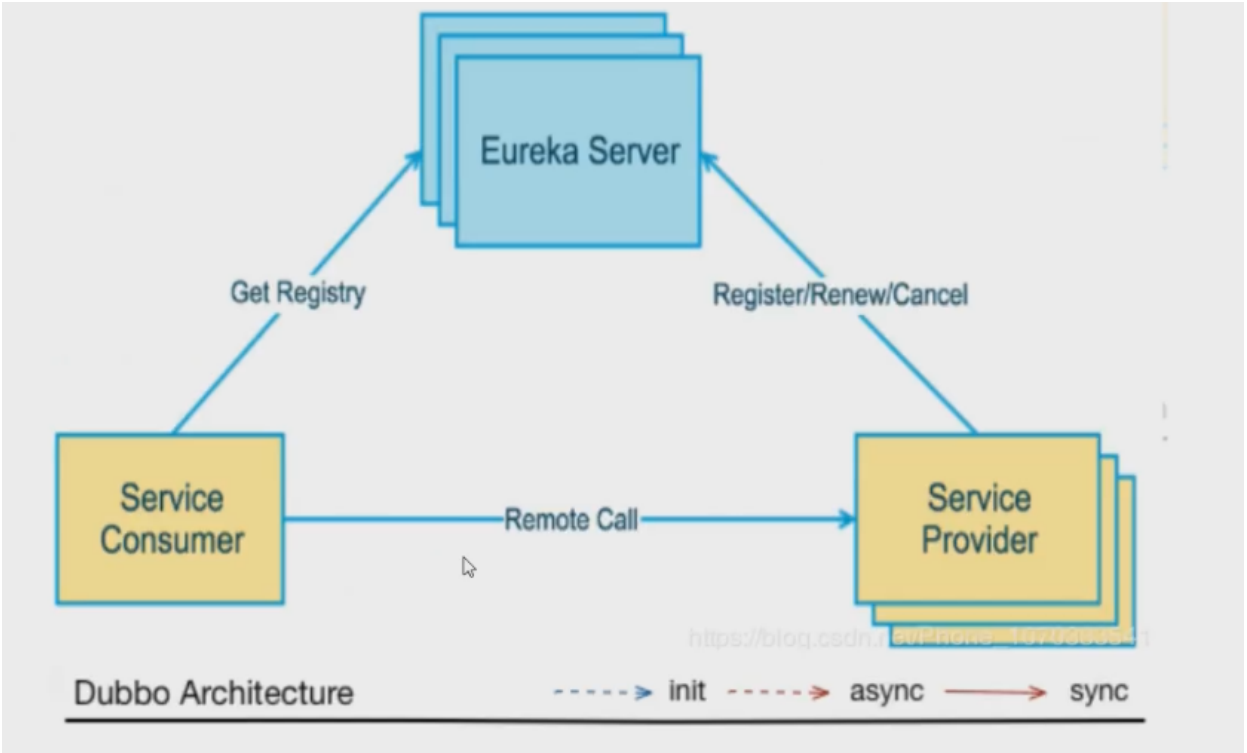
Eureka服务注册与发现

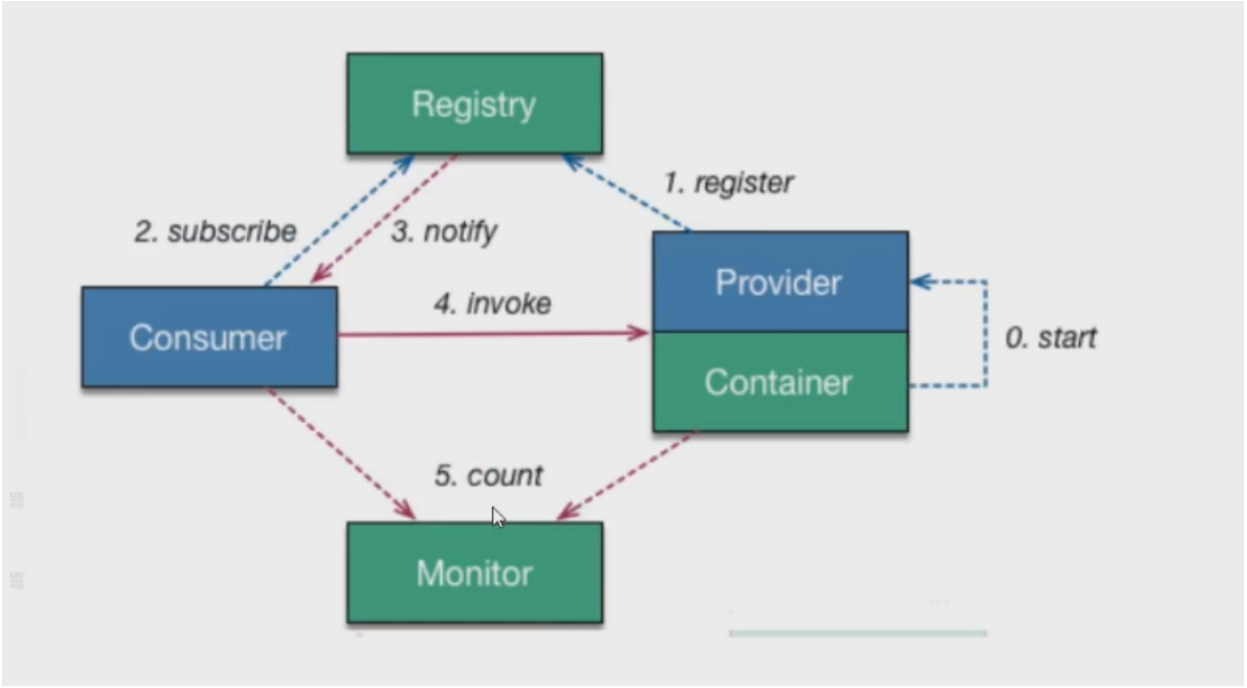
5.1、什么是Eureka

- Eureka：怎么读?
- Netflix 在设计Eureka时，遵循的就是AP原则
- Eureka是Netflix的一个子模块，也是核心模块之一。Eureka是一个基于REST的服务，用于定位服务，以实现云端中间层服务发现和故障转移，服务注册与发现对于微服务来说是非常重要的，有了服务发现与注册，只需要使用服务的标识符，就可以访问到服务，而不需要修改服务调用的配置文件了，功能类似于Dubbo的注册中心，比如Zookeeper；

5.2、原理讲解

- Eureka的基本架构
  - SpringCloud 封装了NetFlix公司开发的Eureka模块来实现服务注册和发现（对比Zookeeper）
  - Eureka采用了C-S的架构设计，EurekaServer 作为服务注册功能的服务器，他是服务注册中心
  - 而系统中的其他微服务。使用Eureka的客户端连接到EurekaServer并维持心跳连接。这样系统的维护人员就可以通过EurekaServer来监控系统中各个微服务是否正常运行，SpringCloud的一些其他模块（比如Zuul）就可以通过EurekaServer来发现系统中的其他微服务，并执行相关的逻辑；
  - 和Dubbo架构对比

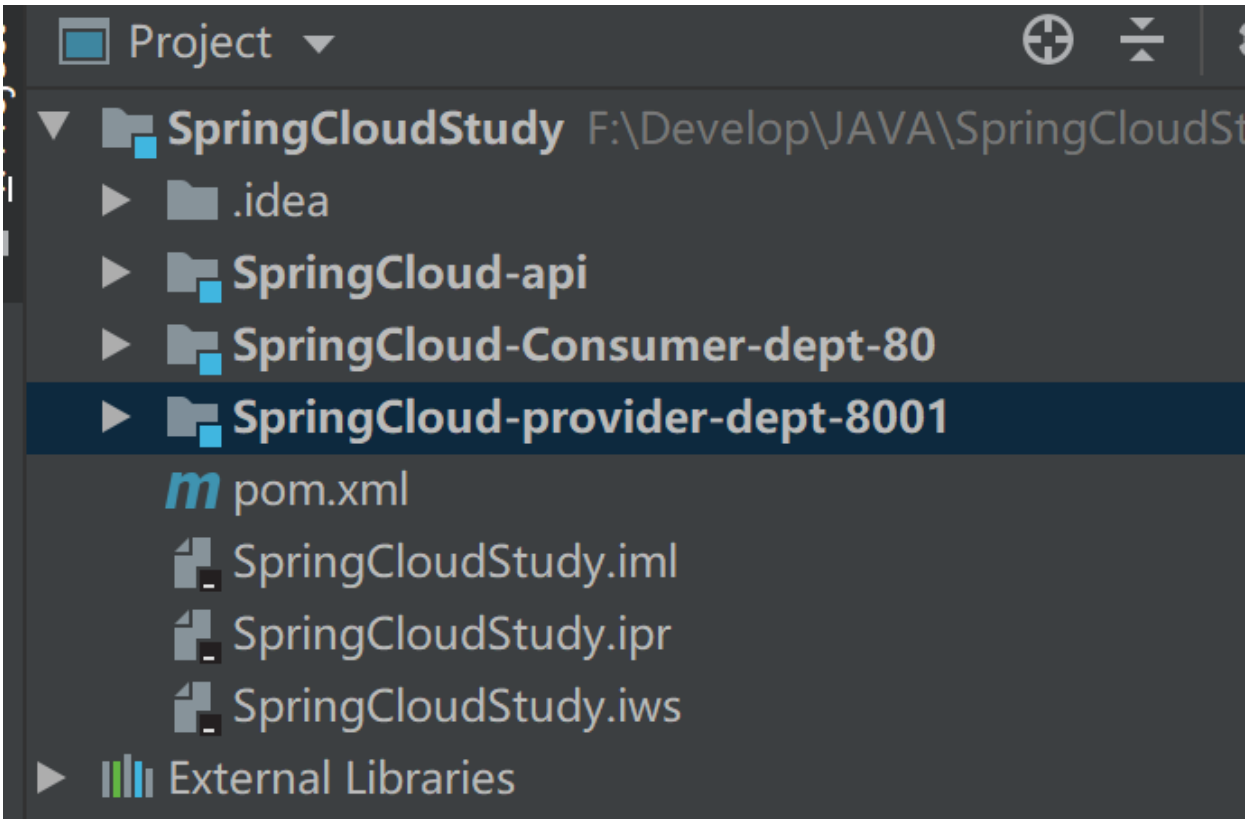




- Eureka 包含两个组件：**Eureka Server** 和 **Eureka Client** 。
- Eureka Server提供服务注册服务，各个节点启动后，会在EurekaServer中进行注册，这样Eureka Server中的服务注册表中将会村粗所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到。
- Eureka Client是一个Java客户端，用于简化EurekaServer的交互，客户端同时也具备一个内置的，使用轮询负载算法的负载均衡器。在应用启动后，将会向EurekaServer发送心跳（默认周期为30秒）。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除掉（默认周期为90秒）

- 三大角色
  - Eureka Server：提供服务的注册于发现。
  - Service Provider：将自身服务注册到Eureka中，从而使消费方能够找到。
  - Service Consumer：服务消费方从Eureka中获取注册服务列表，从而找到消费服务。

目前架构状态。



SpringCloud-Consumer-dept-80：服务消费者

SpringCloud-provider-dept-8001：服务提供者

因此：我们需要创建一个提供注册与发现的服务

演示详解：

一、创建提供注册与发现的服务模块

SpringCloud-Consumer-dept-80：服务消费者

SpringCloud-provider-dept-8001：服务提供者

SpringCloud-EurEka-Server-8002：提供注册与发现的服务

1. 导入依赖

```
<dependencies>
 <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-eureka-server -->
 <dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-eureka-server</artifactId>
 <version>1.4.7.RELEASE</version>
 </dependency>
 <!--热部署-->
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  </dependency>
</dependencies>
##注意到点：版本对应问题，Springboot最好使用2.2.x的
```

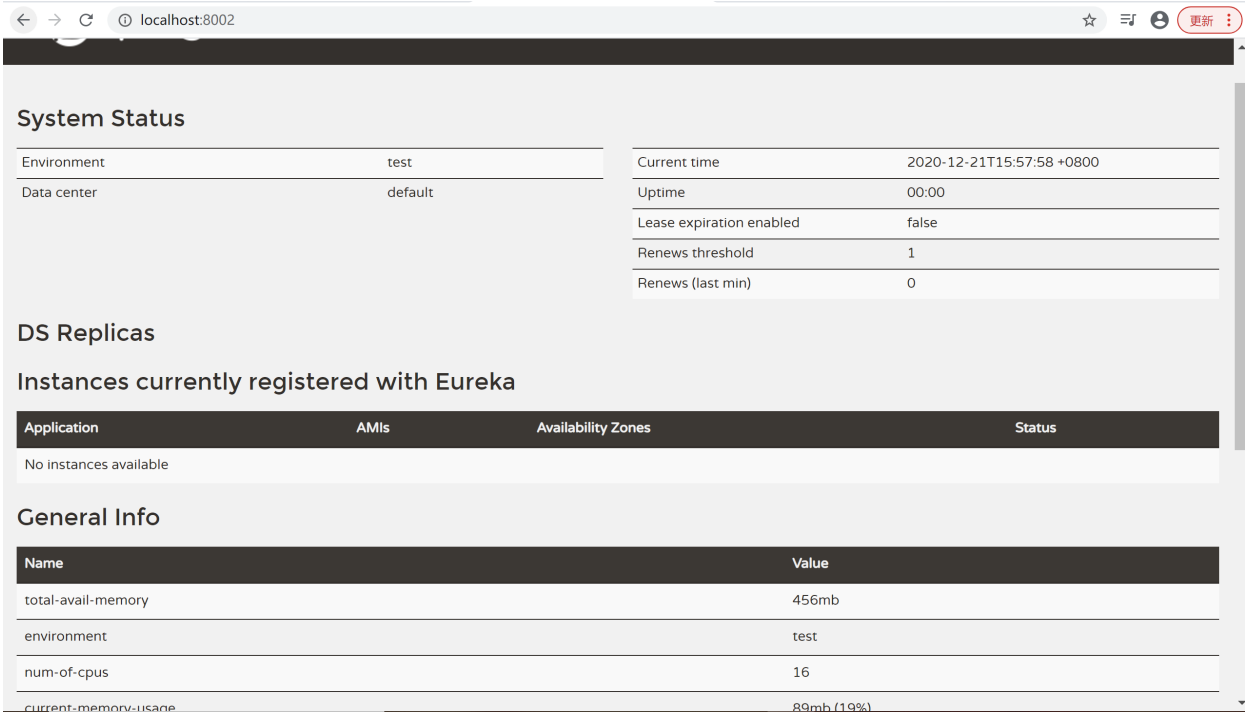
2. 编写配置文件 application.yml

```
#配置端口
server:
 port: 8002

#Eureka配置
eureka:
 instance:
 hostname: localhost #Eureka服务器实例名称
 client:
  register-with-eureka: false #表示是否向Eureka注册自己，既然是server提供注册与发现 所以自己不注册的
  fetch-registry: false #fetch-registry如果为false，则表示自己为注册中心
  service-url: #监控页面 我们访问的 提供监控的页面
  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

3. 开启功能

```
@SpringBootApplication
@EnableEurekaServer //EnableEurekaServer服务端的启动类，可以接受别人注册进来
public class SpringCloudEurekaServer_8002 {
 public static void main(String[] args) {
 SpringApplication.run(SpringCloudEurekaServer_8002.class,args);
  }
}
```

访问监控页面测试，是否开启注册与发现成功！



4. 配置类的实现。（需要的情况）

二、提供服务到注册中心，供消费者查询

SpringCloud-provider-dept-8001：提供服务到注册中心，供消费者查询

1. 导入依赖

```
##在原先的基础上加上eureka客户端和 eureka -->完善监控信息（看个人需求，一般团队开发需要用到监控信息）
<!--eureka客户端-->
<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-netflix-eureka-client -->
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
 <version>2.2.5.RELEASE</version>
</dependency>
<!--eureka -->完善监控信息-->
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

2. 编写配置文件

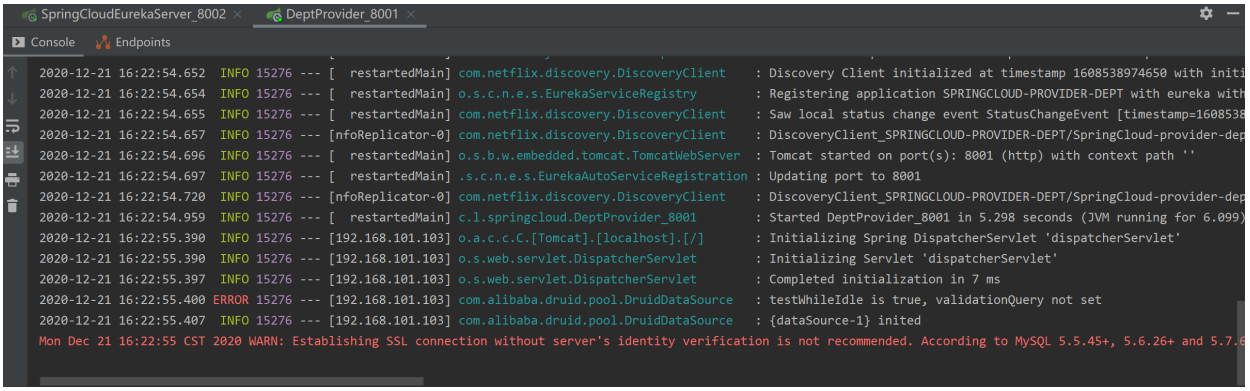
```
#Eureka配置 我们应该将服务提供到哪里去（注册中心）
eureka:
```

```
  client:
  service-url:
  defaultZone: http://localhost:8002/eureka/
  instance: #（这里是下面图示）
  instance-id: SpringCloud-provider-dept-8001 #可以通过这个任意修改Eureka注册实例显示的status信息

#Info配置 也就是监控信息 （这里是下面图示）
info:
 app.name: lidadaibiao
 app.email: 945827167@qq.com
```

3. 开启功能

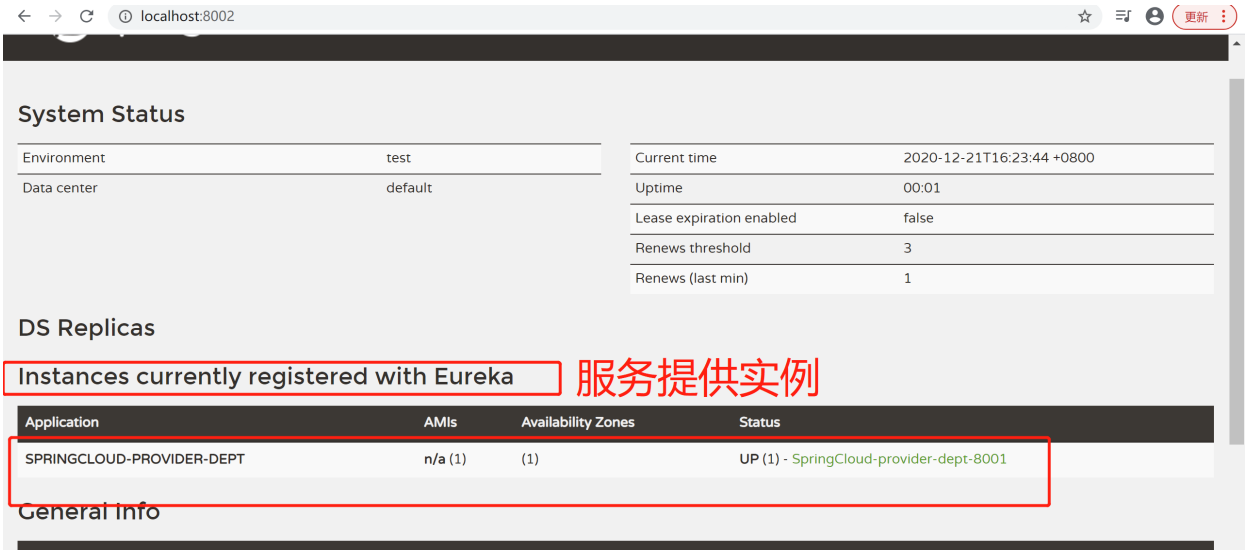
```

@SpringBootApplication
@EnableEurekaClient //在服务启动后自动注册到Eurake中 ---》springcloud-eurake-7001
@EnableDiscoveryClient //服务发现，针对团队开发 下面会记载到。
public class DeptProvider_8001 {
 public static void main(String[] args) {
 SpringApplication.run(DeptProvider_8001.class,args);
 }
}
```

先启动SpringCloud-EurEka-Server-8002：注册与发现 再启动 SpringCloud-provider-dept-8001：提供服务到注册中心，供消费者查询

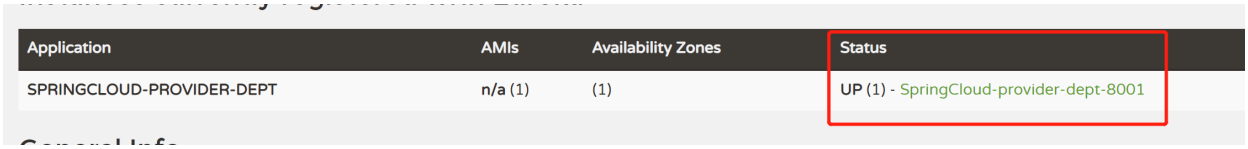


访问测试：

我们可以在监控页面进行服务提供实例查看




```yaml
instance:
  instance-id: SpringCloud-provider-dept-8001 #可以通过这个任意修改Eureka注册实例显示的status信息
```



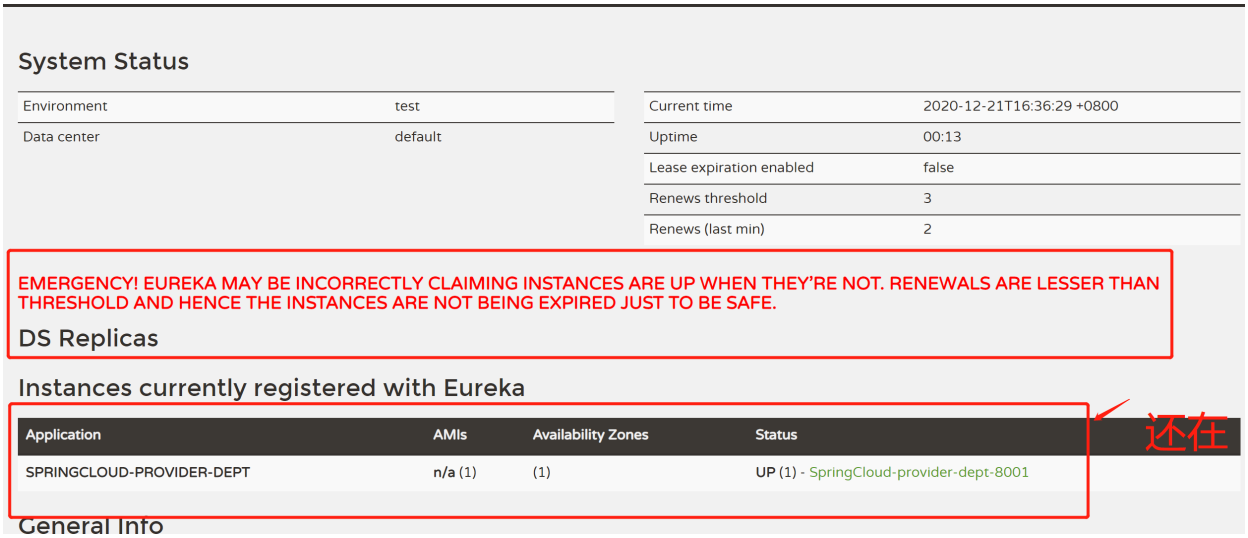
当我们点击进去查看info的时候，会显示我们所配置的监控信息

```yaml
#Info配置 也就是监控信息 （这里是下面图示）
info:
  app.name: lidadaibiao
  app.email: 945827167@qq.com
<!--eureka -->完善监控信息--> 看网址 actuator/info
  <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

4. 配置类的实现。（需要的情况下）

三、Eureka的自我保护机制

场景：当我们突然中断提供服务者，发现注册与发现的监控页面会出现以下情况。



这是Eureka的自我保护机制：

自我保护机制：好死不如赖活着

一句话总结：某时刻某一个微服务不可以用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存！

- 默认情况下，如果EurekaServer在一定时间内没有接收到某个微服务实例的心跳，EurekaServer将会注销该实例（默认90秒）。但是当网络分区故障发生时，微服务与Eureka之间无法正常通行，以上行为可能变得非常危险了--因为微服务本身其实是健康的，**此时本不应该注销这个服务**。Eureka通过 **自我保护机制** 来解决这个问题--当EurekaServer节点在短时间内丢失过多客户端时（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，EurekaServer就会保护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）。当网络故障恢复后，该EurekaServer节点会自动退出自我保护模式。
- 在自我保护模式中，EurekaServer会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该EurekaServer节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话：好死不如赖活着
- 综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮和稳定
- 在SpringCloud中，可以使用`eureka.server.enable-self-preservation = false` 禁用自我保护模式【不推荐关闭自我保护机制】

自我保护机制：好死不如赖活着

一句话总结：某时刻某一个微服务不可以用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存！

- 默认情况下，如果EurekaServer在一定时间内没有接收到某个微服务实例的心跳，EurekaServer将会注销该实例（默认90秒）。但是当网络分区故障发生时，微服务与Eureka之间无法正常通行，以上行为可能变得非常危险了--因为微服务本身其实是健康的，**此时本不应该注销这个服务**。Eureka通过 **自我保护机制** 来解决这个问题--当EurekaServer节点在短时间内丢失过多客户端时（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，EurekaServer就会保护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）。当网络故障恢复后，该EurekaServer节点会自动退出自我保护模式。
- 在自我保护模式中，EurekaServer会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该EurekaServer节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话：好死不如赖活着
- 综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮和稳定
- 在SpringCloud中，可以使用`eureka.server.enable-self-preservation = false` 禁用自我保护模式【不推荐关闭自我保护机制】

四、Eureka的服务发现（Discovery）

- 对于注册到Eureka里面的微服务，我们可以通过服务发现来获得该服务信息(向外暴露服务信息)
- 编写我们提供服务到注册中心的，提供服务者Controller层进行服务发现

```
@Autowired
private DiscoveryClient discoveryClient; #注入服务发现客户端 import org.springframework.cloud.client.discovery.DiscoveryClient;
@GetMapping("/eurake/discovery/")
public Object discovery(){
 //获得微服务列表的清单
 List<String> services = discoveryClient.getServices();
 System.out.println("微服务列表清单："+services);
 //得到一个具体的，微服务实例信息 getInstances(String serviceId); serviceId 服务ID
 List<ServiceInstance> serviceInstanceList = discoveryClient.getInstances("SPRINGCLOUD-PROVIDER-DEPT");
  for (ServiceInstance serviceInstance : serviceInstanceList) {
  System.out.println("URI:"+serviceInstance.getUri());
  System.out.println("port:"+serviceInstance.getPort());
  System.out.println("host:"+serviceInstance.getHost());
  System.out.println("ServiceId:"+serviceInstance.getServiceId());
  }
  return this.discoveryClient;
}
```

- 需要在启动类中开启服务发现

```
@SpringBootApplication
@EnableEurekaClient //在服务启动后自动注册到Eurake中 --》springcloud-eurake
-7001
@EnableDiscoveryClient //开启服务发现，针对团队开发
public class DeptProvider_8001 {
 public static void main(String[] args) {
 SpringApplication.run(DeptProvider_8001.class,args);
 }
}
```

测试：