Spring Cloud 项目建立

* 开发环境：jdk1.8
* 开发工具:IDEA

参考文章：

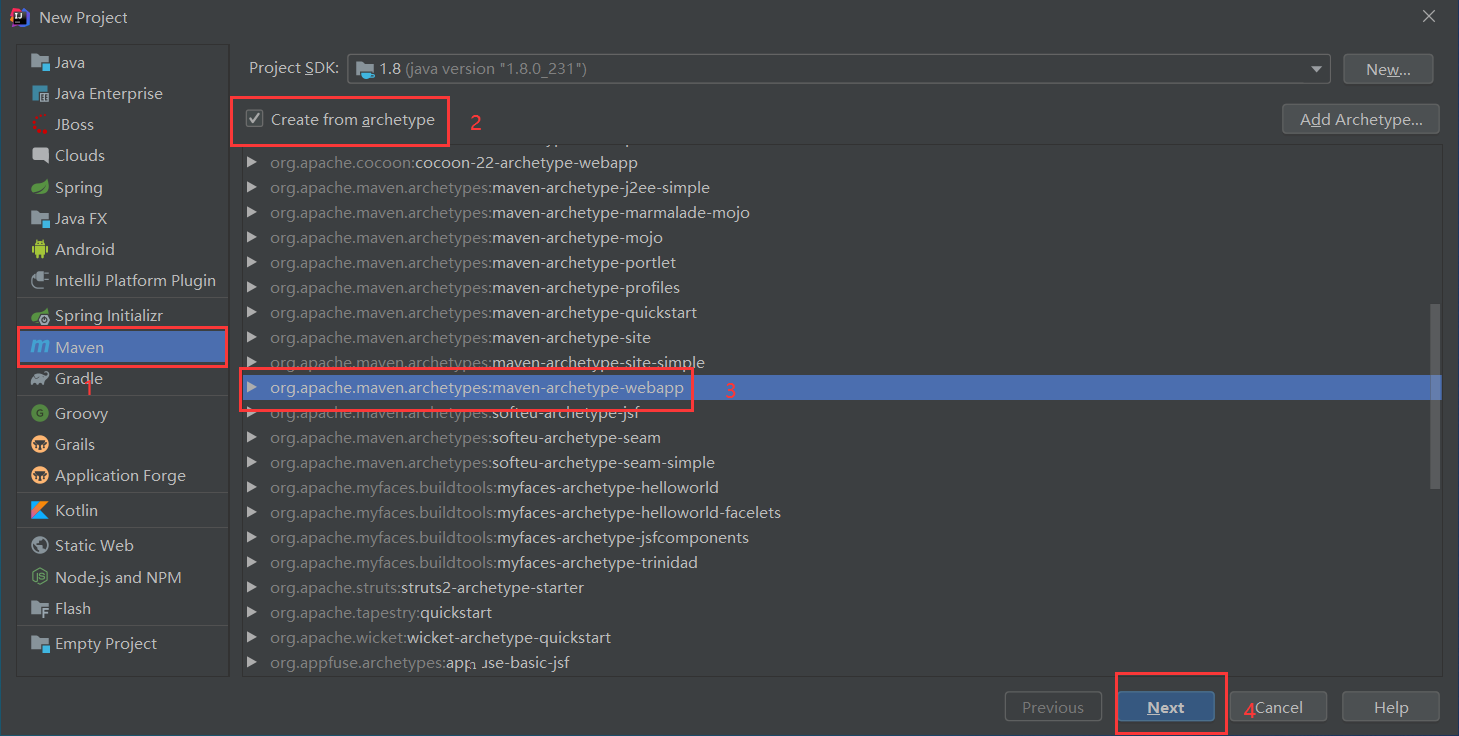
<https://juejin.im/post/5b83466b6fb9a019b421cecc>（了解SpringCloud）

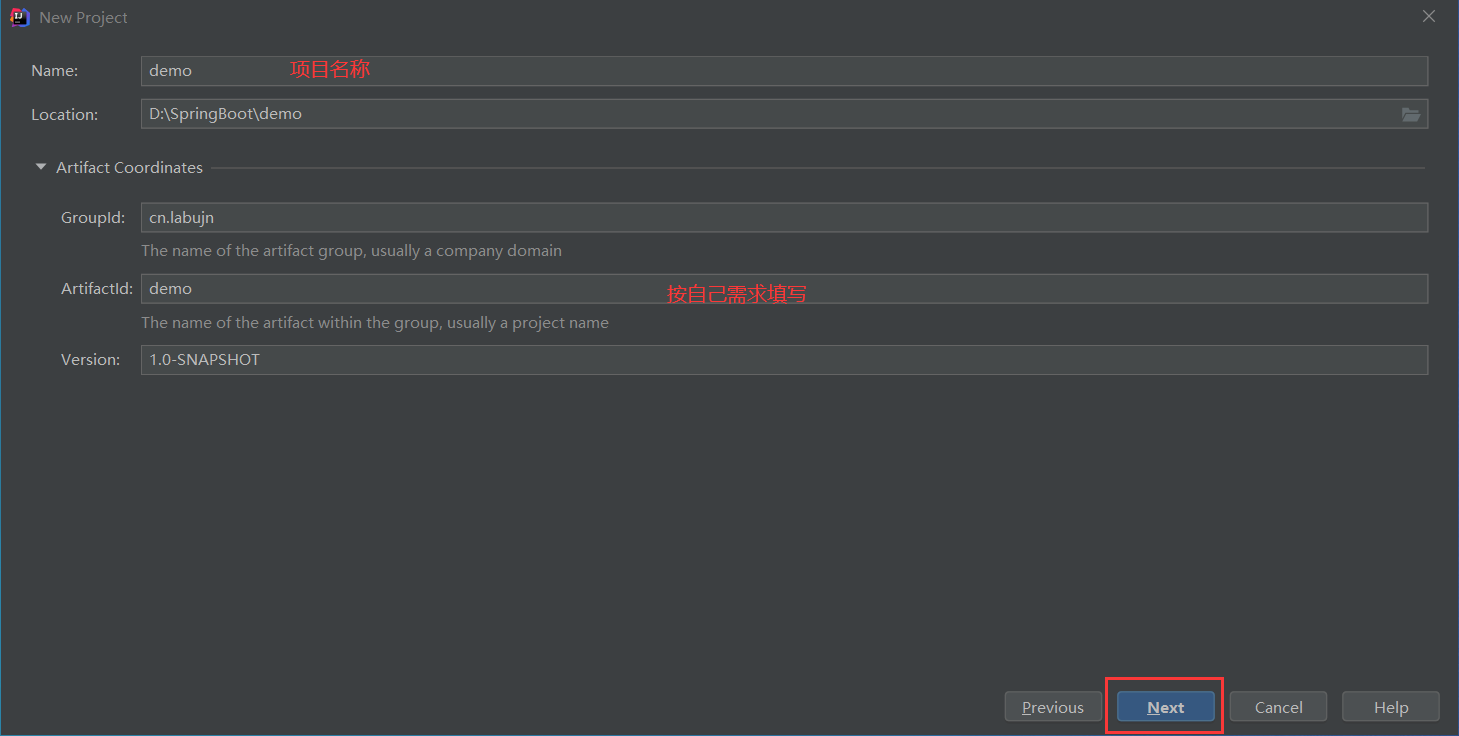
<https://www.fangzhipeng.com/spring-cloud.html>

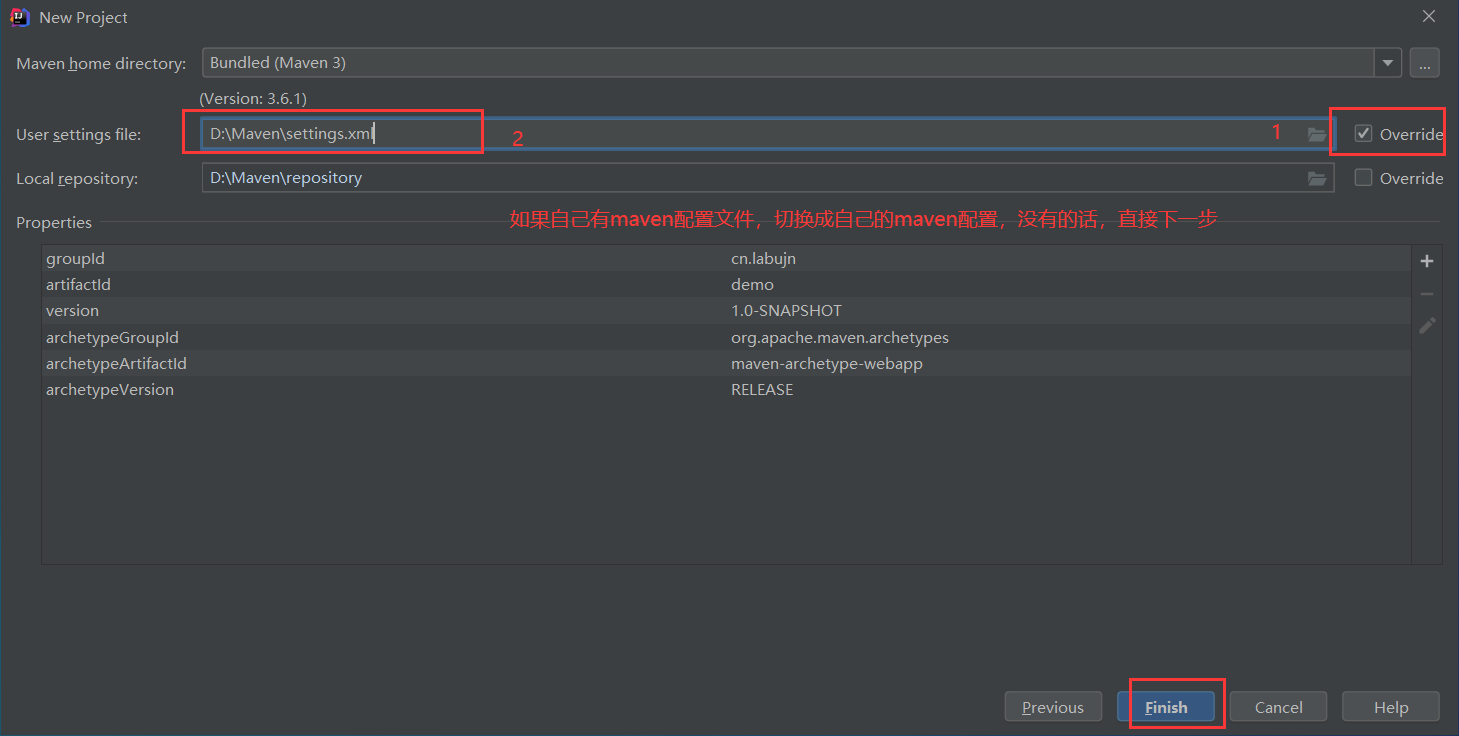
1. 如何建立一个Spring Cloud项目

**(实例代码的git Tag：tag\_2020\_01\_01)**

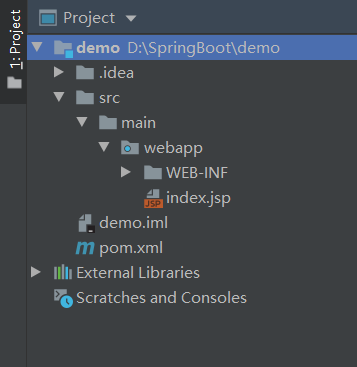
1. 建立一个空Maven项目





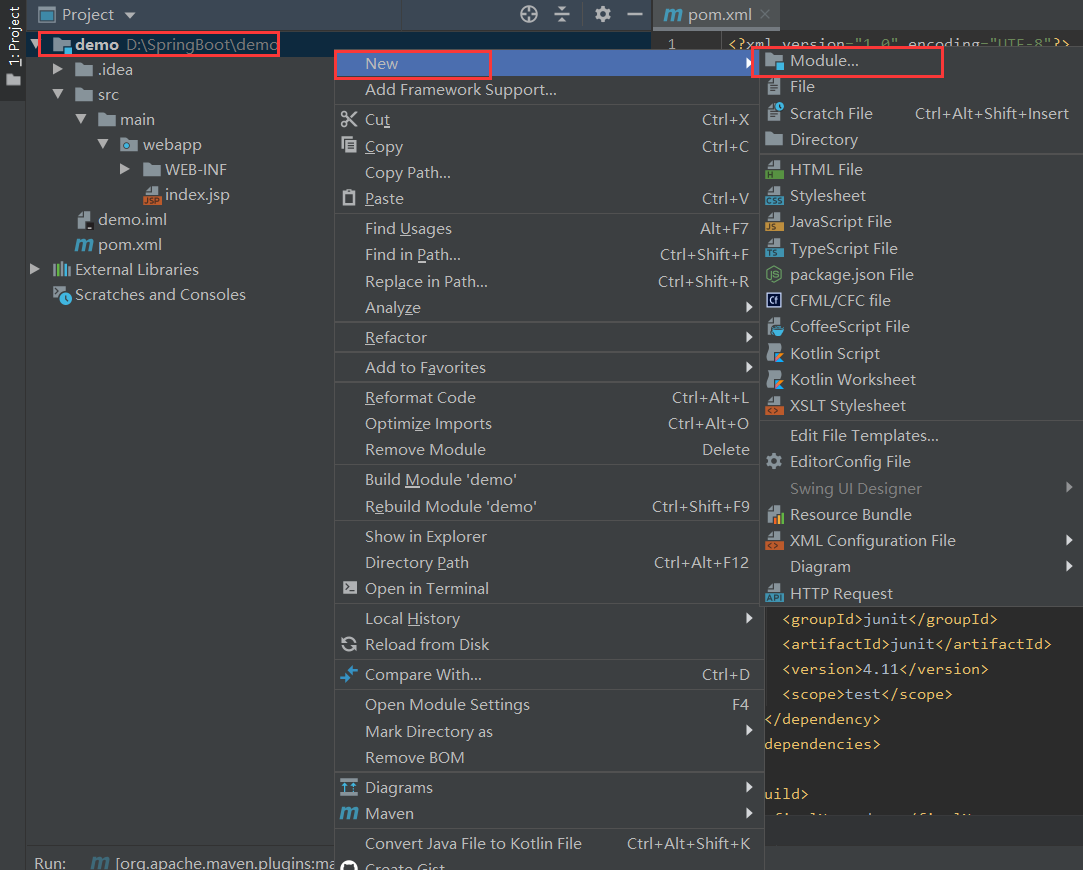


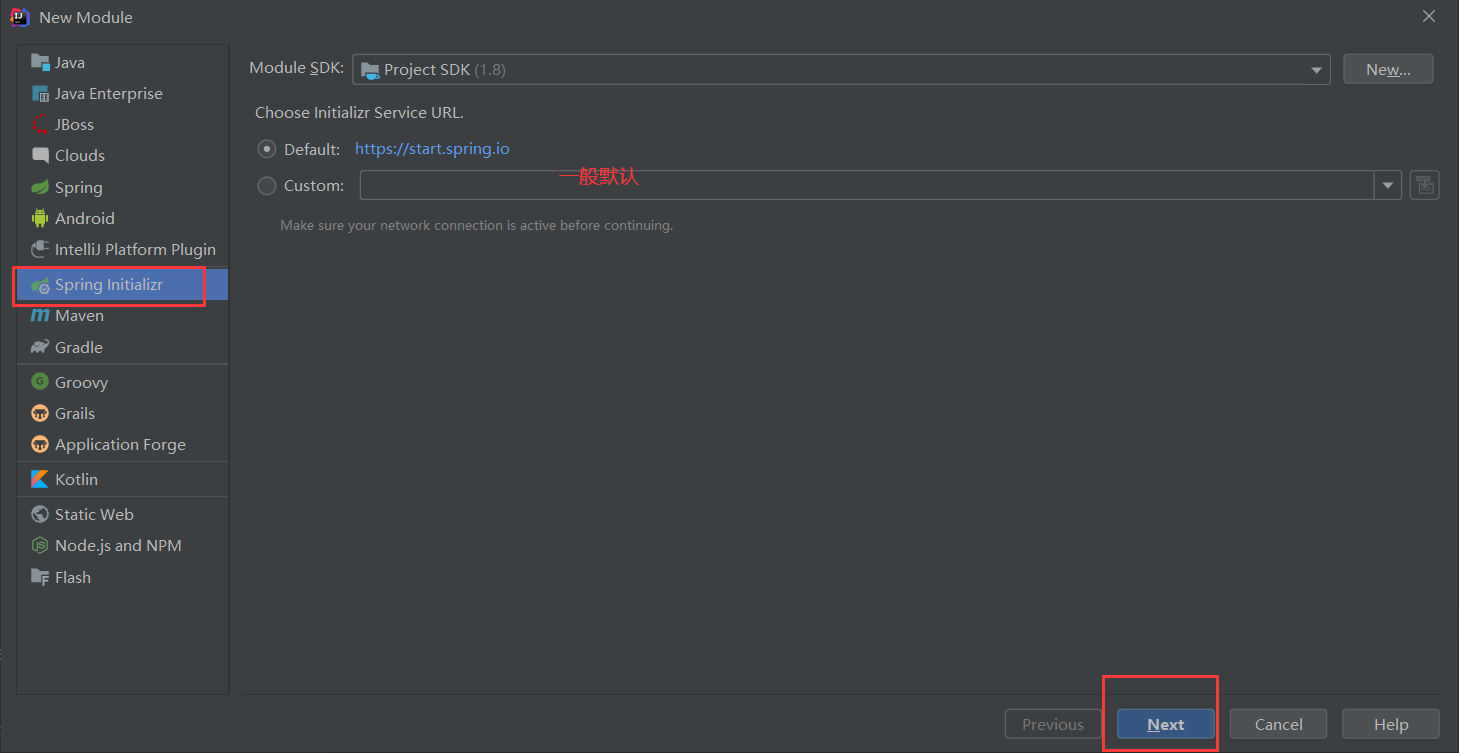
项目目录结构展示：

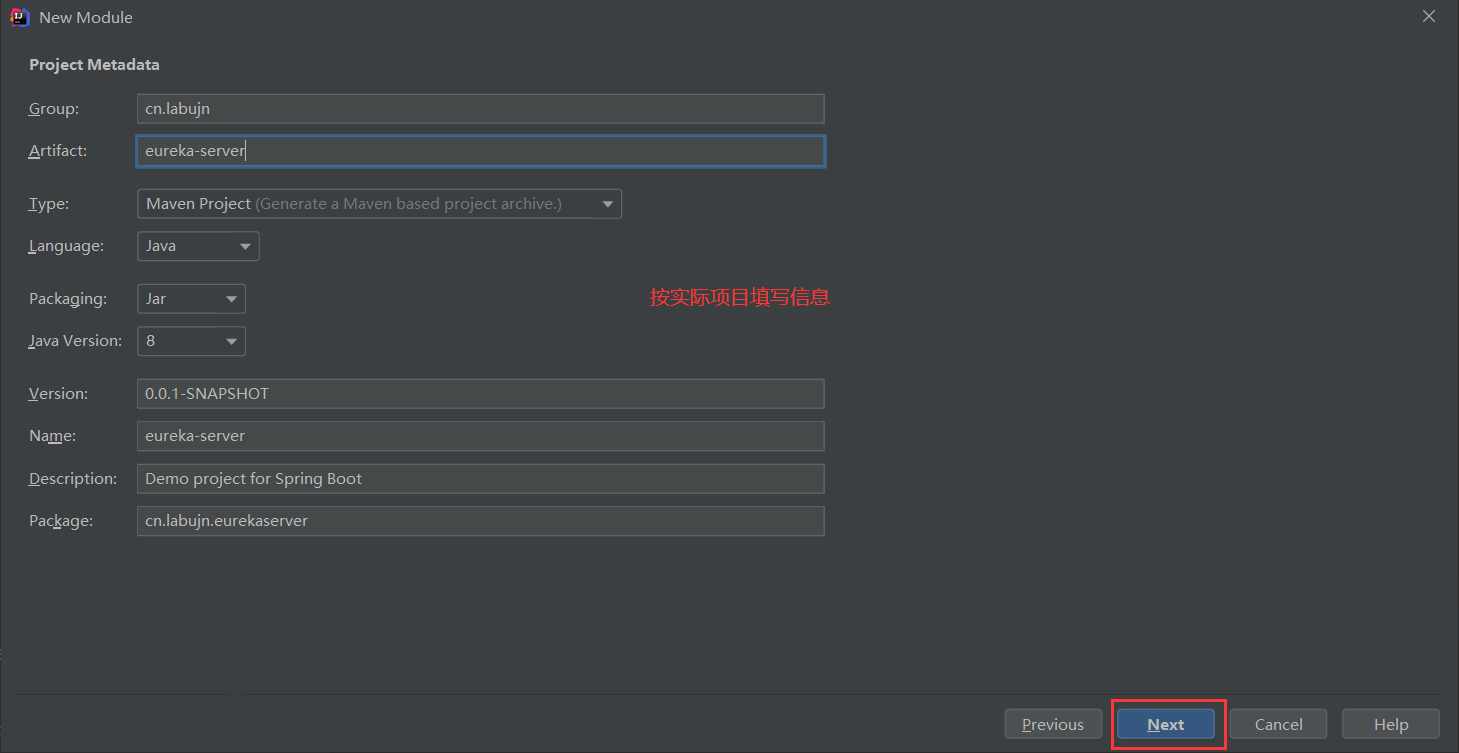


1. 建立服务注册中心（Eureka Server）

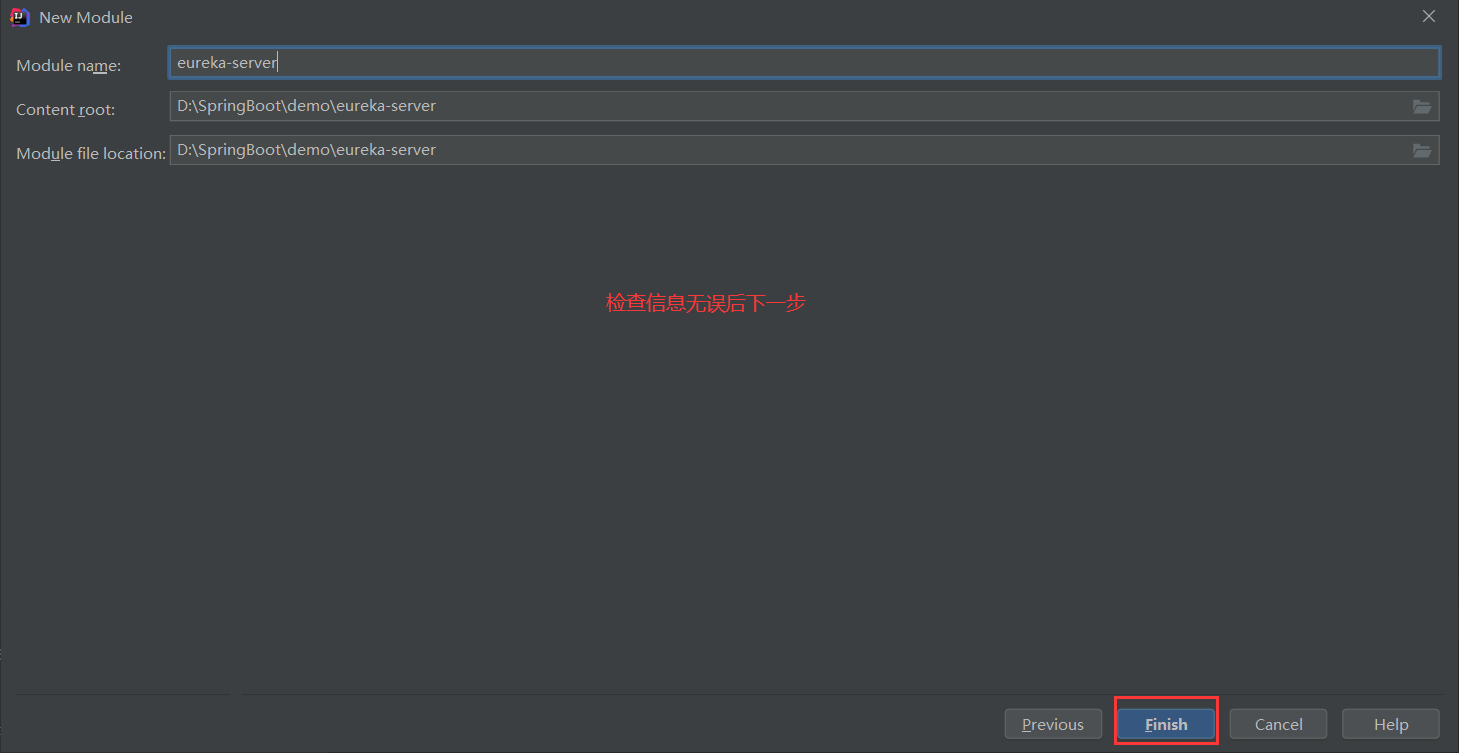
在maven下新建module

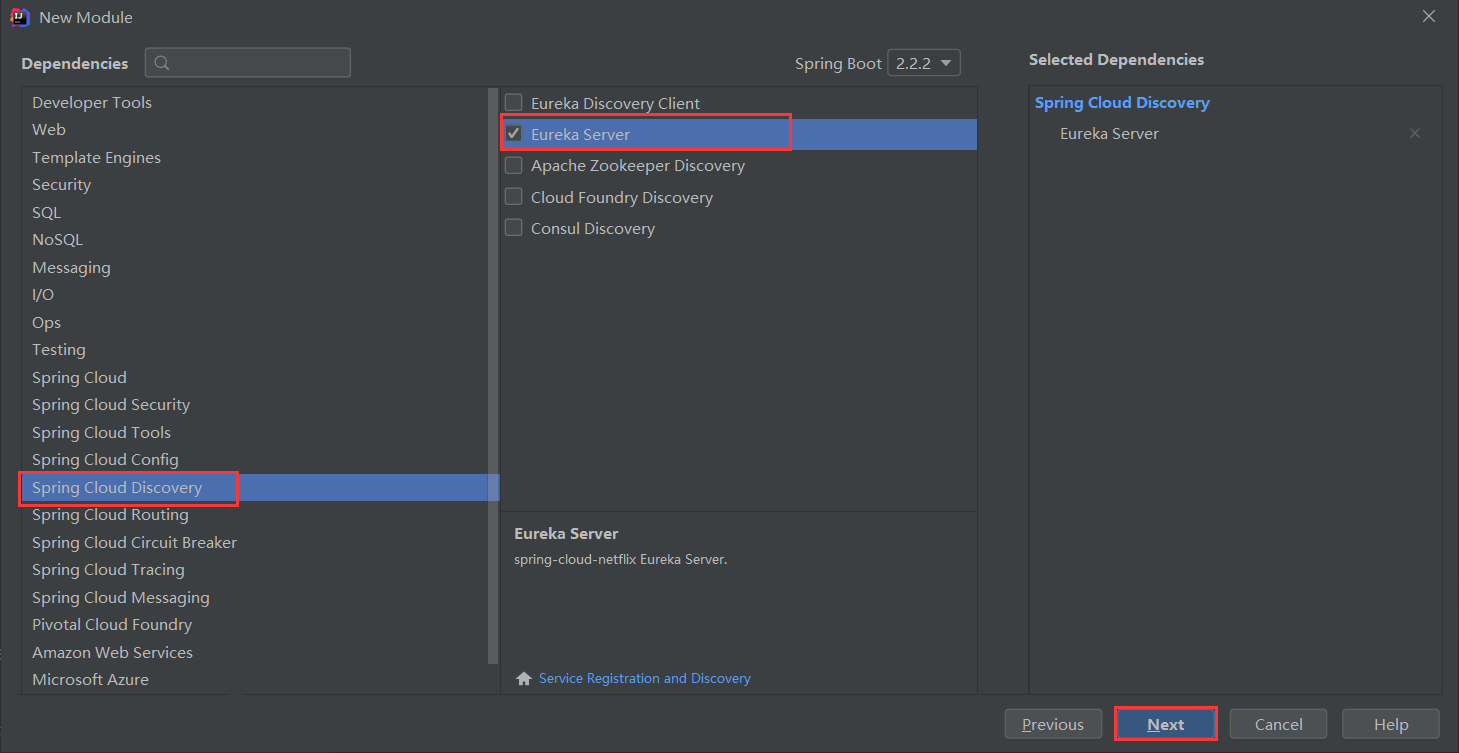




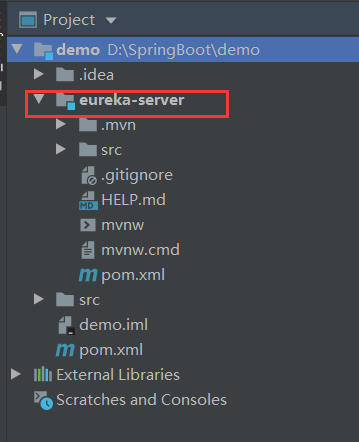


选择Eureka Server





此时项目目录结构如图所示：

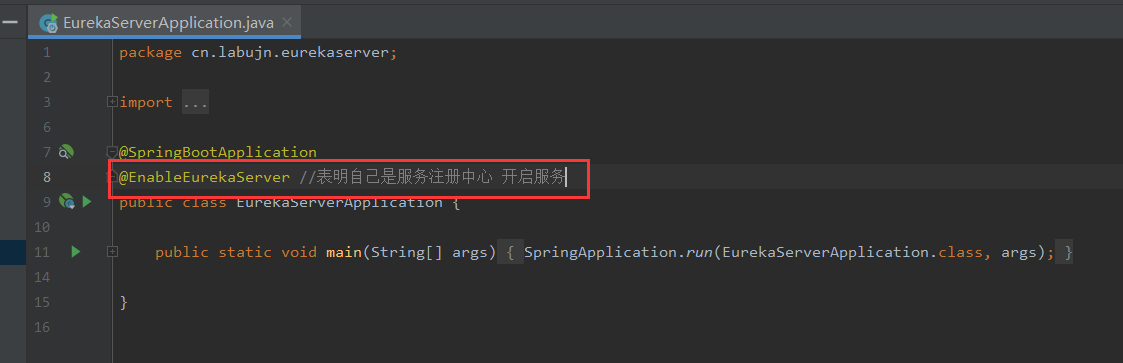


**下面开始配置Eureka Server**

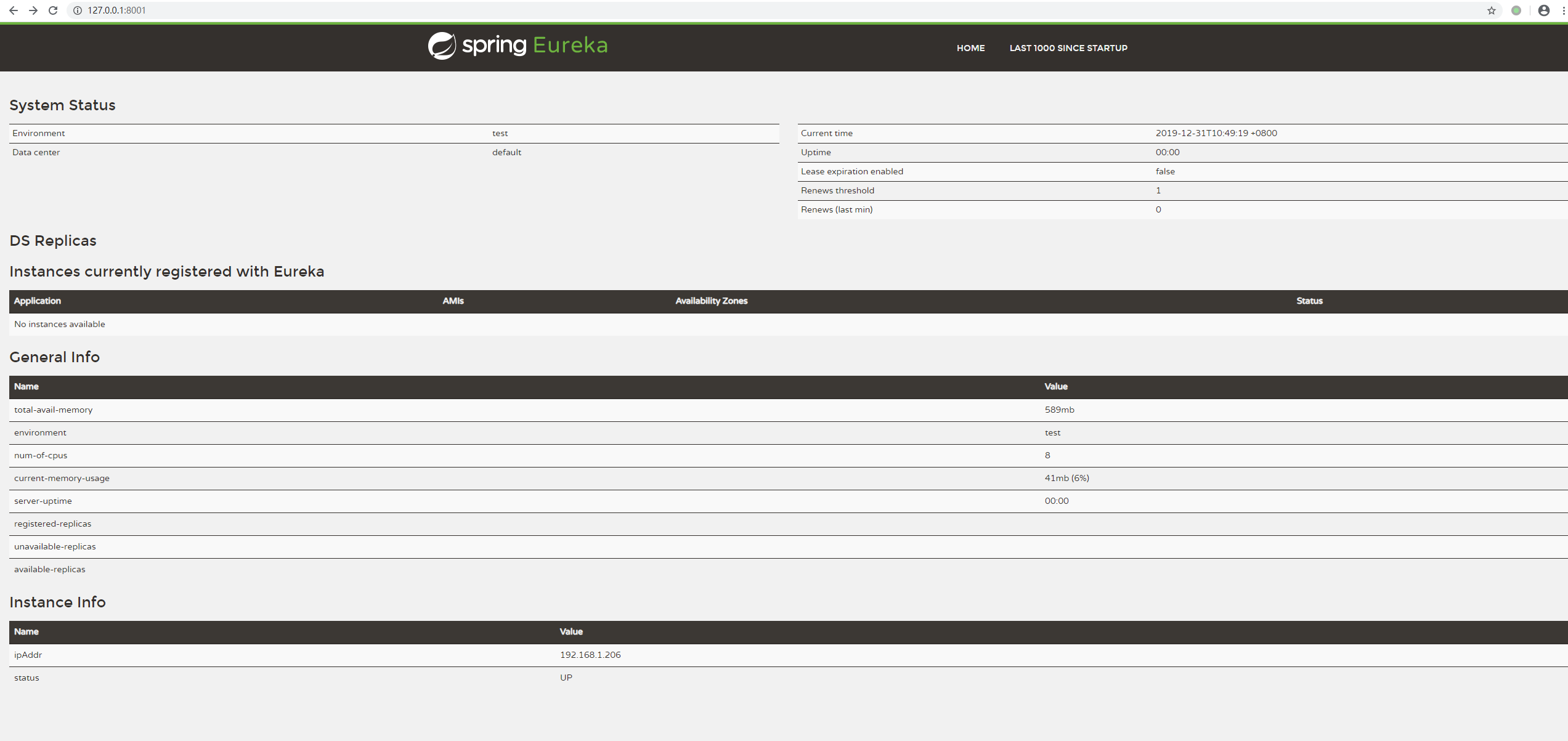
Application.properties文件配置：



项目启动文件添加注解@EnableEurekaServer 开启服务注册中心



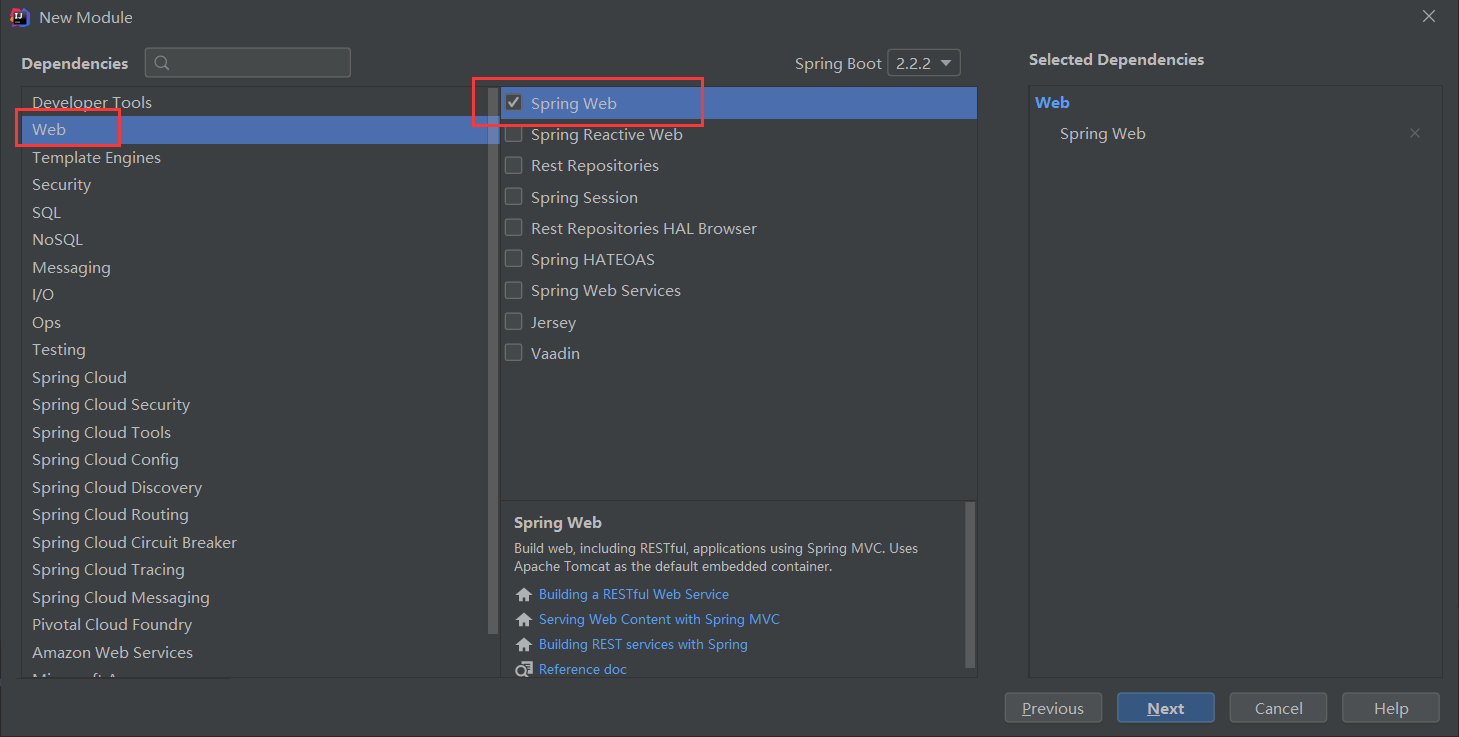
运行项目，访问自己定义的服务端口，看到如下界面代表服务注册中心已经配置完成

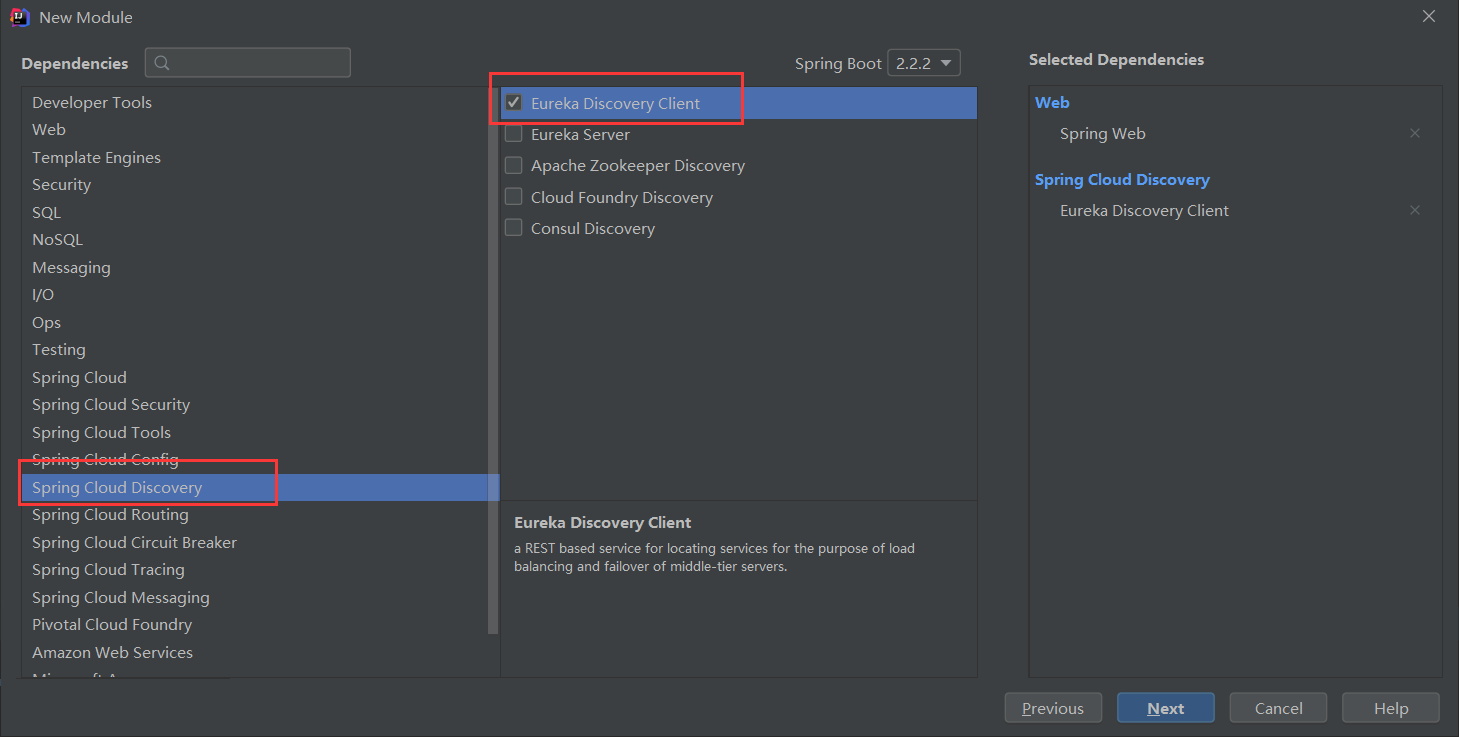


1. 新建服务生产者（Eureka Client）

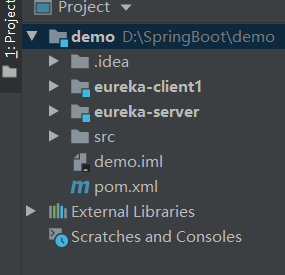
新建服务生产者与建立服务注册中心步骤一致，只是配置不同

**要勾选 Eureka Discover Client 与 springboot 的web组件,其余组件按需求勾选，与建立springboot项目相同**





此时项目目录结构如图所示：

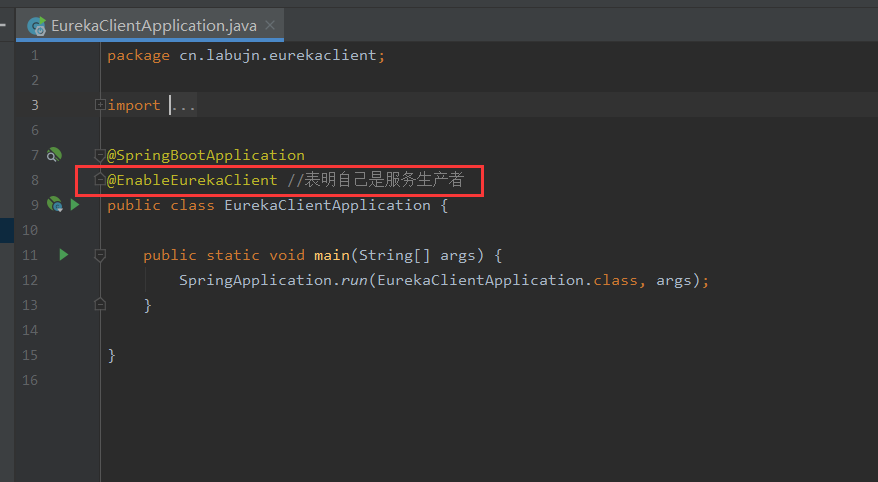


**配置步骤：**

Application.properties文件配置：

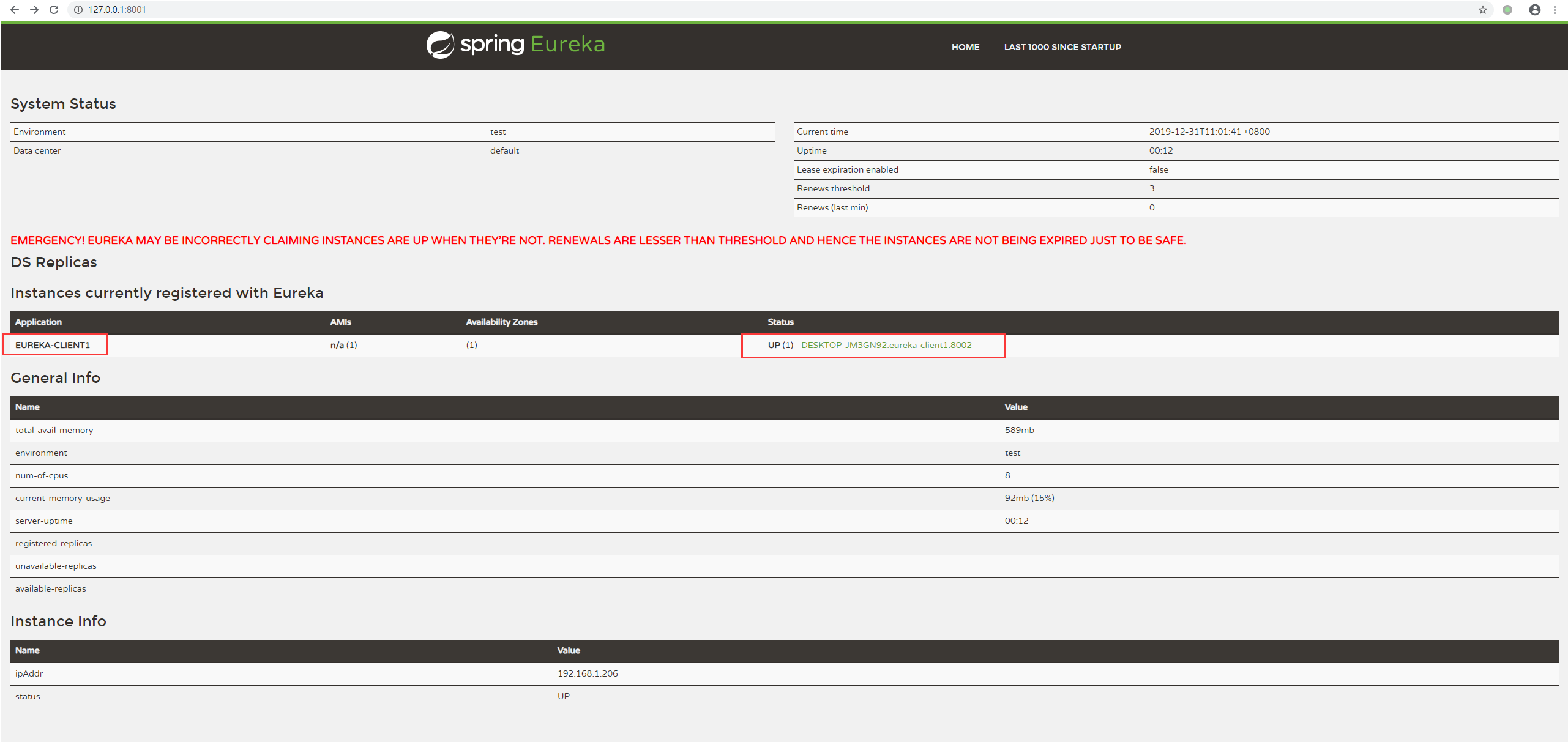


项目启动文件添加注解@ EnableEurekaClient开启服务生产者



运行服务生产者，自己会向服务注册中心注册自己（需要等几秒钟）

此时打开服务注册中心，就会看到自己新建的服务了



应用名称就是自己配置的应用名

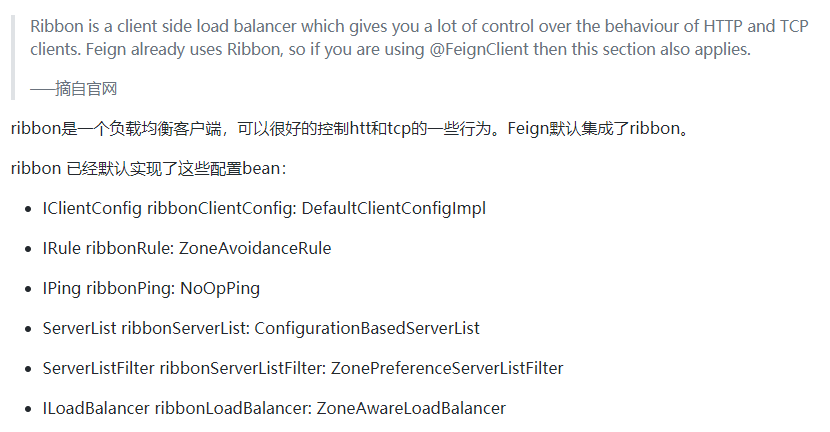
**服务生产者可以按需要配置多个，每一个服务生产者都是一个独立的springboot项目**

服务调用方式

1. ribbon+restTemplate的服务调用方式

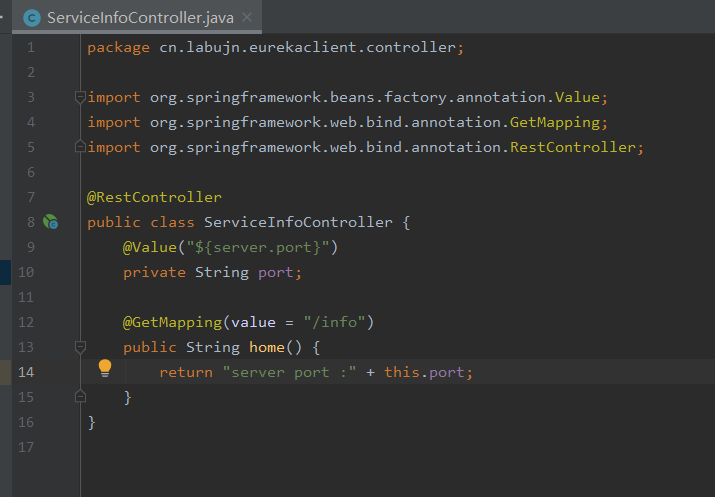
**(实例代码的git Tag：tag\_2020\_01\_01)**

* 1. ribbon简介



* 1. 准备工作
* 首先在服务生产者中写一个服务：

功能：返回当前服务生产者的端口

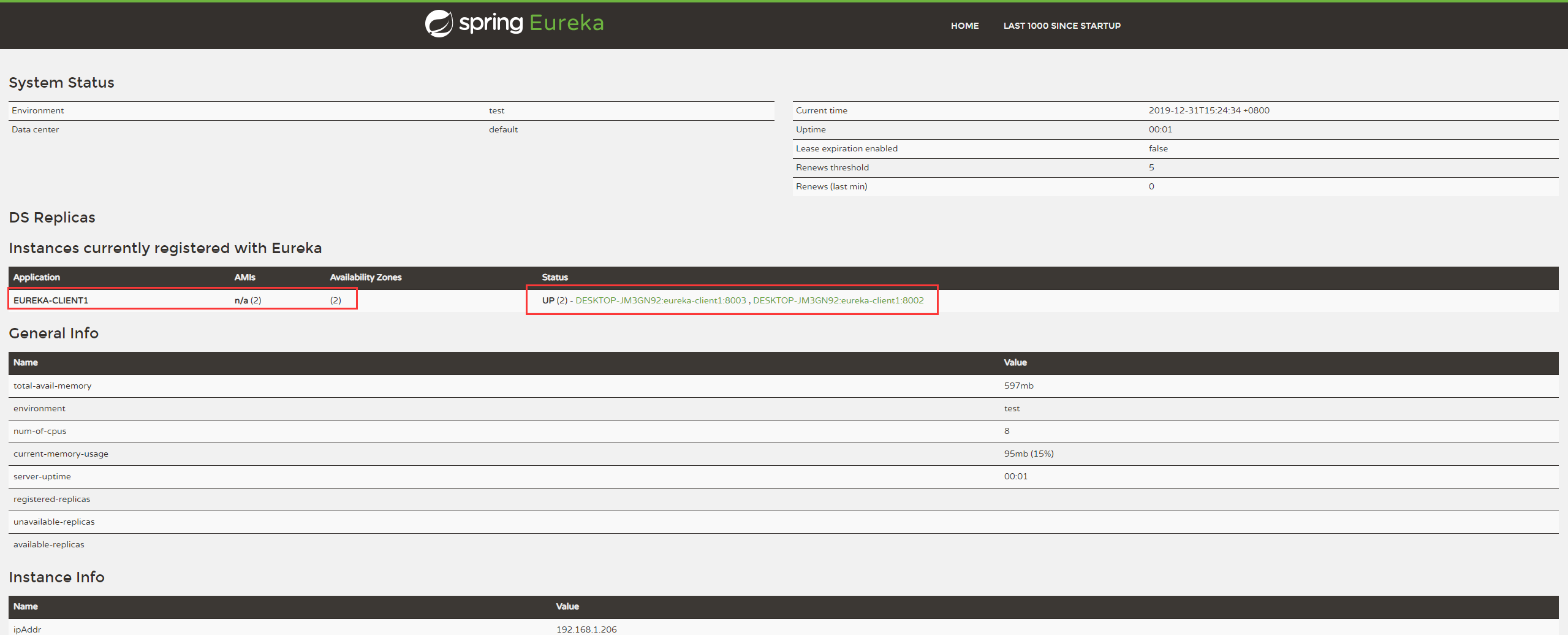


* 启动服务

先启动服务注册中心（eureka server 工程），再启动服务生产者（eureka client1端口为8002），再在配置文件中把端口改为8003，再启动，此时eureka client1会在服务注册中心，注册两个实例，就相当于一个小集群了。

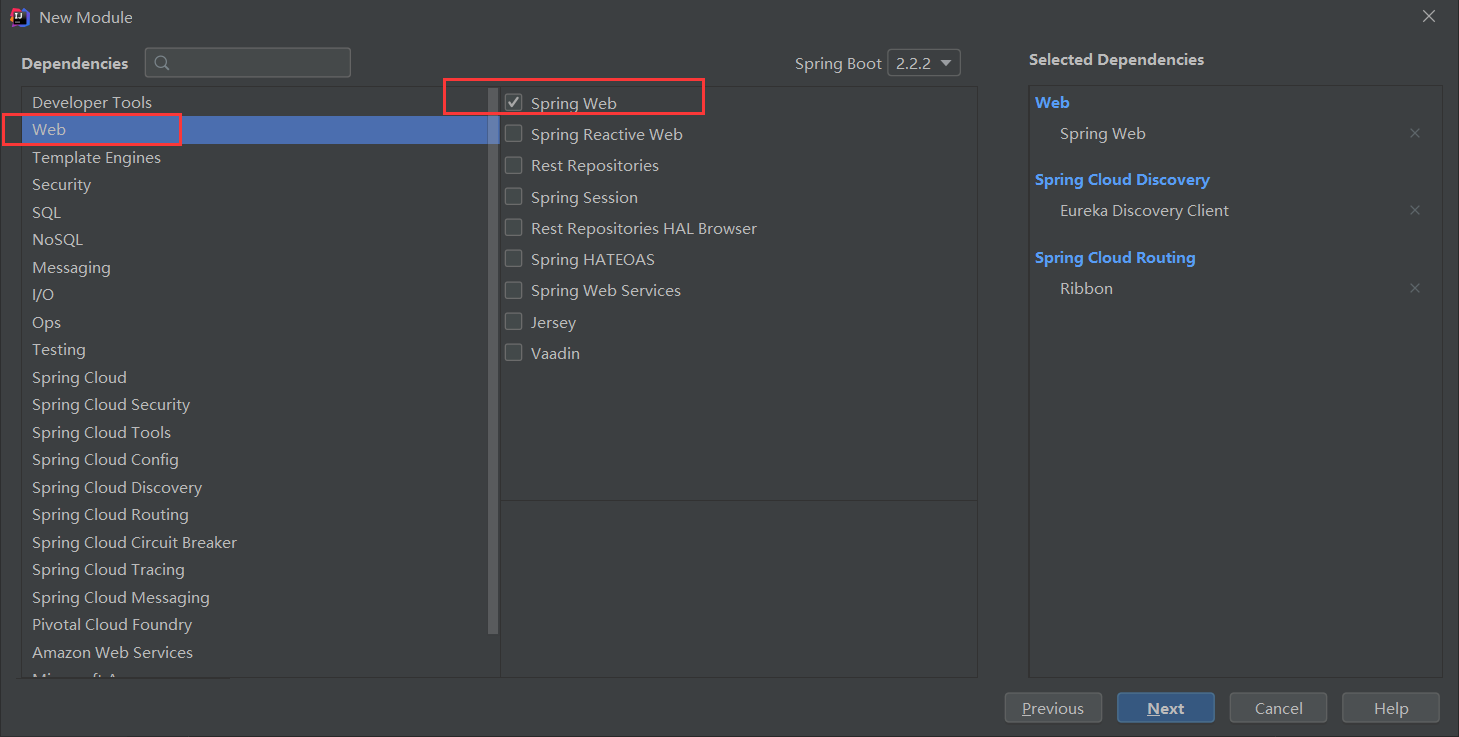
Tip:用idea无法把同一个工程重复启动，可以使用Terminal工具，首先将命令行位置跳转到当前项目的根目录下（demo/erureka-client1），再输入“mvnw spring-boot:run”命令，初次操作maven需要下载插件等待几分钟(mvnw 是项目根目录下的mvnw.cmd,名字可能会变化，请按实际情况自行替换)

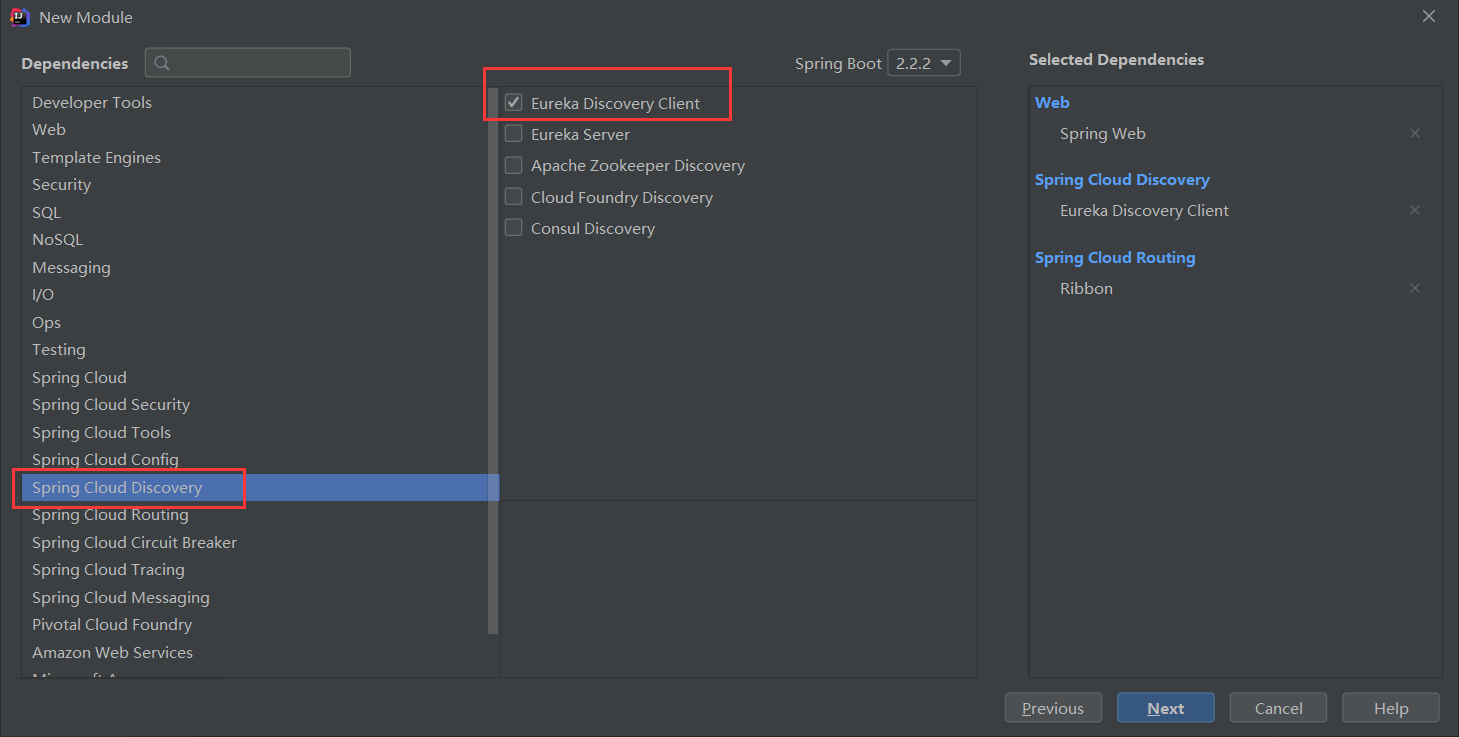
服务注册中心，此时截图如图所示：

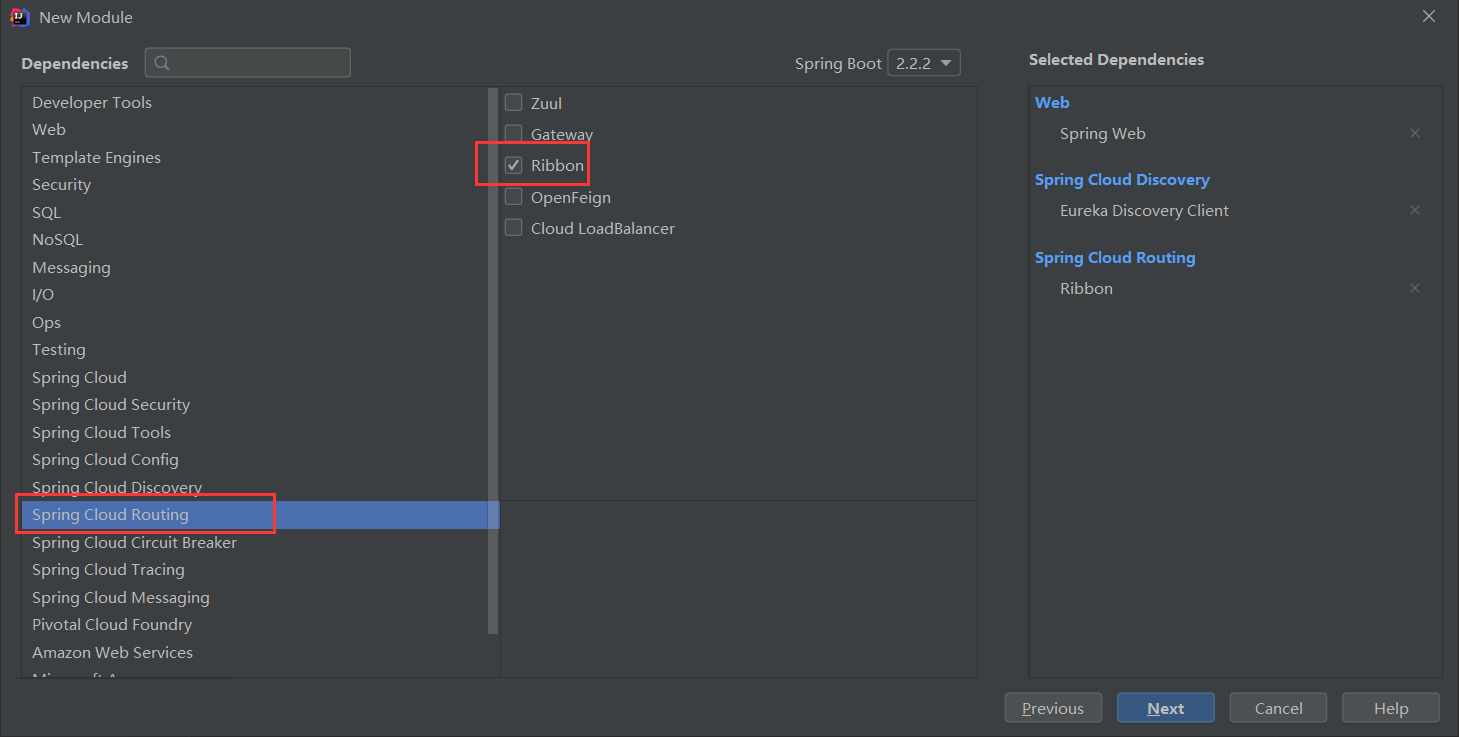


* 1. 创建服务消费者

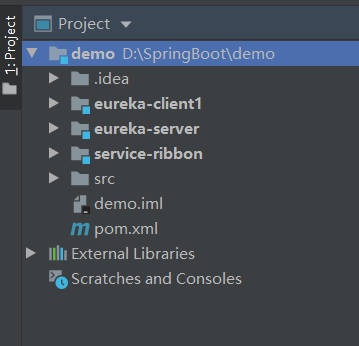
创建步骤跟创建服务者一致，需要勾选ribbon、Eureka Discover Client 与 springboot 的web组件,其余组件按需求勾选，与建立springboot项目相同



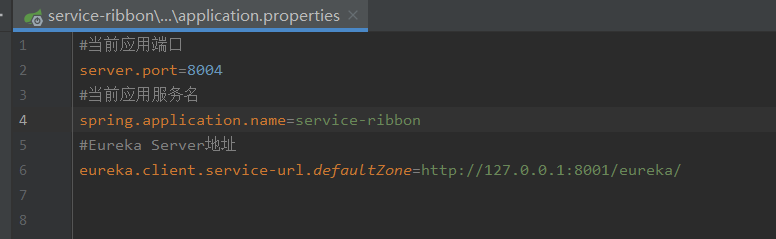




此时项目结构目录如图所示

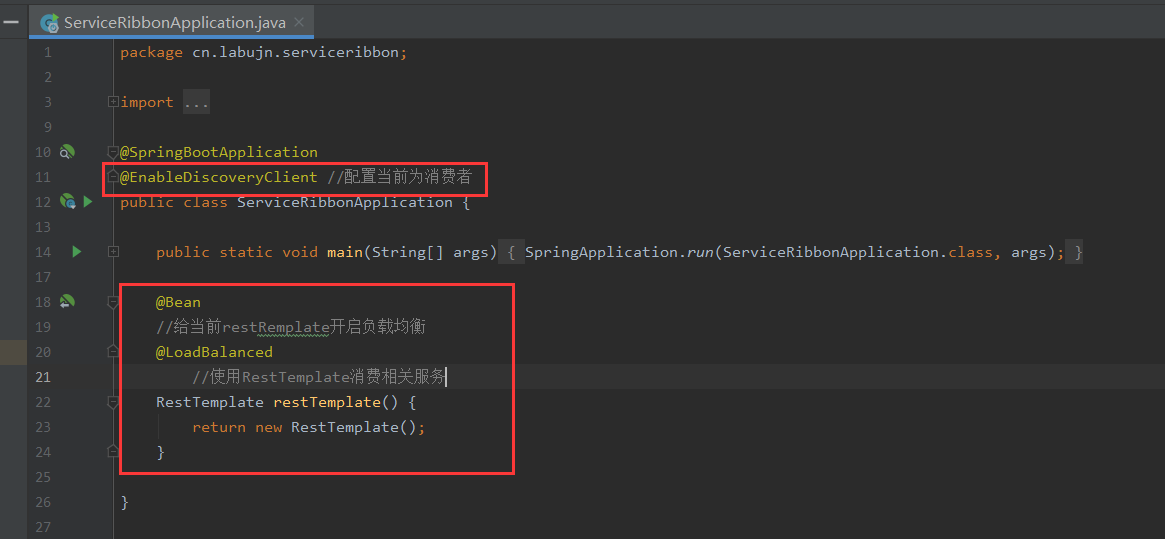


**配置方式：**



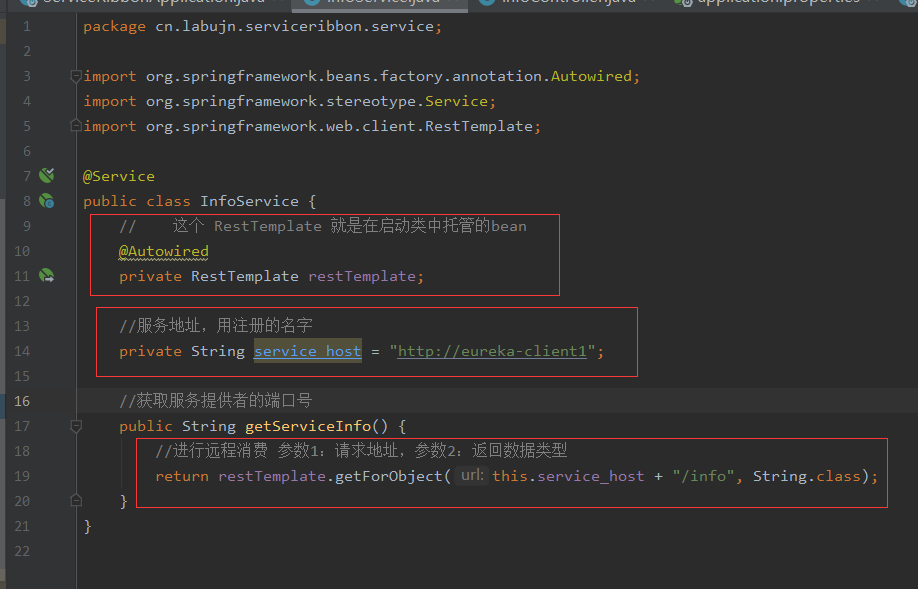
**启动文件：**

**把RestTemplate注册到ICO容器中，供后面消费使用**

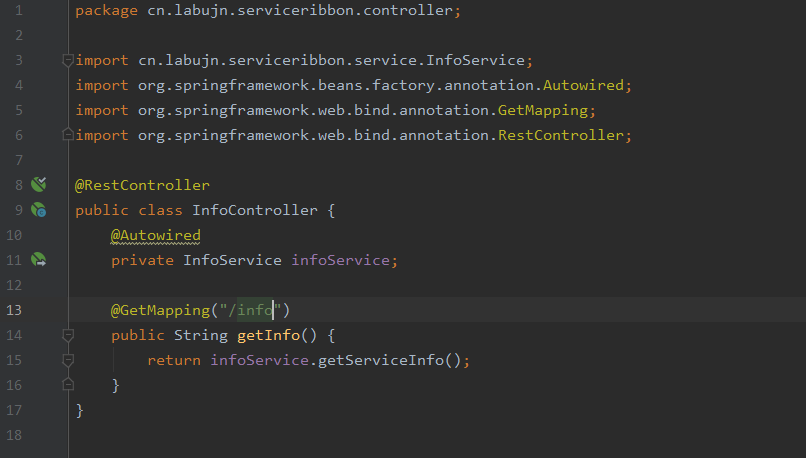


* 1. **消费远程服务**

新建service层，进行远程消费

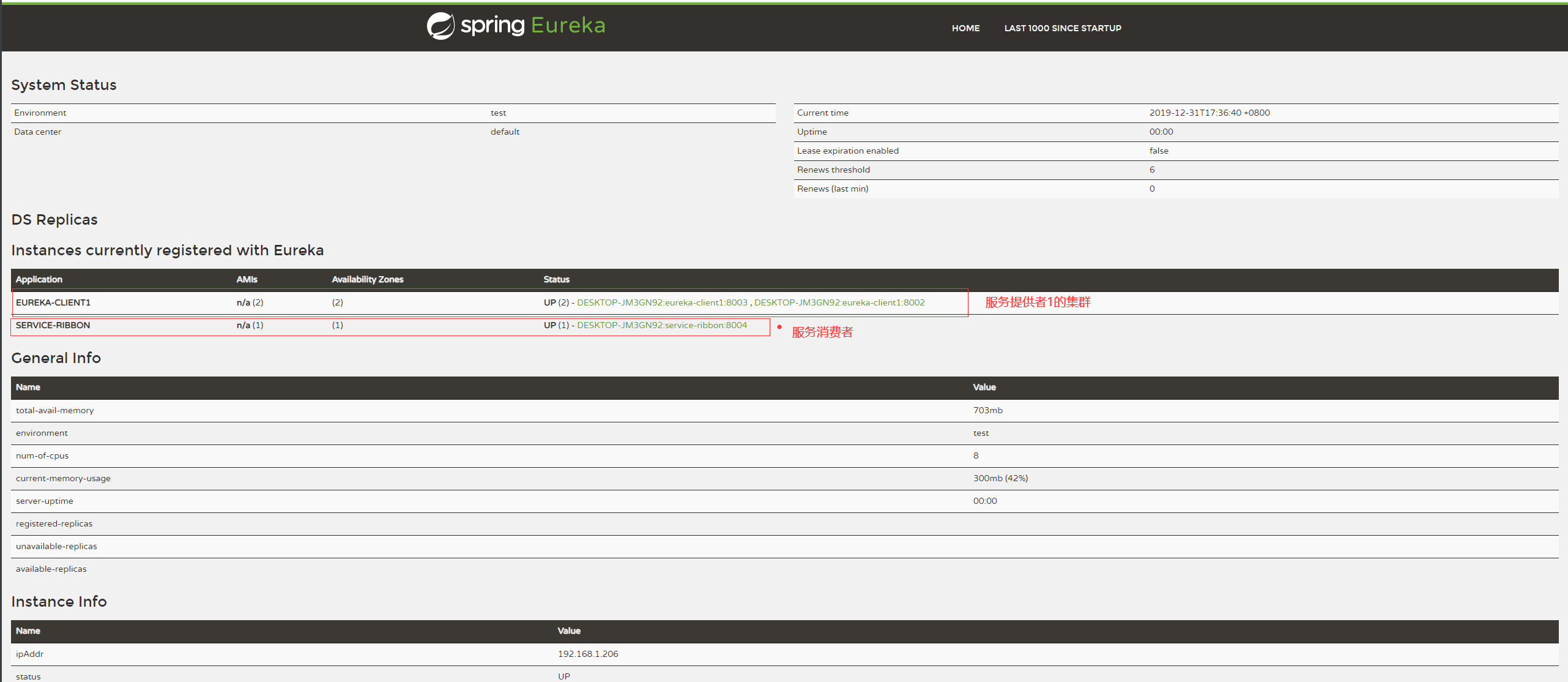


新建controller层

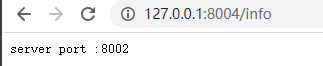
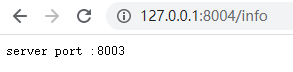


启动服务

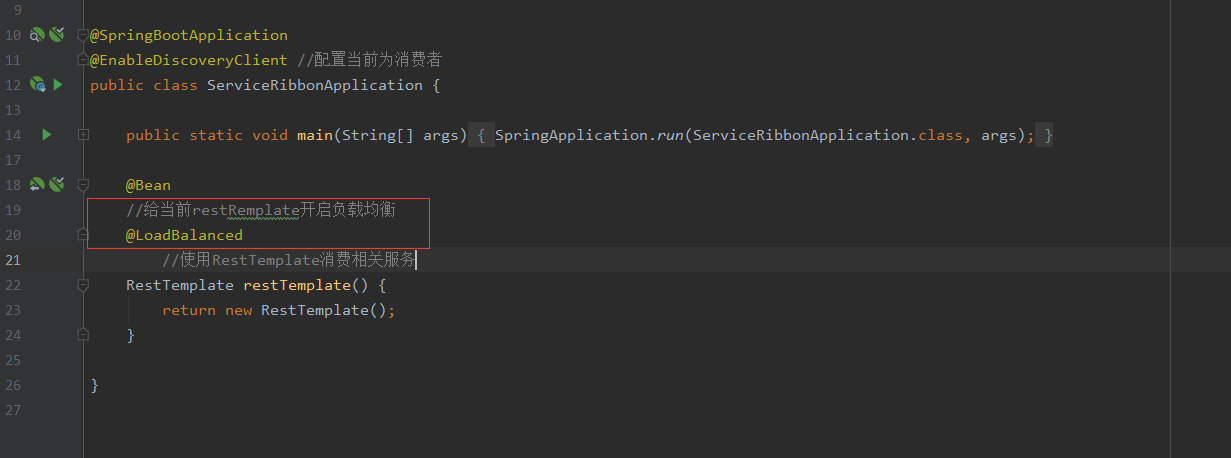
* 1. **测试结果**
* **服务注册中心**



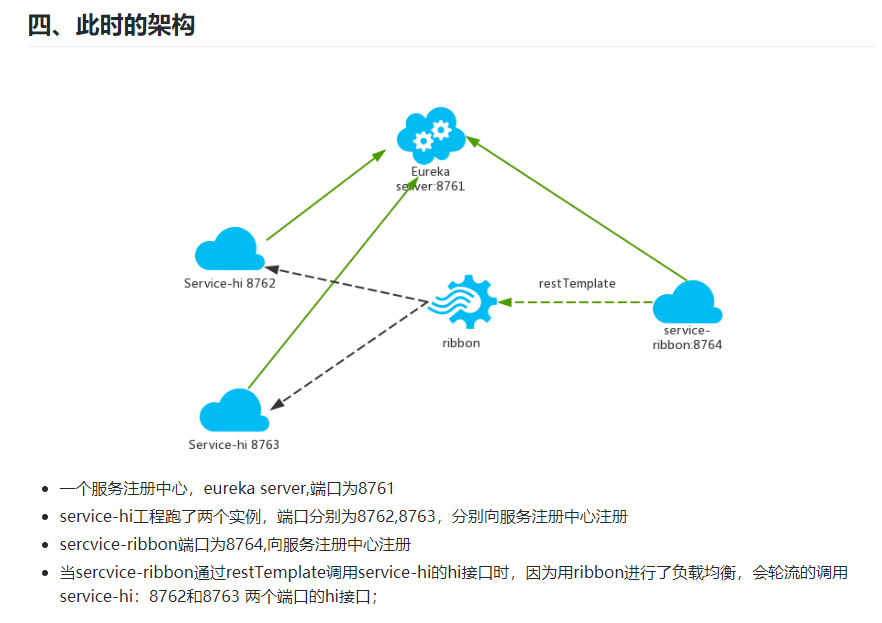
* **访问服务消费者**

多次刷新访问端口会发生变化，因为我们开启了负载均衡，默认是通过轮询的方式进行的，可以通过重写方法，定义自己的策略



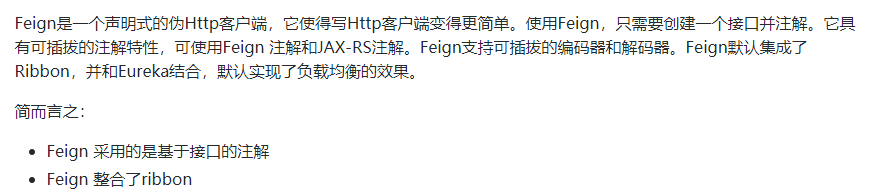
**此时架构参考：**



1. feign的服务调用方式

**(实例代码的git Tag：tag\_2020\_01\_01)**

* 1. **feign简介**

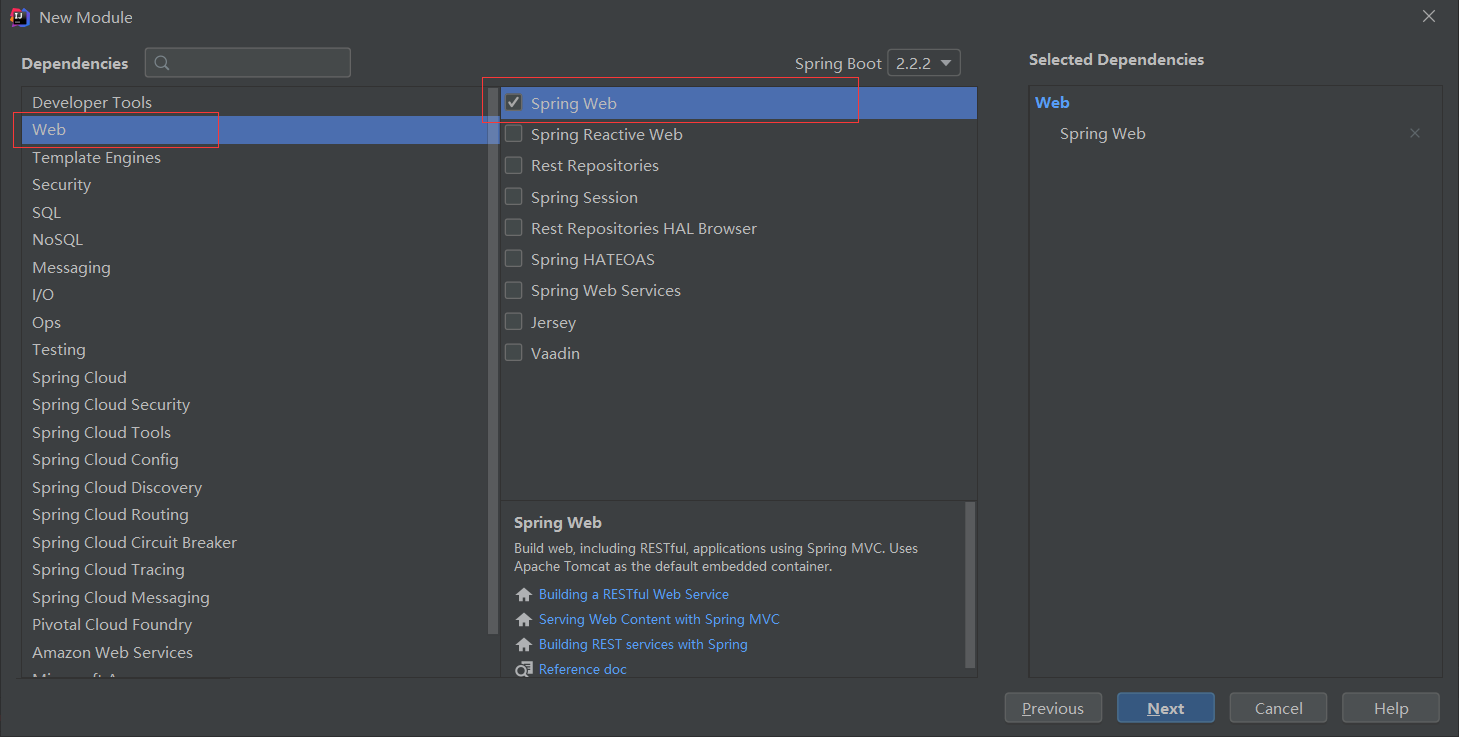


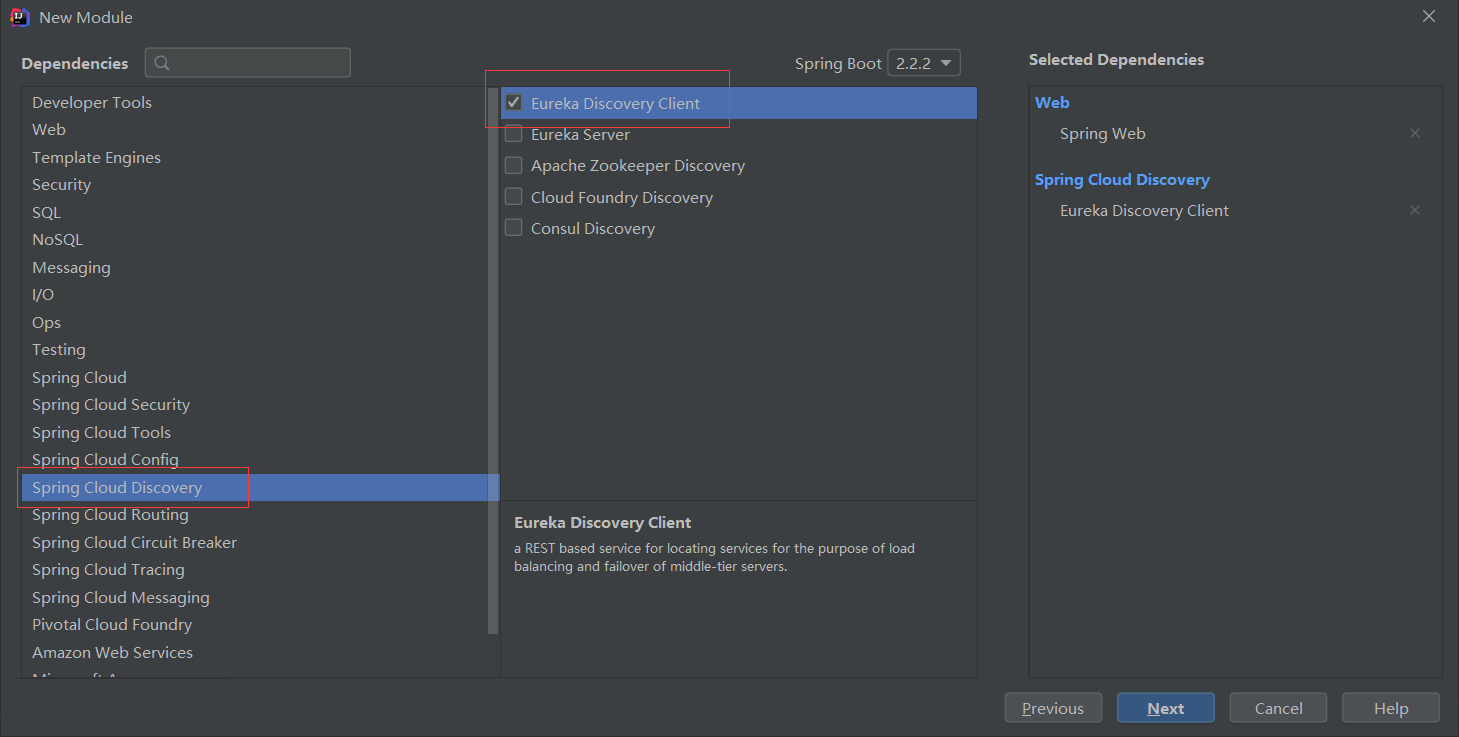
* 1. **准备工作**

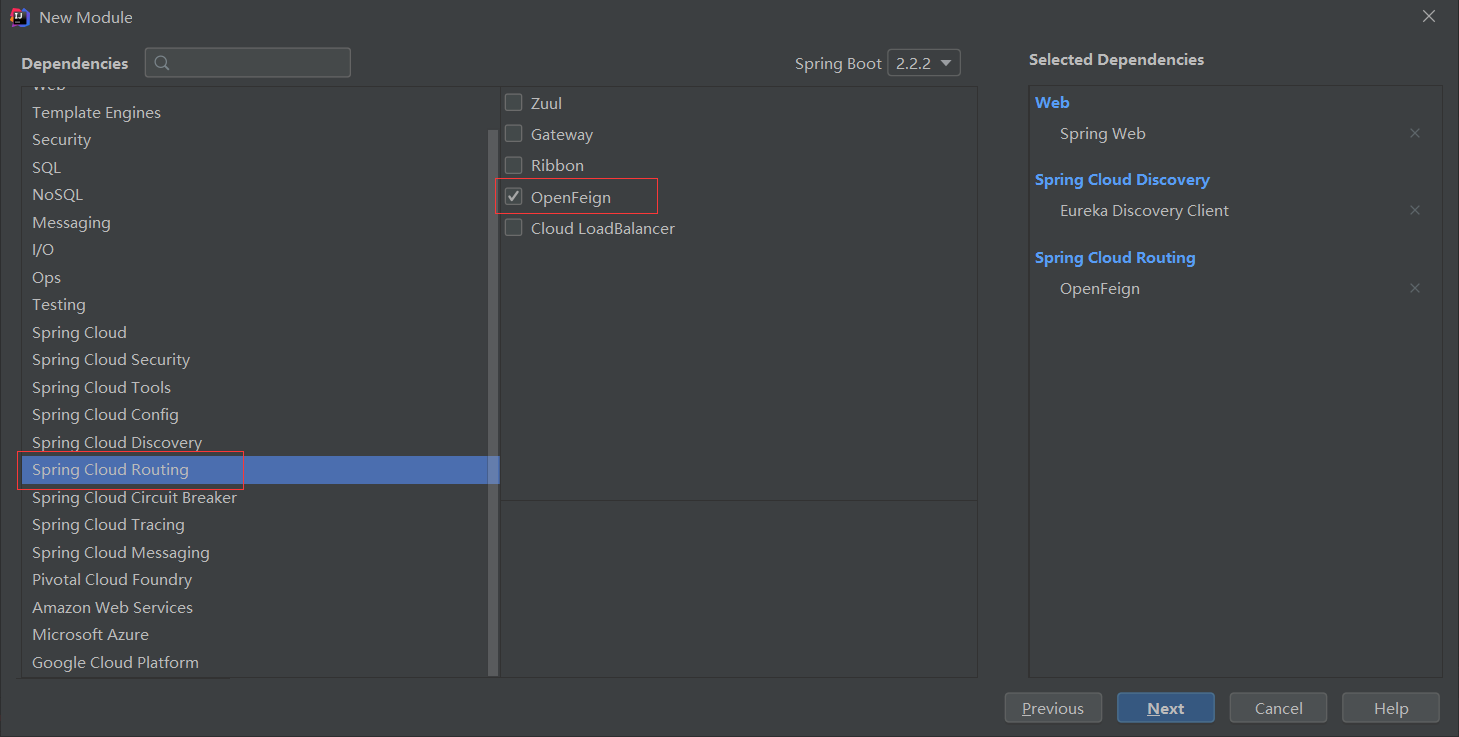
继续使用之前的服务提供者，具体查看前面的配置过程

* 1. **创建服务消费者**

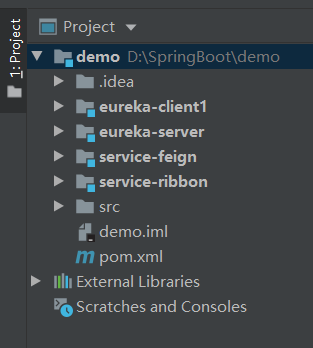
**创建步骤跟创建服务者一致，需要勾选feign、Eureka Discover Client 与 springboot 的web组件,其余组件按需求勾选，与建立springboot项目相同**





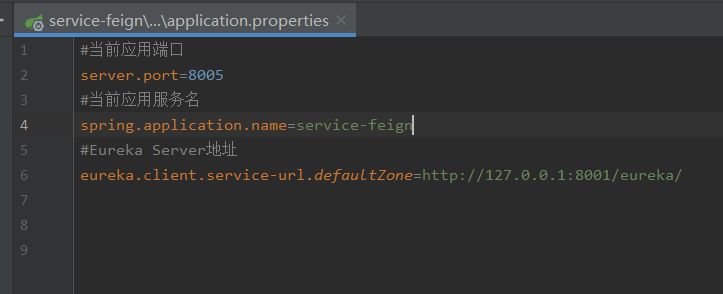


**此时目录结构如图所示：**

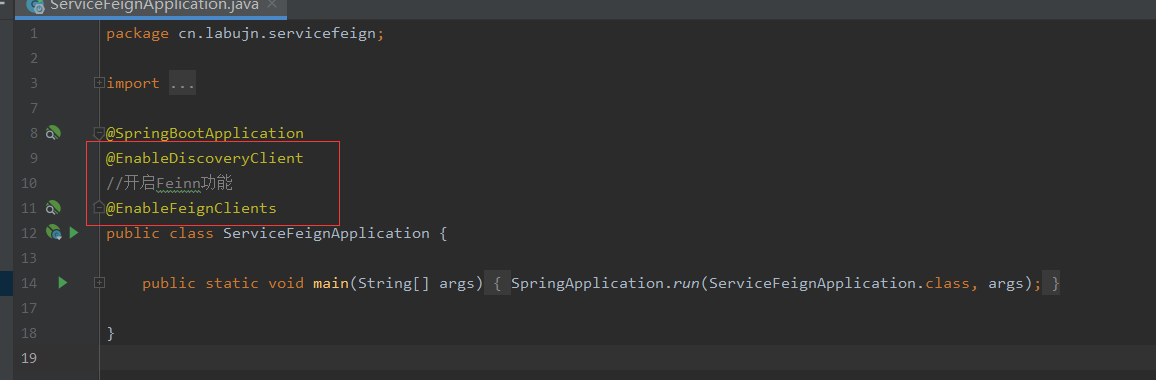


**配置：**

**application.properties**



**启动类：**

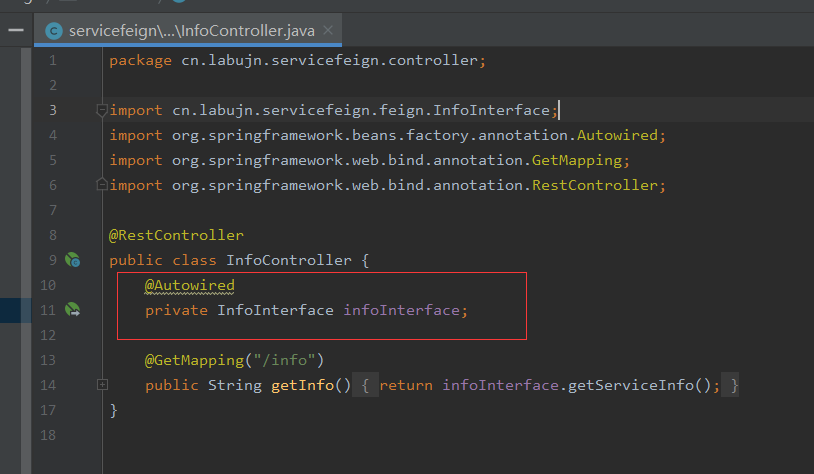


* 1. **定义feign接口，消费远程服务**

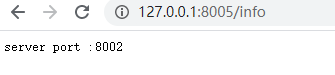
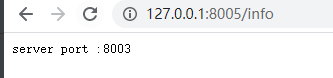
**接口定义：**



**控制器：**

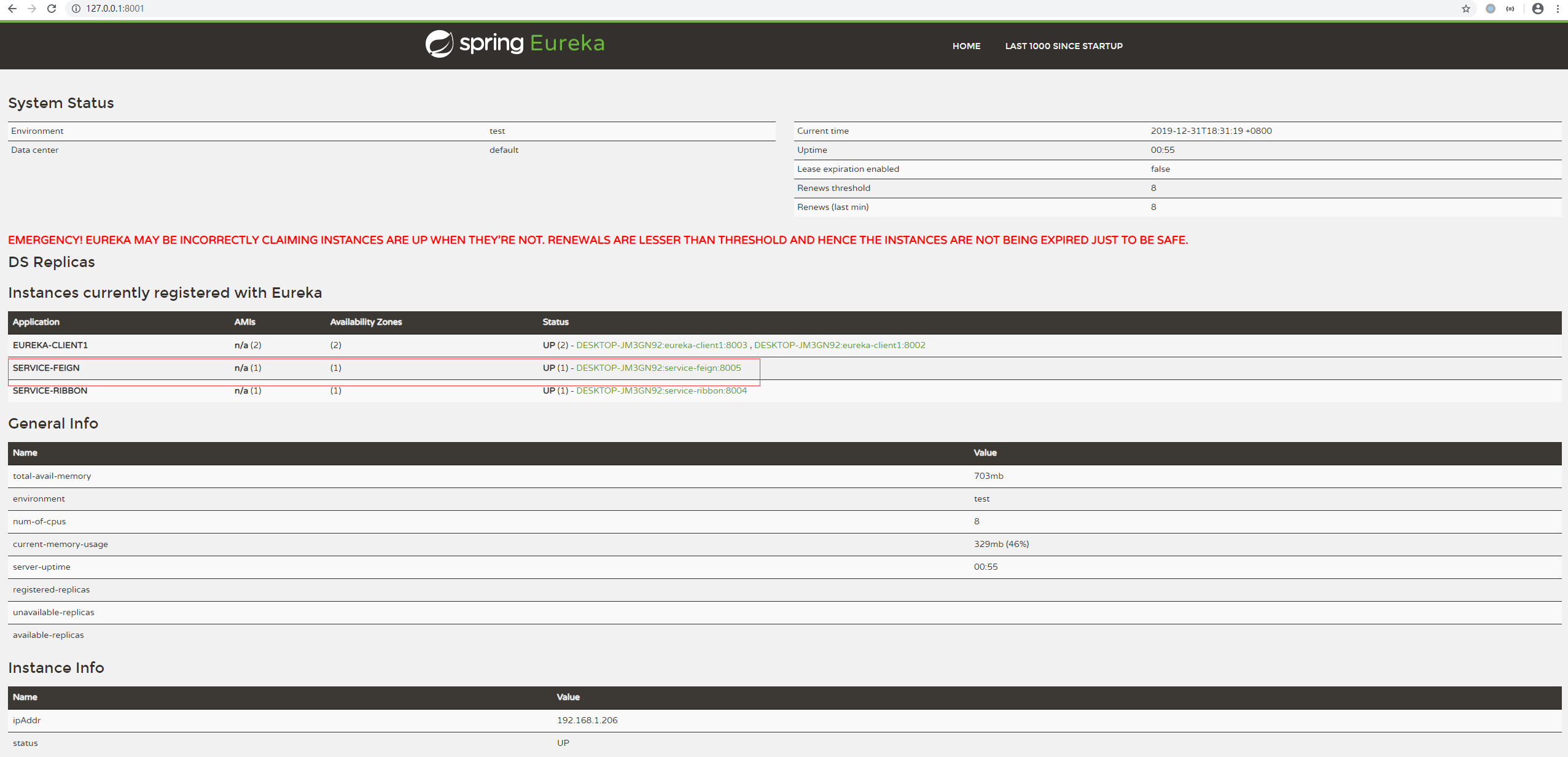


* 1. **测试结果**

访问测试发现，也是有负载均衡的，这是因为Feign默认集成了Ribbon，并和Eureka结合，默认实现了负载均衡的效果。

**此时服务注册中心截图：**



断路器

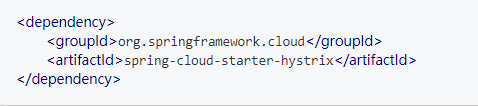
**(实例代码的git Tag：tag\_Hystrix)**

1. 为什么要有断路器

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

为了解决这个问题，业界提出了断路器模型。

1. 断路器Hystrix的使用
   1. **准备工作**
2. 把服务注册中心（8001端口）与服务提供者（8002、8003端口）跑起来
3. 引入hystrix（在服务消费者中引入）

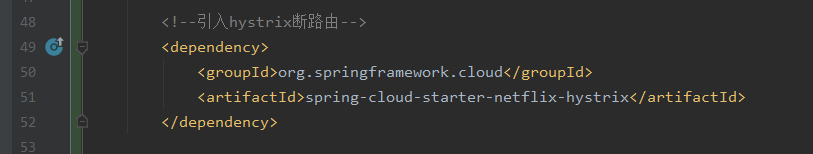


* 1. **在ribbon中使用断路器**

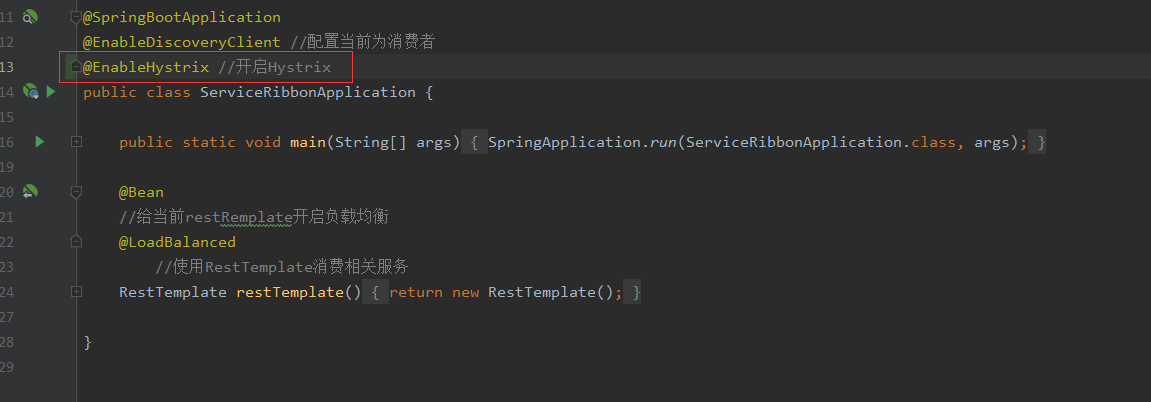
**项目：service-ribbon**

**端口：8004**

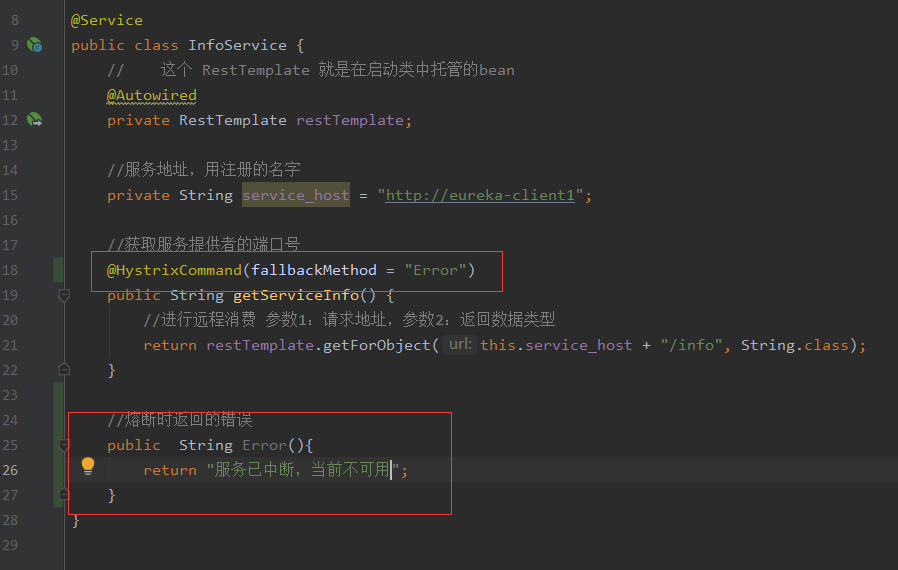
1. **引入依赖pom.xml**



1. **在程序的启动类加@EnableHystrix注解开启Hystrix：**

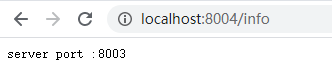


1. **改造InfoService类，在getServiceInfo方法上加上@HystrixCommand注解。该注解对该方法创建了熔断器的功能，并指定了fallbackMethod熔断方法，熔断方法直接返回了一个字符串，代码如下：**

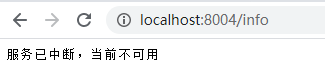


1. **测试**

启动：service-ribbon 工程，当我们访问http://localhost:8004/info,浏览器显示：



此时关闭 service-client1 工程，当我们再访问http://localhost:8004/info，浏览器会显示：



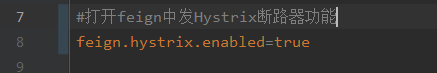
**这就说明当 service-client1工程不可用的时候，service-ribbon调用 service-client1的API接口时，会执行快速失败，直接返回一组字符串，而不是等待响应超时，这很好的控制了容器的线程阻塞。**

* 1. **在feign中使用断路由**

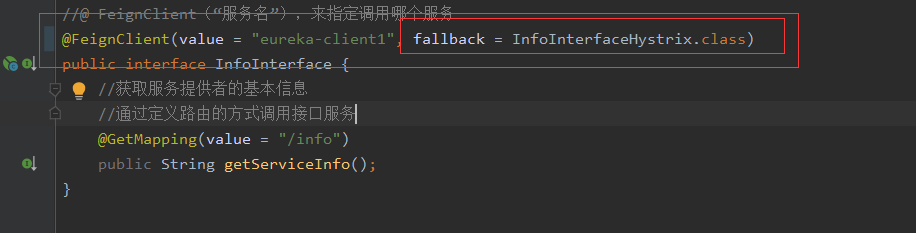
**项目：service-feign**

**端口：8005**

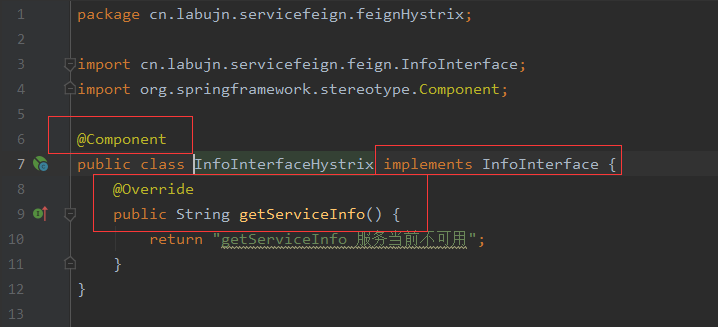
1. Feign是自带断路器的，它没有默认打开。需要在配置文件中配置打开它，在配置文件加以下代码：



1. 基于service-feign工程进行改造，只需要在FeignClient的接口的注解中加上fallback的指定类就行了：

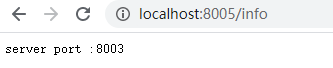


1. InfoInterfaceHystrix需要实现InfoInterface接口，并注入到Ioc容器中，代码如下：

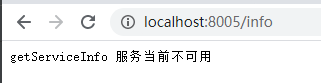


1. 测试：

启动：service-feign 工程，当我们访问http://localhost:8005/info,浏览器显示：



此时关闭 service-client1 工程，当我们再访问http://localhost:8005/info，浏览器会显示：



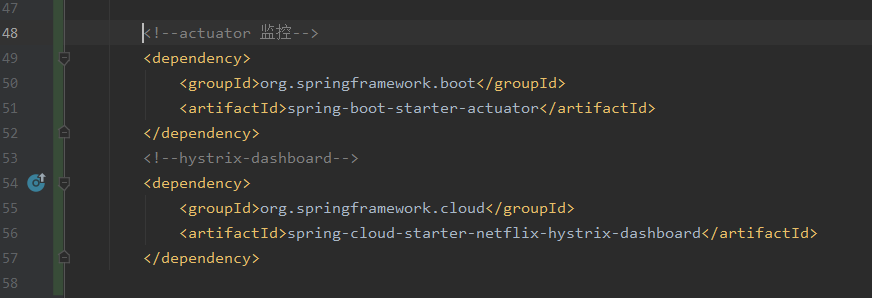
1. 断路器Hystrix的仪表盘

在ribbion中使用与feign中相同，下面用feign作为示例,**区别的地方已经标注**

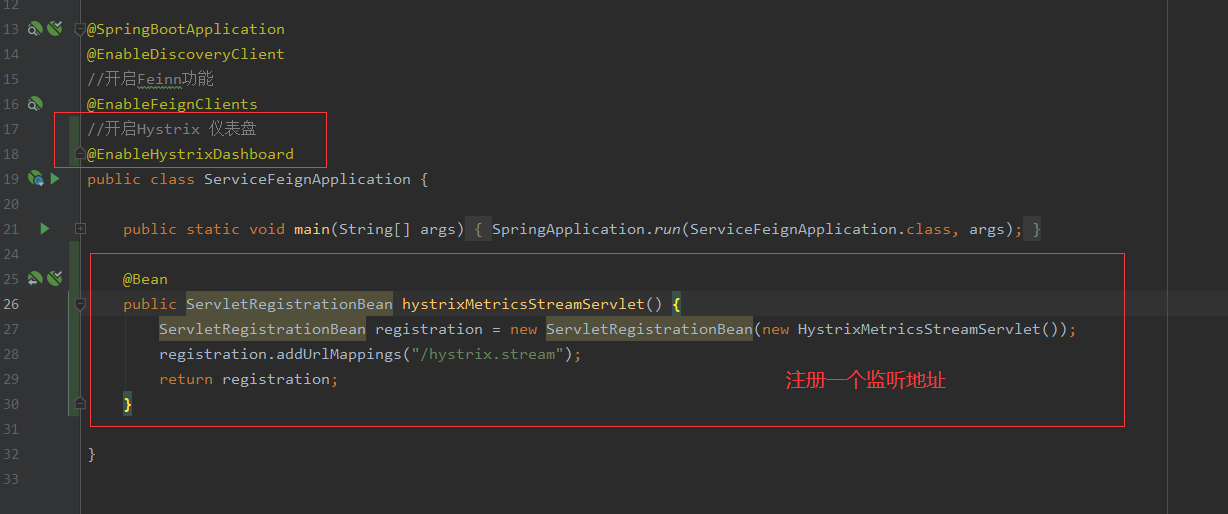
**项目：service-feign**

**端口：8005**

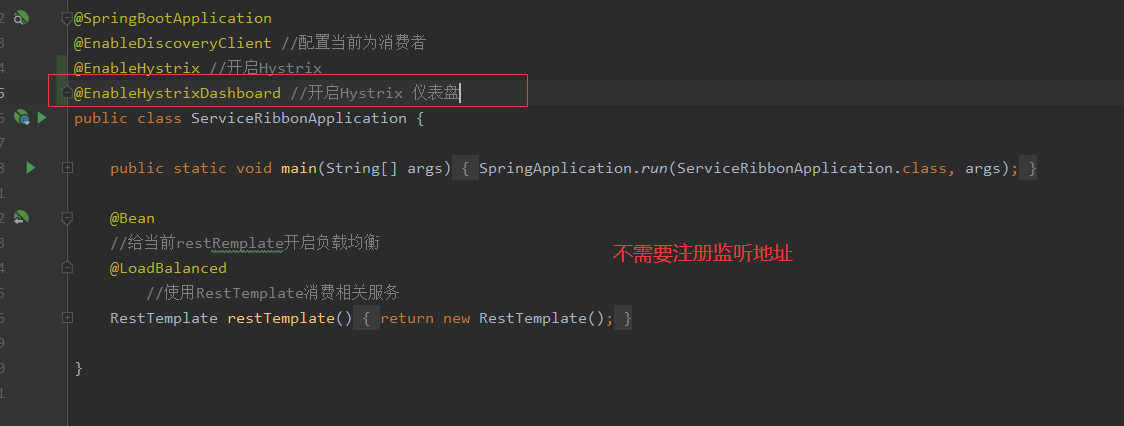
1. 在pom.xml引入Hystrix dashboard（图形化数据）、actuator（获取监控数据）



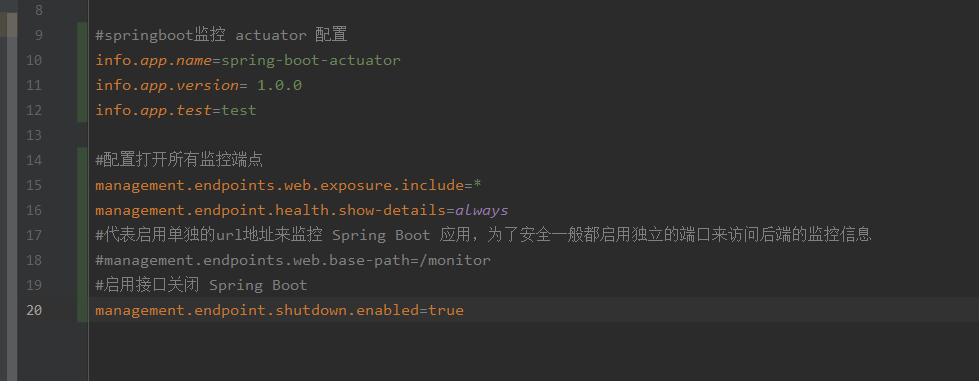
1. 配置好Hystrix后，在启动类加入@EnableHystrixDashboard注解开启仪表盘**(feign)**



配置好Hystrix后，在启动类加入@EnableHystrixDashboard注解开启仪表盘(ribbion)



Actuator常规配置



1. 测试

把eureka-service、eureka-client1、service-feign服务分别开启

先访问服务http://localhost:8005/info

再访问

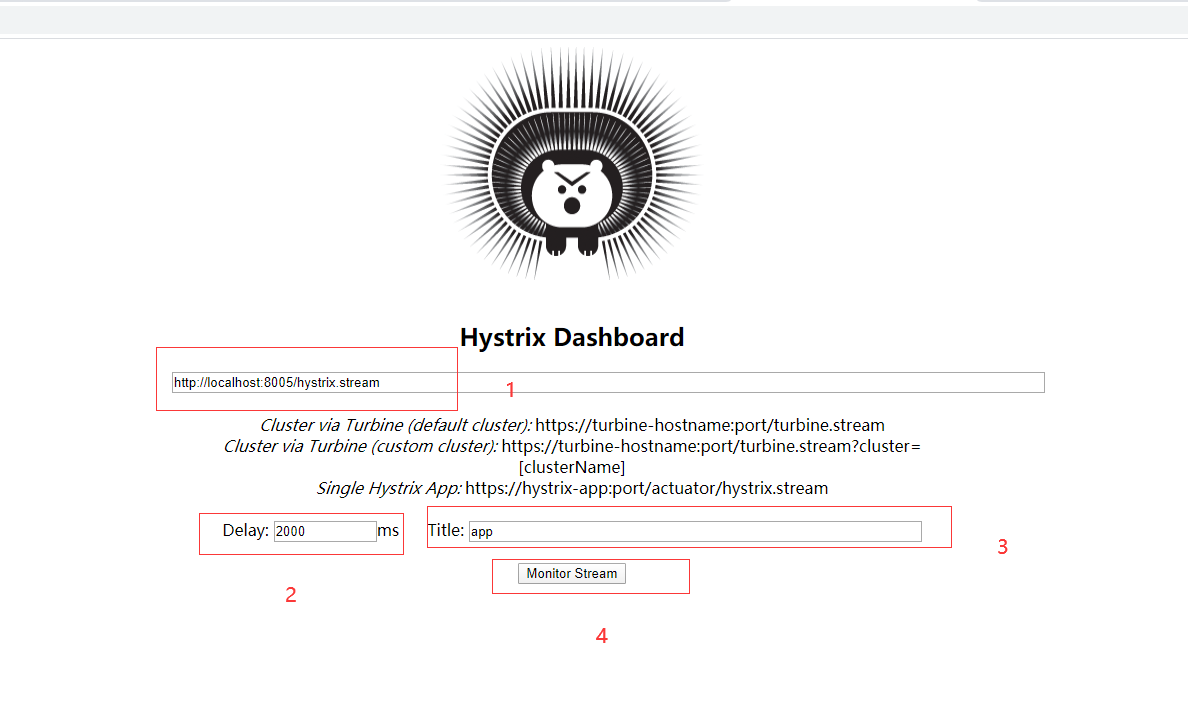
[http://localhost:8005/hystrix.stream](http://localhost:8005/hystrix.stream（feign)**[（feign](http://localhost:8005/hystrix.stream（feign)）**

<http://localhost:8004/actuator/hystrix.stream>**（ribbion）**

可以看到一些具体的数据



打开<http://localhost:8005/hystrix>会看到一个控制面板

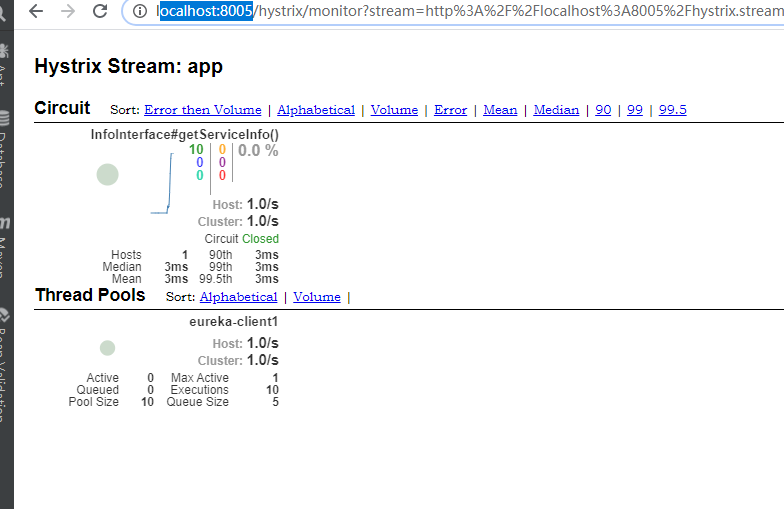


在1处填写<http://localhost:8005/hystrix.stream>

2处填写2000（自定义）

在3处填写标题名字

点击4，可以看到如下页面



这个页面就是把监控地址的数据图形化了

**Hystrix Dashboard Wiki上详细说明了图上每个指标的含义，如下图：**

