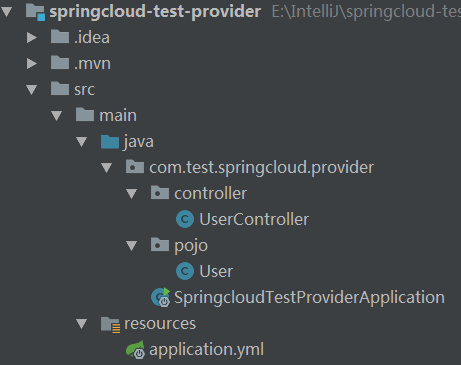
# 环境搭建

在开始SpringCloud之前，先看一下一个简单的服务提供者和服务消费者。服务提供者提供一个REST风格的HTTP接口给服务消费者。

## 普通的提供者者



### pom.xml

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>  
</properties>  
  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
   
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

### 关键代码

实体类：

**package com.test.springcloud.provider.pojo**;  
  
**public class User {  
 private Long id**;  
 **private String username**;  
 **//getter/setter略**  
**}**

Controller

**package com.test.springcloud.provider.controller**;  
  
**import com.test.springcloud.provider.pojo.User**;  
**import org.springframework.web.bind.annotation.GetMapping**;  
**import org.springframework.web.bind.annotation.PathVariable**;  
**import org.springframework.web.bind.annotation.RestController**;  
  
**@RestController  
public class UserController {  
 @GetMapping**(**"/simple/{id}"**)**//rest 风格,微服务一般都使用 rest 风格  
 public User** findById(**@PathVariable Long id**) **{  
 User user=new** User();  
 **user**.setId(**id**);  
 **user**.setUsername(**"hello "+id**);  
 **return user**;  
 **}  
}**

### application.yml

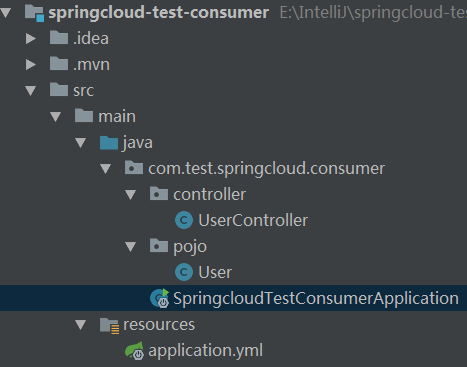
server:  
 port: 8081

启动项目后：

访问：<http://localhost:8081/simple/1>



## 普通的消费者



### pom.xml

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>  
</properties>  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>  
  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

### 关键代码

实体类：

**package com.test.springcloud.consumer.pojo**;  
  
**public class User {  
 private Long id**;  
 **private String username**;  
 **//getter/setter略**  
**}**

Controller

**package com.test.springcloud.consumer.controller**;  
  
**import com.test.springcloud.consumer.pojo.User**;  
**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.RestController**;  
**import org.springframework.web.client.RestTemplate**;  
  
**@RestController  
public class UserController {  
 @Autowired  
 private RestTemplate restTemplate**;**// rest 请求模板类  
  
 @Value**(**"${user.userServicePath}"**)**//从配置文件中读取指定属性,名字和配置中保持一致即可  
 private String userServicePath**;  
  
 **@GetMapping**(**"/user/{id}"**)  
 **public User** findById(**@PathVariable Long id**) **{  
 //调用指定的地址,传递参数过去,将返回的数据解析为 user 格式  
 // return this.restTemplate.getForObject("http://localhost:7900/simple/"+ id, User.class);//硬编码,不好  
 return this**.**restTemplate**.getForObject(**this**.**userServicePath + id**, **User**.**class**);**//通过配置文件注入地址的方式  
 }  
}**

启动类

**package com.test.springcloud.consumer**;  
  
**import org.springframework.boot.SpringApplication**;  
**import org.springframework.boot.autoconfigure.SpringBootApplication**;  
**import org.springframework.context.annotation.Bean**;  
**import org.springframework.web.client.RestTemplate**;  
  
**@SpringBootApplication  
public class SpringcloudTestConsumerApplication {  
 @Bean  
 public RestTemplate** restTemplate() **{  
 return new** RestTemplate();  
 **}  
 public static void** main(**String**[] **args**) **{  
 SpringApplication**.run(**SpringcloudTestConsumerApplication**.**class**, **args**);  
 **}  
}**

### application.yml

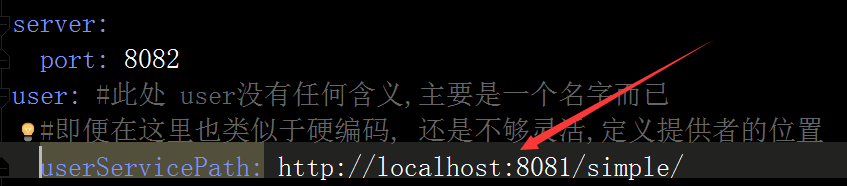
server:  
 port: 8082  
user: #此处 user没有任何含义,主要是一个名字而已  
 #即便在这里也类似于硬编码, 还是不够灵活,定义提供者的位置  
 userServicePath: http://localhost:8081/simple/

启动项目，访问<http://localhost:8082/user/2>

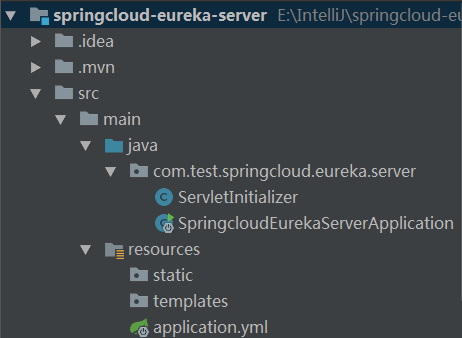


这里的数据其实是消费者调用了服务提供者之后得到的。

## Eureka



在消费者调用服务的时候，服务提供者的地址是以硬编码的形式写在配置文件中的。如果服务端迁移就需要改变地址。我们可以将服务注册到Eureka，消费者不必关心提供者的真实地址，通过Eureka中的服务名直接调用服务即可。

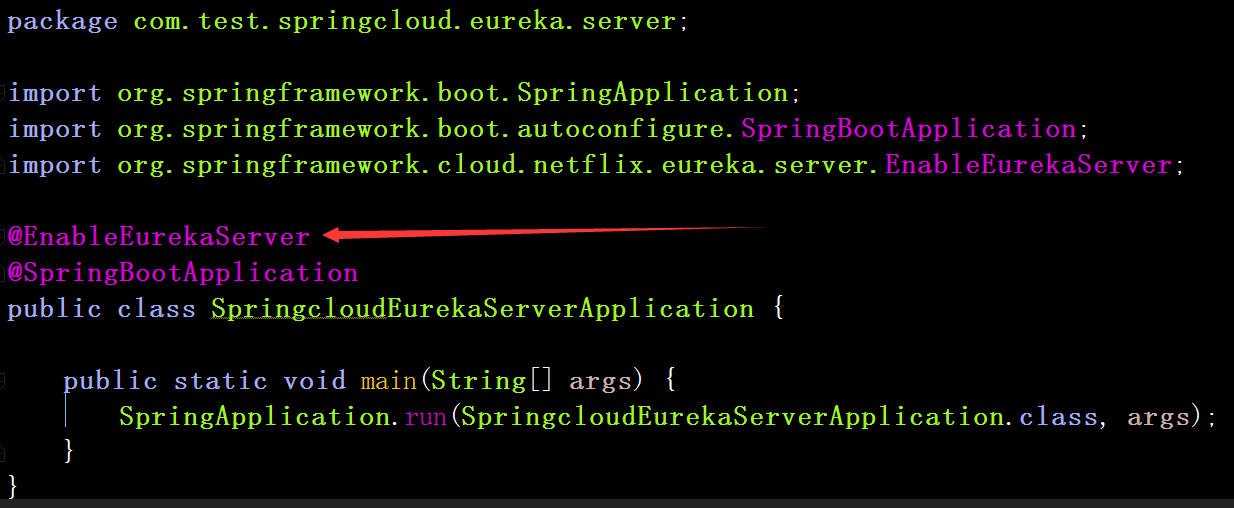


### pom.xml

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>Edgware.SR2</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-server</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>

### 启动类

在启动类上增加@EnableEurekaServer注解，如图：

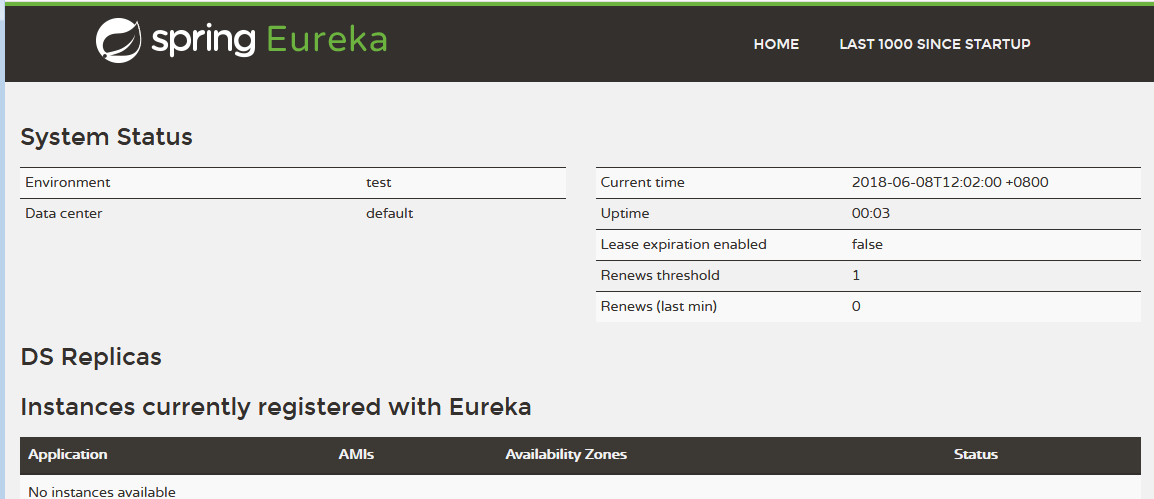


### application.yml

server:  
 port: 8083  
  
eureka:  
 instance:  
 hostname: localhost  
 client:  
 serviceUrl:  
 defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

启动项目，能看见一个管理后台：

访问<http://localhost:8083/>

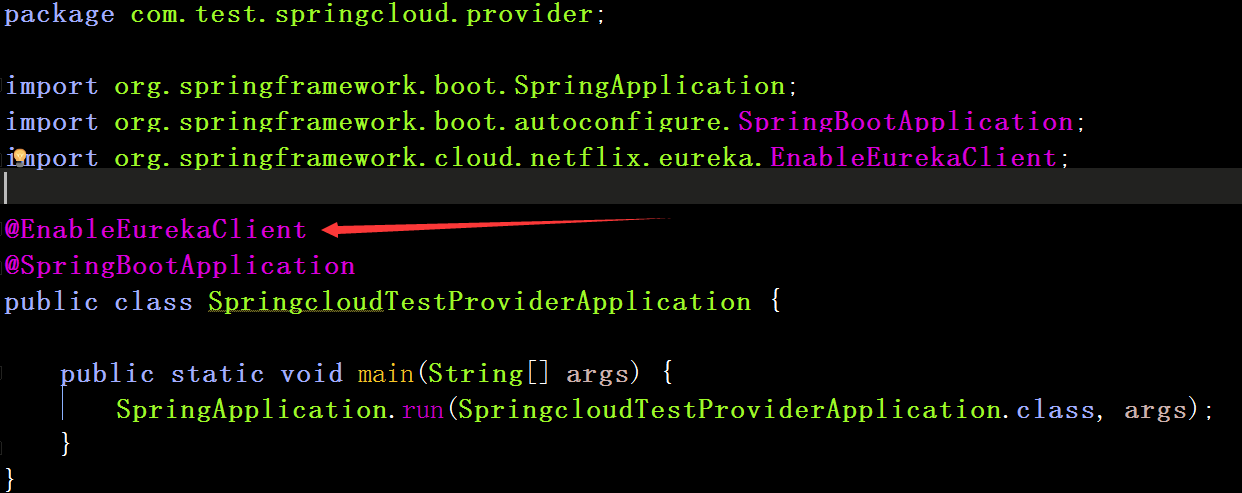


### 修改服务提供者

pom.xml中添加SpringCloud的引用

<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>  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-eureka</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

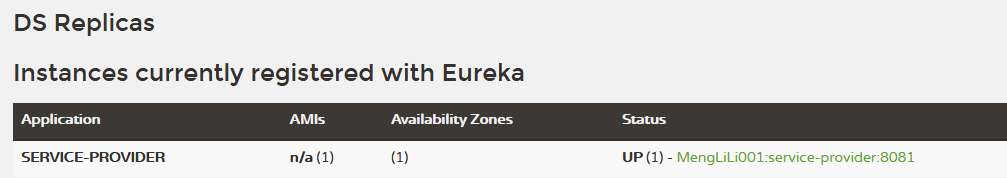
启动类上添加注解@EnableEurekaClient：



完整的application.yml

server:  
 port: 8081  
eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://localhost:8083/eureka/  
spring:  
 application:  
 name: service-provider #这个名字就是调用服务时的名字

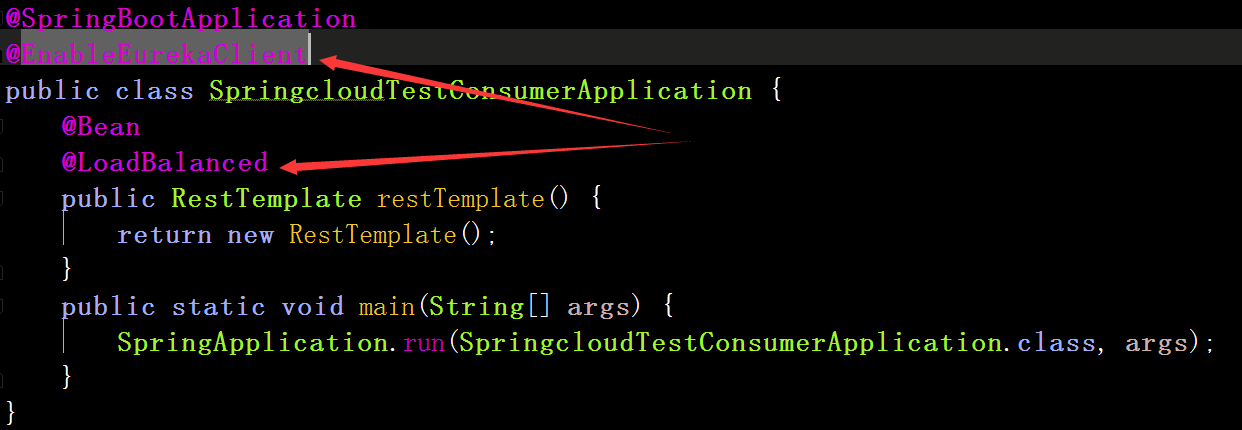
启动提供者，Eureka的控制台可以看见服务：



### 修改消费者

pom.xml同提供者一样

启动类添加注解@EnableEurekaClient还有restTemplate上的@LoadBalanced



完整的application.yml

server:  
 port: 8082  
user: #此处 user没有任何含义,主要是一个名字而已  
 #即便在这里也类似于硬编码, 还是不够灵活,定义提供者的位置  
 #userServicePath: http://localhost:8081/simple/  
 userServicePath: http://SERVICE-PROVIDER/simple/  
eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://localhost:8083/eureka/  
spring:  
 application:  
 name: service-comsumer

### 1.3.4 添加密码

注册中心可以添加密码

需要引入jar包

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

注册中心的配置文件中：

#配置密码，如果不需要，就不写这段  
security:  
 basic:  
 enabled: true  
 user:  
 name: username  
 password: password123  
#配置密码结束

调用者在访问Eureka的时候：

eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://username:password123@localhost:8083/eureka/

defaultZone的格式是

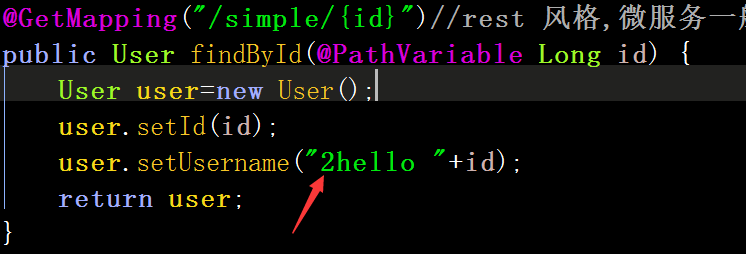
http://用户名:密码@主机地址:端口/eureka

## ribbon负载均衡

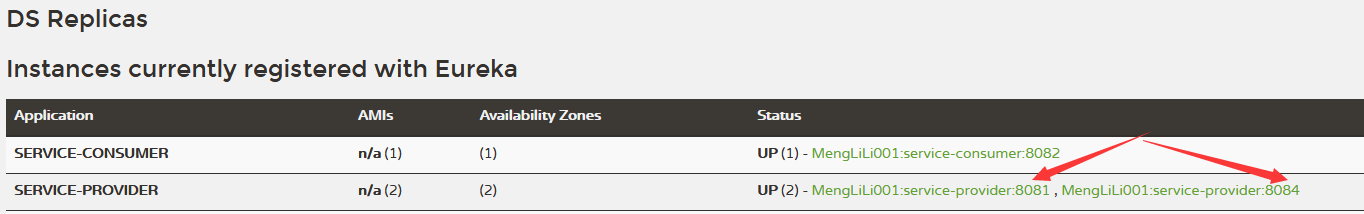
在消费者中添加依赖：

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

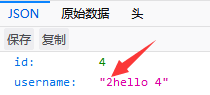
启动两次提供者(一个运行main方法，一个maven启动)，修改端口，模拟同一个服务部署在不同的服务器上，如端口为8084。稍稍修改一些controller中的代码，区分两个服务：



启动服务，可以看到同一个服务有两个提供者：



刷新消费者，可以看到返回结果实际上是实现了负载均衡。

# Feign

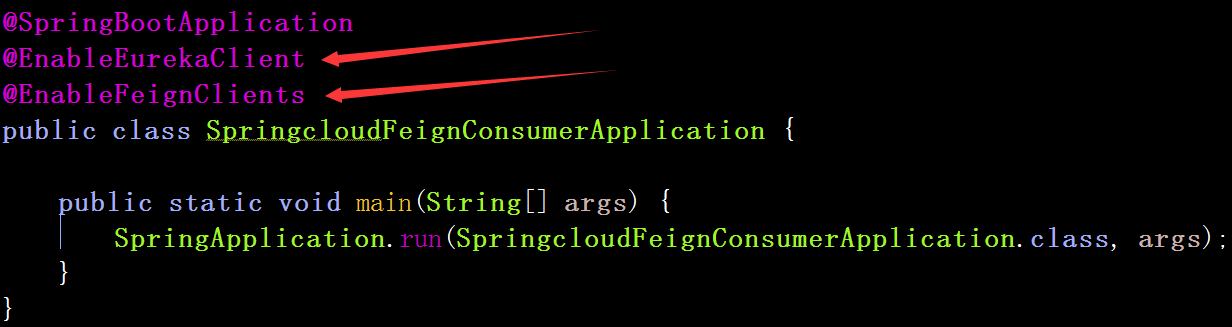
Feign是一个声明式的伪Http客户端，它使得写Http客户端变得更简单。使用Feign，只需要创建一个接口并注解。Feign默认集成了Ribbon，并和Eureka结合，默认实现了负载均衡的效果。

## pom.xml

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>Edgware.SR2</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-eureka</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-feign</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>

## 启动类

启动类添加@EnableEurekaClient注解和@EnableFeignClients



## service

**package com.test.springcloud.feign.consumer.service**;  
  
**import com.test.springcloud.feign.consumer.pojo.User**;  
**import org.springframework.cloud.netflix.feign.FeignClient**;  
**import org.springframework.web.bind.annotation.PathVariable**;  
**import org.springframework.web.bind.annotation.RequestMapping**;  
**import org.springframework.web.bind.annotation.**RequestMethod;  
  
**@FeignClient**(**value = "SERVICE-PROVIDER"**)  
**public interface UserService {  
 @RequestMapping**(**value = "/simple/{id}"**, **method =** RequestMethod.***GET***)  
 **User** getUserById(**@PathVariable**(**value = "id"**) **Long id**);  
**}**

@FeignClient的值是服务提供者在Eureka上的名字。

方法上@RequestMapping的值是请求服务提供者时的路径。参数的传递方式和SpringMVC的Controller接收参数时语法一样，如@PathVariable,@RequestParam

User类同服务提供者中的一样。

## Controller

**package com.test.springcloud.feign.consumer.controller**;  
  
**import com.test.springcloud.feign.consumer.pojo.User**;  
**import com.test.springcloud.feign.consumer.service.UserService**;  
**import org.springframework.beans.factory.annotation.Autowired**;  
**import org.springframework.web.bind.annotation.RequestMapping**;  
**import org.springframework.web.bind.annotation.**RequestMethod;  
**import org.springframework.web.bind.annotation.RequestParam**;  
**import org.springframework.web.bind.annotation.RestController**;  
  
**@RestController  
public class UserController {  
 @Autowired  
 private UserService userService**;  
  
 **@RequestMapping**(**value = "/test"**, **method =** RequestMethod.***GET***)  
 **public User** sayHi(**@RequestParam Long id**) **{  
 return userService**.**getUserById**(**id**);  
 **}  
}**

## application.yml

eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://localhost:8083/eureka/  
server:  
 port: 8085  
spring:  
 application:  
 name: service-feign

启动项目，再启动两个服务提供者，访问

<http://localhost:8085/test/?id=2>



能看见实现了两个服务提供者的负载均衡。

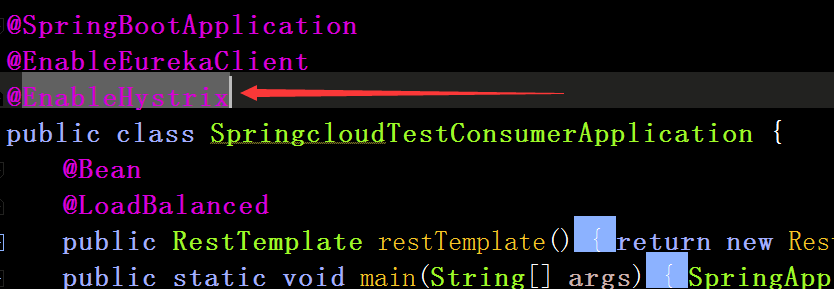
# 断路器Hystrix

## Ribbon使用断路器

在ribbon的消费者工程中加入断路器依赖：

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

启动类上添加注解@EnableHystrix

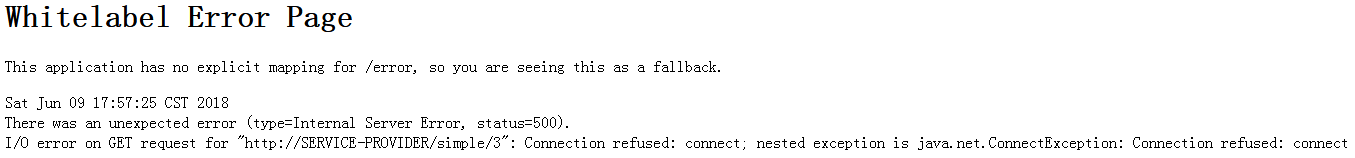




启动项目，在服务提供者挂掉的时候，会看到提示信息：



如果不做熔断，会在请求服务提供者超时之后报错：



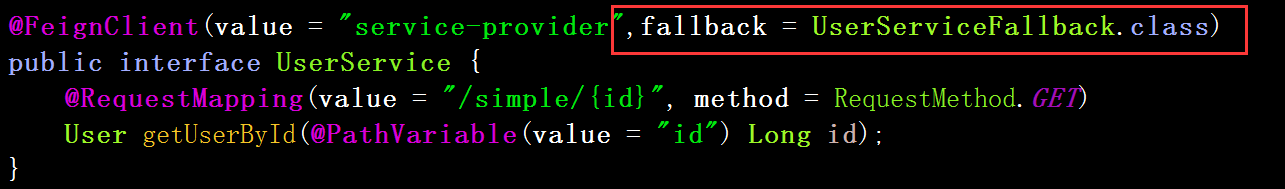
## Feign使用断路器

Feign默认就带有断路器

在application.yml中开启断路器：

feign:  
 hystrix:  
 enabled: true

在接口上添加fallback属性，值是处理断路的类。



UserServiceFallback是UserService的实现类，但是不是业务实现类(不调用dao进行业务操作)。它只做断路时的处理。

**package com.test.springcloud.feign.consumer.service**;  
  
**import com.test.springcloud.feign.consumer.pojo.User**;  
**import org.springframework.stereotype.Component**;  
  
**@Component  
public class UserServiceFallback implements UserService {  
 @Override  
 public User** getUserById(**Long id**) **{  
 User user = new** User();  
 **user**.setUsername(**"sorry"**);  
 **return user**;  
 **}  
}**

当提供者挂掉的时候，访问消费者：

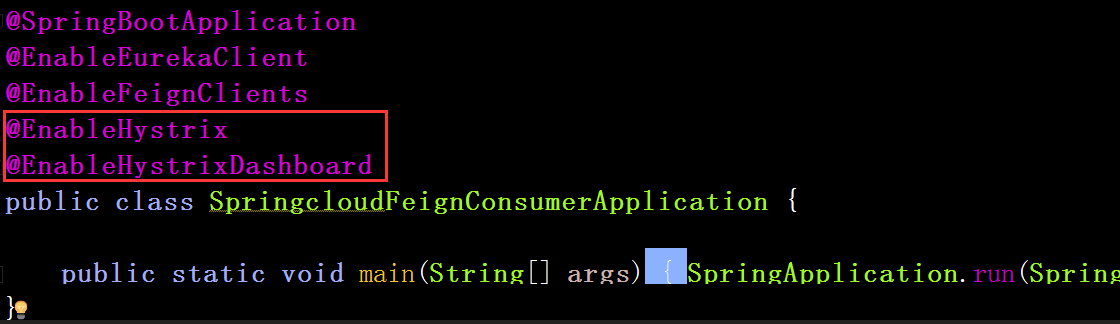


## Hystrix Dashboard (Hystrix 仪表盘)

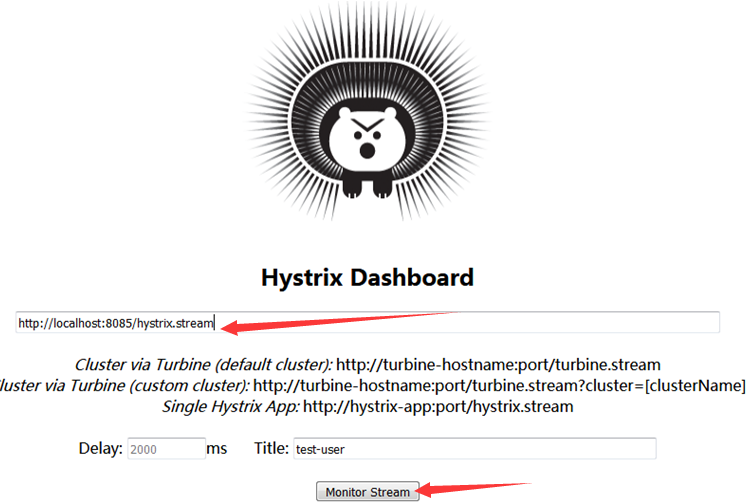
在消费者的一端配置(Feign和Ribbon)：

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-hystrix</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>  
</dependency>

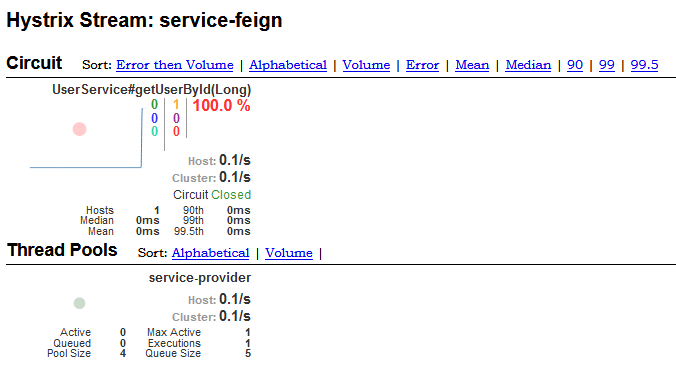
启动类：



浏览器中访问：host:端口/hystrix



访问一次消费者，Hystrix控制台上：



# 路由Zuul

新建一个路由工程

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>Edgware.SR3</spring-cloud.version>  
</properties>  
  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-eureka</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-zuul</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>

application.yml

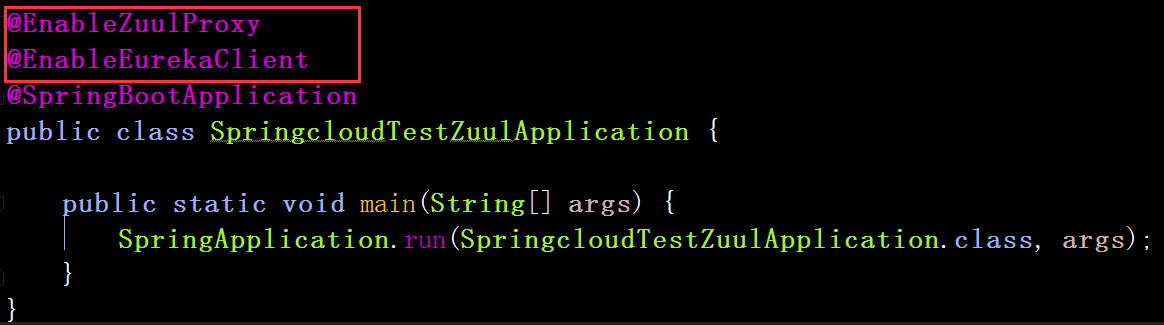
eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://localhost:8083/eureka/  
server:  
 port: 8086  
spring:  
 application:  
 name: service-zuul  
zuul:  
 routes:  
 api-a:  
 path: /api-a/\*\*  
 serviceId: service-consumer  
 api-b:  
 path: /api-b/\*\*  
 serviceId: service-feign

routes定义了请求转发的规则。path是url访问路径，serviceId是对应的Eureka中的服务

<http://localhost:8086/api-a/user/1>会发送到service-consumer对应的工程

<http://localhost:8086/api-b/test?id=2>会发送到feign的工程

启动类：



# 分布式配置中心

SpringCloud Config

## 创建远程配置文件

从远程获取配置文件

可以使用git。我们在此使用码云

在码云上新建一个项目(就是个普通的项目)：



在项目中上传一个config-client-dev.properties文件，里面随意配置个属性，我们测试能不能从远程读取文件，所以属性可以随意配置，但是文件名要符合规范：

profile: profile-default

文件名是{application-name}-{profile}的命名。也就是应用名-运行环境

profile:

dev开发环境

test测试环境

pro正式环境

## 创建Server工程

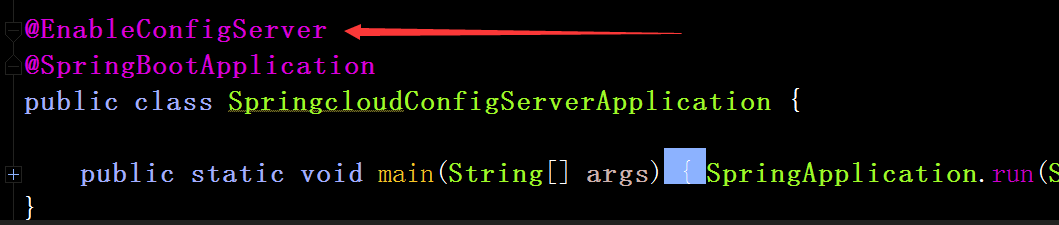
pom.xml

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>Edgware.SR2</spring-cloud.version>  
</properties>  
  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-server</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</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>

application.yml

server:  
 port: 8087  
spring:  
 cloud:  
 config:  
 server:  
 git:  
 uri: https://gitee.com/MengLaoShi/springcloud-config.git

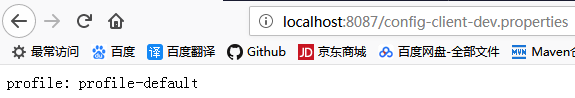
启动类：



启动项目，访问：

<http://localhost:8087/config-client-dev.properties>

可以看到从远程git上读取了配置文件



## 创建Client工程

创建客户端工程：

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.13.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>Edgware.SR2</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-config</artifactId>  
 </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>

application.yml

server:  
 port: 8088

bootstrap.yml 注意连接ConfigServer的配置必须写在bootstrap.yml中

spring:  
 cloud:  
 config:  
 uri: http://localhost:8087  
 profile: dev  
 label: master # 当configserver的后端存储是Git时，默认就是master  
 application: #此属性可以放到 application 中  
 name: config-client  
 #程序 name, 参考路径的匹配规则,会找config-client开头的dev文件

寻找文件的路径是：

/{application}/{profile}[/{label}]

/{application}-{profile}.yml

/{label}/{application}-{profile}.yml

/{application}-{profile}.properties

/{label}/{application}-{profile}.properties

application代表程序名称, profile 代表哪个文件 lable 代表版本,比如 git 是 master

Controller

**@RestController  
public class TestController {  
 @Value**(**"${profile}"**)  
 **private String profile**;  
  
 **@RequestMapping**(**value = "/test"**)  
 **public String** test() **{  
 return profile**;  
 **}  
}**

启动项目，访问<http://localhost:8088/test> 能读取到配置文件中的值



## 整合Eureka

### 修改Server工程

修改ConfigServer，添加Eureka的依赖：

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

application.yml



### 修改Client工程

添加Eureka依赖

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

boostrap.yml



启动工程。可以看到和以前同样的运行效果。Client没有直接访问Server，而是调用了Eureka中的服务。能实现高可用。