

东软睿道内部公开

文件编号: D000-

Spring Cloud微服务架构

版本: 1.0.0

第7章 Zuul 路由网关

东软睿道教育信息技术有限公司
(版权所有, 翻版必究)

Copyright © Neusoft Educational Information Technology Co., Ltd
All Rights Reserved



本章教学目标

- ✓ 了解服务网关；
- ✓ 了解Zuul简介；
- ✓ 了解过滤器类型与请求生命周期；
- ✓ 了解过滤器禁用；
- ✓ 掌握Zuul微服务网关编写；
- ✓ 掌握微服务网关的路由测试、负载均衡测试、容错与监控测试；
- ✓ 掌握Zuul过滤器的编写和测试；

本章教学内容

节	知识点	掌握程度	难易程度	教学形式	对应在线微课
Zuul简介	为什么使用服务网关	了解		线下	
	服务网关	了解		线下	
	Zuul简介	了解		线下	
编写Zuul微服务网关	编写Zuul微服务网关	掌握		线下	
	路由测试	掌握		线下	
	负载均衡测试	掌握		线下	
	Hystrix容错与监控测试	掌握		线下	
	路由访问映射规则	掌握		线下	
Zuul的过滤器	过滤器类型与请求生命周期	了解		线下	
	编写Zuul过滤器	掌握	难	线下	
	Zuul过滤器测试	掌握		线下	
	禁用Zuul过滤器	了解		线下	

CONTENTS

目录

01

Zuul简介

02

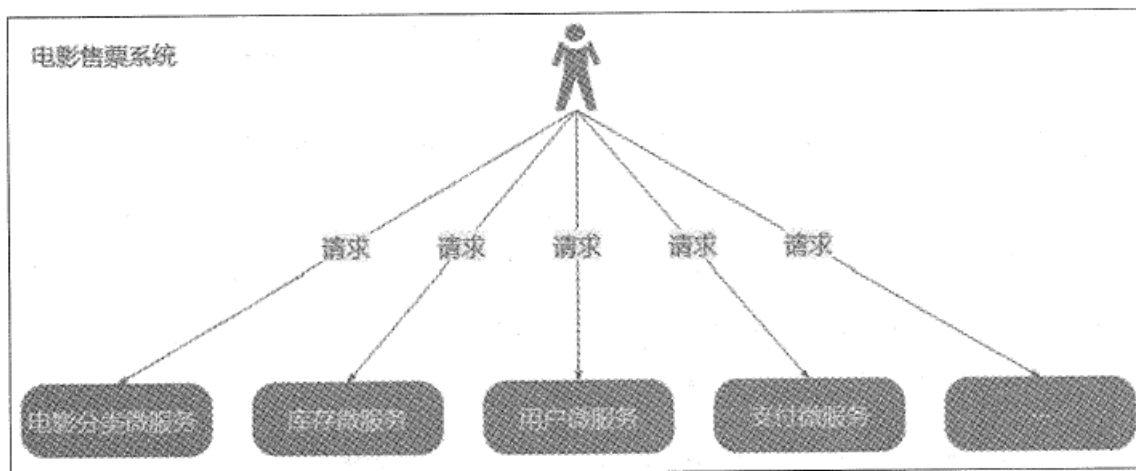
编写Zuul微服务网关

03

Zuul的过滤器

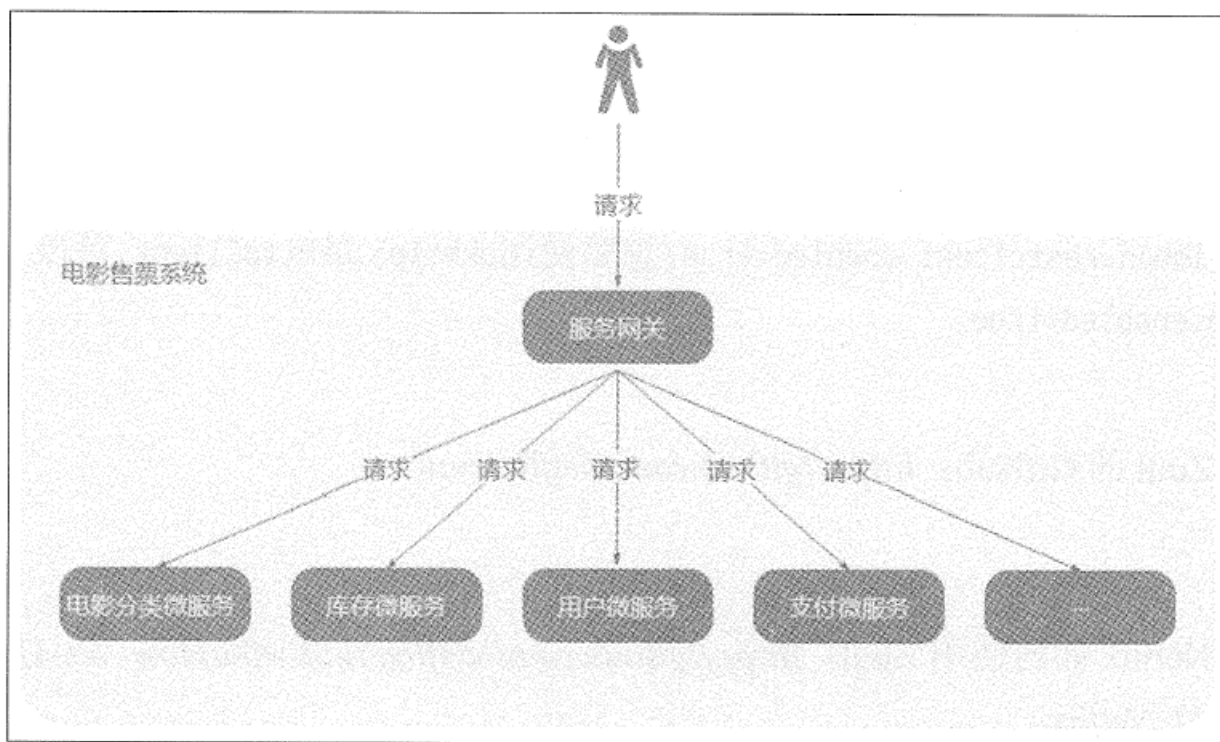
为什么使用服务网关

- ❖ 微服务场景下，每一个微服务对外暴露了一组细粒度的服务。客户端的请求可能会涉及到一串的服务调用，如果将这些微服务都暴露给客户端，那么客户端需要多次请求不同的微服务才能完成一次业务处理，增加客户端的代码复杂度，如下图。另外，对于微服务我们可能还需要服务调用进行统一的认证和校验等等。微服务架构虽然可以将我们的开发单元拆分的更细，降低了开发难度，但是如果不能够有效的处理上面提到的问题，可能会造成微服务架构实施的失败。



为什么使用服务网关

- ❖ 针对以上问题，可以借助微服务网关解决。
- ❖ 微服务网关介于客户端和服务端之间的中间层，所有的外部请求都会先经过微服务网关。如下图：



服务网关

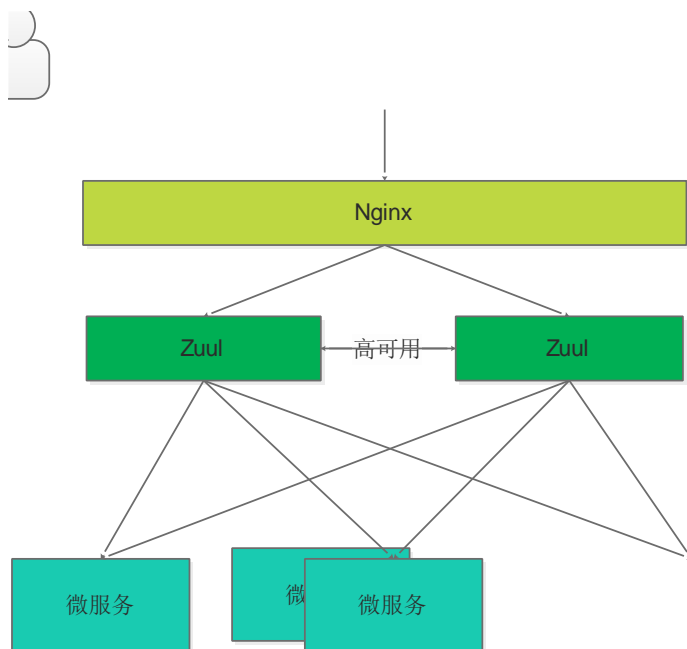
- ❖ 服务网关是在微服务前边设置一道屏障，请求先到服务网关，网关会对请求进行过虑、校验、路由等处理。有了服务网关可以提高微服务的安全性，校验不通过的请求将被拒绝访问。
- ❖ 前边介绍的Ribbon客户端负载均衡技术可以不用经过网关，因为通常使用Ribbon完成微服务与微服务之间的内部调用，而对那些对外提供服务的微服务，比如：用户登录、提交订单等，则必须经过网关来保证微服务的安全。

Zuul 简介

- ❖ Spring Cloud Zuul是整合Netflix公司的Zuul开源项目实现的微服务网关，它实现了请求路由、负载均衡、校验过滤等功能。
 - ▶ 其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础
 - ▶ 而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础
- ❖ Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。
- ❖ Zuul功能：过滤+路由

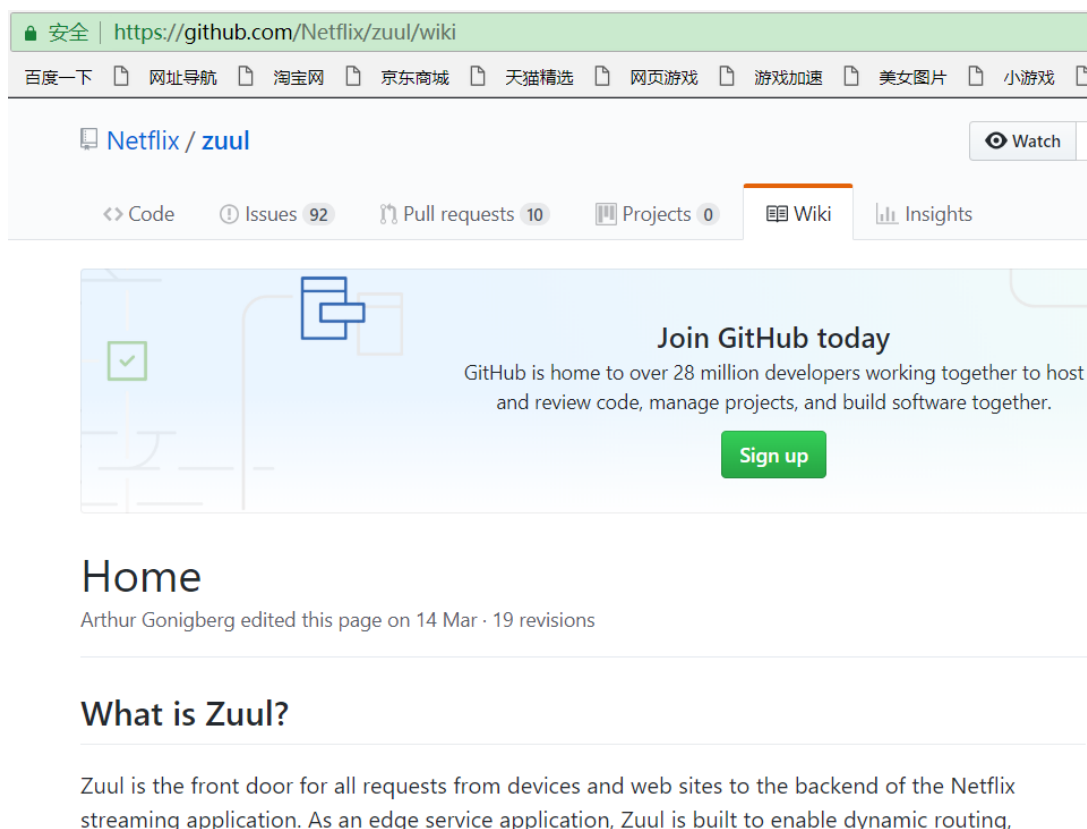
Zuul 简介

❖ 架构图



Zuul 简介

- ❖ Zuul项目在github上托管: <https://github.com/Netflix/zuul/>
 - ▶ 可以参考学习



CONTENTS

目录

01

Zuul简介

02

编写Zuul微服务网关

03

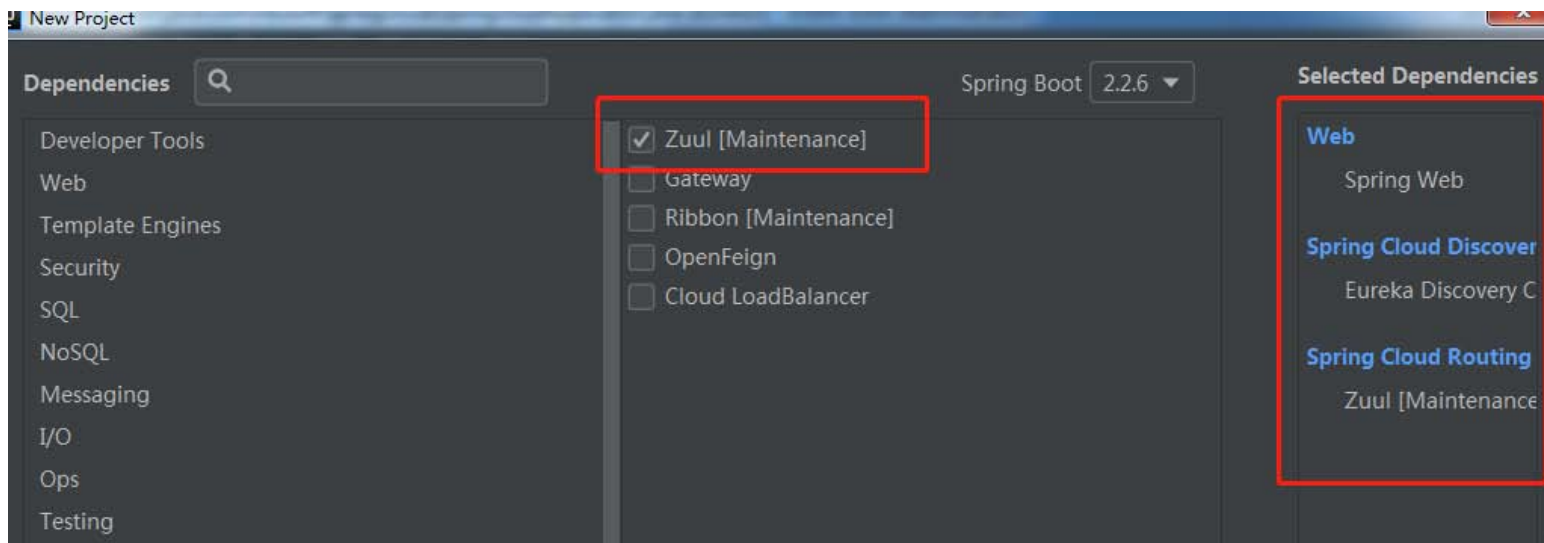
Zuul的过滤器

编写Zuul微服务网关

- ❖ 新建gateway工程
 - ▶ 全局配置文件application.yml
 - ▶ Hosts修改
 - ▶ 主启动类，添加@EnableZuulProxy

编写Zuul 微服务网关

❖ 新建工程gateway



编写Zuul 微服务网关

❖ 全局配置文件application.yml

```
server:
  port: 6001
spring:
  application:
    name: gateway
eureka:
  client:
    service-url:
      defaultZone:
        http://eureka.com:7001/eureka/, http://eureka2.com:7002/eureka/, http://eureka3.com:7003/eureka/
    instance:
      instance-id: gateway.com
# 访问路径可以显示IP地址
  prefer-ip-address: true
```

编写Zuul微服务网关

- ❖ hosts修改
 - ▶ 添加zuul网关映射

★ 127.0.0.1

myzuul.com

编写Zuul 微服务网关

- ❖ 主启动类
 - ▶ 添加@EnableZuulProxy

路由测试

- ❖ 先启动3个eureka集群
- ❖ 再启动provider, 端口8001
- ❖ 最后启动gateway
- ❖ 浏览器
 - ▶ 不用路由 `http://localhost:8001/user/findUserById/1`
 - ▶ 启用路由 `http://myzuul.com:6001/provider/user/findUserById/1`
 - ★ provider为生产者微服务的服务名 (spring.application.name)

← → ↻ ⓘ localhost:6001/provider/user/findUserById/1

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis"}
```

路由测试

❖ 可见Zuul网关已经注册到Eureka server中

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OT
DS Replicas

eureka2.com

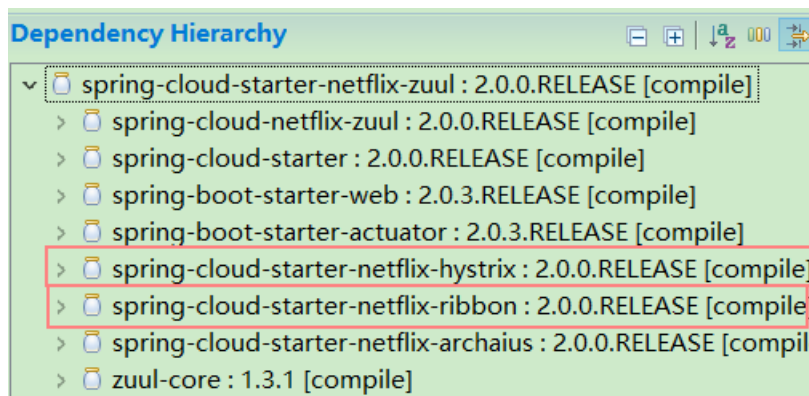
eureka3.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - gateway.com
PROVIDER	n/a (1)	(1)	UP (1) - user-provider

负载均衡测试

- ❖ Zuul整合了ribbon负载均衡和hystrix容错处理



负载均衡测试

- ❖ 启动3个eureka集群
- ❖ 启动3个provider, 8001、8002、8003
- ❖ 启动gateway, 6001
- ❖ 浏览器
 - ▶ 多次访问

★ <http://myzuul.com:6001/provider/user/findUserById/1>

← → ↻ ⓘ 不安全 | myzuul.com:6001/provider/user/findUserById/1

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis"}
```

← → ↻ ⓘ 不安全 | myzuul.com:6001/provider/user/findUserById/1

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis2"}
```

← → ↻ ⓘ 不安全 | myzuul.com:6001/provider/user/findUserById/1

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis3"}
```

负载均衡测试

- ❖ 多次刷新，数据来自于不同的provider
- ❖ 说明Zuul可以使用Ribbon达到负载均衡的效果

Hystrix容错与监控测试

- ❖ 先启动3个eureka集群
- ❖ 启动gateway, 6001
- ❖ 启动dashboard, 9001; 以及dashboard中的provider, 8001、8002、8003
- ❖ 启动turbine, 9002
- ❖ 浏览器
 - ▶ <http://myzuul.com:6001/provider/user/findUserById/1>
 - ▶ <http://myzuul.com:6001/provider/user/findUserById/888>

← → ↻ ⓘ 不安全 | myzuul.com:6001/provider/user/findUserById/888

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购 京东打折 梯度下降法的推导...

```
{"id":888,"loginName":"The user id 888 is not found!","username":null,"password":null,"dbSource":"no this data in Database"}
```

Hystrix容错与监控测试

❖ 查看注册服务

THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK
DS Replicas

eureka2.com

eureka3.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - gateway.com
PROVIDER	n/a (3)	(3)	UP (3) - user-provider1 , user-provider2 , user-provider3
TURBINE	n/a (1)	(1)	UP (1) - PC-20200320LFAL:turbine:9002

Hystrix容错与监控测试

- ❖ 在浏览器输入<http://localhost:9001/hystrix>进入监控面板主页面
 - ▶ 输入<http://localhost:9002/turbine.stream>进入默认集群面板页



Hystrix Dashboard

<https://hostname:port/turbine/turbine.stream>

Cluster via Turbine (default cluster): <https://turbine-hostname:port/turbine.stream>
 Cluster via Turbine (custom cluster): [https://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](https://turbine-hostname:port/turbine.stream?cluster=[clusterName])
 Single Hystrix App: <https://hystrix-app:port/actuator/hystrix.stream>

Delay: ms Title:

Hystrix Stream: test

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)

findUserById				
0	0	0	0	0.0 %
0	0	0	0	
0	0	0	0	
Host: 0.0/s				
Cluster: 0.0/s				
Circuit Closed				
Hosts	2	90th	0ms	
Median	0ms	99th	0ms	
Mean	0ms	99.5th	0ms	

Thread Pools Sort: [Alphabetical](#) | [Volume](#)

UserController				
Host: 0.0/s				
Cluster: 0.0/s				
Active	0	Max Active	0	
Queued	0	Executions	0	
Pool Size	2	Queue Size	5	

❖ 浏览器

- ▶ 多次访问 <http://myzuul.com:6001/provider/user/findUserId/1>

Hystrix容错与监控测试

❖ 启动集群监控测试

Hystrix Stream: test

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)



Host: 0.4/s

Cluster: 1.1/s

Circuit **Closed**

Hosts	3	90th	16ms
Median	7ms	99th	16ms
Mean	8ms	99.5th	16ms

Thread Pools Sort: [Alphabetical](#) | [Volume](#) |

UserController

Host: 0.4/s

Cluster: 1.1/s

Active	0	Max Active	3
Queued	0	Executions	32
Pool Size	30	Queue Size	5

路由访问映射规则

- ❖ 前面讲解的是路由基本配置，通常需要在全局配置文件中设置映射规则。
- ❖ 自定义代理名称
 - ▶ `zuul.routes.XX.path`=通配地址
 - ▶ `zuul.routes.XX.serviceId`=服务名称
 - ▶ `zuul:`
 `routes:`
 `user:`
 `serviceId: provider`
 `path: /user/**`
- ❖ 测试：
 - ▶ 不用路由 `http://localhost:8001/user/findUserId/1`
 - ▶ 使用服务名 `http://myzuul.com:6001/provider/user/findUserId/1`
 - ▶ 使用代理名 `http://myzuul.com:6001/user/user/findUserId/1`

路由访问映射规则

- ❖ 考虑到安全性，通常需要将原真实服务名忽略
- ❖ 忽略真实服务名

- ▶ `zuul.ignored-services=provider`
- ▶ 单个具体，多个可以用"*"

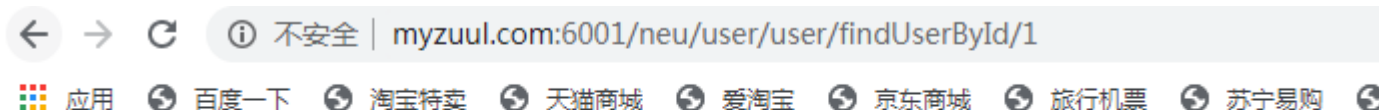
- ❖ 设置统一公共前缀

- ▶ `zuul.prefix=/neu`
- ▶ `zuul:`
 - `routes:`
 - `user:`
 - `serviceId: provider`
 - `path: /user/**`
 - `ignored-services: provider`
 - `# ignored-services: "*"`
 - `prefix: /neu`

- ❖ 测试:

- ▶ 访问只有一个入口

`http://myzuul.com:6001/neu/user/user/findUserById/1`



```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis2"}
```

CONTENTS

目录

01

Zuul简介

02

编写Zuul微服务网关

03

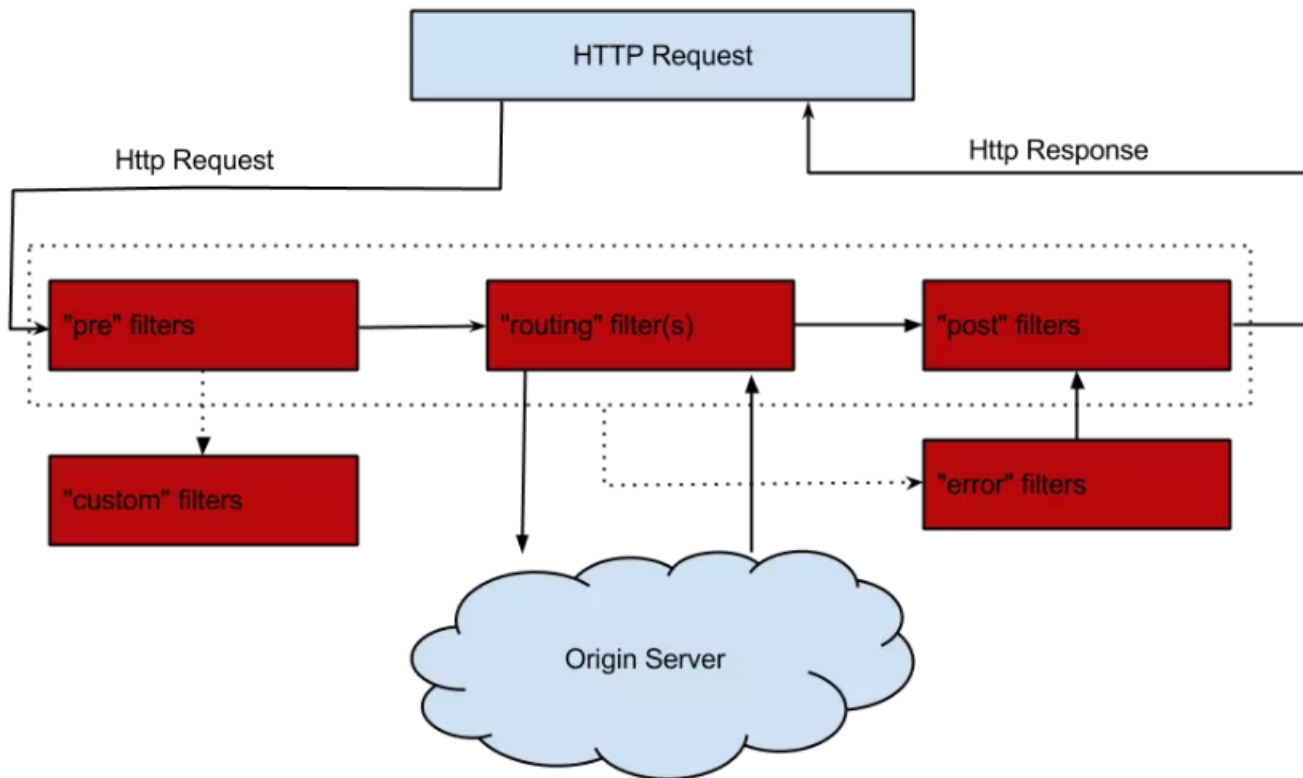
Zuul的过滤器

过滤器类型与请求生命周期

- ❖ 过滤器是Zuul的核心组件。
- ❖ 过滤器主要是用来进行权限管理，即通过过滤器来进行请求的拦截和过滤，过滤器中可以进行鉴权、签名校验、权限验证等。
- ❖ Zuul中定义了4种标准过滤器类型，这些类型对应于请求的典型生命周期。
 - ▶ pre: 可以在请求被路由之前调用
 - ▶ routing: 在路由请求时候被调用
 - ▶ post: 在routing和error过滤器之后被调用
 - ▶ error: 处理请求时发生错误时被调用

过滤器类型与请求生命周期

❖ 请求生命周期



编写Zuul过滤器

- ❖ 继承抽象类ZuulFilter，实现4个抽象方法
 - ▶ Object run()：主过滤逻辑处理
 - ▶ boolean shouldFilter()：判断该过滤器是否需要被执行
 - ▶ int filterOrder()：过滤器的执行顺序
 - ▶ String filterType()：过滤器类型
 - ★ pre
 - ★ route
 - ★ post
 - ★ Error

编写Zuul过滤器

- ❖ 示例，在gateway工程中新建立2个过滤器
 - ▶ 新建AccessTokenFilter
 - ★ 设置需要添加参数accessToken，执行顺序为1；
 - ▶ 新建AccessTokenFilter1
 - ★ 设置需要添加参数accessToken且参数不能等于zuul，执行顺序为2；

编写Zuul过滤器

❖ 过滤器AccessTokenFilter

@Component

```
public class AccessTokenFilter extends ZuulFilter {  
    // run: 过滤器的具体逻辑。  
    // 通过ctx.setSendZuulResponse(false)令zuul过滤该请求，不对其进行路由，  
    // 然后通过ctx.setResponseStatusCode(401)设置了其返回的错误码，  
    // 也可以进一步优化比如，通过ctx.setResponseBody(body)对返回body内容进行编辑等  
    @Override  
    public Object run() throws ZuulException {  
        RequestContext ctx = RequestContext.getCurrentContext();  
        HttpServletRequest request = ctx.getRequest();  
  
        Object accessToken = request.getParameter("accessToken");  
        System.out.println("accessToken:" + accessToken);  
        if (accessToken == null) {  
            ctx.setSendZuulResponse(false);  
            ctx.setResponseStatusCode(401);  
            return null;  
        }  
        return null;  
    }  
}
```

编写Zuul过滤器

❖ 过滤器AccessTokenFilter

```
// shouldFilter: 判断该过滤器是否需要被执行。  
// true表示该过滤器对所有请求都会生效。  
// 实际运用中我们可以利用该函数来指定过滤器的有效范围。
```

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

// filterOrder: 过滤器的执行顺序。当请求在一个阶段中存在多个过滤器时，需要根据该方法返回的值来依次执行。

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

```
// filterType: 过滤器的类型，它决定过滤器在请求的哪个生命周期中执行。  
// pre: 可以在请求被路由之前调用  
// route: 在路由请求时候被调用  
// post: 在route和error过滤器之后被调用  
// error: 处理请求时发生错误时被调用
```

```
@Override  
public String filterType() {  
    return "pre";  
}
```

编写Zuul过滤器

❖ 过滤器AccessTokenFilter1

@Component

```
public class AccessTokenFilter1 extends ZuulFilter {
```

```
    //设置需要添加参数accessToken且参数不能等于zuul
```

```
    @Override
```

```
    public Object run() throws ZuulException {
```

```
        RequestContext ctx = RequestContext.getCurrentContext();
```

```
        int code = ctx.getResponseStatusCode();
```

```
        System.out.println("accessToken1 response code:" + code);
```

```
        if(code == 401)
```

```
            return null;
```

```
        HttpServletRequest request = ctx.getRequest();
```

```
        Object accessToken = request.getParameter("accessToken");
```

```
        System.out.println("accessToken1:" + accessToken);
```

```
        if (accessToken == null || "zuul".equals(accessToken)) {
```

```
            ctx.setSendZuulResponse(false);
```

```
            ctx.setResponseStatusCode(402);
```

```
            return null;
```

```
        }
```

```
        return null;
```

```
    }
```

编写Zuul过滤器

❖ 过滤器AccessTokenFilter1

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

//执行顺序为2

```
@Override
public int filterOrder() {
    return 1;
}
```

```
@Override
public String filterType() {
    return "pre";
}
```

```
}
```

Zuul 过滤器测试

- ❖ 先启动3个eureka集群
- ❖ 启动gateway, 6001
- ❖ 启动dashboard中的provider, 8001、8002、8003
- ❖ 浏览器
 - ▶ 不用路由 <http://localhost:8001/user/findUserById/1>



```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis"}
```

Zuul 过滤器测试

❖ 浏览器

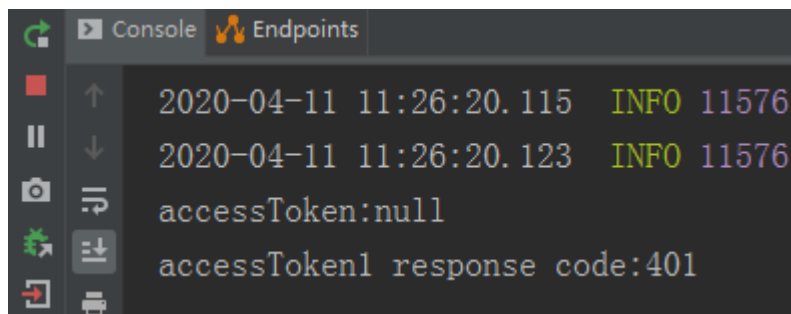
- ▶ 使用路由唯一访问路径

`http://myzuul.com:6001/neu/user/user/findUserId/1`

- ▶ 因为加入了Filter，所以没有正确的token，请求被拦截

← → ↻ ⓘ myzuul.com:6001/neu/user/user/findUserId/1

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购



```
2020-04-11 11:26:20.115 INFO 11576
2020-04-11 11:26:20.123 INFO 11576
accessToken:null
accessToken response code:401
```



该网页无法正常工作

如果问题仍然存在，请与网站所有者联系。

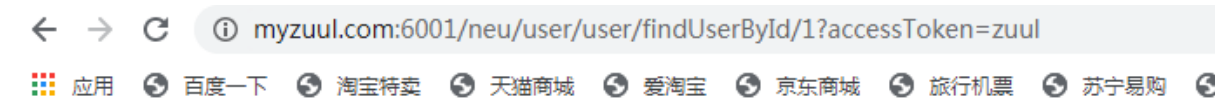
HTTP ERROR 401

重新加载

Zuul 过滤器测试

❖ 浏览器

- ▶ `http://myzuul.com:6001/neu/user/user/findUserById/1?accessToken=zuul`



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



该网页无法正常运行

如果问题仍然存在，请与网站所有者联系。

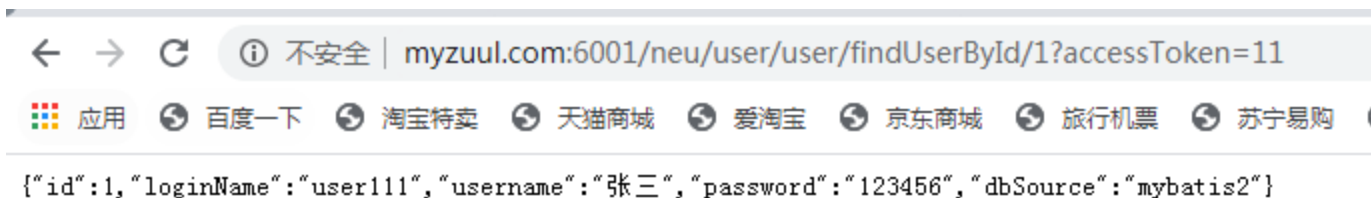
HTTP ERROR 402

重新加载

Zuul 过滤器测试

❖ 浏览器

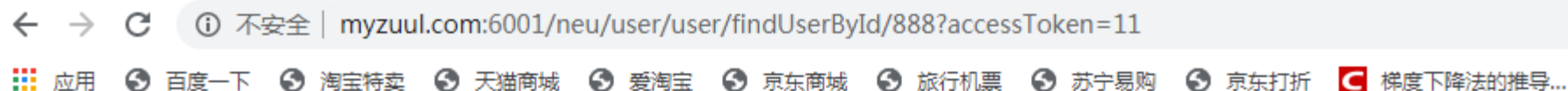
- ▶ `http://myzuul.com:6001/neu/user/user/findUserById/1?accessToken=11`



Zuul 过滤器测试

❖ 浏览器

- ▶ `http://myzuul.com:6001/neu/user/user/findUserById/888?accessToken=11`



```
{"id":888,"loginName":"The user id 888 is not found!","username":null,"password":null,"dbSource":"no this data in Database"}
```

禁用过滤器

- ❖ 一些场景下，如果要禁用部分过滤器，只需设置
 - ▶ `zuul.<SimpleClassName>.<filterType>.disable=true`，即可禁用 `SimpleClassName` 所对应的过滤器。
 - ▶ 例如，想要禁用 `org.springframework.cloud.netflix.zuul.filters.post.SendResponseFilter` 过滤器
 - ▶ 设置 `zuul.SendResponseFilter.post.disable=true` 即可

zuul:

```
  routes:
    user:
      serviceId: provider
      path: /user/**
  ignored-services: provider
# ignored-services: "*"
  prefix: /neu
  AccessTokenFilter:
    pre:
      disable: true
  AccessTokenFilter1:
    pre:
      disable: true
```

Zuul 总结

- ❖ springcloud zuul 包含了对请求的路由和过滤两个功能，其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础；而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验，服务聚合等功能的基础。
- ❖ 然而实际上，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。所以说，过滤器可以说是zuul实现api网关功能最核心的部件，每一个进入zuul的http请求都会经过一系列的过滤器处理链得到请求响应并返回给客户端。

本章重点总结

- ❖ 了解服务网关；
- ❖ 了解Zuul简介；
- ❖ 了解过滤器类型与请求生命周期；
- ❖ 了解过滤器禁用；
- ❖ 掌握Zuul微服务网关编写；
- ❖ 掌握微服务网关的路由测试、负载均衡测试、容错与监控测试；
- ❖ 掌握Zuul过滤器的编写和测试；

课后作业【必做任务】

- ❖ 1、独立完成课件中的示例

