1. 软件架构的演变

软件架构的发展经历了由单体架构，垂直架构，SOA架构到微服务架构的演变过程。

1. 单体架构
2. 概述

单体架构就是将所有的功能都放在一个项目中。

1. 优点

单体架构结构简单，前期的开发成本低，开发周期短，适合小型项目。

1. 缺点

（1）对于大型项目来说，全部的功能都集成在一个工程中，不易开发，拓展和维护。

（2）技术栈受限，只能使用一种语言开发

（3）系统性能扩展只能通过扩展集群节点，成本比较高。

1. 垂直架构
2. 概述

所谓的垂直架构就是按照业务的逻辑将原有的项目切割为多个小的单体项目。

1. 优点

垂直架构通过切割之后，每个小的单体项目可以使用不同的技术栈。

1. 缺点
2. 所有功能依然集中在一个项目中，不利于开发，扩展，维护
3. 系统扩张只能通过集群
4. 项目之间功能冗余，耦合性强
5. SOA架构
6. 概述

SOA的全程为Service-Oriented Architecture，即面向服务的架构。它可以根据需求通过网络对松散耦合的粗粒度应用组件（服务）进行分布式部署，组合和使用。一个服务通常以独立的形式存在于操作系统进程中。

服务都是可复用的，通过服务的编排可以实现业务的快速再生，把原先固有的业务功能转变为通用的业务服务，实现业务逻辑的快速复用。在项目和服务之间使用ESB（企业服务总线）的形式作为通信的桥梁。

1. 优点
2. 重复功能或模块抽取为服务，提高开发效率
3. 可重用性高，任何业务都可以调用通用的服务
4. 可维护性高，维护方便，便于定位错误位置
5. 缺点
6. 在需求分析时，由于各系统之间业务不同，很难确认功能或模块是重复的
7. 抽取服务的粒度较大
8. 系统和服务之间耦合度高
9. 微服务架构
10. 概述

微服务架构与SOA的区别是服务粒度更细，且服务层完全独立出来，抽取为一个一个的微服务。用户在前端界面中发出请求，通过服务网关，向服务层的各个微服务进行请求，微服务再向数据层进行查询。

1. 优点
2. 服务拆分粒度更细，有利于提高开发效率
3. 可以针对不同服务制定对应的优化方案
4. 产品迭代周期更短
5. 缺点

粒度太细导致服务太多，维护成本较高。

1. Apache Dubbo
2. 概述
3. Dubbo概述

Apache Dubbo 是一款高性能的Java RPC框架，前身是阿里巴巴公司开源的一个高性能，轻量级的开源Java RPC框架，可以和Spring框架无缝集成。

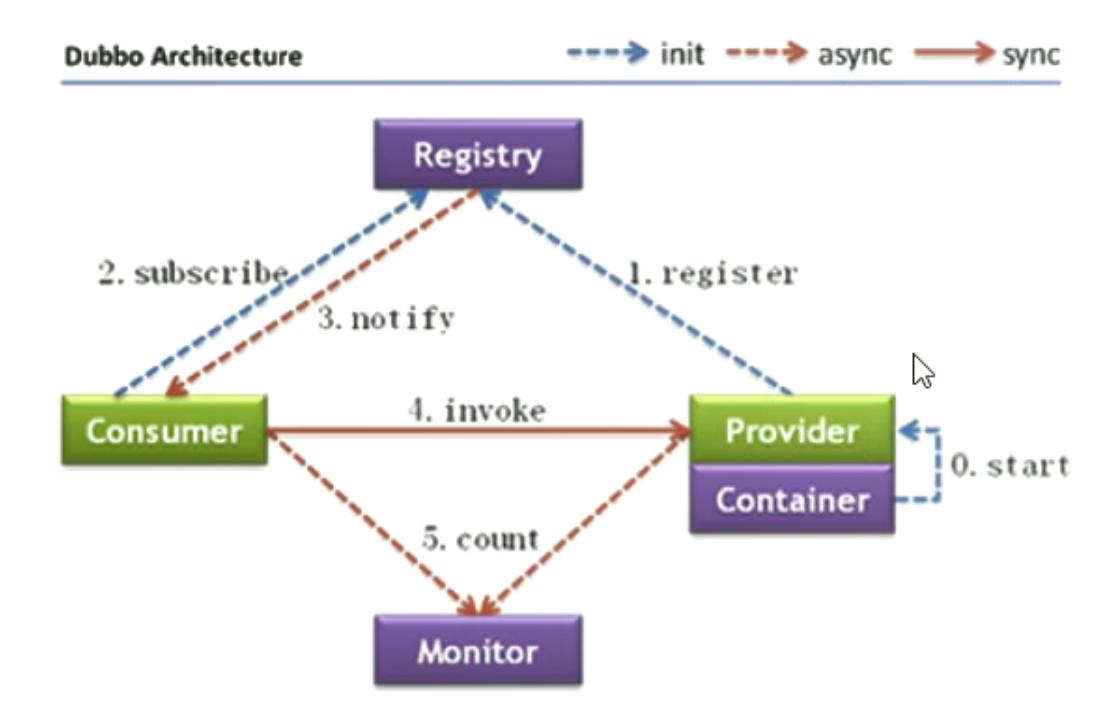
1. RPC概述

RPC全称为Remote Procedure Call，即远程过程调用，它不是一个具体的技术，而是一种网络远程调用过程概念。如两台服务器上部署着不同的应用A和B，A想要调用B应用的方法，因此需要通过网络来表达调用的语义和传达调用的数据。

各种语言都有自己的RPC框架，Java中的RPC框架较多，广泛使用的有RMI，Hessian，Dubbo等。Dubbo提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

1. Dubbo架构

Dubbo的架构图如下所示：



1. 节点说明

Provider 服务的提供方

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 统计服务的调用次数和调用时间的监控中心

Container 服务运行容器

1. 关联分析
2. 服务提供方在容器中启动服务，并且将服务注册到注册中心中。
3. 消费者要先调用该服务，首先要想注册中心订阅该服务，注册中心将服务提供者地址列表返回给消费者，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. 消费者从服务提供者的地址列表中，基于软负载均衡算法，选择一台提供者进行调用，如果调用失败，再选另一台进行调用。
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。
6. 注意事项

架构图片中的虚线代表着异步访问，实现代表着同步访问，只有消费者调用服务是同步访问的，其他都是异步访问的。

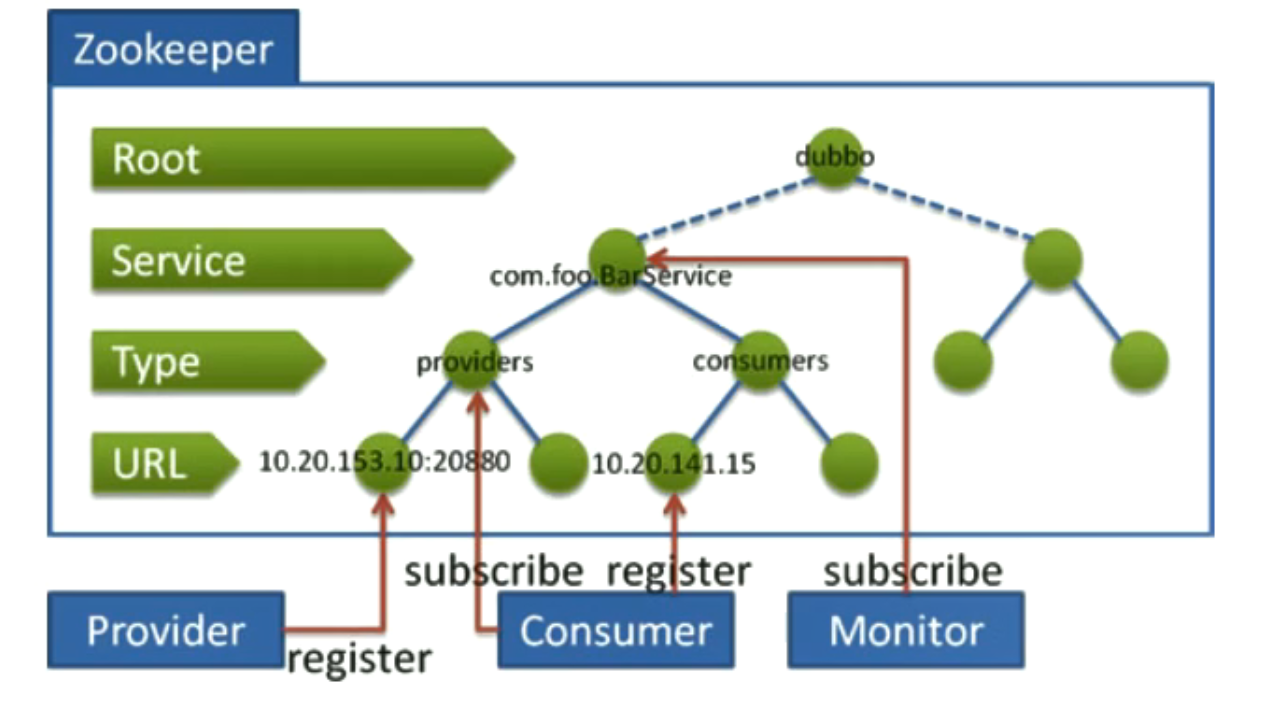
蓝色虚线代表着在启动时完成的功能，红色虚线（实现）代表着程序在运行过程中执行的功能。

1. 服务注册中心Zookeeper
2. 概述

Zookeeper是Apache Hadoop的子项目，是一个树形的目录服务，支持变更推送，适合作为Dubbo的注册中心。

1. Zookeeper的架构

Zookeeper的属性目录架构如下图所示：



可以看到，根节点为dubbo，服务层节点就是一个个的服务，每个服务下面又分为了服务提供者节点和服务消费者节点，每个提供者和消费者下又分为了多个URL地址，表示着多个对应的地址。

这样，服务提供者在服务启动时，就可以在对应的服务的提供者节点下注册自己的URL地址；

服务消费者在启动时，向注册中心订阅需要的服务，即订阅该服务节点下的提供者节点的URL地址，同时在该服务的消费者节点下写入自己的URL地址；

监控中心在启动时，需要订阅已启动服务节点下的所有提供者和消费者的URL地址。

1. Zookeeper的安装
2. 从官网下载Zookeeper的压缩包之后，将压缩包上传到linux系统中。
3. 解压缩压缩包

tar -zxvf zookeeper-3.4.6.tar.gz

1. 进入zookeeper-3.4.6目录，创建data目录，作为数据目录

mkdir data

1. 进入conf目录，把zoo\_sample.cfg改名为zoo,cfg

cd conf

mv zoo\_sample.cfg zoo.cfg

1. 打开zoo.cfg文件，修改data属性为刚创建的data目录路径

dataDir=/usr/local/zookeeper-3.4.6/data

1. Zookeeper的启动

进入Zookeeper安装目录/bin/zkServer.sh文件，后面接start表示启动服务：

./zkServer.sh start

使用 status 表示查看服务状态：

./zkServer.sh status

使用 stop表示停止服务

./zkServer.sh stop

1. Dubbo入门案例
2. 案例概述

本案例要创建两个应用，一个作为服务的提供方，一个作为服务的消费方，通过Dubbo来实现服务消费方远程调用服务提供方的方法。

1. 服务提供方开发
2. 创建Maven工程dubboDemo\_Provider

创建一个新的Maven工程，由于这是一个web项目，因此我们可以使用webapp的骨架进行搭建Maven项目。

1. 配置pom.xml

在pom.xml中导入相关的坐标，如spring相关的坐标，dubbo相关的坐标，以及数据库相关的坐标。

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</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-webmvc</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-aspects</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jms</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context-support</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <!-- dubbo相关 -->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.6.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.zookeeper</groupId>  
 <artifactId>zookeeper</artifactId>  
 <version>3.4.7</version>  
 </dependency>  
 <dependency>  
 <groupId>com.github.sgroschupf</groupId>  
 <artifactId>zkclient</artifactId>  
 <version>0.1</version>  
 </dependency>  
 <dependency>  
 <groupId>javassist</groupId>  
 <artifactId>javassist</artifactId>  
 <version>3.12.1.GA</version>  
 </dependency>  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.47</version>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.11</version>  
 </dependency>  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.1.6</version>  
 </dependency>  
 <dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis-spring</artifactId>  
 <version>1.3.2</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.11</version>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

1. 配置web.xml

这是一个web项目，因此我们需要配置web.xml文件，添加一个监听器，当web项目启动时，自动读取配置文件application.xml，配置文件的地址通过<context-param>标签给出。

<!DOCTYPE web-app PUBLIC  
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
 "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  
  
<web-app>  
 <display-name>Archetype Created Web Application</display-name>  
  
 <context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:application\*.xml</param-value>  
 </context-param>  
 <listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
 </listener>  
</web-app>

1. 创建服务接口

创建一个HelloService接口，声明一个方法，同时创建它的实现类HelloServiceImpl，实现接口的方法，并且在该实现类的上面用@Service注解标注，注意这里使用的@Service注解不是spring框架中的注解，而是Dubbo提供的@Service注解，因为要发布服务必须使用Dubbo提供的@Service注解。

package com.dubbo.service.impl;  
  
import com.alibaba.dubbo.config.annotation.Service;  
import com.dubbo.service.HelloService;  
  
@Service //发布服务使用Dubbo提供的Service注解  
public class HelloServiceImpl implements HelloService {  
  
 @Override  
 public String sayHello(String name) {  
 return "hello " + name;  
 }  
}

1. 在src/main/resources下创建applicationContext-service.xml

这是spring服务的配置文件，我们可以在其中配置dubbo的各种设置。

dubbo的应用名称，如现在需要配置的是服务提供方的应用名称dubboDemo\_Provider：

<dubbo:application name=”dubboDemo\_Provider”/>

注册中心的地址，地址中的zookeeper为固定写法

<dubbo:registry address=”zookeeper://192.168.134.129:2181”/>

注册dubbo的协议和端口号，端口号默认是20880

<dubbo:protocol name=”dubbo” port=”20880”></dubbo:protocol>

扫描指定的包，那些加入了@Service注解的类会被发布为服务

<dubbo:annotation package=”com.dubbo.service.impl”>

1. 服务消费方开发
2. 创建Maven项目

同理，依然是依据webapp骨架创建一个Maven项目。

1. 配置pom.xml

配置情况与服务提供方相同，不过注意在Tomcat插件的端口处要和服务提供方不同。

1. 配置web.xml

服务消费方的web.xml配置和服务提供方是不同的，消费方是要调用服务提供方的服务的，因此配置的是SpringMVC中的前端控制器。

这里以servlet作为前端控制器，当有以do为后缀的请求时，servlet将请求转发到对应的Controller上，并且指定服务启动时加载的配置文件。

<!DOCTYPE web-app PUBLIC  
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
 "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  
  
<web-app>  
 <display-name>Archetype Created Web Application</display-name>  
  
 <servlet>  
 <servlet-name>springmvc</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:applicationContext-web.xml</param-value>  
 </init-param>  
 <load-on-startup>1</load-on-startup>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>springmvc</servlet-name>  
 <url-pattern>\*.do</url-pattern>  
 </servlet-mapping>  
</web-app>

1. 将服务提供方的HelloService接口复制到当前的服务消费方中

直接复制服务提供方的接口。

1. 编写Controller

服务消费方使用过Controller来调用服务提供方的服务方法的，因此创建一个Contoller的类，远程调用服务的方法sayHello。

这里，我们按照往常一样在类上设置@Controller注解和@RequestMapping注解。并且注入要调用的服务接口 HelloService，但是这里不能使用@Autowired自动注入，因为远程调用服务提供方的服务，该服务不在内存中，自然不能自动注入。需要使用Dubbo提供的@Reference注解来引用注册中心中的服务。

package com.dubbo.controller;  
  
import com.alibaba.dubbo.config.annotation.Reference;  
import com.dubbo.service.HelloService;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
@Controller  
@RequestMapping(".demo")  
public class HelloController {  
  
 @Reference  
 private HelloService helloService;  
  
 @RequestMapping("/hello")  
 @ResponseBody  
 public String getName(String name){  
 //远程调用  
 String result = helloService.sayHello(name);  
 System.*out*.println(result);  
 return result;  
 }  
}

1. 配置applicationContext-web.xml

同理，需要配置spring web层的配置文件，同样是配置应用名称，链接注册中心的地址，这个地址是我们在linux服务器中启动的，以及扫描包。

<?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:p="http://www.springframework.org/schema/p"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://code.alibabatech.com/schema/dubbo  
 http://code.alibabatech.com/schema/dubbo/dubbo.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd">  
 <dubbo:application name="dubboDemo\_Consumer"></dubbo:application>  
 <dubbo:registry address="zookeeper://192.168.211.128:2181"></dubbo:registry>  
 <dubbo:annotation package="com.dubbo.controller"></dubbo:annotation>  
 <dubbo:consumer check="false"></dubbo:consumer>  
</beans>

1. 测试

注册中心zookeeper服务启动后，还要启动服务提供方的项目，服务消费方的项目，如果项目的启动都没有问题的话，访问服务消费方的地址，并且访问Controller的方法，并且添加参数name:

<http://localhost:8084/demo/hello.do?name=wwxx>

得到页面的显示结果为：

hello wwxx

测试成功！！！

可能会出现Zookeeper连接不上的问题，首先要检查Linux中安装的Zookeeper的版本与项目中导入的版本是否一致；然后再检查Ubuntu 中是否关闭了防火墙，如果没有，需要将防火墙关闭！！！

1. 思考
2. 之前直接将服务提供方的接口复制到服务消费方中，这是一个不好的做法。更好的解决方法是单独创建一个Maven工程，将该接口创建在这个Maven工程中。需要依赖此接口的工程只需要在自己的工程pom.xml中引入maven坐标即可。
3. 服务消费方只是引用了接口，并没有提供实现类，Dubbo是如何远程调用的？

Dubbo底层是基于代理技术为接口创建代理对象，远程调用是通过此代理对象完成的。可以通过开发工具的debug功能查看此代理对象的内部结构，另外，Dubbo实现网络传输底层是基于Netty框架实现的。

1. 如何防止Zookeeper单点故障？

Zookeeper是支持集群模式的，可以配置Zookeeper集群来达到Zookeeper服务的高可用性，防止出现单点故障。

1. Dubbo管理控制台
2. 概述

在开发中，我们需要知道Zookeeper注册中心都注册了哪些服务，有哪些消费者来消费这些服务，我们可以通过部署一个管理中心来实现。这个管理中心是一个web应用，部署在Tomcat服务器上。

1. 安装步骤
2. 将下载好的 dubbo-admin-2.6.0.war文件复制到本地Tomcat安装文件夹的webapps目录下。
3. 在Tomcat的安装目录下的bin目录中，点击startup.bat批处理文件，启动Tomcat，这个Dubbo管理控制台的war包会自动被解压成文件夹。
4. 在同名文件夹中，修改WEB-INF目录下的dubbo.properties文件，将Zookeeper的ip地址改为当前使用的Zookeeper的ip地址和端口号。

dubbo.registry.address=zookeeper://192.168.211.128:2181

dubbo.admin.password=root

dubbo.admin.guest.password=guest

注意，这里有两个用户，一个是root用户，一个是guest用户，root用户的密码为root，guest用户的密码为guest。

1. 重新启动Tomcat
2. Dubbo相关配置说明
3. 包扫描

服务提供者和服务消费者都需要配置包扫描，作用是扫描指定包（包括子包）下的类。

<dubbo:annotation package=”com.dubbo.service”>

如果不适用包扫描，也可以使用配置的方式来发布服务。如

服务提供方通过注册服务：

<bean id=”helloService” class=”com.dubbo.service.impl.HelloServiceImpl”/>

<dubbo:service interface=”com.dubbo.service.HelloService” ref=”helloService”>

服务消费方，通过配置引用服务：

<dubbo:reference id=”helloService” interface=”com.dubbo.service.helloService”/>

这种通过配置来发布和引用服务的做法，一个配置项（dubbo:service，dubbo:reference）只能发布或者引用一个服务，如果有多个服务，这种方式就非常地繁琐，这里推荐使用地是包扫描方式。

1. 协议

在服务提供者的配置中，可以指定使用的协议名称和端口号：

<dubbo:protocal name=”dubbo” port=”20880”/>

Dubbo支持的协议有：dubbo, rmi, hessian, http, webservice, rest, redis等，官方推荐使用的是dubbo协议。

dubbo协议采用单一长连接和NIO异步通信，是和于小数据量大并发的服务调用，以及服务者机器数远大于服务提供者机器数的情况，不适合传送大数据量的服务，比如传文件，视频。

我们也可以在同一个工程中配置多个协议，在不同服务中可以使用不同的协议，直接通过注解设置或者在配置文件中设置：

注解设置多协议：

在服务提供方的服务注解@Service中设置protocol属性即可。

配置文件设置多协议：

<dubbo:protocol name=”dubbo” port=”20880”/>

<dubbo:protocol name=”rmi” port=”1099”/>

<dubbo:service interface=”com.dubbo.service.HelloService” ref=”helloService” protocol=”dubbo”/>

<dubbo:service interface=”com.dubbo.service.DemoService” ref=”demoService” protocol=”rmi”/>

1. 启动时检查

我们之前是先启动服务提供方的服务器，再启动服务消费方的服务器，这样服务消费方能够检测到服务提供方注册的服务。若我们将两者的启动顺序调换，就会出错，这是服务消费方的启动时检查的结果，它会检查依赖的服务时是否可用，不可用时会抛出异常，阻止Spring初始化完成。

为了尽快上线，及早地发现问题，我们在开发阶段可以关闭启动时检查，在生产环境下改为true。

<dubbo:consumer check=”false”/>

1. 负载均衡

负载均衡是将请求分摊到多个操作单元上进行执行，从而共同完成工作任务。在集群负载均衡时，Dubbo提供了多种均衡策略，包括危机，轮训，最少活跃调用数，一致性Hash，默认为随机调用。

配置负载均衡策略，既可以在服务提供者一方进行配置，也可以在服务消费者一方进行配置，如：

在服务消费者一方配置负载均衡

@Controller

@RequestMapping(“/demo”)

public class HelloController{

@Reference(check=false,loadbalance=”random”)

private HelloService helloService;

@RequestMapping(“/hello”)

@ResponseBody

public String getName(String name){

String result = helloService.sayHello(name);

System.out.println(result);

return result;

}

}

在服务提供者一方配置负载均衡

@Service(loadbalance=”random”)

public class HelloServiceImpl implements HelloService{

public String sayHello(String name){

return “hello” + name;

}

}

1. 解决Dubbo无法发布被事务代理的Service问题

如果在服务提供者类上添加 @Transactional事务控制注解后，服务就发布失败了，原因是事务控制的底层原理是为服务提供者创建代理对象，而默认情况下Spring是基于JDK 动态代理方式创建代理对象，而此代理对象的全类名为 com.sun.proxy.$Proxy42，导致Dubbo在发布服务前进行包匹配时无法完成，因为我们在发布服务时扫描的包围com.dubbo.service，所以真正发布服务的代码未执行。

解决方法：

1. 修改applicationContext-service.xml配置文件

开启事务控制注解支持时指定proxy-target-class属性，值为true，其作用是使用cglib代理方式为Service类创建代理对象。

<tx::annotation-driven transaction-manager=”transactionManager” proxy-target-class=”true”/>

1. 修改服务提供方的实现类

在@Service注解中加入interfaceClass属性，值为HelloService.class，作用是指定服务的接口类型，这样，服务提供方就会知道创建代理对象的接口为HelloService了，而不是SpringProxy，该服务能够发布成功。

服务消费方：

@Service(interfaceClass=helloService.class)

@Transactional

public class HelloSericeImpl implements implements HelloService{

public String sayHello(String name){

return “hello” + name;

{

}