# 基于微服务架构的服务治理

一、微服务与服务治理相关技术分析

二、基本功能实现

本次实验实现了商品微服务，模拟商家在销售中对自身中对自身库存进行查询、得到指定商品的库存数目以及罗列出目前仓库中所有商品的库存、进货后对商品的库存进行更改。

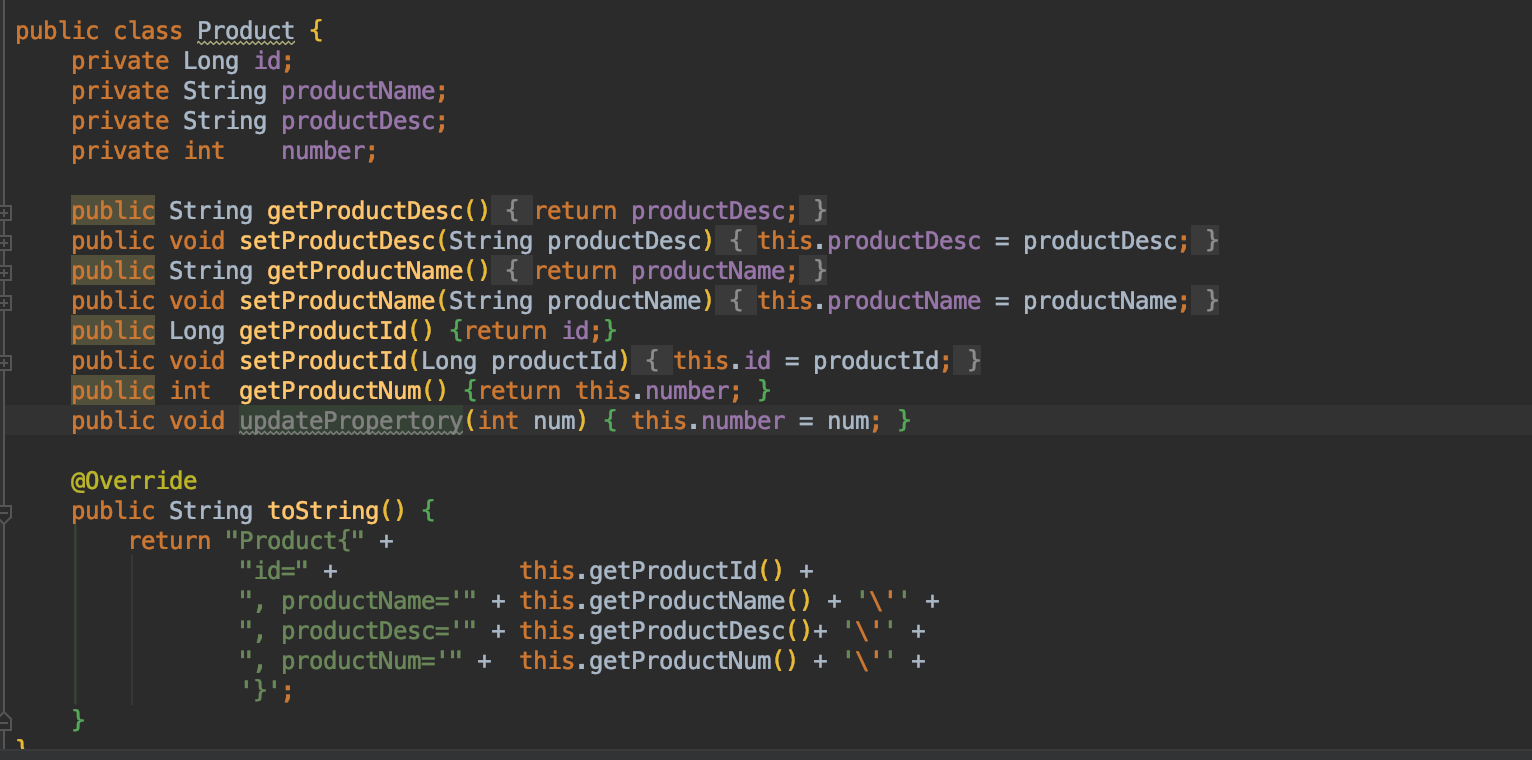
服务提供者通过查询本地的数据库获得指定商品的库存、列举指定商品的信息和所有商品的信息、进货修改本地库存的数量。在这种场景下，商品微服务是一个服务提供者，如下图所示：

图片包含 游戏机, 画, 桌子

描述已自动生成

商品类的属性包含id（唯一的标识，不能相同），商品的名称，商品的描述信息以及当前库存的数量。实验中通过product\_common实现了商品的基本类，并实现了对商品库存量的查询、商品的进货后库存的修改、根据id查询指定商品的信息。如下图所示：

图片包含 游戏机, 截图

描述已自动生成 

实验中定义的数据库如下：

图片包含 截图, 游戏机, 播放器, 男人

描述已自动生成

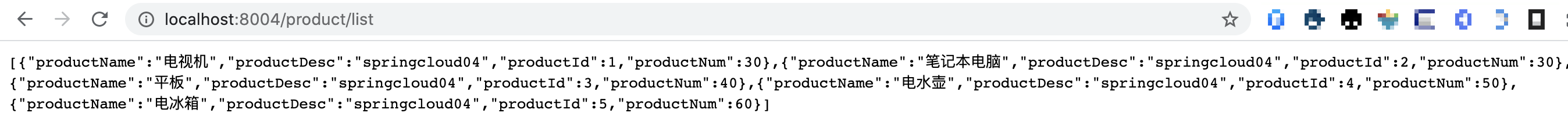
该仓库目前所拥有的的商品主要有电视机、笔记本电脑、平板、电水壶、电冰箱等，库存数量间number所在行。通过product\_provider实现了服务提供者对应的微服务，主要的操作如下：

a.查询指定商品（<http://localhost:8004/product/get/1>）：

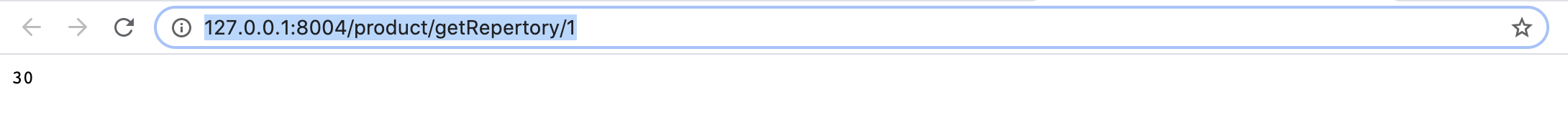
图片包含 游戏机, 截图

描述已自动生成

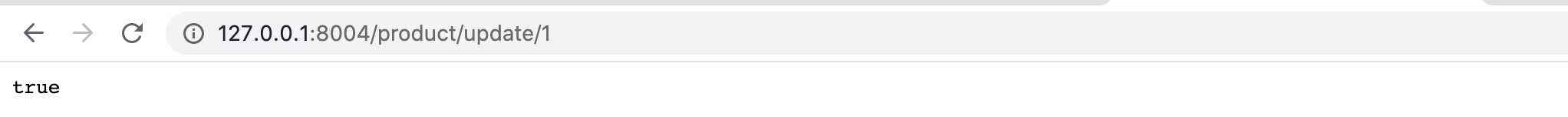
b.查询目前仓库中所有的商品情况（<http://localhost:8004/product/list>）：

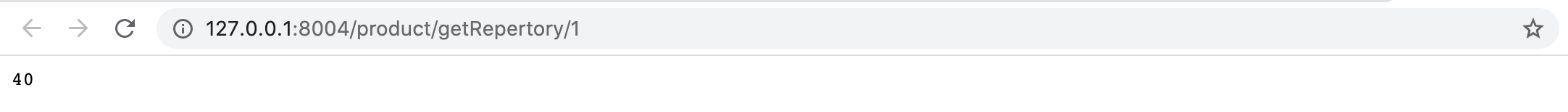


c.查询指定商品的库存(<http://127.0.0.1:8004/product/getRepertory/1>)：



d.进货后更新商品库存(<http://127.0.0.1:8004/product/update/1>)：





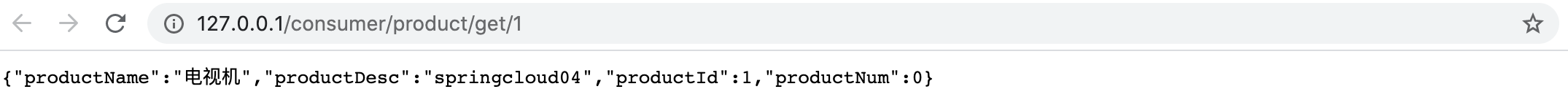
备注，此时进货默认每件货进10件，商品的库存数量增加10，返回进货成功与否。

用户不能每次都到提供者所提供的端口和地址去访问，此时用户可以通过创建服务消费者的微服务，消费者和提供者通过网络连接起来，用户通过消费者调用商品微服务的相关操作。本次实验中，通过定义product\_consumer来调用提供者所提供的服务，此时的调用关系如下：

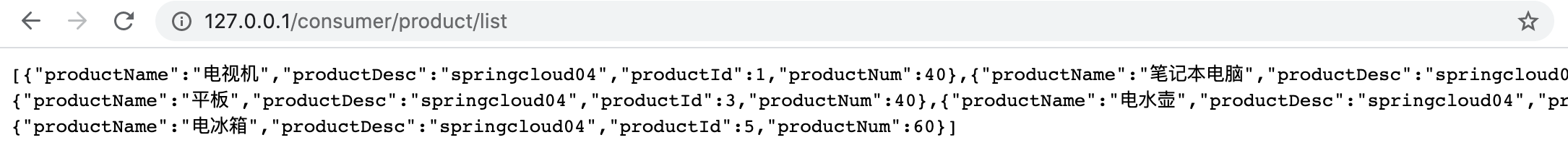


product\_cosumer在80端口通过网络访问商品微服务在8004端口提供的微服务，进一步将服务解耦，保护后端程序的安全性和隐私性，用户只能拿到提供者想让他看的。功能实现如下：

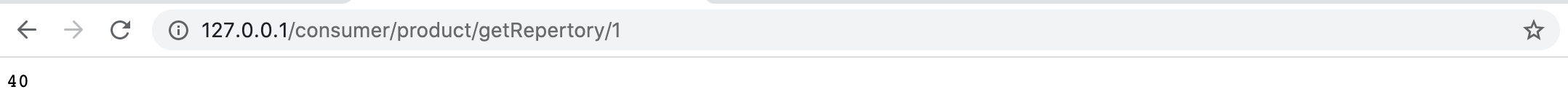
a.查询指定商品(<http://127.0.0.1/consumer/product/get/1>)



b.查询目前仓库中所有的商品情况(<http://127.0.0.1/consumer/product/list>)



c.查询指定商品的库存(<http://127.0.0.1/consumer/product/getRepertory/1>)



d.进货后更新商品库存

图片包含 游戏机, 截图, 鸟

描述已自动生成

对用户权限进行控制，不允许其通过服务消费者进货后修改库存。

三、微服务进阶功能

第二节已经实现了基本的服务和功能调用，但是随时用户访问数量的增多和服务的增多，为保证服务的健壮性和安全性，需要进一步将服务进行划分，主要实现了以下功能:

1. 服务注册（Eureka Server）

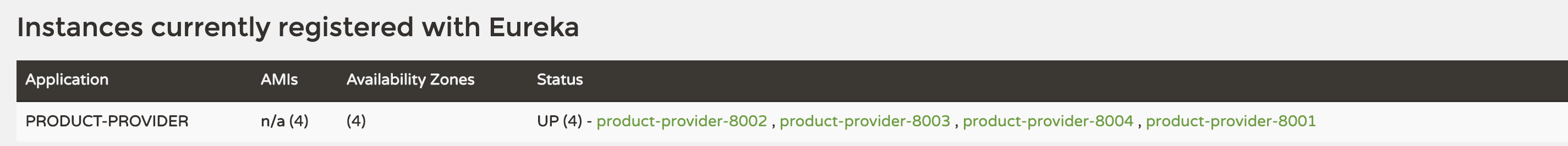
注册中心可以说是微服务架构中的”通讯录“，它记录了服务和服务地址的映射关系。在分布式架构中，服务会注册到这里，当服务需要调用其它服务时，就这里找到服务的地址，进行调用。

服务注册中心是微服务架构非常重要的一个组件，在微服务架构里主要起到了协调者的一个作用，可以实现服务注册和发现、服务配置和服务健康监测等功能。主要的服务注册中心有Zookeeper, Eureka, Consul, Nacos, 本次实验主要基于经典的Zookeeper进行实现。

图片包含 游戏机, 标志

描述已自动生成

本次实验由四个同名的商品微服务分别在8001、8002、8003、8004端口提供微服务，他们将向服务注册中心（端口为7001）注册服务，用户可以根据服务名称在注册中心查询得到服相应的服务并调用服务。

注册结果如下所示：  


如图所示，四个名称为product-provider的服务已经注册到7001的注册中心，后续可以直接通过该名称访问该服务。

2. 服务发现与使用（Eureka Client）

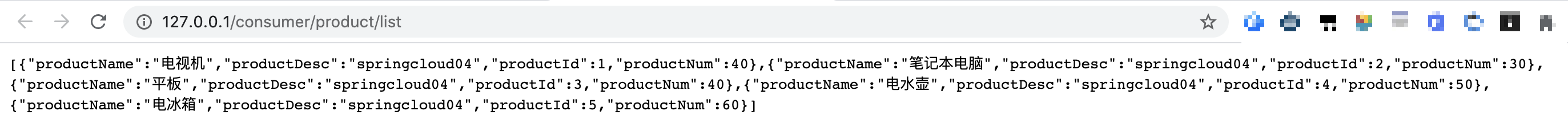
第二节客户端在80端口通过客户消费者访问服务提供者的所提供的商品微服务，但消费者到提供者是通过连接提供者的ip静态连接的。静态连接的方式使得服务方ip变动以后客户端必须相应作出修改，否则不能访问；另外，当服务方较多时需要多次绑定且容易出错。本实验基于3.1节的服务注册，通过服务提供者提供的商品名称，动态匹配服务提供者的路由，增加了灵活性和可用性。

实现过程的部分代码如下图：

图片包含 截图, 室内, 屏幕, 监控

描述已自动生成

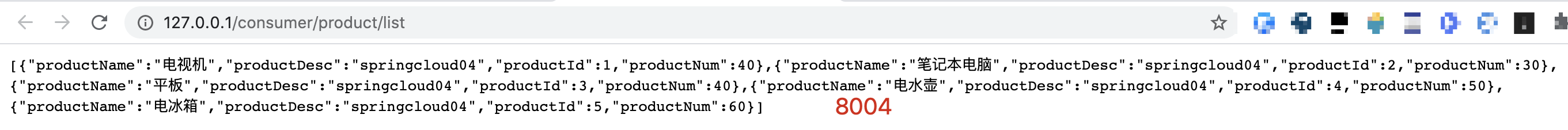
客户端通过消费者访问的结果如下：

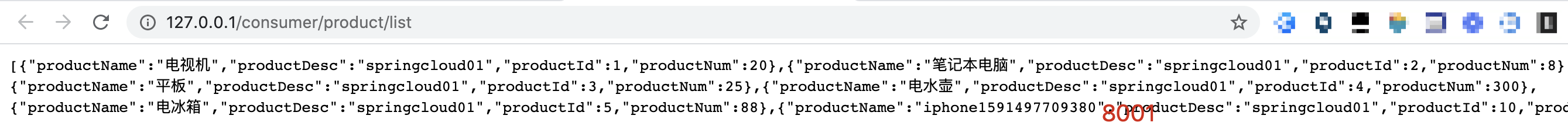


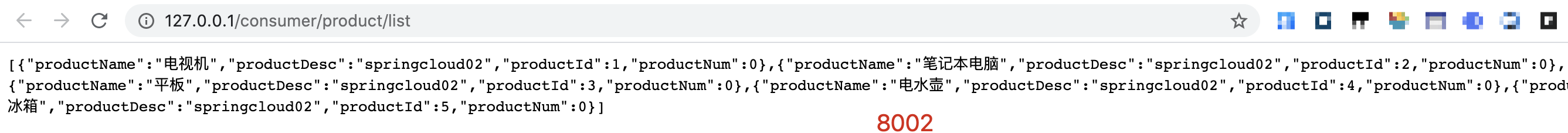
此时通过查询注册中心注册服务动态匹配访问的服务提供者地址，结果正确。

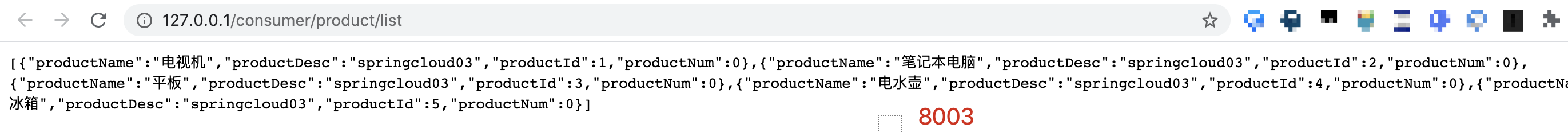
3. 负载均衡（Ribbon和Feign）

目前已经实现了服务的注册和服务发现。当启动某个服务的时候，可以通过HTTP的形式将服务注册到注册中心，并且可以通过SpringCloud提供的工具获取注册中心的服务列表。但是当访问请求逐渐增多时如何选择后端的微服务，这就是负载均衡需要解决的问题。本次实验实现了基于Ribbon和Feign的负载均衡方法，服务提供者有四个，采用“轮询”的负载均衡策略顺序分配客户端的调用请求，执行结果如下：

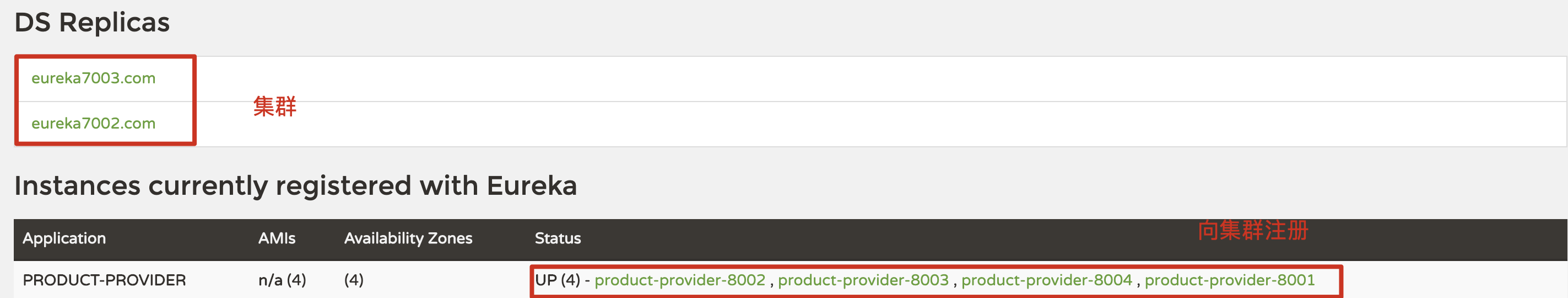


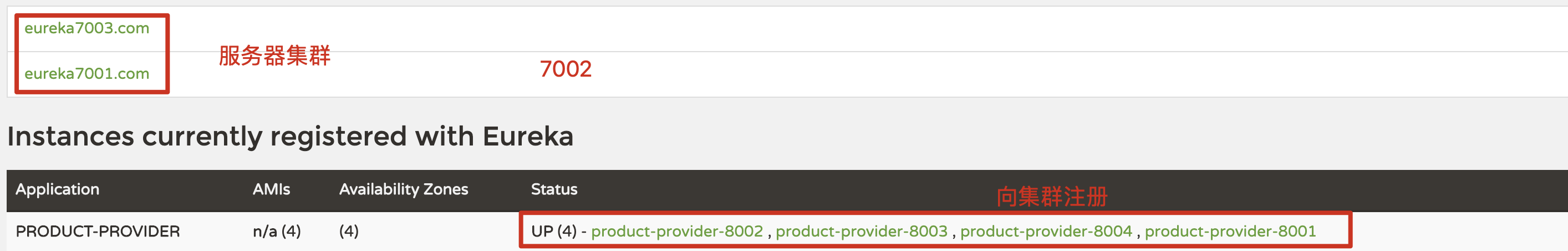


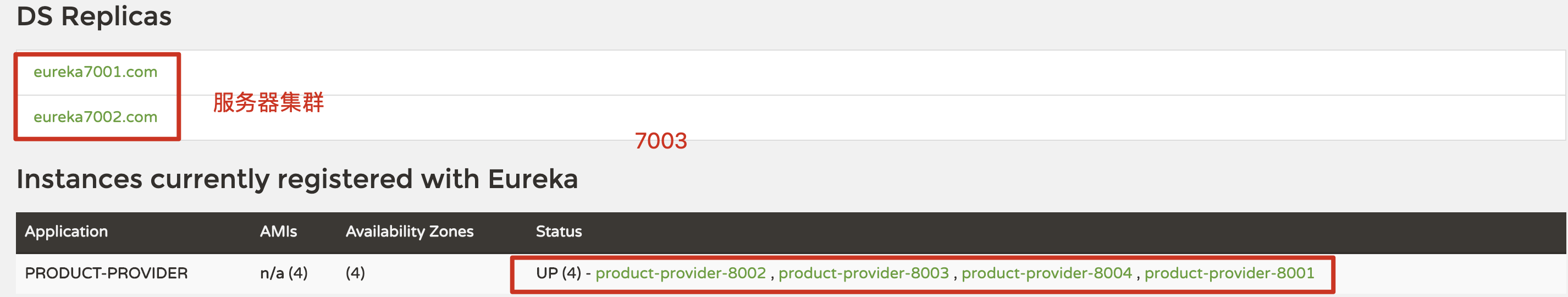


4.注册中心集群化

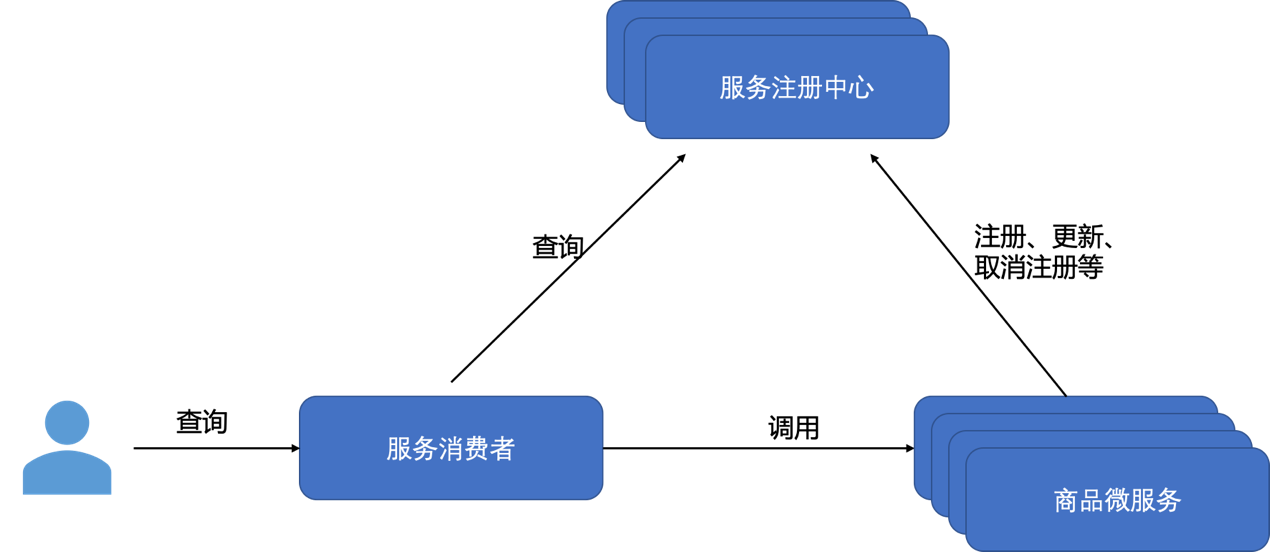
服务提供者需要将自己所提供的服务注册到注册中心，服务消费者需要访问服务注册中心得到所请求服务的地址。此时，Eureka服务注册中心一旦被破坏，整个系统就不能正常工作。为解决注册中心被破坏或者受到拒绝服务攻击使得系统瘫痪，本实验将注册中心由一个扩展成三个注册中心组成的注册中心集群，服务注册时向集群注册并且请求服务时向集群查询，任意注册中心被破坏后不影响系统的整体作用，并且注册查询的效率有所提升。结果如下所示：







如上图所示，服务注册中心由原来的7001端口的单个微服务，组合形成了三个注册中心组成的的集群（端口分别是7001，7002和7003）。整体的体系结构如下图所示：



5. 服务熔断（Hystrix）

6. 服务监控（Dashboard）

7. 网关服务（Zuul）

8. 服务过滤（Zuul Filter）

9. 服务限流（Gateway Filter）

10. 服务降级（Feign Circuit）

11. 安全（Security）