学成在线 第9天 讲义-课程预览 Eureka Feign

1 Eureka注册中心

1.1 需求分析

在前后端分离架构中，服务层被拆分成了很多的微服务，微服务的信息如何管理？Spring Cloud中提供服务注册中心来管理微服务信息。

如果使用配置文件，则需要配每个微服务的IP地址和端口，非常麻烦

为什么 要用注册中心？

1、微服务数量众多，要进行远程调用就需要知道服务端的ip地址和端口，注册中心帮助我们管理这些服务的ip和端口。

上报心跳

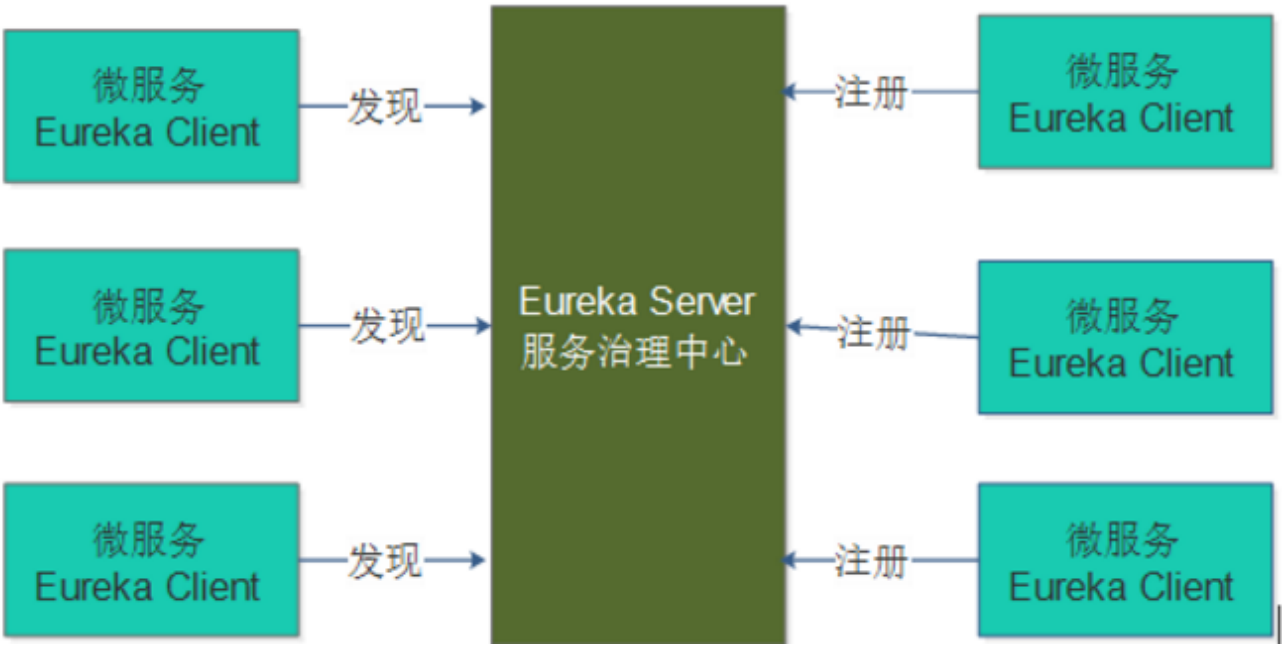
2、微服务会实时上报自己的状态，注册中心统一管理这些微服务的状态，将存在问题的服务踢出服务列表，客户端获取到可用的服务进行调用。

1.3 Eureka注册中心

1.3.1 Eureka介绍

Spring Cloud Eureka 是对Netflix公司的Eureka的二次封装，它实现了服务治理的功能，Spring Cloud Eureka提供服务端与客户端，服务端即是Eureka服务注册中心，客户端完成微服务向Eureka服务的注册与发现。服务端和客户端均采用Java语言编写。下图显示了Eureka Server与Eureka Client的关系：

每个微服务都有一个客户端



1、Eureka Server是服务端，负责管理各各微服务结点的信息和状态。

1、EurekaServer是服务端，负责管理各各微服务结点的信息和状态。

2、在微服务上部署Eureka Client程序，远程访问Eureka Server将自己注册在Eureka Server。

3、微服务需要调用另一个微服务时从Eureka Server中获取服务调用地址，进行远程调用。

1.3.2 Eureka Server搭建

1.3.2.1 单机环境搭建

1、创建xc-govern-center工程：

包结构：com.xuecheng.govern.center

2、添加依赖

在父工程添加：（有了则不用重复添加） 依赖版本控制

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-dependencies</artifactId>
    <version>Finchley.SR1</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

在Eureka Server工程添加： 现在还不能使用注解@SpringBootApplication,
因为父类只有版本控制,没有引入任何包

```
<dependencies>
    <!-- 导入Eureka服务的依赖 -->
    <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
```

3、启动类

```
@EnableEurekaServer//标识这是一个Eureka服务
@SpringBootApplication
public class GovernCenterApplication {
    public static void main(String[] args) {
        SpringApplication.run(GovernCenterApplication.class, args);
    }
}
```

4、@EnableEurekaServer

需要在启动类上用@EnableEurekaServer标识此服务为Eureka服务

5、从其它服务拷贝application.yml和logback-spring.xml。

application.yml的配置内容如下：

```
server:
  port: 50101 #服务端口

spring:
  application:
    name: xc-govern-center #指定服务名

eureka:
  client:
    registerWithEureka: false #服务注册，是否将自己注册到Eureka服务中
    fetchRegistry: false #服务发现，是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端的交互地址，高可用状态配置对方的地址，单机状态配置自己（如果不配置则默认本机8761端口）
      defaultZone: http://localhost:50101/eureka/
  server:
    enable-self-preservation: false #是否开启自我保护模式
    eviction-interval-timer-in-ms: 60000 #服务注册表清理间隔（单位毫秒，默认是60*1000）
```

registerWithEureka：被其它服务调用时需向Eureka注册

fetchRegistry：需要从Eureka中查找要调用的目标服务时需要设置为true

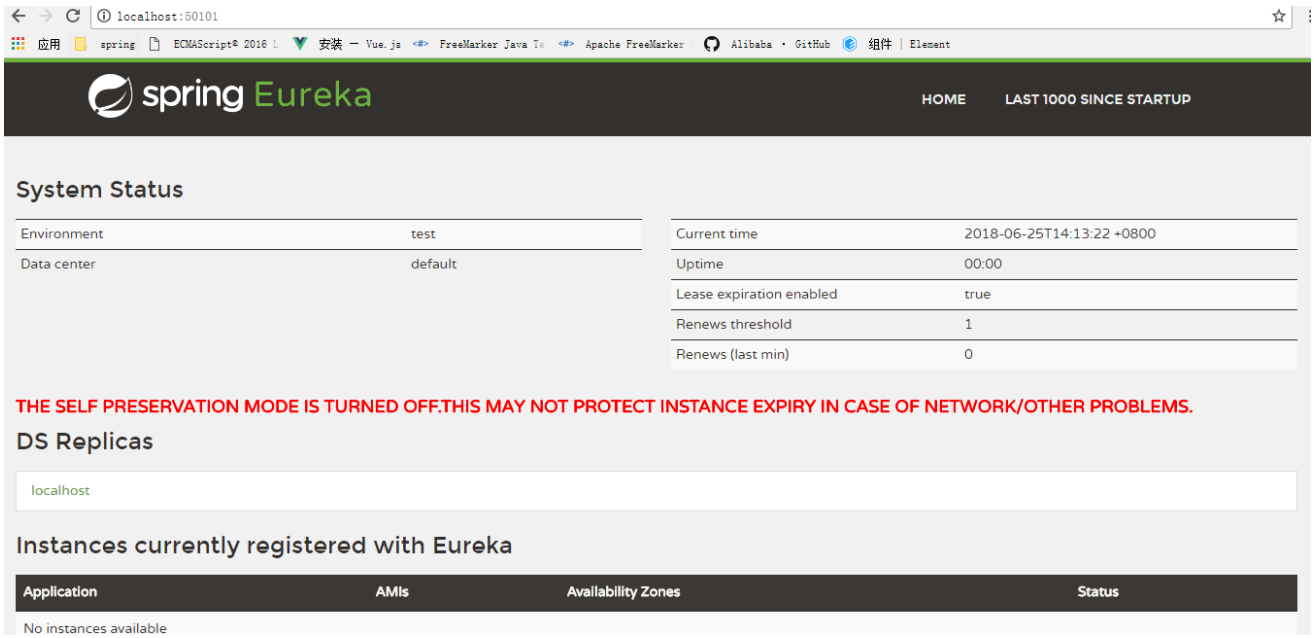
serviceUrl.defaultZone 配置上报Eureka服务地址高可用状态配置对方的地址，单机状态配置自己

enable-self-preservation：自保护设置，下边有介绍。

eviction-interval-timer-in-ms：清理失效结点的间隔，在这个时间段内如果没有收到该结点的上报则将结点从服务列表中剔除。

5、启动Eureka Server

启动Eureka Server，浏览50101端口。



说明：

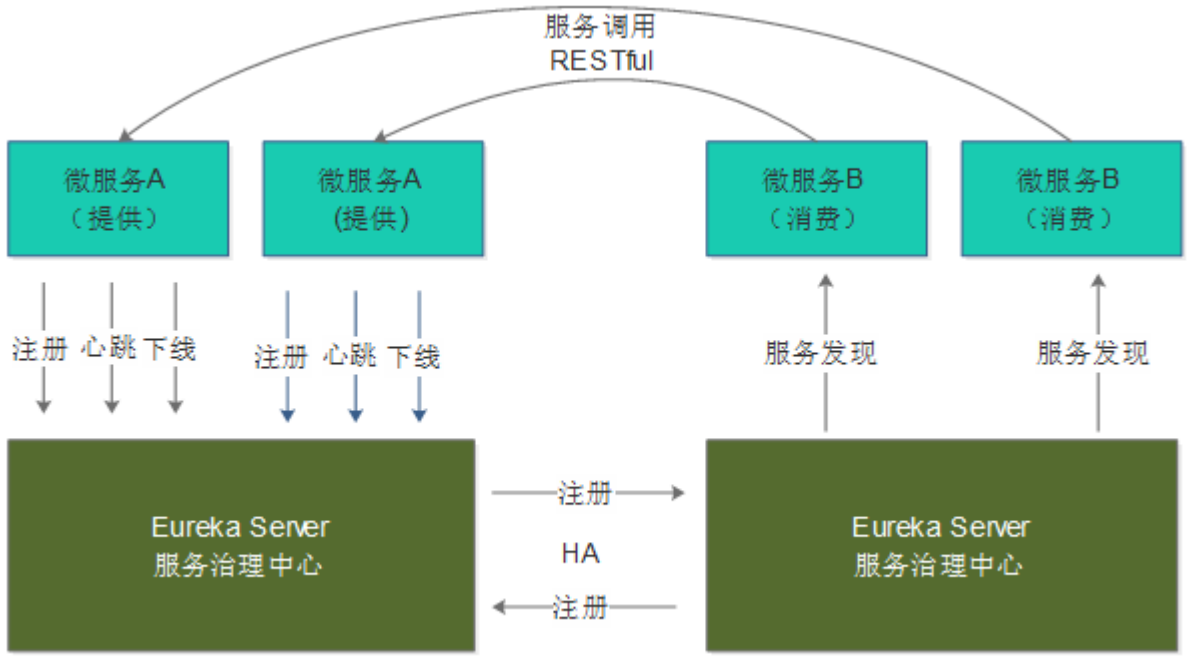
上图红色提示信息：
THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.
自我保护模式被关闭。在网络或其他问题的情况下可能不会保护实例失效。

Eureka Server有一种自我保护模式，当微服务不再向Eureka Server上报状态，Eureka Server会从服务列表将此服务删除，如果出现网络异常情况（微服务正常），此时Eureka server进入自保护模式，不再将微服务从服务列表删除。

在开发阶段建议关闭自保护模式。 开发阶段不会出现网络问题,但需要频繁开关微服务

1.3.2.2 高可用环境搭建

Eureka Server 高可用环境需要部署两个Eureka server，它们互相向对方注册。如果在本机启动两个Eureka需要注意两个Eureka Server的端口要设置不一样，这里我们部署一个Eureka Server工程，将端口可配置，制作两个Eureka Server启动脚本，启动不同的端口，如下图：



1、在实际使用时Eureka Server至少部署两台服务器，实现高可用。

2、两台Eureka Server互相注册。

3、微服务需要连接两台Eureka Server注册，当其中一台Eureka死掉也不会影响服务的注册与发现。

4、微服务会定时向Eureka server发送心跳，报告自己的状态。

5、微服务从注册中心获取服务地址以RESTful方式发起远程调用。

配置如下：

1、端口可配置

```
server:
  port: ${PORT:50101} #服务端口
```

2、Eureka服务端的交互地址可配置

```
eureka:
  client:
    registerWithEureka: true #服务注册，是否将自己注册到Eureka服务中
    fetchRegistry: true #服务发现，是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端的交互地址，高可用状态配置对方的地址，单机状态配置自己（如果不配置则默认本机8761端口）
      defaultZone: ${EUREKA_SERVER:http://eureka02:50102/eureka/}
```

和上面不一样

3、配置hostname

Eureka 组成高可用，两个Eureka互相向对方注册，这里需要通过域名或主机名访问，这里我们设置两个Eureka服务的主机名分别为 eureka01、eureka02。

完整的eureka配置如下：

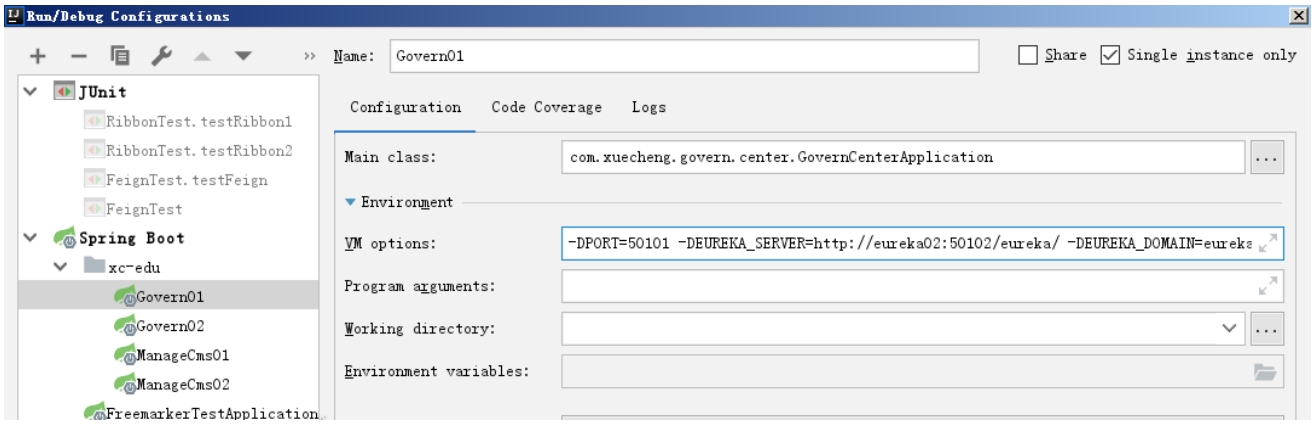
```
eureka:
  client:
    registerWithEureka: true #服务注册，是否将自己注册到Eureka服务中
    fetchRegistry: true #服务发现，是否从Eureka中获取注册信息
    serviceUrl: #Eureka客户端与Eureka服务端的交互地址，高可用状态配置对方的地址，单机状态配置自己（如果不配置则默认本机8761端口）
      defaultZone: ${EUREKA_SERVER:http://eureka02:50102/eureka/}
  server:
    enable-self-preservation: false #是否开启自我保护模式
    eviction-interval-timer-in-ms: 60000 #服务注册表清理间隔（单位毫秒，默认是60*1000）
  instance:
    hostname: ${EUREKA_DOMAIN:eureka01}
```

当前主机,可以理解为域名

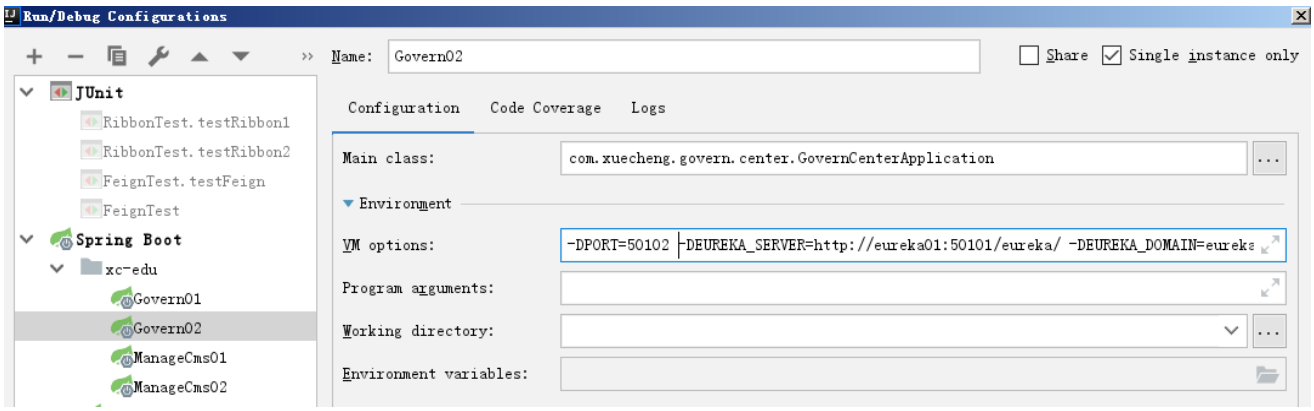
配置host文件，域名映射

4、在IDEA中制作启动脚本

启动1：



启动2：

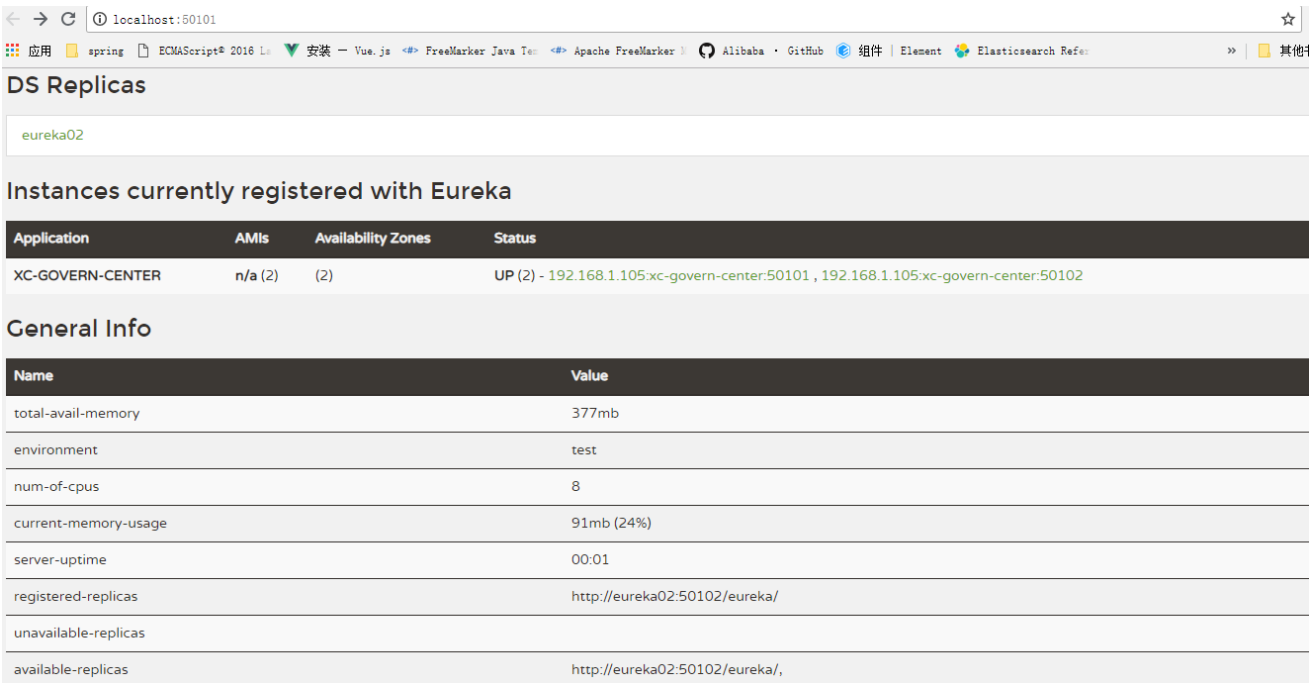


运行两个启动脚本，分别浏览：

http://localhost:50101/

http://localhost:50102/

Eureka主画面如下：



1.3.3 服务注册

1.3.3.1 将cms注册到Eureka Server

下边实现cms向Eureka Server注册。

1、在cms服务中添加依赖

```
<!-- 导入Eureka客户端的依赖 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

2、在application.yml配置

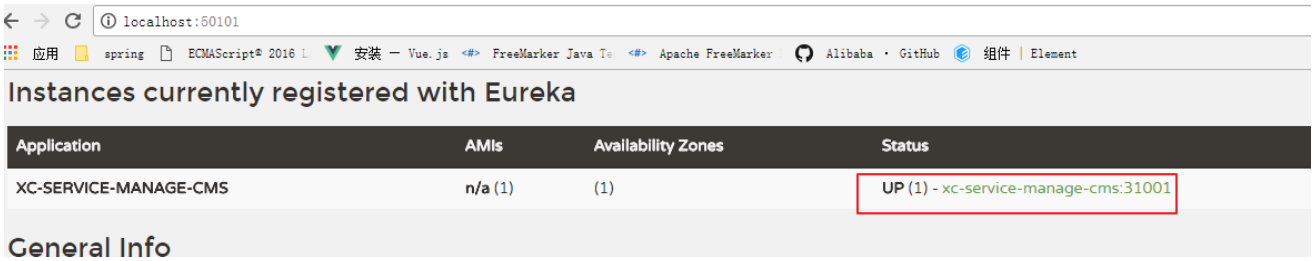
```
eureka:
  client:
    registerWithEureka: true #服务注册开关
    fetchRegistry: true #服务发现开关
    serviceUrl: #Eureka客户端与Eureka服务端进行交互的地址，多个中间用逗号分隔
      defaultZone: ${EUREKA_SERVER:http://localhost:50101/eureka/, http://localhost:50102/eureka/}
                                    alternative url,并不是同时的
  instance:
    prefer-ip-address:  true  #将自己的ip地址注册到Eureka服务中
    #ip-address:${IP_ADDRESS:127.0.0.1}# 主动填写自己的的ip地址
    instance-id: ${spring.application.name}:${server.port} #指定实例id
```

3、在启动类上添加注解

在启动类上添加注解 @EnableDiscoveryClient ，表示它是一个Eureka的客户端

代替 ：@SpringCloudApplication 会有异常

4、刷新Eureka Server查看注册情况



1.3.3.2 将manage-course注册到Eureka Server

方法同上。

1、在manage-course工程中添加spring-cloud-starter-eureka依赖：

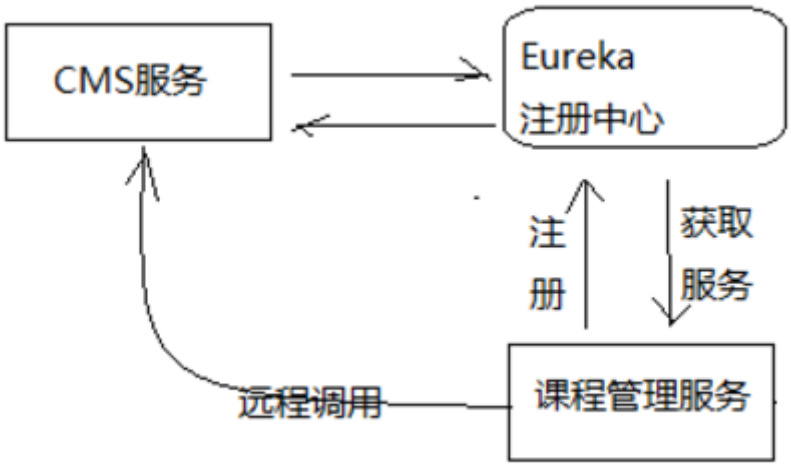
2、在application.yml配置eureka

3、在启动类上添加注解 @EnableDiscoveryClient

2 Feign远程调用

在前后端分离架构中，服务层被拆分成了很多的微服务，服务与服务之间难免发生交互，比如：课程发布需要调用CMS服务生成课程静态化页面，本节研究微服务远程调用所使用的技术。

下图是课程管理服务远程调用CMS服务的流程图：



工作流程如下：

1、cms服务将自己注册到注册中心。

2、课程管理服务从注册中心获取cms服务的地址。

3、课程管理服务远程调用cms服务。

2.1 Ribbon

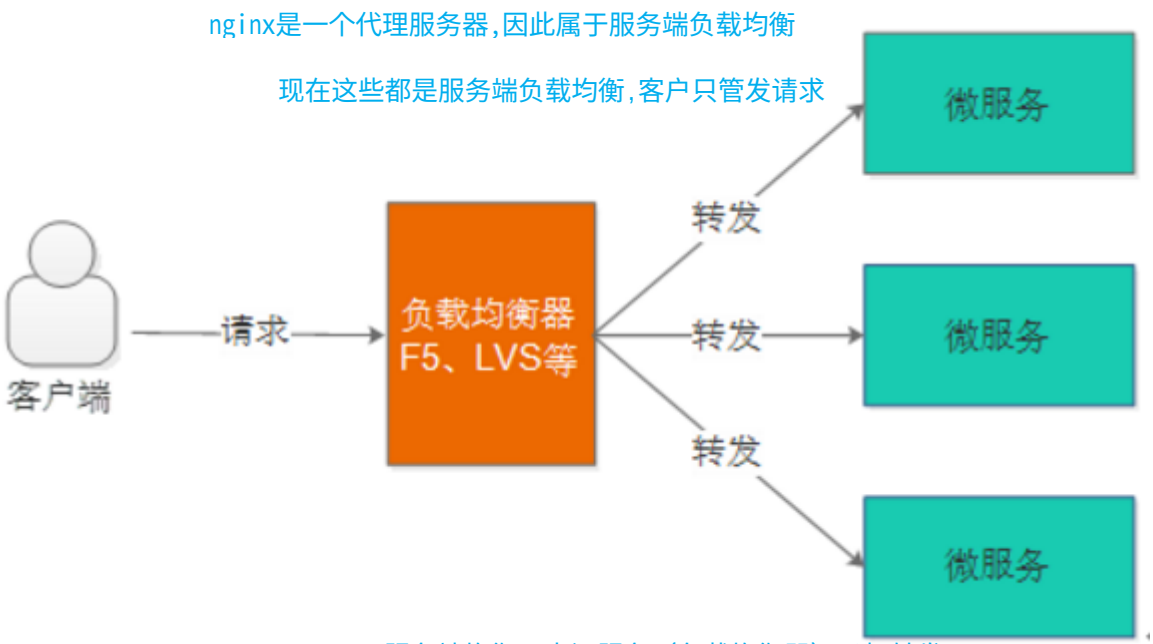
2.1.1 Ribbon介绍

Ribbon是Netflix公司开源的一个负载均衡的项目（https://github.com/Netflix/ribbon），它是一个基于HTTP、TCP的客户端负载均衡器。

1、什么是负载均衡？

负载均衡是微服务架构中必须使用的技术，通过负载均衡来实现系统的高可用、集群扩容等功能。负载均衡可通过硬件设备及软件来实现，硬件比如：F5、Array等，软件比如：LVS、Nginx等。

如下图是负载均衡的架构图：



服务端均衡：中间服务（负载均衡器）承担转发

用户请求先到达负载均衡器（也相当于一个服务），负载均衡器根据负载均衡算法将请求转发到微服务。负载均衡算法有：轮训、随机、加权轮训、加权随机、地址哈希等方法，负载均衡器维护一份服务列表，根据负载均衡算法将请求转发到相应的微服务上，所以负载均衡可以为微服务集群分担请求，降低系统的压力。

2、什么是客户端负载均衡？

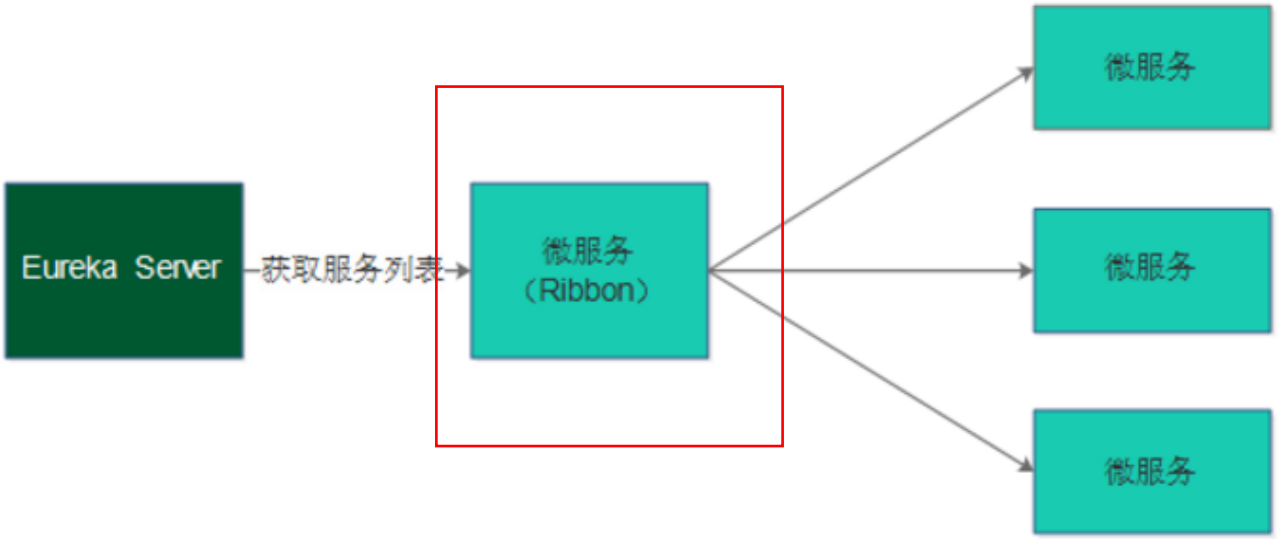
客户端均衡： 没有中间服务，客户端负责承担转发

上图是服务端负载均衡，客户端负载均衡与服务端负载均衡的区别在于客户端要维护一份服务列表，Ribbon从Eureka Server获取服务列表，Ribbon根据负载均衡算法直接请求到具体的微服务，中间省去了负载均衡服务。

如下图是Ribbon负载均衡的流程图：请求完全后应该会把服务列表返回给eureka

不合理，相同的微服务如何保证服务列表相同，并且每个微服务都要维护一份，太麻烦了，直接Eureka Server维护不好吗，一份就可以了，

答： 列表很简单，但是转发就比较复杂了



1、在消费微服务中使用Ribbon实现负载均衡，Ribbon先从EurekaServer中获取服务列表。

2、Ribbon根据负载均衡的算法去调用微服务。

2.1.2 Ribbon测试

复杂用法source

Spring Cloud引入Ribbon配合 restTemplate 实现客户端负载均衡。Java中远程调用的技术有很多，如：webservice、socket、rmi、Apache HttpClient、OkHttp等，互联网项目使用基于http的客户端较多，本项目使用OkHttp。

manage-course微服务中

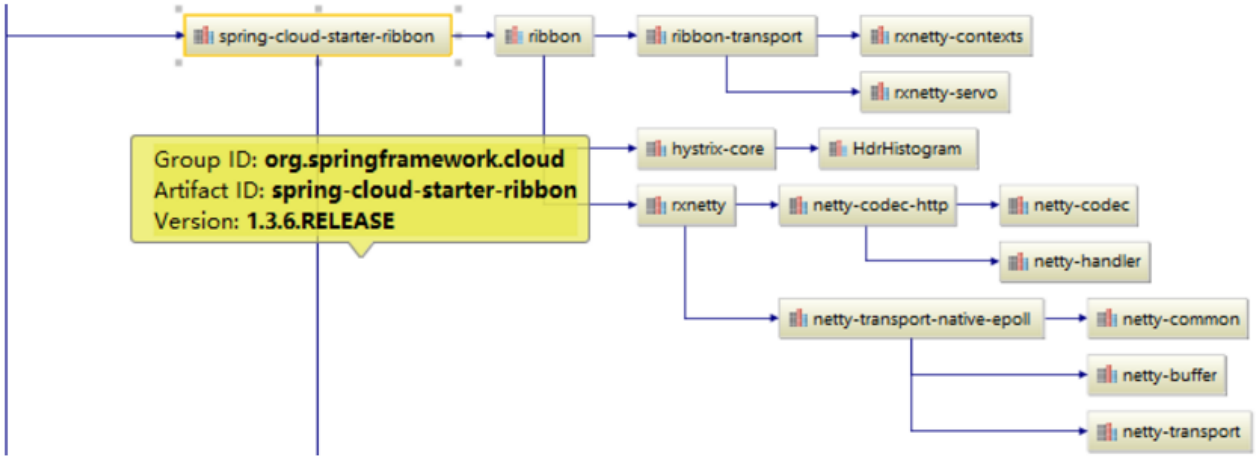
1、在客户端添加Ribbon依赖：

这里在课程管理服务配置ribbon依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
</dependency>
```

为什么这里要加入版本?父类pom没有添加版本能控制

由于依赖了spring-cloud-starter-eureka，会自动添加spring-cloud-starter-ribbon依赖



2、配置Ribbon参数

这里在课程管理服务的application.yml中配置ribbon参数　　是一个独立的配置

```
ribbon:
  MaxAutoRetries: 2 #最大重试次数，当Eureka中可以找到服务，但是服务连不上时将会重试
  MaxAutoRetriesNextServer: 3 #切换实例的重试次数
  OkToRetryOnAllOperations: false  #对所有操作请求都进行重试，如果是get则可以，如果是post，put等操作没有实现幂等的情况下是很危险的,所以设置为false
  ConnectTimeout: 5000  #请求连接的超时时间
  ReadTimeout: 6000 #请求处理的超时时间
```

3、负载均衡测试

1）启动两个cms服务，注意端口要不一致

启动完成观察Eureka Server的服务列表

Application	AMIs	Availability Zones	Status
XC-SERVICE-MANAGE-CMS	n/a (2)	(2)	UP (2) - xc-service-manage-cms:31001 , xc-service-manage-cms:31002

2）定义RestTemplate，使用@LoadBalanced注解

在课程管理服务的启动类中定义RestTemplate

```
@Bean
@LoadBalanced    可以不用这个注解
public RestTemplate restTemplate() {
    return new RestTemplate(new OkHttp3ClientHttpRequestFactory());
}
```

3）测试代码

在课程管理服务工程创建单元测试代码，远程调用cms的查询页面接口：

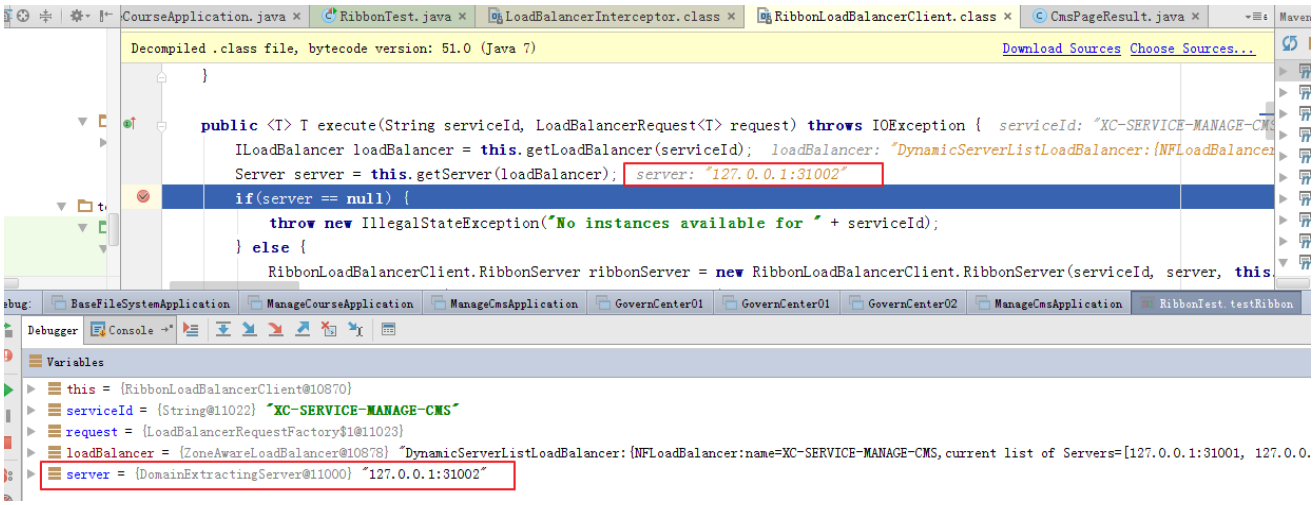
```
    //负载均衡调用
    @Test
    public void testRibbon() {
        //服务id
        String serviceId = "XC-SERVICE-MANAGE-CMS";注意pdf的-和其他的-是不一样的
        for(int i=0;i<10;i++){
            //通过服务id调用
            ResponseEntity<CmsPage> forEntity = restTemplate.getForEntity("http://" + serviceId
+ "/cms/page/get/5a754adf6abb500ad05688d9", CmsPage.class);
            CmsPage cmsPage = forEntity.getBody();
            System.out.println(cmsPage);
        }
    }
```

```
@Autowired
LoadBalancerClient loadBalancerClient;
```

```
//从eureka中获取认证服务的地址（因为spring security在认证服务中）
//从eureka中获取认证服务的一个实例的地址
ServiceInstance serviceInstance = loadBalancerClient.choose(XcServiceList.XC_SERVICE_UCENTER_AUTH);
//此地址就是http://ip:port
URI uri = serviceInstance.getUri();
//令牌申请的地址 http://localhost:40400/auth/oauth/token
```

4）负载均衡测试

添加@LoadBalanced注解后，restTemplate会走LoadBalancerInterceptor拦截器，此拦截器中会通过RibbonLoadBalancerClient查询服务地址，可以在此类打断点观察每次调用的服务地址和端口，两个cms服务会轮流被调用。



2.2 Feign

2.2.1 Feign介绍

Feign是Netflix公司开源的轻量级rest客户端，使用Feign可以非常方便的实现Http 客户端。Spring Cloud引入Feign并且集成了Ribbon实现客户端负载均衡调用。

2.2.2 Feign测试

1、在客户端添加依赖

在课程管理服务添加下边的依赖：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>com.netflix.feign</groupId>
    <artifactId>feign-okhttp</artifactId>
</dependency>
```

自动依赖ribbon

2、定义FeignClient接口

client包表示都是客户端

```
@EnableFeignClients(basePackages = "mall.cloud.api")
```

参考Swagger文档定义FeignClient，注意接口的Url、请求参数类型、返回值类型与Swagger接口一致。

在课程管理服务中创建client包，定义查询cms页面的客户端该用接口，

可以直接在spi中写getMapping然后再再该模块中继承使用

```
@FeignClient(value = XcServiceList.XC_SERVICE_MANAGE_CMS)
public interface CmsPageClient {

    @GetMapping("/cms/page/get/{id}")
    public CmsPage findById(@PathVariable("id") String id);
}
```

集成了SpringMVC的功能

使得这里的http客户端更加关注业务逻辑，
(这里不需要json数据)，更好地与controller对接，
不需要关注http远程调用问题

3、启动类添加@EnableFeignClients注解

```
@EnableFeignClients(basePackages = "mall.cloud.api")
```

4、测试

feign不经过loadbalance

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class FeignTest {
    @Autowired
    CmsPageClient cmsPageClient;
    @Test
    public void testFeign() {
        //通过服务id调用cms的查询页面接口
        CmsPage cmsPage = cmsPageClient.findById("5a754adf6abb500ad05688d9");
        System.out.println(cmsPage);
    }

}
```

生成代理对象

Feign工作原理如下：

对应包路径

1、 启动类添加@EnableFeignClients注解，Spring会扫描标记了@FeignClient注解的接口，并生成此接口的代理对象

2、 @FeignClient(value = XcServiceList.XC_SERVICE_MANAGE_CMS)即指定了cms的服务名称，Feign会从注册中心获取cms服务列表，并通过负载均衡算法进行服务调用。

3、在接口方法 中使用注解@GetMapping("/cms/page/get/{id}")，指定调用的url，Feign将根据url进行远程调用。

2.2.4 Feign注意点

SpringCloud对Feign进行了增强兼容了SpringMVC的注解 ，我们在使用SpringMVC的注解时需要注意：

1、feignClient接口 有参数在参数必须加@PathVariable("XXX")和@RequestParam("XXX")

2、feignClient返回值为复杂对象时其类型必须有无参构造函数。