# 认识Apache Dubbo

## 为什么要使用Apache Dubbo

服务于服务之间的远程通信是分布式架构最基本的组成部分，但传统意义上的远程通信， 更多的时候是解决信息孤岛及数据互联问题的，它主要关注的是数据的共享。随着SOA 生态的不断完善以及微服务架构思想的落地，服务于服务之间的远程通信需求更多来自 服务的解耦。同时，业务规模的不断增粘会使得微服务数量增加，那么问题也就随之产 生了，比如：

·如何协调线上运行的服务，以及保障服务的高可用性

·如何根据不同服务的访问情况来合理地调控服务器资源，提高机器的利用率

·线上出现故障时，如何动态地对故障业务做降低、流量控制等

·如何动态地更新服务中的配置信息，比如限流阈（yu读声）值、降级开关等

·如何实现大规模服务集群所带来的服务地址的管理和服务上下线的动态感知

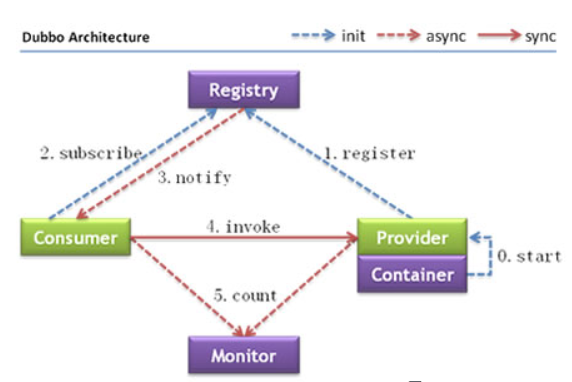
为了解决这些问题，就需要一个统一的服务治理框架对服务进行统一、有效的管控，从 而保障服务的高效、健康运行，而Dubbo就是这样一个框架。Dubbo是阿里巴巴内部 使用的一个分布式服务治理框架，于2012年开源。由于Dubbo在服务治理这一领域的 优势，以及它本身在阿里巴巴经过大规模的业务验证，所以在很多的时间内，Dubbo 就被很多互联网公司采用，笔者所在公司内部把WebService切换到Dubbo框架。后 Dubbo进入Apache，这意味着Dubbo不再只是阿里巴巴的Dubbo，而是属于开源社区 的，也意味着会有更多的开源贡献者参与到Dubbo的开发中来。

促使Apache Dubbo框架产生的原因有两个：

1. 在大规模服务化之后，服务越来越多，服务消费者在调用服务提供者的服务时，需要在配置文件中维护服务提供者的URL地址，当服务提供者出现故障或者动态扩容时，所有相关的消费者都需要更新本地配置的URL地址，这种维护成本非常高。这个时候，实现服务的上下动态感知及服务地址的动态维护就显得非常重要了。
2. 随着用户的访问量增加，后端服务为了支撑更大的访问量，会通过增加服务器来扩容。但是，哪些服务要扩容，哪些服务要缩容，需要一个判断依据，也就是说需要知道每个服务的调用量及响应时间，这个时候，就需要有一种监控手段，使用监控的数据作为容量规划的参考值，从而实现根据不同服务的访问情况来合理地调控服务器资源，提高机器的利用率。

## 什么是Apache Dubbo

Apache Dubbo是一个开源的分布式服务框架，主要实现多个系统之间的高性能、透明 化调用，简单来说它就是一个RPC框架，但是和普通的RPC框架不同的是，它提供了 服务治理功能，比如服务注册、监控、路由、容错等。从如下图所示的Apache Dubbo 架构图也能够清晰地看出，除了基本的RPC框架的只能，它的核心功能便是监控和 服务注册。



inti（初始化） async（异步） sync（同步）

subscribe（订阅） notify（通知） invoke（调用） count（计数）

Provider：暴露服务方称之为“服务提供者”

Consumer：调用远程服务方称之为“服务消费者”

Registry：服务注册与发现的中心目录服务称之为“服务注册中心”

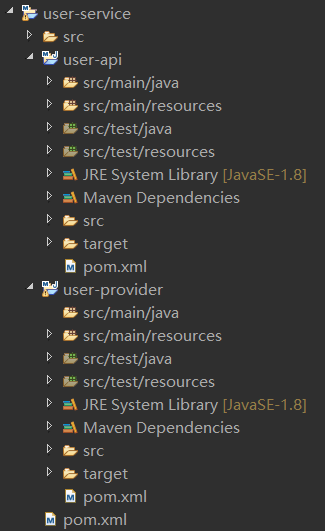
Monitor：统计服务的调用次数和调用时间的日志服务称之为“服务监控中心”

Container：服务运行容器

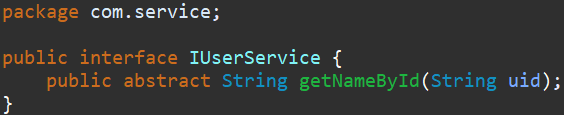
## Apache Dubbo实现远程通信

创建两个普通的Maven工程，分别为order-service和user-service，代表订单服务和 用户服务，这两个服务之间在实际业务场景中会存在相互依赖的情况，比如订单服务中 的某个功能可能需要查询用户信息等，就需要调用用户服务指定的接口来完成。

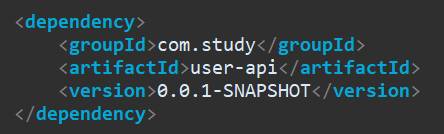
1. **user-service的实现流程**
2. 在user-service服务中定义了两个模块，分别为user-api和user-provider，前者用来定义当前服务对外提供的接口，这个模块会部署到Maven的远程仓库中，便于服务调用者依赖，后者是针对这个接口的实现，该实现会独立部署在服务上。

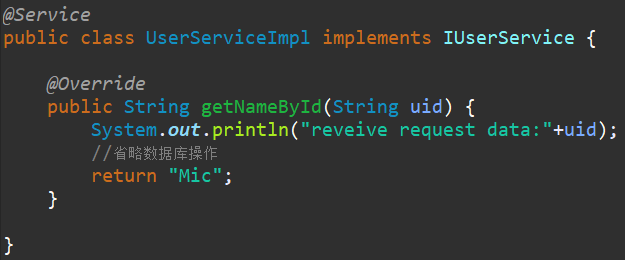


1. 在user-api模块中定义一个接口，执行mvn install将其打包成Jar包安装到本地仓库中，本地环境中的其他项目就可以找到该依赖；当然，如果自己搭建了私服，可以通过mvn deploy发布。

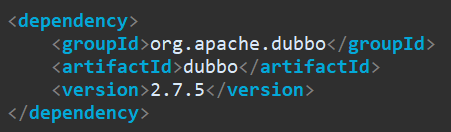


1. 在user-provider模块中编写实现，这里需要注意的是，user-provider中需要用到user-api中定义的IUserService接口，所以需要先在user-provider模块的pom.xml文中添加user-api的maven依赖。

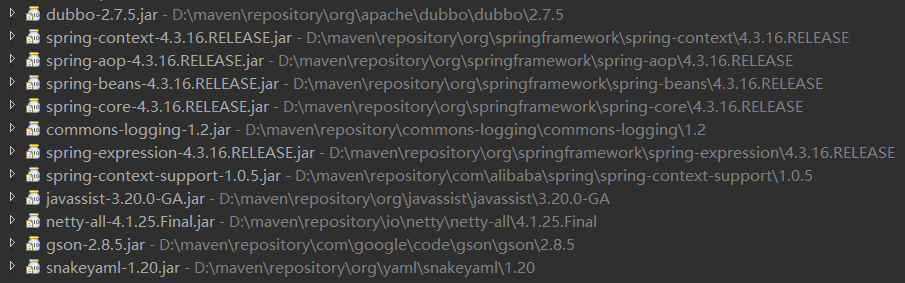




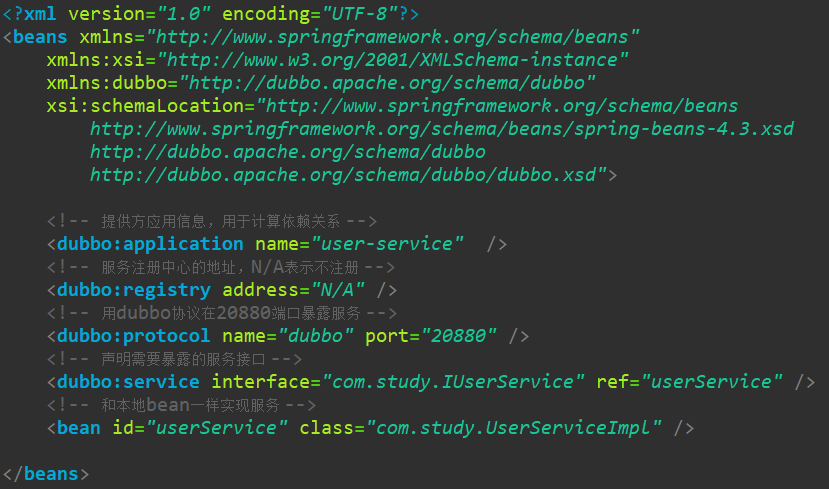
1. 在user-provider模块中添加Dubbo的依赖



添加dubbo依赖后，会自动导入其他依赖。



1. 创建配置文件resources/META-INF/spring/user-provider.xml，把服务发布到网络上，让其他进程可以访问。因为Dubbo采用了Spring配置的扩展类来实现透明化的服务发布和服务消费，所以它的配置基本上和以上通过XML形式描述Bean差不多。



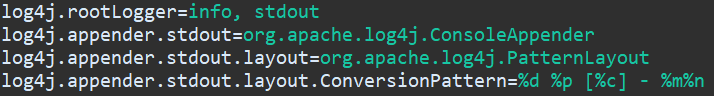
dubbo:application用来描述服务提供方的应用信息，比如应用名称、维护人、版本 等，其中应用名称是必填项。开发者或运维人员可以通过监控平台查看这些信息来 更快速地定位和解决问题。

dubbo:registry配置注册中心的地址，如果不需要注册中心，可以设置为N/A。 Dubbo支持多种注册中心，比如ZooKeeper、Nacos等。

dubbo:protocol配置服务提供者的协议信息、暴露端口，Dubbo支持多种协议来发 布服务，默认采用Dubbo协议，可选的协议有很多，比如Hessian、WebService、 Thrift等。这意味着如果公司之前采用的协议是WebService，想切换到Dubbo上来， 几乎没有太大的迁移成本。

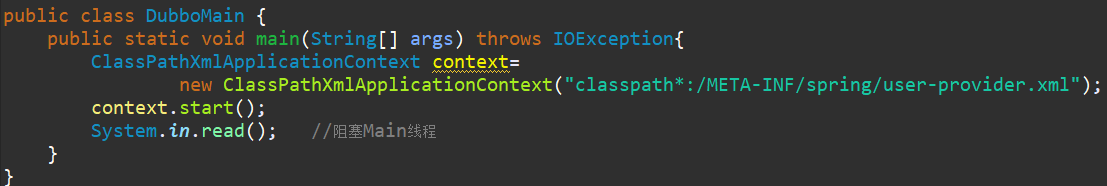
dubbo:service描述需要发布的服务接口，也就是这个接口可以供本网络上的其他进 程访问，interface表示定义的要暴露的接口，ref表示这个接口的实现

1. 在resources目录下创建log4j.properties文件，文件内容如下：

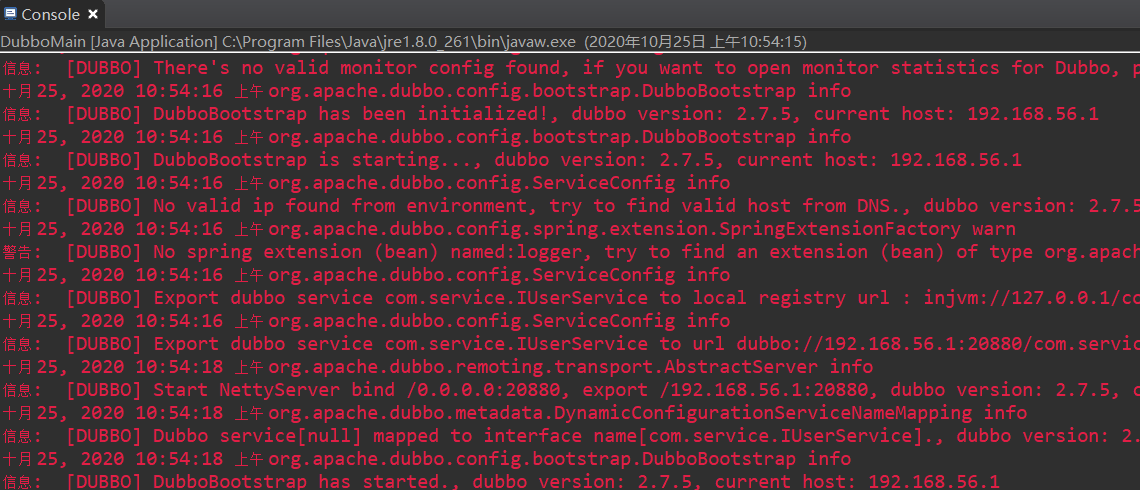


表示在控制台打印Dubbo的日志信息。

1. 加载Spring的XML文件，可以通过ClassPathXmlApplicaton来完成加载启动的过程，也可以通过Main.main(args)来启动。两者在本质上没有任何区别，只是Dubbo做了一层封装，简化了开发者的使用。



启动之后，可以在控制台看到如下信息，说明服务已经发布成功：

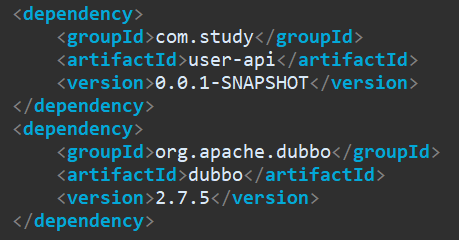


而且还打印了Dubbo发布的地址dubbo://192.168.56.1:20880/com.service.IuserServi ce，这个地址是一个远程通信地址，服务调用者可以基于该地址来访问该服务完成 远程通信。

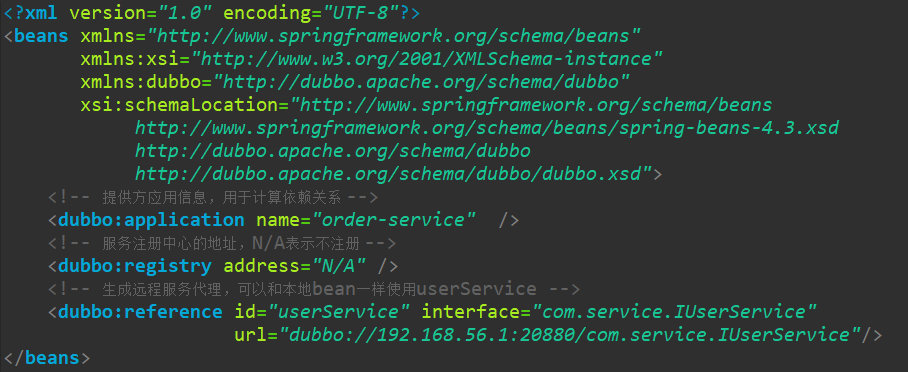
**2、order-service的实现流程**

1）添加user-api和Dubbo的Maven依赖，前者是用户访问IUserService接口的方

法，后者通过通过远程代理完成远程通信过程。



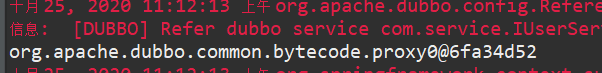
1. 在resources/META-INF/spring/consumer.xml中配置远程服务的引用，主要关注一下dubbo:reference这个配置，它会生成一个针对当前interface的远程服务的代理，指向的远程服务地址是user-service发布的Dubbo协议的URL地址。



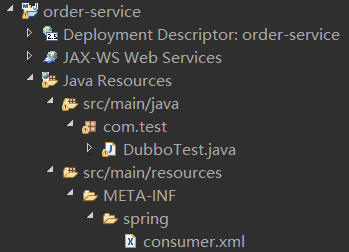
1. 加载Spring配置文件，使用方式和本地Bean一样，通过从IoC容器中获取一个示例对象进行调用，需要注意的是，这里的IUserServie返回的是一个代理对象，它的底层会基于网络通信来实现远程服务的调用。



运行程序，查看控制台打印：



此时，便可调用远程服务。完整的项目目录结构如下：



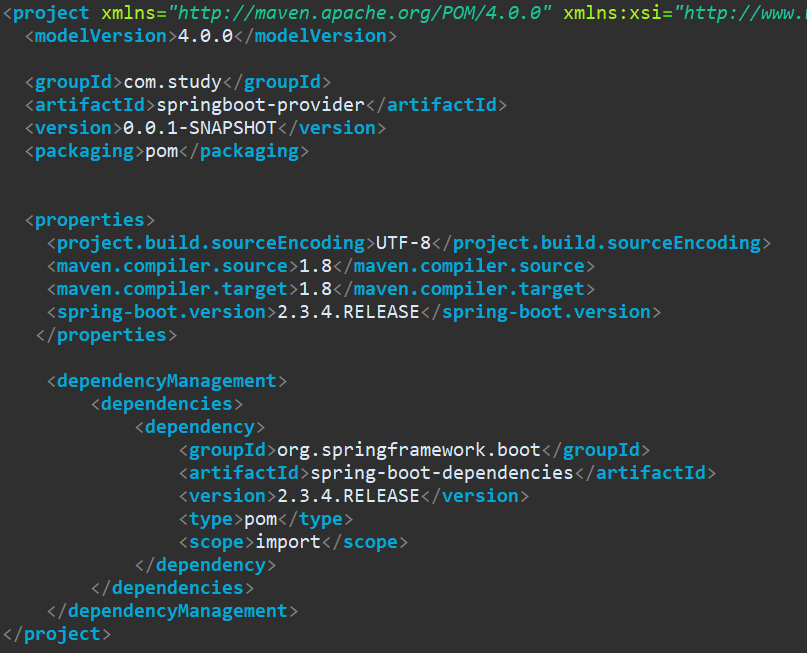
以上案例演示的仅仅是点对点的通信形式。整体看来，由于Dubbo天然地集成了Spring， 并且在此基础上做了标签的扩展，所以整体的配置方式和Spring相差不大，开发者在 使用Dubbo的时候几乎没有太多的学习成本。基于XML形式的服务发布和服务消费方 式还是比较繁琐的，而且在发布的服务接口比较多的情况下，配置会非常复杂，所以 Apache Dubbo也提供了对注解的支持。

## Spring Boot集成Apache Dubbo

在上一节中使用Apache Dubbo实现远程通信是在Spring框架架构中实现的，可以看出 Apache Dubbo不需要依赖Spring Boot也是可以实现微服务的，集成到Spring Boot的好 处是可以享受Spring Boot生态的框架和技术支持，也就是基于Spring Boot实现了标准 化，并统一了开发、部署、运维的形态。在Spring Boot中可以使用Dubbo Spring Boot 组件轻松集成Apache Dubbo，它整合了Spring Boot的自动装配、健康检查、外部化配 置等功能。

**1、服务提供者开发流程**

1）创建一个普通的Maven工程springboot-provider，做为父工程。同时由于要创建的 模块中存在Spring Boot工程类型的模块，所以，在父工程的pom.xml文件中，就需 要使用dependencyManagement来实现，这样Spring Boot工程类型模块就可以不 用继承spring-boot-starter-parent了，但是Java的版本、编码的格式等都需要开发 者手动配置。

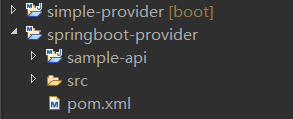


完成，将该父工程打包安装到本地仓库或远程仓库中，供其他项目或Spring Boot

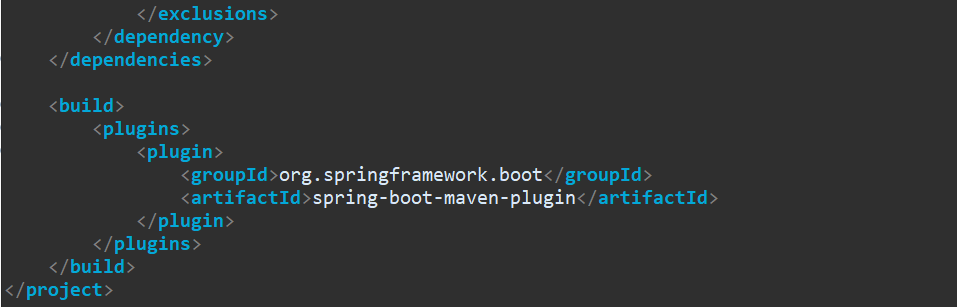
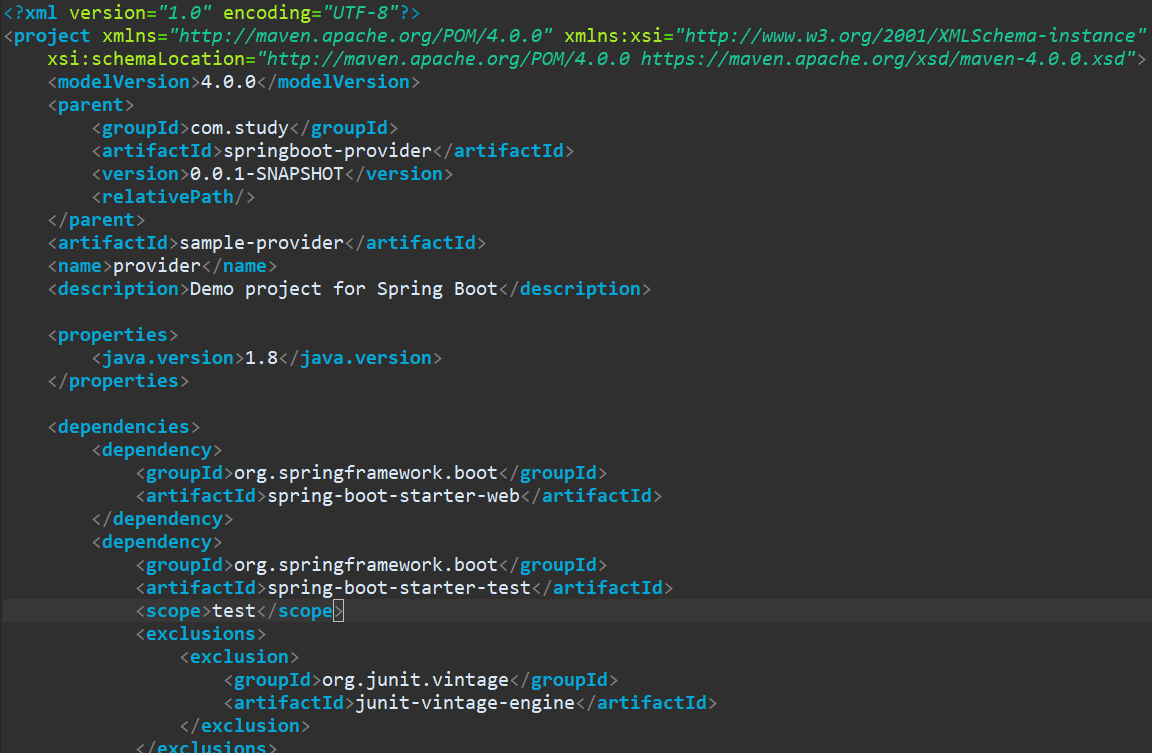
类型模块依赖。

2）创建两个模块，sample-api和sample-provider，其中sample-provider模块是一个

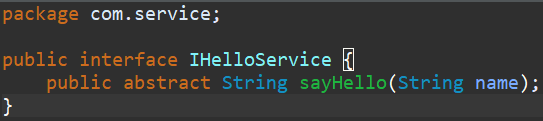
Spring Boot工程。



sample-provider模块是一个Spring Boot工程，pom.xml代码如下：

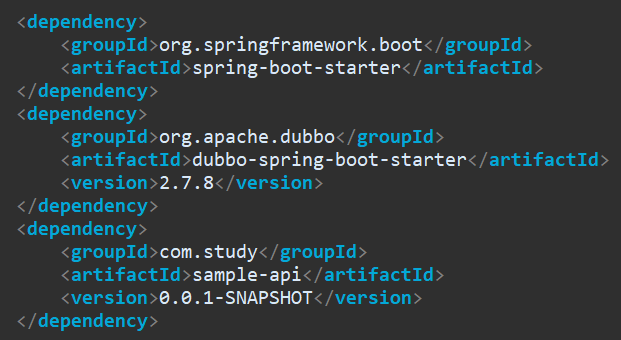
 

3）在sample-api模块中定义一个接口



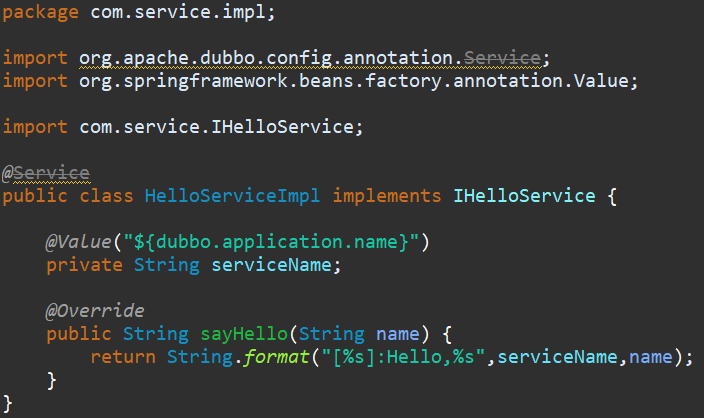
之后通过mvn install安装到本地仓库或远程仓库。

1. 在sample-provider模块中引入依赖，其中包括 dubbo

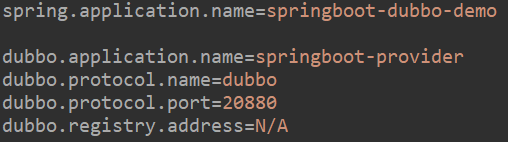


spring-boot-starter是创建Spring Boot时就有的，如果是web项目，则是 spring-b oot-starter-web。dubbo-spring-boot-starter是Apache Dubbo官方提供的开箱即用 组件，sample-api是sample-api模块。

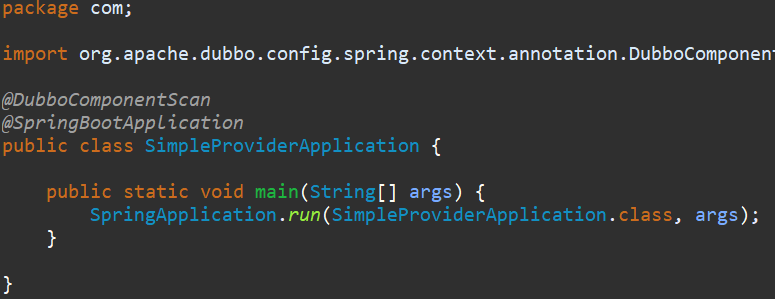
1. 在sample-provider模块中定义HelloServiceImpl实现IHelloService接口，并且使用Dubbo中提供的@Service注解发布服务。



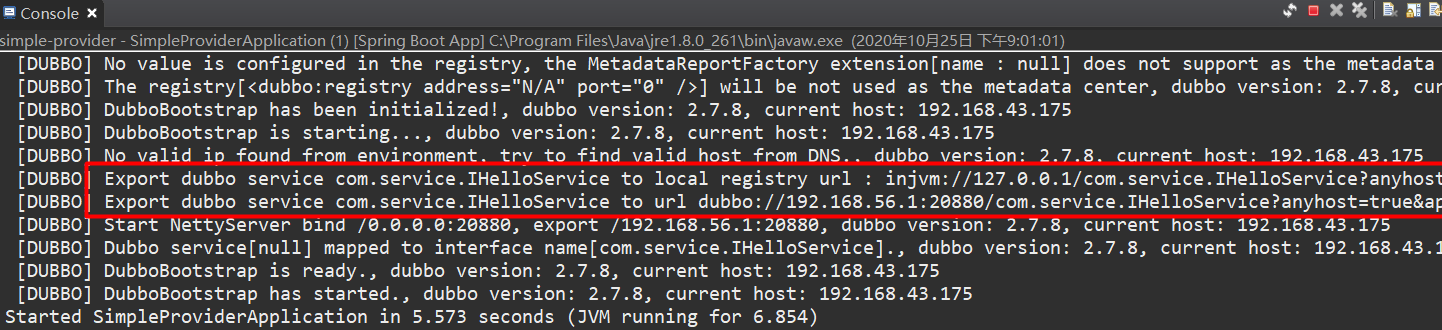
1. 在sample-provider模块中的application.properties文件中添加Dubbo服务的配置信息，配置元素在前面的章节中讲过，只是换了一种配置形式。



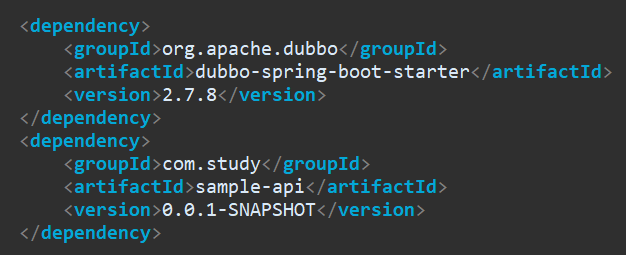
1. 启动Spring Boot，需要注意的是，需要在其启动方法上添加@DubboComponentScan注解，它的作用和Spring Framework提供的@ComponentScan一样，只不过这里扫描的是Dubbo中提供的@Service注解。



运行后，在控制台会打印Dubbo给出的地址：



1. **服务调用者开发流程**
2. 创建一个Spring Boot项目，名为springboot-consumer，添加jar包依赖

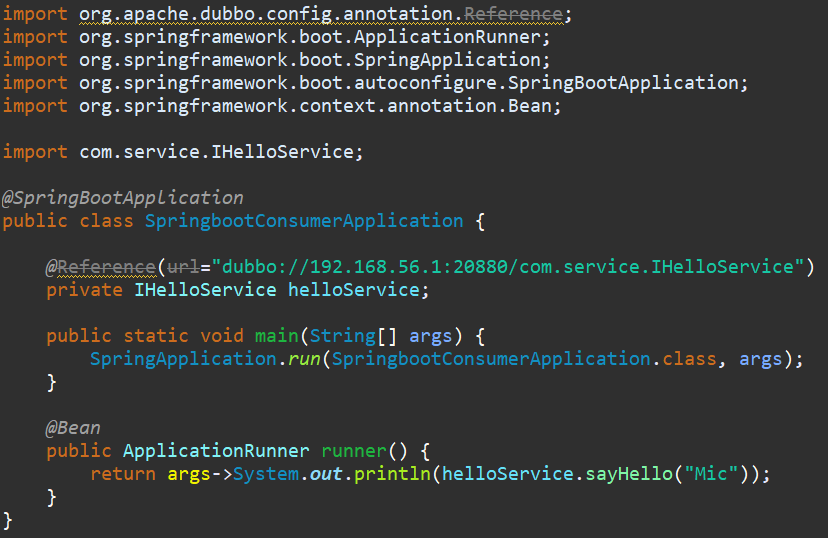


1. 在application.properties中配置Dubbo的项目名称



因为它是必选的。

1. 在Spring Boot启动类中，使用Dubbo提供的@Reference注解来获得一个远程代理对象。



服务提供者项目运行后，运行该项目，观察控制台打印：



调用远程服务成功。

该案例和上一小节中的案例都是使用Dubbo以点对点的形式来实现服务之间的通信。 相比于基于XML的形式来说，基于dubbo-spring-boot-starter组件来使用Dubbo完成服 务发布和服务消费会使得开发更加简单。另外，官方来提供了dubbo-spring-boot-actuator 模块，可以实现针对Dubbo服务健康检查；还可以通过Endpoints实现Dubbo服务信息 的查询和控制等，为生产环境中对Dubbo服务的监控提供了良好支持。

# 快速上手ZooKeeper

## 认识Zookeeper

Zookeeper是一个高性能的分布式协调中间件，所谓的分布式协调中间件的作用类似于多线程环境中通过并发工具来协调线程的访问控制，只是分布式协调中间件主要解决分布式环境中各个服务进程的访问控制问题，比如访问顺序控制。所以，在这里需要强调的是，Zookeeper并不是注册中心，只是基于ZooKeeper本身的特性可以实现注册中心这个场景而已。

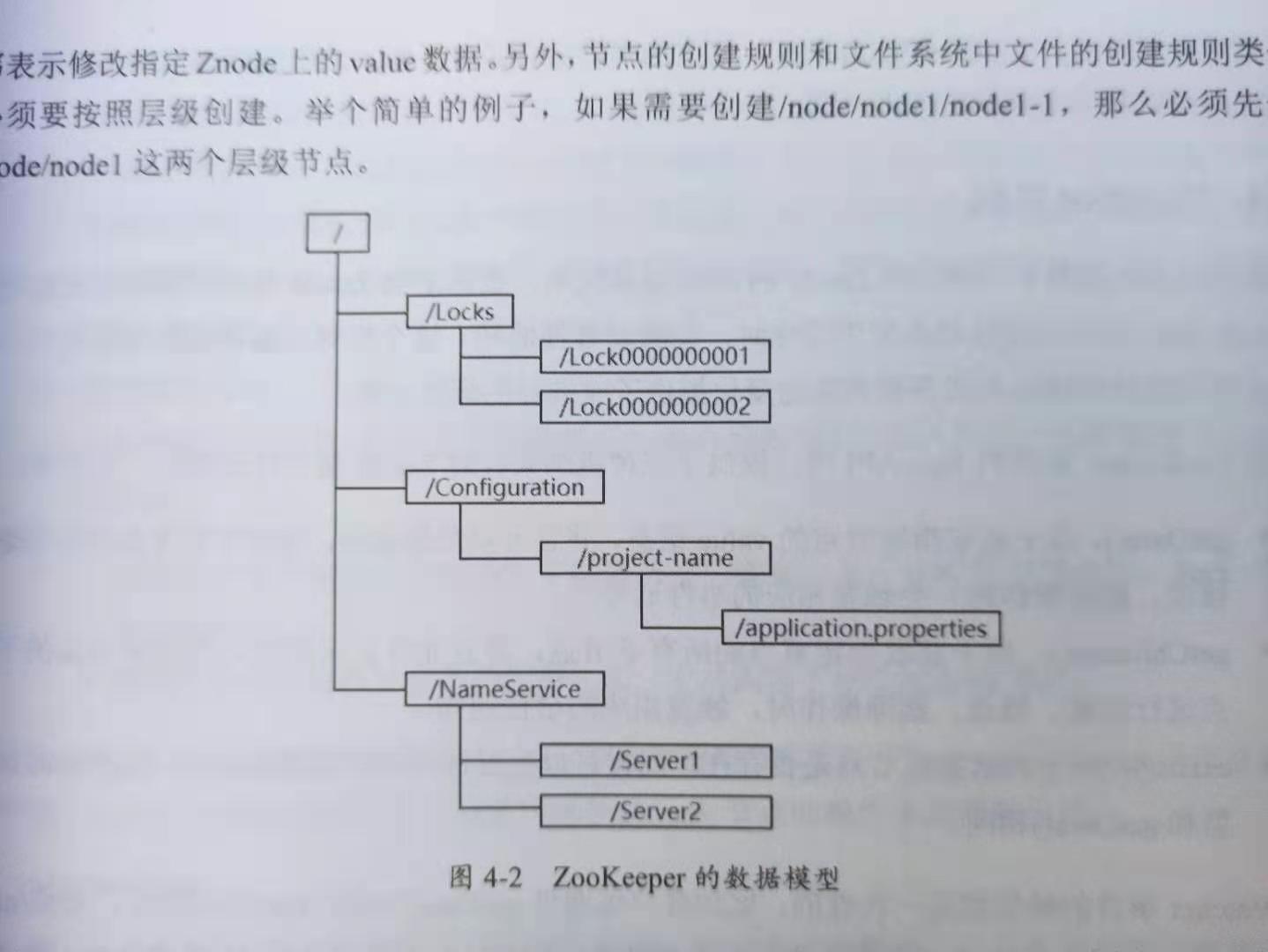
## ZooKeeper的安装

ZooKeeper的安装非常简单，需要注意的是，由于ZooKeeper是使用Java编写的，所以在安装之前必须要安装Java运行环境。另外，ZooKeeper支持单击部署和集群部署，由于本书并不是专门讲解ZooKeeper的，所以只会简单演示单击环境的安装过程，便于完成Dubbo和ZooKeeper的集成。安装步骤如下：

1. 在Apache官网上下载ZooKeeper，作者写作该书时最新版本为3.5.6，而我下载的是3.5.8。
2. 将下载好的安装包解压缩到指定目录，解压缩后可以看到ZooKeeper包含很多目录，其中conf是存放配置文件的目录，bin是ZooKeeper提供的可执行脚本的目录。
3. ${ZooKeeper\_HOME }\conf目录下提供了ZooKeeper核心配置的样例文件zoo\_sample.cfg，如果要将ZooKeeper运行起来，需要将其名称修改为zoo.cfg，内容可以暂时不用修改。
4. 在${ZooKeeper\_HOME}\lib目录下，执行sh zkServer.sh start，启动服务。
5. 启动服务后，就可以通过默认发布的2181端口来访问。如果是在同一台机器上访问，在${ZooKeeper\_HOME}/bin目录下执行**sh zkCli.sh**即可连接到ZooKeeper服务器。如果要连接到不同机器上的ZooKeeper服务，需要增加-server参数，即在${ZooKeeper\_HOME}/bin目录下执行**sh zkCli.sh -server target-server-ip:2181**，其中target-server-ip是IP地址。（注意，虽然在其他机器上连接ZooKeeper服务器不用开放端口号，但是在Apache Dubbo集成ZooKeeper时ZooKeeper服务器是需要开放端口的，所以建议，安装完ZooKeeper后开放端口号）

## ZooKeeper的数据结构

ZooKeeper的数据模型和分布式文件系统类似，是一种层次化的属性结构，如下图所示：



ZooKeeper树中的每个节点被称为Znode，Znode维护一个stat状态信息，其中包含数据变化的时间和版本等。并且每个Znode可以设置一个value值，ZooKeeper并不用于通用的数据库或大容量的对象存储，它只是管理和协调有关的数据，所以value的数据大小不建议设置的非常大，较大的数据会带来更大的网络开销。

ZooKeeper上的每个节点都是允许读和写的，读表示获得指定Znode上的value数据，写表示修改指定Znode上的value数据。另外，节点的创建规则和文件系统中文件的创建规则类似，必须要按照层级创建。举个简单的例子，如果需要创建/node/node/node-1，那么必须先创建/node/node1这两个层级节点。

## ZooKeeper的特性

ZooKeeper中的Znode在被创建的时候，需要指定节点的类型，节点的类型分为：

·持久化节点：节点的数据会持久化到磁盘

·临时节点：节点的生命周期和创建该节点的客户端的生命周期保持一致，一旦该客户

端的会话结束，则该客户端所创建的临时节点就会被自动删除。

·有序节点：在创建的节点中后面会增加一个递增的序列，该序列在同一级父节点之下

是唯一的。持久化节点和临时节点也是可以设置为有序节点的，也就是持久化有序节点或者临时有序节点。

在3.5.3版本之后，又增加了两种节点类型，分别是：

·容器节点：当容器节点下的最后一个子节点被删除时，容器节点就会被自动删除。

·TTL节点：针对持久化节点或者持久化有序节点，我们可以设置一个存活时间，如果

在存活时间之内该节点没有任何修改并且没有任何子节点，它就会被自动删除。

需要注意的是，在同一级目录下，节点的名称必须是唯一的，就像我们在同一个目录中

不能创建两个有相同名字的文件夹一样。

## Watcher机制

1. ZooKeeper提供了一种针对Znode的订阅/通知机制，也就是当Znode节点状态发生变化时或者ZooKeeper客户端连接状态发生变化时，会触发事件通知。这个机制在服务注册与发现中，针对服务调用者及时感知到服务提供者的变化提供了非常好的解决方案。
2. 在ZooKeeper提供的Java API中，提供了三种机制来针对Znode进行注册监听，分别是：
3. getData( )：用于获取指定节点的value信息，并且可以注册监听，当监听的节

点进行创建、修改、删除操作时，会触发相应的事件通知。

1. getChildren( )：用于获取指定节点的所有子节点，并且允许注册监听，当监听节点的子节点进行此、修改、删除操作时，触发相应的事件通知。
2. exists( )：用于判断指定节点是否存在，同样可以注册针对指定节点的监听，监听的时间类型和getData( )相同。
3. Watcher事件的触发都是一次性的，比如客户端通过getData(“/node”,true)注册监听，如果/node节点发生数据修改，那么该客户端会收到一个修改事件通知，但是/node再次发送变化时，客户端无法收到Watcher事件，为了解决这个问题，客户端必须在收到的事件回调中再次注册事件。

## 常用应用场景分析

基于ZooKeeper中节点的特性，可以为多种应用场景提供解决方案。

1. **分布式锁**

用户多现场的读者应该知道锁，比如Synchronized或者Lock，它们主要用于解决多线程环境下共享资源访问的数据安全性，但是它们所处理的范围时线程级别的。在分布式架构中，多个进程对同一共享资源的访问，也存在数据安全性问题，因此也需要使用锁的形式来解决这类问题，而解决分布式环境下多进程对于共享资源访问带来的安全性问题的方案就是使用分布式锁。锁的本质是排他性的，也就是避免在同一时刻多个进程同时访问某一个共享资源。如果使用ZooKeeper实现分布式锁达到排他性，只需要用到节点的特性：临时节点，以及同级节点的唯一性。

**·获得锁的过程**

在获得排他锁时，所有客户端可以去ZooKeeper服务器上/Exclusive\_Locks

节点下创建一个临时节点/lock。ZooKeeper基于同级节点的唯一性，会保证所有客户端中只有一个客户端能创建成功，创建成功的客户端获得了排他锁，没有获得锁的客户端就需要通过Watcher机制监听/Exlusive\_Locks节点下的子节点的变更事件，用于实时监听/lock节点的变化情况以做出反应。

**·释放锁的过程**

在获得锁的过程中，我们定义的锁节点/lock为临时节点，那么在以下两

种情况下会触发锁的释放。

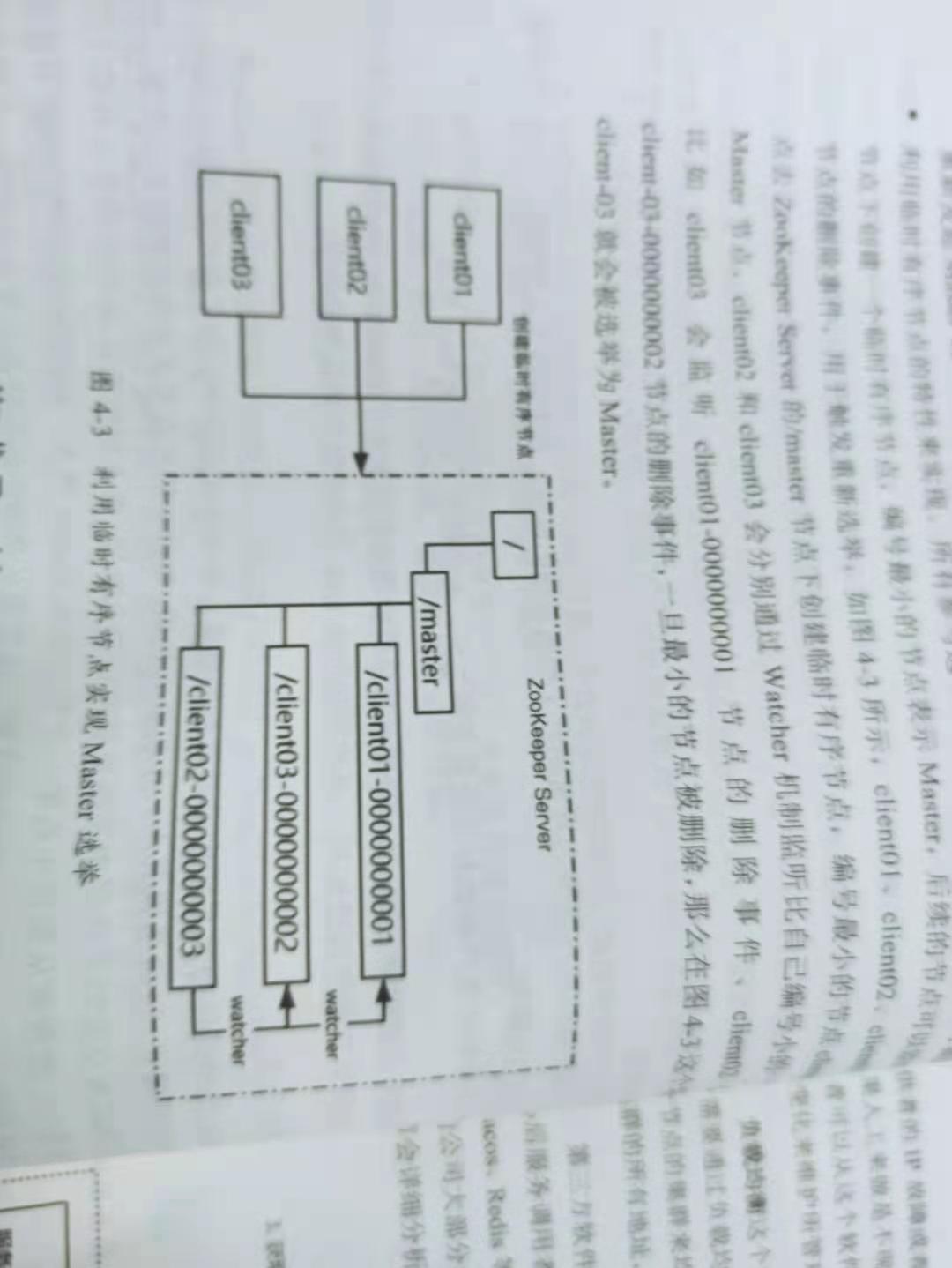
1. 获得锁的客户端因为异常断开了和服务端的连接，基于临时节点的特性，/lock节点会被自动删除。
2. 获得所的客户端执行完业务逻辑之后，主动删除了创建的/lock节点。

当/lock节点被删除之后，Zookeeper服务器会通知所有监听了/Exclusive\_Locks子节点变化的客户端。这些客户端收到通知后，再次发起创建/lock节点的操作来获得排他锁。

1. **Master选举**

Master选举是分布式中非常常见的场景，在分布式架构中，为了保证服务的可用性，通常会采用集群模式，也就是当其中一个机器宕机后，集群中的其他节点会接替故障节点继续工作。在这种场景下，就需要从集群中选举一个节点作为Master节点，剩余的节点都可以作为备份节点随时待命。当原有的Master节点出现故障之后，还需要从集群中的其他备份节点中选举一个节点作为Master节点继续提供服务。

ZooKeeper就可以帮助集群中的节点失效Master选举。具体而言，ZooKeeper有两种方式来实现Master选举这一场景：

1. 同一级节点不能重复创建一个已存在的节点，这个有点类似于分布式锁的实现场景，其实Master选举的场景也是如此。假设集群中有3个节点，需要选举出Master，那么这三个节点同时去ZooKeeper服务器上创建一个临时节点/master-election，由于节点的特殊，只会有一个客户端创建成功，创建成功的客户端所在的机器就成了Master。同时，其他没有创建成功的客户端，针对该节点注册Watcher事件，用于监控当前的Master机器是否存活，一旦发现Master“挂了”，也就是/master-election节点被删除了，那么其他的客户端将会重新发起Master选举操作。
2. 利用临时有序节点的特性来实现。所有参与选举的客户端在ZooKeeper服务器的/master节点下创建一个临时有序节点，编号最小的节点表示Master，后续的节点可以监听前一个节点的删除事件，用于触发重新选举，如下图所示：

client01、cllient02、client03三个节点去ZooKeeper Server的/master节点下创建临时有序节点，编号最小的节点client01表示Master节点，client02和client03会分被通过Watcher机制监听比自己编号小的一个节点，比如client03会监听client01-0000000001节点的删除事件、client02会监听client03-0000000002节点的删除事件，一旦最小的节点被删除，那么在上图所示场景中，client-03就会被选举为Master。

# Apache Dubbo集成ZooKeeper实现服务注册

## 为什么要集成ZooKeeper

大规模服务化之后，在远程RPC通信过程中，会遇到两个比较尖锐的问题。

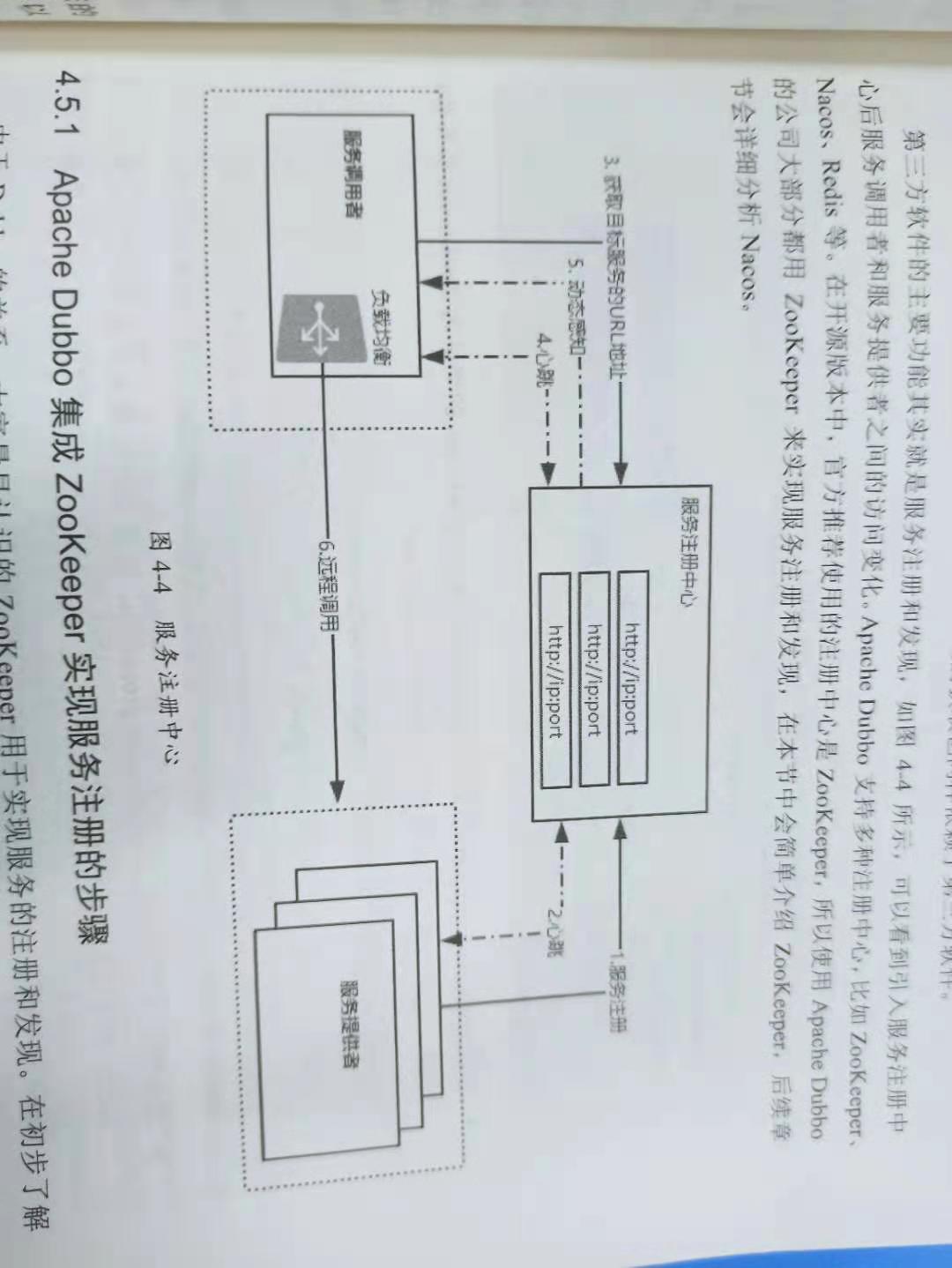
·服务动态上下线感知

·负载均衡

1. **服务动态上下线感知**

服务动态上下线感知，就是服务调用者要感知到服务提供者上下线的变化。按照以往传统形式，服务调用者如果要调用服务提供者，必须要知道服务提供者的地址信息及映射参数。以Webservice为例，服务调用者需要在配置文件中维护一个http://ip:port/service?wsdl地址，但是如果服务提供者是一个集群节点，那么服务调用者需要维护多个这样的地址。问题来了，一旦服务提供者的IP故障或者集群中某个节点下线了，服务调用者需要同步更新这些地址，但是这个操作如果人工来做是不现实的，所以需要一个具有服务注册和发现功能的第三方软件来统一管理服务提供者的URL地址，服务调用者可以从这个软件中获得目标服务的相关地址，并且第三方软件需要动态感知服务提供者状态的变化来维护所管理的URL，进而使得服务调用者能够及时感知到变化而做出相应的处理。

1. **负载均衡**

负载均衡就是当服务提供者是由多个节点组成的集群环境时，服务调用者需要通过负载均衡算法来动态选择一台目标服务器进行远程通信。负载均衡的主要目的是通过多个节点的集群来均衡服务器的访问压力，提升整体性能。实现负载均衡的前提是，要得到目标服务集群的所有地址，在服务调用者端进行计算，而地址的获取也同样依赖于具有服务注册与发现功能的第三方软件。

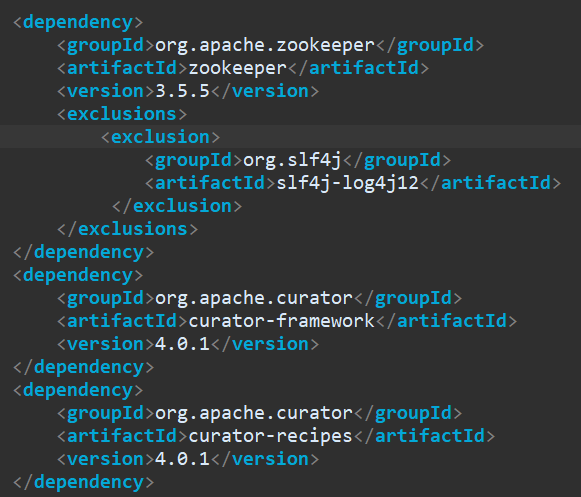
这个第三方软件的主要功能其实就是服务注册与发现，如下图所示：

可以看到引入服务注册中心后服务调用者和服务提供者之间的访问变化。Apache Dubbo支持多种注册中心，比如ZooKeeper、Nacos、Redis等。在开源版本中，官方推荐的注册中心是ZooKeeper，所以使用Apache Dubbo的公司大部分都用ZooKeeper来实现服务注册与发现。（本章会简单介绍ZooKeeper，后续章节会详细介绍Nacos）

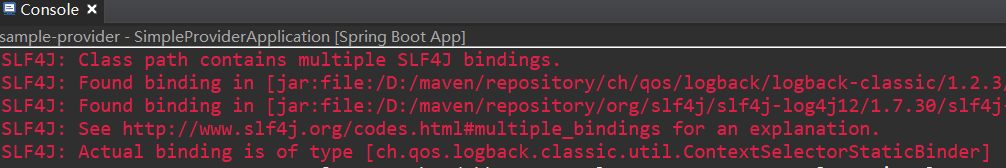
## 集成ZooKeeper实现服务注册

由于Dubbo的关系，最早认识的ZooKeeper用于实现服务的注册和发现。在初步了解ZooKeeper之后，我们就可以将ZooKeeper集成进来实现Dubbo服务的注册和动态感知。还是以前面的Spring Boot集成Apache Dubbo的案例作为演示。

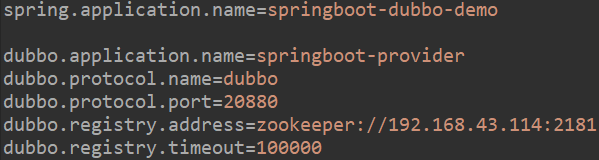
1. 在spring-provider项目的sample-provider模块中添加ZooKeeper相关依赖，如下：



在添加zookeeper依赖时，将slf4j-log4j12排除了，这是因为它会与spring-boot-starter或spring-boot-starter-web依赖中slf4j产生冲突，如果不排除，则在项目启动时有可能会出现以下警告：

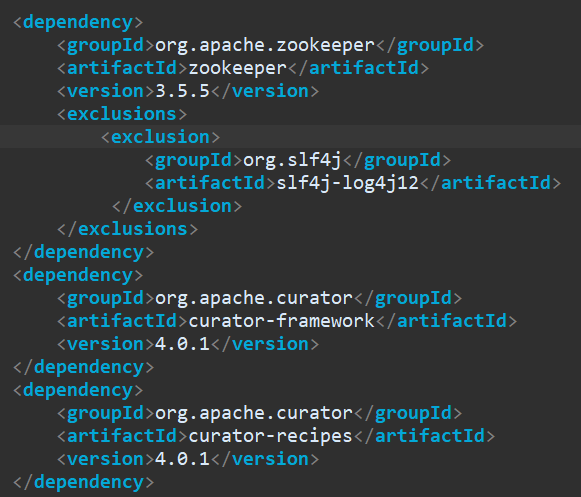


1. 修改application.properties文件，修改dubbo.registry.address的地址为Zookeeper服务器的地址，表示当前Dubbo服务需要注册到ZooKeeper上。

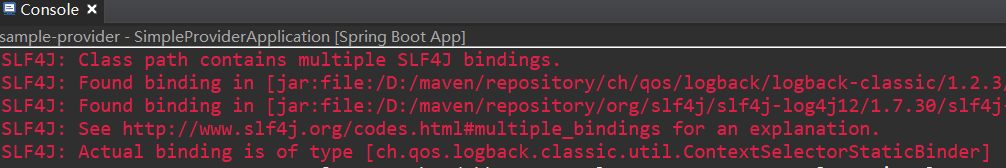


在这里设置了一个新的属性dubbo.registr.timeout，它表示远程连接ZooKeeper服务器的超时时间，单位毫秒，如果不设置，则默认为3000毫秒，而在远程连接ZooKeeper的过程中，3秒可能过短，此时将会抛出异常，导致项目启动失败，所以，可以稍微将值设大一点。

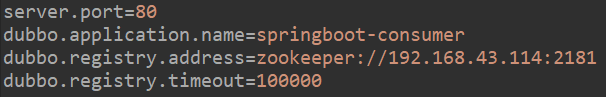
1. 在调用者中添加依赖，如下：



在添加zookeeper依赖时，将slf4j-log4j12排除了，这是因为它会与spring-boot-starter或spring-boot-starter-web依赖中slf4j产生冲突，如果不排除，则在项目启动时有可能会出现以下警告：



1. 调用者只需要修改application.properties，设置Dubbo服务注册中心的地址即可，当Dubbo调用方发起远程调用时，会去注册中心获取目标服务的URL地址已完成最终通信。



在这里设置了一个新的属性dubbo.registr.timeout，它表示远程连接ZooKeeper服务器的超时时间，单位毫秒，如果不设置，则默认为3000毫秒，而在远程连接ZooKeeper的过程中，3秒可能过短，此时将会抛出异常，导致项目启动失败，所以，可以稍微将值设大一点。

## ZooKeeper注册中心的实现原理

Dubbo服务注册到ZooKeeper上之后，可以在ZooKeeper服务器上可以查看到树形结构。

当Dubbo服务启动时，会去ZooKeeper服务器上的/dubbo/ com.service.IHelloService/provides目录下创建当前服务的URL，其中com.service.IHelloService是发布服务的接口全路径名称（包类路径），providers表示服务提供者的类型，dubbo://ip:port表示该服务发布的协议类型及访问地址。其中URL是临时节点，其他持久化节点。在这里使用临时节点的好处在于，如果注册该节点的服务器下线了，那么这个服务器的URL地址就会从ZooKeeper服务器上被移除。

当Dubbo服务器消费者启动时，会对/dubbo/ com.service.IHelloService/providers节点下的子节点注册Watcher监听，这样便可以感知到服务提供方节点的上下线变化，从而防止请求发送到已经下线的服务器造成访问失败。同时，服务消费者会在dubbo/ com.service.IHelloService/comsumers下写入自己的URL，这样做的目的是可以在监控平台上看到某个Dubbo服务正在被哪些服务调用。最重要的是，Dubbo服务的消费者如果需要调用IHelloService服务，那么它会先去/dubbo/com.service.IService/provides路径下获得所有该服务的提供方URL列表，然后通过负载均衡算法计算出一个地址进行远程访问。

整体看来，服务注册和动态感知的功能用到了ZooKeeper中的临时节点、持久化节点、Watcher等，回过头看前面分析的ZooKeeper的应用场景可以发现，几乎所有的场景都是基于这些来完成的。另外，不得不提的是，Dubbo还可以针对不同的情况来实现以下功能。

·基于临时节点的特性，当服务提供者宕机或者下线时，注册中心会自动删除该服

务提供者的信息。

·注册中心重启时，Dubbo能够自动恢复注册数据及订阅要求。

·为了保证节点操作的安全性，ZooKeeper提供了ACL权限控制，在Dubbo中可以

通过dubbo，registry.username/dubbo.registry.password设置节点的验证信息。

·注册中心默认的根节点是/dubbo，如果需要针对不同环境设置不同的根节点，可

以适应dubbo.registry.group修改根节点名称。

# 实战Dubbo Spring Cloud

Spring Cloud为Java环境中解决微服务问题提供了非常完整的方案，所以在最近几年里Spring Cloud成了很多公司首选的技术方案。但是随着运用规模的扩大，Spring Cloud在微服务领域的局限性逐步显露出来。相对来说，在服务治理方面，Apache Dubbo有着非常大的优势，而且在Spring Cloud出现之前，它就已经被很多公司作为服务治理及微服务基础设施的首选框架，Dubbo Spring Cloud的出现，使得Dubbo既能够完全整合到Spring Cloud的技术栈中，享受Spring Cloud生态中的技术支持和标准化输出，又能弥补Spring Cloud中服务治理这方面的短板。

## 什么是Dubbo Spring Cloud

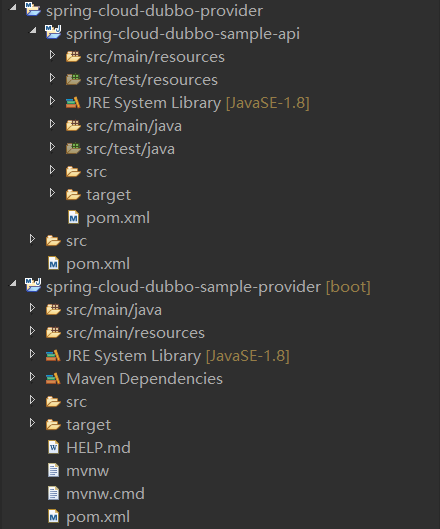
Dubbo Spring Cloud是Spring Cloud Alibaba的核心组件，它构建在原生的Spring Cloud标准之上，不仅覆盖了Spring Cloud原生特性，还提供了更加稳定和成熟的实现。

## 实现Dubbo服务提供方

细心的读者应该会发现，对于服务提供者而言，都会存在一个API声明，因为服务的调用者需要访问服务提供者的声明的接口，为了确保契约的一致性，Dubbo官方推荐的做法是把服务接口打成jar包发布到仓库上。服务调用者可以依赖该jar包，通过接口调用方式完成远程通信。对于服务提供者来说，也需要依赖该jar包完成接口的实现。

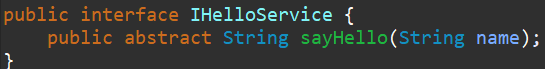
1）创建一个Maven工程，并在该工程创建两个模块：spring-cloud-dubbo-sample-api和

spring-cloud-dubbo-sample-provider。其中spring-cloud-dubbo-sample-api是一个普通的Maven工程，用于提供服务接口，sprng-cloud-dubbo-sample-provider是一个Spring Boot工程。

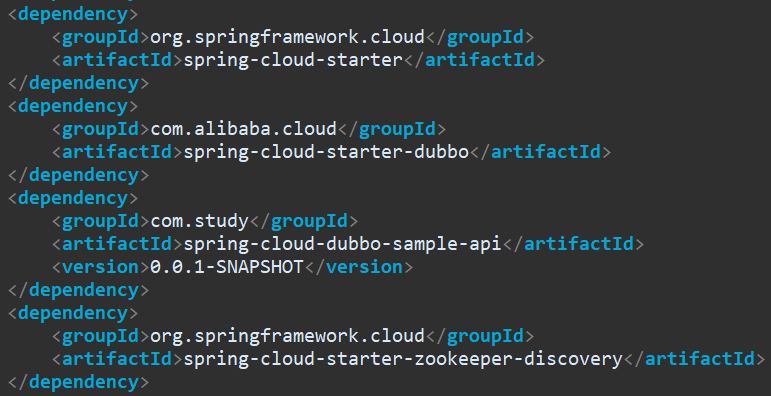


2）在spring-cloud-dubbo-sample-api模块中声明接口，并执行mvn install将该模块打成

jar包安装到仓库，供其他模块和服务消费者依赖。



3）在spring-cloud-dubbo-sample-provider中添加依赖



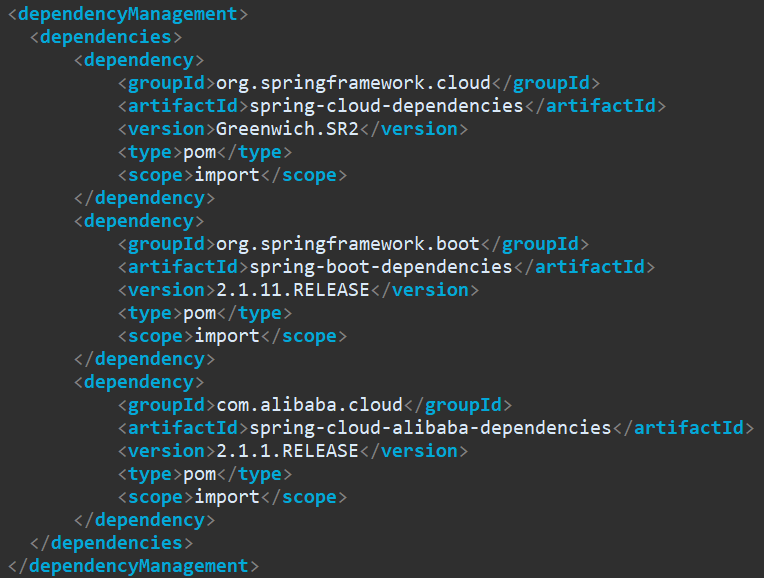
spring-cloud-starter：Spring Cloud核心包

spring-cloud-dubbo-sample-api：API接口模块

spring-cloud-starter-dubbo：引入Spring Cloud Alibaba Dubbo

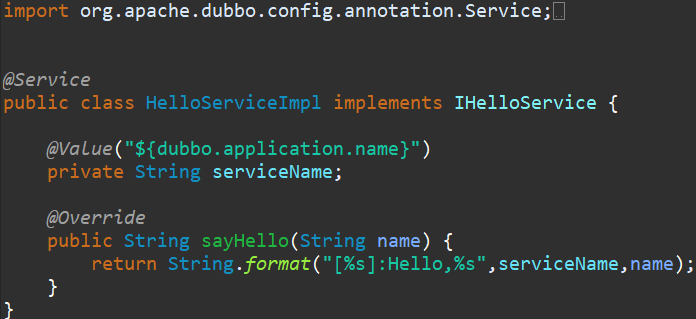
spring-cloud-starter-zookeeper-discovery：基于ZooKeeper实现服务注册与发现的项目。

需要注意的是，上述依赖并没有指定版本，所以需要在父pom.xml中声明版本管理。如下：

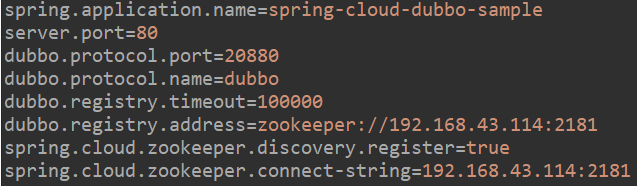


4）在spring-cloud-dubbo-sample-provider中创建接口的实现类HelloServiceImpl，其中

@Service是Dubbo服务的注解，表示当前服务会发布为一个远程服务。



5）在application.properties中配置Dubbo相关信息



spring.application.name：该服务在服务注册中心使用的名称

server.port=80：该SpringBoot项目使用的端口号，因为学习将服务提供者和服务消费者都在本机上运行，所以为了避免端口冲突，设置了端口号。

dubbo.protocol.name=dubbo：配置服务提供者的协议信息，Dubbo默认采用Dubbo协议，除此之外，还支持Webservice等。

dubbo.protocol.port=20880：配置Dubbo端口号，默认为20880端口

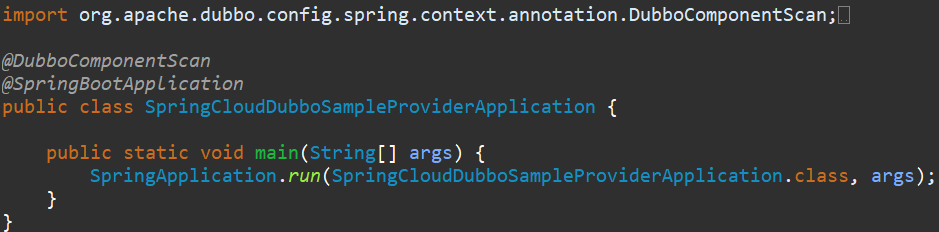
dubbo.registry.address=zookeeper://192.168.43.114:2181：注册中心的地址。

dubbo.registry.timeout=100000：注册超时时间，如果超时则项目报错。

spring.cloud.zookeeper.discovery.register=true：服务是否需要注册到注册中心

spring.cloud.zookeeper.connect-string=192.168.43.114.2181：表示ZooKeeper的连接字符串

1. 在启动类中声明@DubboComponentScan注解，并可以启动服务。



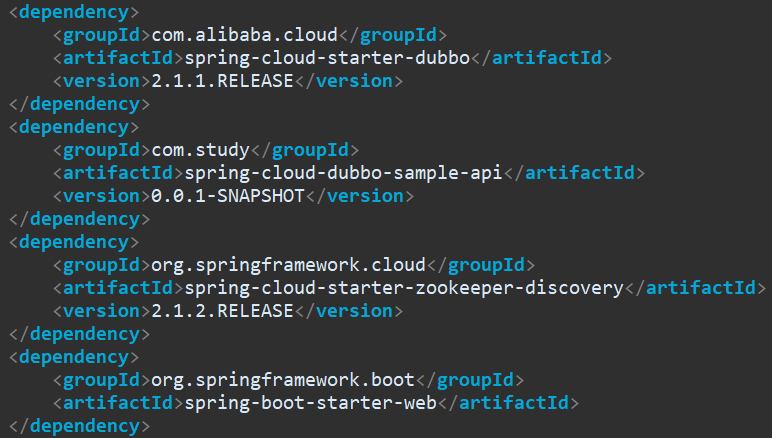
@DubboComponentScan用于扫描当前注解所在的包路径下@org.apache.dubbo.config.spring.context.annotation.DubboComponentScan注解，实现服务的发布。发布完成之后，可以在ZooKeeper服务器上看到一个/service/${project-name}节点，这个节点中保存了服务提供方相关的地址信息。



## 实现Dubbo服务调用方

Dubbo服务提供方spring-cloud-dubbo-sample已经准备完毕，创建一个名为spring-cloud-dubbo-consumer的Spring Boot项目，作为服务调用方。

1. 创建一个名为spring-cloud-dubbo-consumer的Spring Boot工程，添加如下依赖



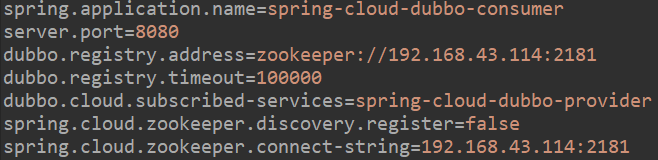
spring-cloud-dubbo-sample-api：服务提供者的API接口

spring-cloud-starter-dubbo：引入Spring Cloud Alibaba Dubbo

spring-cloud-starter-zookeeper-discovery：基于ZooKeeper实现服务注册与发现的项目

spring-cloud-starter-web：为了演示需要，增加了spring-boot-starter-web组件，表示这是一个Web项目

1. 在applicatin.properties文件中配置Dubbo相关配置



spring.application.name：该服务在注册中心的名称。

server.port：指定端口号，因为学习所示是在本机上同时启动服务提供者和服务调

用者，为了避免端口重复，故指定端口号。

dubbo.registry.address：注册中心的地址。

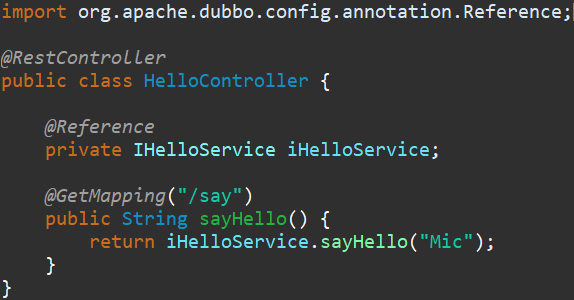
dubbo.registry.timeout：表示在ZooKeeper注册超时时间，单位毫秒，一旦超时，项目启动失败并报错。

dubbo.cloud.subscribe-services：表示服务调用者订阅的服务提供方的应用名称列表，如果有多个应用名称，可以通过“，”分隔，默认值为“\*”，不推荐使用默认值，如果使用，控制台的日志中会输入一段警告信息

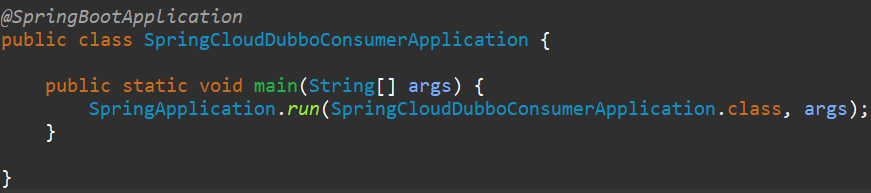
spring.cloud.zookeeper.discovery.register=false：表示当前服务不需要注册到ZooKeeper上，默认值为true。

spring.cloud.zookeeper.onnect-string=192.168.43.114.2181：表示ZooKeeper的连接字符串。

1. 创建HelloController类，暴露一个/say服务，来消费Dubbo服务提供的IHelloService服务。



1. 启动Spring Boot服务



服务启动后，通过curl命令或通过浏览器访问暴露的接口：



响应成功。

# Apache Dubbo的高级应用

前面的学习中，我们只是了解了Apache Dubbo作为RPC通信框架的使用方法，以及服务注册中心的应用及原理，但这仅仅是它的冰山一角，Apache Dubbo更像一个生态，它提供了很多比较主流框架的集成，比如：

1. 支持多种协议的服务发布，默认是dubbo://，还可以支持rest://、webservice://、thrift://等。
2. 支持多种不同的注册中心，如Nacos、ZooKeeper、Redis，未来还将会支持Consul、Eureka、Etcd等。
3. 支持多种序列化技术，如avro、fst、fastjson、hessian2、kryo等。

除此之外，Apache在服务治理方面的功能非常完善，比如集群容错、服务路由、负载均衡、服务降级、服务限流、服务监控、安全验证等。它的常用功能可以关注Apache Dubbo官网，相对于国外资料，它最大的优势是支持中文，所以对读者来说也能够很好理解。

## 集群容错

1. **什么是集群容错**

在分布式架构的网络通信中，容错能力是必须具备的。容错从字面来看，就是服务容忍错误的能力。在网络通信中会存在很多不确定的因素导致请求失败，比如网络延迟、网络中断、服务异常等。当服务调用者（消费者）调用服务提供者的接口时，如果因为这些原因导致请求失败，那对于服务调用者来说，需要一种机制来应付。Dubbo中提供了集群容错的机制来优雅的处理这种错误。

1. **容错模式**

Dubbo默认提供了6种容错模式，默认为Failover cluster。如果这6种容错模式不能满足实际需求，可以自行扩展。这也是Dubbo的强大之一，几乎所有功能都提供了插拔式的扩展。

·Failover Cluster，失败自动切换。当服务调用失败后，会切换到集群中的其他机器重试，默认重试次数为2，通过属性retries=2可以修改次数，但是重试次数增加会带来更多的响应延迟。这种容错模式通常用于读操作，因为事务型操作会带来数据重复问题。

·Failfast Cluster，快速失败。当服务调用失败后，立即报错，也就是只发起一次调用，通常用于一些幂等的写操作，比如新增数据，因为当服务调用失败时，很可能这个请求已经在服务器端处理成功，只是因为网络延迟导致响应失败，为了避免在结果不确定的情况下导致数据重复插入的问题，可以使用这种容错机制。

·Failsafe Cluster：快速安全。也就是出现异常时，直接忽略异常。

·Failback Cluster失败后自动恢复。服务调用出现异常时，在后台记录这条失败的

请求定时重发。这种模式适合用于消息通知操作，保证这个请求一定发送成功。

·Forking Cluster，并行调用集群中的多个服务，只要其中一个成功就返回。可以

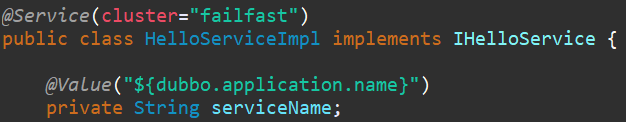
通过forks=2来设置最大并行数。

·Boradcase Cluster，广播调用所有的服务提供者，任意一个服务报错则表示服务

调用失败，这种机制通常用于通知所有的服务提供者更新缓存或者本地资源信息。

**3、配置方式**

只需要在指定服务的@Service注解上增加一个参数即可。在@Service注解增加cluster=”failfast”参数，表示当前服务的容错方式为快速失败。



**4、使用建议**

在实际应用中，查询语句容错策略建议使用默认的Failover Cluster，而增删改操作建议使用Failfast Cluster或者Failover Cluster（retries=”0”）策略，防止出现数据重复添加等其他问题！建议在设计接口的时候把查询接口方法单独做成一个接口提供查询。

## 负载均衡

**1、什么是负载均衡**

在访问量较大的情况下，我们会通过水平扩容的方式增加多个节点来平衡请求的流量，从而提升服务的整体性能。比如一个服务节点的TPS是100，那么如果增加到5个节点的集群，意味着整个集群的TPS可以达到500。当服务调用者面对5个节点组成的服务提供方集群时，请求应该分发到集群中的哪个节点，取决于负载均衡算法，通过该算法可以让每个服务器节点获得适合自己处理能力的负载。负载均衡可以分为硬件负载均衡和软件负载均衡，硬件负载均衡比较常见的是F5，软件负载均衡目前比较主流的是Nginx。

**2、Dubbo的负载均衡策略**

在Dubbo中提供了4种负载均衡策略，默认负载均衡策略时random。同样，如果这4种策略不能满足实际需求，我们可以基于Dubbo中额SPI机制来扩展。

·Random LoadBalance，随机算法。可以针对性能较好的服务器设置比较大的权重

值，权重值越大，随机的概率也会越大。

·RoundRobin LoadBalance，轮询。按照公约后的权重设置轮询比例。

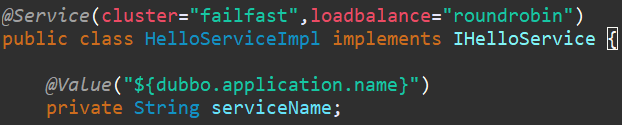
·LeastActive LoadBalance，最少活跃调用书。处理较慢的节点会收到更少的请求。

·ConsiStenHash LoadBalance，一致性Hash。相同参数的请求总是发送到同一个服

务提供者。

**3、配置方式**

在@Service注解上增加loadbalance参数：



## 服务降级

**1、什么是服务降级**

服务降级是一种系统保护策略，当服务器访问压力较大时，可以根据当前业务情况

对不重要的业务进行降级，以保证核心服务的正常运行。所谓的降级，就是把一些非必要的功能在流量较大的时间段暂时关闭，比如在双11大促时，淘宝会把查看历史订单、商品评论功能等关闭，从而释放更多的资源来保障大部分用户能够正常完成交易。

**2、降级分类**

降级有多个层面的分类：

·按照是否自动化可分为自动降级和人工降级。

·按照功能可分为读服务降级和写服务降级。

人工降级一般具有一定的前置性，比如在电商大促之前，暂时关闭某些非核心服务，如评价、推荐等。而自动降级更多的来自于系统出现某些异常的时候自动触发“兜底的流畅”，比如：

·故障降级，调用的远程服务“挂了”，网络故障或者RPC服务返回异常。

这类情况在业务允许的情况下可以通过设置兜底数据响应给客户端。

·限流降级，不管是什么样的系统，它所支撑的流量是有限的，为了保护

系统不被压垮，在系统中会针对核心业务进行限流。当请求流量达到阈值时，

后续的请求会被拦截，这类请求可以进入排队系统，比如12306。也可以直

接返回降级页面，比如返回“活动太火爆，请稍后再来”页面。

**3、Dubbo实现服务降级**

Dubbo提供一种Mock配置来实现服务降级，也就是说当服务提供方出现网络异常

无法访问时，客户端不抛出异常，而是通过降级配置返回兜底数据。操作步骤如下：

1）在spring-cloud-dubbo-consumer项目中创建MockHelloService类，这个只需要

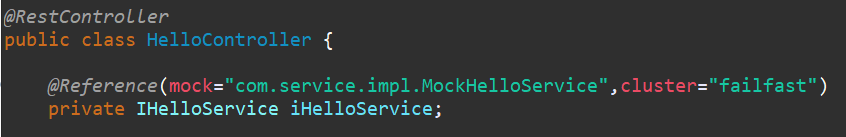
实现自动降级的接口即可，然后重写接口中的抽象方法实现本地数据的返回。



IHelloService为服务提供者提供的接口。

2）在HelloController类中修改@Reference注解增加Mock参数。其中设置了属性

cluster=”failfast”，因为默认的容错策略会发起两次重试，等待的时间较长。



3）在不启动Dubbo服务端（服务提供者）或者服务端的返回值超过默认的超时时

间时，访问/say接口得到的结构就是MockHelloService中返回的数据。

## 主机绑定规则

**1、认识主机绑定**

主机绑定表示的是Dubbo服务对外发布的IP地址，默认情况下Dubbo会按照以下顺序来查找并绑定主机的IP地址。

·查找环境变量中的DUBBO\_IP\_TO\_BIND属性配置的IP地址

·查找dubbo.protocol.host属性配置的IP地址，默认是空，如果没有配置或者IP

地址不合法，则继续向下查找。

·通过LocalHost.getHostAddress获取本机IP地址，如果获取失败，则继续往下查

找。

·如果配置了注册中心的地址，则使用Socket通信连接到注册中心的地址后，使

用for循环通过socket.getLocalAddress( ).getHostAddress( )扫描各个网卡获取网卡

的IP地址。

上述过程中，任意一个步骤检测到合法的IP地址，便会将其返回作为对外暴露的服务IP地址。需要注意的是，获取的IP地址并不是写入注册中心的地址，默认情况下，写入注册中心的地址优先选择环境变量中DUBBO\_IP\_TO\_REGISTRY属性配置的IP地址。在这个属性没有配置的情况下，才会选取前面获得的IP地址并写入注册中心。

**2、后果与解决**

使用默认的主机绑定规则，可能会存在获取的IP地址不正确的情况，导致服务消费者与注册中心大道的URL地址进行通信。因为Dubbo检测本地IP地址的策略是先调用LocalHost.getHostAddress，这个方法的原理是通过获取本机的hostname映射IP地址，如果它指向的是一个错误的IP地址，那么这个错误的地址会作为服务发布的地址注册到ZooKeeper节点上，虽然Dubbo服务能够正常启动，但是服务消费者却无法正常调用。按照Dubbo中IP地址的查找规则，如果遇到这种情况，可以使用很多种方式来解决。

·在/etc/hosts中配置及其名对应正确的IP地址映射

·在环境变量中添加DUBBO\_IP\_TO\_BIND或者DUBBO\_IP\_TO\_REGISTRY属性，

value值为绑定的主机地址。

·通过dubbo.protocol.host设置主机地址。

除获取绑定主机IP地址外，对外发布的端口也是需要注意的，Dubbo框架中针对不同的协议都提供了默认的端口号：

·Dubbo协议的默认端口号为20880

·Webservice协议的默认端口号为80

在实际使用过程中，建议指定一个端口号，避免和其他Dubbo服务的端口产生冲

突。