# 微服务专题-一站式微服务架构SpringCloud

主讲：Cat 老师

**北京动力节点教育科技有限公司**  
**2019 • 北京**

**动力节点版权所有，禁止私自传播**

# 一点理论概念的梳理（也很重要）

在系统架构与设计的实践中，从宏观上可以总结为三个阶段；

**集中式架构**：就是把所有的功能、模块都集中到一个项目中，部署在一台服务器上，从而对外提供服务（单体架构、单体服务、单体应用）；

直白一点：就是只有一个项目，只有一个war；

**分布式架构：**就是把所有的功能、模块拆分成不同的子项目，部署在多台不同的服务器上，这些子项目相互协作共同对外提供服务。

直白一点：就是有很多项目，有很多war包，这些项目相互协作完成需要的功能，不是一个war能完成的，一个war包完成不了；

比如：

Shop项目：单体应用

Shop项目：拆分--> (user-center, order-center, trade-center) 分布式应用

**微服务架构：**分布式强调系统的拆分，微服务也是强调系统的拆分，微服务架构属于分布式架构的范畴；

并且到目前为止，微服务并没有一个统一的标准的定义，那么微服务究竟是什

么？

微服务一词源于 Martin Fowler（马丁.福勒）的名为 Microservices 的博文，

可以在他的官方博客上找到这篇文章：

<http://martinfowler.com/articles/microservices.html>

中文翻译版本：

<https://www.martinfowler.cn/articles/microservices.html>

简单地说， 微服务是系统架构上的一种设计风格， 它的主旨是将一个原本独立

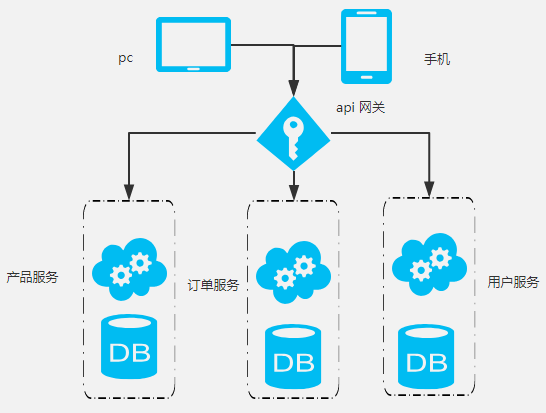
的系统拆分成多个小型服务，这些小型服务都在各自独立的进程中运行，服务之

间通过基于 HTTP 的 RESTful API 进行通信协作； （dubbo -->dubbo协议 ）

RESTful API (controller --> 调用 congtroller)

被拆分后的每一个小型服务都**专注**于完成系统中的**某一项业务功能**，**职责单一，** 并且每个服务都是一个独立的项目，可以进行独立的测试、开发和部署等；

由于各个独立的服务之间使用的是基于 HTTP 的 JSON 作为数据通信协作的基础，所以这些微服务也可以使用不同的语言来开发；



比如：项目里面有User模块和Order模块，但是User模块和Order模块并没有直接关系，仅仅只是一些数据需要交互，那么就可以把这2个模块单独分开来，当user需要调用order的时候，order是一个服务方，但是order需要调用user的时候，user又是服务方了， 所以，它们并不在乎谁是服务方谁是调用方，他们都是2个独立的服务，这就是微服务的概念；

## 经典面试：分布式和微服务有什么区别？

分布式，就是将巨大的一个系统划分为多个模块，这一点和微服务是一样的，都是要把系统进行拆分，部署到不同机器上，因为一台机器可能承受不了这么大的访问压力，或者说要支撑这么大的访问压力需要采购一台性能超级好的服务器，其财务成本非常高，有这些预算完全可以采购很多台普通的服务器了，分布式系统各个模块通过接口进行数据交互，其实分布式也是一种微服务，因为都是把模块拆分变为独立的单元，提供接口来调用，那么它们本质的区别是什么？

它们的本质的区别体现在“目标”上， 何为目标，就是你采用分布式架构或者采用微服务架构，你最终是为了什么，要达到什么目的？

分布式架构的目标是什么？ 就是访问量很大一台机器承受不了，或者是成本问题，不得不使用多台机器来完成服务的部署；

而微服务的目标是什么？只是让各个模块拆分开来，不会被互相影响，比如模块的升级或者出现BUG或者是重构等等都不要影响到其他模块，微服务它是可以在一台机器上部署；

但是：分布式也是微服务的一种，微服务也属于分布式；

面试：微服务与Spring-Cloud的关系或区别？

微服务只是一种项目的架构方式、架构理念，或者说是一种概念，就如同我们的MVC架构一样， 那么Spring Cloud便是对这种架构方式的技术落地实现；

面试：微服务一定要使用Spring Cloud吗？

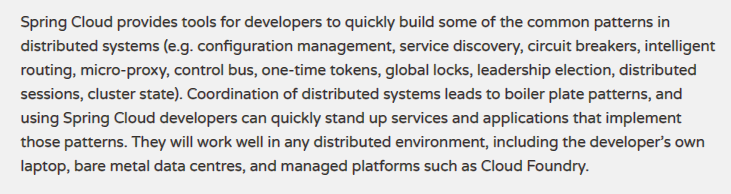
微服务只是一种项目的架构方式、架构理念，所以任何技术都可以实现这种架构理念，只是微服务架构里面有很多问题需要我们去解决，比如：负载均衡，服务的注册与发现，服务调用，服务路由，服务熔断等等一系列问题，如果你自己从0开始实现微服务的架构理念，那头发都掉光了，所以Spring Cloud 帮我们做了这些事情，Spring Cloud将处理这些问题的的技术全部打包好了，我们只需要开箱即用；

# What is Spring Cloud？

官网：<https://spring.io/projects/spring-cloud>

版本：Greenwich SR3

出自官方：



Spring Cloud为开发人员提供了一些工具用来快速构建分布式系统中的一些常见模式和解决一些常见问题（例如配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性令牌、全局锁、领导选举、分布式会话、群集状态）。分布式系统的协调导致了很多样板式的代码（很多固定套路的代码），使用Spring Cloud开发人员可以快速建立实现这些模式的服务和应用程序。它们在任何分布式环境中都能很好地运行，包括开发人员自己的笔记本电脑、裸机数据中心和云计算等托管平台；

## Spring Cloud特性

Spring Cloud为分布式系统开发的典型应用场景提供良好的开箱即用的功能，比如：

分布式/版本化配置

服务注册和发现

路由

服务与服务间的调用

负载均衡

断路器

全局锁

领导选举与集群状态

分布式消息传递

## Spring Cloud下的主要项目

Spring Cloud Config

Spring Cloud Netflix

Spring Cloud Bus

Spring Cloud Cloudfoundry

Spring Cloud Open Service Broker

Spring Cloud Cluster

Spring Cloud Consul

Spring Cloud Security

Spring Cloud Sleuth

Spring Cloud Data Flow

Spring Cloud Stream

Spring Cloud Stream App Starters

Spring Cloud Task

Spring Cloud Task App Starters

Spring Cloud Zookeeper

Spring Cloud AWS

Spring Cloud Connectors

Spring Cloud Starters

Spring Cloud CLI

Spring Cloud Contract

Spring Cloud Gateway

Spring Cloud OpenFeign

Spring Cloud Pipelines

Spring Cloud Function

## Spring Cloud的版本

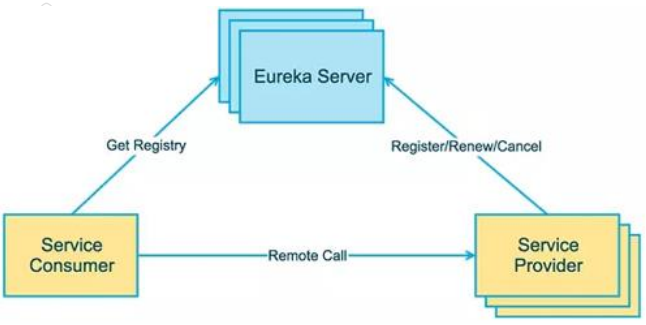
Spring Cloud是由一系列独立项目组成的，每个独立的项目具有不同的发布节奏，每次Spring Cloud发布版本时，就会组合这一系列的子项目，Spring Cloud为了避免大家对版本号的误解，避免与子项目版本号混淆，所以Spring Cloud发布的版本是一个按照字母顺序的伦敦地铁站的名字（“天使”是第一个版本，“布里克斯顿”是第二个），字母顺序是从A-Z，目前最新稳定版本**Greenwich SR3**，当Spring Cloud里面的某些子项目出现关键性bug或重大更新，则发布序列将推出名称以“.SRX”结尾的版本，其中“X”是一个数字，比如：Greenwich SR1、Greenwich SR2、Greenwich SR3；

Spring Cloud是微服务开发的一整套解决方案，采用Spring Cloud开发，每个项目依然是使用Spring Boot；

Spring Cloud 与 Spring Boot的兼容版本

|  |  |
| --- | --- |
| **Spring Cloud版本** | **Spring Boot版本** |
| Hoxton | 2.2.x |
| Greenwich | 2.1.x |
| Finchley | 2.0.x |
| Edgware | 1.5.x |
| Dalston | 1.5.x |
| Camden | 1.4.x 或 1.5.x |
| Brixton | 1.3.x 或 1.4.x |
| Angel | 1.2.x |

# Spring Cloud 的整体架构



Service Provider： 暴露服务的服务提供方。

Service Consumer：调用远程服务的服务消费方。

EureKa Server： 服务注册中心和服务发现中心。

# 3、服务消费方直接调用服务提供方

我们知道，SpringCloud 构建微服务是基于 SpringBoot 开发的。

1、创建一个 SpringBoot 工程，并且添加 SpringBoot 的相关依赖；

2、创建服务提供者的访问方法，也就是后续消费者如何访问提供者；

Spring Cloud 是基于 rest 的访问，所以我们添加一个 Controller，在该

Controller 中提供一个访问入口：

@RestController  
**public class** GoodsController {  
  
 @Autowired  
 **private** GoodsService **goodsService**;  
 */\*\*  
 \* 查询所有商品  
 \*  
 \** ***@param model*** *\** ***@return*** *\*/* @GetMapping(**"/service/goods"**)  
 **public** ResultObject goods(Model model) {  
 List<Goods> goodsList = **goodsService**.getAllGoods();  
 **return new** ResultObject(Constant.***ZERO***, **"查询成功"**, goodsList);  
 }  
}

3、启动运行该 SpringBoot 程序，访问该 controller；

服务消费者也是一个 SpringBoot 项目，服务消费者主要用来消费服务提供者提 供的服务；

1、创建一个 SpringBoot 工程，并且添加 SpringBoot 的相关依赖；

2、开发一个消费者方法，去消费服务提供者提供的服务，这个消费者方法也是

一个 Controller:

@RequestMapping(**"/cloud/goods"**)  
**public** @ResponseBody Object goods(Model model) {  
 ResponseEntity<ResultObject> responseEntity = **restTemplate**.getForEntity(***GOODS\_URL\_01***, ResultObject.**class**);  
 **int** statusCodeValue = responseEntity.getStatusCodeValue();  
 HttpStatus httpStatus = responseEntity.getStatusCode();  
 HttpHeaders httpHeaders = responseEntity.getHeaders();  
 ResultObject body = responseEntity.getBody();  
  
 System.***out***.println(statusCodeValue);  
 System.***out***.println(httpStatus);  
 System.***out***.println(httpHeaders);  
 System.***out***.println(body);  
  
 model.addAttribute(**"goodsList"**, body.getData());  
 **return** body;  
}

3、启动该 SpringBoot 程序，测试服务消费者调用服务提供者；

# 4、服务注册与发现

前面的例子，我们看到了，是通过手动指定每个服务来实现调用的，这是相当低效的，当服务接口增多，这种手动指定接口地址的方式变得非常难以维护，Spring Cloud 提供了多种服务注册与发现的实现方式，例如：Eureka、Consul、Zookeeper。

Spring Cloud 支持得最好的是 Eureka，其次是 Consul，再次是 Zookeeper。

**什么是服务注册？**

服务注册：将服务所在主机、端口、版本号、通信协议等信息登记到注册中心上；

**什么是服务发现？**

服务发现：服务消费者向注册中心请求已经登记的服务列表，然后得到某个服务

的主机、端口、版本号、通信协议等信息，从而实现对具体服务的调用；

**Eureka是什么？**

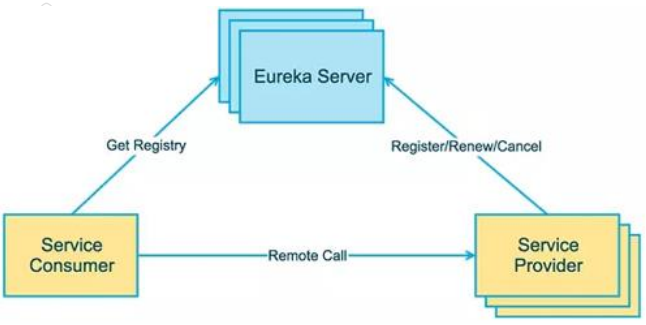
Eureka是Netflix的子模块之一，也是一个核心的模块，Eureka 采用了 C-S（客户端/服务端）的设计架构，也就是 Eureka 由两个组件组成：Eureka 服务端和 Eureka 客户端。

Eureka Server(一个独立的项目) 用于注册服务以及实现服务的负载平衡和故障转移，它是服务的注册中心，Eureka Client（我们的微服务） 它是用于与Eureka Server交互，获取其上注册的服务，使得交互变得非常简单，只需要通过服务标识符即可拿到服务。

### **与spring-cloud的关系：**

Eureka 是 Netflix 公司开发的（一家做版权视频和云服务的公司），Spring Cloud 封装了 Netflix 公司开发的Eureka 模块来实现服务注册和发现，也就是说 Spring Cloud 对 Netflix Eureka 做了二次封装；

**角色关系图：**



**搭建与配置 Eureka 服务注册中心**

Spring Cloud 要使用 Eureka 注册中心非常简单和方便，Spring Cloud 中的

Eureka 服务注册中心实际上也是一个 Spring Boot 工程，我们只需通过引入相关依赖和注解配置就能让 Spring Boot 构建的微服务应用轻松地与 Eureka 进行整合。

具体步骤如下：

1、创建一个 SpringBoot 项目，并且添加 SpringBoot 的相关依赖；

34-springcloud-service-eureka

2、添加 eureka 的依赖：

*<!--Spring Cloud 的 eureka-server 起步依赖-->*

<**dependency**>

<**groupId**>org.springframework.cloud</**groupId**>

<**artifactId**>spring-cloud-starter-netflix-eureka-server</**artifactId**>

</**dependency**>

3、在 Spring Boot 的入口类上添加一个@EnableEurekaServer 注解，用于

开启 Eureka 注册中心服务端

4、在 application.properties 文件中配置 Eureka 服务注册中心信息：

*#内嵌定时tomcat的端口***server.port**=**8761***#设置该服务注册中心的hostname***eureka.instance.hostname**=**localhost***#由于我们目前创建的应用是一个服务注册中心，而不是普通的应用，默认情况下，这个应用会向注册中心（也是它自己）注册它自己，设置为false表示禁止这种自己向自己注册的默认行为***eureka.client.register-with-eureka**=**false***#表示不去从服务端检索其他服务信息，因为自己就是服务端，服务注册中心本身的职责就是维护服务实例，它不需要去检索其他服务***eureka.client.fetch-registry**=**false***#指定服务注册中心的位置***eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka**

**启动与测试 Eureka 服务注册中心**

1、完成上面的项目搭建后，我们就可以启动 SpringBoot 程序，main 方法运行；

2、启动成功之后，通过在浏览器地址栏访问我们的注册中心；

**向 Eureka 服务注册中心注册服务**

我们前面搭建了服务提供者项目，接下来我们就可以将该服务提供者注册到

Eureke 注册中心，步骤如下：

1、在该服务提供者中添加 eureka 的依赖，因为服务提供者向注册中心注册服

务，需要连接 eureka，所以需要 eureka 客户端的支持；

*<!--spring-cloud-starter-netflix-eureka-client-->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
</**dependency**>

2、激活 Eureka 中的 EnableEurekaClient 功能：

在 Spring Boot 的入口函数处，通过添加@EnableEurekaClient 注解来表明自己是一个 eureka 客户端，让我的服务提供者可以连接 eureka 注册中心；

3、配置服务名称和注册中心地址

*#每间隔2s，向服务端发送一次心跳，证明自己依然"存活"***eureka.instance.lease-renewal-interval-in-seconds**=**2***#告诉服务端，如果我10s之内没有给你发心跳，就代表我故障了，将我踢出掉***eureka.instance.lease-expiration-duration-in-seconds**=**10***#告诉服务端，服务实例以IP作为链接，而不是取机器名***eureka.instance.prefer-ip-address**=**true***#告诉服务端，服务实例的名字***eureka.instance.instance-id**=**34-springcloud-service-goods***#eureka注册中心的连接地址***eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka**

4、启动服务提供者 SpringBoot 程序的 main 方法运行；

5、启动运行之后，通过在浏览器地址栏访问我们之前搭建好的 eureka 注册中

心，就可以看到有一个服务已经注册成功了；

**从 Eureka 服务注册中心发现与消费服务**

我们已经搭建一个服务注册中心，同时也向这个服务注册中心注册了服务，接下

来我们就可以发现和消费服务了，这其中服务的发现由 eureka 客户端实现，而服务的消费由 Ribbon 实现，也就是说服务的调用需要 eureka 客户端和 Ribbon，两者配合起来才能实现；

Eureka 客户端是一个 Java 客户端，用来连接 Eureka 服务端，与服务端进行交互、负载均衡，服务的故障切换等；

Ribbon 是一个基于 HTTP 和 TCP 的客户端负载均衡器，当使用 Ribbon 对服务进行访问的时候，它会扩展 Eureka 客户端的服务发现功能，实现从Eureka

注册中心中获取服务端列表，并通过 Eureka 客户端来确定服务端是否己经启动。

Ribbon 在 Eureka 客户端服务发现的基础上，实现了对服务实例的选择策略，

从而实现对服务的负载均衡消费。

接下来我们来让服务消费者去消费服务：

我们前面搭建了服务消费者项目，接下来我们就可以使用该服务消费者通过注册

中心去调用服务提供者，步骤如下：

1、在该消费者项目中添加 eureka 的依赖，因为服务消费者从注册中心获取服

务，需要连接 eureka，所以需要 eureka 客户端的支持；

*<!--spring-cloud-starter-netflix-eureka-client-->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
</**dependency**>

2、激活 Eureka 中的 EnableEurekaClient 功能：

在 Spring Boot 的入口函数处，通过添加@EnableEurekaClient 注解来表明自己是一个 eureka 客户端，让我的服务消费者可以使用 eureka 注册中心；

3、配置服务的名称和注册中心的地址：

spring.application.name=34-springcloud-service-portal

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

4、前面我介绍了服务的发现由 eureka 客户端实现，而服务的真正调用由 ribbon实现，所以我们需要在调用服务提供者时使用 ribbon 来调用：

@LoadBalanced*//使用Ribbon实现负载均衡的调用*@Bean  
**public** RestTemplate restTemplate() {  
 **return new** RestTemplate();  
}

加入了 ribbon 的支持，那么在调用时，即可改为使用服务名称来访问：

**restTemplate**.getForEntity(**"http://34-SPRINGCLOUD-SERVICE-GOODS/service/goods"**,

String.**class**).getBody();

5、完成上面的步骤后，我们就可以启动消费者的 SpringBoot 程序，main 方法运行；

6、启动成功之后，通过在浏览器地址栏访问我们的消费者，看是否可以正常调

用远程服务提供者提供的服务；

**Eureka 与 Zookeeper 的比较**

著名的 CAP 理论指出，一个分布式系统不可能同时满足 C(一致性)、A(可用性) 和 P(分区容错性)；

由于分区容错性在是分布式系统中必须要保证的，因此我们只能在 A 和 C 之间进行权衡，在此 Zookeeper 保证的是 CP, 而 Eureka 则是 AP。

p 全称：Partition tolerance （分区容忍）

主要是指网络问题， 比如：A 、B、C 三台机器之间相互ping不通、网络不通，这种情况在分布式系统里面是允许的，也是很有可能发生的，我们要容忍这种情况的出现，在这种情况出现的时候，我们 是选择 “一致性的C” 还是选择 “可用性的A”，就看应用场景。

**Zookeeper 保证 CP**

在 ZooKeeper 中，当 master 节点因为网络故障与其他节点失去联系时，剩余节点会重新进行 leader 选举，但是问题在于，选举 leader 需要一定时间, 且选举期间整个 ZooKeeper 集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得 ZooKeeper 集群失去 master 节点是大概率事件，虽然服务最终能够恢复，但是在选举时间内导致服务注册长期不可用是难以容忍的。

**Eureka 保证 AP**

Eureka 优先保证可用性，Eureka 各个节点是平等的，某几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而 Eureka 的客户端在向某个 Eureka 注册或时如果发现连接失败，则会自动切换至其它节点，只要有一台 Eureka 还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。

所以 Eureka 在网络故障导致部分节点失去联系的情况下，只要有一个节点可用， 那么注册和查询服务就可以正常使用，而不会像 zookeeper 那样使整个注册服务瘫痪，Eureka 优先保证了可用性；

# Eureka 注册中心高可用集群

在微服务架构的这种分布式系统中，我们要充分考虑各个微服务组件的高可用性

问题，不能有单点故障，由于注册中心 eureka 本身也是一个服务，如果它只有一个节点，那么它有可能发生故障，这样我们就不能注册与查询服务了，所以我们需要一个高可用的服务注册中心，这就需要通过注册中心集群来解决。

eureka 服务注册中心它本身也是一个服务，它也可以看做是一个提供者，又可

以看做是一个消费者，我们之前通过配置：

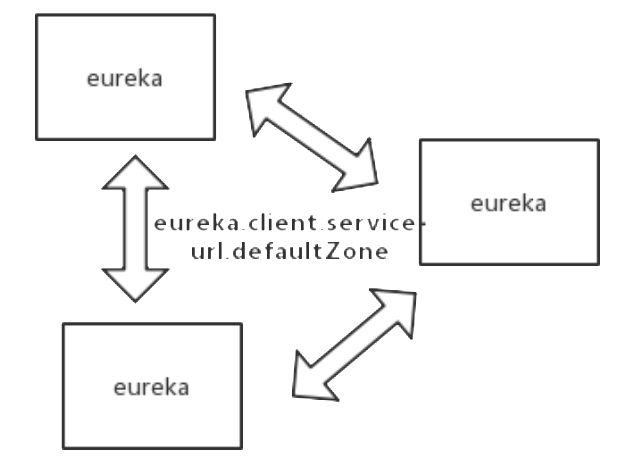
eureka.client.register-with-eureka=false 让注册中心不注册自己，但是我们

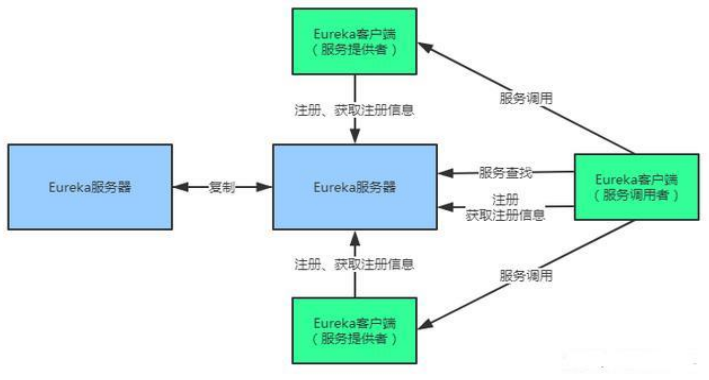
可以向其他注册中心注册自己；

Eureka Server 的高可用实际上就是将自己作为服务向其他服务注册中心注册

自己，这样就会形成一组互相注册的服务注册中心，进而实现服务清单的互相同

步，往注册中心 A 上注册的服务，可以被复制同步到注册中心 B 上，所以从任何一台注册中心上都能查询到已经注册的服务，从而达到高可用的效果。





**Eureka 注册中心高可用集群搭建**

我们知道，Eureka 注册中心高可用集群就是各个注册中心相互注册，所以：

在 8761 的配置文件中，让它的 service-url 指向 8762和8763，在 8762 的配置文件中让它的 service-url 指向 8761和8763， 在 8763 的配置文件中让它的 service-url 指向 8761和8762；

由于两两互相指向对方，实际上我们构建了一个三节点的服务注册中心集群

**eureka.client.service-url.defaultZone**=**http://eureka8762:8762/eureka/,http://eureka8763:8763/eureka/**

**eureka.client.service-url.defaultZone**=**http://eureka8761:8761/eureka/,http://eureka8763:8763/eureka/**

**eureka.client.service-url.defaultZone**=**http://eureka8761:8761/eureka/,http://eureka8762:8762/eureka/**

然后在本地 hosts 文件配置：C:\Windows\System32\drivers\etc\hosts

**127.0.0.1 eureka8761**

**127.0.0.1 eureka8762**

**127.0.0.1 eureka8763**

运行时，在运行配置项目 Program Arguments 中配置：

**--spring.profiles.active=eureka8761**

**--spring.profiles.active=eureka8762**

**--spring.profiles.active=eureka8763**

分别启动三个注册中心，访问三个注册中心页面，观察注册中心页面是否正常；

**Eureka 注册中心高可用集群测试**

在要进行注册的服务中配置：

*#eureka注册中心的连接地址***eureka.client.service-url.defaultZone**=**http://eureka8761:8761/eureka,http://eureka8762:8762/eureka,http://eureka8763:8763/eureka**

启动服务提供者服务，然后观察注册中心页面，可以看到服务会在三个注册中心

上都注册成功；

**集群的注册中心打包发布**

在真实项目中，需要将Eureka发布到具体服务器上进行执行，打包部署其实和springboot里面的一样，对于properties文件，不同的环境会有不同的配置文件；

运行：

java -jar springcloud-eureka-server.jar --spring.profiles.active=eureka8762；

java -jar springcloud-eureka-server.jar --spring.profiles.active=eureka8762；

java -jar springcloud-eureka-server.jar --spring.profiles.active=eureka8763