### Eureka服务注册与发现

**书签：**

[(1)Eureka服务启动](#_Eureka服务启动)

[(2)Eureka客户端注册服务](#_Eureka客户端注册服务)

[(3)Eureka客户端发现服务](#_Eureka客户端发现服务)

[(4)Eureka服务集群](#_Eureka服务集群)

[(5)Eureka服务其他细节](#_Eureka服务其他细节)

###### Eureka服务启动

新建项目eureka-server，

pom.xml 配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>cn.ywj</groupId>  <artifactId>eureka-provider</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>eureka-provider</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.5.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </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>   <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.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>   <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>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

application.yml 配置如下：

|  |
| --- |
| #应用名 spring:  application:  name: eureka-server # 端口号 server:  port: 8080 # eureka配置 eureka:  instance:  hostname: localhost  client:  fetch-registry: false # 禁止到Eureka器中获取注册信息  register-with-eureka: false # 禁止将自己作为客户端注册信息到Eureka服务器  serviceUrl:  defaultZone: http://127.0.0.1:8080/eureka/ # 告诉别人服务的注册和发现地址 |

启动类上加入注解@EnableEurekaServer 即可：

|  |
| --- |
| package cn.ywj.eurekaserver;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  @SpringBootApplication @EnableEurekaServer public class EurekaServerApplication {   public static void main(String[] args) {  SpringApplication.run(EurekaServerApplication.class, args);  } } |

启动项目，访问http://127.0.0.1:8080/ 如下：

|  |
| --- |
|  |

OK.

###### Eureka客户端注册服务

新建项目：eureka-provider

pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>cn.ywj</groupId>  <artifactId>eureka-provider</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>eureka-provider</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.5.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </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>   <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.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>   <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>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

application.yml 配置：

|  |
| --- |
| #应用名 spring:  application:  name: eureka-provider # 端口号 server:  port: 8081 # eureka配置 eureka:  instance:  hostname: localhost  client:  serviceUrl:  defaultZone: http://127.0.0.1:8080/eureka/ # 服务的注册地址 |

启动类加入注解：@EnableEurekaClient

|  |
| --- |
| package cn.ywj.eurekaprovider;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication @EnableEurekaClient public class EurekaProviderApplication {   public static void main(String[] args) {  SpringApplication.run(EurekaProviderApplication.class, args);  } } |

启动eureka-server项目，再启动eureka-provider，访问http://127.0.0.1:8080/ 如下图：

Instances currently registered with Eureka 有刚才eureka-provider项目的注册信息

|  |
| --- |
|  |

###### Eureka客户端发现服务

新那一个项目：eureka-consumer

pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>cn.ywj</groupId>  <artifactId>eureka-consumer</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>eureka-consumer</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.5.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </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>   <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.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>   <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>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

application.yml 配置

|  |
| --- |
| #应用名 spring:  application:  name: eureka-consumer # 端口号 server:  port: 8082 # eureka配置 eureka:  instance:  hostname: localhost  client:  serviceUrl:  defaultZone: http://127.0.0.1:8080/eureka/ # 服务的发现地址 |

修改eureka-provider项目，提供一个接口：

|  |
| --- |
| package cn.ywj.eurekaprovider;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.client.discovery.EnableDiscoveryClient; import org.springframework.cloud.netflix.eureka.EnableEurekaClient; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController;  @SpringBootApplication @EnableEurekaClient @RestController public class EurekaProviderApplication {   @GetMapping("/t")  public Object t(){// 提供一个接口  return "msg from provider";  }   public static void main(String[] args) {  SpringApplication.run(EurekaProviderApplication.class, args);  } } |

eureka-consumer项目启动类修改：

|  |
| --- |
| package cn.ywj.eurekaconsumer;  import com.netflix.appinfo.InstanceInfo; import com.netflix.discovery.EurekaClient; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.context.annotation.Bean; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import org.springframework.web.client.RestTemplate;  @SpringBootApplication @RestController public class EurekaConsumerApplication {   @Bean  public RestTemplate getRestTemplate () {// 注入一个RestTemplate  return new RestTemplate() ;  }   @Autowired  private RestTemplate restTemplate;   @Autowired  private EurekaClient eurekaClient;   @RequestMapping("/t")  public Object t(){   InstanceInfo instanceInfo = eurekaClient.getNextServerFromEureka("EUREKA-PROVIDER", false);// 通过服务名可以拿到地址  String str = restTemplate.getForObject(instanceInfo.getHomePageUrl()+"/t", String.class);  return str;  }   public static void main(String[] args) {  SpringApplication.run(EurekaConsumerApplication.class, args);  } } |

分别启动项目：eureka-server、eureka-provider、eureka-consumer 访问：<http://localhost:8082/t> 即可获取信息：

msg from provider

###### Eureka服务集群

只需修改eureka-server的application.yml中的端口号和defaultZone即可，idea可以一个项目多次启动，所以可以不用创建多个server项目，每次修改application.yml后重新启动项目即可

先配置一下hosts文件：

|  |
| --- |
| 127.0.0.1 eureka0  127.0.0.1 eureka1  127.0.0.1 eureka2 |

eureka-server项目下创建3个yml:

application-yml8080.yml

application-yml8180.yml

application-yml8280.yml

配置中除了port、hostname、defaultZone 相对应的变化，其他的不变

application-yml8080.yml

|  |
| --- |
| #应用名 spring:  application:  name: eureka-server  profiles:  active: yml8080 # 端口号 server:  port: 8080 # eureka配置 eureka:  instance:  hostname: eureka0  client:  fetch-registry: false # 禁止到Eureka器中获取注册信息  register-with-eureka: false # 禁止将自己作为客户端注册信息到Eureka服务器  serviceUrl:  defaultZone: http://eureka1:8180/eureka/,http://eureka2:8280/eureka/ |

application-yml8180.yml

|  |
| --- |
| #应用名 spring:  application:  name: eureka-server  profiles:  active: yml8180 # 端口号 server:  port: 8180 # eureka配置 eureka:  instance:  hostname: eureka1  client:  fetch-registry: false # 禁止到Eureka器中获取注册信息  register-with-eureka: false # 禁止将自己作为客户端注册信息到Eureka服务器  serviceUrl:  defaultZone: http://eureka0:8080/eureka/,http://eureka2:8280/eureka/ |

application-yml8280.yml

|  |
| --- |
| #应用名 spring:  application:  name: eureka-server  profiles:  active: yml8280 # 端口号 server:  port: 8280 # eureka配置 eureka:  instance:  hostname: eureka2  client:  fetch-registry: false # 禁止到Eureka器中获取注册信息  register-with-eureka: false # 禁止将自己作为客户端注册信息到Eureka服务器  serviceUrl:  defaultZone: http://eureka0:8080/eureka/,http://eureka1:8180/eureka/ |

然后在idea里设置program arguments : --spring.profiles.active=yml8280 (三个启动器，分别对应)

|  |
| --- |
|  |

然后分别启动这3个启动器即可，访问：<http://eureka0:8080/>

|  |
| --- |
|  |

接着eureka-provider项目yml修改defaultZone为三个地址即可：

|  |
| --- |
| defaultZone: http://eureka0:8080/eureka/,http://eureka1:8180/eureka/,http://eureka2:8280/eureka/ |

###### Eureka服务其他细节

1、eureka-provider显示Ip:

|  |
| --- |
| eureka:  instance:  prefer-ip-address: true |

1. 自我保护功能关闭:

|  |
| --- |
| eureka:  server:  enable-self-preservation: false #关闭 |

### Ribbon负载均衡（客户端）

**书签：**

[(1)@LoadBalanced 负载均衡](#_@LoadBalanced 负载均衡)

[(2)Ribbon自带的负载规则](#_Ribbon自带的负载规则)

[(3)Ribbon自定义负载规则](#_Ribbon自定义负载规则)

###### @LoadBalanced 负载均衡

复制项目eureka-provider，生成eureka-provider-2，修改eureka-provider-2的端口成8091即可，请求返回的信息改变一下：

|  |
| --- |
| @GetMapping("/t") public Object t(){  return "msg from provider-2";// 后面加个2，方便对比 } |

修改eureka-consumer项目启动类：主要是加上注解@LoadBalanced;请求服务提供者用服务名

|  |
| --- |
| package cn.ywj.eurekaconsumer;  import com.netflix.appinfo.InstanceInfo; import com.netflix.discovery.EurekaClient; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.client.loadbalancer.LoadBalanced; import org.springframework.cloud.netflix.ribbon.RibbonClient; import org.springframework.context.annotation.Bean; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import org.springframework.web.client.RestTemplate;  @SpringBootApplication @RestController public class EurekaConsumerApplication {   @Bean  @LoadBalanced  public RestTemplate getRestTemplate () {// 注入一个RestTemplate  return new RestTemplate() ;  }   @Autowired  private RestTemplate restTemplate;   //@Autowired  //private EurekaClient eurekaClient;   @RequestMapping("/t")  public Object t(){   //InstanceInfo instanceInfo = eurekaClient.getNextServerFromEureka("EUREKA-PROVIDER", false);// 通过服务名可以拿到地址  String str = restTemplate.getForObject("http://EUREKA-PROVIDER/t", String.class);// 直接用服务名请求  return str;  }   public static void main(String[] args) {  SpringApplication.run(EurekaConsumerApplication.class, args);  } } |

启动eureka-consumer项目，访问：<http://localhost:8082/t> 返回的结果有时是：msg from provider，有时是：msg from provider-2，轮来轮去。

###### Ribbon自带的负载规则

|  |
| --- |
| package cn.ywj.eurekaconsumer;  import com.netflix.loadbalancer.\*; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;  @Configuration public class RibbonConfiguration {   @Bean  public IRule ribbonRule() {  return new RoundRobinRule();// 轮循算法   /\*  相关算法   一、RoundRobinRule ：系统默认的规则，通过简单地轮询服务列表来选择服务器   二、AvailabilityFilteringRule 该规则会忽略以下服务器   1、无法连接的服务器 在默认情况下，如果 次连接失败，该服务器将会被置为“路”的状态，该状态将持续 30 秒；如果再次连接失败， “短路”状态的持续时间  将会以几何级数增加。可以通过修改 niws.loadbalancer. <clientName>.connectionFailureCountThreshold  属性，来配置连接失败的次数   2、并发数过高的服务器：如果连接到该服务器的并发数过高，也会被这个规则忽略，可以通过修改＜ lientName> .ribbon.ActiveConnectionsLirnit 属性  来设最高并发数   三、WeightedResponseTimeRule ：为每个服务器赋予一个权重值 服务器的响应时间越长，该权重值就越少，这个规则会随机选择服务器，权重值有可能会决定服务器选择   四、ZoneAvoidanceRule ：该规则以区域、可用服务器为基础进行服务器选择。使用 Zone对服务器进行分类，可以理解为机架或者机房   五、BestAvailableRule ：忽略“短路”的服务器，并选择并发数较低的服务器。   六、RandomRule ：顾名思义，随机选择可用的服务器。   七、RerRule 含有重试的选择逻辑，如果使用 RoundRobinRule 选择的服务器无法连接，那么将会重新选择服务器。\*/  } } |

###### Ribbon自定义负载规则

参考RandomRule.java类，简单自定义一个规则 -- 只访问第一个服务

创建DiyRibbon.java

DiyRibbon.java不能被@ComponentScan扫到!

DiyRibbon.java不能被@ComponentScan扫到!

DiyRibbon.java不能被@ComponentScan扫到!

|  |
| --- |
| package diy.ribbon;  import com.netflix.client.config.IClientConfig; import com.netflix.loadbalancer.AbstractLoadBalancerRule; import com.netflix.loadbalancer.ILoadBalancer; import com.netflix.loadbalancer.Server;  import java.util.List; import java.util.concurrent.ThreadLocalRandom;  /\*\*  \* 此类不能被@ComponentScan扫到  \*/ public class DiyRibbon extends AbstractLoadBalancerRule {    @Override  public void initWithNiwsConfig(IClientConfig iClientConfig) {   }   @Override  public Server choose(Object o) {  return choose(getLoadBalancer(), o);  }   protected int chooseRandomInt(int serverCount) {  return ThreadLocalRandom.current().nextInt(serverCount);  }  public Server choose(ILoadBalancer lb, Object key) {  if (lb == null) {  return null;  }  List<Server> upList = lb.getReachableServers();  System.out.println("--->"+upList.size());  if(upList != null && upList.size() > 0){  return upList.get(0);// 拿第一个  }  return null;  } } |

创建配置类：RibbonConfiguration.java

|  |
| --- |
| package cn.ywj.eurekaconsumer;  import com.netflix.loadbalancer.\*; import diy.ribbon.DiyRibbon; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;  @Configuration public class RibbonConfiguration {   @Bean  public IRule ribbonRule() {   return new DiyRibbon();// 自定义负载规则   //return new RoundRobinRule();// 轮循算法   /\*  相关算法   一、RoundRobinRule ：系统默认的规则，通过简单地轮询服务列表来选择服务器   二、AvailabilityFilteringRule 该规则会忽略以下服务器   1、无法连接的服务器 在默认情况下，如果 次连接失败，该服务器将会被置为“路”的状态，该状态将持续 30 秒；如果再次连接失败， “短路”状态的持续时间  将会以几何级数增加。可以通过修改 niws.loadbalancer. <clientName>.connectionFailureCountThreshold  属性，来配置连接失败的次数   2、并发数过高的服务器：如果连接到该服务器的并发数过高，也会被这个规则忽略，可以通过修改＜ lientName> .ribbon.ActiveConnectionsLirnit 属性  来设最高并发数   三、WeightedResponseTimeRule ：为每个服务器赋予一个权重值 服务器的响应时间越长，该权重值就越少，这个规则会随机选择服务器，权重值有可能会决定服务器选择   四、ZoneAvoidanceRule ：该规则以区域、可用服务器为基础进行服务器选择。使用 Zone对服务器进行分类，可以理解为机架或者机房   五、BestAvailableRule ：忽略“短路”的服务器，并选择并发数较低的服务器。   六、RandomRule ：顾名思义，随机选择可用的服务器。   七、RerRule 含有重试的选择逻辑，如果使用 RoundRobinRule 选择的服务器无法连接，那么将会重新选择服务器。\*/  } } |

启动类加入注解：@RibbonClient

|  |
| --- |
| package cn.ywj.eurekaconsumer;  import com.netflix.appinfo.InstanceInfo; import com.netflix.discovery.EurekaClient; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.client.ServiceInstance; import org.springframework.cloud.client.loadbalancer.LoadBalanced; import org.springframework.cloud.client.loadbalancer.LoadBalancerClient; import org.springframework.cloud.netflix.ribbon.RibbonClient; import org.springframework.context.annotation.Bean; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import org.springframework.web.client.RestTemplate;  import java.net.MalformedURLException; import java.net.URI;  @SpringBootApplication @RestController @RibbonClient(name = "eureka-provider", configuration = RibbonConfiguration.class)// name是服务注册者 public class EurekaConsumerApplication {   @Bean  @LoadBalanced  public RestTemplate getRestTemplate () {// 注入一个RestTemplate  return new RestTemplate() ;  }   @Autowired  private RestTemplate restTemplate;   //@Autowired  //private EurekaClient eurekaClient;   @RequestMapping("/t")  public Object t(){   //InstanceInfo instanceInfo = eurekaClient.getNextServerFromEureka("EUREKA-PROVIDER", false);// 通过服务名可以拿到地址  String str = restTemplate.getForObject("http://EUREKA-PROVIDER/t", String.class);// 直接用服务名请求  return str;  }   public static void main(String[] args) {  SpringApplication.run(EurekaConsumerApplication.class, args);  } } |

### Feign声明式服务调用客户端//接口方式

**书签：**

**[1、小小例子](#_小小例子)**

**[2、Get多个请求参数处理](#_Get多个请求参数处理)**

###### 小小例子

新建项目：eureka-consumer-feign

pom核心配置：

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId> </dependency> |

创建接口：FeignTest.java

|  |
| --- |
| package cn.ywj.eurekaconsumerfeign;  import org.springframework.cloud.openfeign.FeignClient; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RequestMethod;  @FeignClient("eureka-provider")// 指定要访问的服务提供者 public interface FeignTest {   @RequestMapping(method = RequestMethod.GET, value = "/t")// 服务提供的请求地址  public String t(); } |

项目启动类：EurekaConsumerFeignApplication.java, 核心注解：@EnableFeignClients

|  |
| --- |
| package cn.ywj.eurekaconsumerfeign;  import com.netflix.loadbalancer.IRule; import com.netflix.loadbalancer.RandomRule; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.openfeign.EnableFeignClients; import org.springframework.context.annotation.Bean; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController;  @SpringBootApplication @RestController @EnableFeignClients public class EurekaConsumerFeignApplication {   @Autowired  private FeignTest feignTest;   @RequestMapping("/t")  public String t(){  return feignTest.t();  }   public static void main(String[] args) {  SpringApplication.run(EurekaConsumerFeignApplication.class, args);  } } |

运行项目访问，一样可以得到结果。Feign集成了Ribbon,也可以注入IRule使用各种负载均衡功能。

###### Get多个请求参数处理

单参数请求参数时，没什么问题，多参数时，有会问题，，，

当FeignTest接口中添加新的方法

|  |
| --- |
| @RequestMapping(method = RequestMethod.GET, value = "/t2") public User t2(User user); |

然后Controller请求

|  |
| --- |
| @RequestMapping("/t2") public String t2(User user){  User user1 = feignTest.t2(user);  return user1 == null ? "null" : user1.getName()+":"+user1.getAge(); } |

<http://127.0.0.1:8083/t2?name=ywj&age=17>

此时会报错：

|  |
| --- |
| content:{"timestamp":"2018-10-10T09:14:43.547+0000","status":405,"error":"Method Not Allowed","message":"Request method 'POST' not supported","path":"/t2"} |

处理方法：

1. 第一种：把user参数拆分成name,age

|  |
| --- |
| **FeignTest.java**  @RequestMapping(method = RequestMethod.GET, value = "/t3") public User t3(@RequestParam("name") String name, @RequestParam("age") Integer age);  **eureka-provider**  @GetMapping("/t3") public User t3(@RequestParam("name") String name, @RequestParam("age") Integer age){  User user = new User();  user.setName(name);  user.setAge(age);  return user; } |

1. 第二种：换成Post请求

|  |
| --- |
| **FeignTest.java**  @RequestMapping(method = RequestMethod.POST, value = "/t5") public User t5(User user);  **eureka-provider**  @PostMapping("/t5") public User t5(@RequestBody User user){  return user; } |

### Hystrix服务容错 - 服务熔断降级

**书签：**

**[1、小小例子](#_小小例子_1)**

**[2、指定Class回退](#_指定Class回退)**

###### 小小例子

A服务访问B服务，B服务可能超时或者响应异常，A服务停止对B的访问（1小段时间），调用自定义Back方法，而不等死或抛异常，堵资源。

pom.xml

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId> </dependency> |

接着eureka-consumer-feign项目时行修改：

启动类EurekaConsumerFeignApplication.java主要添加注解@EnableCircuitBreaker和@HystrixCommand

|  |
| --- |
| package cn.ywj.eurekaconsumerfeign;  import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker; import org.springframework.cloud.openfeign.EnableFeignClients; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController;  @SpringBootApplication @RestController @EnableFeignClients @EnableCircuitBreaker public class EurekaConsumerFeignApplication {   @Autowired  private FeignTest feignTest;   @RequestMapping("/t")  public String t(){  return feignTest.t();  }   @RequestMapping("/t2")  public String t2(User user){  User user1 = feignTest.t2(user);  return user1 == null ? "null" : user1.getName()+":"+user1.getAge();  }   @RequestMapping("/t3")  public String t3(User user){  User user1 = feignTest.t3(user.getName(), user.getAge());  return user1 == null ? "null" : user1.getName()+":"+user1.getAge();  }   @RequestMapping("/t5")  public String t5(User user){  User user1 = feignTest.t5(user);  return user1 == null ? "null" : user1.getName()+":"+user1.getAge();  }   @RequestMapping("/t6")  @HystrixCommand(fallbackMethod = "fallbackMethod")//服务熔断后会调用fallbackMethod方法  public String t6(){  return feignTest.t6();  }   public String fallbackMethod(){  return "t6 from fallbackMethod";  }   public static void main(String[] args) {  SpringApplication.run(EurekaConsumerFeignApplication.class, args);  } } |

访问t6方法，当provider正常跑时，可以返回相关的结果，当把provider关闭后，返回的信息是”t6 from fallbackMethod”.

###### 指定Class回退

创建一个类FeiginHystrixTest.java 实现接口 FeignTest.java

|  |
| --- |
| package cn.ywj.eurekaconsumerfeign;  import org.springframework.stereotype.Component;  @Component public class FeiginHystrixTest implements FeignTest{  @Override  public String t() {  return null;  }   @Override  public User t2(User user) {  return null;  }   @Override  public User t3(String name, Integer age) {  return null;  }   @Override  public User t5(User user) {  return null;  }   @Override  public String t6() { // 回退方法  return "msg from FeiginHystrixTest";  } } |

FeignTest.java上的@FeignClient注解修改如下：

|  |
| --- |
| @FeignClient(value = "eureka-provider",fallback = FeiginHystrixTest.class)// 指定要访问的服务提供者与降级处理方法 |

原来的Controller变成普通的样子

|  |
| --- |
| @RequestMapping("/t6") public String t6(){  return feignTest.t6(); } |

application.yml添加配置

|  |
| --- |
| feign:  hystrix:  enabled: true #声明 Feign 的 Hystrix 支持 |

完毕。

### 网关服务--ZUUL

zuul的主要作用是代理，路由、过滤（如权限），对外提供统一访问入口。

**书签：**

**[1、小小例子](#_小小例子_2)**

**[2、路由规则](#_路由规则)**

**[3、小小过滤](#_小小过滤)**

###### 小小例子

测试路由功能：新建一个项目：gateway-zuul

主要依赖：pom.xml

|  |
| --- |
| <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-zuul</artifactId> </dependency> |

主启动类加入注解：@EnableZuulProxy

|  |
| --- |
| package cn.ywj.gatewayzuul;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.netflix.zuul.EnableZuulProxy;  @SpringBootApplication @EnableZuulProxy public class GatewayZuulApplication {   public static void main(String[] args) {  SpringApplication.run(GatewayZuulApplication.class, args);  } } |

application.yml 修改配置

|  |
| --- |
| #应用名 spring:  application:  name: gateway-zuul # 端口号 server:  port: 8803 # eureka配置 eureka:  instance:  hostname: localhost  prefer-ip-address: true  client:  serviceUrl:  defaultZone: http://eureka0:8080/eureka/,http://eureka1:8180/eureka/,http://eureka2:8280/eureka/   # 上面是普通的基本配置  zuul:  routes:   xxx-provider: # 指定一个路由，名字随便写，最好和服务名一样  path: /ep/\*\* #当访问/ep/\*\* url时，就会请求到下面的serviceId对应的服务的请求中，如请求/ep/abc.do后会请求到eureka-provider的abc.do方法里  serviceId: eureka-provider # 配合上面的path跳转使用 |

完毕。启动访问http://127.0.0.1:8803/ep/t 即可看到请求到结果。

###### 路由规则

1. 多个写法，在配置文件里配置多个routes子节点即可

|  |
| --- |
| zuul:  routes:  xxx-provider:   path: /ep/\*\*  serviceId: eureka-provider  yyy-provider:  path: /ep2/\*\*  serviceId: eureka-provider |

1. 忽略原服务名请求方式，之前可以通过http://127.0.0.1:8803/ep/t 访问，也可以通过http://127.0.0.1:8803/eureka-provider/t 访问，现在想禁止到通过服务名请求的方式，加上配置：ignored-services

|  |
| --- |
| zuul:  routes:   xxx-provider:   path: /ep/\*\*  serviceId: eureka-provider   yyy-provider:  path: /ep2/\*\*  serviceId: eureka-provider   ignored-services: eureka-provider #忽略服务名请求，多个就有逗号,隔开;如果想全部忽略，就用 \* 符号 |

此时http://127.0.0.1:8803/ep/t可以访问，而http://127.0.0.1:8803/eureka-provider/t 不再可以访问

3、访问加前缀： prefix

|  |
| --- |
| zuul:  routes:   xxx-provider:   path: /ep/\*\*  serviceId: eureka-provider   yyy-provider:  path: /ep2/\*\*  serviceId: eureka-provider  prefix: /ywj #访问加前缀，也就是最终访问要在path前面加上 /ywj  ignored-services: eureka-provider #忽略服务名请求，多个就有逗号,隔开;如果想全部忽略，就用 \* 符号 |

此时访问为：[http://127.0.0.1:8803//ywj/ep/t](http://127.0.0.1:8803/ywj/ep/t)

###### 小小过滤

模拟过滤操作，也可以做相关权限功能啦

创建一个类MyFilter继承ZuulFilter

|  |
| --- |
| package cn.ywj.gatewayzuul;  import com.netflix.zuul.ZuulFilter; import com.netflix.zuul.context.RequestContext; import com.netflix.zuul.exception.ZuulException; import org.springframework.http.HttpStatus; import org.springframework.stereotype.Component;  import javax.servlet.http.HttpServletRequest;  /\*\*  \* 参考:http://www.spring4all.com/article/303  \* 参考：https://www.cnblogs.com/qdhxhz/p/9601170.html  \*/ @Component public class MyFilter extends ZuulFilter {   private static String FILTER\_TYPE = "pre";   @Override  public String filterType() {  return FILTER\_TYPE;  }   @Override  public int filterOrder() {  return 0;  }   @Override  public boolean shouldFilter() {   RequestContext requestContext = RequestContext.getCurrentContext();  HttpServletRequest request = requestContext.getRequest();  String url = request.getRequestURI();  // 这里可以放开登录之类的请求，return false  return true;  }   @Override  public Object run() throws ZuulException {   RequestContext requestContext = RequestContext.getCurrentContext();  HttpServletRequest request = requestContext.getRequest();   // 想怎么搞都行  String p = request.getParameter("p");  if(!"ywj".equalsIgnoreCase(p)){   // 权限不对之类的就设置下面信息就不会再访问下去了   // 过滤该请求，不对其进行路由  requestContext.setSendZuulResponse(false);  //返回错误代码  requestContext.setResponseStatusCode(HttpStatus.UNAUTHORIZED.value());  // 返回的描述信息  requestContext.setResponseBody("{\"result\":\"fk\"}");  }  // 一切没问题，ok，放开访问  return null;  } } |

Ok,完毕。