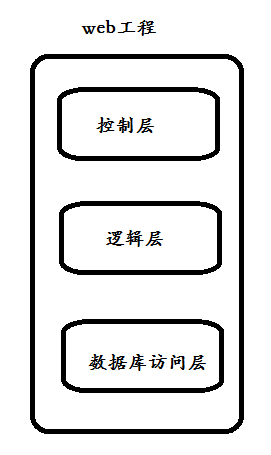
**SpringCloud微服务基础**

# 单点系统架构

## 传统项目架构

传统项目分为三层架构，将业务逻辑层、数据库访问层、控制层放入在一个项目中。

优点:适合于个人或者小团队开发，不适合大团队开发。



## 分布式项目架构

根据业务需求进行拆分成N个子系统，多个子系统相互协作才能完成业务流程子系统之间通讯使用RPC远程通讯技术。

优点:

1.把模块拆分，使用接口通信，降低模块之间的耦合度。

2.把项目拆分成若干个子项目，不同的团队负责不同的子项目。

3.增加功能时只需要再增加一个子项目，调用其它系统的接口就可以。

4.可以灵活的进行分布式部署。

有优点就有缺点，缺点如下：

1.系统之间交互需要使用远程通信，接口开发增加工作量。

2.各个模块有一些通用的业务逻辑无法共用。

为了解决上面分布式架构的缺点，我们引入了soa架构，SOA：Service Oriented Architecture面向服务的架构。也就是把工程拆分成服务层、表现层两个工程。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。

## 什么是项目集群

多台服务器部署相同应用构成一个集群

作用：通过负载均衡设备共同对外提供服务

## RPC远程调用

RPC 的全称是 Remote Procedure Call 是一种进程间通信方式。  
它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即无论是调用本地接口/服务的还是远程的接口/服务，本质上编写的调用代码基本相同。  
比如两台服务器A，B，一个应用部署在A服务器上，想要调用B服务器上应用提供的函数或者方法，由于不在一个内存空间，不能直接调用，这时候需要通过就可以应用RPC框架的实现来解决

### restful、soap、rpc

（1）restful是一种架构设计风格，提供了设计原则和约束条件，而不是架构。而满足这些约束条件和原则的应用程序或设计就是 RESTful架构或服务。  
（2）soap象访问协议是一种数据交换协议规范，  
是一种轻量的、简单的、基于XML的协议的规范。SOAP协议和HTTP协议一样，都是底层的通信协议，只是请求包的格式不同而已，SOAP包是XML格式的。  
soap

基于xml并封装成了符合http协议，因此，它符合任何路由器、 防火墙或代理服务器的要求。  
soap可以使用任何语言来完成，只要发送正确的soap请求即可，基于soap的服务可以在任何平台无需修改即可正常使用。  
（3）RPC就是从一台机器（客户端）上通过参数传递的方式调用另一台机器（服务器）上的一个函数或方法（可以统称为服务）并得到返回的结果。  
RPC 会隐藏底层的通讯细节（不需要直接处理Socket通讯或Http通讯）  
RPC 是一个请求响应模型。客户端发起请求，服务器返回响应（类似于Http的工作方式）  
RPC 在使用形式上像调用本地函数（或方法）一样去调用远程的函数（或方法）。

### rpc远程调用框架

几种比较典型的RPC的实现和调用框架。   
（1）RMI实现，利用java.rmi包实现，基于Java远程方法协议(Java Remote Method Protocol)   
和java的原生序列化。   
（2）Hessian，是一个轻量级的remoting onhttp工具，使用简单的方法提供了RMI的功能。 基于HTTP协议，采用二进制编解码。   
（3）thrift是一种可伸缩的跨语言服务的软件框架。thrift允许你定义一个描述文件，描述数据类型和服务接口。依据该文件，编译器方便地生成RPC客户端和服务器通信代码。

（4）SpringCloud 为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等。

（4） Dubbo是阿里巴巴公司开源的一个高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 Spring框架无缝集成。

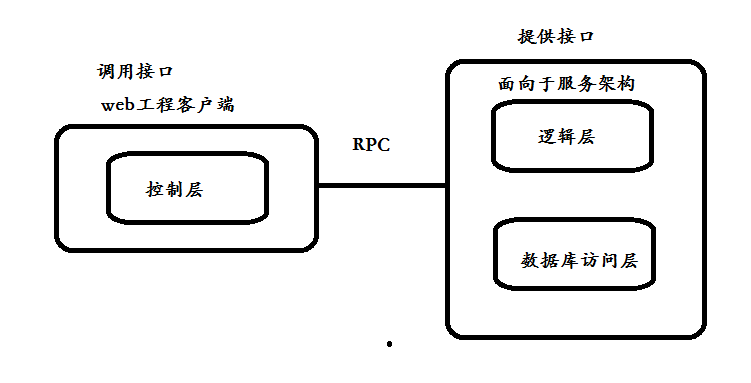
# 面向于服务架构

## 什么是SOA

业务系统分解为多个组件，让每个组件都独立提供离散，自治，可复用的服务能力

通过服务的组合和编排来实现上层的业务流程

作用：简化维护,降低整体风险,伸缩灵活



# 微服务架构

## 什么是微服务架构

架构设计概念,各服务间隔离（分布式也是隔离）,自治（分布式依赖整体组合）其它特性(单一职责,边界,异步通信,独立部署)是分布式概念的跟严格执行

SOA到微服务架构的演进过程

作用：各服务可独立应用，组合服务也可系统应用(巨石应用[monolith]的简化实现策略-平台思想)

## SOA架构与微服务架构区别

SOA架构主要针对企业级、采用ESB服务（ESB企业服务总线），非常重，需要序列化和反序列化，采用XML格式传输。

微服务架构主要互联网公司，轻量级、小巧，独立运行，基于Http+Rest+JSON格式传输。

ESB也可以说是传统中间件技术与XML、Web服务等技术相互结合的产物。

# SpringCloud

SpringCloud 为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、服务发现、断路器、路由、负载均衡、微代理、事件总线、全局锁、决策竞选、分布式会话等等。它运行环境简单，可以在开发人员的电脑上跑。另外说明spring cloud是基于Springboot的，所以需要开发中对Springboot有一定的了解，如果不了解的话可以看蚂蚁课堂SpringBoot课程。

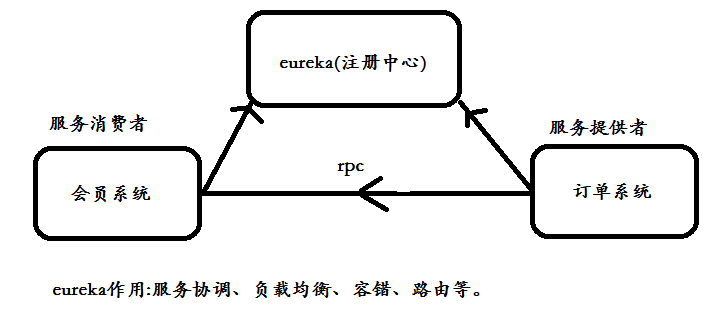
## 服务提供者与消费关系

服务提供者:提供服务被人调用

消费者:调用被人服务

## 服务的注册与发现(Eureka )

在这里，我们需要用的的组件上Spring Cloud Netflix的Eureka ,eureka是一个服务注册和发现模块。



### 什么是Eureka

官方的介绍在这里[Eureka wiki](https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance)。Eureka是Netflix开源的一个RESTful服务，主要用于服务的注册发现。Eureka由两个组件组成：Eureka服务器和Eureka客户端。Eureka服务器用作服务注册服务器。Eureka客户端是一个java客户端，用来简化与服务器的交互、作为轮询负载均衡器，并提供服务的故障切换支持。Netflix在其生产环境中使用的是另外的客户端，它提供基于流量、资源利用率以及出错状态的加权负载均衡。  
在我看来，Eureka的吸引力来源于以下几点：

**开源**：大家可以对实现一探究竟，甚至修改源码。

**可靠**：经过Netflix多年的生产环境考验，使用应该比较靠谱省心

**功能齐全**：不但提供了完整的注册发现服务，还有Ribbon等可以配合使用的服务。

**基于Java**：对于Java程序员来说，使用起来，心里比较有底。

**spring cloud**可以使用Spring Cloud, 与Eureka进行了很好的集成，使用起来非常方便。

### 实现服务注册

#### 创建EureKaserver 项目

#### Maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <!--eureka server -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  </dependency>  <!-- spring boot test -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Dalston.RC1</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  <repositories>  <repository>  <id>spring-milestones</id>  <name>Spring Milestones</name>  <url>https://repo.spring.io/milestone</url>  <snapshots>  <enabled>false</enabled>  </snapshots>  </repository>  </repositories> |

#### 配置application.yml

|  |
| --- |
| server:  port: 8888  eureka:  instance:  hostname: localhost  client:  registerWithEureka: **false**  fetchRegistry: **false**  serviceUrl:  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/ |

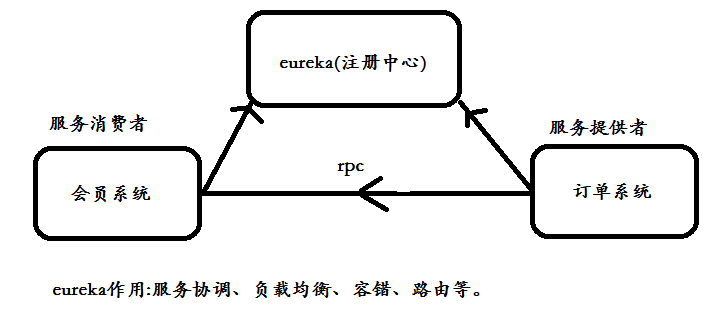
#### 启动EurekaServer

|  |
| --- |
| @SpringBootApplication  @EnableEurekaServer  **public** **class** App {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(App.**class**, args);  }  } |

#### 打开eureka server 界面的

|  |
| --- |
| [http://localhost:8761](http://localhost:8761/) ,界面如下：  Paste_Image.png  No application available 没有服务被发现 ……^\_^  因为没有注册服务当然不可能有服务被发现了。 |

## 实现案例订单服务调用会员服务查询用户信息



## 服务提供者

创建一个服务提供者 会员服务工程 (eurekaMember),提供会员查询服务信息

### 创建项目service-member

### Maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Dalston.RC1</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  <repositories>  <repository>  <id>spring-milestones</id>  <name>Spring Milestones</name>  <url>https://repo.spring.io/milestone</url>  <snapshots>  <enabled>false</enabled>  </snapshots>  </repository>  </repositories> |

### application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8762  spring:  application:  name: service-member |

### 服务接口

|  |
| --- |
| @RestController  **public** **class** MemberController {  @RequestMapping("/getUserList")  **public** List<String> getUserList() {  List<String> listUser = **new** ArrayList<String>();  listUser.add("zhangsan");  listUser.add("lisi");  listUser.add("yushengjun");  **return** listUser;  }  } |

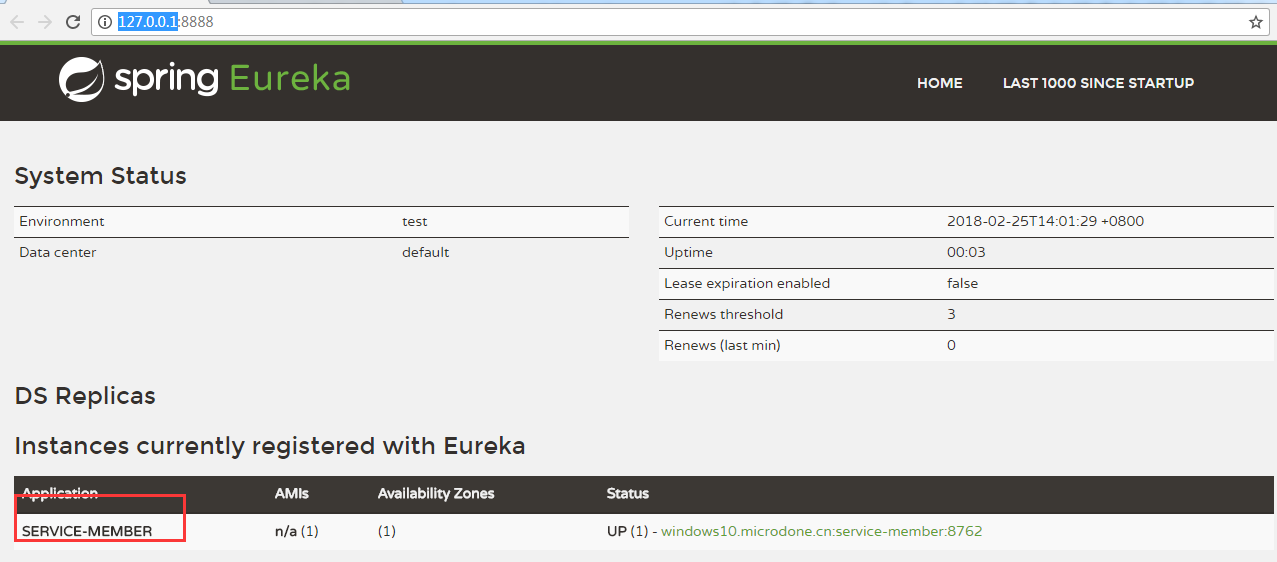
### 发布服务

通过注解@EnableEurekaClient 表明自己是一个eurekaclient.

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  **public** **class** AppMember {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppMember.**class**, args);  }  } |

### 演示效果

需要指明spring.application.name,这个很重要，这在以后的服务与服务之间相互调用一般都是根据这个name 。   
启动工程，打开127.0.0.1:8888 ，即eureka server 的网址：



你会发现一个服务已经注册在服务中了，服务名为SERVICE-HI ,端口为7862

这时打开 http://127.0.0.1:8762/getUserList ，你会在浏览器上看到 :

["zhangsan","lisi","yushengjun"]

## 服务消费者

### 创建项目sercice-order

### Maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-ribbon</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Dalston.RC1</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  <repositories>  <repository>  <id>spring-milestones</id>  <name>Spring Milestones</name>  <url>https://repo.spring.io/milestone</url>  <snapshots>  <enabled>false</enabled>  </snapshots>  </repository>  </repositories> |

### application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8764  spring:  application:  name: service-order |

### 编写service,调用service-member

|  |
| --- |
| @SuppressWarnings("unchecked")  @Service  **public** **class** MemberService {  @Autowired  RestTemplate restTemplate;  **public** List<String> getOrderByUserList() {  **return** restTemplate.getForObject("http://service-member/getUserList", List.**class**);  }  } |

### 演示效果

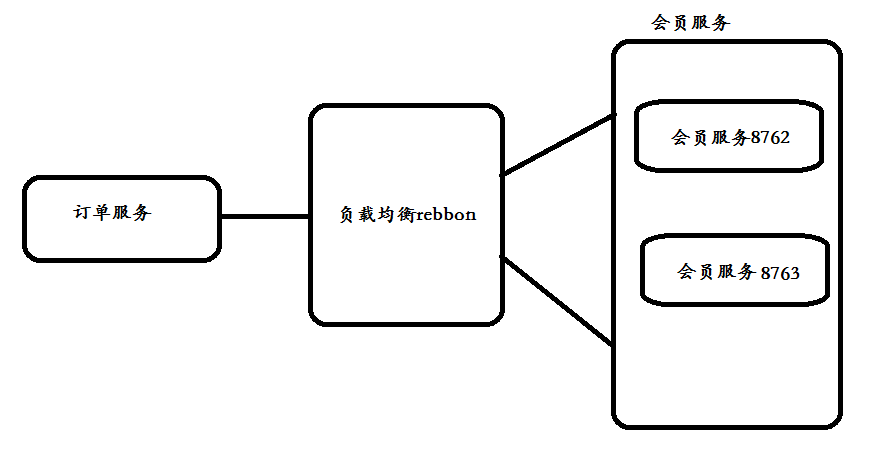
|  |
| --- |
| @EnableEurekaClient  @SpringBootApplication  **public** **class** AppOrder {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppOrder.**class**, args);  }  @Bean  @LoadBalanced  RestTemplate restTemplate() {  **return** **new** RestTemplate();  }  } |

|  |
| --- |
|  |

在工程的启动类中,通过@EnableDiscoveryClient向服务中心注册；并且向程序的ioc注入一个bean: restTemplate;并通过@LoadBalanced注解表明这个restRemplate开启负载均衡的功能。

## 使用ribbon实现负载均衡

启动两个会员服务工程，端口号分别为8762、8763，订单服务 使用负载均衡策略轮训到会员服务接口。



### 什么是ribbon

ribbon是一个负载均衡客户端 类似nginx反向代理，可以很好的控制htt和tcp的一些行为。Feign默认集成了ribbon。

### 修改会员服务工程代码区分端口项目

|  |
| --- |
| @Value("${server.port}")  **private** String serverPort;  @RequestMapping("/getUserList")  **public** List<String> getUserList() {  List<String> listUser = **new** ArrayList<String>();  listUser.add("zhangsan");  listUser.add("lisi");  listUser.add("yushengjun");  listUser.add("端口号:"+serverPort);  **return** listUser;  } |

### 开启ribbon

@LoadBalanced注解表明这个restRemplate开启负载均衡的功能。

# [服务消费者（Feign）](http://blog.csdn.net/forezp/article/details/69808079)

## 什么是Feign

Feign是一个声明式的伪Http客户端，它使得写Http客户端变得更简单。使用Feign，只需要创建一个接口并注解。它具有可插拔的注解特性，可使用Feign 注解和JAX-RS注解。Feign支持可插拔的编码器和解码器。Feign默认集成了Ribbon，并和Eureka结合，默认实现了负载均衡的效果。

简而言之：

* Feign 采用的是基于接口的注解
* Feign 整合了ribbon

## 创建service-order-feign工程

## Maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.2.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-ribbon</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-feign</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Dalston.RC1</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  <repositories>  <repository>  <id>spring-milestones</id>  <name>Spring Milestones</name>  <url>https://repo.spring.io/milestone</url>  <snapshots>  <enabled>false</enabled>  </snapshots>  </repository>  </repositories> |

## application.yml配置

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8888/eureka/  server:  port: 8765  spring:  application:  name: service-order-feign |

### 编写service,调用service-member

|  |
| --- |
| @FeignClient("service-member")  **public** **interface** MemberFeign {  @RequestMapping("/getUserList")  **public** List<String> getOrderByUserList();  } |

@FeignClient 需要调用服务名称，@RequestMapping服务请求名称

### 演示效果

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  @EnableFeignClients  **public** **class** OrderFeignApp {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(OrderFeignApp.**class**, args);  }  } |