

# Dubbo

Not Given

2022 年 7 月 24 日

## 目录

<b>第一章：分布式系统中的相关概念</b>	<b>1</b>
一、传统项目与互联网项目 . . . . .	1
二、集群与分布式 . . . . .	1
三、架构的演变 . . . . .	2
<b>第二章：Dubbo 的简介与基本使用</b>	<b>3</b>
一、简介 . . . . .	3
二、ZooKeeper . . . . .	3
三、基本使用方式 . . . . .	3
<b>第三章：Dubbo 的高级特性</b>	<b>5</b>
一、Dubbo-admin . . . . .	5
二、序列化 . . . . .	5
三、地址缓存 . . . . .	6
四、超时与重试 . . . . .	6
五、多版本 . . . . .	7
六、负载均衡 . . . . .	7
七、集群容错 . . . . .	8
八、服务降级 . . . . .	8

# 第一章：分布式系统中的相关概念

## 一、传统项目与互联网项目

1. 传统项目：公司内部系统，主要使用群体为公司员工
2. 互联网项目：微信、百度等，使用群体为网民
3. 互联网项目面临的挑战：用户基数大，高并发，用户忍耐力低
4. 互联网项目的设计目标

- **高性能**：提供快速的访问体验。
- **高可用**：网站服务一直可以正常访问。
- **可伸缩**：通过硬件增加/减少，提高/降低处理能力。
- **高可扩展**：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。

### 5. 衡量网站的性能指标

#### 衡量网站的性能指标：

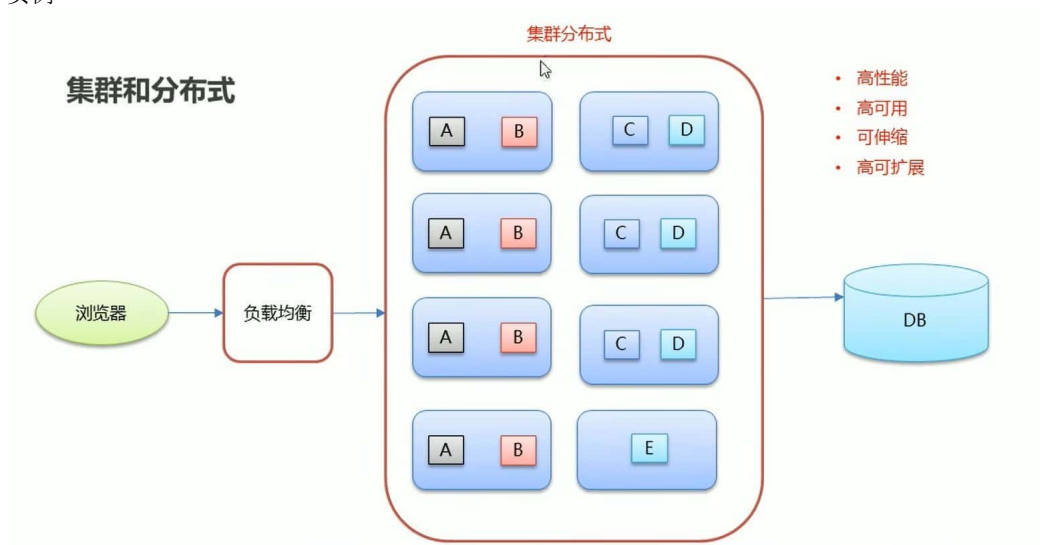
- **响应时间**：指执行一个请求从开始到最后收到响应数据所花费的总体时间。
- **并发数**：指系统同时能处理的请求数量。
  - **并发连接数**：指的是客户端向服务器发起请求，并建立了TCP连接。每秒钟服务器连接的总TCP数量
  - **请求数**：也称为QPS(Query Per Second) 指每秒多少请求。
  - **并发用户数**：单位时间内有多少用户
- **吞吐量**：指单位时间内系统能处理的请求数量。
  - **QPS**：Query Per Second 每秒查询数。
  - **TPS**：Transactions Per Second 每秒事务数。
  - 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。

## 二、集群与分布式

### 1. 基本概念

- **集群**：很多“人”一起，干一样的事。
- **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。

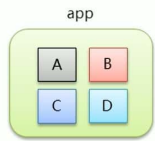
### 2. 实例



三、架构的演变

1. 单体架构

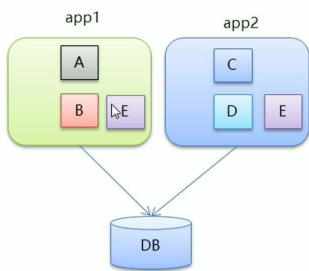
架构演进--单体架构



- 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
  - 扩展性和可维护性差
  - 性能低

2. 垂直架构

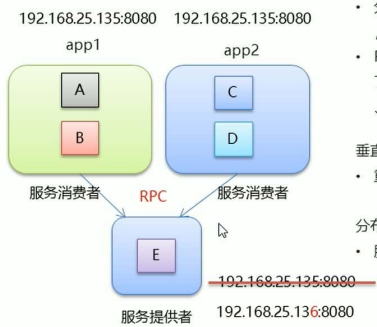
架构演进--垂直架构



- 垂直架构是指将单体架构中的多个模块拆分为多个独立的项目, 形成多个独立的单体架构。
- 单体架构存在的问题:
- 项目启动慢
  - 可靠性差
  - 可伸缩性差
  - 扩展性和可维护性差
  - 性能低
- 垂直架构存在的问题:
- 重复功能太多

3. 分布式架构

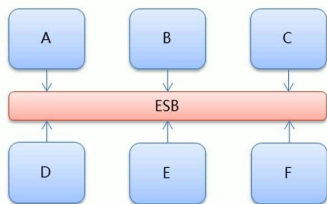
架构演进--分布式架构



- 分布式架构是指在垂直架构的基础上, 将公共业务模块抽取出来, 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。
  - 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。
- 垂直架构存在的问题:
- 重复功能太多
- 分布式架构存在的问题:
- 服务提供方一旦产生变更, 所有消费方都需要变更。

4. SOA 架构

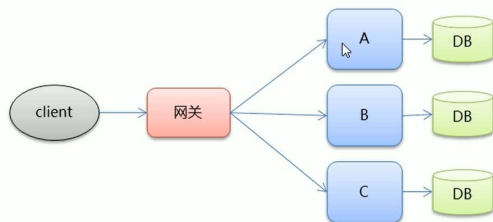
架构演进--SOA架构



- 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。
  - ESB: (Enterprise 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。
- 分布式架构存在的问题:
- 服务提供方一旦产生变更, 所有消费方都需要变更

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

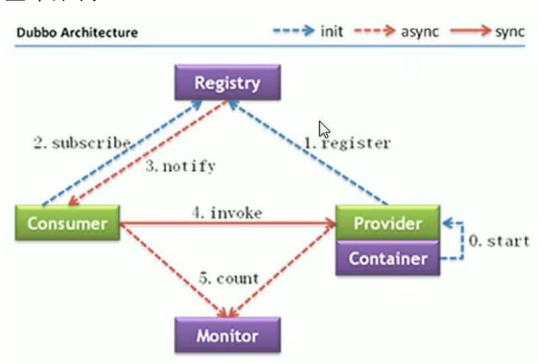


- 特点:
- 服务实现组件化: 开发者可以自由选择开发技术, 也不需要协调其他团队
- 服务之间交互一般使用REST API
- 去中心化: 每个微服务有自己私有的数据库持久化业务数据
- 自动化部署: 把应用拆分成为一个一个独立的单个服务, 方便自动化部署、测试、运维

## 第二章：Dubbo 的简介与基本使用

### 一、简介

1. 概念：一款高性能、轻量级的 RPC 开发框架
2. 基本架构



### 二、ZooKeeper

1. 概念：一种供 Dubbo 使用的注册中心
2. 安装方式：<https://www.bilibili.com/video/BV1VE411q7dX?p=7>

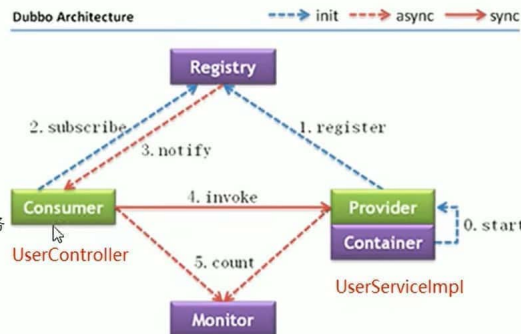
### 三、基本使用方式

1. 实现步骤

#### Dubbo快速入门

##### 实现步骤:

- ① 创建服务提供者Provider模块
- ② 创建服务消费者Consumer模块
- ③ 在服务提供者模块编写 UserServiceImpl 提供服务
- ④ 在服务消费者中的 UserController 远程调用 UserServiceImpl 提供的服务
- ⑤ 分别启动两个服务，测试



2. 创建项目：用 Spring 和 SpringMVC 创建两个项目，分别作为 Service 模块和 Web 模块，其中 Service 可以调用 Web 模块。但是这时项目还不算分布式的，因为 Service 模块不能独立运行
3. 配置 Service 模块

#### (1) 导入坐标

```
<!--Dubbo的起步依赖，版本2.7之后统一为org.apache.dubbo-->
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>${dubbo.version}</version>
</dependency>
<!--ZooKeeper客户端实现-->
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>${zookeeper.version}</version>
</dependency>
<!--ZooKeeper客户端实现-->
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>${zookeeper.version}</version>
</dependency>
```

#### (2) 设置方法注解：目的为将提供服务的类对外发布

```
@Service//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
public class UserServiceImpl implements UserService {
    public String sayHello() { return "hello dubbo!"; }
}
```

### (3) Dubbo 配置

```
<!--dubbo的配置-->
<!--1.配置项目的名称,唯一-->
<dubbo:application name="dubbo-service"/>
<!--2.配置注册中心的地址-->
<dubbo:registry address="zookeeper://192.168.149.135:2181"/>
<!--3.配置dubbo包扫描-->
<dubbo:annotation package="com.itheima.service.impl" />
```

## 4. 配置 Web 模块

### (1) 导入坐标

```
<!--Dubbo的起步依赖,版本2.7之后统一为org.apache.dubb -->
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>${dubbo.version}</version>
</dependency>
<!--Zookeeper客户端实现 -->
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>${zookeeper.version}</version>
</dependency>
<!--Zookeeper客户端实现 -->
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>${zookeeper.version}</version>
</dependency>
```

### (2) 设置方法注解：目的为进行远程调用

```
//注入Service
//@Autowired//本地注入

/*
    1. 从zookeeper注册中心获取userService的访问url
    2. 进行远程调用RPC
    3. 将结果封装为一个代理对象。给变量赋值
*/

@Reference//远程注入
private UserService userService;

@RequestMapping("/sayHello")
public String sayHello(){
    return userService.sayHello();
}
```

### (3) Dubbo 配置

```
<!--dubbo的配置-->
<!--1.配置项目的名称,唯一-->
<dubbo:application name="dubbo-service"/>
<!--2.配置注册中心的地址-->
<dubbo:registry address="zookeeper://192.168.149.135:2181"/>
<!--3.配置dubbo包扫描-->
<dubbo:annotation package="com.itheima.service.impl" />
```

## 5. 配置公共接口：在两个模块的 pom.xml 文件中分别进行配置，让它们都依赖于公共接口模块，这样可以避免两个项目引入的接口不一致

```
<!--依赖公共的接口模块-->
<dependency>
    <groupId>com.itheima</groupId>
    <artifactId>dubbo-interface</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

## 第三章：Dubbo 的高级特性

### 一、Dubbo-admin

#### 1. 概念：一种图形化的服务管理平台

- dubbo-admin 管理平台，是图形化的服务管理页面
- 从注册中心中获取到所有的提供者 / 消费者进行配置管理
- 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
- 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

#### 2. 安装方式：<https://www.bilibili.com/video/BV1VE411q7dX?p=11>

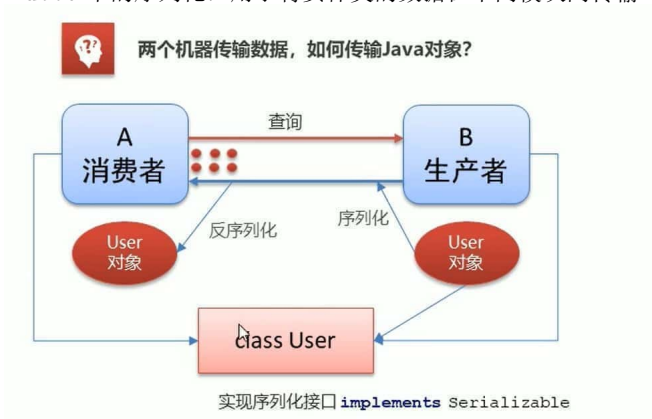
#### 3. 基本使用：目前只有查询功能



### 二、序列化

#### 1. 概念：将 java 对象转化为流的数据，然后进行传输

#### 2. Dubbo 中的序列化：用于将实体类的数据在不同模块间传输



#### 3. 示例



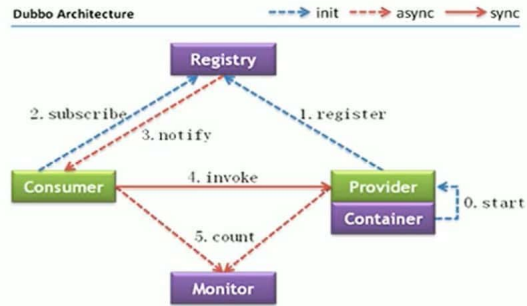
### 三、地址缓存

#### 地址缓存



注册中心挂了，服务是否可以正常访问？

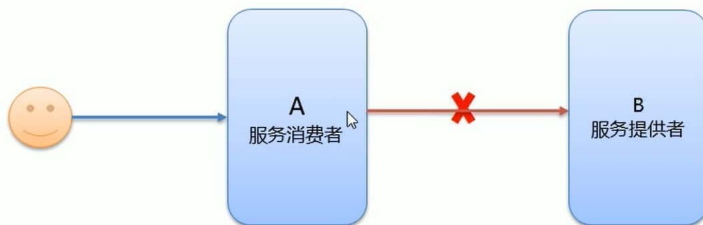
- 可以，因为dubbo服务消费者在第一次调用时，会将服务提供方地址缓存到本地，以后在调用则不会访问注册中心。
- 当服务提供者地址发生变化时，注册中心会通知服务消费者。



### 四、超时与重试

1. 超时机制：在一定时间内请求不到服务，端口会被释放。一般在 Service 中配置

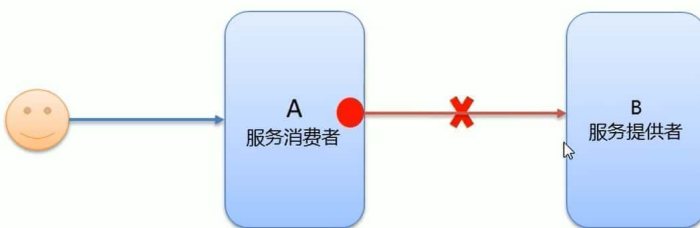
#### 超时与重试



- 服务消费者在调用服务提供者的时候发生了阻塞、等待的情形，这个时候，服务消费者会一直等待下去。
- 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。
- 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。

2. 重试机制：在网络抖动或超时后重新发送请求

#### 超时与重试



- 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。
- 如果出现网络抖动，则这一次请求就会失败。
- Dubbo 提供重试机制来避免类似问题的发生。
- 通过 `retries` 属性来设置重试次数。默认为 2 次。

### 3. 示例

```
//@Service//将该类的对象创建出来，放到Spring的IOC容器中 bean定义
//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
@Service(timeout = 3000, retries = 0)//当前服务 3秒超时
public class UserServiceImpl implements UserService {

    public String sayHello() { return "hello dubbo hello!~"; }

    public User findUserById(int id) {
        //查询User对象
        User user = new User( id: 1, username: "zhangsan", password: "123");
        return user;
    }
}
```

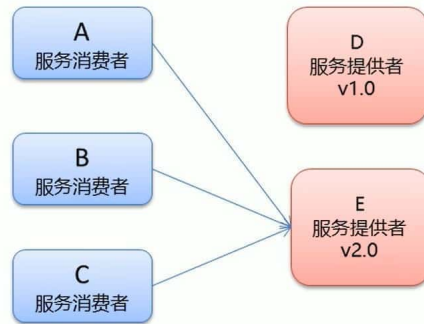


## 五、多版本

### 1. 概念：设置调用统一接口的不同版本

#### 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



### 2. 示例

```
//@Service//将该类的对象创建出来，放到Spring的IOC容器中 bean定义
//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
@Service(version = "v2.0")
public class UserServiceImpl2 implements UserService {

    public String sayHello() { return "hello dubbo hello!~"; }

    public User findUserById(int id) {
        System.out.println("new...");
        //查询User对象
        User user = new User( id: 1, username: "zhangsan", password: "123");
        return user;
    }
}
```

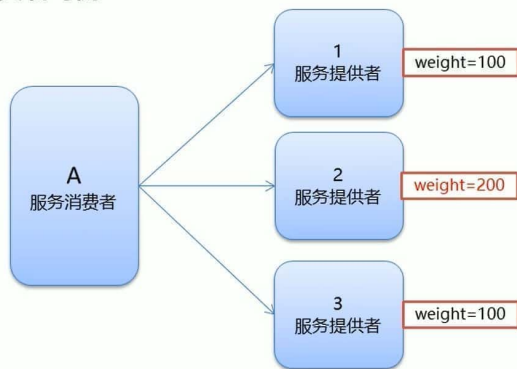
```
@Reference(version = "v2.0")//远程注入
private UserService userService;
```

## 六、负载均衡

### 1. 概念：将服务请求进行分流

### 2. Random

#### 负载均衡

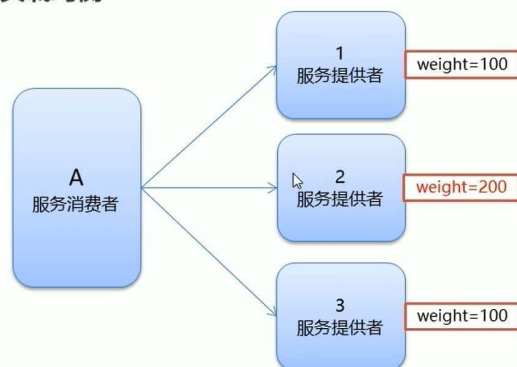


负载均衡策略（4种）：

- Random：按权重随机，默认值。按权重设置随机概率。

### 3. RoundRobin

#### 负载均衡



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

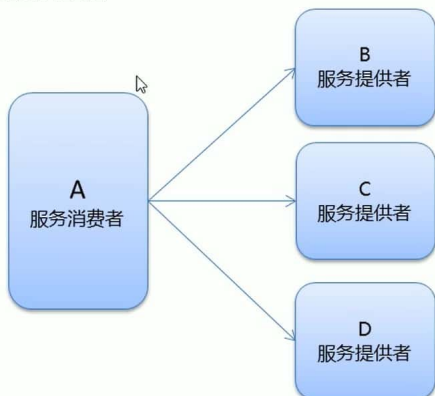


## 七、集群容错

1. 概念：出错后的应对策略

2. 分类

### 集群容错



集群容错模式：

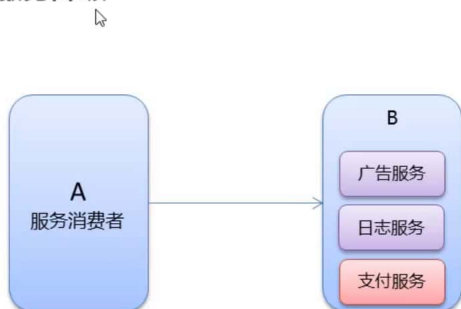
- Failover Cluster：失败重试。默认值。当出现失败，重试其它服务器，默认重试2次，使用 `retries` 配置。一般用于读操作。
- Failfast Cluster：快速失败，只发起一次调用，失败立即报错。通常用于写操作。
- Failsafe Cluster：失败安全，出现异常时，直接忽略。返回一个空结果。
- Failback Cluster：失败自动恢复，后台记录失败请求，定时重发。
- Forking Cluster：并行调用多个服务器，只要一个成功即返回。
- Broadcast Cluster：广播调用所有提供者，逐个调用，任意一台报错则报错。

## 八、服务降级

1. 概念：当负载过度时，将不重要的服务关掉，释放资源以保证主要服务可以正常运行

2. 分类

### 服务降级



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。