1. 导入maven坐标

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.0.1.RELEASE</version>  
 <relativePath />  
</parent>  
  
<properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Finchley.SR1</spring-cloud.version>  
</properties>  
  
<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

1. 建立Eureka服务

建立一个独立的子模块

取名为eureka

导入坐标，导入eureka的坐标，并且导入Ribbon的坐标，Ribbon是负载均衡的,导入spring的安全框架坐标，配置eureka对应的访问权限

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

<!--spring 的安全中心-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>  
</dependency>  
<dependency>  
 <groupId>com.squareup.okhttp3</groupId>  
 <artifactId>okhttp</artifactId>  
</dependency>

</dependencies>

并且写入以下代码：

package com.sc;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@SpringBootApplication  
@EnableEurekaServer  
public class EurekaApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaApplication.class, args);  
 }  
}

新版本的eureka要求增加一个对应的类在eureka服务中，我们起名为WebSecurityConfig，类的信息如下：

package com.sceureka;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
  
@EnableWebSecurity  
@Configuration  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.csrf().disable();  
 super.configure(http);  
 }  
}

配置完成后，需要在yml文件中加入对应的配置，如下：

*#Eureka服务注册中心  
#作为一个实例也包括定期心跳到注册表（通过客户端的serviceUrl），  
#默认持续时间为30秒。在实例，服务器和客户端在其本地缓存中都具有相同的元数据（因此可能需要3个心跳）之前，  
#客户端才能发现服务。您可以使用eureka.instance.leaseRenewalIntervalInSeconds更改期限，这将加快客户端连接到其他服务的过程。  
#在生产中，最好坚持使用默认值，因为服务器内部有一些计算可以对租赁更新期进行假设。  
#  
#  
#***server:  
 port:** 9000  
**spring:  
 application:  
 name:** sc-euruka  
 *# 配置eureka的安全中心* **security:  
 user:  
 name:** 123qwe  
 **password:** 123qwe  
*# Eureka的默认注册地址***eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://${**spring.security.user.name**}:${**spring.security.user.password**}@${**eureka.instance.hostname**}:${**server.port**}/eureka/  
*# http://name:pwd@localhost:9001/eureka  
 # 禁止eureka向自己注册，eureka会自己向自己注册服务，如果禁止那么就是单机版* **register-with-eureka: false  
 fetch-registry: false***# 初始化Eureka名称，如果不初始化名称，那么就以application的name为名称* **instance:  
 hostname:** localhost  
 **instanceId:** ${**spring.application.name**}  
 **server:***# eureka 关闭自我保护机制，如果是网络原因，那么eureka可以有自己的缓存能够维持继续访问* **enable-self-preservation: false**

解释：

Server.port:应用启动的端口号

Spring.application.name:应用名称

Spring.security.user.name:登入或注册eureka的用户名称

Spring.security.user.password:登入或注册eureka的密码

Eureka.client.serviceUrl.defaultZone:注册的eureka地址

Register-with-eureka:是否注册到eureka上

Fetch.registery:是否从eureka上获取注册客户端的信息

Enable.self.preservation:是否启动自我保护机制

Ip-address:如果配置地址，那么注册到的eureka中心就是对应的ipi了

Prefer-ip-address:注册到eureka上的地址是否展示

1. 配置完成eureka后，我们写入两个客户端

3.1第一个客户端为order模块

导入pom.xml文件，pom文件的坐标有三个，一个是eureka的客户端，一个是ribbon的负载均衡，一个是hystrix的熔断器

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <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-ribbon</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.squareup.okhttp3</groupId>  
 <artifactId>okhttp</artifactId>  
 </dependency>  
</dependencies>

新建一个类为OrderApplication.java

其中的主类为信息为：

package com.sc;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
import org.springframework.cloud.netflix.hystrix.EnableHystrix;  
import org.springframework.cloud.openfeign.EnableFeignClients;  
  
@SpringBootApplication  
@EnableDiscoveryClient  
@EnableHystrix  
@EnableFeignClients  
public class OrderApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(OrderApplication.class, args);  
 }  
}

注解：一个是注册到eureka上，一个是启动熔断器，一个是使用feign客户端。

可进行重试和相应的次数：配置如下：

package com.sc;  
  
import feign.Retryer;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class FeignConfig {  
 @Bean  
 public Retryer feignRetryer () {  
 return new Retryer.Default(100, 1000, 2);  
 }  
}

new Retryer.Default(100, 1000, 2); 100是超时的毫秒数，1000最大毫秒数，2尝试2次

使用方式，在使用注解@FeignClient(value=””, configureation=FeignConfig.class)即可

同时配置RestTemplate的注解，在主类对应的方法下，提供如下配置，并且在pom.xml中导入对应的okhttp远程访问，对应的restTemplate的访问就可以进行使用okhttp进行访问了。

所以配置如下（如果需要的话）:

Pom.xml文件配置

<dependency>  
 <groupId>com.squareup.okhttp3</groupId>  
 <artifactId>okhttp</artifactId>  
</dependency>

对应的主类加入：

@Bean  
@LoadBalanced

@Primary  
RestTemplate restTemplate () {  
 return new RestTemplate(new OkHttp3ClientHttpRequestFactory());  
}

1. 以同样的方式建立一个商品端：
2. 我们在双方的项目下，建立一个controller提供对外的访问。

package com.sc.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cloud.client.discovery.DiscoveryClient;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class OrderController {  
  
 @Autowired  
 DiscoveryClient discoveryClient;  
 @Autowired  
 private OrderInterface orderInterface;  
  
 @GetMapping("/order/{id}")  
 public String goods (@PathVariable("id") String id) {  
  
 return orderInterface.goodsById(id);  
 }  
  
}

以上是order的controller

接下来是feign客户端的接口，也就是order中调用goods中的对应的类

package com.sc.controller;  
  
import com.sc.FeignConfig;  
import org.springframework.cloud.openfeign.FeignClient;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
  
@FeignClient(value = "goodsApplication", configuration = FeignConfig.class)  
public interface OrderInterface {  
  
 @GetMapping("/goods/{id}")  
 String goodsById (@PathVariable("id") String id);  
  
}

goods中有一个相当于springmvc的映射，直接映射对应即可。

1. RestTemplate结合Hystrix
2. Feign结合Hystrix，feign结合它的话，首先在yml文件中进行配置，配置如下：

当我们调用失败的时候，进行对应的调用错误的方法，最开始要加如对应的配置文件信息，为：

**feign:  
 hystrix:  
 enabled:** true

首先建立一个实现feign接口调用其他客户端的实现类，具体的话，如下：

首先feign调用的接口如下：

package com.sc.controller;  
  
import com.sc.FeignConfig;  
import org.springframework.cloud.openfeign.FeignClient;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
  
@FeignClient(value = "goodsApplication", configuration = FeignConfig.class, fallback = ErrorOrder.class)  
public interface OrderInterface {  
  
 @GetMapping("/goods/{id}")  
 String goodsById (@PathVariable("id") String id);  
  
}

在此处，fallback是指的回调的类，那么我们就要建立对应的类的信息，如下;

package com.sc.controller;  
  
import org.springframework.stereotype.Component;  
  
@Component  
public class ErrorOrder implements OrderInterface {  
 @Override  
 public String goodsById(String id) {  
 return "error";  
 }  
}

此类要放到容器中，使用对应的注解，同时要实现feign调用的客户端的接口即可。然后再实现对应的方法的内容就OK了。当我们调用失败的时候，我们就可以对应的调用此类的对应的方法了。

1. 创建一个Zuul网关。网关主要是用来拦截请求，并发送到对应的服务上。

首先创建一个服务：

导入pom.xml文件，如下：

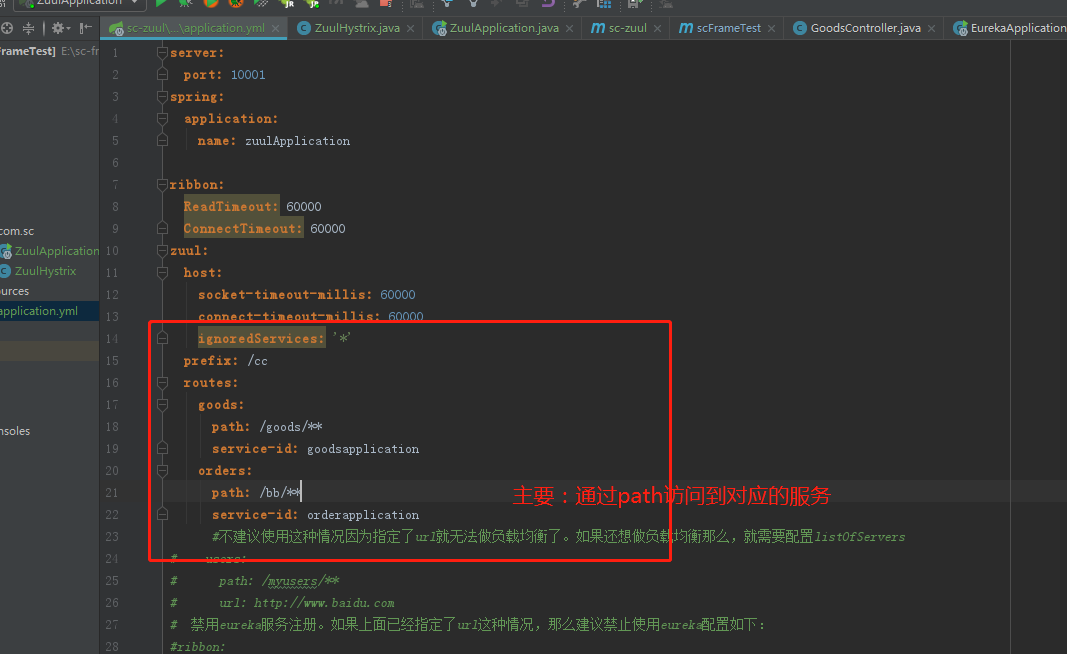
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-zuul</artifactId>  
 </dependency>  
 <dependency>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

导入完成pom.xml后，接下来我们写一个主函数，如下：

package com.sc;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;  
  
*/\*\*  
 \** ***@Author*** *ZhPJ  
 \** ***@Date*** *2018/11/2 00029:21  
 \** ***@Version*** *1.0  
 \** ***@Description:*** *\*/*@SpringBootApplication  
@EnableZuulProxy  
@EnableDiscoveryClient  
public class ZuulApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(ZuulApplication.class, args);  
 }  
  
}

完成后，对应的application.yml文件配置如下：

**server:  
 port:** 10001  
**spring:  
 application:  
 name:** zuulApplication  
  
**ribbon:  
 ReadTimeout:** 60000  
 **ConnectTimeout:** 60000  
**zuul:  
 host:  
 socket-timeout-millis:** 60000  
 **connect-timeout-millis:** 60000  
 **ignoredServices:** '\*'  
 **prefix:** /cc  
 **routes:  
 goods:  
 path:** /goods/\*\*  
 **service-id:** goodsapplication  
 **orders:  
 path:** /bb/\*\*  
 **service-id:** orderapplication  
 *#不建议使用这种情况因为指定了url就无法做负载均衡了。如果还想做负载均衡那么，就需要配置listOfServers  
# users:  
# path: /myusers/\*\*  
# url: http://www.baidu.com  
# 禁用eureka服务注册。如果上面已经指定了url这种情况，那么建议禁止使用eureka配置如下：  
#ribbon:  
# eureka:  
# enabled: false  
# 配置Ribbon的轮训机制 users应该对应的是routes中的users，这样完成了负载均衡  
#users:  
# ribbon:  
# listOfServers: example.com,google.com  
# 扼杀模式和本地前进，在此配置中，我们可以看到url:后面跟到了forward:/secord 那么，所有的/third/\*\* 访问都转发到了/3rd 路径上  
#zuul:  
# routes:  
# first:  
# path: /first/\*\*  
# url: http://first.example.com  
# second:  
# path: /second/\*\*  
# url: forward:/second  
# third:  
# path: /third/\*\*  
# url: forward:/3rd***hystrix:  
 command:  
 default:  
 execution:  
 isolation:  
 thread:  
 timeoutInMilliseconds:** 60000  
**eureka:  
 client:  
 service-url:  
 defaultZone:** http://123qwe:123qwe@127.0.0.1:9000/eureka/  
 **register-with-eureka: true  
 fetch-registry: true  
 instance:  
 prefer-ip-address: true**



完成后，对应的zuul访问失败时，回调对应的返回信息，如下：

新建一个类，并实现一个接口，实现对应的方法：如下：

package com.sc;  
  
import org.springframework.cloud.netflix.zuul.filters.route.FallbackProvider;  
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;  
  
@Component  
public class ZuulHystrix implements FallbackProvider {  
 @Override  
 public String getRoute() {  
 return "\*";  
 }  
  
 @Override  
 public ClientHttpResponse fallbackResponse(String route, Throwable throwable) {  
 return new ClientHttpResponse() {  
 @Override  
 public HttpStatus getStatusCode() throws IOException {  
 return HttpStatus.*OK*;  
 }  
  
 @Override  
 public int getRawStatusCode() throws IOException {  
 return 200;  
 }  
  
 @Override  
 public String getStatusText() throws IOException {  
 return "OK";  
 }  
  
 @Override  
 public void close() {  
  
 }  
  
 @Override  
 public InputStream getBody() throws IOException {  
 return new ByteArrayInputStream("fallback".getBytes());  
 }  
  
 @Override  
 public HttpHeaders getHeaders() {  
 HttpHeaders headers = new HttpHeaders();  
 headers.setContentType(MediaType.*APPLICATION\_JSON*);  
 return headers;  
 }  
 };  
 }  
}

调用失败的时候，会自动调用getBody方法，实现回显

可以设置对应的返回值什么的。

1. 总线配置