

一、实验目的

1.1 相关背景

传统的 Java Web 都是采用单体架构的方式来进行开发、部署、运维的，所谓单体架构就是将 Application 的所有业务模块全部打包在一个文件中进行部署。但是随着互联网的发展、用户数量的激增、业务的极速扩展，传统的单体应用方式的缺点就逐渐显现出来了，给开发、部署、运维都带来了极大的难度，工作量会越来越大，难度越来越高。

因为在单体架构中，所有的业务模块的耦合性太高，耦合性过高的同时项目体量又很大势必会给各个技术环节带来挑战。项目越进行到后期，这种难度越大，只要有改动，整个应用都需要重新测试，部署，极大的限制了开发的灵活性，降低了开发效率。同时也带来了更大的安全隐患，如果某个模块发生故障无法正常运行就有可能导致整个项目崩溃，单体应用的架构如下图所示。

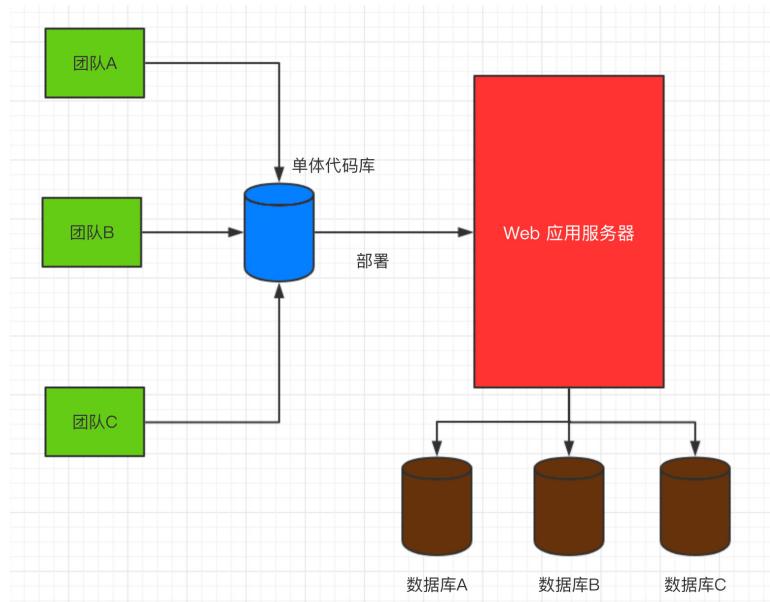


图 1 单体应用的架构

随着业务的发展，开发变得越来越复杂。修改、新增某个功能，需要对整个系统进行测试，重新部署。一个模块出现问题，很可能导致整个系统崩溃。多团队同时对数据进行管理，容易产生安全漏洞。各模块使用同一种技术框架，很难根据具体业务需求选择更合适的框架，局限性太大。模块内容太复杂，如果员工离职，可能需要很长时间才能完成任务交接。

为了解决上述问题，微服务架构应运而生，简单来说，微服务就是将一个单

体应用拆分成若干个小型的服务，协同完成系统功能的一种架构模式，在系统架构层面进行解耦合，将一个复杂问题拆分成若干个简单问题。这样的好处是对于每一个简单问题，开发、维护、部署的难度就降低了很多，可以实现自治，可自主选择最合适的技术框架，提高了项目开发的灵活性。

微服务就像一个饭店，原本一个厨师负责切菜洗锅颠勺烹饪卤煮所有的活，现在把厨房里的活分模块的包给好多个厨师专门负责，然后多个厨师互相协作维持整个厨房的工作流程。

微服务架构不仅是单纯的拆分，拆分之后的各个微服务之间还要进行通信，否则就无法协同完成需求，也就失去了拆分的意义。不同的微服务之间可以通过某种协议进行通信，相互调用、协同完成功能，并且各服务之间只需要制定统一的协议即可，至于每个微服务是用什么技术框架来实现的，统统不需要关心。这种松耦合的方式使得开发、部署都变得更加灵活，同时系统更容易扩展，降低了开发、运维的难度，单体应用拆分之后的架构如下图所示。

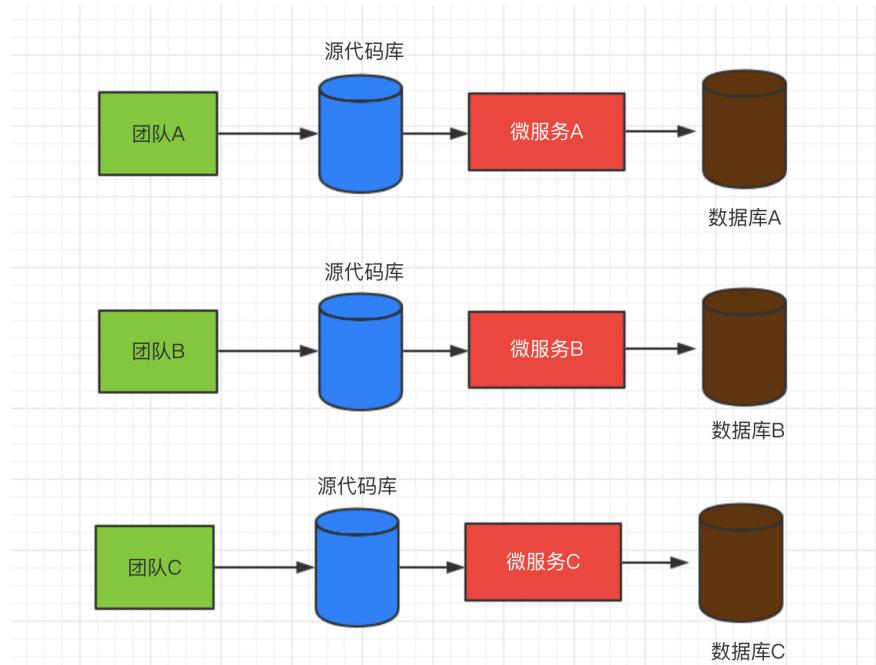


图 2 单体应用拆分之后的架构

1.2 微服务架构的核心组件

1.2.1 服务治理(服务注册、服务发现)

拆分之后的微服务首先需要完成的工作就是实现服务治理，包括服务注册和服务发现。这里我们把微服务分为两类：提供服务的叫做服务提供者，调用服务

的叫做服务消费者。一个服务消费者首先需要知道有哪些可供调用的微服务，以及如何来调用这些微服务。所以就需要将所有的服务提供者在注册中心完成注册，记录服务信息，如 IP 地址、端口等，然后服务消费者可以通过服务发现获取服务提供者的这些信息，从而实现调用。

1.2.2 服务负载均衡

微服务间的负载均衡是必须要考虑的，服务消费者在调用服务提供者的接口时，可根据配置选择某种负载均衡算法，从服务提供者列表中选择具体要调用的实例，从而实现服务消费者与服务提供者之间的负载均衡。

1.2.3 微服务网关

在一个微服务架构中会包含多个微服务实例，每个微服务实例都有其特定的网络信息，如 IP 地址、端口等，很多时候完成一个需求需要多个微服务来协同工作，那么客户端就需要频繁请求不同的 URL，不便于统一管理，可以使用微服务网关来解决这一问题，为客户端提供统一的入口，系统内部由网关将请求映射到不同的微服务。

1.2.4 微服务容错

前面我们提到过，各个服务之间是通过远程调用的方式来完成协作任务的，如果因为某些原因使得远程调用出现问题，导致微服务不可用，就有可能产生级联反应，造成整个系统崩溃。这个问题我们可以使用微服务的熔断器来处理，熔断器可以防止整个系统因某个微服务调用失败而产生级联反应导致系统崩溃的情况。

1.2.5 分布式配置

每个微服务都有其对应的配置文件，在一个大型项目中管理这些配置文件也是工作量很大的一件事情，为了提高效率、便于管理，我们可以对各个微服务的配置文件进行集中统一管理，将配置文件集中保存到本地系统或者 Git 仓库，再由各个微服务读取自己的配置文件。

1.2.6 服务监控

一个分布式系统中往往会部署很多个微服务，这些服务彼此之间会相互调用，整个过程就会较为复杂，我们在进行问题排查或者优化的时候工作量就会比较大。如果我们能准确跟踪到每一个网络请求，了解它整个运行流程，经过了哪

些微服务、是否有延迟、耗费时间等，这样的话我们分析系统性能，排查解决问题就会容易很多，这就是服务监控。

1.3 Spring Cloud

微服务是一种分布式软件架构设计方式，具体的落地框架有很多，如阿里的 Dubbo、Google 的 gRPC、新浪的 Motan、Apache 的 Thrift 等，都是基于微服务实现分布式架构的技术框架，本课程我们选择的框架的是 Spring Cloud，Spring Cloud 基于 Spring Boot 使得整体的开发、配置、部署都非常方便，可快速搭建基于微服务的分布式应用，Spring Cloud 相当于微服务各组件的集成者。

Spring Boot 和 Spring Cloud 的关系可大致理解为，Spring Boot 快速搭建基础系统，Spring Cloud 在此基础上实现分布式系统中的公共组件，如服务注册、服务发现、配置管理、熔断器、控制总线等，服务调用方式是基于 REST API，整合了各种成熟的产品和架构。

对于 Java 开发者而言，Spring 框架已成为事实上的行业标准，Spring 全家桶系列产品优势在于功能齐全、简单好用、性能优越、文档规范，实际开发中就各方面综合因素来看，Spring Cloud 是微服务架构中非常优秀的实现方案，本课程就将为各位读者详细解读如何快速上手 Spring Cloud，Spring Cloud 的核心组件如下图所示。

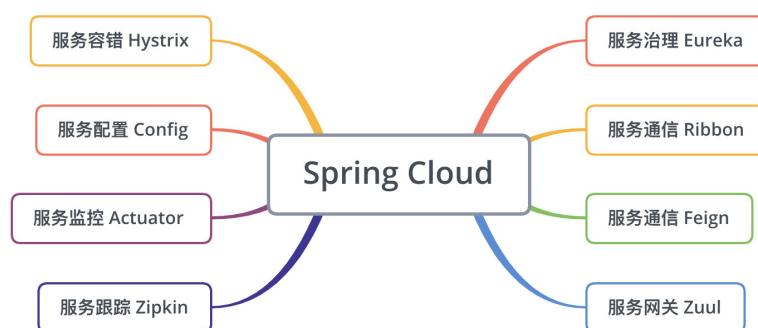


图 3 Spring Cloud 的核心组件

服务治理的核心组成有三部分：服务提供者，服务消费者，注册中心。

在分布式系统架构中，每个微服务（服务提供者、服务消费者）在启动时，将自己的信息存储在注册中心，把这个过程称之为服务注册。服务消费者要调用服

务提供者的接口，就得找到服务提供者，从注册中心查询服务提供者的网络信息，并通过此信息来调用服务提供者的接口，这个过程就是服务发现。

既然叫服务治理就不仅仅是服务注册与服务发现，同时还包括了服务的管理，即注册中心需要对记录在案的微服务进行统一管理，如何来具体实现管理呢？各个微服务与注册中心通过心跳机制完成通信，每间隔一定时间微服务就会向注册中心汇报一次，如果注册中心长时间无法与某个微服务通信，就会自动销毁该微服务。当某个微服务的网络信息发生变化时，会重新注册。同时，微服务可以通过客户端缓存将需要调用的服务地址保存起来，并通过定时任务更新的方式来保证服务的时效性，这样可以降低注册中心的压力，如果注册中心出现问题，也不会影响微服务之间的调用。

1.4 实验目的

本实验使用 Spring Cloud 框架完成电商项目的商品管理微服务。包含了 Spring Cloud 核心子项目：

1. Spring Cloud Netflix：核心组件，可以对多个 Netflix OSS 开源套件进行整合，包括以下几个组件：
 - Eureka：服务治理组件，包含服务注册与发现；
 - Hystrix：容错管理组件，实现了熔断器；
 - Ribbon：客户端负载均衡的服务调用组件；
 - Feign：基于 Ribbon 和 Hystrix 的声明式服务调用组件；
 - Zuul：网关组件，提供智能路由、访问过滤等功能；
 - Archaius：外部化配置组件。
2. Spring Cloud Config：配置管理工具，实现应用配置的外部化存储，支持客户端配置信息刷新、加密/解密配置内容等；

通过对上述 Spring Cloud 核心项目的使用，能够提高对这一类微服务的使用能力。

二、实验内容

功能要求：商品的增删改查，其中查需要有按照商品 ID、厂家 ID、价格等查询，最好能设计实现多条件查询，要求使用 Eureka 注册中心，使用 Feign。（至少两个注册中心，两个生产者，都需配置成负载均衡）。

进阶要求：在项目中使用容错（包括 DashBoard 和 TURBINE）、路由网关（ZUUL）、配置中心（Config）、扩展功能。

三、实验环境

MySql、JKD、IDEA、Google Chrome。

四、实验过程与分析

4.1 数据库(MySql)

首先建立一个数据库，有 goodsId、goodsName、goodsPrice、goodsStock、goodsFactory 这 5 个字段名，代表了商品 ID、商品名、商品价格、商品库存、商品供应商（ID）。并且插入了 10 条数据，如下图。

goodsId	goodsName	goodsPrice	goodsStock	goodsFactory
1	手机	999	423	小米
2	电脑	4999	234	联想
3	鼠标	129	253	罗技
4	AK-47	4999	234	北方工业
5	海洋之心	999999	1	英国石材厂
6	光刻机	9999999	2	ASML
7	哈哈	20	56	国家
8	哈哈	20	56	国家
9	傻妞	24	1	北京

图 4 goods 数据库

4.2 服务提供者、服务消费者、注册中心的关联

4.2.1 服务治理（Eureka、Provider）

首先启动 3 个 Eureka 注册中心。再启动 3 个服务提供者 Provider，在注册中心注册可以提供的服务。

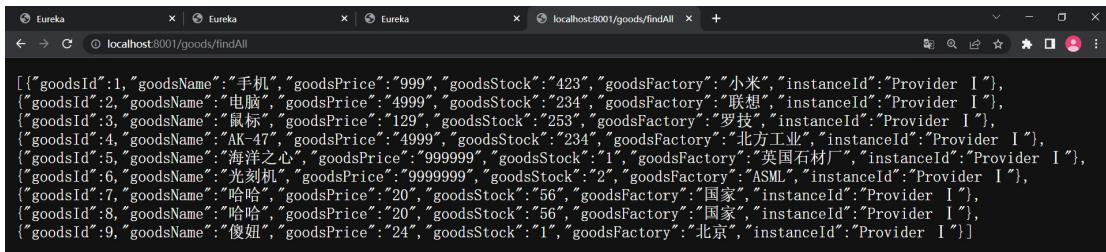
The screenshot shows the Spring Eureka registration center interface running on localhost:7001. It includes sections for System Status, DS Replicas, Instances currently registered with Eureka, and General Info.

- System Status:** Shows environment as 'test', data center as 'default', current time as 2022-06-06T14:01:37+0800, uptime as 00:01, lease expiration enabled as true, renew threshold as 6, and renew count as 3.
- DS Replicas:** Lists two replicas: eureka2.com and eureka3.com.
- Instances currently registered with Eureka:** A table showing registered instances. One instance named 'PROVIDER' is listed under Application, AMIs, Availability Zones, and Status columns.
- General Info:** A table with columns Name and Value.

图 5 Eureka 注册中心面板

然后在浏览器中访问服务提供者 Provider 提供的服务：

1. 查询所有信息: <http://localhost:8001/goods/findAll>



```
[{"goodsId":1,"goodsName":"手机","goodsPrice":999,"goodsStock":423,"goodsFactory":"小米","instanceId":"Provider I"}, {"goodsId":2,"goodsName":"电脑","goodsPrice":4999,"goodsStock":234,"goodsFactory":"联想","instanceId":"Provider I"}, {"goodsId":3,"goodsName":"鼠标","goodsPrice":129,"goodsStock":253,"goodsFactory":"罗技","instanceId":"Provider I"}, {"goodsId":4,"goodsName":"AK-47","goodsPrice":4999,"goodsStock":234,"goodsFactory":"北方工业","instanceId":"Provider I"}, {"goodsId":5,"goodsName":"海洋之心","goodsPrice":999999,"goodsStock":1,"goodsFactory":"英国石材厂","instanceId":"Provider I"}, {"goodsId":6,"goodsName":"光刻机","goodsPrice":9999999,"goodsStock":0,"goodsFactory":"ASML","instanceId":"Provider I"}, {"goodsId":7,"goodsName":"哈哈","goodsPrice":20,"goodsStock":56,"goodsFactory":"国家","instanceId":"Provider I"}, {"goodsId":8,"goodsName":"哈哈","goodsPrice":20,"goodsStock":56,"goodsFactory":"国家","instanceId":"Provider I"}, {"goodsId":9,"goodsName":"傻妞","goodsPrice":24,"goodsStock":1,"goodsFactory":"北京","instanceId":"Provider I"}]
```

图 6 查询所有信息

可以看到，浏览器中显示了查询到的所有商品信息。

2. 根据商品 ID 查询商品: <http://localhost:8002/goods/findById/1>



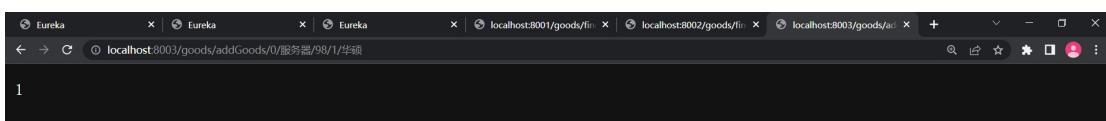
```
{"goodsId":1,"goodsName":"手机","goodsPrice":999,"goodsStock":423,"goodsFactory":"小米","instanceId":"Provider II"}
```

图 7 根据商品 ID 查询商品

同样地，浏览器显示了通过商品 ID 查询到的一条记录。

3. 插入新的商品信息:

<http://localhost:8003/goods/addGoods/0/%E6%9C%8D%E5%A1%A5%99%A8/98/1/%E5%8D%8E%E7%A1%95>



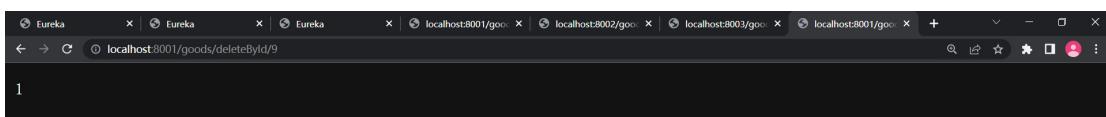
```
1
```

图 8 插入新的商品信息

最后，浏览器显示了插入成功后返回的信息，代表插入成功。

4. 根据商品 ID 删除商品:

<http://localhost:8001/goods/deleteById/9>



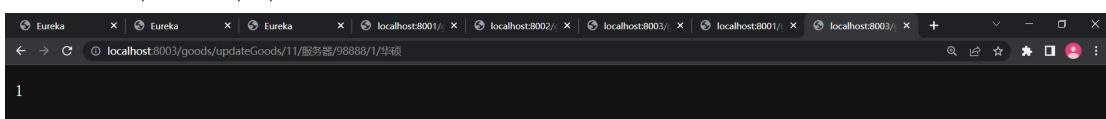
```
1
```

图 9 根据商品 ID 删除商品

最后，浏览器显示了删除成功后返回的信息，代表删除成功。

5. 根据商品 ID 修改商品信息:

<http://localhost:8003/goods/updateGoods/11/%E6%9C%8D%E5%A1%A5%99%A8/98888/1/%E5%8D%8E%E7%A1%95>



```
1
```

图 10 根据商品 ID 修改商品信息

最后，浏览器显示了修改成功后返回的信息，代表修改成功。

6. 根据商品价格查询商品:

<http://localhost:8002/goods/findByPrice/99>

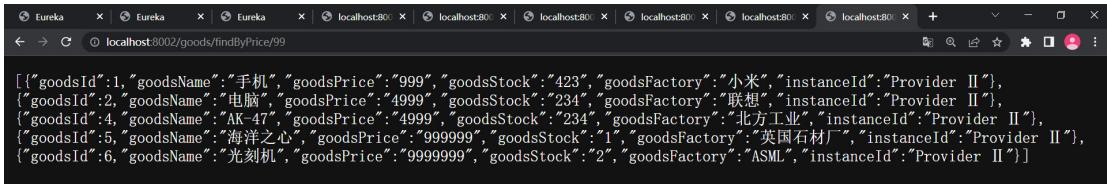


图 11 根据商品价格查询商品

同样地，浏览器显示了通过商品价格查询到的多条记录。

7. 根据商品供应商 (ID) 查询商品：

<http://localhost:8002/goods/findByFactory/%E5%B0%8F%E7%B1%B3>

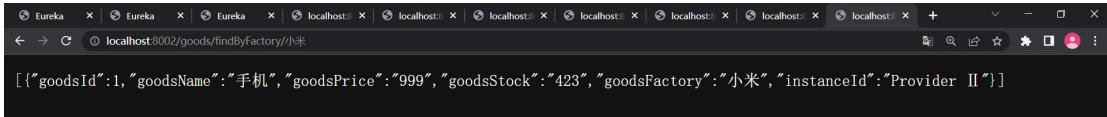


图 12 根据商品供应商 (ID) 查询商品

同样地，浏览器显示了通过商品供应商 (ID) 查询到的一条记录。

4.2.2 服务负载均衡 (Feign)

再启动 Feign，调用任意接口，这里以查询商品 ID 为例：

<http://localhost/consumer/findById/1>，并且多次刷新。

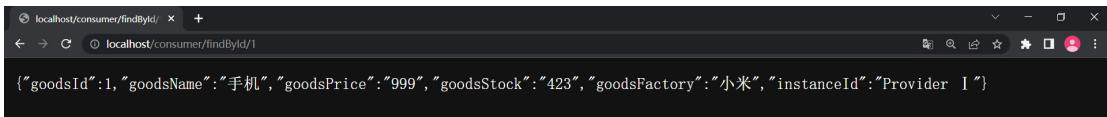


图 13 第一次刷新

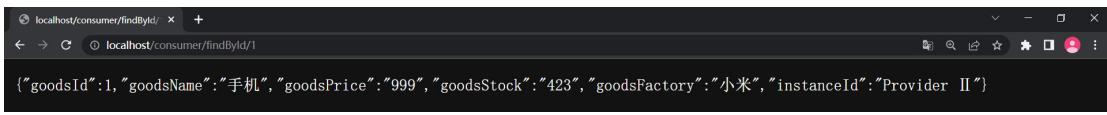


图 14 第二次刷新

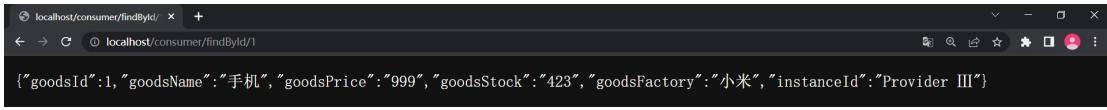


图 15 第三次刷新

可以看到，服务提供者在轮流变化。

4.2.3 微服务容错 (Breaker)

关闭 Feign 和 provider1。启动 Breaker 服务。访问：

<http://localhost:8001/goods/findById/1>

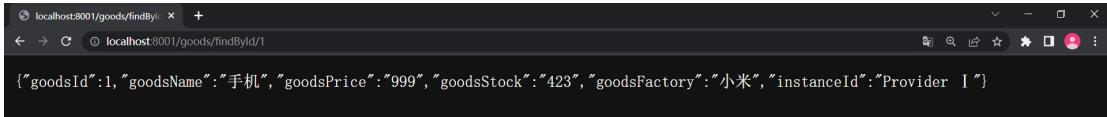


图 16 正常访问

可以看到，可以正常访问。但是当访问一个不存在的记录时，比如访问：

<http://localhost:8001/goods/findById/999>

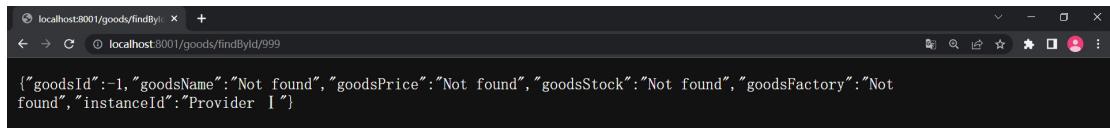


图 17 “Not found”

可以看到，浏览器显示了“Not found”的信息。代表访问不到信息。

4.2.4 微服务熔断 (FallBack)

启动 FallBack 服务。访问：<http://localhost/consumer/findAll>

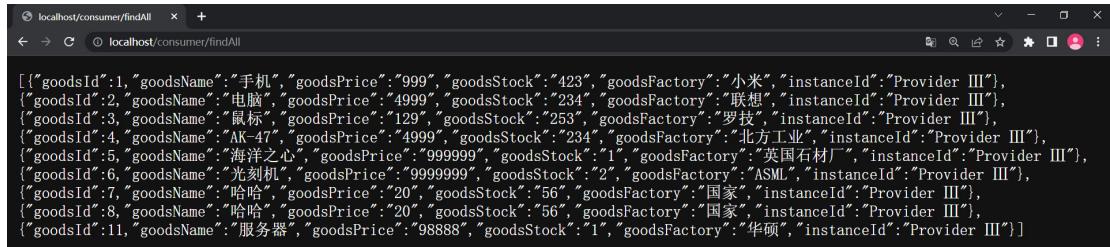


图 18 正常访问

可以看到能够访问到信息。当关闭所有 provider 时，再次访问：

<http://localhost/consumer/findAll>



图 19 “Not found”

可以看到，服务被熔断了，返回了“Not found”的信息。

4.2.5 服务监控 (DashBoard)

启动三个 provider 和 DashBoard 服务。访问：

<http://localhost:9001/hystrix>

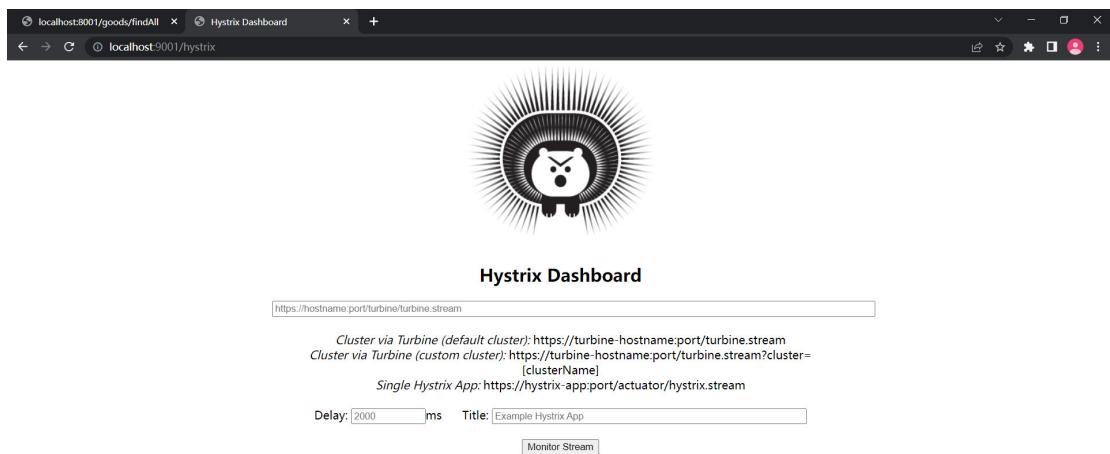


图 20 DashBoard 访问面板

在其中输入 `http://localhost:8001/hystrix.stream` 和 `test8001` 点击检测，并且在检测期间多次访问 `http://localhost:8001/goods/findById/1` 来调用服务：

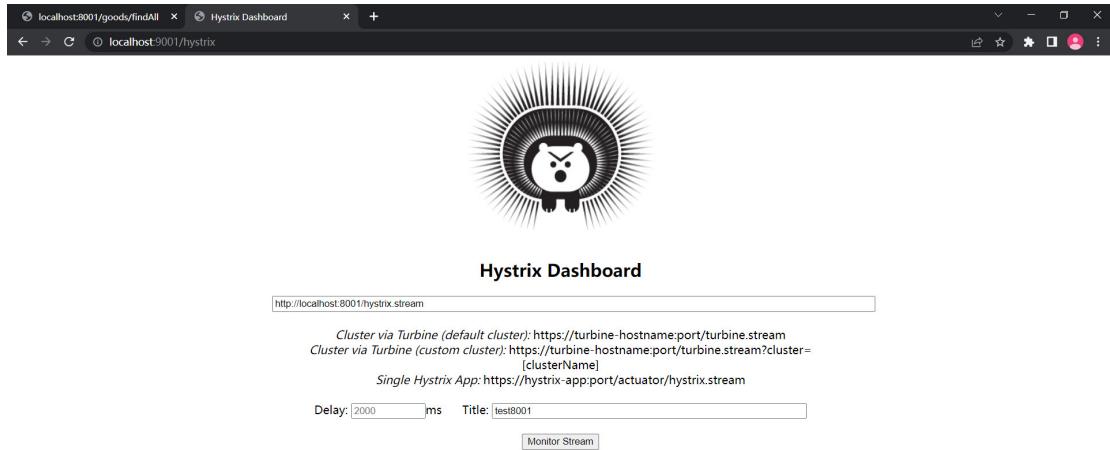


图 21 输入监测信息

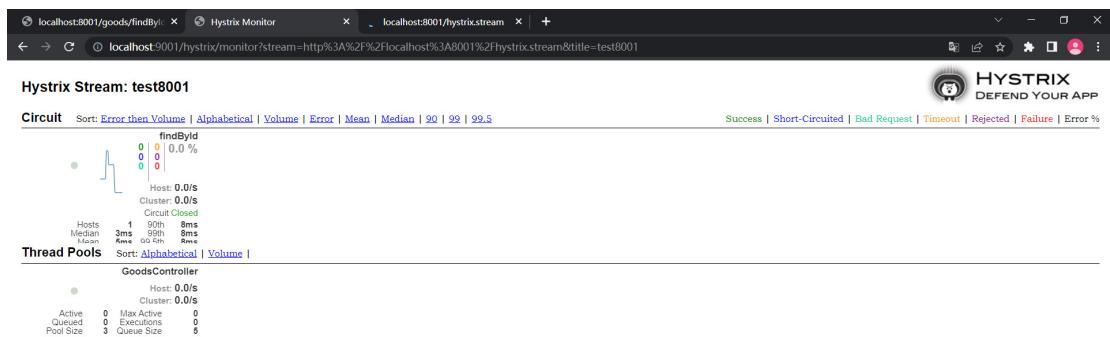


图 22 DashBoard 监测面板

可以看到，控制面板上显示了数据，还有图表，显示了访问的频率。

启动 Turbine。打开

`http://localhost:8001/goods/findById/1`

`http://localhost:8002/goods/findByPrice/99`

`http://localhost:8003/goods/findByFactory/%E5%B0%8F` 和

`http://localhost:8001/hystrix.stream`

`http://localhost:8002/hystrix.stream`

`http://localhost:8003/hystrix.stream`

最后打开 `http://localhost:9001/hystrix`

输入 `http://localhost:9002/turbine.stream` 和 `testTurbine` 信息

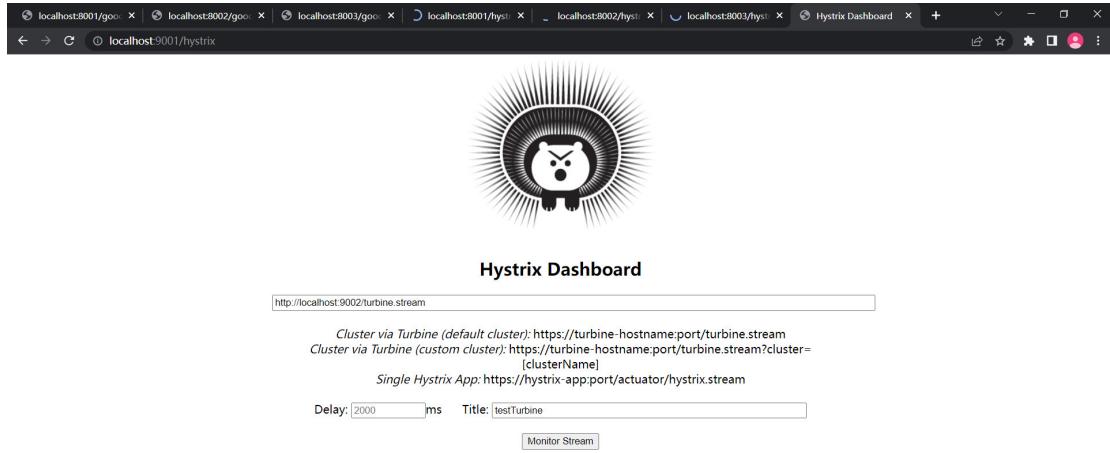


图 23 输入信息

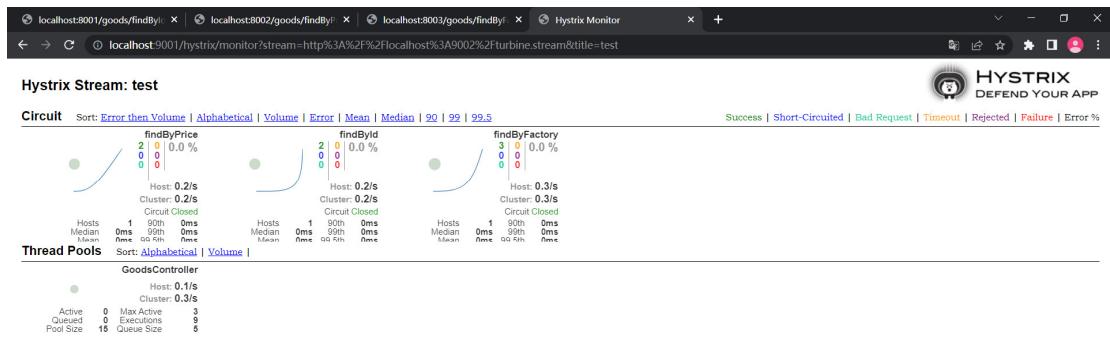


图 24 Turbine 的检测面板

然后到了 Turbine 的检测面板。可以看到访问频率信息。

4.2.6 微服务网关 (Zuul)

启动 GateWay。访问: <http://localhost:8001/goods/findById/1>

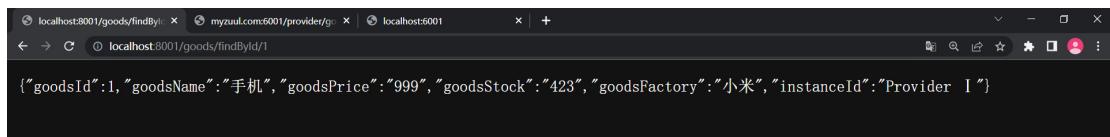


图 25 正常访问

可以正常访问。关闭过滤功能的注解@Component 和注释掉配置文件的 zuul 功能。重启 GateWay，访问:

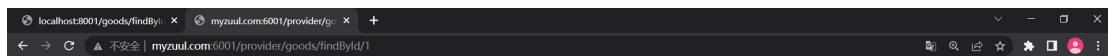
<http://myzuul.com:6001/provider/goods/findById/1>



图 26 路由访问

可以看到，可以通过路由来访问服务。现在开启过滤功能，访问:

http://myzuul.com:6001/provider/goods/findById/1



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Jun 06 16:40:43 CST 2022

There was an unexpected error (type=Not Found, status=404).

图 27 访问失败

可以看到，不能访问。加上 Token 再次访问：

http://myzuul.com:6001/ckp/mall/goods/findById/1

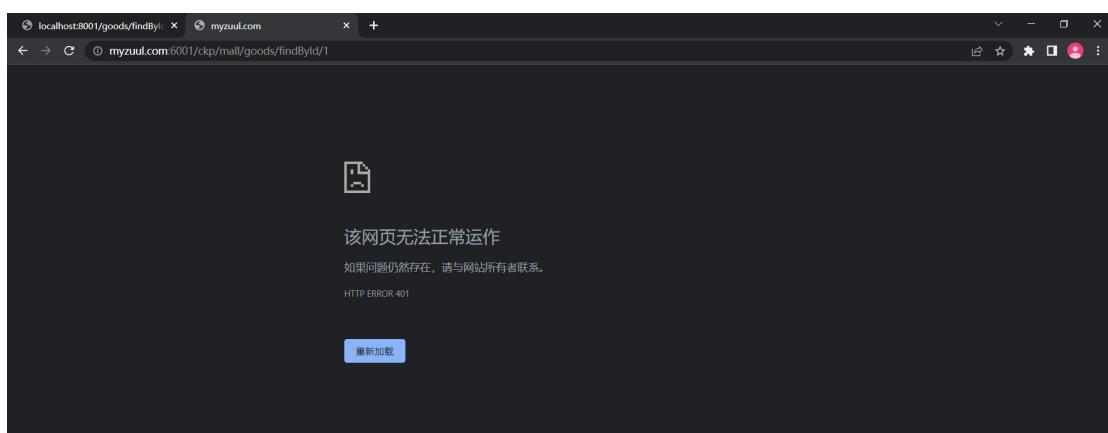


图 28 访问失败

可以看到不能访问，并且控制台输出：

```
accessToken:null  
accessToken1 response code:401
```

图 29 控制台输出

加上 accessToken 再次访问：

http://myzuul.com:6001/ckp/mall/goods/findById/1?accessToken=zuul

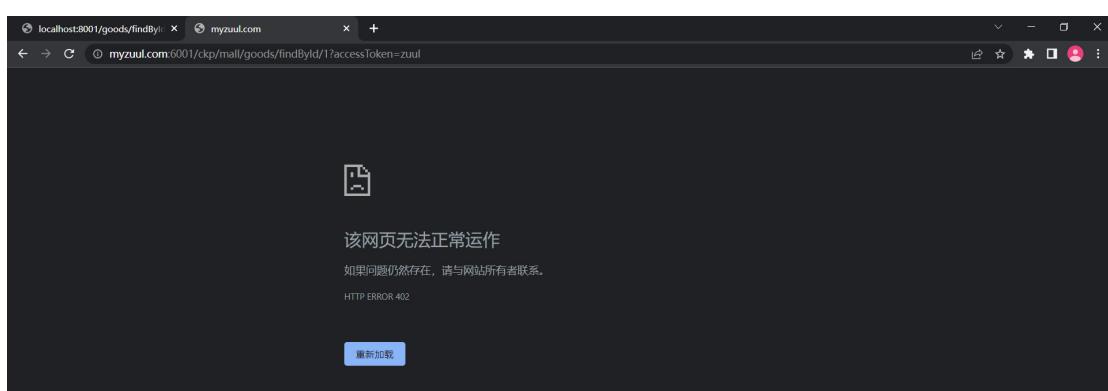


图 30 访问失败

可以看到，还是不能访问。并且控制台输出：

```
accessToken:zuul
accessToken1 response code:500
accessToken1:zuul
```

图 31 控制台输出

换一个 accessToken 再次访问：

<http://myzuul.com:6001/ckp/mall/goods/findById/1?accessToken=xyz>



图 32 成功访问

可以看到成功访问，并且控制台也输出 accessToken：

```
accessToken:xyz
accessToken1 response code:500
accessToken1:xyz
```

图 33 控制台输出

4.2.7 分布式配置 (Config)

启动 ConfigServer。访问：

<http://localhost:7755/mscloud-config-user-client/dev>



图 34 成功访问

成功访问。关闭 provider2，启动 ConfigClient。访问：

<http://localhost:8002/profile>



图 35 成功访问

成功访问。

五、实验创新点

本次实验练习了 SpringCloud 的各种组件：Eureka：服务治理组件，包含服

务注册与发现；**Hystrix**: 容错管理组件，实现了熔断器；**Ribbon**: 客户端负载均衡的服务调用组件；**Feign**: 基于 Ribbon 和 Hystrix 的声明式服务调用组件；**Zuul**: 网关组件，提供智能路由、访问过滤等功能；还有**Spring Cloud Config**: 配置管理工具，实现应用配置的外部化存储，支持客户端配置信息刷新、加密/解密配置内容等；通过对这些组件的学习与使用，我极大的感受到了 SpringCloud 的奥妙所在，也理解了微服务的名字，大道至简，化繁为简，这样极大的提高了应用开发效率。