# zuul简介

网关组件,实现整个微服务集群对外访问的唯一入口。

# zuul功能

## 路由

根据请求地址不同、网关会计算路由然后调用后端对应的微服务。

## 过滤（拦截）

网关中实现请求的鉴权工作(符不符合参数要求，是否携带身份信息，是否合法)。

# 网关工程的创建

## 创建项目

## pom.xml

### 继承 (spring-boot-parent)

|  |
| --- |
| <!--继承springboot--> *<*parent*>  <*groupId*>*org.springframework.boot*</*groupId*>  <*artifactId*>*spring-boot-starter-parent*</*artifactId*>  <*version*>*2.3.0.RELEASE*</*version*>  <*relativePath*/> <!-- lookup parent from repository --> </*parent*>* |

### 依赖

* eureka-client （通过网关路由到微服务，就必须要通过客户端从注册中心获取微服务信息）
* zuul依赖（内部包括了ribbon）

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

#### application.yml

|  |
| --- |
| server:  port: 8763  *#给功能起一个服务名称* spring:  application:  name: gate-way eureka:  instance:  *#ip优先相互通信* prefer-ip-address: true  *#访问的注册中心接口 会使用name到注册中心实现服务注册* client:  service-url:  defaultZone: http://127.0.0.1:8761/eureka *#网关路由* zuul:  routes:  api-a: *#自定义的名称* path: /zuul-user/\*\* *#外部访问路径匹配* service-id: ouu-user *#匹配后找对应的服务* api-b: *#自定义的名称* path: /zuul-order/\*\* *#外部访问路径匹配* service-id: ouu-order *#匹配后找对应的服务* |

## 创建启动类

|  |
| --- |
| @SpringBootApplication @EnableEurekaClient @EnableZuulProxy//开启zuul代理 目的就是引入zuul的过滤逻辑 //封装了ribbon /restTemplate public class StarterGateway {  public static void main(String[] args) {  SpringApplication.*run*(StarterGateway.class,args);  } } |

# 网关路由配置详解

|  |
| --- |
| *#网关路由* zuul:  routes:  api-a: *#自定义的名称* path: /zuul-user/\*\* *#外部访问路径匹配* service-id: ouu-user *#匹配后找对应的服务* api-b: *#自定义的名称* path: /zuul-order/\*\* *#外部访问路径匹配* service-id: ouu-order *#匹配后找对应的服务* |

## zuul.routes

固定的路由配置前缀

## api-a/api-b:

自定义的路由名称，一般这里配置的路由名称和调用的功能有关,一对路由配置这个名字保持一致

## path

ant匹配规范,匹配的是访问到网关的请求uri地址是否满足这里的规范

/zuul-a/\*\*表示只要请求到网关的uri地址是以/zuul-a/开始的就满足匹配

例如/zuul-a/haha,/zuul-a/a/b,/zuul-a/.其中ANT匹配规范如下

|  |  |
| --- | --- |
| ? | 表示匹配一级(/为1级)单个字符，例如：path=/zuul-a/?,可以匹配到/zuul-a/a,/zuul-a/b,不能匹配/zuul-a/abc,/zuul-a/a/b/c |
| \* | 表示匹配一级任意字符串，例如：path=/zuul-a/\*,可以匹配到/zuul-a/a,/zuul-a/abc,不能匹配/zuul-a/a/b/c |
| \*\* | 表示匹配任意多级任意字符串，例如：path=/zuul-a/\*\*,可以匹配到/zuul-a/a,/zuul-a/abc,/zuul-a/a/b/c.没有不能匹配的 |

## serviceId（微服务名称）

路由匹配在zuul中一对一对出现,path匹配请求地址,一旦匹配上将会调用serviceId的微服务

## 优先级问题

如果功能非常复杂,多个微服务配置路由时path有包含关系,相当于一个请求到网关的功能匹配到多个path(优先级)

* 例如:

|  |
| --- |
| zuul.routes.api-a.path=/zuul/\*  zuul.routes.api-a.service-id=service1  zuul.routes.api-b.path=/zuul/\*\*  zuul.routes.api-b.service-id=service2 |

* 请求网关:

localhost:8103/zuul/abc 到底在网关当中调用是service1 service2

有可能是service1调用,也有可能是service2(properties格式)

zuul中对应包含关系的匹配规则path具有优先级的判断,先加载到内存的路由优先级高于后加载的.

properties的格式文件,先后加载和手写顺序未必一致(hashMap).可以通过yml格式编写,定义优先级,yml格式就是按照先写先加载的逻辑,读取属性.

但是实际场景中,不太可能使用到有包含关系的path.

# 网关集群和结构

|  |
| --- |
|  |

最终经过eureka治理组件和ribbon组件形成微服务集群,并且实现服务调用服务的功能

通过网关组件实现所有的微服务功能,必须经过网关才能访问

引入nginx实现外界访问入口网关的集群负载均衡

这种结构才是springcloud最基本的完整的结构

# Js请求到微服务整体流程

## js请求起始地址：

<http://www.ou.com/user/query/point>

|经过hosts文件ip地址映射找到127.0.0.1

[http://127.0.0.1/user](http://127.0.0.1/orderuser01/pay.action)/query/point

## nginx接收请求

|经过监听80端口 监听www.ou.com满足要求，进入server的location

|location =/user/query/point 匹配到这个location 剩余uri字符串/

|proxy\_pass [http://127.0.0.1:8103/zuul](http://127.0.0.1:8103/zuul-pay/pay.action)-tuser/user/query/point/

|从nginx发送出去的请求 zuul网关

## 进入zuul网关

|http://127.0.0.1:8103/zuul-tuser/user/query/point/

|path匹配路径 /zuul-tuser/\*\* 剩余url地址 /user/query/point/

|path匹配成功找到路由规则api-a 对应找到了服务名称ouu-user

|http:// ouu-user/user/query/point

zuul网关内部ribbon拦截生效将ouu-userr找到对应实例(负载均衡)详细信息ip:port访问后端微服务

从zuul网关出去:

<http://127.0.0.1:9001/user/query/point>

## URL变化

进入nginx请求

<http://www.ou.com/user/query/point>

进入zuul网关请求

<http://127.0.0.1:8103/zuul-tuser/user/query/point>

进入服务提供者请求

<http://127.0.0.1:9001/user/query/point>

# zuul的过滤功能

网关作为微服务的集群唯一入口,非法的请求经过过滤,没有身份的请求拦截.

过滤：判断力度较小

拦截：力度较大

应用场景：鉴权作用.

本身网关的所有功能就是一套过滤机制.自定义过滤器class可以直接添加到网关系统

## 编写实现类并重写相关方法

下例中判断请求的url是否以/zuul-a/开始，若是则判断是否有请求参数name，有则放行，否则自定义消息返回

重要 使得过滤器的类生效,必须让他成为容器的一个bean

添加类的注解@Component

或配置类的方法上@Bean生成

|  |
| --- |
| /\*\*  \* @作者 舒新胜  \* @项目 easymall-2002-all  \* @创建时间 2020/6/12 11:32  \*/  *@Component* public class MyZuulFilter extends ZuulFilter {  */\*\*  \* 过滤器类型  \* @return  \*/* @Override  public String filterType() {  return null;  }   /\*\*  \*多个过滤器顺序  \* @return  \*/@Override  public int filterOrder() {  return 0;  }   /\*\*  \* 是否启动过滤  \* @return  \*/@Override  public boolean shouldFilter() {  //获取zuul上下文  RequestContext context=RequestContext.*getCurrentContext*();  //请求对象  *HttpServletRequest* request = context.getRequest();  //除去域名的上下文请求地址  String requestURI = request.getRequestURI();  //是否以/zuul-a/开始  boolean b = requestURI.startsWith("/zuul-a/");  return b;  }   @Override  public Object run() {  //过滤逻辑,判断name是否为参数  //拿到当前请求request,拦截,获取respone  //获取上下文  RequestContext context = RequestContext.*getCurrentContext*();  //请求对象  *HttpServletRequest* request = context.getRequest();  //响应对象  *HttpServletResponse* response = context.getResponse();  //从request拿到参数  String name = request.getParameter("name");  if(name==null){  //请求中没有任何形式的name的参数  //进行拦截,防止后续调用逻辑   //请求是否执行后续逻辑  context.setSendZuulResponse(false);  //设置响应码  context.setResponseStatusCode(403);  //设置响应编码  response.setContentType("text/html;charset=utf-8");  //手动响应返回数据  context.setResponseBody  ("{\"status\":\"201\",\"msg\":\"参数数据格式不正确\"}");  }  return null;  } } |

## 过滤器重写的四个方法介绍

### String filterType

返回值表示过滤器的类型，类型不同过滤器使用的位置不同

"pre":进入zuul网关时加载的过滤器

"route":计算路由时记载的过滤器

"error":在上述2个过滤器中任何一个出现问题异常时，会进入这个过滤器

"post":最终的过滤器，无论如何都会经过

|  |
| --- |
| 计算机生成了可选文字: 请 求 链  request  response  P re  route  主 要 用 来 改 变 路 由 规  error  request  response  post |

### int filterOrder()

如果自定义了多个同一种类型的过滤器,可以通过这里的返回值定义执行顺序，返回值可正可负,值越小，顺序执行越靠前

### boolean shouldFilter()

主要是判断当前请求该不该进入到过滤逻辑run()方法.例如:

鉴权工作,并不是所有的请求都要执行鉴权的

返回false:表示不需要过滤逻辑执行

返回true:表示需要过滤逻辑执行,进入到run方法

### Object run()

过滤的核心逻辑对应的方法,返回值没有任何效果,一般都是在run里处理request和response

# 微服务框架下的错误排查

## 问题定位

在微服务框架下启动的程序功能如果出现访问错误，应该从不同层面先去排查各自的问题。目前结构中由3个层面

* 第一层：微服务提供者，单独访问出现问题，不用在看网关nginx

<http://127.0.0.1:9001/user/query/point?userId=1>

保证这里返回的数据，执行的功能是正确的，如果不正确检查代码项目

* 第二层：添加网关后，通过路由匹配，服务调用访问后端微服务提供者

<http://127.0.0.1:8103/zuul-tuser/user/query/point?userId=1>

保证单独访问网关能够正常调用后端微服务，如果出错，一定路由匹配错误，测试请求地址和路由匹配不配套（检查path，检查serviceId值，检查访问网关是否匹配path）

* 第三层：nginx做负载均衡访问网关，单独测试nginx请求，就是js请求地址

<http://www.ou.com/user/query/point?userId>

保证返回值也是正确的，如果出现错误，检查nginx. conf(proxy\_pass是否正确访问网关的路由匹配规则)，hosts

|  |
| --- |
| 计算机生成了可选文字: nglnx 负载均衡 zuul网关 zuul网关 由匹配 务调用 order 具亻本工力自 user 具亻本工力自 |

## 核心能力

了解所有功能的正确流转过程

起始地址js发起的请求 F12看浏览器开发者模式network

请求进入nginx如何发送出

进入zuul网关如调用哪个微服务哪个功能

## 常见的问题

### Forwarding error/ Load balancer does not have available server for client: test-use

zuul网关负载均衡无法找到一个微服务名称叫做test-use

原因：当进入zuul网关经过路由匹配确实成功了，但是无法找对应调用的微服务，zuul启动时从eureka没有抓取这个服务

application.properties 微服务serviceId 拼错了

zuul网关在微服务之前启动（等一段时间就好了）

### read time out 访问超时

原因：启动所有工程后，相互间的联系通过eureka还没有建立稳定

等一段时间/重启一下网关进程

测试时访问不同层此的url本身就是错的

### com.sun.jersey.api.client.ClientHandlerException: java.net.ConnectException: Connection refused: connect

### com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server

微服务提供者启动时,作为eureka-client携带自己的信息,到注册中心注册,每隔30秒心跳,每隔30秒中抓取注册中心服务

访问注册中心地址,同时一个属性指定

eureka.client.service-url.defaultZone=http://127.0.0.1:8761/eureka

微服务提供者报上述异常:

注册中心没启动

微服务提供者配置属性地址错了

注册中心报上述异常

高可用注册中心相互注册,必定至少报一次这个异常--正常的

注册中心把自己当成客户端在自己注册,必定会报异常至少一次---正常