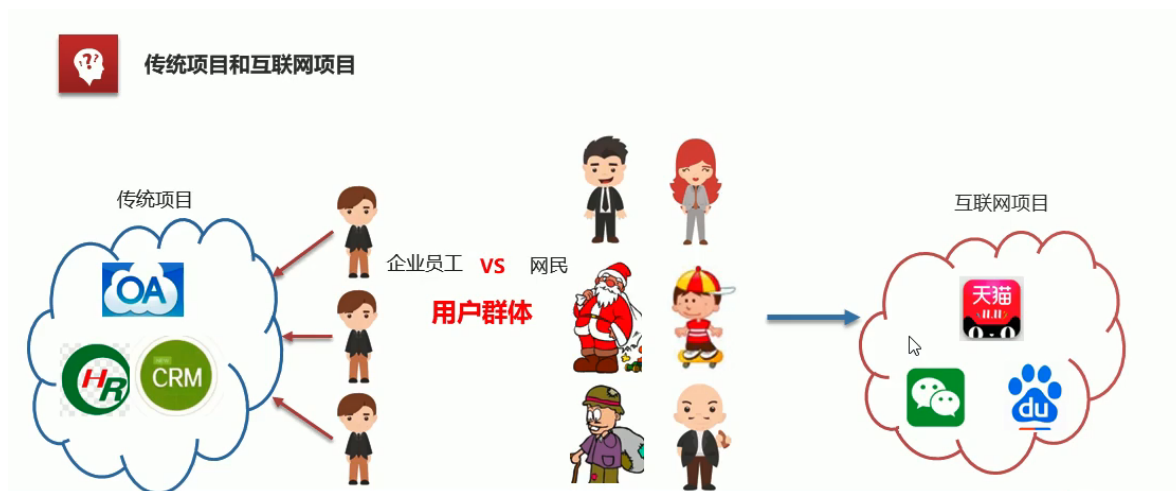


Dubbo

1.分布式系统的相关概念 ☆

1.1 大型互联网项目架构目标



1.用户群体：

传统项目：企业职工

互联网项目：广大网民

2.用户体验要求：

美观、功能、速度、稳定性

衡量一个网站速度是否快：

打开一个新页面一瞬间完成;页面内跳转，-刹那间完成。

根据佛经《僧祇律》记载:一刹那者为-念,二十念为-瞬,二十瞬为-弹指,二十弹指为-罗预,二十罗预为-须臾,一日一夜有三十须臾。

经过周密的计算:一瞬间为0.36秒,一刹那为0.018秒

1.1.1 互联网项目特点：

- 用户多
- 流量大，并发高
- 海量数据
- 易受攻击
- 功能繁琐
- 变更快

1.1.2 衡量网站的性能指标:

- **响应时间:**指执行一个请求从开始到最后收到响应数据所花费的总体时间。
- **并发数:**指系统同时能处理的请求数量。

并发连接数: 指的是客户端向服务器发起请求, 并建立了TCP连接。每秒钟服务器连接的总TCP数量

请求数: 也称为QPS(Query Per Second)指每秒多少请求.请求数是大于等于并发连接数的。

并发用户数: 单位时间内有多少用户

- **吞吐量:** 指单位时间内系统能处理的请求数量。

- QPS: Query Per Second每秒查询数。
 - TPS: Transactions Per Second每秒事务数。
 - 一个事务是指一个客户机向服务器发送请求然后服务器做出反应的过程。客户机在发送请求时开始计时, 收到服务器响应后结束计时, 以此来计算使用的时间和完成的事务个数。
 - 一个页面的一次访问, 只会形成一个TPS; 但一次页面请求, 可能产生多次对服务器的请求, 就会有多个QPS
- $QPS \geq \text{并发连接数} \geq TPS$

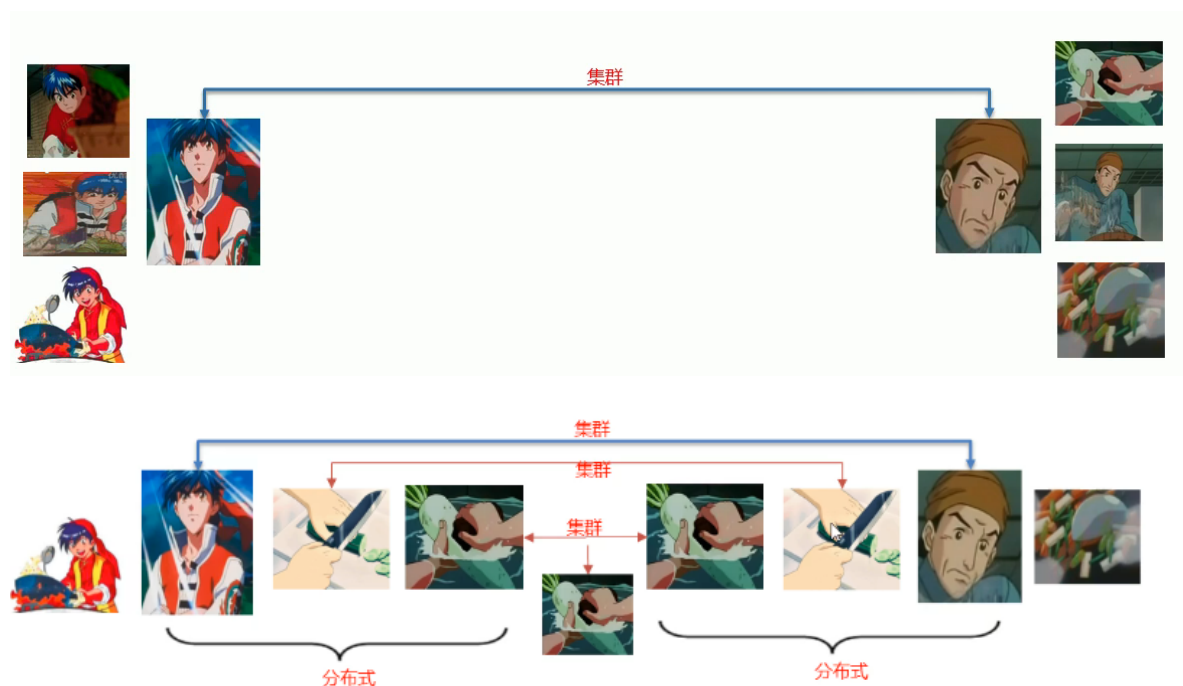
总结: 大型互联网根据以上性能指标进行衡量, 最终想实现的目标为:

- 高性能: 提供快速的访问体验
- 高可用: 网站服务可以一直正常访问
- 可伸缩: 通过硬件增加/减少, 提高/降低处理能力
- 高可拓展性: 系统间耦合低, 方便的通过新增/移除方式, 增加/减少新的功能/模块
- 安全性: 提供网站安全访问和数据加密, 安全存储等策略
- 敏捷性: 按需应变, 快速响应

1.2 集群和分布式相关概念

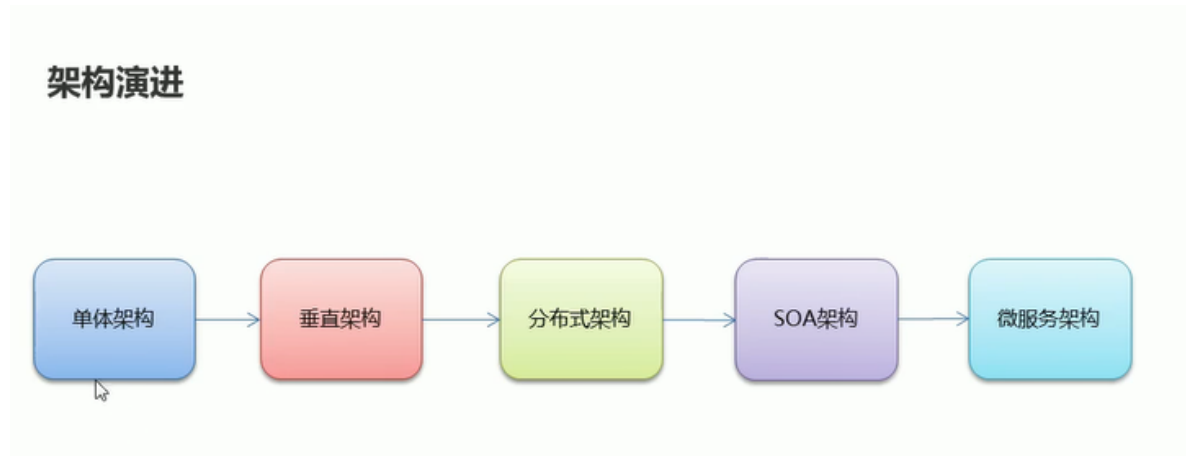
集群和分布式,

- 集群: 很多“人”一起, 干一样的事。实际上就是一个业务模块, 部署在多台服务器上。
- 分布式: 很多“人”一起, 干不一样的事。这些不一样的事, 合起来是一件大事。一个大的业务系统, 拆分为小的业务模块, 分别部署在不同的机器上。



1.3 分布式系统架构演进

架构演变历史：



1.3.1 单体架构

优点:

简单:开发部署都很方便，小型项目首选

缺点:

- 项目启动慢
- 可靠性差
- 可伸缩性差
- 拓展性和可维护性差
- 性能差

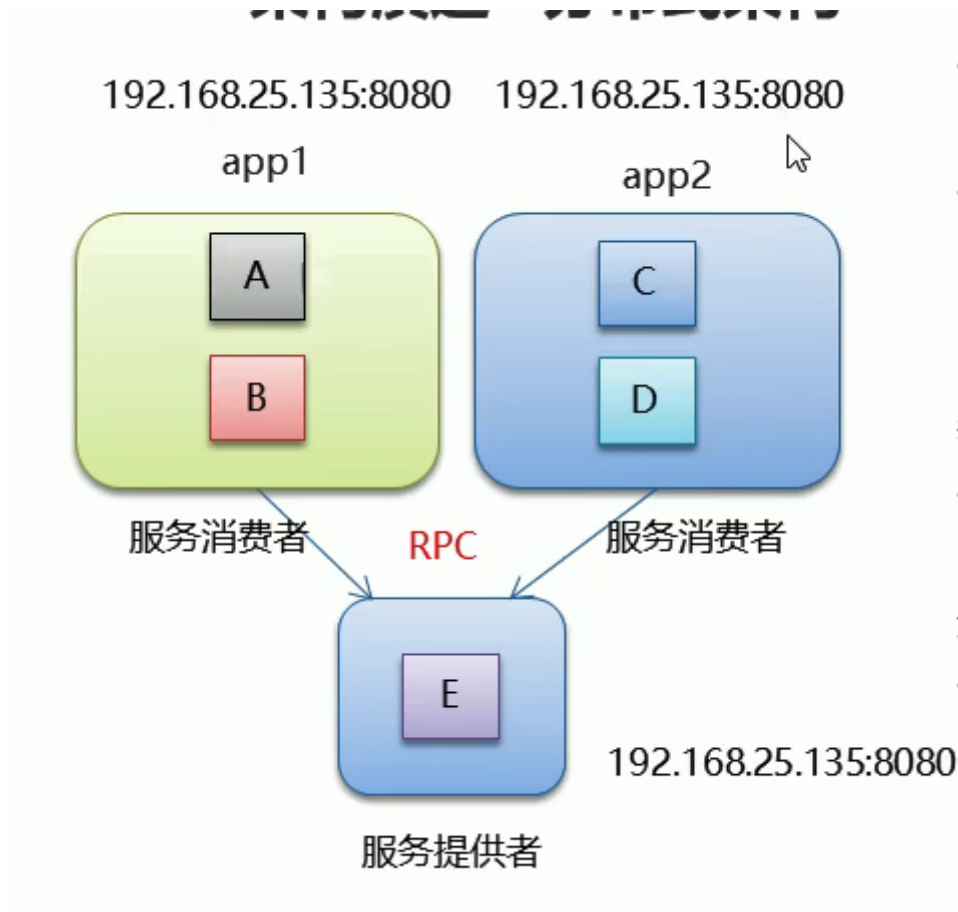
1.3.2 垂直架构

垂直架构是将单体架构中的多个模块拆分成为多个独立的架构，形成多个独立的单体架构。

垂直架构存在的问题：

- 重复功能太多啦

1.3.3 分布式架构



分布式服务是指在垂直架构的基础上，将公共的业务模块抽取出来，作为独立的服务，供其他调用者消费，以实现服务的共享和重用。底层通过RPC（远程过程调用实现）

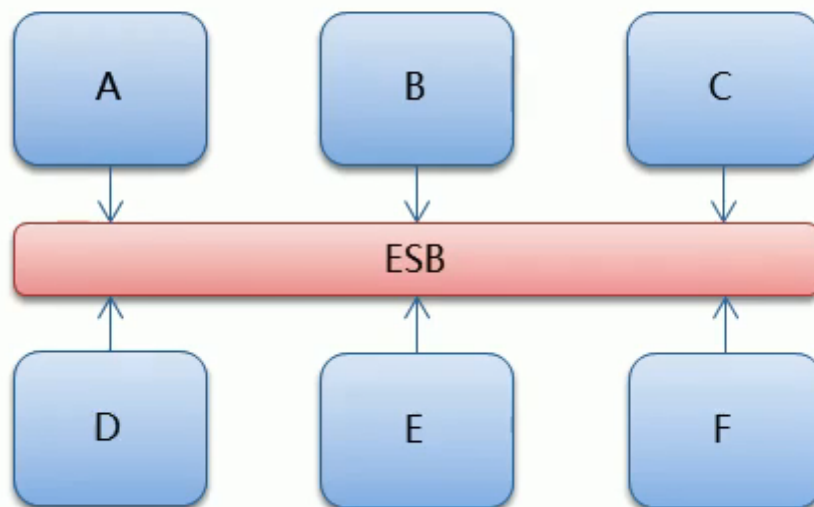
RPC: Remote Procedure Call 远程过程调用。有非常多的协议和技术来都实现了RPC的过程。比如: HTTP REST风格, Java RMI规范、WebService SOAP协议Hession等等。

分布式架构存在的问题:

- 服务提供方--旦产生变更,所有消费方都需要变更。比如服务提供方的IP端口等变更,由于分布式系统架构是直接调用,此时需要在调用方进行变更。

1.3.4 SOA架构

架构演进--SOA架构



SOA: (Service- Oriented Architecture,面向服务的架构)：是一个组件模型,它将应用程序的不同功能单元(称为服务)进行拆分,并通过这些服务之间定义良好的接口和契约联系起来。也就是服务消费方不需要直接知道服务提供方的IP端口等信息,因为不再直接调用。只需要通过良好的接口和契约告知调用哪个服务即可。

ESB: (Enterpraise Service Bus)：企业服务总线,服务中介。主要是提供了一个服务于服务之间的交互。ESB包含的功能如:负载均衡,流量控制,加密处理,服务的监控,异常处理,监控告急等等。

1.3.5 微服务架构

微服务架构是在SOA上做的升华,微服务架构强调的一个重点是“业务需要彻底的组件化和服务化”,原有的单个业务系统会拆分为多个可以独立开发、设计、运行的小应用。这些小应用之间通过服务完成交互和集成。

微服务架构= 80%的SOA服务架构思想+ 100%的组件化架构思想+ 80%的领域建模思想

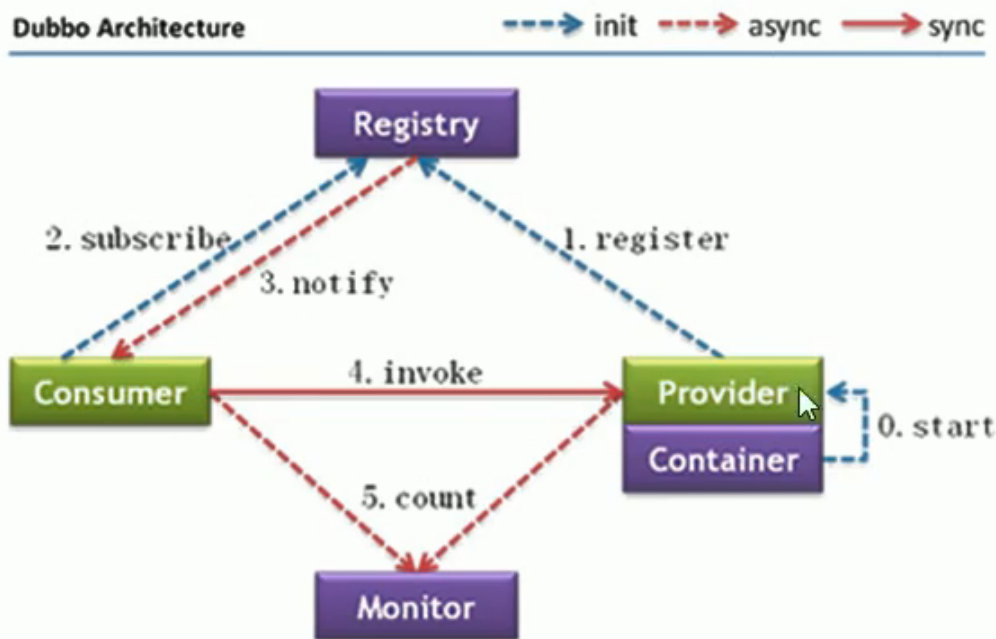
特点:

- 服务实现组件化:开发者可以自由选择开发技术。也不需要协调其他团队
- 服务之间交互一般使用REST API
- 去中心化:每个微服务有自己私有的数据库持久化业务数据
- 自动化部署:把应用拆分成为一个一个独立的单个服务,方便自动化部署、测试、运维

2.Dubbo概述☆

- Dubbo是阿里巴巴公司开源的一个高性能、轻量级的Java RPC框架。
- 致力于提供高性能和透明化的RPC远程服务调用方案,以及SOA服务治理方案。
- 官网: <http://dubbo.apache.org>

Dubbo架构



节点角色说明: .

节点	角色名称
Provider	暴露服务的提供方，这个选项运行在容器内部，通常是运行在Spring容器中
Container	服务运行容器，Spring容器启动服务
Consumer	调用远程服务的消费方,需要去subscribe订阅服务
Registry	服务注册与发现的注册中心，服务提供者有变动，也会notify通知给服务提供者
Monitor	统计服务的调用次数和调用时间的监控中心，监控中心不是必须的

虚线都是异步调用，实线是同步调用。

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。

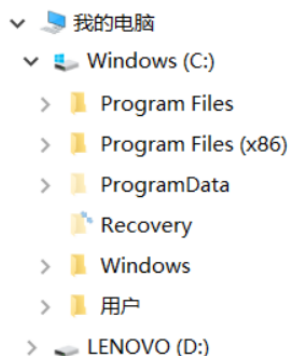
4. 服务消费者，从服务提供者列表中，基于负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

3.Zookeeper注册中心☆🐑

3.1 Zookeeper介绍

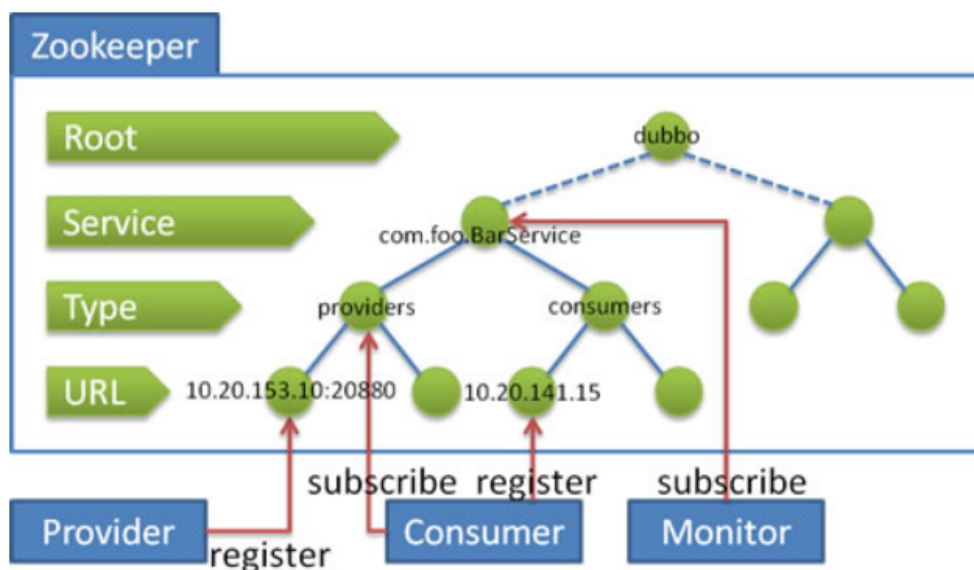
Zookeeper 是 Apache Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用。

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 Zookeeper安装

下载地址: <http://archive.apache.org/dist/zookeeper/>

本课程使用的Zookeeper版本为3.5.6, 下载完成后可以获得名称为zookeeper-3.4.6.tar.gz的压缩文件。

安装步骤:

第一步: 安装 jdk (略)

第二步: 把 zookeeper 的压缩包 (zookeeper-3.4.6.tar.gz) 上传到 linux 系统

```
[root@localhost ~]# ll
总用量 175048
-rw-r--r--. 1 root root 9230052 6月 6 12:41 apache-zookeeper-3.5.6-bin.tar.gz
lrwxrwxrwx. 1 root root 7 4月 15 2020 bin -> usr/bin
```

第三步: 解压缩压缩包

```
1 tar -zxvf apache-zookeeper-3.5.6-bin.tar.gz -C /usr/local/
```

```
[root@localhost ~]# cd /usr/local/
[root@localhost local]# ll
总用量 0
drwxr-xr-x. 6 root root 134 6月 6 12:44 apache-zookeeper-3.5.6-bin
drwxr-xr-x. 2 root root 134 11月 1 2021 bin
drwxr-xr-x. 2 root root 6 11月 5 2016 etc
drwxr-xr-x. 2 root root 6 11月 5 2016 games
drwxr-xr-x. 2 root root 6 11月 5 2016 include
drwxr-xr-x. 2 root root 6 11月 5 2016 lib
drwxr-xr-x. 2 root root 6 11月 5 2016 lib64
drwxr-xr-x. 2 root root 6 11月 5 2016 libexec
drwxr-xr-x. 2 root root 6 11月 5 2016 sbin
drwxr-xr-x. 5 root root 49 4月 15 2020 share
drwxr-xr-x. 2 root root 6 11月 5 2016 src
```

第四步: 进入zookeeper-3.4.6目录, 创建data目录

```
[root@localhost apache-zookeeper-3.5.6-bin]# mkdir data
[root@localhost apache-zookeeper-3.5.6-bin]# ll
总用量 32
drwxr-xr-x. 2 1000 1000 232 10月 9 2019 bin
drwxr-xr-x. 2 1000 1000 70 6月 6 12:52 conf
drwxr-xr-x. 2 root root 6 6月 6 12:53 data
drwxr-xr-x. 5 1000 1000 4096 10月 9 2019 docs
drwxr-xr-x. 2 root root 4096 6月 6 12:44 lib
-rw-r--r--. 1 1000 1000 11358 10月 5 2019 LICENSE.txt
-rw-r--r--. 1 1000 1000 432 10月 9 2019 NOTICE.txt
-rw-r--r--. 1 1000 1000 1560 10月 9 2019 README.md
-rw-r--r--. 1 1000 1000 1347 10月 5 2019 README_packaging.txt
```

```
1 mkdir data
```

此时data目录为: /usr/local/apache-zookeeper-3.5.6-bin/data

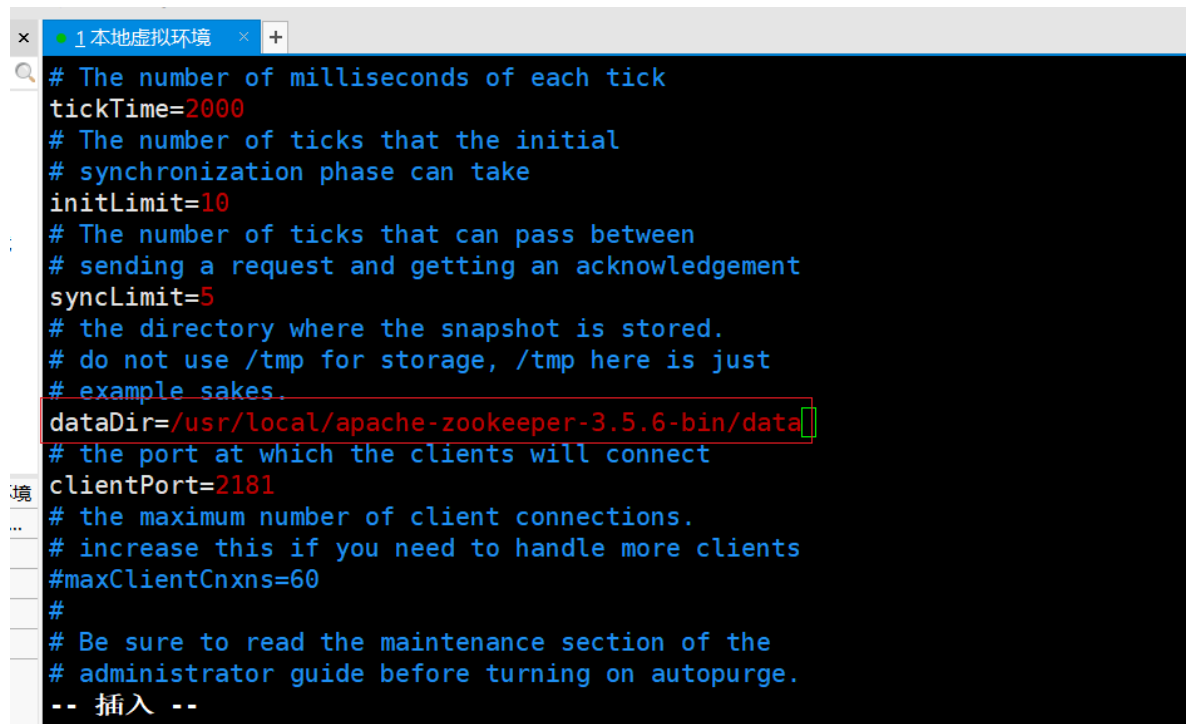
第五步: 进入conf目录, 把zoo_sample.cfg 改名为zoo.cfg

```
总用量 32
drwxr-xr-x. 2 1000 1000 232 10月 9 2019 bin
drwxr-xr-x. 2 1000 1000 77 10月 9 2019 conf
drwxr-xr-x. 5 1000 1000 4096 10月 9 2019 docs
drwxr-xr-x. 2 root root 4096 6月 6 12:44 lib
-rw-r--r--. 1 1000 1000 11358 10月 5 2019 LICENSE.txt
-rw-r--r--. 1 1000 1000 432 10月 9 2019 NOTICE.txt
-rw-r--r--. 1 1000 1000 1560 10月 9 2019 README.md
-rw-r--r--. 1 1000 1000 1347 10月 5 2019 README_packaging.txt
[root@localhost apache-zookeeper-3.5.6-bin]# cd conf
```



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

第六步：打开zoo.cfg文件，修改data属性：dataDir=/usr/local/apache-zookeeper-3.5.6-bin/data



```
# 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 sake.
dataDir=/usr/local/apache-zookeeper-3.5.6-bin/data
# 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.
-- 插入 --
```

3.3 启动、停止Zookeeper

zookeeper的默认端口是2181.

进入Zookeeper的bin目录，启动服务命令

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

停止服务命令

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

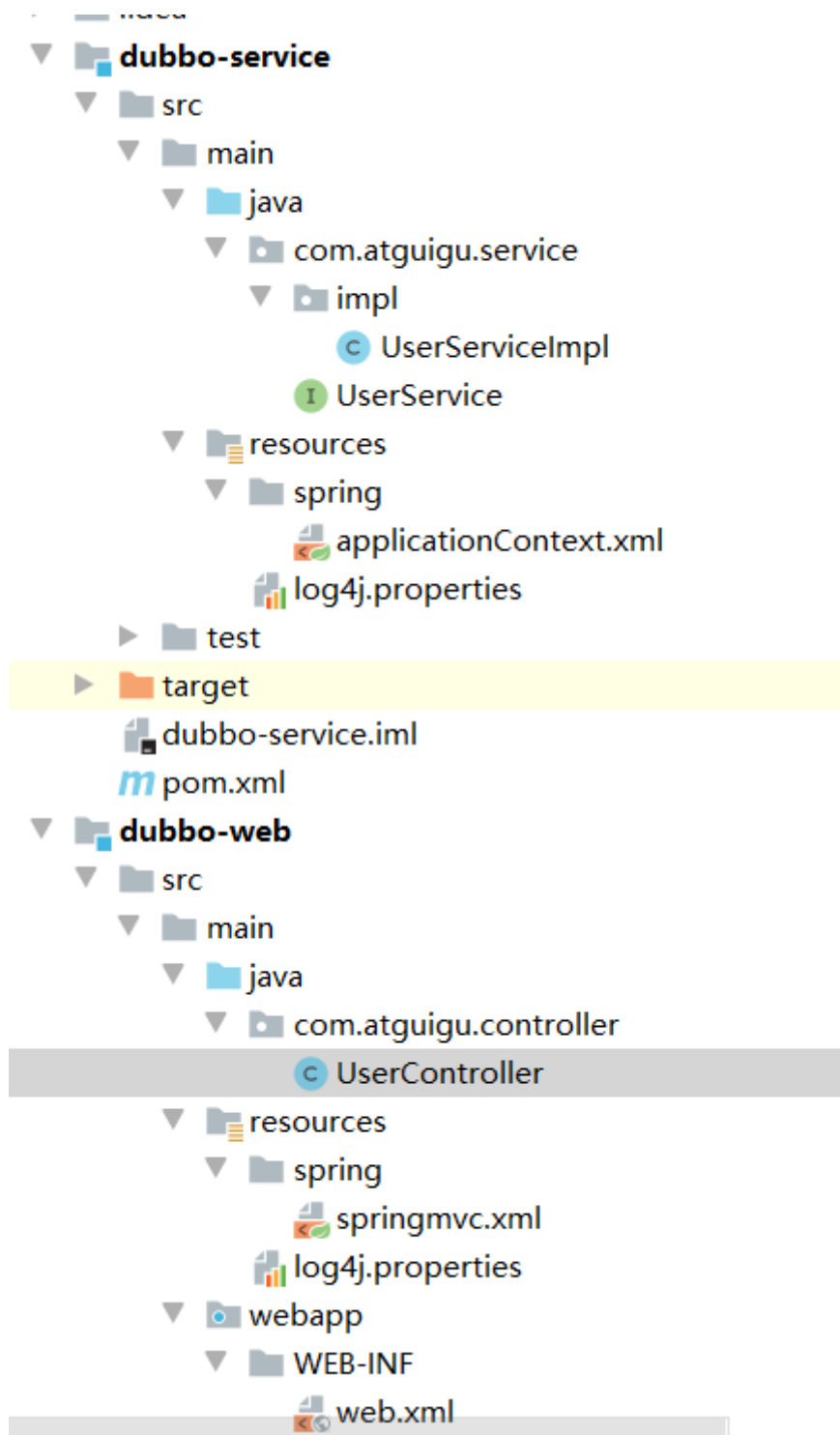
查看服务状态：

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

4.Dubbo案例 ☆

4.1 基础案例搭建

我们先创建两个工程：dubbo-service和dubbo-web.其中dubbo-service是一个jar工程，被web工程dubbo-web所依赖。



dubbo-service:

pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5 http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>IdeaProject1</artifactId>
8         <groupId>org.example</groupId>
9         <version>1.0-SNAPSHOT</version>
10     </parent>
11     <modelVersion>4.0.0</modelVersion>
```

```
12     <artifactId>dubbo-service</artifactId>
13
14     <properties>
15         <spring.version>5.1.9.RELEASE</spring.version>
16         <dubbo.version>2.7.4.1</dubbo.version>
17         <zookeeper.version>4.0.0</zookeeper.version>
18
19     </properties>
20
21     <dependencies>
22         <!-- servlet3.0规范的坐标 -->
23         <dependency>
24             <groupId>javax.servlet</groupId>
25             <artifactId>javax.servlet-api</artifactId>
26             <version>3.1.0</version>
27             <scope>provided</scope>
28         </dependency>
29         <!--spring的坐标-->
30         <dependency>
31             <groupId>org.springframework</groupId>
32             <artifactId>spring-context</artifactId>
33             <version>${spring.version}</version>
34         </dependency>
35         <!--springmvc的坐标-->
36         <dependency>
37             <groupId>org.springframework</groupId>
38             <artifactId>spring-webmvc</artifactId>
39             <version>${spring.version}</version>
40         </dependency>
41
42         <!--日志-->
43         <dependency>
44             <groupId>org.slf4j</groupId>
45             <artifactId>slf4j-api</artifactId>
46             <version>1.7.21</version>
47         </dependency>
48         <dependency>
49             <groupId>org.slf4j</groupId>
50             <artifactId>slf4j-log4j12</artifactId>
51             <version>1.7.21</version>
52         </dependency>
53
54
55
56         <!--Dubbo的起步依赖，版本2.7之后统一为rg.apache.dubb -->
57         <dependency>
58             <groupId>org.apache.dubbo</groupId>
59             <artifactId>dubbo</artifactId>
60             <version>${dubbo.version}</version>
61         </dependency>
62         <!--ZooKeeper客户端实现 -->
63         <dependency>
64             <groupId>org.apache.curator</groupId>
65             <artifactId>curator-framework</artifactId>
66             <version>${zookeeper.version}</version>
67         </dependency>
68         <!--ZooKeeper客户端实现 -->
69         <dependency>
```

```

70         <groupId>org.apache.curator</groupId>
71         <artifactId>curator-recipes</artifactId>
72         <version>${zookeeper.version}</version>
73     </dependency>
74
75 </dependencies>
76 </project>

```

配置文件

```

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:dubbo="http://dubbo.apache.org/schema/dubbo"
5        xmlns:context="http://www.springframework.org/schema/context"
6        xsi:schemaLocation="http://www.springframework.org/schema/beans
7        http://www.springframework.org/schema/beans/spring-beans.xsd
8        http://dubbo.apache.org/schema/dubbo
9        http://dubbo.apache.org/schema/dubbo/dubbo.xsd
10       http://www.springframework.org/schema/context
11       https://www.springframework.org/schema/context/spring-context.xsd">
12      <context:component-scan base-package="com.atguigu.service">
13      </context:component-scan>
14
15
16 </beans>

```

```

1  # DEBUG < INFO < WARN < ERROR < FATAL
2  # Global logging configuration
3  log4j.rootLogger=info, stdout,file
4  # My logging configuration...
5  #log4j.logger.com.tocersoft.school=DEBUG
6  #log4j.logger.net.sf.hibernate.cache=debug
7  ## Console output...
8  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
9  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
10 log4j.appender.stdout.layout.ConversionPattern=%5p %d %C: %m%n
11
12 log4j.appender.file=org.apache.log4j.FileAppender
13 log4j.appender.file.File=../logs/iask.log
14 log4j.appender.file.layout=org.apache.log4j.PatternLayout
15 log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %l %m%n

```

接口

```

1  package com.atguigu.service;
2
3  public interface UserService {
4      public String sayHello();
5  }

```

实现类

```
1 package com.atguigu.service.impl;
2
3 import com.atguigu.service.UserService;
4 import org.springframework.stereotype.Service;
5
6 @Service
7 public class UserServiceImpl implements UserService {
8     @Override
9     public String sayHello() {
10         return "hello dubbo";
11     }
12 }
```

dubbo-web

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>IdeaProject1</artifactId>
8         <groupId>org.example</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12
13    <artifactId>dubbo-web</artifactId>
14
15    <!-- 这个项目需要打成war包-->
16    <packaging>war</packaging>
17
18    <properties>
19        <spring.version>5.1.9.RELEASE</spring.version>
20        <dubbo.version>2.7.4.1</dubbo.version>
21        <zookeeper.version>4.0.0</zookeeper.version>
22    </properties>
23
24    <dependencies>
25        <!-- servlet3.0规范的坐标 -->
26        <dependency>
27            <groupId>javax.servlet</groupId>
28            <artifactId>javax.servlet-api</artifactId>
29            <version>3.1.0</version>
30            <scope>provided</scope>
31        </dependency>
32        <!--spring的坐标-->
33        <dependency>
34            <groupId>org.springframework</groupId>
35            <artifactId>spring-context</artifactId>
36            <version>${spring.version}</version>
37        </dependency>
38        <!--springmvc的坐标-->
39        <dependency>
40            <groupId>org.springframework</groupId>
```

```
40         <artifactId>spring-webmvc</artifactId>
41         <version>${spring.version}</version>
42     </dependency>
43
44     <!--日志-->
45     <dependency>
46         <groupId>org.slf4j</groupId>
47         <artifactId>slf4j-api</artifactId>
48         <version>1.7.21</version>
49     </dependency>
50     <dependency>
51         <groupId>org.slf4j</groupId>
52         <artifactId>slf4j-log4j12</artifactId>
53         <version>1.7.21</version>
54     </dependency>
55
56
57
58     <!--Dubbo的起步依赖，版本2.7之后统一为rg.apache.dubb -->
59     <dependency>
60         <groupId>org.apache.dubbo</groupId>
61         <artifactId>dubbo</artifactId>
62         <version>${dubbo.version}</version>
63     </dependency>
64     <!--ZooKeeper客户端实现 -->
65     <dependency>
66         <groupId>org.apache.curator</groupId>
67         <artifactId>curator-framework</artifactId>
68         <version>${zookeeper.version}</version>
69     </dependency>
70     <!--ZooKeeper客户端实现 -->
71     <dependency>
72         <groupId>org.apache.curator</groupId>
73         <artifactId>curator-recipes</artifactId>
74         <version>${zookeeper.version}</version>
75     </dependency>
76
77     <dependency>
78         <groupId>org.example</groupId>
79         <artifactId>dubbo-service</artifactId>
80         <version>1.0-SNAPSHOT</version>
81     </dependency>
82 </dependencies>
83
84
85 <build>
86     <plugins>
87         <!--tomcat插件-->
88         <plugin>
89             <groupId>org.apache.tomcat.maven</groupId>
90             <artifactId>tomcat7-maven-plugin</artifactId>
91             <version>2.1</version>
92             <configuration>
93                 <port>8000</port>
94                 <path>/</path>
95             </configuration>
96         </plugin>
97     </plugins>
```

```

98     </build>
99
100 </project>

```

配置文件

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xmlns="http://java.sun.com/xml/ns/javaee"
4         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6         version="2.5">
7
8      <!-- spring -->
9      <context-param>
10         <param-name>contextConfigLocation</param-name>
11         <param-value>classpath*:spring/applicationContext*.xml</param-value>
12     </context-param>
13     <listener>
14         <listener-
15         class>org.springframework.web.context.ContextLoaderListener</listener-class>
16     </listener>
17
18     <!-- Springmvc -->
19     <servlet>
20         <servlet-name>springmvc</servlet-name>
21         <servlet-
22         class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
23         <!-- 指定加载的配置文件 ， 通过参数contextConfigLocation加载-->
24         <init-param>
25             <param-name>contextConfigLocation</param-name>
26             <param-value>classpath:spring/springmvc.xml</param-value>
27         </init-param>
28     </servlet>
29
30     <servlet-mapping>
31         <servlet-name>springmvc</servlet-name>
32         <!--
33             如果在 web.xml 中配置的 *.do, 则会拦截所有的 .do 请求去匹配
34             但是在 Controller 中的 RequestMapping 的 Value 如果是字符串没有不是 .do 结尾,
35             那么 Spring MVC 会默认加上 .do,
36             web.xml 中配置了url-pattern后, 会起到两个作用:
37             (1) 是限制 url 的后缀名, 只能为 ".do"。
38             (2) 就是在没有填写后缀时, 默认在你配置的 Controller 的 RequestMapping 中添
39             加 ".do" 的后缀。
40         -->
41         <url-pattern>*.do</url-pattern>
42     </servlet-mapping>
43
44 </web-app>

```

```

1  # DEBUG < INFO < WARN < ERROR < FATAL
2  # Global logging configuration
3  log4j.rootLogger=info, stdout,file
4  # My logging configuration...
5  #log4j.logger.com.tocersoft.school=DEBUG

```



```

6  #log4j.logger.net.sf.hibernate.cache=debug
7  ## Console output...
8  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
9  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
10 log4j.appender.stdout.layout.ConversionPattern=%5p %d %C: %m%n
11
12 log4j.appender.file=org.apache.log4j.FileAppender
13 log4j.appender.file.File=../logs/iask.log
14 log4j.appender.file.layout=org.apache.log4j.PatternLayout
15 log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %l %m%n

```

```

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:dubbo="http://dubbo.apache.org/schema/dubbo"
5      xmlns:mvc="http://www.springframework.org/schema/mvc"
6      xmlns:context="http://www.springframework.org/schema/context"
7      xsi:schemaLocation="http://www.springframework.org/schema/beans
8          http://www.springframework.org/schema/beans/spring-beans.xsd
9          http://www.springframework.org/schema/mvc
10         http://www.springframework.org/schema/mvc/spring-mvc.xsd
11         http://dubbo.apache.org/schema/dubbo
12         http://dubbo.apache.org/schema/dubbo/dubbo.xsd
13         http://www.springframework.org/schema/context
14         https://www.springframework.org/schema/context/spring-context.xsd">
15  <mvc:annotation-driven></mvc:annotation-driven>
16  <context:component-scan base-package="com.atguigu.controller">
17  </context:component-scan>
18
19  </beans>

```

代码

```

1  package com.atguigu.controller;
2
3
4  import com.atguigu.service.UserService;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RestController;
8
9  @RestController
10 @RequestMapping("/user")
11 public class UserController {
12     @Autowired
13     private UserService userService;
14
15     @RequestMapping("/sayHello")
16     public String sayHello(){
17         return userService.sayHello();
18     }
19 }
20
21

```

此时我们访问：<http://localhost:8000/user/sayHello.do>，可以看到字符串顺利被打印出来。

此时的Dubbo-service还不是一个独立的web服务，只是被依赖进来啦，上述工程还不算是一个分布式工程。为此我们需要对上述项目进行改造。

4.2 服务提供者和@servcie

对service的pom.xml进行改造

- 1.引入依赖：由于此时的服务提供方也是web工程，所以我们也添加tomcat插件

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5          http://maven.apache.org/xsd/maven-4.0.0.xsd">
6      <parent>
7          <artifactId>IdeaProject1</artifactId>
8          <groupId>org.example</groupId>
9          <version>1.0-SNAPSHOT</version>
10     </parent>
11     <modelVersion>4.0.0</modelVersion>
12     <artifactId>dubbo-service</artifactId>
13     <packaging>war</packaging>
14     <properties>
15         <spring.version>5.1.9.RELEASE</spring.version>
16         <dubbo.version>2.7.4.1</dubbo.version>
17         <zookeeper.version>4.0.0</zookeeper.version>
18     </properties>
19     <dependencies>
20         <!-- servlet3.0规范的坐标 -->
21         <dependency>
22             <groupId>javax.servlet</groupId>
23             <artifactId>javax.servlet-api</artifactId>
24             <version>3.1.0</version>
25             <scope>provided</scope>
26         </dependency>
27         <!--spring的坐标-->
28         <dependency>
29             <groupId>org.springframework</groupId>
30             <artifactId>spring-context</artifactId>
31             <version>${spring.version}</version>
32         </dependency>
33         <!--springmvc的坐标-->
34         <dependency>
35             <groupId>org.springframework</groupId>
36             <artifactId>spring-webmvc</artifactId>
37             <version>${spring.version}</version>
38         </dependency>
39         <!--日志-->
40         <dependency>
41             <groupId>org.slf4j</groupId>
42             <artifactId>slf4j-api</artifactId>
43             <version>1.7.21</version>
44         </dependency>
45     </dependencies>
```

```

49         <groupId>org.slf4j</groupId>
50         <artifactId>slf4j-log4j12</artifactId>
51         <version>1.7.21</version>
52     </dependency>
53
54
55
56     <!--Dubbo的起步依赖，版本2.7之后统一为rg.apache.dubb -->
57     <dependency>
58         <groupId>org.apache.dubbo</groupId>
59         <artifactId>dubbo</artifactId>
60         <version>${dubbo.version}</version>
61     </dependency>
62     <!--ZooKeeper客户端实现 -->
63     <dependency>
64         <groupId>org.apache.curator</groupId>
65         <artifactId>curator-framework</artifactId>
66         <version>${zookeeper.version}</version>
67     </dependency>
68     <!--ZooKeeper客户端实现 -->
69     <dependency>
70         <groupId>org.apache.curator</groupId>
71         <artifactId>curator-recipes</artifactId>
72         <version>${zookeeper.version}</version>
73     </dependency>
74
75 </dependencies>
76
77 <build>
78     <plugins>
79         <!--tomcat插件-->
80         <plugin>
81             <groupId>org.apache.tomcat.maven</groupId>
82             <artifactId>tomcat7-maven-plugin</artifactId>
83             <version>2.1</version>
84             <configuration>
85                 <port>9999</port>
86                 <path>/</path>
87             </configuration>
88         </plugin>
89     </plugins>
90 </build>
91 </project>

```

- 2.编写web.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xmlns="http://java.sun.com/xml/ns/javaee"
4          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5          http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6          version="2.5">
7      <!-- spring -->
8      <context-param>
9          <param-name>contextConfigLocation</param-name>
10         <param-value>classpath*:spring/applicationContext*.xml</param-value>
11     </context-param>
12     <listener>

```

```

12         <listener-
    class>org.springframework.web.context.ContextLoaderListener</listener-class>
13     </listener>
14
15 </web-app>

```

- 3.编写Spring配置文件需要加入注册中心相关信息

```

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:dubbo="http://dubbo.apache.org/schema/dubbo"
    xmlns:context="http://www.springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
6      http://dubbo.apache.org/schema/dubbo
    http://dubbo.apache.org/schema/dubbo/dubbo.xsd
    http://www.springframework.org/schema/context
    https://www.springframework.org/schema/context/spring-context.xsd">
7  <!--      <context:component-scan base-package="com.atguigu.service">
    </context:component-scan>-->
8
9      <!--dubbo的配置 -->
10     <!--1.配置项目的名称，项目的名称需要唯一 -->
11     <dubbo:application name="dubbo-service"></dubbo:application>
12     <!--2.配置注册中心的地址 -->
13     <dubbo:registry address="zookeeper://192.168.248.132:2181"></dubbo:registry>
14     <!--3.配置Dubbo的包扫描。扫描Service接口所在包 -->
15     <!-- 扫描指定包，加入@Service注解的类会被发布为服务 -->
16     <dubbo:annotation package="com.atguigu.service"></dubbo:annotation>
17
18 </beans>

```

- 4.service接口

```

1  package com.atguigu.service;
2
3  public interface UserService {
4      public String sayHello();
5  }

```

- 5.Service实现类，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

```

1  package com.atguigu.service.impl;
2
3  import com.atguigu.service.UserService;
4  import org.apache.dubbo.config.annotation.Service;
5
6  @Service // 这个是Dubbo提供的Service注解，将这个类提供的方法（服务）对外发布，将访问的地址，ip，端口，路径注册到注册中心中
7  public class UserServiceImpl implements UserService {
8      @Override
9      public String sayHello() {
10         return "hello dubbo";
11     }
12 }

```

4.3 服务消费者和@Reference

- 1.pom.xml文件

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5          http://maven.apache.org/xsd/maven-4.0.0.xsd">
6      <parent>
7          <artifactId>IdeaProject1</artifactId>
8          <groupId>org.example</groupId>
9          <version>1.0-SNAPSHOT</version>
10     </parent>
11     <modelVersion>4.0.0</modelVersion>
12     <artifactId>dubbo-web</artifactId>
13
14     <!-- 这个项目需要打成war包-->
15     <packaging>war</packaging>
16
17     <properties>
18         <spring.version>5.1.9.RELEASE</spring.version>
19         <dubbo.version>2.7.4.1</dubbo.version>
20         <zookeeper.version>4.0.0</zookeeper.version>
21     </properties>
22
23     <dependencies>
24         <!-- servlet3.0规范的坐标 -->
25         <dependency>
26             <groupId>javax.servlet</groupId>
27             <artifactId>javax.servlet-api</artifactId>
28             <version>3.1.0</version>
29             <scope>provided</scope>
30         </dependency>
31         <!--spring的坐标-->
32         <dependency>
33             <groupId>org.springframework</groupId>
34             <artifactId>spring-context</artifactId>
35             <version>${spring.version}</version>
36         </dependency>
37         <!--springmvc的坐标-->
38         <dependency>
39             <groupId>org.springframework</groupId>
40             <artifactId>spring-webmvc</artifactId>
41             <version>${spring.version}</version>
42         </dependency>
43
44         <!--日志-->
45         <dependency>
46             <groupId>org.slf4j</groupId>
47             <artifactId>slf4j-api</artifactId>
48             <version>1.7.21</version>
49         </dependency>
50         <dependency>
51             <groupId>org.slf4j</groupId>
52             <artifactId>slf4j-log4j12</artifactId>
53             <version>1.7.21</version>
```

```

54         </dependency>
55
56
57
58         <!--Dubbo的起步依赖，版本2.7之后统一为rg.apache.dubb -->
59         <dependency>
60             <groupId>org.apache.dubbo</groupId>
61             <artifactId>dubbo</artifactId>
62             <version>${dubbo.version}</version>
63         </dependency>
64         <!--ZooKeeper客户端实现 -->
65         <dependency>
66             <groupId>org.apache.curator</groupId>
67             <artifactId>curator-framework</artifactId>
68             <version>${zookeeper.version}</version>
69         </dependency>
70         <!--ZooKeeper客户端实现 -->
71         <dependency>
72             <groupId>org.apache.curator</groupId>
73             <artifactId>curator-recipes</artifactId>
74             <version>${zookeeper.version}</version>
75         </dependency>
76
77     </dependencies>
78
79
80     <build>
81         <plugins>
82             <!--tomcat插件-->
83             <plugin>
84                 <groupId>org.apache.tomcat.maven</groupId>
85                 <artifactId>tomcat7-maven-plugin</artifactId>
86                 <version>2.1</version>
87                 <configuration>
88                     <port>8000</port>
89                     <path>/</path>
90                 </configuration>
91             </plugin>
92         </plugins>
93     </build>
94
95 </project>

```

- 2.web.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xmlns="http://java.sun.com/xml/ns/javaee"
4          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5                              http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6          version="2.5">
7
8      <!-- Springmvc -->
9      <servlet>
10         <servlet-name>springmvc</servlet-name>
11         <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

```

```

12         <init-param>
13             <param-name>contextConfigLocation</param-name>
14             <param-value>classpath:spring/springmvc.xml</param-value>
15         </init-param>
16     </servlet>
17
18     <servlet-mapping>
19         <servlet-name>springmvc</servlet-name>
20         <!--
21             如果在 web.xml 中配置的 *.do, 则会拦截所有的 .do 请求去匹配
22             但是在 Controller 中的 RequestMapping 的 Value 如果是字符串没有不是 .do 结尾,
23             那么 Spring MVC 会默认的加上 .do,
24             web.xml 中配置了url-pattern后, 会起到两个作用:
25             (1) 是限制 url 的后缀名, 只能为".do"。
26             (2) 就是在没有填写后缀时, 默认在你配置的 Controller 的 RequestMapping 中添
27             加".do"的后缀。
28             -->
29         <url-pattern>*.do</url-pattern>
30     </servlet-mapping>
31 </web-app>

```

- 3.springmvc的配置文件, 配置文件需要加入注册中心相关信息

```

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:dubbo="http://dubbo.apache.org/schema/dubbo"
5         xmlns:mvc="http://www.springframework.org/schema/mvc"
6         xmlns:context="http://www.springframework.org/schema/context"
7         xsi:schemaLocation="http://www.springframework.org/schema/beans
8             http://www.springframework.org/schema/beans/spring-beans.xsd
9             http://www.springframework.org/schema/mvc
10             http://www.springframework.org/schema/mvc/spring-mvc.xsd
11             http://dubbo.apache.org/schema/dubbo
12             http://dubbo.apache.org/schema/dubbo/dubbo.xsd
13             http://www.springframework.org/schema/context
14             https://www.springframework.org/schema/context/spring-context.xsd">
15  <mvc:annotation-driven></mvc:annotation-driven>
16  <context:component-scan base-package="com.atguigu.controller">
17  </context:component-scan>
18  <!--dubbo的配置 -->
19  <!--1.配置项目的名称, 项目的名称需要唯一 -->
20  <dubbo:application name="dubbo-web"></dubbo:application>
21  <!--2.配置注册中心的地址 -->
22  <dubbo:registry address="zookeeper://192.168.248.132:2181"></dubbo:registry>
23  <!--3.配置Dubbo的包扫描。-->
24  <dubbo:annotation package="com.atguigu.controller"></dubbo:annotation>
25  </beans>

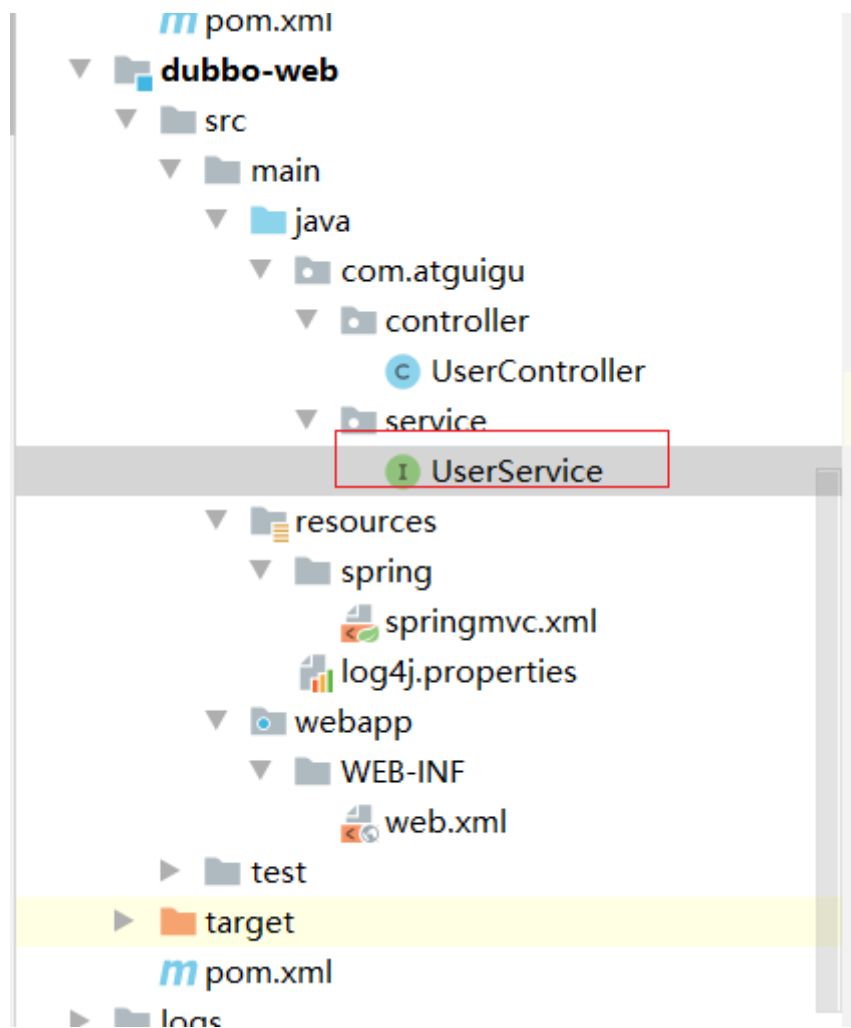
```

- 4.编写Service: 由于此时的Dubbo-service和Dubbo-web是两个不同的模块, 故此时依赖中是没有Dubbo-service的代码的, 我们暂时是将service接口写在了dubbo-web工程, 也就是:


```

1 package com.atguigu.service;
2
3 public interface UserService {
4     public String sayHello();
5 }

```



但是后面我们会解决这个问题。

- 5.编写controller

```

1 package com.atguigu.controller;
2
3
4 import com.atguigu.service.UserService;
5 import org.apache.dubbo.config.annotation.Reference;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 @RequestMapping("/user")
12 public class UserController {
13     // @Autowired // 本地注入
14     /**
15      * 1.从zookeeper注册中心获取userService的访问
16      * 2.进行远程调用RPC
17      * 3.将结果封装为一个代理对象，给变量赋值
18      */
19     @Reference //远程注入

```

```

20     private UserService userService;
21
22     @RequestMapping("/sayHello")
23     public String sayHello(){
24         return userService.sayHello();
25     }
26
27 }

```

4.3.1 解决案例日志报错的问题

我们在改造上述案例进行访问的时候，发现虽然访问成功，但是日志报错：

```

INFO 2022-06-06 15:37:51.436 org.apache.catalina.connector.impl.EnsembleTacker: new config event received. 17
ERROR 2022-06-06 15:37:51.633 org.apache.dubbo.qos.server.Server: [DUBBO] qos-server can not bind localhost:22222, dubbo version: 2.7.4.1, curr
java.net.BindException: Address already in use: bind
    at sun.nio.ch.Net.bind0(Native Method)
    at sun.nio.ch.Net.bind(Net.java:433)

```

这是由于服务提供方和服务消费方都会使用22222作为qos-server的端口，我们让其中一个工程的qos-server的端口为其他值即可！

```

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:dubbo="http://dubbo.apache.org/schema/dubbo"
5      xmlns:mvc="http://www.springframework.org/schema/mvc"
6      xmlns:context="http://www.springframework.org/schema/context"
7      xsi:schemaLocation="http://www.springframework.org/schema/beans
8          http://www.springframework.org/schema/beans/spring-beans.xsd
9          http://www.springframework.org/schema/mvc
10         http://www.springframework.org/schema/mvc/spring-mvc.xsd
11         http://dubbo.apache.org/schema/dubbo
12         http://dubbo.apache.org/schema/dubbo/dubbo.xsd
13         http://www.springframework.org/schema/context
14         https://www.springframework.org/schema/context/spring-context.xsd">
15  <mvc:annotation-driven></mvc:annotation-driven>
16  <context:component-scan base-package="com.atguigu.controller">
17  </context:component-scan>
18  <!--dubbo的配置 -->
19  <!--1.配置项目的名称，项目的名称需要唯一 -->
20  <dubbo:application name="dubbo-web">
21      <dubbo:parameter key="qos.port" value="44444"></dubbo:parameter>
22  </dubbo:application>
23  <!--2.配置注册中心的地址 -->
24  <dubbo:registry address="zookeeper://192.168.248.132:2181"></dubbo:registry>
25  <!--3.配置Dubbo的包扫描。-->
26  <dubbo:annotation package="com.atguigu.controller"></dubbo:annotation>
27  </beans>

```

思考一： 上面的Dubbo入门案例中我们是将Service接口从服务提供者工程(dubbo_service)复制到服务消费者工程(dubbo-web)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(dubbo-web)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Service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。

思考三：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。

4.4 Dubbo配置说明

4.4.1 包扫描

```
1 <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：

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

作为服务消费者，可以通过如下配置来引用服务：

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

4.4.2 协议

```
1 <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：

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

5.Dubbo高级特性 ☆ 🐘

5.1 管理控制台Dubbo-admin

dubbo-admin管理平台，是 **图形化的服务管理页面**，提供了

- 从 **注册中心** 中获取到所有的提供者/消费者进行配置管理，故dubbo-admin中需要配置注册中心地址；
- 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

实际上，dubbo-admin是一个前后端分离的项目。前端使用vue，后端使用springboot，我们安装dubbo-admin其实就是部署该项目。

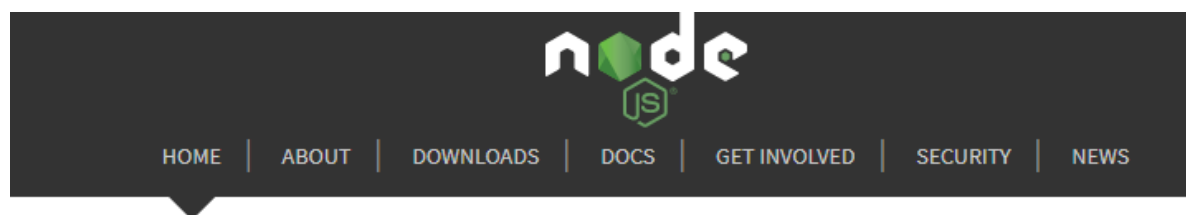
1、环境准备

dubbo-admin 是一个前后端分离的项目。前端使用vue，后端使用springboot，安装 dubbo-admin 其实就是部署该项目。我们将dubbo-admin安装到开发环境上。要保证开发环境有jdk, maven, nodejs

安装node(如果当前机器已经安装请忽略)

因为前端工程是用vue开发的，所以需要安装node.js，node.js中自带了npm，后面我们会通过npm启动
下载地址

```
1 https://nodejs.org/en/
```



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

12.14.0 LTS

Recommended For Most Users

13.5.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

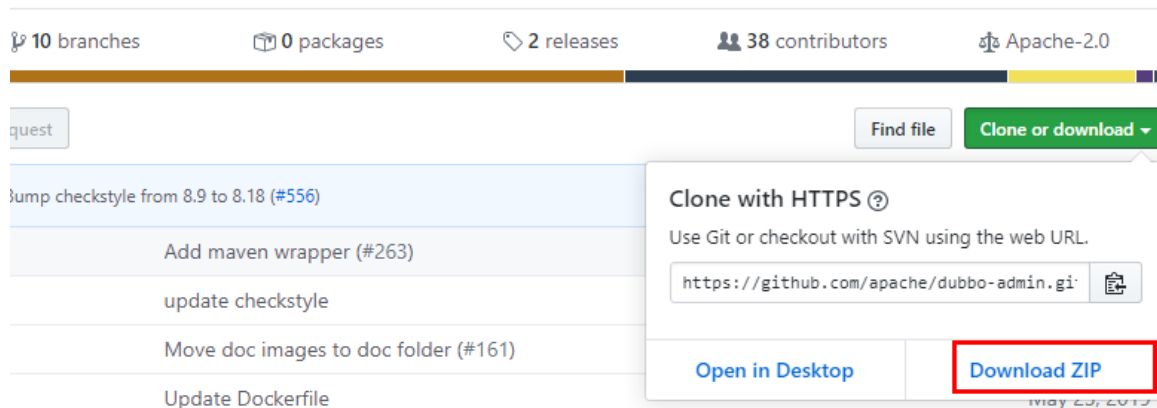
[Other Downloads](#) | [Changelog](#) | [API Docs](#)

2、下载 Dubbo-Admin

进入github，搜索dubbo-admin

```
1 https://github.com/apache/dubbo-admin
```

下载：



3、把下载的zip包解压到指定文件夹(解压到那个文件夹随意)

名称

dubbo-admin-develop
dubbo-admin-develop.zip

4、修改配置文件

我们需要在配置文件中修改注册中心的配置等信息。

admin.registry.address注册中心

admin.config-center 配置中心

admin.metadata-report.address元数据中心

5、打包项目

在 dubbo-admin-develop 目录执行打包命令

```
1 mvn clean package
```

```
main:
[copy] Copying 1 file to E:\software\dubbo-admin-develop\dubbo-admin-distribution\target
[INFO] Executed tasks
[INFO] -----
[INFO] Reactor Summary for dubbo-admin 0.1:
[INFO] dubbo-admin ..... SUCCESS [ 2.907 s]
[INFO] dubbo-admin-ui ..... SUCCESS [01:24 min]
[INFO] dubbo-admin-server ..... SUCCESS [02:21 min]
[INFO] dubbo-admin-distribution ..... SUCCESS [ 4.226 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 03:54 min
[INFO] Finished at: 2020-01-06T16:47:01+08:00
[INFO] -----
```

6、启动后端

切换到目录

```
1 dubbo-Admin-develop\dubbo-admin-distribution\target>
```

执行下面的命令启动 dubbo-admin, dubbo-admin后台由SpringBoot构建。

```
1 java -jar .\dubbo-admin-0.1.jar
```

7、前台后端

dubbo-admin-ui 目录下执行命令

```
1 npm run dev
```

```
DONE Compiled successfully in 1407ms
```

16:50:55

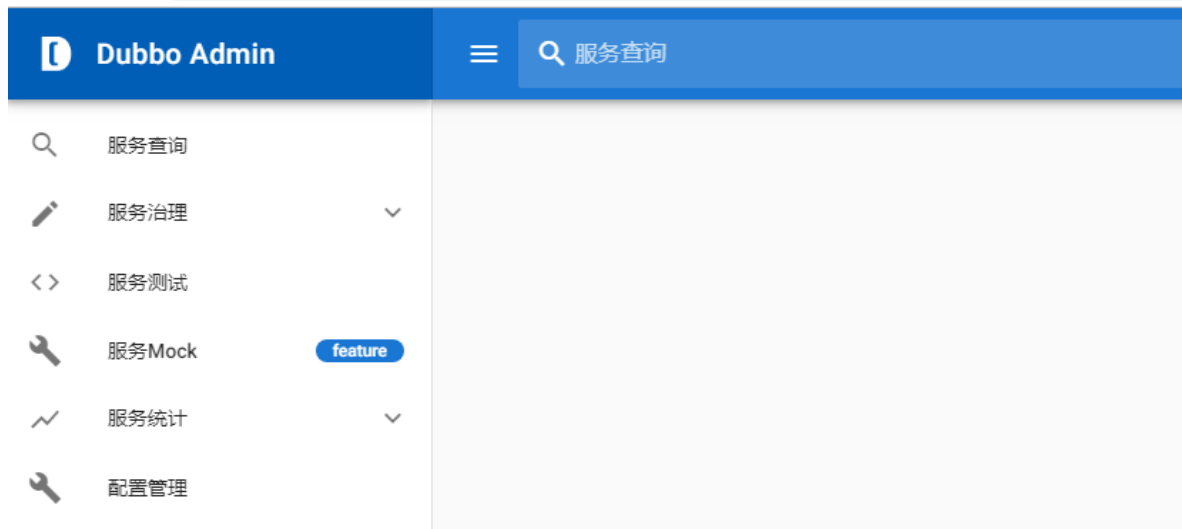
```
1 Your application is running here: http://localhost:8081
```

8、访问

浏览器输入。用户名密码都是root

```
1 http://localhost:8081/
```

← → ↻ localhost:8081/#/



5.2 Dubbo自带的控制台

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。这个是Dubbo自带的，不如上面美观。

5.2.1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://192.168.253.124:2181
```

```
dubbo.admin.root.password=root
```

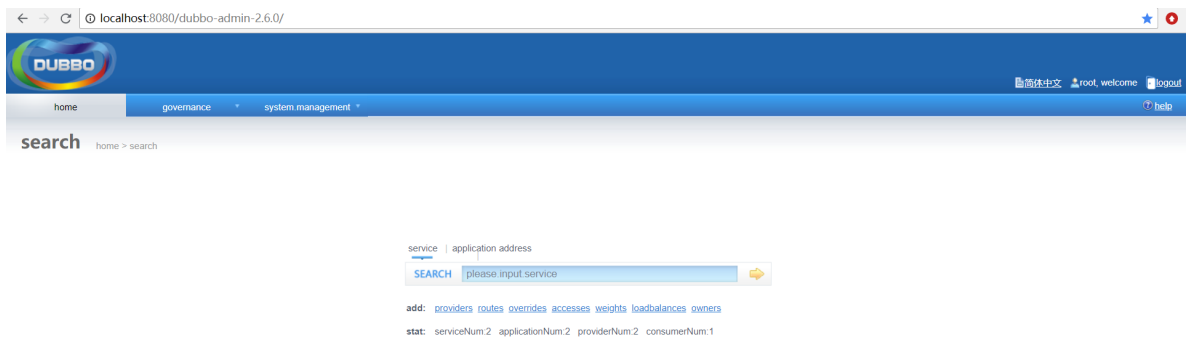
```
dubbo.admin.guest.password=guest
```

- (4) 重启tomcat

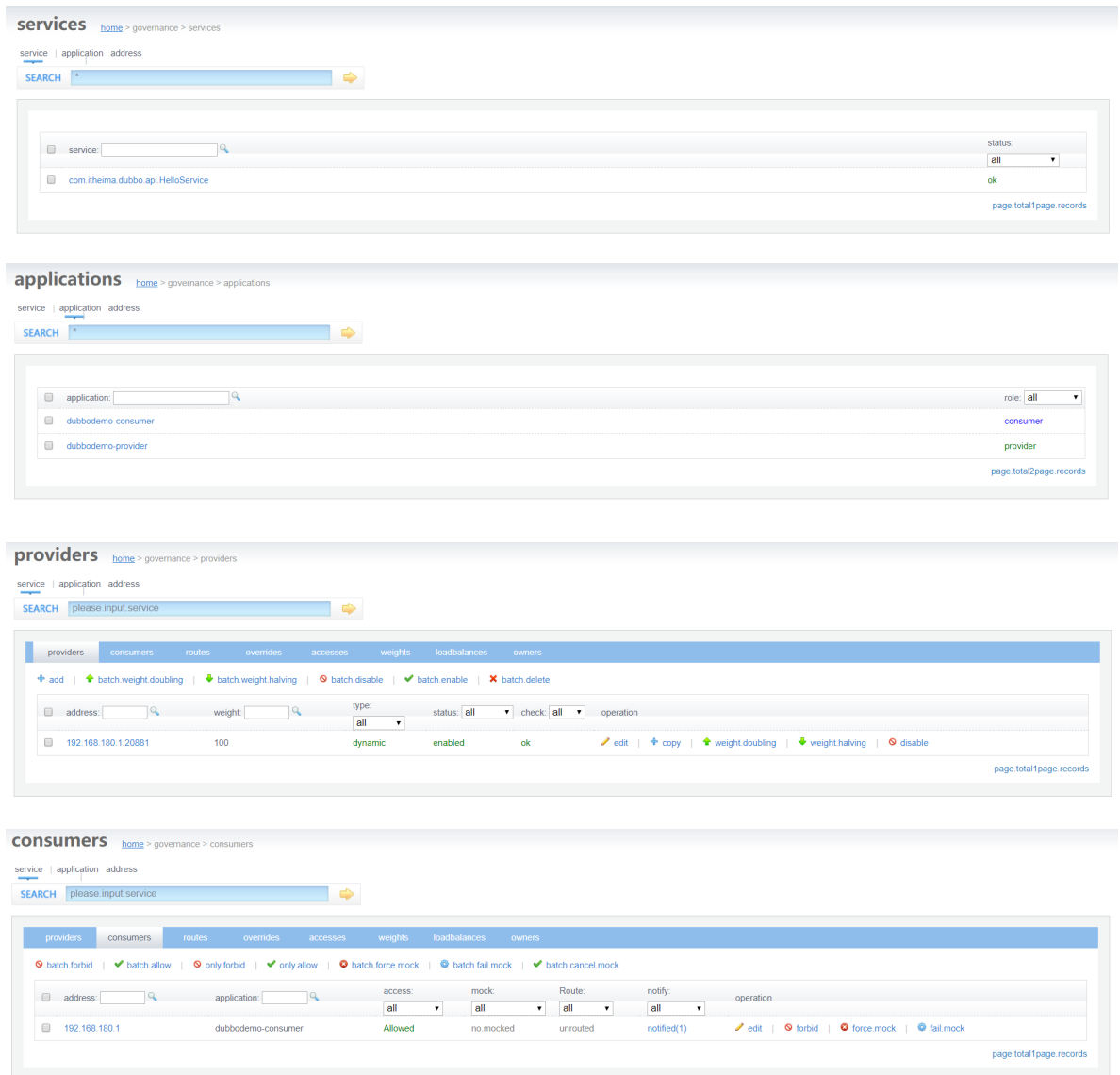
5.2.2使用

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



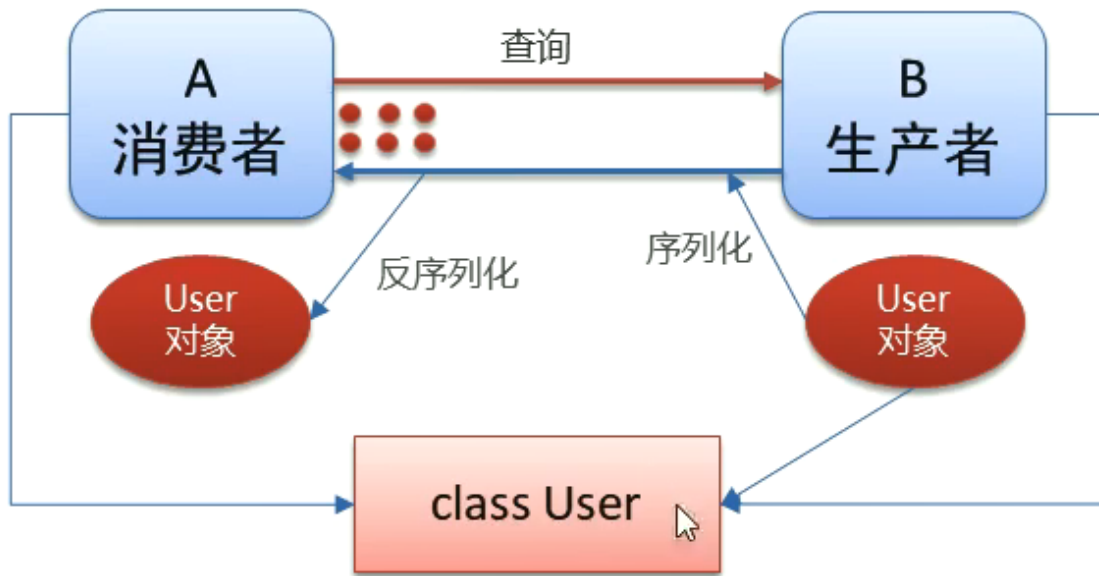
5.3 序列化☆🐼

我们在生产者和消费者之间如果需要传递java对象，此时就需要用到 **java的序列化**。

1. dubbo 内部已经将序列化和反序列化的过程内部封装了
2. 我们只需要在定义pojo类时**实现serializable接口**即可
3. 一般会定义一个公共的pojo模块,让生产者和消费者都依赖该模块。



两个机器传输数据，如何传输Java对象？



实现序列化接口 `implements Serializable`

我们先创建一个实体类模块，让服务端和客户端都依赖这个实体类模块。实体类模块工程如下：dubbo-pojo

到时候服务消费端和服务提供端都引入这个模块依赖：

```
1 <dependency>
2     <groupId>org.example</groupId>
3     <artifactId>dubbo-pojo</artifactId>
4     <version>1.0-SNAPSHOT</version>
5 </dependency>
```

```
1 package com.atguigu.pojo;
2
3 import java.io.Serializable;
4
5 /**
6  * 注意：将来所有的pojo类都需要实现Serializable接口！
7  */
8 public class User implements Serializable {
9     private String id;
10    private String username;
11    private String password;
12
13    public User() {
14    }
15
16    public User(String id, String username, String password) {
17        this.id = id;
18        this.username = username;
19        this.password = password;
20    }
21
22    public String getId() {
```

```

23         return id;
24     }
25
26     public void setId(String id) {
27         this.id = id;
28     }
29
30     public String getUsername() {
31         return username;
32     }
33
34     public void setUsername(String username) {
35         this.username = username;
36     }
37
38     public String getPassword() {
39         return password;
40     }
41
42     public void setPassword(String password) {
43         this.password = password;
44     }
45 }

```

此时我们对Service工程做改造。

接口

```

1  package com.atguigu.service;
2
3  import com.atguigu.pojo.User;
4
5  public interface UserService {
6      public String sayHello();
7
8      /**
9       * 查询用户
10      */
11     public User findUserById(String id);
12 }

```

实现类

```

1  package com.atguigu.service.impl;
2
3  import com.atguigu.pojo.User;
4  import com.atguigu.service.UserService;
5  import org.apache.dubbo.config.annotation.Service;
6
7  @Service // 这个是Dubbo提供的Service注解，将这个类提供的方法（服务）对外发布，将访问的地址，ip，端口，路径注册到注册中心中
8  public class UserServiceImpl implements UserService {
9
10     public String sayHello() {
11         return "hello dubbo";
12     }
13
14     public User findUserById(String id) {

```

```

15         // 查询user对象
16         User user = new User(id, "zhangsang", "12");
17         System.out.println(user.hashCode());
18         return user;
19     }
20 }
21
22

```

我们再将消费端接口做改造：

控制器方法

```

1  package com.atguigu.controller;
2
3
4  import com.atguigu.pojo.User;
5  import com.atguigu.service.UserService;
6  import org.apache.dubbo.config.annotation.Reference;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 @RequestMapping("/user")
12 public class UserController {
13     // @Autowired // 本地注入
14     /**
15      * 1.从zookeeper注册中心获取userService的访问
16      * 2.进行远程调用RPC
17      * 3.将结果封装为一个代理对象，给变量赋值
18      */
19     @Reference // 远程注入
20     private UserService userService;
21
22     @RequestMapping("/sayHello")
23     public String sayHello(){
24         return userService.sayHello();
25     }
26
27     @RequestMapping("/find")
28     public User find(){
29         User user = userService.findUserById("2");
30         System.out.println(user.hashCode());
31         return user;
32     }
33 }
34
35

```

注意：实体类一定要实现Serializable接口，否则会出现异常

```

] with root cause
org.apache.dubbo.remoting.RemotingException: Failed to send response: Response [id=2, version=2.0.2, status=20, event=false, err
java.lang.IllegalStateException: Serialized class com.atguigu.pojo.User must implement java.io.Serializable
    at com.alibaba.com.caucho.hessian.io.SerializerFactory.getDefaultSerializer(SerializerFactory.java:405)
    at com.alibaba.com.caucho.hessian.io.SerializerFactory.getSerializer(SerializerFactory.java:379)

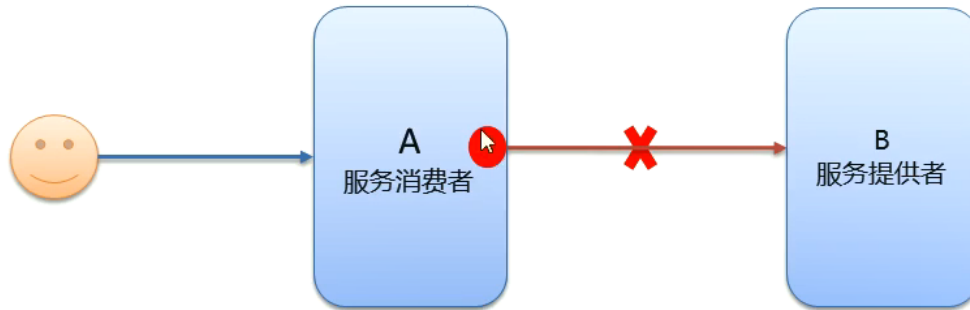
```

5.4 地址缓存 ☆ 🐼

注册中心挂了，服务是否可以正常访问？

1. 可以，因为dubbo服务消费者在第一次调用时，会将服务提供方地址缓存到本地，以后在调用则不会访问注册中心。
2. 当服务提供者地址发生变化时，注册中心会通知服务消费者。

5.5 超时 ☆ 🐼



- 服务消费者在调用服务提供者时发生了阻塞、等待的情形,这个时候,服务消费者会直等待下去。
- 在某个峰值时刻，大量的请求都在同时请求服务消费者,会造成线程的大量堆积，势必会造成雪崩。

为了解决上述可能出现的问题，dubbo采用了如下方案：

dubbo利用超时机制来解决这个问题，设置一个超时时间, 在这个时间段内，无法完成服务访问,则自动断开连接。我们还可以设置超时时间：我们可以在@Service注解或者@Reference注解上配置，这个超时时间默认是1000毫秒。**建议配置在服务提供方：谁定义服务，谁才知道服务大概耗时。**

```
1 //timeout 超时时间 单位毫秒 retries 重试次数
2 @Service(timeout = 3000,retries=0)
```

5.6 重试 ☆ 🐼

超时机制的规则是如果在一定的时间内，provider没有返回，则认为本次调用失败，

重试机制在出现调用失败时，会再次调用。如果在配置的调用次数内都失败，则认为此次请求异常，抛出异常。

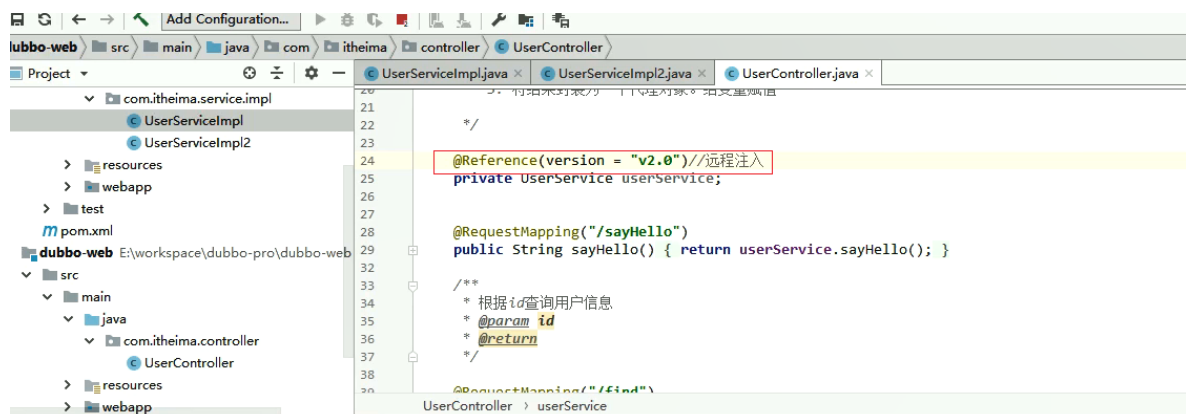
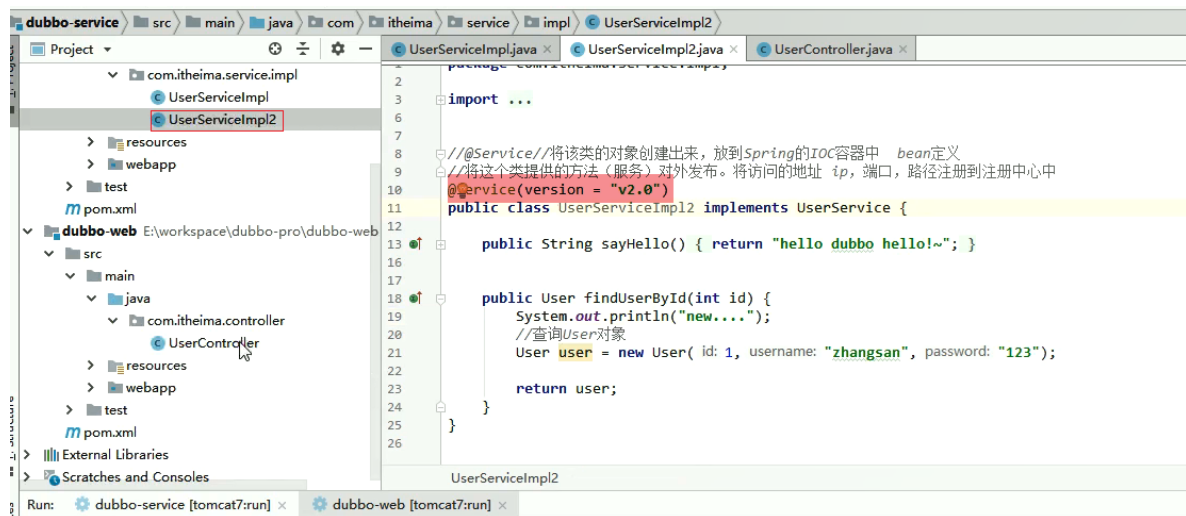
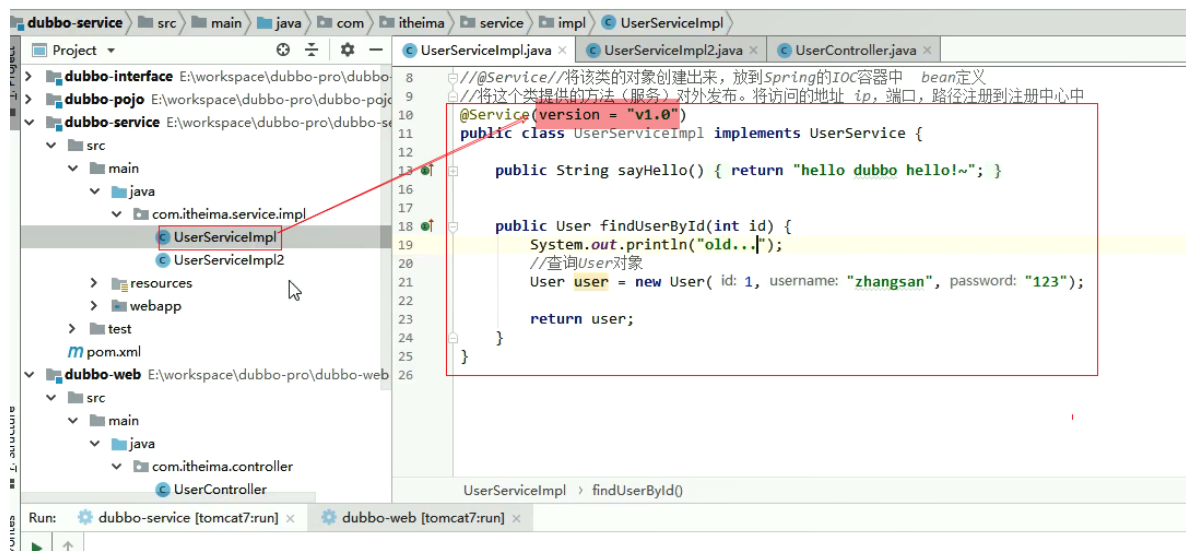
1. 设置了超时时间，在这个时间段内，无法完成服务访问,则自动断开连接。
2. 如果出现网络抖动,则这一次请求就会失败。
3. Dubbo提供重试机制来避免类似问题的发生。
4. 通过retries属性来设置重试次数。默认为2次
5. 如果消费者配置了重试次数，提供者也配置了重试次数，则以消费者为准；

```
1 //timeout 超时时间 单位毫秒 retries 重试次数
2 @Service(timeout = 3000,retries=0)
```

5.7 多版本 ☆

灰度发布:当出现新功能时,会让一部分用户先使用新功能，用户反馈没问题时，再将所有用户迁移到新功能。

dubbo中使用version属性来设置和调用同一个接口的不同版本



5.8 负载均衡 ☆ 🐼

配置在消费端。

为了演示负载均衡的效果，我们可以让服务提供者启动三次，每次注意修改和端口有关的配置，这样就相当于搭建了一个伪集群。

修改1: pom.xml的tomcat插件的端口

```

1  <build>
2      <plugins>
3          <!--tomcat插件-->
4          <plugin>
5              <groupId>org.apache.tomcat.maven</groupId>
6              <artifactId>tomcat7-maven-plugin</artifactId>

```

```

7         <version>2.1</version>
8         <configuration>
9             <port>8000</port>
10            <path>/</path>
11        </configuration>
12    </plugin>
13 </plugins>
14 </build>

```

修改2: 修改protocol和qos.port的端口。

```

<dubbo:protocol port="20880"></dubbo:protocol>
<dubbo:application name="dubbo-service1"> 注意。集群中的application.name不用修改
    <dubbo:parameter key="qos.port" value="22222"></dubbo:parameter>
</dubbo:application>

```

负载均衡策略(4种):

- Random: 按权重随机, 默认值。按权重设置随机概率。
- RoundRobin: 按权重轮询。
- LeastActive: 最少活跃调用数, 相同活跃数的随机。谁最快调用谁
- ConsistentHash: 一致性Hash, 相同参数的请求总是发到同一提供者。

负载均衡的配置如下:

```

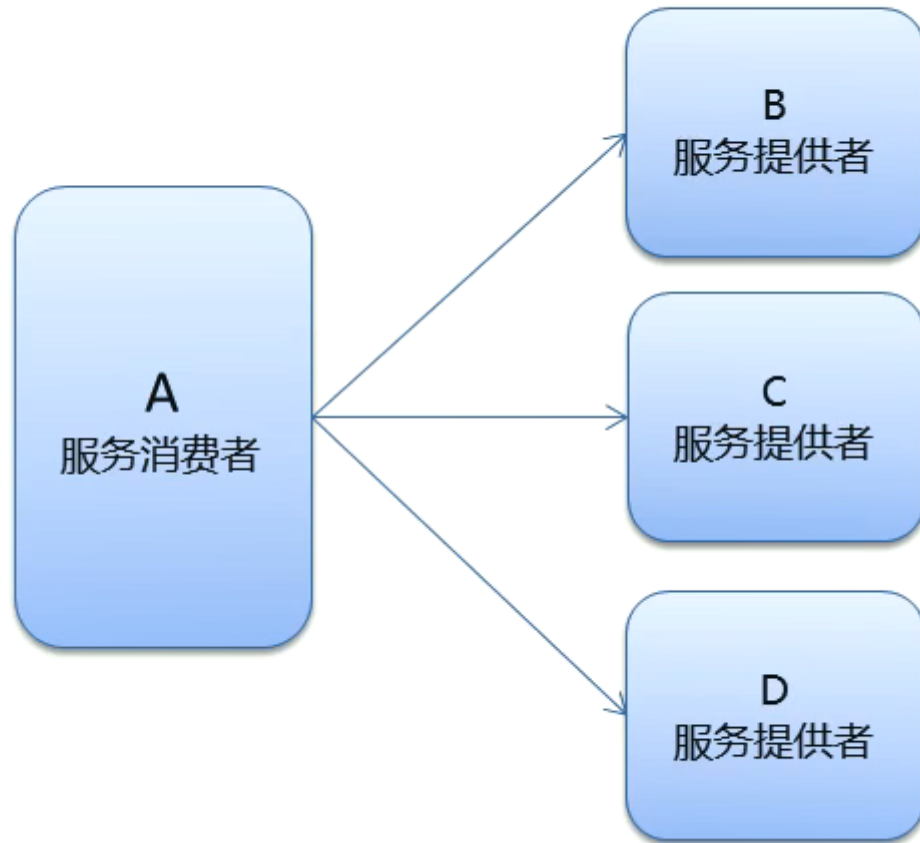
1 // @Reference(loadbalance = "roundrobin")
2 // @Reference(loadbalance = "leastactive")
3 // @Reference(loadbalance = "consistenthash")
4 @Reference(loadbalance = "random") // 默认 按权重随机
5 private UserService userService;

```

5.9 集群容错 ☆ 🐼

容错策略配置在消费端。

集群容错



策略如下：

Failover Cluster: 失败重试。默认值。当出现失败，重试其它服务器，默认重试2次，使用retries配置。一般用于读操作

Failfast Cluster : 快速失败,发起一次调用，失败立即报错。通常用于写操作。也就是不重试，直接报错。

Failsafe Cluster: 失败安全，出现异常时，直接忽略。返回一个空结果。

Failback Cluster: 失败自动恢复,后台记录失败请求,定时重发。也就是一定要成功，不成功一定会继续发送，直到成功

Forking Cluster : 并行调用多个服务器，只要一个成功即返回。广撒网！

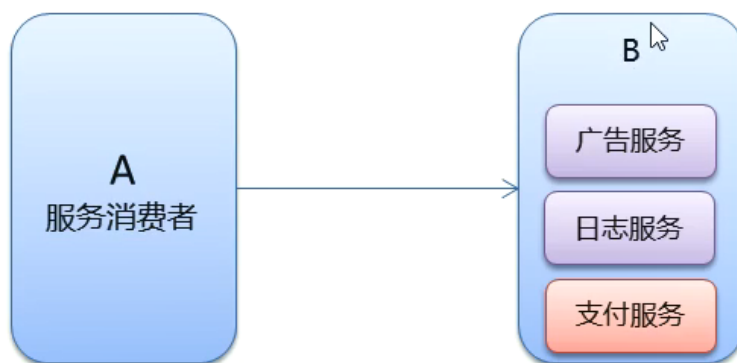
Broadcast Cluster: 广播调用所有提供者,逐个调用，任意一台报错则报错。同时成功才算成功！

其策略的代码配置如下：

```
1 @Reference(cluster = "failover")//远程注入
2 private UserService userService;
```

5.10 服务降级

假如说一个机器中提供有多个服务，如下所示：



此时为了性能考虑，可能会关闭不太重要的非核心服务。

服务降级：当服务器压力剧增的情况下，根据实际业务情况及流量，对一些服务和页面有策略的不处理或换种简单的方式处理，从而释放服务器资源以保证**核心交易**正常运作或高效运作

服务降级方式：

mock= force:return null：表示消费方对该服务的方法调用都直接返回null值,不发起远程调用。用来屏蔽不重要服务不可用时对调用方的影响。

mock= fail:return null：表示消费方对该服务的方法调用在失败后，再返回null值,不抛异常。用来容忍不重要服务不稳定时对调用方的影响

服务降级在2.2.0版本后才有这个功能。

其代码的配置如下：

```
1 //远程注入
2 @Reference(mock = "force:return null")//不再调用userService的服务
3 private UserService userService;
```

6.Dubbo早期版本的问题

Dubbo无法发布被事务代理的Service问题:

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。

6.1 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

```

1  <dependency>
2    <groupId>mysql</groupId>
3    <artifactId>mysql-connector-java</artifactId>
4    <version>5.1.47</version>
5  </dependency>
6  <dependency>
7    <groupId>com.alibaba</groupId>
8    <artifactId>druid</artifactId>
9    <version>1.1.6</version>
10 </dependency>
11 <dependency>
12   <groupId>org.mybatis</groupId>
13   <artifactId>mybatis-spring</artifactId>
14   <version>1.3.2</version>
15 </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

```

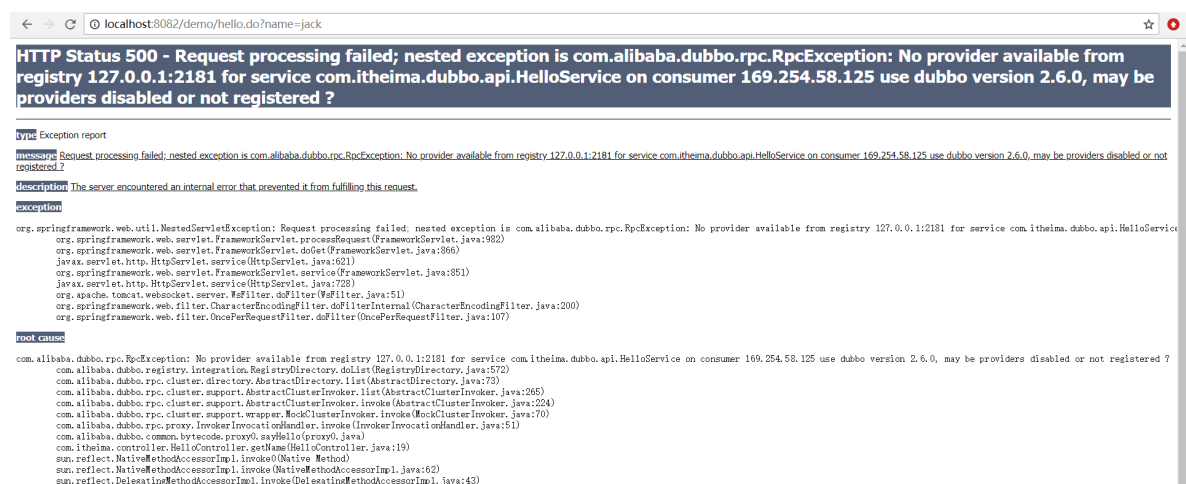
1  <!--数据源-->
2  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" destroy-
method="close">
3    <property name="username" value="root" />
4    <property name="password" value="root" />
5    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
6    <property name="url" value="jdbc:mysql://localhost:3306/test" />
7  </bean>
8  <!-- 事务管理器 -->
9  <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

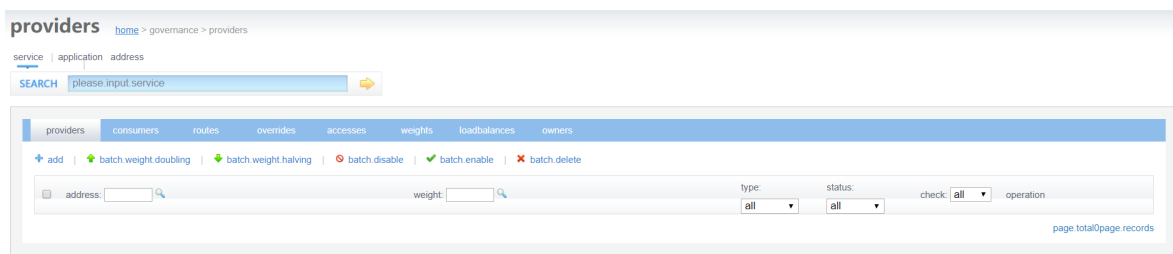
(3) 在HelloServiceImpl类上加入@Transactional注解

(4) 启动服务提供者和服务消费者，并访问

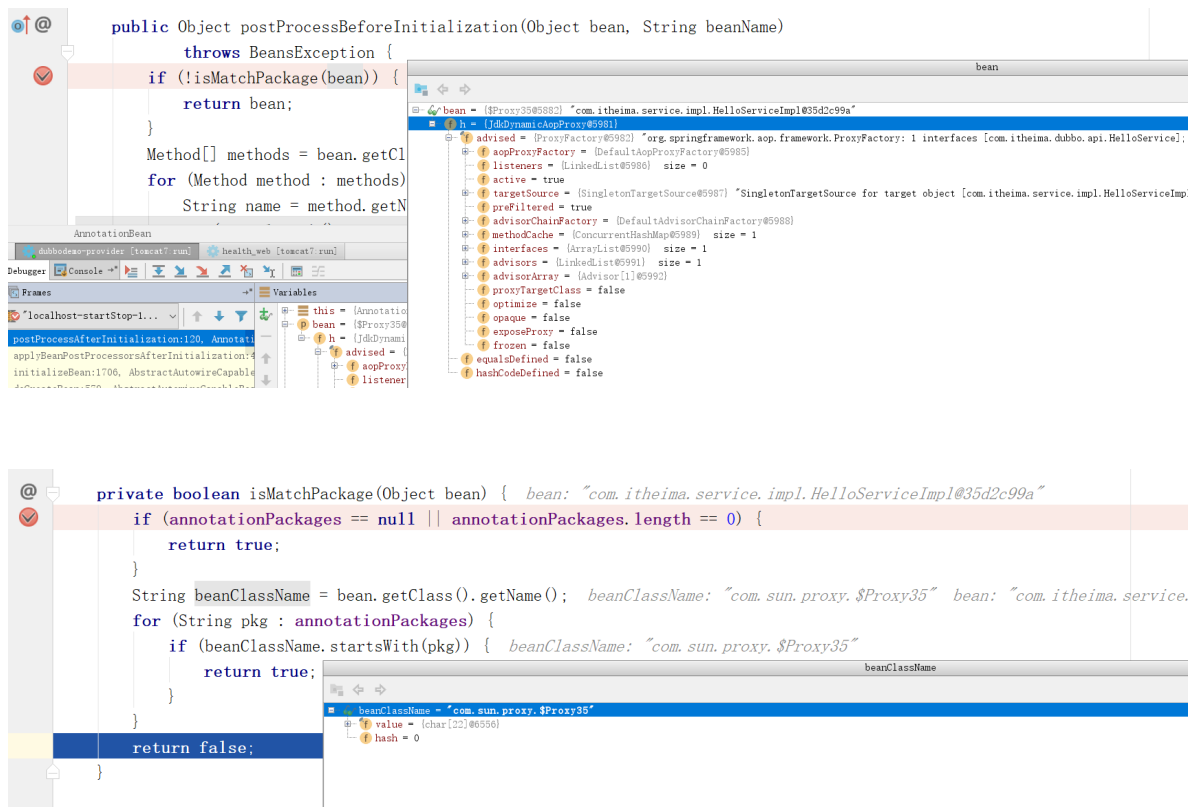


上面的错误为没有可用的服务提供者

查看dubbo管理控制台发现服务并没有发布，如下：



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



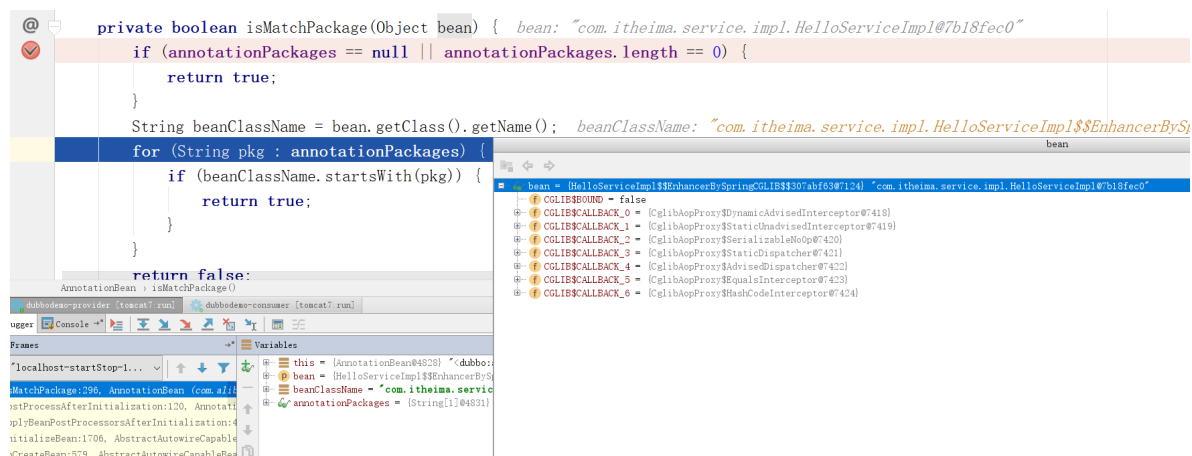
6.2 解决方案

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

- 1 <!--开启事务控制的注解支持，强制使用cglib创建代理对象-->
- 2 <tx:annotation-driven transaction-manager="transactionManager" proxy-target-class="true"/>



(2) 修改服务提供类，在Service注解中加入interfaceClass属性，值为接口类.class，作用是指定服务的接口类型

```
1 @Service(interfaceClass = HelloService.class)
2 @Transactional
3 public class HelloServiceImpl implements HelloService {
4     public String sayHello(String name) {
5         return "hello " + name;
6     }
7 }
```

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，这是由于创建的代理对象也是要实现其他接口的，故需要我们具体指定注册的接口为哪一个，如下：

