Eureka是Spring Cloud Netflix微服务套件中的一部分，提供了服务治理的功能。Eureka集群中的节点基于REST服务进行通信，如使用HttpClient与RestTemplate，Spring Cloud 套件中的Feign模块提供了更为简洁的服务访问。

Zuul是一个网关，能够将集群的服务隐藏到网关后面，统一对外提供服务，功能上类似Nginx（部分类似），提升了集群的安全性。

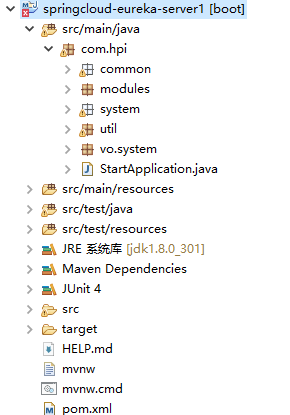
集群结构

本示例共4个项目，分别为eureka-server、service-provider、service-invoker和zuul-server。其中，eureka-server为服务注册中心，server1端口号为8761；service-provider为服务提供者，provider1端口号为9000；service-invoker为服务调用者，端口号为8000；service-zuul为网关，端口号为8080。

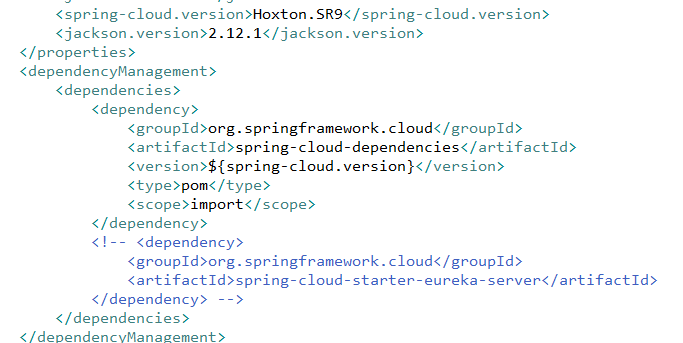
# 构建注册中心Eureka服务器

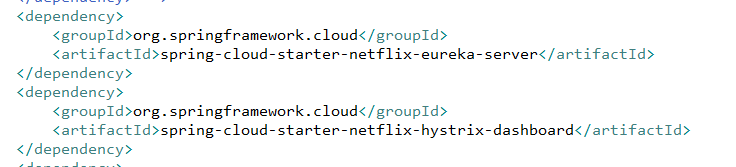
## 创建项目

创建名为springcloud-eureka-server的maven项目



Pom文件如下：主要增加如下坐标







详情如下：

<?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>com.hpi</groupId>

<artifactId>springboot-start</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- 打包机制，如pom,jar,maven-plugin,ejb,war,ear,rar,par 默认为jar方式 ；war包配置-->

<packaging>jar</packaging>

<!--<packaging>war</packaging>-->

<name>springboot-start</name>

<description>springcloud-server1，Eureka服务器，注册中心</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.12.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>

<maven.compiler.encoding>UTF-8</maven.compiler.encoding>

<java.version>1.8</java.version>

<spring-cloud.version>Hoxton.SR9</spring-cloud.version>

<jackson.version>2.12.1</jackson.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>

<!-- <dependency>

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

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency> -->

</dependencies>

</dependencyManagement>

<dependencies>

<!-- spring-boot 基础依赖包 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions><!-- 去掉默认配置 -->

<!-- jar 需要注释掉spring-boot-starter-tomcat相关依赖,war 必须放开 ；war包配置-->

<!-- <exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>-->

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-jul</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.17.0</version>

</dependency>

<!-- 工具包 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.78</version>

</dependency>

<!--lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--swagger2 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-text</artifactId>

<version>1.9</version>

</dependency>

<!-- <dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.10.0</version>

</dependency> -->

</dependencies>

<build>

<finalName>eureka-server1</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<mainClass>com.hpi.StartApplication</mainClass>

<includeSystemScope>true</includeSystemScope>

</configuration>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

<!-- 改成jar取消，war必须放开 ；war包配置，同时会导致系统导入的jar丢失，需做配置，在百度搜-->

<!-- <plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

</plugin> -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<verbose>true</verbose>

<!-- 项目编码-->

<encoding>UTF-8</encoding>

<fork>true</fork>

<executable>${JAVA8\_HOME}/bin/javac</executable>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skipTests>true</skipTests> <!--打包过程默认关掉单元测试 -->

</configuration>

</plugin>

<!--<plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-surefire-plugin</artifactId>

<configuration> <skip>true</skip>跳过单元测试 <testFailureIgnore>true</testFailureIgnore>这个网上很多的解决方式是这个,其实这个,其实这个配置后打包还是会编译单元测试类的,只是忽略编译单元测试类的错误.

</configuration> </plugin> -->

</plugins>

<resources>

<!-- 打包时将jsp文件拷贝到META-INF目录下 -->

<resource>

<!-- 指定resources插件处理哪个目录下的资源文件 -->

<directory>src/main/templates</directory>

<!--注意此次必须要放在此目录下才能被访问到 -->

<targetPath>META-INF/resources</targetPath>

<includes>

<include>\*\*/\*\*</include>

</includes>

</resource>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<!-- 是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*\*</include>

</includes>

<!--是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

</resources>

<defaultGoal>compile</defaultGoal>

</build>

<!--注意：这里必须要添加，否则各种依赖有问题-->

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/libs-milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

## 编写启动类

编写一个Springboot的启动类，并加上@EnableEurekaServer即可将此项目声明为Eureka服务器。启动类如下所示：

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.**class**})//取消数据库配置

//@ServletComponentScan(basePackages = "com.hpi.system.filters")

@SpringBootApplication

@ComponentScan(basePackages

= {

"com.hpi.common",

"com.hpi.modules",

"com.hpi.system"

})

@EnableCaching // 开启缓存注解

@EnableEurekaServer//支持eureka服务端

@EnableHystrixDashboard

**public** **class** StartApplication **extends** SpringBootServletInitializer

{

**public** **static** **void** main(String[] args)

{

// System.setProperty("spring.devtools.restart.enabled", "false");//完全关闭重启支持

SpringApplication.*run*(StartApplication.**class**, args);

}

}



## 编写application.properties文件

在文件中增加如下：

spring.application.name= eureka-server1

server.port=8761

#eureka

eureka.instance.hostname=localhost

#false表示不向服务中心注册自己

eureka.client.register-with-eureka=false

#false表示自己就是注册中心，不需要从eureka server获取注册信息,需要检索服务

eureka.client.fetch-registry=false

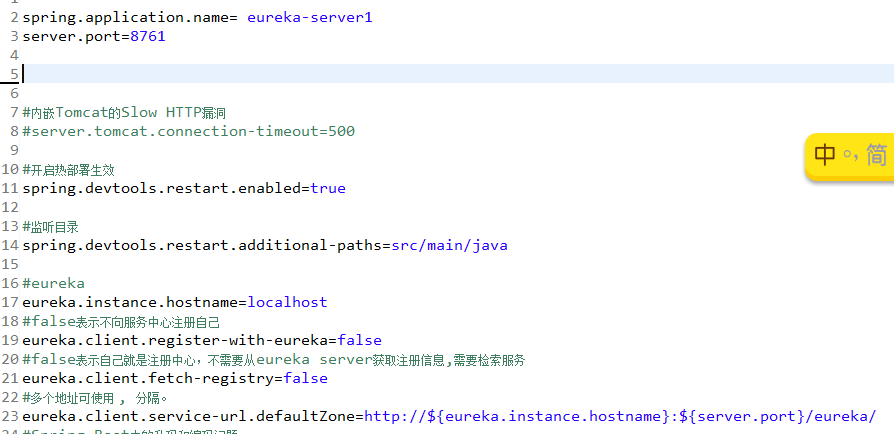
#多个地址可使用 , 分隔。

eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/

如果有security，则增加

spring.security.user.name=admin

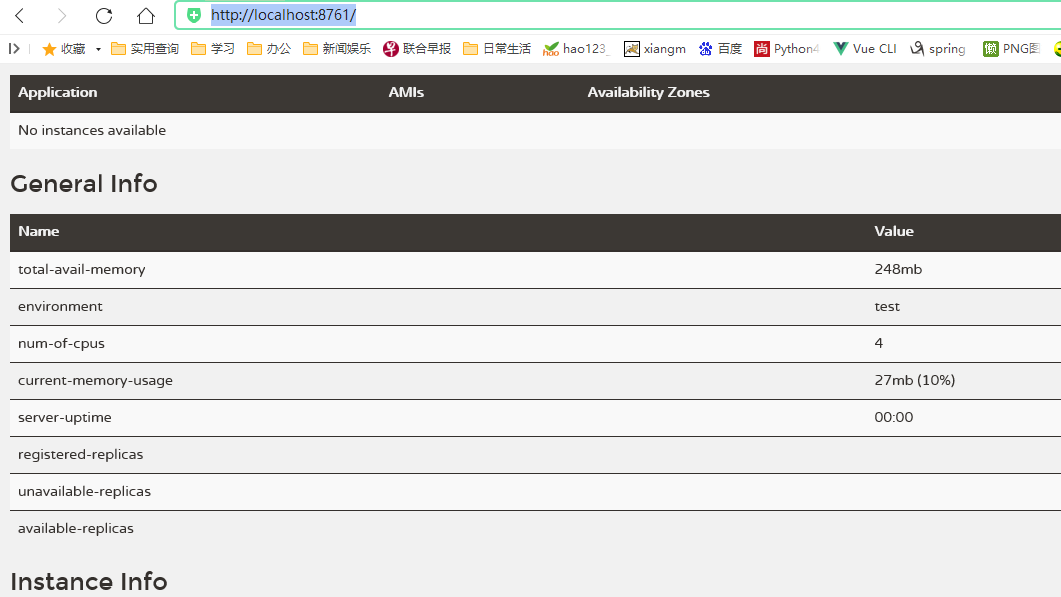
spring.security.user.password=123456



因为本例只有一台eureka服务器（可以是多台），所以register-with-eureka和fetch-registry这两项为false

## 启动Eureka服务，进入注册中心

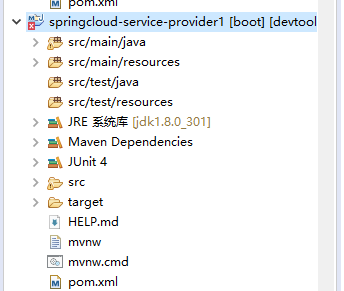
输入<http://localhost:8761/>，进入注册中心



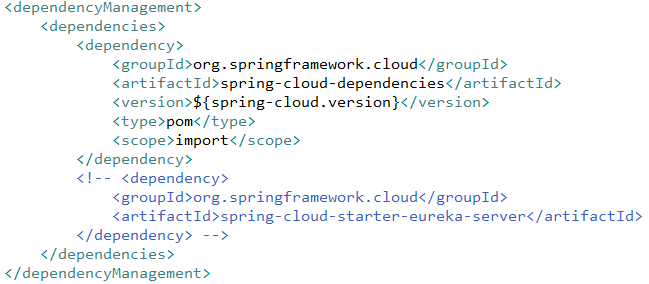
# 编写服务提供者（provider）

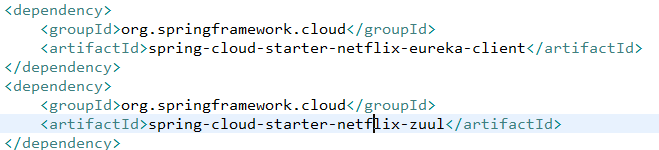
## 创建provider项目

创建名为springcloud-service-provider的maven项目。



pom.xml文件如下：主要增加如下





详情如下：

<?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>com.hpi</groupId>

<artifactId>springboot-start</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- 打包机制，如pom,jar,maven-plugin,ejb,war,ear,rar,par 默认为jar方式 ；war包配置-->

<packaging>jar</packaging>

<!--<packaging>war</packaging>-->

<name>springboot-start</name>

<description>service-provider,服务提供者，springcloud-client1,向eureka注册中心注册</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.12.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>

<maven.compiler.encoding>UTF-8</maven.compiler.encoding>

<java.version>1.8</java.version>

<spring-cloud.version>Hoxton.SR9</spring-cloud.version>

<jackson.version>2.12.1</jackson.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>

<!-- <dependency>

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

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency> -->

</dependencies>

</dependencyManagement>

<dependencies>

<!-- spring-boot 基础依赖包 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions><!-- 去掉默认配置 -->

<!-- jar 需要注释掉spring-boot-starter-tomcat相关依赖,war 必须放开 ；war包配置-->

<!-- <exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>-->

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</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-zuul</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-jul</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!--为了测试，使用jdbcTemplate jdbcTemplate -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<!--doGetAuthorizationInfo 不生效 AOP环境 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

<!--

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>-->

<!--freemarker支持 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-freemarker</artifactId>

</dependency>-->

<!-- <dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-juli</artifactId>

<version>${tomcat.version}</version>

</dependency>-->

<!-- mybatis plus -->

<!--<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>${mybatis-plus.version}</version>

</dependency> -->

<!-- rabbitMQ -->

<!--

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

-->

<!-- shiro -->

<!--<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.7.1</version>

</dependency>-->

<!-- https://mvnrepository.com/artifact/com.auth0/java-jwt -->

<!-- 动态数据源 -->

<!--

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>dynamic-datasource-spring-boot-starter</artifactId>

<version>2.5.4</version>

</dependency>-->

<!-- mybatis plus 代码生成器依赖 -->

<!--<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-generator</artifactId>

<version>${mybatis-plus.version}</version>

</dependency>-->

<!-- Velocity 模板-->

<!-- Activiti生成流程图 -->

<!-- junit -->

<!-- <dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

默认的版本为3.8.1，修改为4.x,因为3.x使用的为编程的方式，4.x为注解的形式。

</dependency> -->

<!-- 支持jsp的jar包用于编译jsp -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

<!-- jstl标签 依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- tomcat 的支持 . jar 屏蔽掉，不需要 ；war包需要 ;war包配置-->

<!--打包的时候可以不用包进去，别的设施会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。 相当于compile，但是打包阶段做了exclude操作 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>-->

<!--引入thymeleaf的依赖 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!--Spring Boot数据库资源 -->

<!-- https://mvnrepository.com/artifact/com.oracle/ojdbc6 -->

<!--<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.3</version>

</dependency>-->

<!-- 工具包 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.78</version>

</dependency>

<!-- Guava Jave库 -->

<dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>29.0-jre</version>

</dependency>

<!--lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--swagger2 -->

<!-- 添加 Scheduled 坐标 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpcore</artifactId>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-text</artifactId>

<version>1.9</version>

</dependency>

<dependency>

<groupId>cn.hutool</groupId>

<artifactId>hutool-all</artifactId>

<version>5.7.4</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.10.0</version>

</dependency>

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.4</version>

</dependency>

</dependencies>

<build>

<finalName>eureka-client1</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<mainClass>com.hpi.StartApplication</mainClass>

<includeSystemScope>true</includeSystemScope>

</configuration>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

<!-- 改成jar取消，war必须放开 ；war包配置，同时会导致系统导入的jar丢失，需做配置，在百度搜-->

<!-- <plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

</plugin> -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<verbose>true</verbose>

<!-- 项目编码-->

<encoding>UTF-8</encoding>

<fork>true</fork>

<executable>${JAVA8\_HOME}/bin/javac</executable>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skipTests>true</skipTests> <!--打包过程默认关掉单元测试 -->

</configuration>

</plugin>

<!--<plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-surefire-plugin</artifactId>

<configuration> <skip>true</skip>跳过单元测试 <testFailureIgnore>true</testFailureIgnore>这个网上很多的解决方式是这个,其实这个,其实这个配置后打包还是会编译单元测试类的,只是忽略编译单元测试类的错误.

</configuration> </plugin> -->

</plugins>

<resources>

<!-- 打包时将jsp文件拷贝到META-INF目录下 -->

<resource>

<!-- 指定resources插件处理哪个目录下的资源文件 -->

<directory>src/main/templates</directory>

<!--注意此次必须要放在此目录下才能被访问到 -->

<targetPath>META-INF/resources</targetPath>

<includes>

<include>\*\*/\*\*</include>

</includes>

</resource>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<!-- 是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*\*</include>

</includes>

<!--是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

</resources>

<defaultGoal>compile</defaultGoal>

</build>

<!--注意：这里必须要添加，否则各种依赖有问题-->

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/libs-milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

## 编写一个Controller，提供REST服务

@RestController

@RequestMapping("/user")

* **public** **class** UserController

{

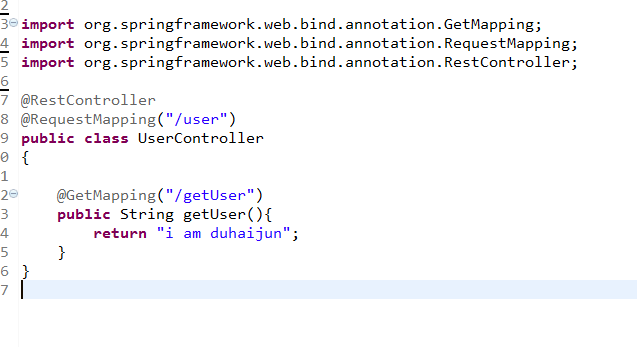
@GetMapping("/getUser")

**public** String getUser(){

**return** "i am duhaijun";

}

}



## 编写Springboot启动器

编写Springboot启动器，需要加上@EnableEurekaClient注解声明该项目为eureka的客户端。

启动类标注上@EnableEurekaClient或者@EnableDiscoveryClient二选一，eureka注册中心建议是EnableEurekaClient，其他建议是EnableDiscoveryClient

代码如下：

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.**class**})//取消数据库配置

//@ServletComponentScan(basePackages = "com.hpi.system.filters")

@SpringBootApplication

@ComponentScan(basePackages

= {

"com.hpi.common",

"com.hpi.modules",

"com.hpi.system"

})

@EnableCaching // 开启缓存注解

@EnableEurekaClient

//@EnableDiscoveryClient//其他注册中心用这个

**public** **class** StartApplication **extends** SpringBootServletInitializer

{

/\*\*

\* **@方法说明**: **TODO**

\* **@参数**： **@param** args

\* **@返回值**： void

\* **@异常**：

\* **@作者**： duhj

\* **@创建日期** 2019-6-17

\*

\* 历史记录

\* 1、修改日期：

\* 修改人：

\* 修改内容：

\*/

**public** **static** **void** main(String[] args)

{

// System.setProperty("spring.devtools.restart.enabled", "false");//完全关闭重启支持

SpringApplication.*run*(StartApplication.**class**, args);

}

}



## 修改application.properties

增加如下：

spring.application.name=servcie-provider

server.port=9000

#eureka

#eureka.instance.hostname=localhost

#false表示不向服务中心注册自己

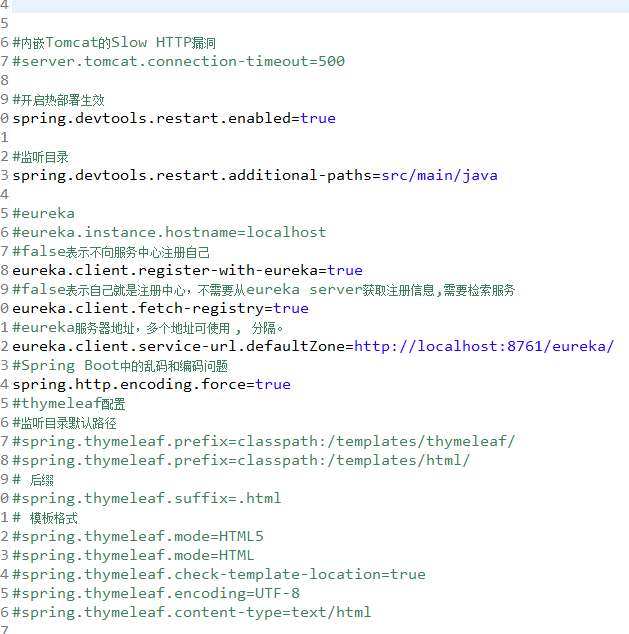
eureka.client.register-with-eureka=true

#false表示自己就是注册中心，不需要从eureka server获取注册信息,需要检索服务

eureka.client.fetch-registry=true

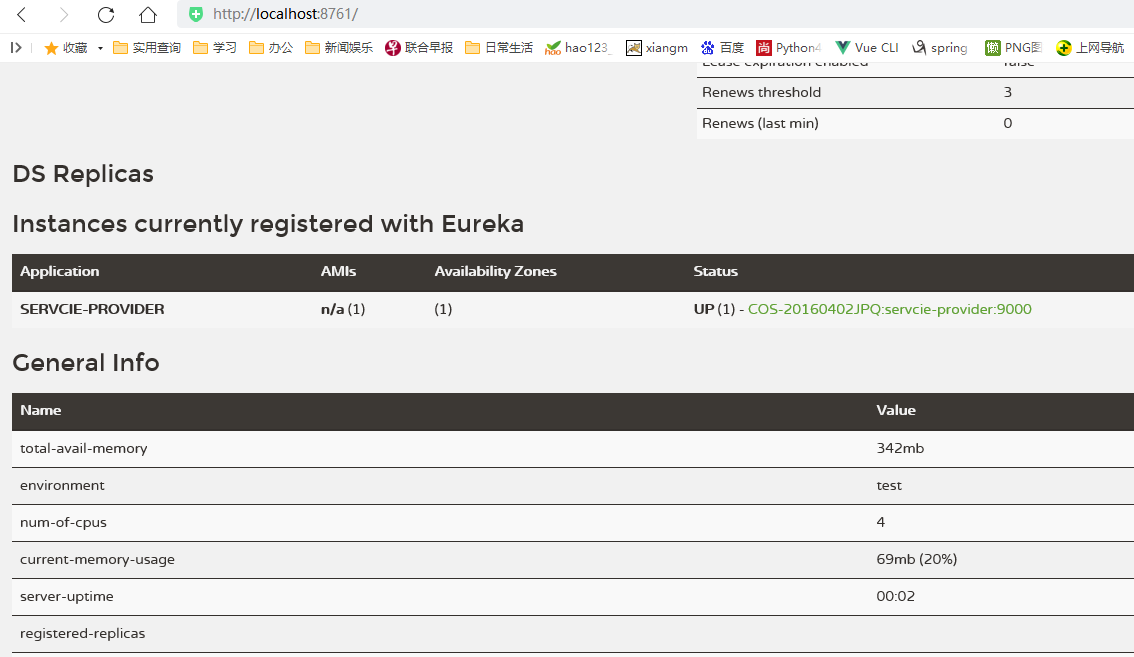
#eureka服务地址，多个地址可使用 , 分隔。

eureka.client.service-url.defaultZone=http://localhost:8761/eureka/



## 启动服务，看是否注册成功

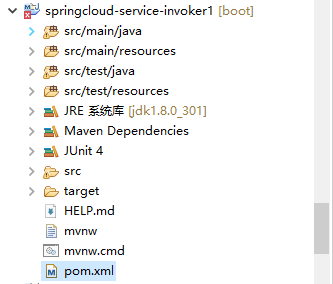
启动后，打开Eureka服务器地址<http://localhost:8761/>，看是否有该服务



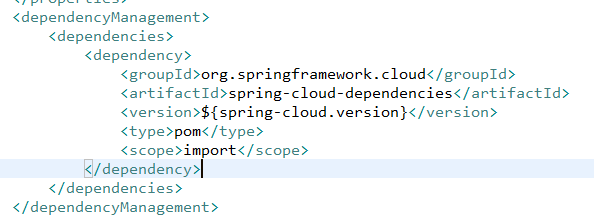
# 编写服务调用者（invoker）（使用openfeign）

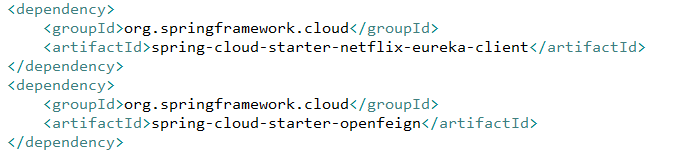
服务接口调用（客户端调用服务的简化工具），openfeign算是 Ribbon（负载均衡）和RestTemplate（服务和服务之间调用）的结合体。

## 创建invoker项目



Pom增加如下内容：





具体如下：

<?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>com.hpi</groupId>

<artifactId>springboot-start</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- 打包机制，如pom,jar,maven-plugin,ejb,war,ear,rar,par 默认为jar方式 ；war包配置-->

<packaging>jar</packaging>

<!--<packaging>war</packaging>-->

<name>springboot-start</name>

<description>service-invoker,服务调用者,Fegin的使用（默认使用了Ribbon自动调用负载均衡作用）</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.12.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>

<maven.compiler.encoding>UTF-8</maven.compiler.encoding>

<java.version>1.8</java.version>

<spring-cloud.version>Hoxton.SR9</spring-cloud.version>

<jackson.version>2.12.1</jackson.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>

<dependencies>

<!-- spring-boot 基础依赖包 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions><!-- 去掉默认配置 -->

<!-- jar 需要注释掉spring-boot-starter-tomcat相关依赖,war 必须放开 ；war包配置-->

<!-- <exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>-->

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</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-openfeign</artifactId>

</dependency>

<!-- <dependency>

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

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

<version>1.4.7.RELEASE</version>

</dependency>

<dependency>

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

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

<version>1.4.7.RELEASE</version>

</dependency> -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-jul</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.17.0</version>

</dependency>

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency> -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency> -->

<!--为了测试，使用jdbcTemplate jdbcTemplate -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency> -->

<!--doGetAuthorizationInfo 不生效 AOP环境 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency> -->

<!--

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>-->

<!--freemarker支持 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-freemarker</artifactId>

</dependency>-->

<!-- <dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-juli</artifactId>

<version>${tomcat.version}</version>

</dependency>-->

<!-- mybatis plus -->

<!--<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>${mybatis-plus.version}</version>

</dependency> -->

<!-- rabbitMQ -->

<!--

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

-->

<!-- shiro -->

<!--<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.7.1</version>

</dependency>-->

<!-- https://mvnrepository.com/artifact/com.auth0/java-jwt -->

<!-- 动态数据源 -->

<!--

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>dynamic-datasource-spring-boot-starter</artifactId>

<version>2.5.4</version>

</dependency>-->

<!-- mybatis plus 代码生成器依赖 -->

<!--<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-generator</artifactId>

<version>${mybatis-plus.version}</version>

</dependency>-->

<!-- Velocity 模板-->

<!-- Activiti生成流程图 -->

<!-- junit -->

<!-- <dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

默认的版本为3.8.1，修改为4.x,因为3.x使用的为编程的方式，4.x为注解的形式。

</dependency> -->

<!-- 支持jsp的jar包用于编译jsp -->

<!-- <dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

jstl标签 依赖

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency> -->

<!-- tomcat 的支持 . jar 屏蔽掉，不需要 ；war包需要 ;war包配置-->

<!--打包的时候可以不用包进去，别的设施会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。 相当于compile，但是打包阶段做了exclude操作 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>-->

<!--引入thymeleaf的依赖 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>-->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency> -->

<!--Spring Boot数据库资源 -->

<!-- https://mvnrepository.com/artifact/com.oracle/ojdbc6 -->

<!--<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.3</version>

</dependency>-->

<!-- 工具包 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.78</version>

</dependency>

<!-- Guava Jave库 -->

<!-- <dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>29.0-jre</version>

</dependency> -->

<!--lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--swagger2 -->

<!-- 添加 Scheduled 坐标 -->

<!-- <dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpcore</artifactId>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

</dependency> -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-text</artifactId>

<version>1.9</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.10.0</version>

</dependency>

</dependencies>

<build>

<finalName>feign</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<mainClass>com.hpi.StartApplication</mainClass>

<includeSystemScope>true</includeSystemScope>

</configuration>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

<!-- 改成jar取消，war必须放开 ；war包配置，同时会导致系统导入的jar丢失，需做配置，在百度搜-->

<!-- <plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

</plugin> -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<verbose>true</verbose>

<!-- 项目编码-->

<encoding>UTF-8</encoding>

<fork>true</fork>

<executable>${JAVA8\_HOME}/bin/javac</executable>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skipTests>true</skipTests> <!--打包过程默认关掉单元测试 -->

</configuration>

</plugin>

<!--<plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-surefire-plugin</artifactId>

<configuration> <skip>true</skip>跳过单元测试 <testFailureIgnore>true</testFailureIgnore>这个网上很多的解决方式是这个,其实这个,其实这个配置后打包还是会编译单元测试类的,只是忽略编译单元测试类的错误.

</configuration> </plugin> -->

</plugins>

<resources>

<!-- 打包时将jsp文件拷贝到META-INF目录下 -->

<resource>

<!-- 指定resources插件处理哪个目录下的资源文件 -->

<directory>src/main/templates</directory>

<!--注意此次必须要放在此目录下才能被访问到 -->

<targetPath>META-INF/resources</targetPath>

<includes>

<include>\*\*/\*\*</include>

</includes>

</resource>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<!-- 是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*\*</include>

</includes>

<!--是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

</resources>

<defaultGoal>compile</defaultGoal>

</build>

<!-- 添加私服仓库地址，否则找不到包 -->

<!-- <repositories>

<repository>

<id>alfresco</id>

<name>Activiti Releases</name>

<url>https://artifacts.alfresco.com/nexus/content/repositories/activiti-releases/</url>

<releases>

<enabled>true</enabled>

</releases>

</repository>

</repositories> -->

<!--注意：这里必须要添加，否则各种依赖有问题-->

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/libs-milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

## 修改application.properties

主要增加如下：

spring.application.name=server-invoker

server.port=8000

#主机名

eureka.instance.hostname= localhost

#eureka服务器地址，多个地址可使用 , 分隔。

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

## 编写远程调用其他微服务的接口

package com.hpi.modules.user.service;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.web.bind.annotation.GetMapping;

//@FeignClient注解中的值表示调用服务名为service-provider的节点,即注册中心注册中心Application 的name值，@RequestMapping注解声明了调用该节点的哪个路由。

//@FeginClient标注value属性写提供者provider 中的application.yml里面的spring.application.name，也是注册中心Application name 的值，这三者是一至的。

@FeignClient("servcie-provider")

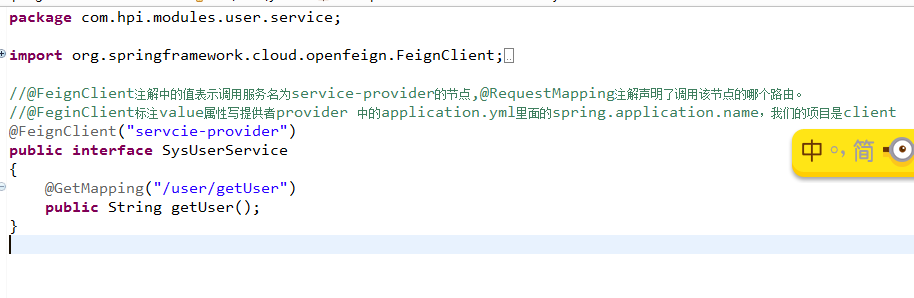
public interface SysUserService

{

@GetMapping("/user/getUser")

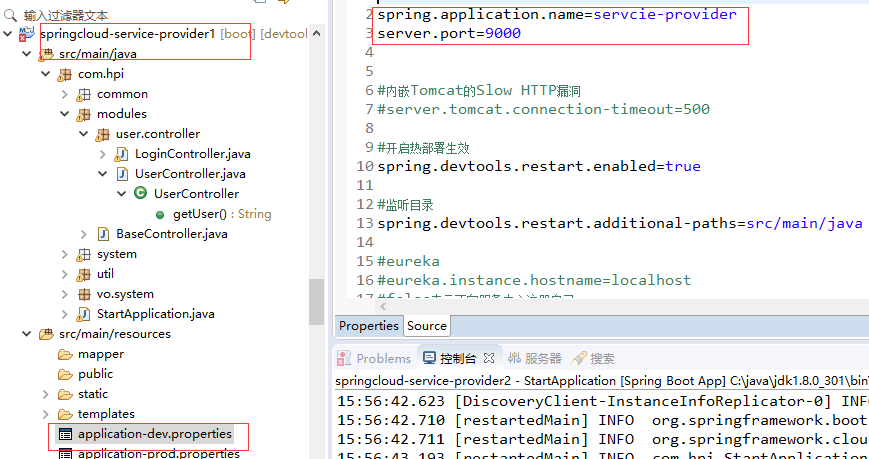
public String getUser();

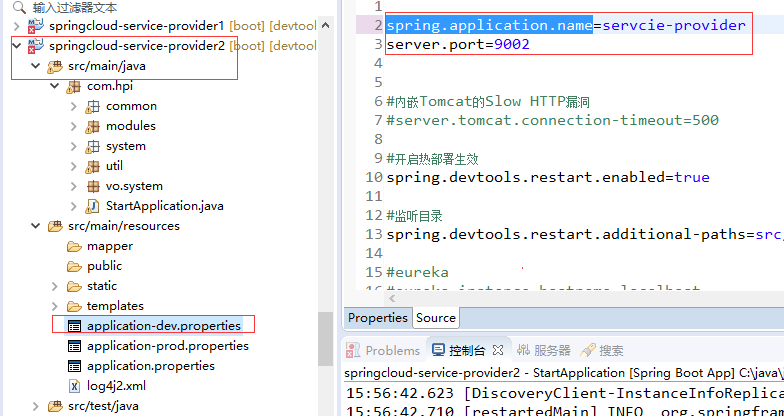
}



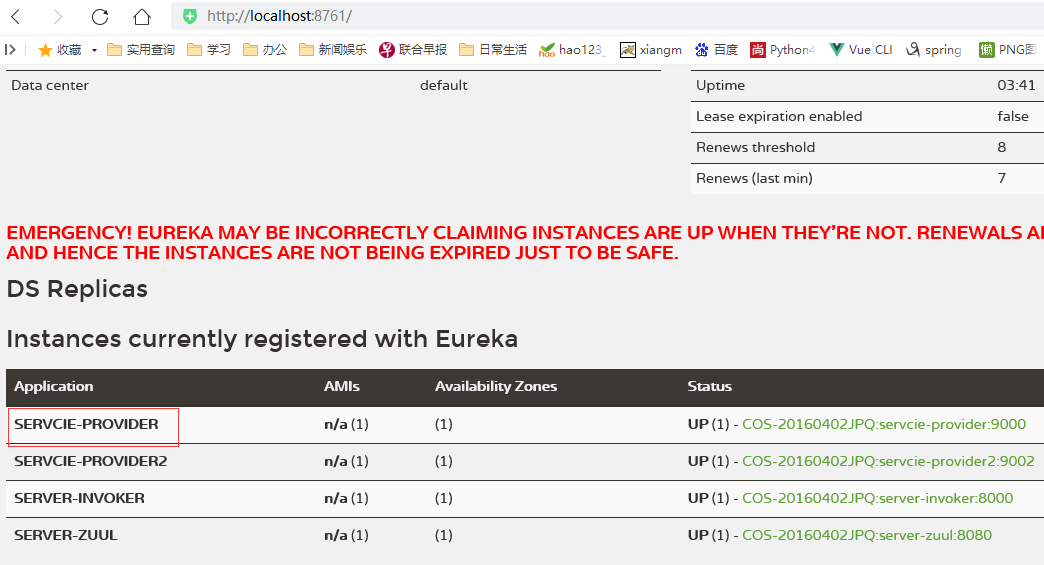
注意的是：

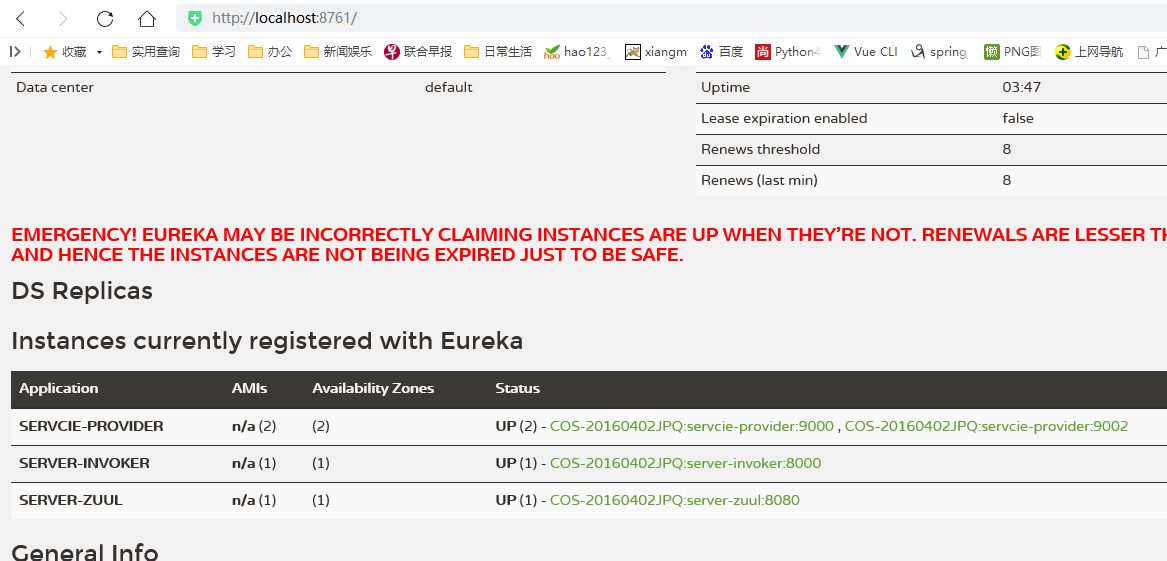
@FeginClient标注value属性写服务提供者（provider）application.yml里面的spring.application.name值，也是注册中心Application name 的值，这三者是一至的表示调用服务名为service-provider的节点





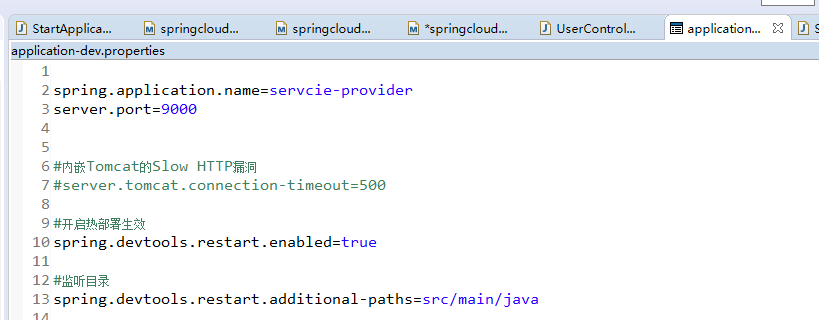
和注册中心Application对应的一致





@RequestMapping或者GetMapping注解声明了调用该节点的哪个路由。

即



如果调用的服务接口有接收参数一定要带上@RequestParam之类的标注否则会传送不了参数

## 编写Controller，对外提供服务

编写一个Controller，对外提供服务，使用SysUserService接口（代理对象）完成服务调用。

**package** com.hpi.modules.user.controller;

**import** org.springframework.beans.factory.annotation.Autowired;

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

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

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

**import** com.hpi.modules.user.service.SysUserService;

@RestController

@RequestMapping("/user")

**public** **class** UserController

{

@Autowired

**private** SysUserService sysUserService;

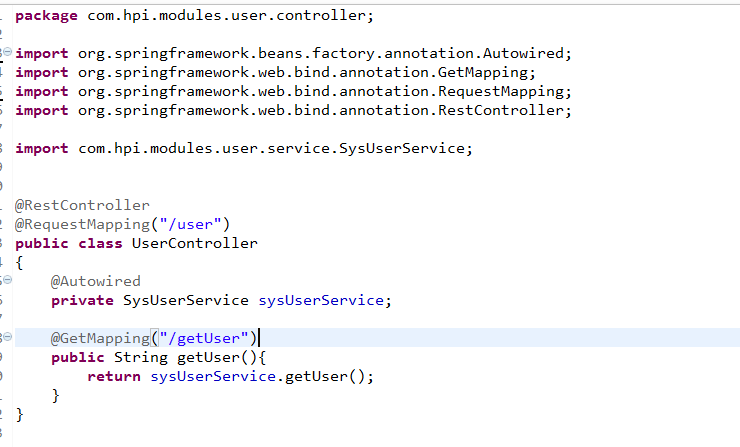
@GetMapping("/getUser")

**public** String getUser(){

**return** sysUserService.getUser();

}

}



## 编写Springboot启动器

编写Springboot启动器，需要加上@EnableDiscoveryClient注解声明有能力发现eureka中的服务，加上@EnableFeignClients打开Feign开关。启动器代码如下：

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.**class**})//取消数据库配置

//@ServletComponentScan(basePackages = "com.hpi.system.filters")

@SpringBootApplication

@ComponentScan(basePackages

= {

"com.hpi.common",

"com.hpi.modules",

"com.hpi.system"

})

@EnableEurekaClient

@EnableFeignClients(basePackages = "com.hpi.modules")

**public** **class** StartApplication **extends** SpringBootServletInitializer

{

/\*\*

\* **@方法说明**: **TODO**

\* **@参数**： **@param** args

\* **@返回值**： void

\* **@异常**：

\* **@作者**： duhj

\* **@创建日期** 2019-6-17

\*

\* 历史记录

\* 1、修改日期：

\* 修改人：

\* 修改内容：

\*/

**public** **static** **void** main(String[] args)

{

// System.setProperty("spring.devtools.restart.enabled", "false");//完全关闭重启支持

SpringApplication.*run*(StartApplication.**class**, args);

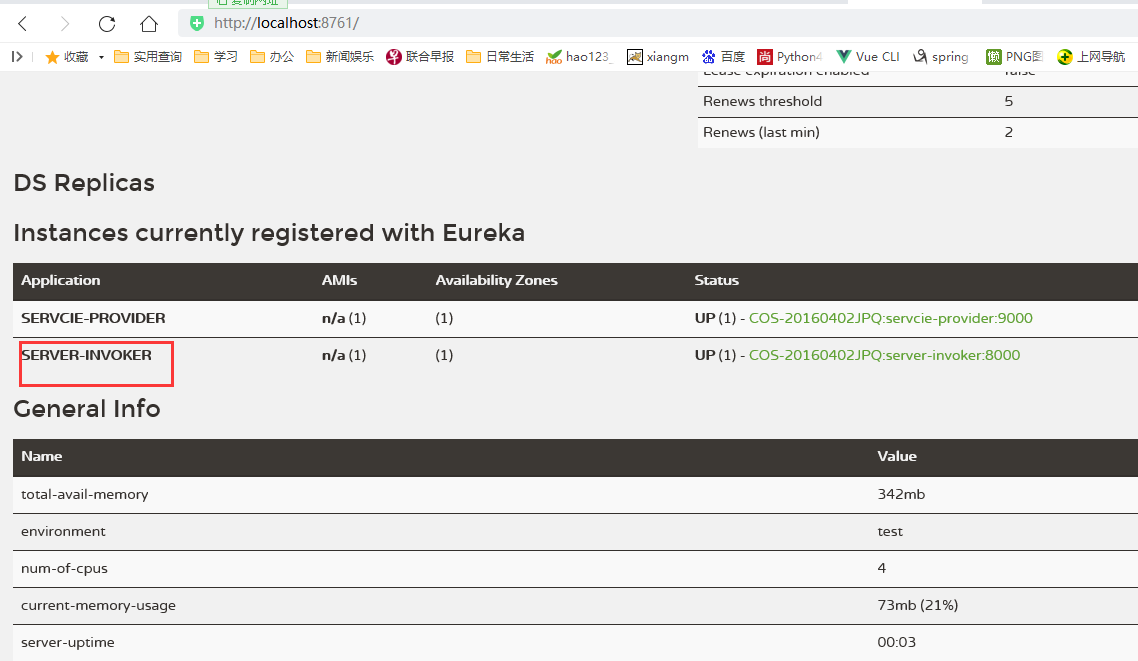
}

}

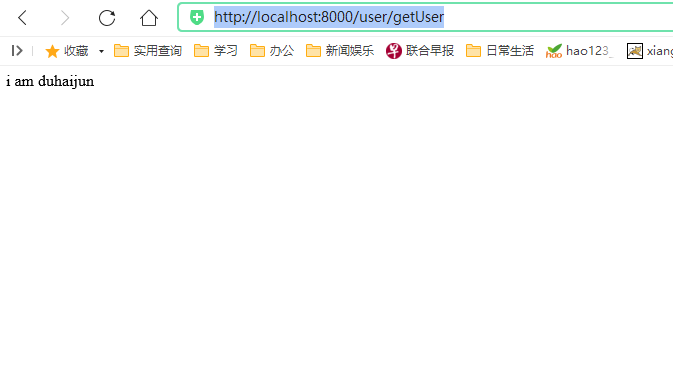


## 启动服务，检查是否注册成功

输入Eureka服务器地址（http://localhost:8761/），进入注册中心，看是否注册成功



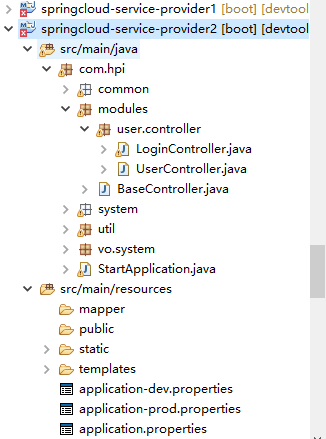
访问<http://localhost:8000/user/getUser>



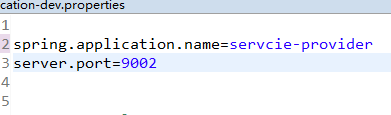
表明注册成功

## 路由测试

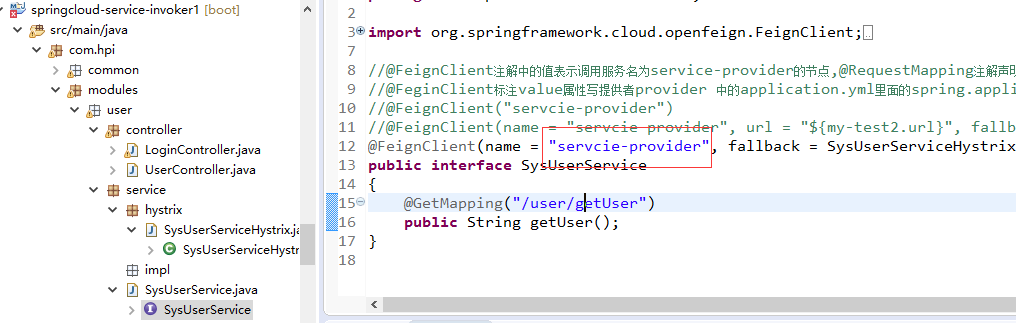
创建多个服务提供者（provider），比如我们再创建springcloud-service-provider2，



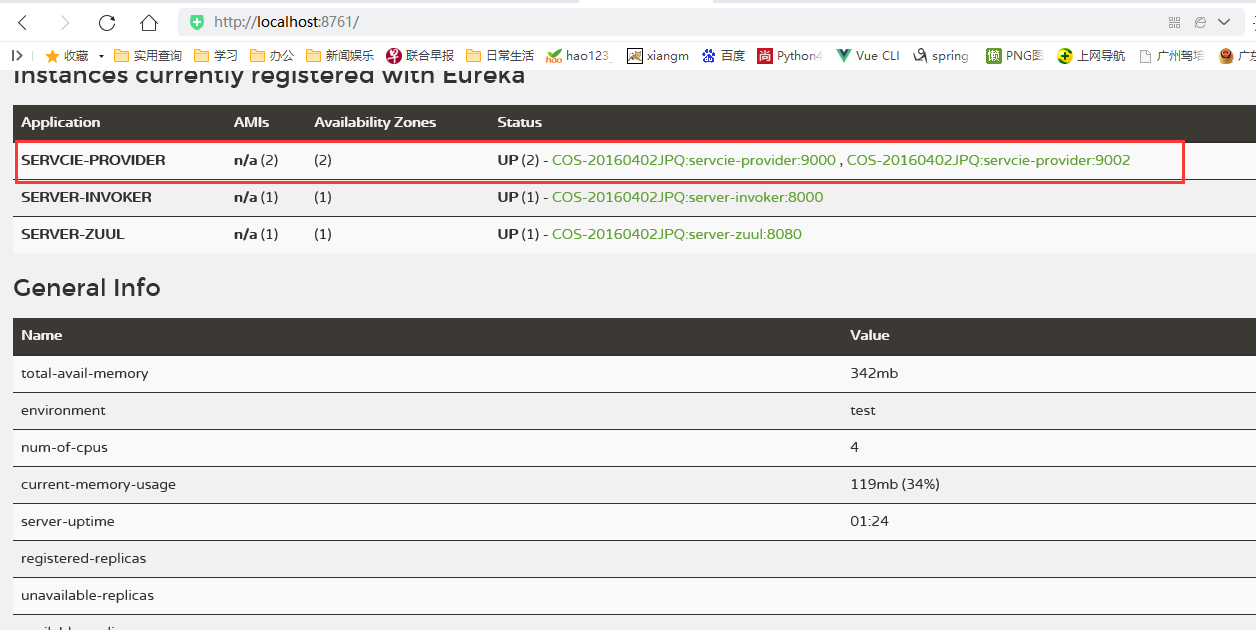
修改端口为9002，其他不变



注意：spring.application.name值不能变，要和调用者FeignClient 中的name 一致，否则不能路由。如下图：

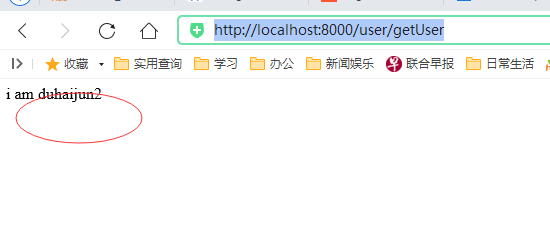


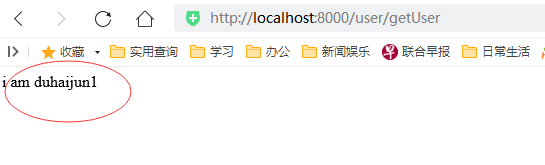
进入服务，就看到2个提供者服务



测试<http://localhost:8000/user/getUser>

会看到在两个提供者之间切换





## Hystrix（熔断器）配置 -普通降级

Hystrix是一个用于处理分布式系统的延迟和容错的开元库,在分布式系统里,许多以来不可避免的会调用失败,比如超时,异常等,Hystrix能够保证在一个依赖出问题的情况下,不会导致整体服务失败,避免级联鼓掌,以提高分布式系统的弹性

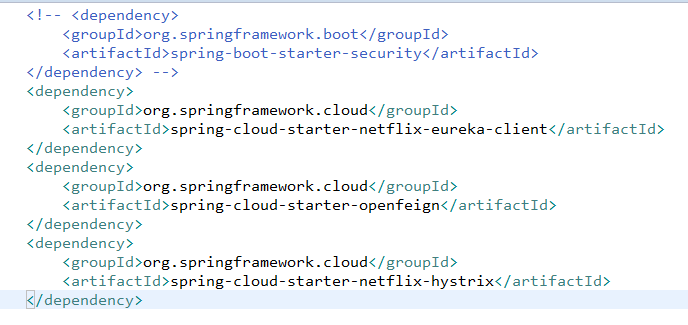
### 3.8.1给pom加入Hystrix坐标

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

****

### 3.8.2在启动类加上@EnableCircuitBreaker或@EnableHystrix

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.**class**})//取消数据库配置

//@ServletComponentScan(basePackages = "com.hpi.system.filters")

@SpringBootApplication

@ComponentScan(basePackages

= {

"com.hpi.common",

"com.hpi.modules",

"com.hpi.system"

})

@EnableEurekaClient

@EnableFeignClients(basePackages = "com.hpi.modules")

@EnableHystrix

**public** **class** StartApplication **extends** SpringBootServletInitializer

{

/\*\*

\* **@方法说明**: **TODO**

\* **@参数**： **@param** args

\* **@返回值**： void

\* **@异常**：

\* **@作者**： duhj

\* **@创建日期** 2019-6-17

\*

\* 历史记录

\* 1、修改日期：

\* 修改人：

\* 修改内容：

\*/

**public** **static** **void** main(String[] args)

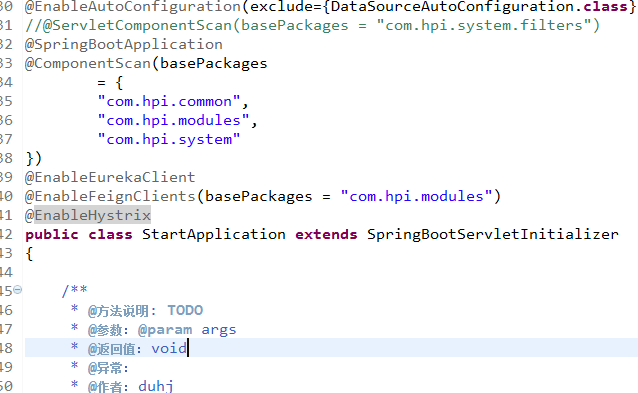
{

// System.setProperty("spring.devtools.restart.enabled", "false");//完全关闭重启支持

SpringApplication.*run*(StartApplication.**class**, args);

}

}



### 3.8.3修改配置文件application.properties

给配置文件增加如下：

#开启feign支持hystrix

#开启熔断器hystrix

feign.hystrix.enabled=true

#指的是建立连接后从服务器读取到可用资源所用的时间。

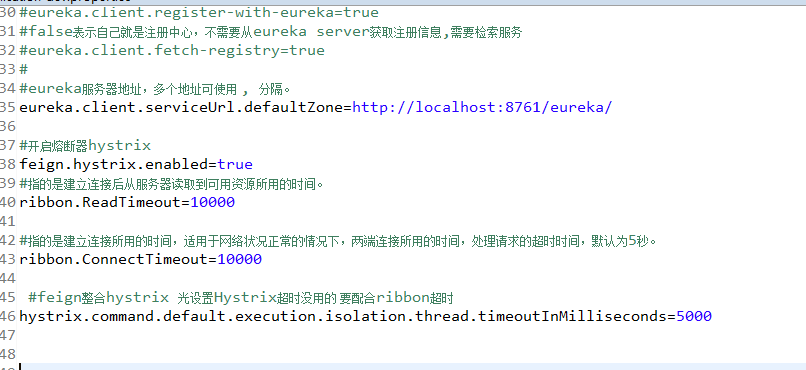
ribbon.ReadTimeout=10000

#指的是建立连接所用的时间，适用于网络状况正常的情况下，两端连接所用的时间，处理请求的超时时间，默认为5秒。

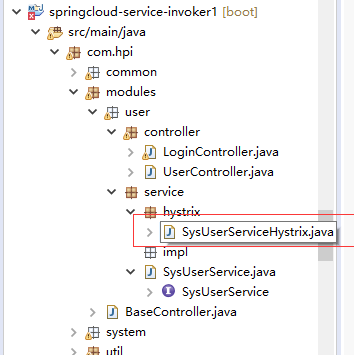
ribbon.ConnectTimeout=10000

#feign整合hystrix 光设置Hystrix超时没用的 要配合ribbon超时

hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=5000



### 3.8.4创建一个类实现服务FeignClient接口



package com.hpi.modules.user.service.hystrix;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.stereotype.Component;

import com.hpi.modules.user.service.SysUserService;

@Component

public class SysUserServiceHystrix implements SysUserService

{

@Override

public String getUser()

{

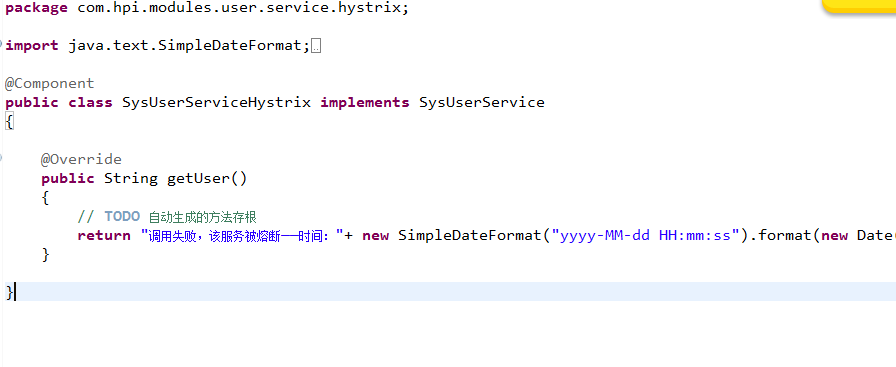
// TODO 自动生成的方法存根

return "调用失败，该服务被熔断——时间："+ new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());

}

}

如下图



### 3.8.5 Fegin接口的@FeignClient注解加上fallback属性

@FeignClient(name = "servcie-provider", fallback = SysUserServiceHystrix.**class**)

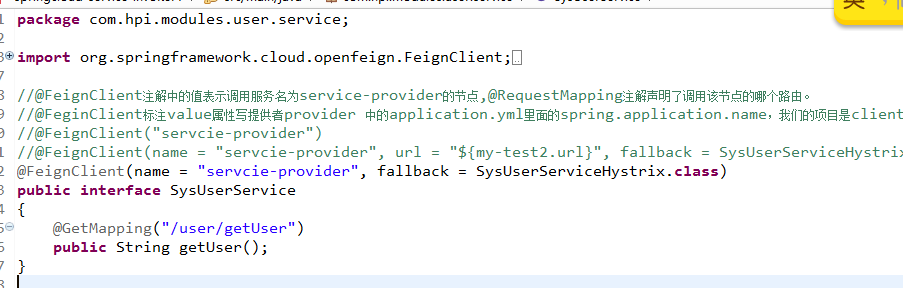
**public** **interface** SysUserService

{

@GetMapping("/user/getUser")

**public** String getUser();

}



### 3.8.6模拟测试

为了测试服务超时场景，我们在springcloud-service-provider1的控制器中增加超时*sleep*

@RequestMapping("/user")

**public** **class** UserController

{

@GetMapping("/getUser")

**public** String getUser(){

//用来模拟服务超时

**try** {

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

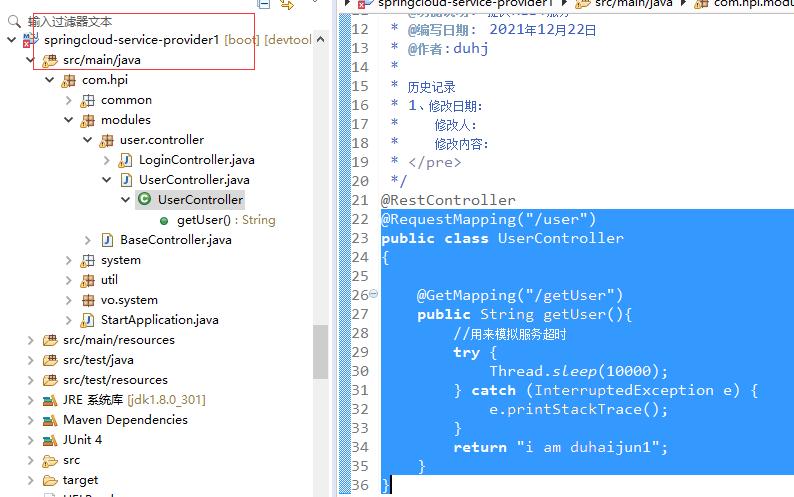
e.printStackTrace();

}

**return** "i am duhaijun1";

}

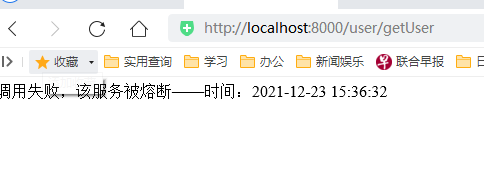
}



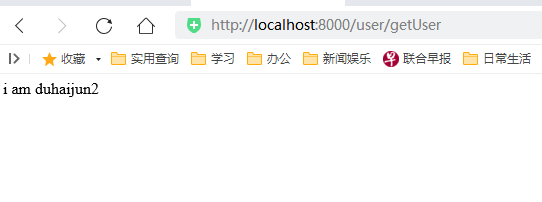
超过配置文件中hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds时间

测试：

如果是provider1，则

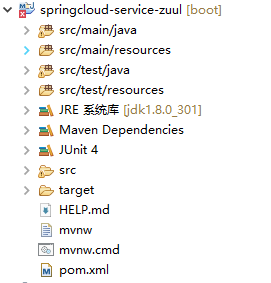


如果是如果是provider2，则正常

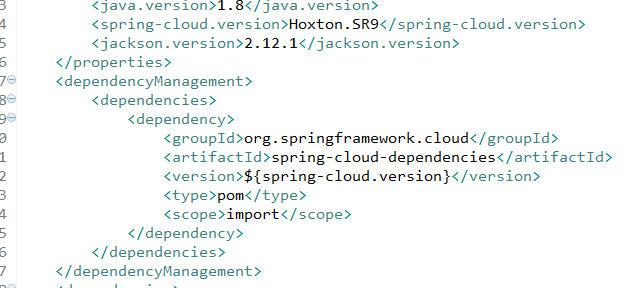


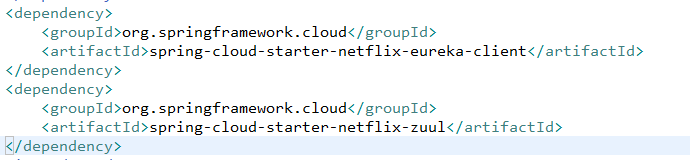
# 编写Zuul网关（service-zuul）

## 创建Zuul项目



Pom增加如下：





具体如下：

<?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>com.hpi</groupId>

<artifactId>springboot-start</artifactId>

<version>0.0.1-SNAPSHOT</version>

<!-- 打包机制，如pom,jar,maven-plugin,ejb,war,ear,rar,par 默认为jar方式 ；war包配置-->

<packaging>jar</packaging>

<!--<packaging>war</packaging>-->

<name>springboot-start</name>

<description>service-zuul，Zuul网关</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.12.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>

<maven.compiler.encoding>UTF-8</maven.compiler.encoding>

<java.version>1.8</java.version>

<spring-cloud.version>Hoxton.SR9</spring-cloud.version>

<jackson.version>2.12.1</jackson.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>

<dependencies>

<!-- spring-boot 基础依赖包 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions><!-- 去掉默认配置 -->

<!-- jar 需要注释掉spring-boot-starter-tomcat相关依赖,war 必须放开 ；war包配置-->

<!-- <exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>-->

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</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-zuul</artifactId>

</dependency>

<!-- <dependency>

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

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

<version>1.4.7.RELEASE</version>

</dependency>

<dependency>

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

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

<version>1.4.7.RELEASE</version>

</dependency> -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-jul</artifactId>

<version>2.17.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.17.0</version>

</dependency>

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency> -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency> -->

<!--为了测试，使用jdbcTemplate jdbcTemplate -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency> -->

<!--doGetAuthorizationInfo 不生效 AOP环境 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency> -->

<!--

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-cache</artifactId>

</dependency>-->

<!--freemarker支持 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-freemarker</artifactId>

</dependency>-->

<!-- <dependency>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-juli</artifactId>

<version>${tomcat.version}</version>

</dependency>-->

<!-- mybatis plus -->

<!--<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>${mybatis-plus.version}</version>

</dependency> -->

<!-- rabbitMQ -->

<!--

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

-->

<!-- shiro -->

<!--<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.7.1</version>

</dependency>-->

<!-- https://mvnrepository.com/artifact/com.auth0/java-jwt -->

<!-- 动态数据源 -->

<!--

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>dynamic-datasource-spring-boot-starter</artifactId>

<version>2.5.4</version>

</dependency>-->

<!-- mybatis plus 代码生成器依赖 -->

<!--<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-generator</artifactId>

<version>${mybatis-plus.version}</version>

</dependency>-->

<!-- Velocity 模板-->

<!-- Activiti生成流程图 -->

<!-- junit -->

<!-- <dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

默认的版本为3.8.1，修改为4.x,因为3.x使用的为编程的方式，4.x为注解的形式。

</dependency> -->

<!-- 支持jsp的jar包用于编译jsp -->

<!-- <dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

jstl标签 依赖

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency> -->

<!-- tomcat 的支持 . jar 屏蔽掉，不需要 ；war包需要 ;war包配置-->

<!--打包的时候可以不用包进去，别的设施会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。 相当于compile，但是打包阶段做了exclude操作 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>-->

<!--引入thymeleaf的依赖 -->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>-->

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency> -->

<!--Spring Boot数据库资源 -->

<!-- https://mvnrepository.com/artifact/com.oracle/ojdbc6 -->

<!--<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.3</version>

</dependency>-->

<!-- 工具包 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.78</version>

</dependency>

<!-- Guava Jave库 -->

<!-- <dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>29.0-jre</version>

</dependency> -->

<!--lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--swagger2 -->

<!-- 添加 Scheduled 坐标 -->

<!-- <dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpcore</artifactId>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

</dependency> -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-text</artifactId>

<version>1.9</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.10.0</version>

</dependency>

</dependencies>

<build>

<finalName>zuul</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<mainClass>com.hpi.StartApplication</mainClass>

<includeSystemScope>true</includeSystemScope>

</configuration>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

<!-- 改成jar取消，war必须放开 ；war包配置，同时会导致系统导入的jar丢失，需做配置，在百度搜-->

<!-- <plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

</plugin> -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<verbose>true</verbose>

<!-- 项目编码-->

<encoding>UTF-8</encoding>

<fork>true</fork>

<executable>${JAVA8\_HOME}/bin/javac</executable>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<skipTests>true</skipTests> <!--打包过程默认关掉单元测试 -->

</configuration>

</plugin>

<!--<plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-surefire-plugin</artifactId>

<configuration> <skip>true</skip>跳过单元测试 <testFailureIgnore>true</testFailureIgnore>这个网上很多的解决方式是这个,其实这个,其实这个配置后打包还是会编译单元测试类的,只是忽略编译单元测试类的错误.

</configuration> </plugin> -->

</plugins>

<resources>

<!-- 打包时将jsp文件拷贝到META-INF目录下 -->

<resource>

<!-- 指定resources插件处理哪个目录下的资源文件 -->

<directory>src/main/templates</directory>

<!--注意此次必须要放在此目录下才能被访问到 -->

<targetPath>META-INF/resources</targetPath>

<includes>

<include>\*\*/\*\*</include>

</includes>

</resource>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.properties</include>

<include>\*\*/\*.xml</include>

</includes>

<!-- 是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*\*</include>

</includes>

<!--是否替换资源中的属性-->

<filtering>false</filtering>

</resource>

</resources>

<defaultGoal>compile</defaultGoal>

</build>

<!-- 添加私服仓库地址，否则找不到包 -->

<!-- <repositories>

<repository>

<id>alfresco</id>

<name>Activiti Releases</name>

<url>https://artifacts.alfresco.com/nexus/content/repositories/activiti-releases/</url>

<releases>

<enabled>true</enabled>

</releases>

</repository>

</repositories> -->

<!--注意：这里必须要添加，否则各种依赖有问题-->

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/libs-milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

## 编写Springboot的启动类

编写一个Springboot的启动类，并加上@EnableZuulProxy和EnableDiscoveryClient即可将此项目声明为zuul服务器。

在启动类上添加注解@EnableZuulProxy ，声明一个Zuul代理。该代理使用Ribbon来定位注册在Eureka Server中的微服务；

同时，该代理还整合了 Hystrix,从而实现了容错，所有经过Zuul的请求都会在Hystrix命令中执行。

启动类如下所示：

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

import org.springframework.context.annotation.ComponentScan;

/\*\*

\*

\* <pre>

\* @业务名:

\* @功能说明:

\* @编写日期: 2019-7-9

\* @作者: duhj

\*

\* 历史记录

\* 1、修改日期：

\* 修改人：

\* 修改内容：

\* </pre>

\*/

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})//取消数据库配置

//@ServletComponentScan(basePackages = "com.hpi.system.filters")

@SpringBootApplication

@ComponentScan(basePackages

= {

"com.hpi.common",

"com.hpi.modules",

"com.hpi.system"

})

@EnableZuulProxy

@EnableDiscoveryClient

public class StartApplication extends SpringBootServletInitializer

{

/\*\*

\* @方法说明: TODO

\* @参数： @param args

\* @返回值： void

\* @异常：

\* @作者： duhj

\* @创建日期 2019-6-17

\*

\* 历史记录

\* 1、修改日期：

\* 修改人：

\* 修改内容：

\*/

public static void main(String[] args)

{

// System.setProperty("spring.devtools.restart.enabled", "false");//完全关闭重启支持

SpringApplication.run(StartApplication.class, args);

}



## 修改application.properties

主要加入下：

spring.application.name=server-zuul

server.port=8080

#eureka

eureka.instance.hostname=localhost

#eureka服务器地址，多个地址可使用 , 分隔。

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

#配置下zuul的超时时间，因zuul启用了ribbon的负载均衡，还需要设置ribbon的超时时间，注意ribbon的超时时间要小于zuul超时时间 。

#HTTP连接超时要比Hystrix的大

zuul.host.connect-timeout-millis=15000

#socket超时

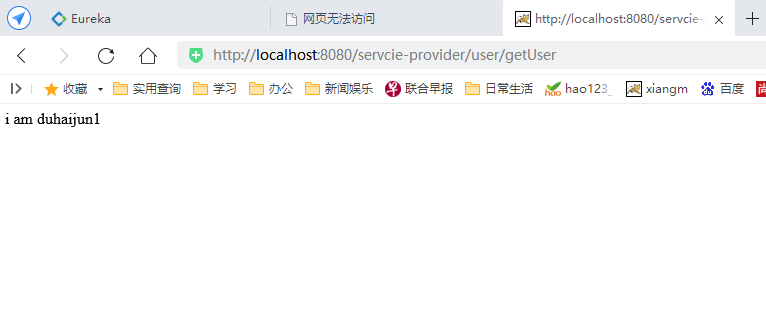
zuul.host.socket-timeout-millis=60000

ribbon.ReadTimeout=10000

ribbon.ConnectTimeout=10000

## 启动测试

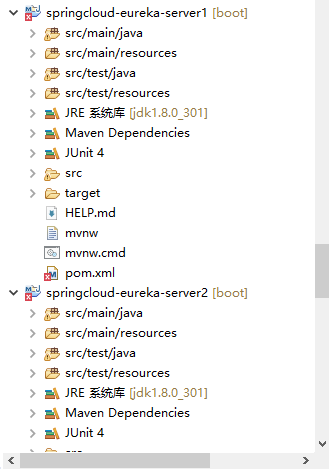
访问http://localhost:8080/servcie-provider/user/getUser, 请求方式为http://localhost:Zuul的端口/服务名/请求资源



# 集群

## 创建注册中心集群

创建两个集群中心，以实现集群为例。两个注册中心基本一致，不一致的是配置文件yml或者properties的配置方式变化，下图为两个注册中心

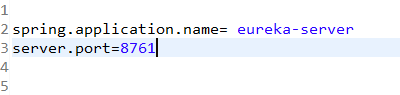


配置文件变化是：

eureka-server1和eureka-server1区别是：

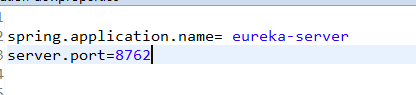
1. 端口不一致，暂定义eureka-server1端口是8761，eureka-server2端口是8762
2. eureka.client.service-url.defaultZone，这两个要相互注册，不要自己注册自己，多个以逗号隔开，

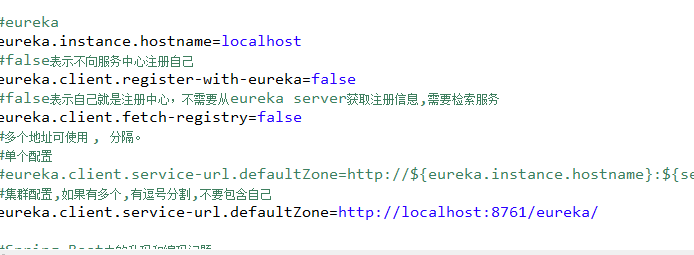
**eureka-server1的配置**





**eureka-server2的配置**





## 创建多个服务提供者provider

我们以两个服务者为例



两个服务者不同的也是yml或者properties配置文件的不同

区别是：

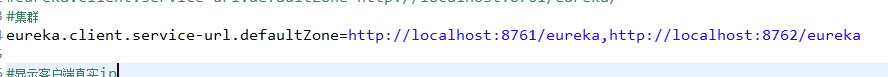
1. 端口

service-provider1端口是9000

service-provider2端口是9002

其他都一致，尤其是spring.application.name一致，才能为调用者提供相同的路由

eureka.client.service-url.defaultZone要配置成集群环境，如下图



具体配置如下：

**service-provider1：**



**service-provider2：**



## 创建多个服务调用者invoker

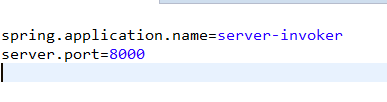
我们创建两个调用者，分别是service-invoker1和service-invoker2



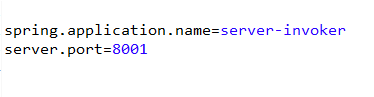
service-invoker1区别也是yml或者properties配置文件的不同

主要是端口的不同

service-invoker1端口是 8001

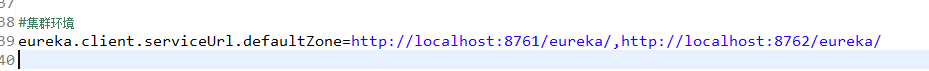


service-invoker2 端口是8001



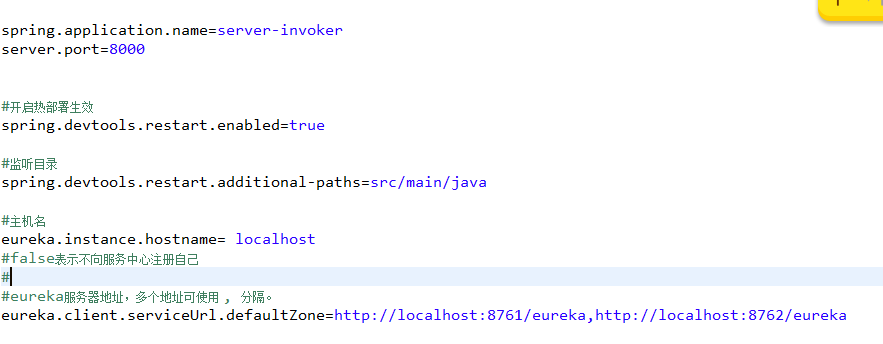
其他都一直

同样eureka.client.serviceUrl.defaultZone必须是集群环境

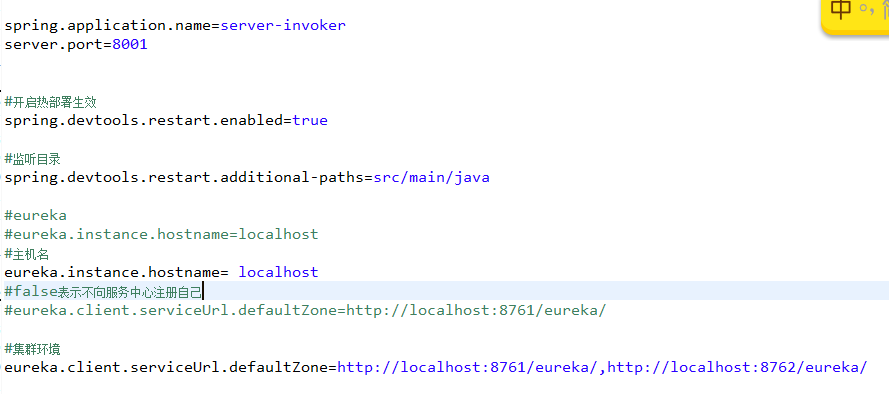


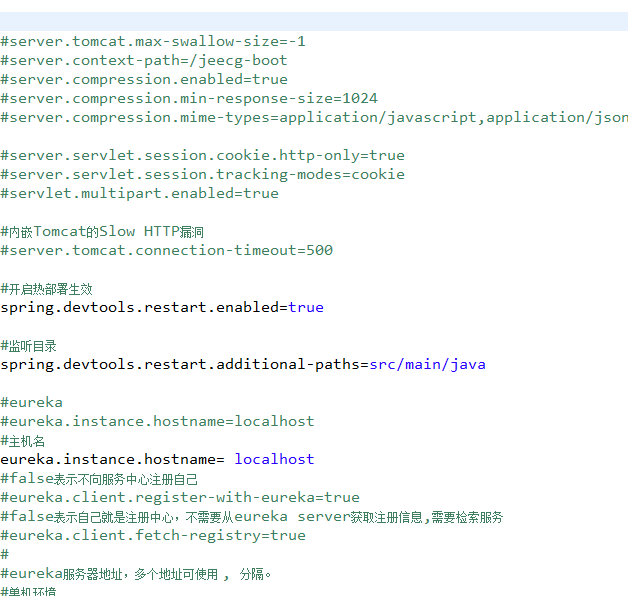
具体如下：

**service-invoker1：**



**service-invoker2：**





测试：

调用者1：<http://localhost:8000/user/getUser>

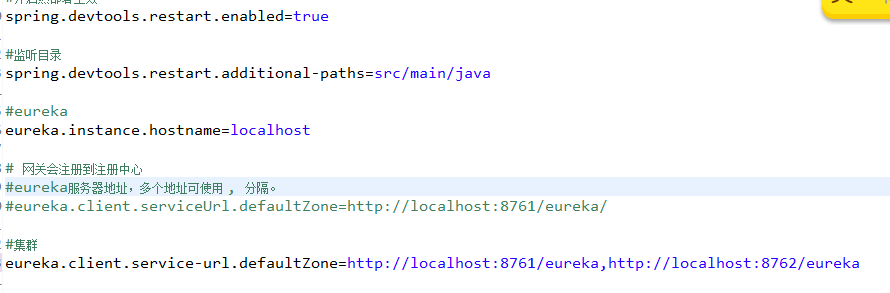
调用者2：http://localhost:8001/user/getUser

## 处理zuul server

只需修改yml或者properties的eureka.client.service-url.defaultZone

#集群

eureka.client.service-url.defaultZone=http://localhost:8761/eureka,http://localhost:8762/eureka



# 集成Security

## 注册中心Eureka

**1）加入spring-boot-starter-security依赖坐标**

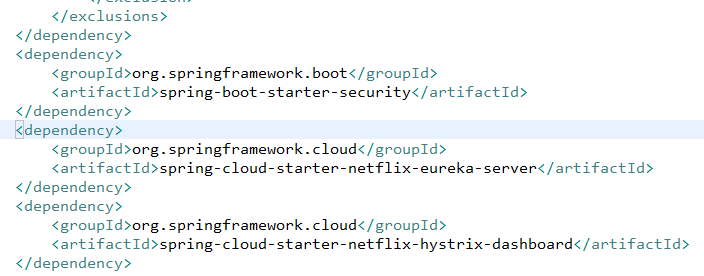
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

如下图：

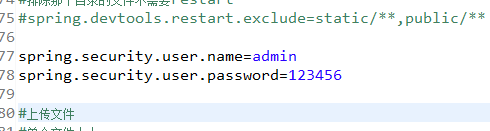


**2）修改配置文件yml或者properties**

在配置文件中增加配置用户名密码

spring.security.user.name=admin

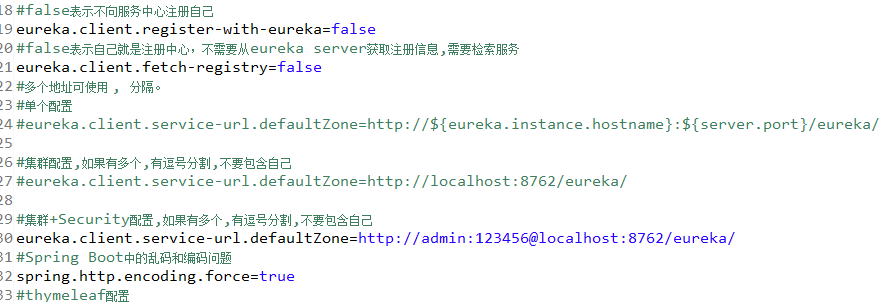
spring.security.user.password=123456



如果是集群，则要修改eureka.client.service-url.defaultZone

#集群+Security配置,如果有多个,有逗号分割,不要包含自己

eureka.client.service-url.defaultZone=http://admin:123456@localhost:8762/eureka/



**3）Eureka Server关闭csrf检验**

Security默认启用了csrf检验，  
CSRF一般指跨站请求伪造攻击，  
要在Eureka Server端配置关闭csrf检验，  
否则Eureka Client无法访问注册中心,  
新建类WebSecurityConfig.java如下：

package com.hpi.system.config;

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

public class WebSecurityConfig extends WebSecurityConfigurerAdapter

{

@Override

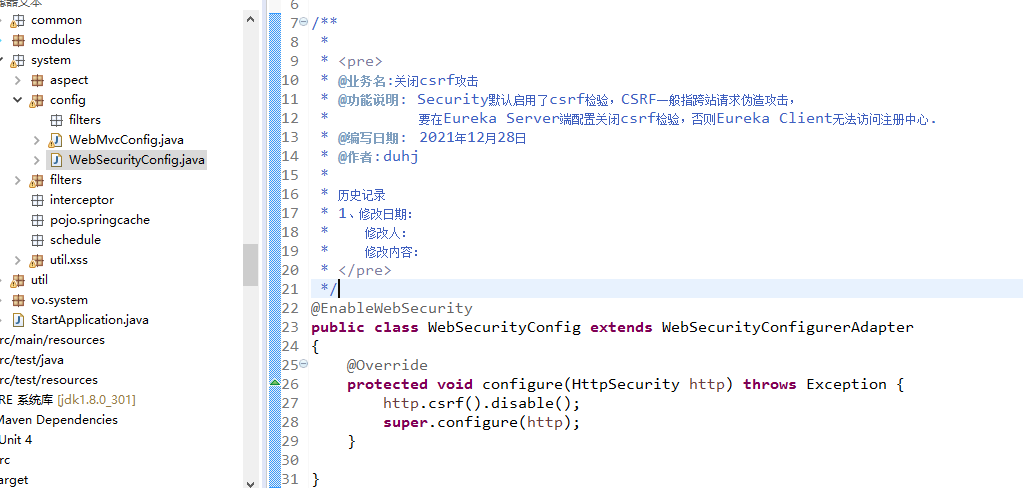
protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable();

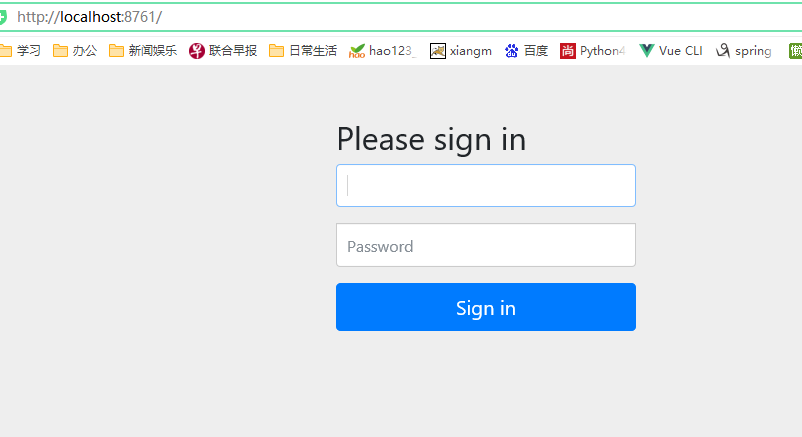
super.configure(http);

}

}



**4）Eureka Server，测试验证**



输入配置的用户名和密码，登录如下

admin/123456

## 提供者provider

对于提供者只需修改yml或者properties的eureka.client.service-url.defaultZone

修改如下：

eureka.client.service-url.defaultZone=http://admin:123456@localhost:8761/eureka,http://admin:123456@localhost:8762/eureka



## 调用者invoker

对于盗用这者也是只需修改yml或者properties的eureka.client.service-url.defaultZone

修改如下：

#集群+Security

eureka.client.service-url.defaultZone=http://admin:123456@localhost:8761/eureka,http://admin:123456@localhost:8762/eureka



## Zuul server

只需修改yml或者properties的eureka.client.service-url.defaultZone

#集群+Security

eureka.client.service-url.defaultZone=http://admin:123456@localhost:8761/eureka,http://admin:123456@localhost:8762/eureka



# 附件

## 访问url

进入注册中心

<http://localhost:8761/>

调用者调用举例：必须启动注册中心，提供者和调用者

<http://localhost:8000/user/getUser>

<http://localhost:8000/user/findUser?name=du1&&password=123>

新增用户 POST

<http://localhost:8000/user/user>

ZUUL

<http://localhost:8080/servcie-provider/user/getUser>

<http://localhost:8080/servcie-provider/user/findUser?name=du1&&password=123>

## Postman 使用

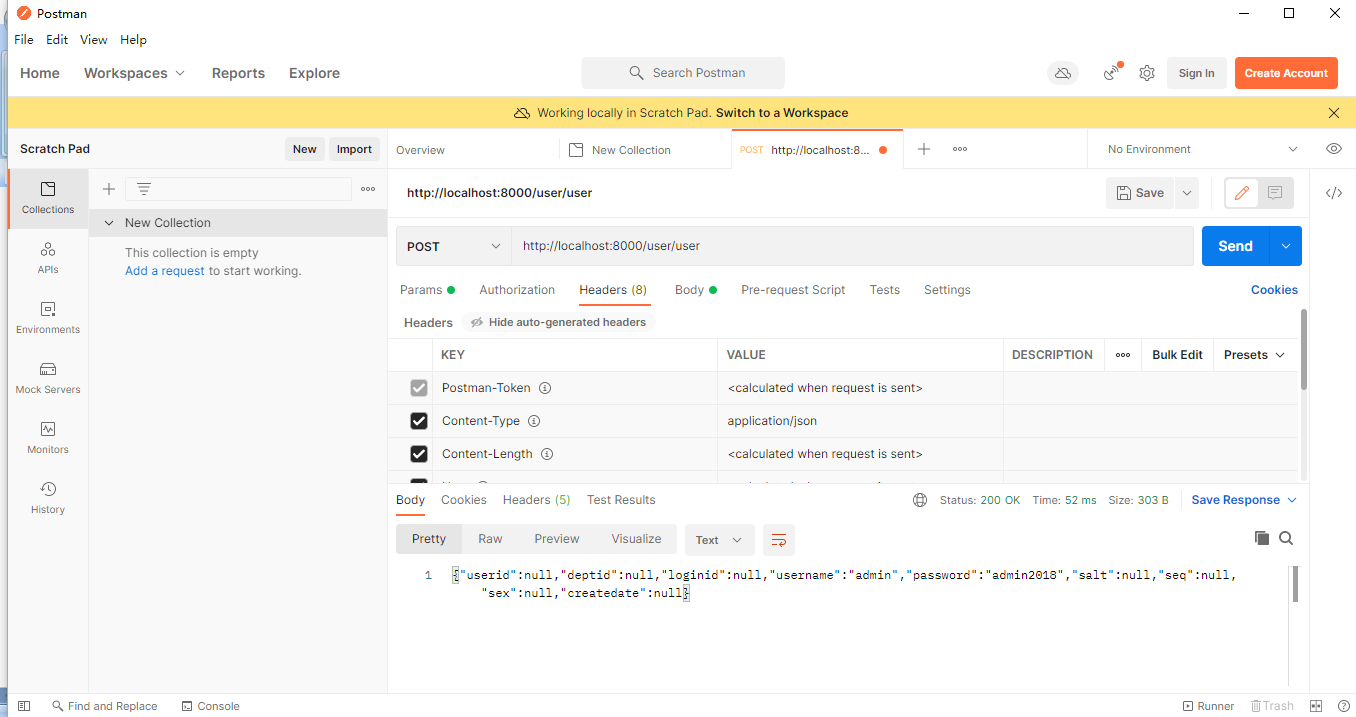
### 7.2.1下载

* Postman是google开发的一款功能强大的网页调试与发送网页HTTP请求，并能运行Chrome插件。其主要功能包括模拟各种HTTP requests

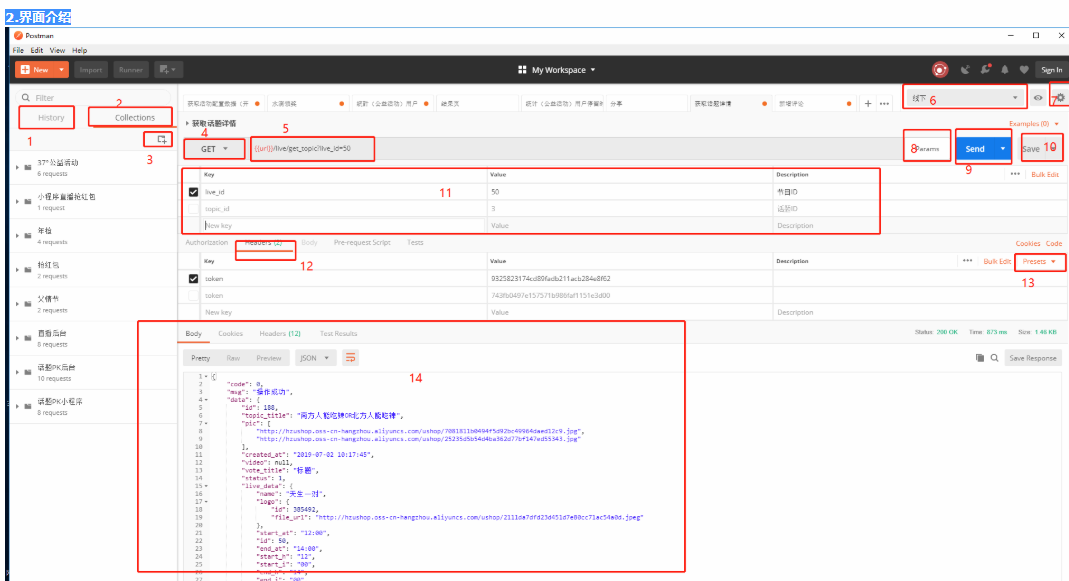
下载地址：<https://www.getpostman.com/>



安装完成后界面如下



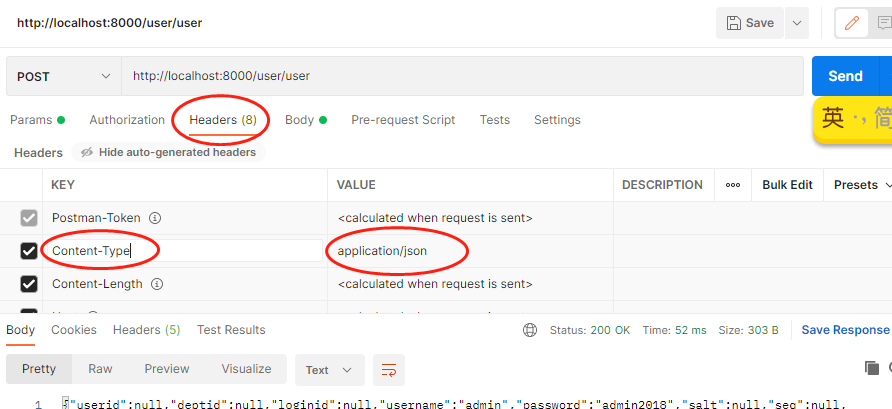
### 7.2.2界面介绍



### 7.2.2发送JSON 格式的POST请求

1. 选择“POST”方式，

在“headers”添加key:Content-Type  , value:application/json



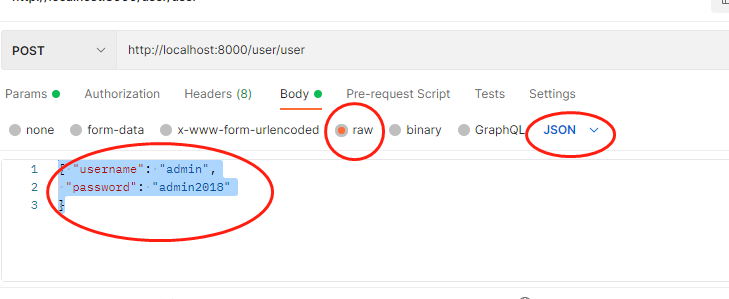
1. 点击"body",''raw''并设定为JSON

添加：

{ "username": "admin",

 "password": "admin2018"

}



然后点击发送即可

## 测试建议

**单机版建议启动**

* 注册中心1 个 springcloud-eureka-server1，
* 提供者2个，分别是：springcloud-service-provider1和springcloud-service-provider2 服务名都是servcie-provider
* 调用者（invoker）1个，springcloud-service-invoker1

也可以启动springcloud-service-invoker2

其实他们提供的都是一样的，如果启动两个，其中一个宕机了，另一个可以提供服务，因为@FeignClient(name = "servcie-provider")都指向的是servcie-provider

输入<http://localhost:8000/user/getUser>测试

如果启动了springcloud-service-invoker2，也可以输入<http://localhost:8001/user/getUser>测试

* 启动网关，springcloud-service-zuul

测试：<http://localhost:8080/servcie-provider/user/getUser>

**集群版建议启动**

* 注册中心2 个 springcloud-eureka-server1和springcloud-eureka-server2，一个宕机另外一个提供服务
* 提供者4个，分别是：springcloud-service-provider1和springcloud-service-provider2 服务名都是servcie-provider
* springcloud-service-provider7和springcloud-service-provider8服务名都是servcie-provider1
* 调用者（invoker）2个，springcloud-service-invoker1和springcloud-service-invoker2

springcloud-service-invoker1的@FeignClient(name = "servcie-provider")指向的是servcie-provider ，即springcloud-service-provider1和springcloud-service-provider2

springcloud-service-invoker2的@FeignClient(name = "servcie-provider1")指向的是servcie-provider1,即springcloud-service-provider7和springcloud-service-provider8

* 启动网关，springcloud-service-zuul

分别访问两个提供者

<http://localhost:8080/servcie-provider/user/getUser>

http://localhost:8080/servcie-provider1/user/getUser