分布式系统架构解决方案—Dubbo

---- 老孙

课程目标：

1、dubbo概述

2、快速入门

3、监控中心

4、综合实战

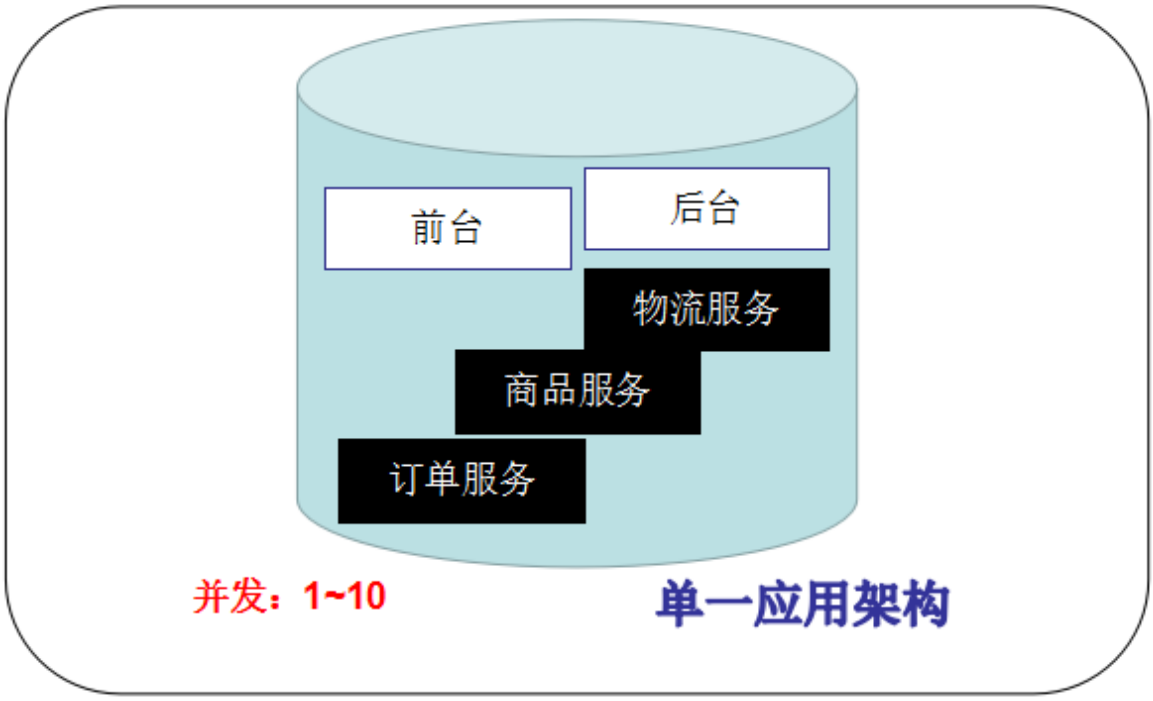
1. dubbo概述

1.1 什么是分布式系统？

- 《分布式系统原理与范型》定义：
  - “分布式系统是若干独立计算机的集合，这些计算机对于用户来说就像单个相关系统”
  - 分布式系统（distributed system）是建立在网络之上的软件系统。
  - 简单来说：多个（不同职责）人共同来完成一件事！
  - 任何一台服务器都无法满足淘宝的双十一的数据吞吐量，一定是很多台服务器公共来完成的。
- 歇后语：“三个臭皮匠赛过诸葛亮”，就是分布式系统的真实写照

1.1.1 单一应用架构

- 当网站流量很小时，只需要一个应用，将所有的功能部署到一起（所有业务都放在一个tomcat里），从而减少部署节点和成本；
- 此时，用于简化 增删改查 工作量的数据访问框架 （**ORM**）是关键；
- 例如：某个超市的收银系统，某个公司的员工管理系统

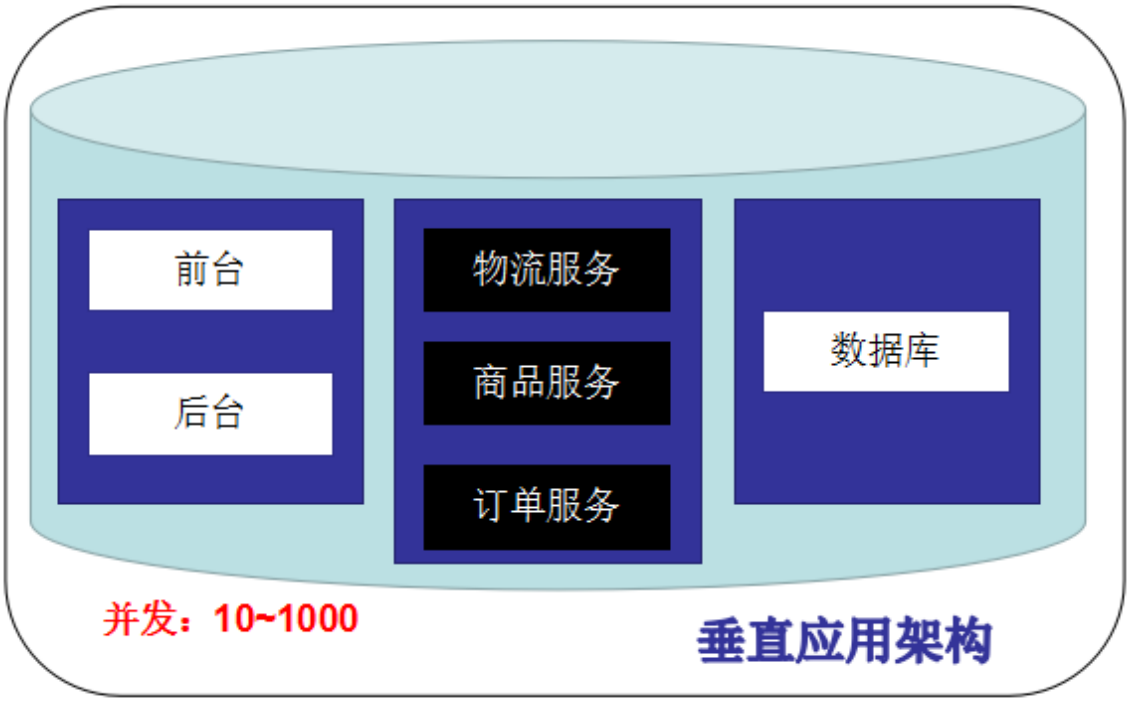


ORM：对象关系映射（Object Relational Mapping）

- **优点**
  - 小项目开发快 成本低
  - 架构简单
  - 易于测试
  - 易于部署
- **缺点**
  - 大项目模块耦合严重 不易开发 维护 沟通成本高
  - 新增业务困难
  - 核心业务与边缘业务混合在一块，出现问题互相影响

1.1.2 垂直应用架构

- 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成几个**互不相干**的几个应用,以提高效率；
- 大模块按照mvc分层模式，进行拆分成多个互不相关的小模块，并且每个小模块都有独立的服务器
- 此时，用于加速前端页面开发的web框架（**MVC**）是关键；因为每个小应用都有独立的页面

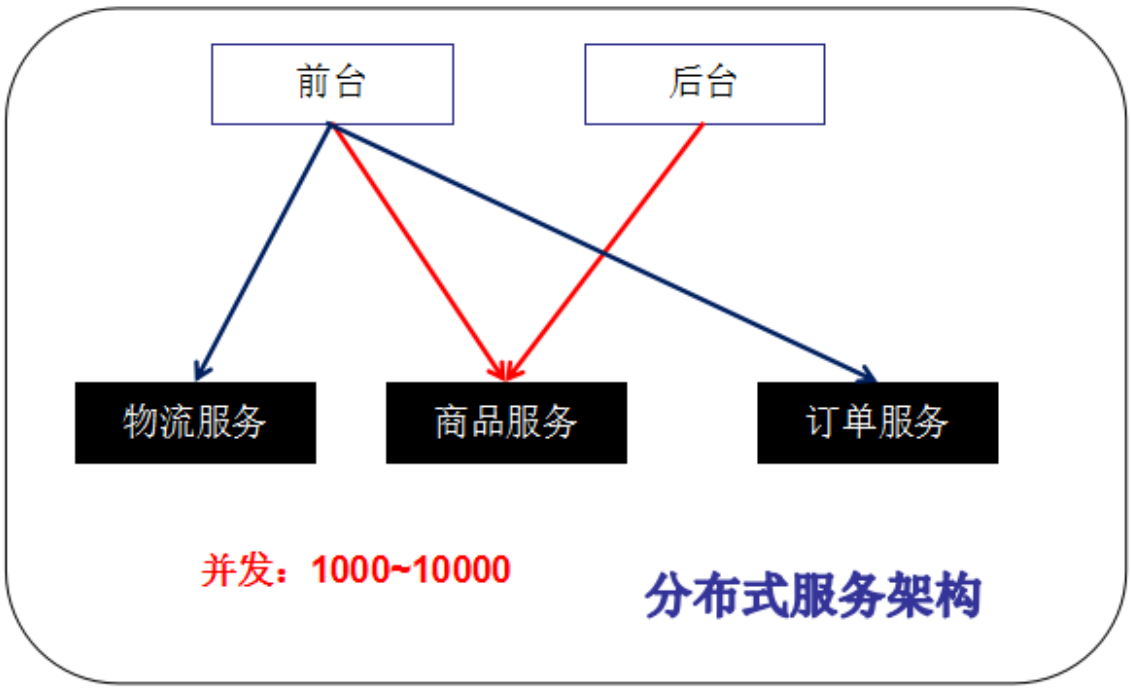


MVC：模型视图控制器（Model View Controller）

- 缺点：
  - 模块之间不可能完全没有交集，公用模块无法重复利用，开发性的浪费

1.1.3 分布式服务架构

- 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的业务，逐渐形成稳健的服务中心，使前端应用能更快速的响应多变的市场需求；
- 此时，用户提高业务复用及整合的分布式服务框架（**RPC**）远程调用是关键；



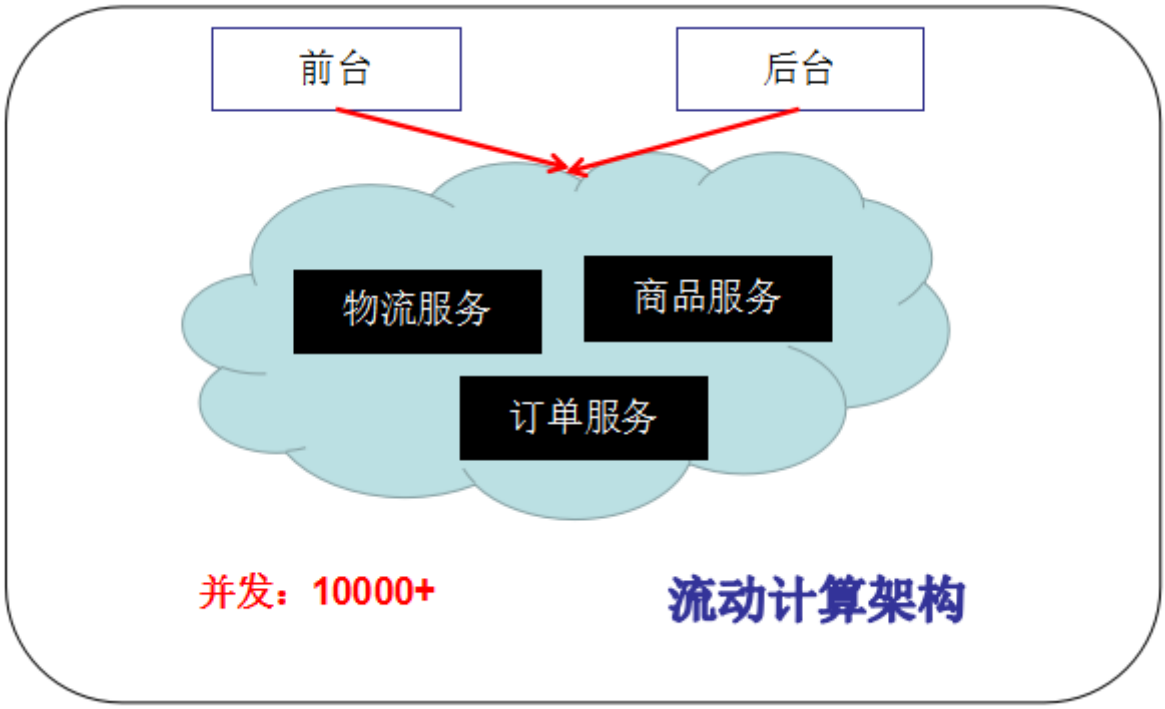
RPC：独立的应用服务器之间，要依靠RPC（Romote Procedure Call）才能调用

- 物流服务不忙，有100台服务器； 商品服务特别忙，也是100台服务器；

- 如何做到资源优化调配？↓

1.1.4 流动计算架构

- 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐呈现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率；
- 此时，用于提高机器利用率的资源调度和治理中心（**SOA**）是关键；



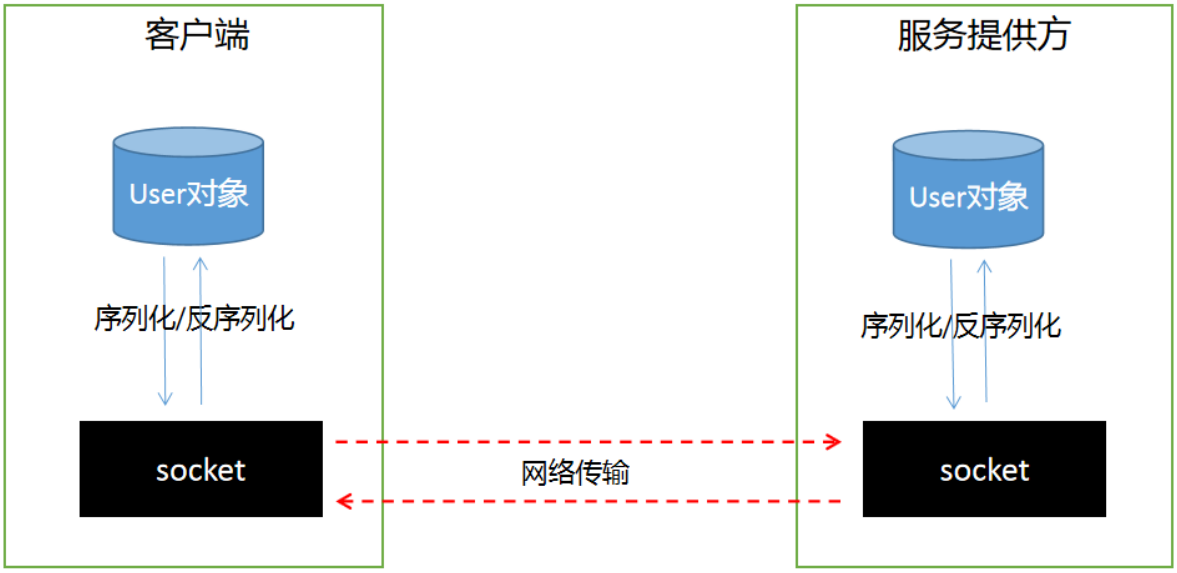
SOA：面向服务架构（Service-Oriented Architecture），简单理解就是“服务治理”，例如：公交车站的“调度员”

1.2 Dubbo简介

- Dubbo是分布式服务框架，是阿里巴巴的开源项目，现交给apache进行维护
- Dubbo致力于提高性能和透明化的RPC远程服务调用方案，以及SOA服务治理方案
- 简单来说，dubbo是个服务框架，如果没有分布式的需求，是不需要用的

1.2.1 RPC

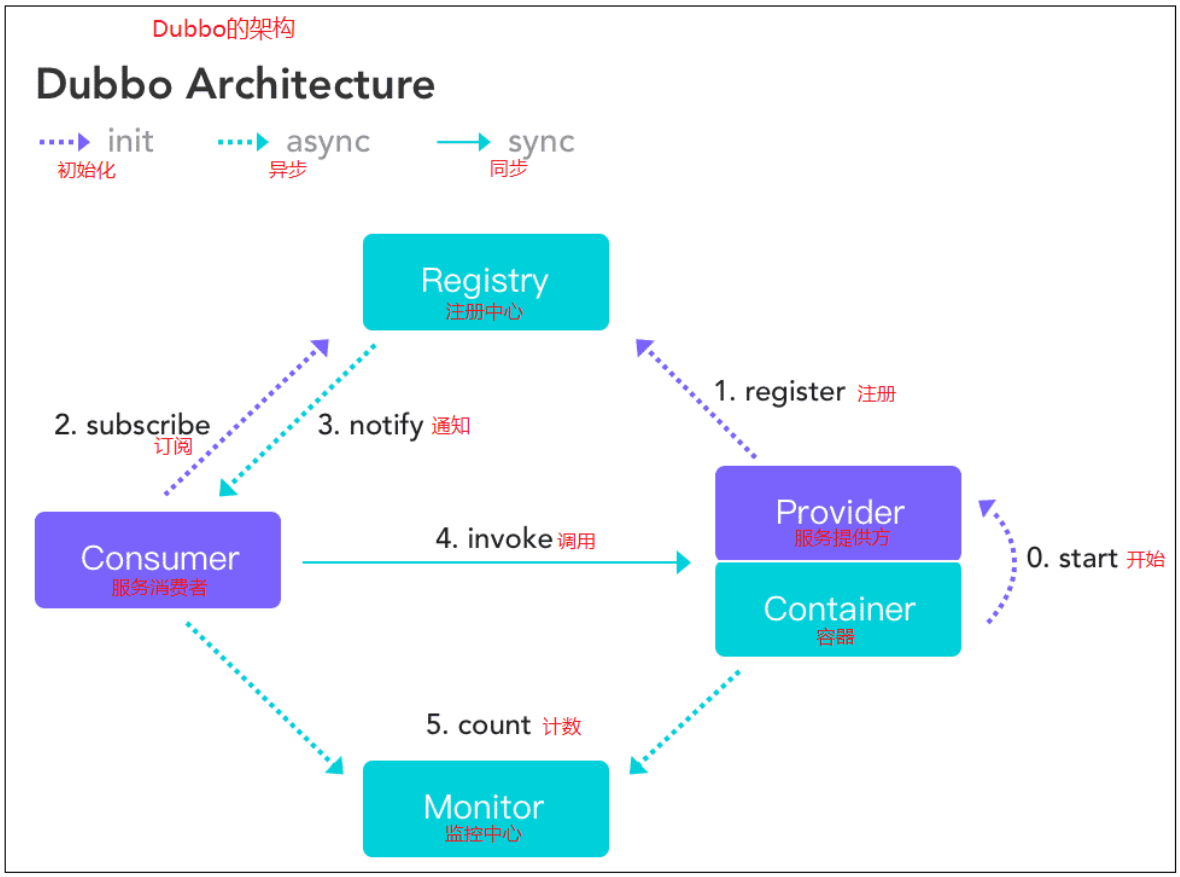
- **RPC**【Remote Procedure Call】是指远程过程调用，是一种进程间通信方式
- **RPC基本的通信原理**
  1. 在客户端将对象进行**序列化**
  2. 底层通信框架使用netty（基于tcp协议的socket），将序列化的对象发给服务方提供方
  3. 服务提供方通过socket得到数据文件之后，进行反序列化，获得要操作的对象
  4. 对象数据操作完毕，将新的对象序列化，再通过服务提供方的socket返回给客户端
  5. 客户端获得序列化数据，再反序列化，得到最新的数据对象，至此，完成一次请求



- RPC两个核心模块：**通讯**（socket），**序列化。**

1.2.2 节点角色

节点	角色说明
Provider	服务的提供方（洗浴中心）
Consumer	服务的消费方（客人）
Registry	服务注册与发现的注册中心（便民服务中心，所有的饭店娱乐场所都在已在本中心注册）
Monitor	监控服务的统计中心（统计服务被调用的次数）
Container	服务运行容器（烧烤一条街，洗浴一条街）

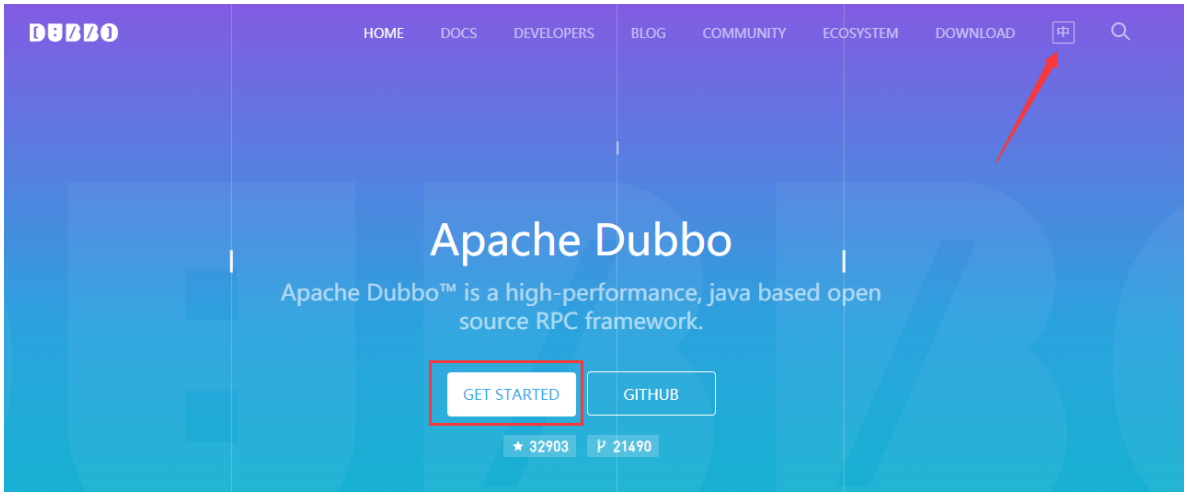


1.2.3 调用关系

1.服务容器负责启动，加载，运行服务提供者；
2.服务提供者在启动时，向注册中心注册自己提供的服务；
3.服务消费者在启动时，向注册中心订阅自己所需的服务；
4.在注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者；
5.服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用；
6.服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心；

2. 快速入门

http://dubbo.apache.org/



2.1 注册中心

2.1.1 Zookeeper

- 官方推荐使用zookeeper注册中心；
- 注册中心负责服务地址的注册与查找，相当于目录服务；
- 服务提供者和消费者只在启动时与注册中心交互，注册中不转发请求，压力较小；
- 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的服务注册中心，工业强度较高，可用于生产环境；

dubbo即是求职的人，也是招聘单位，而zookeeper就是人才市场/招聘网站；

2.1.2 安装

1、安装jdk

2、拷贝apache-zookeeper-3.6.0-bin.tar.gz到opt目录

3、解压安装包

```
[root@localhost opt]# tar -zxvf apache-zookeeper-3.6.0-bin.tar.gz
```

4、重命名

```
[root@localhost opt]# mv apache-zookeeper-3.6.0-bin zookeeper
```

5、在/opt/zookeeper/这个目录上创建zkData和zkLog目录

```
[root@localhost zookeeper]# mkdir zkData
[root@localhost zookeeper]# mkdir zkLog
```

6、进入/opt/zookeeper/conf这个路径，复制一份 zoo_sample.cfg 文件并命名为 zoo.cfg

```
[root@localhost conf]# cp zoo_sample.cfg zoo.cfg
```

7、编辑zoo.cfg文件，修改dataDir路径：

```
dataDir=/opt/zookeeper/zkData
dataLogDir=/opt/zookeeper/zkLog
```

8、启动Zookeeper

```
[root@localhost bin]# ./zkServer.sh start
```

9、查看状态：

```
[root@localhost bin]# ./zkServer.sh status
```

2.2 服务提供方

1、一个空的maven项目

2、提供一个服务接口即可

2.2.1 服务方的pom.xml

各种依赖请严格按照下面的版本

```
<packaging>war</packaging>

<properties>
    <spring.version>5.0.6.RELEASE</spring.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!--dubbo -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.5.7</version>
    </dependency>
    <dependency>
        <groupId>org.apache.zookeeper</groupId>
        <artifactId>zookeeper</artifactId>
```

```
        <version>3.4.6</version>
    </dependency>
    <dependency>
        <groupId>com.github.sgroschupf</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.1</version>
    </dependency>
    <dependency>
        <groupId>javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.11.0.GA</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven </groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <configuration>
                <port>8001</port>
                <path>/</path>
            </configuration>
             <executions>
                <execution>
                    <!-- 打包完成后,运行服务 -->
                    <phase>package</phase>
                    <goals>
                        <goal>run</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

2.2.2 服务方接口

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

2.2.3 服务方实现

```
@com.alibaba.dubbo.config.annotation.Service
public class HelloServiceImpl implements HelloService {
    @Override
    public String sayHello(String name) {
        return "Hello," + name + "!!!";
    }
}
```

2.2.4 服务方的配置文件spring.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:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://code.alibabatech.com/schema/dubbo
       http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!--1.服务提供方在zookeeper中的“别名”-->
    <dubbo:application name="dubbo-server"/>
    <!--2.注册中心的地址-->
    <dubbo:registry address="zookeeper://192.168.204.141:2181"/>
    <!--3.扫描类（将什么包下的类作为服务提供类）-->
    <dubbo:annotation package="service.impl"/>
</beans>
```

2.2.5 服务方的web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         id="WebApp_ID" version="3.1">
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring/spring.xml</param-value>
    </context-param>
</web-app>
```

2.3 服务消费方

2.3.1 消费方的pom.xml

与服务方一致，只需要修改tomcat的端口为8002

2.3.2 消费方的Controller

```
@RestController
public class HelloAction {
    @com.alibaba.dubbo.config.annotation.Reference
    private HelloService hs;
    @RequestMapping("hello")
    @ResponseBody
    public String hello( String name){
        return hs.sayHello(name);
    }
}
```

2.3.3 消费方的接口

注意：
controller中要依赖HelloService，所以我们创建一个接口；
这里是消费方，不需要实现，因为实现会让服务方为我们搞定！

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

2.3.4 消费方的web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         id="WebApp_ID" version="3.1">

    <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:spring/spring.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

2.3.5 消费方的springmvc.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:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://code.alibabatech.com/schema/dubbo
       http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <!--Dubbo的应用名称，通常使用项目名 -->
    <dubbo:application name="dubbo-consumer" />
    <!--配置Dubbo的注册中心地址 -->
    <dubbo:registry address="zookeeper://192.168.204.141:2181" />
    <!--配置Dubbo扫描类，将这个类作为服务进行发布 -->
    <dubbo:annotation package="controller" />
</beans>
```

2.4 启动服务测试

首先启动服务方，再启动消费方。

访问：http://localhost:8002/hello?name=james

3. 监控中心

我们在开发时，需要知道注册中心都注册了哪些服务，以便我们开发和测试。

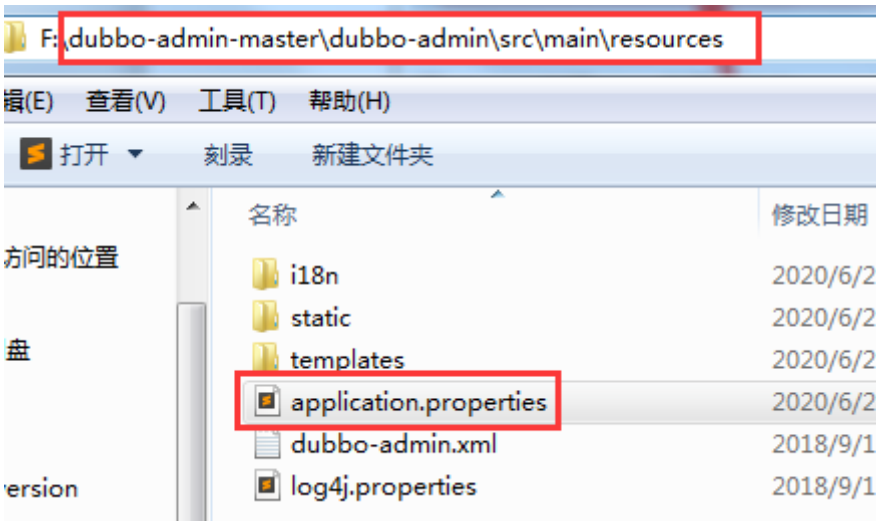
图形化显示注册中心的中 服务列表

我们可以通过部署一个web应用版的管理中心来实现。

3.1 服务管理端

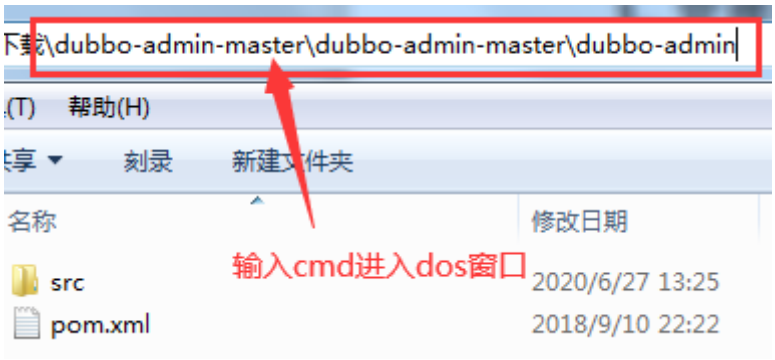
3.1.1 安装管理端

1. 解压 dubbo-admin-master.zip
2. 修改配置文件

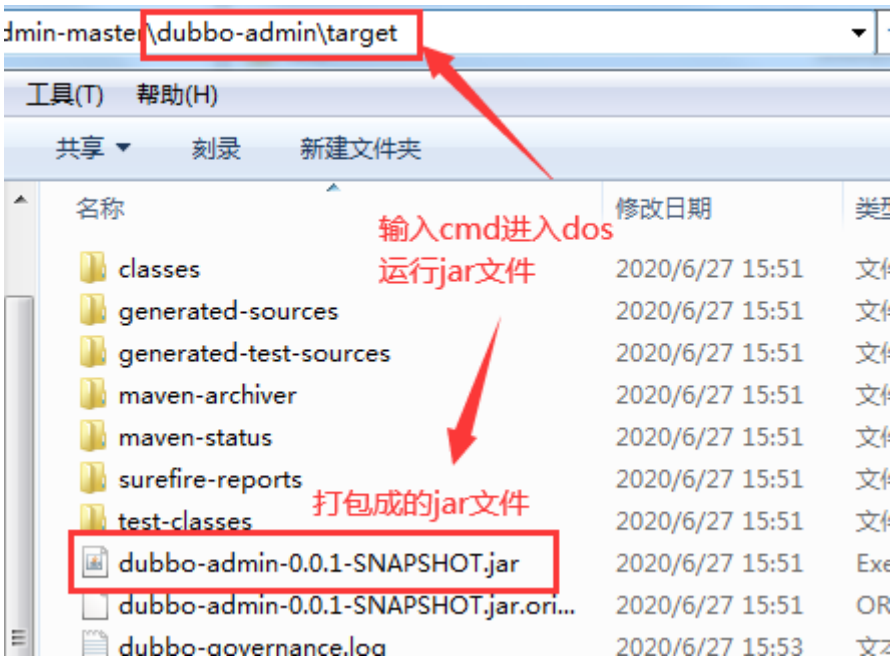


```
server.port=7001        管理端的端口
spring.velocity.cache=false
spring.velocity.charset=UTF-8
spring.velocity.layout-url=/templates/default.vm
spring.messages.fallback-to-system-locale=false
spring.messages.basename=i18n/message
spring.root.password=root
spring.guest.password=guest
                        注册中心的地址
dubbo.registry.address=zookeeper://192.168.31.78:2181
```

3. 返回到项目根目录，使用maven打包：**mvn clean package**



4. 在dos下运行target目录中的jar文件：**java -jar dubbo-admin-0.0.1-SNAPSHOT.jar**



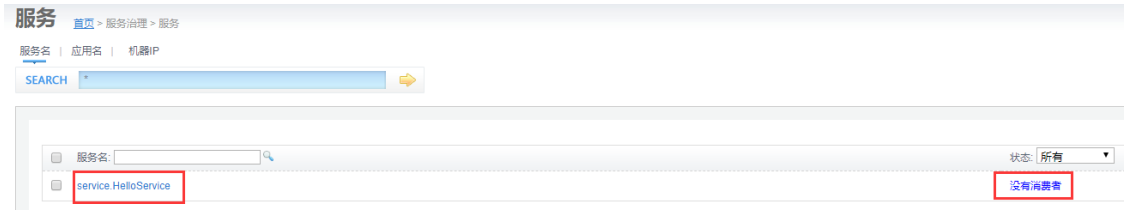
5. 此时打开浏览器输入：http://localhost:7001/ ；

   第一次访问时，需要登录，帐号密码都是root

3.2.1 管理端使用

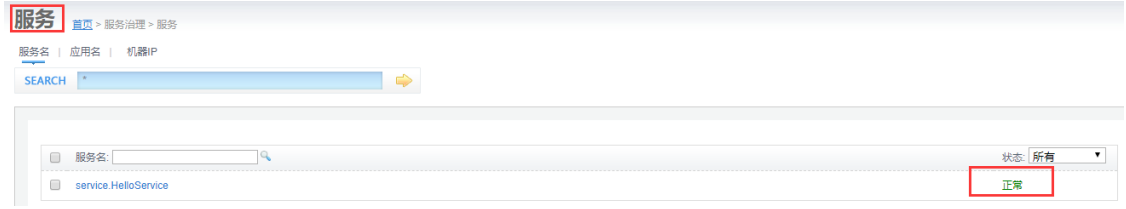
1. 启动服务方，将服务注册到zookeeper

2. 启动dubbo-server服务方后，刷新管理端，服务注册成功，只是没有消费者



3. 点击服务名，进入服务提供者页面

4. 把消费者也运行起来，刷新服务，显示正常



5. 查看消费者

3.2 监控统计中心

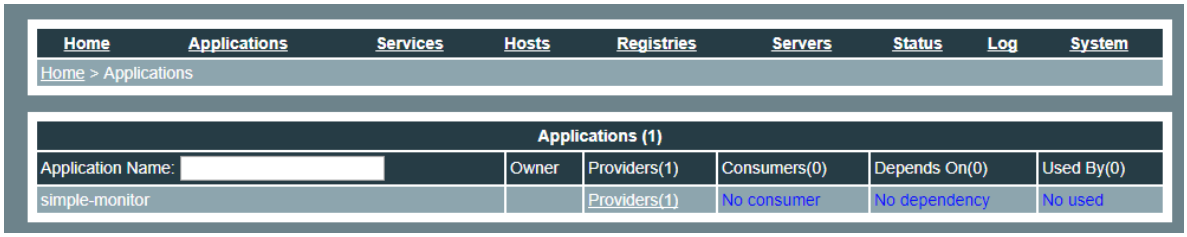
Monitor：**统计**中心 ，记录服务被调用多少次等

1. 解压dubbo-monitor-simple-2.5.3.zip
2. 修改dubbo-monitor-simple-2.5.3\conf\dubbo.properties

```
dubbo.container=log4j,spring,registry,jetty
dubbo.application.name=simple-monitor
dubbo.application.owner=
dubbo.registry.address=zookeeper://192.168.204.141:2181
#dubbo.registry.address=zookeeper://127.0.0.1:2181
#dubbo.registry.address=redis://127.0.0.1:6379
#dubbo.registry.address=dubbo://127.0.0.1:9090
dubbo.protocol.port=7070
dubbo.jetty.port=8080
dubbo.jetty.directory=${user.home}/monitor
dubbo.charts.directory=${dubbo.jetty.directory}/charts
dubbo.statistics.directory=${user.home}/monitor/statistics
dubbo.log4j.file=logs/dubbo-monitor-simple.log
dubbo.log4j.level=WARN
```

3. 双击运行dubbo-monitor-simple-2.5.3\bin\start.bat
4. 分别修改dubbo-server和dubbo-consumer的spring.xml，加入下面标签

```
<!-- 让监控 去注册中心 自动找服务 -->
<dubbo:monitor protocol="registry"/>
```



4. 综合实战

4.1 配置说明

4.1.1 启动时检查

- 启动时会在注册中心检查依赖的服务是否可用，不可用时会抛出异常
- 在消费方编写初始化容器的main方法启动（tomcat启动方式，必须访问一次action才能初始化spring）

```
public class Test {
    public static void main(String[] args) throws IOException {
        ClassPathXmlApplicationContext context =
                new
ClassPathXmlApplicationContext("classpath:spring/spring.xml");
        System.in.read();
    }
}
```

```
<!--默认是true:抛异常；false:不抛异常-->
<dubbo:consumer check="false" />
```

- 系统级别日志，需要配合log4j才输出，在resources下添加log4j.properties，内容如下：

```
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %m%n

log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=dubbo.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %l  %m%n

log4j.rootLogger=error, stdout,file
```

4.1.2 超时时间

- 由于网络或服务端不可靠，会导致调用过程中出现不确定的阻塞状态（超时）
- 为了避免超时导致客户端资源（线程）挂起耗尽，必须设置超时时间
- 在服务提供者添加如下配置：

```
<!--设置超时时间为2秒，默认为1秒-->
<dubbo:provider timeout="2000"/>
```

- 可以将服务实现HelloServiceImpl.java中加入模拟的网络延迟进行测试：

```
@Service
public class HelloServiceImpl  implements HelloService {
    public String sayHello(String name) {
        try {
            Thread.sleep(3000);
        }catch (Exception e){
            e.printStackTrace();
        }
        return "Hello,"+name+"!!!!!";
    }
}
```

- 超时设置2秒，而模拟的网络延迟有3秒，超出时限，报错！
- **配置原则**：
    - dubbo推荐在Provider上尽量多配置Consumer端属性：
        1. 作服务的提供者，比服务使用方更清楚服务性能参数，如调用的超时时间，合理的重试次数，等等
        2. 在Provider配置后，Consumer不配置则会使用Provider的配置值，即Provider配置可以作消费者的**缺省值**。

4.1.3 重试次数

- 当出现失败，自动切换并重试其它服务器，dubbo重试的缺省值是2次，我们可以自行设置
- 在provider提供方配置：

```
<!-- 消费方连接第1次不算，再来重试3次，总共重试4次 -->
<dubbo:provider timeout="2000" retries="3"/>
```

```
@Service
public class HelloServiceImpl  implements HelloService {
    public String sayHello(String name) {
        System.out.println("==============被调用 1 次===============");
        try {
            Thread.sleep(3000);
        }catch (Exception e){
            e.printStackTrace();
        }
        return "Hello,"+name+"!!!!!";
    }
}
```

- 并不是所有的方法都适合设置重试次数
    - **幂等**方法：适合（当参数一样，无论执行多少次，结果是一样的，例如：查询，修改）
    - **非幂等**方法：不适合（当参数一样，执行结果不一样，例如：删除，添加）
- 单独设置某个方法

1. 提供方接口添加sayNo()方法并实现

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

```
@Service
public class HelloServiceImpl  implements HelloService {
    public String sayHello(String name) {
        System.out.println("==============hello被调用一次
===============");
        try {
            Thread.sleep(3000);
        }catch (Exception e){
            e.printStackTrace();
        }
        return "Hello,"+name+"!!!!!";
    }

    public String sayNo() {
        System.out.println("-------no被调用一次-------");
        return "no!";
    }
}
```

2. 消费方接口添加sayNo()方法声明

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

3. 消费方controller

```
@Controller
 public class HelloAction {

    //@Reference  此注解已经在xml文件中被<dubbo:reference>顶替，所以自动注入
即可
    @Autowired
    private HelloService helloService;

    @GetMapping("hello")
    @ResponseBody
    public String sayHi(String name){
        return helloService.sayHello(name);
    }

    @GetMapping("no")
    @ResponseBody
    public String no(){
```

```
        return helloService.sayNo();
    }
}
```

4. 消费方配置方法重试次数

```
<dubbo:reference interface="service.HelloService" id="helloService">
    <dubbo:method name="sayHello" retries="3"/>
    <dubbo:method name="sayNo" retries="0"/> <!-- 不重试 -->
</dubbo:reference>
```

4.1.4 多版本

- 一个接口，多个（版本的）实现类，可以使用定义版本的方式引入
- 为HelloService接口定义两个实现类，提供者修改配置：

```
<dubbo:service interface="service.HelloService"
class="service.impl.HelloServiceImpl01" version="1.0.0"/>
<dubbo:service interface="service.HelloService"
class="service.impl.HelloServiceImpl02" version="2.0.0"/>
```

- 消费者就可以根据version的版本，选择具体的服务版本

```
<dubbo:reference interface="service.HelloService" id="helloService"
version="2.0.0">
    <dubbo:method name="sayHello" retries="3"/>
    <dubbo:method name="sayNo" retries="0"/>
</dubbo:reference>
```

- 注意：消费者的控制层要改为自动注入，因为@Reference注解和 <dubbo:reference>在这里冲突

```
@Controller
public class HelloAction {
    @Autowired
    private HelloService helloService;
}
```

- 当消费者的版本修改为 **version="*"**，那么就会随机调用服务提供者的版本

```
-------1.0被调用一次-------
-------2.0被调用一次-------
-------1.0被调用一次-------
-------1.0被调用一次-------
-------1.0被调用一次-------
-------2.0被调用一次-------
```

4.1.5 本地存根

- 目前我们的分布式架构搭建起来有一个严重的问题，就是所有的操作全都是 消费者发起，由服务提供者执行
- 消费者动动嘴皮子却什么活都不干，这样会让提供者很累，例如简单的参数验证，消费者完全能够胜任，把合法的参数再发送给提供者执行，效率高了，提供者也没那么累了
- 例如：去房产局办理房屋过户，请带好自己的证件和资料，如果什么都不带，那么办理过户手续会很麻烦，得先调查你有什么贷款，有没有抵押，不动产证是不是你本人，复印资料等操作。一天肯定办不完。明天还要来。如果你能提前将这些东西准备好，办理过户，1个小时足矣，这就是“房产中介办事效率高的原因”
- 话不多说，先在消费者处理一些业务逻辑，再调用提供者的过程，就是“本地存根”
- 代码实现肯定在 消费者，创建一个HelloServiceStub类并且实现HelloService接口
- 注意：**必须使用构造方法的方式注入**

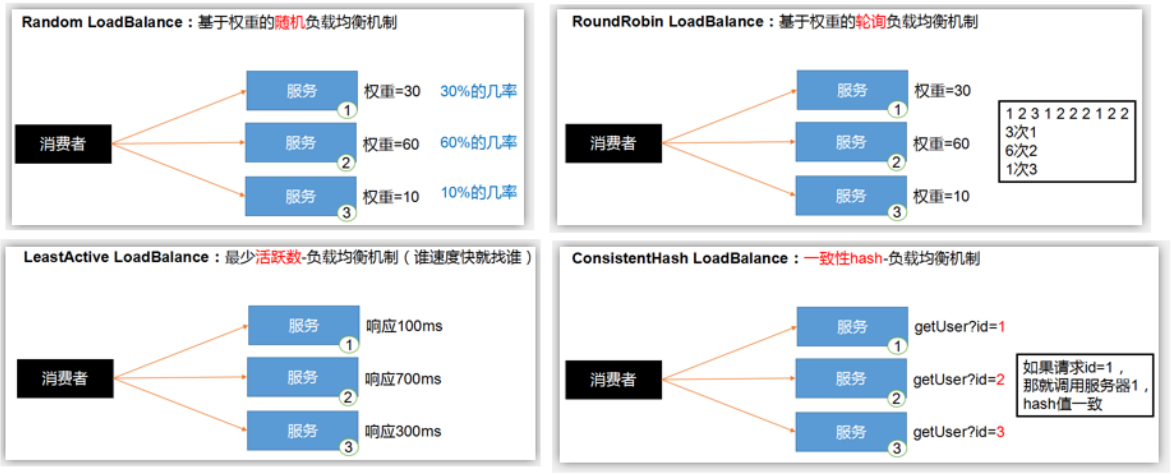
```
public class HelloServiceStub implements HelloService {

    private HelloService helloService;
    // 注入HelloService
    public HelloServiceStub(HelloService helloService) {
        this.helloService = helloService;
    }

    public String sayHello(String name) {
        System.out.println("本地存根数据验证。。。");
        if(!StringUtils.isEmpty(name)){
            return helloService.sayHello(name);
        }
        return "i am sorry!";
    }

    public String sayNo() {
        return helloService.sayNo();
    }
}
```

- 修改消费者配置：

```
<dubbo:reference interface="service.HelloService" id="helloService"
version="1.0.0" stub="service.impl.HelloServiceStub">
    <dubbo:method name="sayHello" retries="3"/>
    <dubbo:method name="sayNo" retries="0"/>
</dubbo:reference>
```

4.2 负载均衡策略

- 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。
- 简单的说，好多台服务器，不能总是让一台服务器干活，应该“雨露均沾”
- dubbo一共提供4种策略，缺省为 random 随机分配调用



- 修改提供者配置并启动3个提供者，让消费者对其进行访问
  - tomcat端口8001，8002，8003
  - provider端口20881，20882，20883

```
<dubbo:provider timeout="2000" retries="3" port="20881"/>
```

  - HelloServiceImpl01类，服务器1，服务器2，服务器3

```
public String sayNo() {
    System.out.println("----服务器1---1.0被调用一次-------");
    return "no!";
}
```

  - 启动consumer进行测试
- 消费方修改权重

```
<dubbo:reference loadbalance="roundrobin" interface="service.HelloService"
id="helloService" version="2.0.0" stub="stub.HelloServiceStub">
    <dubbo:method name="sayHello" retries="3"/>
    <dubbo:method name="sayNo" retries="0"/>
</dubbo:reference>
```

- 最好使用管理端修改权重

4.3 高可用

4.3.1 zookeeper宕机

- zookeeper注册中心宕机，还可以消费dubbo暴露的服务

    - 监控中心宕掉不影响使用，只是丢失部分采样数据
    - 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
    - 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
    - **注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯**
    - 服务提供者无状态，任意一台宕掉后，不影响使用
    - 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复
- 测试：
    - 正常发出请求
    - 关闭zookeeper：./zkServer.sh stop
    - 消费者仍然可以正常消费

4.4 服务降级

- 壁虎遇到危险会自动脱落尾巴，目的是损失不重要的东西，保住重要的
- 服务降级，就是根据实际的情况和流量，对一些服务有策略的停止或换种简单的方式处理，从而释放服务器的资源来保证核心业务的正常运行

4.4.1 为什么要服务降级

- 而为什么要使用服务降级，这是防止分布式服务发生**雪崩效应**
- 什么是雪崩？就是蝴蝶效应，当一个请求发生超时，一直等待着服务响应，那么在高并发情况下，很多请求都是因为这样一直等着响应，直到服务资源耗尽产生宕机，而宕机之后会导致分布式其他服务调用该宕机的服务也会出现资源耗尽宕机，这样下去将导致整个分布式服务都瘫痪，这就是雪崩。

4.4.2 服务降级实现方式

- 在 **管理控制台**配置服务降级：**屏蔽**和**容错**
- **屏蔽**：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。
- **容错**：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。

4.5 整合MyBatis实现用户注册

4.5.1 初始化数据库

```
CREATE DATABASE smd
USE smd
CREATE TABLE users(
  uid INT(11) AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  PASSWORD VARCHAR(50) NOT NULL,
  phone VARCHAR(50) NOT NULL,
  createtime VARCHAR(50) NOT NULL
)
```

4.5.2 创建聚合项目-项目模块化

- lagou-dubbo（项目目录）
- lagou-dubbo-parent（父工程，聚合项目：定义所有模块用的依赖版本）

```
<modelVersion>4.0.0</modelVersion>

<groupId>com.sunguoan</groupId>
<artifactId>sun-parent</artifactId>
<version>1.0-SNAPSHOT</version>

<packaging>pom</packaging>

<properties>
    <spring.version>5.0.6.RELEASE</spring.version>
</properties>

<dependencies>
    <!-- JSP相关 -->
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <scope>provided</scope>
        <version>2.5</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jsp-api</artifactId>
        <scope>provided</scope>
        <version>2.0</version>
    </dependency>
    <!-- Spring -->
    <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>
    <!-- Mybatis -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.2.8</version>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.2.2</version>
    </dependency>
    <!-- 连接池 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.0.9</version>
    </dependency>
    <!-- 数据库 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>
    <!--dubbo -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.5.7</version>
    </dependency>
    <dependency>
        <groupId>org.apache.zookeeper</groupId>
        <artifactId>zookeeper</artifactId>
        <version>3.4.6</version>
    </dependency>
    <dependency>
        <groupId>com.github.sgroschupf</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.1</version>
    </dependency>
    <dependency>
```

```
        <groupId>javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.11.0.GA</version>
    </dependency>
    <!-- fastjson -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.47</version>
    </dependency>
    <!-- junit -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>${spring.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- lagou-dubbo-entity（实体工程，jar项目）
- lagou-dubbo-dao（数据访问层工程，jar项目）
- lagou-dubbo-interface（服务接口定义工程，jar项目）
- lagou-dubbo-service（privoder服务提供者工程，jar项目）

```
<parent>
    <groupId>com.sunguoan</groupId>
    <artifactId>sun-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
</parent>
<modelVersion>4.0.0</modelVersion>

<artifactId>sun-service</artifactId>
<packaging>war</packaging>
<dependencies>
    <dependency>
        <groupId>com.sunguoan</groupId>
        <artifactId>sun-interface</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.sunguoan</groupId>
        <artifactId>sun-dao</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
```

```
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <configuration>
                <port>8001</port>
                <path>/</path>
            </configuration>
            <executions>
                    <execution>
                        <!-- 打包完成后,运行服务 -->
                        <phase>package</phase>
                        <goals>
                            <goal>run</goal>
                        </goals>
                    </execution>
                </executions>
        </plugin>
    </plugins>
</build>
```

- lagou-dubbo-web（consumer服务消费者工程，war项目）

```
<!-- 解决post乱码 -->
<filter>
    <filter-name>charset</filter-name>
 <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>charset</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<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:spring/spring-mvc.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

4.5.3 启动测试

1. 先选择父项目（聚合工程）进行全员安装成jar
2. 启动服务service
3. 启动调用者web