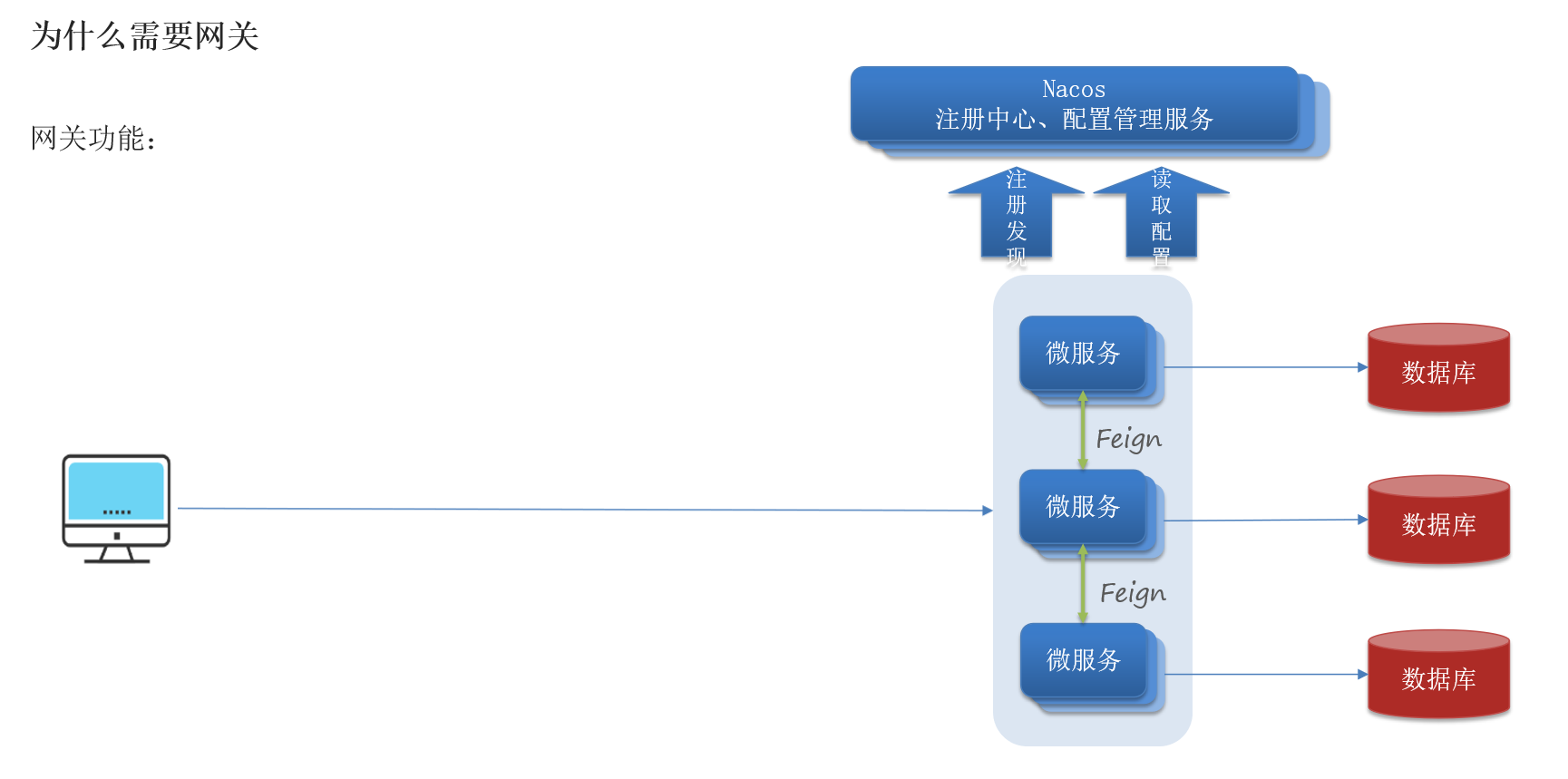
# SpringCloudAlibaba-Gateway

## 统一网关Gateway：

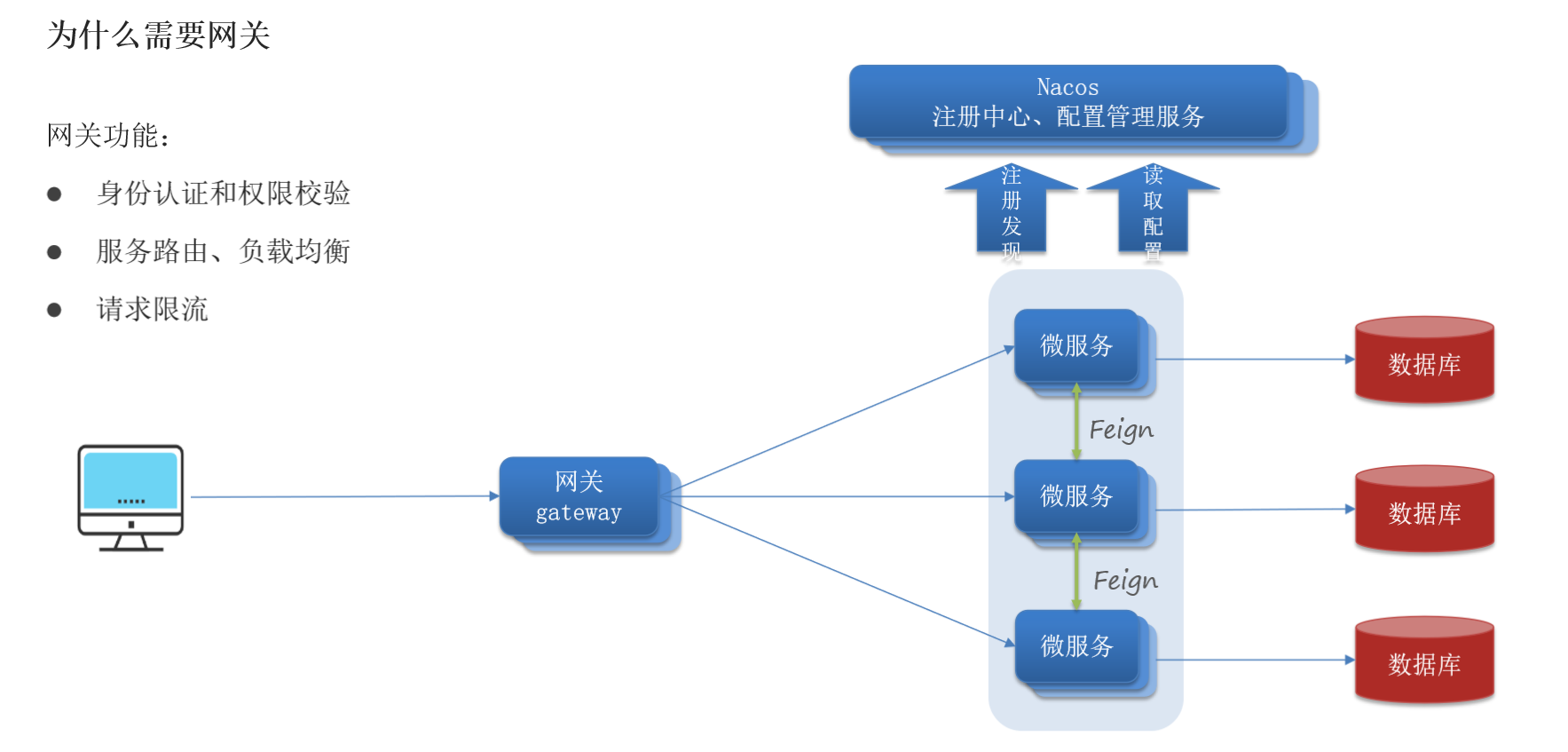
### 什么是网关？为什么需要网关？

先来看一下现在的微服务结构，我们有很多个不同的服务，每个服务都需要去访问数据库完成自己的业务，并且微服务都可以到Nacos里完成服务的注册、配置的管理，这个时候，当我们的微服务内部有相互掉欧阳那个关系时，就可以用Feign组件来完成，而当有外部用户访问服务的时候，显示是让外部请求直接发请求到微服务！



同学们，这样是有问题的，现在的问服务直接暴露摆在那里，允许任何用户发请求来访问，是不是有点不太安全啊！要知道不是所有的业务/服务都是对外公开的！有很多的业务属于公司内部的工作人员、管理人员才可以访问的，结果现在微服务都暴露放在那里，赤裸裸地放在那里，允许任何人来参观、任何人来访问，是不对的！

所以我们应该对访问用户的身份进行验证！如果是工作人员、或者是内部人员，才允许你进去看一看敏感业务，否则拦住不许进去。做这种事的就是统一网关组件Gateway！**一切请求必须先到网关再到微服务！**



网关的作用：

1：身份认证和权限校验，请求来了先得经过网关判断一下，你是谁，一但身份认证通过，网关就会放行、转发、路由请求到对应的微服务去；

2：服务路由、负载均衡，比如请求是做用户查询功能，网是不能处理用户请求的业务，肯定要把请求转发到对应的处理用户查询的服务：user-service，因此网关必须做一件事：得根据你的请求，判断将来给你扔到user-service还是扔到order-service，这个动作弄错了就出问题了，这个动作就是服务的路由；并且同样是user-service可能有多个实例，那请求往后走的过程中还需要做一个负载均衡，错多个实例中挑一个

2：请求限流，什么是限流，比方说微服务能够允许的用户请求量就是500，结果现在来了2000，网关如果都放行，服务就崩了，网关必须要起到限流，超过500，请求就不被放行，要么拒绝、要么等待。其实就是对微服务的一种保护！

以上说明网关的作用很重要，其实就是对微服务的保护

小总结

网关的作用：

1：对用户请求做身份认证、权限校验；

2：将用户请求路由到微服务，并实现负载均衡；

3：对用户请求做限流

### Gateway快速入门

SpringCloud中网关的实有两种：

1：gateway

2：zuul

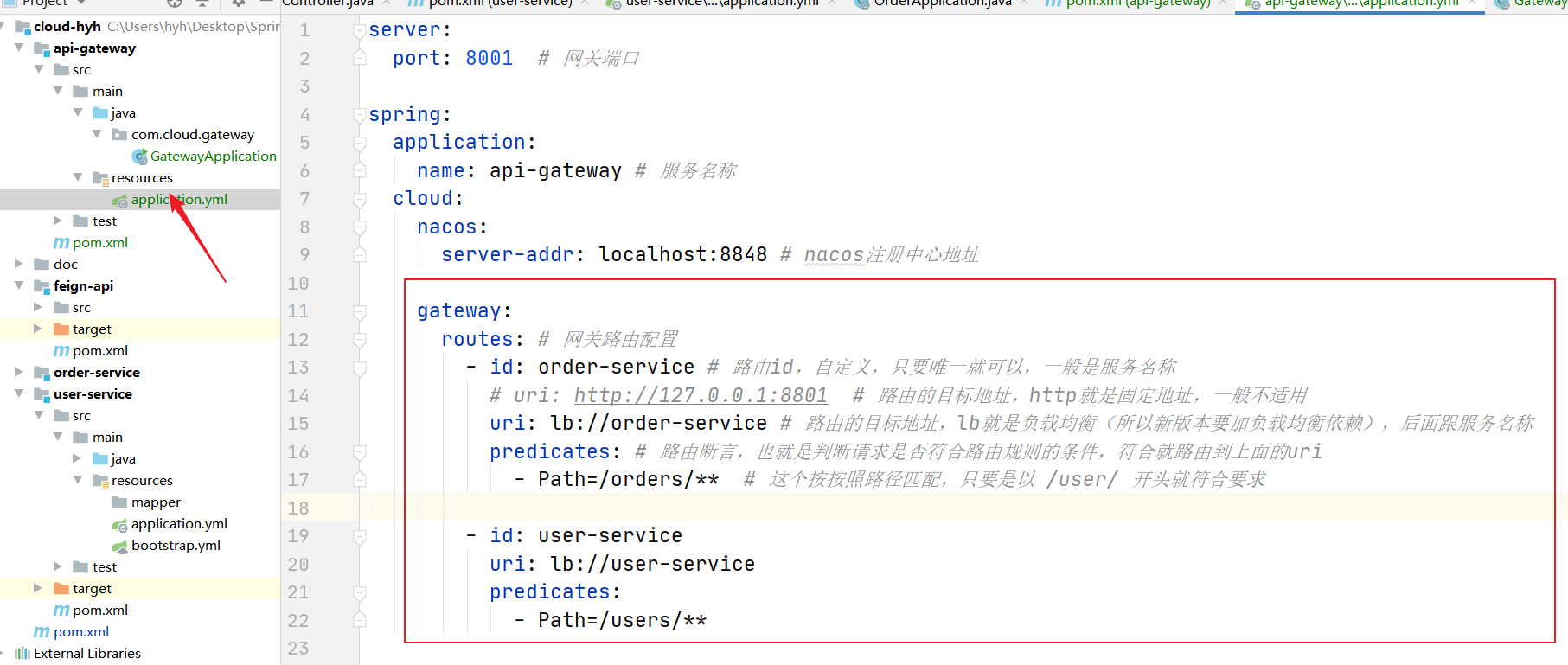
Zuul是基于Servlet的实现，属于阻塞式编程，而gateway则是基于Sprign5中提供的WebFlux，属于响应式编程的实现，具备更好的性能!

#### 搭建网关服务

1：创建新的module，引入SpringCloudGateway的依赖和Nacos的服务发现依赖



2：编写路由配置和Nacos地址



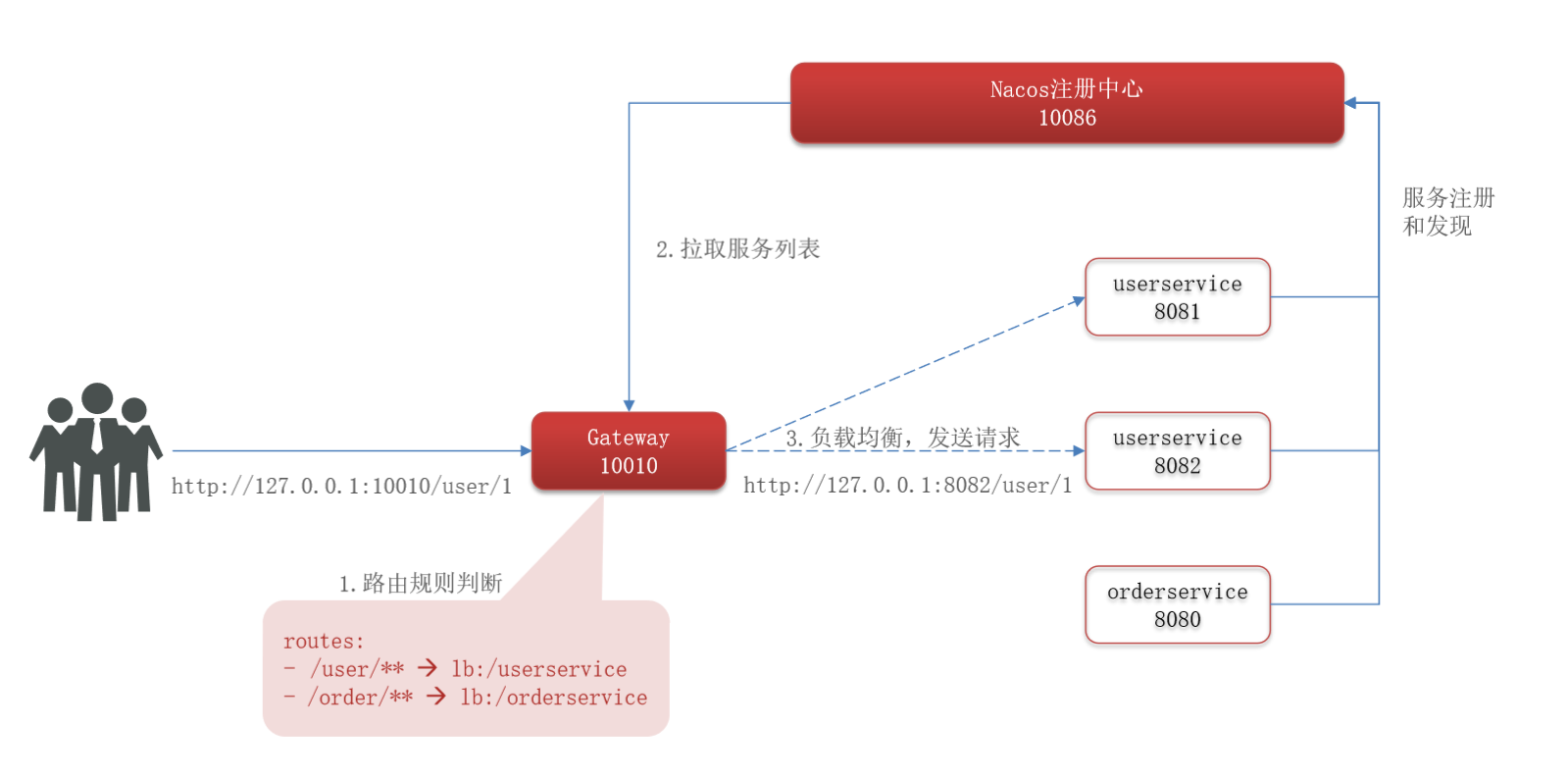
3：启动服务，查看效果，访问网关，成功查到order信息、user信息，我们的网关是没有做任何业务逻辑的，但是却能查到order信息、user信息，其实这就是把请求从网关路由到了对应的order-service、user-service：





总结：请求的时候发生了什么？请看流程图：

我们有两个User-service实例、一个order-service实例、一个Nacos注册中心、一个gateway网关服务，我们的微服务都会注册到Nacos注册中心，当用户发送了用户请求：http://localhost:8001/users/1，因为访问的8001端口，所以网关服务api-gateway的端口就是8001，所以请求先到达网关服务，而网关又无法处理这个查询用户请求的业务，所以只能基于路由规则去判断，我们定义了两个路由规则：一个是把users请求开头的路径代理到user-service,一个是把orders请求开头的路径代理到order-service，那么此时的请求符合第一条路由规则，所以gateway就应该把请求代理到user-service，那么gateway就会拿着user-service服务名称去注册中心拉取对应的服务列表，然后负载均衡挑选出一个user-service实例，比如8201实例，那么请求就会放行、路由、发送到8201的user-sevice实例，整个流程就完成了：



#### 小总结

Gateway网关路由配置:

1：路由id，自定义，唯一即可，一般是服务名称

2：路由的目标地址uri，http代表固定地址，lb就是负载均衡（就是网关做请求转发路由的时候要做负载均衡），后面跟上服务名称，因此gateway会拿服务名去注册中心拉取服务列表，而后做负载均衡，实现请求路由

3：路由断言predicates（就是判断），也就是判断请求是否符合路由规则的条件

4：路由过滤器filters：对请求或响应做处理

微服务概述（分布式+集群）：

1：用户在浏览器或者手机发起请求，请求先到nginx代理服务器，ngixn做负载均衡后选出一个gateway网关服务实例；

2：gateway网关对进来的请求做路由转发，先根据服务名从注册中心拉取服务列表，然后负载均衡挑选出要转发的服务，从而完成请求路由转发；

3：服务之间的调用，服务消费者根据服务命从注册中心拉取服务列表，根据配置或者默认的负载均衡策略选出服务提供者实例，使用Feign完成服务调用。

**你看！你看！！你看！！！以上就有三次负载均衡了！！！！**

### 路由断言工厂Route Predicate Factory

路由断言其实就是路由规则！

我们知道，在网关路由中可以配置的内容主要包括：

1：路由id：路由唯一标识；

2：uri：路由目的地，支持lb和http两种；

3：predicates：路由断言，判断请求是否符合要求，符合则转发到路由目的地；

4：filters：路由过滤器，处理请求或响应

这节课重点学习断言predicates以及断言工厂Predicate Factory

我们在配置文件中写的断言规则只是字符串，这些字符串会被断言工厂Predicate Factory读取并处理，转变为路由判断的条件。

所以断言工厂Predicate Factory的作用就是：读取我们用户定义的断言规则predicates，然后解析出来，并且对请求做出判断，比方说我们写的：

Path=/users/\*\*是按照路径匹配，我们只是写了这样的字符串，这个规则是由下面这个断言工厂类来处理：

**org.springframework.cloud.gateway.handler.predicate.PathRoutePredicateFactory**

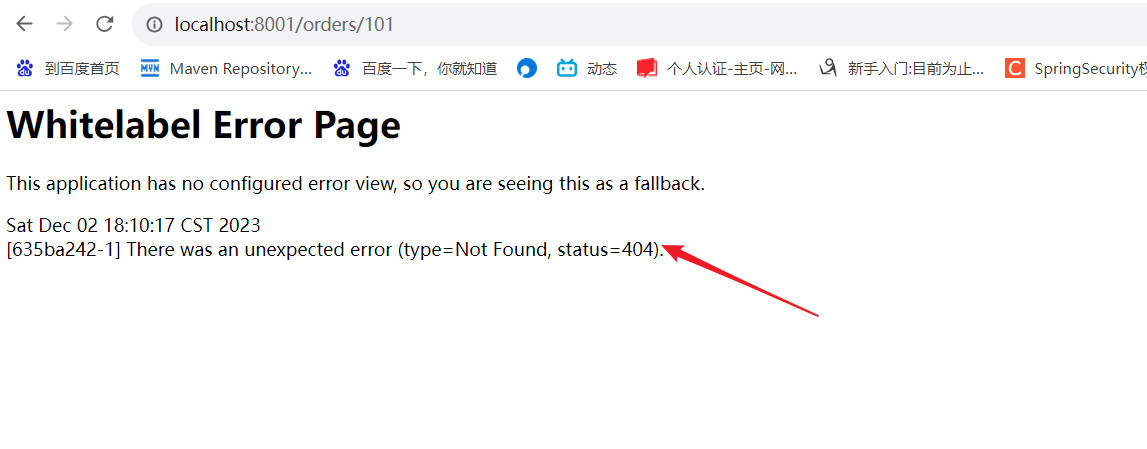
常见的断言工厂：



示例：After和Before



重启服务，只要Before的时间改成不符合，就404了



小总结：

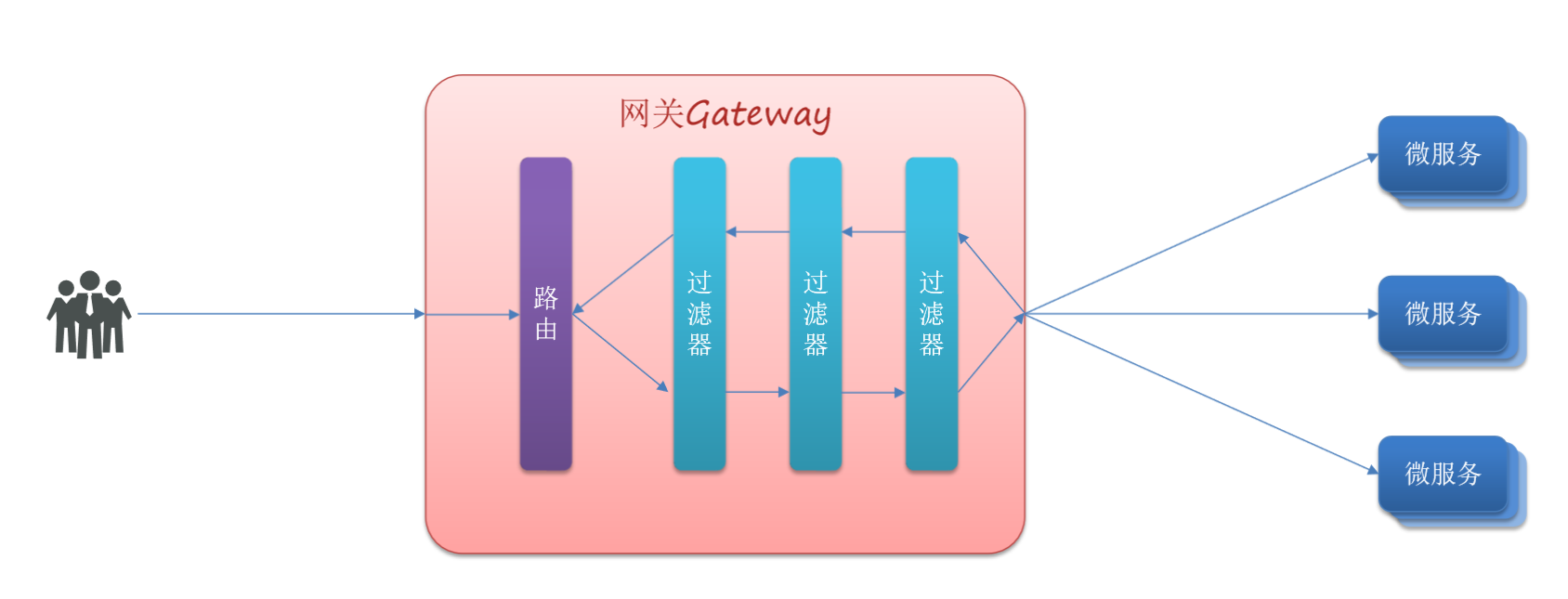
PredicateFactory的作用是什么？

读取用户配置的断言规则，而后解析成对应的判断条件，将来用户请求进来了做判断

### 路由过滤器GatewayFilter

GatewayFilter是网关提供的一种过滤器，可以对**进入网关的请求和微服务返回的响应做处理**

对请求前做前置处理，对响应后做后置处理，请看流程图：



我们已经有很多的微服务，用户想要访问微服务，不能直接访问，必须经过网关，问题来了，当用户向网关发起请求时到底经历了什么呢？我们已经知道的是，请求进入网关一定要做路由，我们会有一个断言工厂PredicateFactory，可以对基于我们配置的断言规则完成请求路由，判断一下用户请求到底要到那个微服务，但是路由之后是不是立即就可以向微服务发出请求了呢？不是这样的！

因为在网管里面，我们还可以给我们的路由请求配置各种各样的过滤器，这些过滤器形成一个过滤器链，你的路由请求一定要经过这些过滤器链，然后才能到达微服务！那么在这个过程中，我们的过滤器就可以干对进入网关的请求做各种处理：比如对请求头做什么处理、请问求参数做什么处理。。。。。。

当请求放行、路由到微服务之后，微服务处理完要返回用户一个结果，响应结果也是先到达网关，响应同样会经过过滤器链来逐层处理响应结果，最终才会返回给用户，那么在这个过程钟，过滤器可以对响应做什么事呢?比如：对响应头加点什么、响应体加点什么。。。。

这就是网关过滤器的作用：对请求或响应做各种各样的处理

那具体可以做什么样的处理呢？那就要看Spring提供了什么过滤器工厂了GatewayFilterFactory

Spring提供了三十多种不同的路由过滤器工厂：



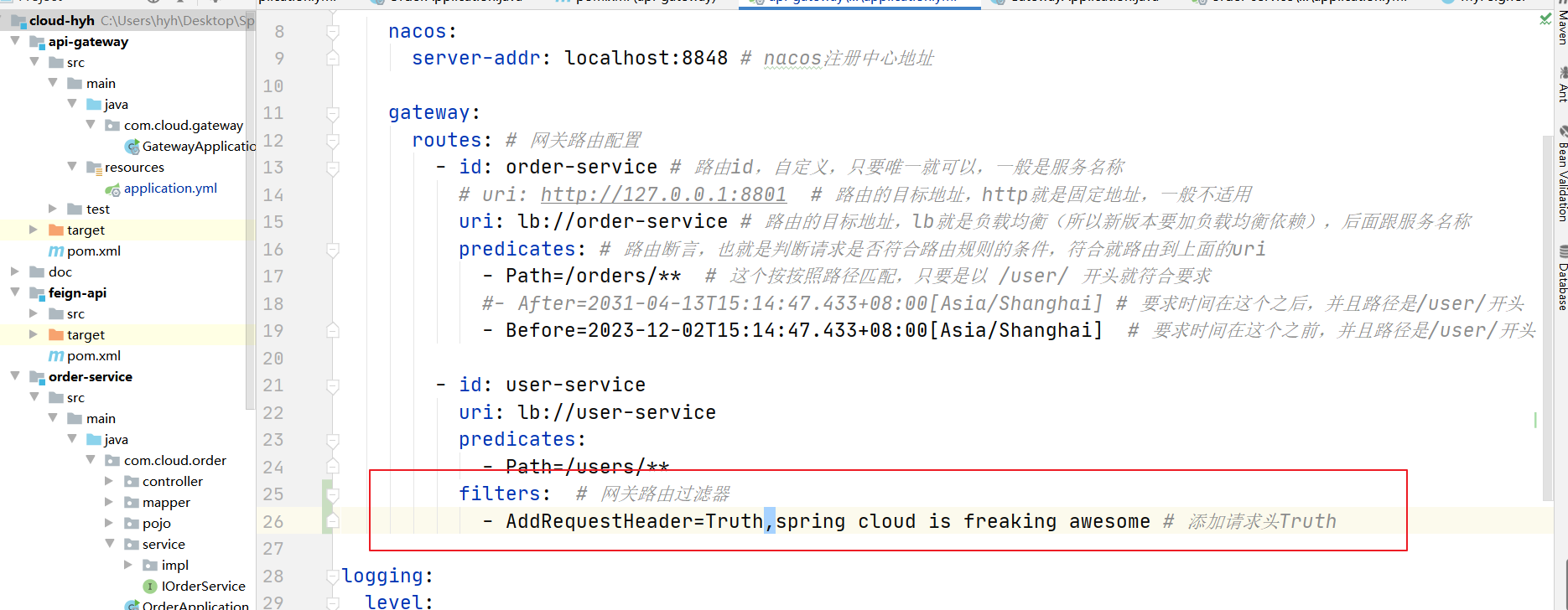
#### 路由过滤器

演示案例一：给所有进入user-service的请求添加一个请求头：

Truth=spring cloud is freaking awesome

实现方式：

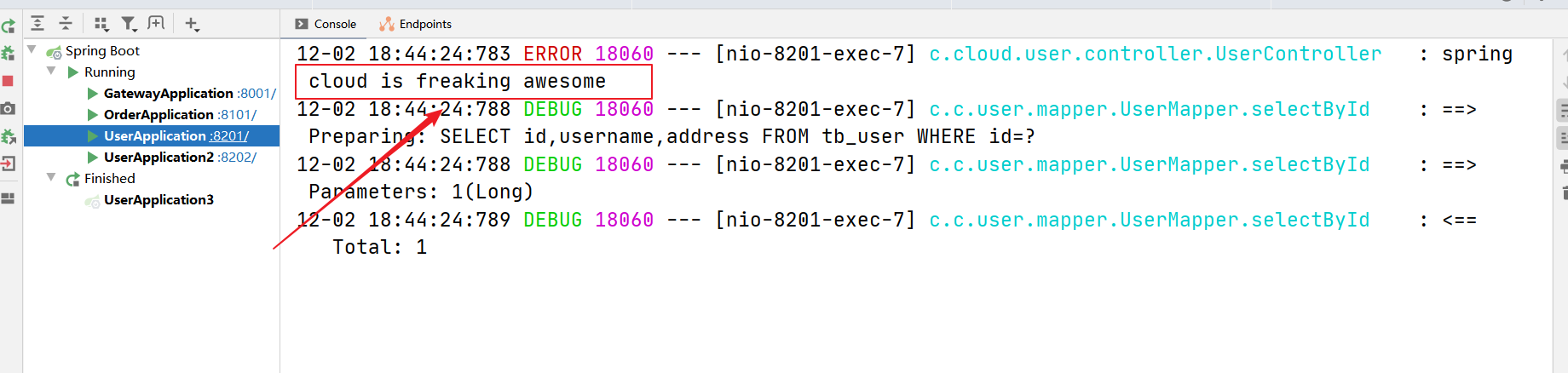
1：在gateway种修改application.yml文件。给user-service的路由添加过滤器：



2：在user-service验证：



3：重启api-gateway和user-service：



#### 默认过滤器

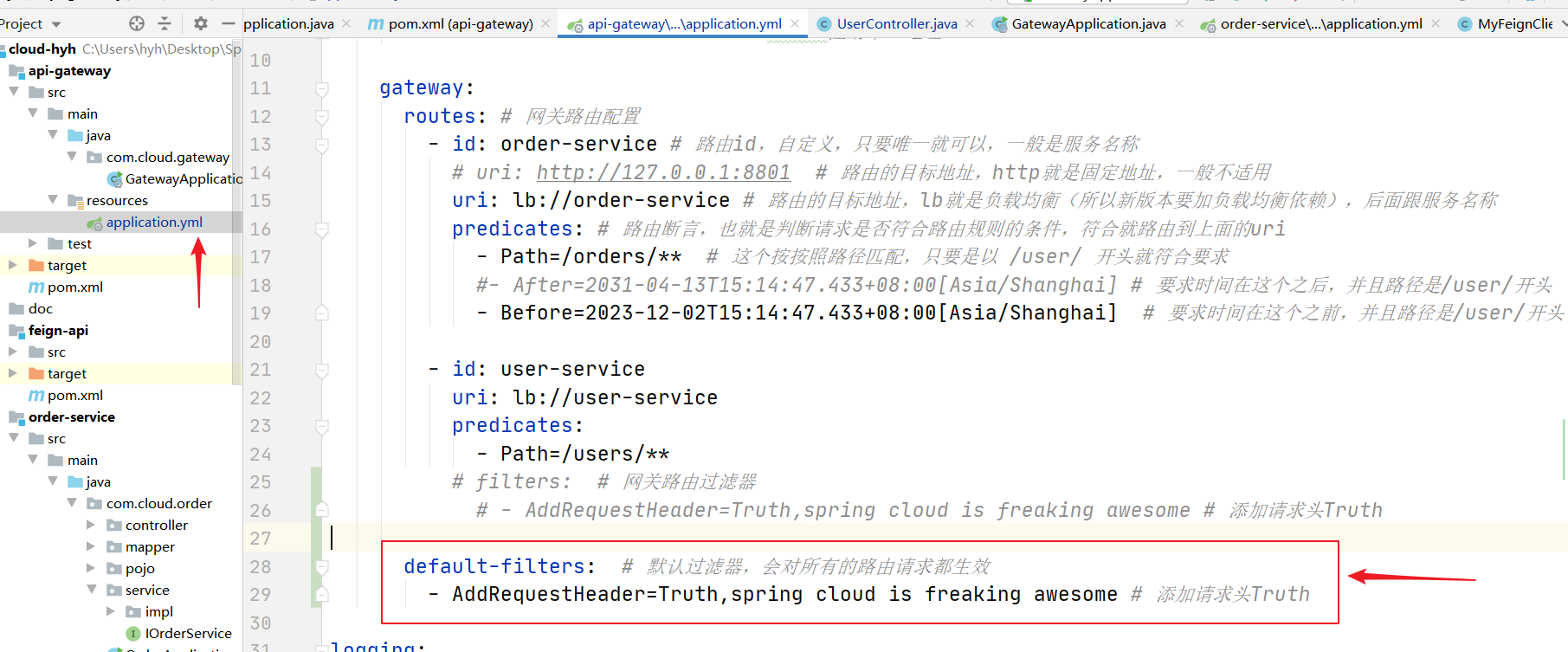
默认过滤器：如果要对所有的路由都生效，则可以将过滤器工厂写到default下，即过滤器只配置某个路由下，只对针对这个路由生效，配在default就针对所有请求都生效。

演示案例二：给所有进入微服务的请求添加一个请求头：

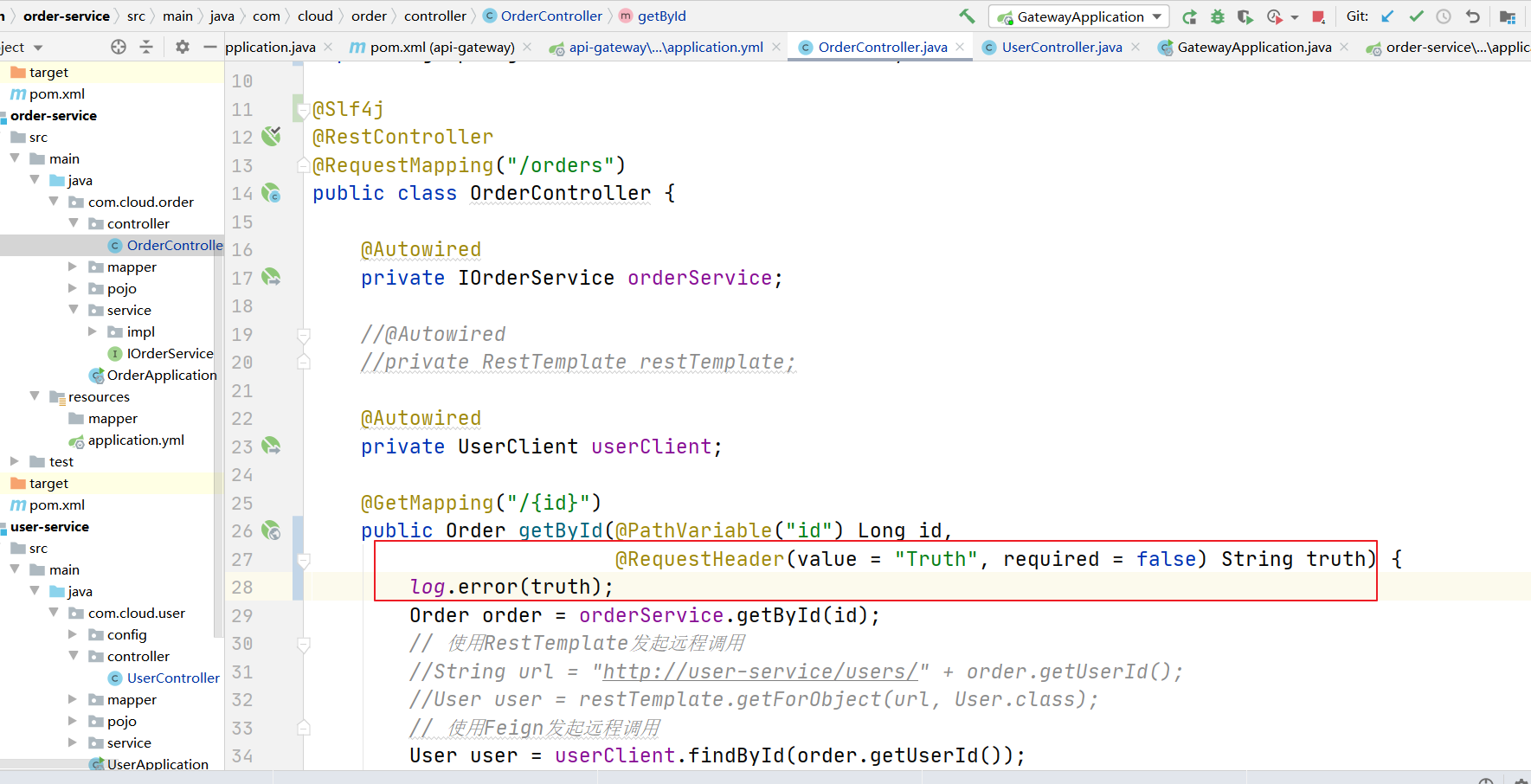
Truth=spring cloud is freaking awesome

实现方式：

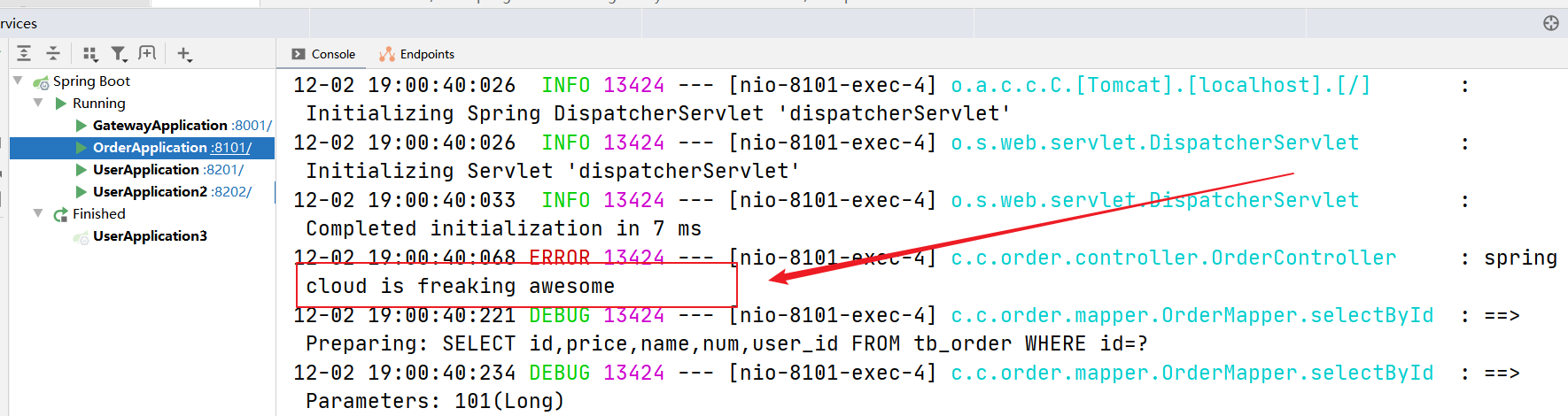
1：在gateway种修改application.yml文件，配置和routes同级的默认过滤器default-filters：



2：在order-service验证:



3：重启api-gateway和order-service：



小总结：

过滤器的作用是什么：

1；对路由的而请求或响应做加工处理，比如添加请求头；

2：配置在路由下的过滤器只对当前路由器的请求生效

DefaultFilters的作用是什么：

1：针对所有的路由都生效的过滤器

#### 全局过滤器GlobalFilter

全局过滤器的作用也是处理一切进入网关的请求和微服务响应，与GatewayFilter的默认过滤器DefaultFilters的作用一样！

但之前学的两种GatewayFilter过滤器是通过在配置文件的配置定义，配置的仅仅是参数，处理逻辑是固定的；而GlobalFilter的逻辑需要我们自己写代码实现，比如用户是否具有访问权限这样复杂的逻辑。

步骤：

1：实现GlobalFilter接口，这个接口只有一个方法：filter，故名意思就是过滤，这个fliter方法里有两个参数：

一个是exchange请求上下文，这个上下文就是从你请求到达网关开始一直到结束为止，整个流程当中都可以共享exchange对象，里面可以获取Request、Response等信息，即请求相关的信息，响应相关的信息，甚至可以往exchange里存个东西；

一个是chain，就是过滤器链，这个链条上处理你这个过滤器还有其他过滤器，作用就是放行，意思就是你调用这个过滤器链，让请求往后走，等于你这个过滤器里的逻辑就处理完了，交给别人处理了。

因此这两个参数的作用是：第一个参数的作用是让我们来编写整个过滤器的业务逻辑的，你需要的信息在参数里都有；第二个参数是让我们放行请求来用的，把请求交给下一个过滤器去处理，处理完之后有个返回值Mono，是WebFlux里的api。

案例：定义全局过滤器，拦截并判断用户身份

需求：定义全局过滤器，拦截请求，判断请求的参数是否满足下面的条件：

1：参数中是否有authorization；

2：authorization参数值是否为admin

如果满足则放行，否则拦截

**package com.cloud.gateway.filter;**

**import org.springframework.cloud.gateway.filter.GatewayFilterChain;**

**import org.springframework.cloud.gateway.filter.GlobalFilter;**

**import org.springframework.core.Ordered;**

**import org.springframework.http.HttpStatus;**

**import org.springframework.http.server.reactive.ServerHttpRequest;**

**import org.springframework.stereotype.Component;**

**import org.springframework.util.MultiValueMap;**

**import org.springframework.web.server.ServerWebExchange;**

**import reactor.core.publisher.Mono;**

**@Component // 过滤器生效，让AuthorizeFilter作为一个bean注入到Spring容器中**

**// @Order(-1) // 顺序注解，可能有很多过滤器，这里定义过滤器执行顺序，值越小优先级越高**

**public class AuthorizeFilter implements GlobalFilter, Ordered {**

**@Override**

**public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {**

**// 1：获取请求参数**

**ServerHttpRequest request = exchange.getRequest();**

**MultiValueMap<String, String> queryParams = request.getQueryParams();**

**// 2：获取参数中的authorization参数**

**String auth = queryParams.getFirst("authorization");**

**// 3：判断参数值是否等于admin**

**if ("admin".equals(auth)) {**

**// 4：是则放行**

**return chain.filter(exchange);**

**}**

**// 5：否则拦截**

**// 5.1：注意设置状态码用户体验友好一点**

**exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);**

**// 5.2：拦截请求**

**return exchange.getResponse().setComplete();**

**}**

**/\*\***

**\* 定义顺序注解，可能有很多过滤器，这里定义过滤器执行顺序，值越小优先级越高**

**\***

**\* @return**

**\*/**

**@Override**

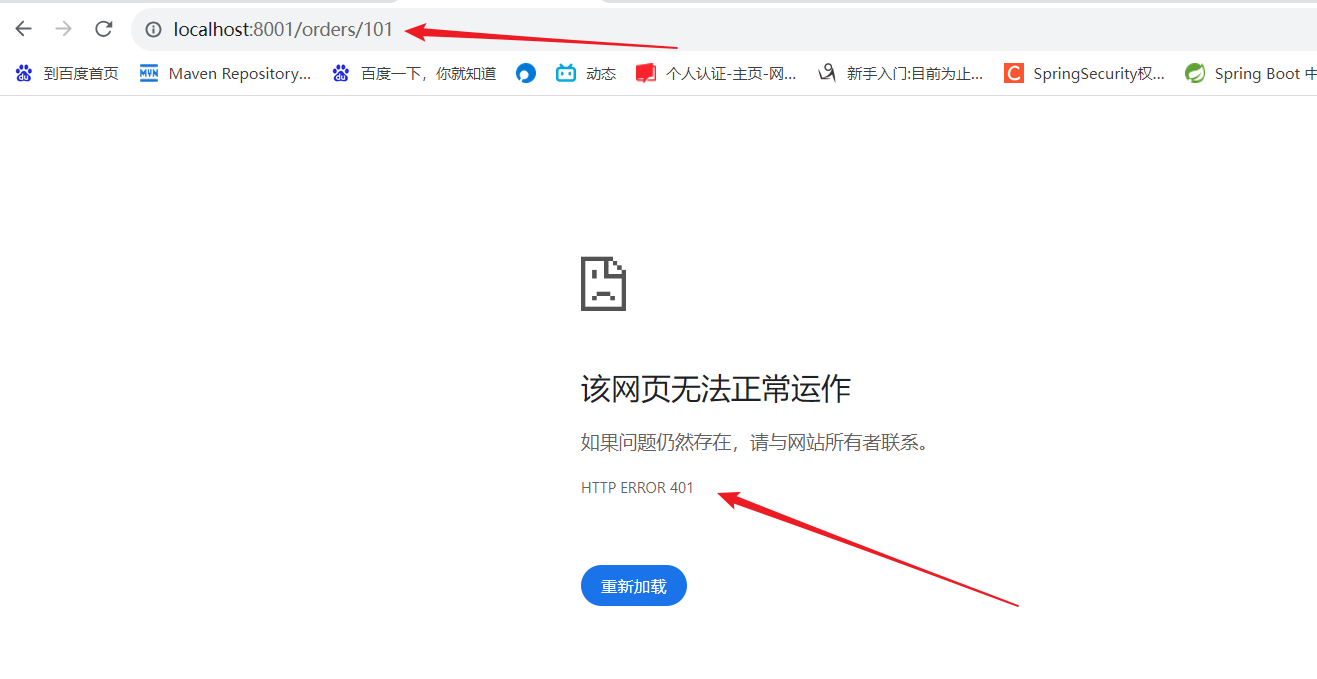
**public int getOrder() {**

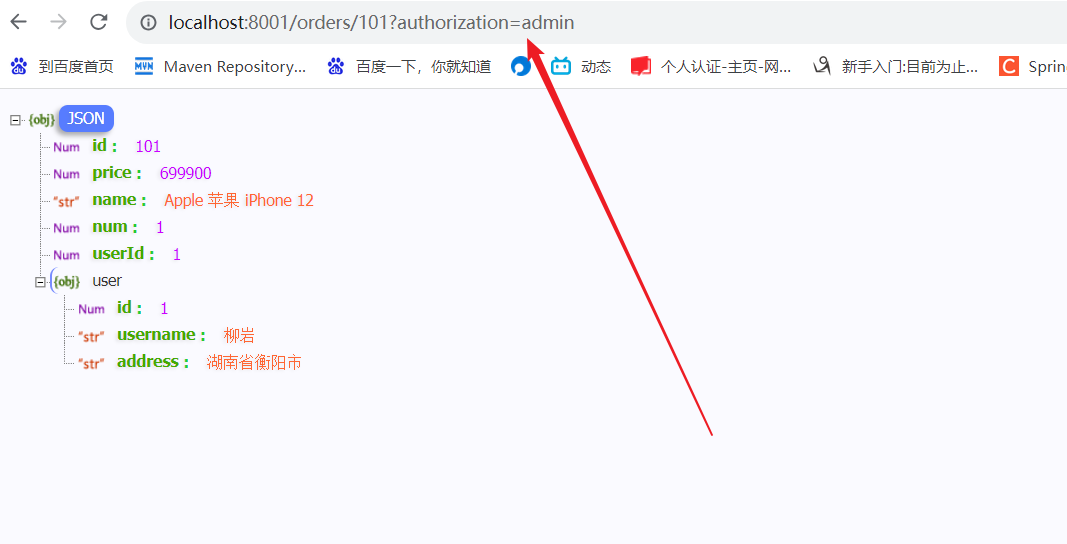
**return -1;**

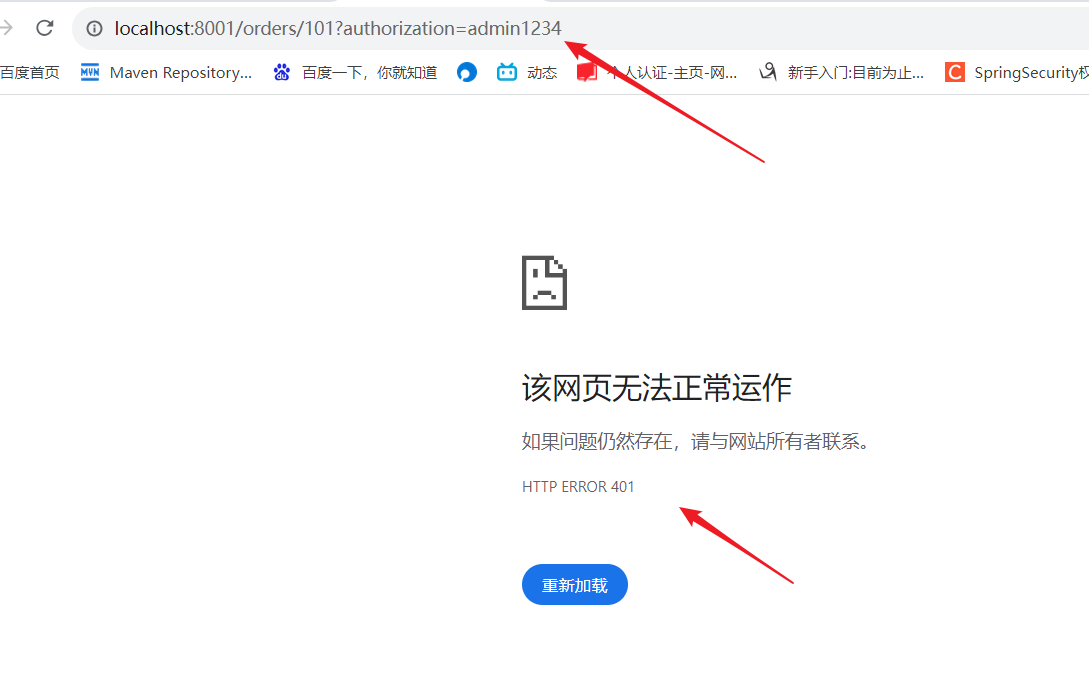
**}**

**}**

重新启动api-gateway，查看效果，可以看到不带authorization参数是会被拦截的，或者参数值不是admin也会被拦截的：







小总结：

全局过滤器的作用是什么？

1：对所有路由都生效的过滤器，并且可以自定义处理逻辑

实现全局过滤器的步骤？

1：实现GlobalFilter接口

2：添加@Order注解或实现Ordered接口

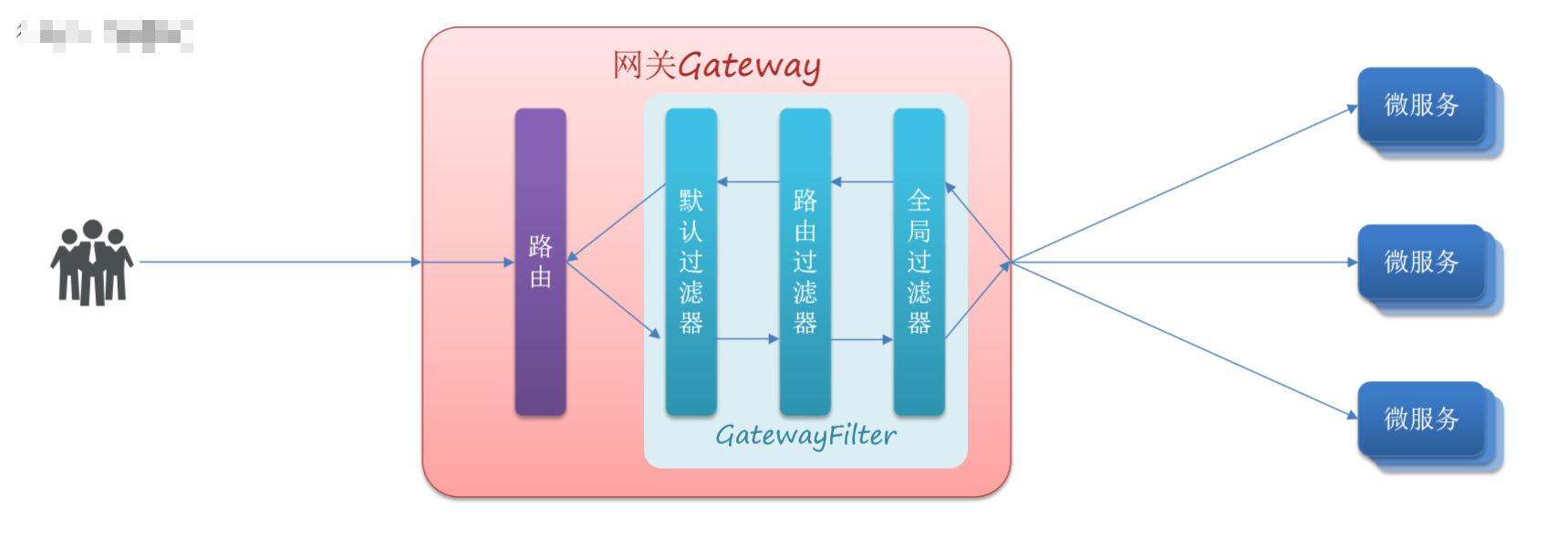
3：编写处理逻辑

#### 过滤器的执行顺序

请求进入网关会碰到三类过滤器：当前路由过滤器、DefaultFilter、GlobalFilter

问：那么这三种过滤器在网关中的执行顺序关系是什么样子的呢？

答：首先当请求路由后，注意了，一定要强调请求路由后，因为当前路由的过滤器只有路由确认之后才知道有哪些过滤器生效，因为不同路由有不同的当前路由过滤器，所以先当请求路由后，网关会将当前路由过滤器再结合DefaultFilter和GlobalFilter，合并到一个过滤器链（集合）中，而后对这些过滤器做排序后依次执行每个过滤器！



问题：你可能会有疑问，当前路由过滤器、DefaultFilter、GlobalFilter这三个过滤器类型都不一样，你怎么能把他们扔到一个集合里，还排序？

傻逼，其实DefaultFilter和路由过滤器这两个是非常接近的，请看配置：



你看你看你看，DefaultFilter和路由过滤器的配置是不是很像！区别就是一个放在路由内，一个放在默认过滤器default-filters里，所以在Java底层来看，DefaultFilter和路由过滤器的本质是一样的，只不过作用范围不一样，比如途中的都是：

AddRequestHeaderGatewayFilterFactory，这个过滤器工厂就会读取配置文件，最后生成一个真正的过滤器GatewayFilter，所以我们任务**路由过滤器和defaultFilter是同一类，都是GatewayFilter**！

问题是全局过滤器GlobalFilter可不是Gateway啊！但是源码里面有一个GatewayFilterAdapter适配器，反正就是GlobalFilter和GatewayFilter存在密切关系就对了！也就是说**在网关当中，所有的GlobalFilter都可以通过GatewayFilterAdapter适配器适配成GatewayFilter**

**所以可以认为，网关中的所有过滤器最终都是GatewayFilter类型！既然是同一种类型，那么**路由过滤器、DefaultFilter、GlobalFilter这三种过滤器自然可以扔到一个集合里去做排序啊！

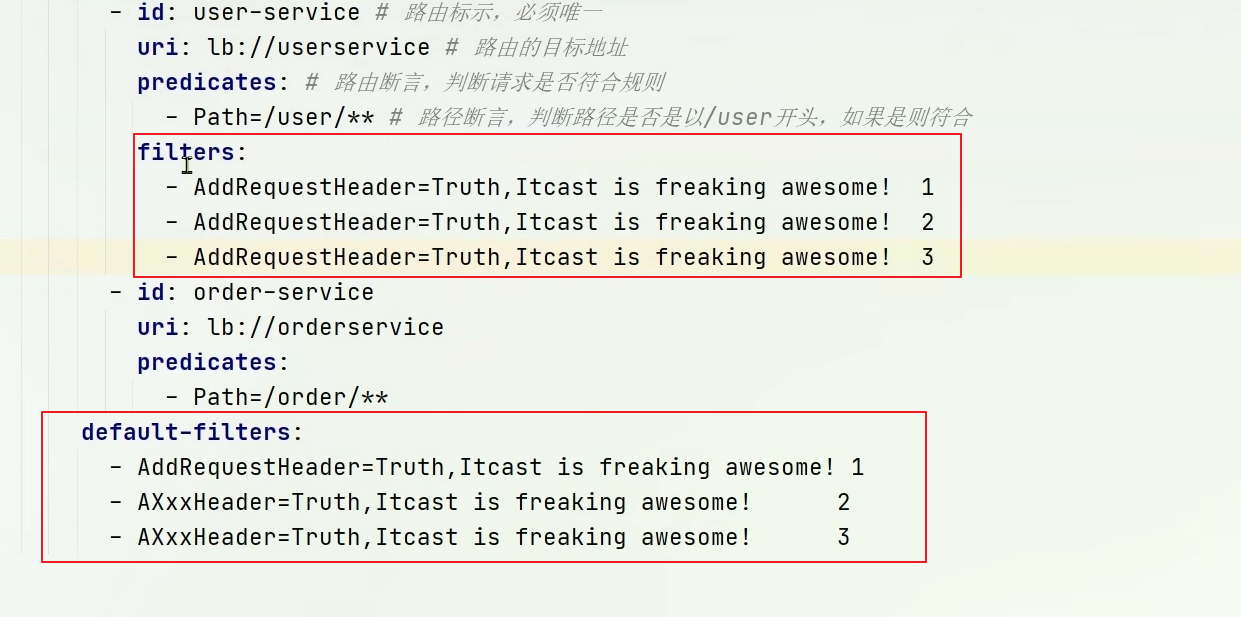
新问题：过滤器执行顺序怎么排序？

首先我们知道，过滤器的执行顺序是由一个int值的order决定的。每一个过滤器都必须指定一个int类型的order值，order值越小，优先级越高，执行顺序越靠前：

1：GlobalFilter通过实现Ordered接口，或者添加@Order注解来指定order值，由我们自己指定；

2：路由过滤器和DefaultFilter的order由Spring指定的，默认是按照声明的顺序从1递增，即最小是1；

3：当过滤器的order值一样时，会按照defaultFilter > 路由过滤器 > GlobalFilter的顺序执行！



总结：路由过滤器、defaultFilter、全局过滤器的执行顺序？

1：order值越小，优先级越高；

2：当order值一样时，顺序是defaultFilter最先，然后是局部的路由过滤器，最后是全局过滤器

### 网关Gateway跨域问题处理

#### 网关的跨域问题处理：

以前在java web阶段，已经学习过跨域问题以及跨域问题的解决方案了，在微服务里为什么要学这个问题呢？**因为在微服务里，所有请求，都要先经过网关，再到微服务，也就是说，跨域请求不需要在每个微服务中都去处理，仅仅在网关处理就可以了**，但是网关Gateway又和我们之前的实现不一样，因为Gateway是基于WebFlux实现的，没有Servlet相关的api，因此，以前在java web学习的跨域问题的解决方案，不一定能够适用。那网关里面该怎么处理跨域问题？

#### 何为跨域：

域名不一致就是跨域（域名不一致；域名相同，端口不用）

跨域就一定有问题吗？这可不一定，就本例来说，order-service的端口和user-service的端口不一致，在order-service中照样可以调用（访问）user-service。

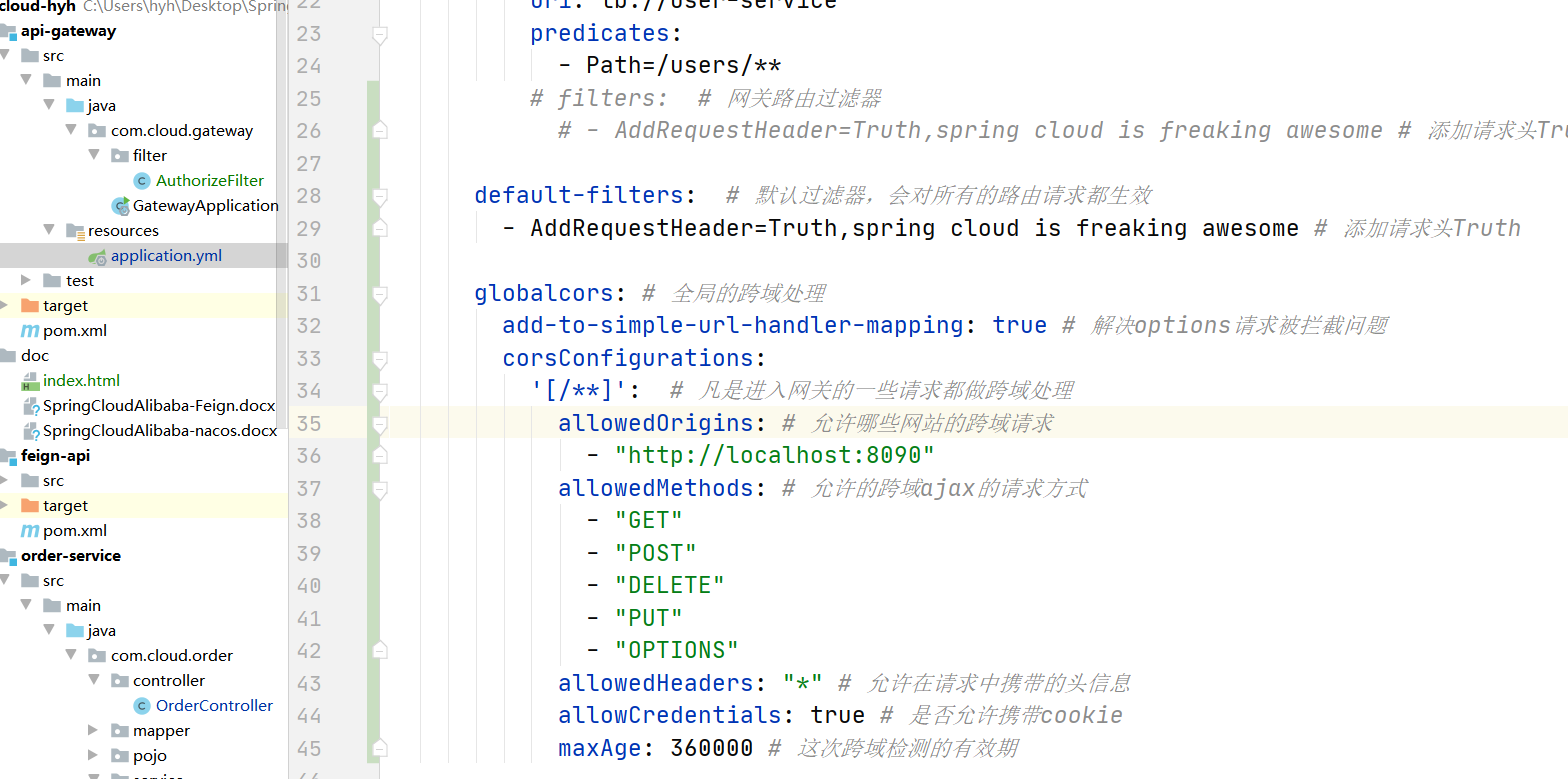
跨域问题：浏览器禁止请求的发起者与服务端发生跨域ajax请求，请求被浏览器拦截的问题！（关键点：浏览器禁止ajax跨域，order-service到user-service跟浏览器没任何关系，也不是ajax，所以不是跨域），性情查看doc文档中的index.html文件：



如何解决跨域问题：CORS，即浏览器去问一下服务端，允不允许ajax跨域，如果让，告诉我浏览器一声，我就不拦截!

所以服务器一般要配置各种信息：允许谁跨越，

Gateway网关其实已经帮我们做好了CORS的底层逻辑，我们只需要简单配置即可！见配置文件application.yml



小总结：

CORS跨域要配置的参数包括哪几个？

1：允许那些域名跨域？

2：允许哪些请求头？

3：允许哪些请求方式？

4：是否允许使用cookie？

5：有效期是多久？