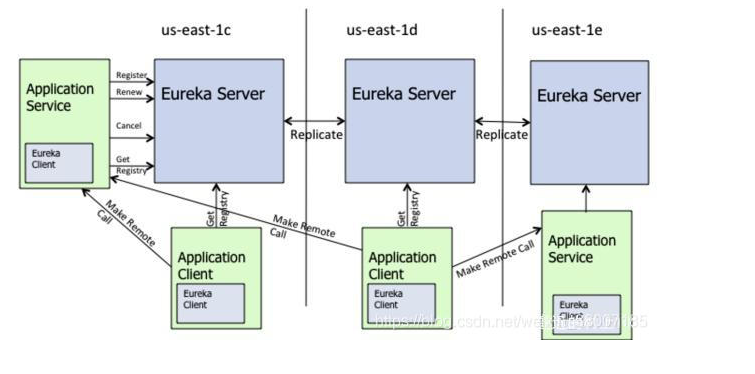
# 基本概念

Eureka是 Spring Cloud Netflix 组件之一，是服务注册中心，全部服务都需要进行注册才能使用，也是微服务架构中必不可少的一个组件。系统中的其他服务使用Eureka的客户端将其连接到Eureka Service中，并且保持心跳，这样可以通过Eureka Service来监控各个微服务是否运行正常。

# Eureka架构



从图中我们可以看出，Eureka 组件分为两部分：Eureka server和 Eureka client。而客户端又分为 Application Service 客户端和 Application Client 客户端两种。Eureka Client是一个Java客户端，用于简化与Eureka Server的交互；Eureka Server提供服务发现的能力，各个微服务启动时，会通过Eureka Client向Eureka Server进行注册自己的信息（例如网络信息），Eureka Server会存储该服务的信息；微服务启动后，会周期性地向Eureka Server发送心跳（默认周期为30秒）以续约自己的信息。如果Eureka Server在一定时间内没有接收到某个微服务节点的心跳，Eureka Server将会注销该微服务节点（默认90秒）；每个Eureka Server同时也是Eureka Client，多个Eureka Server之间通过复制的方式完成服务注册表的同步；Eureka Client会缓存Eureka Server中的信息。即使所有的Eureka Server节点都宕掉，服务消费者依然可以使用缓存中的信息找到服务提供者。

Make Remote Call，可以简单理解为调用RESTful API，us-east-1c、us-east-1d等都是zone，它们组成us-east-1这个服务注册中心。

Eureka 的工作机制每个 region 都有自己的 Eureka 服务器集群，每个 zone 至少要有一个 Eureka 服务器以应对 zone 瘫痪。

## 组成角色

简要描述了Eureka的基本架构，由3个角色组成：

### 1、Eureka Server

提供服务注册和发现

### 2、Service Provider

服务提供方将自身服务注册到Eureka，从而使服务消费方能够找到。

### 3、Service Consumer

服务消费方从Eureka获取注册服务列表，从而能够消费服务。

# 微服务为什么需要注册中心

## 1、简单易用

最好对开发者透明

## 2、高可用

几台注册中心坏掉不会导致整个服务瘫痪，注册服务整体持续可用。

## 3、避免跨越机房调用

最好调用优先同一个机房的服务以减少网络延迟

## 4、跨语言

允许开发者使用多种编程语言构建微服务

# spring-cloud-starter-eureka-server和spring-cloud-starter-netflix-eureka-server的区别

spring-cloud-starter-[eureka](https://so.csdn.net/so/search?q=eureka&spm=1001.2101.3001.7020)-server 在1.5版本中可以使用，在2.0版本中废弃，建议使用spring-cloud-starter-netflix-eureka-server

# 示例

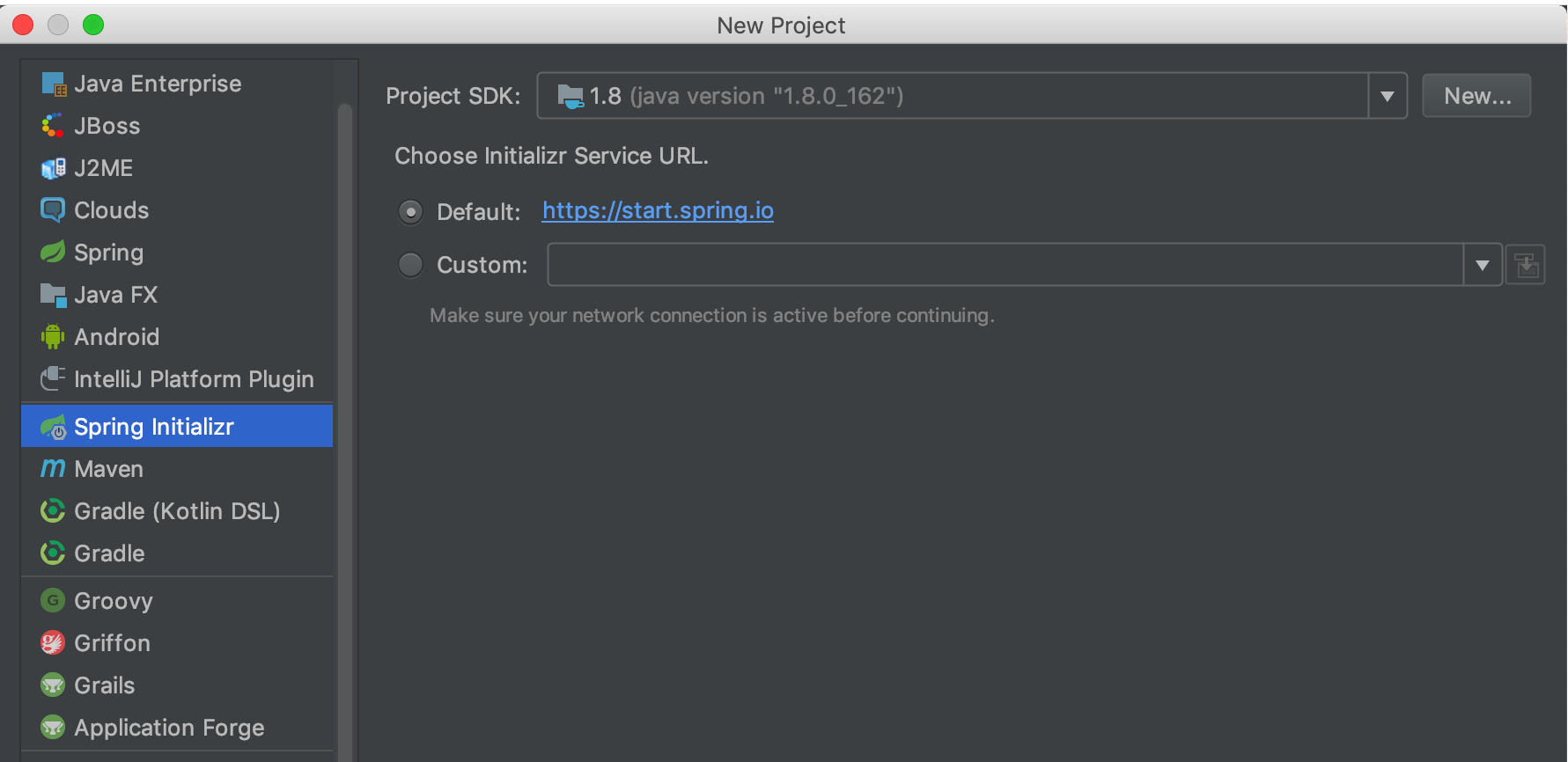
## 背景

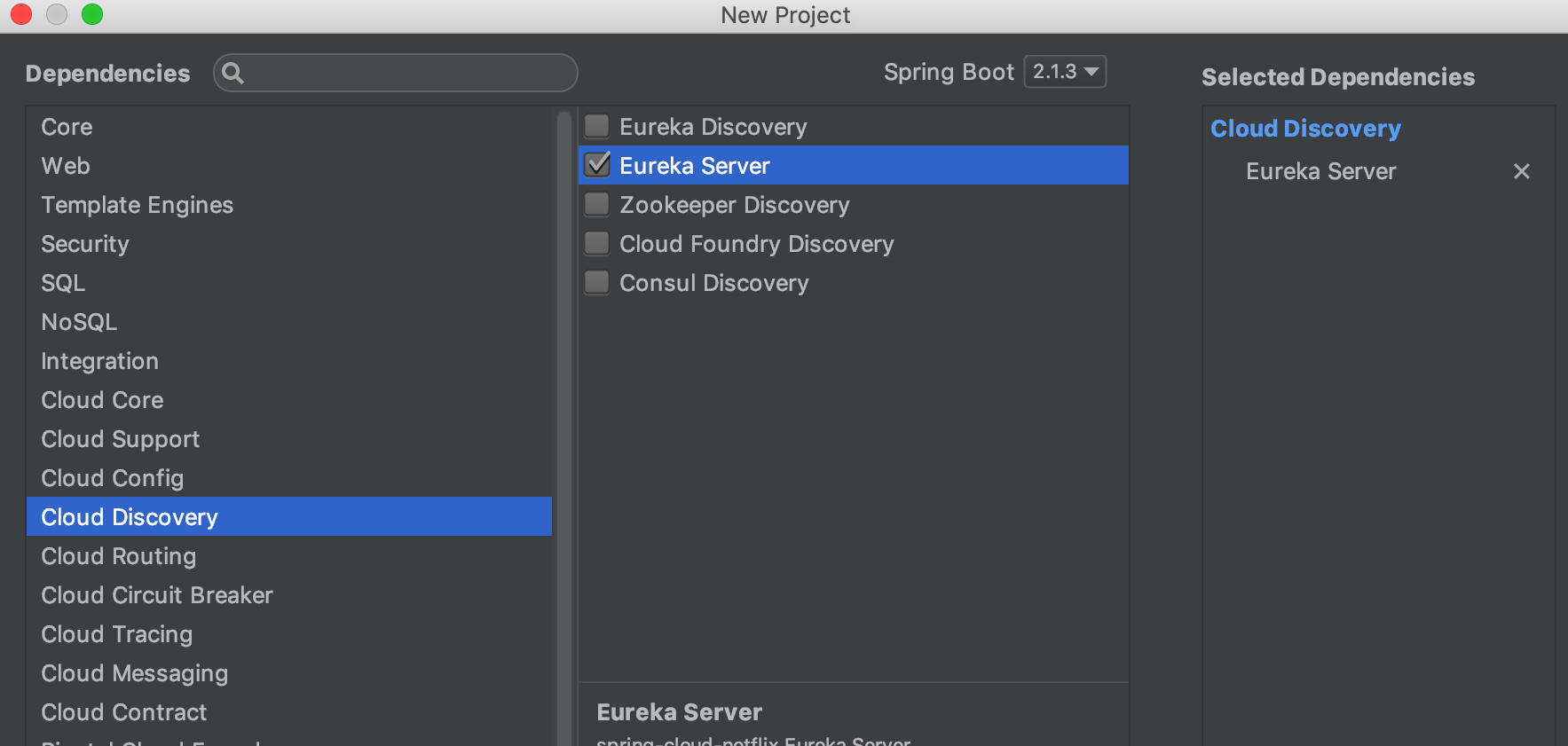
服务：1、注册中心服务端  2、商品服务  3、订单服务

需求：订单服务需要查询商品信息完成下单服务， 商品服务提供商品查询的接口并且注册到服务中心，订单服务通过服务中心调用商品服务，从而拿到商品信息。

## 创建Eureka服务端

1、使用IDEA快速创建Spring Boot项目，并且选择Eureka服务端(记得把Web的引用也勾选上，不然会出现项目启动不了的情况)





完整的pom.xml文件

文本

描述已自动生成

文本

描述已自动生成

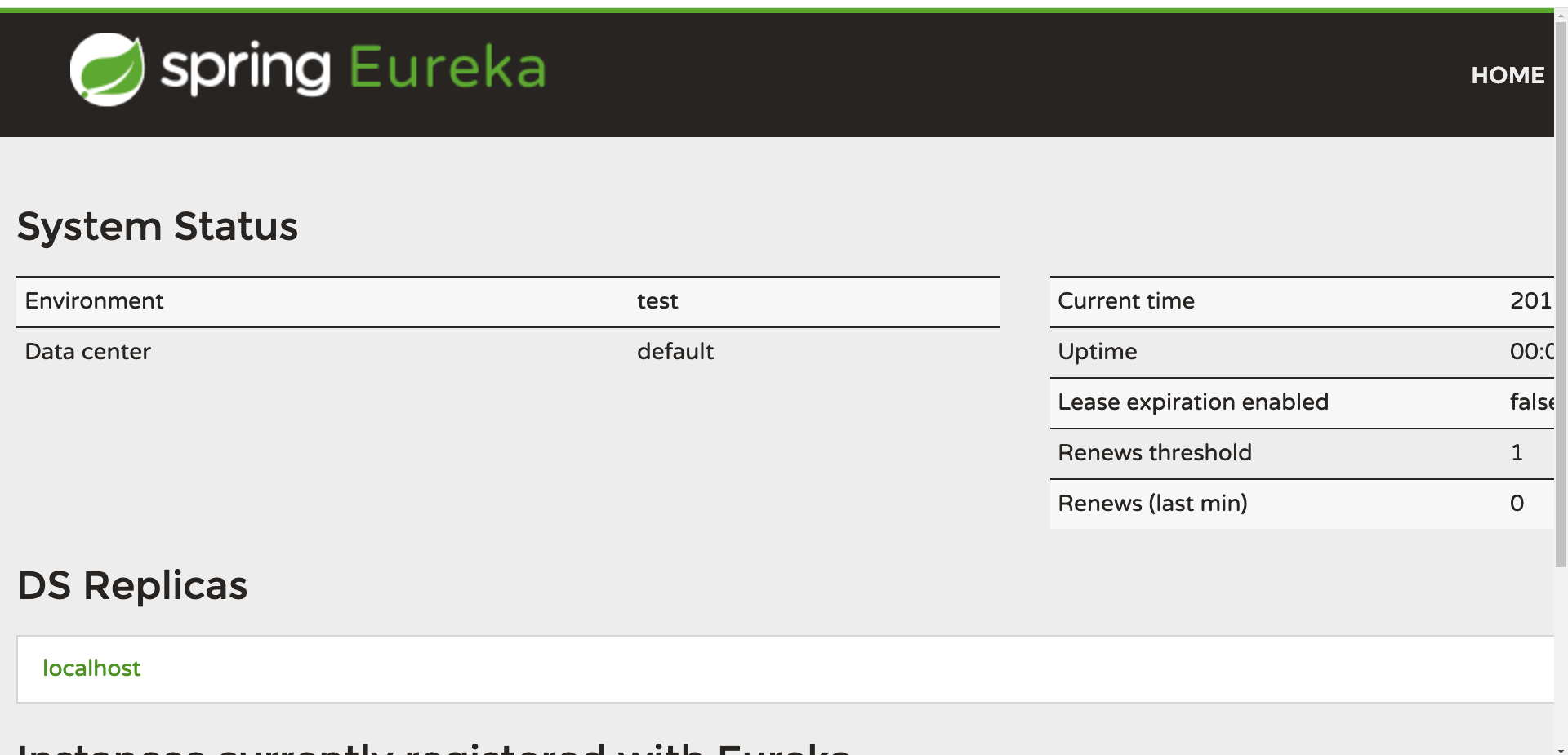
文本

描述已自动生成

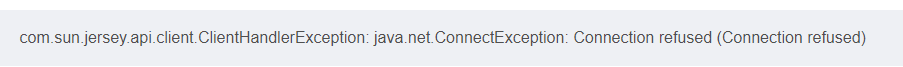
2、在Spring Boot启动类上加Eureka服务端注解（@EnableEurekaServer）



3、直接启动,默认端口应该是8080，[http://localhost:8080](http://localhost:8080/)，看到如下页面则表示服务端启动成功。

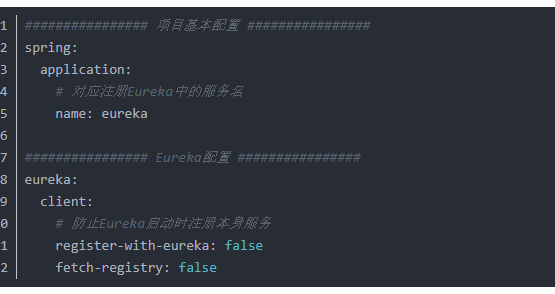


4、如果按照以上步骤启动成功，控制台应该会抛出以下一段异常信息。这个异常表示Eureka启动时会注册自己本身的服务，而本身服务却还没启动成功，则抛出异常。



解决方法：

修改Eureka默认配置，默认习惯使用yml格式的配置文件，修改配置如下：

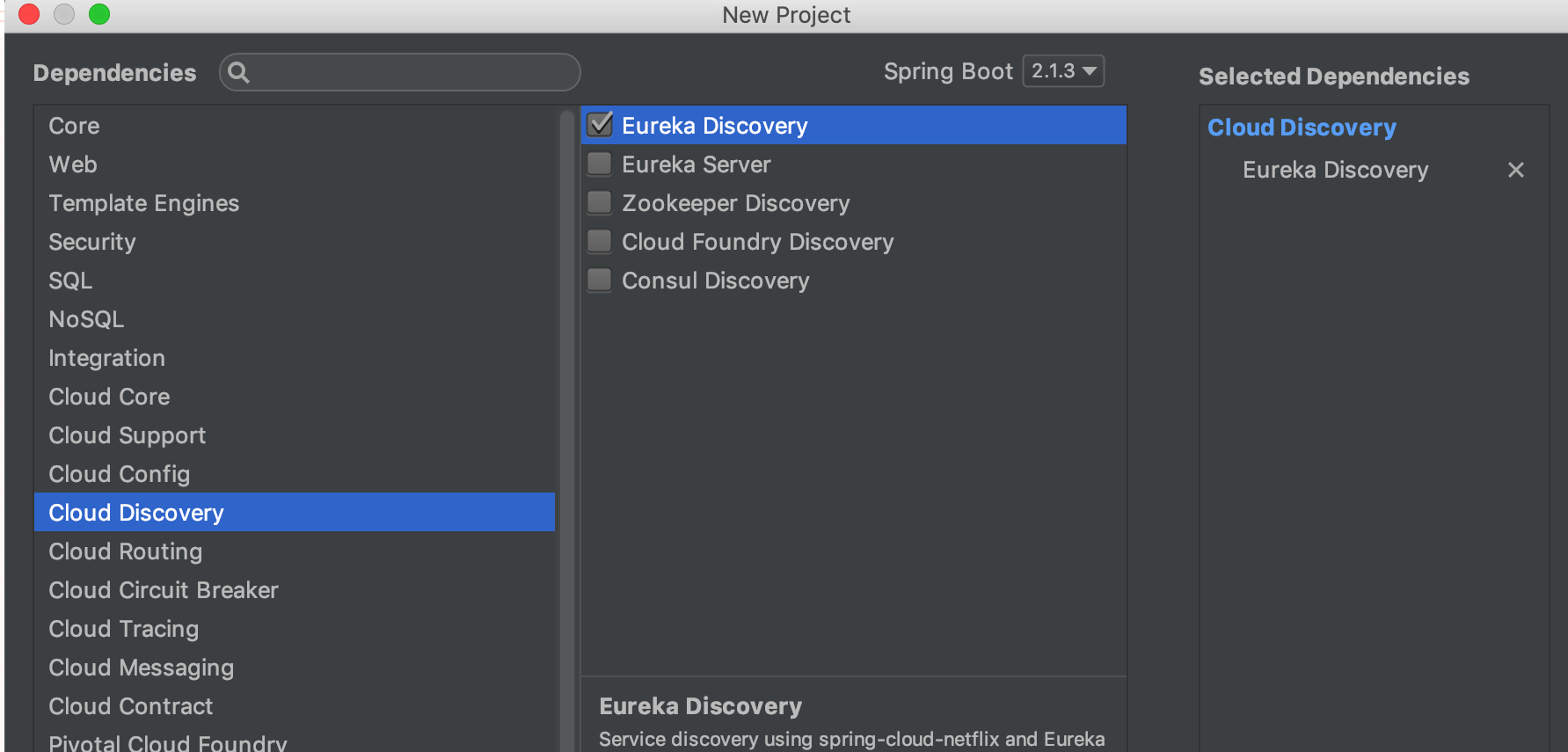


## 创建Eureka客户端

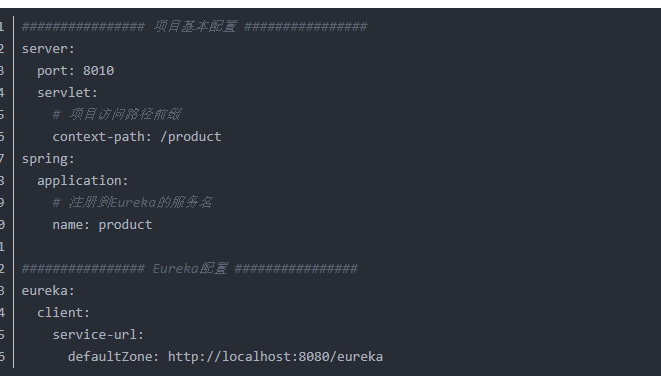
针对实战案例来说，需要创建一个商品项目、订单项目并且都需要注册到Eureka服务中心，以商品服务为例：

### 创建商品服务

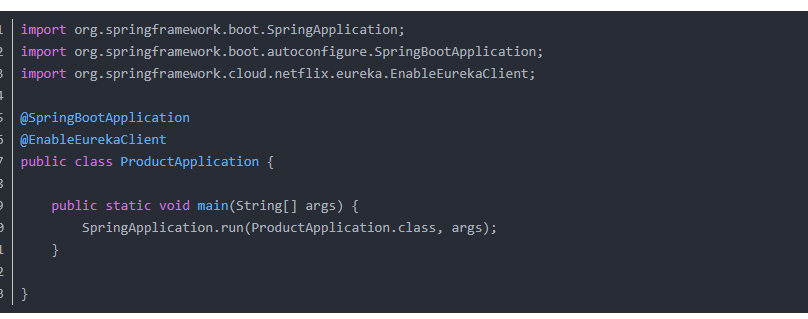
1、采用IDEA快速创建Spring Boot项目，并且选择Eureka客户端依赖(记得把Web也勾选上，不然会出现项目启动不了)



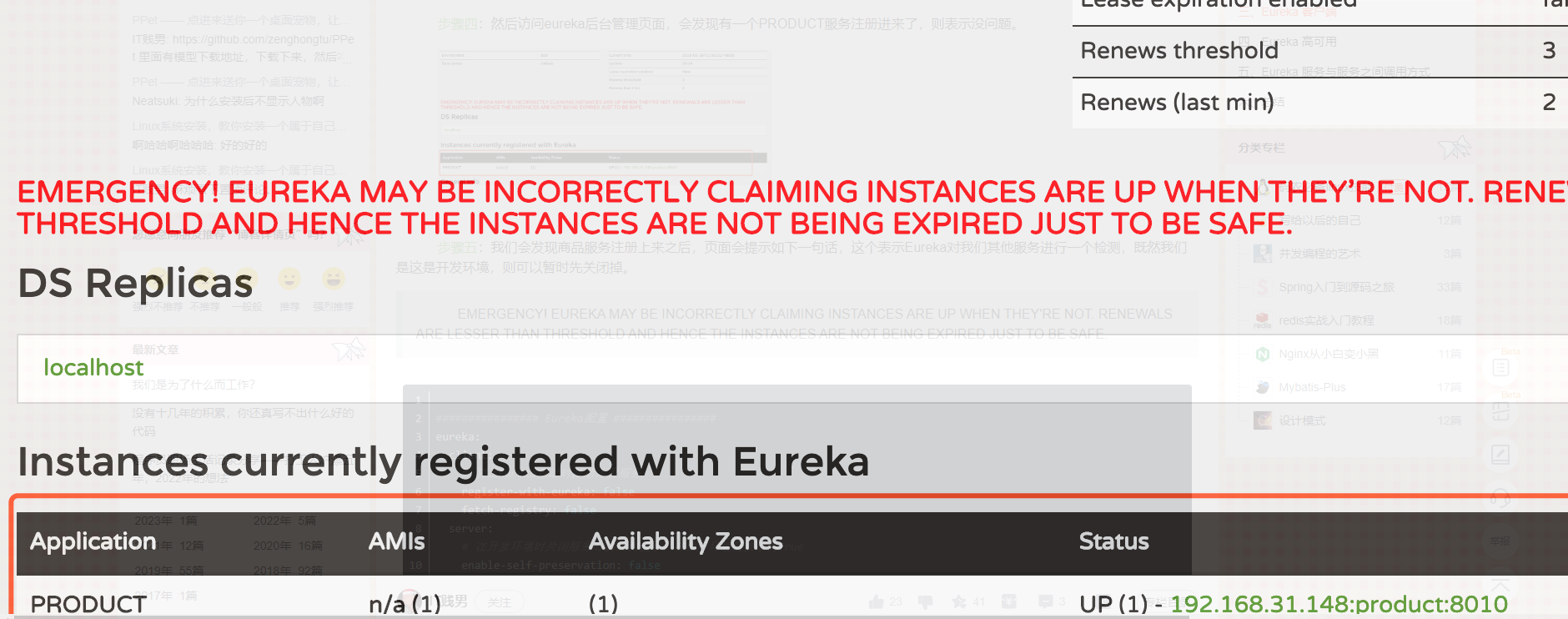
2、配置Eureka服务端地址，还是yml格式的。



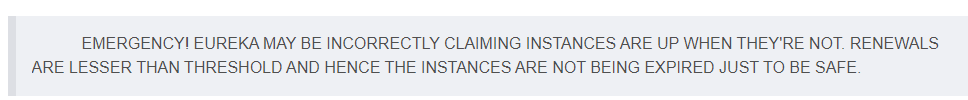
3、SpringBoot启动类上加上Eureka客户端（@EnableEurekaClient）注解，直接启动。



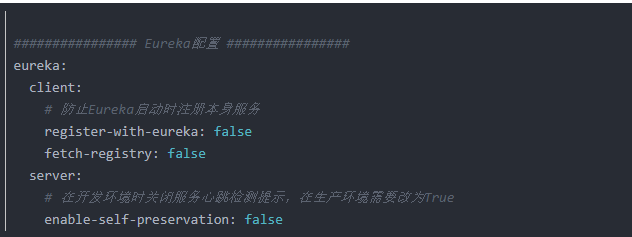
4、然后访问eureka后台管理页面，会发现有一个PRODUCT服务注册进来了，则表示没问题。



5、我们会发现商品服务注册上来之后，页面会提示如下一句话，这个表示Eureka对我们其他服务进行一个检测，既然我们这是开发环境，则可以暂时先关闭掉。

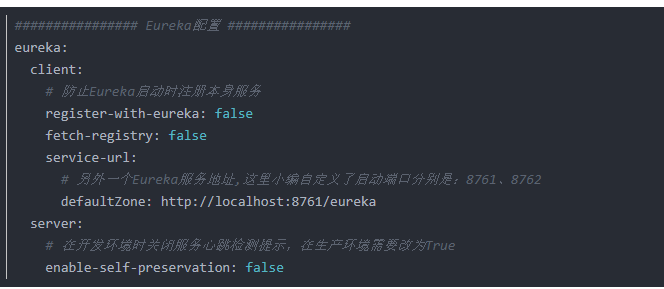


说明：在eureka服务端项目中的配置文件修改。



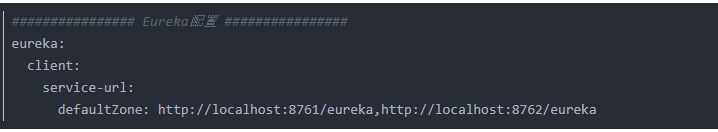
## Eureka高可用

既然我们Eureka控制全场服务，那么单个服务注册中心肯定不够用，万一出现单点故障那就凉了。所以再来说说Eureka怎么实现高可用，看Eureka服务端配置。



启动两个eureka服务端的项目，并且相互注册即可，然后我们把商品服务往其中一个Eureka服务端注册，如果看到两个Eureka服务端后台页面都有PRODUCT服务注册进来了，则表示成功。

有一个问题，现在我们商品服务只是往其中一个Eureka服务进行注册了，如果高可用，其中一台Eureka挂掉了，那么商品服务再次注册的时候就会有问题，所以我们的商品服务、订单服务注册Eureka的配置就需要改动，如下所示：

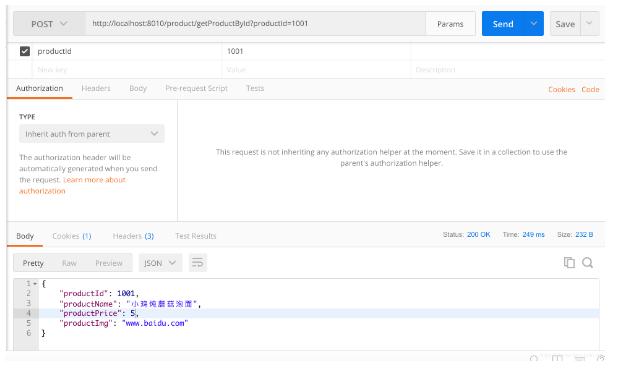


## Eureka服务与服务之间调用方式

现在订单服务下单，订单服务需要通过前端传来的商品ID，调用商品服务从而获取商品的金额、库存等。

商品服务：需要提供一个接口，通过商品id查询商品的具体信息，我们需要编写一个查询商品的接口，这里就不写了。重点在于订单服务怎么调用商品服务的接口。

这是查询商品的接口，如图所示：



订单服务：已知商品ID，需要获取商品的信息。

### 方法一：使用RestTemplate

第一种方式很简单，直接调用RestTemplate的方法即可。



http://localhost:8010/product/getProductById

### 方法二：使用LoadBalancerClient对象

通过LoadBalancerClient对象，获取你调用的服务，从而获得该服务的IP地址、端口信息等。



http://%s:%s/product/getProductById

### 方法三：使用@LoadBalanced注解

需要配置一个Bean，主要是标注了@LoadBalanced该注解。需要使用通过注解直接注入RestTemplate该对象即可使用，代码如下：

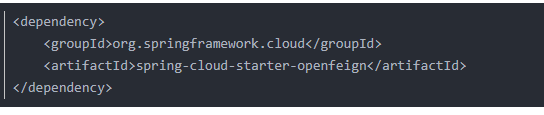




http://Product/product/getProductById

### 方法四：使用Feign

使用Feign调用方式，首先我们需要引入依赖的Jar



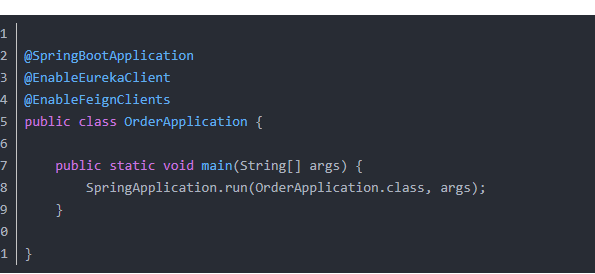
配置每个服务的接口类，参数和路径都和商品服务一一对应。小编个人感觉这种方式稍微比较规范一点，实际项目中也使用的偏多。



使用方式：注入服务接口定义类即可。



最后千万要记住，SpringBoot启动类上需要加上Feign的注解（@EnableFeignClients）。



最后效果是能通过订单服务获取到商品的信息

# DS Replicas

# eureka默认端口

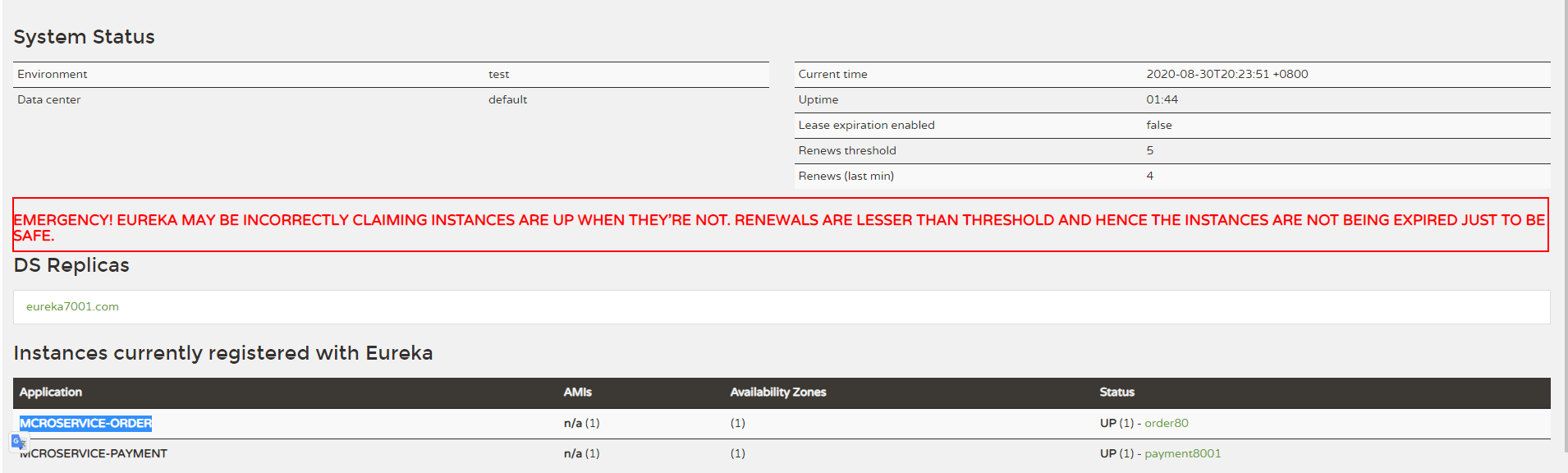
8761

# Eureka怎么实现高可用

创建集群，注册多台Eureka，然后把SpringCloud服务互相注册，客户端从Eureka获取信息时，按照Eureka的顺序来访问。

# 自我保护模式

默认情况下当Eureka server在一定时间内没有收到实例的心跳，便会把该实例从注册表中删除（默认是90秒），但是，如果短时间内丢失大量的实例心跳，便会触发eureka server的自我保护机制。比如在开发测试时，需要频繁地重启微服务实例，但是我们很少会把eureka server一起重启（因为在开发过程中不会修改eureka注册中心），当一分钟内收到的心跳数大量减少时，会触发该保护机制。可以在eureka管理界面看到Renews threshold和Renews(last min)，当后者（最后一分钟收到的心跳数）小于前者（心跳阈值）的时候，触发保护机制，会出现红色的警告：



从警告中可以看到，eureka认为虽然收不到实例的心跳，但它认为实例还是健康的，eureka会保护这些实例，不会把它们从注册表中删掉。在该模式下Eureka Service会保护服务注册表中的信息，不在删除注册表中的数据，当网络故障恢复后，Eureka Servic 节点会自动退出自我保护模式。

# DiscoveryClient的作用

可以从注册中心中根据服务别名获取注册的服务器信息。

# Eureka和ZooKeeper都可以提供服务注册与发现的功能,请说说两个的区别

1、ZooKeeper中的节点服务挂了就要选举，在选举期间注册服务瘫痪，虽然服务最终会恢复,但是选举期间不可用的。

2、Eureka各个节点是平等关系，服务器挂了没关系，只要有一台Eureka就可以保证服务可用，数据都是最新的，如果查询到的数据并不是最新的，就是因为Eureka的自我保护模式导致的。

3、Eureka本质上是一个工程，而ZooKeeper只是一个进程。

4、Eureka可以很好的应对因网络故障导致部分节点失去联系的情况,而不会像ZooKeeper 一样使得整个注册系统瘫痪。

5、ZooKeeper保证的是CP，Eureka保证的是AP。

CAP： C：一致性，A：可用性，P：分区容错性>Partition tolerance

# 已停止的微服务不注销或注销有延迟

## 基本说明

在使用Eureka Client时，可能会发现当微服务（Eureka Client）已经停止了，而注册中心仍然显示该服务处于正常状态，或者过段时间才会注销。然而，这种情况在实际应用中是大家不太希望看见的，希望一旦服务发生异常或宕机，注册中心应该理解体现出来。这是由于Eureka Server注销无效节点周期、自我保护模式的因素造成的，因此会出现服务不注销或注销有延迟。解决方法分为服务端解决和客户端解决。

## Eureka Server

关闭自我保护模式，并配置注销无效节点周期时间间隔。



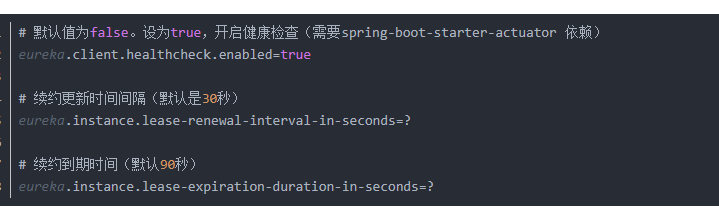
Eureka自我保护模式：

主要用于一组客户端和Eureka Server之间存在网络分区场景下的保护。一旦进入保护模式，Eureka Server将会尝试保护其服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。

## Eureka Client

当Client端开启健康检查时，可以适当的按需配置续约更新时间和到期时间。这样做智能在一定程度上缓解注销延迟的程度，但不能真正解决立即注销，可以在进行微服务(Eureka Client)异常或关机时，主动调用[Eureka Rest API](https://github.com/Netflix/eureka/wiki/Eureka-REST-operations)来注销该服务，注销接口：

http://localhost:8761/eureka/apps的DELETE 请求方式



# Eureka的UNKNOWN问题

注册中心(Eureka Server)中服务状态，常见的有UP、DOWN，但有时会出现另外一种UNKNOWN状态，或者UNKNOWN服务名。

## UNKNOWN服务名

问题显而易见，只需在服务(Eureka Client)配置中配置明确的服务名即可。配置参数为spring.application.name 或者 eureka.instance.appname ，如果这两个参数都不配置，则将会出现UNKNOWN服务名。温馨提示：为了程序的可读性，极力推荐配置明确的服务名。

## UNKNOWN状态

服务为UNKNOWN状态，是很少见的问题。服务状态用于辨别服务是否可用（可调用），来源于对服务的健康检查(需要spring-boot-starter-actuator的依赖)所致。当服务(Eureka Client)配置参数eureka.client.healthcheck.enabled=true时，必须要配置在application.yml或application.properties配置文件中，而不能配置在bootstrap.yml或bootstrap.properties中，否则在一些情况下可能会出现UNKNOWN状态的问题。

# 客户端启动时如何注册到服务端

Eureka客户端在启动时，首先会创建一个心跳的定时任务，定时向服务端发送心跳信息，服务端会对客户端心跳做出响应，如果响应状态码为404时，表示服务端没有该客户端的服务信息，那么客户端则会向服务端发送注册请求，注册信息包括服务名、ip、端口、唯一实例ID等信息。

# 服务端如何保存客户端的服务信息

客户端通过Jersey框架（亚马逊的一个http框架）将服务实例信息发送到服务端，服务端将客户端信息放在一个ConcurrentHashMap对象中。

# 客户端如何拉取服务端已保存的服务信息（是需要使用的时候再去拉取，还是先拉取保存本地，使用的时候直接从本地获取）

客户端拉取服务端服务信息是通过一个定时任务定时拉取的，每次拉取后刷新本地已保存的信息，需要使用时直接从本地直接获取。

# 如何构建高可用的Eureka集群

1、搭建高可用的Eureka集群，只需要在注册中心的配置文件中配置其他注册中心的地址。

2、在eureka的高可用状态下，这些注册中心是对等的，他们会互相将注册在自己的实例同步给其他的注册中心，同样是通过问题1的方式将注册在自己上的实例注册到其他注册中心去。

3、那么问题来了，一旦其中一个eureka收到一个客户端注册实例时，既然eureka注册中心将注册在自己的实例同步到其他注册中心中的方式和客户端注册的方式相同，那么在接收的eureka注册中心一端，会不会再同步回给注册中心（或者其他注册中心），从而导致死循环。

注册中心收到注册信息后会判断是否是其他注册中心同步的信息还是客户端注册的信息，如果是客户端注册的信息，那么他将会将该客户端信息同步到其他注册中心去；否则收到信息后不作任何操作。通过此机制避免集群中信息同步的死循环。

# 服务续约

在注册服务完成以后，服务提供者会维持一个心跳(每30s定时向EurekaServer 发起请求)告诉EurekaServer "我还活着"。

# 失效剔除

有时候,我们的服务提供方并不一定是正常下线,可能是内存溢出,网络故障等原因导致服务无法正常工作，EurekaServer会将这些失效的服务剔除服务列表。因此它会开启一个定时任务，每隔60秒会对失效的服务进行一次剔除。

# Eureka包含CAP中的哪些

CAP理论:一个分布式系统不可能同时满足C (一致性),A(可用性),P(分区容错性).由于分区容错性P在分布式系统中是必须要保证的,因此我们只能从A和C中进行权衡，Eureka 遵守 AP。

Eureka各个节点都是平等的，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 注册或查询是如果发现连接失败,则会自动切换至其他节点。只要有一台Eureka还在,就能保证注册服务可用(保证可用性),只不过查的信息可能不是最新的，不保证强一致性。

# Eureka默认的负载均衡策略

默认为轮询

## 1、随机策略 RandomRule

就像是抽签，闭眼睛随便抓一个。

## 2、轮询策略 RoundRobinRule

按顺序循环选择每个实例。

## 3、重试策略 RetryRule

在轮询策略的基础上增加了重试机制，如果当前选择的实例不可用，就继续按照轮询策略选择下一个实例，直到找到可用的实例。但注意重试是有时间限制的，不能不停的重试。

## 4、最低并发策略 BestAvailableRule

选择并发最小的那个实例，也就是看谁清闲就找谁。

## 5、可用过滤策略 AvailabilityFilteringRule

过滤掉那些故障率高、压力大的实例。

## 6、加权轮询策略 WeightedResponseTimeRule

加权就是为每个实例打个分，处理性能好的分值就越高，然后按照分值从高到低的顺序来轮询。

## 7、区域权衡策略 ZoneAvoidanceRule

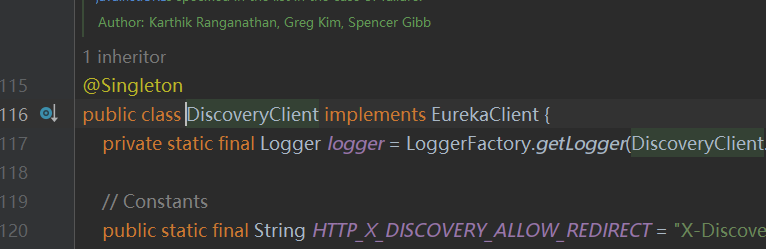
在大型系统中，可能会在多个区域进行部署，那么就可以根据各个区域的服务处理能力来选择实例。

# 常用的类

## DiscoveryClient

### 基本概念

### 源码



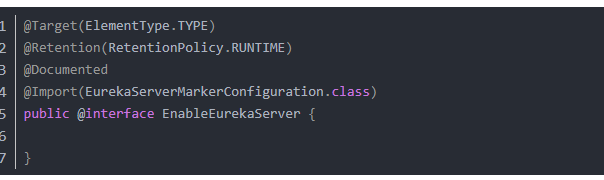
# 涉及的注解

## @EnableEurekaServer

### 基本概念

@EnableEurekaServer 注解是 Spring Cloud Netflix Eureka 提供的核心注解，用于启用 Eureka 服务注册中心功能。

### 源码



### 作用

1、**启用 Eureka Server 功能**：将当前 Spring Boot 应用转换为一个 Eureka 服务注册中心

2、自动配置：自动配置 Eureka Server 相关的所有组件和配置

### 核心功能

#### 1、服务注册与发现

（1）服务注册：其他微服务可以向这个 Eureka Server 注册自己

（2）服务发现：客户端可以从 Eureka Server 查询可用的服务实例

（3）健康检查：定期检查注册服务的健康状态

#### 2、自动装配的组件

（1）EurekaServerConfig：Eureka Server 配置

（2）InstanceRegistry：实例注册表

（3）PeerAwareInstanceRegistry：集群感知的实例注册表

（4）EurekaController：提供 REST API 接口

#### 3、提供的 REST API

Eureka Server 会自动提供以下 REST 端点：

（1）/eureka/apps - 获取所有应用信息

（2）/eureka/apps/{appName} - 获取特定应用信息

（3）/eureka/apps/{appName}/{instanceId} - 服务实例操作

（4）/eureka/status - 服务器状态信息

### 工作原理

1、启动时：创建 Eureka Server 实例，初始化服务注册表

2、运行时：

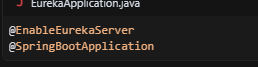
接收服务注册请求

处理心跳续约

清理过期服务实例

同步集群数据（如果是集群模式）

### 代码中的体现



1、应用启动后成为 Eureka 注册中心

2、在 7900 端口提供服务注册与发现功能

3、提供 Web 管理界面（可通过浏览器访问 http://localhost:7900）

4、支持其他微服务的注册、注销、心跳检测等功能

### 总结

这个注解是构建微服务架构中服务注册中心的关键，没有它，应用就只是一个普通的 Spring Boot 应用，而不具备 Eureka Server 的功能。

## @EnableEurekaClient

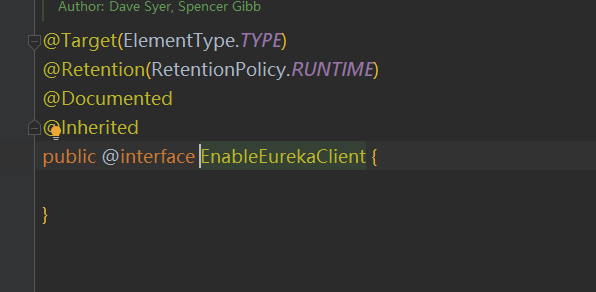
### 基本概念

@EnableEurekaClient是Spring Cloud Netflix中用于启用Eureka客户端功能的注解。

### 说明

spring-cloud-netflix-eureka-client-2.0.2.RELEASE.jar包中

### 源码



### 作用

#### 1、服务注册

自动将当前应用注册到Eureka服务器

#### 2、服务发现

获取其他已注册服务的信息，支持负载均衡调用

### 使用方式



### 配置要求

需在application.yml中配置Eureka服务器地址



### 工作机制

1、**启动时**：向Eureka Server发送注册请求

2、**运行中**：定期发送心跳维持注册状态（默认30秒）

3、**关闭时**：发送下线通知

### 与@EnableDiscoveryClient区别

1、@EnableEurekaClient：专门用于Eureka，功能更丰富

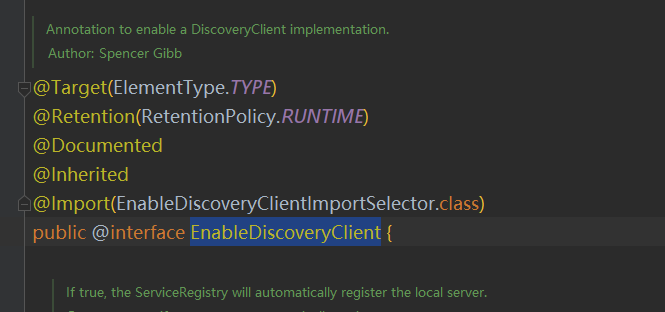
2、@EnableDiscoveryClient：通用服务发现接口，支持多种注册中心

### 自动配置

Spring Boot 2.x后，如果classpath中有eureka-client依赖，会自动启用客户端功能，该注解变为可选。该注解本质上是激活了Spring Cloud的服务治理能力，让微服务具备自动注册发现的分布式特性。

## @EnableDiscoveryClient

### 源码



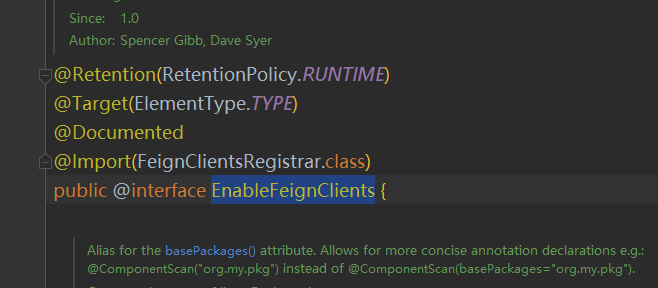
### 作用

## @EnableEurekaClient和@EnableDiscoveryClient区别

@EnableEurekaClient是在spring-cloud-netflix中，仅仅适合于Eureka注册中心。@EnableDiscoveryClient是在spring-cloud-commons中，适合Eureka、Consul、Zookeeper注册中心。在使用时要注意使用依赖的包，否则会发现找不到该注解。

## @EnableFeignClients

### 源码



### 作用