**如何搭建基于SpringCloud的微服务框架**

电子科技大学成都学院 何春

# 环境准备

1. JDK 1.8
2. Maven 3.5.3
3. Eclipse Oxygen.3a

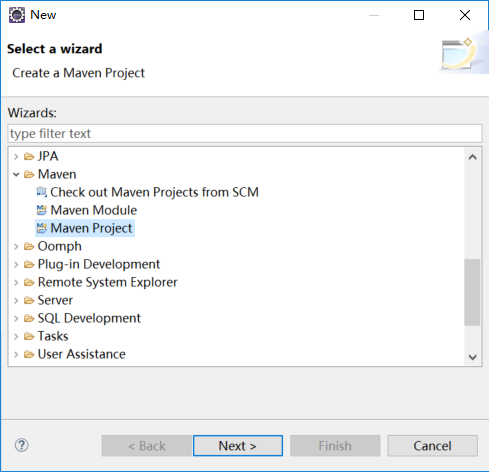
基本的安装和配置过程略。

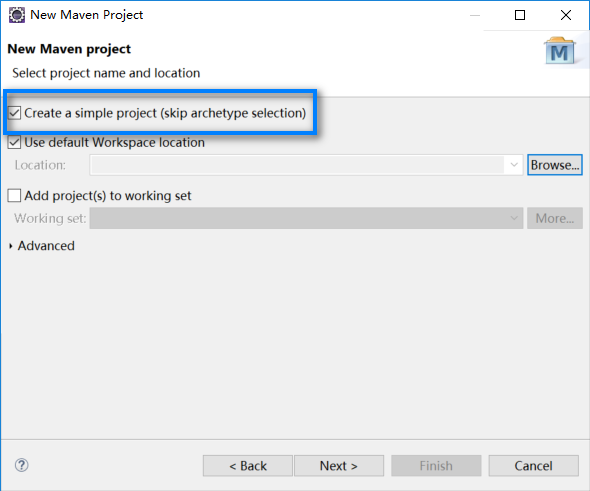
主要的两点：

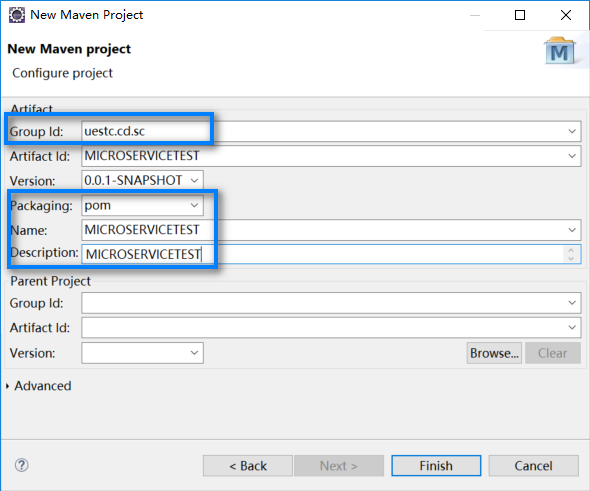
1. 将Maven的源设置为国内比较稳定的源（比如阿里云）。
2. Eclipse使用1中的Maven配置（而不使用Eclipse自带的配置）

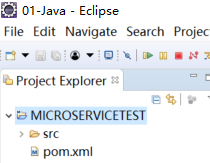
# 创建Maven项目

## 创建空白项目









删掉src文件夹，一个空白的Maven项目就创建成功了。

## 增加对Spring Cloud的支持

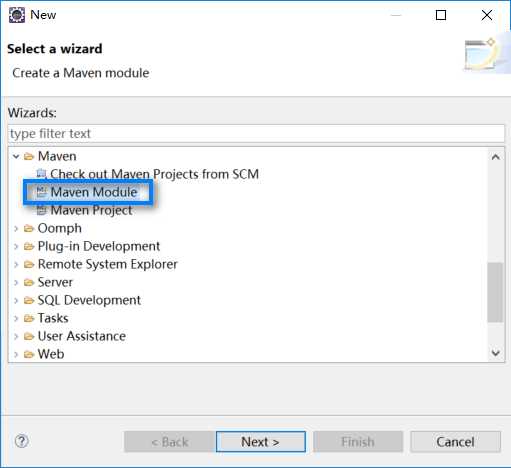
修改项目的POM文件，增加对Spring Cloud的支持：

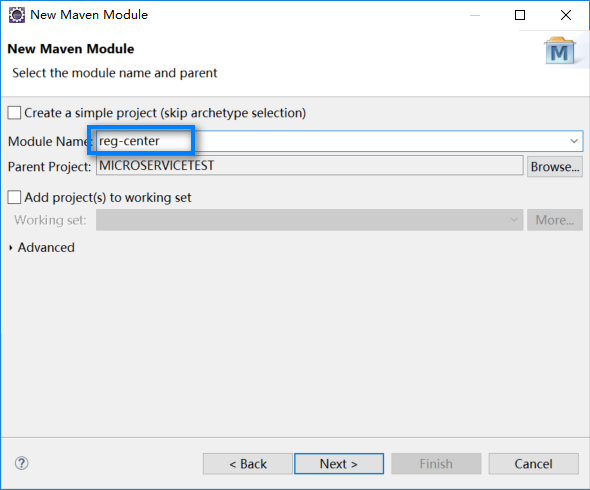
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>uestc.cd.sc</groupId>  <artifactId>MICROSERVICETEST</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <name>MICROSERVICETEST</name>  <description>MICROSERVICETEST</description>  <modules>  <module>reg-center</module>  </modules>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  <spring.boot.version>2.0.2.RELEASE</spring.boot.version>  <spring.cloud.version>Finchley.SR2</spring.cloud.version>  </properties>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <!-- spring boot 默认支持logback日志框架，logging是关联包 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-logging</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>    <!-- 加入该配置后，spring boot类型的子项目，可不用配置org.springframework.boot的<parent>标签 -->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-dependencies</artifactId>  <version>${spring.boot.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring.cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  </project> |

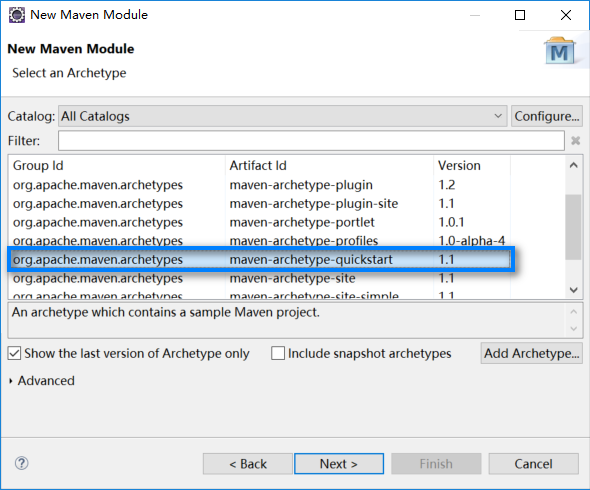
# 创建服务注册中心模块

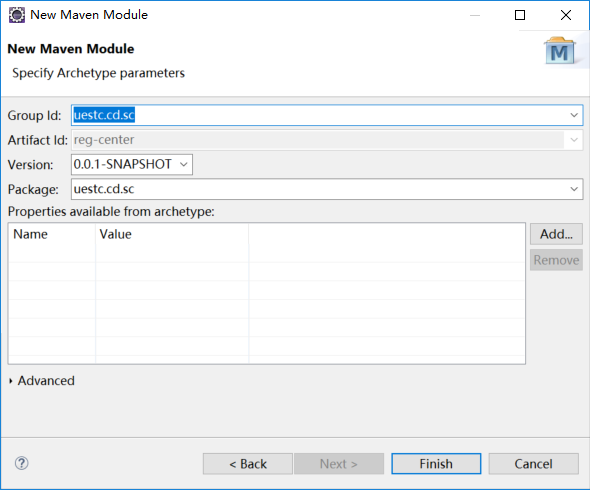
## 创建空白注册中心模块

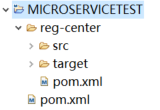
在Eclipse左侧Project Explorer的MICROSERVICETEST项目上点右键，选择Maven Module。











说明：

对于一个应用来讲，它可能包含很多个微服务模块，但是注册中心只需要一个即可（不是每个微服务都需要自己创建一个注册中心，日常开发过程中可以不考虑集群要求）。

这里给出注册中心的创建指导，只是为了方便同学们在自己电脑上学习微服务框架和日常调试。

## 增加对Spring Cloud Eureka的支持

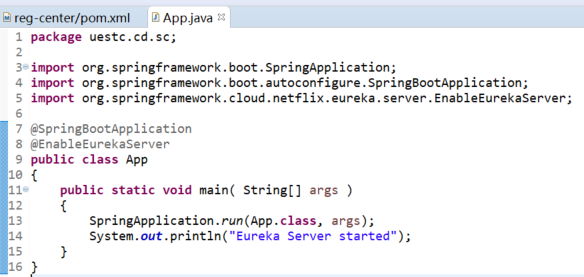
修改项目的POM文件，增加对Spring Cloud Eureka的支持：

|  |
| --- |
| <?xml version=*"1.0"*?>  <project xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"* xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>uestc.cd.sc</groupId>  <artifactId>MICROSERVICETEST</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>    <artifactId>reg-center</artifactId>  <name>reg-center</name>  <url>http://maven.apache.org</url>    <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>    <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  <version>1.4.6.RELEASE</version>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>    <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

在主程序中增加@EnableEurekaServer注解：

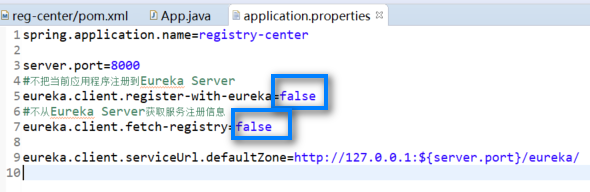
|  |
| --- |
| @SpringBootApplication  @EnableEurekaServer  **public** **class** App  {  **public** **static** **void** main( String[] args )  {  SpringApplication.run(App.class, args);  System.out.println("Registry center started");  }  } |

实际效果如下图所示：

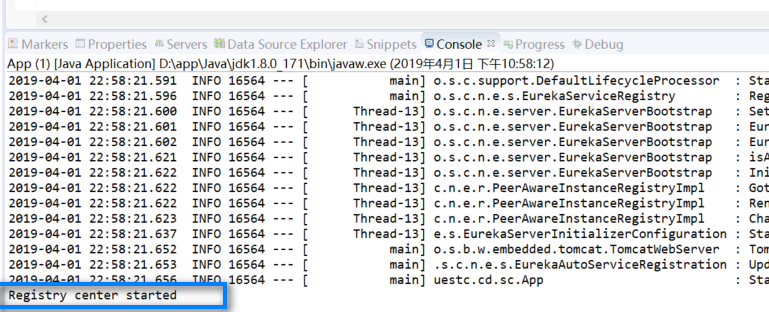


在src/main目录下新增一个resources文件夹，并在其中新建一个application.properties的配置文件，内容如下：

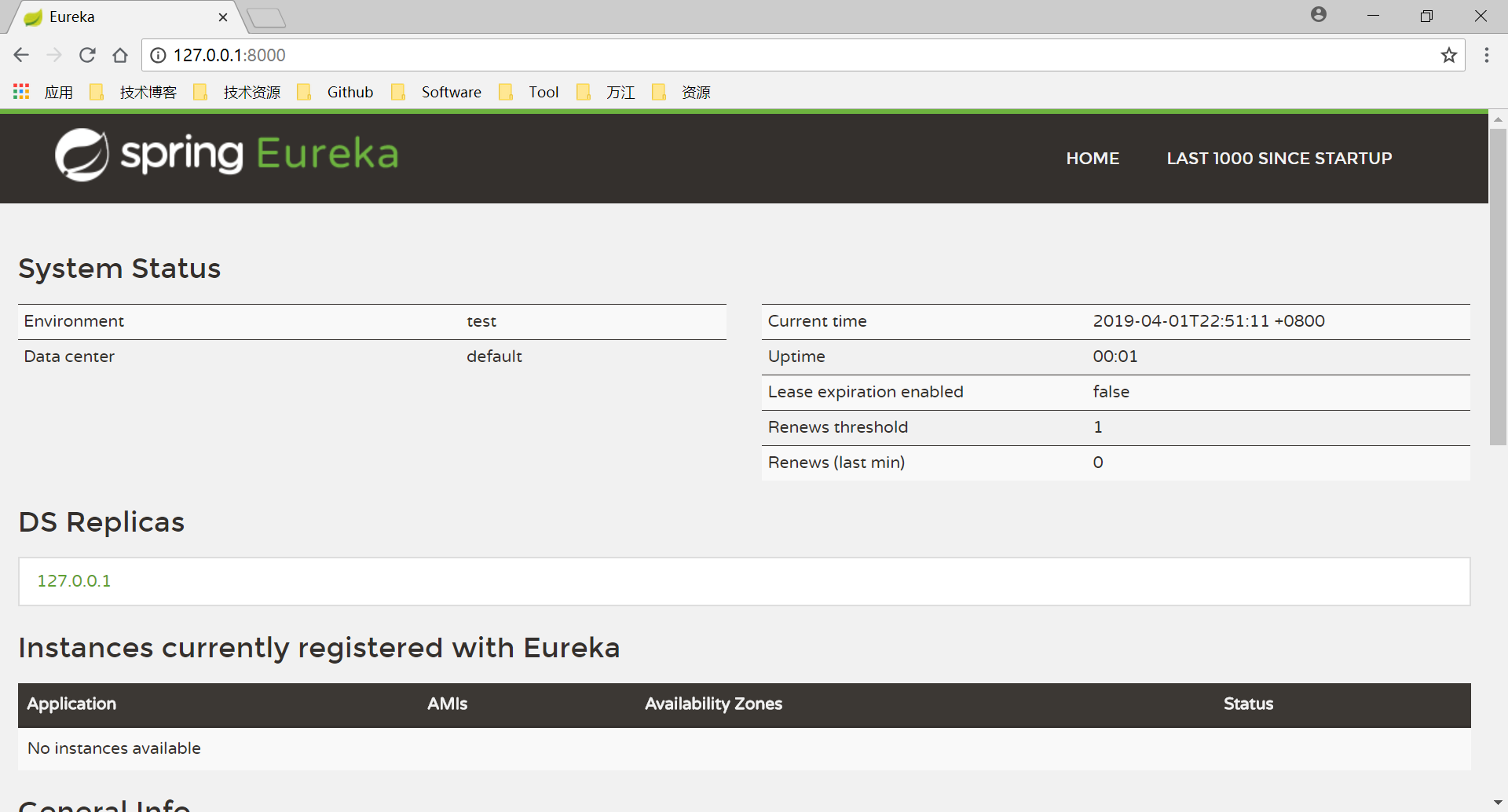
|  |
| --- |
| spring.application.name=registry-center  server.port=8000  #不把当前应用程序注册到Eureka Server  eureka.client.register-with-eureka=false  #不从Eureka Server获取服务注册信息  eureka.client.fetch-registry=false  eureka.client.serviceUrl.defaultZone=http://127.0.0.1:${server.port}/eureka/ |



构建此项目并运行主程序。



在浏览器中输入<http://127.0.0.1:8000>，即可看到注册中心的运行状态。



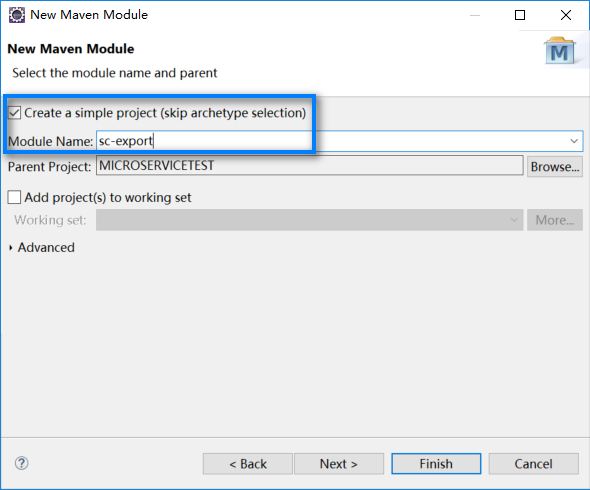
因为当前尚未有服务注册到注册中心，所以Instances区域为空。

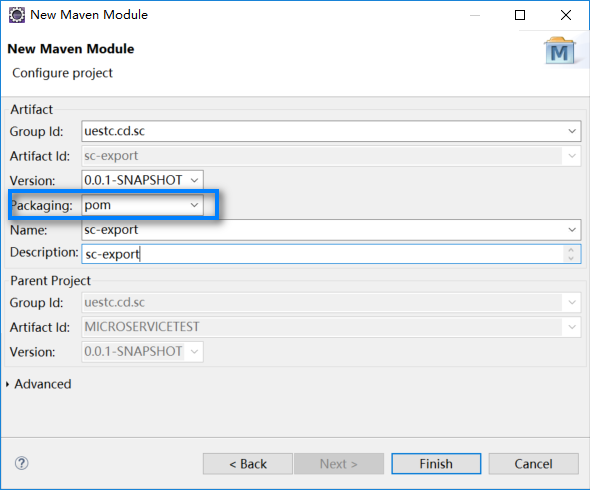
# 创建生产者模块

生产者主要提供业务逻辑处理。

## 创建空白生产者模块

在MICROSERVICETEST项目上点右键，选择Maven Module，创建sc-export模块。

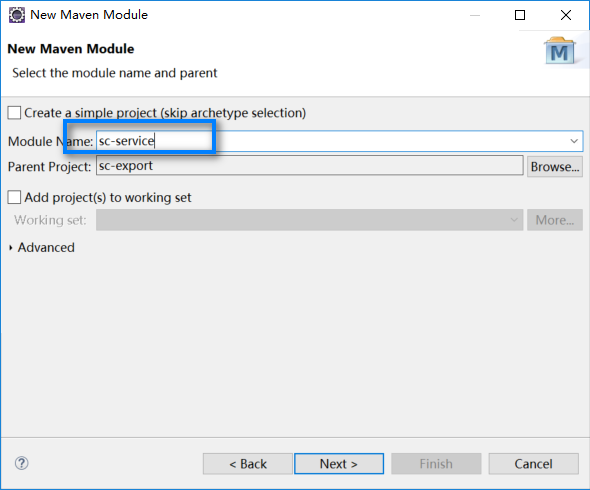


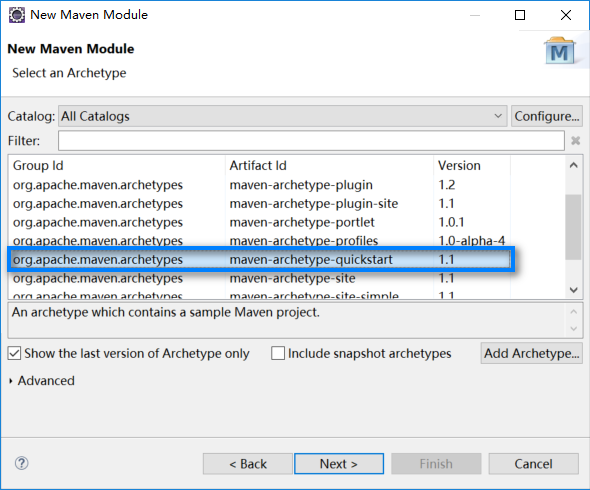


删除src文件夹。

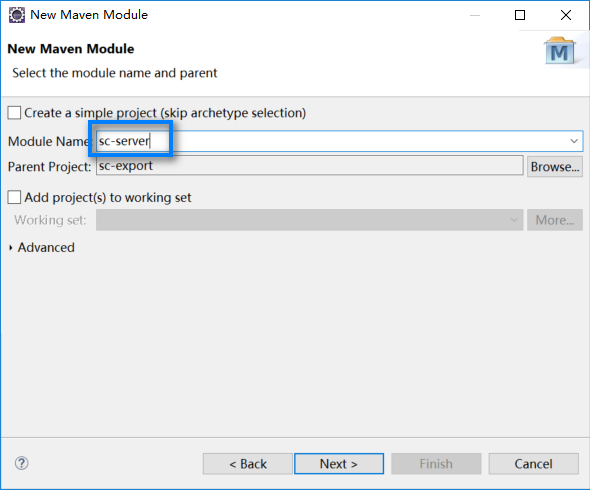


在sc-export模块上点右键，选择Maven Module，创建sc-service模块（这一步需要保留src文件夹），**后续的实体类和业务逻辑接口都放在这个模块中**。

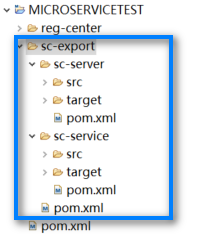




用同样的方法，在sc-export上创建sc-server模块（同样需要保留src文件夹）。



最后的项目结构如下：

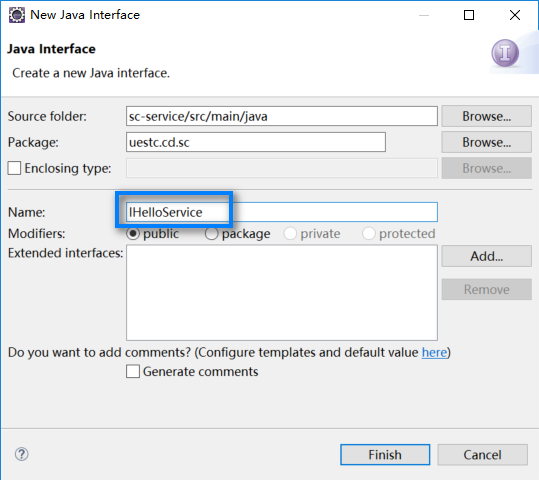


## 增加生产者业务逻辑

在sc-service中删除App.java。

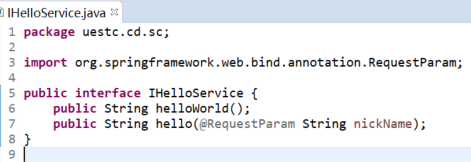


新增接口IHelloService。



内容如下：

|  |
| --- |
| **public** **interface** IHelloService {  **public** String helloWorld();  **public** String hello(@RequestParam String nickName);  } |



修改sc-server模块的POM文件，增加对sc-service和相关类库的依赖：

|  |
| --- |
| <?xml version=*"1.0"*?>  <project xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"* xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>uestc.cd.sc</groupId>  <artifactId>sc-export</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>sc-server</artifactId>  <name>sc-server</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>    <dependencies>  <dependency>  <groupId>uestc.cd.sc</groupId>  <artifactId>sc-service</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>    <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  <version>1.4.6.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

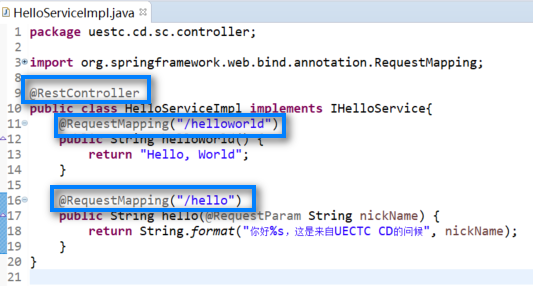
**注意：**

**在sc-server的POM文件中，增加了对sc-service的依赖。**

在sc-server中新建一个类，实现IHelloService中的接口：

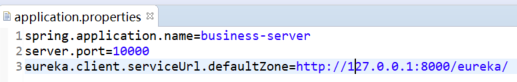
|  |
| --- |
| @RestController  **public** **class** HelloServiceImpl **implements** IHelloService{  @RequestMapping("/helloworld")  **public** String helloWorld() {  **return** "Hello, World";  }    @RequestMapping("/hello")  **public** String hello(@RequestParam String nickName) {  **return** String.*format*("你好%s，这是来自UECTC CD的问候", nickName);  }  } |

注意增加相关的注解：



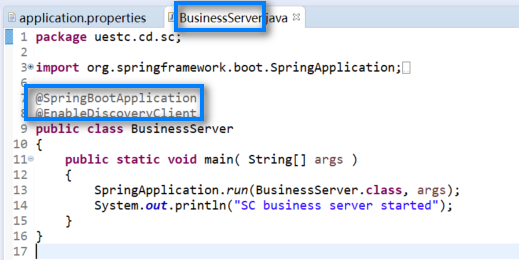
在src/main目录下新增一个resources文件夹，并在其中新建一个application.properties的配置文件，内容如下：

|  |
| --- |
| spring.application.name=business-server  server.port=10001  eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8000/eureka/ |



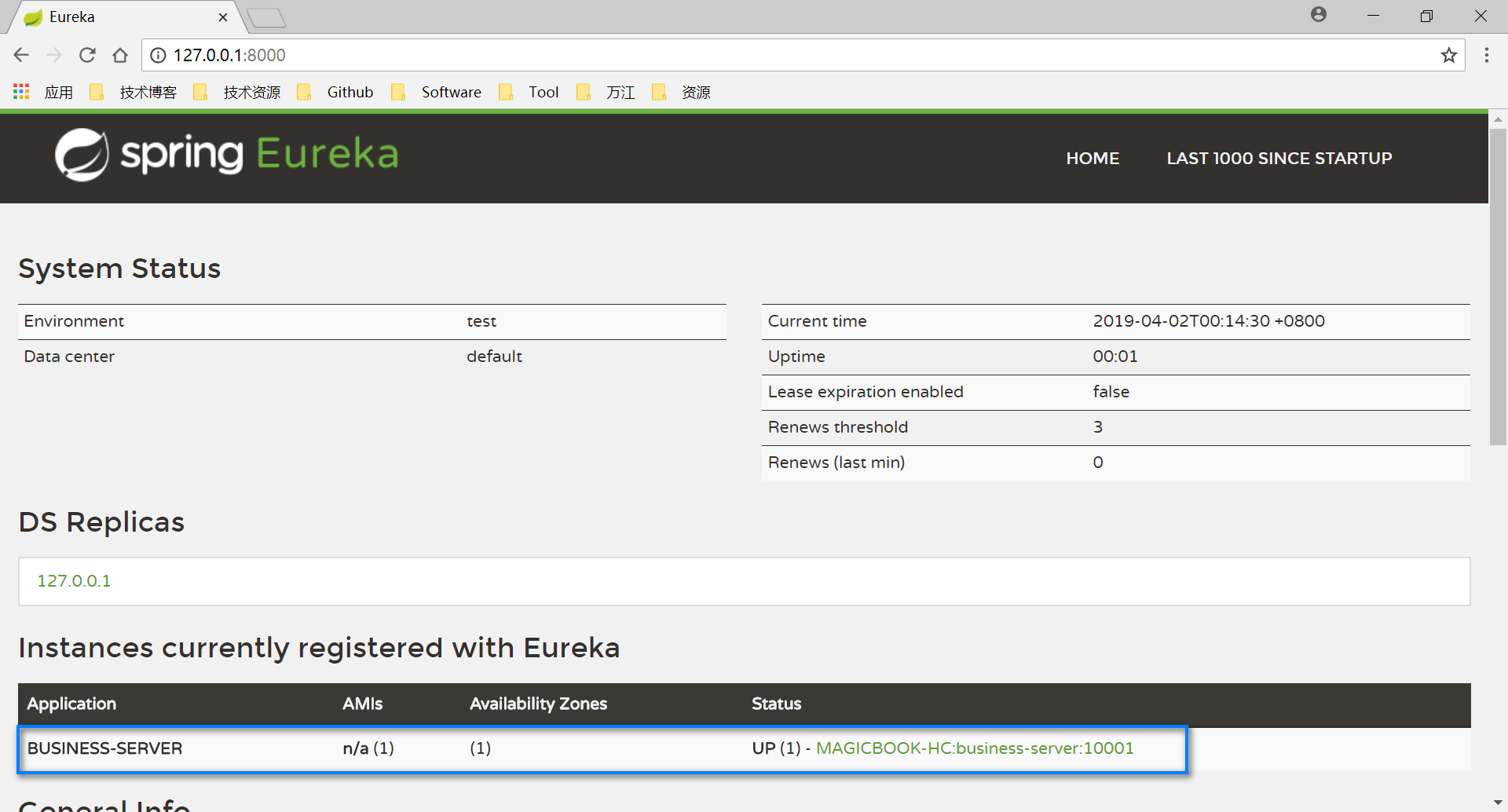
将sc-server默认的主程序重命名并增加相关的注解：

|  |
| --- |
| package uestc.cd.sc;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  @SpringBootApplication  @EnableDiscoveryClient  public class BusinessServer  {  public static void main( String[] args )  {  SpringApplication.run(BusinessServer.class, args);  System.out.println("SC business server started");  }  } |



运行主程序后，生产者即可将其信息自动注册到注册中心。

此时刷新注册中心页面<http://127.0.0.1:8000>，就可看到Instances列表中新的服务BUSINESS-SERVER（这个名字就是sc-server的application.properties中的spring.application.name）。

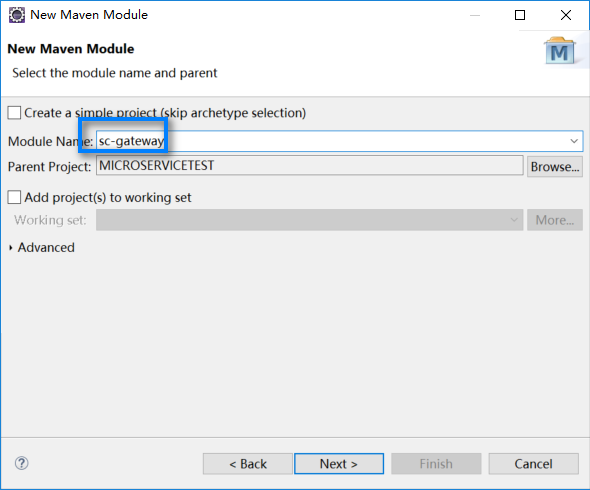


# 创建消费者模块

## 创建空白消费者模块

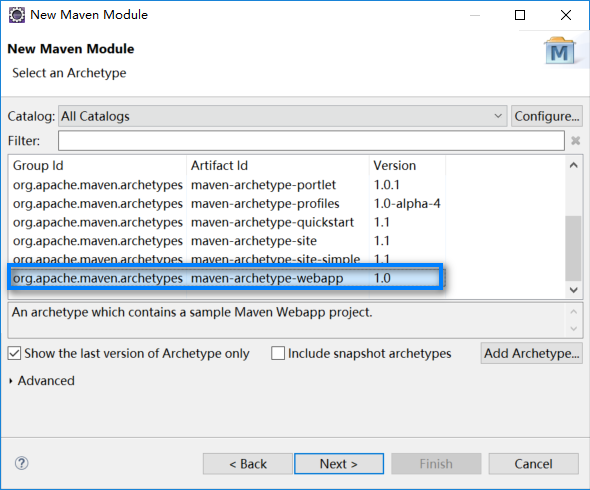
消费者模块的创建流程总体上与sc-server一致，但由于消费者模块需要暴露REST接口，所以在选择类型时，我们会选择webapp。

在MICROSERVICETEST项目上点右键，选择Maven Module，创建sc-gateway模块。



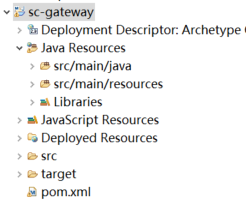
Spring Boot的默认打包格式是jar，jar包会封装一个tomcat的核心，从而可以通过java -jar的形式提供web服务。但是很多时候我们也会需要将消费者模块打包成war包，然后手工部署在tomcat容器里。

下面的打包格式就是war：



如果是需要打包成jar，则Archetype可以参考生产者模块的创建方式，选择maven-archetype-quickstart。

在src\main下新建java文件夹，用于后续存放相关的rest接口及其实现。

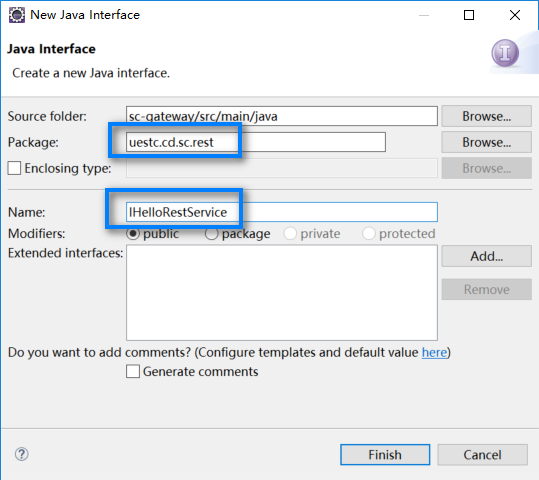


## 增加消费者REST接口

修改sc-gateway模块的POM文件，增加对相关类库的依赖：

|  |
| --- |
| <?xml version=*"1.0"*?>  <project xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"* xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>uestc.cd.sc</groupId>  <artifactId>MICROSERVICETEST</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <artifactId>sc-gateway</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>sc-gateway Maven Webapp</name>  <url>http://maven.apache.org</url>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-feign</artifactId>  <version>1.4.6.RELEASE</version>  </dependency>    <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  <version>1.4.6.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>    <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <finalName>sc-gateway</finalName>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

创建一个IHelloRestService的REST接口类。

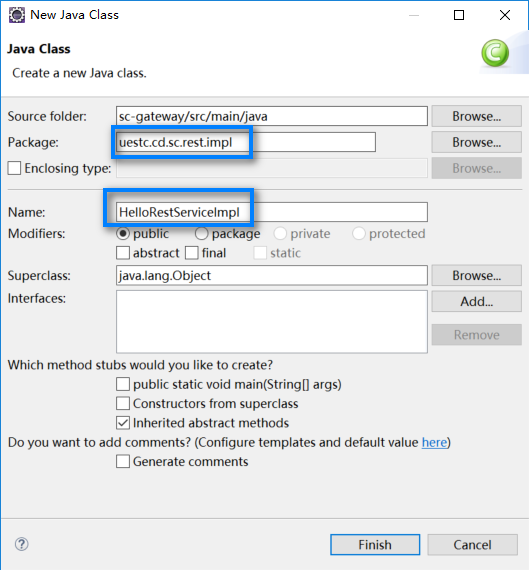


接口内容如下：

|  |
| --- |
| **public** **interface** IHelloRestService {  **public** String helloworld();    **public** String hello(@PathVariable("nickName") String nickName);  } |

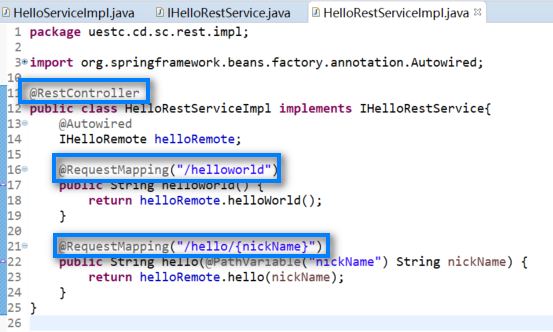


然后创建IHelloRestService的实现类：



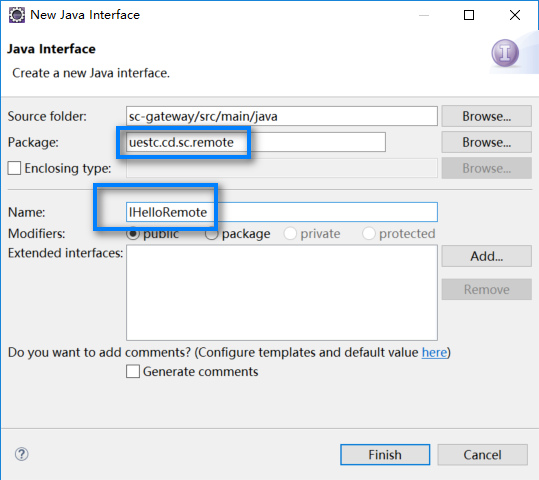
实现内容如下：

|  |
| --- |
| **public** **class** HelloRestServiceImpl **implements** IHelloRestService{  @Autowired  IHelloRemote helloRemote;    @RequestMapping("/helloworld")  **public** String helloworld() {  **return** helloRemote.helloWorld();  }    @RequestMapping("/hello/{nickName}")  **public** String hello(@PathVariable("nickName") String nickName) {  **return** helloRemote.hello(nickName);  }  } |



在HelloRestServiceImpl中，向Controller中注入了生产者的远程调用接口IHelloRemote。

所以，还需要新建IHelloRemote接口：



生产者远程调用接口的内容如下：

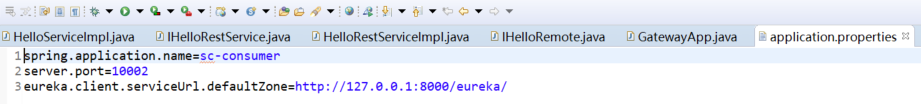
|  |
| --- |
| @FeignClient(name= "business-server")  **public** **interface** IHelloRemote {  @RequestMapping("/helloworld")  **public** String helloWorld();    @RequestMapping("/hello")  **public** String hello(@RequestParam(value="nickName", required = **true**) String nickName);  } |



此处添加了FeignClient的注解，接口中声明的方法和生产者HelloServiceImpl中的方法名、参数都是保持一致的。

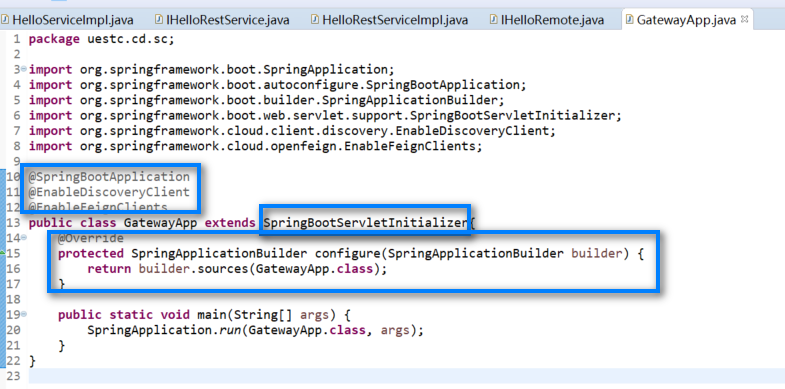
在src/main/resources文件夹中，新建一个application.properties的配置文件，内容如下：

|  |
| --- |
| spring.application.name=sc-consumer  server.port=10002  eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8000/eureka/ |



修改消费者模块创建时自动生成的App类，使其支持在tomcat容器中可以正常初始化并运行。

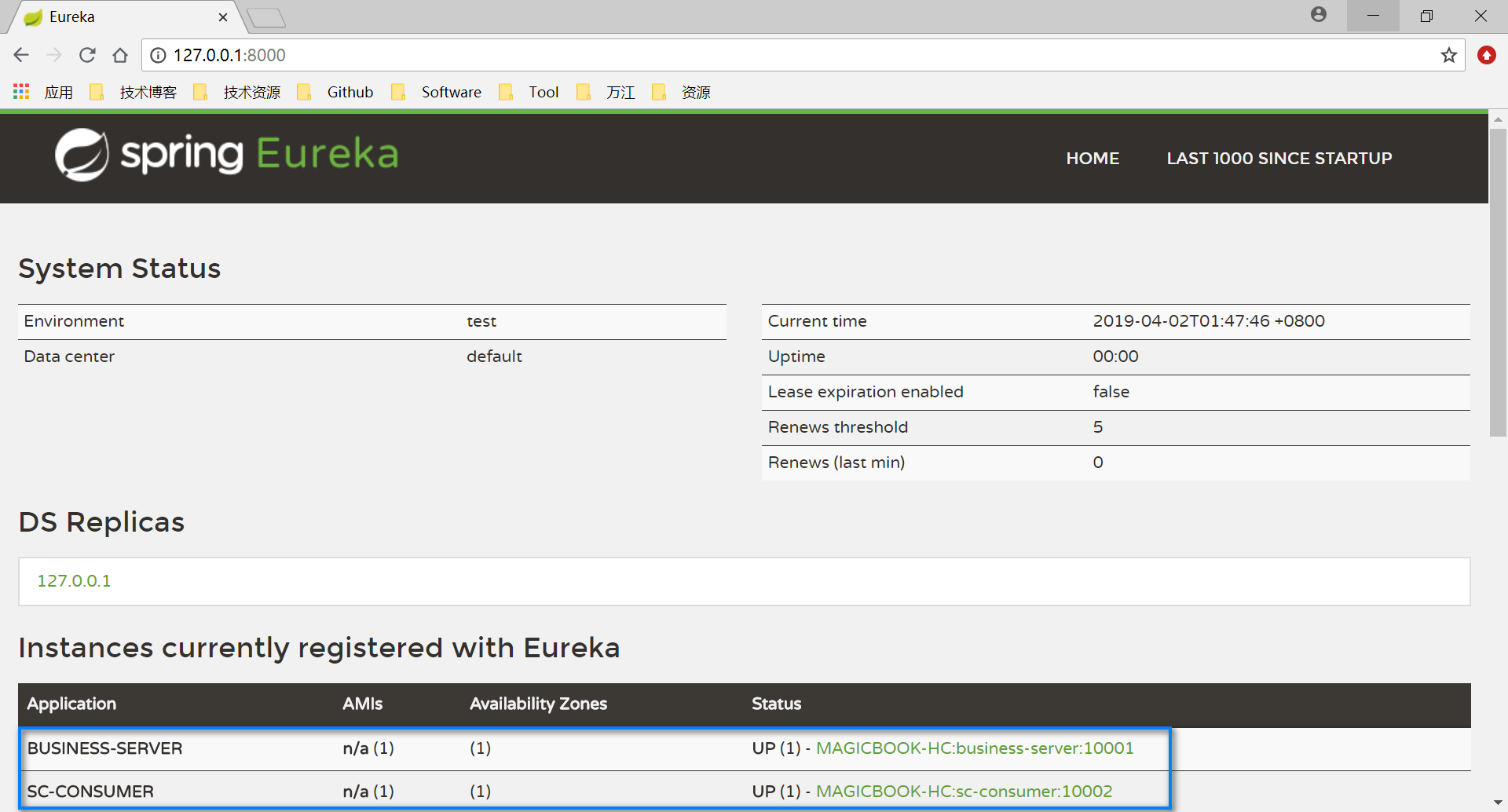
|  |
| --- |
| @SpringBootApplication  @EnableDiscoveryClient  @EnableFeignClients  **public** **class** GatewayApp **extends** SpringBootServletInitializer{  @Override  **protected** SpringApplicationBuilder configure(SpringApplicationBuilder builder) {  **return** builder.sources(GatewayApp.**class**);  }    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(GatewayApp.**class**, args);  }  } |



启动注册中心reg-center。

构建sc-gateway项目，在tomcat中RUN起来。

此时如果访问注册中心页面，就可以看到SC-CONSUMER已经注册成功了。



## 增加消费者WebUI

当我们在创建空白消费者模块sc-gateway时，eclipse已经帮我们生成了web应用必须的基本框架，存放在src/main/webapp下面，这个框架包括了WEB-INF的配置和一个空白的index.jsp页面。接下来我们就用一个最简单的页面来代表我们的WebUI。

对index.jsp稍作改造，使其可以通过异步方式调用消费者提供的REST接口：

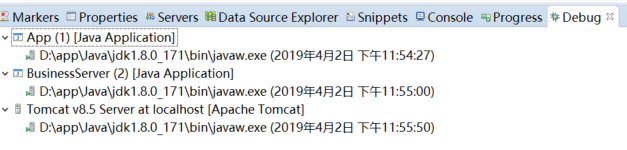
|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=utf-8"*  pageEncoding=*"utf-8"*%>  <!DOCTYPE html>  <html lang=*"zh-CN"*>  <head>  <meta charset=*"utf-8"*>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <meta http-equiv=*"X-UA-Compatible"* content=*"IE=edge"*>  <meta name=*"viewport"* content=*"width=device-width, initial-scale=1"*>  <script src=*"https://cdn.bootcss.com/jquery/1.12.1/jquery.min.js"*></script>  <script type=*""*>  $(document).ready(function(){  $("#btnHello").click(function(){  var param = {};  $.ajax({  type: "get",  url: "helloworld",  data: param,  async: false,  error: function() {  alert("ERROR");  },  success: function(data) {  alert(data);  }  });  });  $("#btnSubmit").click(function(){  var param = {};  $("#result").html("");  var nickName = $("#txtNickName").val();  $.ajax({  type: "get",  url: "hello/" + nickName,  data: param,  async: false,  error: function() {  alert("ERROR");  },  success: function(data) {  $("#result").html(data);  }  });  });  });  </script>  </head>  <body>  <h2>Web UI</h2>  <p style="cursor: *pointer*;" id=*"btnHello"*>点击这里，获取hello</p>  <p>输入昵称：</p>  <p><input type=*"text"* id=*"txtNickName"*><span style="cursor: *pointer*;" id=*"btnSubmit"*>提交</span></p>  <div id=*"result"*></div>  </body>  </html> |

这里的功能很简单：

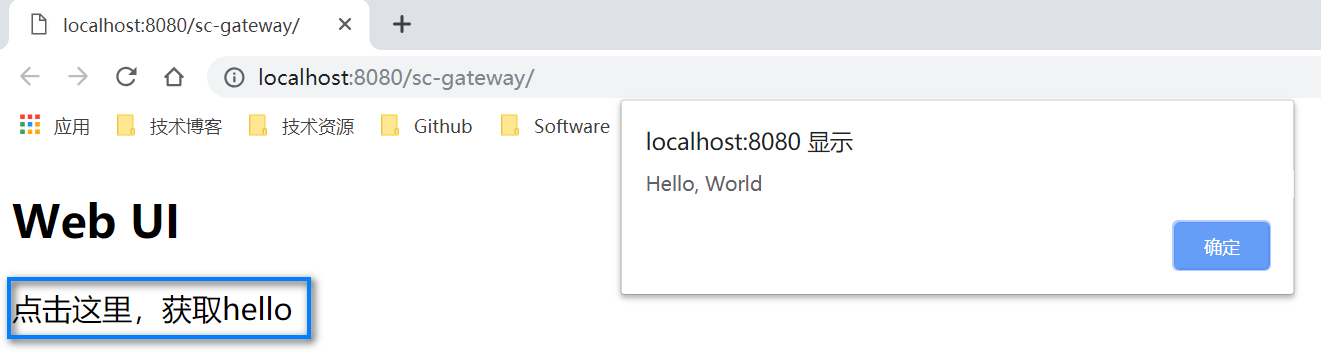
1.点击一段文字，前端框架就异步调用消费者的helloworld接口，从生产者的业务接口中读取相应的数据并回显给用户。

2.填写一个昵称，点击提交按钮后，前端框架就异步调用消费者的hello/xxx接口，从生产者的业务接口中读取相应的数据并回显给用户。

现在把reg-center、sc-server和sc-gateway依次运行起来：



在浏览器上访问<http://localhost:8080/sc-gateway/>，即可体验到微服务的效果：



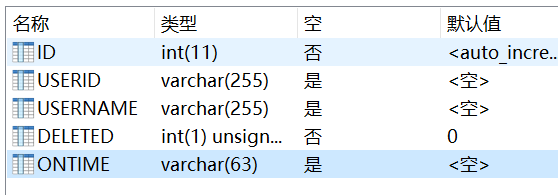


# 进阶1 读取数据库

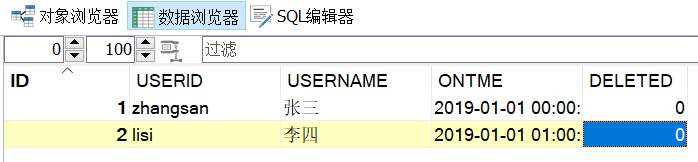
## 环境准备

安装MySQL数据库及相关的管理程序

## 设计数据库

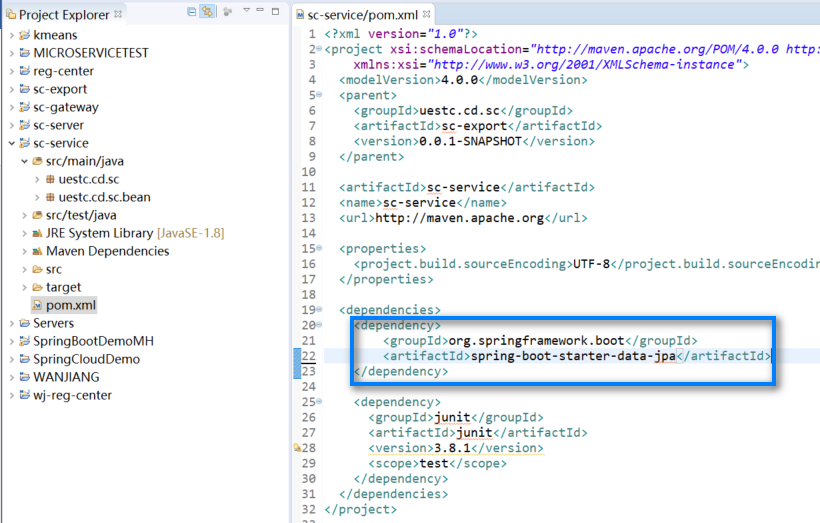


填入两条基本数据



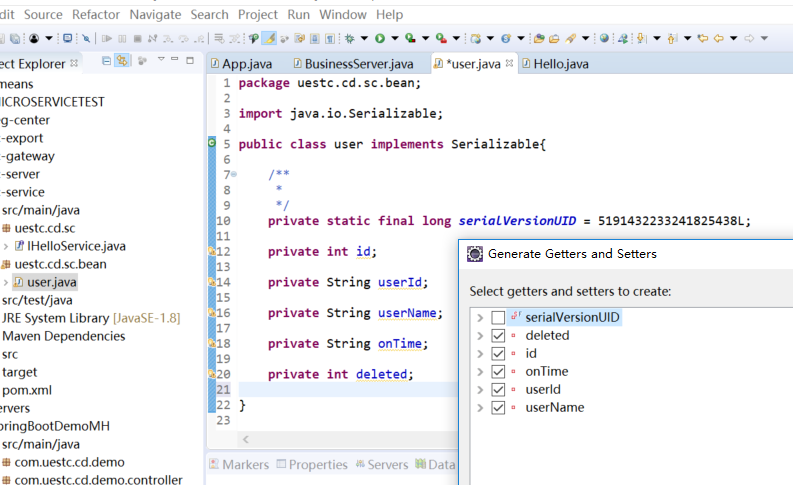
## 创建实体类

在sc-service中完善依赖

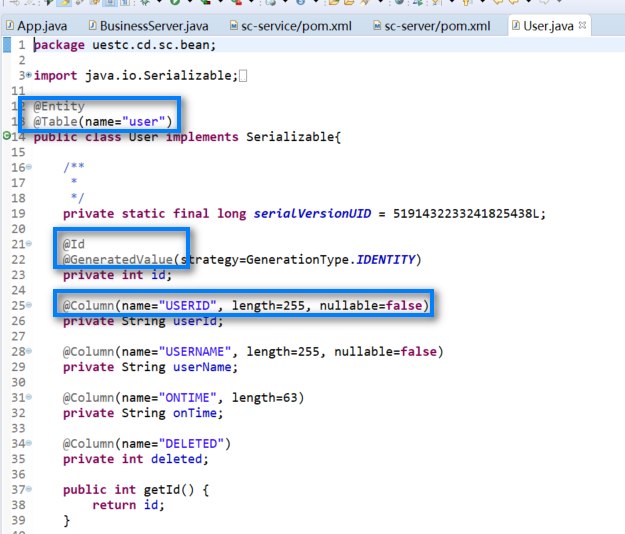


创建实体类user添加相关的注解

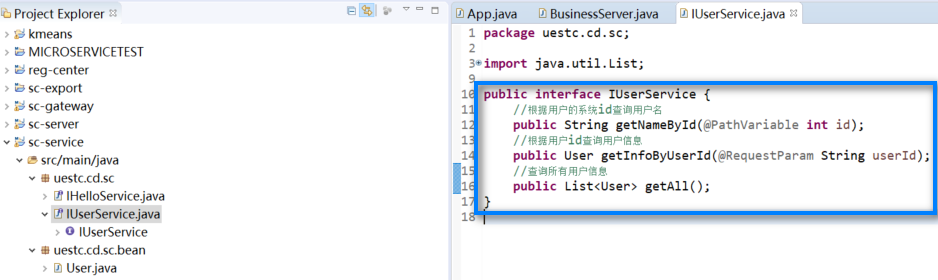




增加相关的注解

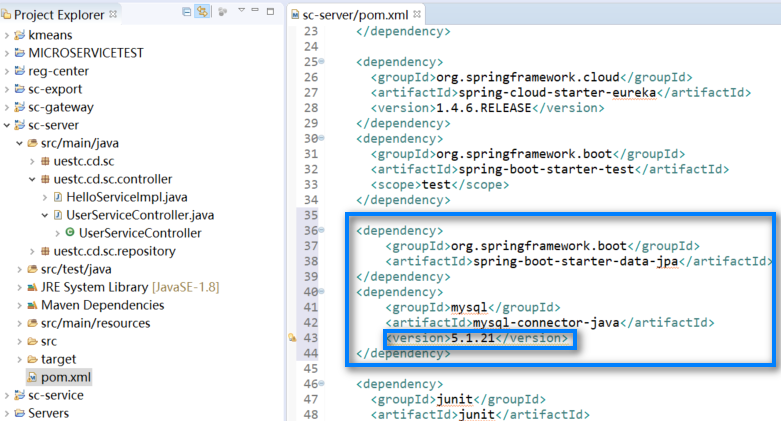


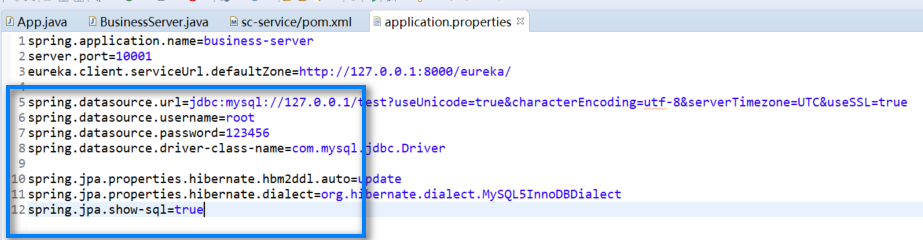
创建IUserService接口，定义相关的方法：



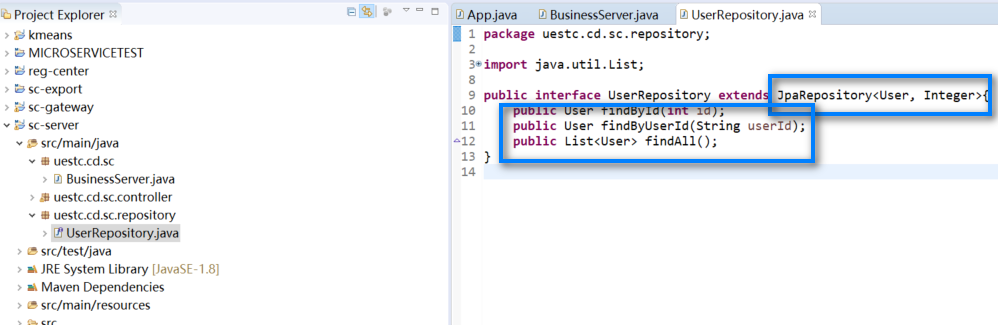
## 增加服务端DAO和业务逻辑实现

完善sc-server的依赖（这里使用jpa）

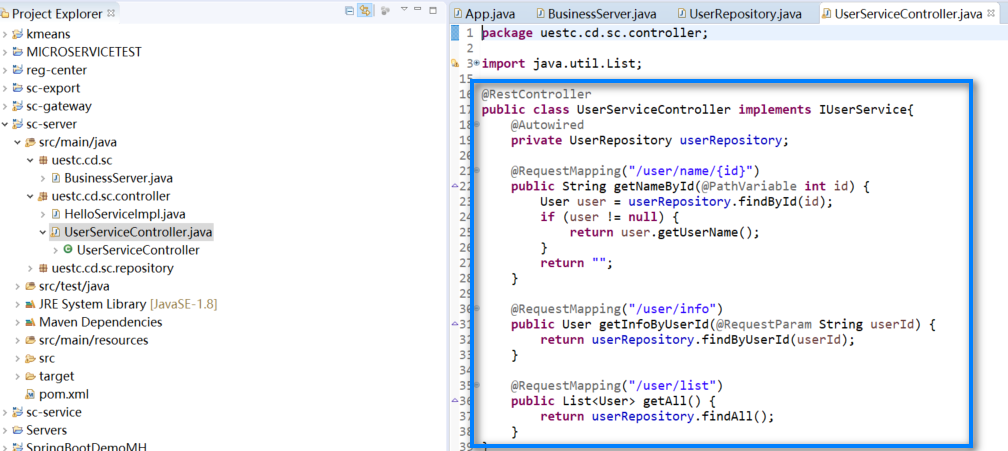
在application.properties中增加MySQL数据库相关的配置



根据sc-service中IUserService中的接口定义创建DAO层接口（继承JpaRepository）：



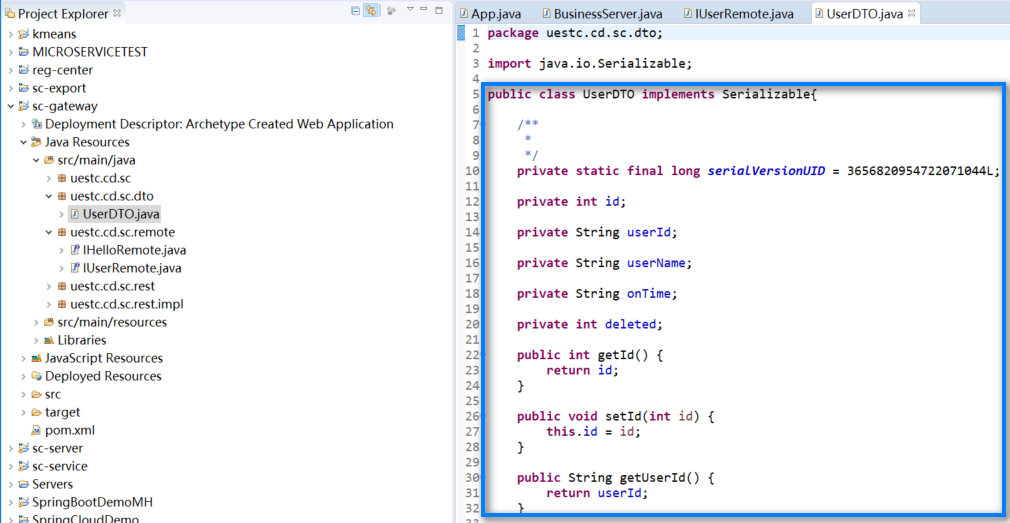
创建UserServiceController，注入UserRepository，实现IUserService的业务逻辑并暴露接口：



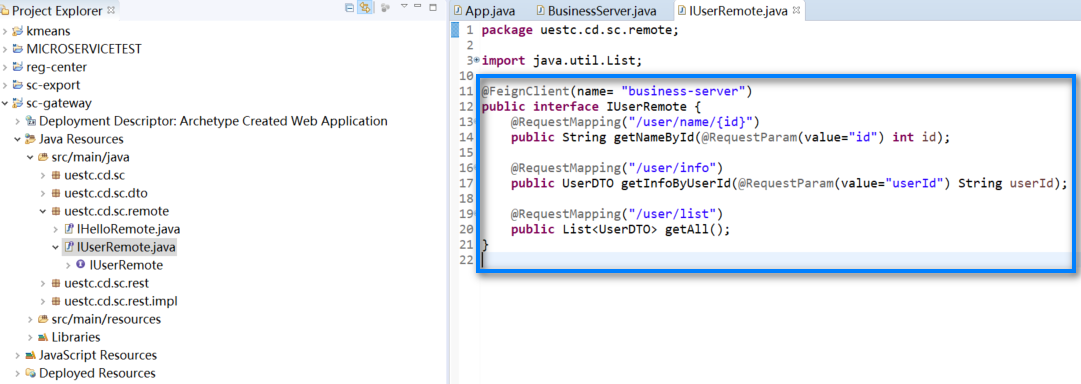
## 增加网关接口和WebUI

增加生产者的远程调用接口IUserRemote：

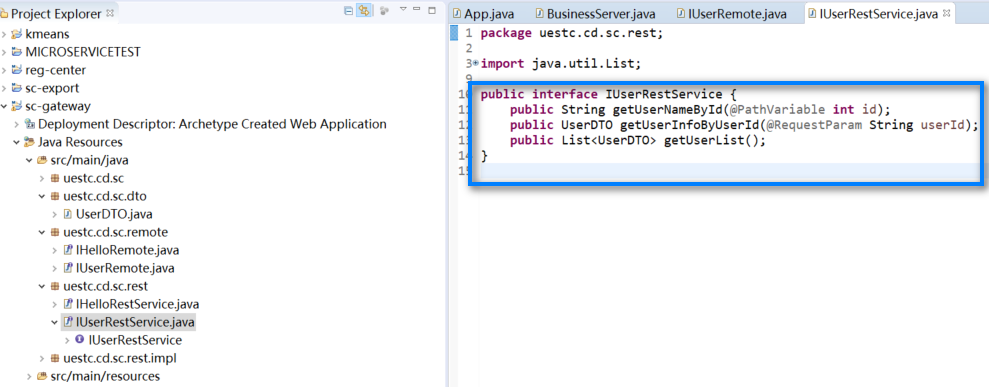
由于IUserRemote只暴露生产者的远程调用接口，且在sc-service中我们已经为实体类User添加了相关的注解，所以这里不能直接引入User类。Name需要我们在创建IUserRemote接口之前，先创建一个和User实体类属性相同的DTO（数据传输对象），名字为UserDTO，其属性方法均与实体类User一致，但是不包含任何注解。



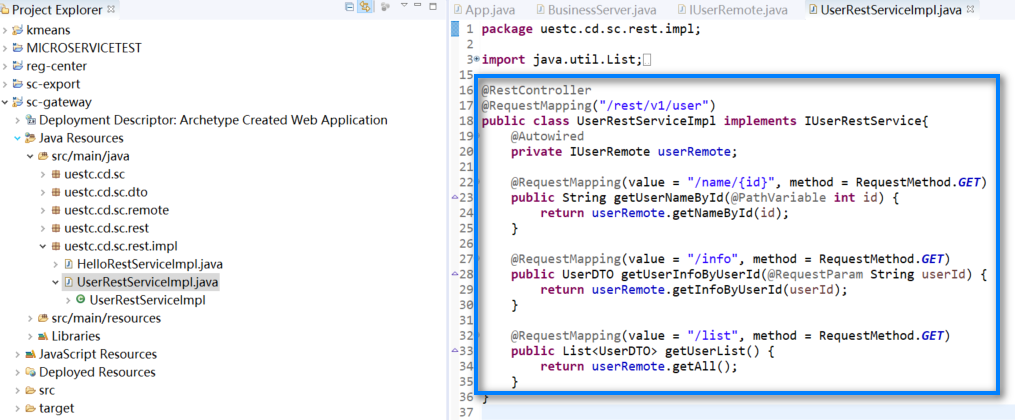
然后创建IUserRemote：



定义网关的rest接口类IUserRestService：



创建UserRestServiceImpl类，实现并暴露rest接口：



验证接口，分别运行reg-center、sc-server和sc-gateway，在浏览器中访问

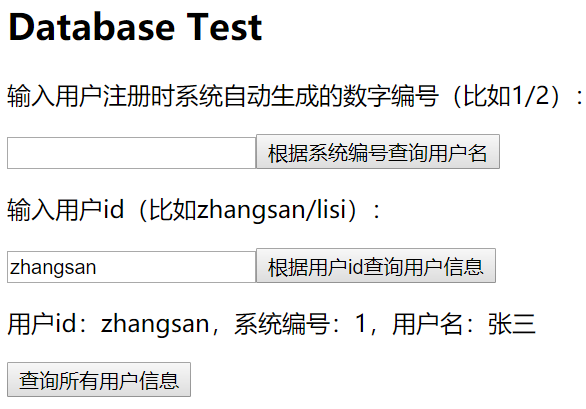
<http://127.0.0.1:8080/sc-gateway/rest/v1/user/name/1>

<http://127.0.0.1:8080/sc-gateway/rest/v1/user/info?userId=lisi>

<http://127.0.0.1:8080/sc-gateway/rest/v1/user/list>

即可查看从服务端返回的数据

写个简单的web页面，验证上面的三个接口：



**回顾“进阶1”部分的程序逻辑，会发现sc-service已经没有太大的存在价值了，请同学们课后将其从项目中删除相关的代码移植到sc-server中，后续我们也将不再创建sc-export模块。**

**另外，请自行增加一个添加和更新用户信息的接口。**

# 参考资料

## 关于微服务框架

<https://www.cnblogs.com/wintersun/p/6219259.html>

<https://www.jianshu.com/p/009d98e30b2a>

<https://www.cnblogs.com/imyalost/p/6792724.html>

<https://blog.csdn.net/u010664947/article/details/80007767>

<https://segmentfault.com/a/1190000005142470>

<http://dubbo.apache.org/zh-cn/docs/user/quick-start.html>

## 关于Spring Boot

<http://www.ityouknow.com/spring-boot.html>

## 关于Spring Cloud

<https://spring.io/projects/spring-cloud>

<https://springcloud.cc/spring-cloud-dalston.html>

<http://www.ityouknow.com/spring-cloud.html>

<https://www.cnblogs.com/chry/p/7248947.html>