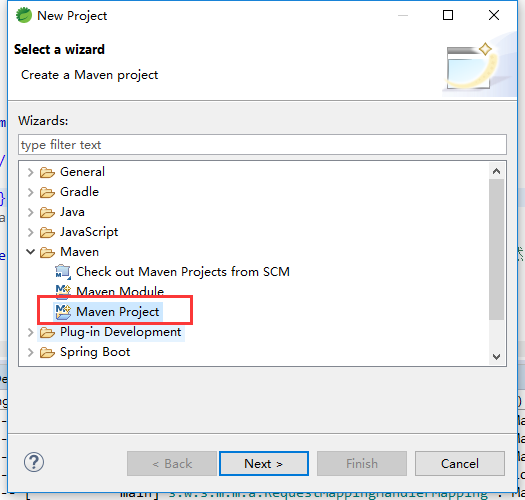
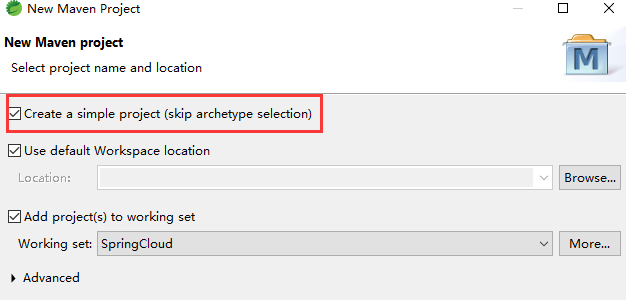
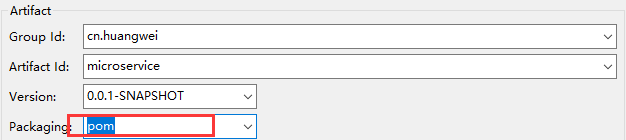
**一、微服务之provider和consumer**

新建父工程





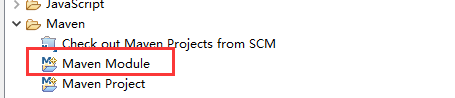


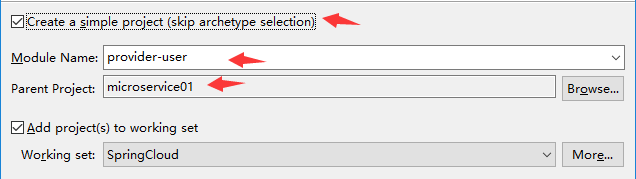
对于父工程，编辑pom文件

|  |
| --- |
| <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-dependencies</artifactId>  <version>1.5.9.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>    <!-- 指定maven编译的版本 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>    <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  </plugins>  </build> |

新建两个子工程

provider-user





编辑provider-user的pom文件

|  |
| --- |
| <!-- 使用springmvc和spring的jar包 -->  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <!-- <version>1.5.9.RELEASE</version> 父工程已经有版本了，所以不用再指定 -->  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-configuration-processor</artifactId>  <optional>true</optional>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>    <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  </dependencies> |

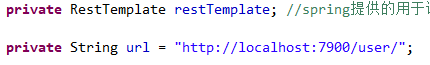
|  |
| --- |
| **新建启动类ProviderUser.java**  package cn.huangwei.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication(scanBasePackages = "cn.huangwei.controller")  public class ProviderUser {  public static void main(String[] args) {  SpringApplication.run(ProviderUser.class, args);  }  }  新建User和UserController  **package** cn.huangwei.entity;  **import** java.util.Date;  **public** **class** User {  **private** **long** id;  **private** Date date;  **public** **long** getId() {  **return** id;  }  **public** **void** setId(**long** id) {  **this**.id = id;  }  **public** Date getDate() {  **return** date;  }  **public** **void** setDate(Date date) {  **this**.date = date;  }  **public** User(**long** id) {  **this**.id = id;  **this**.date = **new** Date();  }    **public** User() {    }    }  **package** cn.huangwei.controller;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** cn.huangwei.entity.User;  @RestController  **public** **class** UserController {    @RequestMapping("/user/{id}")  **public** User getUser(@PathVariable **long** id) {  **return** **new** User(id);  }  } |

创建第二个子工程consumer-order

具体内容如下：

|  |
| --- |
| Pom文件与上一个子工程一致  启动类：  **package** cn.huangwei.app;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.web.client.RestTemplate;  @SpringBootApplication(scanBasePackages = "cn.huangwei.controller")  **public** **class** ConsumerOrder {  @Bean//相当于xml中的bean标签，主要用于调用当前方法获取指定对象  **public** RestTemplate getTemplate() {  **return** **new** RestTemplate();  }    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(ConsumerOrder.**class**, args);  }  }  **package** cn.huangwei.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** org.springframework.web.client.RestTemplate;  **import** cn.huangwei.entity.User;  @RestController  **public** **class** OrderController {  @Autowired  **private** RestTemplate restTemplate; //spring提供的用于访问rest接口的模板对象    **private** String url = "http://localhost:7900/user/";    @RequestMapping("/order/{id}")  **public** User getOrder(@PathVariable **long** id) {  //访问提供者，获取数据  User user = restTemplate.getForObject(url + id, User.**class**);//通过访问rest访问json数据，然后转换为rest  **return** user;  }  }  User也一致，启动的时候两个工程都要启动起来，实现了项目间的相互访问 |

**问题分析：**

此处地址已经写死，可以这么修改

然后再application.yml文件中配置





**二、springcloud之eureka**

如果想使用eureka，需要将我们的两个子项目变为SpringCloud项目

在父工程中添加，springcloud的依赖

|  |
| --- |
| <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Edgware.SR2</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  此时要注意版本问题，如果Springboot的版本为1.x，那么Springcloud的版本也要是1.x，否则会出现下面异常  java.lang.NoSuchMethodError: org.springframework.boot.builder.SpringApplicationBuilder.<init>([Ljava/lang/Class;)V  这个是由于springboot和springcloud版本不匹配的原因 |

新建子工程作为eureka的服务器-eurekaServer

添加相关依赖：

|  |
| --- |
| <dependencies>  <!-- eurekaserver依赖 -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  </dependency>  <!-- 安全依赖 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency>  </dependencies> |

添加application.yml文件

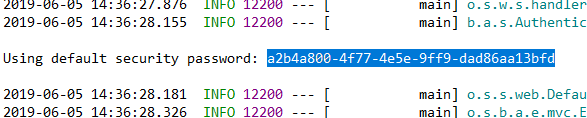
|  |
| --- |
| server:  port: 10001 #eureka server运行再10000端口  #因为当前的eureka是单机的，我们需要一些配置,两者为false，表明我们将服务作为注册中心  #两者为true的时候，表明集群为注册中心  eureka:  client:  register-with-eureka: false  fetch-registry: false  #是一个默认的注册中心地址。配置该选项后，可以在服务中心进行注册  service-url:  defaultZone: http://localhost:10001/eureka |

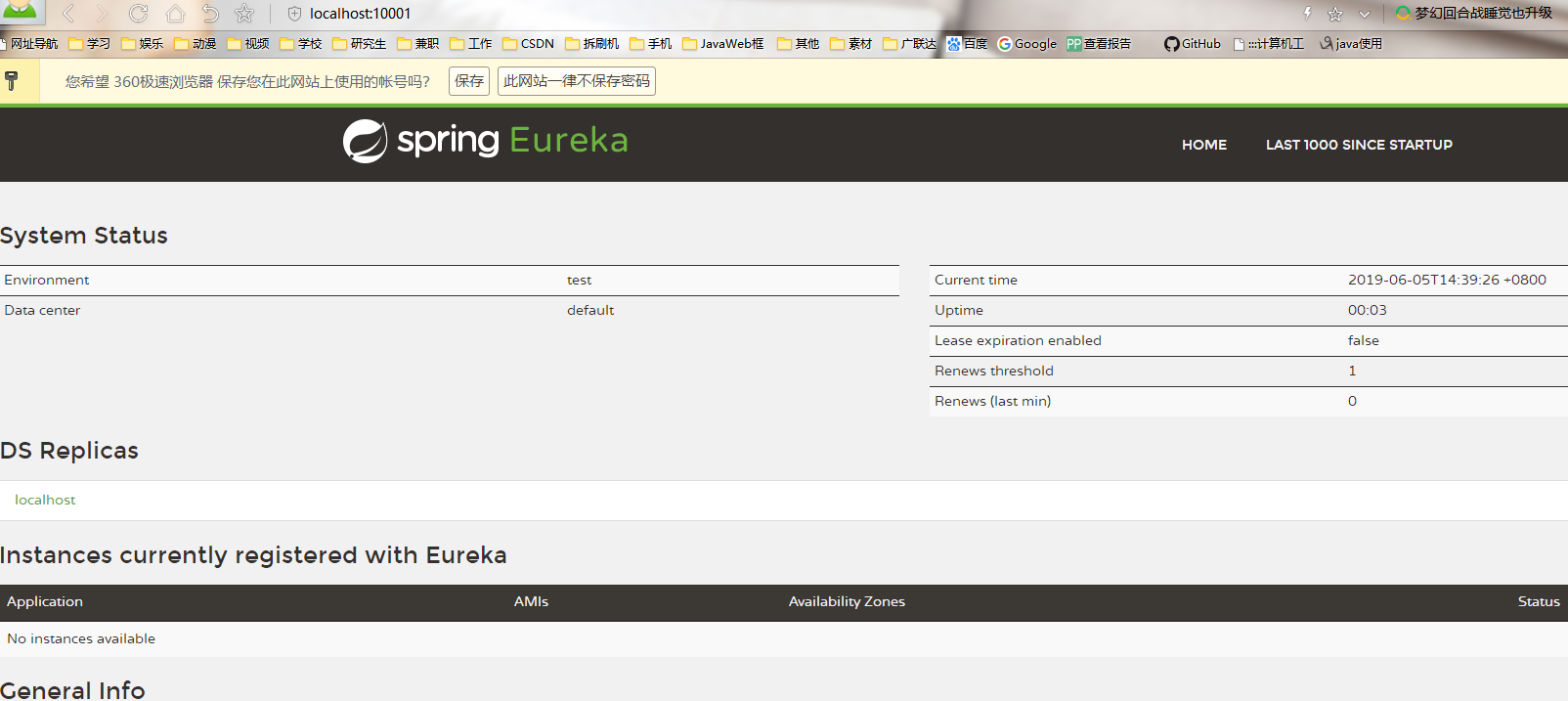
新建一个启动类，并添加注解enableEurekaServer

|  |
| --- |
| **package** cn.huangwei.app;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  @SpringBootApplication()  @EnableEurekaServer//将当前项目标记为eureka server  **public** **class** EurekaApp {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(EurekaApp.**class**, args);  }  } |

运行之后，输入<http://localhost:10001>，提示用户名密码

用户名：user

密码：



如果需要自定义用户名和密码，可以在application.yml加入以下配置

|  |
| --- |
| #开启安全配置，需要密码，如果不需要设置为false  security:  basic:  enabled: true  user:  name: user  password: 123456 #配置用户名和密码后，可以修改地址的访问风格为curl风格 |

配置客户端

在provider-user的pom文件中加入eureka-client依赖

|  |
| --- |
| <!-- 用于监控服务的健康状况 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency> |

配置application.yml文件

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://user:123456@localhost:10001/eureka  #用于显示status列表中的格式  instance:  prefer-ip-address: true  instance-id: ${spring.cloud.client.ipAddress}:${server.port} |

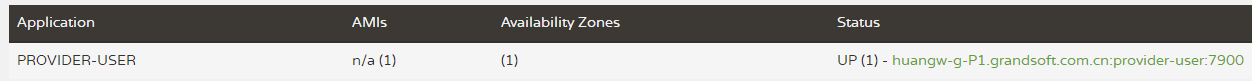
在启动类ProviderUser中添加@EnableEurekaClient注解

|  |
| --- |
| package cn.huangwei.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication(scanBasePackages = "cn.huangwei.controller")  @EnableEurekaClient//标记启用eureka客户端  public class ProviderUser {  public static void main(String[] args) {  SpringApplication.run(ProviderUser.class, args);  }  } |

在UserController中添加测试的方法

|  |
| --- |
| **package** cn.huangwei.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.netflix.appinfo.InstanceInfo;  **import** com.netflix.discovery.EurekaClient;  **import** cn.huangwei.entity.User;  @RestController  **public** **class** UserController {  @Autowired  **private** EurekaClient discoveryClient;    @GetMapping("/user/{id}")  **public** User getUser(@PathVariable **long** id) {  **return** **new** User(id);  }    @GetMapping("/eureka\_info")  **public** String info() {  InstanceInfo instance = discoveryClient.getNextServerFromEureka("PROVIDER-USER", **false**);  **return** instance.getHomePageUrl();  }  } |

运行eurekaServer和provider-user项目



输入<http://huangw-g-p1.grandsoft.com.cn:7900/eureka_info>得到



同理，对consumer-order项目也是这么配置，此时修改orderController中的getUser方法

|  |
| --- |
| package cn.huangwei.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import org.springframework.web.client.RestTemplate;  import com.netflix.appinfo.InstanceInfo;  import com.netflix.discovery.EurekaClient;  import cn.huangwei.entity.User;  @RestController  public class OrderController {  @Autowired  private RestTemplate restTemplate; //spring提供的用于访问rest接口的模板对象    @Autowired  private EurekaClient discoveryClient;    // private String url = "http://localhost:7900/user/";  @Value("${user.url}")  private String url;    @GetMapping("/order/{id}")  public User getOrder(@PathVariable long id) {  //访问提供者，获取数据  InstanceInfo instance = discoveryClient.getNextServerFromEureka("PROVIDER-USER", false);  String homePageUrl = instance.getHomePageUrl();  User user = restTemplate.getForObject(homePageUrl + "/user/" + id, User.class);//通过访问rest访问json数据，然后转换为rest  return user;  }  } |

完成了简单的eureka工作。

**三、springcloud之ribbon**

拷贝consumer-order工程为consumer-order-ribbon

并修改对应的yml文件和pom文件（或者直接新建一个子工程，拷贝相关的文件）

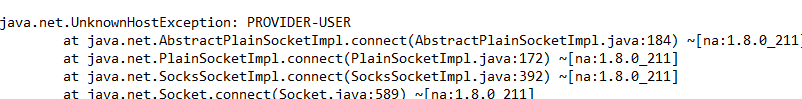
在启动类中添加注解@RibbonClient

|  |
| --- |
| @SpringBootApplication(scanBasePackages = "cn.huangwei.controller")  @EnableEurekaClient  @RibbonClient("PROVIDER-USER")//启用ribbon并对PROVIDER-USER进行负载均衡  **public** **class** ConsumerOrder {  @Bean//相当于xml中的bean标签，主要用于调用当前方法获取指定对象  **public** RestTemplate getTemplate() {  **return** **new** RestTemplate();  }    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(ConsumerOrder.**class**, args);  }  } |

修改orderController文件，测试负载均衡效果

|  |
| --- |
| package cn.huangwei.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.cloud.client.ServiceInstance;  import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RestController;  import org.springframework.web.client.RestTemplate;  import com.netflix.discovery.EurekaClient;  import cn.huangwei.entity.User;  @RestController  public class OrderController {  @Autowired  private RestTemplate restTemplate; //spring提供的用于访问rest接口的模板对象    @Autowired  private EurekaClient discoveryClient;    @Value("${user.url}")  private String url;    @Autowired  private LoadBalancerClient loadBalanceClient;    @GetMapping("/order/{id}")  public User getOrder(@PathVariable long id) {  //访问提供者，获取数据  User user = restTemplate.getForObject("http://PROVIDER-USER/user/" + id, User.class);//通过访问rest访问json数据，然后转换为rest  return user;  }    @GetMapping("/test")  public void test() {  ServiceInstance serviceInstance = loadBalanceClient.choose("PROVIDER-USER");//查找负载均衡后对应的服务实例  System.out.println("1111" + serviceInstance.getServiceId() + "--" + serviceInstance.getHost() + "--" + serviceInstance.getPort());  }  } |

启动eureka项目，两个provider-user项目，consumer-order-ribbon项目，发现出现如下问题，



这个说明发现使用Ribbon的时候，必须要在RestTemplate bean配置中添加负载均衡注解；

重新修改启动类

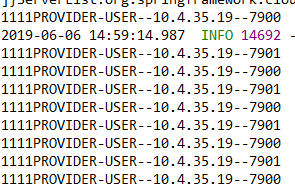


在浏览器中输入<http://localhost:8900/order/4>

发现可以正常访问provider-user中的user/2

输入<http://localhost:8900/test>

可以看到控制台的输出消息

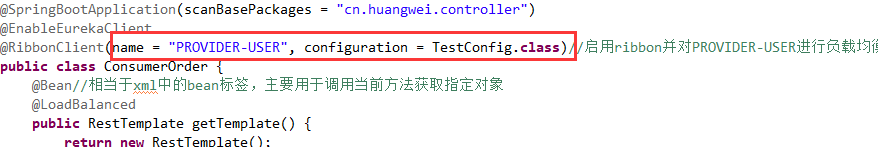
**默认轮询负载均衡机制**。

自定义负载均衡算法；

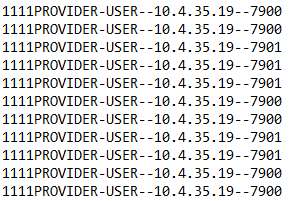
定义一个TestConfig类，这个类使用@Configuration注解，这个注解最好不要放在与SpringBootApplication与ComponentScan注解在同一包路径下，最好放在不同的包级别。

|  |
| --- |
| **package** cn.huangwei;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** com.netflix.client.config.IClientConfig;  **import** com.netflix.loadbalancer.IRule;  **import** com.netflix.loadbalancer.RandomRule;  @Configuration  **public** **class** TestConfig {  @Autowired  **private** IClientConfig clientConfig;    /\*\*  \* 创建负载均衡算法的方法  \* **@return**  \*/  @Bean  **public** IRule ribbonRule() {  **return** **new** RandomRule();//返回随机算法  }  } |

修改consumer-order-ribbon的启动类



输入url地址，发现负载均衡随机



启动类上的注解，在工程名为PROVIDER-USER的项目，会使用TestConfig文件中的负载均衡算法进行负载均衡，其他PROVIDER-USER类似的项目，默认还是使用轮询负载均衡算法。



如果非要将@Configuration注解的文件放到Spring会扫描到的路径下，那么需要做如下配置

新建一个工程consumer-ribbon，内容和consumer-order-ribbon一样；将TestConfig移动到和启动了同一级；

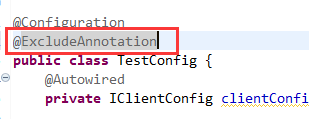
新建一个annotation，该annotation修饰的类，不要被扫描

|  |
| --- |
| **package** cn.huangwei.app;  **public** **@interface** ExcludeAnnotation {  } |

在consumer-ribbon的启动类中，添加componentScan,，具体如下

|  |
| --- |
| package cn.huangwei.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.client.loadbalancer.LoadBalanced;  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  import org.springframework.cloud.netflix.ribbon.RibbonClient;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.FilterType;  import org.springframework.web.client.RestTemplate;  @SpringBootApplication(scanBasePackages = "cn.huangwei.controller")  @EnableEurekaClient  @RibbonClient(name = "PROVIDER-USER", configuration = TestConfig.class)//启用ribbon并对PROVIDER-USER进行负载均衡  **@ComponentScan(excludeFilters = {@ComponentScan.Filter(type = FilterType.ANNOTATION, value = ExcludeAnnotation.class)})**  public class ConsumerOrder {  @Bean//相当于xml中的bean标签，主要用于调用当前方法获取指定对象  @LoadBalanced  public RestTemplate getTemplate() {  return new RestTemplate();  }    public static void main(String[] args) {  SpringApplication.run(ConsumerOrder.class, args);  }  } |

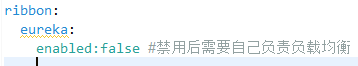
然后再TestConfig类上添加该注解



配置文件的方式配置ribbon, 新建consumer-ribbon-yml的项目，其他不变，修改application.yml文件

|  |
| --- |
| server:  port: 8900  spring:  application:  name: consumer-ribbon  user:  url: http://localhost:7900/user/  eureka:  client:  serviceUrl:  defaultZone: http://user:123456@localhost:10001/eureka  instance:  prefer-ip-address: true  instance-id: ${spring.cloud.client.ipAddress}:${server.port}    PROVIDER-USER1: #指定需要负载均衡的服务客户端的名字  ribbon:  NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule #指定负载均衡使用的算法 |

springcloud已经内置默认使用ribbon，如果不想使用ribbon，可以将其禁用



eureka server高可用

**四、springcloud之feign**

如果想要调用各个微服务之间的方法，可以使用spring提供的RestTemplete，或者使用客户端的服务分发组件eureka，但是这些的便捷性和优雅性都不如feign，下面介绍feign的简单操作以及坑位。

项目列表

-- consumer-order-feign

--provider-user

consumer-order-feign，基本功能与consumer-order类似，springboot版本1.5.9，springcloud版本为Edgware.SR2。

|  |
| --- |
| **App启动类**  package cn.huangwei.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.feign.EnableFeignClients;  import org.springframework.context.annotation.Bean;  import org.springframework.web.client.RestTemplate;  @SpringBootApplication(scanBasePackages = "cn.huangwei")  @EnableFeignClients(basePackages = "cn.huangwei.controller")  public class ConsumerOrder {  @Bean//相当于xml中的bean标签，主要用于调用当前方法获取指定对象  public RestTemplate getTemplate() {  return new RestTemplate();  }    public static void main(String[] args) {  SpringApplication.run(ConsumerOrder.class, args);  }  }  可能由于版本的问题，此时的SpringBootApplication注解上，必须手动添加scanPackage的扫描包，如果不添加，将不会扫描相关注解；同时EnableFeignClients，也要添加basePackages的路径，否则接下来的FeignTest的bean实例将不会产生，提示FeignTest对象not found。  **OrderController类**  package cn.huangwei.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RestController;  import cn.huangwei.entity.User;  @RestController  public class OrderController {  @Autowired  private FeignTest feignTest;    @GetMapping("/order/{id}")  public User getOrder(@PathVariable long id) {  User user = feignTest.getUser(id);  return user;  }  }  该controller的作用就是通过访问order/id，实质上通过feign去调用provider-user的相关服务  **FeignTest类**  package cn.huangwei.controller;  import org.springframework.cloud.netflix.feign.FeignClient;  import org.springframework.stereotype.Component;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import cn.huangwei.entity.User;  @FeignClient(name = "provider-user", url = "http://localhost:7903")  public interface FeignTest {  @GetMapping(value = "/user/{id}")  public User getUser(@PathVariable Long id);  }  Pom文件    **Application.yml**  server:  port: 8900  spring:  application:  name: consumer-order-feign |

provider-user项目，简单的提供user的服务

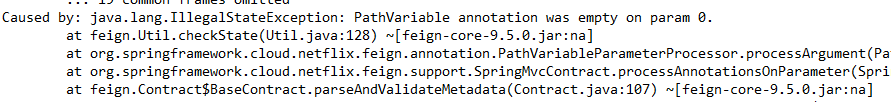
|  |
| --- |
| **启动类app**  package cn.huangwei.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication(scanBasePackages = "cn.huangwei")  public class ProviderUser {  public static void main(String[] args) {  SpringApplication.run(ProviderUser.class, args);  }  }  **Controller类**  **package** cn.huangwei.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.netflix.appinfo.InstanceInfo;  **import** com.netflix.discovery.EurekaClient;  **import** cn.huangwei.entity.User;  @RestController  **public** **class** UserController {  @Autowired  **private** EurekaClient discoveryClient;    @GetMapping("/user/{id}")  **public** User getUser(@PathVariable **long** id) {  **return** **new** User(id);  }  }  Pom文件    **Application.yml**  server:  port: 7903  spring:  application:  name: provider-user-swagger |

运行项目；

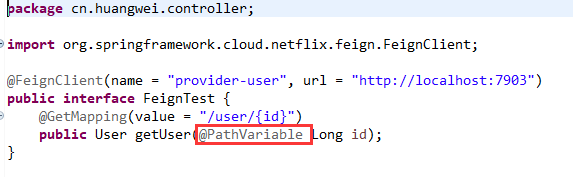
先启动provider-user项目，正常启动

然后启动consumer-order-feign

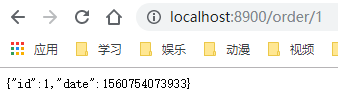
提示：



原因出在这里：



如果使用了getMapping等Restful风格的注解，那么@PathVariable也需要使用restful风格，变为@PathVariable(“id”);

访问成功

**feign坑位**

**1.@RequestMapping(value = "/user/{id}", method = RequestMethod.GET)**

**@GetMapping("/user/{id}")**

这两个注解的效果是等价的，但是在Feign使用中，只能用上面的那种方式，不能直接用@GetMapping，下面我们将前面的那个示例中，改成@GetMapping注解看下效果，我们发现，修改注解后重新启动服务的时候，抛了如下异常：

Caused by: java.lang.IllegalStateException: Method findById not annotated with HTTP method type (ex. GET, POST)

由于存在版本问题，这个情况可能会出现，也可能不出现

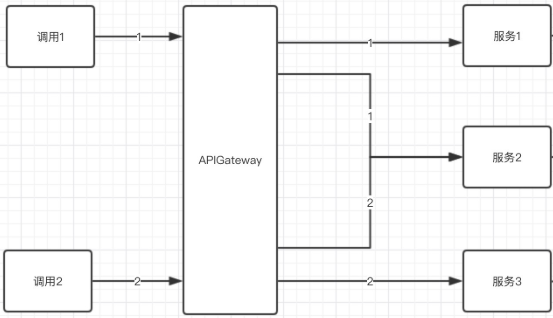
2.多参数问题

|  |
| --- |
| @RequestMapping(value="/user/name", method=RequestMethod.GET)  User findByUsername(final String userName, final String address);  启动服务后，提示  Caused by: java.lang.IllegalStateException: Method has too many Body parameters  当使用feign时，如果发送的时get请求，那么需要在请求参数前加上@RequestParam，表示该请求参数时get请求的参数，如果不标注，可能会提示Request method 'POST' not supported  因为我们使用的是get请求，如果userName没有被@RequestParam注解修饰，会自动当成requestbody来处理。只要有body，feign认为是post请求，例如参数是一个复杂对象User，那么也会当成post请求，从而报错 |

**五、SpringCloud之zuul**

1. APIGateway是什么？

APIGateway即API网关，所有请求首先会经过这个网关，然后到达后端服务。API网管作为系统接口对外的统一出口，可以减少调用方对服务实现的感知。



2. APIGateway的主要作用有：

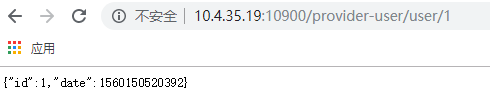
2.1 统一对外接口：利用APIGateway可以让用户在不感知服务边缘的情况下，利用统一的接口组装服务。当内部服务修改时，可以通过APIGateway进行适配，不需要调用方进行调整减少对外暴露服务。

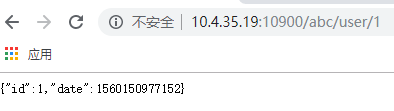
2.2 统一鉴权：通过APIGateway对访问进行统一鉴权，不需要每个应用单独对调用方进行鉴权。

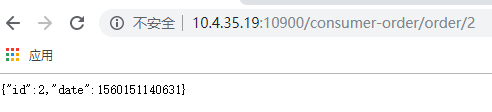
2.3 服务注册与授权：可以控制调用方使用或不能使用某些服务

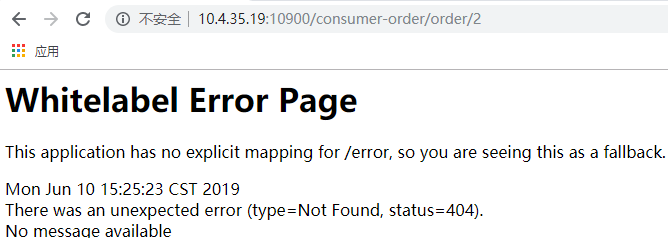
2.4 服务限流：通过APIGateway可以对调用方调用每个接口的每日调用及总调用次数限制

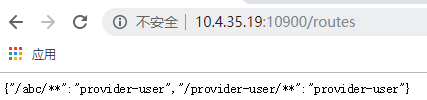
2.5 全链路跟踪：通过APIGateway提供的唯一请求Id，监控调用流程，以及调用的相应时间。

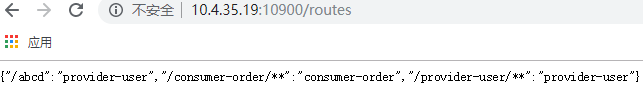


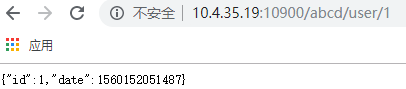


说明zuul能访问所有eureka上的代理服务，一个个代理过去





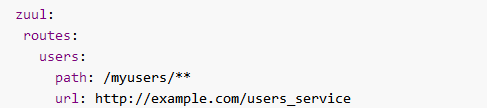




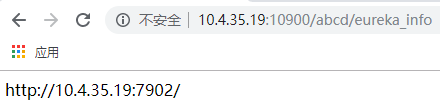
/abc/\* 表示只能匹配一级，即/abc/user

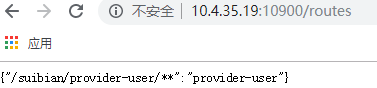
/abc/\*\* 表示能匹配多级，即/abc/user/1, 而上一种不行

还有一种使用url方式



如果使用url方式，将不包含hystrixCommand（快速熔断）机制，也不包含ribbon的负载均衡机制，因此如果要上面的功能，建议使用serviceId





prefix: /abc表示给所有服务请求添加前缀；例如原请求为 /user/1-----》/abc/user/1



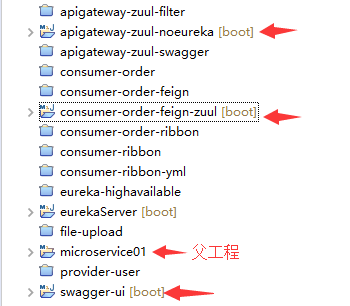
表示如果通过zuul请求的url为<http://10.4.35.19:10900/abcd/user/1>; 此时的stripPrefix为false，那么请求到服务的真正的url为<http://10.4.35.19:7901/abcd/user/1>; 因此此时返回404



表示如果通过zuul请求的url为<http://10.4.35.19:10900/abcd/user/1>;此时的stripPrefix为true，那么请求到服务的真正的url为<http://10.4.35.19:7901/user/1>;，这样可以正常请求，默认spring cloud的值为true

**六．feign+zuul实现生成user对象**

工程目录：



apigateway-zuul-noeureka为服务，作为路由转发的服务

|  |
| --- |
| **Pom文件**    这里需要使用consumer-order-feign-zuul工程中的东西，因此需要导入consumer-order-feign-zuul工程  **Application.yml**  server:  port: 10900  spring:  application:  name: apigateway-zuul  #用于显示status列表中的格式  instance:  prefer-ip-address: true  instance-id: ${spring.cloud.client.ipAddress}:${server.port}  security: #配置访问zuul的routes的时候使用  user:  name: user  password: abc  zuul:  routes:  users:  path: /consumer-order-feign-zuul/\*\*  url: http://localhost:8902  stripPrefix: true  app启动类  package cn.huangwei.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.feign.EnableFeignClients;  import org.springframework.cloud.netflix.zuul.EnableZuulProxy;  @SpringBootApplication(scanBasePackages = "cn.huangwei")  @EnableZuulProxy //启用zuul，自带熔断和自动注册到eureka中，注意需要导入eureka-client依赖  @EnableFeignClients(basePackages = "cn.consumer.controller")  public class ZuulApp {  public static void main(String[] args) {  SpringApplication.run(ZuulApp.class, args);  }  }  package cn.huangwei.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RestController;  import cn.consumer.controller.FeignTest;  import cn.consumer.entity.User;  @RestController  public class OrderController {  @Autowired  private FeignTest feignTest;    @GetMapping("/order/{id}")  public User getOrder(@PathVariable long id) {  User user = feignTest.getUser(id);  return user;  }  } |

运行服务，在网址上输入localhost:10900/routes,显示配置转发的路由



输入local host:10900/order/1

