分布式RPC框架Apache Dubbo

学习目标

- [ ] 了解软件架构的演进过程
- [ ] 掌握Dubbo框架的架构[重点-面试]
- [ ] 能够使用命令启动和停止Zookeeper
- [ ] 掌握Dubbo服务提供者和消费者开发[重点]
- [ ] 了解Dubbo管理控制台dubbo-admin

第一章-软件架构的演进过程

知识点-软件架构的演进过程

1.目标

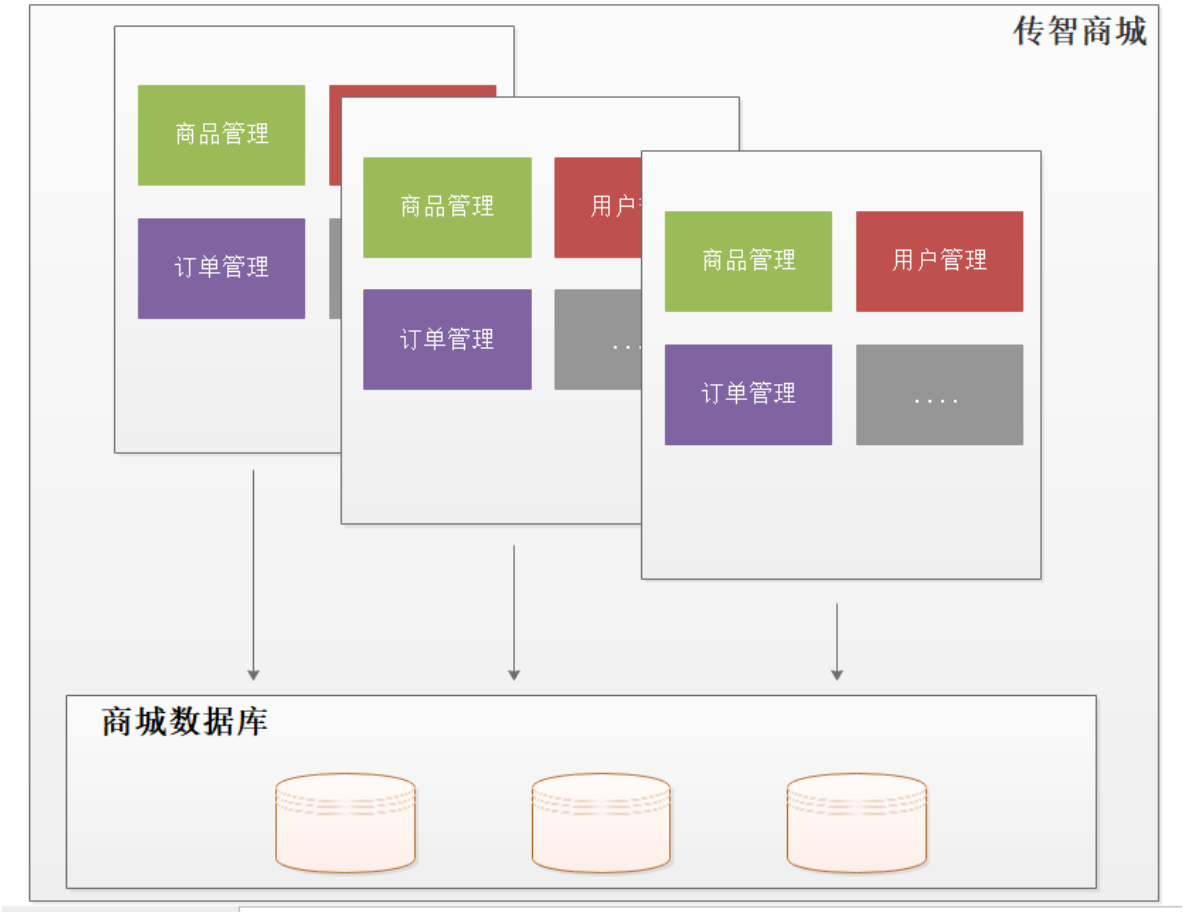
软件架构的发展经历了由单体架构、垂直架构、【SOA架构,微服务】架构的演进过程.我们需要对软件架构的演进过程有一定的了解.

2.路径

- 单体架构
- 垂直架构
- SOA架构
- 微服务架构

3.讲解

3.1单体架构



1. 架构说明

   全部功能集中在一个项目里面(All in one).

2. 架构优点

   项目架构简单，前期开发成本低，周期短，小型项目的首选。

3. 架构缺点

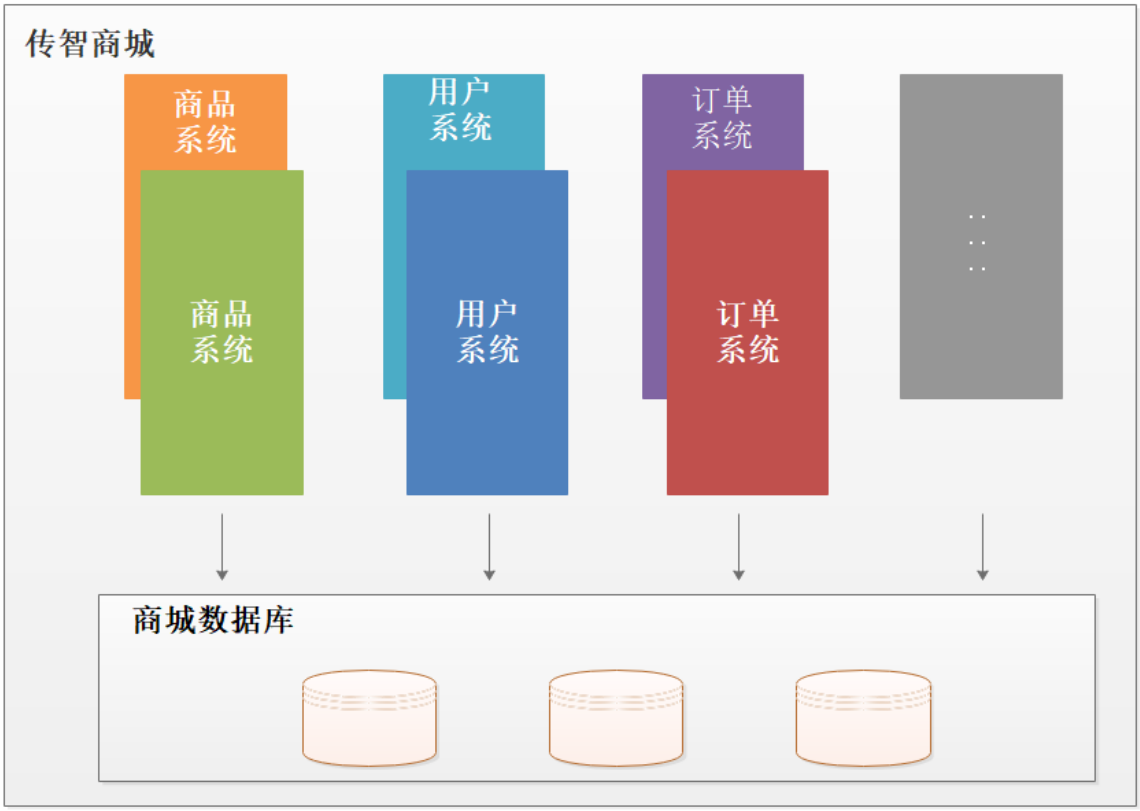
   耦合度比较高,不好维护和扩展

   对于大项目,部署后性能比较低

   技术栈受限,只能使用一种语言开发

   解决高并发只能通过集群,成本高

3.2垂直架构



随着互联网的发展，用户越来越多，软件技术也得到了很大的发展，人们开始研究一些技术使其与底层硬件交互会更加友好等。及某系统流量访问某模块占比很高，而其他模块没有什么流量访问，如果都部署到一起占用的资源就浪费了，如果分开部署，流量高的部署到一台高性能服务器，而流量低的部署到一台普通的服务器，两个模块之间的交互用WebService、RPC等方式进行访问。那样就可以解决上述传统架构的缺点问题.

1. 架构说明

   按照业务进行切割，形成小的项目，项目直接通过RPC等方式通信，交换数据等。

2. 架构优点

   耦合度降低

   技术不会受限制,不同系统可以使用不同语言开发

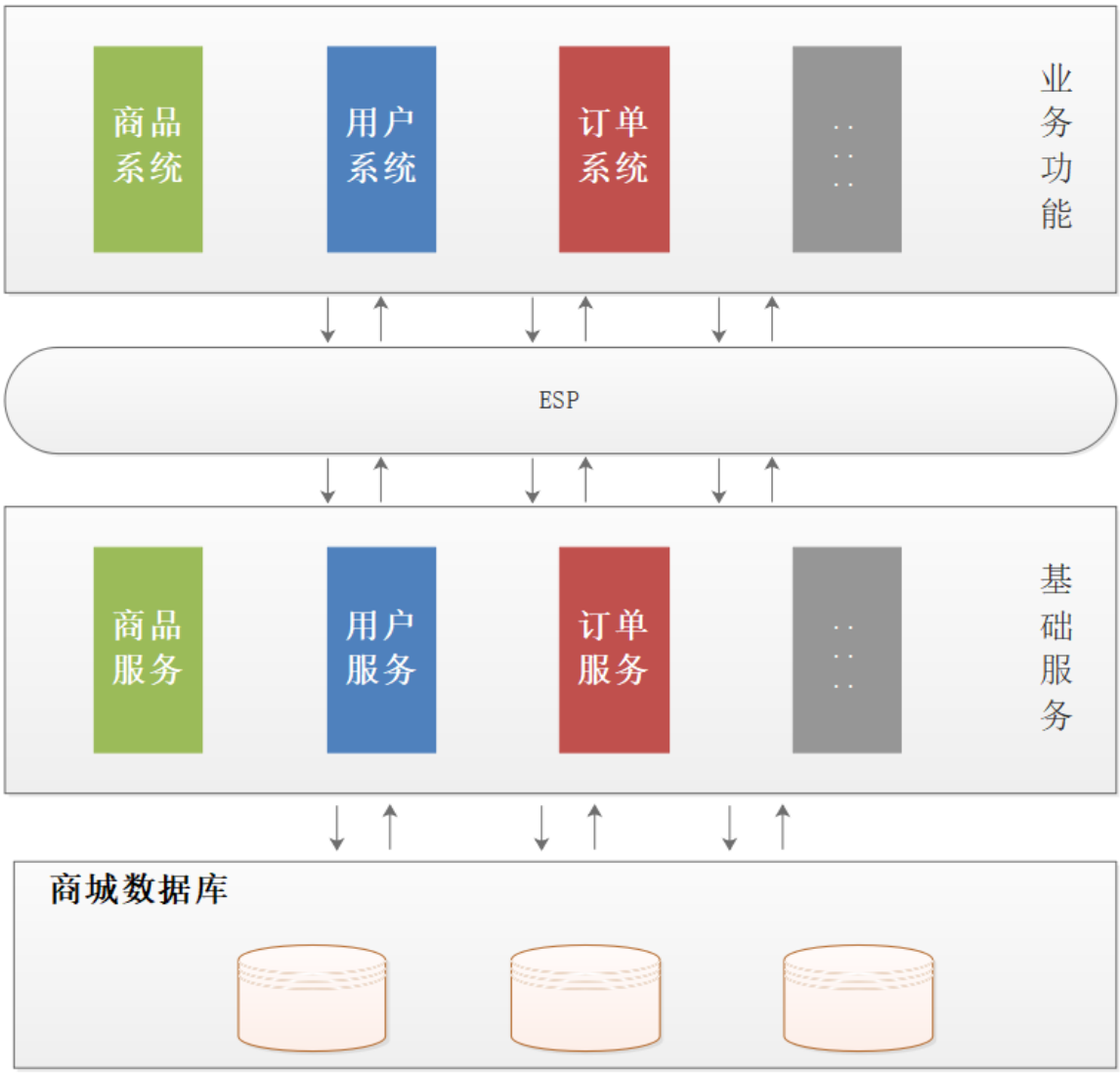
   项目不会无限扩大

3. 架构缺点

   项目间公共的逻辑重复,没办法复用

   界面和业务逻辑没有分离

3.3SOA架构



在垂直架构中可以看到,项目间公共的逻辑重复,没办法复用. 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.

SOA是Service-Oriented Architecture的首字母简称，它是一种支持面向服务的架构样式。从服务、基于服务开发和服务的结果来看，面向服务是一种思考方式。其实SOA架构更多应用于互联网项目开发。

1. 架构说明

   将重复功能或模块抽取成组件的形式，对外提供服务，在项目与服务之间使用ESB（企业服务总线）的形式作为通信的桥梁，使用RPC等方式进行通信。

2. 架构优点

   重复功能或模块抽取为服务，提高开发效率、可重用性高、可维护性高

   可以针对不同服务制定对应的技术方案。

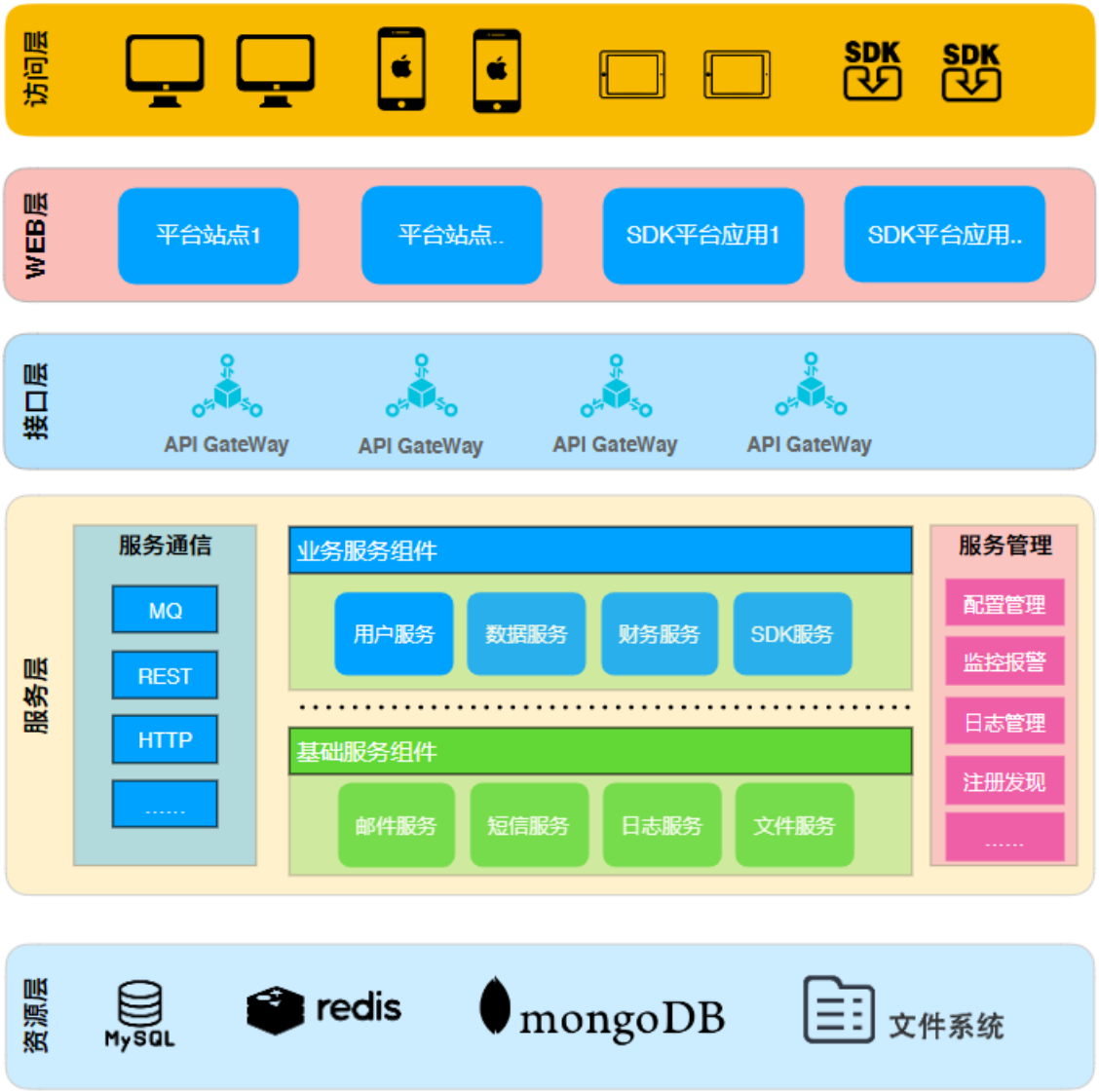
   界面和业务逻辑实现分离

3. 架构缺点

   各系统之间业务不同，因此很难确认功能或模块是重复的，不利于开发和维护

   抽取服务的粒度大

3.4微服务架构



SOA架构有局限性，就是所有的接口都需要走ESB，如果不同的编程语言开发子系统，而这个编程语言对于某种RCP协议支持是最友好的，而ESB规则限定其只能使用ESB的规定协议。

1. 架构说明

   在服务治理架构上延伸，抽取的粒度更细，尽量遵循单一原则，采用轻量级框架协议(HTTP协议)传输。

2. 架构优点

   - 去中心化的思想，不在使用ESB作为通信的桥梁，服务、系统之间可以相互访问。
   - 粒度更细，有利于提高开发效率。
   - 可以针对不同服务制定对应的技术方案。
   - 适用于产品迭代周期短

3. 架构缺点

   - 粒度太细导致服务太多，维护成本高。
   - 负载均衡、事务等问题对技术团队的挑战及成本问题。 分布式事务

4.小结

1. 单体架构: 所有的模块全部写在一个项目里面
2. 垂直架构: 根据业务抽取成一个个系统
3. SOA架构: 把业务抽取成一个个服务(war)

4. 微服务: 前后端分离, 后台只开发微服务, 分库

第二章-Apache Dubbo

知识点-Dubbo概述

1.目标

- 知道什么是Dubbo以及RPC

2.路径

- Dubbo介绍
- RPC介绍

3.讲解

3.1Dubbo介绍

Apache Dubbo是一款高性能的Java RPC框架。其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。

Dubbo官网地址：http://dubbo.apache.org

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。

3.2RPC介绍

RPC全称为remote procedure call，即远程过程调用。

比如两台服务器A和B，A服务器上部署一个应用，B服务器上部署一个应用，A服务器上的应用想调用B服务器上的应用提供的方法，由于两个应用不在一个内存空间，不能直接调用，所以需要通过网络来表达调用的语义和传达调用的数据

RPC是一个泛化的概念，严格来说一切远程过程调用手段都属于RPC范畴。各种开发语言都有自己的RPC框架。Java中的RPC框架比较多，广泛使用的有RMI、Hessian、Dubbo等。

> 需要注意的是RPC并不是一个具体的技术, 不是协议，而是指整个网络远程调用过程。

4.小结

1. Dubbo: 是一个RPC的java框架.
2. RPC: 远程调用的过程, 是一种方式, 不是协议

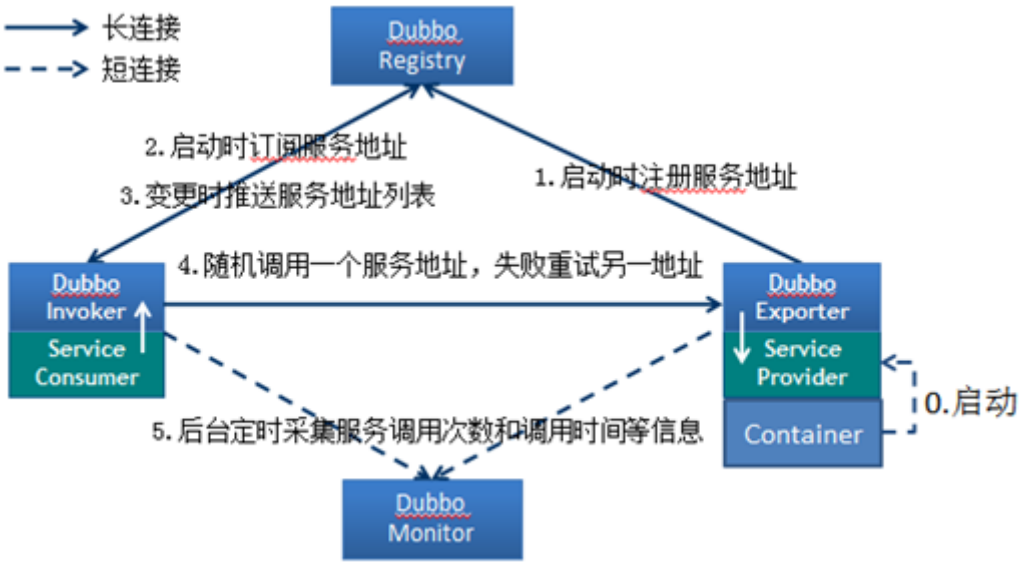
知识点-Dubbo架构[重点-面试]

1.目标

- 掌握Dubbo架构

2.讲解

Dubbo架构图（Dubbo官方提供）如下：



节点角色说明：

节点	角色名称
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	统计服务的调用次数和调用时间的监控中心
Container	服务运行容器

虚线都是异步访问，实线都是同步访问
蓝色虚线:在启动时完成的功能
红色虚线(实线)都是程序运行过程中执行的功能

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。

3.小结

1. 注册中心, 服务的提供者(service), 服务的消费者(controller)
2. 可以把服务的提供者(service), 服务的消费者(controller)都注册到注册中心里面
3. 服务的消费者(controller)可以进行订阅, 从注册中心里面获得服务的提供者(service)的信息, 就可以进行远程调用了

知识点-服务注册中心Zookeeper

1.目标

通过前面的Dubbo架构图可以看到，Registry（服务注册中心）在其中起着至关重要的作用。Dubbo官方推荐使用Zookeeper作为服务注册中心。

2.路径

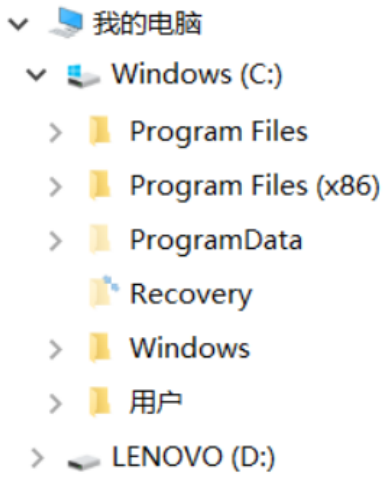
- Zookeeper介绍
- 安装Zookeeper
- 启动、停止Zookeeper

3.讲解

3.1什么是Zookeeper

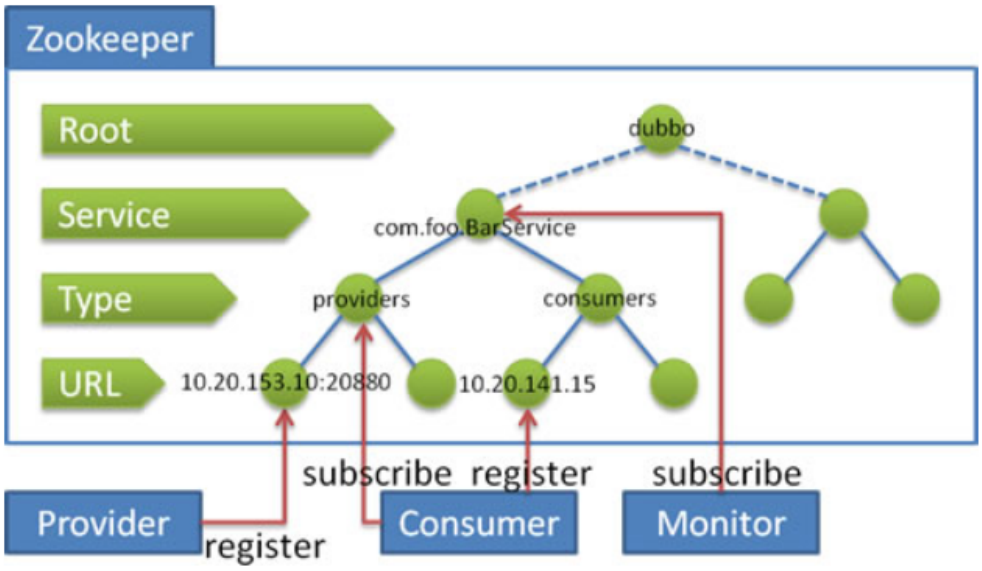
Zookeeper 是 Apache Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用 。

为了便于理解Zookeeper的树型目录服务，我们先来看一下我们电脑的文件系统(也是一个树型目录结构)：



我的电脑可以分为多个盘符（例如C、D、E等），每个盘符下可以创建多个目录，每个目录下面可以创建文件，也可以创建子目录，最终构成了一个树型结构。通过这种树型结构的目录，我们可以将文件分门别类的进行存放，方便我们后期查找。而且磁盘上的每个文件都有一个唯一的访问路径，例如：C:\Windows\itcast\hello.txt。

Zookeeper树型目录服务：



流程说明：

- 服务提供者(Provider)启动时: 向 `/dubbo/com.foo.BarService/providers` 目录下写入自己的 URL 地址
- 服务消费者(Consumer)启动时: 订阅 `/dubbo/com.foo.BarService/providers` 目录下的提供者 URL 地址。并向 `/dubbo/com.foo.BarService/consumers` 目录下写入自己的 URL 地址
- 监控中心(Monitor)启动时: 订阅 `/dubbo/com.foo.BarService` 目录下的所有提供者和消费者 URL 地址

3.2安装Zookeeper

下载地址：http://archive.apache.org/dist/zookeeper/. 本课程使用的Zookeeper版本为3.4.11

3.2.1window版本

- 解压zookeeper(解压到没有中文和空格目录下)

zookeeper-3.4.11.zip

- 修改zoo.cfg配置文件,将conf下的zoo_sample.cfg复制一份改名为zoo.cfg,需要配置:
    - dataDir=./ 临时数据存储的目录（可写相对路径）
    - clientPort=2181 zookeeper的端口号
- 进入bin目录,运行zkServer.cmd

zkServer.cmd

3.2.2 Linux版本

- 第一步:安装 jdk（略）
- 第二步: 把 zookeeper 的压缩包（zookeeper-3.4.11.tar.gz）上传到 linux 系统
- 第三步：解压缩压缩包

```
tar -zxvf zookeeper-3.4.11.tar.gz
```

- 第四步：进入zookeeper-3.4.11目录，创建data目录

```
mkdir data
```

- 第五步：进入conf目录，把zoo_sample.cfg 改名为zoo.cfg

```
cd conf
mv zoo_sample.cfg zoo.cfg
```

- 第六步：打开zoo.cfg文件，修改data属性：dataDir=/root/zookeeper-3.4.11/data

3.3启动、停止Zookeeper

进入bin目录，启动服务命令

```
 ./zkServer.sh start
```

停止服务命令

```
./zkServer.sh stop
```

查看服务状态

```
./zkServer.sh status
```

4.小结

1. Zookeeper: 一个项目, 作为服务存在, dubbo建议使用Zookeeper作为注册中心
2. Zookeeper安装
    - Window: 解压到一个没有中文和空格的目录下, 点击 bin/zkServer.cmd
    - Linux

知识点-Dubbo快速入门

1.目标

Dubbo作为一个RPC框架，其最核心的功能就是要实现跨网络的远程调用。本小节就是要创建两个应用，一个作为服务的提供者，一个作为服务的消费者。

通过Dubbo来实现服务消费者远程调用服务提供者的方法。

2.步骤

2.1服务提供者开发

1. 创建Maven工程(war),添加坐标
2. 创建service接口和实现类, 进行注册
3. 创建applicationContext-service.xml(配置dubbo)
4. 配置web.xml

2.2服务消费者开发

1. 创建Maven工程(war),添加坐标
2. 创建Controller 进行注册
3. 拷贝Service接口, 在Controller 里面注入service
4. 创建applicationContext-web.xml(配置dubbo)
5. 配置web.xml

3.实现

3.1服务提供者开发

【1】创建maven工程（打包方式为war）dubbodemo_provider

【2】在pom.xml文件中导入如下坐标

```
<?xml version="1.0" encoding="UTF-8"?>
<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.itheima</groupId>
    <artifactId>dubbodemo_provider</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <spring.version>5.0.2.RELEASE</spring.version>
    </properties>
    <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>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <!-- 指定端口 -->
                    <port>8080</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

【3】创建服务接口和服务实现类

- HelloService

```java
package com.itheima.service;


public interface HelloService {
    String sayHello(String name);
}
```

- HelloServiceImpl

```java
package com.itheima.service.impl;

import com.alibaba.dubbo.config.annotation.Service;
import com.itheima.service.HelloService;


@Service
public class HelloServiceImpl implements HelloService {
    @Override
    public String sayHello(String name) {
        return "hello..."+name;
    }
}
```

> 注意：服务实现类上使用的Service注解是Dubbo提供的，用于对外发布服务

【4】在src/main/resources下创建applicationContext-service.xml

```xml
<?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">

    <!--配置应用名称 name属性: 随便取,不要重复 建议写应用名-->
    <dubbo:application name="dubbodemo_provider"/>
    <!--配置注册中心地址 zookeeper所在服务器的ip地址端口号为2181-->
    <dubbo:registry address="zookeeper://127.0.0.1:2181" />
    <!--配置注册 协议和port   端口默认是20880-->
    <dubbo:protocol name="dubbo" port="20881"/>
    <!--配置包扫描 加入@Service注解的类会被发布为服务-->
    <dubbo:annotation package="com.itheima.service"/>

</beans>
```

【5】配置web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext*.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>
</web-app>
```

3.2服务消费者开发

【1】创建maven工程（打包方式为war）dubbodemo_consumer，

【2】在pom.xml添加依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<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.itheima</groupId>
    <artifactId>dubbodemo_consumer</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <spring.version>5.0.2.RELEASE</spring.version>
    </properties>

    <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>

        <!--jackson-->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.0</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
```

```
            <artifactId>jackson-core</artifactId>
            <version>2.9.0</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.9.0</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <!-- 指定端口 -->
                    <port>8081</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>


</project>
```

【3】创建服务接口

```
package com.itheima.service;

public interface HelloService {
    String sayHello(String name);
}
```

【4】创建Controller

```
@Controller
@RequestMapping("/dubbo")
public class HelloController {

    @Reference
    private HelloService helloService;

    @RequestMapping("/hello")
    @ResponseBody
    public Map getName(String name){
```

```
        //远程调用
        String result = helloService.sayHello(name);
        System.out.println(result);
        Map map =  new HashMap<>();
        map.put("result",result);
        return map;
    }
```

> 注意：Controller中注入HelloService使用的是Dubbo提供的@Reference注解

【5】在src/main/resources下创建applicationContext-web.xml

```
<?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">

    <!-- 当前应用名称，用于注册中心计算应用间依赖关系，注意：消费者和提供者应用名不要一样
-->
    <dubbo:application name="dubbodemo_consumer" />
    <!-- 连接服务注册中心zookeeper ip为zookeeper所在服务器的ip地址-->
    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
    <!-- 扫描的方式暴露接口  -->
    <dubbo:annotation package="com.itheima.controller" />

    <mvc:annotation-driven></mvc:annotation-driven>
</beans>
```

【6】配置web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>

```

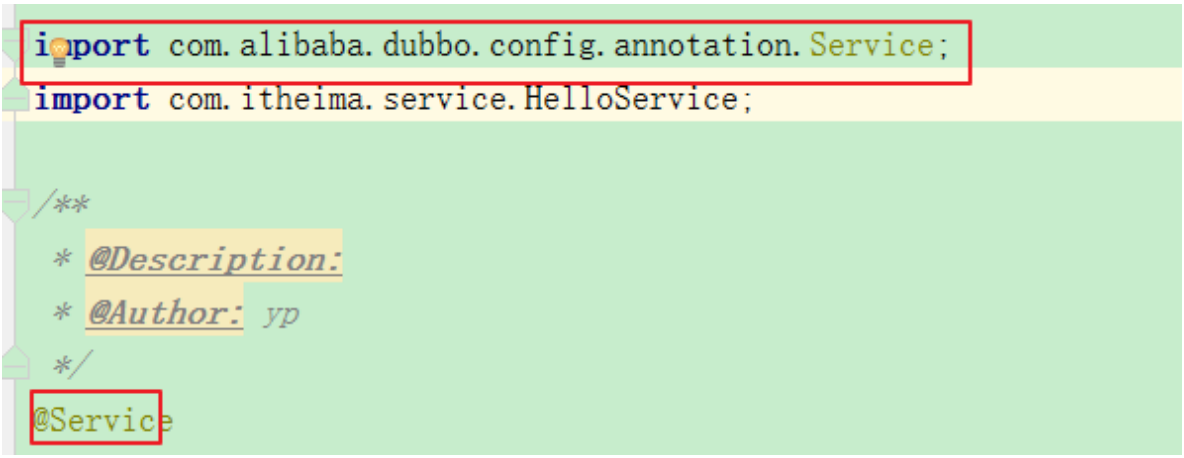
```
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- 指定加载的配置文件 ，通过参数contextConfigLocation加载 -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <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>
```

4.小结

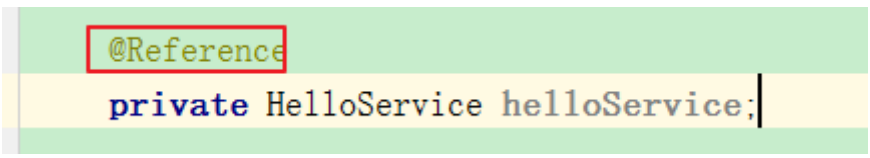
4.1提供者

- 注册用的@Service是dubbo的



4.2消费者

- 注入Service用的是@Reference



5.思考任务

- 思考一:上面的Dubbo入门案例中我们是将HelloService接口从服务提供者工程(dubbodemo_provider)复制到服务消费者工程(dubbodemo_consumer)中，这种做法是否合适？还有没有更好的方式？

答:这种做法显然是不好的，同一个接口被复制了两份，不利于后期维护。更好的方式是单独创建一个maven工程，将此接口创建在这个maven工程中。需要依赖此接口的工程只需要在自己工程的pom.xml文件中引入maven坐标即可。

- 思考二：在服务消费者工程(dubbodemo_consumer)中只是引用了HelloService接口，并没有提供实现类，Dubbo是如何做到远程调用的?

  答:Dubbo底层是基于代理技术为HelloService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框架完成的。

- 上面的Dubbo入门案例中我们使用Zookeeper作为服务注册中心，服务提供者需要将自己的服务信息注册到Zookeeper，服务消费者需要从Zookeeper订阅自己所需要的服务，此时Zookeeper服务就变得非常重要了，那如何防止Zookeeper单点故障呢?

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

知识点-Dubbo管理控制台

1.目标

我们在开发时，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。其实管理中心就是一个web应用，部署到tomcat即可

2.步骤

- Dubbo管理控制台的安装
- Dubbo管理控制台的使用

3.讲解

3.1安装

安装步骤：

（1）将资料中的dubbo-admin-2.6.0.war文件复制到tomcat的webapps目录下

（2）启动tomcat，此war文件会自动解压

（3）修改WEB-INF下的dubbo.properties文件，注意dubbo.registry.address对应的值需要对应当前使用的Zookeeper的ip地址和端口号

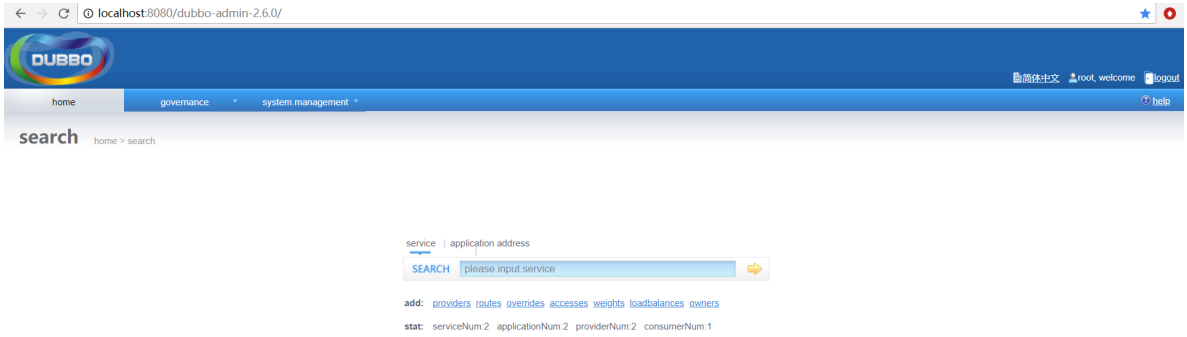
```
dubbo.registry.address=zookeeper://127.0.0.1:2181
dubbo.admin.root.password=root
dubbo.admin.guest.password=guest
```

（4）重启tomcat

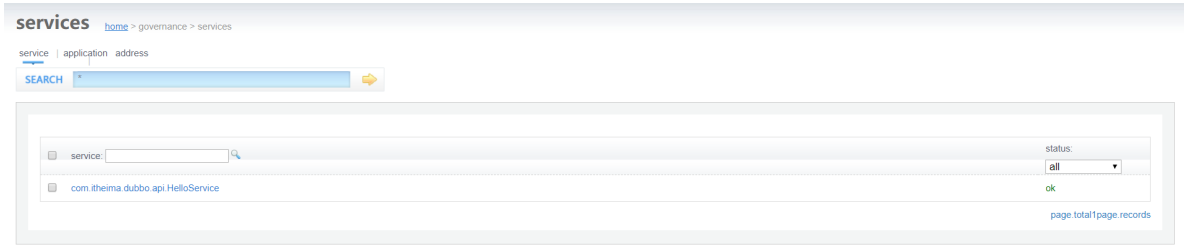
3.2使用

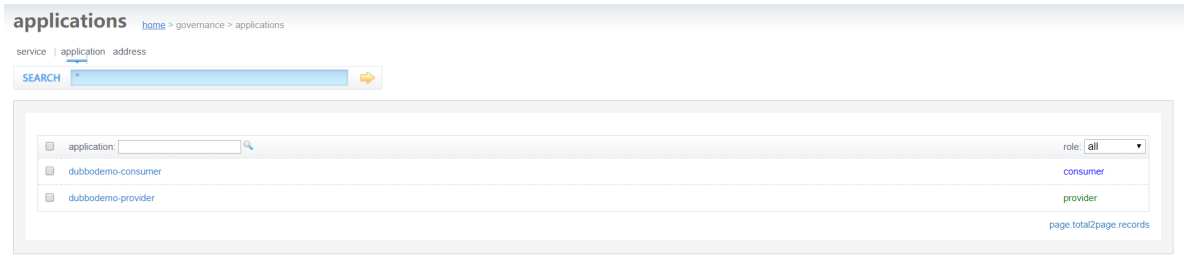
操作步骤：

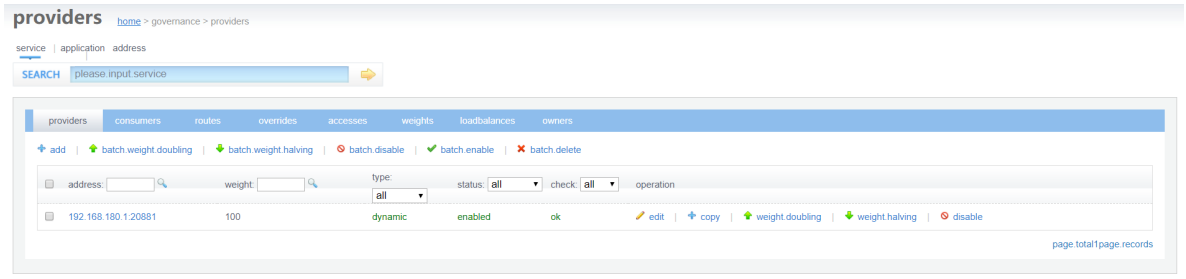
（1）访问http://localhost:8080/dubbo-admin-2.6.0/，输入用户名(root)和密码(root)

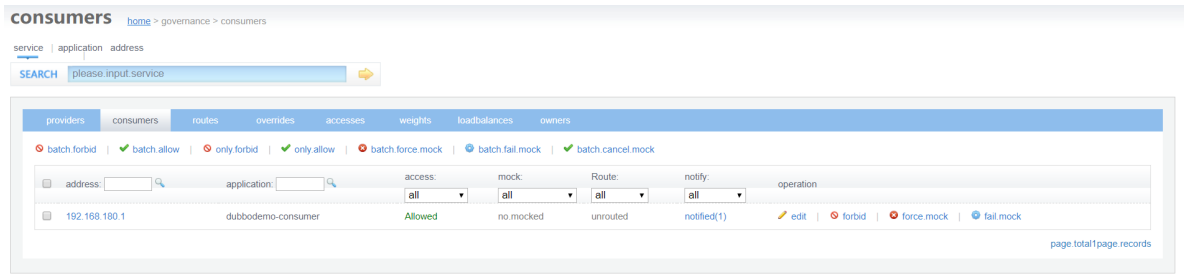


（2）启动服务提供者工程和服务消费者工程，可以在查看到对应的信息









4.小结

1. Dubbo管理控制台: 一个可视化工具, 可以查看提供者,消费者信息的
2. 版本
    - war, 发布到tomcat里面
    - jar 把dubbo-admin-0.0.1-SNAPSHOT.jar和start_dubbo_admin.bat一起拷贝到一个没有中文和空格目录, 点击start_dubbo_admin.bat就可以使用了

知识点-Dubbo相关配置说明

1.目标

- [ ] 掌握Dubbo相关配置

2.路径

- 包扫描
- 协议
- 启动时检查
- 负载均衡

3.讲解

3.1包扫描

```
<dubbo:annotation package="com.itheima.service" />
```

服务提供者和服务消费者都需要配置，表示包扫描，作用是扫描指定包(包括子包)下的类。如果不使用包扫描，也可以通过如下配置的方式来发布服务.

作为服务提供者, 可以通过如下配置来暴露服务[了解]:

```
<bean id="helloService" class="com.itheima.service.impl.HelloServiceImpl">
</bean>
<dubbo:service interface="com.itheima.service.HelloService"
ref="helloService"></dubbo:service>
```

作为服务消费者，可以通过如下配置来引用服务[了解]:

```
<!-- 生成远程服务代理，可以和本地bean一样使用helloService -->
<dubbo:reference id="helloService" interface="com.itheima.api.HelloService"
/>
```

在Controller里面使用@AutoWired注入

```
@Controller
@RequestMapping("/dubbo")
public class HelloController {
    //@Reference
    @Autowired
    private HelloService helloService;
```

上面这种方式发布和引用服务，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，这种方式就比较繁琐了。推荐使用包扫描方式。

3.2协议【面试题】

```xml
<dubbo:protocol name="dubbo" port="20880"/>
```

一般在服务提供者一方配置，可以指定使用的协议名称和端口号。

其中Dubbo支持的协议有：dubbo、rmi、hessian、http、webservice、rest、redis等。

推荐使用的是dubbo协议。

dubbo 协议采用单一长连接和 NIO 异步通讯，适合于小数据量大并发的服务调用，以及服务消费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，传视频等，除非请求量很低。

也可以在同一个工程中配置多个协议，不同服务可以使用不同的协议，例如：

```xml
<!-- 多协议配置 -->
<dubbo:protocol name="dubbo" port="20880" />
<dubbo:protocol name="rmi" port="1099" />
<!-- 使用dubbo协议暴露服务 -->
<dubbo:service interface="com.itheima.api.HelloService" ref="helloService"
protocol="dubbo" />
<!-- 使用rmi协议暴露服务 -->
<dubbo:service interface="com.itheima.api.DemoService" ref="demoService"
protocol="rmi" />
```

3.3启动时检查

```xml
<dubbo:consumer check="false"/>
```

上面这个配置需要配置在服务消费者一方，如果不配置默认check值为true。Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，以便上线时，能及早发现问题。可以通过将check值改为false来关闭检查。

可以在开发阶段将check值设置为false，在生产环境下改为true。

3.4负载均衡

负载均衡（Load Balance）：其实就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随机(random)、轮询(roundrobin)、最少活跃调用数(leastactive)、一致性Hash(consistenthash)，缺省为random随机调用。

配置负载均衡策略，既可以在服务提供者一方配置，也可以在服务消费者一方配置，如下：

```java
@Controller
@RequestMapping("/dubbo")
public class HelloController {
    @Reference(loadbalance = "random")
    private HelloService helloService;

    @RequestMapping("/hello")
    @ResponseBody
    public Map getName(String name){
        //远程调用
        String result = helloService.sayHello(name);
        System.out.println(result);
        Map map =  new HashMap<>();
```

```
        map.put("result",result);
        return map;
    }
}

```

```
//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;
    }
}
```

可以通过启动多个服务提供者来观察Dubbo负载均衡效果。注意：因为我们是在一台机器上启动多个服务提供者，所以需要修改tomcat的端口号和Dubbo服务的端口号来防止端口冲突。在实际生产环境中，多个服务提供者是分别部署在不同的机器上，所以不存在端口冲突问题。

> 参考:http://dubbo.apache.org/zh-cn/docs/source_code_guide/loadbalance.html

4.小结

1. 包扫描: 建议使用注解方式
2. 协议: dubbo协议, 通讯使用的是TCP. 适合量少 高并发的情况, 不适合数据量大的情况(eg: 下载,上传)
3. 启动时检查: 默认情况下, 没有启动提供方, 直接启动消费方, 这个时候消费方会报错. 通过配置:

```
<!--配置启动不检查-->
<dubbo:consumer check="false"/>
```

```
@Reference(check = false, loadbalance = "roundrobin")
private HelloService helloService;
```

4. 负载均衡
    - 随机(random)【默认】
    - 轮询(roundrobin)
    - 最少活跃调用数(leastactive)
    - 一致性Hash(consistenthash)

```
//负载均衡算法为轮询
@Reference(loadbalance = "roundrobin")
private HelloService helloService;
```

扩展知识点-Dubbo优化配置

1.目标

- [ ] 掌握Dubbo优化配置

2.步骤

- 超时配置
- 重连配置
- 缓存测试

3.讲解

3.1超时配置

在消费方调用服务方接口服务时，会发生如下超时错误,Dubbo消费方在调用服务时，超时时间默认是1000毫秒，这个时间可能比较短，如果一些复杂业务需要很长时间完成,这里会发生超时错误 ,我们可以手动设置一下超时时间，加大超时时间设置。

- 消费端

  全局配置：在spring-dubbo.xml中加入如下代码

```
<!--全局参数配置,超时时间：5秒-->
<dubbo:consumer timeout="5000" />
```

- 服务端(提供者)

  全局配置：在spring.xml中加入如下代码

```
<!--全局参数配置,超时时间：5秒-->
<dubbo:provider timeout="5000" />
```

3.2重连配置

dubbo在调用服务不成功时，默认会重试2次请求。 如果此时做的是幂等操作(修改、删除或者查询)，对真实数据不会造成影响，但如果是非幂等(增加)操作，此时会导致数据库数据多出2条记录。

幂等操作：在编程中一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相同。 （例如：删、改、查）

我们可以根据需要设置重连次数，修改消费方的spring-dubbo.xml,增加retries="0",表示不重连。

```
<!--全局参数配置,超时时间：5秒-->
<dubbo:consumer timeout="5000" retries="0" />
```

3.3 缓存测试【重点】

dubbo每次生成对应的代理对象后，都会将远程信息缓存到本地，即便zookeeper注册中心宕机了，也可以继续调用。只是新的服务无法再次注册。

4.小结

知识点-解决Dubbo无法发布被事务代理的Service问题

1.目标

前面我们已经完成了Dubbo的入门案例，通过入门案例我们可以看到通过Dubbo提供的标签配置就可以进行包扫描，扫描到@Service注解的类就可以被发布为服务。

但是我们如果在服务提供者类上加入@Transactional事务控制注解后，服务就发布不成功了。原因是事务控制的底层原理是为服务提供者类创建代理对象，而默认情况下Spring是基于JDK动态代理方式创建代理对象，而此代理对象的完整类名为com.sun.proxy.$Proxy42（最后两位数字不是固定的），导致Dubbo在发布服务前进行包匹配时无法完成匹配，进而没有进行服务的发布。

2.步骤

- 问题展示
- 问题解决

3.讲解

3.1 问题展示

在入门案例的服务提供者dubbodemo_provider工程基础上进行展示

操作步骤：

（1）在pom.xml文件中增加maven坐标

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</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>
```

（2）在applicationContext-service.xml配置文件中加入数据源、事务管理器、开启事务注解的相关配置

```
    <!--数据源-->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" destroy-method="close">
        <property name="username" value="itcast" />
        <property name="password" value="12345678" />
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/vue" />
    </bean>
    <!-- 事务管理器  -->
    <bean id="transactionManager"
```

```
10
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
11          <property name="dataSource" ref="dataSource"/>
12      </bean>
13      <!--开启事务控制的注解支持-->
14      <tx:annotation-driven transaction-manager="transactionManager"/>
```

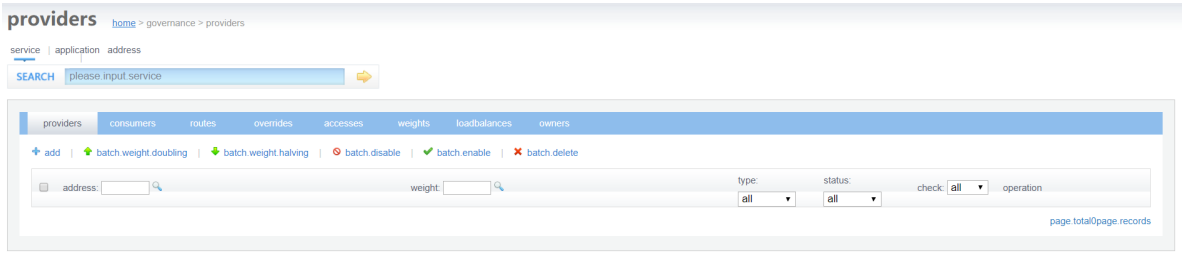
上面连接的数据库可以自行创建

（3）在HelloServiceImpl类上加入@Transactional注解

（4）启动服务提供者和服务消费者，并访问

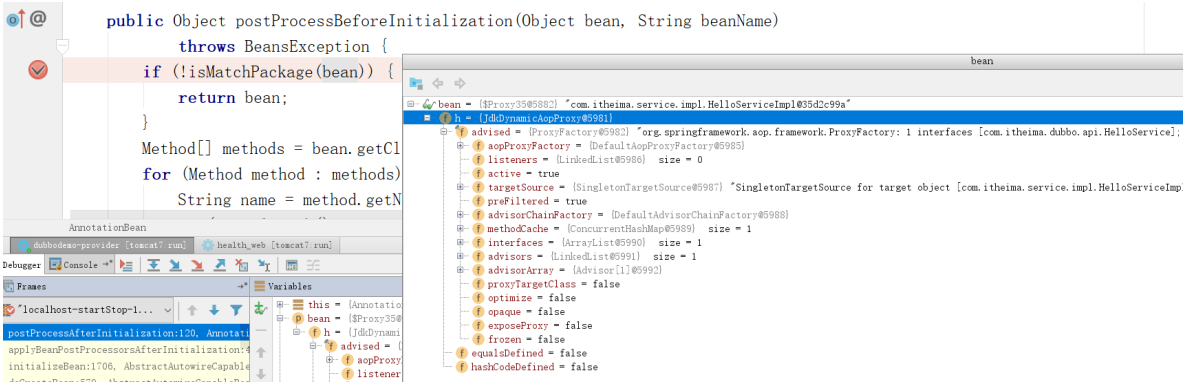
上面的错误为没有可用的服务提供者

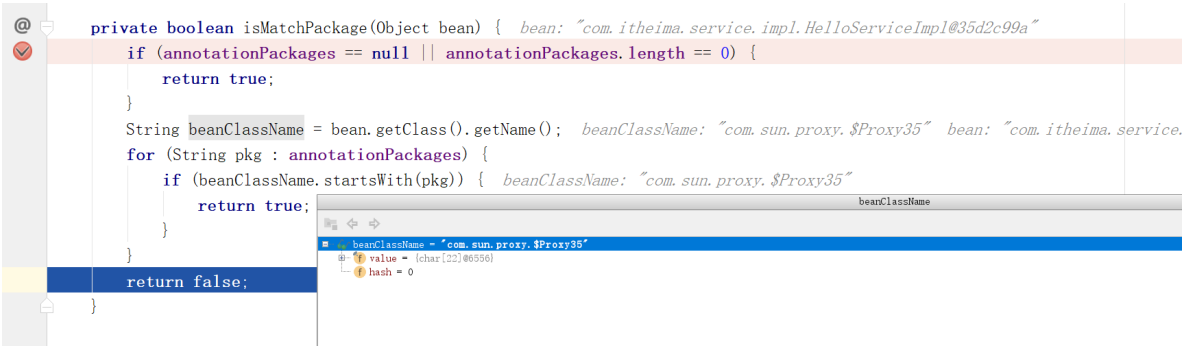
查看dubbo管理控制台发现服务并没有发布，如下：



原因: 加了@Transactional注解之后, Spring会创建代理对象.这个对象不在com.itheima.service包里面

可以通过断点调试的方式查看Dubbo执行过程，Dubbo通过AnnotationBean的postProcessAfterInitialization方法进行处理





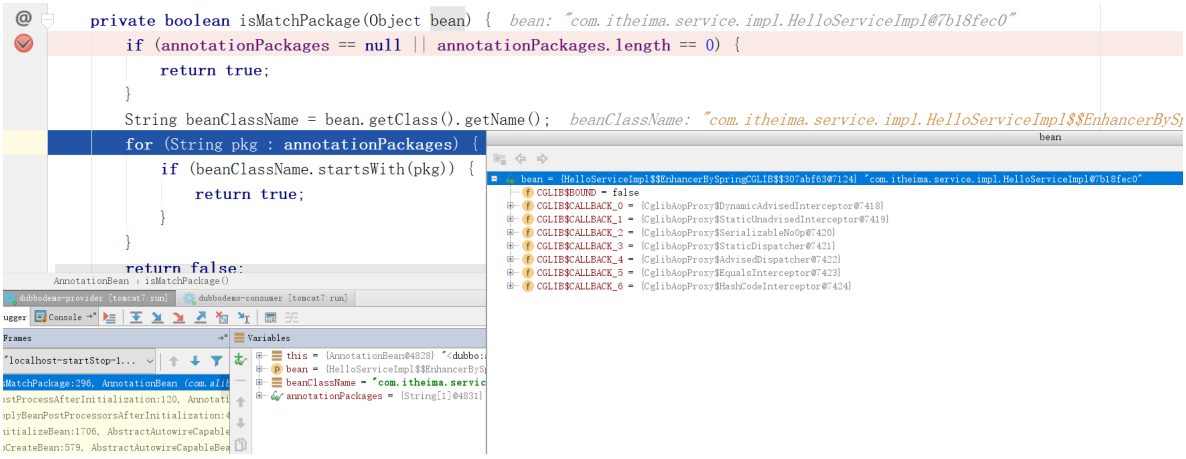
3.2 解决方案

通过上面的断点调试可以看到，在HelloServiceImpl类上加入事务注解后，Spring会为此类基于JDK动态代理技术创建代理对象，创建的代理对象完整类名为com.sun.proxy.$Proxy35，导致Dubbo在进行包匹配时没有成功（因为我们在发布服务时扫描的包为com.itheima.service），所以后面真正发布服务的代码没有执行。

解决方式操作步骤：

（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

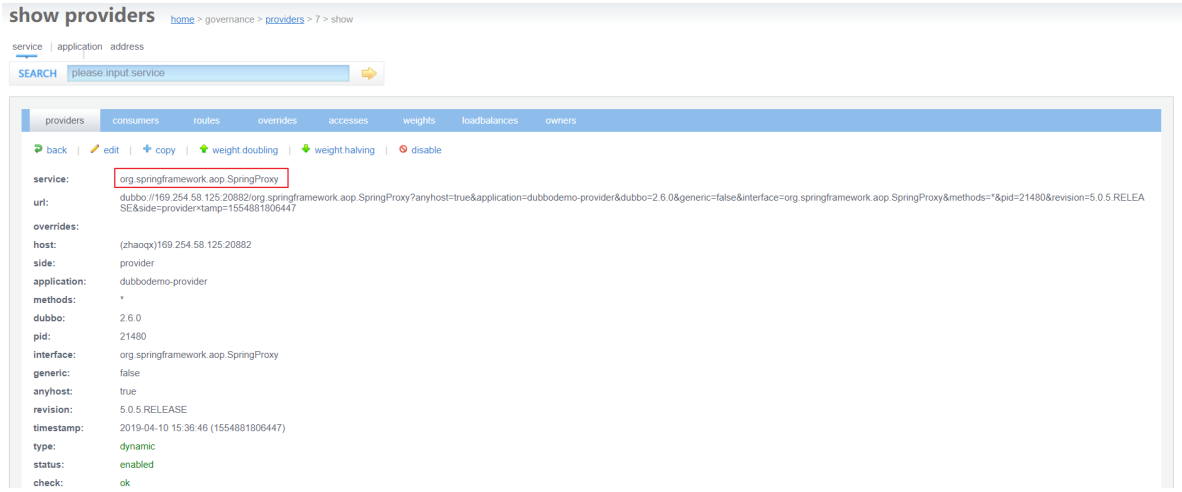
```
<!--开启事务控制的注解支持-->
<tx:annotation-driven transaction-manager="transactionManager" proxy-target-class="true"/>
```



（2）修改HelloServiceImpl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

```
@Service(interfaceClass = HelloService.class)
@Transactional
public class HelloServiceImpl implements HelloService {
    public String sayHello(String name) {
        return "hello " + name;
    }
}
```

此处也是必须要修改的，否则会导致发布的服务接口为SpringProxy，而不是HelloService接口，如下：



4.小结

4.1解决

1. 修改配置文件 改成cgLib方式动态代理

```
<!--开启事务控制的注解支持
    proxy-target-class="true": 手动指定cglib动态代理
-->
<tx:annotation-driven transaction-manager="transactionManager" proxy-target-class="true"/>
```

2. 指定动态代理的类型

```
@Transactional
@Service(interfaceClass = HelloService.class) //使用的Cglib产生
public class HelloServiceImpl implements HelloService {
```

总结

1.架构模式

- 单体架构
- 垂直架构
- SOA
- 微服务

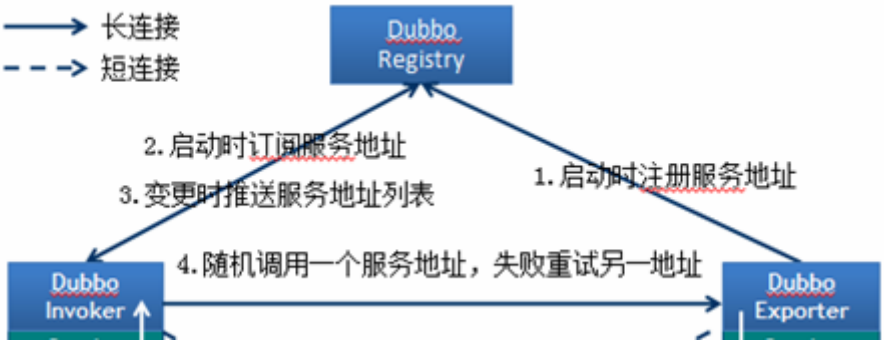
2.Dubbo理论

2.1Dubbo

一款RPC java框架

> RPC : 远程调用 是方式 不是协议

2.2Dubbo架构



3.Zookeeper

3.1什么Zookeeper

就是一个项目 以服务存在，一般用做dubbo的注册中心

3.2Zookeeper安装

4.Dubbo入门【重点】

4.1提供者 dubbo_service

4.2消费者 dubbo_web

5.Dubbo配置

- 包扫描
- 协议 (dubbo协议 tcp)
- 启动时候检查
- 负载均衡(4种)

6.Dubbo事物问题解决

1. 强转使用CgLib动态代理
2. 指定动态代理的类型是Service接口类型

7.扩展_dubbo高级配置

1. 超时
2. 重试(不幂等)
3. 缓存 【面试题】