基于SpringCloud的企业级微服务架构入门（一）：微服务的注册发现

1、简要概述

Spring Cloud 微服务的核心模块主要包括两类：一个是微服务注册中心，用于管理所有注册在其中的微服务；另一个是微服务功能模块，用于提供实际的服务，注册在注册中心上，后续会逐步提到的网关服务、配置服务等都是不同变种的微服务功能模块。

在我们的案例中，我们使用的组件是Spring Cloud Netflix的Eureka。包括Eureka Server和Eureka Client，分别对应注册中心和功能模块。



2、开发环境：

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

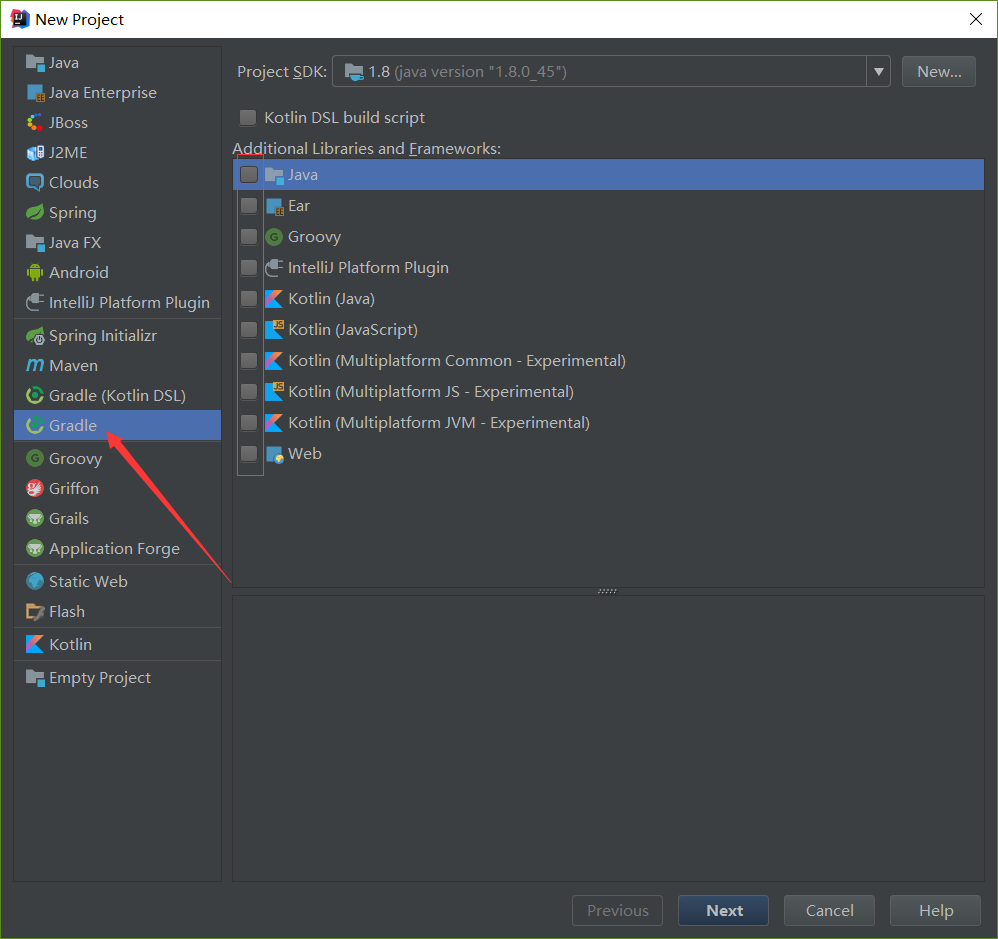
springCloudVersion = 'Edgware.SR2'

Github地址 https://github.com/LittleFlowerPig/lfp-jec.git

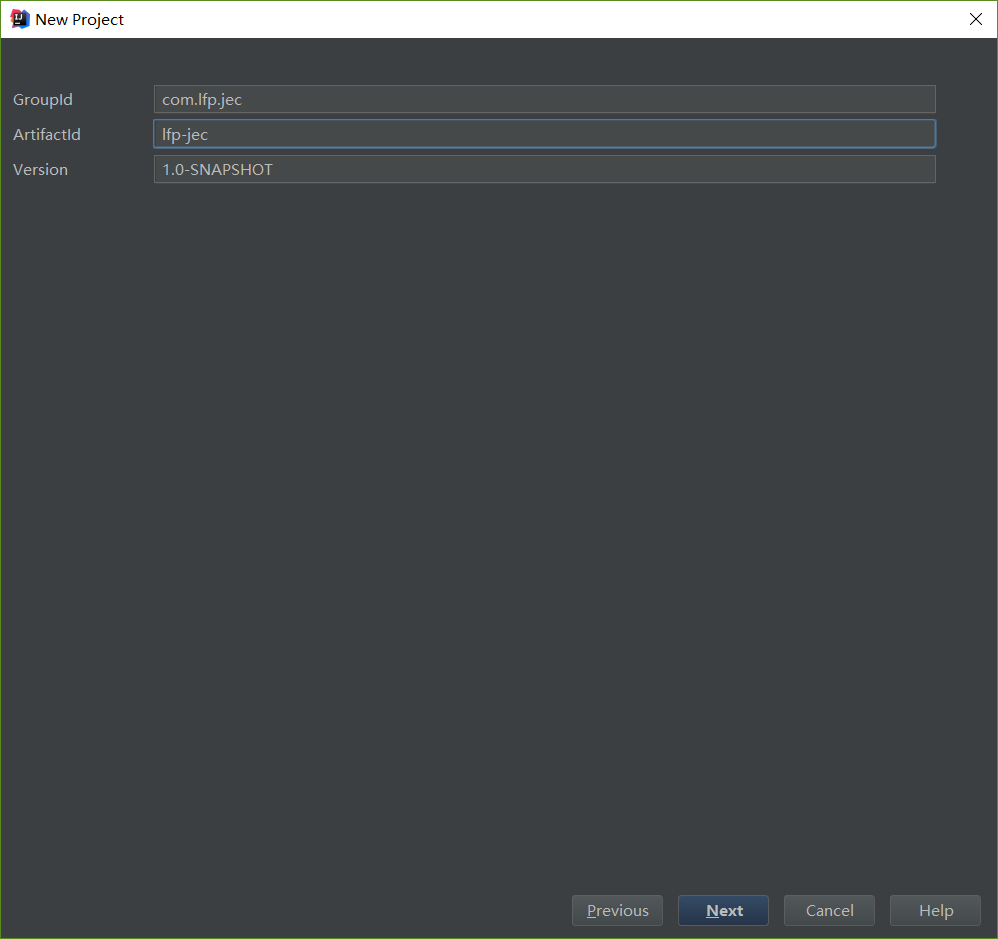
3、实际操作

3.1、创建一个Gradle主工程。

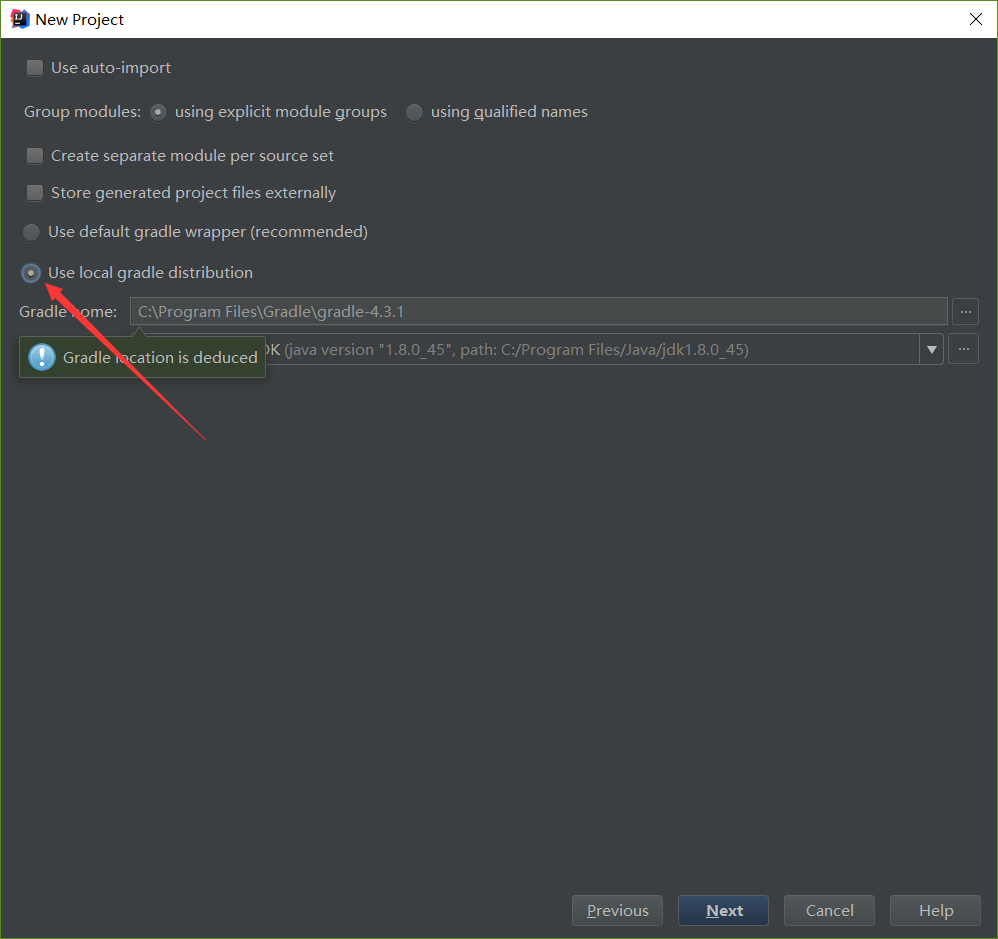
新建新项目，选择Gradle，不选任何框架。



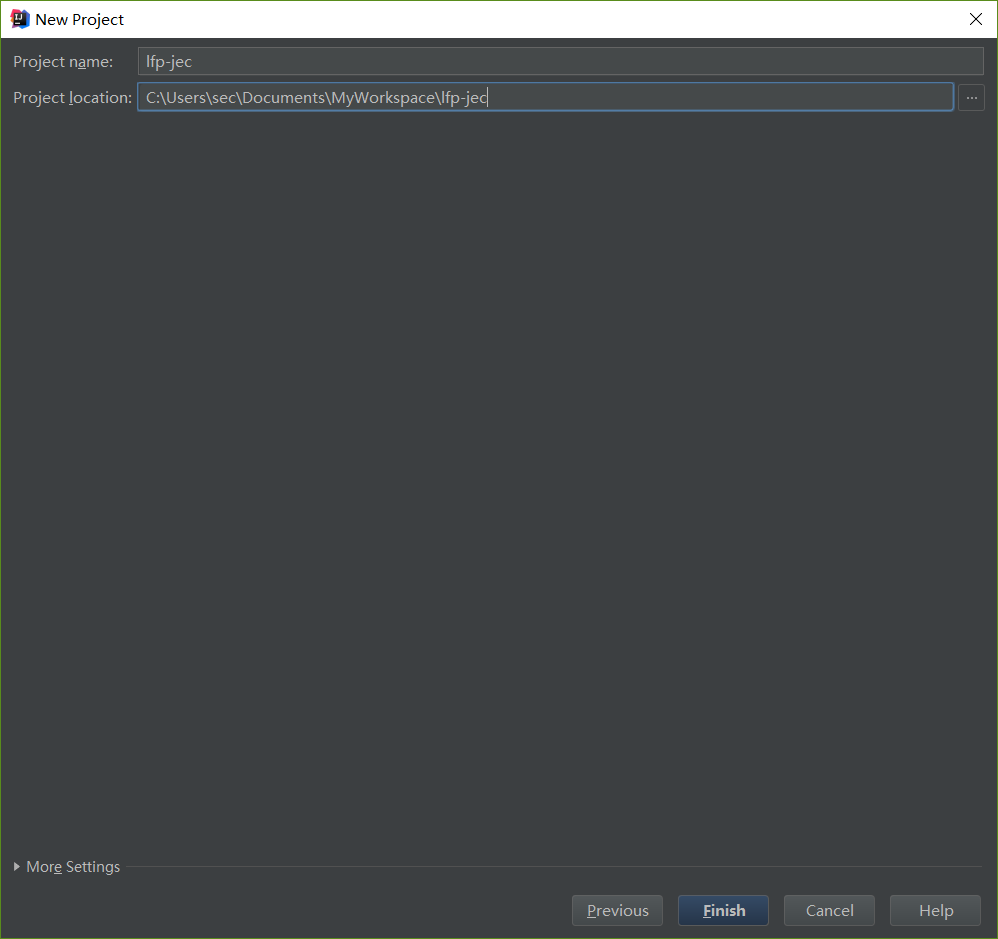
输入GroupId和ArtifactId。



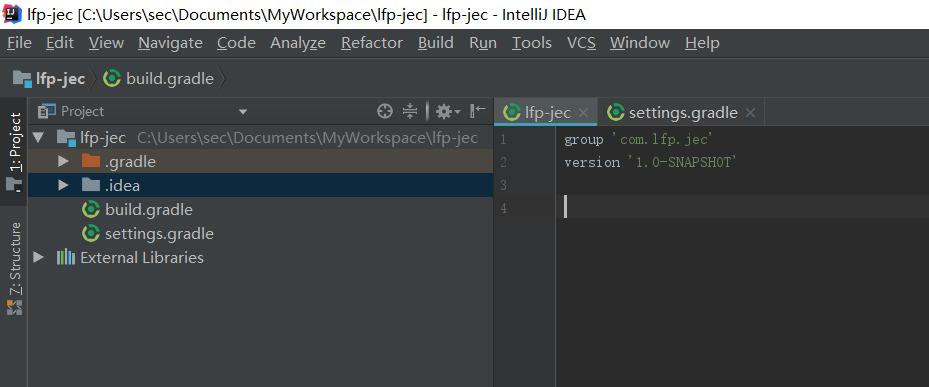
选择本地的gradle。



设置项目的位置。

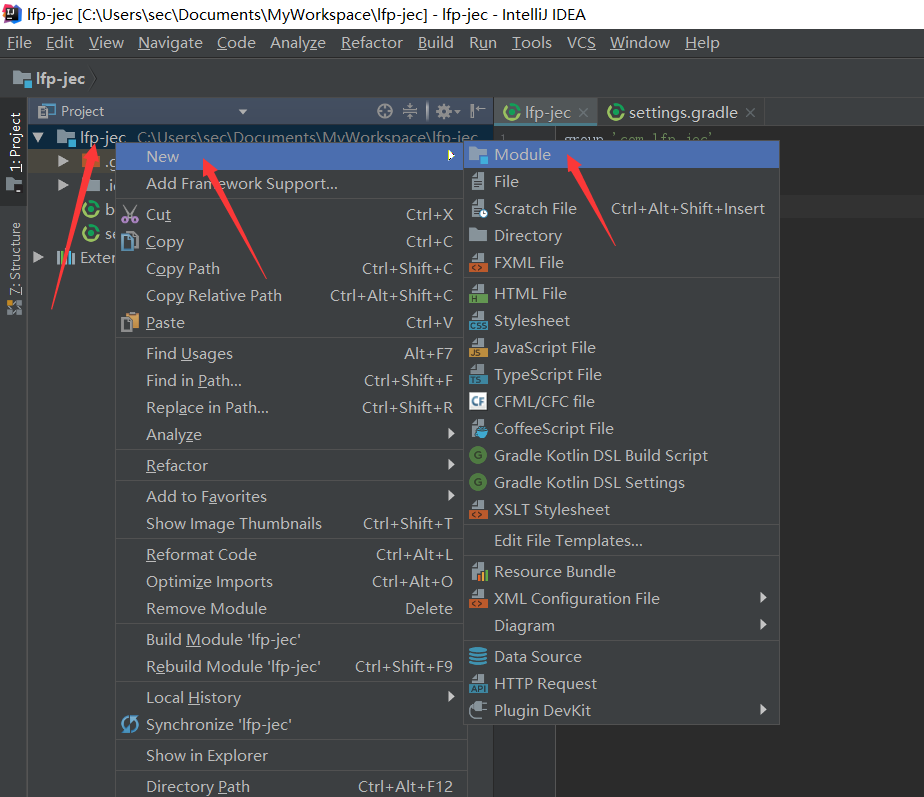


获得Gradle主工程。

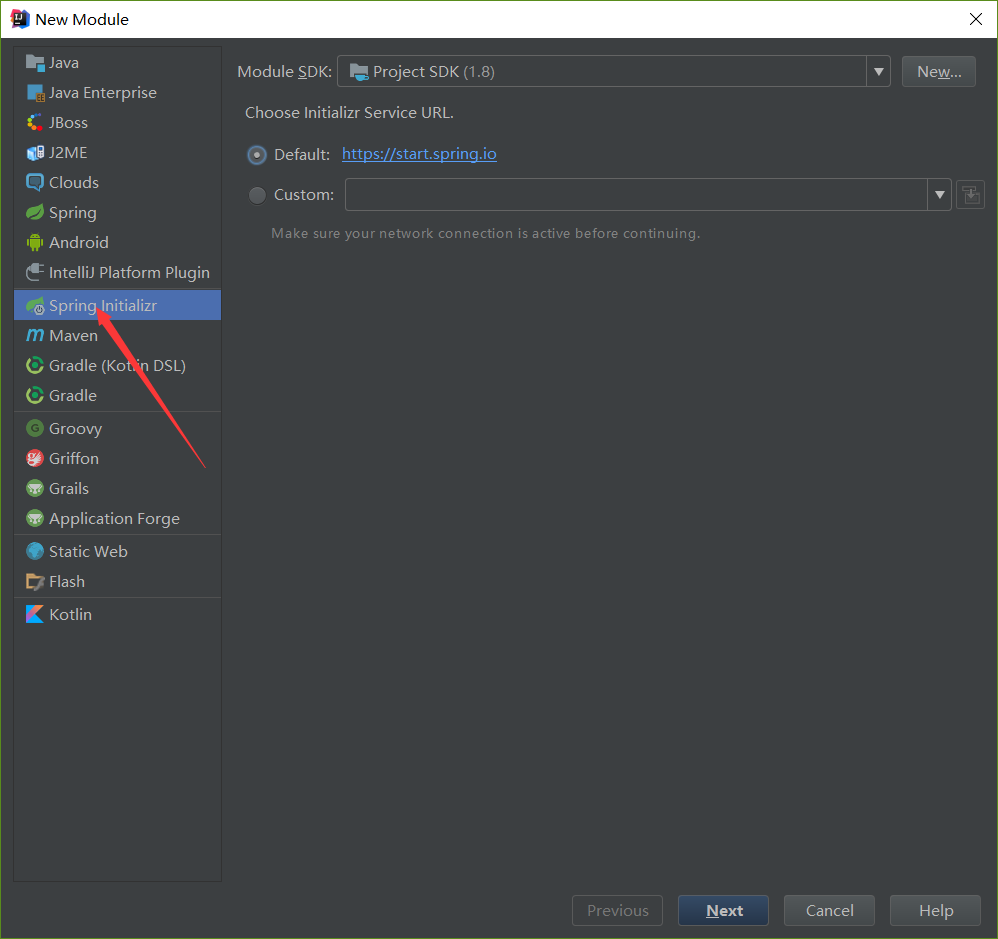


3.2、创建微服务注册中心。

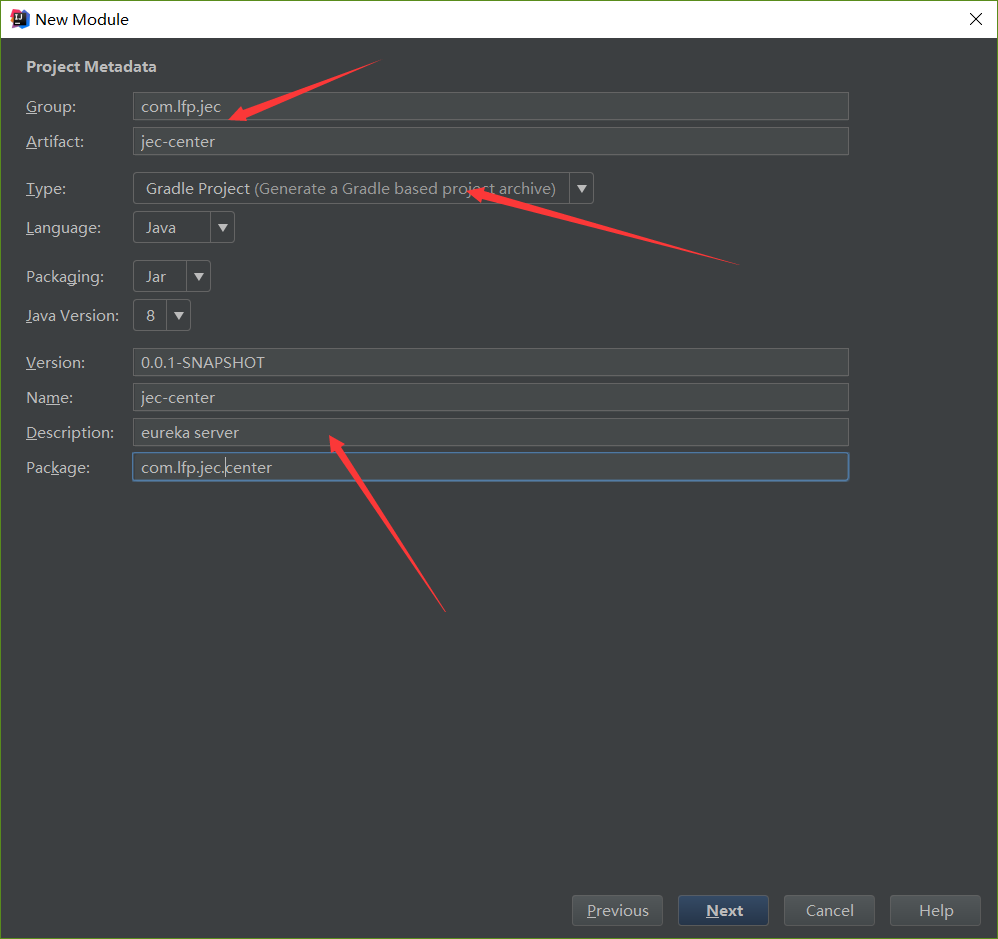
在主项目上右键，选择穿件新模块。



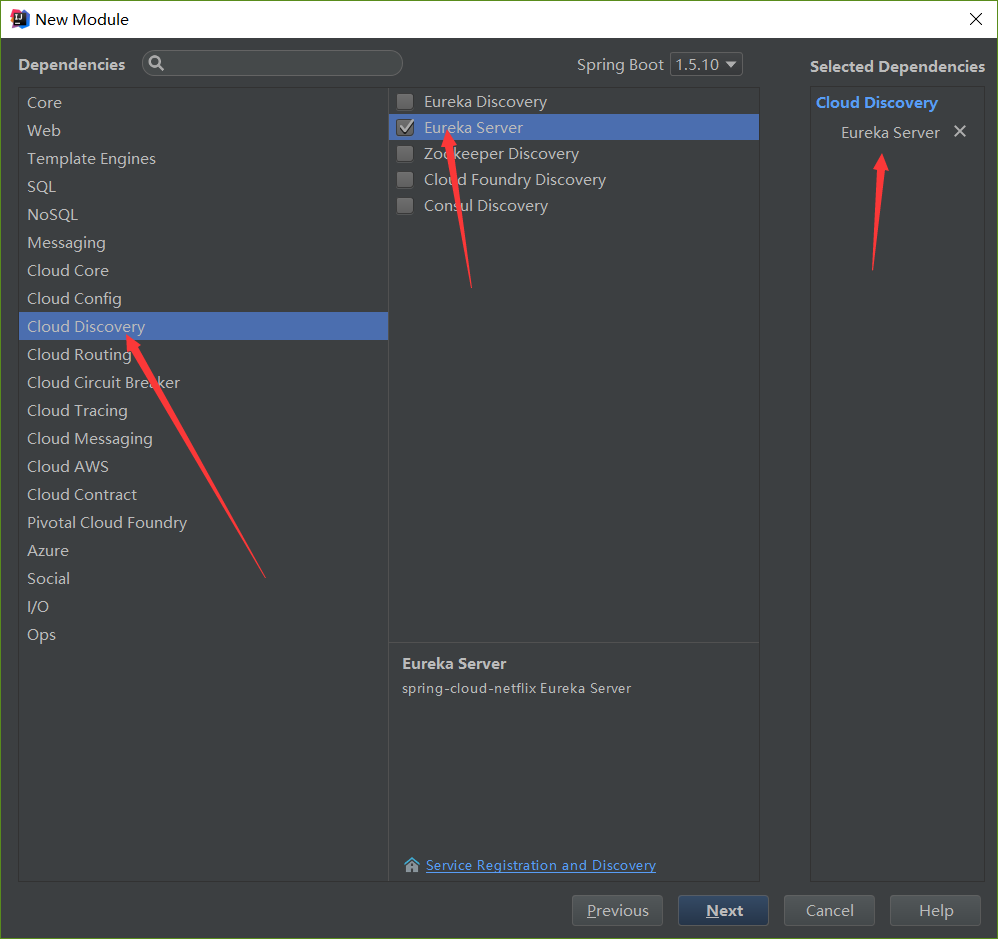
选择Spring Initializr。



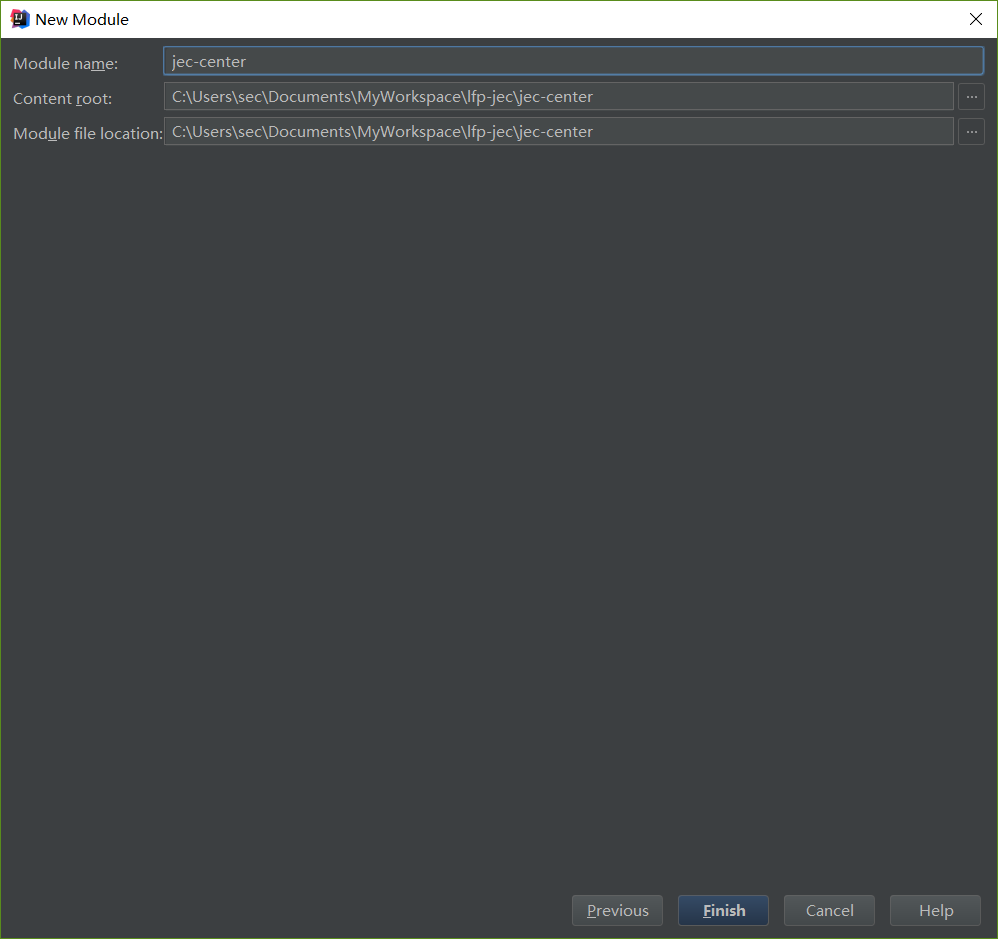
输入模块的group和artifact等信息。



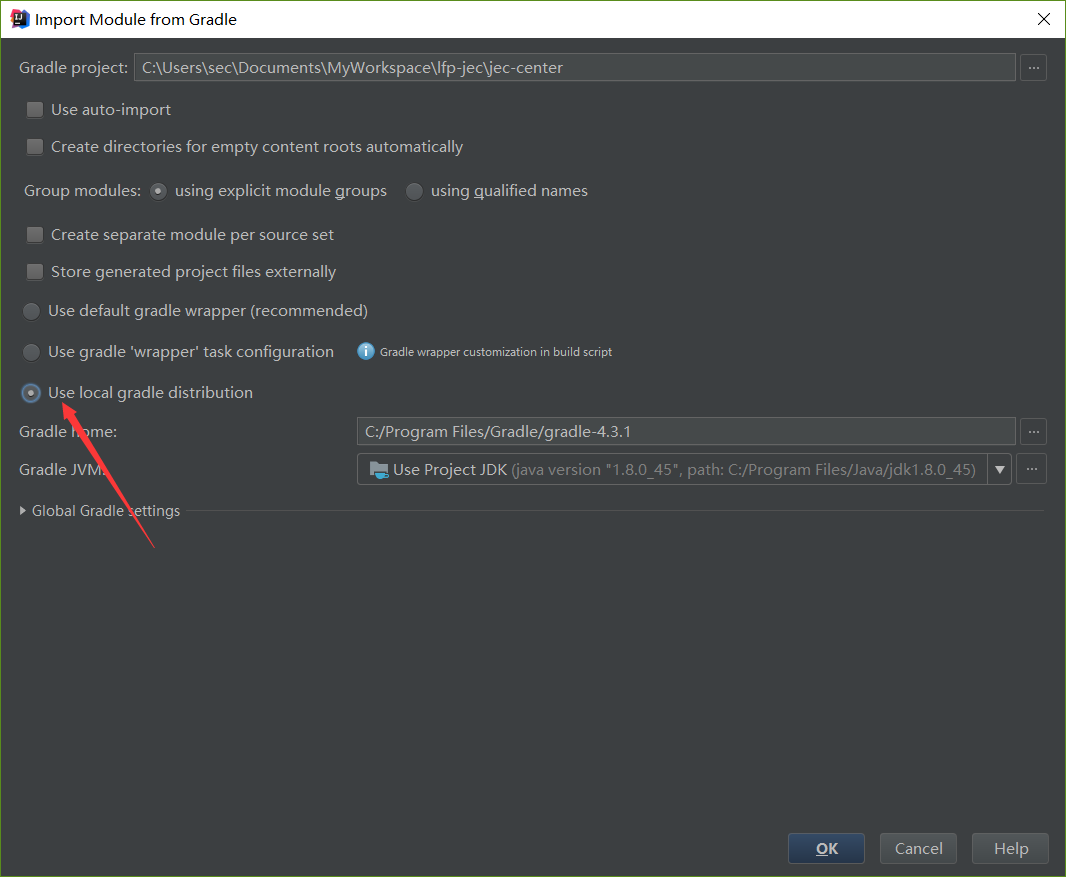
选择Eureka Server起步。



设置模块的目录位置，默认即可。



因为选择了Gradle管理，会再弹出创建Gradle模块的界面，与创建主工程相同。



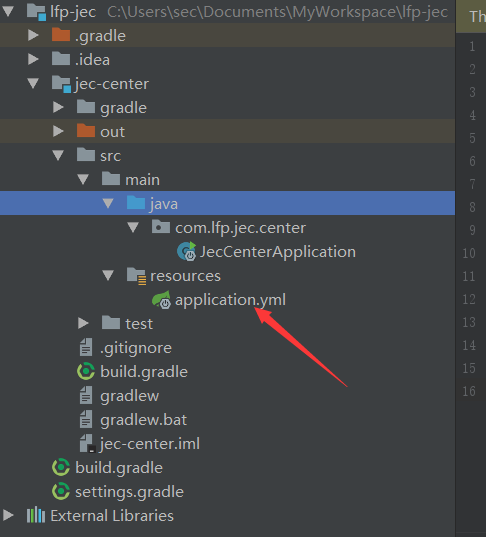
完成创建后，IDE会自动生成gradle文件，声明了该模块的所有构建信息。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册中心  
 compile('org.springframework.cloud:spring-cloud-starter-eureka-server')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

我们还需要在springboot工程的启动application类上加一个注解@EnableEurekaServer，这个注解声明该工程是一个微服务注册中心。

package com.lfp.jec.center;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@SpringBootApplication  
//微服务 - 注册中心  
@EnableEurekaServer  
public class JecCenterApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecCenterApplication.class, args);  
 }  
}

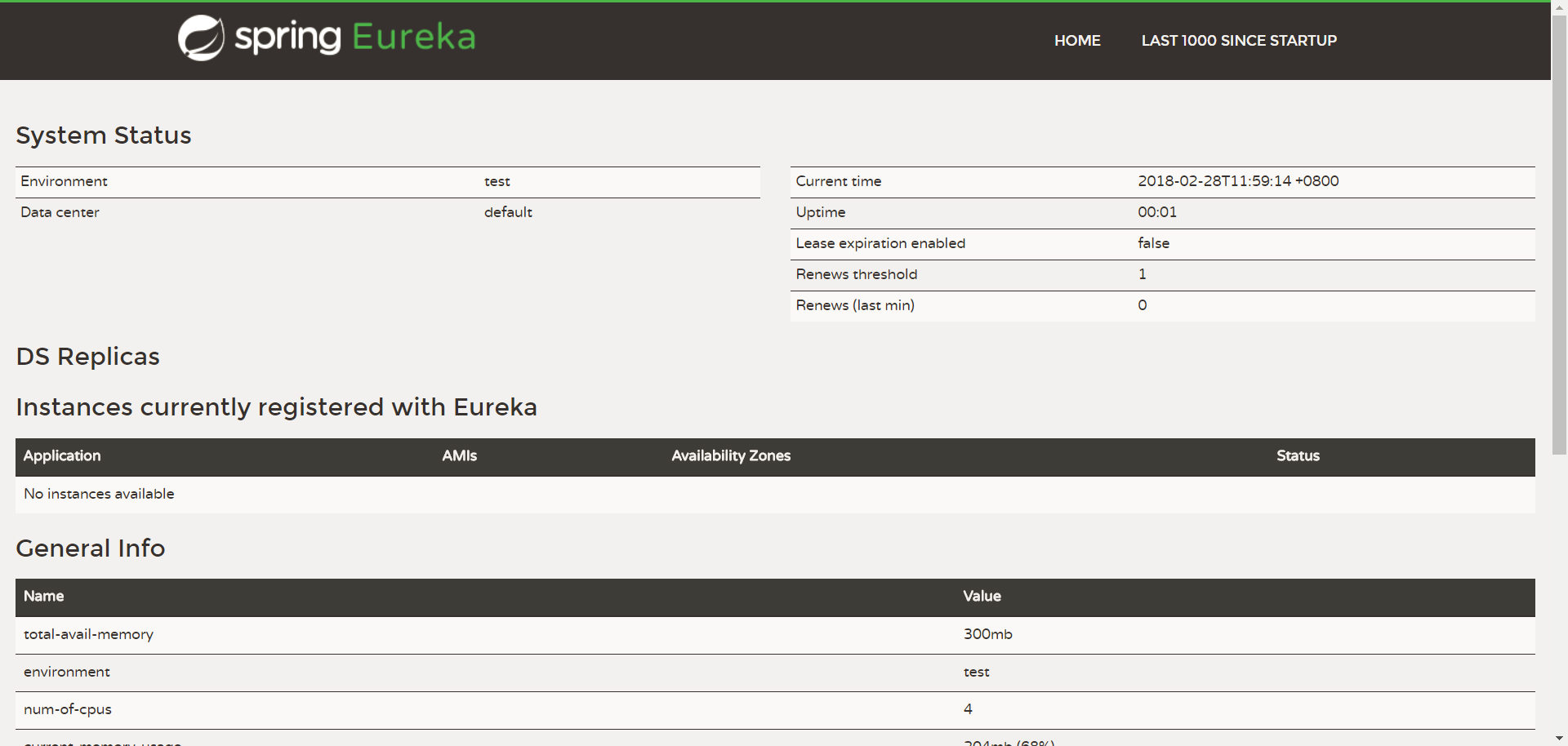
在resources中添加配置文件application.yml，也可以是application.properties。



该文件中记录工程的配置，包括配置为注册中心，端口等。

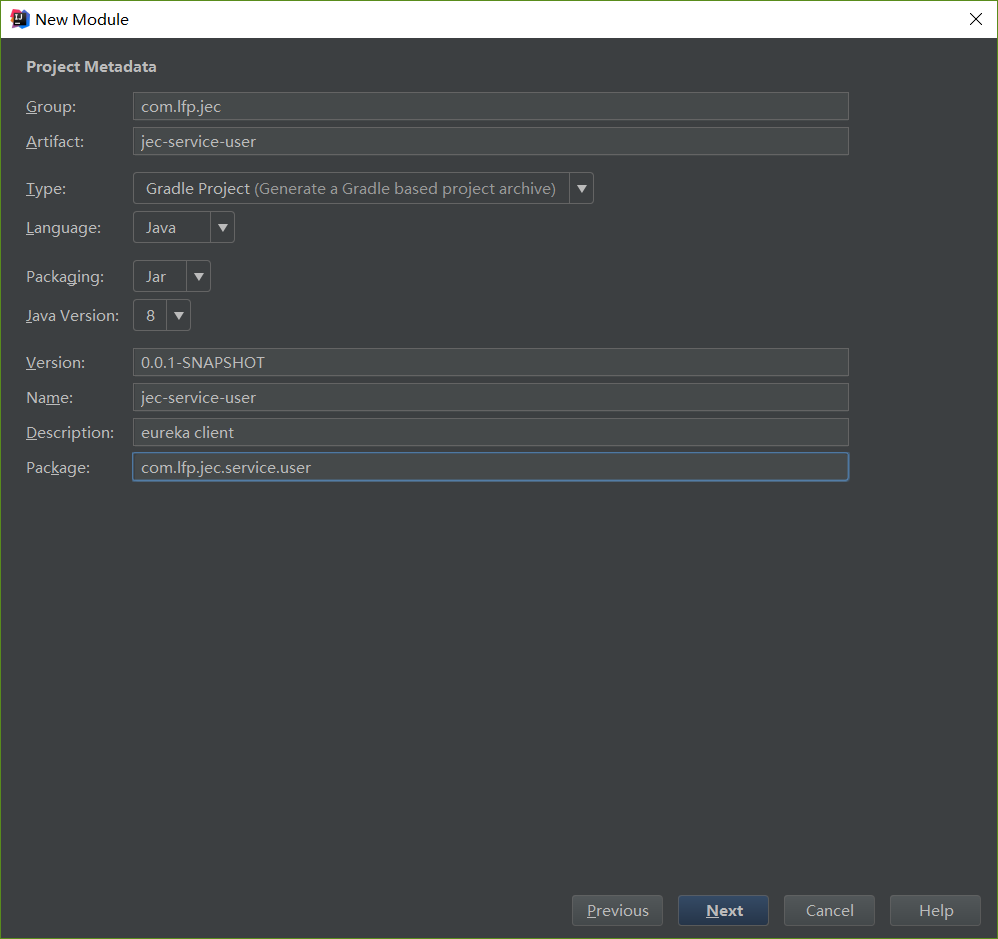
**server:  
 port:** 10000  
  
**eureka:  
 instance:  
 hostname:** localhost  
 **client:  
 registerWithEureka: false  
 fetchRegistry: false  
 serviceUrl:  
 defaultZone:** http://${**eureka.instance.hostname**}:${**server.port**}/eureka/

启动工程，打开浏览器访问：http://localhost:10000/，界面如下：

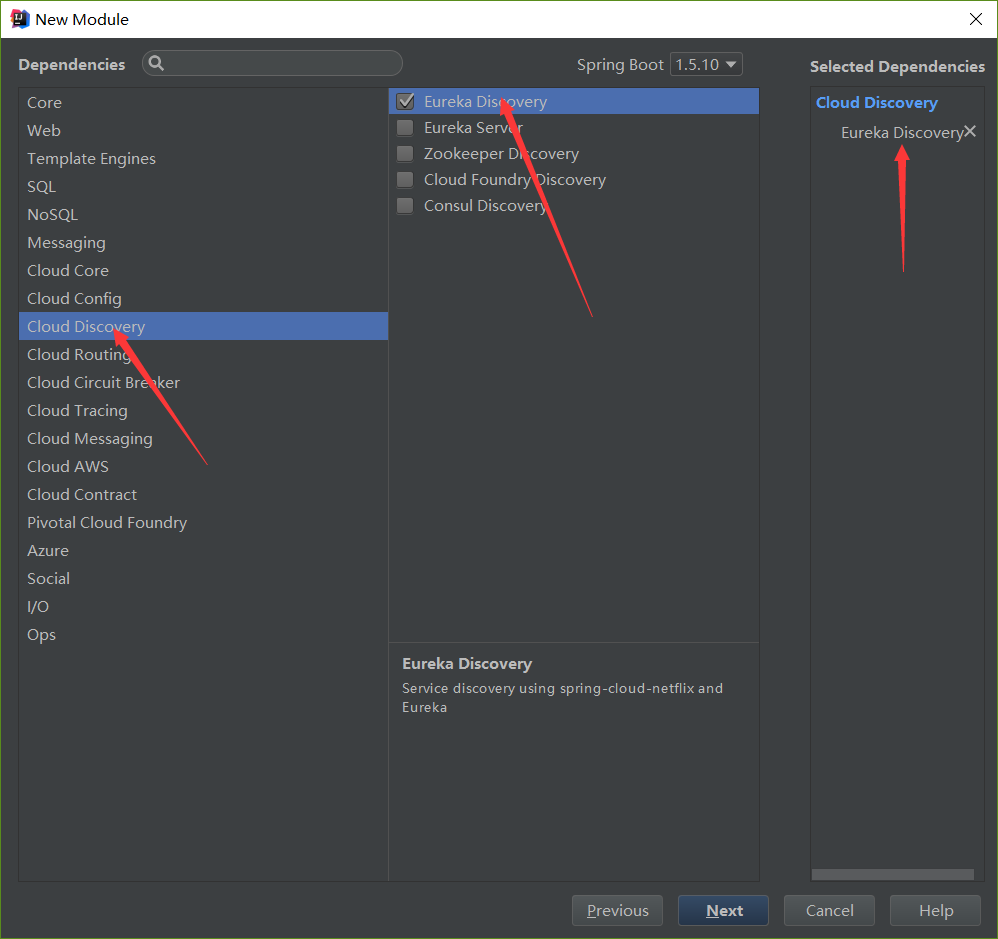


3.3、创建微服务注册服务。

与创建注册中心相同，同样选择Spring initializr，输入注册服务的必要信息。



这里在选择spring cloud起步依赖时，选择Eureka Discovery，声明为Eureka Client。



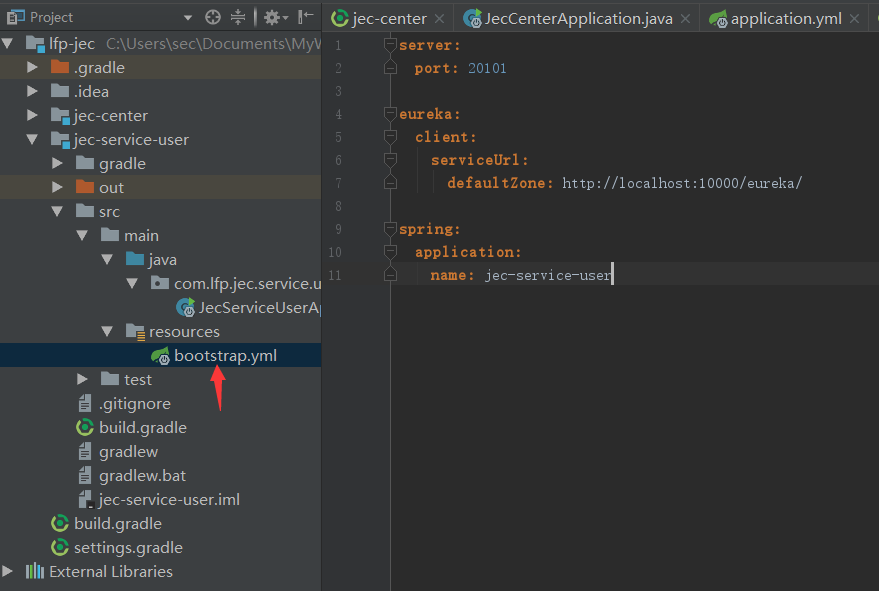
完成后会自动生成gradle文件。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

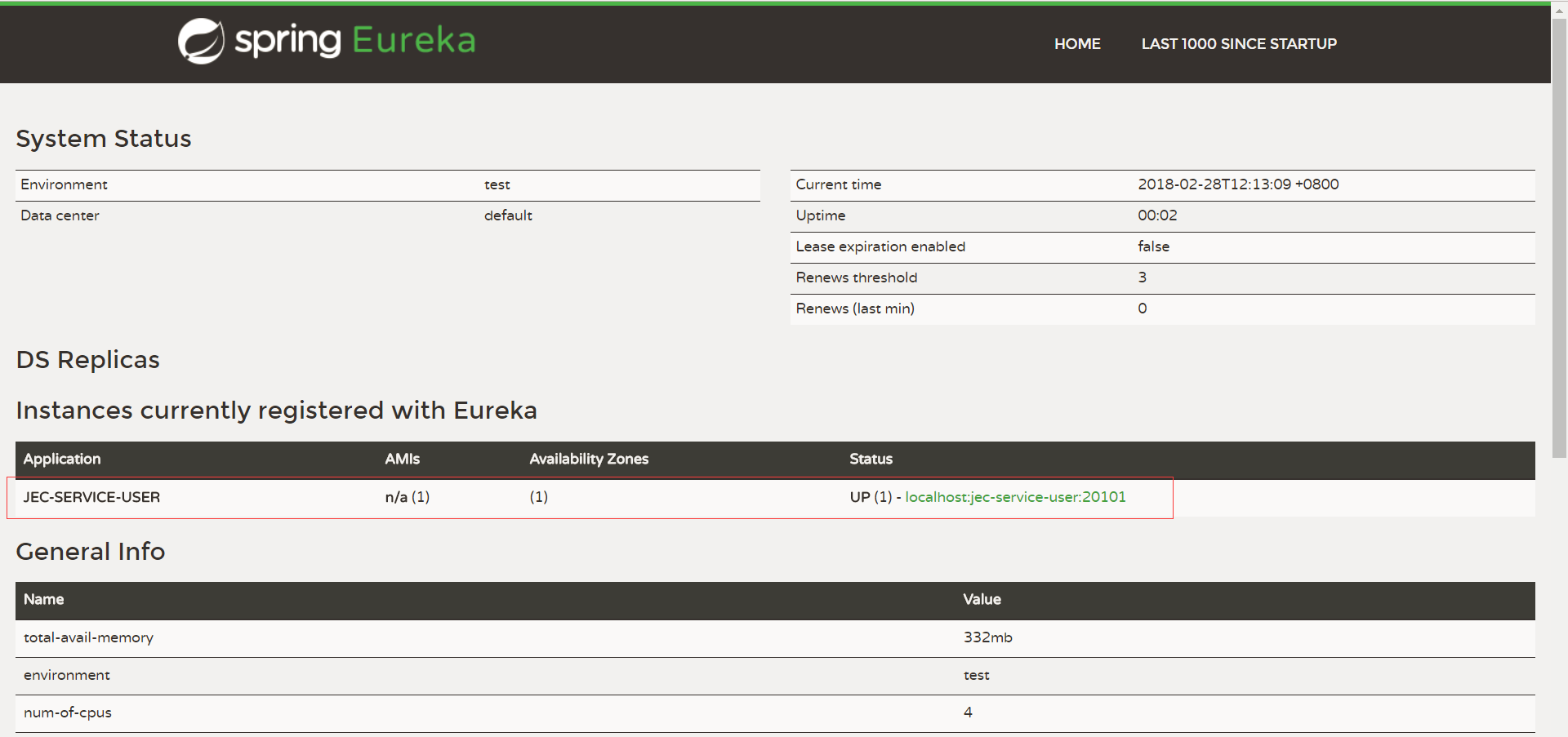
同样，我们还需要在springboot工程的启动application类上加一个注解@EnableEurekaClient，这个注解声明该工程是一个微服务注册服务。

package com.lfp.jec.service.user;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
public class JecServiceUserApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecServiceUserApplication.class, args);  
 }  
}

在resources中添加配置文件bootstrap.yml，也可以是bootstrap.properties。（这里也可以使用application.yml或application.properties，不过在存在非8888端口的配置服务时，application的部分配置无法生效，这个后续内容会涉及。）



先后启动注册中心和注册服务后，开启注册中心界面，可以看到注册服务已经注册到注册中心了。



基于SpringCloud的企业级微服务架构入门（二）：微服务的负载均衡

**1、简要概述**

在微服务架构中，业务会被拆分成一个个独立的功能服务，服务与服务的通讯是基于http restful的。Spring cloud主要有两种服务调用方式，一种是ribbon + restTemplate，另一种是feign，而feign默认也集成了ribbon。

对于部分高频访问的服务单元，我们更希望可以部署多个应用副本，从而进行有效的负载均衡，而ribbon就是一个负载均衡客户端，我们就是利用ribbon负载均衡的能力构建微服务的负载均衡。



**2、开发环境：**

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

springCloudVersion = 'Edgware.SR2'

Github <https://github.com/LittleFlowerPig/lfp-jec.git>

**3、实际操作**

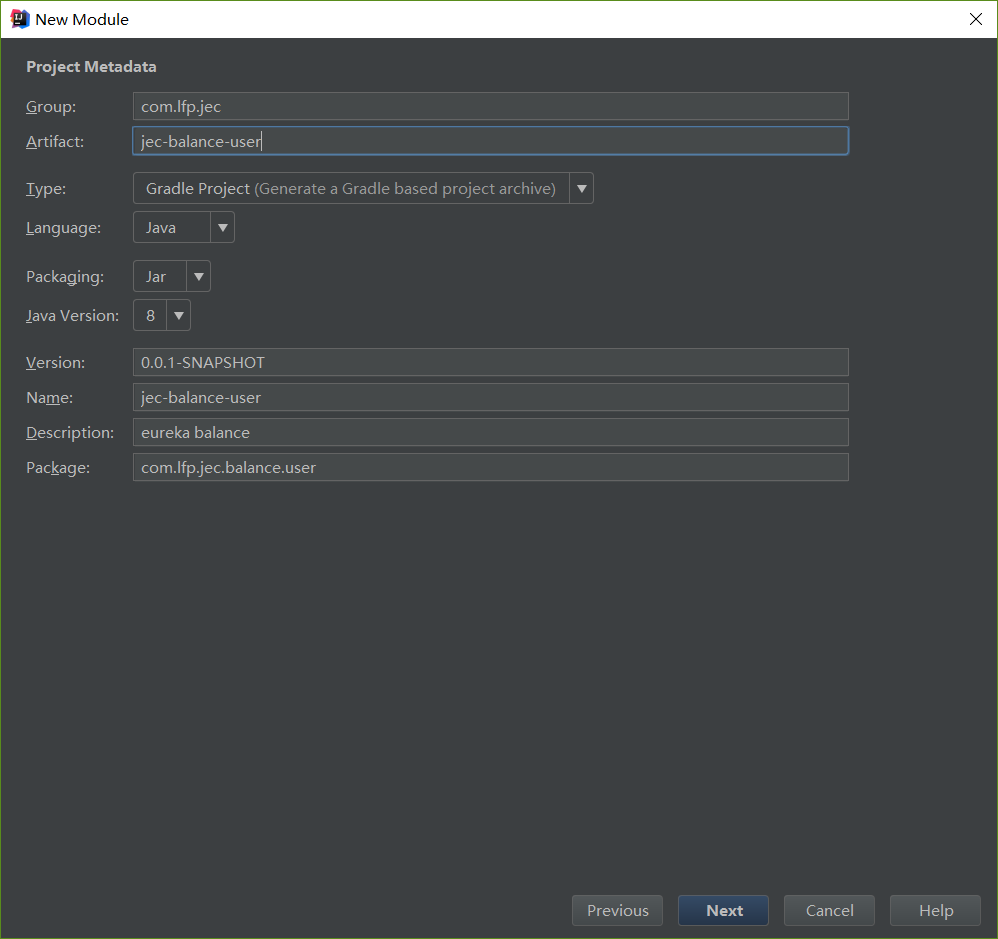
3.1、完善注册服务的功能接口

在jec-service-user工程中添加business包，并添加IndexApi类，用于模拟功能接口，输出信息。

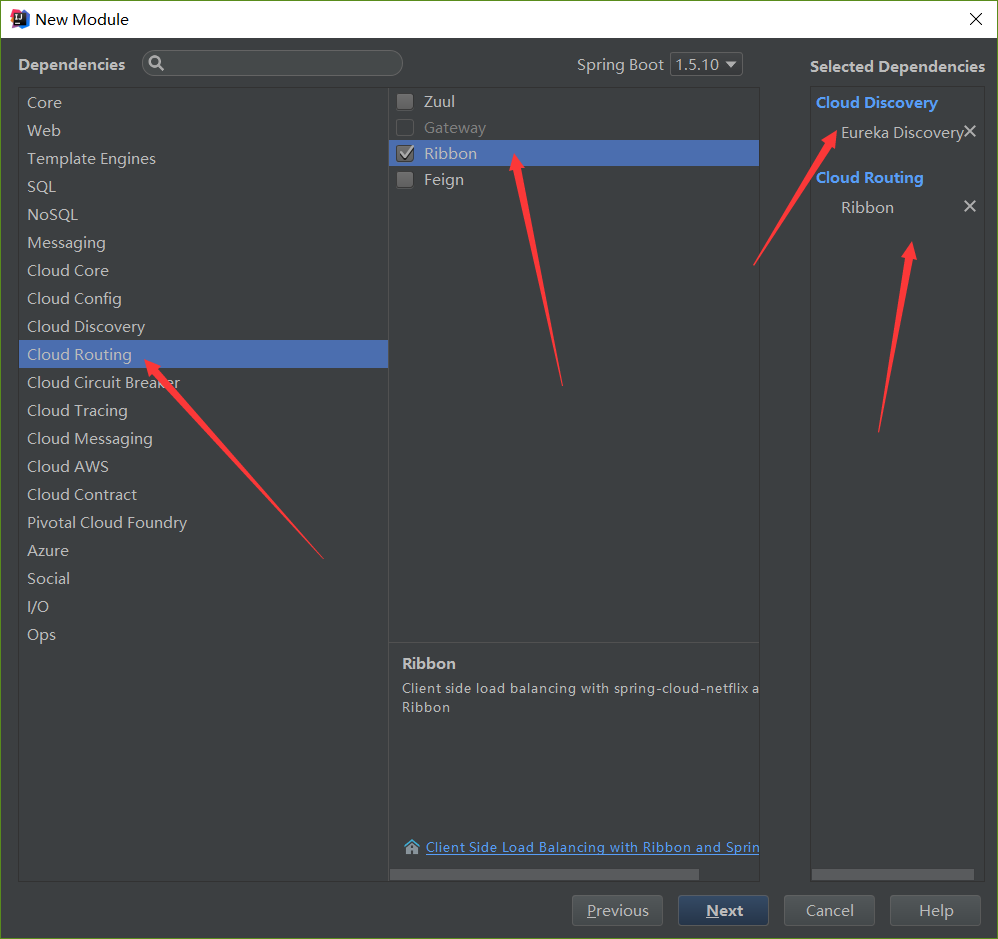
package com.lfp.jec.service.user.business;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class IndexApi {  
  
 @Value("${server.port}")  
 private String port;  
  
 @RequestMapping("/")  
 public String index() {  
 return "Hello World! I am jec-service-user at port: "+ port;  
 }  
  
}

3.2、创建微服务负载均衡（Ribbon）

继续上一篇的工程，添加新的模块工程。一样选择Spring initializr，输入负载均衡的必要信息。



选择Eureka Discovery和Ribbon同时作为该工程的起步依赖。



完成后IDE生成的gradle文件。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 //负载均衡 - ribbon  
 compile('org.springframework.cloud:spring-cloud-starter-ribbon')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

同样，在resources中添加配置文件application.yml。

**server:  
 port:** 20100  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-balance-user

我们还需要在springboot工程的启动application类上加一个注解@EnableEurekaClient，同时需要声明一个具有负载均衡能力的RestTemplate。

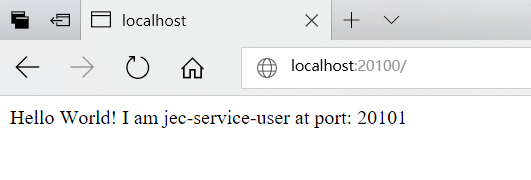
package com.lfp.jec.balance.user;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.loadbalancer.LoadBalanced;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.context.annotation.Bean;  
import org.springframework.web.client.RestTemplate;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
public class JecBalanceUserApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecBalanceUserApplication.class, args);  
 }  
  
 @Bean  
 @LoadBalanced  
 RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
  
}

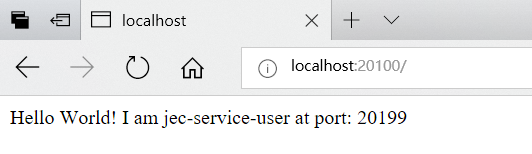
添加对注册服务功能接口的调用，在jec-balance-user工程中也添加business包，并添加IndexApi类，用于调用jec-service-user中IndexApi类提供的接口方法。

package com.lfp.jec.balance.user.business;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import org.springframework.web.client.RestTemplate;  
  
@RestController  
public class IndexApi {  
  
 @Autowired  
 RestTemplate restTemplate;  
  
 @RequestMapping("/")  
 public String index() {  
 return restTemplate.getForObject("http://jec-service-user/", String.class);  
 }  
  
}

该类中注入具有负载均衡能力的RestTemplate，调用服务名为“jec-service-user”的“/”地址的接口，并作为自己的结果返回。

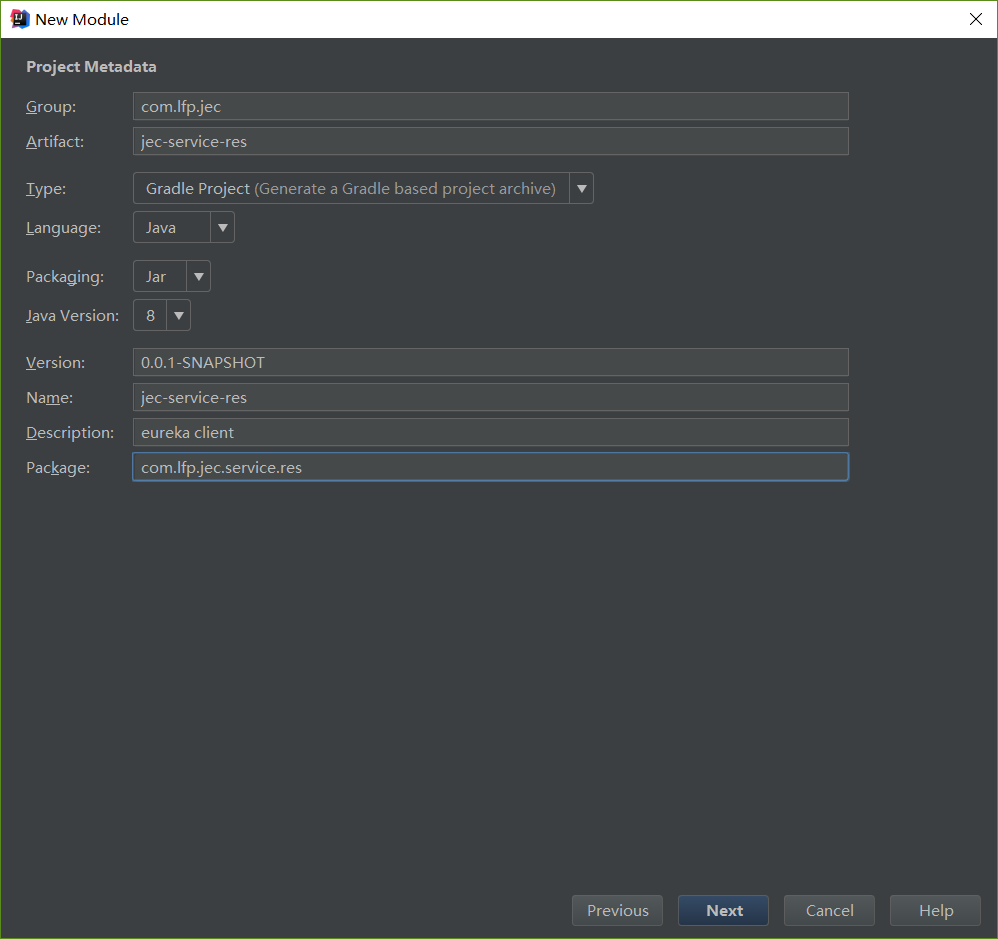
依次启动jec-center、jec-service-user(改变端口启动多个实例)、jec-balance-user。反复刷新，可以看到访问的不同jec-service-user实例提供的服务。





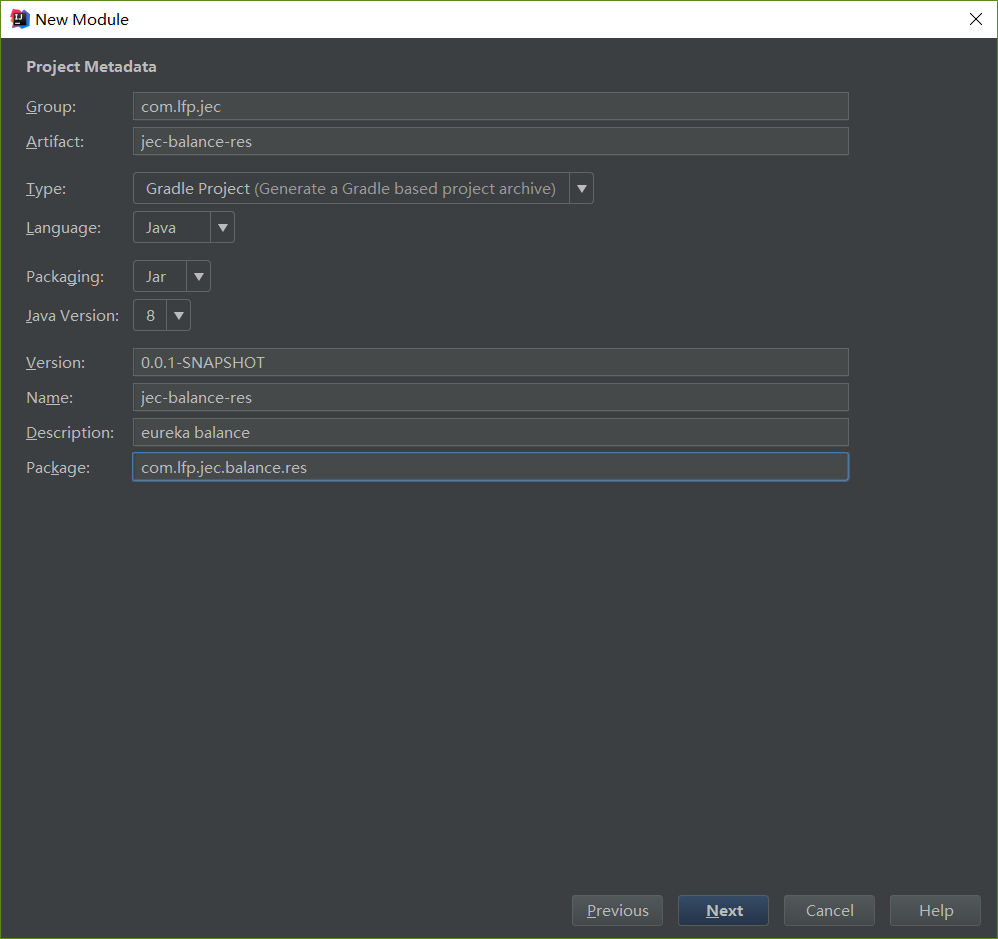
3.3、创建微服务注册服务jec-service-res

仿照jec-service-user，创建另一个注册服务jec-service-res，将来用于管理资源权限。

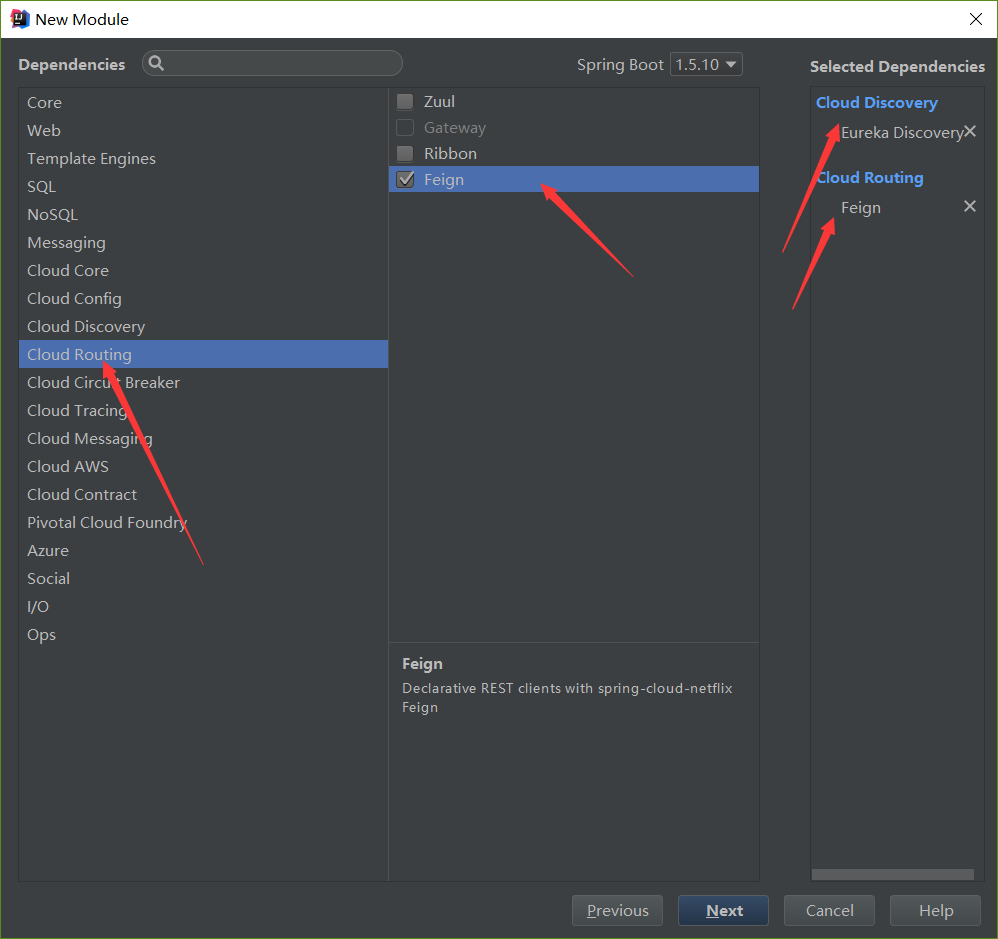


3.4、创建微服务负载均衡（Feign）

添加新的模块工程。一样选择Spring initializr，输入负载均衡的必要信息。



选择Eureka Discovery和Feign同时作为该工程的起步依赖。



完成后IDE生成的gradle文件。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 //负载均衡 - feign  
 compile('org.springframework.cloud:spring-cloud-starter-feign')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

同样，在resources中添加配置文件application.yml。

**server:  
 port:** 29900  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-balance-res

需要在springboot工程的启动application类上额外添加一个注解@EnableFeignClients，声明其为Feign的客户端。

package com.lfp.jec.balance.res;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.cloud.netflix.feign.EnableFeignClients;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
//负载均衡 - feign  
@EnableFeignClients  
public class JecBalanceResApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecBalanceResApplication.class, args);  
 }  
}

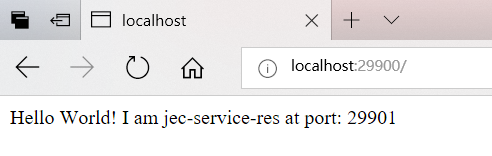
定义一个feign接口，通过@FeignClient（“服务名”），来指定调用的服务。比如在代码中调用了jec-service-res服务的“/”接口，代码如下：

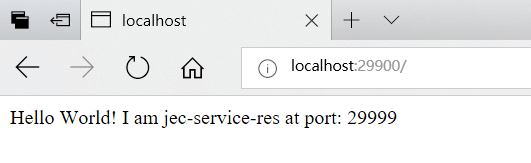
package com.lfp.jec.balance.res.business;  
  
import org.springframework.cloud.netflix.feign.FeignClient;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@FeignClient(value = "jec-service-res")  
public interface Index {  
  
 @RequestMapping("/")  
 String index();  
  
}

添加对注册服务功能接口的调用，在jec-balance-res工程中添加IndexApi类，用于调用feign接口中声明的方法。

package com.lfp.jec.balance.res.business;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class IndexApi{  
  
 @Autowired  
 private Index index;  
  
 @RequestMapping("/")  
 public String index() {  
 return index.index();  
 }  
  
}

依次启动jec-center、jec-service-res(改变端口启动多个实例)、jec-balance-res。反复刷新，可以看到访问的不同jec-service-res实例提供的服务。





基于SpringCloud的企业级微服务架构入门（三）：微服务的路由网关

**1、简要概述**

一个应用在被拆分为多个微服务后，最终仍然需要按照一定的规则装配在固定的站点向外界提供功能接口。外界请求访问统一的站点，由该站点根据请求的路径信息进行路由，分配到不同的微服务单元中进行处理，在spring cloud中担任此角色的正是zuul。

Zuul的主要功能是路由转发和过滤器。路由功能是微服务的一部分，比如／api/user转发到到user服务，/api/res转发到到res服务。



**2、开发环境：**

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

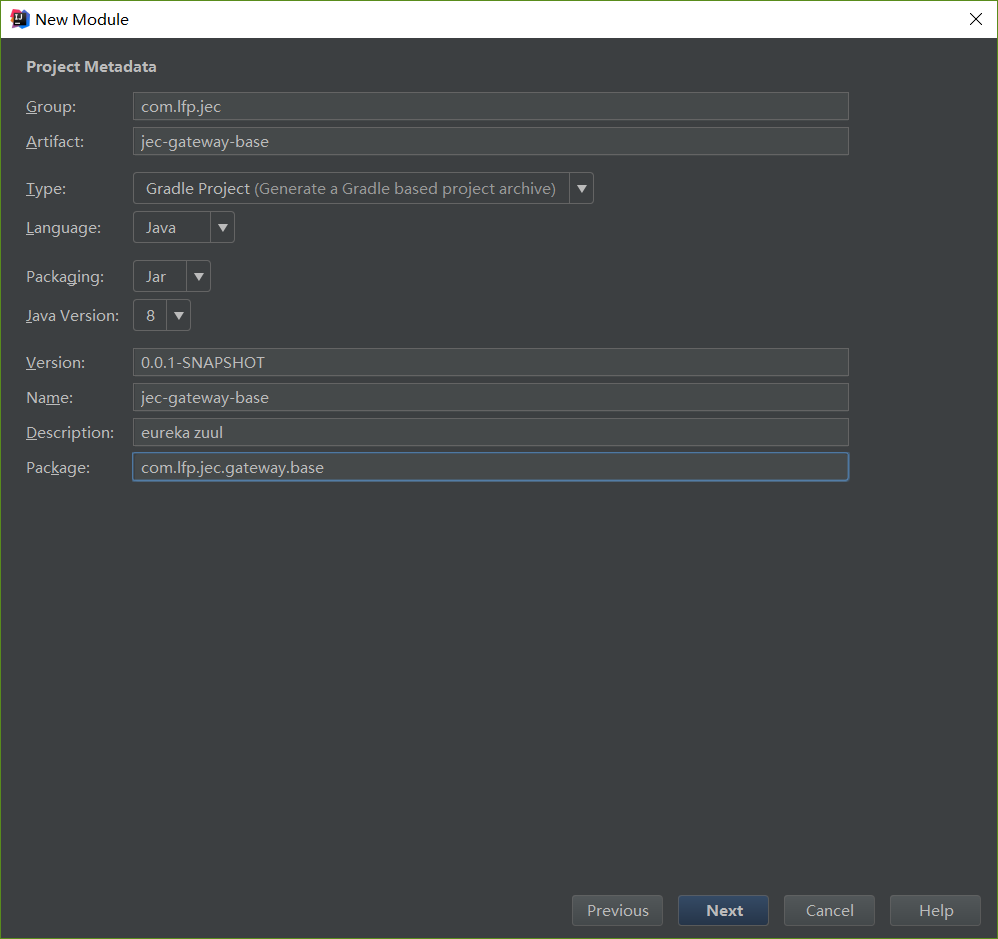
springCloudVersion = 'Edgware.SR2'

Github <https://github.com/LittleFlowerPig/lfp-jec.git>

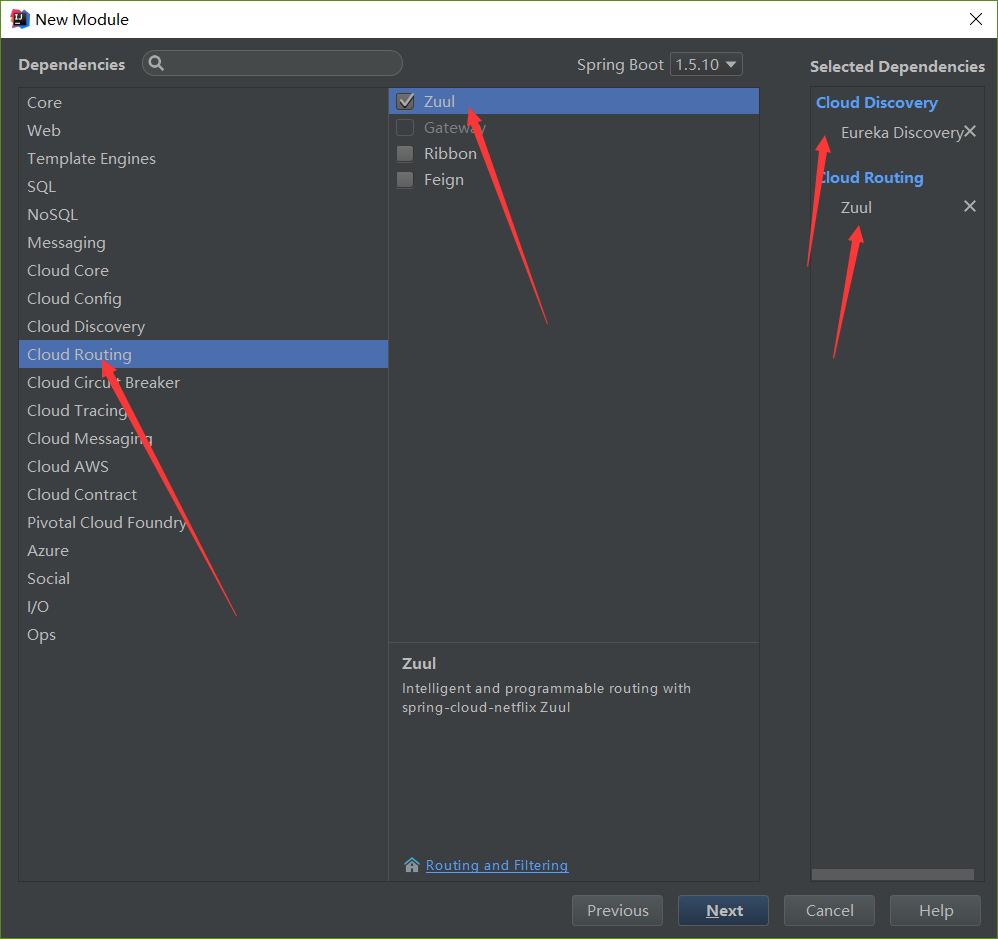
**3、实际操作**

3.1、创建微服务路由网关。

继续上一篇的工程，添加新的模块工程。一样选择Spring initializr，输入路由网关的必要信息。



选择Eureka Discovery和Zuul同时作为该工程的起步依赖。



完成后IDE生成的gradle文件。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 //API网关 - zuul  
 compile('org.springframework.cloud:spring-cloud-starter-zuul')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

同样，在resources中添加配置文件application.yml。在配置文件中添加了zuul的静态路由配置。

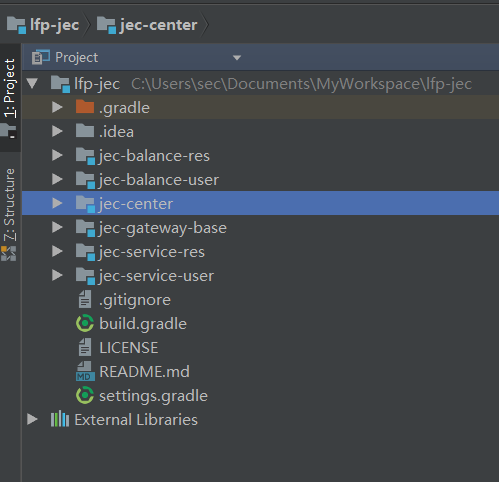
**server:  
 port:** 30101  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-gateway-base  
  
**zuul:  
 routes:  
 user:  
 path:** /user/\*\*  
 **serviceId:** jec-balance-user  
 **res:  
 path:** /res/\*\*  
 **serviceId:** jec-balance-res

需要在springboot工程的启动application类上额外添加一个注解@EnableZuulProxy，声明其为Zuul路由网关。

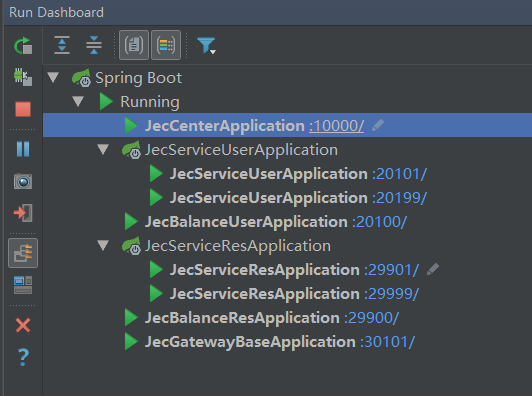
package com.lfp.jec.gateway.base;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
//API网关 - zuul  
@EnableZuulProxy  
public class JecGatewayBaseApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecGatewayBaseApplication.class, args);  
 }  
}

3.2、启动并验证

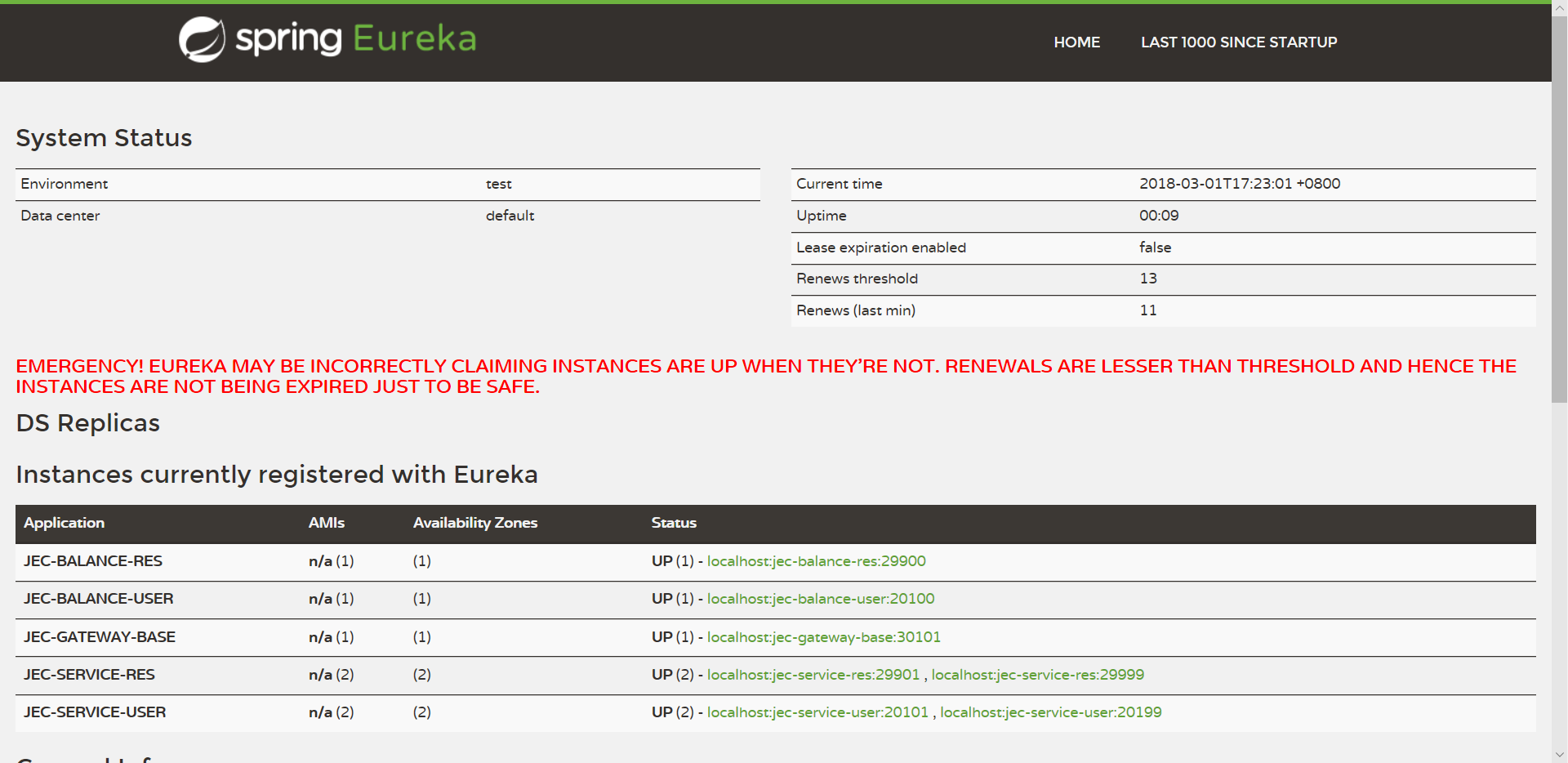
目前的工程目录情况如下，先后启动这些应用，其中service需要启动多实例。



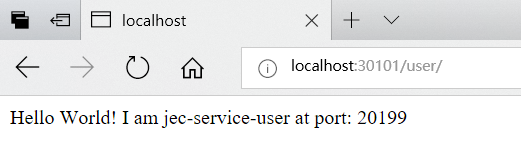
启动后的结果如下。

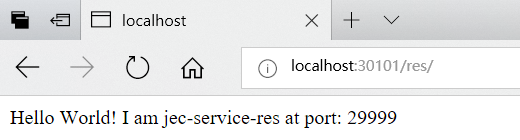


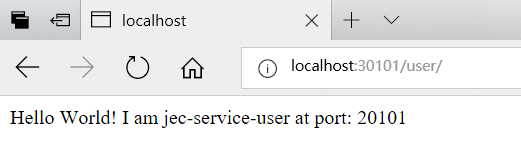
注册中心管理界面如下

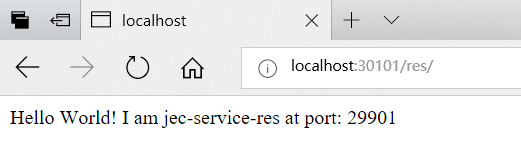


依次输入地址可以路由到指定的微服务中。









基于SpringCloud的企业级微服务架构入门（四）：微服务的配置服务

**1、简要概述**

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloud config，它支持配置服务放在配置服务的本地，也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，即配置服务，二是config client，一般即为常规的注册服务。

配置服务连接了注册服务获取配置文件的桥梁，这样通过配置服务获取/加工过的配置信息可以直接在注册服务中注解使用。有关注册服务使用配置信息将在下一篇中详细介绍。



**2、开发环境：**

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

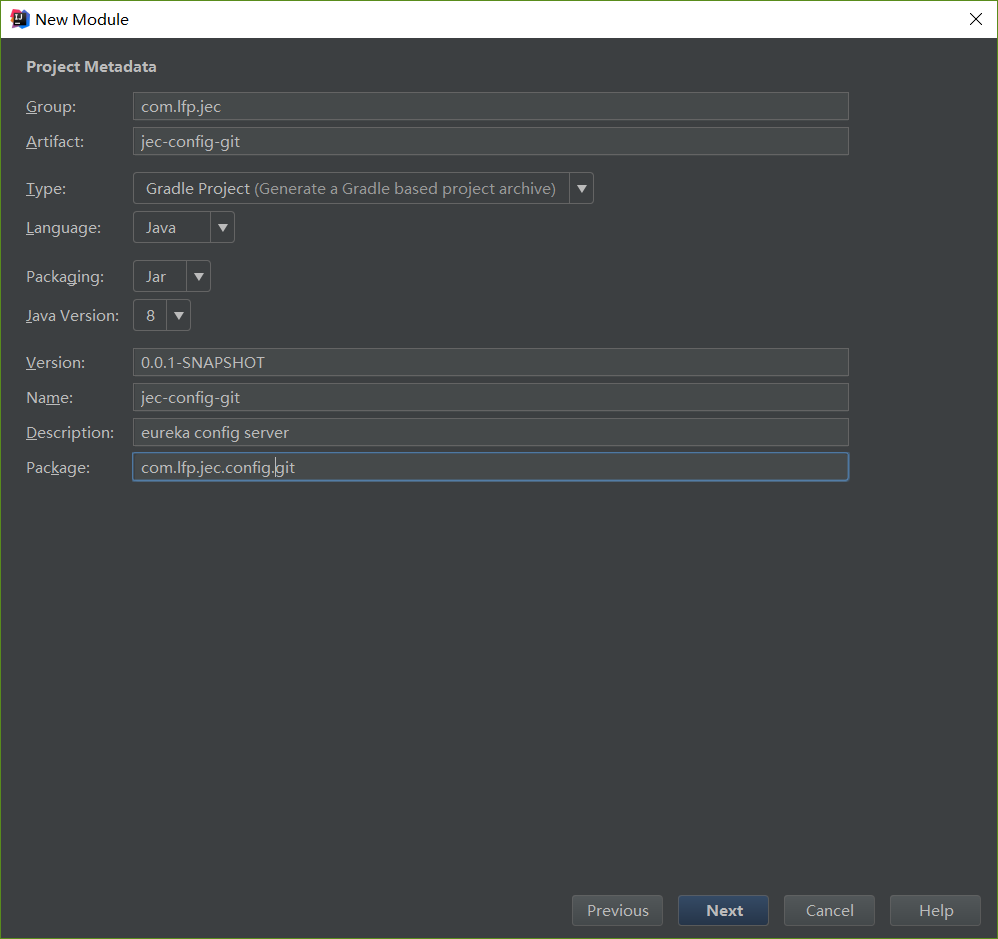
springCloudVersion = 'Edgware.SR2'

Github <https://github.com/LittleFlowerPig/lfp-jec.git>

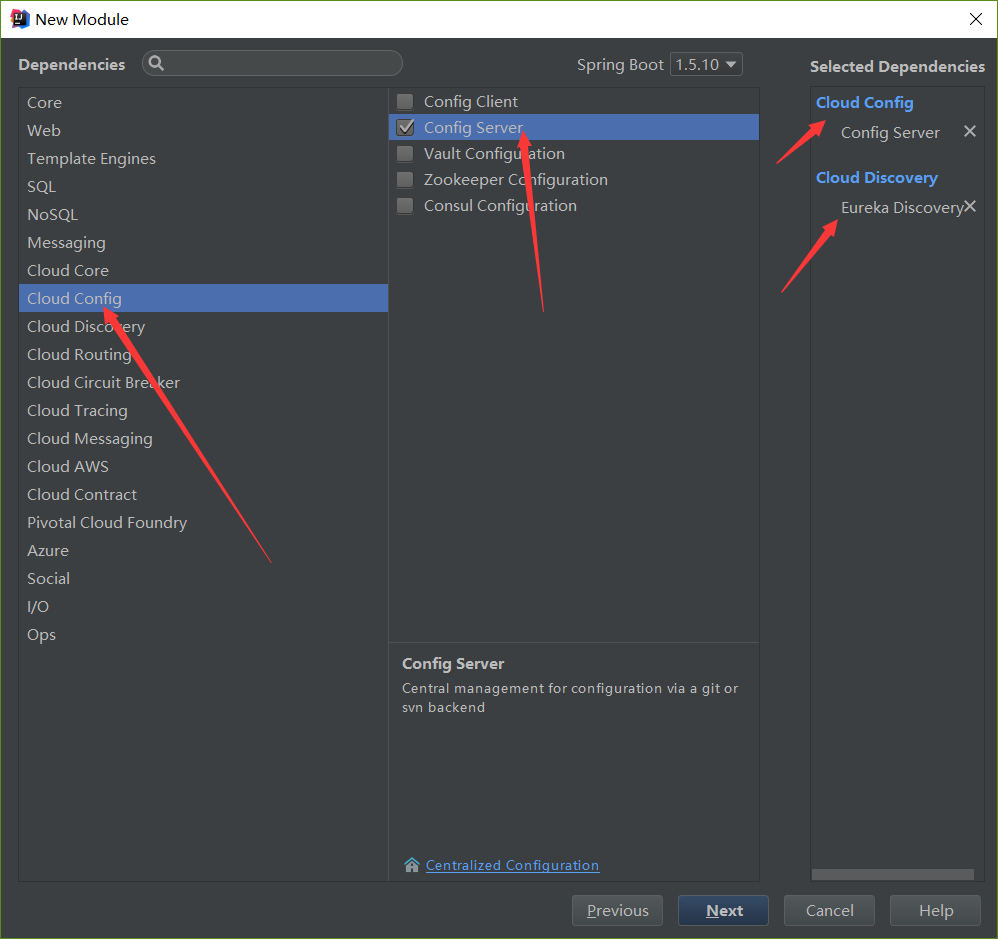
**3、实际操作**

3.1、创建微服务配置服务（Git）

继续上一篇的工程，添加新的模块工程。一样选择Spring initializr，输入配置服务的必要信息。



选择Eureka Discovery和Config Server同时作为该工程的起步依赖。



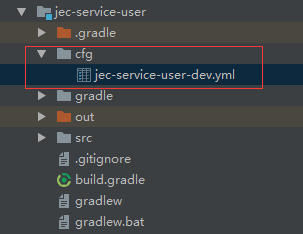
完成后IDE生成的gradle文件。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 //配置服务 - 服务端  
 compile('org.springframework.cloud:spring-cloud-config-server')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

同样，在resources中添加配置文件application.yml。在配置文件中添加读取配置文件的信息。

**server:  
 port:** 40101  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-config-git  
 **cloud:  
 config:  
 label:** master  
 **server:  
 git:  
 uri:** https://github.com/LittleFlowerPig/lfp-jec.git  
 **search-paths:** jec-service-\*/cfg/

根据配置指出，将从git版本库的文件夹jec-service-\*/cfg/中搜索相应的配置文件加载。我们则在jec-service-user中创建cfg/jec-service-user-dev.yml文件。



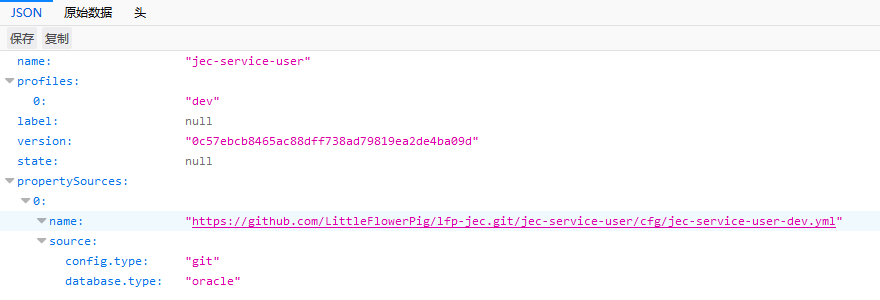
该配置文件中进行必要的配置项的书写。

**config:  
 type:** git  
**database:  
 type:** oracle

需要在springboot工程的启动application类上额外添加一个注解@EnableConfigServer，声明其为配置服务的服务端。

package com.lfp.jec.config.git;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.config.server.EnableConfigServer;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
//配置服务 - 服务端  
@EnableConfigServer  
public class JecConfigGitApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecConfigGitApplication.class, args);  
 }  
}

3.2、配置服务验证（Git）

依次启动jec-center和jec-config-git，浏览器输入地址<http://localhost:40101/jec-service-user/dev>，则在github中jec-service-user中的配置文件jec-service-user-dev.yml 被读出。

http请求地址和资源文件映射有一定的关系，请求地址为/{application}/{profile}[/{label}]，将被检索如下文件：

/{application}-{profile}.yml

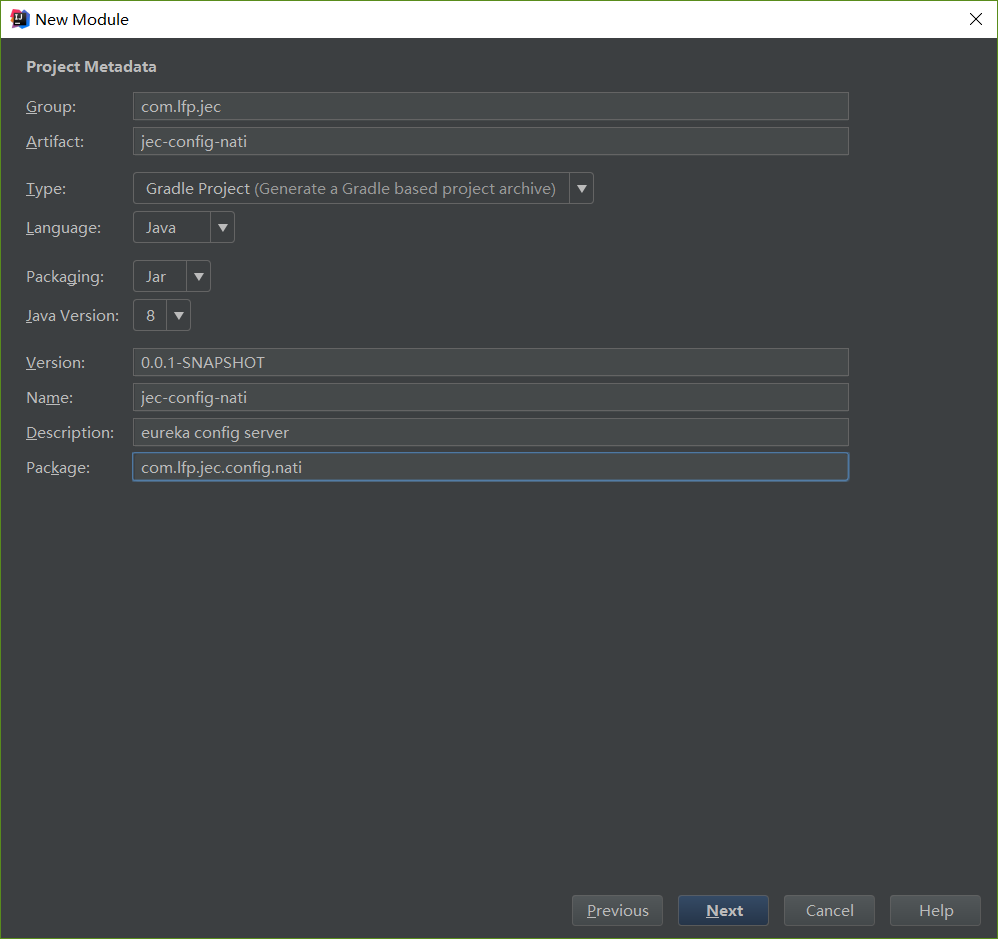
/{application}-{profile}.properties

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

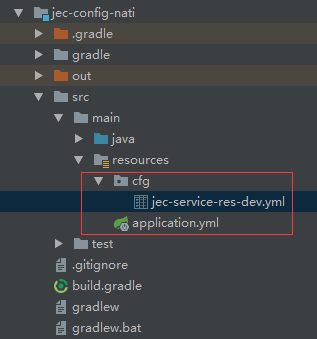
/{label}/{application}-{profile}.properties

3.3、创建微服务配置服务（Native）

添加新的模块工程。一样选择Spring initializr，输入配置服务的必要信息。



剩余部分与基于Git的配置服务创建时完全相同。不过此时配置文件不用散落在各个微服务中，可以在配置服务中集中管理。



同样，在resources中添加配置文件application.yml。在配置文件中添加读取配置文件的信息。

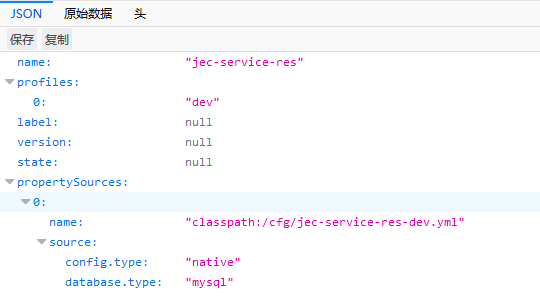
**server:  
 port:** 49999  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-config-nati  
 **profiles:  
 active:** native  
 **cloud:  
 config:  
 server:  
 native:  
 searchLocations:** classpath:/cfg/

在resources文件夹中创建cfg文件夹用于放置各种配置文件，配置文件的命名也与Git版的相同。例如我们添加了jec-service-res-dev.yml

**config:  
 type:** native  
**database:  
 type:** mysql

3.4、配置服务验证（Native）

依次启动jec-center和jec-config-nati，浏览器输入地址http://localhost:49999/jec-service-res/dev，则在cfg/jec-service-res中的配置文件jec-service-res-dev.yml 被读出。



基于SpringCloud的企业级微服务架构入门（五）：微服务的注册服务

**1、简要概述**

在第一篇中介绍的微服务的注册服务只是简单的特例，在介绍了这么多微服务的组件后，本篇将介绍整合这些功能后的较为完整的微服务的注册服务，这是实现微服务系统业务功能的核心。

注册服务首先作为配置服务的客户端config client，从配置服务中动态获取相应的配置信息，自身组织功能实现。同时相同的注册服务会启动多实例，并利用服务发现实现负载均衡，向上挂靠在对应的路由网关中，由路由网关分配相应的注册服务进行响应。

在注册服务使用配置服务时，存在配置文件改变而需要热切换的功能，Spring Cloud也提供消息总线的功能可以实现事件信息的分发。Spring Cloud Bus 将分布式的节点用轻量的消息代理连接起来。它可以用于广播配置文件的更改或者服务之间的通讯，也可以用于监控。本文要讲述的是用Spring Cloud Bus实现通知微服务架构的配置文件的更改。



**2、开发环境：**

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

springCloudVersion = 'Edgware.SR2'

Github <https://github.com/LittleFlowerPig/lfp-jec.git>

**3、实际操作**

3.1、改造注册服务使用配置服务

在jec-service-user的gradle中添加配置服务客户端的起步依赖。

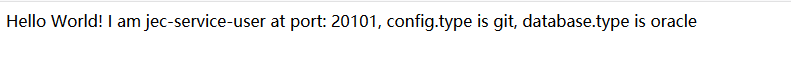
compile('org.springframework.cloud:spring-cloud-config-client')

在bootstrap.yml中添加对配置服务的使用

**server:  
 port:** 20101  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-service-user  
 **cloud:  
 config:  
 label:** master  
 **profile:** dev  
 **discovery:  
 enabled: true  
 service-id:** jec-config-git

package com.lfp.jec.service.user.business;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class IndexApi {  
  
 @Value("${server.port}")  
 private String port;  
  
 @Value("${config.type}")  
 private String config;  
  
 @Value("${database.type}")  
 private String database;  
  
 @RequestMapping("/")  
 public String index() {  
 return "Hello World! I am jec-service-user at port: "+ port+  
 ", config.type is "+config+  
 ", database.type is "+database;  
 }  
  
}

依次启动jec-center、jec-config-git和jec-service-user，浏览器输入地址<http://localhost:20101/>。成功将git配置文件中的配置信息读出并使用。



3.2、改造注册服务刷新配置服务

按照3.1的方法同样改造jec-service-res，不过是使用jec-config-nati的配置服务。

在jec-service-res的gradle中添加配置服务消息总线的起步依赖。

compile('org.springframework.cloud:spring-cloud-starter-bus-amqp')

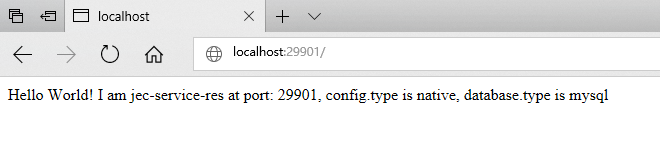
并在bootstrap.yml中配置相应的消息队列作为消息总线。

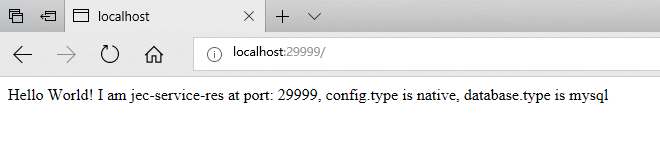
**server:  
 port:** 29901  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-service-res  
 **cloud:  
 config:  
 label:** master  
 **profile:** dev  
 **discovery:  
 enabled: true  
 service-id:** jec-config-nati  
 **rabbitmq:** *## 消息队列配置* **host:** 192.168.0.1  
 **port:** 5672  
 **virtual-host:** xxxxx  
 **username:** admin  
 **password:** 123456  
  
**management:  
 security:** *## 刷新时，关闭安全验证* **enabled: false**

在IndexApi 中添加注解@RefreshScope，声明其中使用的配置项都是可以动态变化的。

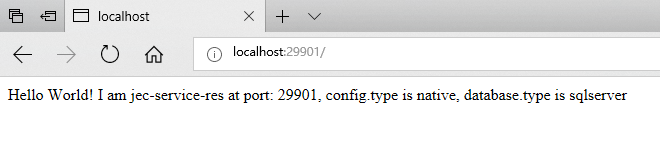
package com.lfp.jec.service.res.business;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.cloud.context.config.annotation.RefreshScope;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
//配置可以动态刷新  
@RefreshScope  
public class IndexApi {  
  
 @Value("${server.port}")  
 private String port;  
  
 @Value("${config.type}")  
 private String config;  
  
 @Value("${database.type}")  
 private String database;  
  
 @RequestMapping("/")  
 public String index() {  
 return "Hello World! I am jec-service-res at port: "+port+  
 ", config.type is "+config+  
 ", database.type is "+database;  
 }  
  
}

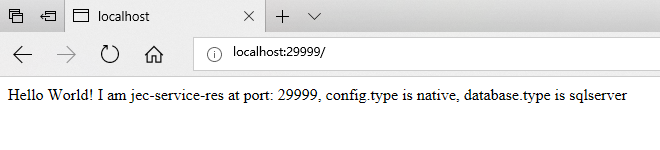
依次启动jec-center、jec-config-nati和jec-service-res（多实例），浏览器输入地址<http://localhost:29901/>或者http://localhost:29999/。成功将jec-config-nati的本地配置文件中的配置信息读出并使用。





此时修改配置文件jec-service-res-dev.yml，修改database.type为sqlserver。发现<http://localhost:29901/>或者<http://localhost:29999/>的值并未发生变化。此时，利用浏览器POST请求<http://localhost:29901/bus/refresh>。完成后再访问<http://localhost:29901/>和<http://localhost:29999/>，发现配置均已经发生了变化。





同时，我们也可以将刷新配置的功能放到配置服务中，需要改造配置服务，由其发起属性配置时间消息，而各个注册服务接收该消息后进行各自刷新。

基于SpringCloud的企业级微服务架构入门（六）：微服务的断路监控

1、简要概述

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用。为了保证高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。为了解决这个问题，业界提出了断路器模型。

Netflix开源了Hystrix组件，实现了断路器模式，Spring Cloud对这一组件进行了整合。较底层的服务如果出现故障，当对特定的服务的调用的不可用达到一个阀值（Hystric 是5秒20次） 断路器将会被打开。断路打开后，可用避免连锁故障，fallback方法可以直接返回一个固定值。

断路器的状况反应了一个程序的可用性和健壮性，它是一个重要指标。Hystrix Dashboard是作为断路器状态的一个组件，提供了数据监控和友好的图形化界面。而当我们有很多个服务的时候，这就需要聚合所有服务的Hystrix Dashboard的数据了。这就需要用到Spring Cloud的另一个组件了，即Hystrix Turbine。

2、开发环境：

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

springCloudVersion = 'Edgware.SR2'

Github地址 https://github.com/LittleFlowerPig/lfp-jec.git

3、实际操作

3.1、在Ribbon使用断路器

在jec-balance-user的gradle中添加断路器Hystrix的起步依赖。

//断路器 - 故障返回  
compile('org.springframework.cloud:spring-cloud-starter-hystrix')

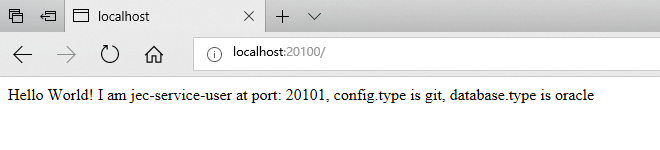
在Application类上加一个注解@EnableHystrix，开启断路器功能。

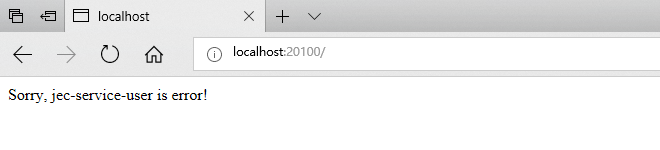
package com.lfp.jec.balance.user;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.loadbalancer.LoadBalanced;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.cloud.netflix.hystrix.EnableHystrix;  
import org.springframework.context.annotation.Bean;  
import org.springframework.web.client.RestTemplate;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
//断路器 - 故障返回  
@EnableHystrix  
public class JecBalanceUserApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecBalanceUserApplication.class, args);  
 }  
  
 @Bean  
 @LoadBalanced  
 RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
  
}

修改IndexApi，在原方法上添加注解@HystrixCommand(fallbackMethod = "indexError")，表明异常时访问indexError方法，并提供该方法。

package com.lfp.jec.balance.user.business;  
  
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import org.springframework.web.client.RestTemplate;  
  
@RestController  
public class IndexApi {  
  
 @Autowired  
 RestTemplate restTemplate;  
  
 @RequestMapping("/")  
 @HystrixCommand(fallbackMethod = "indexError")  
 public String index() {  
 return restTemplate.getForObject("http://jec-service-user/", String.class);  
 }  
  
 public String indexError() {  
 return "Sorry, jec-service-user is error!";  
 }  
  
}

依次运行jec-center、jec-config-git、jec-service-user、jec- balance -user。访问负载均衡后的地址，结果可以访问jec-service-user的内容。此时断开jec-service-user，返回为自定义的异常返回。





3.2、在Feign使用断路器

因为Feign整合了Hystrix，无需再添加gradle依赖。不过需要在配置文件中开启。

**server:  
 port:** 29900  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-balance-res  
  
**feign:  
 hystrix:  
 enabled:** true

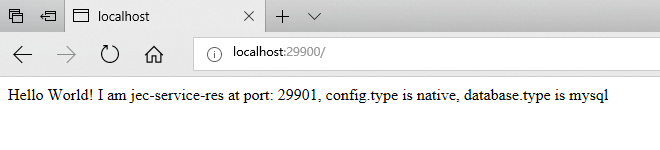
在接口的注解中添加@FeignClient(name = "jec-service-res", fallback = IndexError.class)，声明fallback指向的类。

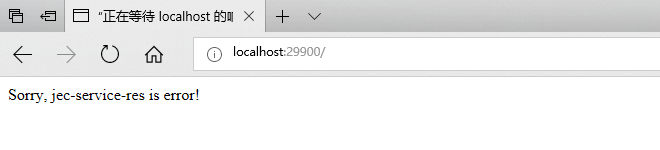
package com.lfp.jec.balance.res.business;  
  
import org.springframework.cloud.netflix.feign.FeignClient;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@FeignClient(name = "jec-service-res", fallback = IndexError.class)  
public interface Index {  
  
 @RequestMapping("/")  
 String index();  
  
}

添加该熔断使用类IndexError

package com.lfp.jec.balance.res.business;  
  
import org.springframework.stereotype.Component;  
@Component  
public class IndexError implements Index {  
  
 @Override  
 public String index() {  
 return "Sorry, jec-service-res is error!";  
 }  
  
}

依次运行jec-center、jec-config-nati、jec-service-res、jec-balance-res。访问负载均衡后的地址，结果可以访问jec-service-res的内容。此时断开jec-service-res，返回为自定义的异常返回。





3.3、断路器监控看板

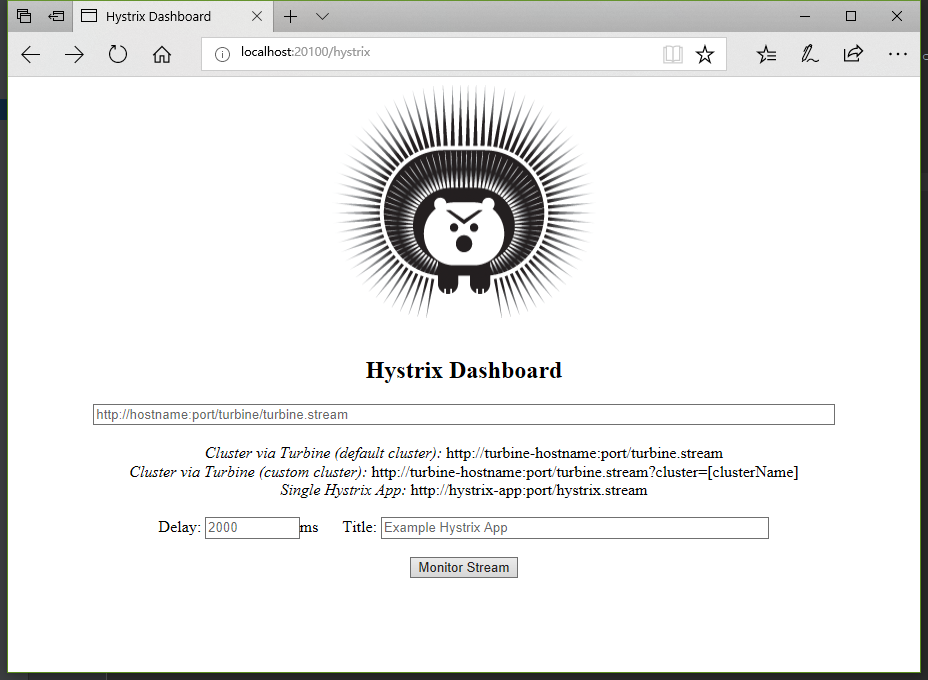
在jec-balance-user的gradle中添加断路器看板hystrix-dashboard的起步依赖。

//断路器 - 故障返回  
compile('org.springframework.cloud:spring-cloud-starter-hystrix')  
//断路器 – 监控看板  
compile('org.springframework.cloud:spring-cloud-starter-hystrix-dashboard')  
//健康监控  
compile('org.springframework.boot:spring-boot-starter-actuator')

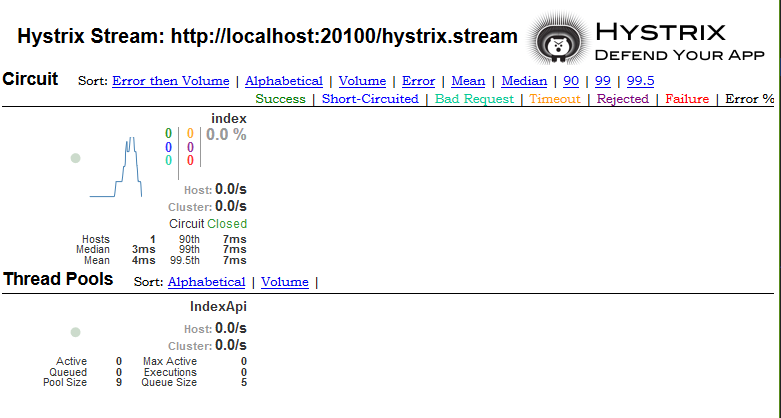
在Application类上加一个注解@EnableHystrixDashboard，开启断路器监控看板功能。

package com.lfp.jec.balance.user;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.loadbalancer.LoadBalanced;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import org.springframework.cloud.netflix.hystrix.EnableHystrix;  
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;  
import org.springframework.context.annotation.Bean;  
import org.springframework.web.client.RestTemplate;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
//断路器 - 服务故障  
@EnableHystrix  
//断路器 - 监控看板  
@EnableHystrixDashboard  
public class JecBalanceUserApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecBalanceUserApplication.class, args);  
 }  
  
 @Bean  
 @LoadBalanced  
 RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
  
}

此时启动服务后，访问负载均衡的应用，<http://localhost:20100/hystrix>，可以看到Hystrix Dashboard的主界面。

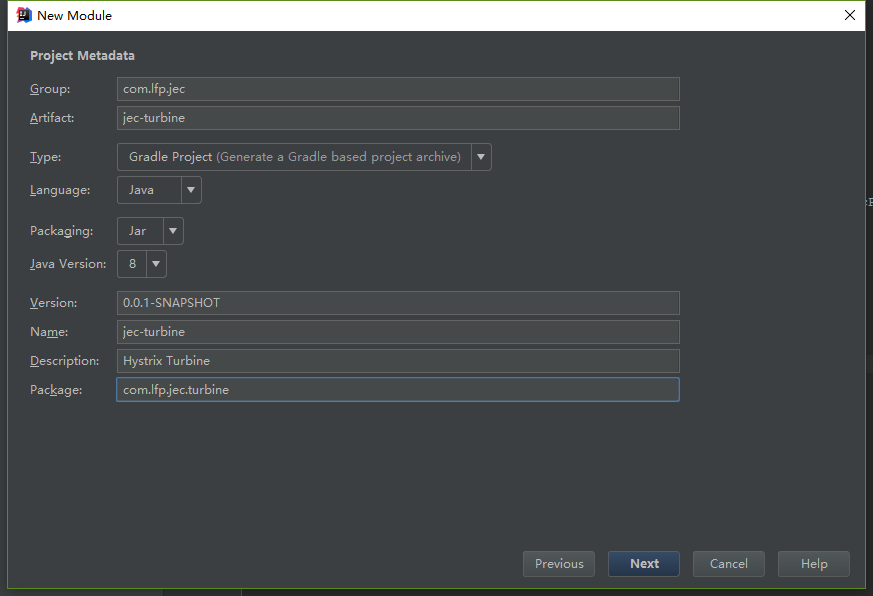


按照提示输入监控内容<http://localhost:20100/hystrix.stream>，可以看到监控的详细情况，此时访问接口应用，监控看板均有数据变化。

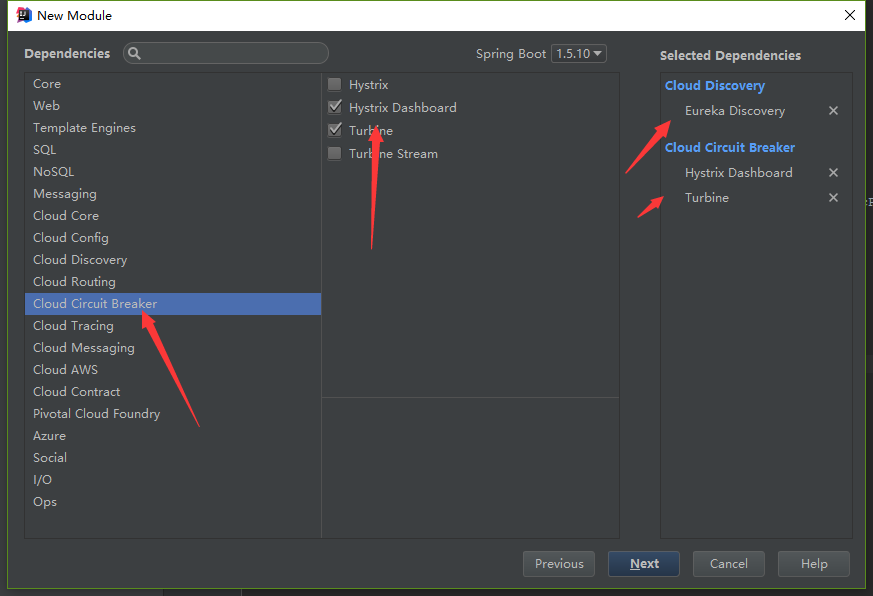


3.4、断路器聚合监控

添加新的模块工程。一样选择Spring initializr，输入断路聚合监控的必要信息。



选择Eureka Discovery和Hystrix Dashboard、Turbine同时作为该工程的起步依赖。



完成后IDE生成的gradle文件。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 //断路器 – 监控看板  
 compile('org.springframework.cloud:spring-cloud-starter-hystrix-dashboard')  
 //断路器 - 聚合监控  
 compile('org.springframework.cloud:spring-cloud-starter-turbine')  
  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

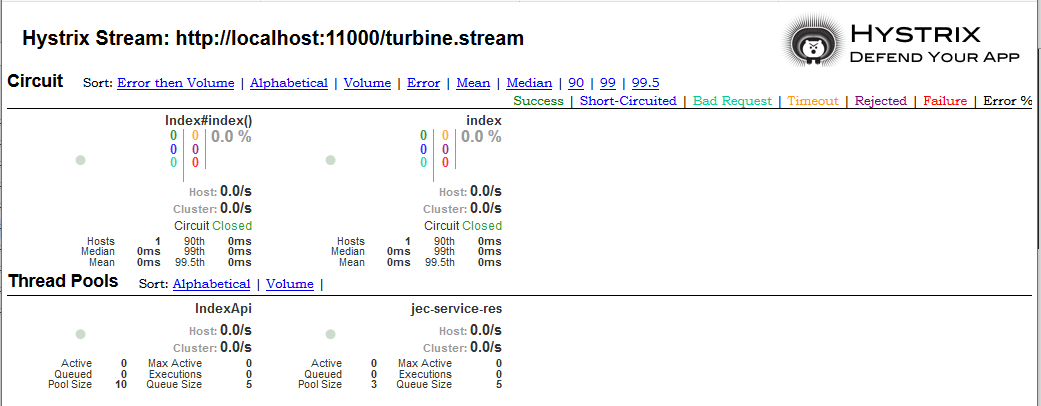
同样，在resources中添加配置文件application.yml。在配置文件中配置turbine的相关信息。

**server:  
 port:** 11000  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-turbine  
  
**turbine:  
 aggregator:** *# 指定聚合哪些集群，多个使用","分割，默认为default。可使用http://.../turbine.stream?cluster={clusterConfig之一}访问* **clusterConfig:** default  
 *### 配置Eureka中的serviceId列表，表明监控哪些服务* **appConfig:** jec-balance-user,jec-balance-res  
 *# 1. clusterNameExpression指定集群名称，默认表达式appName；此时：turbine.aggregator.clusterConfig需要配置想要监控的应用名称  
 # 2. 当clusterNameExpression: default时，turbine.aggregator.clusterConfig可以不写，因为默认就是default  
 # 3. 当clusterNameExpression: metadata['cluster']时，假设想要监控的应用配置了eureka.instance.metadata-map.cluster: ABC，则需要配置，同时turbine.aggregator.clusterConfig: ABC* **clusterNameExpression:** new String("default")

在其入口Application类加上注解@EnableTurbine，开启turbine，@EnableTurbine注解包含了@EnableDiscoveryClient注解，即开启了注册服务。添加@EnableHystrixDashboard开启看板。

package com.lfp.jec.turbine;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;  
import org.springframework.cloud.netflix.turbine.EnableTurbine;  
  
@SpringBootApplication  
//断路器 - 监控看板  
@EnableHystrixDashboard  
//断路器 - 聚合监控  
@EnableTurbine  
public class JecTurbineApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecTurbineApplication.class, args);  
 }  
}

依次运行jec-center、jec- balance -user、jec-balance-res、jec-turbine。访问jec-turbine监控看板地址的<http://localhost:11000/hystrix/>，输入聚合监控Hystrix Stream: <http://localhost:11000/turbine.stream>。



基于SpringCloud的企业级微服务架构入门（七）：微服务的链路追踪

1、简要概述

微服务架构上通过业务来划分服务的，通过REST调用，对外暴露的一个接口，可能需要很多个服务协同才能完成这个接口功能，如果链路上任何一个服务出现问题或者网络超时，都会形成导致接口调用失败。随着业务的不断扩张，服务之间互相调用会越来越复杂。

Spring Cloud Sleuth 主要功能就是在分布式系统中提供追踪解决方案，并且兼容支持了 zipkin。

Zipkin也支持将数据持久化到内存中。同时我们也可以将http通信改为mq等异步方式通信。

2、开发环境：

IntelliJ IDEA 2017.3.4 (Ultimate Edition)

JDK 1.8.0\_152

Gradle 4.3.1

springBootVersion = '1.5.10.RELEASE'

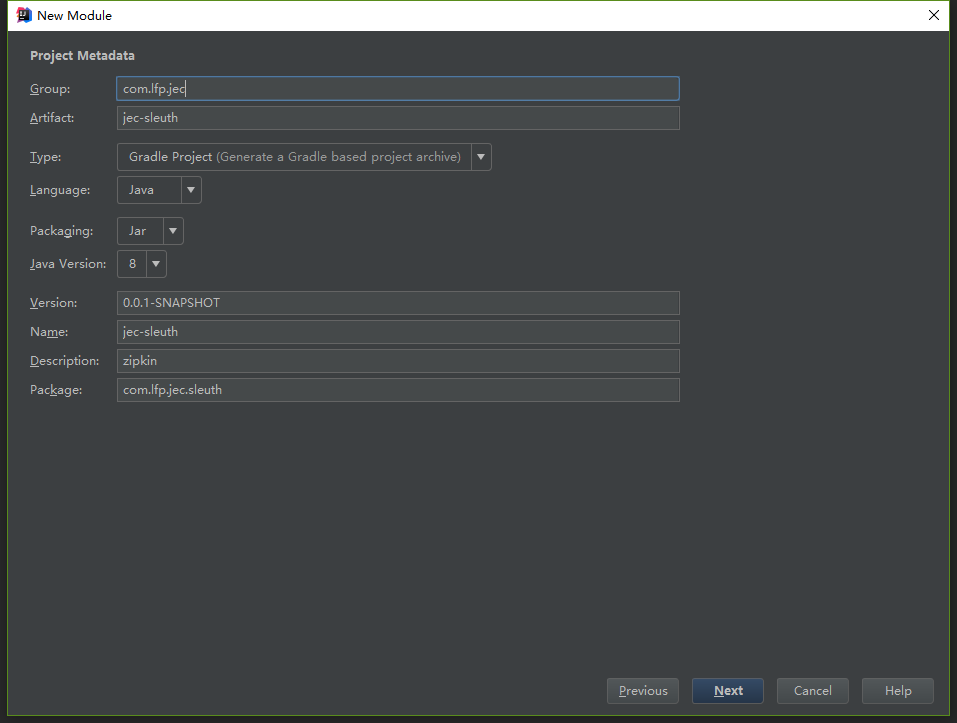
springCloudVersion = 'Edgware.SR2'

Github地址 https://github.com/LittleFlowerPig/lfp-jec.git

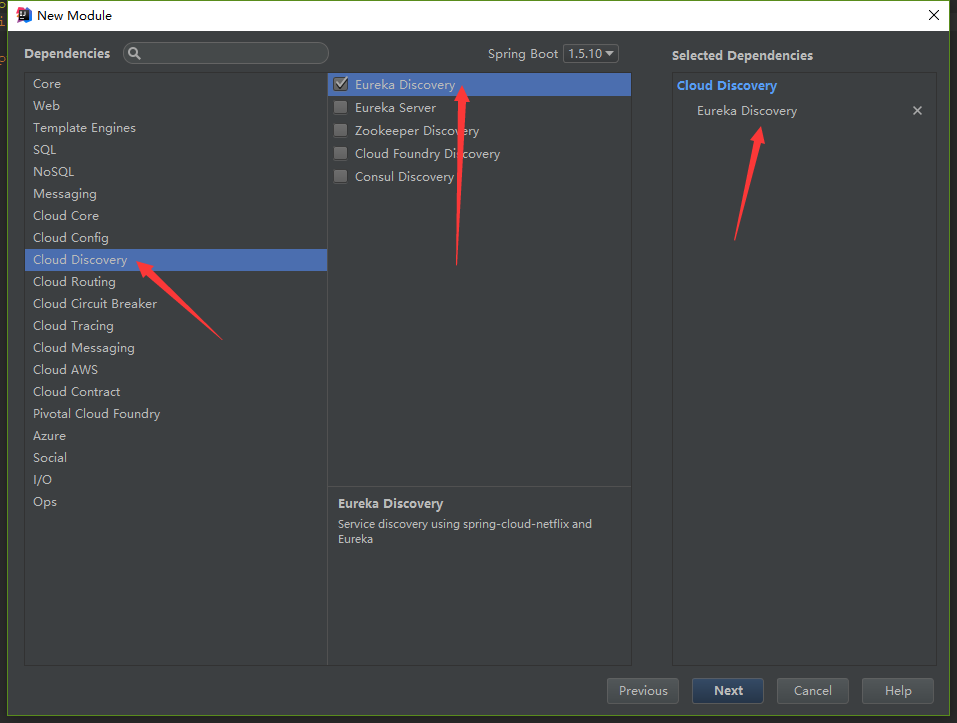
3、实际操作

3.1、创建链路追踪监控服务端

继续上一篇的工程，添加新的模块工程。一样选择Spring initializr，输入链路追踪的必要信息。



我们先只选Eureka Discovery起步，完成后再添加我们需要的zipkin服务端的依赖。



IDE生成gradle文件后，引入zipkin-server依赖，以及zipkin-autoconfigure-ui依赖，这两个依赖提供了Zipkin的功能和Zipkin界面展示的功能。

buildscript {  
 ext {  
 springBootVersion = '1.5.10.RELEASE'  
 }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")  
 }  
}  
  
apply plugin: 'java'  
apply plugin: 'eclipse'  
apply plugin: 'org.springframework.boot'  
  
group = 'com.lfp.jec'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
 mavenCentral()  
}  
  
  
ext {  
 springCloudVersion = 'Edgware.SR2'  
}  
  
dependencies {  
 //微服务 - 注册服务  
 compile('org.springframework.cloud:spring-cloud-starter-eureka')  
 //链路追踪 - 服务端  
 compile('io.zipkin.java:zipkin-server')  
 //链路追踪 - UI展现  
 compile('io.zipkin.java:zipkin-autoconfigure-ui')  
 testCompile('org.springframework.boot:spring-boot-starter-test')  
}  
  
dependencyManagement {  
 imports {  
 mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"  
 }  
}

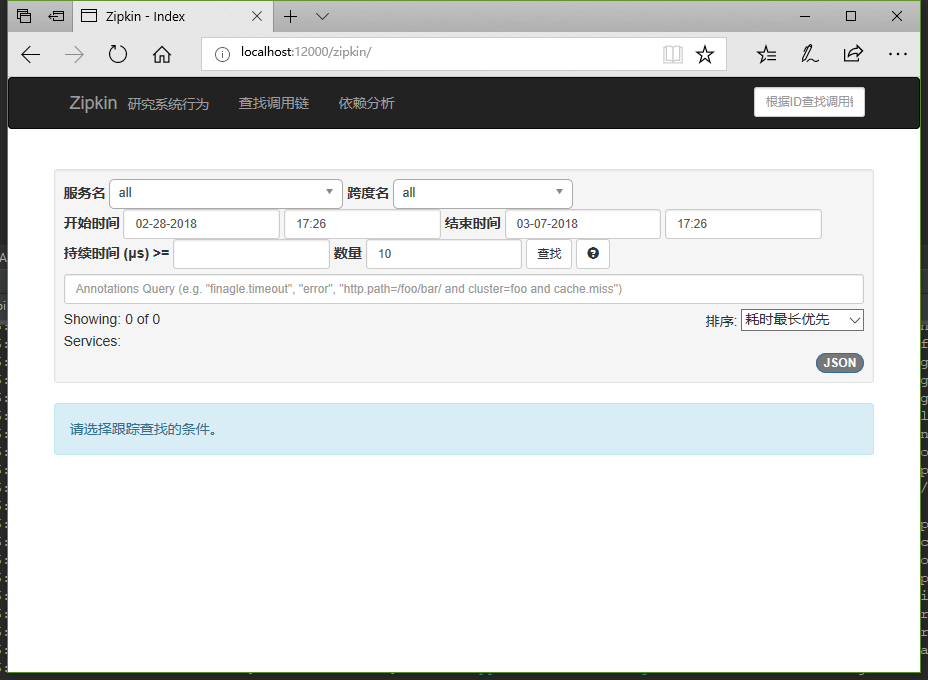
配置文件中添加端口等信息。

**server:  
 port:** 12000  
  
**spring:  
 application:  
 name:** jec-sleuth  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/

需要在springboot工程的启动application类上额外添加一个注解@EnableZipkinServer，声明其为Zipkin链路追踪的服务端。

package com.lfp.jec.sleuth;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
import zipkin.server.EnableZipkinServer;  
  
@SpringBootApplication  
//微服务 - 注册服务  
@EnableEurekaClient  
//链路追踪 - 服务端  
@EnableZipkinServer  
public class JecSleuthApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JecSleuthApplication.class, args);  
 }  
}

jec-sleuth作为服务端只依赖于注册中心，正常启动后，访问地址<http://localhost:12000/>，即可看到相应的界面。



3.2、注册服务接入链路追踪

注册服务接入链路追踪非常方便，需要引入spring-cloud-starter-zipkin依赖。

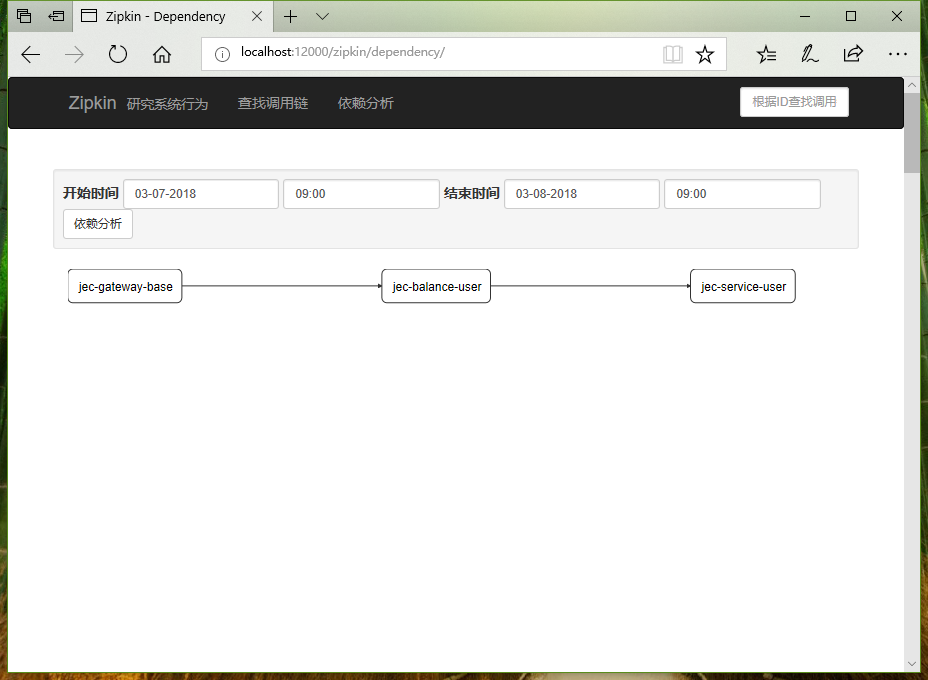
//链路追踪 - 客户端  
compile('org.springframework.cloud:spring-cloud-starter-zipkin')

在配置文件中添加追踪服务端的地址spring.zipkin.base-url就可以了，也可以指定链路的数据上传给Zipkin Server的概率，默认为0.1。

**spring:**  
 **zipkin:  
 base-url:** http://localhost:12000  
 **sleuth:  
 sampler:  
 percentage:** 1.0

按照上述方法分布改造jec-service-user、jec-balance-user、jec-gateway-base。

依次运行jec-center、jec-sleuth、jec-config-git、jec-service-user、jec-balance-user、jec-gateway-base。访问网关地址应用<http://localhost:30101/user>。再访问jec-sleuth依赖监控地址<http://localhost:12000/zipkin/dependency/>，可以看到访问地址间的相互依赖关系。



3.3、持久化链路追踪信息

jec-sleuth的gradle中添加额外的依赖

//链路追踪 - 持久化到数据库  
compile('io.zipkin.java:zipkin-autoconfigure-storage-mysql')  
compile('mysql:mysql-connector-java')  
compile('org.springframework.boot:spring-boot-starter-jdbc')

在配置文件中添加数据源的配置，并指定zipkin使用mysql类型存储数据。

**server:  
 port:** 12000  
  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:10000/eureka/  
  
**spring:  
 application:  
 name:** jec-sleuth  
 **datasource:  
 driver-class-name:** com.mysql.jdbc.Driver  
 **url:** jdbc:mysql://172.20.19.112:3306/jec\_dev?serverTimezone=Hongkong&useUnicode=true&characterEncoding=UTF-8&useSSL=true  
 **username:** root  
 **password:** 123456  
  
**zipkin:  
 storage:  
 type:** mysql

打开依赖包中的io.zipkin.java:zipkin-storage-mysql，其中有mysql.sql初始化语句，执行该语句在数据库中创建相应的表结构。

**CREATE TABLE** IF **NOT EXISTS** zipkin\_spans (  
 `trace\_id\_high` BIGINT **NOT NULL DEFAULT** 0 COMMENT 'If non zero, this means the trace uses 128 bit traceIds instead of 64 bit',  
 `trace\_id` BIGINT **NOT NULL**,  
 `id` BIGINT **NOT NULL**,  
 `name` **VARCHAR**(255) **NOT NULL**,  
 `parent\_id` BIGINT,  
 `debug` **BIT**(1),  
 `start\_ts` BIGINT COMMENT 'Span.timestamp(): epoch micros used for endTs query and to implement TTL',  
 `duration` BIGINT COMMENT 'Span.duration(): micros used for minDuration and maxDuration query'  
) ENGINE=InnoDB ROW\_FORMAT=COMPRESSED **CHARACTER SET**=utf8 **COLLATE** utf8\_general\_ci;  
  
**ALTER TABLE** zipkin\_spans **ADD UNIQUE KEY**(`trace\_id\_high`, `trace\_id`, `id`) COMMENT 'ignore insert on duplicate';  
**ALTER TABLE** zipkin\_spans **ADD** INDEX(`trace\_id\_high`, `trace\_id`, `id`) COMMENT 'for joining with zipkin\_annotations';  
**ALTER TABLE** zipkin\_spans **ADD** INDEX(`trace\_id\_high`, `trace\_id`) COMMENT 'for getTracesByIds';  
**ALTER TABLE** zipkin\_spans **ADD** INDEX(`name`) COMMENT 'for getTraces and getSpanNames';  
**ALTER TABLE** zipkin\_spans **ADD** INDEX(`start\_ts`) COMMENT 'for getTraces ordering and range';  
  
**CREATE TABLE** IF **NOT EXISTS** zipkin\_annotations (  
 `trace\_id\_high` BIGINT **NOT NULL DEFAULT** 0 COMMENT 'If non zero, this means the trace uses 128 bit traceIds instead of 64 bit',  
 `trace\_id` BIGINT **NOT NULL** COMMENT 'coincides with zipkin\_spans.trace\_id',  
 `span\_id` BIGINT **NOT NULL** COMMENT 'coincides with zipkin\_spans.id',  
 `a\_key` **VARCHAR**(255) **NOT NULL** COMMENT 'BinaryAnnotation.key or Annotation.value if type == -1',  
 `a\_value` BLOB COMMENT 'BinaryAnnotation.value(), which must be smaller than 64KB',  
 `a\_type` **INT NOT NULL** COMMENT 'BinaryAnnotation.type() or -1 if Annotation',  
 `a\_timestamp` BIGINT COMMENT 'Used to implement TTL; Annotation.timestamp or zipkin\_spans.timestamp',  
 `endpoint\_ipv4` **INT** COMMENT 'Null when Binary/Annotation.endpoint is null',  
 `endpoint\_ipv6` BINARY(16) COMMENT 'Null when Binary/Annotation.endpoint is null, or no IPv6 address',  
 `endpoint\_port` **SMALLINT** COMMENT 'Null when Binary/Annotation.endpoint is null',  
 `endpoint\_service\_name` **VARCHAR**(255) COMMENT 'Null when Binary/Annotation.endpoint is null'  
) ENGINE=InnoDB ROW\_FORMAT=COMPRESSED **CHARACTER SET**=utf8 **COLLATE** utf8\_general\_ci;  
  
**ALTER TABLE** zipkin\_annotations **ADD UNIQUE KEY**(`trace\_id\_high`, `trace\_id`, `span\_id`, `a\_key`, `a\_timestamp`) COMMENT 'Ignore insert on duplicate';  
**ALTER TABLE** zipkin\_annotations **ADD** INDEX(`trace\_id\_high`, `trace\_id`, `span\_id`) COMMENT 'for joining with zipkin\_spans';  
**ALTER TABLE** zipkin\_annotations **ADD** INDEX(`trace\_id\_high`, `trace\_id`) COMMENT 'for getTraces/ByIds';  
**ALTER TABLE** zipkin\_annotations **ADD** INDEX(`endpoint\_service\_name`) COMMENT 'for getTraces and getServiceNames';  
**ALTER TABLE** zipkin\_annotations **ADD** INDEX(`a\_type`) COMMENT 'for getTraces';  
**ALTER TABLE** zipkin\_annotations **ADD** INDEX(`a\_key`) COMMENT 'for getTraces';  
**ALTER TABLE** zipkin\_annotations **ADD** INDEX(`trace\_id`, `span\_id`, `a\_key`) COMMENT 'for dependencies job';  
  
**CREATE TABLE** IF **NOT EXISTS** zipkin\_dependencies (  
 `day` **DATE NOT NULL**,  
 `parent` **VARCHAR**(255) **NOT NULL**,  
 `child` **VARCHAR**(255) **NOT NULL**,  
 `call\_count` BIGINT,  
 `error\_count` BIGINT  
) ENGINE=InnoDB ROW\_FORMAT=COMPRESSED **CHARACTER SET**=utf8 **COLLATE** utf8\_general\_ci;  
  
**ALTER TABLE** zipkin\_dependencies **ADD UNIQUE KEY**(`day`, `parent`, `child`);

启动应用并访问地址，查看数据库，发现有数据写入了数据库，此时重启zipkin服务端，数据也不会再丢失了。

基于SpringCloud的企业级微服务架构入门（八）：企业应用帮助类库

基于SpringCloud的企业级微服务架构入门（九）：企业应用基础框架

基于SpringCloud的企业级微服务架构入门（十）：企业应用的微服务