标识： 密级

版本：001

微服务培训文档(第六讲)

二○一七年十二月

文档修改记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修改内容描述 | 修改人 | 日期 | 备注 |
| 001 | zuul学习 | 张洪图 | 2017-12-05 | add |
| 002 | zuul的请求分发 | 张洪图 | 2017-12-07 | add |
| 003 | zuul的文件上传和断熔处理 | 张洪图 | 2017-12-10 | add |
| 004 | zuul over | 张洪图 | 2017-12-10 | over |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

[讲解计划 4](#_Toc825273687)

[1.1 zuul简介 4](#_Toc726618291)

[1.2使用zuul 5](#_Toc2120928685)

[S1:创建project 6](#_Toc1396719668)

[S2:创建eureka的module 7](#_Toc2040652145)

[S3:创建网关module 10](#_Toc1428656320)

[S4:创建hello-service的module 12](#_Toc184857340)

[S5:创建hi-service的module 13](#_Toc1787955180)

[1.3使用zull2 19](#_Toc1510720785)

[1.4不经过zuul进行文件上传 24](#_Toc83273619)

[1.5 zuul断熔处理 26](#_Toc1618974988)

# 讲解计划

spring cloud基础入门、eureka服务发现

ribbon负载均衡、hystrix断路器、feign代理

zuul网关、config、bus、stream

说明：spring cloud基础讲解已经完成

在第二讲中,讲解注册中心eureka;在第三讲中,讲解负载均衡ribbon;在第四讲中,讲解断路器hystrix;在第五讲中,讲解代理feign;在第六讲中,讲解网关zuul;今天讲解的内容就是zuul.

zuul和feign内部都包含ribbon和hystrix,可以理解ribbon和hystrix是小组件,zuul和feign内部集成ribbon和hystrix的.然后在项目中可以直接使用zuul和feign就可以.

第六讲就是讲解zuul内容

## 1.1 zuul简介

什么是网关呢?网关就是网络请求的入口,外部服务通过网关进行访问我们内部的请求.

网关的作用是什么呢?网关可以进行安全,权限认证,请求拦截,限制流量,数据校验,api路由转发等功能.针对于公司目前的接口项目,可以进行全局性设置,比如签名验证,安全验证,请求拦截,访问次数限制,请求转发等功能.

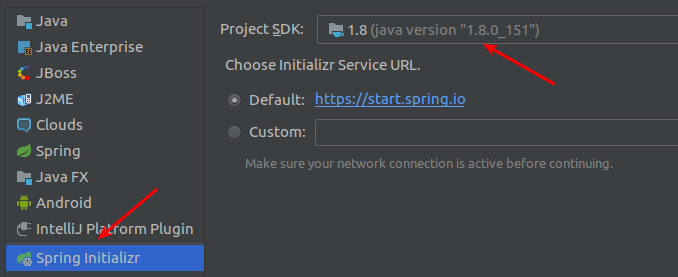
对于安全的处理:处理微服务的安全性问题,需要将权限控制的单元模块,从业务的服务单元中抽取出去,因为这些都是通用的全局设置的内容,最适合安全设置的地方就是在对外访问最前端的地方,需要更加强大负载均衡能力,那就是使用zuul.

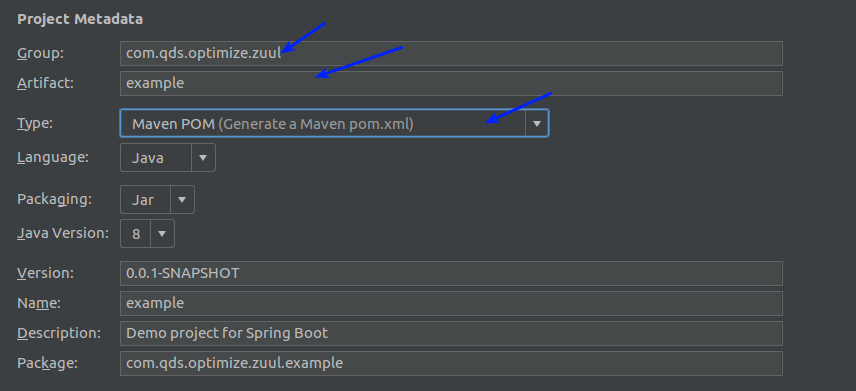
网关在微服务中的地位:首先网关是不可或缺的部分,通过网关可以统一向外提供rest api,除了具备服务路由,负载均衡能力,还有权限控制能力.spring cloud netflix中的zuul就担当这个重要的角色,zuul为微服务家门提供前门保护的作用,同时可以讲这些较重的非业务逻辑内容迁移到服务路由层处理,使得微服务集群主题能够具备更好的可复用性和可测试性.

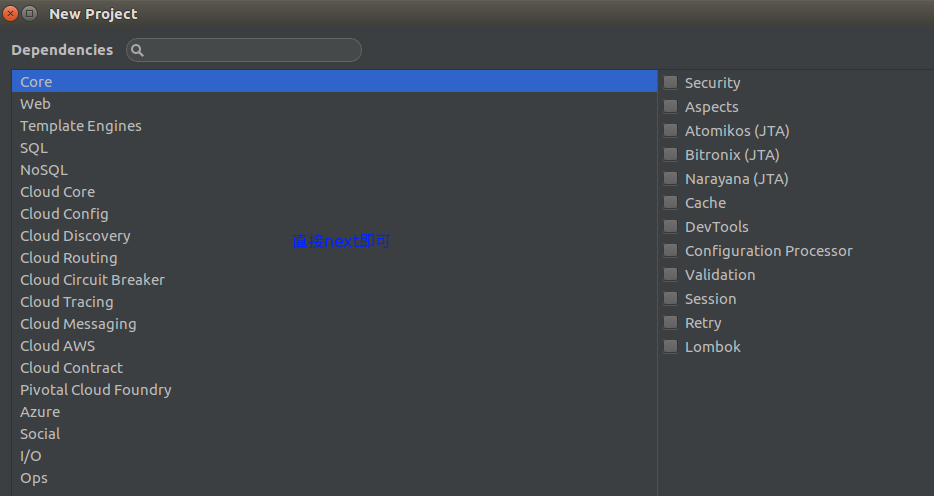
如果没有网关会有什么情况呢?那就是好比现在的安全处理,在商标搜索中,在专利搜索,在接口项目中都有安全限制的代码,都有校验签名的代码,代码没有很好的复用,只是新出现的项目中,导致代码不断地重复,如果后续进行安全升级和改造,那么随着module的增多,改动量就会出现成倍的情况.

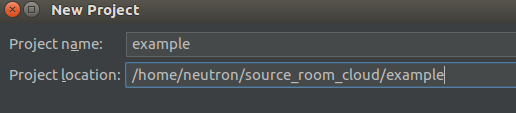
## 1.2使用zuul

### S1:创建project



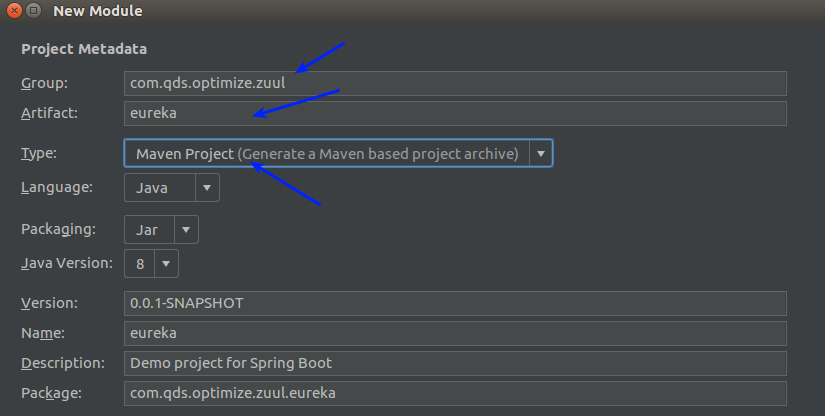


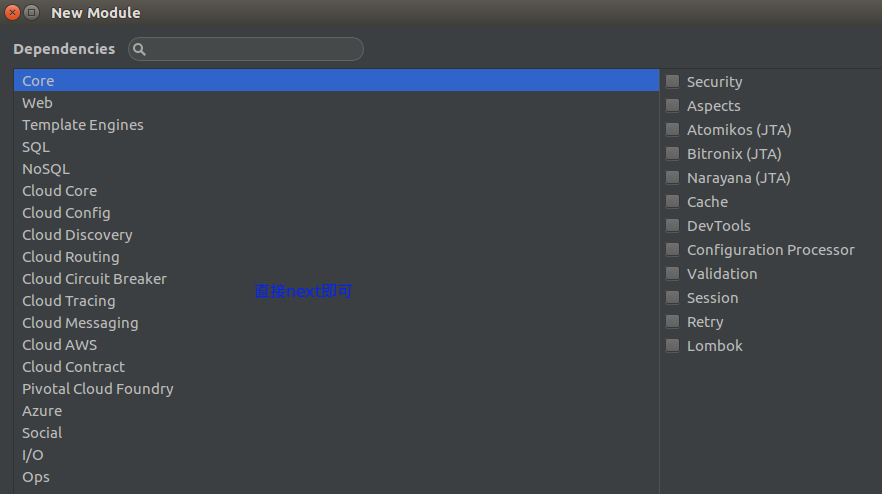


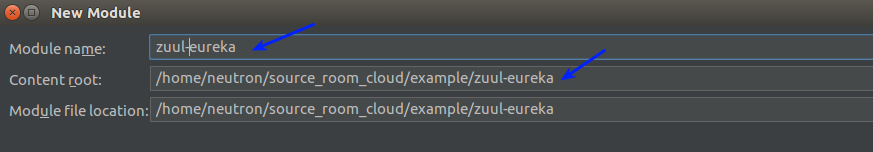


每个人的项目路径自己处理,我重新处理主要是为了不同的知识点放置在不同的路径下面,方便大家查看练习.

### S2:创建eureka的module







因为是zuul的demo,所以都以zuul开头,因此module name设置为zuul-name.请参考之前的module创建,来配置入口类,配置文件内容.

只有在2017年2月版本后的内容才会出现Dashboard界面,之前的版本是没有Dashboard的操作界面的.

查看IDEA的about内容如下:

IntelliJ IDEA 2017.2.5

Build #IU-172.4343.14, built on September 26, 2017

Licensed to lan yu

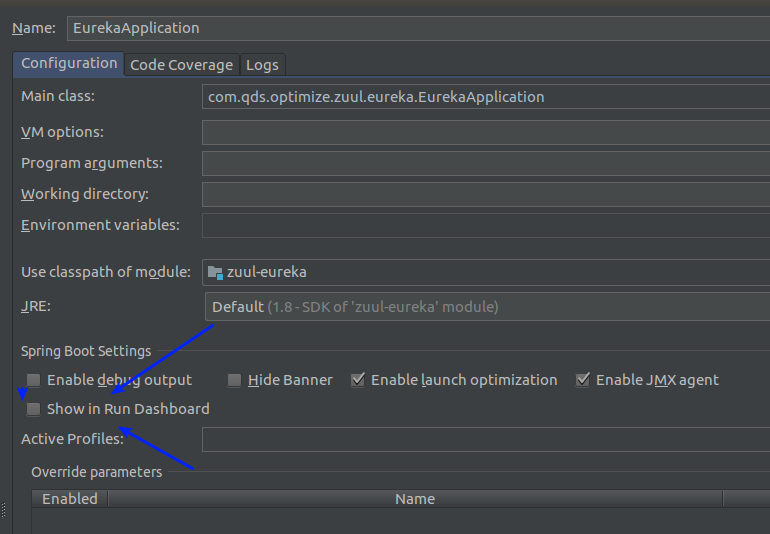
Subscription is active until October 14, 2018

JRE: 1.8.0\_152-release-915-b12 amd64

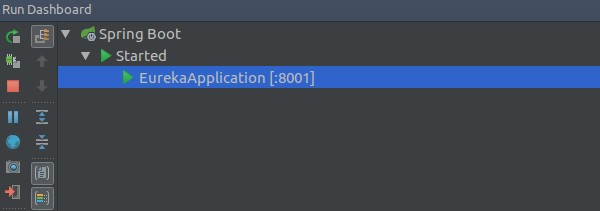
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

Linux 4.10.0-38-generic

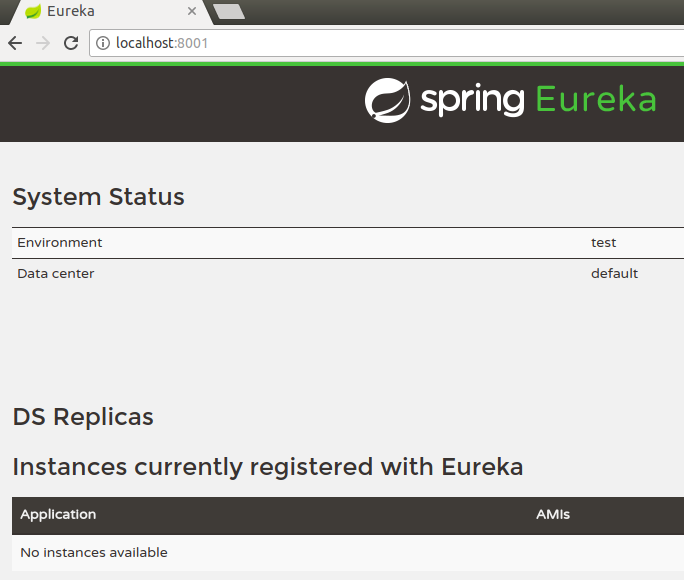
如何出现Dashboard内容.勾选某个module的执行,然后勾选show in Run Dashboard就会出现Dashboard.



然后运行eureka的module

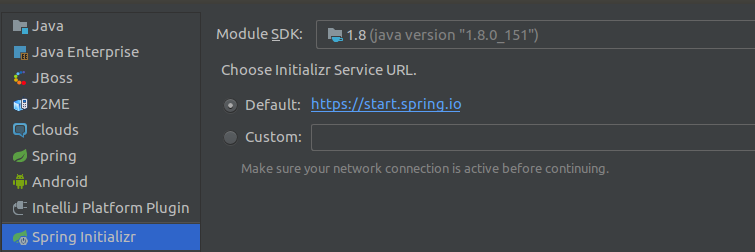


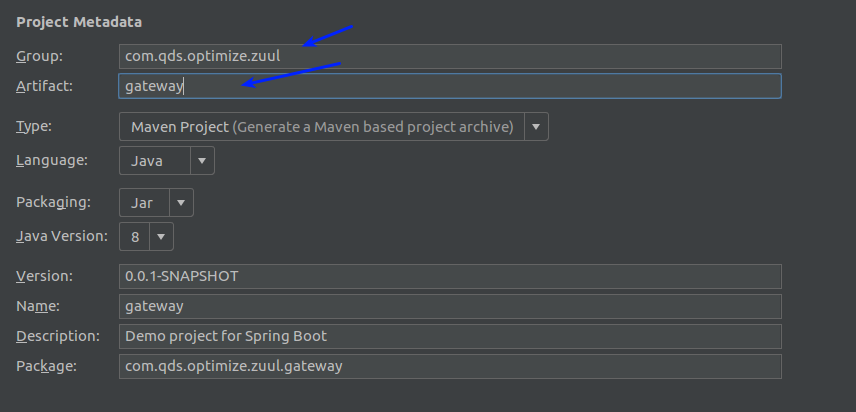
注册中心界面如下:

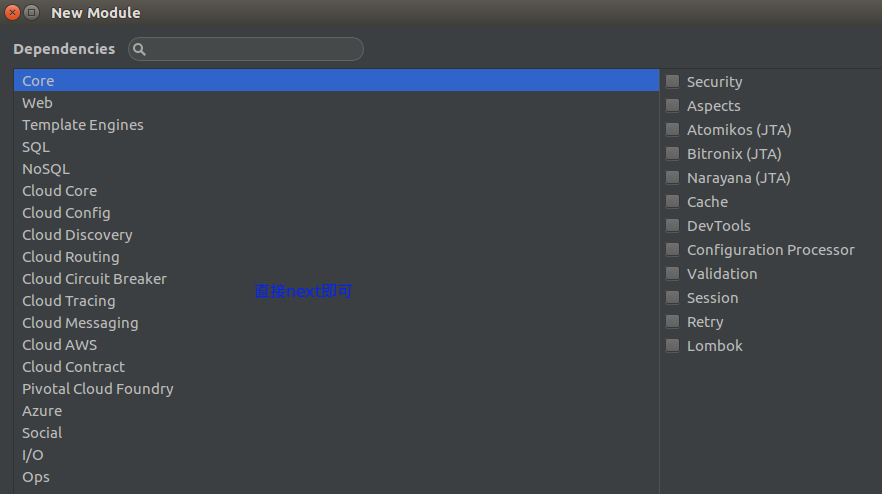


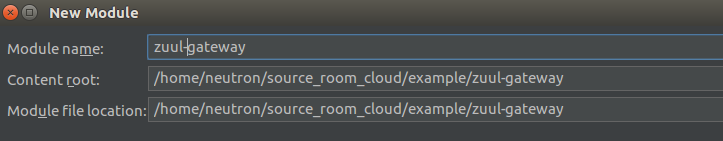
至此注册中心处理完成.

### S3:创建网关module

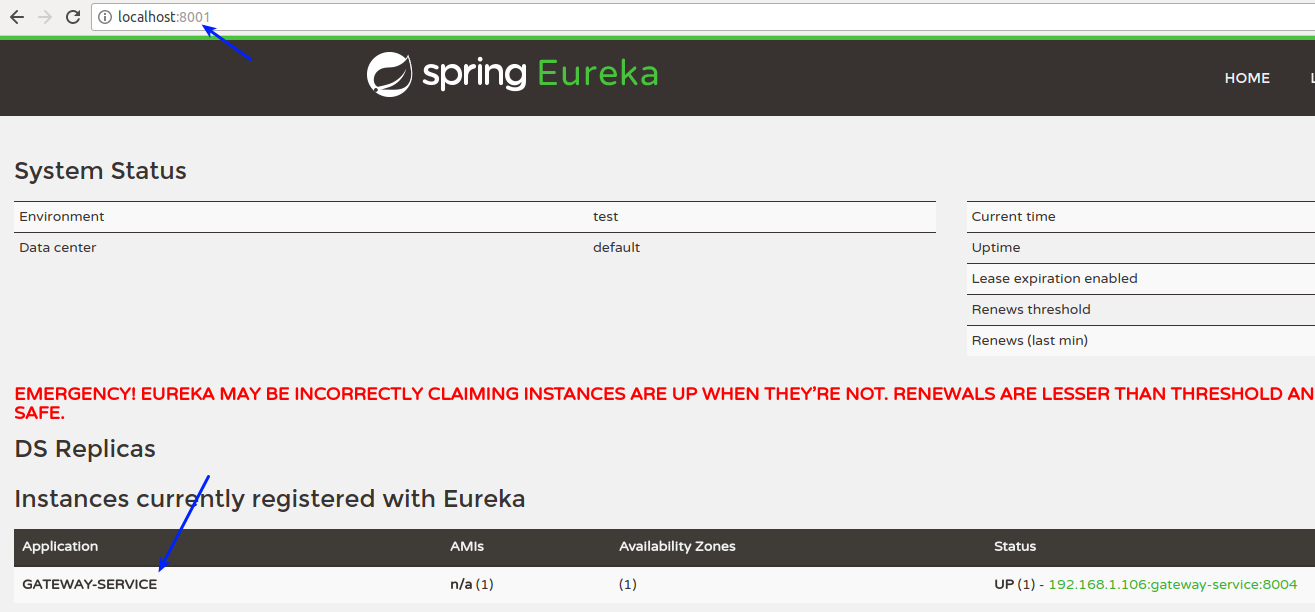




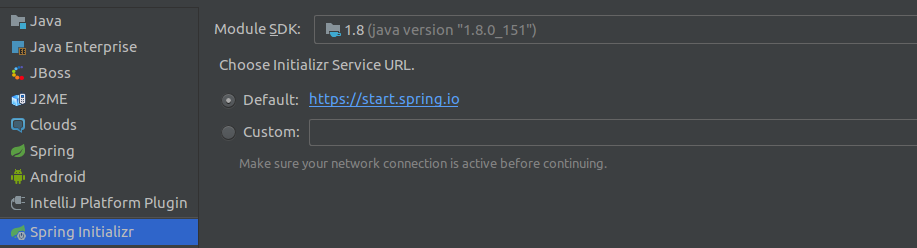


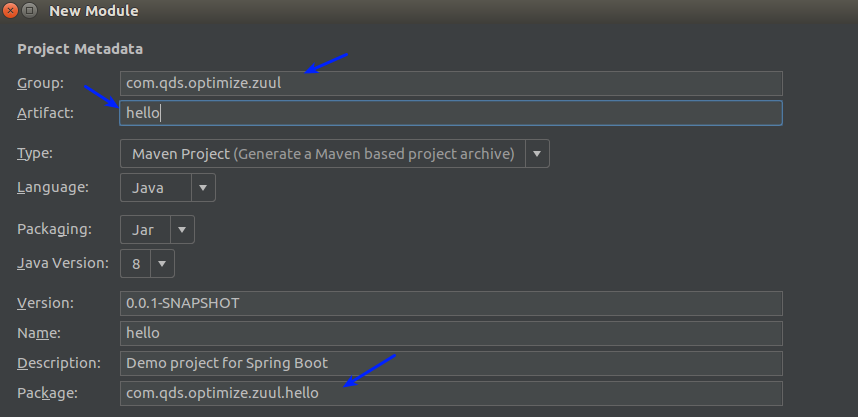


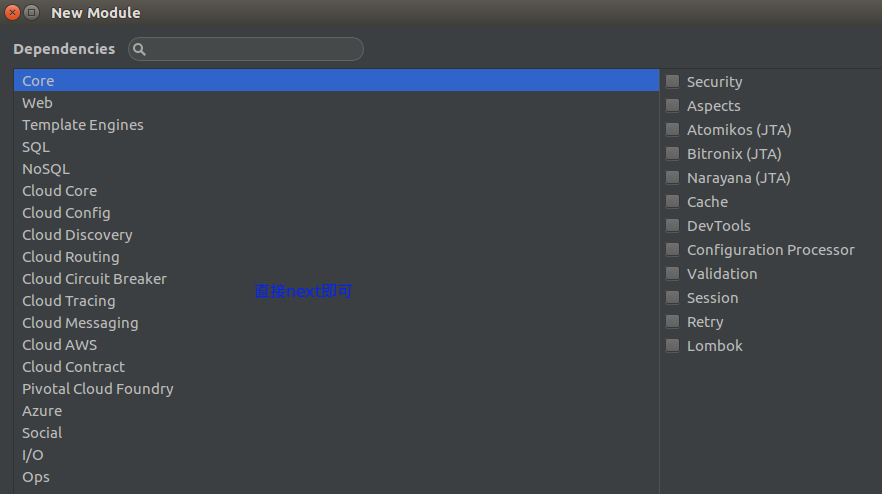
处理网关module的pom.xml,yml文件,入口类内容,查看注册中心如下:

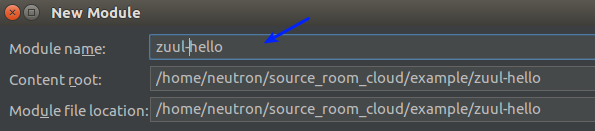


### S4:创建hello-service的module





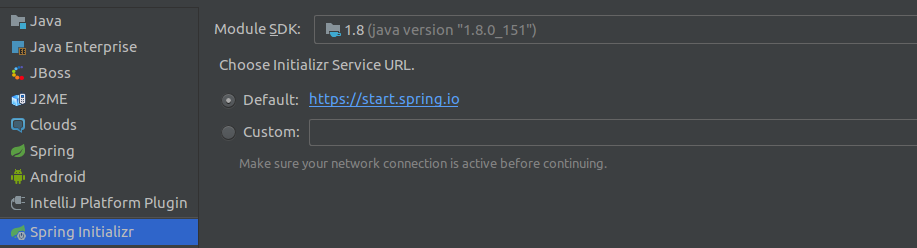


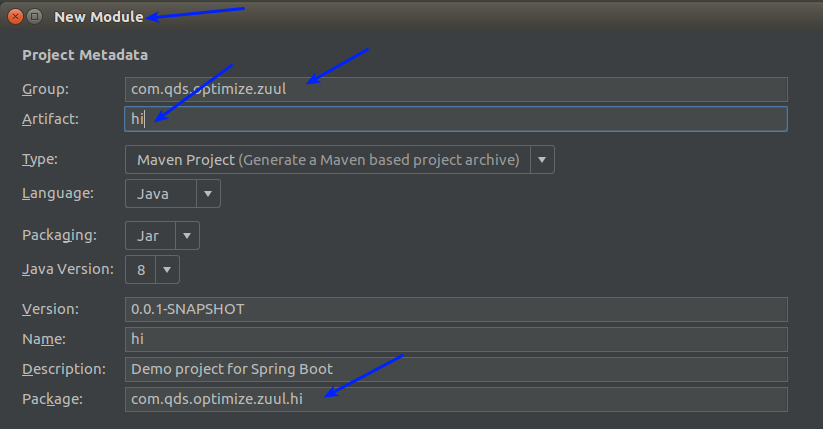


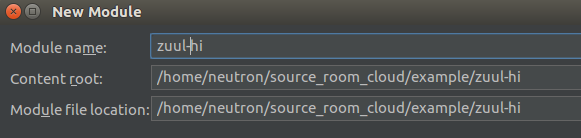
创建完成hello的服务后,修改pom.xml,yml文件,入口类配置

### S5:创建hi-service的module

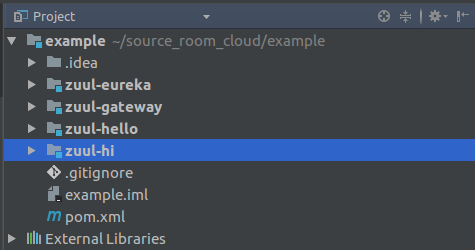
参考创建hello-service的module的内容.



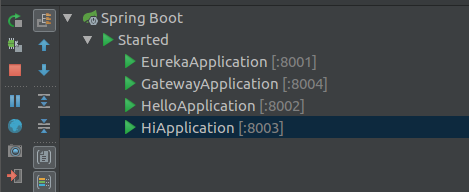




最后的项目结构如下:



启动的Dashboard如下:



参考项目:

第1步:git的第1个commit,创建pom类型的project

第2步:git的第2个提交创建eureka的注册中心的module

第3步:git的第3个提交创建网关gateway的module

第4步:git的第4个提交创建hello的module

第5步:git的第5个提交创建hi的module

在gateway的module中

使用spring cloud zuul非常easy,使用zuul的步骤如下:

1. 引入spring-cloud-starter-zuul,主入口加入注解@EnableZuulProxy即可.

<!-- 网关必须引入spring-cloud-starter-zuul依赖 -->

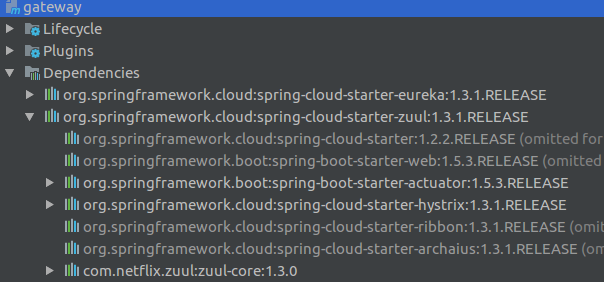
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

通过查看zuul依赖,可以得到以下结论



zuul已经集成web,actuator,hystrix,ribbon的jar依赖.zuul,和feign一样,都有断路器和负载均衡的作用.

使用复合注解@SpringCloudApplication

查看复合注解的内容:

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@SpringBootApplication

@EnableDiscoveryClient

@EnableCircuitBreaker

public @interface SpringCloudApplication {

}

可以使用SpringCloudApplication来代替SpringBootApplication+

EnableDiscoveryClient+EnableCircuitBreaker

1. .配置yml文件,类似的配置如下:

zuul:

routes:

api-hello:

path: /hello-service/\*\*

serviceId: hello-service

api-hi:

path: /hi-service/\*\*

serviceId: hi-service

解说routes的数据类型就是LinkedHashMap,private Map<String, ZuulProperties.ZuulRoute> routes = new LinkedHashMap();

针对于routes的kv来说,都是自己指定的,但是需要有意义的数据,可以参考api-xxx来作为key,value的值是对象

ZuulRoute有以下属性:

private String path;//The path (pattern) for the route, e.g. /foo/\*\*.

private String serviceId;//The service ID (if any) to map to this route. You can specify a physical URL or a service, but not both

private String url;// A full physical URL to map to the route. An alternative is to use a service ID and service discovery to find the physical address.

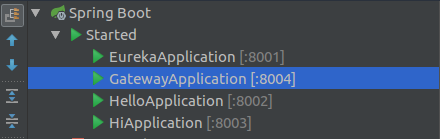
private boolean stripPrefix = true;//Flag to determine whether the prefix for this route (the path, minus pattern patcher) should be stripped before forwarding.

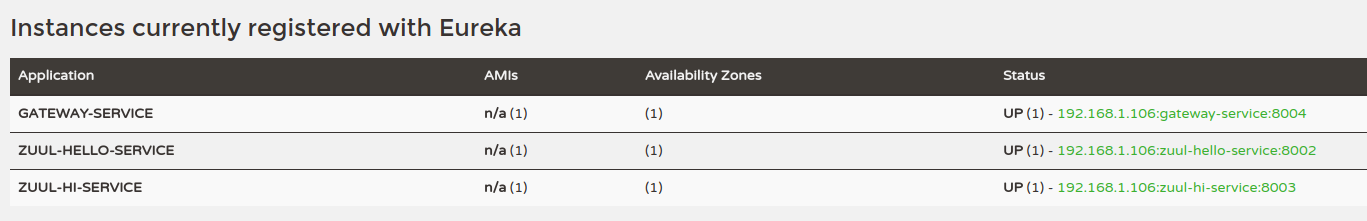
具体参考ZuulRoute类的解说.

继续对参数进行解说,path表示访问必须以该属性值作为前缀,才能访问到serviceId对外提供的服务中,比如必须以hello-service作为前缀,\*\*\*\*表示该前缀下面的所有对外提供的服务,才能访问到serviceId提供的服务.此时serviceId必须对应相应module的spring.application.name的值.

完成相关配置后,快速实现两个service,然后使用zuul的地方进行访问它们,就可以得到进行路由调用的结果.可以使用zuul的默认路由规则,还可以使用正则表达式进行自定义路由规则.

现在通过zuul来访问hello和hi对外提供的方法,开启注册中心,hello,hi,gateway的服务,截图如下:

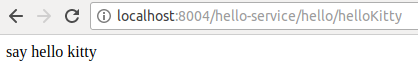


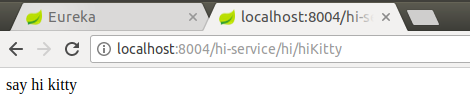


此时我们通过zuul来访问,hello对外提供的helloKitty方法,访问的路径是

localhost:8004/hello-service/hello/helloKitty即可.

其中localhost:8004是zuul的项目访问,hello-service是路由访问策略,hello/helloKitty是hello对外提供的服务内容.





以上就是简单的zuul案例

请注意:在使用zuul的时候,支持ribbon的使用,因此需要设置最大化的全局超时时间,即设置全局的ribbon和hystrix的超时时间,保证包含内容的超时/重试/短熔/降级的时间.

## 1.3使用zull2

在使用zuul的过滤器,可以进行安全认证/权限认证/身份识别等功能,只需要自定义Filter后继承ZoolFilter接口即可.可以通过重写其filterType,filterOrder,shouldFilter以及run方法,可以对当前过滤器进行详细的配置.

filterType表示filter执行的时候,其value值含义如下所示:

pre:在请求被路由之前被调用;

routing:在请求被路由中被调用;

post:在请求被路由之后被调用;

error:处理请求发生错误时被调用;

filterOrder表示执行的优先级,值越小表示优先级越高;

shouldFilter则表示该filter是否需要执行.

拦截器案例如下:

package com.qds.optimize.zuul.gateway;

import com.netflix.zuul.ZuulFilter;

import com.netflix.zuul.context.RequestContext;

import org.apache.commons.lang3.StringUtils;

import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;

import java.util.Map;

/\*\*

\* 设置签名过滤器<br/>

\* 使用component注解表示需要注入到spring容器中

\*

\* @author neutron

\*/

@Component

public class AutographFilter extends ZuulFilter {

/\*\*

\* pre:在请求被路由之前被调用

\* routing:在请求被路由中被调用

\* post:在请求被路由之后被调用

\* error:处理请求发生错误时被调用

\*/

@Override

public String filterType() {

return "pre";

}

/\*\*

\* filterOrder表示执行的优先级,值越小表示优先级越高<br/>

\* 多个filter的执行顺序控制

\*/

@Override

public int filterOrder() {

return 0;

}

/\*\*

\* shouldFilter则表示该filter是否需要执行<br/>

\* true表示需要被执行<br/>

\* false表示不需要执行<br/>

\*/

@Override

public boolean shouldFilter() {

return true;

}

/\*\*

\* 执行Filter真正执行的方法<br/>

\* http请求,run方法,RequestContext

\*/

@Override

public Object run() {

// 根据netflix的请求上下文对象来获取请求

RequestContext context = RequestContext.getCurrentContext();

// 获取request请求对象

HttpServletRequest request = context.getRequest();

// 获取实际请求参数数据

Map<String, String[]> params = request.getParameterMap();

// 对数据进行签名验证

// 比如获取token数据

String token = request.getHeader("x-auth-token");

if (StringUtils.isBlank(token)) {

// 设置返回值信息

context.setSendZuulResponse(false);

// 设置返回值状态码

context.setResponseStatusCode(401);

// 设置返回值body

context.setResponseBody("we need token value");

// 返回null值即可

return null;

} else {

// 如果token是加密,需要解密处理,解密后数据继续处理,

// 然后传递到后面请求处理中

context.addZuulRequestHeader("x-auth-token", token);

}

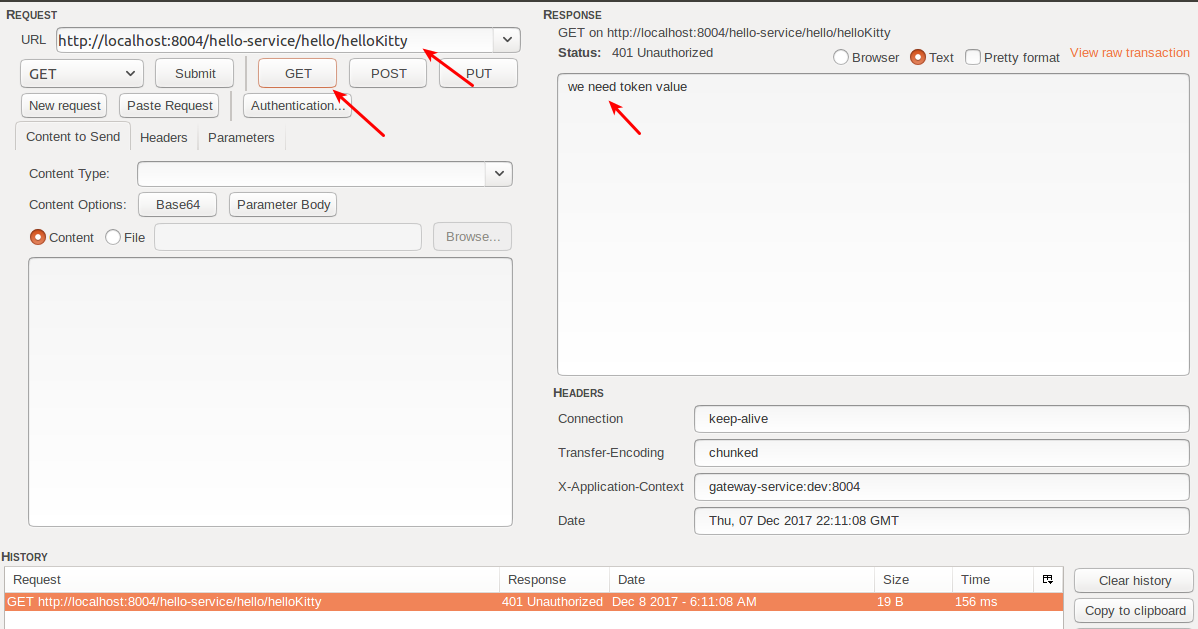
// 如果最后处理成功,返回上线文对象

return context;

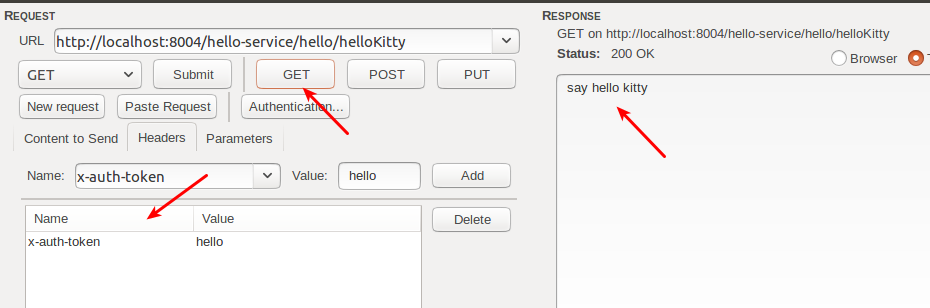
}

}

使用火狐的HttpRequester进行请求访问:



设置token数据:



表示数据处理成功.

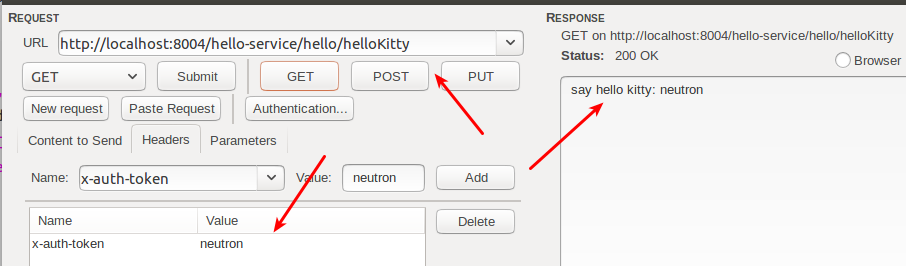
@RequestMapping(value = "/helloKitty", method = {RequestMethod.GET})

public String helloKitty (@RequestHeader("x-auth-token") String token) {

System.err.println("say hello kitty");

return "say hello kitty: " + token;

}



## 1.4不经过zuul进行文件上传

使用zuul进行上传,那么可以不走zuul的dispatchservlet,如何实现的呢?使用/zuul/{spring.application.name}/path,即可以绕开zuul的dispatchservlet.

首先查看controller内容,具体查看hello的module中内容:

package com.qds.optimize.zuul.hello.web;

import org.apache.commons.io.FileUtils;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import java.io.File;

import java.io.IOException;

/\*\*

\* 文件上传对外提供的服务

\*

\* @author neutron

\*/

@RestController

@RequestMapping(value = "/hello")

public class FileApi {

/\*\*

\* 在spring boot下的resources目录下创建METE-INF目录,然后在创建resources目录<br/>

\* 该目录下的文件或者页面可以直接访问,这是servlet3.0规范

\* 如果想走zuul,但是不想走zuul的分发器,那么只需要在请求前面加上/zuul/,<br/>

\* 不然配置zuul,直接调用其他的服务,以及需要配置上传文件的配置

\*/

@RequestMapping(value = "/uploadFile",

method = {RequestMethod.POST})

public String uploadFile (@RequestParam("file") MultipartFile file) {

System.err.println("文件名称:" + file.getOriginalFilename());

try {

FileUtils.writeByteArrayToFile(new File("/tmp"), file.getBytes());

} catch (IOException e) {

e.printStackTrace();

return "upload file fails";

}

return "upload file success";

}

}

界面参考如下:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

</head>

<body>

<div>

<form action="http://localhost:8004/hello-service/upload"

enctype="multipart/form-data" method="post">

<input type="file" name="file" /> <br/>

<input type="submit" value="普通上传" />

</form>

</div>

<div>

<form action="http://localhost:8004/zuul/hello-service/upload"

enctype="multipart/form-data" method="post">

<input type="file" name="file" /> <br/>

<input type="submit" value="绕过zuul的dispatchServlet进行上传" />

</form>

</div>

</body>

</html>

## 1.5 zuul断熔处理

zuul内容已经集成hystrix组件,具有错误断熔的功能,那么如何自定义断熔后的输出结果呢?比如状态吗,响应码,响应报文等内容.只需要自定义类实现ZuulFallbackProvider接口即可,重写下面的方法:

public String getRoute() 自定义熔断的方法

public ClientHttpResponse fallbackResponse() 自定义失败响应信息

我们需要在zuul的gateway项目中增加zuul的断路器内容,首先根据不同的spring.application.name来创建不同的断熔的降级服务.

参考zuul-gateway下的com.qds.optimize.zuul.hystrix.fallback

package com.qds.optimize.zuul.hystrix.fallback;

import org.springframework.cloud.netflix.zuul.filters.route.ZuulFallbackProvider;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.client.ClientHttpResponse;

import org.springframework.stereotype.Component;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStream;

/\*\*

\* zuul的自定义断熔实现类和自定义响应信息内容

\*

\* @author neutron

\*/

@Component

public class HelloServiceFallBack implements ZuulFallbackProvider {

/\*\*

\* 设置服务的断熔名称<br/>

\* 就是spring.application.name的值

\*/

@Override

public String getRoute() {

return "hello-service";

}

@Override

public ClientHttpResponse fallbackResponse() {

return new ClientHttpResponse(){

/\*\*

\* 返回响应头信息

\*/

@Override

public HttpHeaders getHeaders() {

HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.APPLICATION\_JSON);

return headers;

}

/\*\*

\* 返回响应主体信息

\*/

@Override

public InputStream getBody() throws IOException {

return new ByteArrayInputStream("服务不可用,请稍后重试".getBytes());

}

/\*\*

\* 自定义响应状态码,比如200,或者404一类

\*/

@Override

public HttpStatus getStatusCode() throws IOException {

return HttpStatus.OK;

}

/\*\*

\* 返回值是getStatusCode中的value值

\*/

@Override

public int getRawStatusCode() throws IOException {

return this.getStatusCode().value();

}

/\*\*

\* 状态文本信息

\*/

@Override

public String getStatusText() throws IOException {

return this.getStatusCode().getReasonPhrase();

}

/\*\*

\* 关闭内容

\*/

@Override

public void close() {

}

};

}

}

方法的含义和内容,请参考代码中的案例.