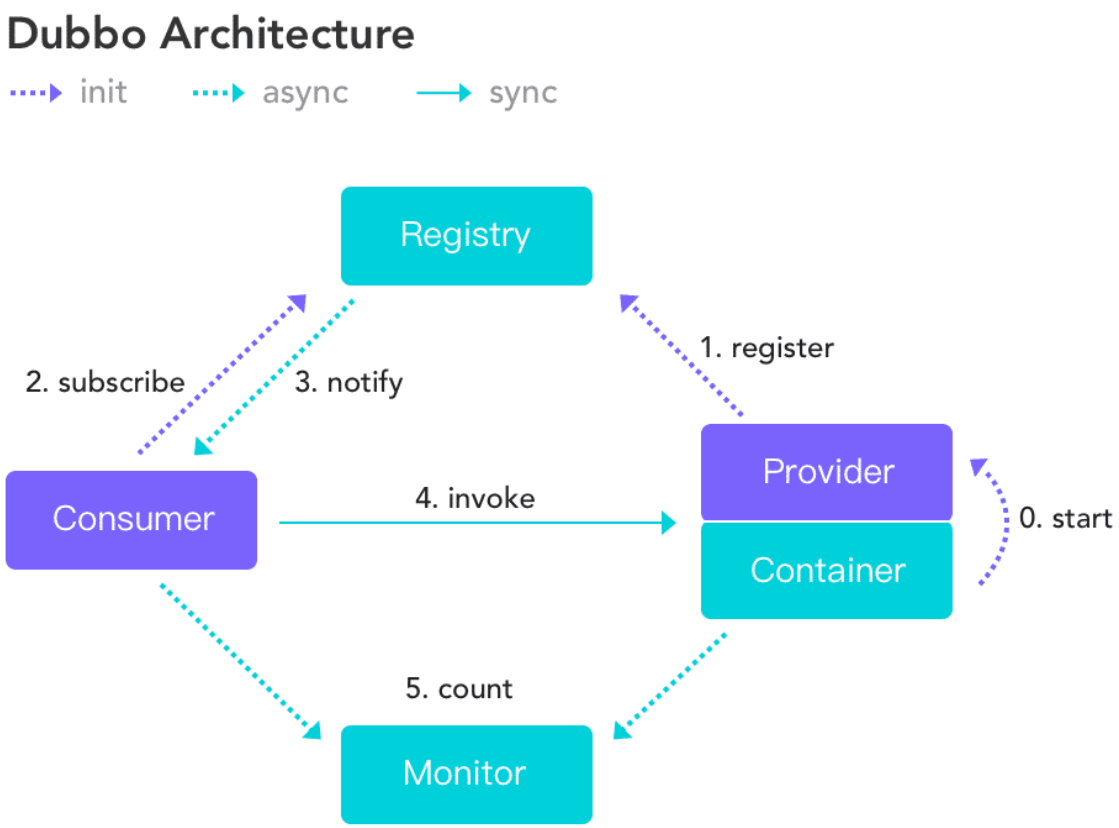
# Dubbo

## 主讲：崔译

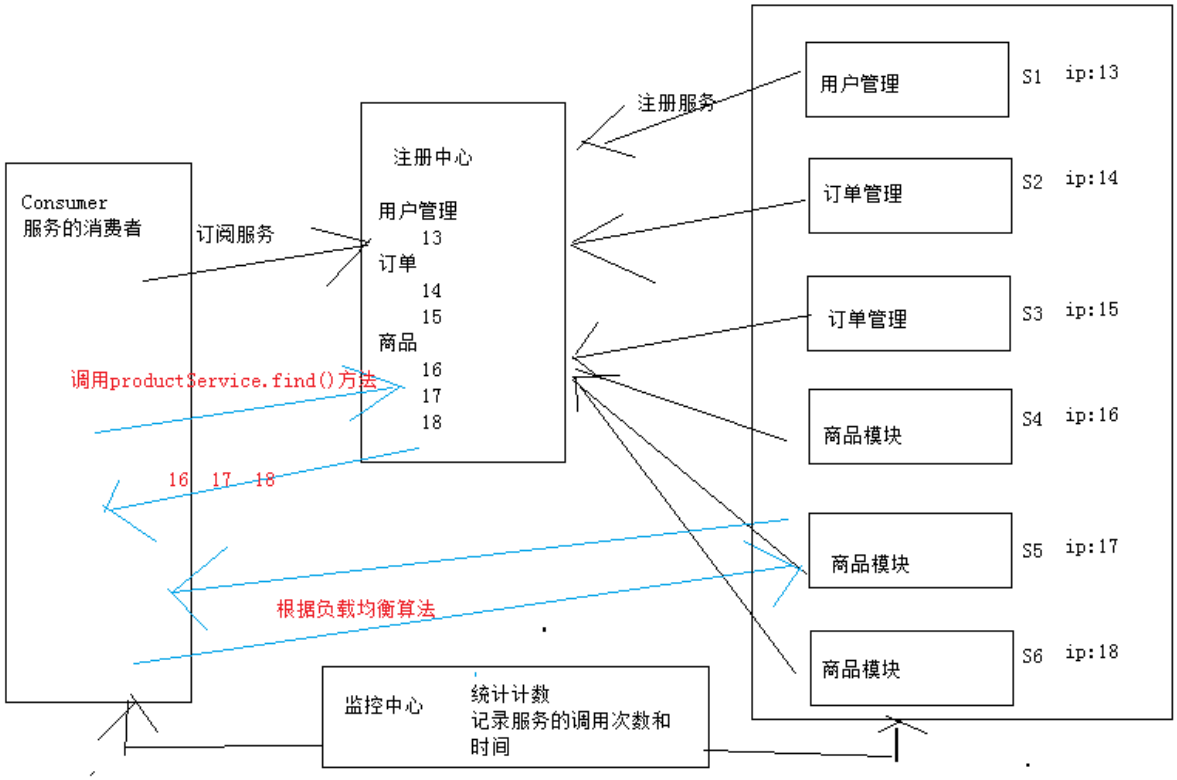
## 一、简介

> Apache Dubbo™ (incubating)是一款高性能Java RPC框架。
>
> Apache Dubbo (incubating) | ˈdʌbəʊ| 是一款高性能、轻量级的开源Java RPC框架，它提供了三大核心能力：
>
> 1. 面向接口的远程方法调用，
> 2. 智能容错和负载均衡，
> 3. 服务自动注册和发现。



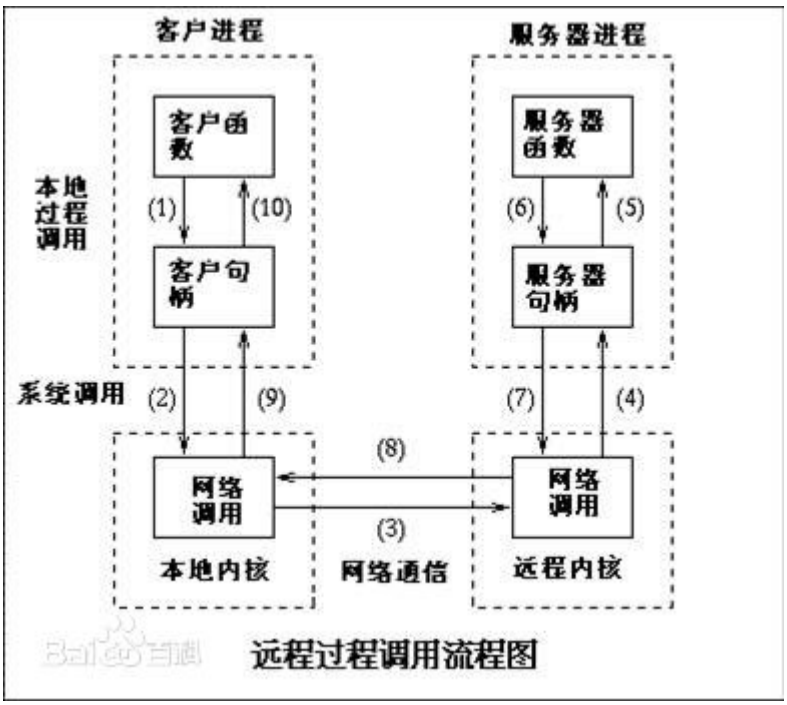

1. Provider  服务的提供者，负责对外提供方法，提供者在启动的时候，会向注册中心注册自己能够提供的服务
2. Consumer 服务的消费者，消费者在启动时，会 向注册中心 `订阅` 自己需要的服务，服务的消费者 从 订阅中心的 地址列表中，基于负载均衡算法，选择一台服务器进行调用，如果调用失败，会选择另一台调用
3. Registry 注册中心，接受注册和订阅，向消费者 提供注册的 服务器地址列表

4. Monitor 监控中心，服务的消费者和提供者 ， 在内存中累计的调用次数和调用时间，定时的（每分钟）向 注册中心发送一次统计数据

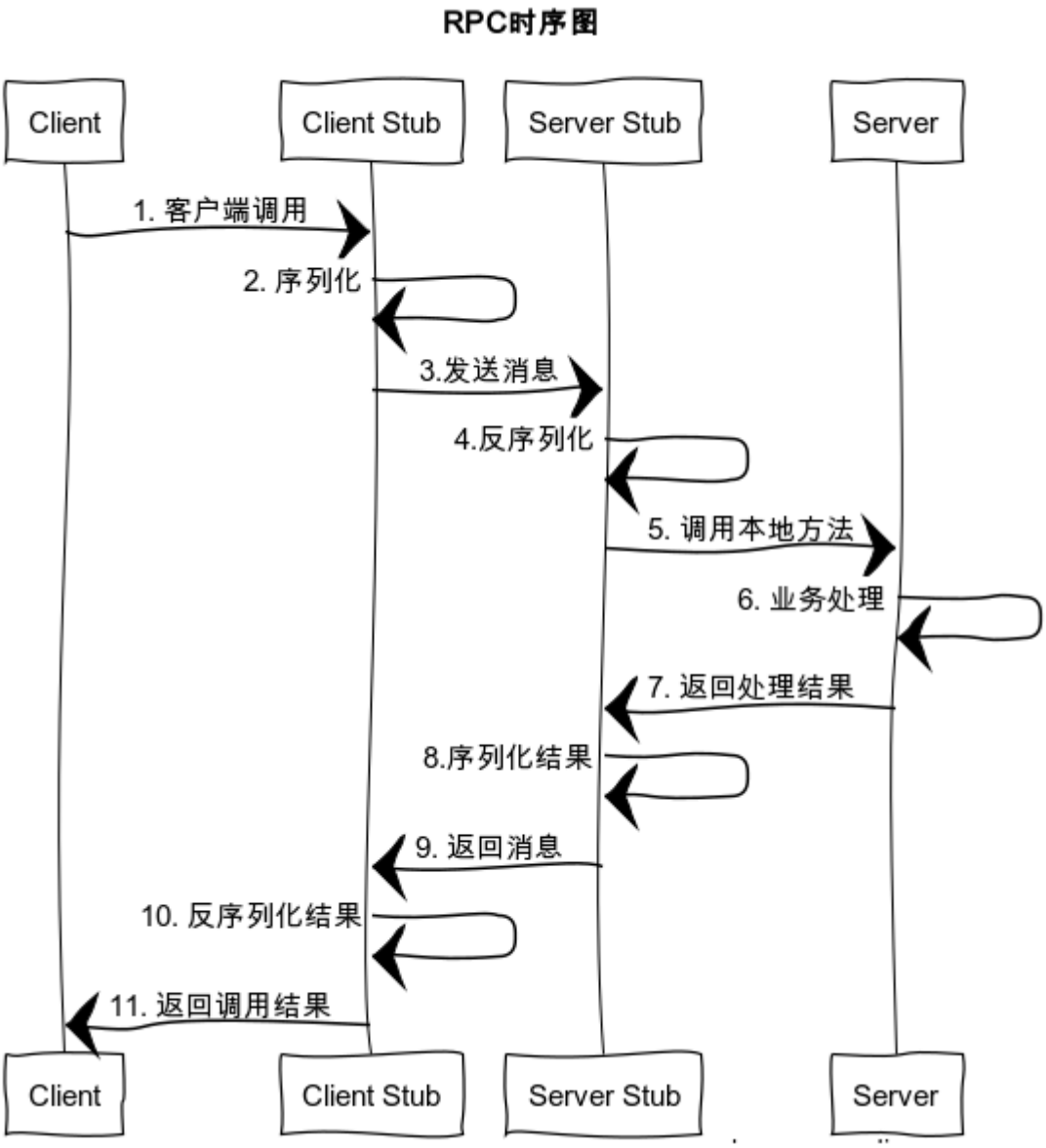



# 二、RPC简介

> Remote Procedure Call 远程过程调用
>
> - 是一种进程间的通信方式
> - 它允许应用程序调用网络上另一个应用程序中的方法（前提：另一个应用程序要允许被调用/要主动暴露方法）
> - 对于 服务消费者（远程方法的调用者）而言，远程调用过程对于 具体代码透明


客户进程
服务器进程
客户函数
服务器函数
本地过程调用
(1)
(10)
(6)
(5)
客户句柄
服务器句柄
系统调用
(2)
(9)
(7)
(4)
(8)
网络调用
网络调用
(3)
本地内核
网络通信
远程内核
远程过程调用流程图

时序图




# 三、分布式系统

> 分布式系统是若干个独立计算机的集合，这些计算机对于终端用户而言，就像单个相关系统

**与集群的区别**

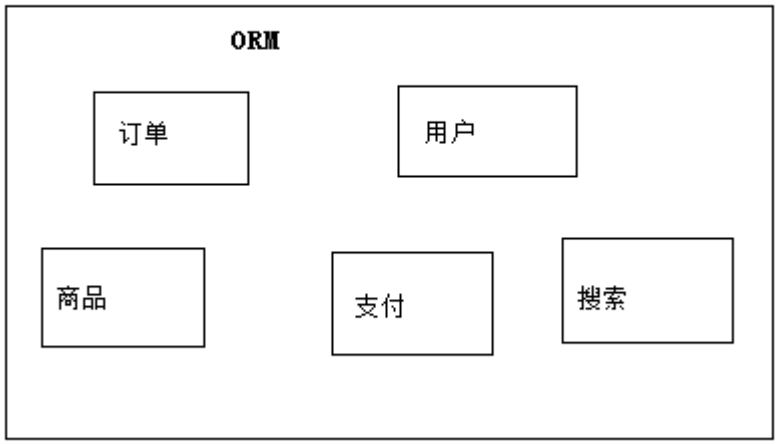
- 集群 是 一种物理上的概念，多态设备进行统一的管理，就叫做集群（至于这多态设备是不是做同样的事情，或者同一件事情，或者同一件事情的不同步骤，不确定）
- 分布式的 所谓的 `若干个独立计算机的集合` 指的是逻辑上的 `集合`

# 四、web程序架构的演变

## 1、单一应用架构

> 当网站访问量（流量）很小时，只需要一个应用程序，将所有功能都部署在一起，以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。

All in One 单一应用架构




**适用于：小型网站： 管理系统，简易办公系统**

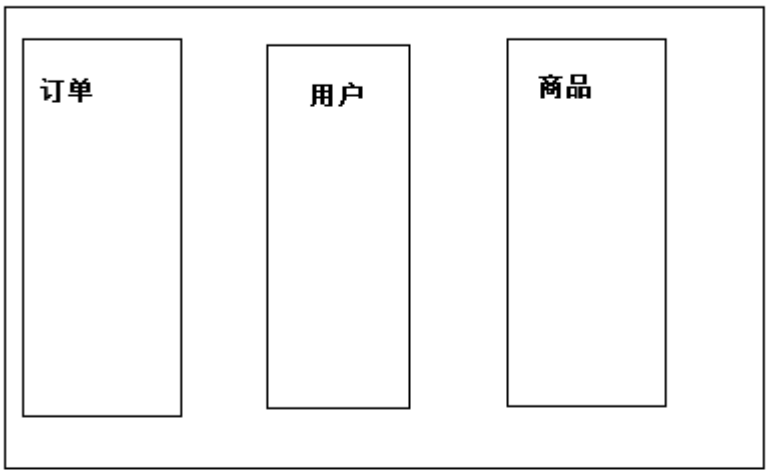
**局限**

1. 扩展性差
2. 不便于协同开发
3. 不利于升级和维护

## 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，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

**垂直应用架构**




通过切分项目业务，实现各个项目模块的独立，降低了 维护和 升级的难度，便于协同开发

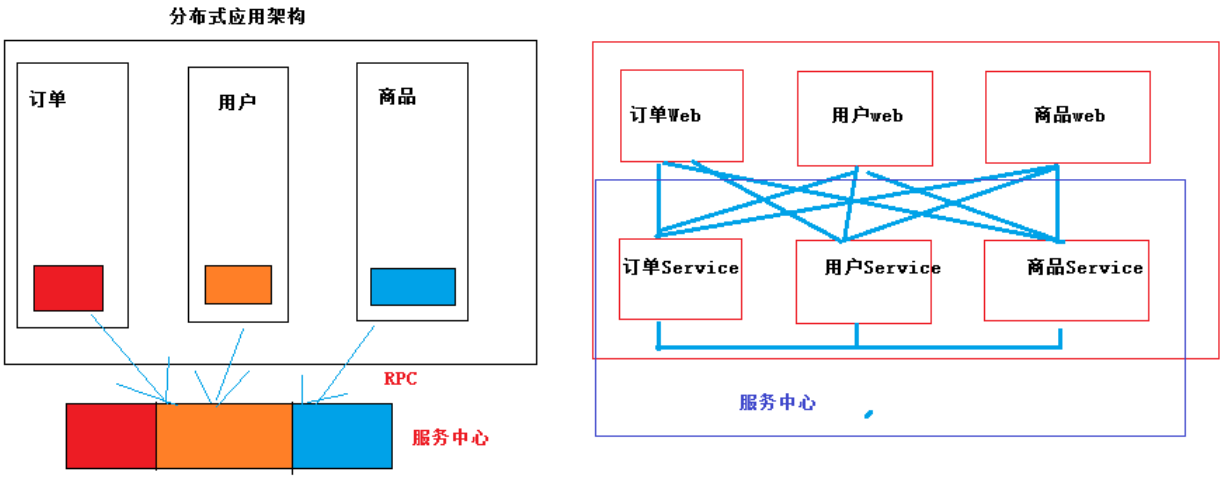
提高了 程序 的 扩展性

**局限**

- 公共资源无法复用，开发成本上的浪费

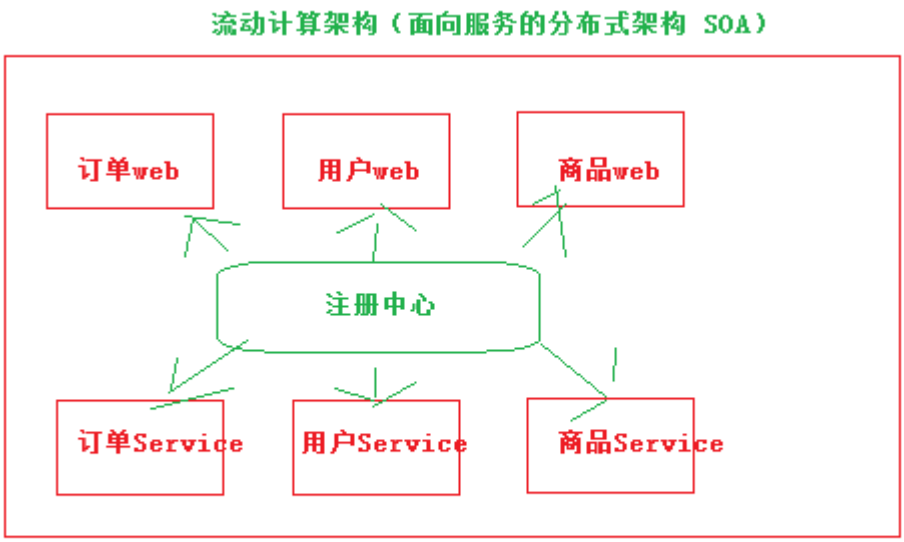
## 3、分布式服务架构

> 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，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

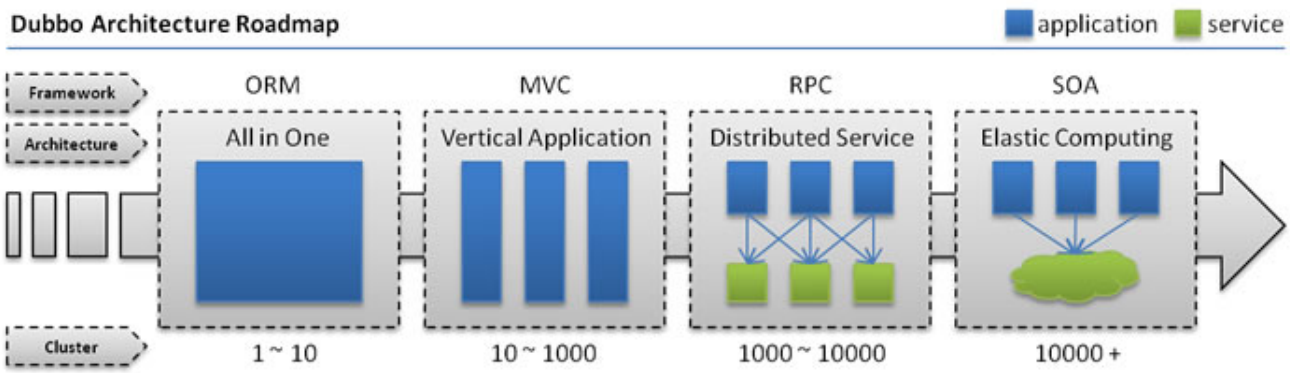



## 4、流动计算架构

> 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显现，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)( Service Oriented Architectre )是关键。




## 5、总结




# 五、使用Dubbo

## 1、准备工作

- 安装JDK
- 安装ZooKeeper
- 下载dubbo-admin
  - 下载

    官方提供的，一个可视化的dubbo的监控管理程序，**非必需品**

    git下载地址
  - 使用Maven 打包 backend 项目

    ```
    mvn clean compile package
    ```
  - 使用npm打包front项目
  - 启动服务

    ```
    java -jar xxxx.jar
    ```
  - 使用浏览器测试

    ```
    localhost:8080
    ```

## 2、Spring-dubbo

### 1、创建spring-common

编写接口和通用类

```
public interface SomeService {
    public User fun(User user);
}
// 必须要实现Serializable接口
public class User  implements Serializable{
    private String username;
    private String address;
}
```

## 2、创建spring-provider

1. 导入common模块

```
<dependency>
  <groupId>com.itany</groupId>
  <artifactId>spring-common</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

2. 编写api的实现类

```
public class SomeServiceImpl implements SomeService {
    @Override
    public User fun(User user) {
        System.out.println(user.getAddress()+","+user.getUsername());

        User temp = new User();
        temp.setAddress("new" + user.getAddress());
        temp.setUsername("new" + user.getUsername());
        return temp;
    }
}
```

## 3、创建spring-consumer

1. 导入common模块

```
<dependency>
  <groupId>com.itany</groupId>
  <artifactId>spring-common</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

2. 编写类使用SomeService

```
public class OtherService {
    private SomeService someService;
    public void test()
    {
        User user = new User("lx","yd");
        User retObj = someService.fun(user);
        System.out.println(retObj);
    }

}
```

3. 编写测试类

```
public class Test {
    public static void main(String[] args) {
        OtherService os = new OtherService();
        os.test();
    }
}
```

## 4、使用Dubbo改造

1. 导入dubbo依赖

```
 <!--dubbo核心包-->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>dubbo</artifactId>
  <version>2.6.2</version>
</dependency>

<!--
      我们使用ZooKeeper作为注册中心，所以在dubbo内容会操作ZooKeeper
      此处需要添加 ZooKeeper的依赖
-->
<dependency>
  <groupId>com.101tec</groupId>
  <artifactId>zkclient</artifactId>
  <version>0.10</version>
</dependency>
<dependency>
  <groupId>org.apache.curator</groupId>
  <artifactId>curator-framework</artifactId>
  <version>2.12.0</version>
</dependency>
```

2. 配置provider

```
<!--当前应用程序的名字-->
<dubbo:application name="spring-provider"></dubbo:application>
<!--指定注册中心地址-->
<dubbo:registry address="zookeeper://127.0.0.1:2181"></dubbo:registry>

<!--配置使用dubbo协议，将服务暴露在特定端口-->
<dubbo:protocol name="dubbo" port="28888"></dubbo:protocol>

<!--指定要暴露的服务-->
<dubbo:service interface="com.itany.api.SomeService" ref="someService">
</dubbo:service>

<!--配置service的bean-->
<bean id="someService" class="com.itany.api.impl.SomeServiceImpl"></bean>
```

3. 配置consumer

```
<!--当前应用程序的名字-->
<dubbo:application name="spring-consumer"></dubbo:application>
<!--指定注册中心地址-->
<dubbo:registry address="zookeeper://127.0.0.1:2181"></dubbo:registry>
<!--配置服务引用-->
<dubbo:reference
                 id="service"
                 interface="com.itany.api.SomeService">

</dubbo:reference>
<bean id="otherService" class="com.itany.service.OtherService">
  <property name="someService" ref="service"></property>
</bean>
```

4. 启动ZooKeeper
5. 编写provider的测试类

```
public class Test {

    public static void main(String[] args) {
        ApplicationContext ac = new ClassPathXmlApplicationContext("app*.xml");

        //阻塞线程
        new Scanner(System.in).next();
    }
}
```

6. 编写consumer的测试类

```
public class Test {
    public static void main(String[] args) {

        ApplicationContext ac = new ClassPathXmlApplicationContext("app*.xml");
        OtherService os = (OtherService) ac.getBean("otherService");
        os.test();

    }
}
```

## 3、Spring-dubbo-注解

### 1、改造provider

```
 <!--当前应用程序的名字-->
<dubbo:application name="spring-provider"></dubbo:application>
<!--指定注册中心地址-->
<dubbo:registry address="zookeeper://127.0.0.1:2181"></dubbo:registry>
<!--配置使用dubbo协议，将服务暴露在特定端口-->
<dubbo:protocol name="dubbo" port="28888"></dubbo:protocol>

<!--
    <dubbo:service interface="com.itany.api.SomeService" ref="someService">
    </dubbo:service>
-->
<!--
    <bean id="someService" class="com.itany.api.impl.SomeServiceImpl">
    </bean>
-->

<!--指定dubbo扫描的包-->
<dubbo:annotation package="com.itany.api.impl"></dubbo:annotation>
```

```
//方式1 显示使用com.alibaba.dubbo.config.annotation.Service注解
//@Service
//@com.alibaba.dubbo.config.annotation.Service
//方式2 使用Component注解替换Spring的Service注解
//@Component
//@Service
//方式3 使用com.alibaba.dubbo.config.annotation.Service注解上的
//    @Inherited 特性：
//      该注解标记的注解A会 自动的加在 使用A的类的子类上
//      使用注解@Inherited可以让指定的注解在某个类上使用后，这个类的子类将自动被该注解标记。
public class SomeServiceImpl extends BaseDoubleService implements SomeService  {
    @Override
    public User fun(User user) {
        System.out.println(user.getAddress()+","+user.getUsername());

        User temp = new User();

        temp.setAddress("new" + user.getAddress());
```

```
        temp.setUsername("new" + user.getUsername());
        return temp;
    }
}

// 如果使用方式3  基础类必须实现相同接口，同时提供interfaceClass 属性
@Service(interfaceClass = SomeService.class)
public class BaseDoubleService implements  SomeService{
}
```

## 2、改造consumer

```
<!--当前应用程序的名字-->
<dubbo:application name="spring-consumer"></dubbo:application>
<!--指定注册中心地址-->
<dubbo:registry address="zookeeper://127.0.0.1:2181"></dubbo:registry>


<!--配置服务引用-->
<!--<dubbo:reference-->
<!--id="service"-->
<!--interface="com.itany.api.SomeService">-->

<!--</dubbo:reference>-->

<!--<bean id="otherService" class="com.itany.service.OtherService">-->
<!--    <property name="someService" ref="service"></property>-->
<!--</bean>-->

<!--扫包，扫描dubbo注解-->
<dubbo:annotation package="com.itany.service"></dubbo:annotation>

<bean id="otherService" class="com.itany.service.OtherService">
    <!--不再需要注入属性，属性由dubbo注入-->
</bean>
```

```
public class OtherService {

    // 使用dubbo 的 @Reference注解 注入 远程服务对象
    @Reference
    private SomeService someService;

    public void test()
    {
        User user = new User("lx","yd");
        User retObj = someService.fun(user);
        System.out.println(retObj);
    }

}
```

# 4、SpringBoot-dubbo

## 1、创建springboot-provider

1. 导入common模块

```
<dependency>
  <groupId>com.itany</groupId>
  <artifactId>spring-common</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

2. 导入dubbo的starter

   github文档

```
<dependency>
  <groupId>com.alibaba.boot</groupId>
  <artifactId>dubbo-spring-boot-starter</artifactId>
  <version>0.2.0</version>
</dependency>
```

| versions | Java | Spring Boot | Dubbo |
| --- | --- | --- | --- |
| 0.2.0 | 1.8+ | 2.0.x | 2.6.2 + |
| 0.1.1 | 1.7+ | 1.5.x | 2.6.2 + |

如果SpringBoot的版本 是2.0+ 需要0.2.0的starter

3. 编写实现类

```
@Service
public class SomeServiceImpl implements SomeService {

    @Override
    public User fun(User user) {
        User temp = new User("boot-"+user.getUsername(),"boot-"+user.getAddress());
        return temp;
    }
}
```

4. 配置yml

```
#<!--当前应用程序的名字  不要和注册到注册中心zookeeper的其他的application重名-->
#<dubbo:application name="spring-provider"></dubbo:application>
dubbo.application.name=springboot-provider
#<!--指定注册中心地址-->
#<dubbo:registry address="zookeeper://127.0.0.1:2181"></dubbo:registry>
dubbo.registry.address=127.0.0.1:2181
dubbo.registry.protocol=zookeeper
```

```
#配置使用dubbo协议，将服务暴露在特定端口
#<dubbo:protocol name="dubbo" port="28888"></dubbo:protocol>
dubbo.protocol.name=dubbo
dubbo.protocol.port=28888
#<!--指定dubbo扫描的包-->
#<dubbo:annotation package="com.itany.api.impl"></dubbo:annotation>
dubbo.scan.base-packages=com.itany.springbootprovider.service.impl
```

## 2、创建springboot-consumer

1. 导入common模块

```
<dependency>
  <groupId>com.itany</groupId>
  <artifactId>spring-common</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

2. 导入dubbo的starter

```
<dependency>
  <groupId>com.alibaba.boot</groupId>
  <artifactId>dubbo-spring-boot-starter</artifactId>
  <version>0.2.0</version>
</dependency>
```

3. 编写控制器类

```
@Controller
@RequestMapping("/sc")
public class SomeController {

    @Reference
    private SomeService someService;

    @RequestMapping("/f1")
    public String f1(User user, Model model)
    {
        User result =  someService.fun(user);
        model.addAttribute("data",result);
        return "user";
    }

}
```

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

    <h1 th:text="${data.username}"></h1>
    <h1 th:text="${data.address}"></h1>

</body>
</html>
```

4. 配置yml

```
#<!--当前应用程序的名字-->
#<dubbo:application name="spring-consumer"></dubbo:application>
dubbo.application.name=springboot-consumer
#<!--指定注册中心地址-->
#<dubbo:registry address="zookeeper://127.0.0.1:2181"></dubbo:registry>
dubbo.registry.protocol=zookeeper
dubbo.registry.address=127.0.0.1:2181

#<!--扫包，扫描dubbo注解-->
#<dubbo:annotation package="com.itany.service"></dubbo:annotation>
dubbo.scan.base-packages=com.itany.springbootconsumer.controller
```

### 3、测试类

```
@SpringBootApplication
@EnableDubbo
public class SpringbootConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootConsumerApplication.class, args);
    }
}
@SpringBootApplication
@EnableDubbo
public class SpringbootProviderApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootConsumerApplication.class, args);
    }
}
```

# 六、安装dubbo-admin

**是dubbo的管理控制台，用于监控dubbo服务，不是必需品**

## 1、下载dubbo-admin

github下载地址

注意：下载的 主版本(master)

## 2、解压缩

## 3、打包

进入到dubbo-admin 根目录（pom文件所在目录），使用maven 打包

```
mvn clean compile packge
```

## 4、启动dubbo-admin

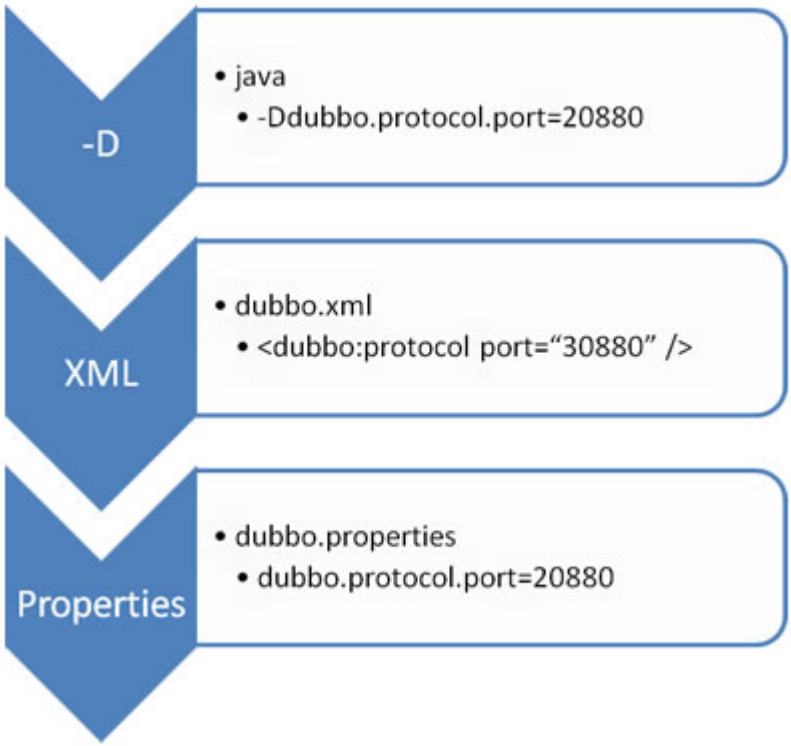
```
cd target
java -jar dubbo-admin-0.0.1-SNAPSHOT.jar
```

## 5、访问

使用`root`/`root`访问`localhost:7001`

# 七、常用配置

## 1、配置原则



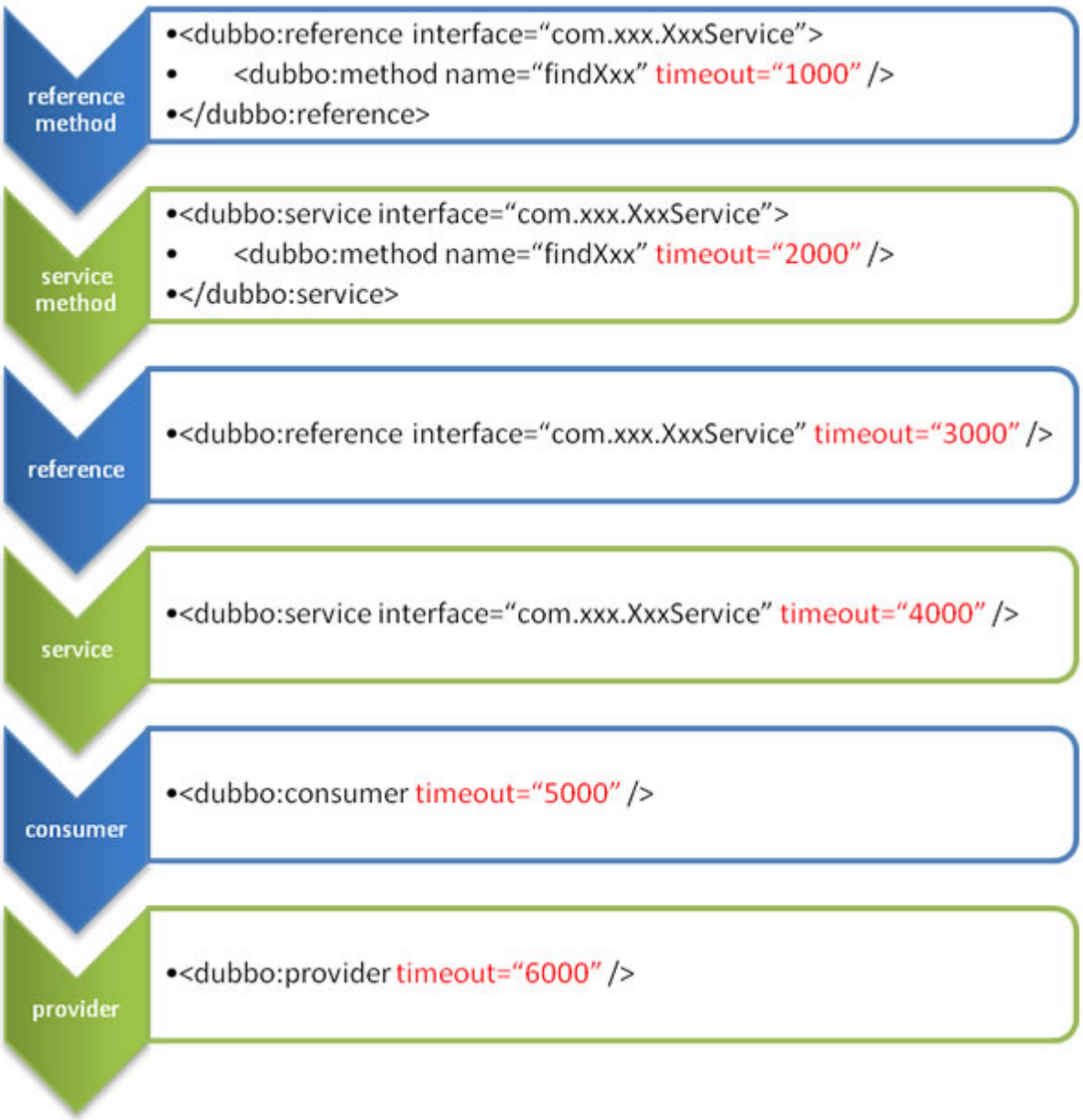

- 关于Dubbo配置方式有三种：
  - 使用虚拟机参数
  - 配置`xml`文件
  - 配置`dubbo.properties`文件
- 优先级最高的是 使用虚拟机参数 `java -D` 指定配置信息

  提供了启动时的参数的重写技术
- 优先级处于中间位置的是 使用xml 配置文件

  常规配置方式
- 优先级最低的是 `dubbo.properties`文件

  一般用于公共的共享配置，默认配置

## 2、配置覆盖关系




- 所有的常用配置，都可配置在六个位置
  - 消费者的 调用的远程方法上
  - 提供者的 提供的远程方法上
  - 消费者的 调用的远程方法所在接口上
  - 提供者的 提供的远程方法所在接口上
  - 消费者的 全局
  - 提供者的 全局
- 优先级从上至下，依次变小
  - 方法级优先，接口级次之，全局配置再次之
  - 如果级别一样，则消费方优先，提供方次之。

## 3、启动时检查

> 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** 处于重试状态（log4）

```
<!--关闭  某个服务  的启动时检查,
    如果不关闭，没有提供者时报错
    关闭了，但是调用远程服务，没有提供者时报错
-->
<dubbo:reference interface="com.foo.BarService" check="false" />
<!--
    dubbo:consumer 用于consumer的全局配置，
        该标签为 <dubbo:reference> 标签的缺省值设置。
    关闭所有服务的启动时检查
-->
<dubbo:consumer check="false" />
<!--关闭注册中心启动时检查-->
<dubbo:registry check="false" />
```

## 4、超时时间

```
<!--全局-->
<dubbo:consumer timeout="2"></dubbo:consumer>
<dubbo:provider timeout="2"></dubbo:provider>
<!--特定接口-->
<dubbo:reference timeout="2"></dubbo:reference>
<dubbo:service timeout="2"></dubbo:service>
 <!--特定方法-->
<dubbo:reference>
    <dubbo:method  timeout="2"></dubbo:method>
</dubbo:reference>
<dubbo:service>
    <dubbo:method  timeout="2"></dubbo:method>
</dubbo:service>
```

## 5、重试次数

> 远程服务调用重试次数，不包括第一次调用，不需要重试请设为0

```
<!--全局-->
<dubbo:consumer retries="2"></dubbo:consumer>
<dubbo:provider retries="2"></dubbo:provider>
<!--特定接口-->
<dubbo:reference retries="2"></dubbo:reference>
<dubbo:service retries="2"></dubbo:service>
 <!--特定方法-->
<dubbo:reference>
    <dubbo:method  retries="2"></dubbo:method>
</dubbo:reference>
<dubbo:service>
    <dubbo:method  retries="2"></dubbo:method>
</dubbo:service>
```

## 6、负载均衡

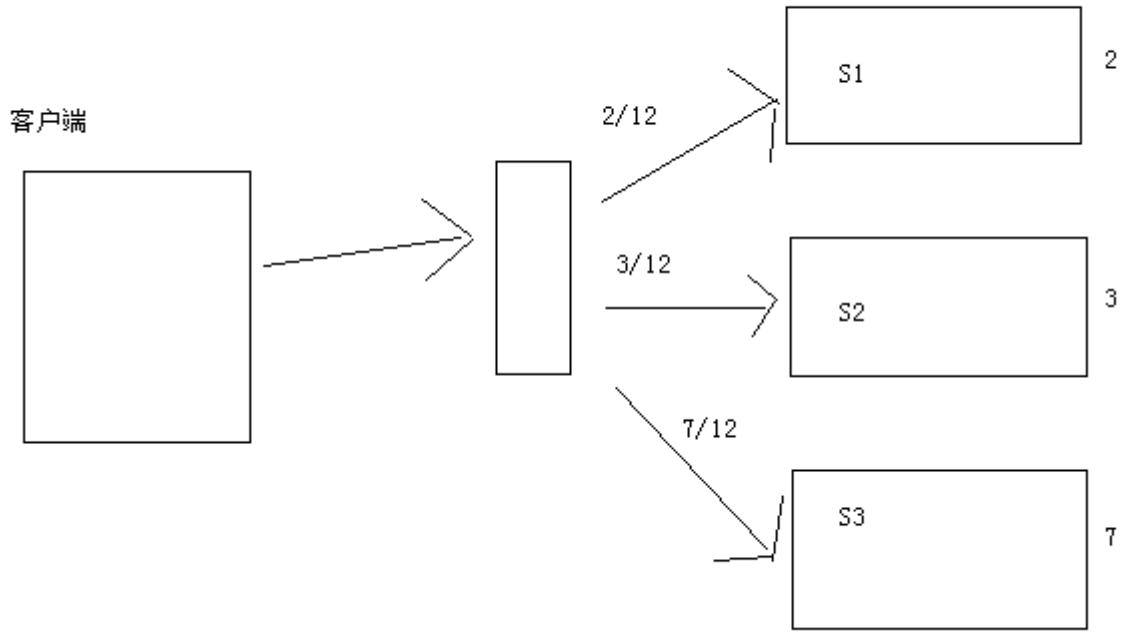
## 1、负载均衡的概念

> Load Balance，其意思就是分摊到多个操作单元上进行执行
>
> 集群中的服务器，通过负载均衡算法，分担 访问压力

## 2、算法-Random LoadBalance

**基于权重的随机负载均衡机制**

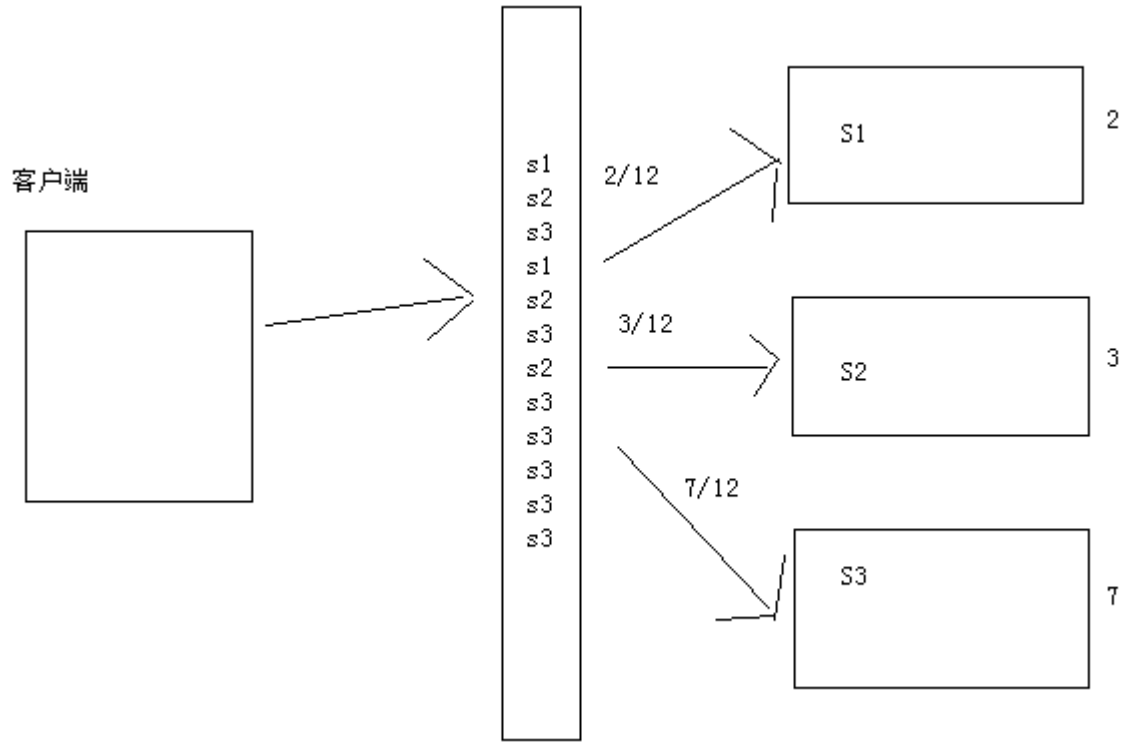
> 随机：按权重设置随机概率，能者多劳
>
> 在一个截面上碰撞的概率高，但**调用量越大分布越均匀**，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重




## 3、算法-RoundRobin LoadBalance

**基于权重的 轮询 负载均衡机制**

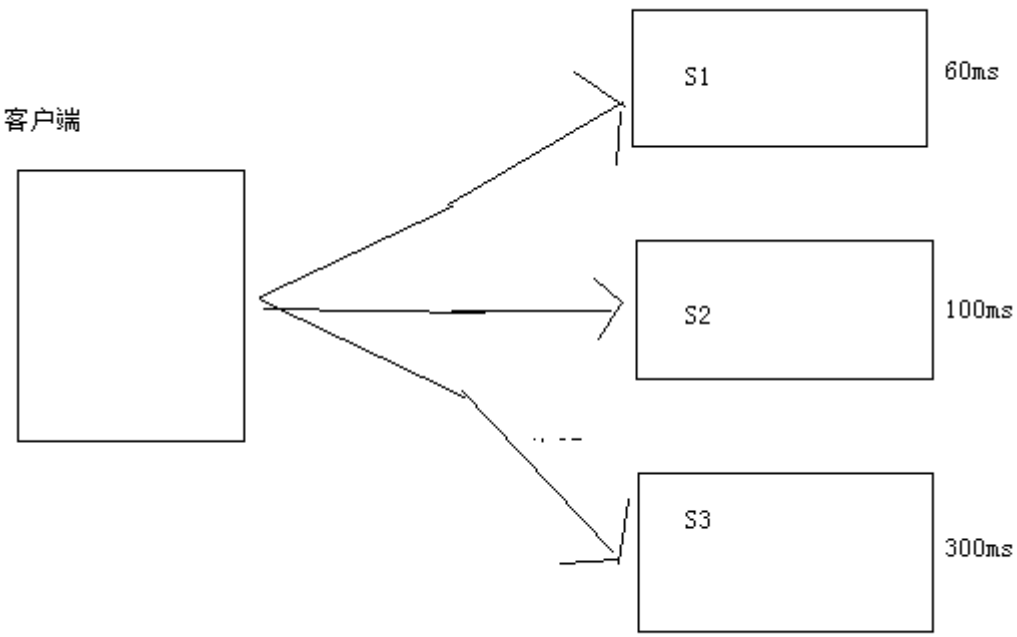
> 轮询按公约后的权重设置轮询比率。
>
> 存在慢的提供者累积请求的问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。




## 4、算法-LeastActive LoadBalance

**最少活跃的 负载均衡机制**

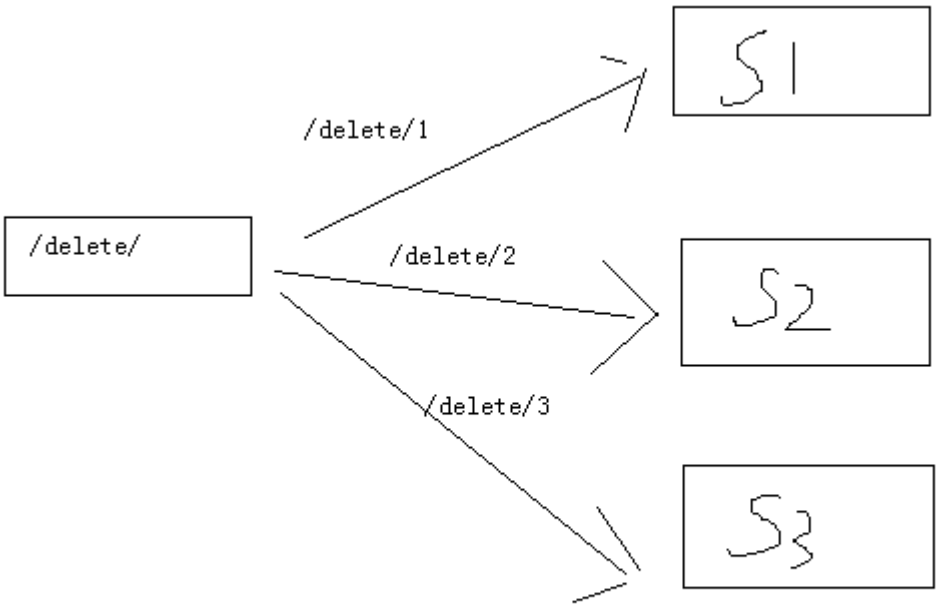
> 最少活跃调用数相同活跃数的随机，活跃数指调用前后计数差。
>
> 使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。




## 5、算法-ConsistentHash LoadBalance

**一致性hash 负载均衡机制**

> 一致性 Hash，**相同参数的请求总是发到同一提供者。**
>
> 当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。
>
> 算法参见：http://en.wikipedia.org/wiki/Consistent_hashing
>
> 缺省只对第一个参数 Hash，如果要修改，请配置 `<dubbo:parameter key="hash.arguments" value="0,1" />`
>
> 缺省用 160 份虚拟节点，如果要修改，请配置 `<dubbo:parameter key="hash.nodes" value="320" />`




## 6、实现方式

```
<dubbo:service loadbalance="roundrobin" weight="7/1/2"
               interface="com.itany.api.SomeService" ref="someService" >
</dubbo:service>
```

```
os.test(1);
Thread.sleep(2000);
os.test(2);
Thread.sleep(2000);
os.test(3);
Thread.sleep(2000);
```