# SpringCloud

## 简介

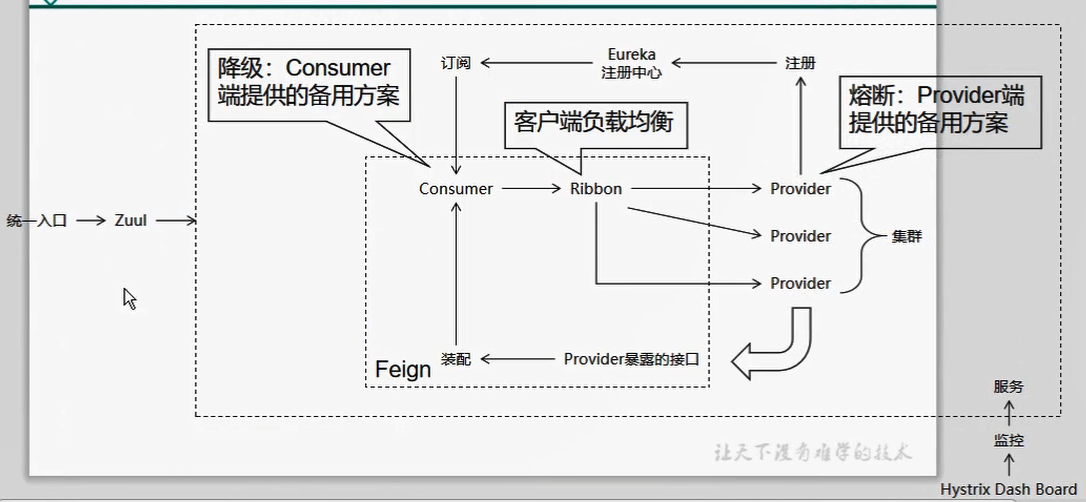
注册中心：Eureka 

负载均衡：Ribbon 

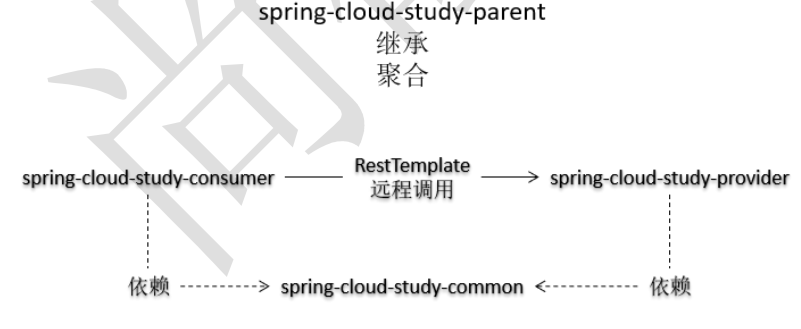
声明式调用远程方法：Feign 

熔断、降级、监控：Hystrix 

网关：Zuul

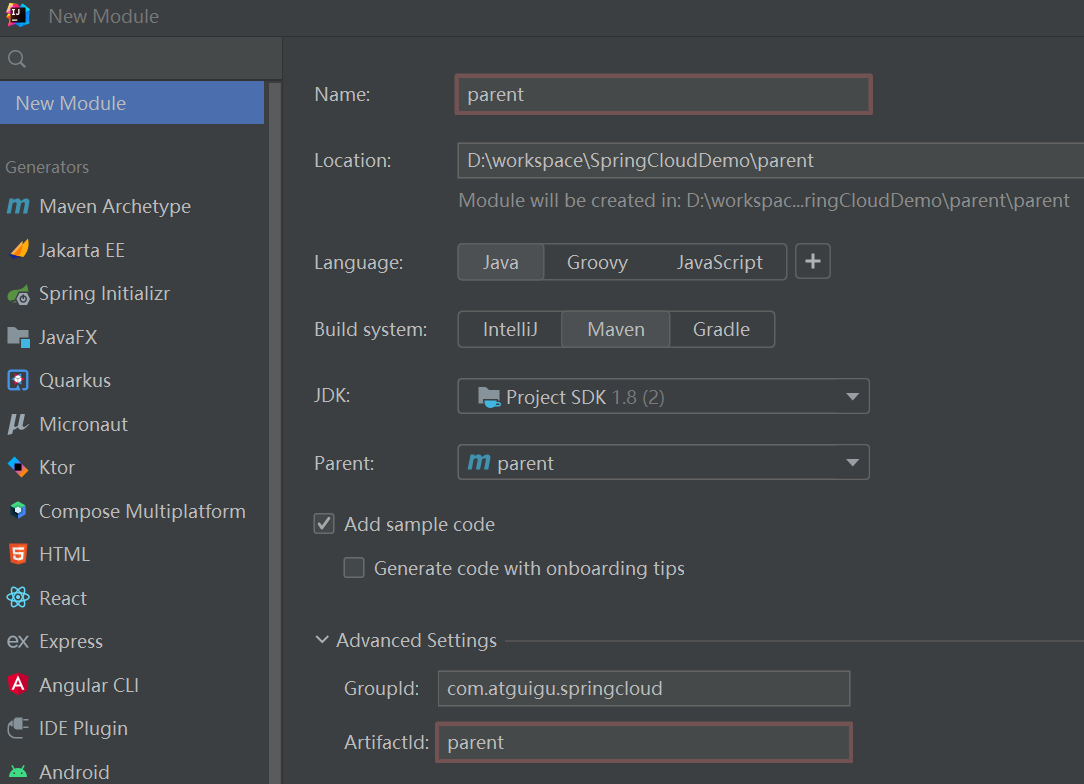


## 基础工程



### 父工程

父工程，用于管理依赖



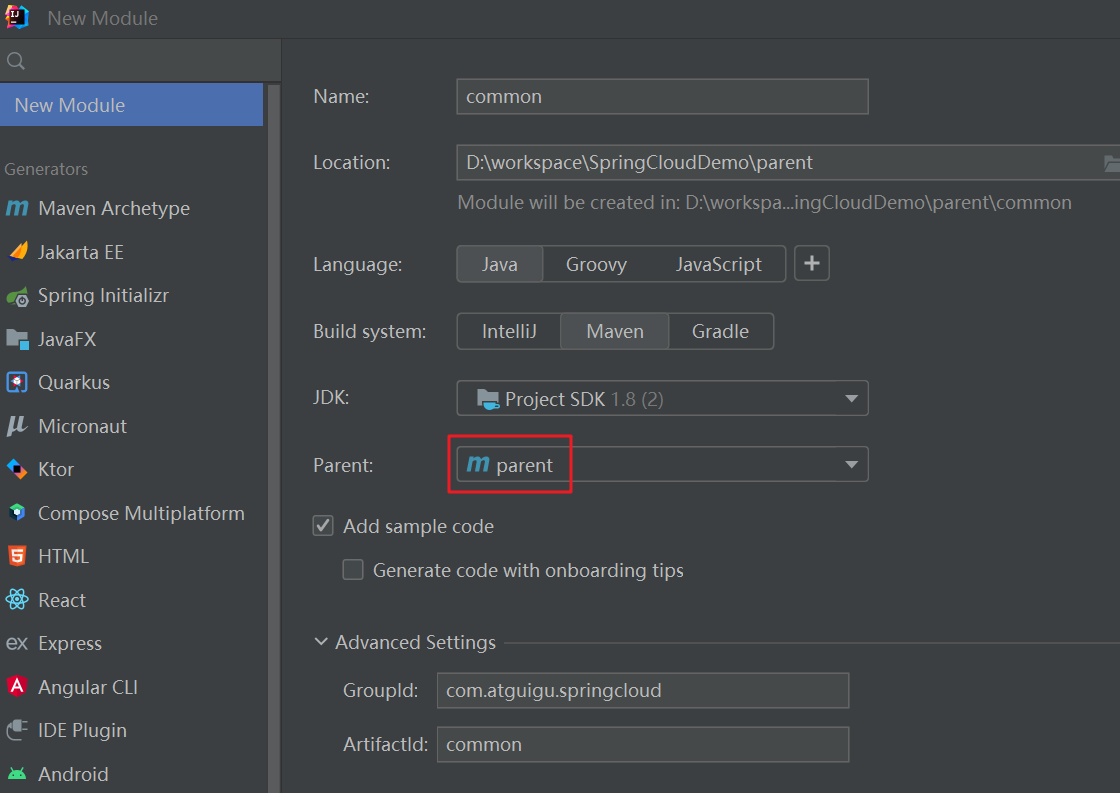
依赖导入：

<dependencyManagement>  
<dependencies>  
 <!-- 导入 SpringCloud 需要使用的依赖信息 -->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Greenwich.SR2</version>  
 <type>pom</type>  
 <!-- import 依赖范围表示将 spring-cloud-dependencies 包中的依赖信息导入--><scope>import</scope>  
 </dependency>  
 <!-- 导入 SpringBoot 需要使用的依赖信息 -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-dependencies</artifactId>  
 <version>2.1.6.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
</dependencies>  
</dependencyManagement>

### common工程

common工程依赖父工程

打jar包



只有一个实体类：

public class Employee {  
 private Integer empId;  
 private String empName;  
 private Double empSalary;

### provider工程

springboot工程，jar包，依赖父工程和common工程

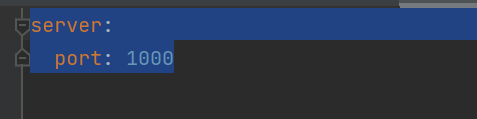
1.引入依赖：

<dependencies>  
 <dependency>  
 <!-- 指定groupId和artifactId即可，版本已在父工程中定义 -->  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.atguigu.springcloud</groupId>  
 <artifactId>common</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
</dependencies>

2.主启动类：

@SpringBootApplication  
public class ProviderMain {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ProviderMain.class);  
 System.*out*.println("Hello world!");  
 }  
}

3.application.yml



4.控制器方法：

@RestController  
public class ProviderHandler {

@RequestMapping("/provider/getReomteEmploee")  
 public Employee getRemoteEmploee(){  
 return new Employee(10001,"shuyun",1000.05);  
 }  
}

### consumer工程

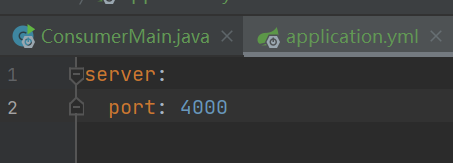
1.引入依赖

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.atguigu.springcloud</groupId>  
 <artifactId>common</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
</dependencies>

2.主启动类

@SpringBootApplication  
public class ConsumerMain {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ConsumerMain.class);  
 System.*out*.println("Hello world!");  
 }  
}

3.application.yml



4.创建配置类，提供RestTemplate，用于远程调用

@Configuration  
public class ConsumerConfig {  
  
 @Bean  
 public RestTemplate getRestTemplate(){  
 return new RestTemplate();  
 }  
}

5.控制器方法：

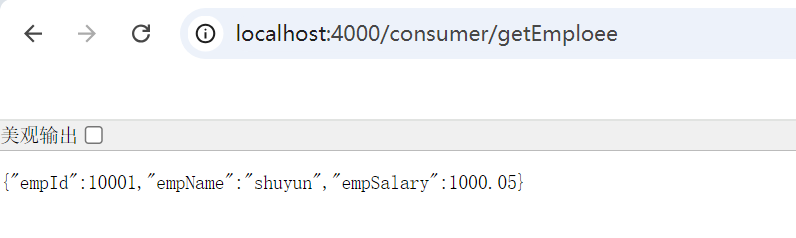
@RestController  
public class HumanResourceHandler {  
 @Autowired  
 private RestTemplate restTemplate;  
  
 @RequestMapping("/consumer/getEmploee")  
 public Employee getEmploeeRemote(){  
  
 //远程调用方法的主机地址  
 String host = "http://localhost:1000";  
  
 //远程调用方法的具体url地址  
 String url = "/provider/getReomteEmploee";  
  
 //通过 restTemplate 调用远程方法  
 return restTemplate.getForObject(host+url, Employee.class);  
  
 }  
}

### 调用测试

1、启动provider工程

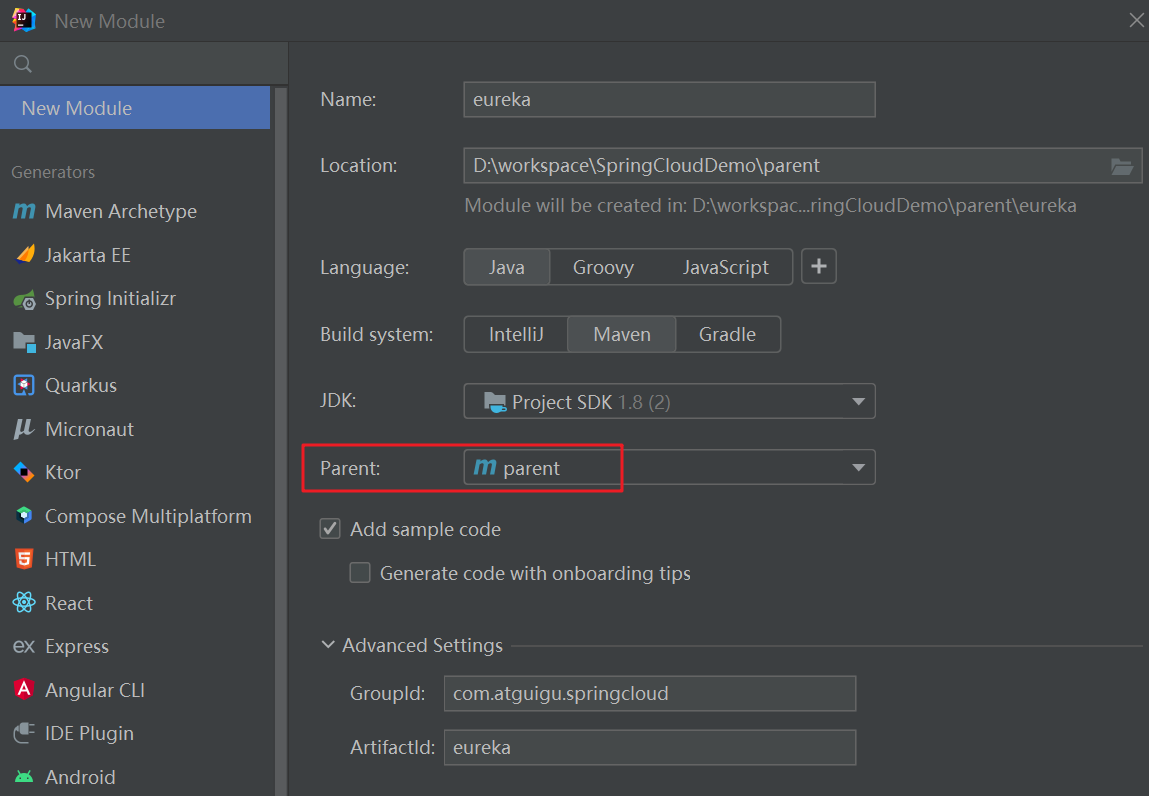
2、启动consumer工程

3、访问consumer工程的 getEmploeeRemote方法，返回的是provider工程的调用结果



## Eureka 注册中心

### 工程创建



### 依赖

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

### 启动类

@EnableEurekaServer 注解表示 启用 Eureka 服务器功能

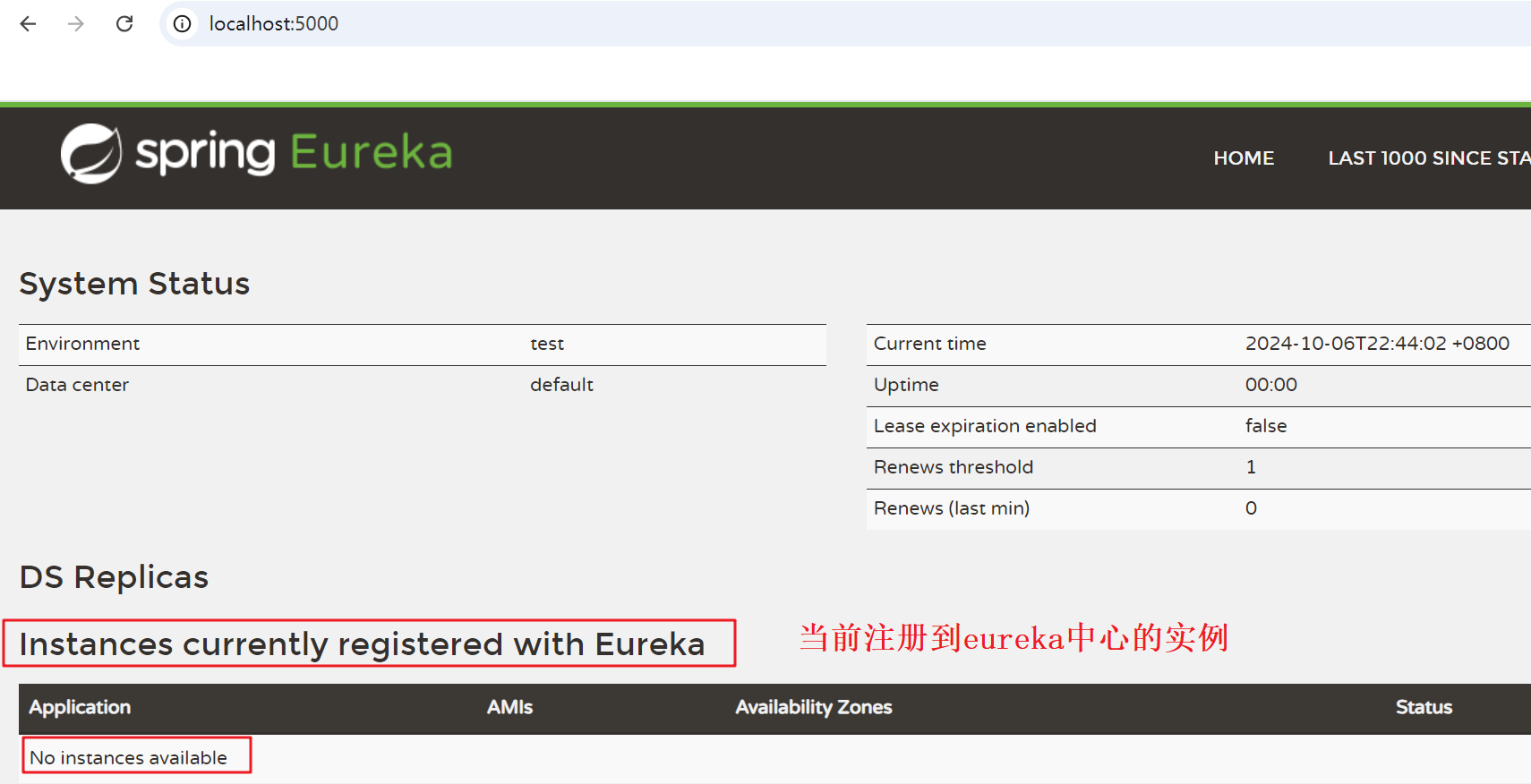
@SpringBootApplication  
@EnableEurekaServer  
public class EurekaMain {  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaMain.class);  
 System.*out*.println("Hello world!");  
 }  
}

### application.yml配置

server:  
 port: 5000  
eureka:  
 instance:  
 hostname: localhost *#配置当前注册中心的服务主机地址* client:  
 register-with-eureka: false *# 自己就是注册中心，所以自己不注册自己* fetch-registry: false *# 自己就是注册中心，所以不需要“从注册中心取回信息”* service-url: *#客户端（consumer和provider）访问 Eureka 时使用的地址* defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

### 启动

启动服务器后，访问 localhost:5000，看到如下界面，说明eureka注册中心创建成功



### 将provider注册到eureka

provider工程做如下修改

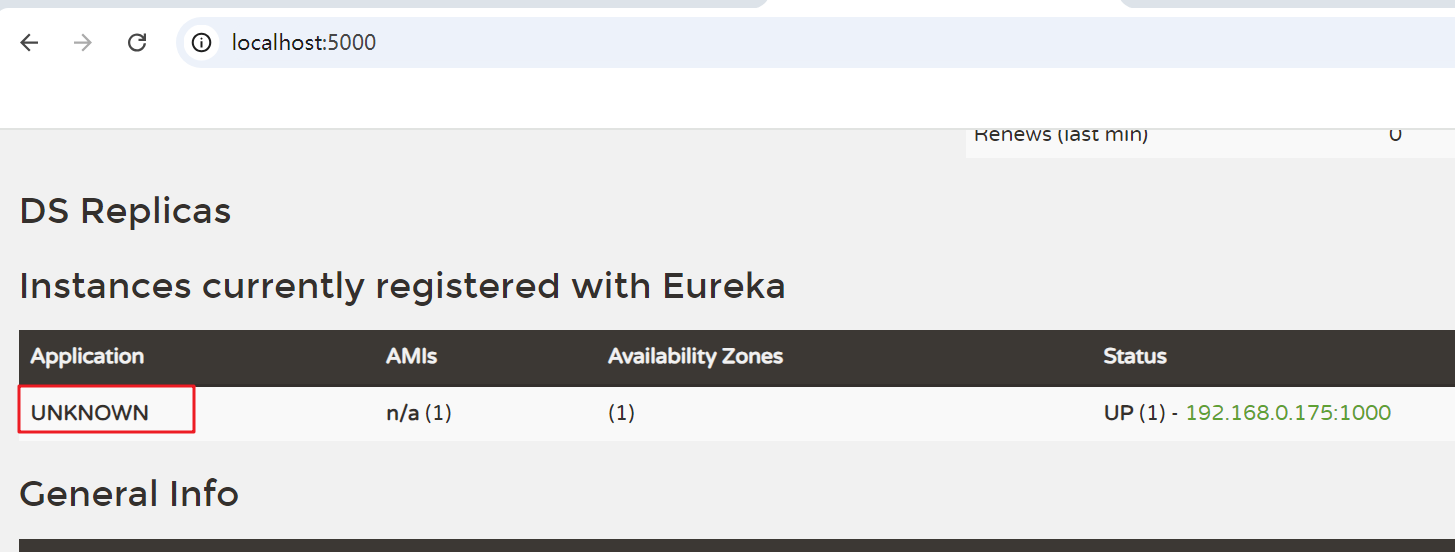
1.增加依赖

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

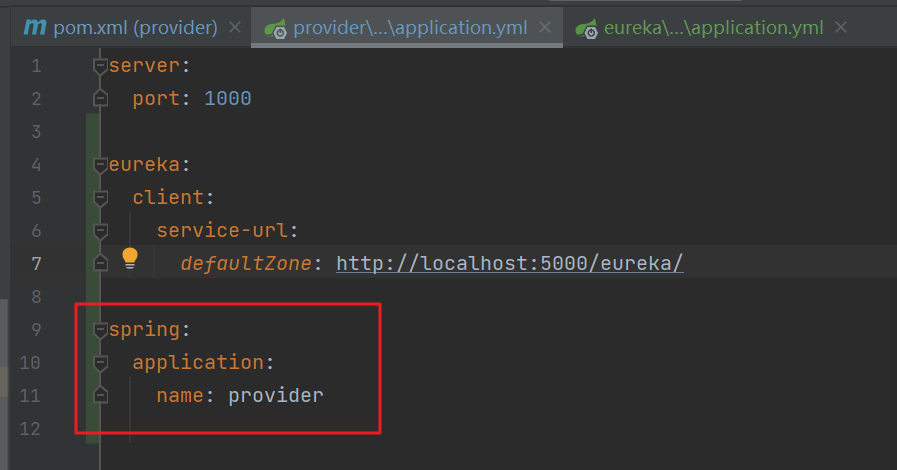
2.增加配置application.yml



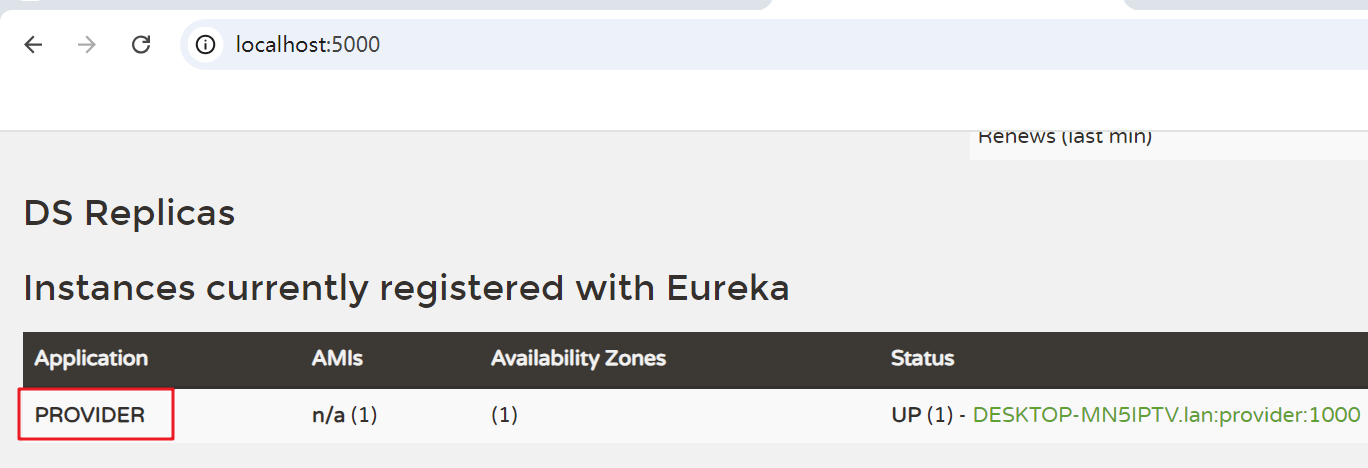
3.重新启动后，刷新eureka注册中心的地址，可以看到如下界面



显示unknown是因为provider 工程没有指定应用名称，指定应用名称配置方式如下：

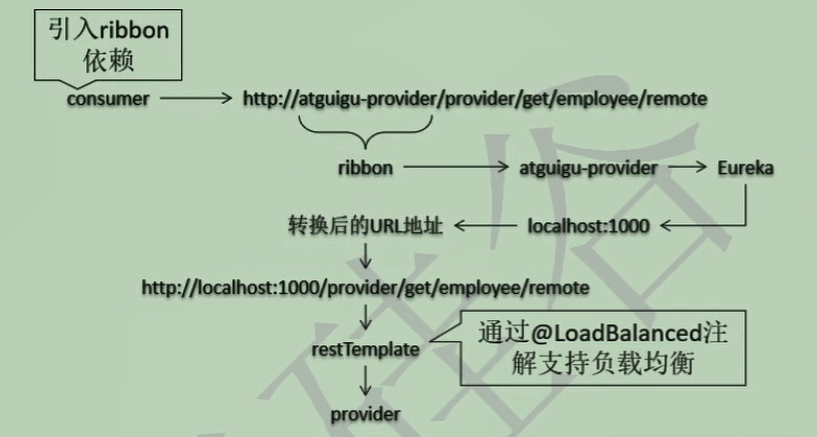


eureka注册中心显示如下：



以后在 SpringCloud 环境下开发，每一个微服务工程都要设置一个应用名称。

### consumer使用微服务名称调用provider



1.引入依赖

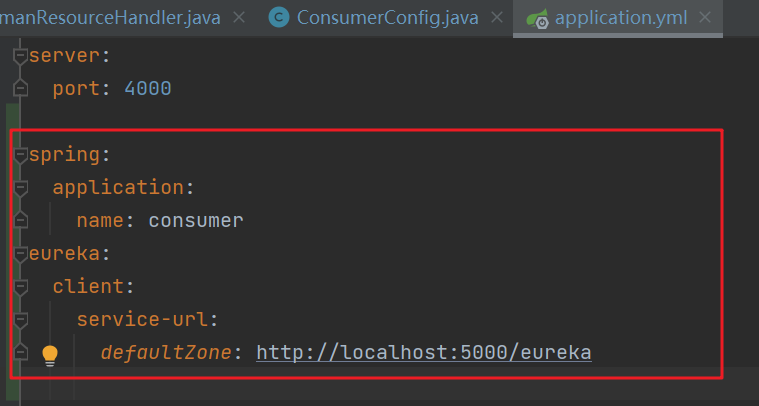
eureka-client是eureka客户端，用来注册到eureka

ribbon是将 微服务名转换成具体的ip和端口，负载均衡方式调用远程微服务

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

2.application.yml配置

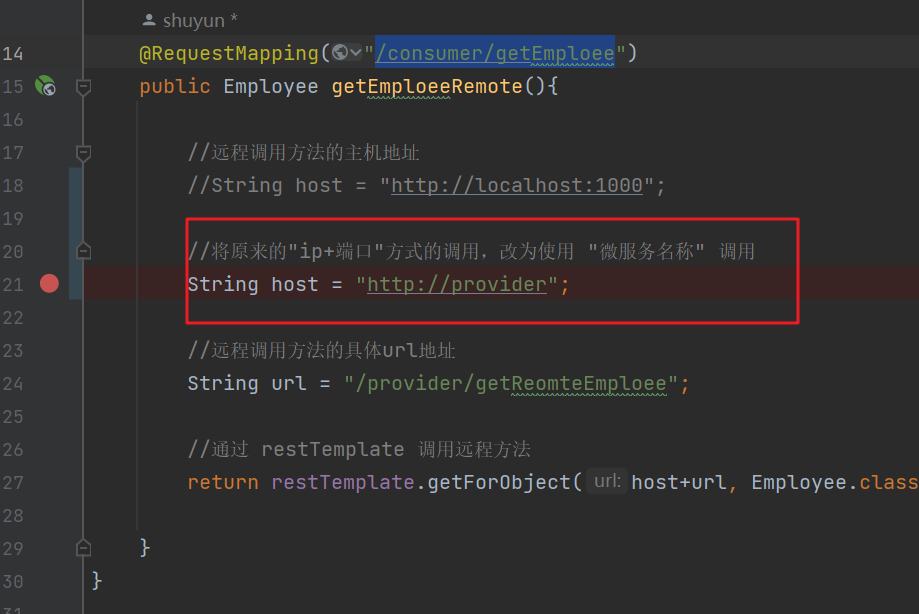
配置微服务名称和eureka服务器地址



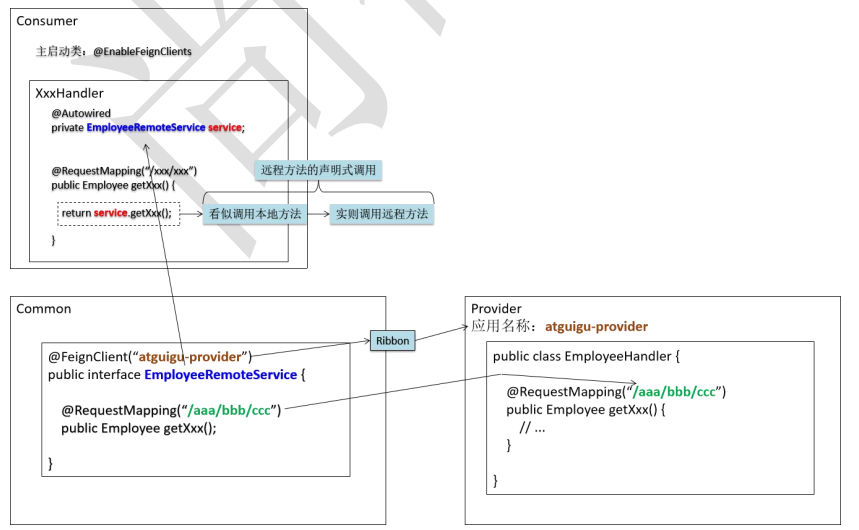
3.获取远程调用方法的Bean加上 @LoadBalance注解

@Configuration  
public class ConsumerConfig {  
  
 @Bean  
 @LoadBalanced  
 public RestTemplate getRestTemplate(){  
 return new RestTemplate();  
 }  
}

4.控制器方法远程调用



## Feign远程方法调用



1.common中声明接口和抽象方法

2.provider中声明与common中的接口相同的实现类，声明具体的方法（不是实现）

3.consumer中装配common中的接口，调用相应的方法。需要在启动类加上@EnableFeignClients注解

### common工程

1.引入依赖

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

2.创建远程调用方法的接口

注意，接口所在包要符合springboot扫描规则，要被扫描到IOC容器里面才能被consumer工程使用

注意：这里的方法要和 Provider中的具体实现的方法的声明要一致

@RequestMapping注解的url地址、@RequestParam、@RequestBody、@PathVariable两边一致

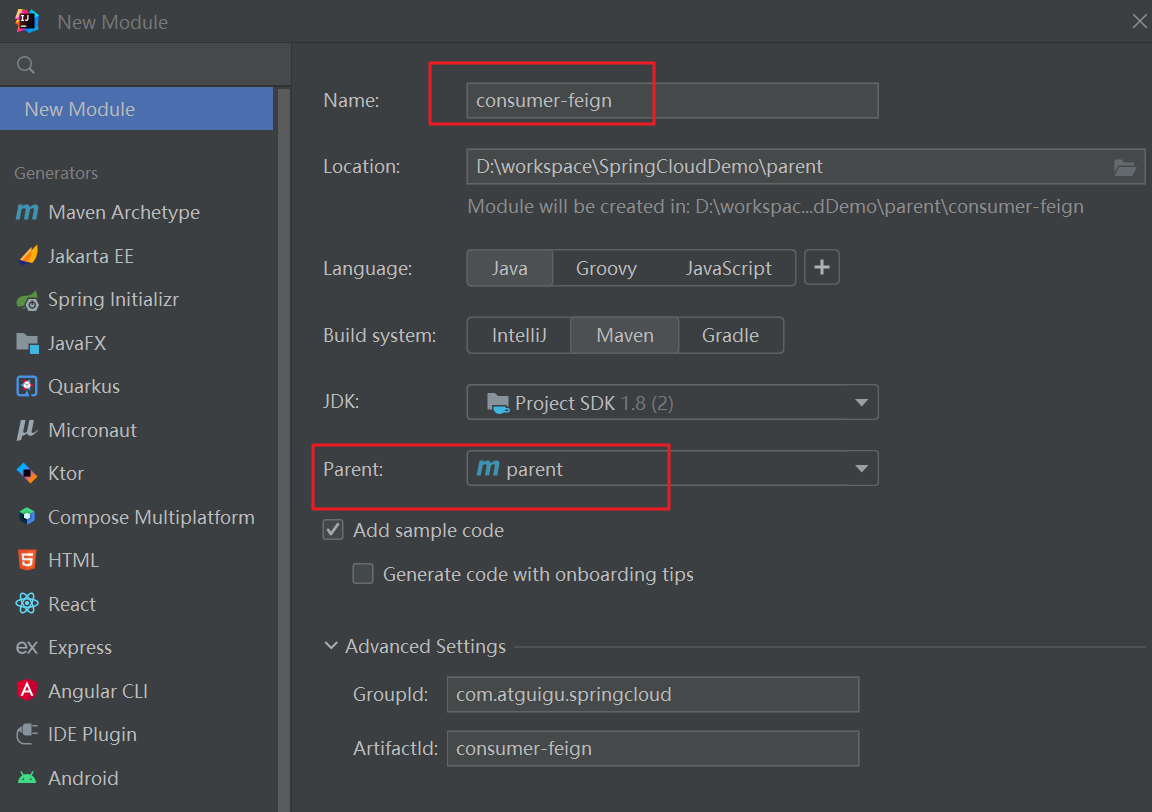
//@FeignClient注解表示和一个provider对应，value属性指定要调用的Provider的微服务名称  
@FeignClient(value = "provider")  
public interface EmployeeRemoteService {  
  
 //注意：这里的方法要和 Provider中的具体实现的方法的声明要一致：

//远程调用的接口方法  
 @RequestMapping("/provider/getReomteEmploee")  
 Employee getRemoteEmploee();  
}

### Feign-consumer工程

1.新建feign-consumer工程

不使用原来的consumer工程了，重新建一个consumer工程



2.引入依赖

ribbon注解在common中的feign已经引入

<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.atguigu.springcloud</groupId>  
 <artifactId>common</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
</dependencies>

3.主启动类

加上@EnableFeignClients注解

@SpringBootApplication  
//启用Feign客户端功能  
@EnableFeignClients  
public class ConsumerFeignMain {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ConsumerFeignMain.class);  
 System.*out*.println("Hello world!");  
 }  
}

4.application.yml配置

配置端口、微服务名称、eureka注册服务器地址

server:  
 port: 8000  
  
spring:  
 application:  
 name: feign-consumer  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:5000/eureka

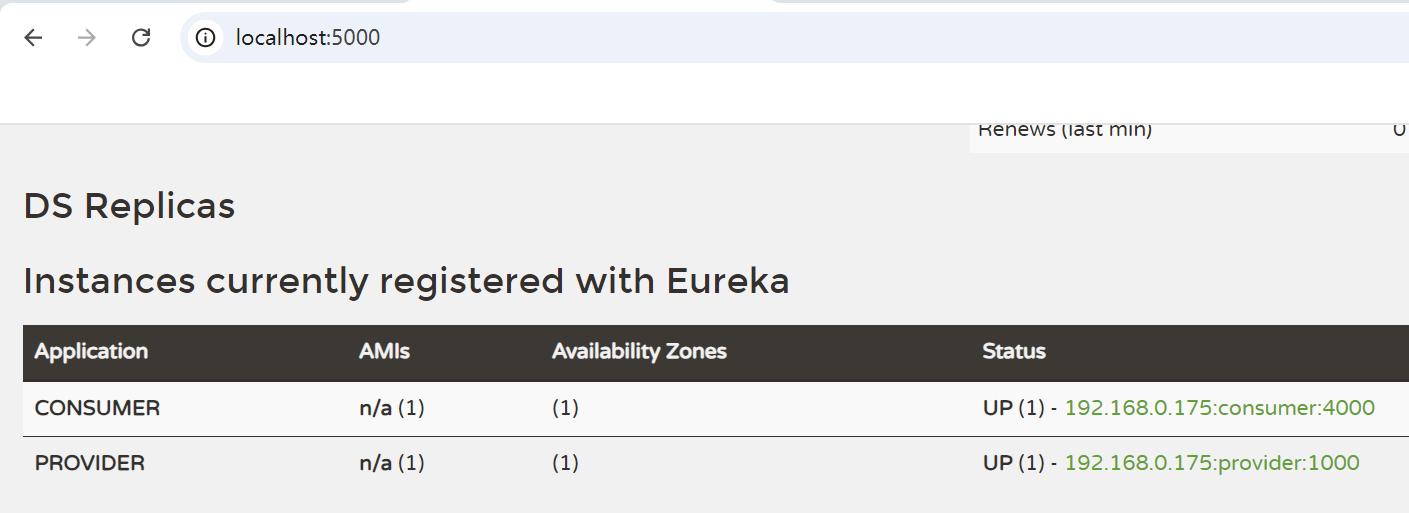
5.控制器方法

@RestController  
public class FeignConsumerHandler {  
  
 //装配远程调用微服务的接口，然后就可以像本地调用一样了  
 @Autowired  
 private EmployeeRemoteService employeeRemoteService;  
  
 @RequestMapping("/feign/getRemoteEmployee")  
 public String getRemoteEmployee(){  
 Employee remoteEmploee = employeeRemoteService.getRemoteEmploee();  
 return remoteEmploee.toString();  
 }  
}

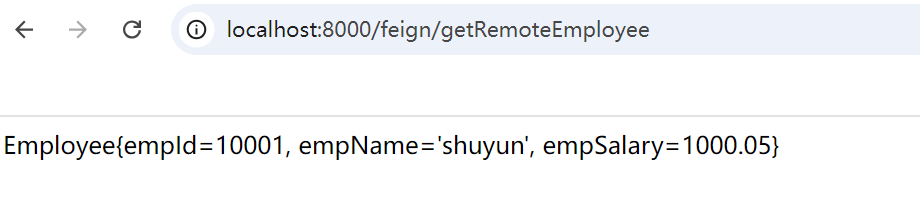
### 测试

先启动eureka，再启动provider和consumer

服务已经注册上



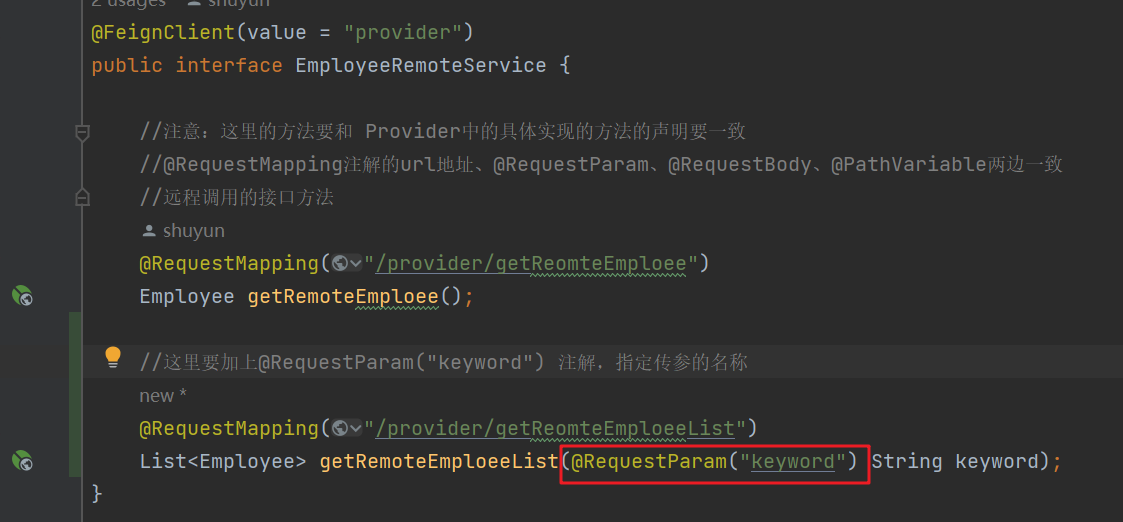
调用consumer的接口



### 简单传参

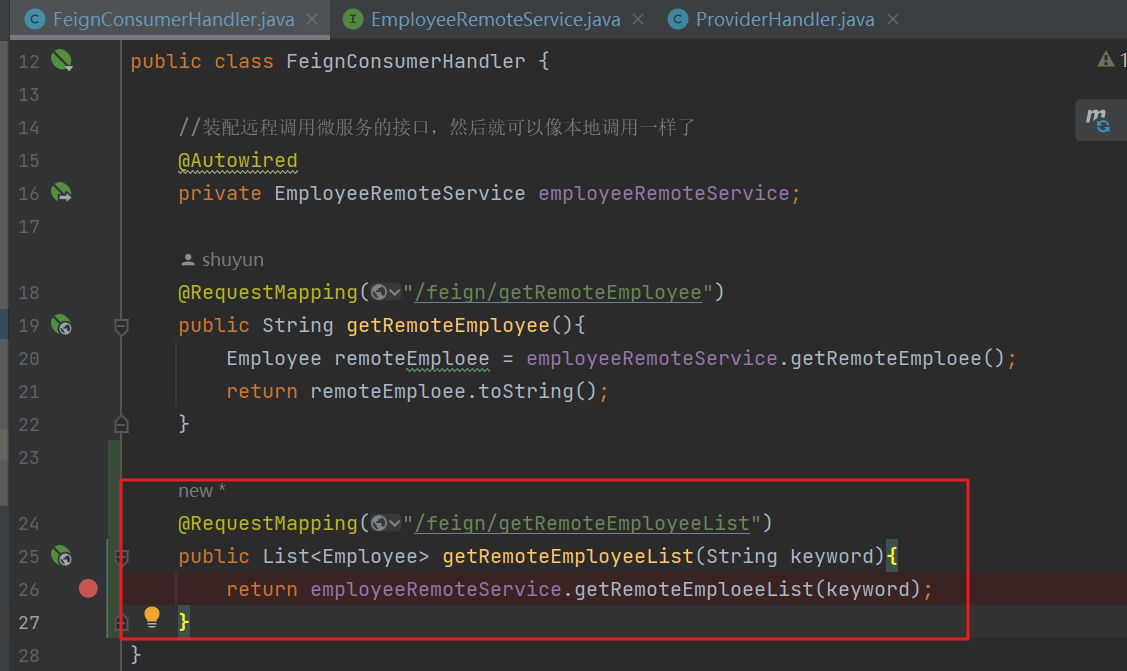
1.common

要加上@RequestParam注解，指定传参的名称



2.consumer

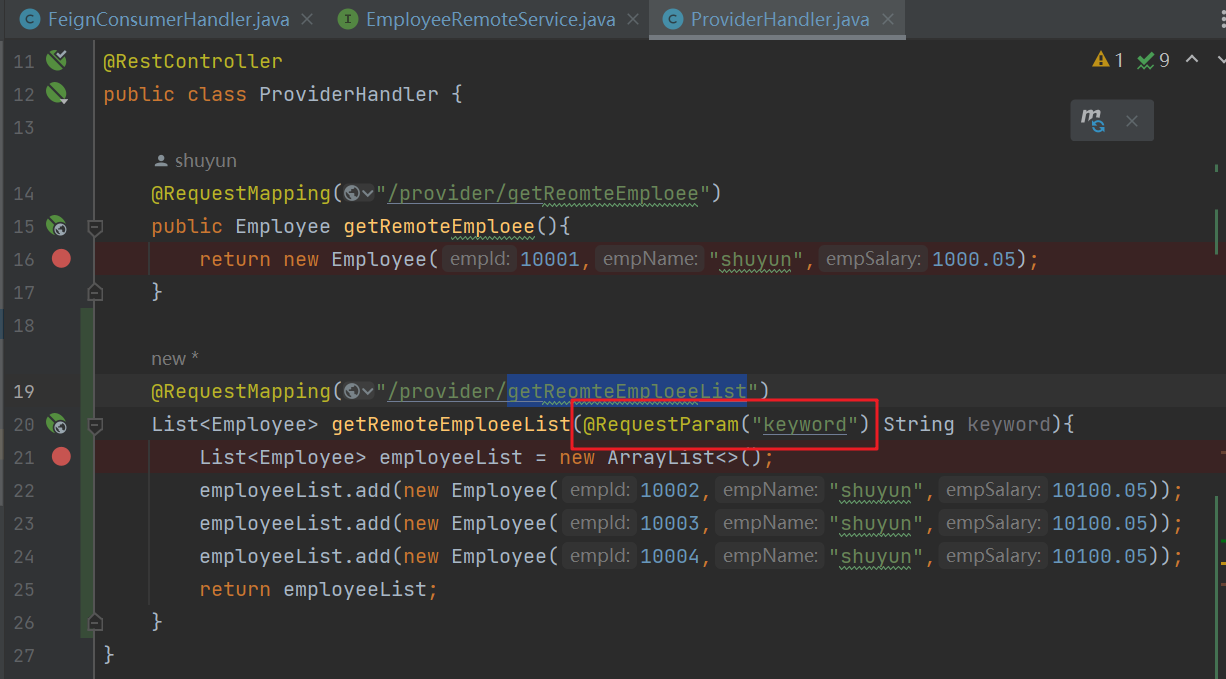
调用远程方法



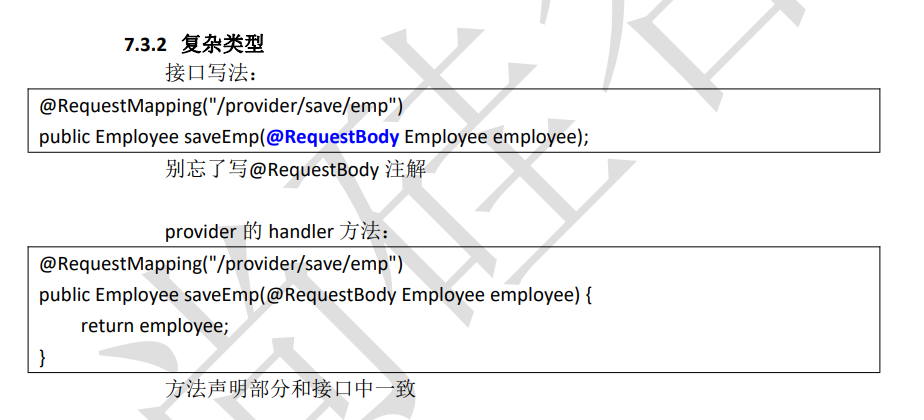
3.provider

方法声明要和common中的一致

@RequestParam注解也要加上

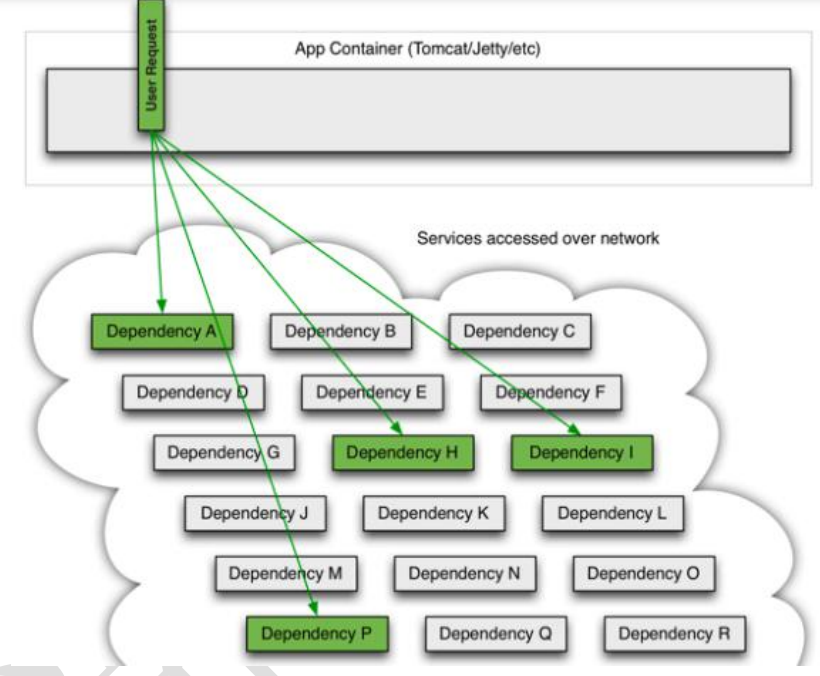


### 复杂传参



## Hystrix

### 分布式系统面临的问题



在微服务架构体系下，服务间的调用错综复杂，交织成一张大网。如果其中某个节点突然无法正常工作，则访问它的众多服务都会被卡住，进而有更多服务被卡住，系统中的线程、CPU、内存等资源有可能被迅速耗尽，最终整个服务体系崩溃。

我们管这样的现象叫服务雪崩。

### Hytrix 介绍

Hystrix 是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖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，避免级联故障，以提高分布式系统的弹性。

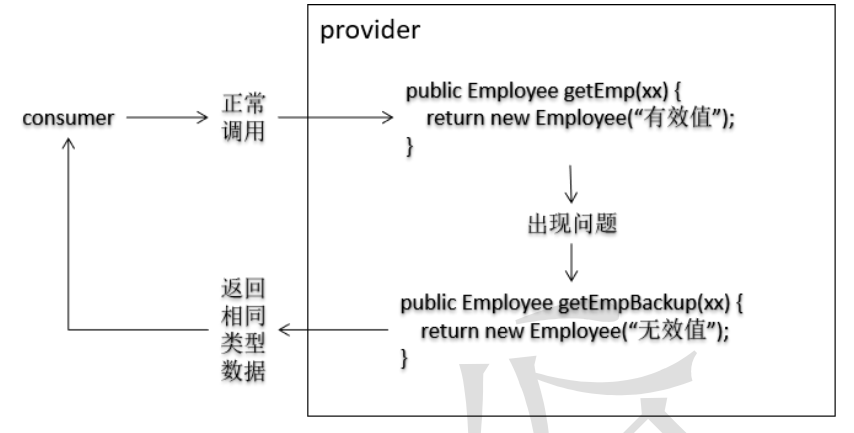
“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

Hytrix 能够提供服务降级、服务熔断、服务限流、接近实时的监控等方面的功能。

### 服务熔断机制

熔断机制是应对雪崩效应的一种微服务链路保护机制。

当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速响应错误信息。当检测到该节点微服务调用响应正常后恢复调用链路。在 SpringCloud 框架里熔断机制通过 Hystrix 实现。Hystrix 会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是 5 秒内 20 次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。



### 熔断-provider

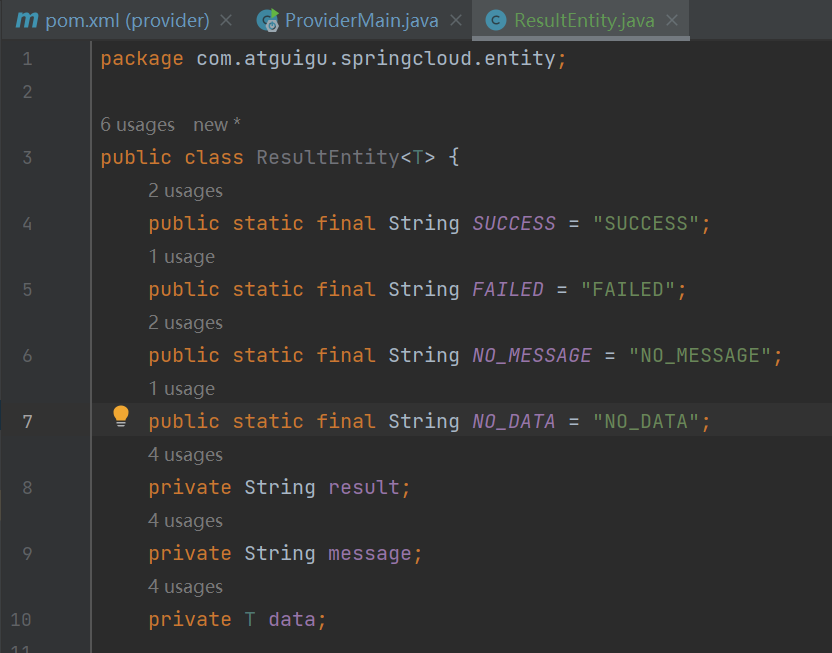
1.在provider工程中引入hystrix依赖

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

2.主启动类加上注解 @EnableCircuitBreaker 启用断路器功能



3.创建ResultEntity，统一作为Ajax请求和远程方法调用返回的类型，放在common工程中



4.provider控制器方法

①在原来的方法上加上注解，指定方法调用出错后调用的熔断方法

@HystrixCommand 注解通过 fallbackMethod 属性指定断路情况下要调用的备份方法

②新增熔断方法

@HystrixCommand(fallbackMethod = "getEmpBackUp")  
@RequestMapping("/provider/hystrix")  
public ResultEntity<Employee> getEmp(@RequestParam("key") String key){  
  
 if ("haha".equals(key)){  
 throw new RuntimeException();  
 }  
  
 return ResultEntity.*successWithData*(new Employee(10002,"shuyun",10100.05));  
}  
  
public ResultEntity<Employee> getEmpBackUp(@RequestParam("key") String key){  
 return ResultEntity.*failed*("触发熔断机制");  
}

## Zuul网关

# 会员系统

## 总体目标

## 整体架构

## 工程概述

# 环境搭建

## 约定

## eureka工程

## entity工程

## MySQL工程

## Redis工程

## 认证工程

## 网关

# 会员登录

# 发起项目

# 展示项目

# 支付案例

# 订单

# 支付

# 部署

# 总结

# 最后