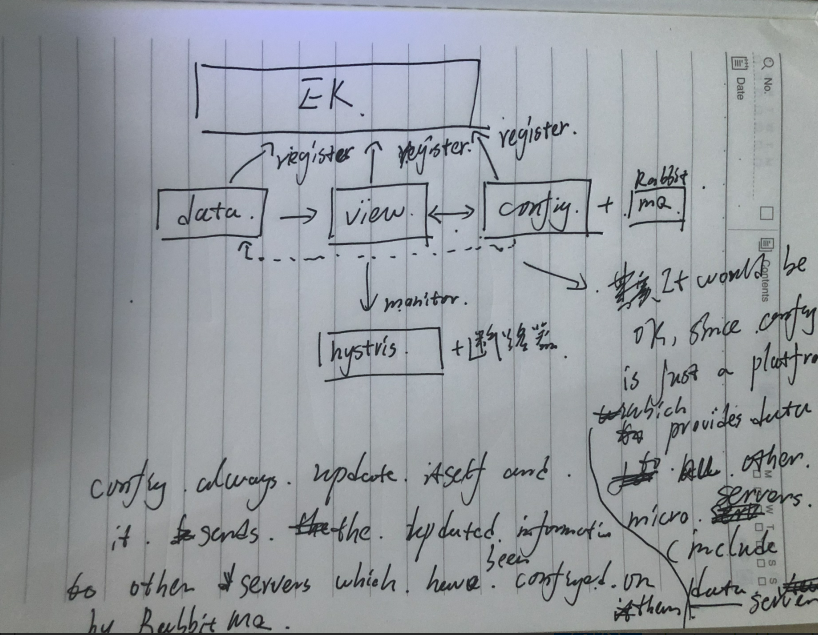
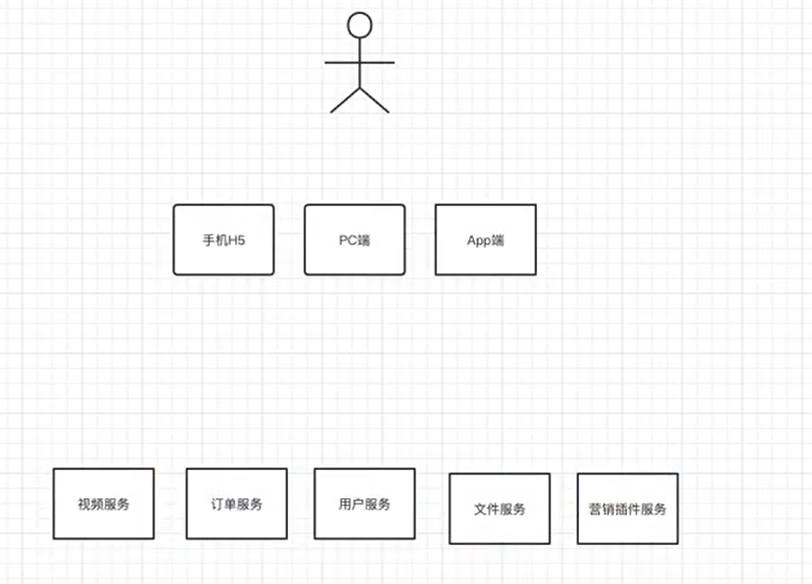
# 什么是分布式集群和SpringCloud





虽然也做过了Spark大数据的项目，但是实际在分布式这一块还是有点懵，在这里站长引入了单体架构的概念，就是典型的一个java应用，可以直接打成jar或者war的。这种单体架构有很多弊端，其主要体现在高并发的时候，一旦1s/2000次访问，这种单体架构就不能用了，怎么办？这个时候分布式集群就可以出来帮忙了。

而SpringCloud，就是一个搭建分布式集群的工具。

原文：

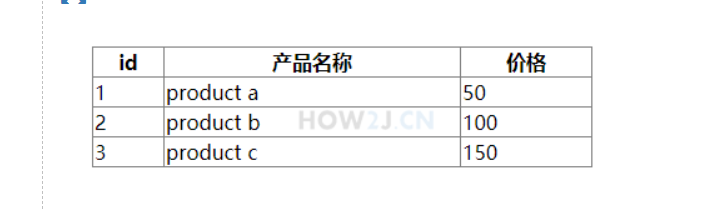
在讲解Springcloud 之前，我们先讲一讲单体架构系统。 所谓的单体架构就是所有功能，都放在一个应用里。  
比如后面要讲的一个单体产品服务应用，提供数据和视图都在一个springboot里。  
  
单体架构系统有其好处，如便于开发，测试，部署也很方便，直接打成一个 jar 或者 war, 就什么都好了。  
  
不过单体架构也有其弊端，最主要体现在高访问，高并发的上限是固定的。 比如一个单体架构，能够承受 1000次访问/秒。 但是访问量达到 2000次/秒的时候，就会非常卡顿，严重影响业务，并且仅仅依靠单体架构本身，很难突破这个瓶颈了。

既然单体架构会有性能上的瓶颈，那么总要解决呀。 解决办法通常就是采用分布式和集群来做。  
可是分布式和集群分别是什么意思呢？  
  
这个就不是一两句话能够说的清楚的啦。。。。 站长又不愿意直接丢一大堆概念给同学们，让没有接触过的同学们更加云里雾里，所以站长会先讲解单体架构，然后对这个单体架构进行逐步改造，最后让其变成一个分布式和集群系统，通过参与这个过程，大家才能够感性地接触和认识到底什么是分布式和集群。

SpringCloud 就是一套工具，帮助大家很容易地搭建出这么一个 集群和分布式的架子出来。  
  
接下来的课程，站长就会从**一个简单的单站系统开始，然后一步一步地把它改造成分布式+集群的系统**。 在改造的过程中，就会逐步引入服务注册中心 Eureka, 客户端Ribbon,Feigh, 断路保护 Hystrix, 配置服务，消息总线等等概念和用法。  
按部就班地跟着站长走一遍，差不多SpringCloud 的初步运用就能掌握啦。

# 2.单体架构与其弊端：

使用站长的一个简单的Springboot项目：



很简单的一个功能，取数据和展示数据。

功能：1. 提供数据 2. 展示数据。 这就是一个典型的单体结构



单体架构的缺点：

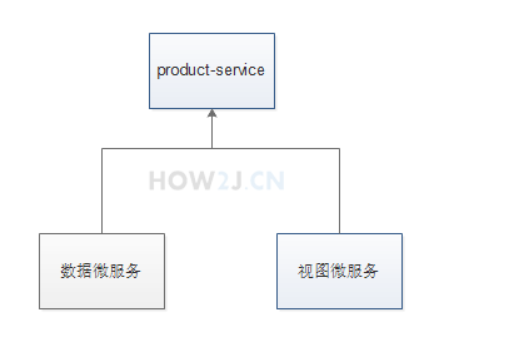
它把两个服务 提供数据和展示数据 放在了一起，这就会出现固有的缺点。  
1. 如果要修改数据部分的代码， 那么必须把整个项目重新编译打包部署。 虽然展示部分，什么都没变但是也会因为重新部署而暂时不能使用，要部署完了，才能使用。（经典的后端稍微改一点就不断重开）

2. 如果提供数据部分出现了问题，比如有的开发人员改错了，抛出了异常，会导致整个项目不能使用，展示数据部分也因此受到影响。（一错全停）

3. 性能瓶颈难以突破  
4. 等等。。。

# 3.微服务基础概念（模块化作业）：

要说springcloud 分布式之前，先引入微服务概念。  
微服务简单说，**一个 springboot 就是一个 微服务**，并且这个 springboot 做的事情很单纯。 比如 product-service 这个项目，就可以拆成两个微服务，分别是 数据微服务，和视图微服务，其实就是俩 springboot, 只是各自做的事情都更单纯~



几乎是“微乎其微”，将每一个能够模块化的地方就模块化。

那么有了很多“微小的”组件服务后，怎么把这些服务串联起来呢？或者说组合成一个完整的“大服务”，这就是Springcloud的微服务注册中心要做的东西了。

叫做：eureka server，这个东西用来实现微服务之间的通信，数据传输。

原文：

那么有了微服务，就存在如何管理这个微服务，以及这两个微服务之间如何通信的问题，所以就要引入一个 微服务注册中心概念，这个微服务注册中心在 springcloud 里就叫做 eureka server, 通过它把就可以把微服务注册起来，以供将来调用。

在业务逻辑上， 视图微服务 需要 数据微服务 的数据，所以就存在一个微服务访问另一个微服务的需要。  
而这俩微服务已经被注册中心管理起来了，所以 视图微服务 就可以通过 注册中心定位并访问 数据微服务了。

# 4.分布式和集群的概念

## 1.分布式：（是指一个完整的项目分成很多小模块“微服务”）

这样，多个不同的部门管理属于自己的“微服务”，他们全部拼起来组成一个完整的服务，就叫做“分布式”

分布式的好处就是他们每一个“微服务”都是属于自己的一个单独模块，比如A模块修改了需要重启，B模块不会因此受到影响需要重启。（类似前后端分离），反正，就是模块化作业的思想。

原文：

系统改造到现在，就是分布式啦~ 简单说，原来是在一个 springboot里就完成的事情，现在分布在多个 springboot里做，这就是初步具备 分布式雏形了。  
那么分布式有什么好处呢？  
1. 如果我要更新数据微服务，视图微服务是不受影响的  
2. 可以让不同的团队开发不同的微服务，他们之间只要约定好接口，彼此之间是低耦合的。  
3. 如果视图微服务挂了，数据微服务依然可以继续使用

## 2.集群：（一个“微服务”有很多的“替身”，形成集群）

这个不陌生，原来的flume1和flume2和flume3，都是一样的功能，都是往指定的文件爬取数据。相当于flume1有两个替身。

集群的好处是：（负载提升，高可用）

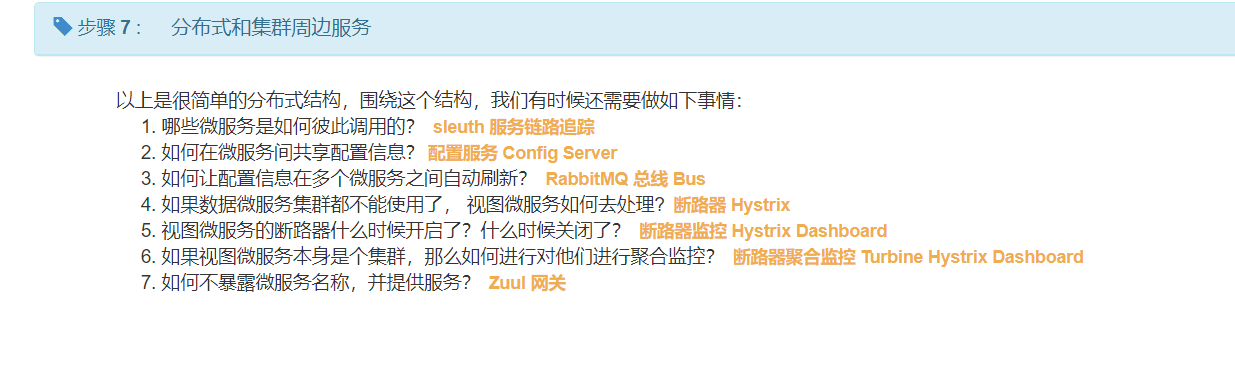
1.横向拓展了负载承担性能，比如现在有3台机器，（即一个微服务和两个替身，他们实现同一个功能），那么这个微服务的负载承担能力就x3，（即微服务1在A机器上抢不到cpu挂了，替身2或者3就开始作业，这就叫负载承担能力。这个微服务的“承受容错”运行内存x3）

2.替身攻击！即使微服务1挂了，还有替身2和3能用，他们只是端口不同而已。

这就叫做高可用



# 5.微服务搭建需要的那些要求？



接下来就是实现他们的教程了。

但首先要实现一个父子集的maven架构。

# 6. Eureka，微服务注册中心（“找到了！”）

## 1.配置eureka微服务

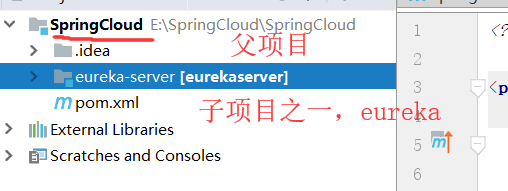
一个项目由很多的微服务组成，那么如何管理这些微服务呢，就需要eureka这个工具了，

Eureka专门用来注册微服务组件。

一个由微服务项目构建的工程，要遵循 父 并 子集的思想，

所以需要使用Maven的父子模式去创建项目，由一个父项目->管理整个子集项目（微服务们）

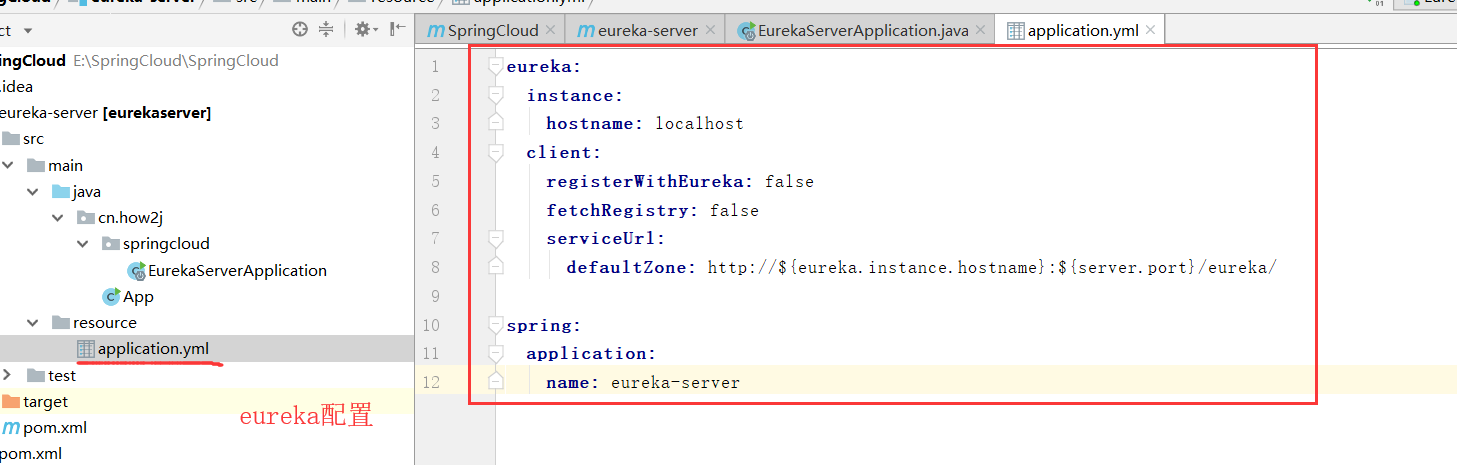
之后，需要单独实现一个Eureka微服务作为管理微服务的组件。



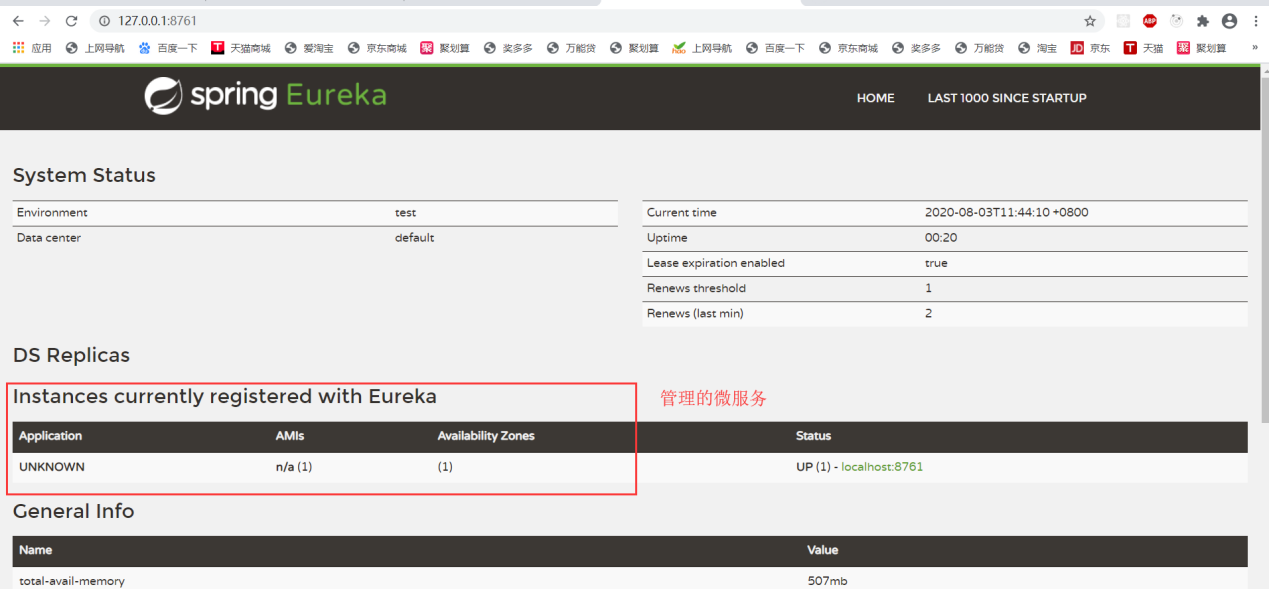
Eureka连接其他微服务的配置：



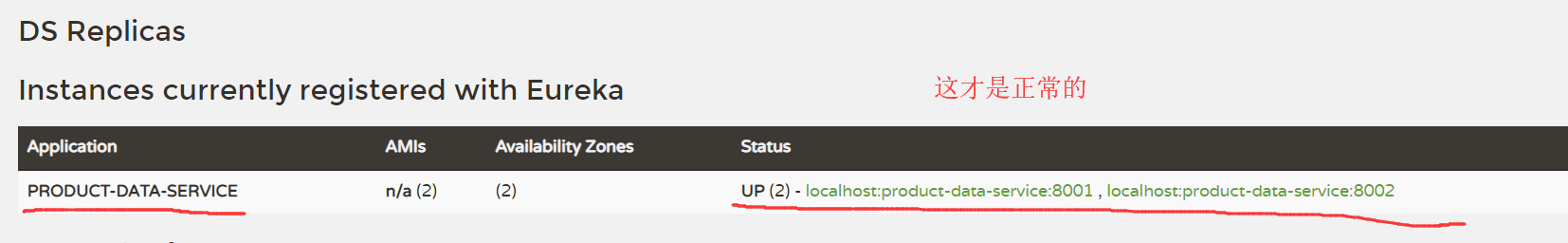
Eureka的配置文件：一定要注意自己配置的resource文件夹要在struct中配置改为resource



效果: <http://127.0.0.1:8761/>



上面那个忘了将resource文件夹注册了。下面这个有Application名字的才是正常的

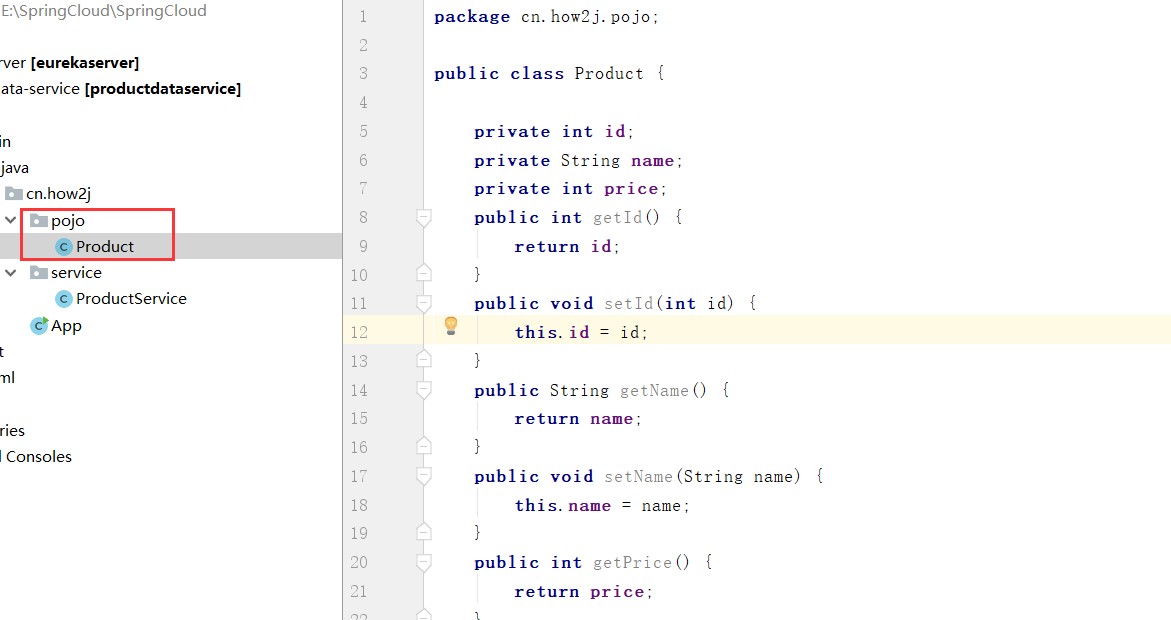


## 2.其他微服务连接eureka（6步）：

1.首先创建一个子项目(比如product-data-service)，再修改其pom文件，使其可以连接eureka



2.定义pojo



3.定义service：

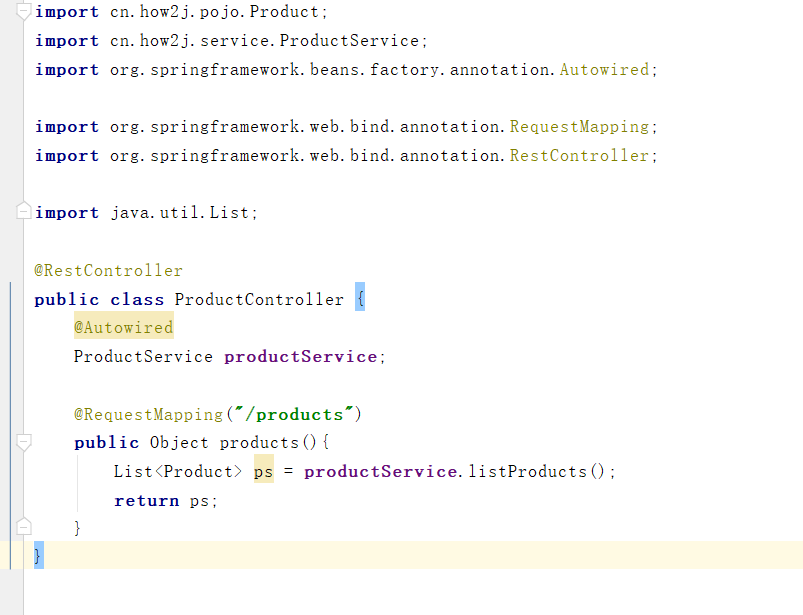


需要注意的是，这里把 **端口号** 放进了产品信息里。

这个数据服务要做成集群，那么访问者为了分辨到底是从哪个数据微服务取的数据，就需要提供个端口号，才能意识到是从不同的微服务得到的数据。

为了分辨是从集群的哪一个微服务拿到的数据。

4.定义Controller:

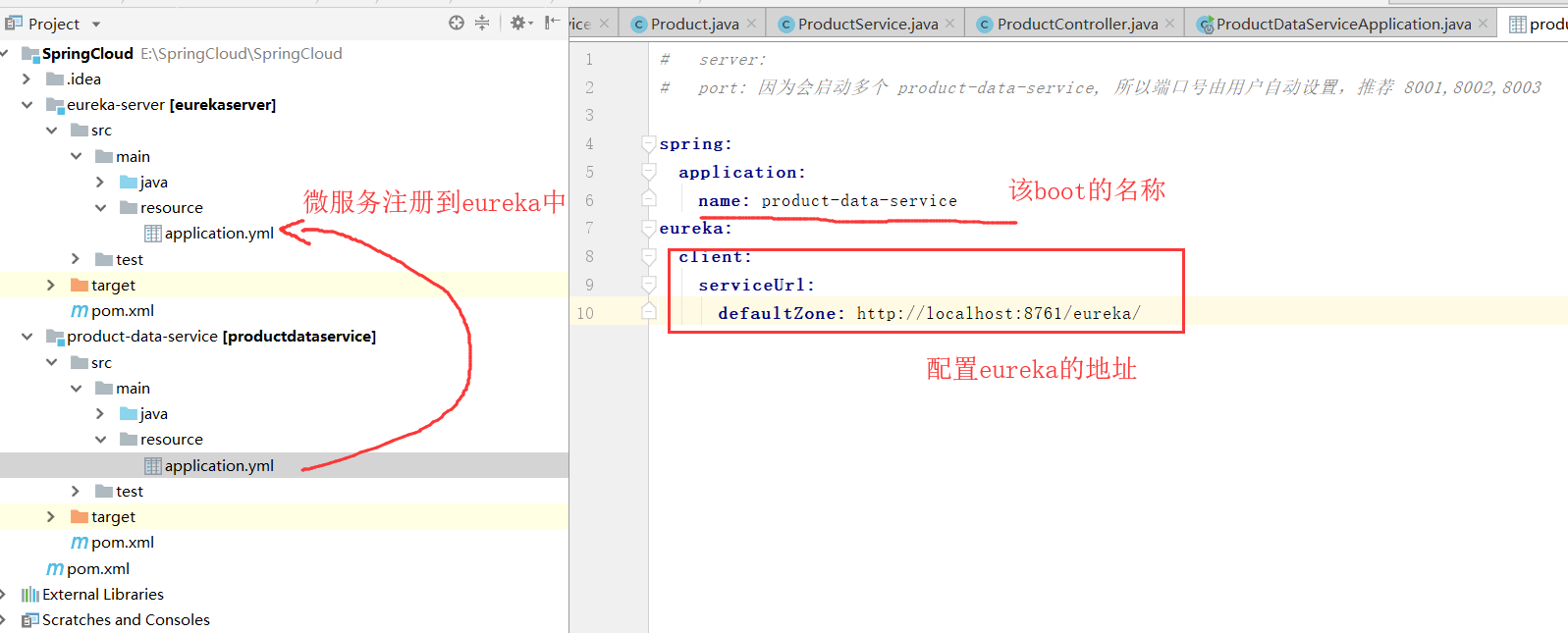


5.启动类，加入Eureka

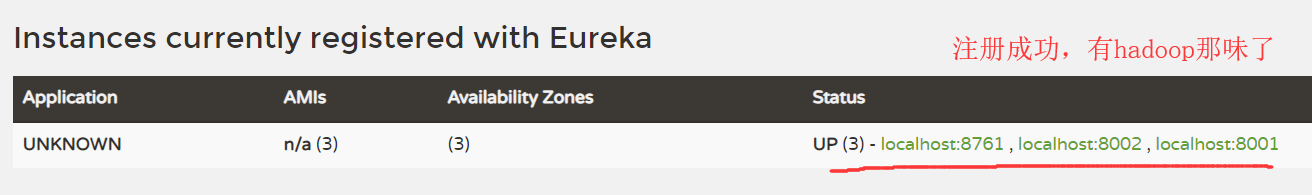




6.配置文件application.yml，注册到Eureka：



效果：



# 7. Ribbon （Feign）微服务之间的数据通信

有同学就会问了，为什么不用前后端分离呢？ 干嘛要用 thymeleaf 做服务端渲染呢？  
原因如下：  
1. 使用前后端分离，站长多半会用 vue.js + axios.js来做，就像 springboot 天猫教程那样。 如果学习者没有这个基础，就会加重学习的负担。

2. 使用前后端分离，是走的 http 协议， 那么就无法演示重要的 微服务端调用了，所以站长这里特意没有用前后端分离，以便于大家观察和掌握微服务的彼此调用

这里主要是展示微服务与微服务之间的通信，所以不会做前后端分离。

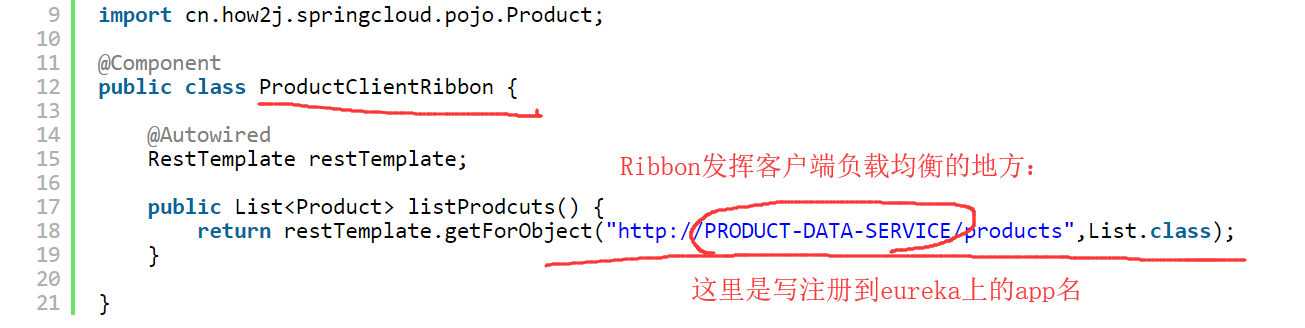
1.客户端负载均衡，选择调用集群中的某一个进行服务：

Ribbon 是使用 restTemplate 进行调用，并进行客户端负载均衡。 什么是客户端负载均衡呢？ 在前面 [注册数据微服务](https://how2j.cn/k/springcloud/springcloud-eureka-client/2039.html) 里，注册了8001和8002两个微服务， Ribbon 会从注册中心获知这个信息，然后由 Ribbon 这个客户端自己决定是调用哪个，这个就叫做客户端负载均衡。

2.配置：

## 1.ClientRibbon（最重要的一步！！！），取得数据的地方

！！！就是这里实现了微服务之间的通信！



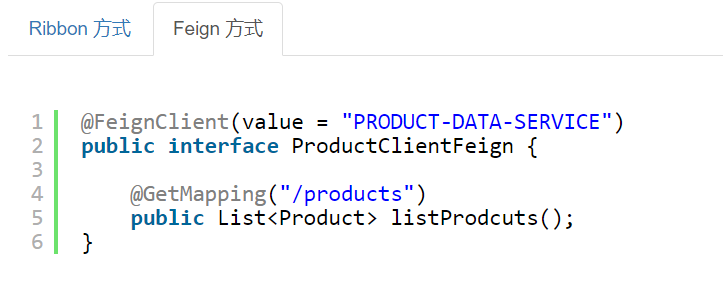
也就是说 这个带有Ribbon的SpringBoot是利用restTemplate这个插件从另一个微服务的接口取到了（xxx/products）的数据：

注意，这个restTemplate组件是在启动类中声明new出来的（见下面）

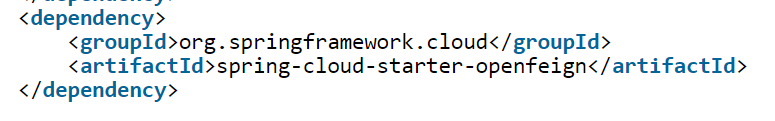


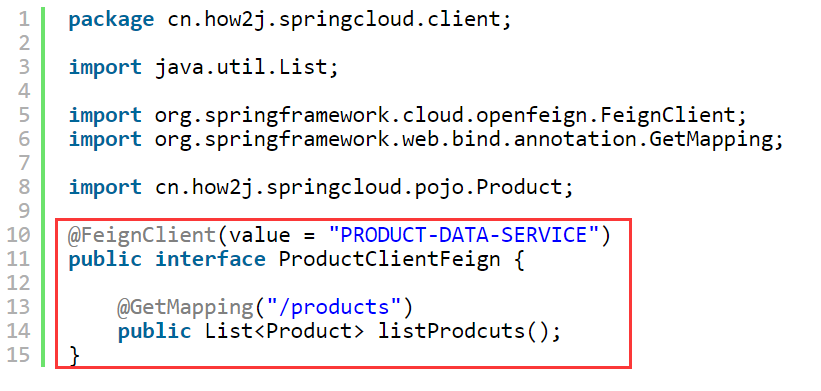
只是这里利用了注册到Eureka上的服务名（product-data-service），从而利用了Eureka得到了接口地址去取数据（eureka的作用这就体现出来了，提供了一个微服务地址查询的平台作用，将注册到这个平台上的所有微服务互相关联起来）

如果使用feign，是这样的：



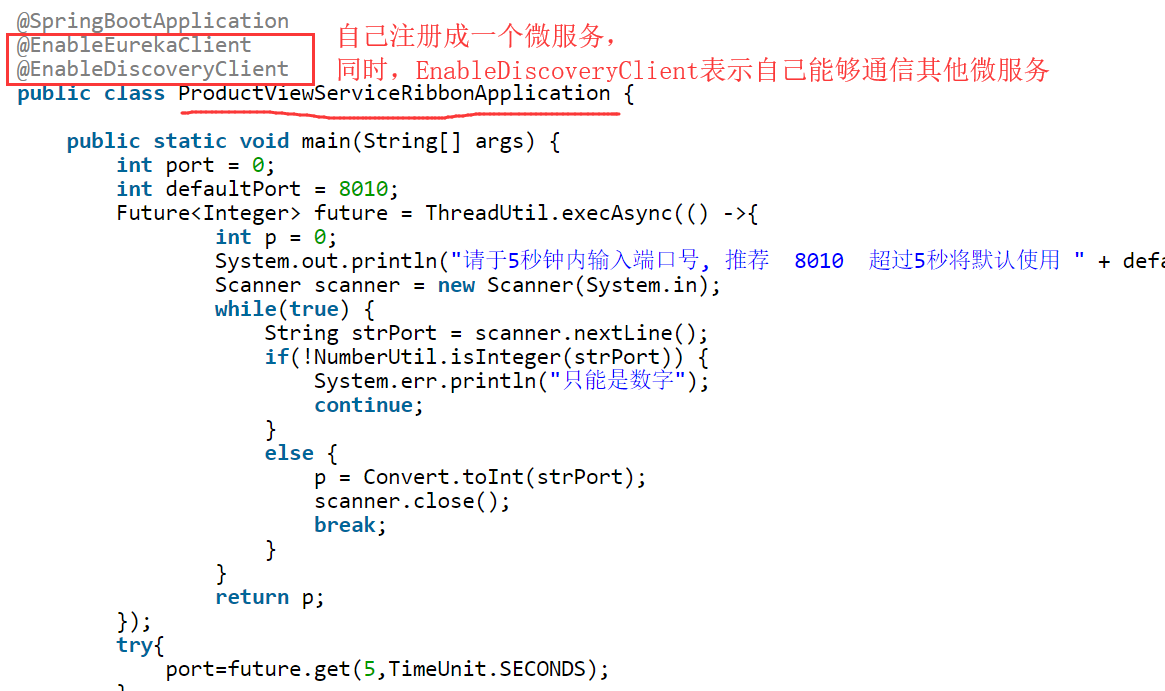
依赖中多一行：

微服务通信clientRibbon做成这样：

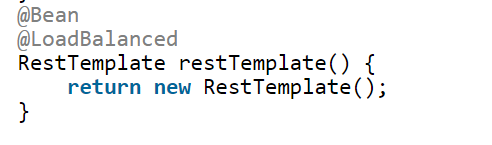


其他的就差不多了

## 2.启动类：（注册通信许可）

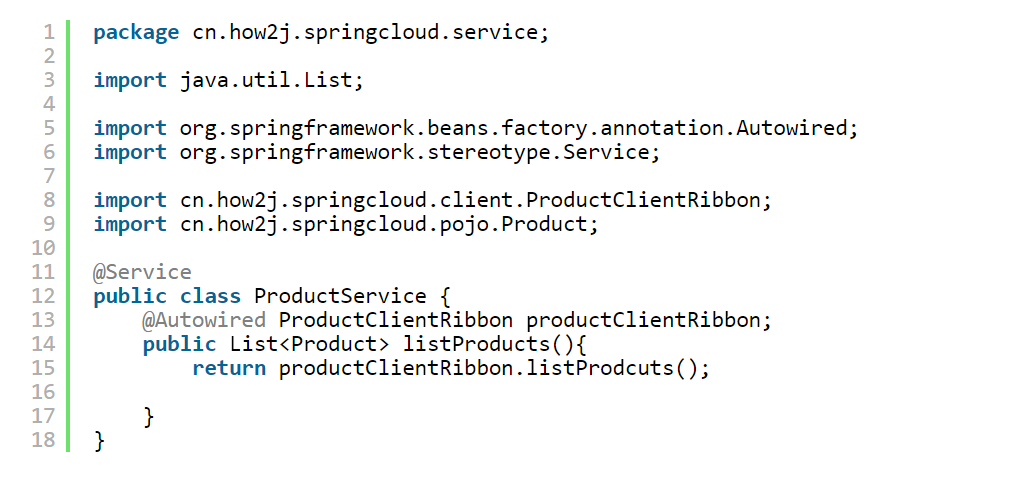


在端口启动下面多了一个，注册restTemplate的地方，这个东西用来实现微服务之间的通信，在上面的ClientRibbon里就主要用了这个来取得其他微服务的数据。

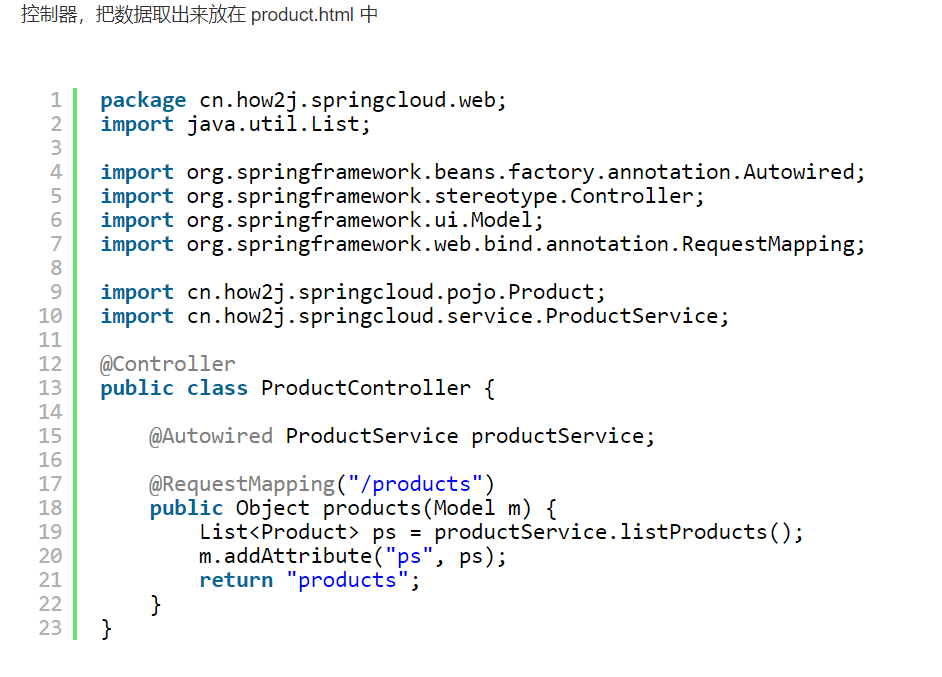


## 3.Service,Controller,html，配置yml:

Service调用ProductClientRibbon从另一个微服务取数据



Controller:



取数据返回数据

Html 展示数据：

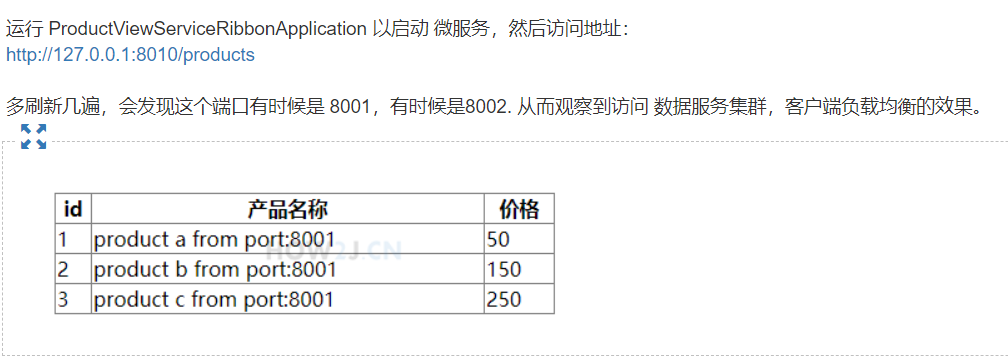


配置文件yml



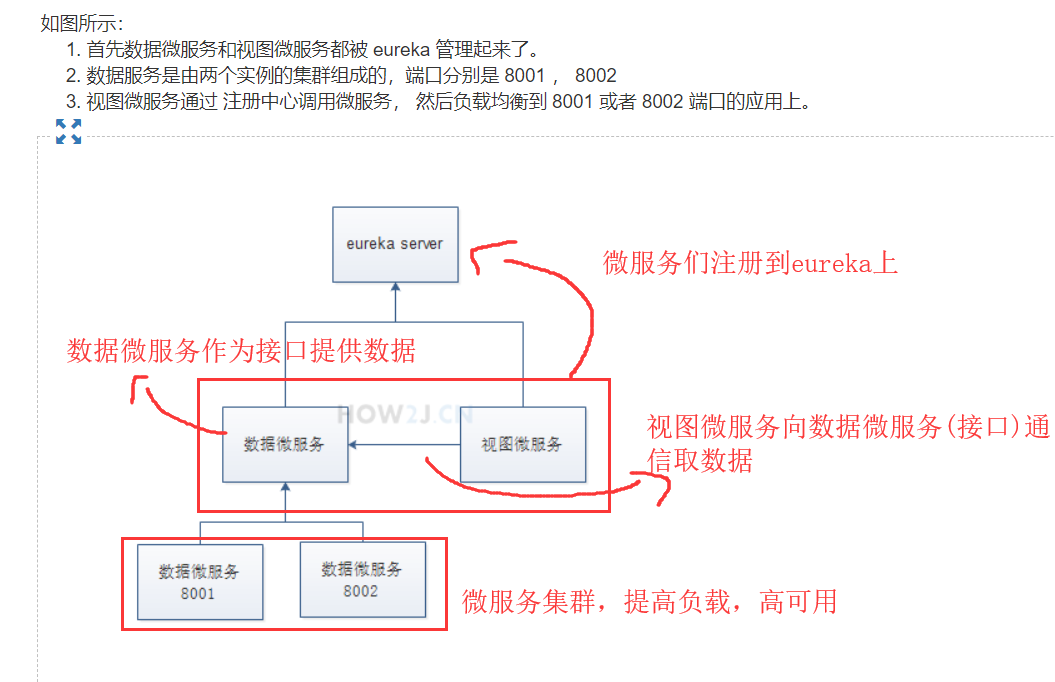
## 4.总结：

效果：



总结就是前1,2步是最重要的，需要将自己注册到eureka并且使用restTemplate进行与其他微服务的通信（同时负载均衡，取数据的微服务会按负载变化）

思维图：



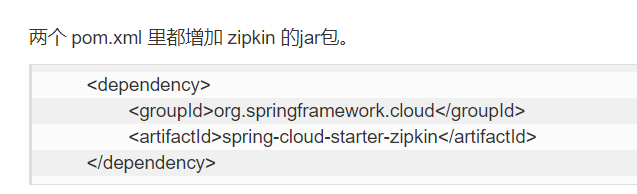
# 8.zipkin服务链路追踪插件（跟踪监控每个微服务的状态）

启动：

java -jar zipkin-server-2.10.1-exec.jar --zipkin.collector.rabbitmq.addresses=localhost

<http://localhost:9411/zipkin/dependency/>

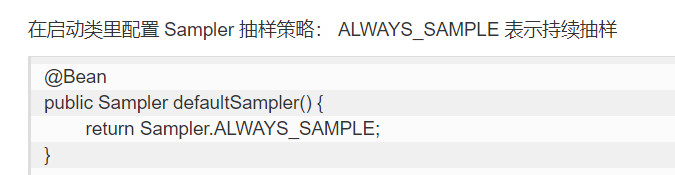
首先加入依赖



配置文件：

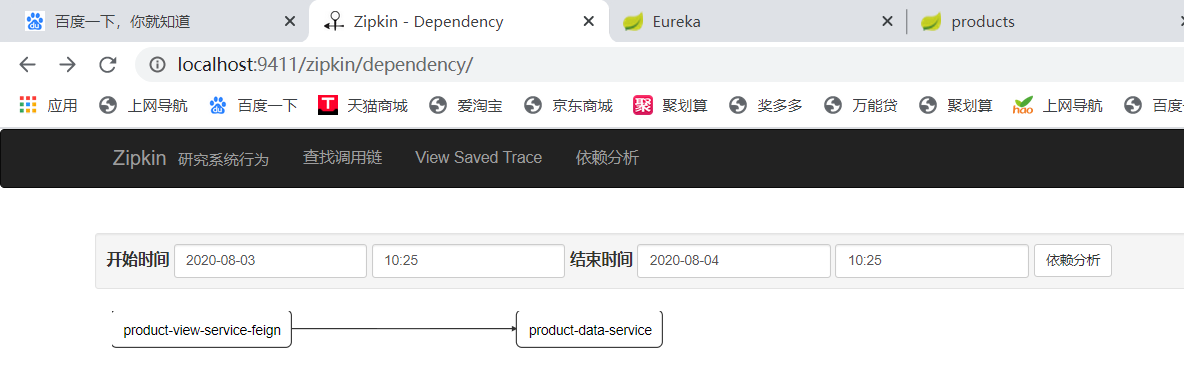


然后在启动类中配置抽样类：这样zipkin就能对其进行抽样并追踪他们的关系了



然后，要先启动zipkin，再启动ek，再启动相关的微服务。

地址:<http://localhost:9411/zipkin/dependency/>



这就是结果

这就相当于一个追踪平台，专门查看各个微服务的连接，调用状态。

# 9.配置服务器ConfigServer：

## 1.搭建configServer：

这是一个特别的SpringCloud微服务组件，专门用来给各个微服务对应的配置信息。

因为对于集群来说，不可能修改了一次配置文件就要全部修改，这很麻烦。所以需要一个专门的配置服务器来管理各个微服务的配置，那些微服务就只需要去配置服务器取各自的配置信息就行了

配置：

Pom文件：

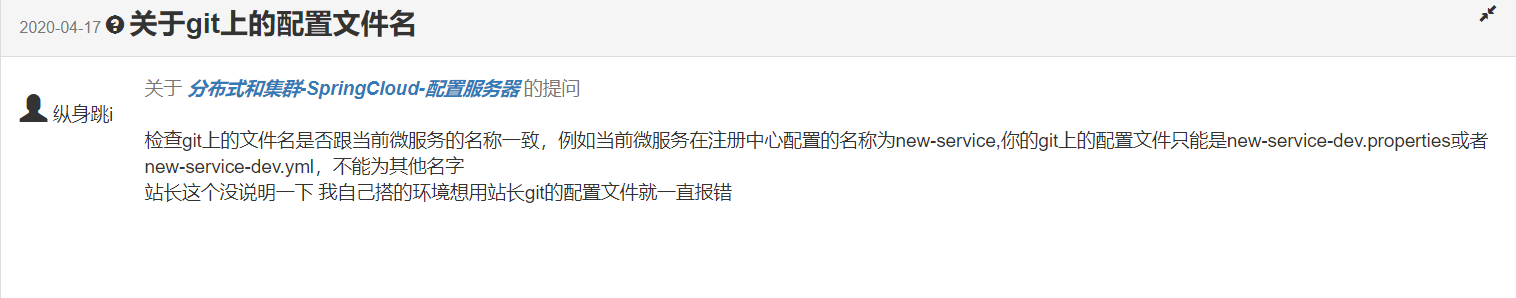




Yml文件：



但是这个从远程git上拉取信息一定要特别注意格式：



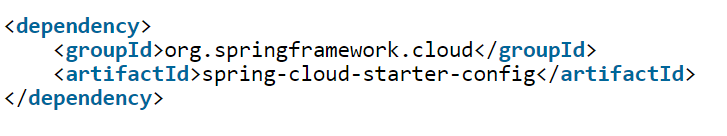
要和当前的微服务名称一致才行。

<http://localhost:8030/version/dev> 拉取地址

## 2.客户端微服务连接到ConfigServer：

其他的微服务怎么样才能访问到配置服务器的信息呢？

1. 首先在pom里添加config依赖：



1. 配置bootstrap.yml，这个和application.yml差不多，都是配置文件

原文：

作为配置客户端，比较。。。特别~ 它需要在 bootstrap.yml 里配置 config-server 的信息，而不是像以前那样在 application.yml 里进行配置。  
bootstrap.yml 和 application.yml 的区别，简单说就是前者先启动，并且一些系统方面的配置需要在 bootstrap.yml 里进行配置。 更多关于他们的区别，请自行百度。  
在 bootstrap.yml 里配置提供了 serviceId: config-server, 这个是配置服务器在 eureka server 里的服务名称，这样就可以定位 config-server了。

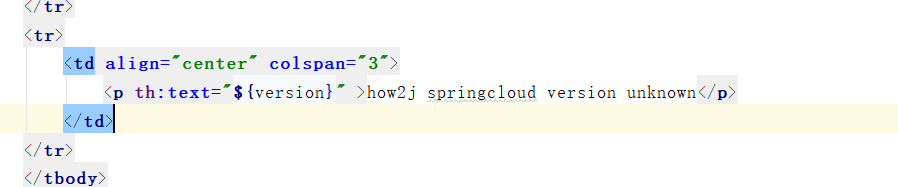
同时application.yml里的ek设置转移到了bs里面



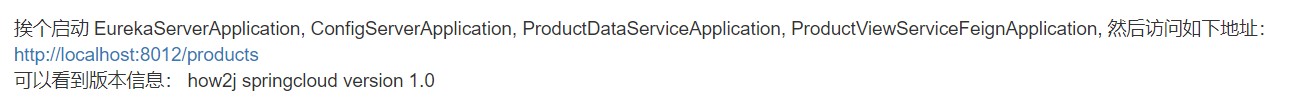
相呼应，就能找到configServer了



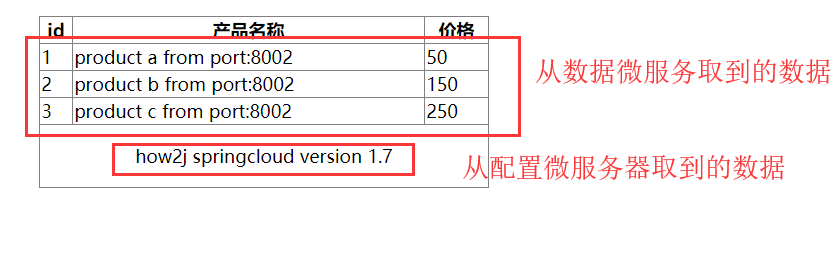
在html展示：



注意启动顺序：

Ek🡪ConfigServer🡪DataServer🡪ViewServer

结果：



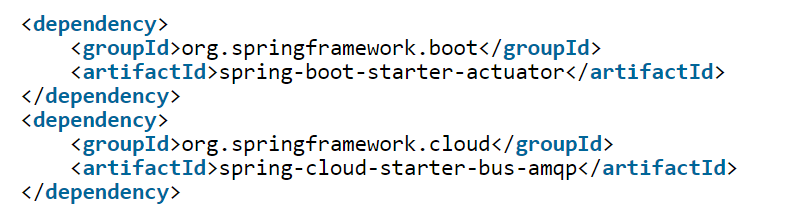
问题：

但现在如果配置服务器连接git上的数据改了之后，需要重启配置服务器以及视图微服务才能展示新的数据，这也太傻了。接下来就是如何解决这个问题：

# 10.消息总线Bus（刷新配置）

之前的教程虽然配置了config-server, 也把 视图服务改造成了配置客户端，但是当需要刷新配置信息的时候，不得不既重启 config-server, 又重启微服务。 这样的体验当然是不太好的。 我们当然是希望一旦 git 上的配置信息修改之后，就可以自动地刷新到微服务里，而不是需要手动重启才可以。

1. 添加依赖：



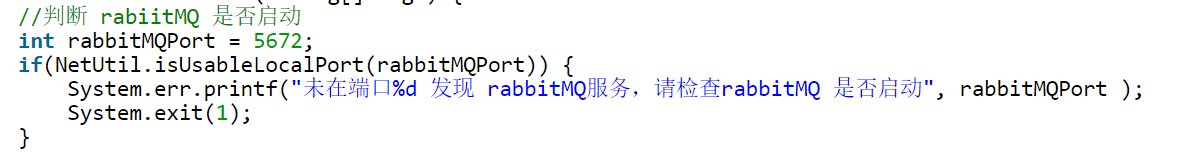
2.配置文件添加消息队列插件：

Bootstrap： Application: 解决跨域问题





3. 启动类添加rabbit判断



1. 设计fresh类来触发刷新：



然后修改了远程的配置文件，只需要启动一下这个fresh类，就可以同步了

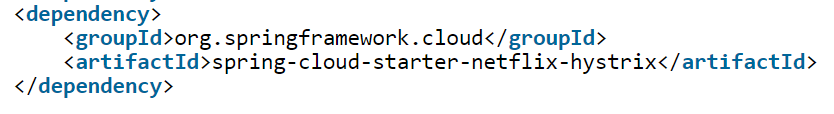
# 11.断路器hystrix（解决后台宕机的情况）

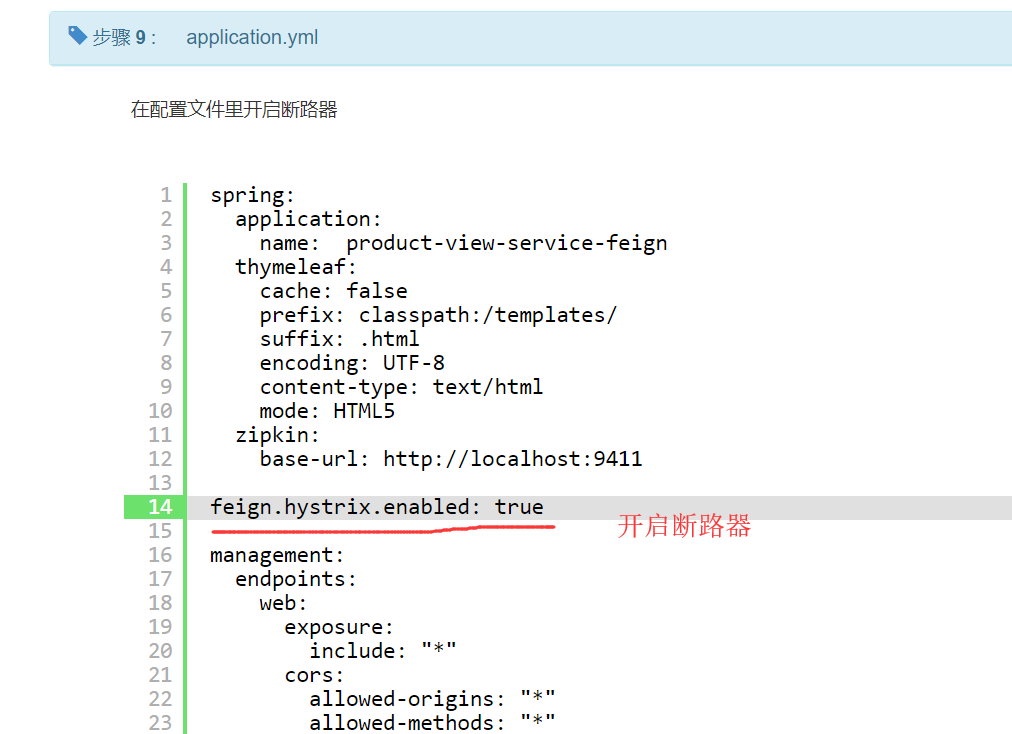
## 1.在View里配置 断路器

当后台数据服务器宕机后，前台目前是没有解决方案的，只能跟着一起断掉。

断路器hystrix就是解决这个问题的，加入断路器之后，当后台数据服务器连接不上时，当前微服务可以有备选方案。

1. 加入依赖：



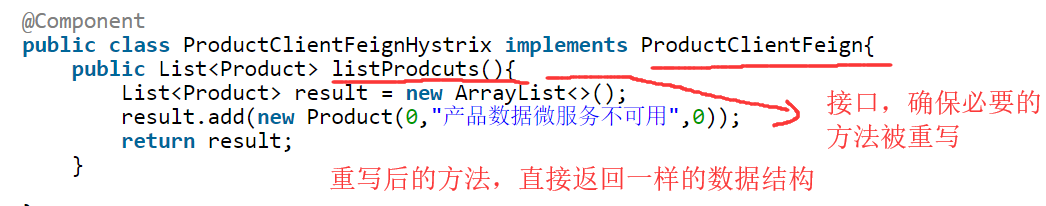


开启断路器

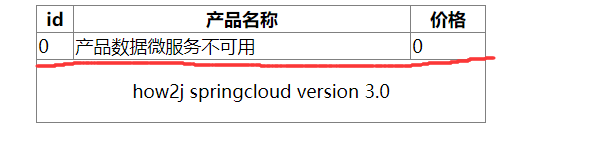
1. 备选方案：



1. 备选方案的编写：



实现接口，并重写需要的方法，返回一样的数据结构，只是内容表示“取不到”原来的



结果就这样了。

## 断路器监控页面访问情况（单微服务）

[https://how2j.cn/k/springcloud/springcloud-dashboard/2043.html#nowhere](https://how2j.cn/k/springcloud/springcloud-dashboard/2043.html" \l "nowhere)

监控地址：<http://localhost:8020/hystrix>

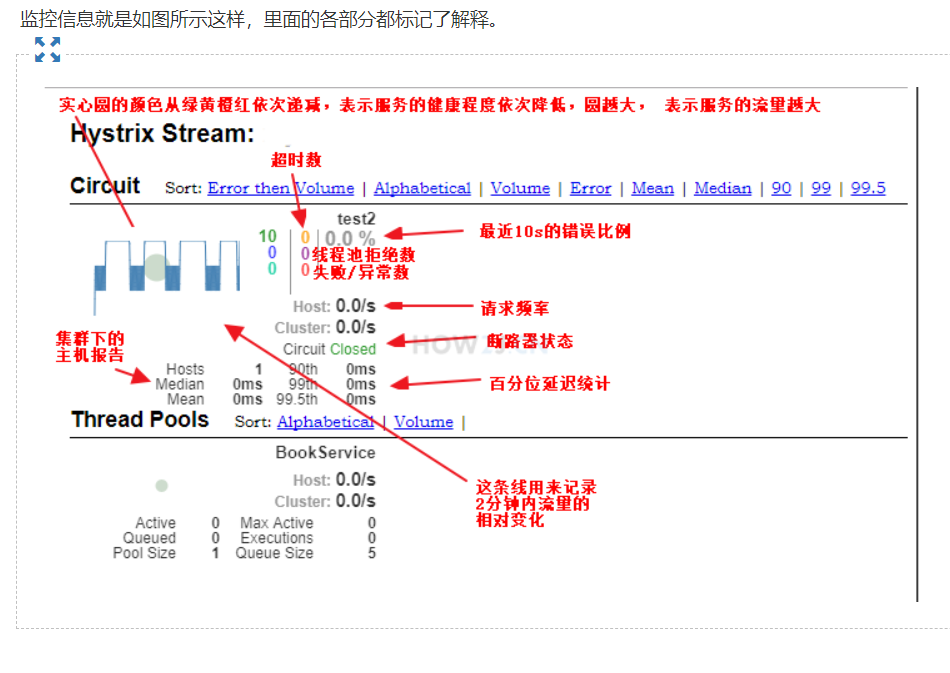
输入：http://localhost:8012/actuator/hystrix.stream

需要单独开一个模块来启动监控器

具体的配置看连接。

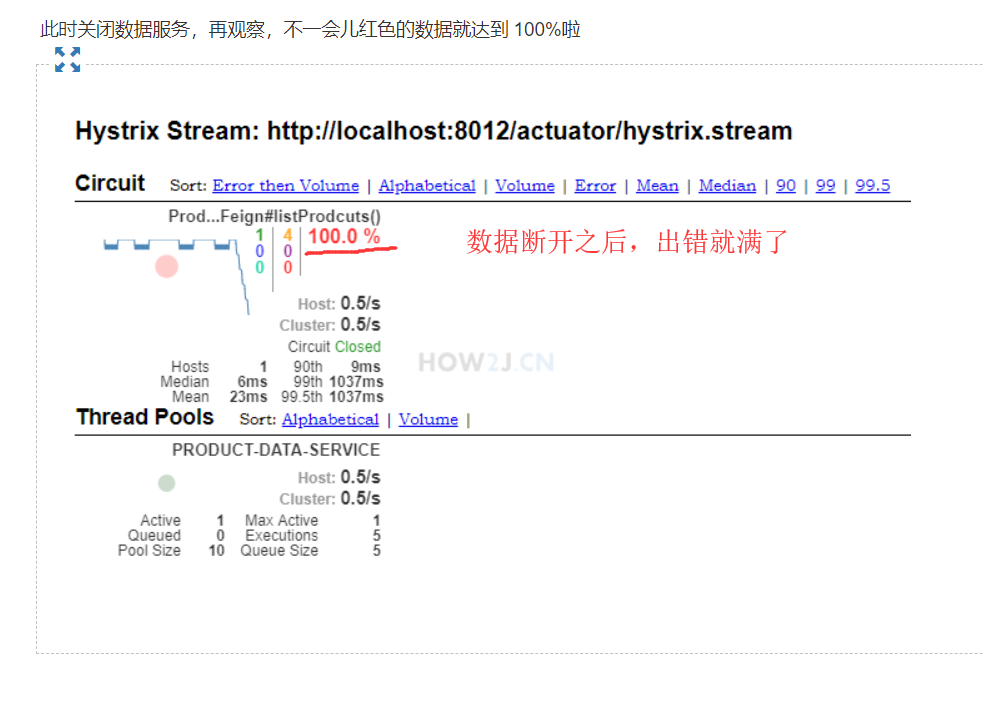


这里主要说下效果：



就是页面流量的访问情况

断开数据访问之后：



## hystrix监控集群：（集群）

上面是针对一个微服务的监控，下面讲如何对集群进行监控：

[https://how2j.cn/k/springcloud/springcloud-dashboard/2043.html#nowhere](https://how2j.cn/k/springcloud/springcloud-dashboard/2043.html" \l "nowhere)

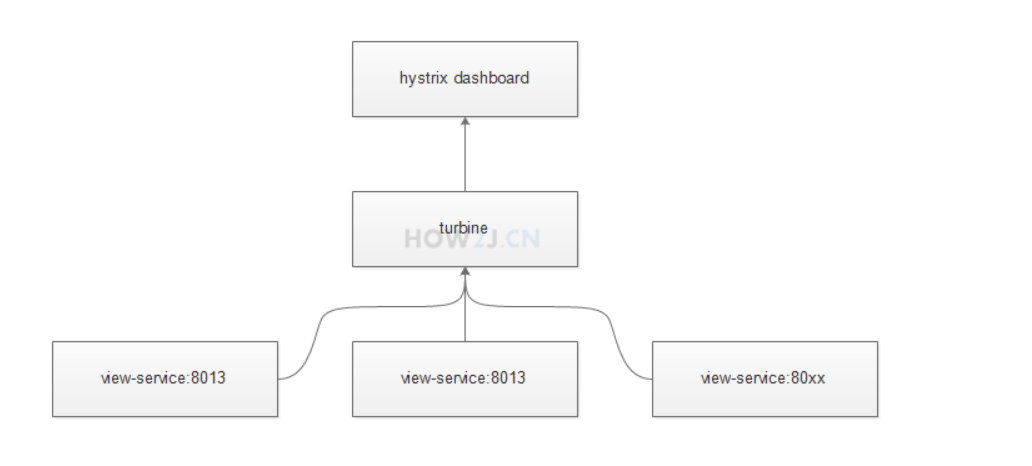
监控地址：<http://localhost:8020/hystrix>

输入：<http://localhost:8021/turbine.stream>

我也不知道为啥是8021，应该启动这个turbin的接口就在8021把

原文：

在上个知识点讲解了**针对一个微服务**的断路器监控，但是微服务通常会是多个实例组成的一个集群。 倘若集群里的实例比较多，难道要挨个挨个去监控这些实例吗？ 何况有时候，根据集群的需要，会动态增加或者减少实例，监控起来就更麻烦了。  
所以为了方便监控集群里的多个实例，springCloud 提供了一个 turbine 项目，它的作用是把一个集群里的多个实例汇聚在一个 turbine里，这个然后再在 断路器监控里查看这个 turbine, 这样就能够在集群层面进行监控啦。



启动步骤: （电脑跑不动了，意会吧，就是一个监控某个集群流量

状态的平台）

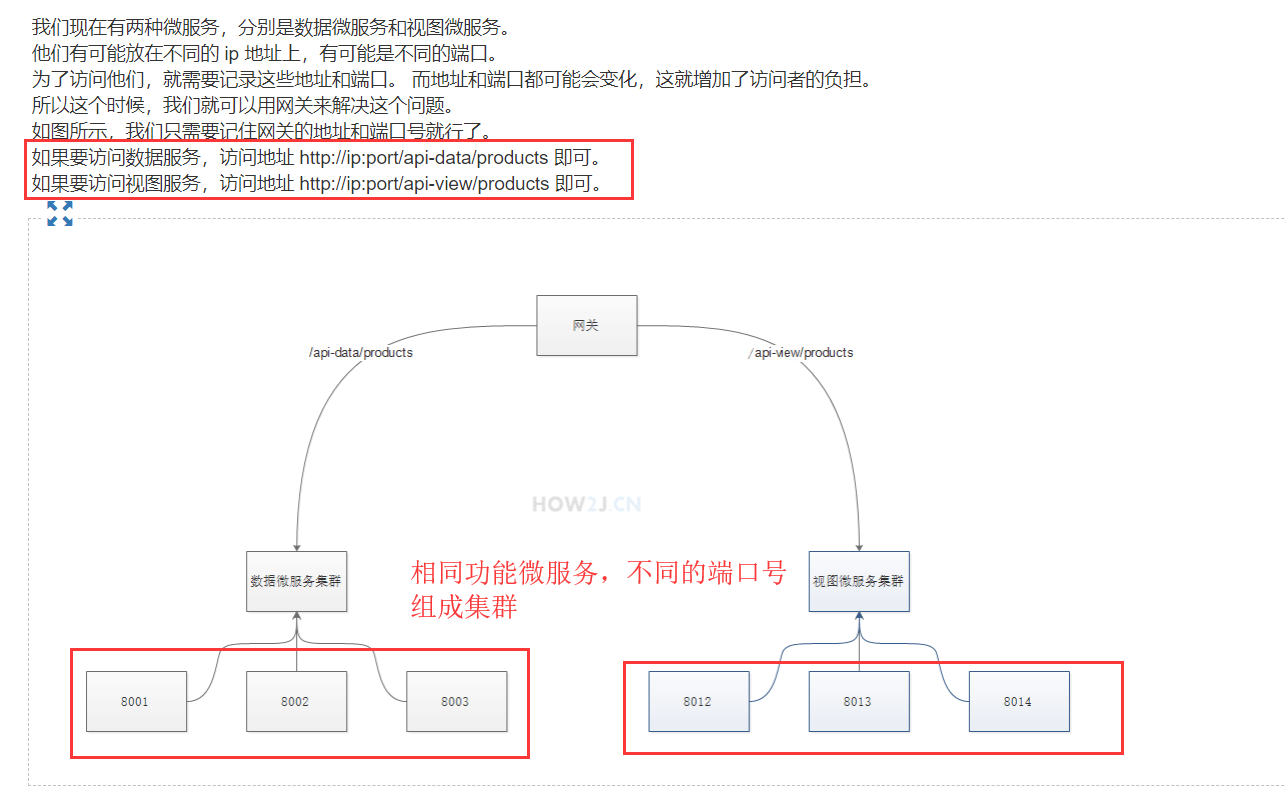
1. 首先挨个运行 EurekaServerApplication, ConfigServerApplication, ProductDataServiceApplication， ProductViewServiceFeignApplication:8012，， ProductViewServiceFeignApplication:8013，ProductServiceHystrixDashboardApplication, ProductServiceTurbineApplication.  
2. 运行视图微服务里的 AccessViewService 来周期性地访问 http://127.0.0.1:8012/products 和 http://127.0.0.1:8013/products。 因为只有访问了，监控里才能看到数据。  
3. 打开监控地址  
<http://localhost:8020/hystrix>  
4. 在最上面输入

http://localhost:8021/turbine.stream

# 12.网关路由：（直通微服务集群，不管ip号）

就是方便了不用记端口号，直接访问集群。

相同的微服务，组成一个集群，里面的微服务虽然功能相同，但是ip号不同或者端口号不同，如果外界需要访问他们的话，就需要输入不同的IP号或者端口号。网关就是用来解决这个问题的，直接访问集群，而不是端口号。

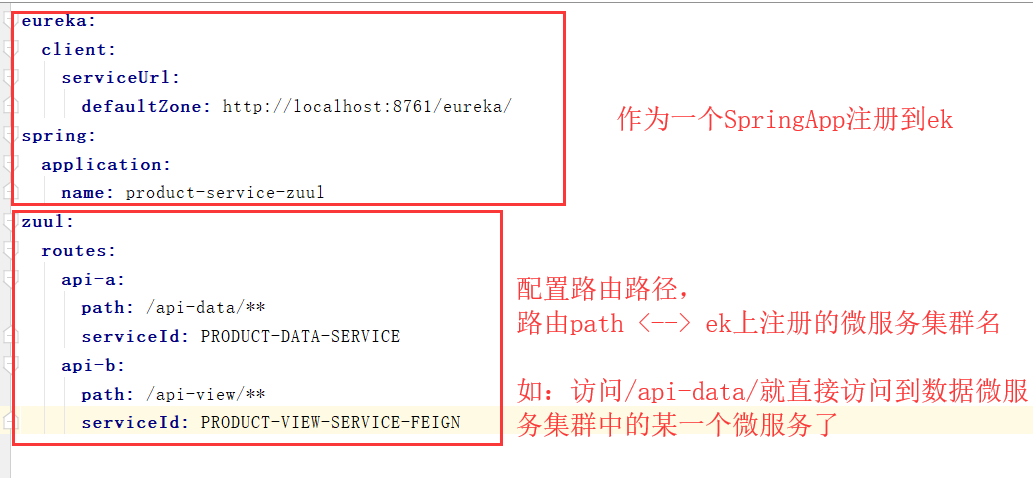


1. 一个新module zuul 放依赖：



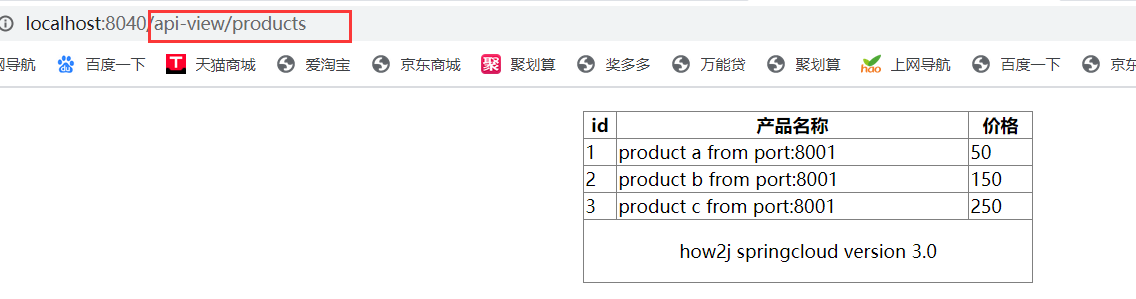
启动类就是启动一个端口，没啥特别

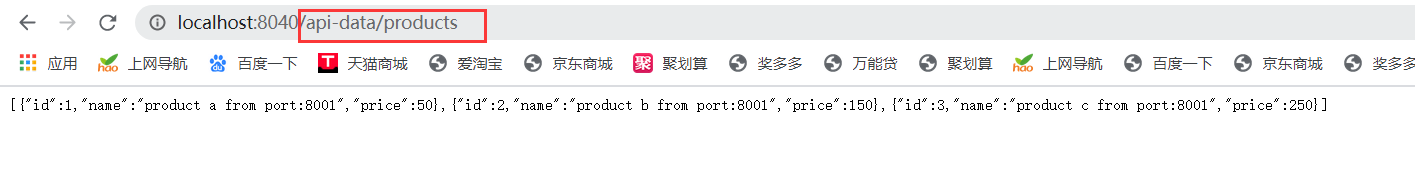
主要看配置文件：路由映射微服务集群



跟随打开zuul

效果：





不需要端口号了