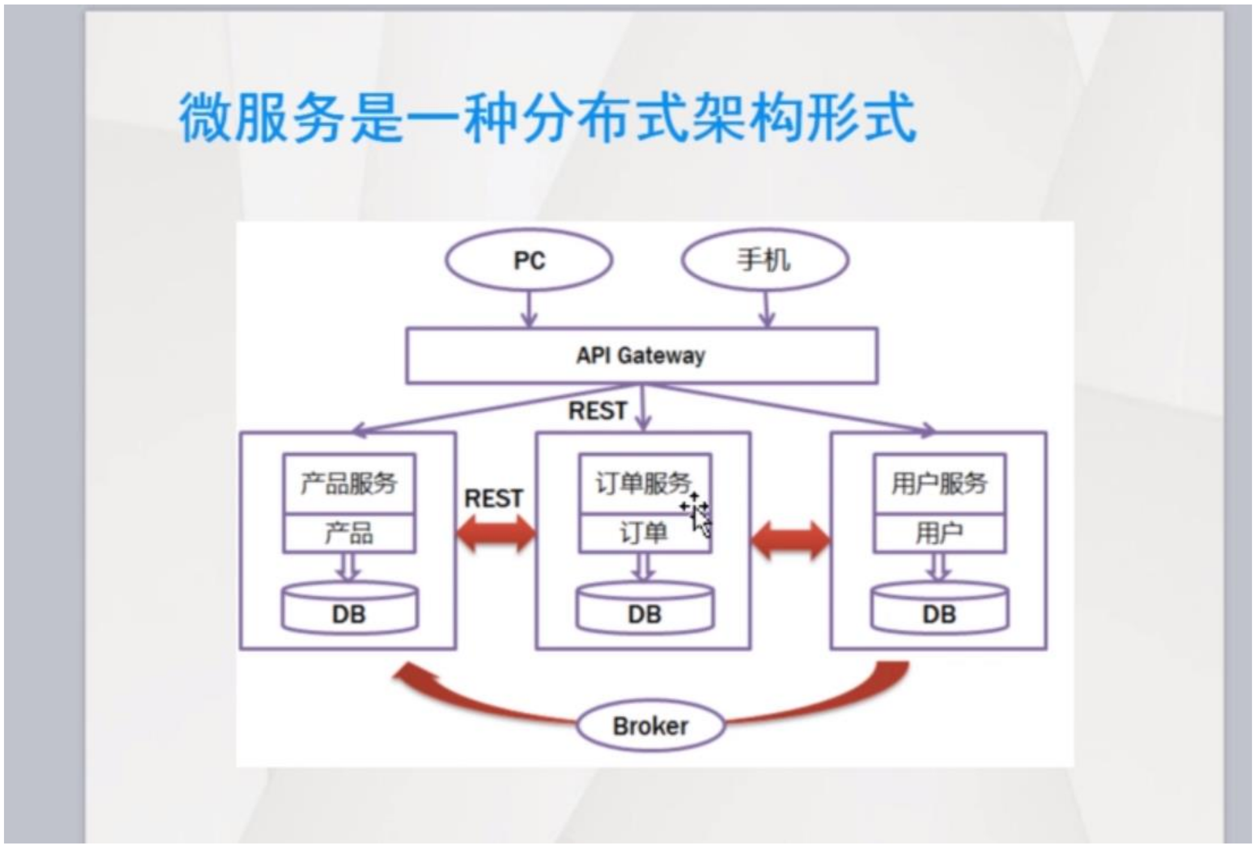
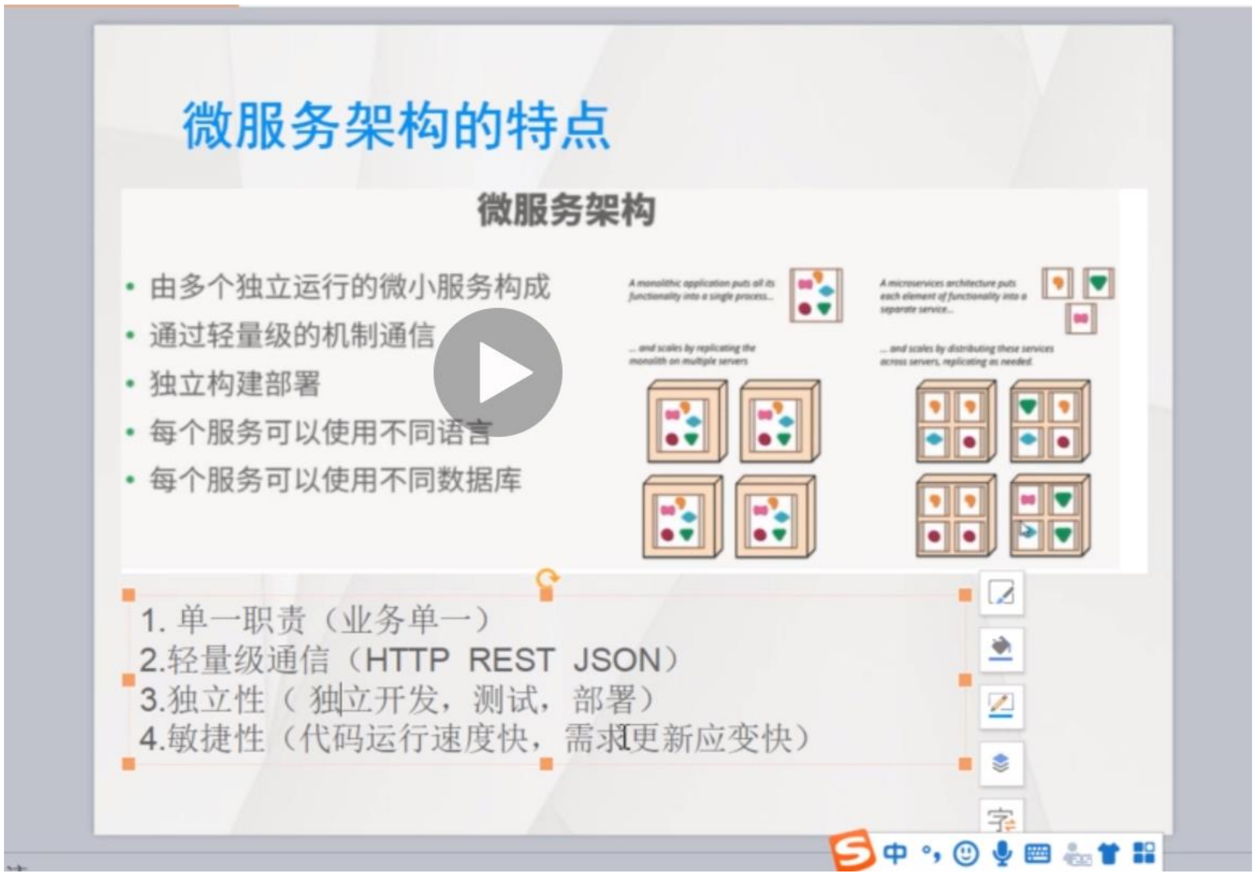


服务之间使用 restful 通信，每个服务可以使用不同的语言和数据库开发。


微服务架构的特点

微服务架构

- 由多个独立运行的微小服务构成
- 通过轻量级的机制通信
- 独立构建部署
- 每个服务可以使用不同语言
- 每个服务可以使用不同数据库

1. 单一职责（业务单一）
2.轻量级通信（HTTP REST JSON）
3.独立性（ 独立开发，测试，部署）
4.敏捷性（代码运行速度快，需求更新应变快）

安装 mysql 镜像，设置密码，端口：

```
docker run -di --name=mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 centos/mysql-57-centos7
```

host 输入 AWS ubuntu IPV4 地址，密码 123456 连接名为 springcloud-mysql

导入 springdataJPA 依赖：   mysqlconnector + spring data jpa

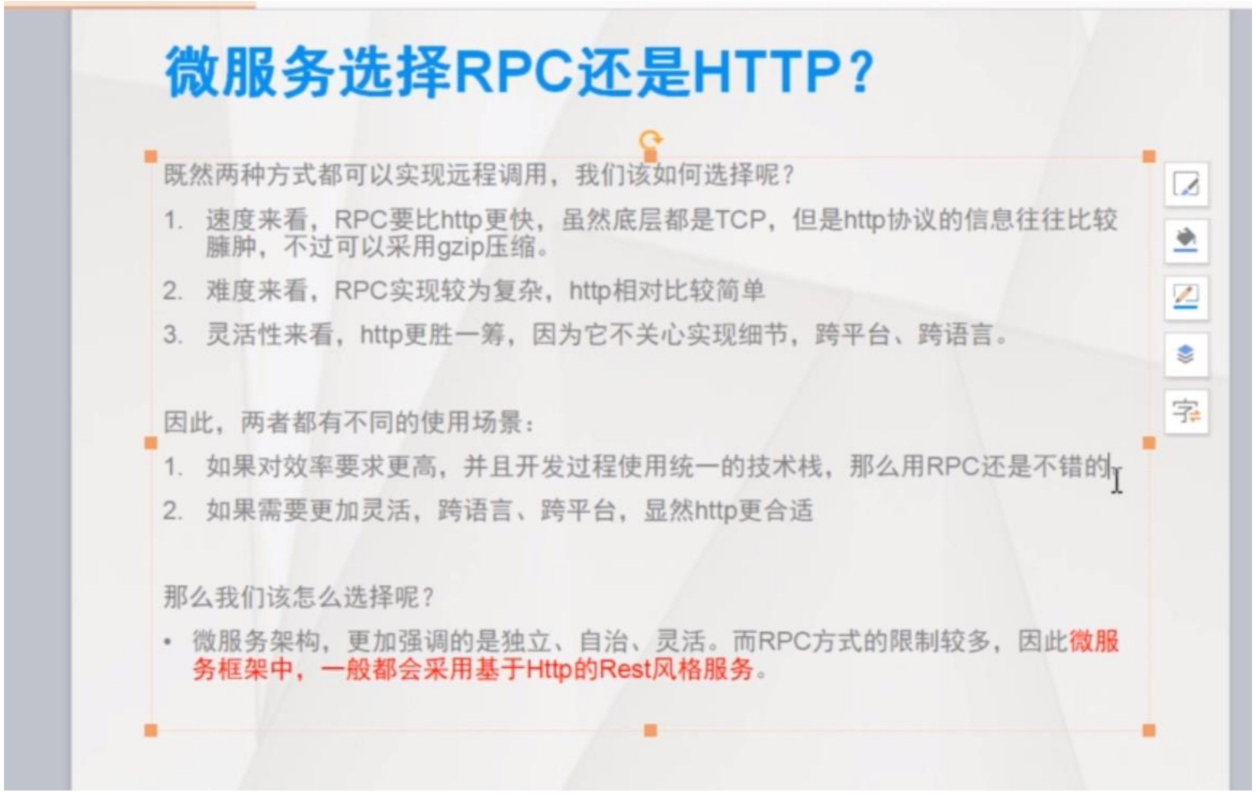
打开 putty.application,输入 AWS Instance ipv4 地址和 key 连接 docker。
打开 E 盘 mysql 命令行,18 191 236 201 是 AWS instance 地址。

```
E:\MySQL\mysql-8.0.17-winx64\bin>mysql -h 18.191.236.201 -P 3306 -u root    -p
```

密码 123456
Intellij 中配置文件写入

```
url: jdbc:mysql://18.191.236.201:3306/springcloud?characterEncoding=UTF8
```

Movie 模板需要调用 user 模块获取用户信息：
两种远程调用：
RPC/HTTP
RTC： 自定义数据格式，socket 通讯，速度快效率高。
HTTP： rest style



微服务选择RPC还是HTTP？

既然两种方式都可以实现远程调用，我们该如何选择呢？

1. 速度来看，RPC要比http更快，虽然底层都是TCP，但是http协议的信息往往比较臃肿，不过可以采用gzip压缩。
2. 难度来看，RPC实现较为复杂，http相对比较简单
3. 灵活性来看，http更胜一筹，因为它不关心实现细节，跨平台、跨语言。

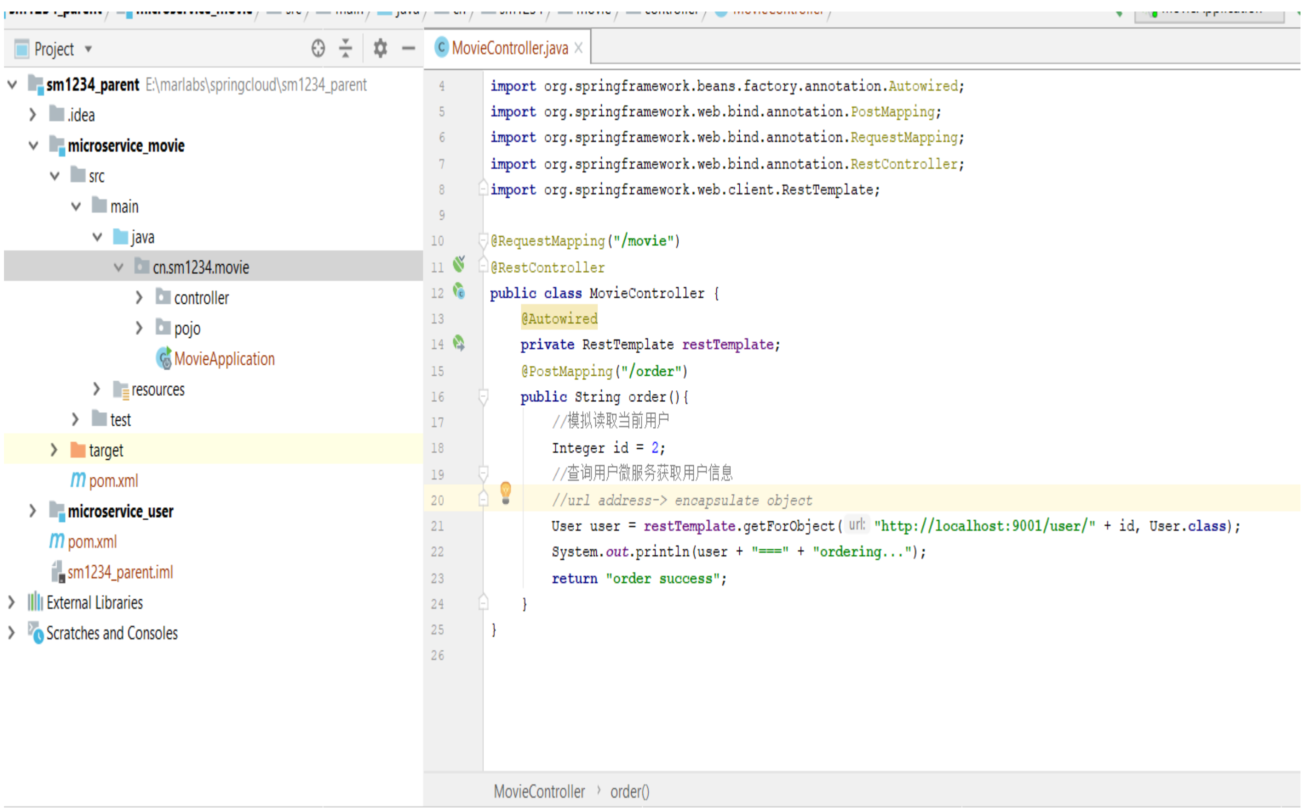
因此，两者都有不同的使用场景：

1. 如果对效率要求更高，并且开发过程使用统一的技术栈，那么用RPC还是不错的。
2. 如果需要更加灵活，跨语言、跨平台，显然http更合适

那么我们该怎么选择呢？

- 微服务架构，更加强调的是独立、自治、灵活。而RPC方式的限制较多，因此微服务框架中，一般都会采用基于Http的Rest风格服务。

模块 module 之间使用 resttemplate 互相调用！！！



```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RequestMapping("/movie")
@RestController
public class MovieController {
    @Autowired
    private RestTemplate restTemplate;
    @PostMapping("/order")
    public String order(){
        //模拟读取当前用户
        Integer id = 2;
        //查询用户微服务获取用户信息
        //url address-> encapsulate object
        User user = restTemplate.getForObject( url: "http://localhost:9001/user/" + id, User.class);
        System.out.println(user + "===" + "ordering...");
        return "order success";
    }
}
```

如何自动注册发现服务?

如何实现状态监管?

如何负载均衡?

如何解决容灾问题?

。。。 ====》 spring cloud

Spring cloud 服务注册与发现?

主要框架:

Spring cloud Netflix:eureka,openFeign,Hystrix,Zuul

Spring cloud config 配置中心

Spring cloud bus 配置实时更新

Spring cloud sleuth 分布式链路跟踪

Eureka=>注册中心

Openfeign=》服务调用组件

Hystrix=》 熔断器

Zuul =>微服务网关

Spring cloud 版本: 使 H.release 版本

依赖 spring boot2.2.2

什么是 eureka?

用于服务注册，分为 eureka server, eureka client

负责管理记录服务提供者信息，

服务提供方与 eureka 之间通过心跳监控

实现了服务自动注册，发现，状态监控。

搭建 eureka server

创建一个新模块

导入依赖：在父工程导入 spring cloud 依赖

```xml
<!-- 定义 spring cloud 版本-->

<repositories>

    <repository>

        <id>spring-snapshots</id>

        <name>Spring Snapshots</name>

        <url>https://repo.spring.io/snapshot</url>

        <snapshots>

            <enabled>true</enabled>

        </snapshots>

    </repository>


    <repository>

        <id>spring-milestones</id>

        <name>Spring Milestones</name>

        <url>https://repo.spring.io/milestone</url>

        <snapshots>

            <enabled>false</enabled>

        </snapshots>

    </repository>


</repositories>



<pluginRepositories>
```

```xml
    <pluginRepository>
        <id>spring-snapshots</id>
        <name>Spring Snapshots</name>
        <url>https://repo.spring.io/snapshot</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </pluginRepository>

    <pluginRepository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>

<!-- 锁定 SpringCloud 版本 -->
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Finchley.M9</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

在子工程导入 eureka server 依赖

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
```

编写 application.yml 配置 eureka

```
server:
  port: 8888
spring:
  application:
    name: eureka-server
#单机版配置
eureka:
  client:
    fetch-registry: false  #是否需要从 eureka 获取注册信息
    register-with-eureka: false #是否需要把该服务注册到 eureka
    service-url:  http://127.0.0.1:${server.port} #暴露 eureka 注册地址
```

编写启动类，添加@EnableEurekaServer 注解

```
package cn.sm1234.eureka;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

//eureka 微服务
@SpringBootApplication
@EnableEurekaServer  //开启服务端自动配置
public class EurekaApplication {
    public static void main(String[] args){
        SpringApplication.run(EurekaApplication.class, args);
    }
}
```

记得更改 spring cloud 到 H 版本！！！

注册服务到 eureka

导入 eureka client 依赖

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

在 application.yml 配置连接 Eureka Server

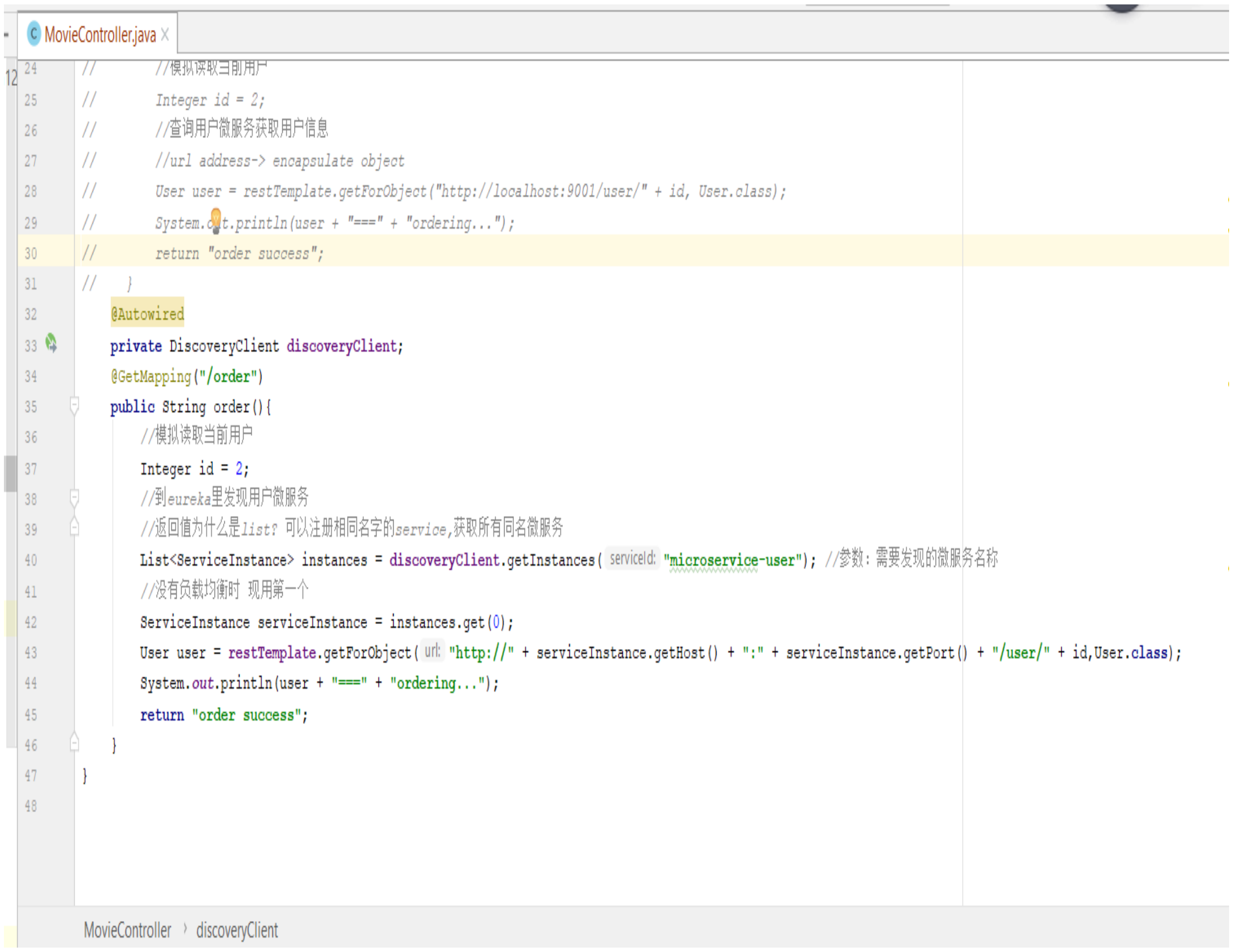
```yaml
eureka:
  client:
    register-with-eureka: true #作为客户端 需要注册到 eureka
    fetch-registry: true   #获取注册信息 从 eureka
    service-url:
      defaultZone: http://127.0.0.1:9003/eureka
  instance:
    prefer-ip-address: true #优先使用该服务的 IP 地址注册到 Eureka
```

启动类加上@EnableEurekaClient 注解

```java
@SpringBootApplication
@EnableEurekaClient //开启 eureka 客户端的自动配置
public class UserApplication {
    public static void main(String[] args){
        SpringApplication.run(UserApplication.class, args);
    }
}
```

解耦写死的 url 地址:

注入 discoveryClient,使用它的 getInstances 方法 拼接 URL



MovieController.java

```
//        //模拟读取当前用户
//        Integer id = 2;
//        //查询用户微服务获取用户信息
//        //url address-> encapsulate object
//        User user = restTemplate.getForObject("http://localhost:9001/user/" + id, User.class);
//        System.out.println(user + "===" + "ordering...");
//        return "order success";
//    }
    @Autowired
    private DiscoveryClient discoveryClient;
    @GetMapping("/order")
    public String order(){
        //模拟读取当前用户
        Integer id = 2;
        //到eureka里发现用户微服务
        //返回值为什么是list? 可以注册相同名字的service,获取所有同名微服务
        List<ServiceInstance> instances = discoveryClient.getInstances( serviceId: "microservice-user"); //参数: 需要发现的微服务名称
        //没有负载均衡时 现用第一个
        ServiceInstance serviceInstance = instances.get(0);
        User user = restTemplate.getForObject( url: "http://" + serviceInstance.getHost() + ":" + serviceInstance.getPort() + "/user/" + id,User.class);
        System.out.println(user + "===" + "ordering...");
        return "order success";
    }
}
```

MovieController › discoveryClient

搭建高可用的 Eureka Server:

一台 eureka 服务器 down =》需要搭建很多 Eureka Server

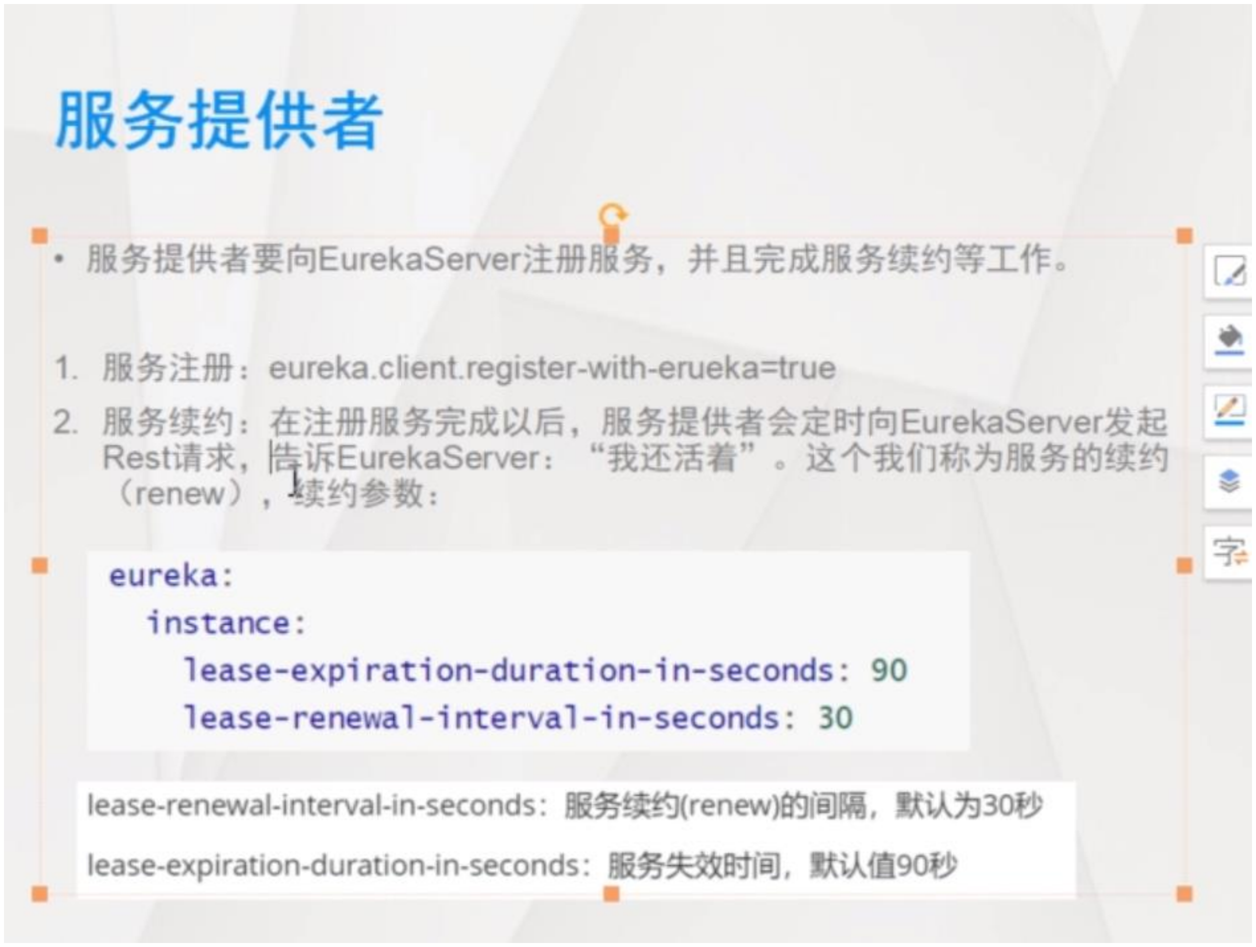
两台 eureka 服务器互相注册!!!

```yaml
server:
  port: 9004
spring:
  application:
    name: eureka-server
    # 单机版配置
#eureka:
#  client:
#    fetch-registry: false # 是否需要从 Eureka 获取注册信息
#    register-with-eureka: false # 是否需要把该服务注册到 Eureka
#    service-url:  # 暴露 Eureka 注册地址
#      defaultZone: http://127.0.0.1:${server.port}/eureka
    #集群配置
eureka:
  client:
    fetch-registry: true # 是否需要从 Eureka 获取注册信息
    register-with-eureka: true # 是否需要把该服务注册到 Eureka
    service-url:  # 暴露 Eureka 注册地址
      defaultZone: http://127.0.0.1:9003/eureka
```

启动一次, 9003 与 9004 互换再启动一次！ 设置为 true 互相注册!!!

服务提供方：userApplication,movieApplication

服务调用方（movie）获取注册信息配置



服务提供者

- 服务提供者要向EurekaServer注册服务，并且完成服务续约等工作。

1. 服务注册：eureka.client.register-with-erueka=true
2. 服务续约：在注册服务完成以后，服务提供者会定时向EurekaServer发起Rest请求，告诉EurekaServer："我还活着"。这个我们称为服务的续约（renew），续约参数：

```yaml
eureka:
  instance:
    lease-expiration-duration-in-seconds: 90
    lease-renewal-interval-in-seconds: 30
```

lease-renewal-interval-in-seconds：服务续约(renew)的间隔，默认为30秒

lease-expiration-duration-in-seconds：服务失效时间，默认值90秒

服务注册：

Eureka.client.register-with-eureka = true

注册服务之后，定时发送心跳：

```
lease-renewal-interval-in-seconds: 5
lease-expiration-duration-in-seconds: 15
```

服务调用方（movie）获取注册信息配置

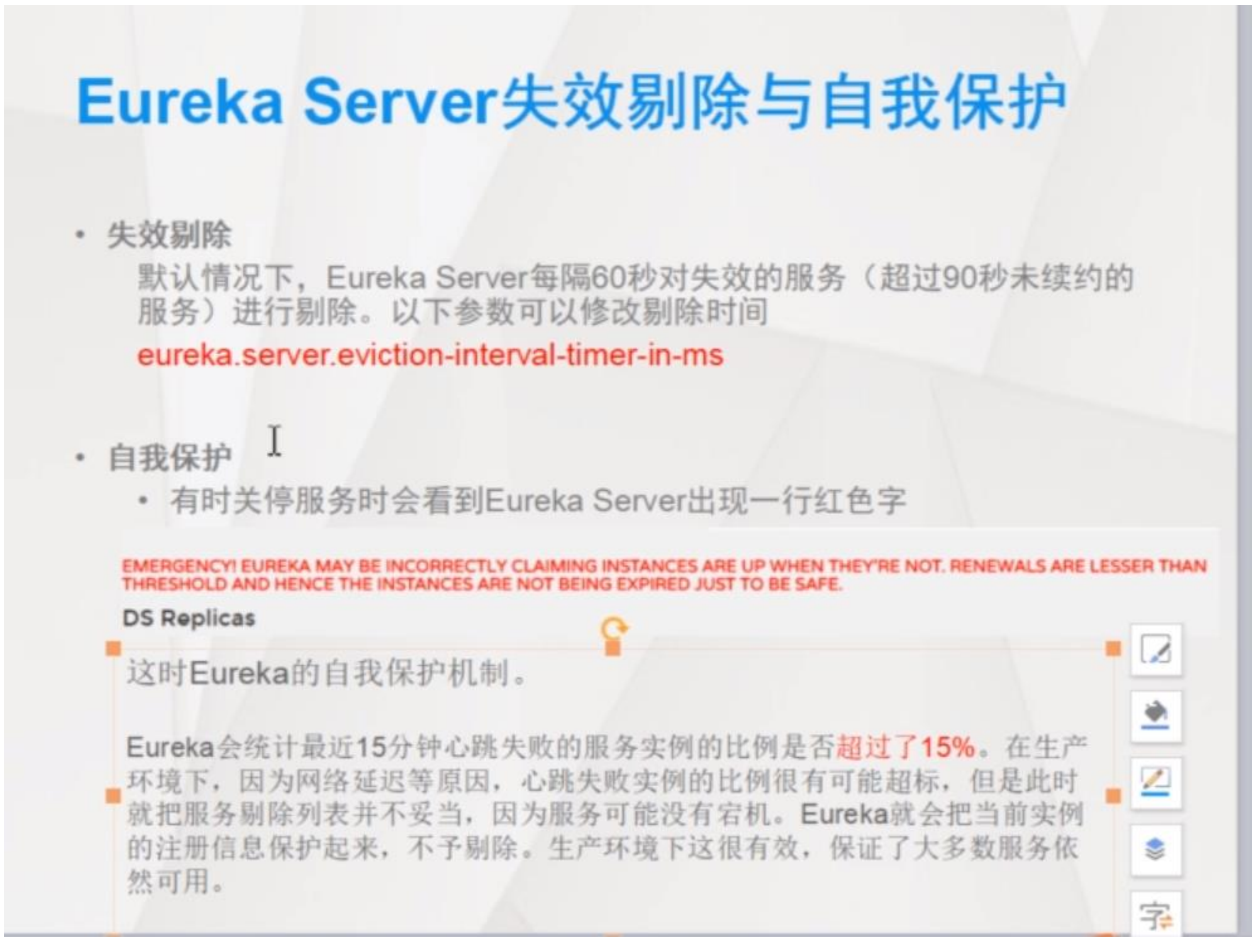
获取服务注册信息：

Eureka.client.fetch-registry=true

默认每隔 30 秒重新获取并更新注册信息，修改参数

Eureka.client.registry-fetch-interval-seconds=5

Eureka Server 的失效剔除与自我保护



Eureka Server失效剔除与自我保护

- 失效剔除
  默认情况下，Eureka Server每隔60秒对失效的服务（超过90秒未续约的服务）进行剔除。以下参数可以修改剔除时间
  eureka.server.eviction-interval-timer-in-ms
- 自我保护
  - 有时关停服务时会看到Eureka Server出现一行红色字

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

这时Eureka的自我保护机制。

Eureka会统计最近15分钟心跳失败的服务实例的比例是否超过了15%。在生产环境下，因为网络延迟等原因，心跳失败实例的比例很有可能超标，但是此时就把服务剔除列表并不妥当，因为服务可能没有宕机。Eureka就会把当前实例的注册信息保护起来，不予剔除。生产环境下这很有效，保证了大多数服务依然可用。

```yaml
eureka:

  client:

    fetch-registry: false # 是否需要从 Eureka 获取注册信息

    register-with-eureka: false # 是否需要把该服务注册到 Eureka

    service-url:  # 暴露 Eureka 注册地址

      defaultZone: http://127.0.0.1:${server.port}/eureka

  server:

    #修改扫描失效服务间隔时间

    eviction-interval-timer-in-ms: 5000

    #取消自我保护机制

    enable-self-preservation: false
```

第三章：负载均衡

使用 openFeign（简化服务调用） + 内置 Ribbon（负载均衡，默认轮询）

Ribbon 是 netflix 发布的负载均衡组件

集成到了 eureka client 里

```
//购票方法，使用 ribbon 负载均衡
@Autowired
private LoadBalancerClient loadBalancerClient;
@GetMapping("/order")
public String order(){
    //模拟读取当前用户
    Integer id = 1;
    //使用 ribbon 帮助选择合适的服务实例
    ServiceInstance serviceInstance = loadBalancerClient.choose("microservice-user");
    User user = restTemplate.getForObject("http://" + serviceInstance.getHost() + ":" +
serviceInstance.getPort() + "/user/" + id,User.class);
    System.out.println(user + "===" + "ordering...");
    return "order success";
}
```

第二种办法：

服务调用者启动类添加 ribbon 负载均衡

```
@Bean
@LoadBalanced //添加 ribbon 负载均衡组件
public RestTemplate restTemplate(){
    return new RestTemplate();
}
```

服务调用者 controller 里直接使用 service name

```
//简化版 ribbon
@GetMapping("/order")
public String order(){
    //模拟读取当前用户
    Integer id = 1;
    User user = restTemplate.getForObject("http://microservice-user/user/" + id,User.class);
    System.out.println(user + "===" + "ordering...");
    return "order success";
}
```

Ribbon 默认负载均衡算法是轮询，如何修改负载均衡算法?

Ribbon 通过 IRule 接口的 choose()方法实现自定义 load balance algorithm,通过实现 choose()方法的方式达到自定义 LB.

Private final static IRule DEFAULT_RULE = new RoundRobinRule(); //默认轮询算法(自增长值 % 总 server 数量)

修改 ribbon LB 算法

在服务调用方配置文件:

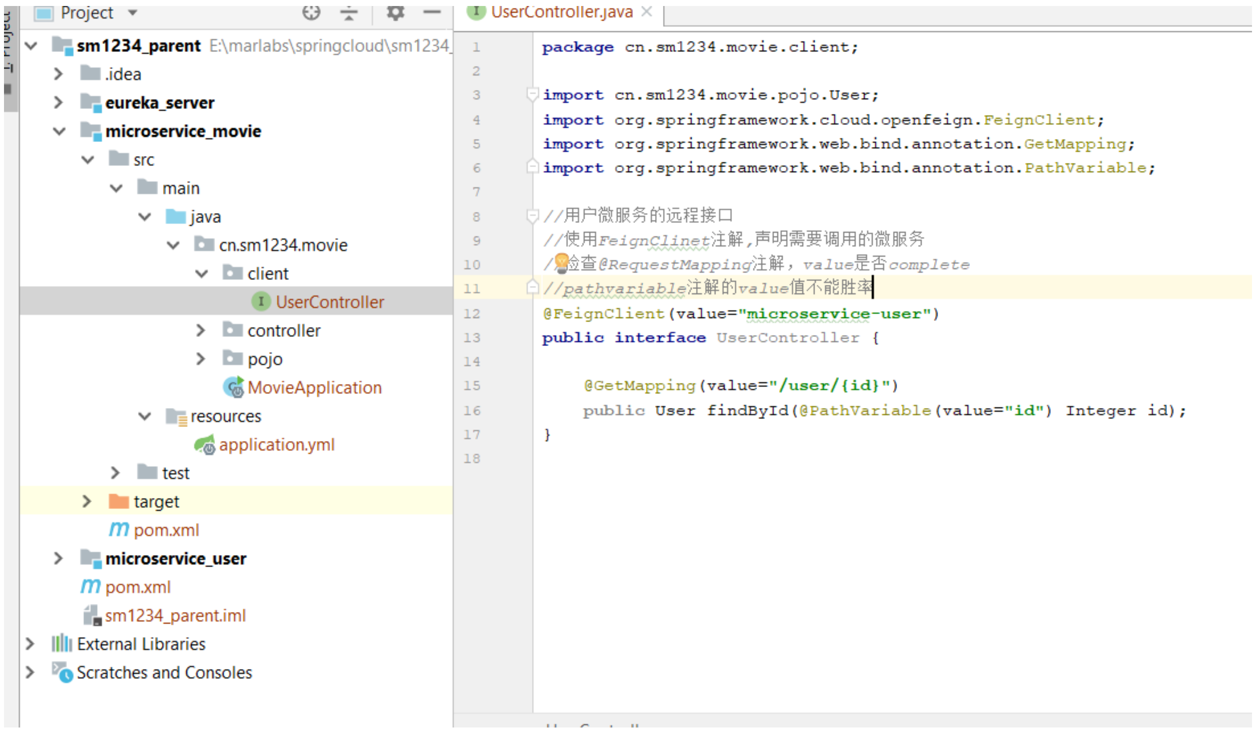
```
microservice-user:
  ribbon:
    NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule
```

服务调用组件: openfeign 四步

服务调用者导入 openfeign 依赖

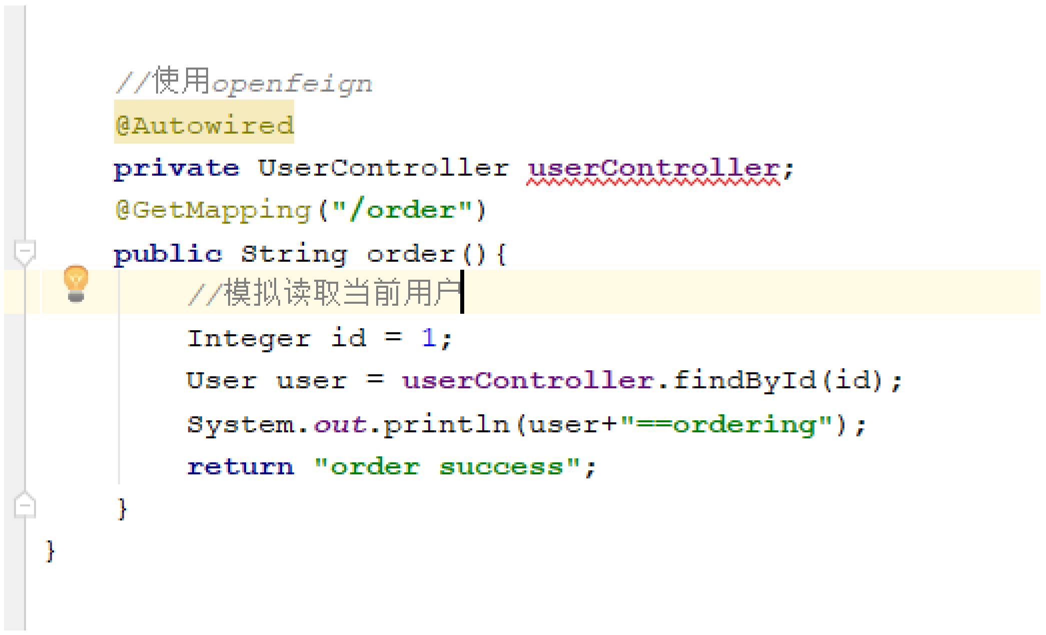
```xml
<!-- 导入 openfeign 依赖-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

创建服务接口



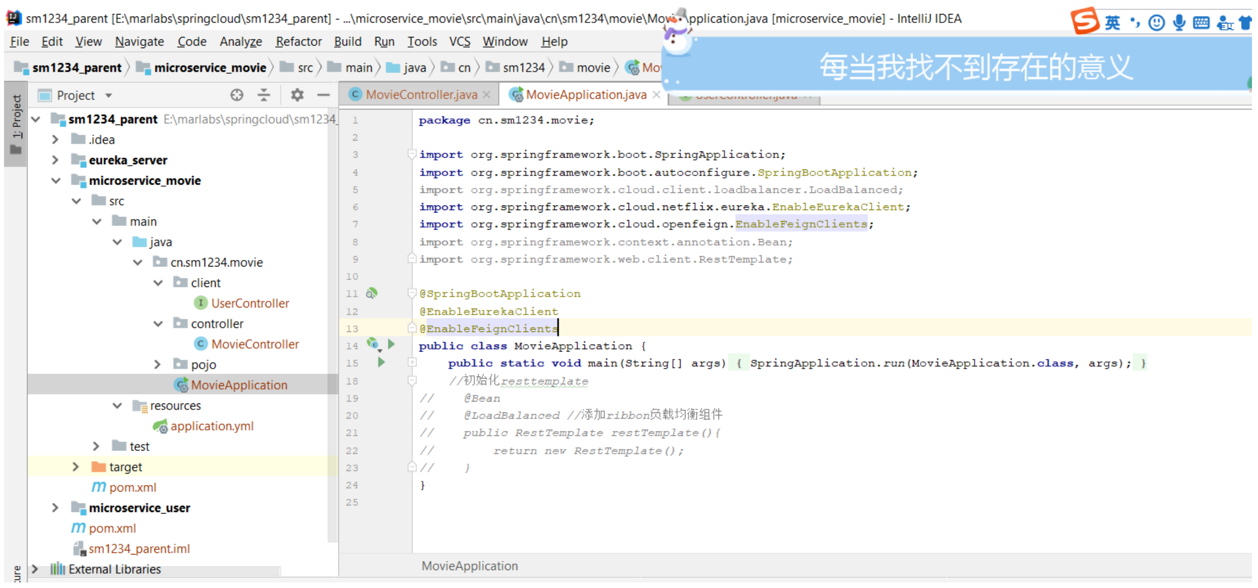
```java
package cn.sm1234.movie.client;

import cn.sm1234.movie.pojo.User;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

//用户微服务的远程接口
//使用FeignClinet注解,声明需要调用的微服务
/检查@RequestMapping注解，value是否complete
//pathvariable注解的value值不能胜率
@FeignClient(value="microservice-user")
public interface UserController {

    @GetMapping(value="/user/{id}")
    public User findById(@PathVariable(value="id") Integer id);
}
```

使用代理接口调用服务



```java
    //使用openfeign
    @Autowired
    private UserController userController;
    @GetMapping("/order")
    public String order(){
        //模拟读取当前用户
        Integer id = 1;
        User user = userController.findById(id);
        System.out.println(user+"==ordering");
        return "order success";
    }
}
```

向启动类中加入@EnableFeignClients 注解



```java
package cn.sm1234.movie;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class MovieApplication {
    public static void main(String[] args) { SpringApplication.run(MovieApplication.class, args); }
    //初始化resttemplate
//    @Bean
//    @LoadBalanced //添加ribbon负载均衡组件
//    public RestTemplate restTemplate(){
//        return new RestTemplate();
//    }
}
```

Openfeign 自带 resttemplate+ribbon

使用 openfeign + 自带 ribbon 作出 LB

Chapter 4 : spring cloud 熔断器

熔断器

作用：

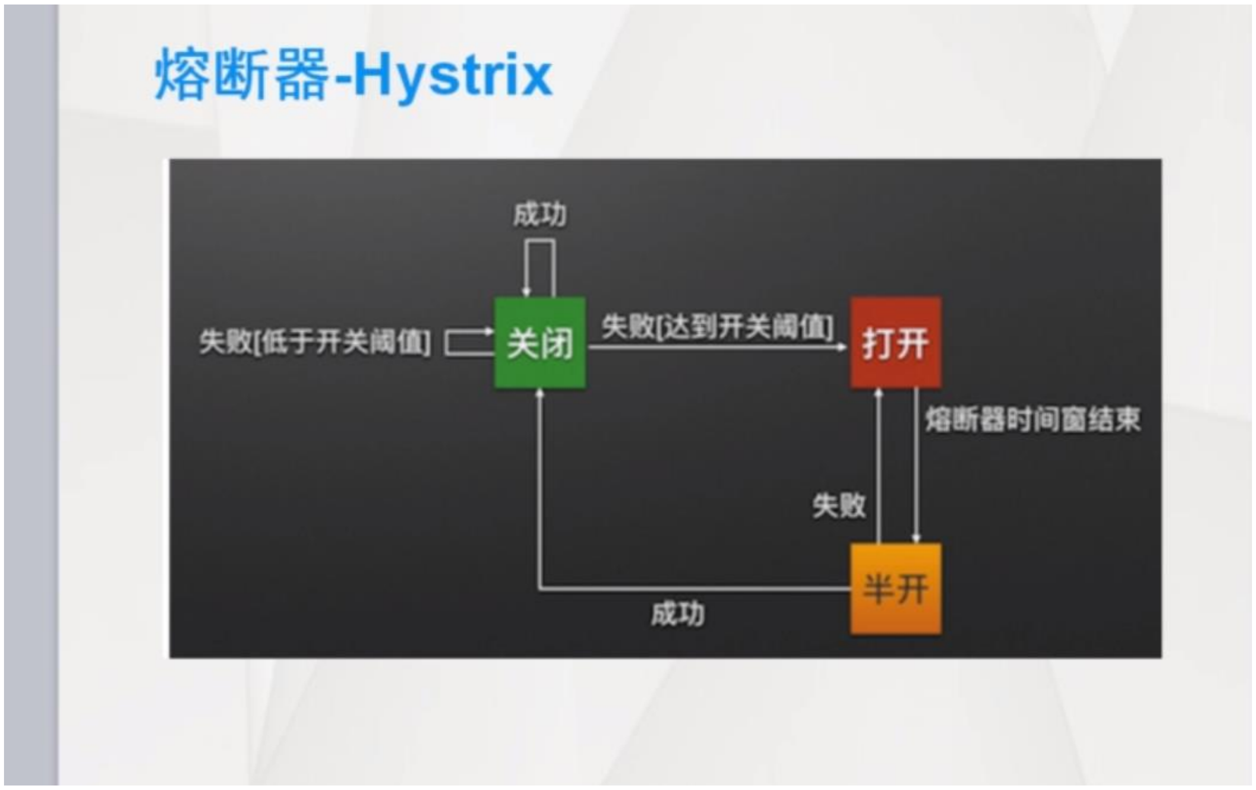
某个服务的单个点的请求故障会导致用户的请求处于阻塞状态，最终的结果就是整个服务的线程资源消耗殆尽。由于服务的依赖性，会导致依赖于该故障服务的其他服务也处于线程阻塞状态，最终导致这些服务的线程资源消耗殆尽 直到不可用，从而导致整个问服务系统都不可用，即雪崩效应。

为了防止雪崩效应，我们采用的熔断器 Hystrix。

工作原理（机制）：

首先，当服务的某个 API 接口的失败次数在一定时间内小于设定的阀值时，熔断器处于关闭状态，该 API 接口正常提供服务 。当该 API 接口处理请求的失败次数大于设定的阀值时， Hystrix 判定该 API 接口出现了故障，打开熔断器，这时请求该 API 接口会执行快速失败的逻辑（即 fall back 回退的逻辑），不执行业务逻辑，请求的线程不会处于阻塞状态。处于打开状态的熔断器一段时间后会处于半打开的状态，并将一定数量的请求执行正常的逻辑。剩余的请求会执行快速失败，若执行正常逻辑的请求失败了，则熔断器继续打开；若成功了，则将熔断器关闭。这样熔断器就具有了自我修复的能力。

熔断器 hystrix



1.RESTTEMPLATE+RIBBON + HYSTRIX 熔断器

2.另一种方法：openfeign 开启熔断器，默认有对 hystrix 的集成

第一种做法：

1. 有四步

   导入 hystrix 依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

   使用@hystrixCommand 声明 fallback 方法

```
//简化版 ribbon
@GetMapping("/order")
@HystrixCommand(fallbackMethod = "fallback")
public String order(){
    //模拟读取当前用户
    Integer id = 1;
    User user = restTemplate.getForObject("http://microservice-user/user/" + id,User.class);
    System.out.println(user + "===" + "ordering...");
    return "order success";
}
//熔断器 fallback 方法
public String fallback(){
    return "service temporal unavailable";
}
```

   编写 fallback 方法逻辑

   在启动类中添加@EnableHystrix 注解

```java
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
@EnableHystrix
public class MovieApplication {
   public static void main(String[] args){
      SpringApplication.run(MovieApplication.class, args);
   }
   //初始化 resttemplate
   @Bean
   @LoadBalanced //添加 ribbon 负载均衡组件
   public RestTemplate restTemplate(){
      return new RestTemplate();
   }

}
```

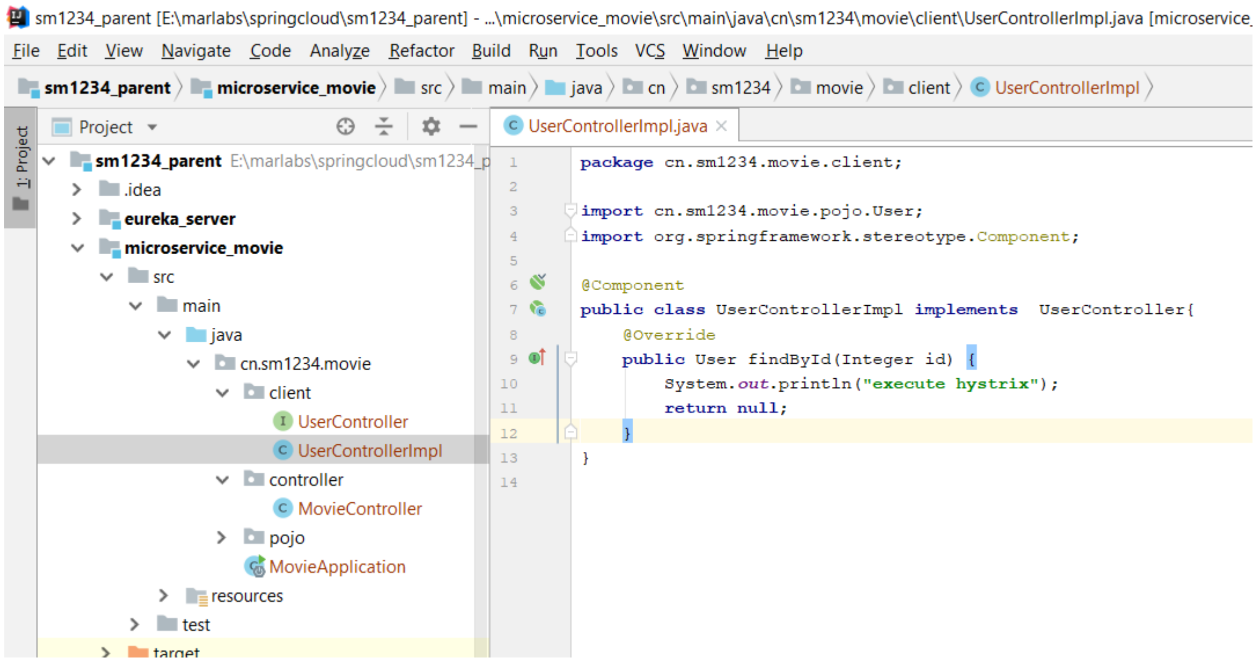
熔断器加在 movie service 里面，因为是它需要被保护 用到了服务提供方 user service。

第二种做法： openfeign 开启熔断器

Openfeign 打开熔断器开关

```yaml
#开启熔断器开关
feign:
  hystrix:
    enabled: true
```

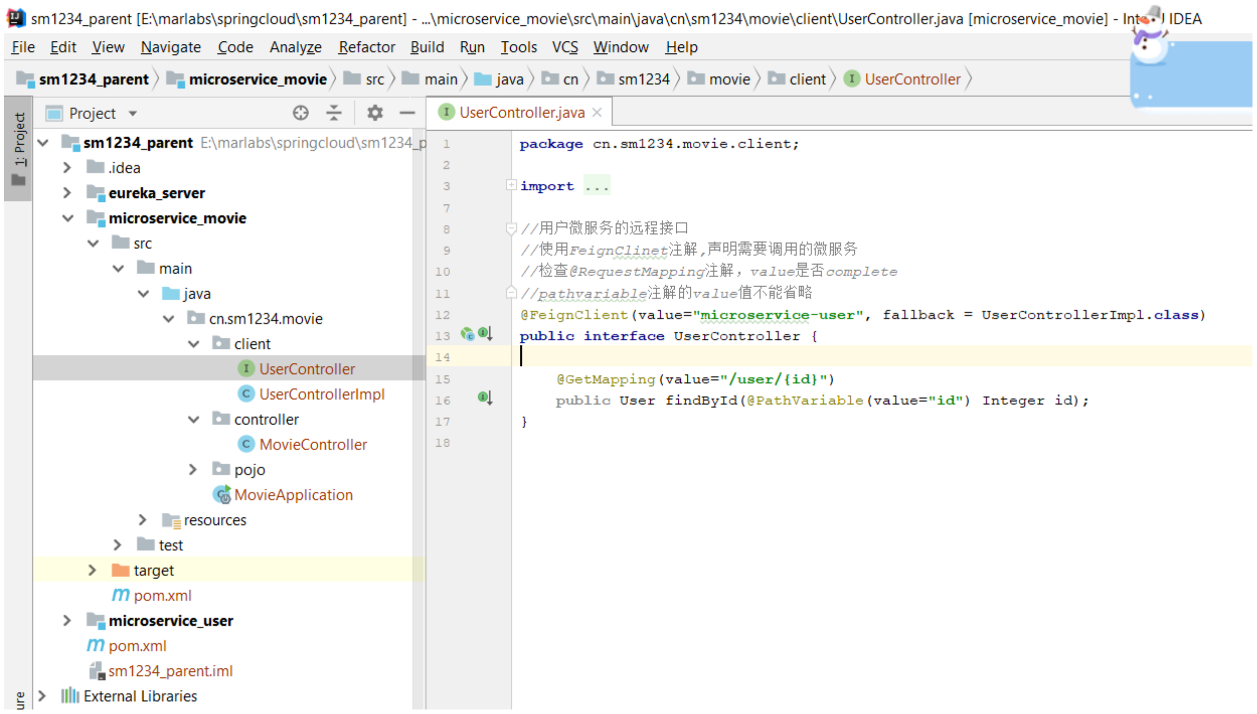
编写 fallback 处理类



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_movie\src\main\java\cn\sm1234\movie\client\UserControllerImpl.java [microservice

```java
package cn.sm1234.movie.client;

import cn.sm1234.movie.pojo.User;
import org.springframework.stereotype.Component;

@Component
public class UserControllerImpl implements  UserController{
    @Override
    public User findById(Integer id) {
        System.out.println("execute hystrix");
        return null;
    }
}
```

指定处理类



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_movie\src\main\java\cn\sm1234\movie\client\UserController.java [microservice_movie] - IntelliJ IDEA

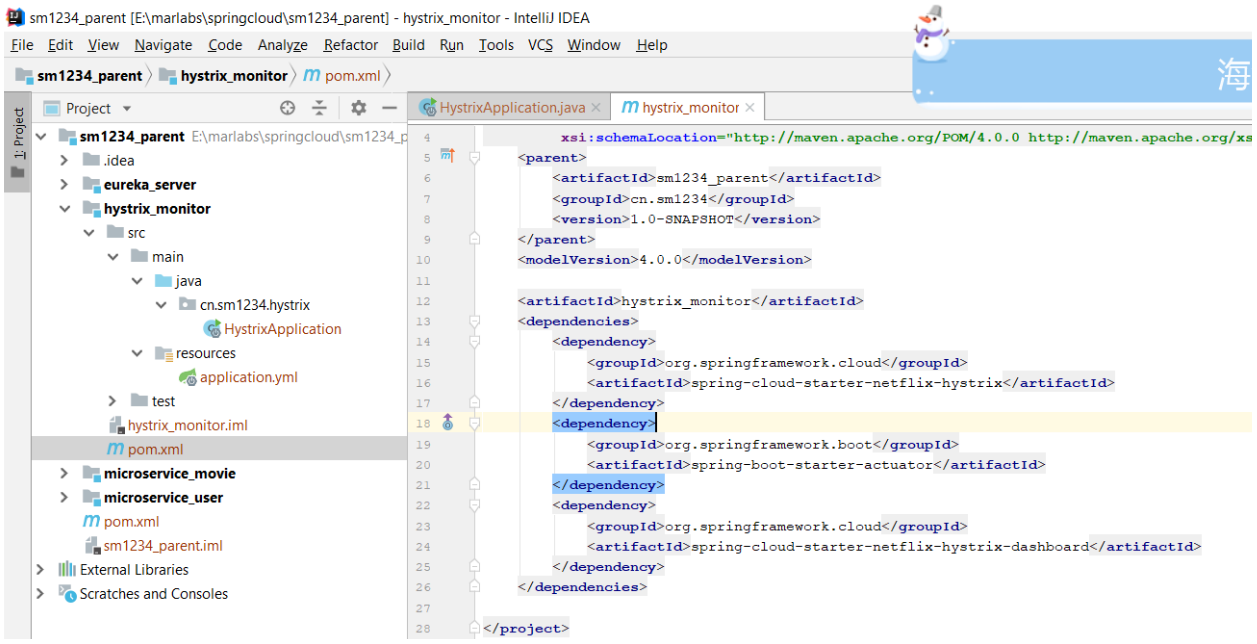
```java
package cn.sm1234.movie.client;

import ...

//用户微服务的远程接口
//使用FeignClinet注解,声明需要调用的微服务
//检查@RequestMapping注解，value是否complete
//pathvariable注解的value值不能省略
@FeignClient(value="microservice-user", fallback = UserControllerImpl.class)
public interface UserController {

    @GetMapping(value="/user/{id}")
    public User findById(@PathVariable(value="id") Integer id);
}
```

Hystrix dashboard 监控面板

监控请求成功失败次数，速度等等。、、

步骤：

1. 搭建 hystrix dashboard 工程（microservice），导入 hystrix dashboard 依赖，启动类添加@EnableHystrixDashboard 注解

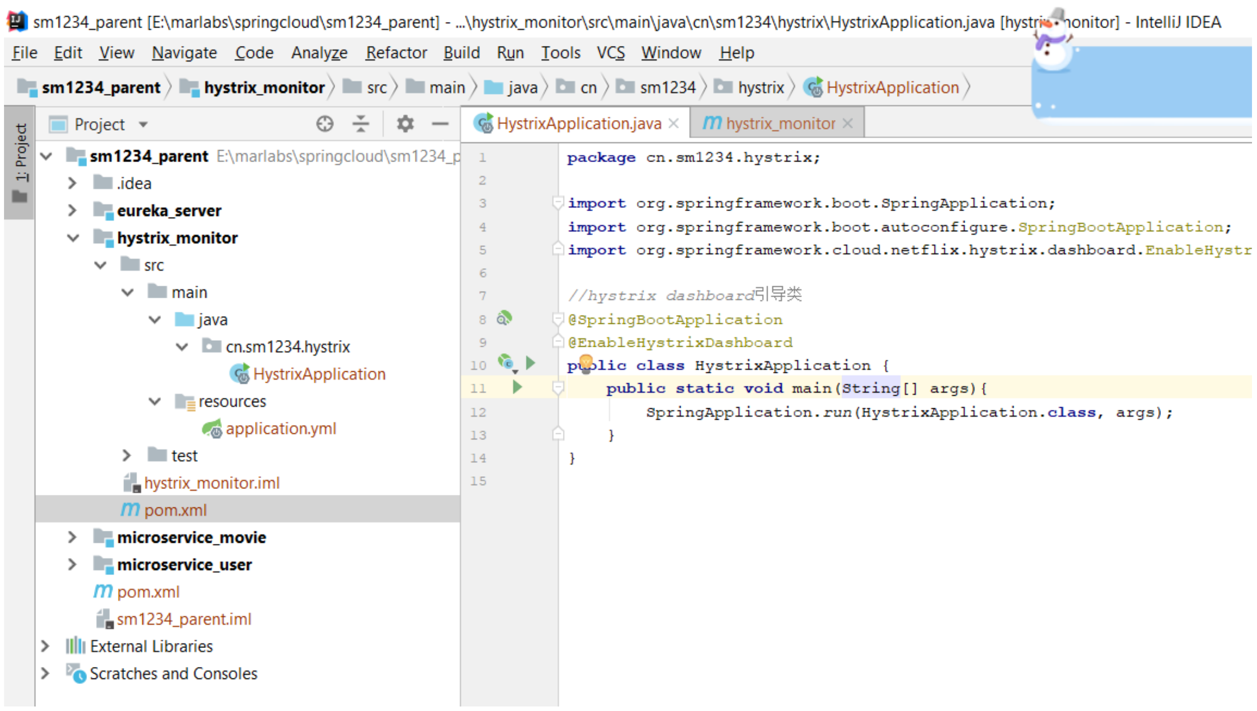


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - hystrix_monitor - IntelliJ IDEA

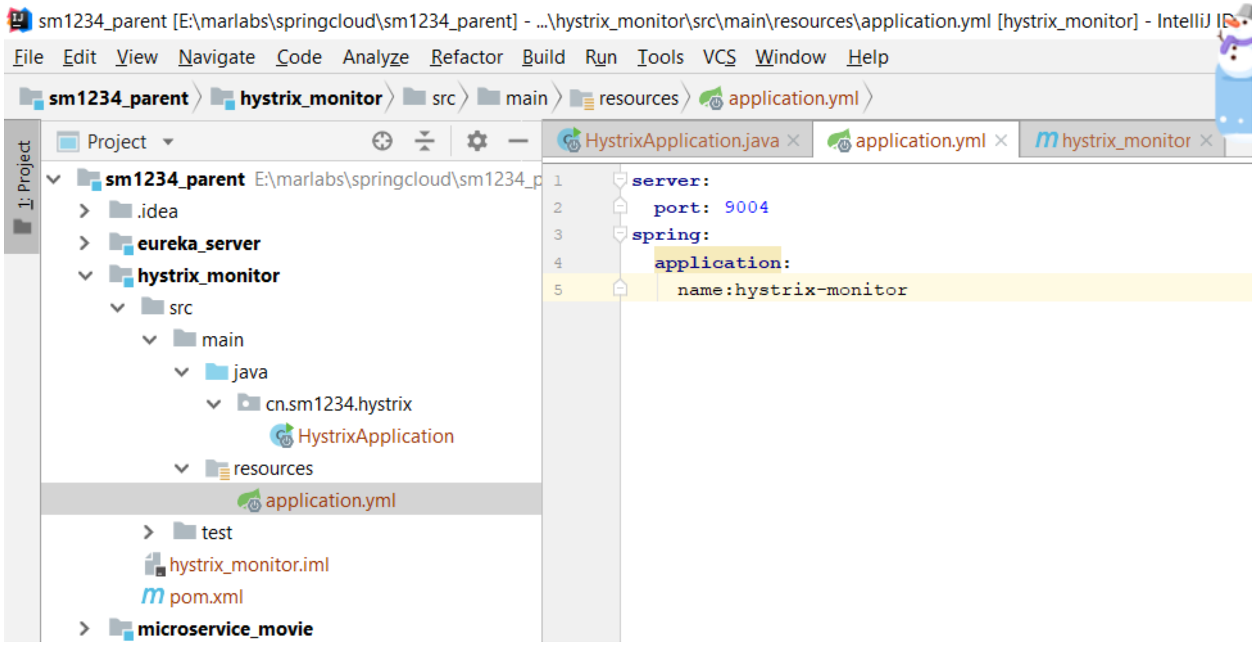
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sm1234_parent 〉 hystrix_monitor 〉 pom.xml
Project
HystrixApplication.java
hystrix_monitor
海

```
sm1234_parent E:\marlabs\springcloud\sm1234_p
  .idea
  eureka_server
  hystrix_monitor
    src
      main
        java
          cn.sm1234.hystrix
            HystrixApplication
        resources
          application.yml
      test
    hystrix_monitor.iml
    pom.xml
  microservice_movie
  microservice_user
  pom.xml
  sm1234_parent.iml
External Libraries
Scratches and Consoles
```

```xml
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xs
    <parent>
        <artifactId>sm1234_parent</artifactId>
        <groupId>cn.sm1234</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>hystrix_monitor</artifactId>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
        </dependency>
    </dependencies>

</project>
```


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\hystrix_monitor\src\main\java\cn\sm1234\hystrix\HystrixApplication.java [hystrix_monitor] - IntelliJ IDEA

```java
package cn.sm1234.hystrix;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystr

//hystrix dashboard引导类
@SpringBootApplication
@EnableHystrixDashboard
public class HystrixApplication {
    public static void main(String[] args){
        SpringApplication.run(HystrixApplication.class, args);
    }
}
```


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\hystrix_monitor\src\main\resources\application.yml [hystrix_monitor] - IntelliJ IDEA

```yaml
server:
  port: 9004
spring:
  application:
    name:hystrix-monitor
```

http://localhost:9004/hystrix 查看 hystrixdashboard 主页

2. 向消费方=》movie service 加入 servlet 监控器，监控调用服务情况

   在 movie 引导类里添加

```
@Bean
public ServletRegistrationBean getServlet(){
   HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();
   ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);
   registrationBean.setLoadOnStartup(1);
   registrationBean.addUrlMappings("hystrix.stream");
   registrationBean.setName("HystrixMetricsStreamServlet");
   return registrationBean;
}
```

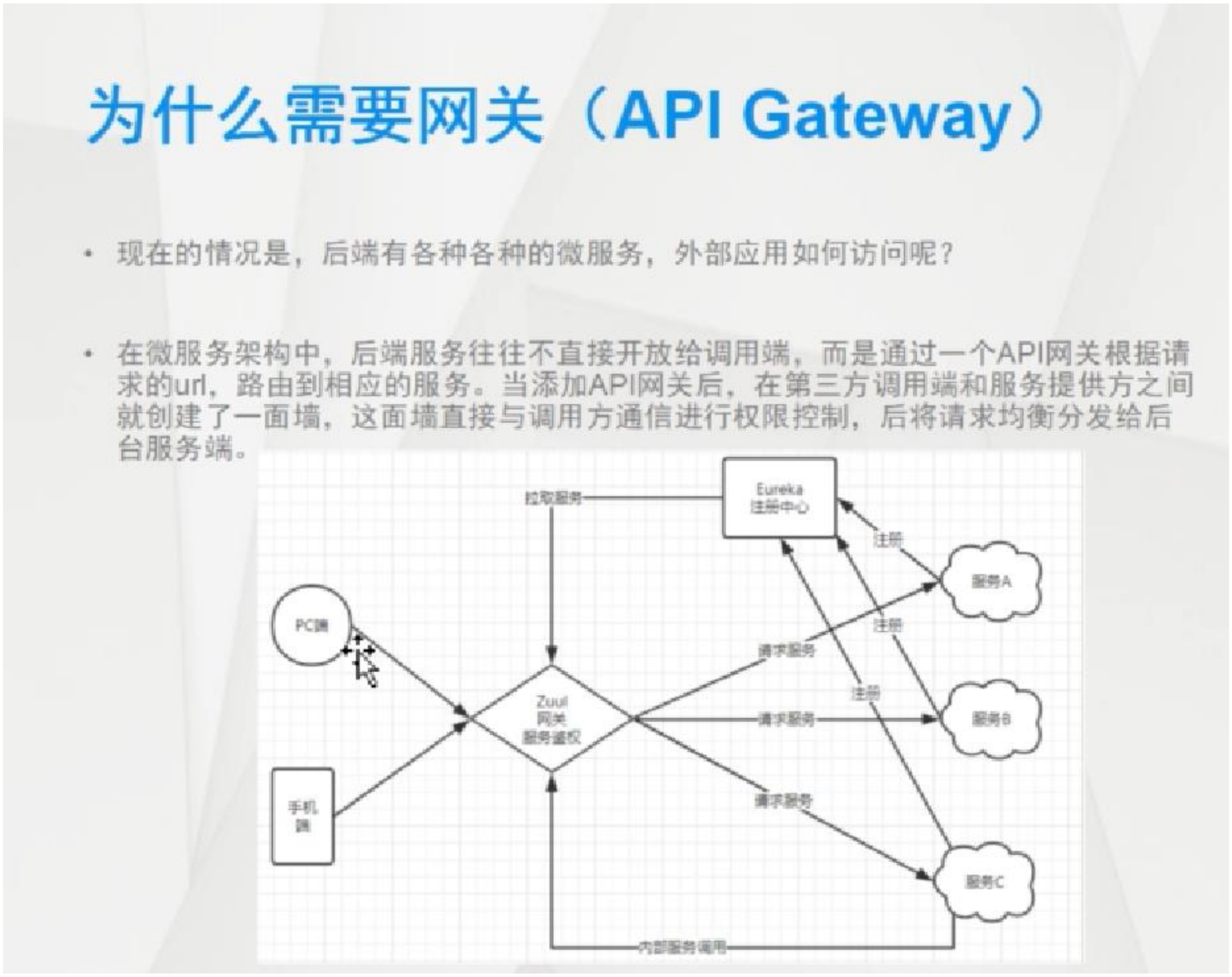
   到 localhost:9005/hystrix 进入 hystrix dashboard

   监控 localhost:9002/hystrix.stream

第五章 spring cloud 网关

Spring cloud API gateway:权限控制， 负载均衡

Permission control， LB



为什么需要网关（API Gateway）

- 现在的情况是，后端有各种各种的微服务，外部应用如何访问呢？
- 在微服务架构中，后端服务往往不直接开放给调用端，而是通过一个API网关根据请求的url，路由到相应的服务。当添加API网关后，在第三方调用端和服务提供方之间就创建了一面墙，这面墙直接与调用方通信进行权限控制，后将请求均衡分发给后台服务端。

Spring cloud zuul:

Zuul 是 netflix 开源的微服务网关，可以和 eureka，ribbon，hystrix 组件配合使用，zuul 核心是一系列的过滤器，这些过滤器具有以下功能：

身份认证与安全：认识每个资源的严正要求，并拒绝那些与要求不符的请求。

审查与监控：在边缘位置追踪有意义的数据和统计结果，从而带来精确的生产试图。

动态路由：动态的将请求路由到不同的后端集群。

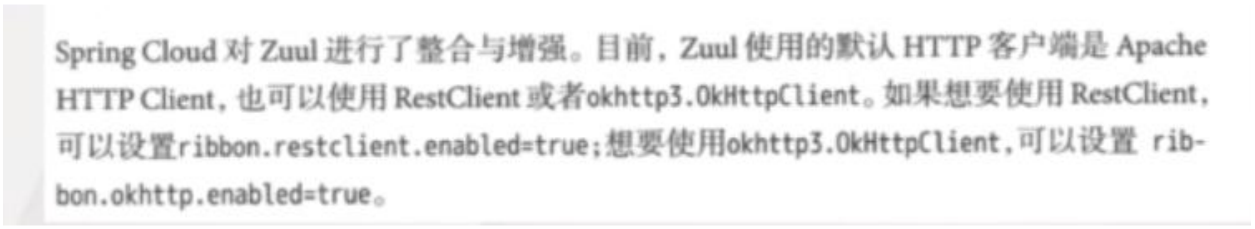
压力测试：逐渐增加指向集群的流量，以了解性能。

负载分配：为每一种负载类型分配对应容量，并且弃用超出限定值的请求。

静态响应处理：在边缘位置直接建立部分响应，从而避免其转发到内部集群。

多区域弹性：跨越 AWS Region 进行请求路由，旨在实现 ELB elastic load balancing 使用的多样化，以及让系统的边缘更贴近系统的使用者。

Spring cloud 对 zuul 进行了整合与增强，zuul 使用默认 HTTP 客户端是 Apache HTTP client,也可以使用 restClient 或者 okhttp3.OKHttpClient



Spring Cloud 对 Zuul 进行了整合与增强。目前，Zuul 使用的默认 HTTP 客户端是 Apache HTTP Client，也可以使用 RestClient 或者okhttp3.OkHttpClient。如果想要使用 RestClient，可以设置ribbon.restclient.enabled=true；想要使用okhttp3.OkHttpClient，可以设置 ribbon.okhttp.enabled=true。

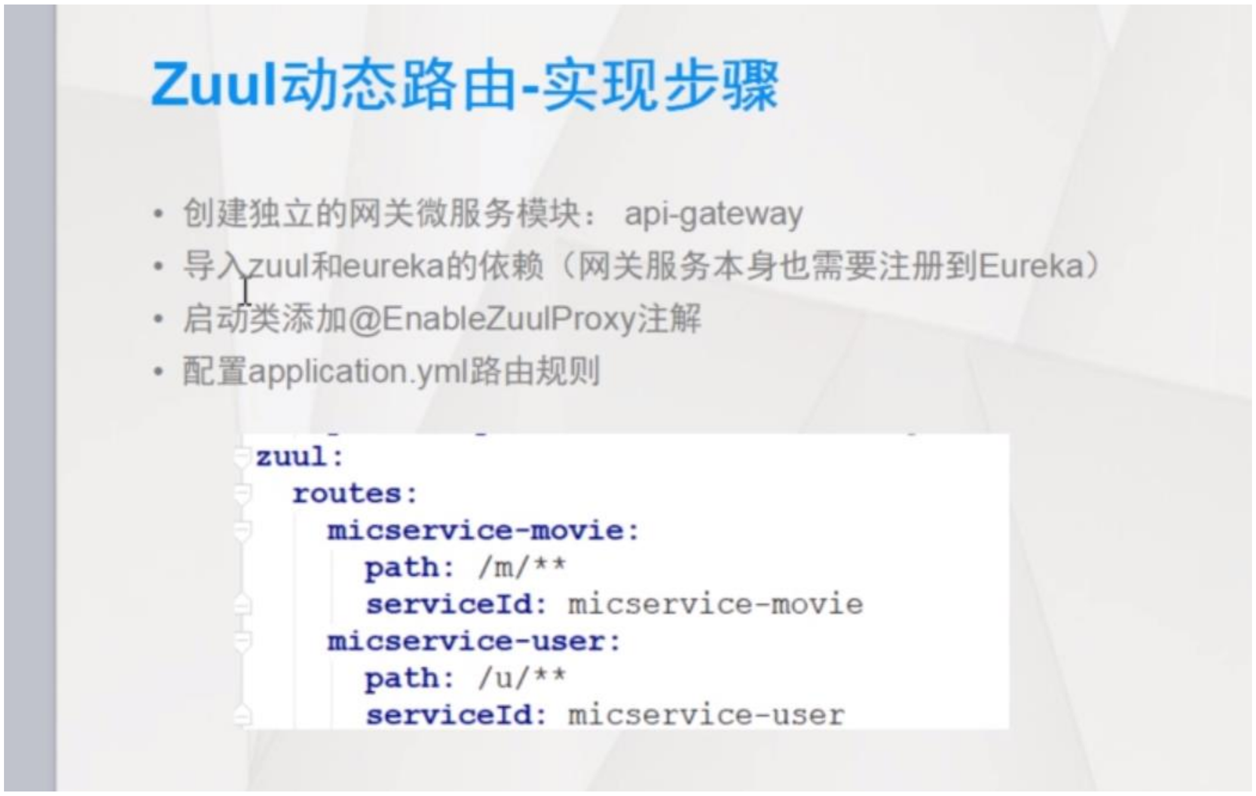
Zuul 动态路由---实现步骤

1.创建独立的网关微服务模块:api-gateway

2.导入 zuul 和 eureka 依赖（网关服务本身也需要注册到 eureka）

3.启动类添加@EnableZuulProxy

4.配置 application.yml 路由规则

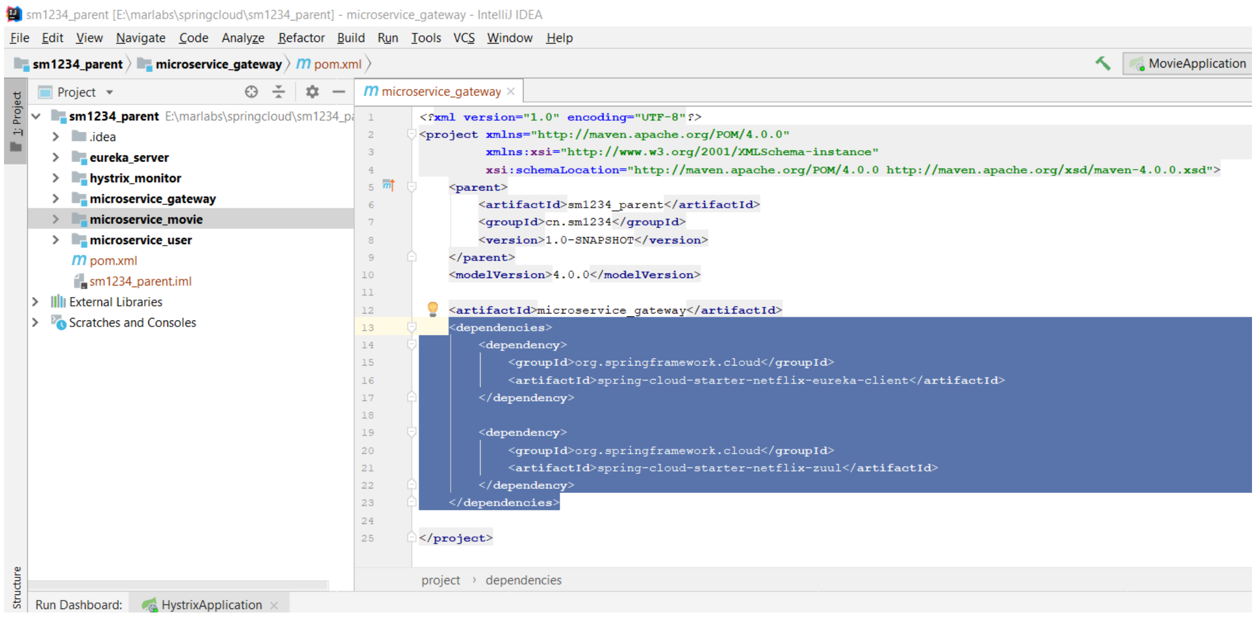


Zuul动态路由-实现步骤

- 创建独立的网关微服务模块： api-gateway
- 导入zuul和eureka的依赖（网关服务本身也需要注册到Eureka）
- 启动类添加@EnableZuulProxy注解
- 配置application.yml路由规则

```yaml
zuul:
  routes:
    micservice-movie:
      path: /m/**
      serviceId: micservice-movie
    micservice-user:
      path: /u/**
      serviceId: micservice-user
```

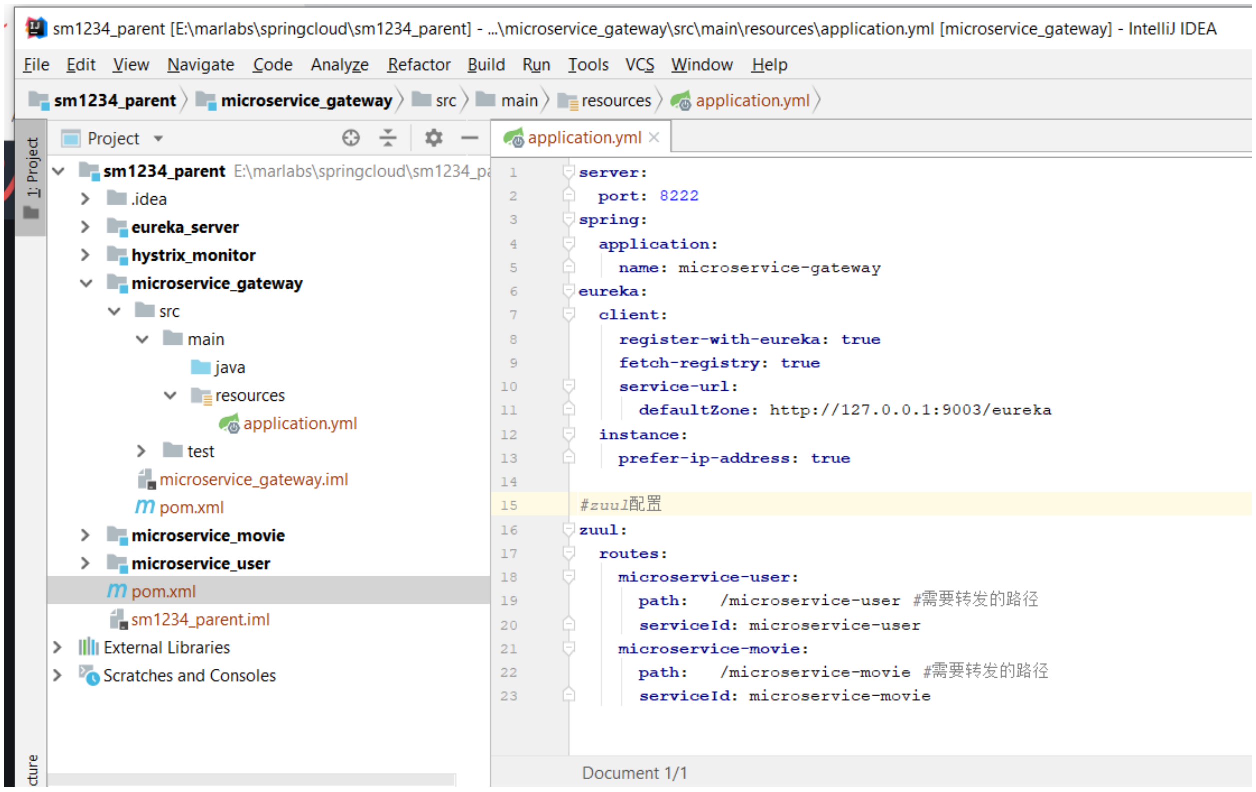
2.



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - microservice_gateway - IntelliJ IDEA

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>sm1234_parent</artifactId>
        <groupId>cn.sm1234</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>microservice_gateway</artifactId>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
        </dependency>
    </dependencies>

</project>
```

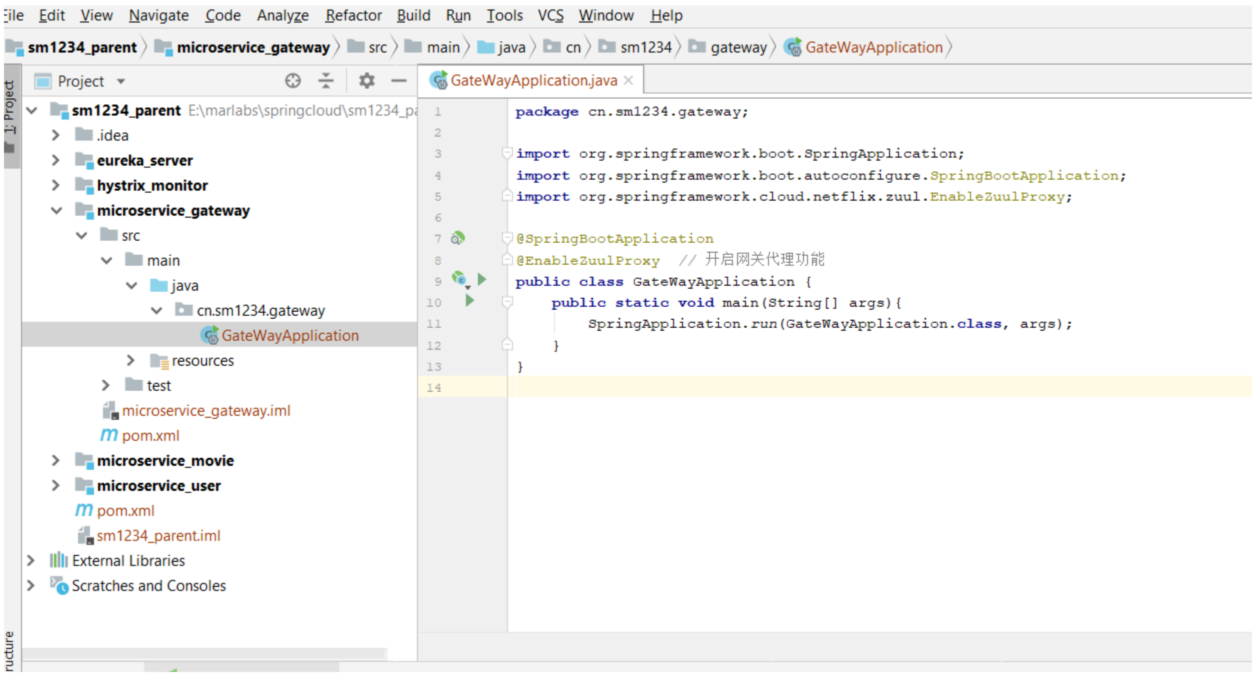
4.



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_gateway\src\main\resources\application.yml [microservice_gateway] - IntelliJ IDEA

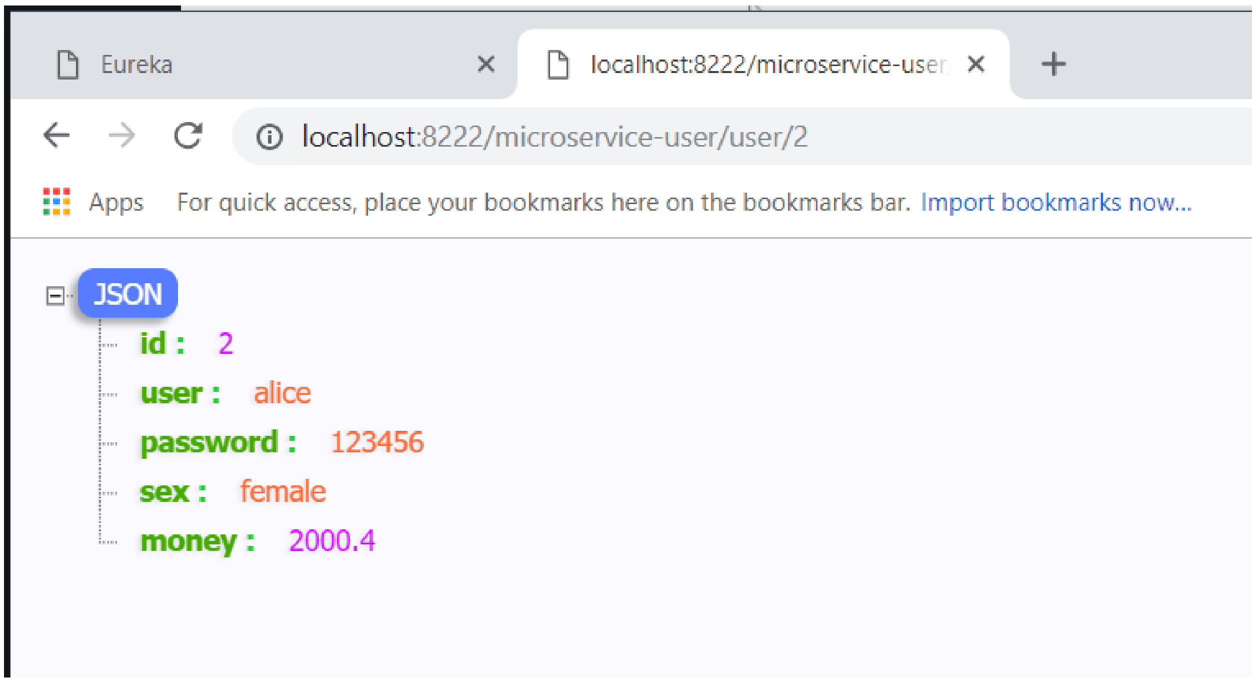
```
server:
  port: 8222
spring:
  application:
    name: microservice-gateway
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://127.0.0.1:9003/eureka
  instance:
    prefer-ip-address: true

#zuul配置
zuul:
  routes:
    microservice-user:
      path:   /microservice-user  #需要转发的路径
      serviceId: microservice-user
    microservice-movie:
      path:   /microservice-movie  #需要转发的路径
      serviceId: microservice-movie
```

3.



```java
package cn.sm1234.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy  // 开启网关代理功能
public class GateWayApplication {
    public static void main(String[] args){
        SpringApplication.run(GateWayApplication.class, args);
    }
}
```

Zuul 网关转发成功


Eureka
localhost:8222/microservice-user
localhost:8222/microservice-user/user/2
Apps
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
JSON

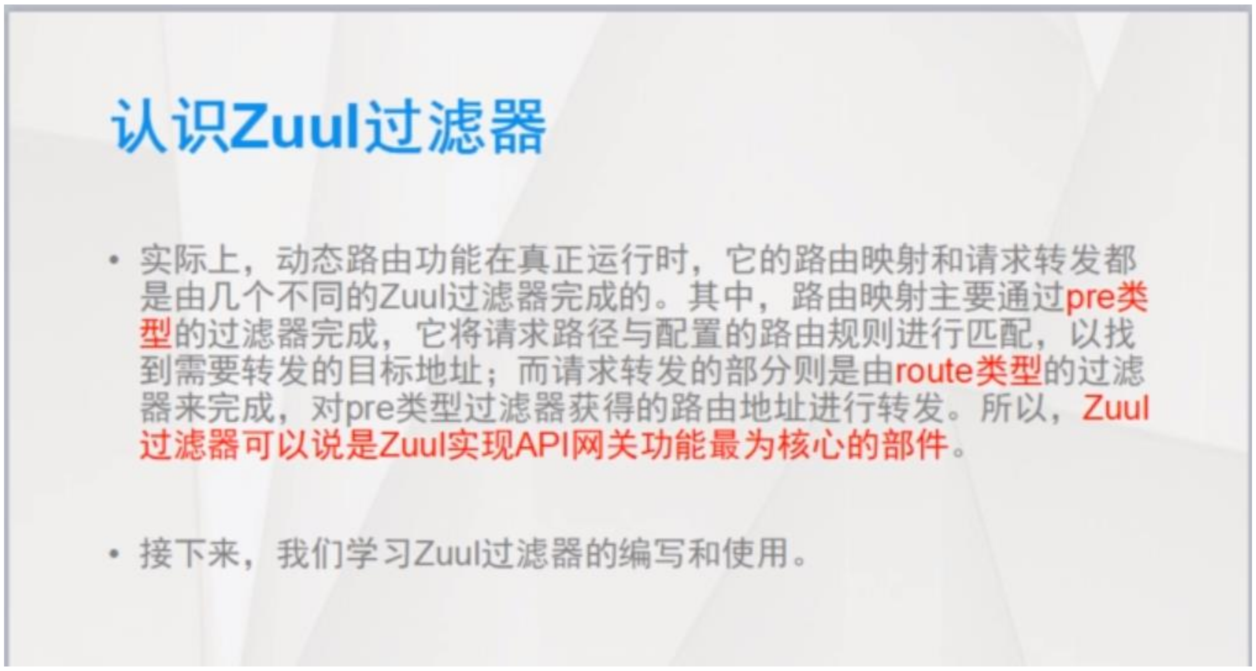
```
id : 2
user : alice
password : 123456
sex : female
money : 2000.4
```

Zuul 实现负载均衡

默认轮询 底层 ribbon 实现， 自动支持 LB

不需要任何配置

Zuul 过滤器



认识Zuul过滤器

- 实际上，动态路由功能在真正运行时，它的路由映射和请求转发都是由几个不同的Zuul过滤器完成的。其中，路由映射主要通过pre类型的过滤器完成，它将请求路径与配置的路由规则进行匹配，以找到需要转发的目标地址；而请求转发的部分则是由route类型的过滤器来完成，对pre类型过滤器获得的路由地址进行转发。所以，Zuul过滤器可以说是Zuul实现API网关功能最为核心的部件。
- 接下来，我们学习Zuul过滤器的编写和使用。

Zuul 过滤器

FilterType:该方法需要返回一个字符串来代表过滤器类型，此类型就是在 HTTP 请求中定义的各个阶段，默认四种不同生命周期的 filtertype，

Pre:可以在请求被路由之前调用

Routing：在路由请求时被调用

Post：routing 和 error 过滤器之后被调用

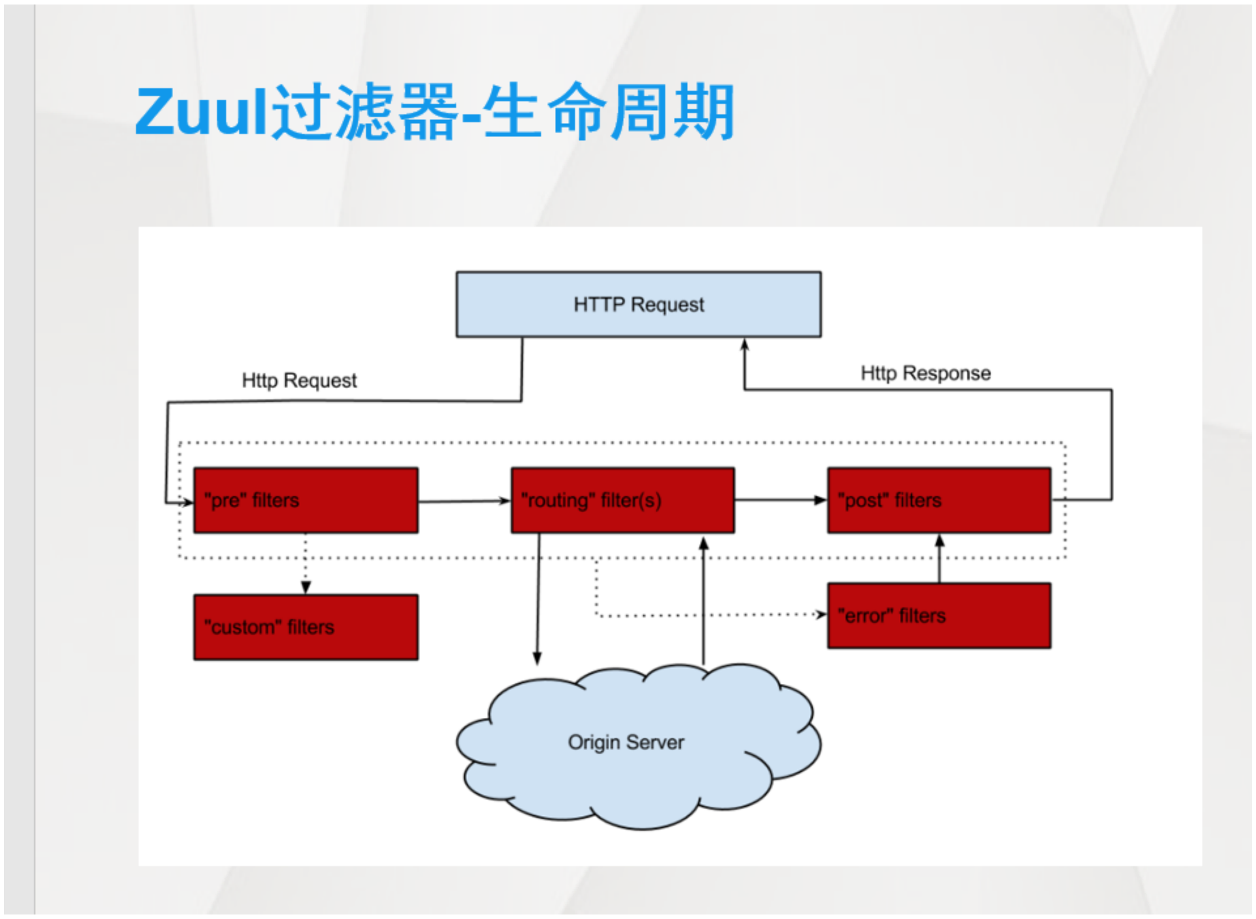
Error：处理请求时发生错误被调用

Filterorder：

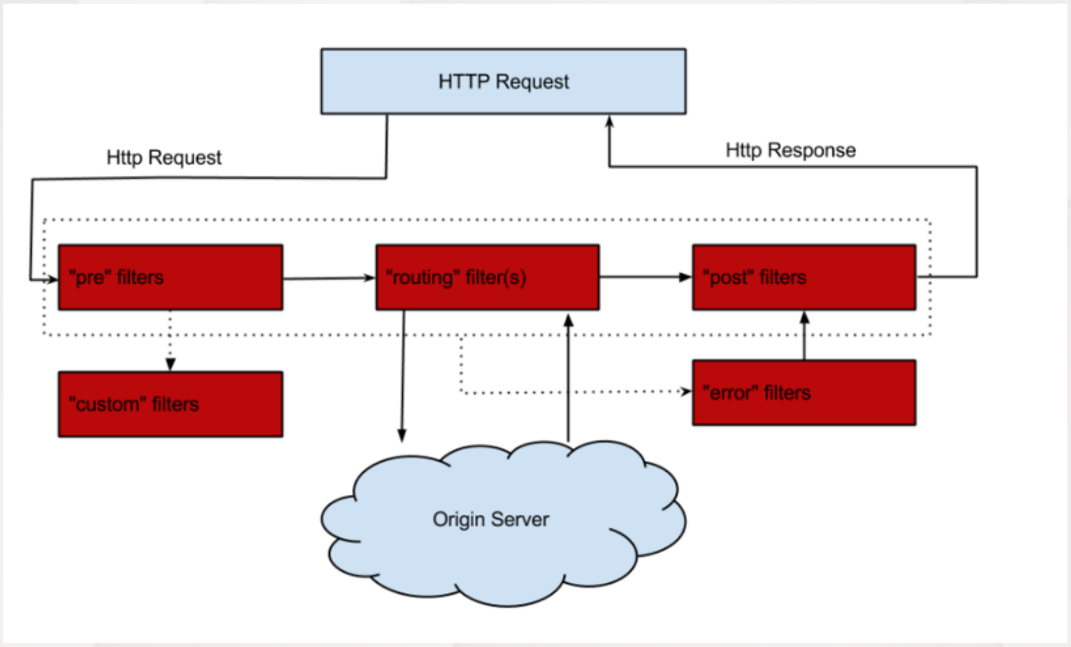
通过 int 值定义过滤器执行顺序，数值越小优先级越高

Shouldfilter：返回一个 boolean 类型来判断该过滤器是否要执行，可以通过此方法指定过滤器的有效范围

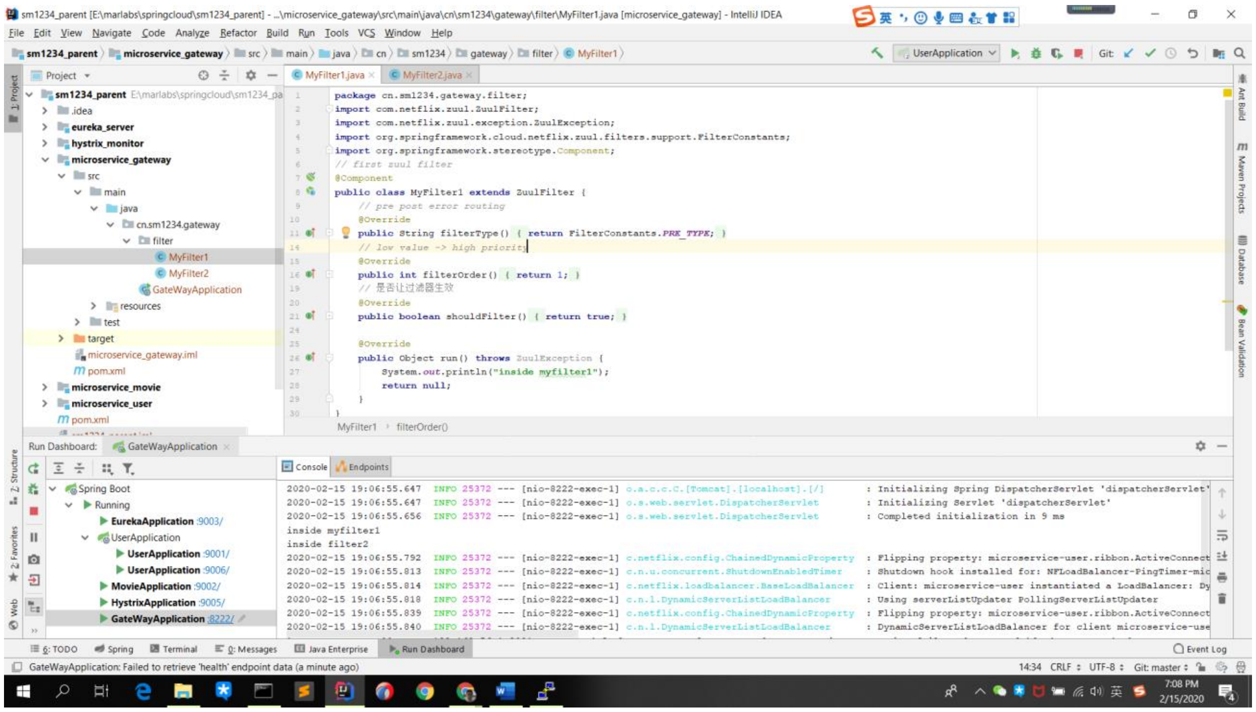
Run：过滤器的具体逻辑，可以实现自定义过滤逻辑，来确定是否拦截当前请求，不对其进行后续的路由，或是在请求路由返回结果之后，对处理结果进行一些加工等。



Zuul过滤器-生命周期



自定义 zuul 过滤器

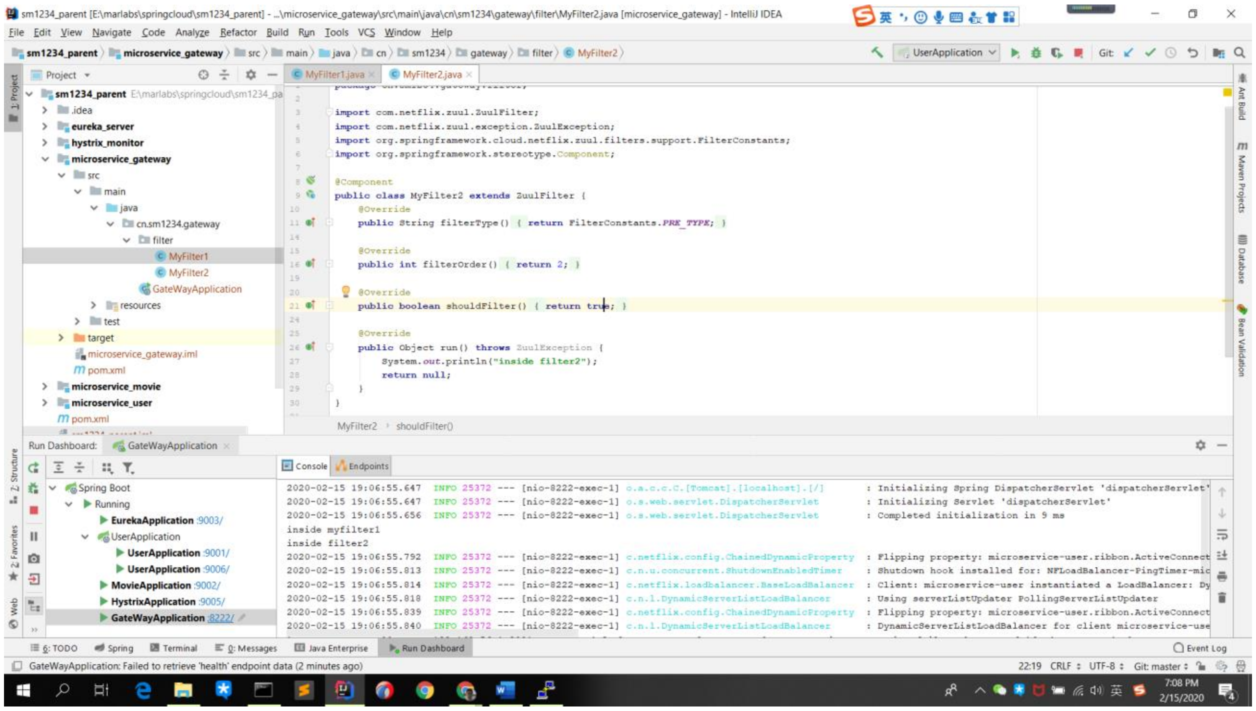


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_gateway\src\main\java\cn\sm1234\gateway\filter\MyFilter1.java [microservice_gateway] - IntelliJ IDEA

```java
package cn.sm1234.gateway.filter;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
// first zuul filter
@Component
public class MyFilter1 extends ZuulFilter {
    // pre post error routing
    @Override
    public String filterType() { return FilterConstants.PRE_TYPE; }
    // low value -> high priority
    @Override
    public int filterOrder() { return 1; }
    // 是否让过滤器生效
    @Override
    public boolean shouldFilter() { return true; }

    @Override
    public Object run() throws ZuulException {
        System.out.println("inside myfilter1");
        return null;
    }
}
```

```
2020-02-15 19:06:55.647  INFO 25372 --- [nio-8222-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-02-15 19:06:55.647  INFO 25372 --- [nio-8222-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-02-15 19:06:55.656  INFO 25372 --- [nio-8222-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 9 ms
inside myfilter1
inside filter2
2020-02-15 19:06:55.792  INFO 25372 --- [nio-8222-exec-1] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:06:55.813  INFO 25372 --- [nio-8222-exec-1] c.n.u.concurrent.ShutdownEnabledTimer    : Shutdown hook installed for: NFLoadBalancer-PingTimer-mic
2020-02-15 19:06:55.814  INFO 25372 --- [nio-8222-exec-1] c.netflix.loadbalancer.BaseLoadBalancer  : Client: microservice-user instantiated a LoadBalancer: Dy
2020-02-15 19:06:55.818  INFO 25372 --- [nio-8222-exec-1] c.n.l.DynamicServerListLoadBalancer      : Using serverListUpdater PollingServerListUpdater
2020-02-15 19:06:55.839  INFO 25372 --- [nio-8222-exec-1] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:06:55.840  INFO 25372 --- [nio-8222-exec-1] c.n.l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client microservice-use
```

GateWayApplication: Failed to retrieve 'health' endpoint data (a minute ago)

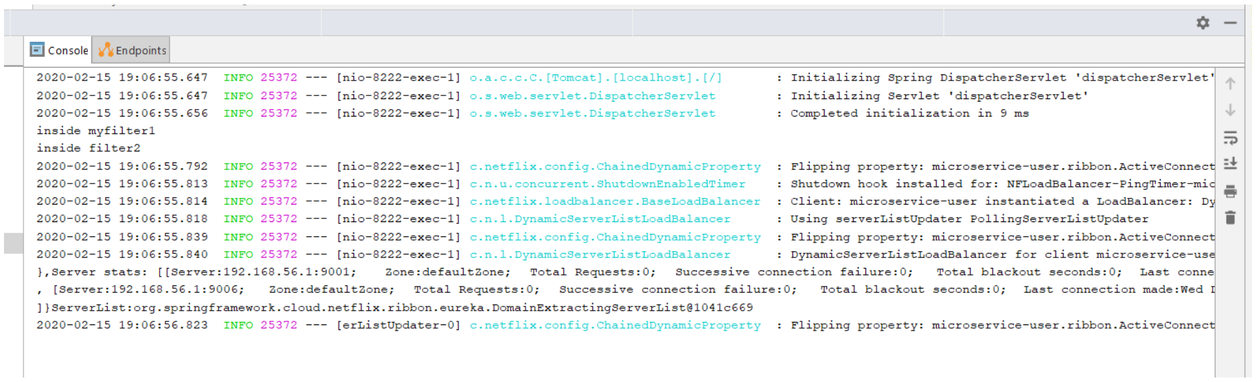


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_gateway\src\main\java\cn\sm1234\gateway\filter\MyFilter2.java [microservice_gateway] - IntelliJ IDEA

```java
package cn.sm1234.gateway.filter;

import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;

@Component
public class MyFilter2 extends ZuulFilter {
    @Override
    public String filterType() { return FilterConstants.PRE_TYPE; }

    @Override
    public int filterOrder() { return 2; }

    @Override
    public boolean shouldFilter() { return true; }

    @Override
    public Object run() throws ZuulException {
        System.out.println("inside filter2");
        return null;
    }
}
```

```
2020-02-15 19:06:55.647  INFO 25372 --- [nio-8222-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-02-15 19:06:55.647  INFO 25372 --- [nio-8222-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-02-15 19:06:55.656  INFO 25372 --- [nio-8222-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 9 ms
inside myfilter1
inside filter2
2020-02-15 19:06:55.792  INFO 25372 --- [nio-8222-exec-1] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:06:55.813  INFO 25372 --- [nio-8222-exec-1] c.n.u.concurrent.ShutdownEnabledTimer    : Shutdown hook installed for: NFLoadBalancer-PingTimer-mic
2020-02-15 19:06:55.814  INFO 25372 --- [nio-8222-exec-1] c.netflix.loadbalancer.BaseLoadBalancer  : Client: microservice-user instantiated a LoadBalancer: Dy
2020-02-15 19:06:55.818  INFO 25372 --- [nio-8222-exec-1] c.n.l.DynamicServerListLoadBalancer      : Using serverListUpdater PollingServerListUpdater
2020-02-15 19:06:55.839  INFO 25372 --- [nio-8222-exec-1] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:06:55.840  INFO 25372 --- [nio-8222-exec-1] c.n.l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client microservice-use
```

GateWayApplication: Failed to retrieve 'health' endpoint data (2 minutes ago)

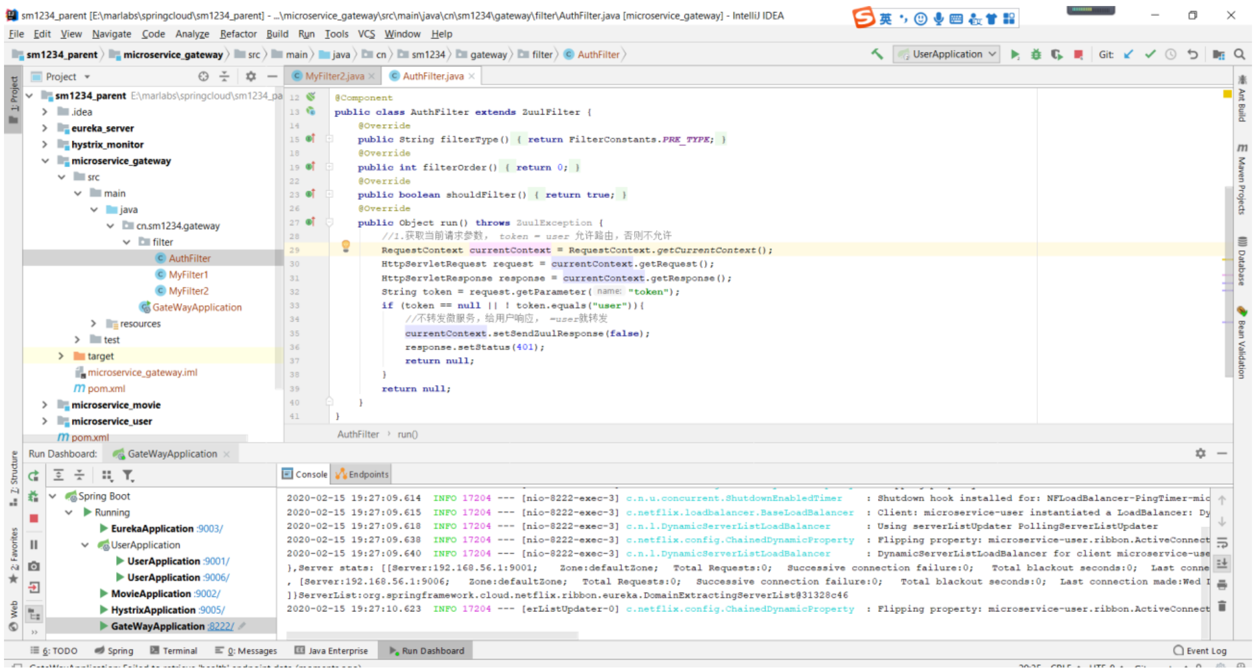


```
2020-02-15 19:06:55.647  INFO 25372 --- [nio-8222-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-02-15 19:06:55.647  INFO 25372 --- [nio-8222-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-02-15 19:06:55.656  INFO 25372 --- [nio-8222-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 9 ms
inside myfilter1
inside filter2
2020-02-15 19:06:55.792  INFO 25372 --- [nio-8222-exec-1] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:06:55.813  INFO 25372 --- [nio-8222-exec-1] c.n.u.concurrent.ShutdownEnabledTimer    : Shutdown hook installed for: NFLoadBalancer-PingTimer-mic
2020-02-15 19:06:55.814  INFO 25372 --- [nio-8222-exec-1] c.netflix.loadbalancer.BaseLoadBalancer  : Client: microservice-user instantiated a LoadBalancer: Dy
2020-02-15 19:06:55.818  INFO 25372 --- [nio-8222-exec-1] c.n.l.DynamicServerListLoadBalancer      : Using serverListUpdater PollingServerListUpdater
2020-02-15 19:06:55.839  INFO 25372 --- [nio-8222-exec-1] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:06:55.840  INFO 25372 --- [nio-8222-exec-1] c.n.l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client microservice-use
},Server stats: [[Server:192.168.56.1:9001;   Zone:defaultZone;  Total Requests:0;  Successive connection failure:0;  Total blackout seconds:0;  Last conne
, [Server:192.168.56.1:9006;   Zone:defaultZone;  Total Requests:0;  Successive connection failure:0;  Total blackout seconds:0;  Last connection made:Wed I
]}ServerList:org.springframework.cloud.netflix.ribbon.eureka.DomainExtractingServerList@1041c669
2020-02-15 19:06:56.823  INFO 25372 --- [erListUpdater-0] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
```

Zuul 实现权限验证：

收到请求，验证权限 =》决定是否路由

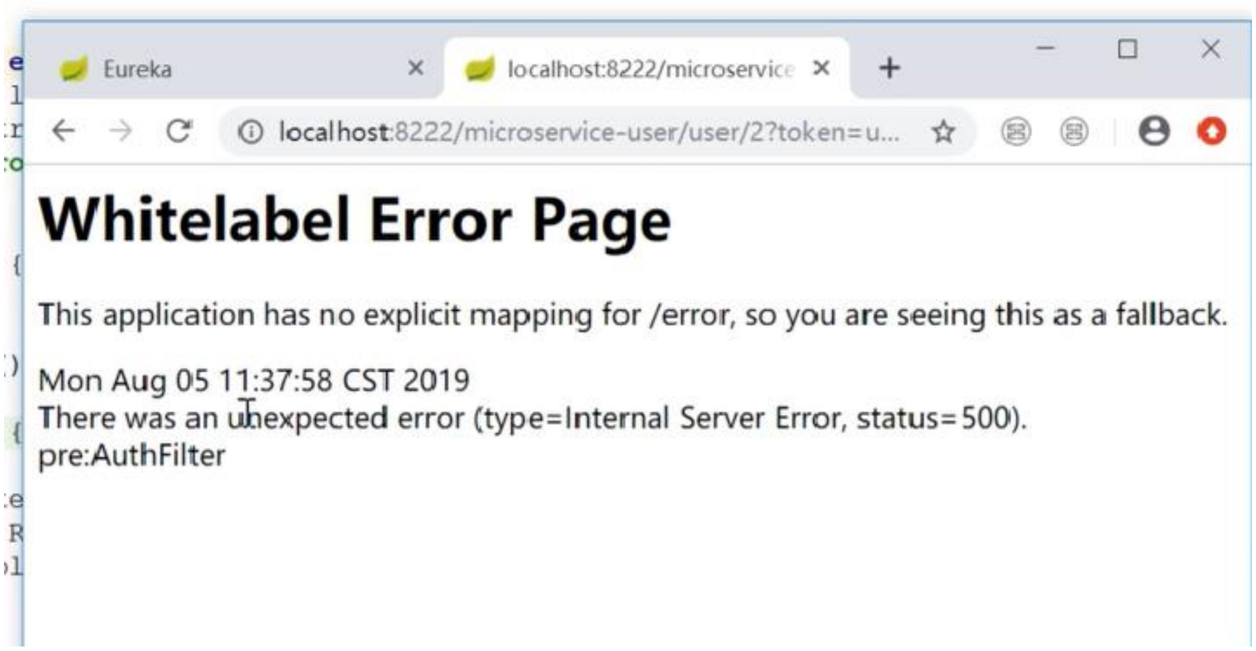
属于前置过滤器 pre 类型



```java
@Component
public class AuthFilter extends ZuulFilter {
    @Override
    public String filterType() { return FilterConstants.PRE_TYPE; }
    @Override
    public int filterOrder() { return 0; }
    @Override
    public boolean shouldFilter() { return true; }
    @Override
    public Object run() throws ZuulException {
        //1.获取当前请求参数， token = user 允许路由，否则不允许
        RequestContext currentContext = RequestContext.getCurrentContext();
        HttpServletRequest request = currentContext.getRequest();
        HttpServletResponse response = currentContext.getResponse();
        String token = request.getParameter( name: "token");
        if (token == null || ! token.equals("user")){
            //不转发微服务，给用户响应， =user就转发
            currentContext.setSendZuulResponse(false);
            response.setStatus(401);
            return null;
        }
        return null;
    }
}
```

```
2020-02-15 19:27:09.614  INFO 17204 --- [nio-8222-exec-3] c.n.u.concurrent.ShutdownEnabledTimer    : Shutdown hook installed for: NFLoadBalancer-PingTimer-mic
2020-02-15 19:27:09.615  INFO 17204 --- [nio-8222-exec-3] c.netflix.loadbalancer.BaseLoadBalancer  : Client: microservice-user instantiated a LoadBalancer: Dy
2020-02-15 19:27:09.618  INFO 17204 --- [nio-8222-exec-3] c.n.l.DynamicServerListLoadBalancer      : Using serverListUpdater PollingServerListUpdater
2020-02-15 19:27:09.638  INFO 17204 --- [nio-8222-exec-3] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
2020-02-15 19:27:09.640  INFO 17204 --- [nio-8222-exec-3] c.n.l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client microservice-use
},Server stats: [[Server:192.168.56.1:9001;   Zone:defaultZone;  Total Requests:0;  Successive connection failure:0;  Total blackout seconds:0;  Last conne
, [Server:192.168.56.1:9006;   Zone:defaultZone;  Total Requests:0;  Successive connection failure:0;  Total blackout seconds:0;  Last connection made:Wed I
]}ServerList:org.springframework.cloud.netflix.ribbon.eureka.DomainExtractingServerList@31328c46
2020-02-15 19:27:10.623  INFO 17204 --- [erListUpdater-0] c.netflix.config.ChainedDynamicProperty  : Flipping property: microservice-user.ribbon.ActiveConnect
```

Zuul 过滤器：异常处理

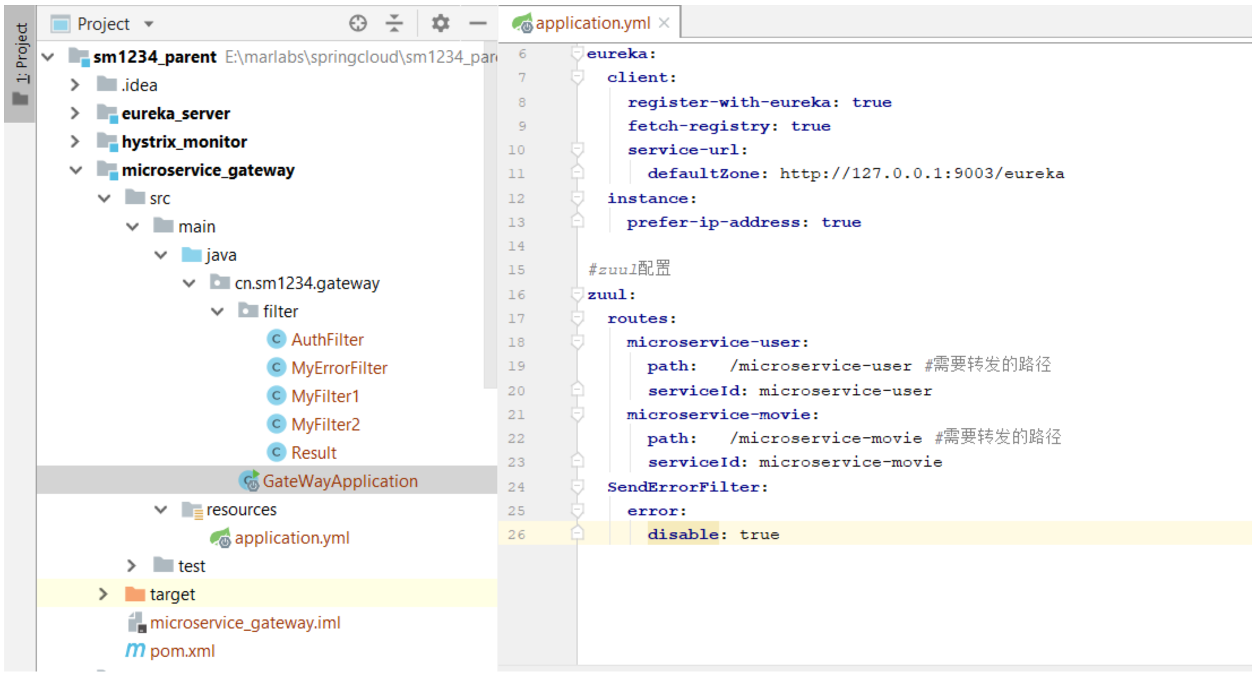


```
Whitelabel Error Page

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

Mon Aug 05 11:37:58 CST 2019
There was an unexpected error (type=Internal Server Error, status=500).
pre:AuthFilter
```

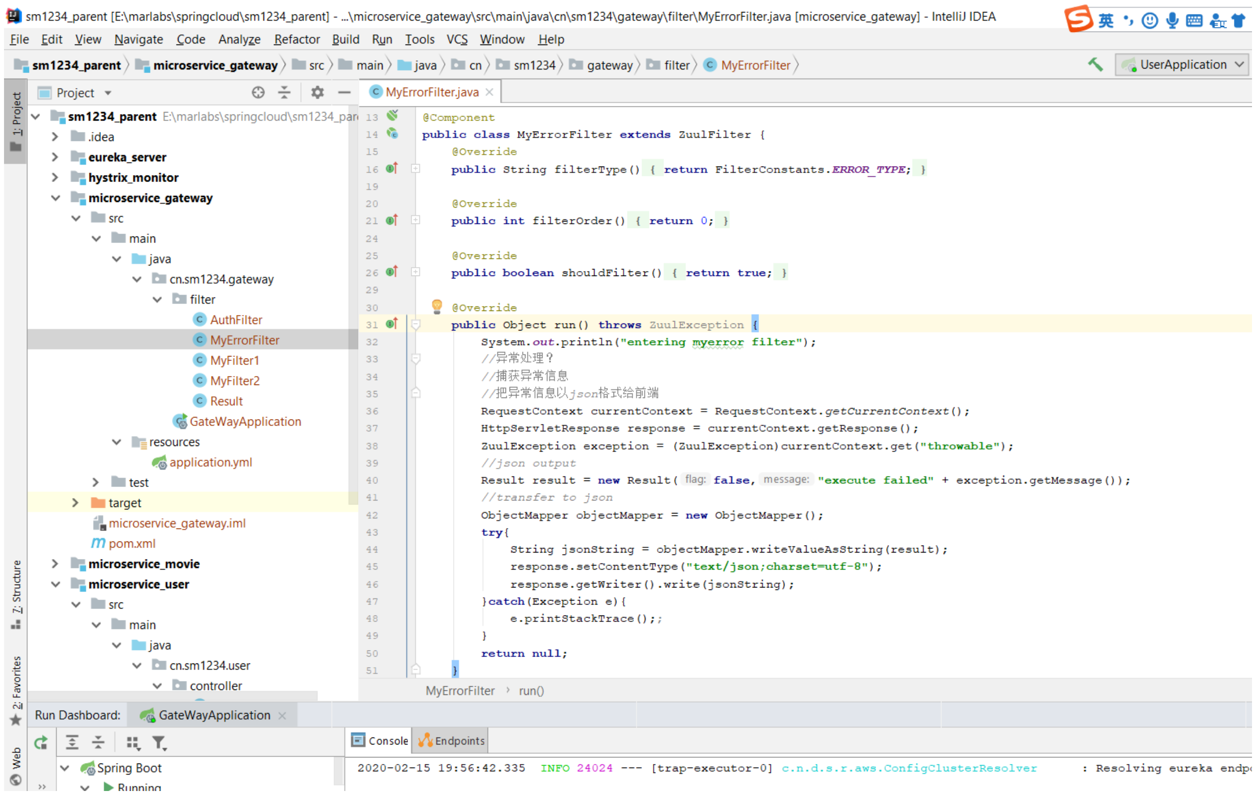
More friendly error page?

自定义异常处理

配置文件里 Disable 掉默认的 error filter，加入自定义 filter



```yaml
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://127.0.0.1:9003/eureka
  instance:
    prefer-ip-address: true

#zuul配置
zuul:
  routes:
    microservice-user:
      path:   /microservice-user #需要转发的路径
      serviceId: microservice-user
    microservice-movie:
      path:   /microservice-movie #需要转发的路径
      serviceId: microservice-movie
  SendErrorFilter:
    error:
      disable: true
```



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_gateway\src\main\java\cn\sm1234\gateway\filter\MyErrorFilter.java [microservice_gateway] - IntelliJ IDEA

```java
@Component
public class MyErrorFilter extends ZuulFilter {
    @Override
    public String filterType() { return FilterConstants.ERROR_TYPE; }

    @Override
    public int filterOrder() { return 0; }

    @Override
    public boolean shouldFilter() { return true; }

    @Override
    public Object run() throws ZuulException {
        System.out.println("entering myerror filter");
        //异常处理？
        //捕获异常信息
        //把异常信息以json格式给前端
        RequestContext currentContext = RequestContext.getCurrentContext();
        HttpServletResponse response = currentContext.getResponse();
        ZuulException exception = (ZuulException)currentContext.get("throwable");
        //json output
        Result result = new Result( flag: false, message: "execute failed" + exception.getMessage());
        //transfer to json
        ObjectMapper objectMapper = new ObjectMapper();
        try{
            String jsonString = objectMapper.writeValueAsString(result);
            response.setContentType("text/json;charset=utf-8");
            response.getWriter().write(jsonString);
        }catch(Exception e){
            e.printStackTrace();;
        }
        return null;
    }
```

```
2020-02-15 19:56:42.335  INFO 24024 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endp
```

Zuul 网关整合 swagger

什么是 swagger?

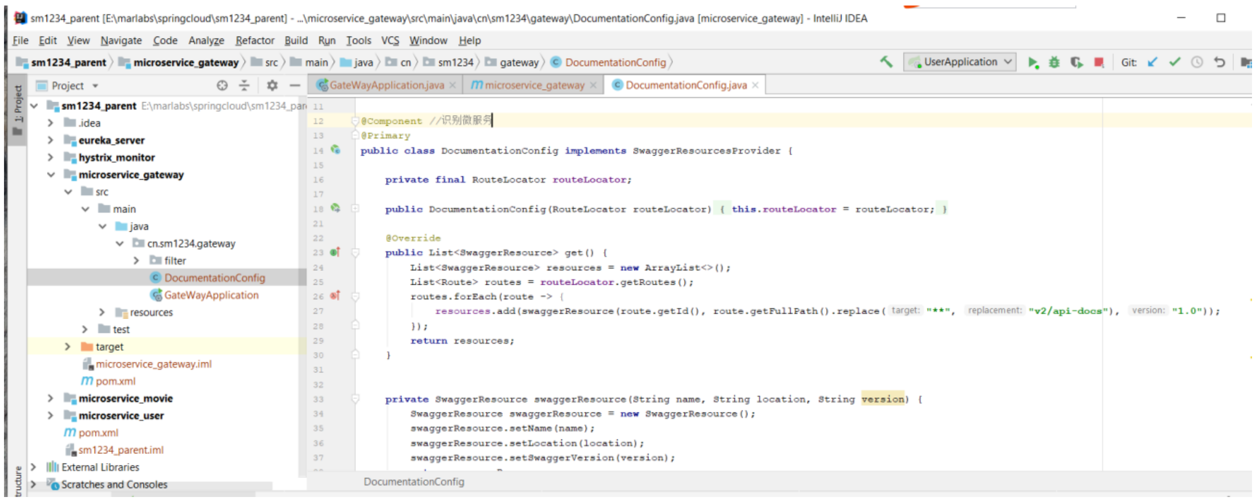
API 文档标准

步骤:

网关引入 swagger 依赖

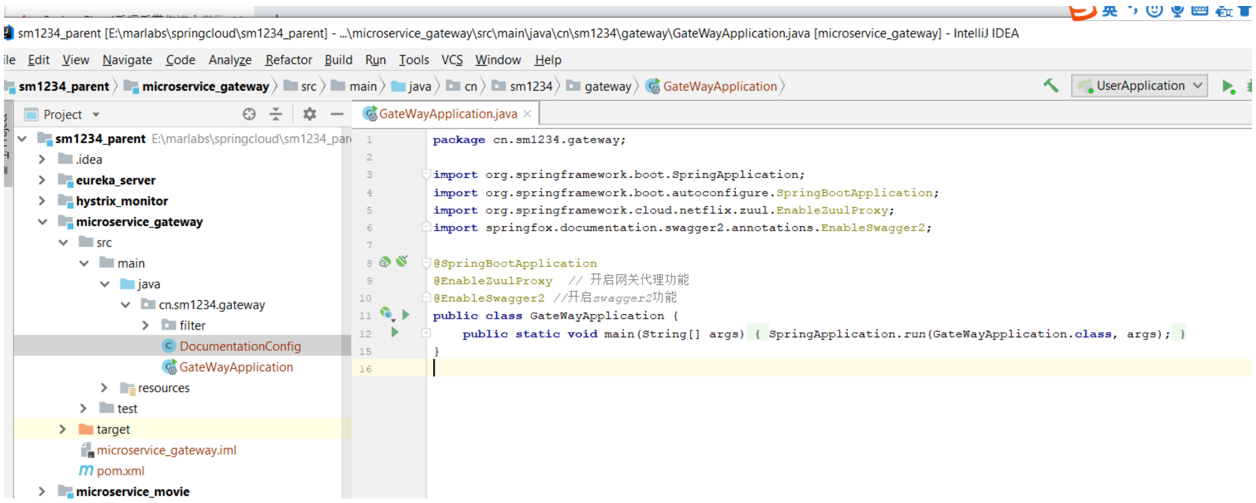
```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

拷贝 config 类


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_gateway\src\main\java\cn\sm1234\gateway\DocumentationConfig.java [microservice_gateway] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sm1234_parent microservice_gateway src main java cn sm1234 gateway DocumentationConfig
UserApplication
Git:
Project
GateWayApplication.java
microservice_gateway
DocumentationConfig.java
sm1234_parent E:\marlabs\springcloud\sm1234_parent
.idea
eureka_server
hystrix_monitor
microservice_gateway
src
main
java
cn.sm1234.gateway
filter
DocumentationConfig
GateWayApplication
resources
test
target
microservice_gateway.iml
pom.xml
microservice_movie
microservice_user
pom.xml
sm1234_parent.iml
External Libraries
Scratches and Consoles
DocumentationConfig

```java
@Component //识别微服务
@Primary
public class DocumentationConfig implements SwaggerResourcesProvider {

    private final RouteLocator routeLocator;

    public DocumentationConfig(RouteLocator routeLocator) { this.routeLocator = routeLocator; }

    @Override
    public List<SwaggerResource> get() {
        List<SwaggerResource> resources = new ArrayList<>();
        List<Route> routes = routeLocator.getRoutes();
        routes.forEach(route -> {
            resources.add(swaggerResource(route.getId(), route.getFullPath().replace( target: "**", replacement: "v2/api-docs"), version: "1.0"));
        });
        return resources;
    }


    private SwaggerResource swaggerResource(String name, String location, String version) {
        SwaggerResource swaggerResource = new SwaggerResource();
        swaggerResource.setName(name);
        swaggerResource.setLocation(location);
        swaggerResource.setSwaggerVersion(version);
```

开启 swagger2 在启动类中



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_gateway\src\main\java\cn\sm1234\gateway\GateWayApplication.java [microservice_gateway] - IntelliJ IDEA

```java
package cn.sm1234.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication
@EnableZuulProxy  // 开启网关代理功能
@EnableSwagger2 //开启swagger2功能
public class GateWayApplication {
    public static void main(String[] args) { SpringApplication.run(GateWayApplication.class, args); }
}
```

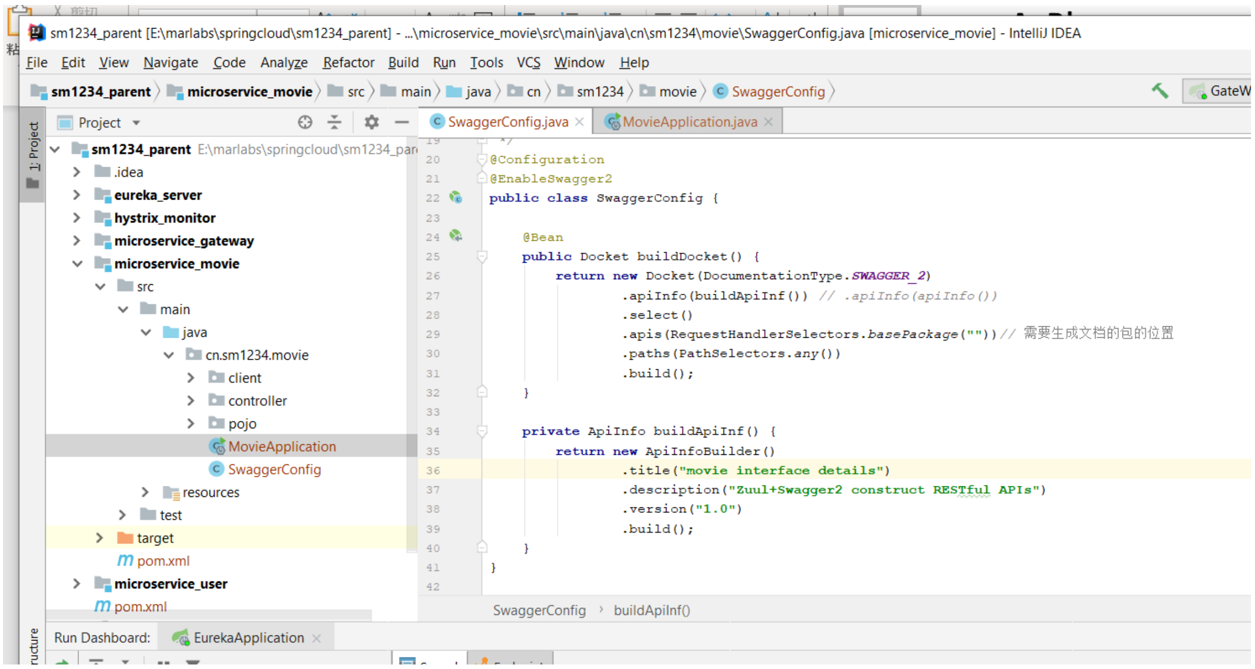
输入 http://localhost:8222/swagger-ui.html 查看 swaggerui 界面

将电影 service 暴露给 swagger2.

导入 pom 中的 swagger 依赖

启动类中@EnableSwagger2

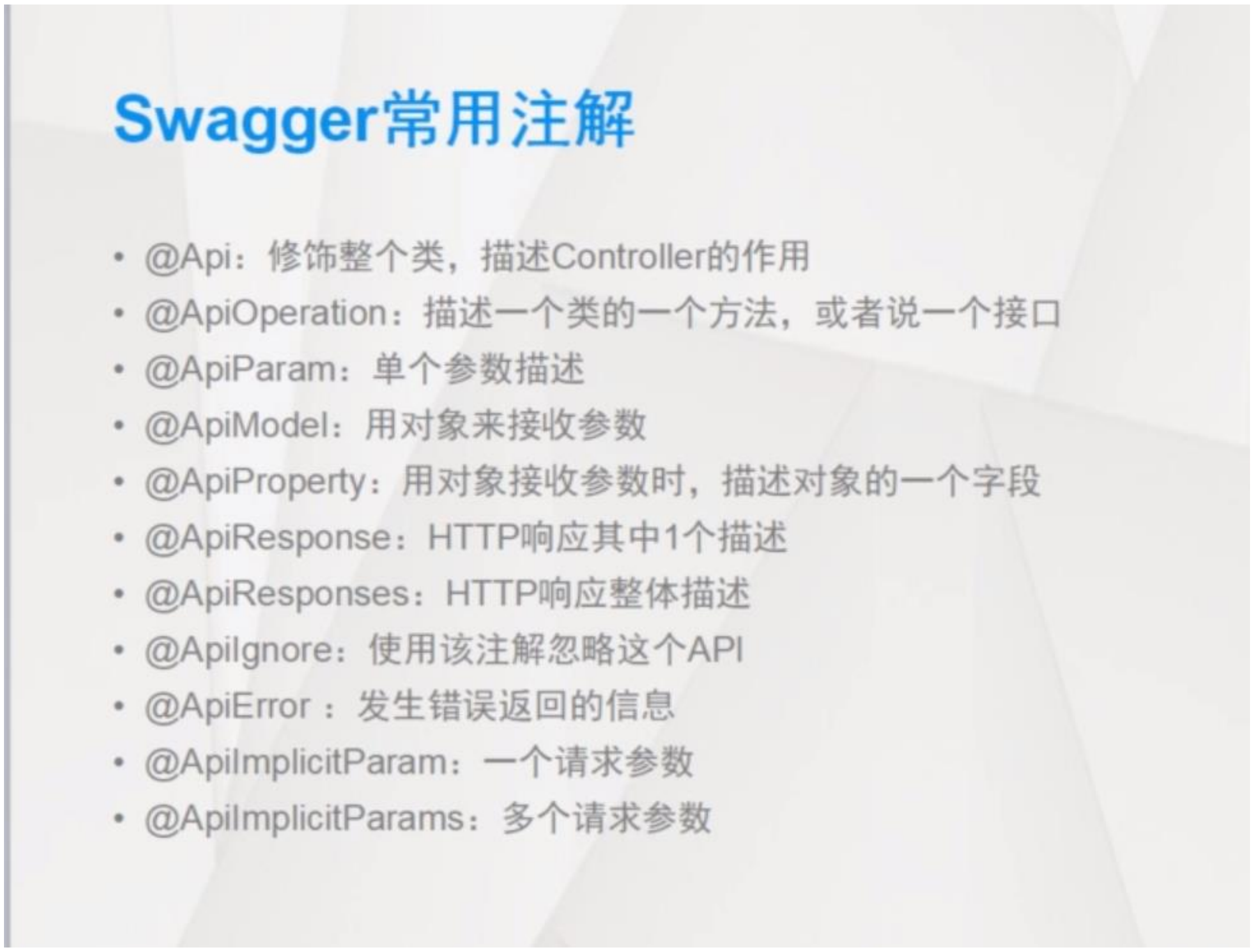
加入 SwaggerConfig 暴露类



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_movie\src\main\java\cn\sm1234\movie\SwaggerConfig.java [microservice_movie] - IntelliJ IDEA

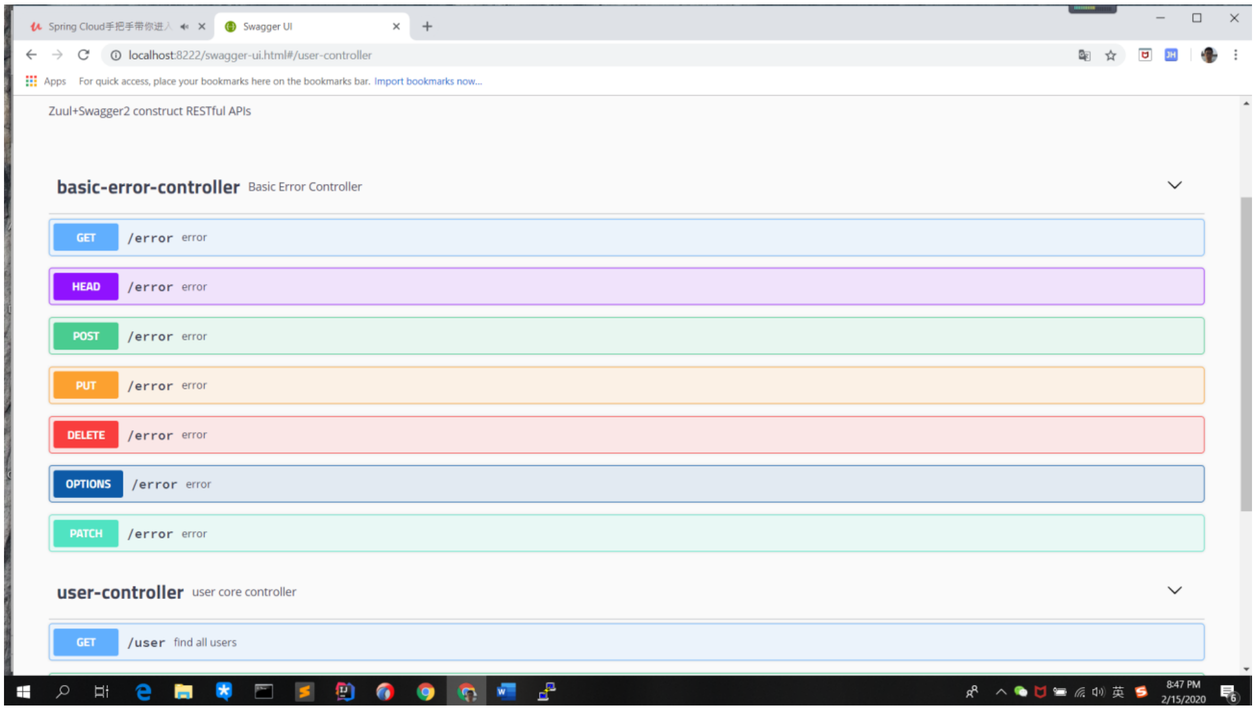
```java
@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Bean
    public Docket buildDocket() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(buildApiInf()) // .apiInfo(apiInfo())
                .select()
                .apis(RequestHandlerSelectors.basePackage(""))// 需要生成文档的包的位置
                .paths(PathSelectors.any())
                .build();
    }

    private ApiInfo buildApiInf() {
        return new ApiInfoBuilder()
                .title("movie interface details")
                .description("Zuul+Swagger2 construct RESTful APIs")
                .version("1.0")
                .build();
    }
}
```

Swagger 常用注解:



Swagger常用注解

- @Api：修饰整个类，描述Controller的作用
- @ApiOperation：描述一个类的一个方法，或者说一个接口
- @ApiParam：单个参数描述
- @ApiModel：用对象来接收参数
- @ApiProperty：用对象接收参数时，描述对象的一个字段
- @ApiResponse：HTTP响应其中1个描述
- @ApiResponses：HTTP响应整体描述
- @ApiIgnore：使用该注解忽略这个API
- @ApiError ：发生错误返回的信息
- @ApiImplicitParam：一个请求参数
- @ApiImplicitParams：多个请求参数



Zuul+Swagger2 construct RESTful APIs

basic-error-controller Basic Error Controller

GET /error error

HEAD /error error

POST /error error

PUT /error error

DELETE /error error

OPTIONS /error error

PATCH /error error

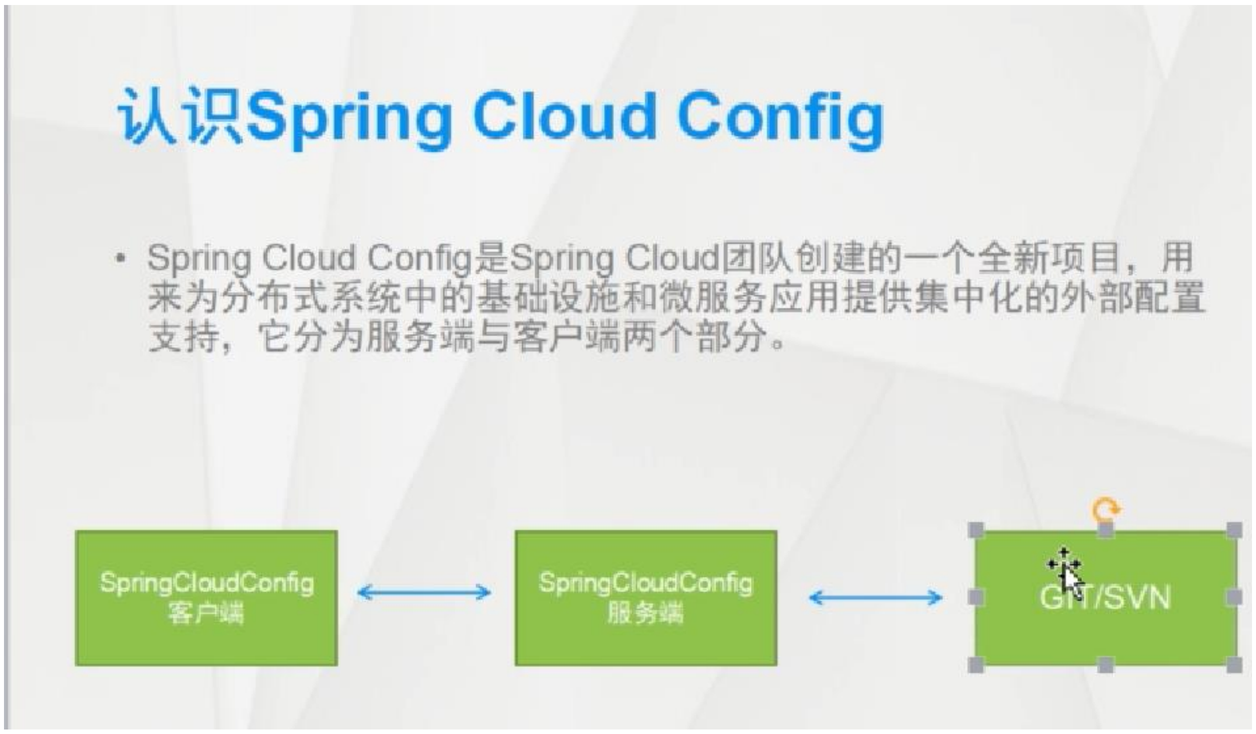
user-controller user core controller

GET /user find all users

Spring Cloud手把手带你进入
Swagger UI
localhost:8222/swagger-ui.html#/user-controller
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
8:47 PM
2/15/2020

第六章:

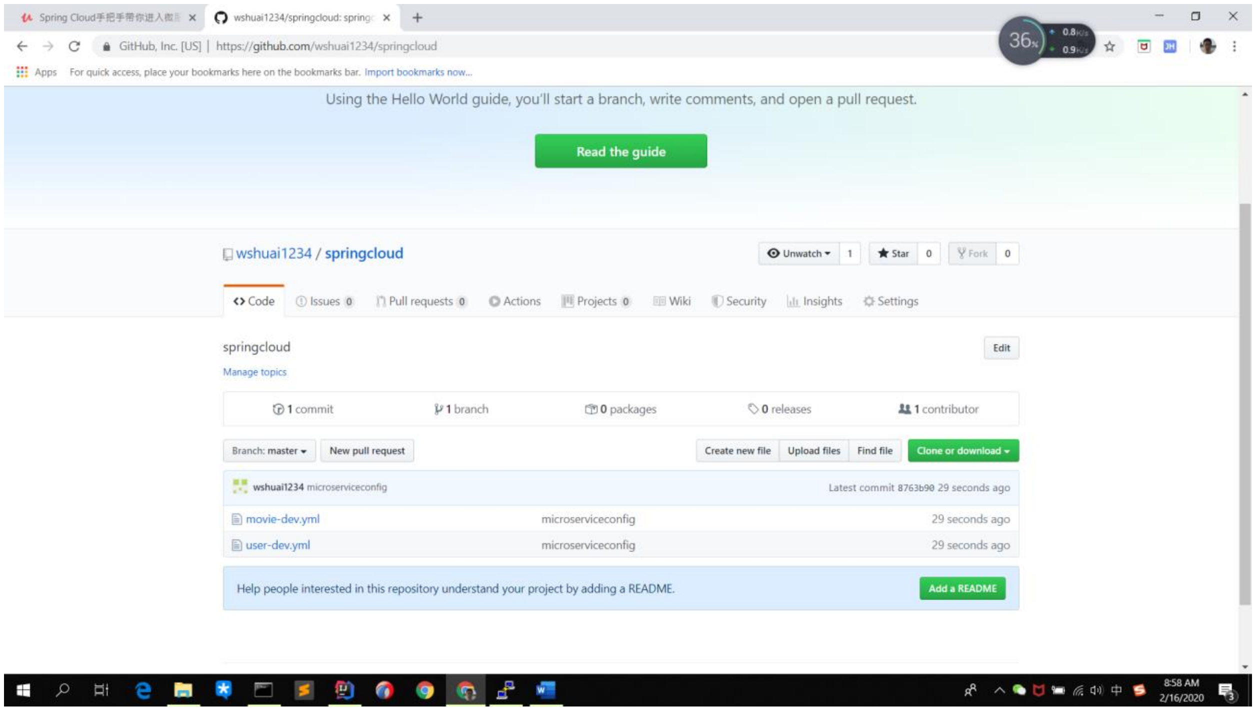
Spring cloud config　集中配置管理中心



认识Spring Cloud Config

- Spring Cloud Config是Spring Cloud团队创建的一个全新项目，用来为分布式系统中的基础设施和微服务应用提供集中化的外部配置支持，它分为服务端与客户端两个部分。

微服务配置上传 github


wshuai1234 / springcloud

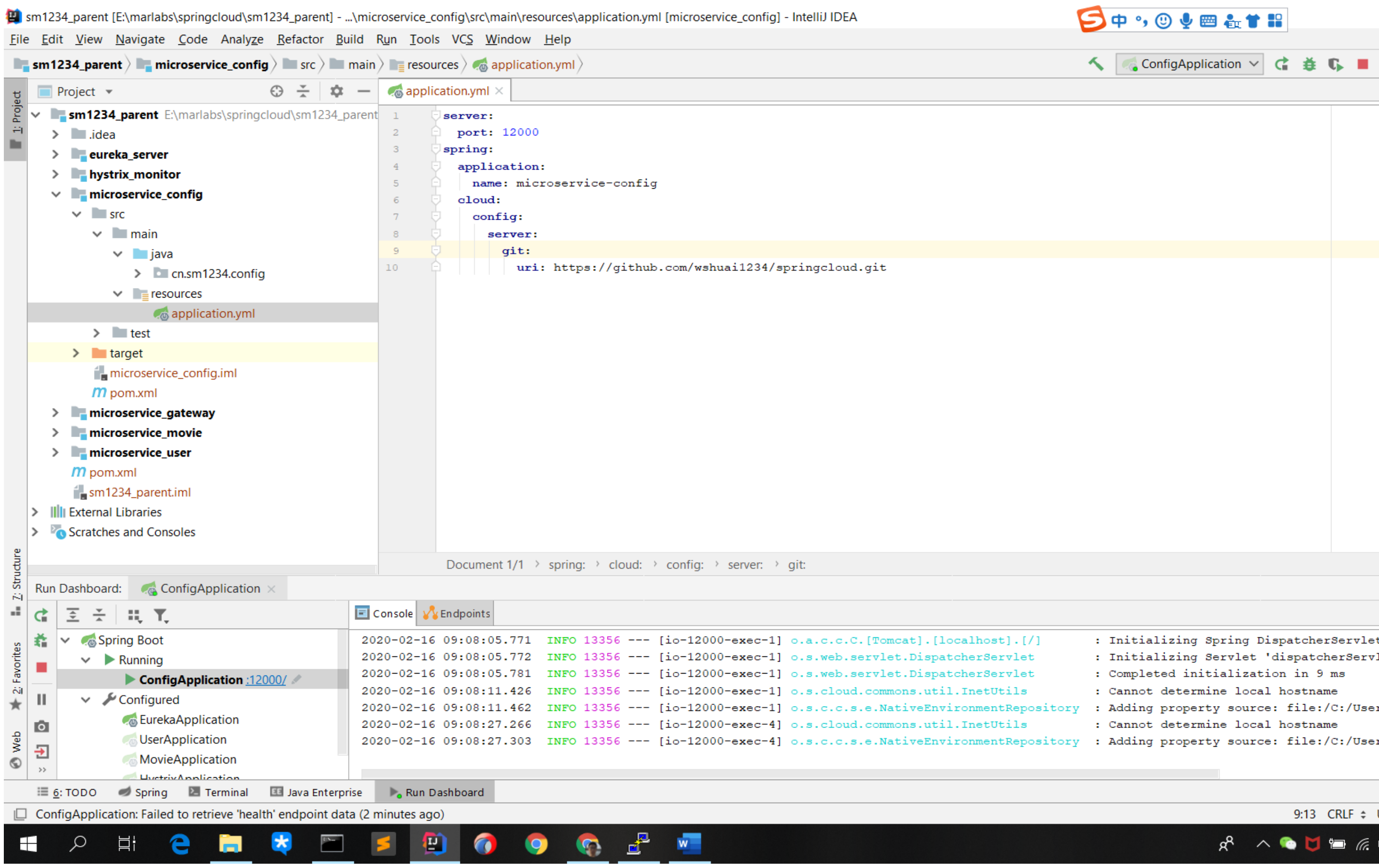
springcloud

1 commit · 1 branch · 0 packages · 0 releases · 1 contributor

wshuai1234 microserviceconfig   Latest commit 8763b90 29 seconds ago

movie-dev.yml	microserviceconfig	29 seconds ago
user-dev.yml	microserviceconfig	29 seconds ago

搭建 springcloudconfig 服务端:连接 github 仓库，获取配置文件



sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_config\src\main\resources\application.yml [microservice_config] - IntelliJ IDEA

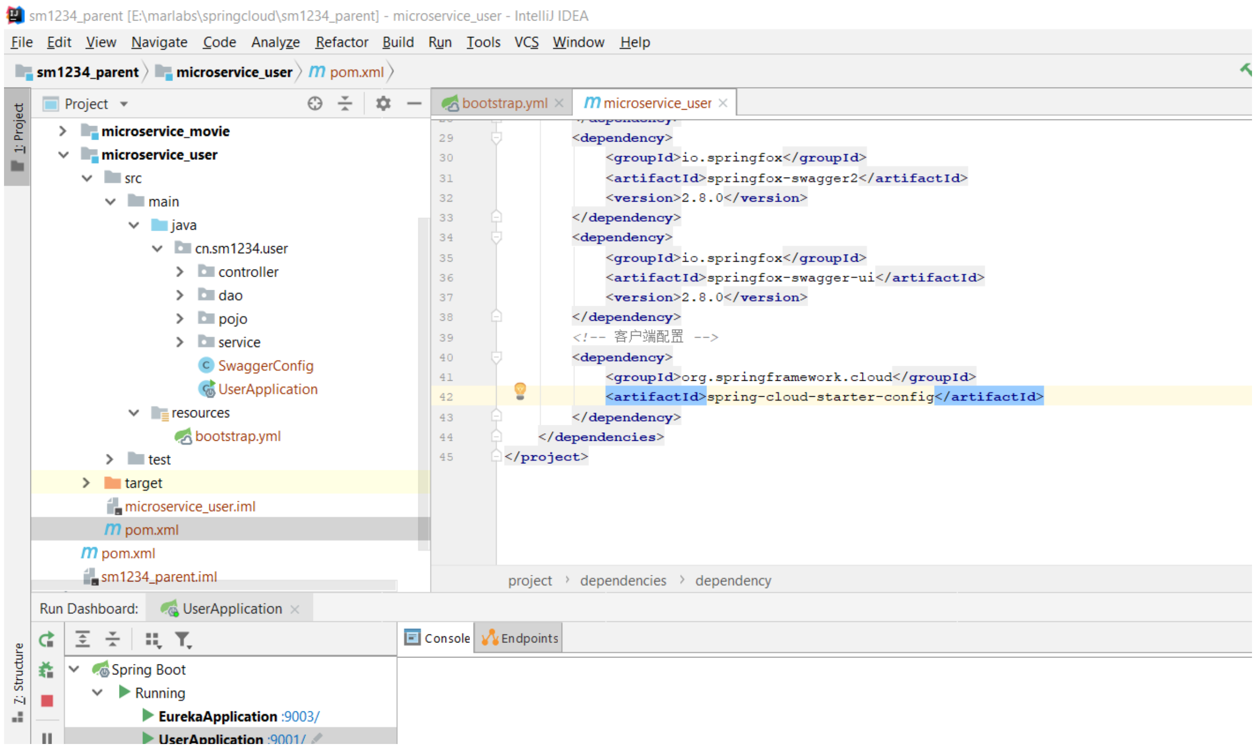
```yaml
server:
  port: 12000
spring:
  application:
    name: microservice-config
  cloud:
    config:
      server:
        git:
          uri: https://github.com/wshuai1234/springcloud.git
```

```
2020-02-16 09:08:05.771  INFO 13356 --- [io-12000-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet
2020-02-16 09:08:05.772  INFO 13356 --- [io-12000-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServl
2020-02-16 09:08:05.781  INFO 13356 --- [io-12000-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 9 ms
2020-02-16 09:08:11.426  INFO 13356 --- [io-12000-exec-1] o.s.cloud.commons.util.InetUtils         : Cannot determine local hostname
2020-02-16 09:08:11.462  INFO 13356 --- [io-12000-exec-1] o.s.c.c.s.e.NativeEnvironmentRepository  : Adding property source: file:/C:/Use
2020-02-16 09:08:27.266  INFO 13356 --- [io-12000-exec-4] o.s.cloud.commons.util.InetUtils         : Cannot determine local hostname
2020-02-16 09:08:27.303  INFO 13356 --- [io-12000-exec-4] o.s.c.c.s.e.NativeEnvironmentRepository  : Adding property source: file:/C:/Use
```

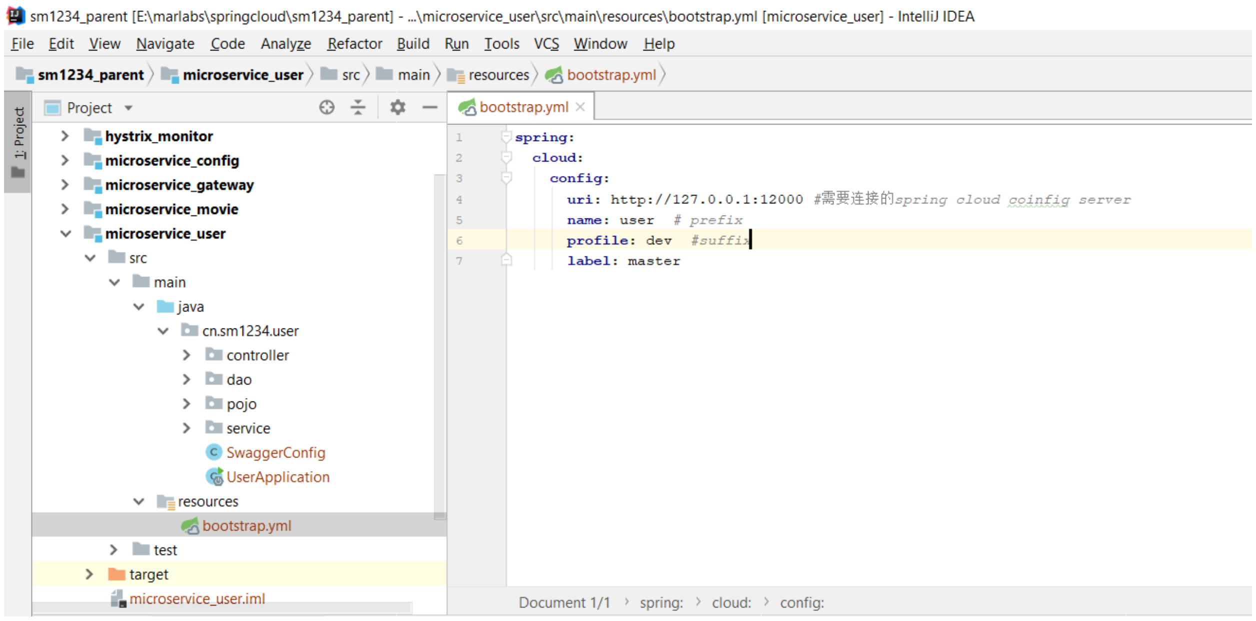
ConfigApplication: Failed to retrieve 'health' endpoint data (2 minutes ago)

搭建客户端

引入 client 依赖，创建配置引导文件


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - microservice_user - IntelliJ IDEA

```xml
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.8.0</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.8.0</version>
        </dependency>
        <!-- 客户端配置 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>
    </dependencies>
</project>
```

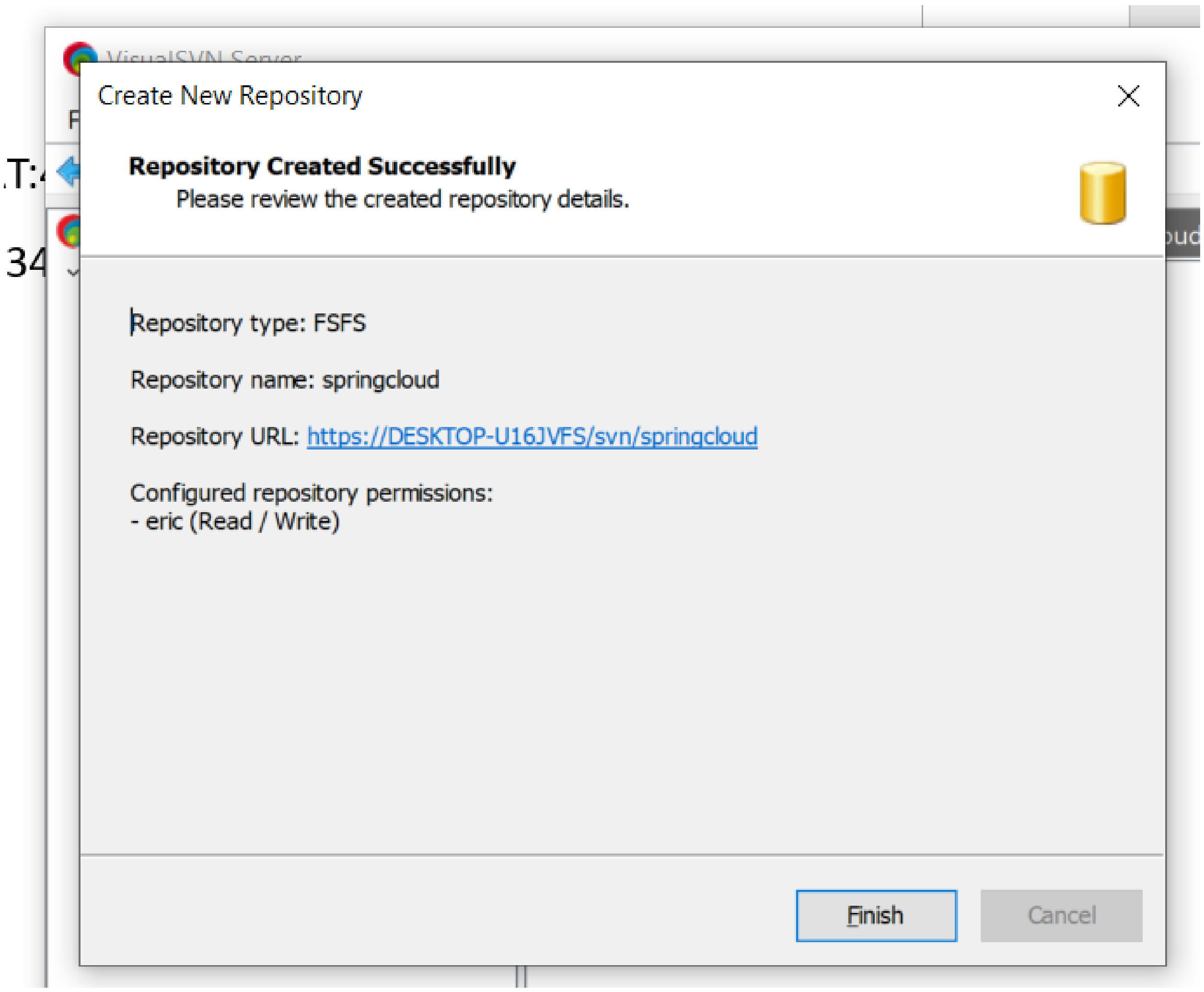

sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_user\src\main\resources\bootstrap.yml [microservice_user] - IntelliJ IDEA

```yaml
spring:
  cloud:
    config:
      uri: http://127.0.0.1:12000  #需要连接的spring cloud coinfig server
      name: user  # prefix
      profile: dev  #suffix
      label: master
```

存放到 SVN 仓库

Visual SVN SERVER PORT:443

创建用户 eric 密码 123456



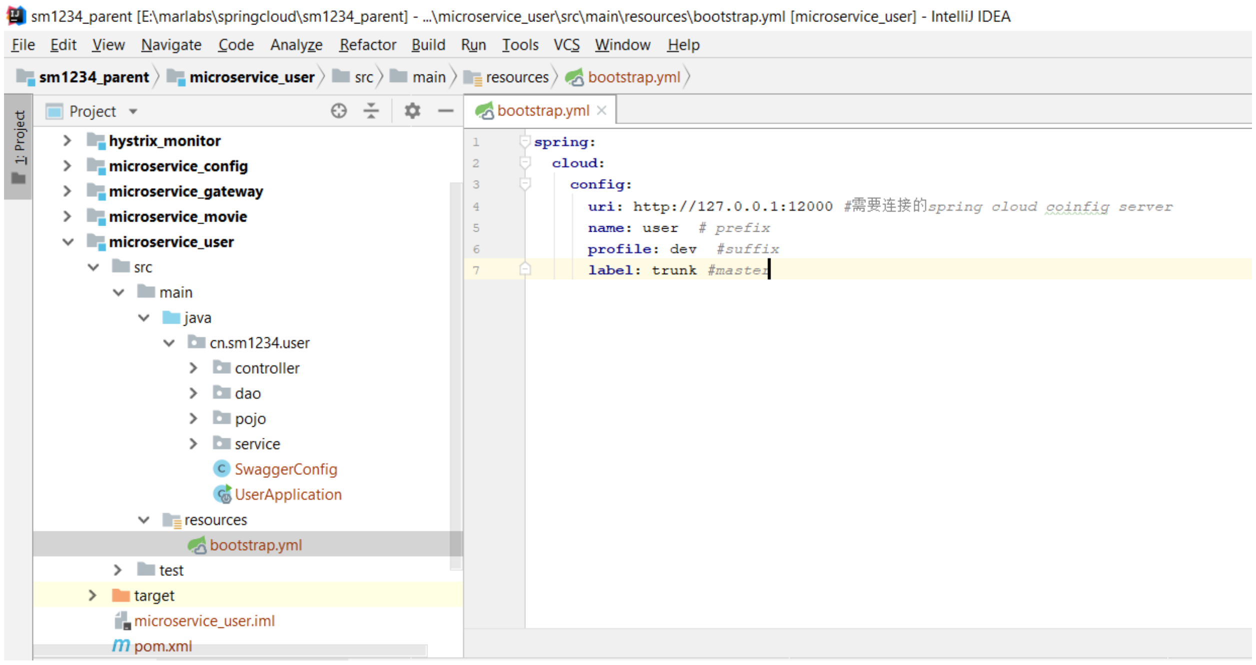
Repository Created Successfully
Please review the created repository details.

Repository type: FSFS

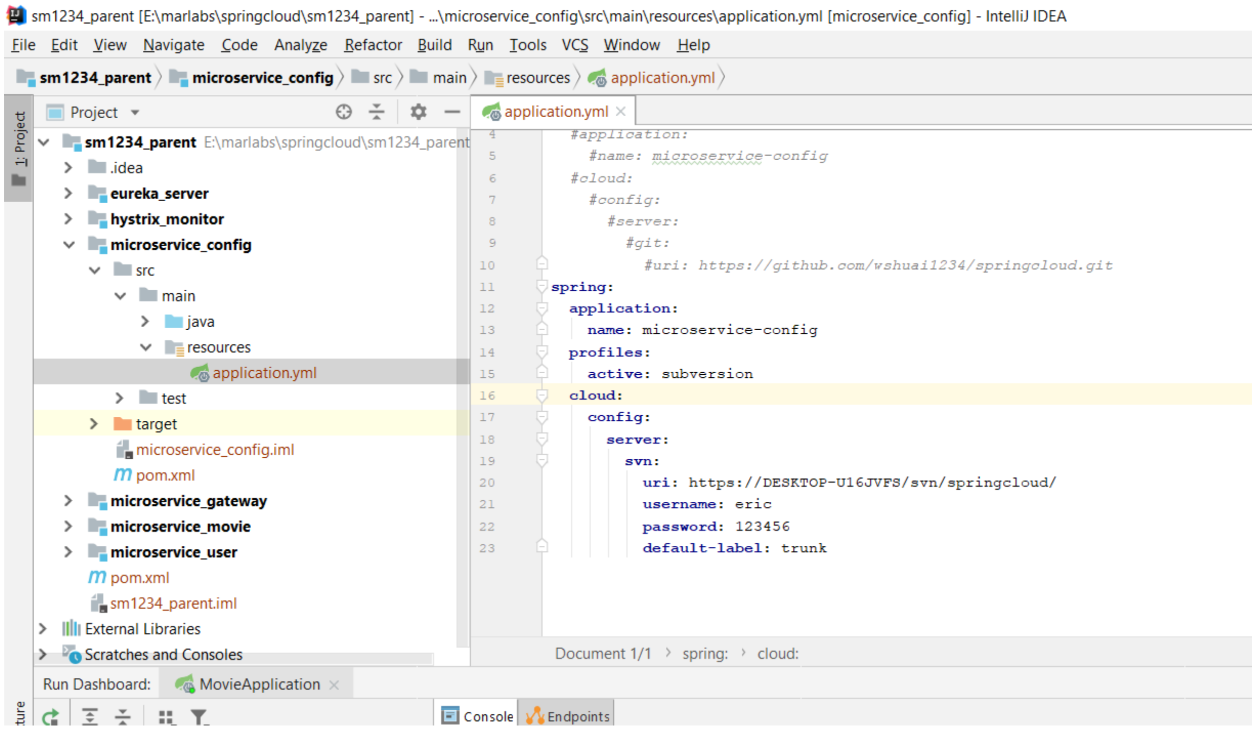
Repository name: springcloud

Repository URL: https://DESKTOP-U16JVFS/svn/springcloud

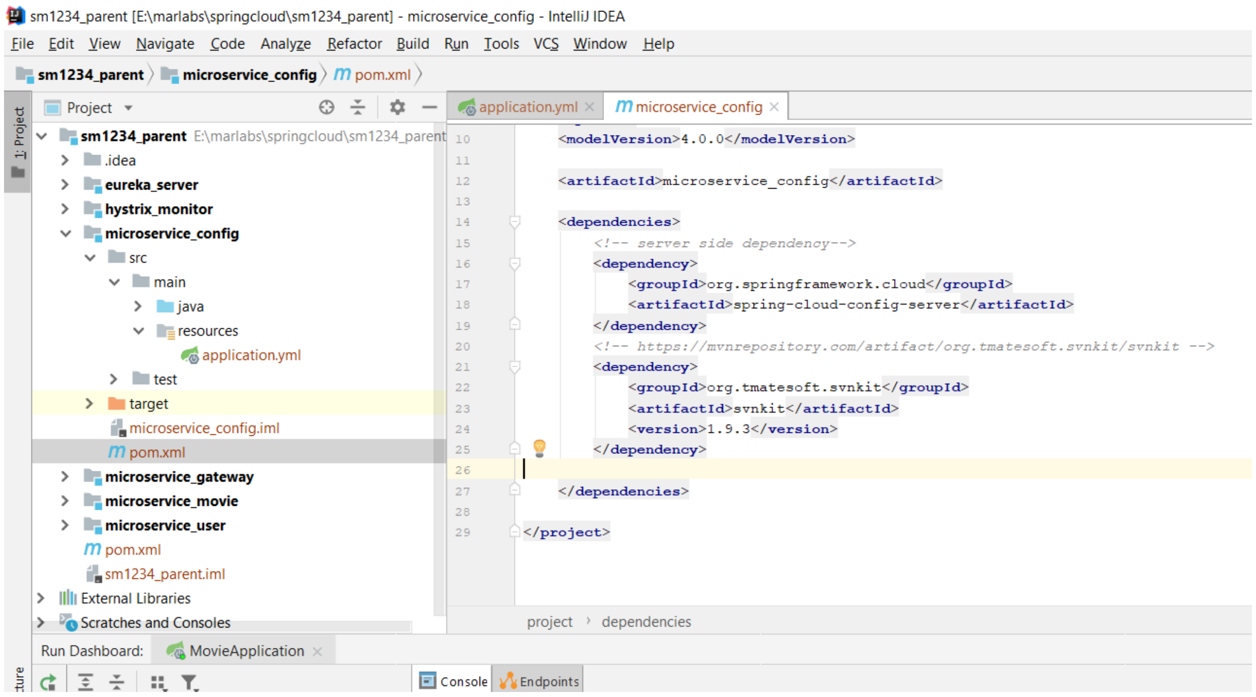
Configured repository permissions:
- eric (Read / Write)


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_user\src\main\resources\bootstrap.yml [microservice_user] - IntelliJ IDEA

```yaml
spring:
  cloud:
    config:
      uri: http://127.0.0.1:12000 #需要连接的spring cloud coinfig server
      name: user  # prefix
      profile: dev  #suffix
      label: trunk #master
```


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_config\src\main\resources\application.yml [microservice_config] - IntelliJ IDEA

```yaml
  #application:
    #name: microservice-config
  #cloud:
    #config:
      #server:
        #git:
          #uri: https://github.com/wshuai1234/springcloud.git
spring:
  application:
    name: microservice-config
  profiles:
    active: subversion
  cloud:
    config:
      server:
        svn:
          uri: https://DESKTOP-U16JVFS/svn/springcloud/
          username: eric
          password: 123456
          default-label: trunk
```



```xml
    <modelVersion>4.0.0</modelVersion>

    <artifactId>microservice_config</artifactId>

    <dependencies>
        <!-- server side dependency-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-config-server</artifactId>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.tmatesoft.svnkit/svnkit -->
        <dependency>
            <groupId>org.tmatesoft.svnkit</groupId>
            <artifactId>svnkit</artifactId>
            <version>1.9.3</version>
        </dependency>

    </dependencies>

</project>
```

搭建高可用配置中心

停止 config 服务无法启动用户和电影微服务 怎么办

注册到 eureka

启动多个 config 服务，端口 12000,12001,12002.。。

Spring cloud bus 消息总线

对配置实时更新的增强

当前每个微服务修改配置文件之后都要重启微服务，微服务很多时如何实时更新?

RabbitMQ 消息队列支撑

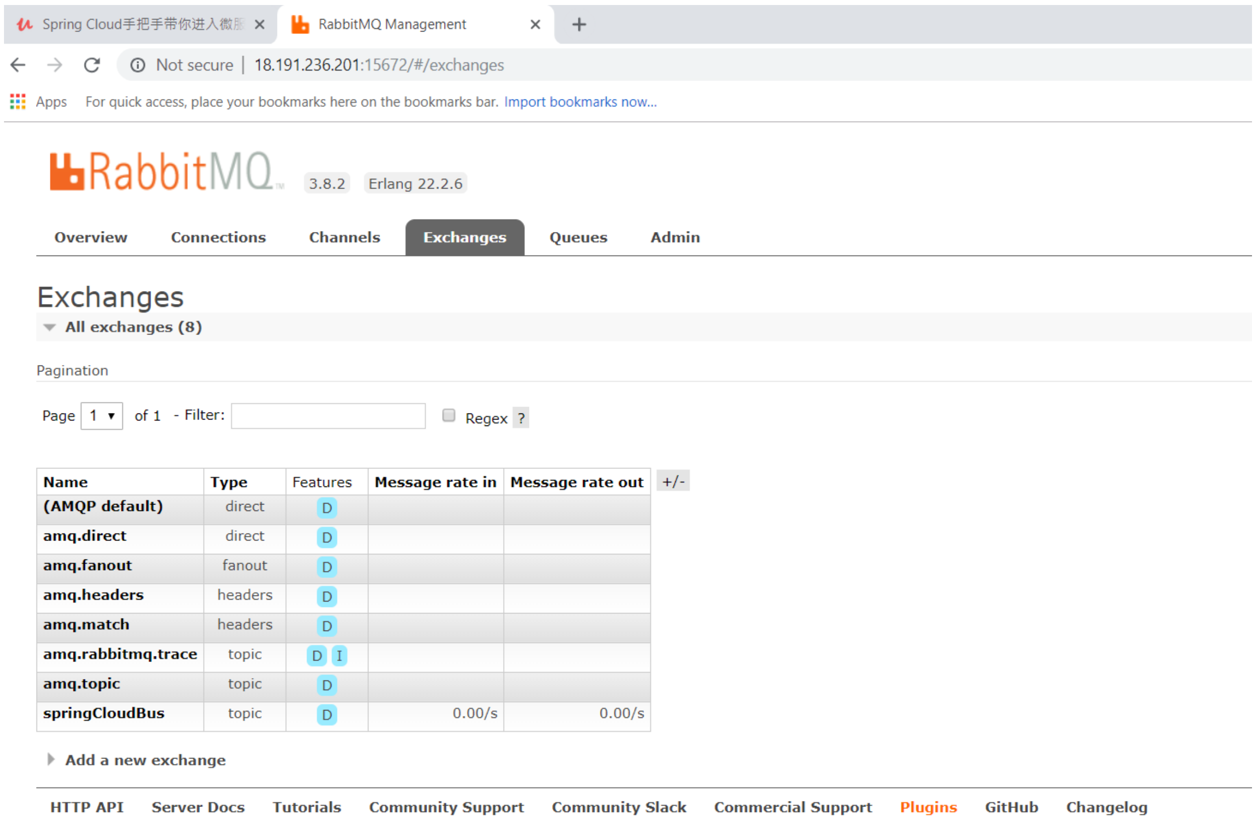
docker pull rabbitmq:management

docker run -di --name=rabbitmq -p 5671:5671 -p 5672:5672 -p 15671:15671 -p 15672:15672 -p 4369:4369 -p 25672:25672 rabbitmq:management

访问

http://18.191.236.201:15672/ 得到 rabbitmq 界面

默认用户名密码都是 guest


Spring Cloud手把手带你进入微服
RabbitMQ Management
Not secure | 18.191.236.201:15672/#/exchanges
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
RabbitMQ 3.8.2 Erlang 22.2.6
Overview Connections Channels Exchanges Queues Admin

Exchanges

All exchanges (8)

Pagination

Page 1 of 1 - Filter: Regex ?

Name	Type	Features	Message rate in	Message rate out
(AMQP default)	direct	D		
amq.direct	direct	D		
amq.fanout	fanout	D		
amq.headers	headers	D		
amq.match	headers	D		
amq.rabbitmq.trace	topic	D I		
amq.topic	topic	D		
springCloudBus	topic	D	0.00/s	0.00/s

Add a new exchange

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

搭建 spring cloudBus 架构

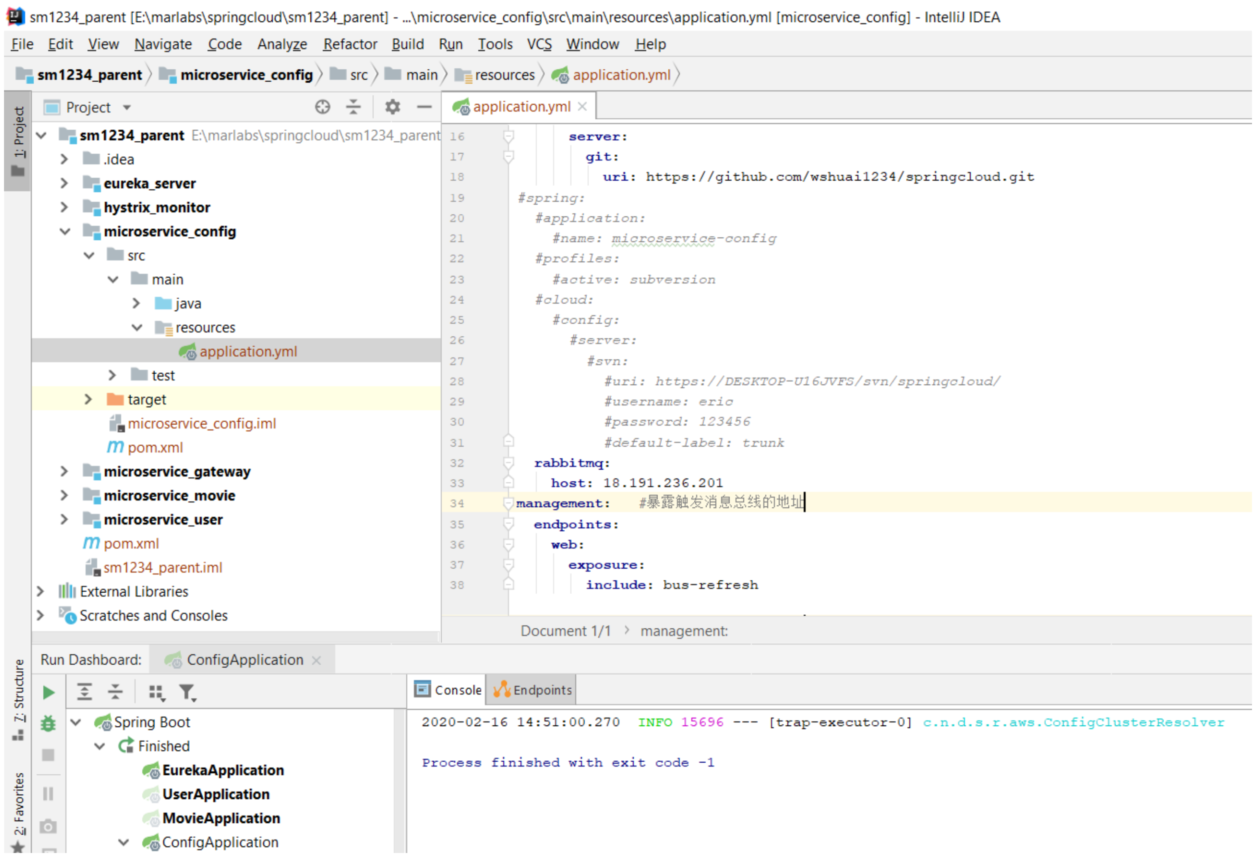
服务端：spring cloud config 里

客户端：消费方和提供方业务服务里面

服务端 config 服务导入 spring cloud bus 依赖

```xml
<!-- spring cloud bus dependency-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-bus</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
</dependency>
```

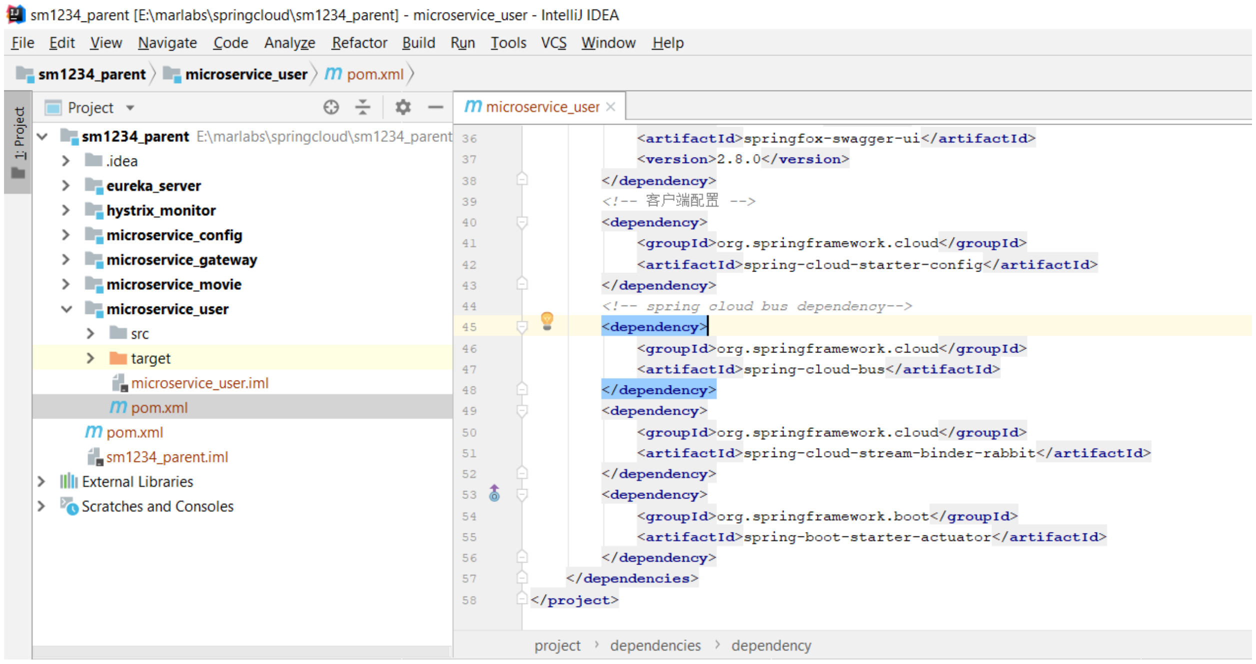
暴露消息总线地址和 URL:


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_config\src\main\resources\application.yml [microservice_config] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sm1234_parent microservice_config src main resources application.yml
Project
sm1234_parent E:\marlabs\springcloud\sm1234_parent
.idea
eureka_server
hystrix_monitor
microservice_config
src
main
java
resources
application.yml
test
target
microservice_config.iml
pom.xml
microservice_gateway
microservice_movie
microservice_user
pom.xml
sm1234_parent.iml
External Libraries
Scratches and Consoles
Document 1/1 › management:
Run Dashboard: ConfigApplication
Spring Boot
Finished
EurekaApplication
UserApplication
MovieApplication
ConfigApplication
Console Endpoints

```yaml
      server:
        git:
          uri: https://github.com/wshuai1234/springcloud.git
#spring:
  #application:
    #name: microservice-config
  #profiles:
    #active: subversion
  #cloud:
    #config:
      #server:
        #svn:
          #uri: https://DESKTOP-U16JVFS/svn/springcloud/
          #username: eric
          #password: 123456
          #default-label: trunk
  rabbitmq:
    host: 18.191.236.201
management:    #暴露触发消息总线的地址
  endpoints:
    web:
      exposure:
        include: bus-refresh
```

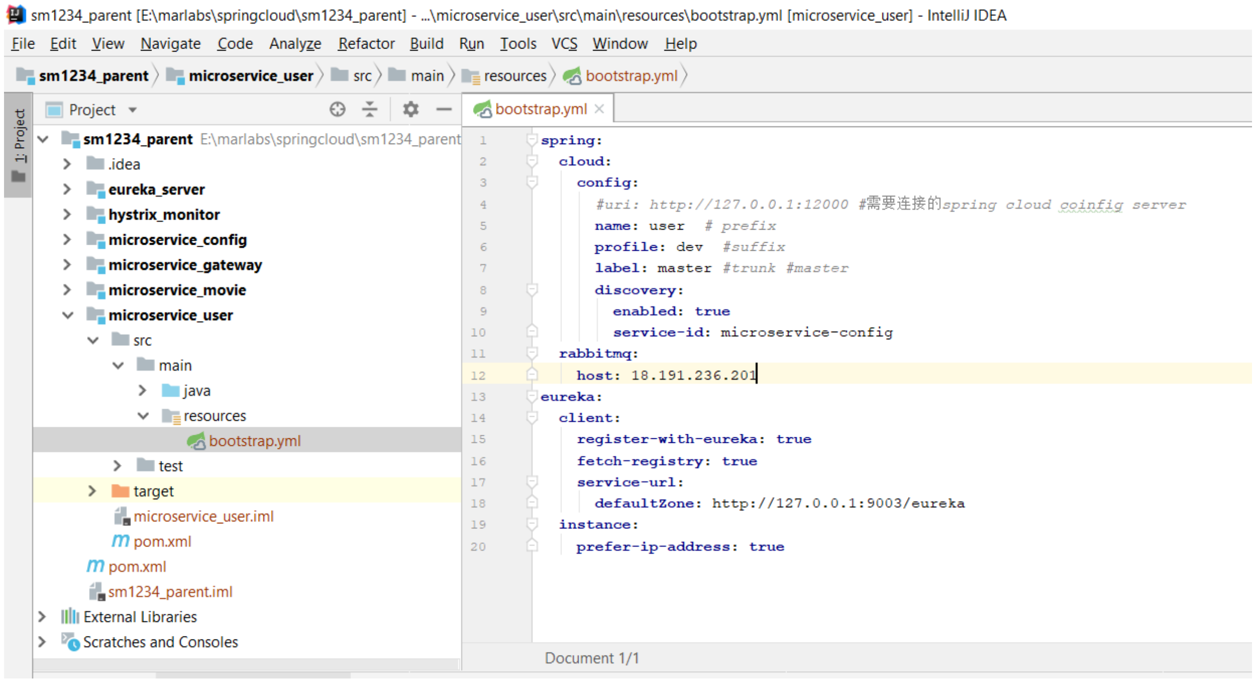
```
2020-02-16 14:51:00.270  INFO 15696 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver

Process finished with exit code -1
```

客户端导入依赖


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - microservice_user - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sm1234_parent 〉 microservice_user 〉 pom.xml
Project
sm1234_parent E:\marlabs\springcloud\sm1234_parent
.idea
eureka_server
hystrix_monitor
microservice_config
microservice_gateway
microservice_movie
microservice_user
src
target
microservice_user.iml
pom.xml
pom.xml
sm1234_parent.iml
External Libraries
Scratches and Consoles
microservice_user
project › dependencies › dependency

```xml
                <artifactId>springfox-swagger-ui</artifactId>
                <version>2.8.0</version>
            </dependency>
            <!-- 客户端配置 -->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-config</artifactId>
            </dependency>
            <!-- spring cloud bus dependency-->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-bus</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-actuator</artifactId>
            </dependency>
        </dependencies>
    </project>
```

客户端服务连接 rabbitmq


sm1234_parent [E:\marlabs\springcloud\sm1234_parent] - ...\microservice_user\src\main\resources\bootstrap.yml [microservice_user] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
sm1234_parent 〉 microservice_user 〉 src 〉 main 〉 resources 〉 bootstrap.yml
Project
sm1234_parent E:\marlabs\springcloud\sm1234_parent
.idea
eureka_server
hystrix_monitor
microservice_config
microservice_gateway
microservice_movie
microservice_user
src
main
java
resources
bootstrap.yml
test
target
microservice_user.iml
pom.xml
pom.xml
sm1234_parent.iml
External Libraries
Scratches and Consoles
bootstrap.yml
Document 1/1

```yaml
spring:
  cloud:
    config:
      #uri: http://127.0.0.1:12000 #需要连接的spring cloud coinfig server
      name: user  # prefix
      profile: dev  #suffix
      label: master #trunk #master
      discovery:
        enabled: true
        service-id: microservice-config
  rabbitmq:
    host: 18.191.236.201
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://127.0.0.1:9003/eureka
  instance:
    prefer-ip-address: true
```

刷新 springcloudbus

修改配置文件数据库名称 或是端口之后，

Post 请求，12000 是 config 微服务的端口号

http://localhost:12000/actuator/bus-refresh

再次访问微服务就会发现已经自动更新，而不需要重启微服务

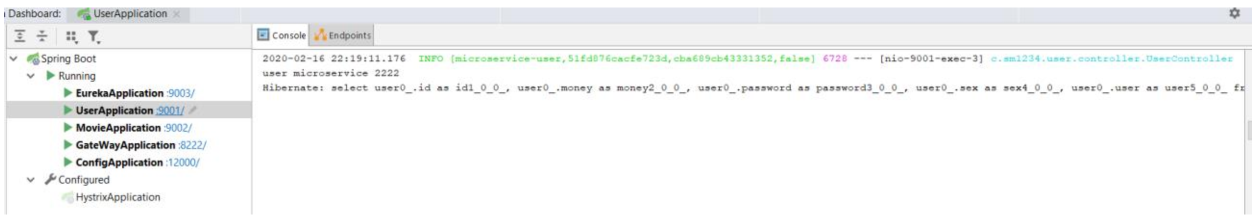
Spring cloud 分布式链路跟踪

微服务架构下，一个请求可能会经过多个服务才会得到结果，如果过程出现了异常，很难定位问题。必须实现分布式链路跟踪功能，直观显示完整的调用过程

使用 spring cloud sleuth 日志跟踪组件追踪链路

导入 sleuth 依赖，向所有链路中经过的服务加入此依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```



```
2020-02-16 22:19:11.176  INFO [microservice-user,51fd876cacfe723d,cba689cb43331352,false] 6728 --- [nio-9001-exec-3] c.sm1234.user.controller.UserController
user microservice 2222
Hibernate: select user0_.id as id1_0_0_, user0_.money as money2_0_0_, user0_.password as password3_0_0_, user0_.sex as sex4_0_0_, user0_.user as user5_0_0_ fr
```

51fd876cacfe723d 最后四位是 trace ID

Sleuth 结合 zipkin 使日志跟踪更方便，zipkin 是日志跟踪可视化界面

Zipkin 用来查看请求链路跟踪