。接下来就是Invoker转换到Exporter的过程。

Dubbo处理服务暴露的关键就在Invoker转换到Exporter的过程(如上图中的红色部分)，下面我们以Dubbo和RMI这两种典型协议的实现来进行说明：

* **Dubbo的实现**

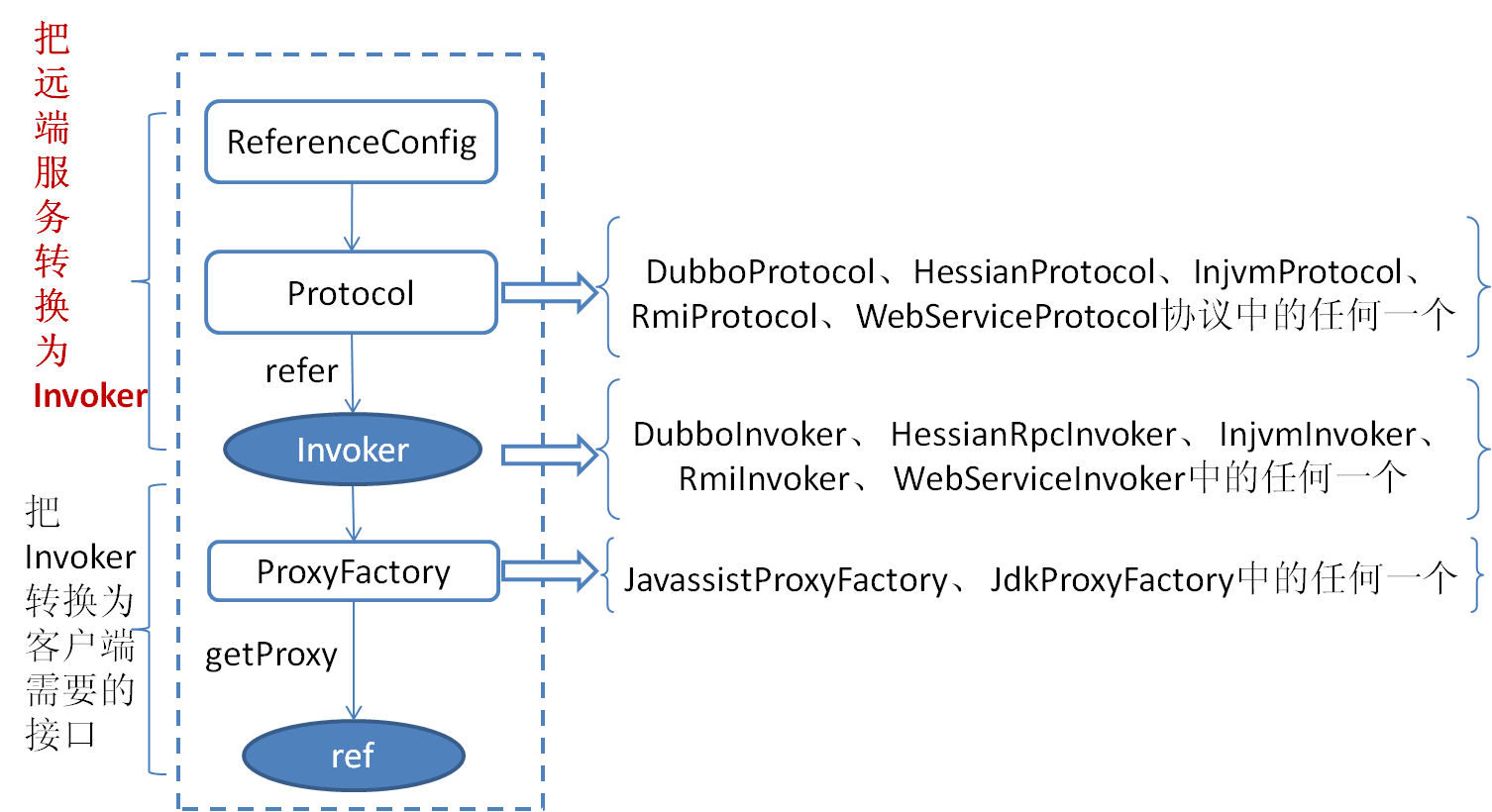
Dubbo协议的Invoker转为Exporter发生在DubboProtocol类的export方法，它主要是打开socket侦听服务，并接收客户端发来的各种请求，通讯细节由Dubbo自己实现。

* **RMI的实现**

RMI协议的Invoker转为Exporter发生在RmiProtocol类的export方法，

它通过Spring或Dubbo或JDK来实现RMI服务，通讯细节这一块由JDK底层来实现，这就省了不少工作量。

### 服务消费者消费一个服务的详细过程



上图是服务消费的主过程：

首先ReferenceConfig类的init方法调用Protocol的refer方法生成Invoker实例(如上图中的红色部分)，这是服务消费的关键。接下来把Invoker转换为客户端需要的接口(如：HelloWorld)。

关于每种协议如RMI/Dubbo/Web service等它们在调用refer方法生成Invoker实例的细节和上一章节所描述的类似。

# 产品环境要求

## 开发环境

### 基础样例

### 开发指南

#### 服务提供者

（1）定义服务接口: (该接口需单独打包，在服务提供方和消费方共享)

**DemoService.java**

|  |
| --- |
| package com.alibaba.dubbo.demo;    public interface DemoService {        String sayHello(String name);    } |

（2）在服务提供方实现接口：(对服务消费方隐藏实现)

**DemoServiceImpl.java**

|  |
| --- |
| package com.alibaba.dubbo.demo.provider;    import com.alibaba.dubbo.demo.DemoService;    public class DemoServiceImpl implements DemoService {        public String sayHello(String name) {          return "Hello " + name;      }    } |

（3）用Spring配置声明暴露服务：

**provider.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans)      xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)      xmlns:dubbo="[http://code.alibabatech.com/schema/dubbo"](http://code.alibabatech.com/schema/dubbo)      xsi:schemaLocation="http://www.springframework.org/schema/beans          http://www.springframework.org/schema/beans/spring-beans.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="hello-world-app"  />        <!-- 使用multicast广播注册中心暴露服务地址 -->      <dubbo:registry address="multicast://224.5.6.7:1234" />        <!-- 用dubbo协议在20880端口暴露服务 -->      <dubbo:protocol name="dubbo" port="20880" />        <!-- 声明需要暴露的服务接口 -->      <dubbo:service interface="com.alibaba.dubbo.demo.DemoService" ref="demoService" />        <!-- 和本地bean一样实现服务 -->      <bean id="demoService" class="com.alibaba.dubbo.demo.provider.DemoServiceImpl" />    </beans> |

（4）加载Spring配置

**Provider.java**

|  |
| --- |
| import org.springframework.context.support.ClassPathXmlApplicationContext;    public class Provider {        public static void main(String[] args) throws Exception {          ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new String[] {"provider.xml"});          context.start();            System.in.read(); // 按任意键退出      }    } |

#### 服务消费者

（1）通过Spring配置引用远程服务

**consumer.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans)      xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)      xmlns:dubbo="[http://code.alibabatech.com/schema/dubbo"](http://code.alibabatech.com/schema/dubbo)      xsi:schemaLocation="http://www.springframework.org/schema/beans          http://www.springframework.org/schema/beans/spring-beans.xsd          http://code.alibabatech.com/schema/dubbo          http://code.alibabatech.com/schema/dubbo/dubbo.xsd          ">        <!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->      <dubbo:application name="consumer-of-helloworld-app"  />        <!-- 使用multicast广播注册中心暴露发现服务地址 -->      <dubbo:registry address="multicast://224.5.6.7:1234" />        <!-- 生成远程服务代理，可以和本地bean一样使用demoService -->      <dubbo:reference id="demoService" interface="com.alibaba.dubbo.demo.DemoService" />    </beans> |

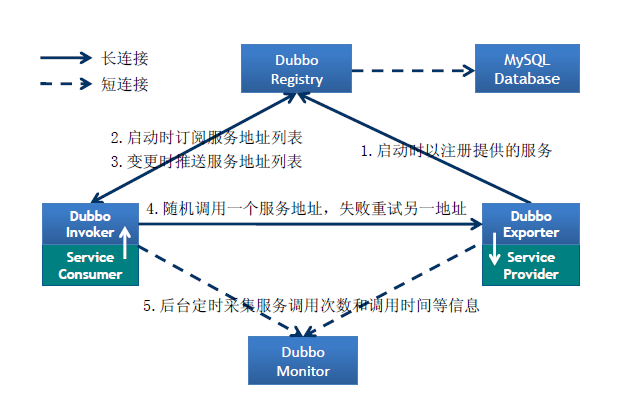
（2）加载Spring配置，并调用远程服务：(也可以使用IoC注入)

**Consumer.java**

|  |
| --- |
| import org.springframework.context.support.ClassPathXmlApplicationContext;  import com.alibaba.dubbo.demo.DemoService;    public class Consumer {        public static void main(String[] args) throws Exception {          ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new String[] {"consumer.xml"});          context.start();            DemoService demoService = (DemoService)context.getBean("demoService"); // 获取远程服务代理          String hello = demoService.sayHello("world"); // 执行远程方法            System.out.println( hello ); // 显示调用结果      }    } |

## 部署环境

### 部署方式



#### 部署DUBBO注册中心（必选）

ZOOKEEPER注册中心安装：

* 建议使用dubbo-2.3.3以上版本的zookeeper注册中心客户端
* [Zookeeper](http://zookeeper.apache.org)是Apache Hadoop的子项目，强度相对较好，建议生产环境使用该注册中心
* Dubbo未对Zookeeper服务器端做任何侵入修改，只需安装原生的Zookeeper服务器即可，所有注册中心逻辑适配都在调用Zookeeper客户端时完成

**安装:**

wget http://www.apache.org/dist//zookeeper/zookeeper-3.3.3/zookeeper-3.3.3.tar.gz

tar zxvf zookeeper-3.3.3.tar.gz

cd zookeeper-3.3.3

cp conf/zoo\_sample.cfg conf/zoo.cfg

**配置：**

vi conf/zoo.cfg

如果需要集群，zoo.cfg的内容如下：(其中data目录和server地址需改成你真实部署机器的信息)

如果不需要集群，zoo.cfg的内容如下：(其中data目录需改成你真实输出目录)

**zoo.cfg**

|  |
| --- |
| tickTime=2000  initLimit=10  syncLimit=5  dataDir=/home/dubbo/zookeeper-3.3.3/data  clientPort=2181 |

如果需要集群，zoo.cfg的内容如下：(其中data目录和server地址需改成你真实部署机器的信息)

**zoo.cfg**

|  |
| --- |
| tickTime=2000  initLimit=10  syncLimit=5  dataDir=/home/dubbo/zookeeper-3.3.3/data  clientPort=2181  server.1=10.20.153.10:2555:3555  server.2=10.20.153.11:2555:3555 |

并在data目录下放置myid文件：(上面zoo.cfg中的dataDir)

|  |
| --- |
| mkdir data  vi myid |

myid指明自己的id，对应上面zoo.cfg中server.后的数字，第一台的内容为1，第二台的内容为2，内容如下：

**myid**

|  |
| --- |
| 1 |

#### 部署DUBBO监控中心（可选）

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:**

|  |
| --- |
| wget http://apache.etoak.com/tomcat/tomcat-6/v6.0.35/bin/apache-tomcat-6.0.35.tar.gz  tar zxvf apache-tomcat-6.0.35.tar.gz  cd apache-tomcat-6.0.35  rm -rf webapps/ROOT |

|  |
| --- |
| wget http://code.alibabatech.com/mvn/releases/com/alibaba/dubbo-admin/2.4.1/dubbo-admin-2.4.1.war  unzip dubbo-admin-2.4.1.war -d webapps/ROOT |

**配置:** (或将dubbo.properties放在当前用户目录下)

|  |
| --- |
| vi webapps/ROOT/WEB-INF/dubbo.properties |

**dubbo.properties**

|  |
| --- |
| dubbo.registry.address=**zookeeper://127.0.0.1:2181**  dubbo.admin.root.password=root  dubbo.admin.guest.password=guest |

**启动:**

|  |
| --- |
| ./bin/startup.sh |

**停止:**

|  |
| --- |
| ./bin/shutdown.sh |

**访问:** (用户:root,密码:root 或 用户:guest,密码:guest)

|  |
| --- |
| <http://127.0.0.1:8080/> |

#### 部署服务提供者

把服务提供者安装JAVA程序的部署方式进行部署。

如果注册中心是ZOOKEEPER， 把dubbo.registry.address的值改为zookeeper://127.0.0.1:2181 （IP地址配置为应用所在的主机IP地址）。

#### 部署服务消费者

把服务消费者安装JAVA程序的部署方式进行部署。

如果注册中心是ZOOKEEPER，把dubbo.registry.address的值改为zookeeper://127.0.0.1:2181 （IP地址配置为应用所在的主机IP地址）。

### 运行及监控方法

需要管理Dubbo的服务时，首先要搜索到这个服务，然后打开它的管理页面：



#### 服务提供者页面



#### 服务消费者页面



#### 服务应用页面



#### 添加路由规则页面



#### 添加动态配置页面

