dubbo

一.架构演进的概念

1. 单体架构:

- 所谓的单体架构呢，就是将我们的整个项目放在一台服务器上部署，例如：一个项目中有很多个模块，比如，用户，支付，商品，购物车，订单，后台管理等多种模块，部署到一块。

2. 水平垂直架构:

- 所谓的水平垂直呢，也就是将单体架构进行水平扩展，和垂直扩展，水平扩展就是当我们一台服务不够支撑用户的并发，我们可以将我们的项目复制一份多搭建几台，在使用负载均衡做个代理转发。垂直扩展：它就是将一台服务器的设备增大，但这些问题都是从硬件做了调整，而技术没有多大的变化，而且我们的项目也不容易扩展。

3. 分布式架构:

- 所谓的分布式架构，就是将项目按照业务进行拆分，模块与模块分离，可以将模块部署到不同服务上，这样更容易对某一个固定的模块进行扩展，精准扩展，但这样的拆分虽好，所带来一些列的技术问题，各个服务之间怎么管理，怎么治理。带来一些列的调用，降级，容错，负载，服务发现和注册，监控等。

4. 微服务架构:

- 所谓的微服务架构，我们需要从一篇论文说起，论文的作者叫马丁福乐，它论文就是将我们的服务更细粒度的差分，形成一个庞大的服务网络，服务拆分粒度更小，但服务与服务之间有这某种通讯方式。

二.RPC的原理（重点）

1. 期待目标：可以自己模拟一个RPC远程调用协议
2. 原理：我们不说大理论，举个例子：

- 我们的dubbo呢一般有三个关键角色，注册中心，服务消费者，服务提供者，服务消费者订阅注册中心的某个服务提供者，服务提供者将自己的服务名，url，地址，协议注册到注册中心（zookeeper）,一旦注册中心的数据发生变化，就会通知订阅者，服务消费者，通过调用接口中的方法，就可以调到服务提供者的实现类。
- 问题：怎么调的，服务器与服务器通信方式一定是由网络通信的，所以服务提供者做的事情：

服务提供者所作的事情 Vue 复制代码

```
1. 接收请求：http请求tomcat,tcp请求netty,就以http请求为列,创建tomcat,
   当然tomcat只认识servlet---->配置一个servlet,在servlet中将请求转化为一个调度对象：
   调度对象是封装了，接口名，接口参数，参数类型，版本等一些信息，
   当然发送方就得封装成这样，回到servlet，调度对象中得到接口名等。
2. 问题：如何接口名找到对应的实现类，可以在服务提供者启动的时候，将接口名，
   接口实现类存到map中，这样就可以在servlet找找到（相当于注册中心的作用）
3. 回到servlet：servlet根据实现类，使用反射调用相应的方法，接收结果，响应请求
```

服务消费者所做的事情 Vue 复制代码

```
1.发送请求：将接口名，接口参数，参数类型，版本等一些信息，封装成一个对象
2.使用http或者java提供的url类发送请求
3.获取响应结果
问题：服务每次发请求都要new调度对象，发送，那dubbo中指需要调用接口，就可以，怎么做到
我们可以搞一个代理对象，发送请求，我们只需要调用代理对象的方法即可，发送到哪里，从注册中
心获取服务List<URl>--->负载均衡算发，发送到某一台服务器.
```

3. 服务消费者和提供者如何统一协议：

- 服务提供者将自己的协议注册到注册中心，当服务消费者订阅了注册中心，就会查到协议版本，使用统一的版本

三.dubbo如何实现远程通信

服务提供者将自己的服务信息注册到注册中心，服务消费者将自己的订阅注册中心的服务即可，注册中心有很多zookeeper,redis等
代码实现方式：

XML 复制代码

```xml
<dependency>
  <groupId>org.apache.dubbo</groupId>
  <artifactId>dubbo-spring-boot-starter</artifactId>
  <version>2.7.3</version>
</dependency>
<dependency>
  <groupId>org.apache.dubbo</groupId>
  <artifactId>dubbo-registry-zookeeper</artifactId>
  <version>2.7.3</version>
  <exclusions>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
    </exclusion>
  </exclusions>
 </dependency>
```

关键代码 Java 复制代码

```java
//暴露接口
public interface SiteService {
 Site getSiteById(Long id);
}
//服务提供者 dubbo里的注解
@Service
impl implements SiteService{
}
//消费者
@Reference
private SiteService service;
//启动类
@EnableDubbo
```

四.dubbo如何进行服务超时，指定版本号，服务容错，负载均衡

- 指定版本号

最关键的是注解 Java 复制代码

```
版本控制
@Service(version = "async")
impl implements SiteService{
}
@Service(version = "default")
impl implements SiteService{
}
@Reference(id = "siteService",version = "async")
private SiteService siteService;
```

- 超时，容错，负载均衡

注释的参数 Java 复制代码

```
@Reference(version = "timeout",
           interfaceName = "timeOutSiteServiceImpl",
           timeout = 1000,
           mock = "fail:return HaHa",
           loadbalance="random")
version: 是当服务接口提供者有多个以上我们可以指定固定版本名
interfaceName: 当服务接口提供者实现了多个，会生成多个SiteService类型的代理对象，
               指定具体的代理对象
timeout: 服务超时，如果在提供者或者消费者端配置一个，默认都是配置一样的超时时间，
         代表的是消费着调用你的接口只需6秒，6秒内你给我响应，不响应我就报错
mock: 做服务降级用的 可以返回抛出的异常类，可以是字符串 可以是接口的拖低数据
loadbalance: 指定负载均衡的策略
```

四.dubbo提供的负载均衡算法

dubbo提供的负载均衡算法：轮询，权重，随机，一致性hash,最少活跃调用

1. 轮询，权重，随机：略
2. 最少活跃调用: 最少活跃调用：dubbo的服务消费者，都会在消费端的缓存记录每个服务提供者的活跃数 用active,如果发送一次请求，active+1,响应回来 active-1根据这个过程推算可知，active越高你的响应越慢，所以叫最少活跃调用
3. 一致性hash: 什么是一致性hash，一致性hash应用与redis,dubbo，我们在一个hash环上有16384个hash槽， 服务提供者的IP有三个，对ip进行hash运算得到三个hash槽,这三个hash就在hash环上，每当来一个请求， redis对key做运算，dubbo对ip运算,得到的hash值%16384得到0~16383个中的一个值，这个hash槽到那个ip近就去访问那个 当然服务提供者的三个iphash也是随机的如何保证均

匀的分再hash环上，可以采用虚拟结点，将三个hash映射出多个虚拟ip,分散在环上这样就能够相对平等分布

五. dubbo支持protocol协议

1. rest协议
2. http协议
3. dubbo协议

实现方式 XML 复制代码

```
dubbo.protocols.protocol1.id=rest
dubbo.protocols.protocol1.name=rest
dubbo.protocols.protocol1.port=8090
dubbo.protocols.protocol1.host=0.0.0.0
dubbo.protocols.protocol2.id=dubbo1
dubbo.protocols.protocol2.name=dubbo
dubbo.protocols.protocol2.port=20882
dubbo.protocols.protocol2.host=0.0.0.0
```

java注解引入协议 Java 复制代码

```java
//dubbo
@Service(version = "default",protocol = "protocol2")
```

注意：其他协议配置请参考官网，和技术文档，日更文档的dubbo中找

六.dubbo集群容错方案

dubbo为集群调用提供了容错方案：

- failover：（默认，推荐）

当出现失败时，会进行重试，默认重试2次，一共三次调用。但是会出现幂等性问题。

虽然会出现幂等性问题，但是依然推荐使用这种容错机制，在业务层面解决幂等性问题：

- 方案一：把数据的业务id作为数据库的联合主键，此时业务id不能重复。

- 方案二（推荐）：使用分布式锁来解决重复消费问题。

- failfast：当出现失败时。立即报错，不进行重试。
- failsafe：失败不报错，记入日志。
- failback：失败就失败，开启定时任务 定时重发。
- forking：并行访问多个服务器，获取某一个结果既视为成功。

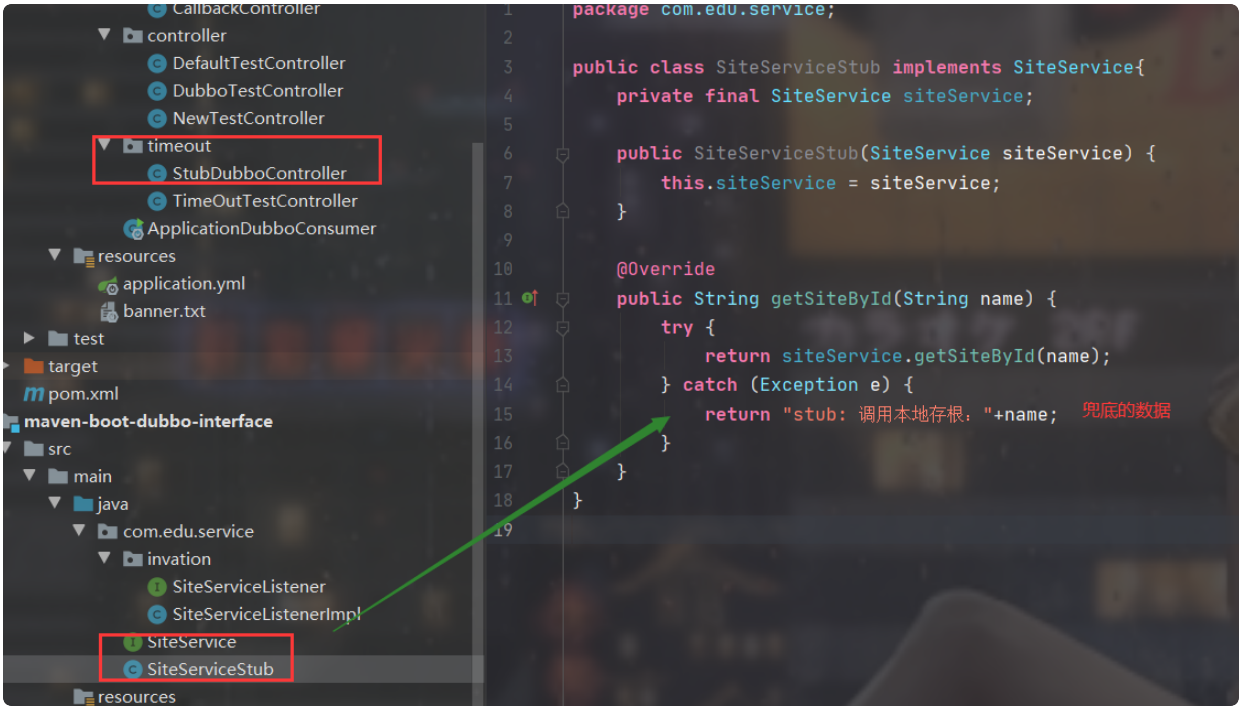
结论：如果使用dubbo，不推荐把重试关掉，而是在非幂等性操作的场景下，服务提供者方要做幂等性的解决方案（保证）。

七.dubbo的本地存根（业务出错，找兜底方案）

- 大白话：什么叫本地存根：就是服务消费者在@Service代理的对象上在包装一层代理对象，让代理对象去调用服务提供者的接口方法，如果正常返回，就正常调用，没有正常返回我们可以做一些错误的兜底处理。
- 官方话： 远程服务后，客户端通常只剩下接口，而实现全在服务器端，但提供方有些时候想在客户端 也执行部分逻辑，比如：做 ThreadLocal 缓存，提前验证参数，调用失败后伪造容错数据等等，此时就需要在 API 中带上 Stub，客户端生成 Proxy 实例，会把 Proxy 通过构造函数传给 Stub 1，然后把 Stub 暴露给用户，Stub 可以决定要不要去调 Proxy。
- 实现方式

本地存根实现 Java 复制代码

```java
stub:  本地存根，会在本服务器找一个SiteServiceStub的实现类去调用，
SiteServiceStub代理去调用远程，如果捕获到异常，返回一个托底数据
 @Reference(version = "timeout",
         interfaceName = "timeOutSiteServiceImpl",
         timeout = 1000,
         stub = "true")
 private SiteService siteService;
```



八.dubbo的参数回调和异步调用

1. 参数回调

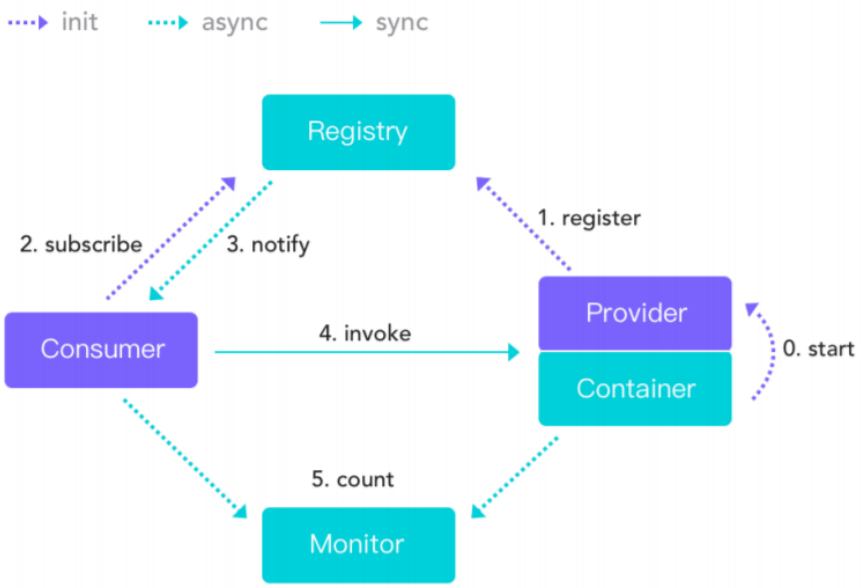
- 大白话：服务消费者调用服务提供者的同时，服务提供者可以调用 服务消费者
- 官方话： 参数回调方式与调用本地 callback 或 listener 相同，只需要在 Spring 的配置文件中声明哪个 参数是 callback 类型即可。Dubbo 将基于长连接生成反向代理，这样就可以从服务器端调用客户端逻辑。

2.异步调用

- 大白话：服务消费者调用服务提供者可以不等响应回来，先执行自己的业务

注意：实现方式参考笔记文档

九.dubbo核心架构图



官方资料：https://dubbo.apache.org/zh/docs/advanced/

视频地址:

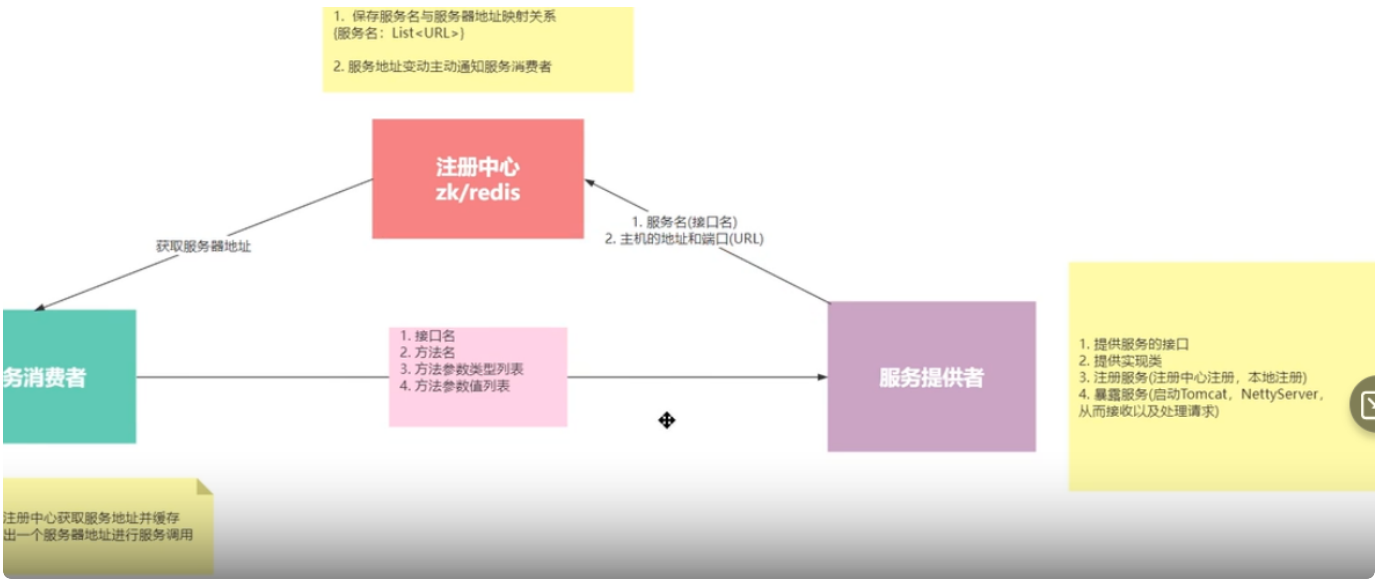
基础课程：https://www.bilibili.com/video/BV1jh41187KQ?p=35&spm_id_from=333.880.my_history.page.click

大佬课程：https://www.bilibili.com/video/BV1J44y1a7Qw?p=9&spm_id_from=333.880.my_history.page.click

学习资料： source_everyday->zookeeper_dubbo

手写RPC远程调用：E:\mavenoffcnidea-work\maven-boot-dubbo-parent

十.手写RPC架构思路图



```
String name = System.getProperty("protocolName");
```

这个代码可以直接获取JVM配置参数–
