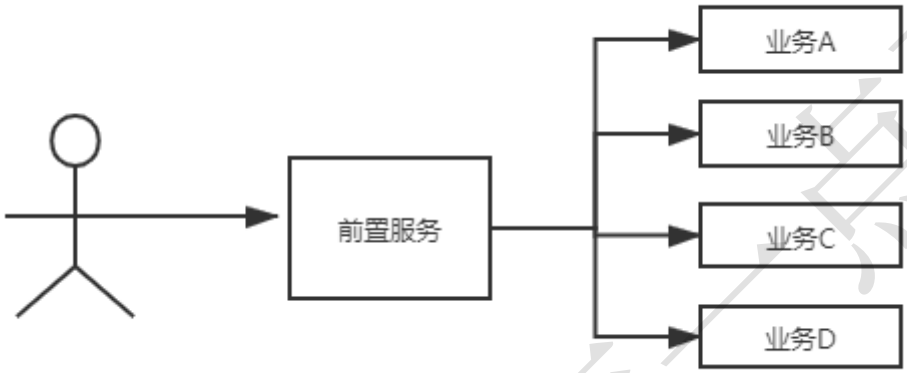
5. Eureka

注册中心

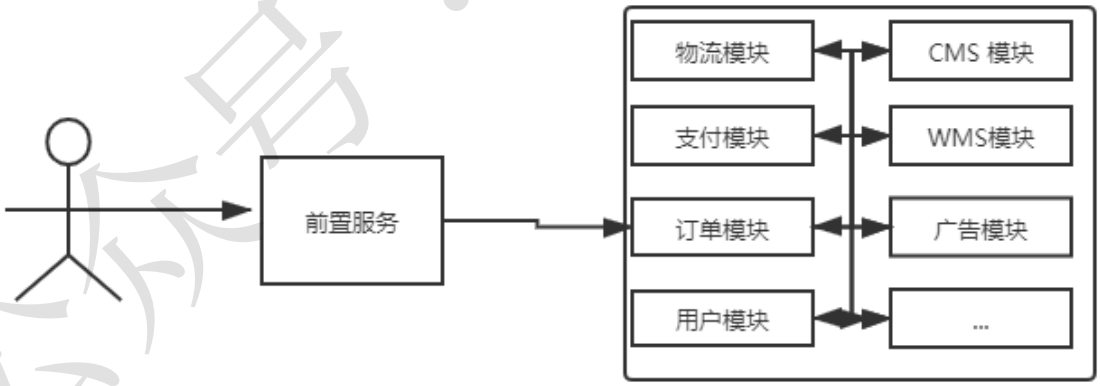
Eureka 是 Spring Cloud 中的注册中心，类似于 Dubbo 中的 Zookeeper。那么到底什么是注册中心，我们为什么需要注册中心?

我们首先来看一个传统的单体应用:



在单体应用中，所有的业务都集中在一个项目中，当用户从浏览器发起请求时，直接由前端发起请求给后端，后端调用业务逻辑，给前端请求做出响应，完成一次调用。整个调用过程是一条直线，不需要服务之间的中转，所以没有必要引入注册中心。

随着公司项目越来越大，我们会将系统进行拆分，例如一个电商项目，可以拆分为订单模块、物流模块、支付模块、CMS 模块等等。这样，当用户发起请求时，就需要各个模块之间进行协作，这样不可避免的要进行模块之间的调用。此时，我们的系统架构就会发生变化:



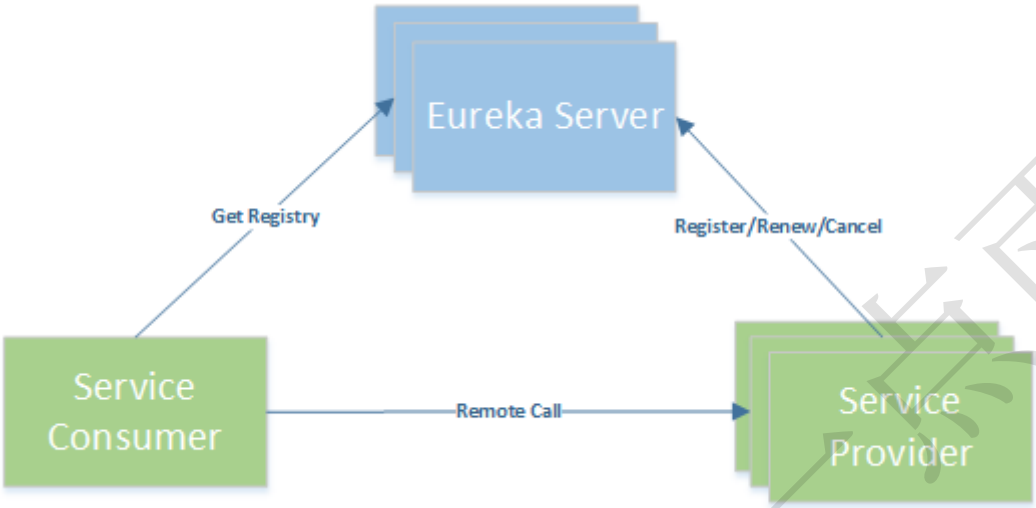
在这里，大家可以看到，模块之间的调用，变得越来越复杂，而且模块之间还存在强耦合。例如 A 调用 B，那么就要在 A 中写上 B 的地址，也意味着 B 的部署位置要固定，同时，如果以后 B 要进行集群化部署，A 也需要修改。

为了解决服务之间的耦合，注册中心闪亮登场。

Eureka

Eureka 是 Netflix 公司提供的一款服务注册中心，Eureka 基于 REST 来实现服务的注册与发现，曾经，Eureka 是 Spring Cloud 中最重要的核心组件之一。Spring Cloud 中封装了 Eureka，在 Eureka 的基础上，优化了一些配置，然后提供了可视化的页面，可以方便的查看服务的注册情况以及服务注册中心集群的运行情况。

Eureka 由两部分：服务端和客户端，服务端就是注册中心，用来接收其他服务的注册，客户端则是一个 Java 客户端，用来注册，并可以实现负载均衡等功能。



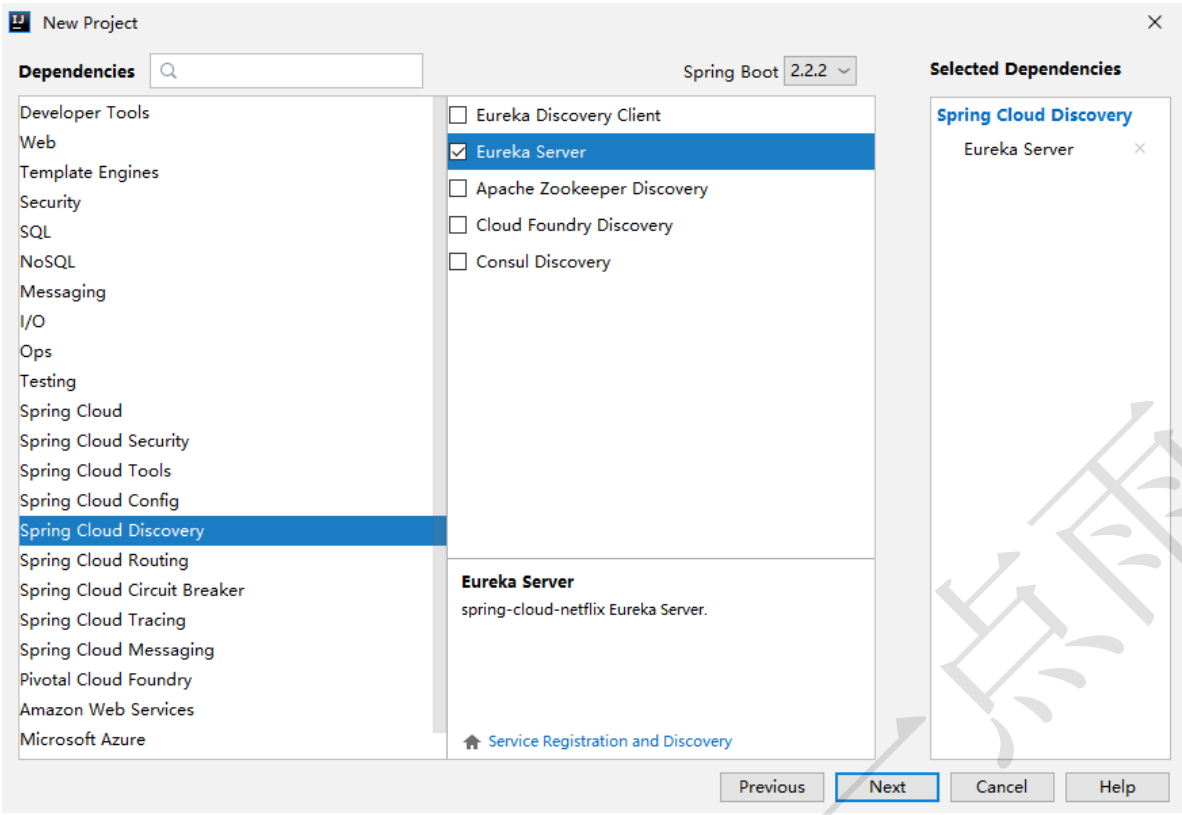
从图中，我们可以看出，Eureka 中，有三个角色：

- Eureka Server：注册中心
- Eureka Provider：服务提供者
- Eureka Consumer：服务消费者

5.1 Eureka 搭建

Eureka 本身是使用 Java 来开发的，Spring Cloud 使用 Spring Boot 技术对 Eureka 进行了封装，所以，在 Spring Cloud 中使用 Eureka 非常方便，只需要引入 spring-cloud-starter-netflix-eureka-server 这个依赖即可，然后就像启动一个普通的 Spring Boot 项目一样启动 Eureka 即可。

创建一个普通的 Spring Boot 项目，创建时，添加 Eureka 依赖：



项目创建成功后，在项目启动类上添加注解，标记该项目是一个 Eureka Server：

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {

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

}
```

`@EnableEurekaServer` 注解表示开启 Eureka 的功能。

接下来，在 application.properties 中添加基本配置信息：

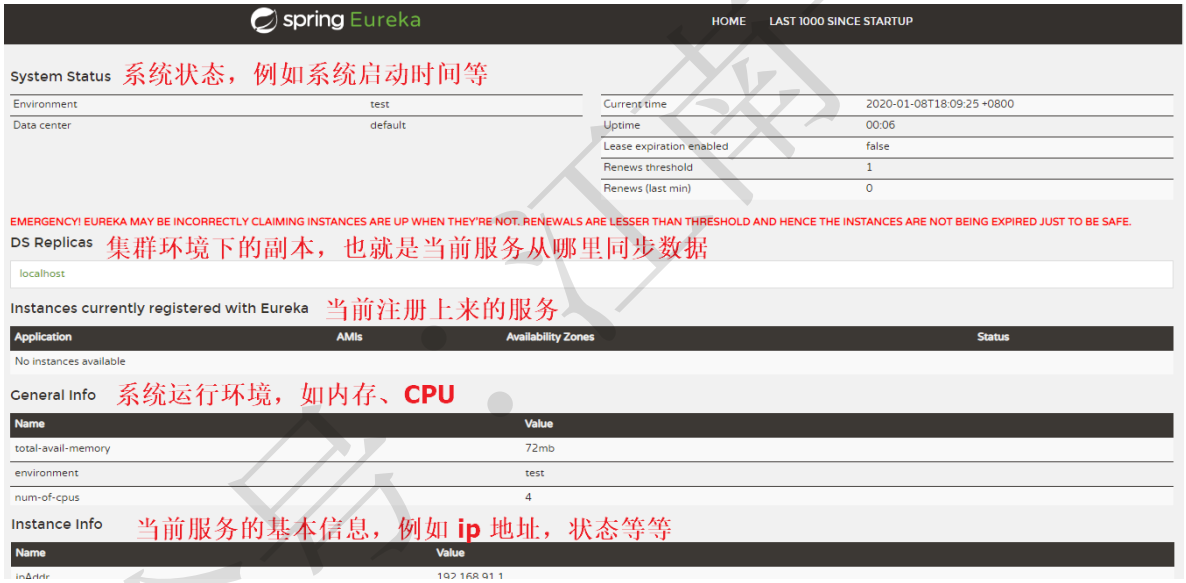
```
# 给当前服务取一个名字
spring.application.name=eureka
# 设置端口号
server.port=1111
# 默认情况下，Eureka Server 也是一个普通的微服务，所以当它还是一个注册中心的时候，他会有两层身份：1.注册中心；2.普通服务，即当前服务会自己把自己注册到自己上面来
# register-with-eureka 设置为 false，表示当前项目不要注册到注册中心上
eureka.client.register-with-eureka=false
# 表示是否从 Eureka Server 上获取注册信息
eureka.client.fetch-registry=false
```

配置完成后，就可以启动项目了。

如果在项目启动时，遇到 `java.lang.TypeNotPresentException: Type javax.xml.bind.JAXBContext not present` 异常，这是因为 JDK9 以上，移除了 JAXB，这个时候，只需要我们手动引入 JAXB 即可。

```
 <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>javax.activation</groupId>
    <artifactId>activation</artifactId>
    <version>1.1.1</version>
</dependency>
```

项目启动成功后，浏览器输入 http://localhost:1111 就可以查看 Eureka 后台管理页面了：



5.2 Eureka 集群

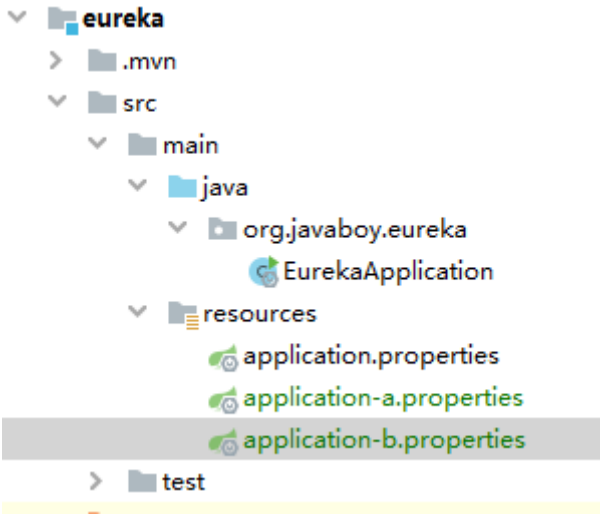
使用了注册中心之后，所有的服务都要通过服务注册中心来进行信息交换。服务注册中心的稳定性就非常重要了，一旦服务注册中心掉线，会影响到整个系统的稳定性。所以，在实际开发中，Eureka 一般都是以集群的形式出现的。

Eureka 集群，实际上就是启动多个 Eureka 实例，多个 Eureka 实例之间，互相注册，互相同步数据，共同组成一个 Eureka 集群。

搭建 Eureka 集群，首先我们需要一点准备工作，修改电脑的 hosts 文件（`C:\Windows\System32\drivers\etc\hosts`）：

```
127.0.0.1 eureka-a eureka-b
```

在 5.1 小节 demo 的基础上，我们在 resources 目录下，再添加两个配置文件，分别为 application-a.properties 以及 application-b.properties:



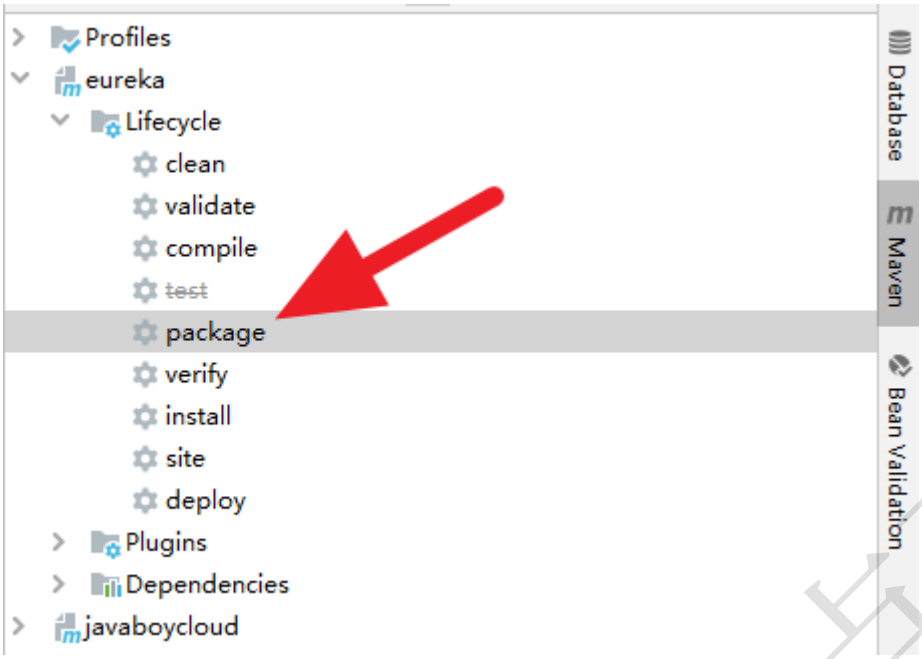
application-a.properites 内如如下：

```
# 给当前服务取一个名字
spring.application.name=eureka
# 设置端口号
server.port=1111
eureka.instance.hostname=eurekaA
# 默认情况下，Eureka Server 也是一个普通的微服务，所以当它还是一个注册中心的时候，他会有两层身份：1.注册中心；2.普通服务，即当前服务会自己把自己注册到自己上面来
# register-with-eureka 设置为 false，表示当前项目不要注册到注册中心上
eureka.client.register-with-eureka=true
# 表示是否从 Eureka Server 上获取注册信息
eureka.client.fetch-registry=true
# A 服务要注册到 B 上面
eureka.client.service-url.defaultZone=http://eurekaB:1112/eureka
```

application-b.properites 内如如下：

```
# 给当前服务取一个名字
spring.application.name=eureka
# 设置端口号
server.port=1112
eureka.instance.hostname=eurekaB
# 默认情况下，Eureka Server 也是一个普通的微服务，所以当它还是一个注册中心的时候，他会有两层身份：1.注册中心；2.普通服务，即当前服务会自己把自己注册到自己上面来
# register-with-eureka 设置为 false，表示当前项目不要注册到注册中心上
eureka.client.register-with-eureka=true
# 表示是否从 Eureka Server 上获取注册信息
eureka.client.fetch-registry=true
eureka.client.service-url.defaultZone=http://eurekaA:1111/eureka
```

配置完成后，对当前项目打包，打成 jar 包：



打包完成后，在命令行启动两个 Eureka 实例。两个启动命令分别如下：

```
java -jar eureka-0.0.1-SNAPSHOT.jar --spring.profiles.active=a
java -jar eureka-0.0.1-SNAPSHOT.jar --spring.profiles.active=b
```

启动成功后，就可以看到，两个服务之间互相注册，共同给组成一个集群。

5.3 Eureka 工作细节

Eureka 本身可以分为两大部分，Eureka Server 和 Eureka Client

5.3.1 Eureka Server

Eureka Server 主要对外提供了三个功能：

1. 服务注册，所有的服务都注册到 Eureka Server 上面来
2. 提供注册表，注册表就是所有注册上来服务的一个列表，Eureka Client 在调用服务时，需要获取这个注册表，一般来说，这个注册表会缓存下来，如果缓存失效，则直接获取最新的注册表
3. 同步状态，Eureka Client 通过注册、心跳等机制，和 Eureka Server 同步当前客户端的状态

5.3.2 Eureka Client

Eureka Client 主要是用来简化每一个服务和 Eureka Server 之间的交互。Eureka Client 会自动拉取、更新以及缓存 Eureka Server 中的信息，这样，即使 Eureka Server 所有节点都宕机，Eureka Client 依然能够获取到想要调用服务的地址（但是地址可能不准确）。

5.3.2.1 服务注册

服务提供者将自己注册到服务注册中心（Eureka Server），需要注意，所谓的服务提供者，只是一个业务上上的划分，本质上他就是一个 Eureka Client。当 Eureka Client 向 Eureka Server 注册时，他需要提供自身的一些元数据信息，例如 IP 地址、端口、名称、运行状态等等。

5.3.2.2 服务续约

Eureka Client 注册到 Eureka Server 上之后，事情没有结束，刚刚开始而已。注册成功后，默认情况下，Eureka CLient 每隔 30 秒就要向 Eureka Server 发送一条心跳消息，来告诉 Eureka Server 我还在运行。如果 Eureka Server 连续 90 秒都有没有收到 Eureka Client 的续约消息（连续三次没发送），它会认为 Eureka Client 已经掉线了，会将掉线的 Eureka Client 从当前的服务注册列表中剔除。

服务续约，有两个相关的属性（一般不建议修改）：

```
eureka.instance.lease-renewal-interval-in-seconds=30
eureka.instance.lease-expiration-duration-in-seconds=90
```

- eureka.instance.lease-renewal-interval-in-seconds 表示服务的续约时间，默认是 30 秒
- eureka.instance.lease-expiration-duration-in-seconds 服务失效时间，默认是 90 秒

5.3.2.3 服务下线

当 Eureka Client 下线时，它会主动发送一条消息，告诉 Eureka Server ，我下线啦。

5.3.2.4 获取注册表信息

Eureka Client 从 Eureka Server 上获取服务的注册信息，并将其缓存在本地。本地客户端，在需要调用远程服务时，会从该信息中查找远程服务所对应的 IP 地址、端口等信息。Eureka Client 上缓存的服务注册信息会定期更新(30 秒)，如果 Eureka Server 返回的注册表信息与本地缓存的注册表信息不同的话，Eureka Client 会自动处理。

这里，也涉及到两个属性，一个是是否允许获取注册表信息：

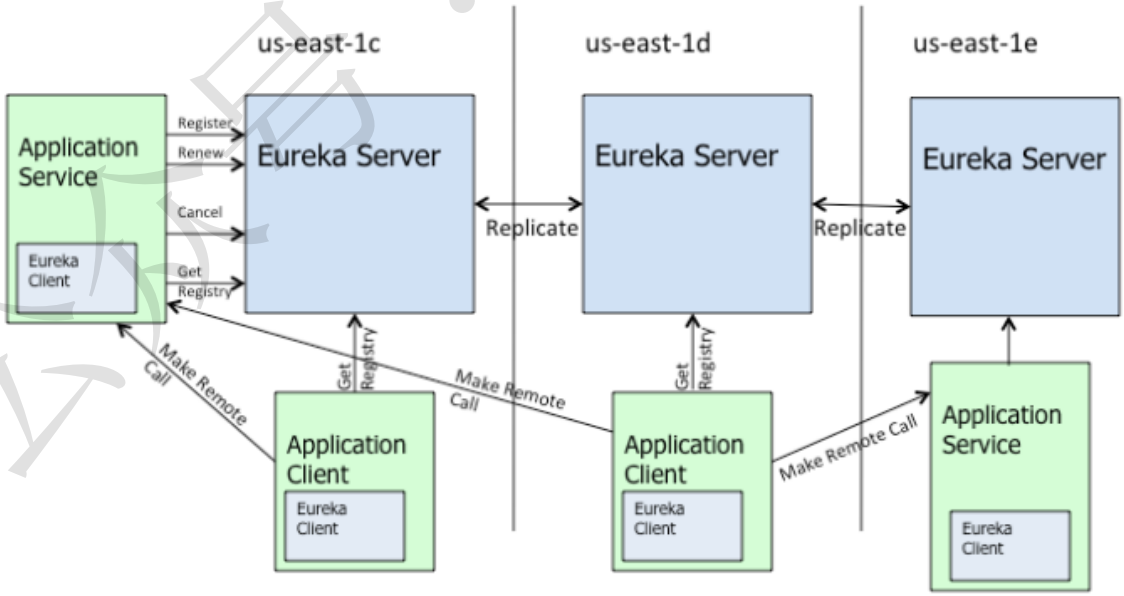
```
eureka.client.fetch-registry=true
```

Eureka Client 上缓存的服务注册信息，定期更新的时间间隔，默认 30 秒：

```
eureka.client.registry-fetch-interval-seconds=30
```

5.4 Eureka 集群原理

我们来看官方的一张 Eureka 集群架构图：



在这个集群架构中，Eureka Server 之间通过 Replicate 进行数据同步，不同的 Eureka Server 之间不区分主从节点，所有节点都是平等的。节点之间，通过置顶 serviceUrl 来互相注册，形成一个集群，进而提高节点的可用性。

在 Eureka Server 集群中，如果有某一个节点宕机，Eureka Client 会自动切换到新的 Eureka Server 上。每一个 Eureka Server 节点，都会互相同步数据。Eureka Server 的连接方式，可以是单线的，就是 A-->b-->C ，此时，A 的数据也会和 C 之间互相同步。但是一般不建议这种写法，在我们配置 serviceUrl 时，可以指定多个注册地址，即 A 可以即注册到 B 上，也可以同时注册到 C 上。

Eureka 分区：

1. region：地理上的不同区域
2. zone：具体的机房