**springcloud**

# 1.springcloud

## eureka

## ribbon

## feign

## hystrix

## spring cloud config

## zuul

# 2.Eureka

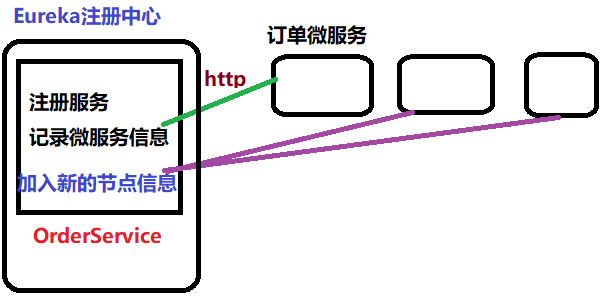
## 2.1 Eureka

eureka是spring cloud 的微服务框架的核心组件(管理单位);是完成当前所有集群微服务的注册与发现的核心进程;可以动态的处理所有负载均衡访问的集群（serverList）的管理工作;

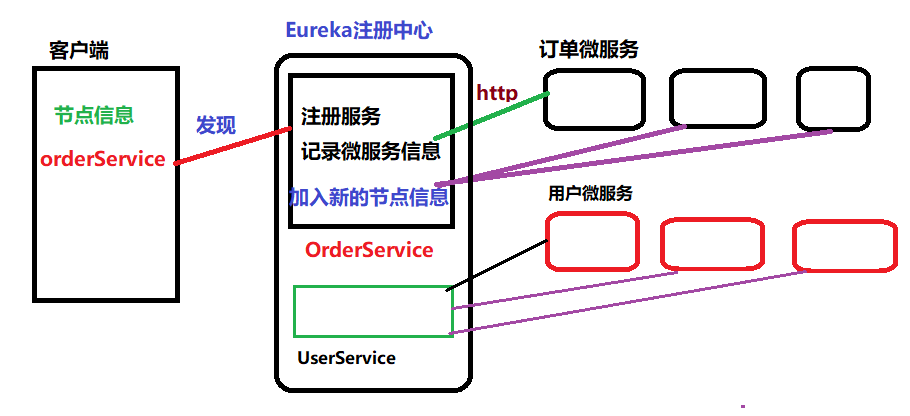
* 服务的注册:微服务节点作为真正的服务,功能的提供者.动态注册在eureka中心。

（nginx是静态的。维护serverList是一个静态文件，当服务器发生变化时，只能手动的修改静态文件，修改完后还要重新启动nginx才能完成。）

图示

描述已自动生成 

* **服务的发现**:**客户端**具备访问eureka的能力,动态的获取某个一个服务的真正提供者信息,从而发起访问调用;



* eureka组件中的角色:
  + 注册中心,称为eureka中的服务端角色
  + 注册在中心的微服务们,称为eureka中的客户端角色

## 2.2搭建一个eureka的注册中心，管理所有的微服务

* **创建一个quickstart工程**

图形用户界面, 应用程序

描述已自动生成

* **pom文件中引入资源(导入springcloud的相关内容,springboot)**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.9.RELEASE</version>

</parent>

引入springcloud资源dependencyManagement import导入

<!-- 引入springcloud资源 Edgware.RELEASE 版本-->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Springcloud与SpringBoot的版本对应

表格

描述已自动生成

引入eureka依赖内容

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

* **配置properties文件**

1）与网络访问有关的主机名称,作为一个eureka的启动实例，可以通过这个hostname进行访问(www.eureka01.com),必须配合网路配置(dns服务器,hosts文件)，如果配置为localhost可以不加。

eureka.instance.hostname=localhost

2）registerWithEureka默认为true，指示该实例是否注册到eureka注册中心，设置false, 表示自己就是eureka服务中心

eureka注册中心本身启动时既具备服务端角色,也可以是本身服务端的客户端。

eureka.**client.**registerWithEureka=false

3）fetch-registry: 默认值为true，指示此客户端是否应该从eureka获取eureka注册表信息，设置为false，表示自己就是注册中心，不用去注册中心获取其他服务的地址

eureka.client.fetchRegistry=false

4）对外暴露的注册地址(接口)

eureka.client.serviceUrl.defaultZone=http://localhost:8761**/eureka**

server.port=8761

#eureka config

eureka.instance.hostname=localhost

eureka.client.registerWithEureka=false

eureka.client.fetchRegistry=false

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

* **启动类编写**

**package** cn.edu.scnu;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer//启动eurekaServer的注解

**public** **class** StarterEurekaServer {

**public** **static** **void** main(String[] args) {

//通过run方法讲反射信息发送给初始化过程

//启动过程加载eureka相关依赖资源，自动配置eureka进程启动

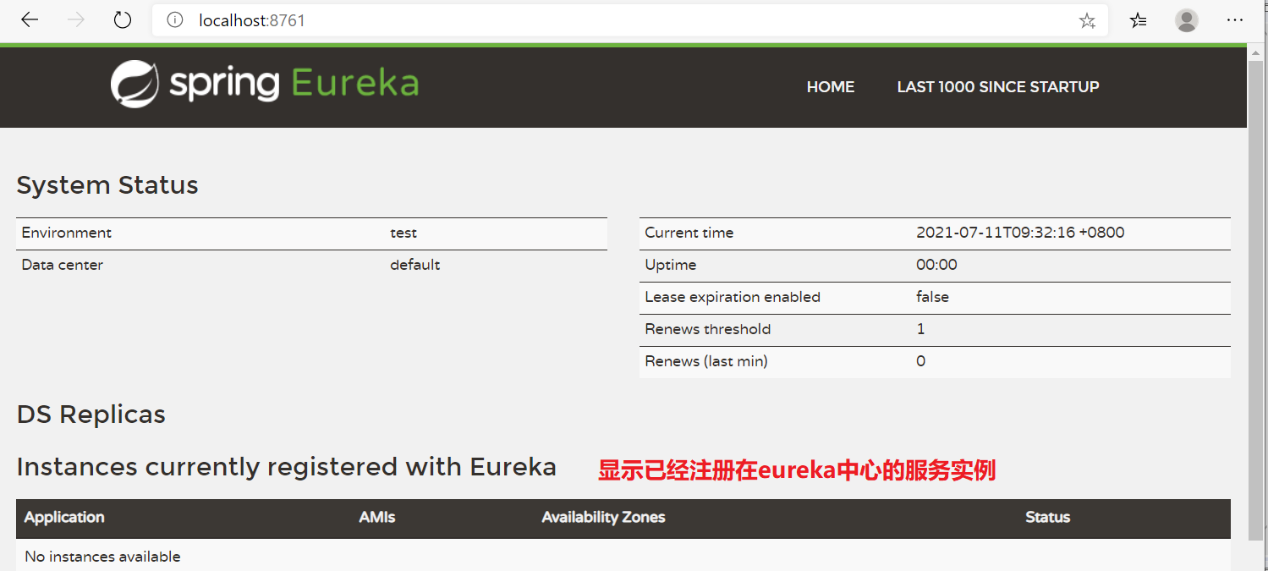
SpringApplication.*run*(StarterEurekaServer.**class**, args);

}

}

## 2.3启动第一个server服务端，观察控制台页面

访问http://localhost:8761/



## 2.4搭建3个 eureka的客户端

**(每个微服务都是eureka客户端)**

* **quickstart**
* **pom文件(依赖spring-cloud-starter-eureka)**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.9.RELEASE</version>

</parent>

<!--Eureka相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<!--springcloud相关依赖 -->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* **properties**

1）3个客户端的端口号分别为9001,9002,9003

2）spring项目,进程的应用名称,在eureka注册阶段携带到注册中心的服务名称. 3个客户端的应用名称都是： **service01**

spring.application.name=**service01**

3）指向注册中心url地址

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

server.port=9001

#eureka client config

spring.application.name=service01

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

图示

描述已自动生成

* **启动类(@EnableEurekaClient注解)**

**package** cn.edu.scnu;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

**public** **class** StarterEurekaClient01 {

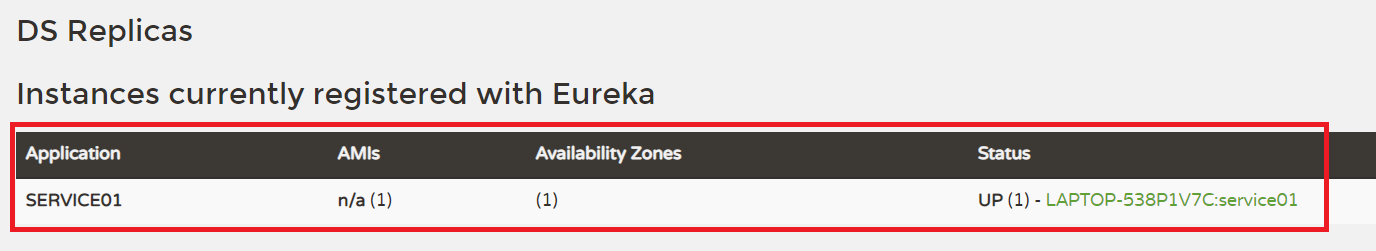
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(StarterEurekaClient01.**class**, args);

}

}

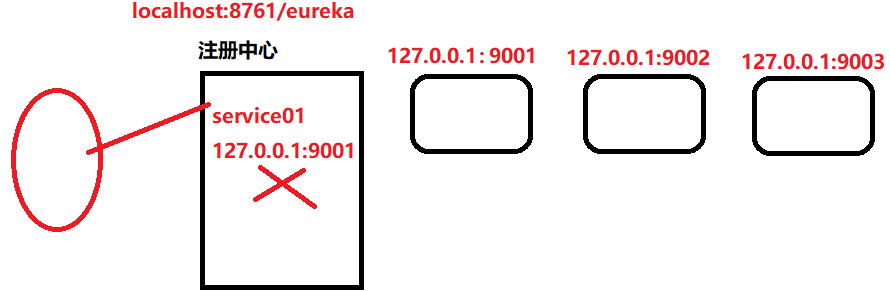
先启动StarterEurekaServer，在启动StarterEurekaClient01



## 2.5注册的逻辑.eureka管理数据

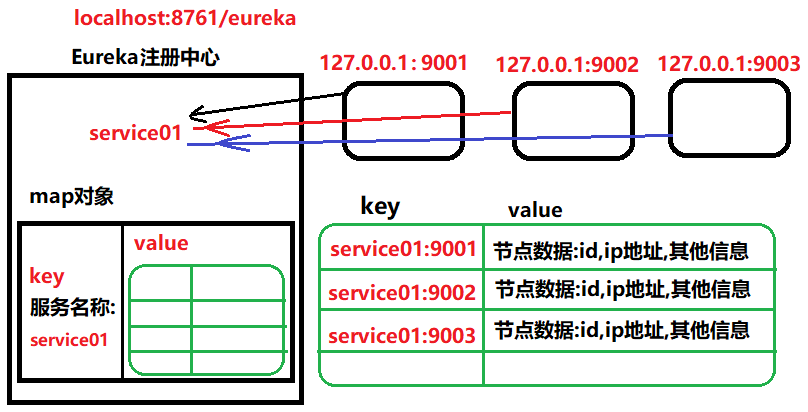
* **续约,和剔除**

启动的每一个eureka客户端进程.都会根据配置到eureka注册中心进行注册,注册之后, eureka客户端（服务）每30秒给注册中心发送一个心跳检测(续约) ,注册中心每60秒检测一次所有的服务续约,一旦发现服务超过90秒没有发送心跳检测的话,将会把服务对应的实例节点剔除;



* **Eureka注册信息结构**

每一个服务的提供者注册到服务端时,访问调用localhost:8761/eureka发送请求.并且携带一些信息,Eureka服务端将这些信息保管在一个双层map中

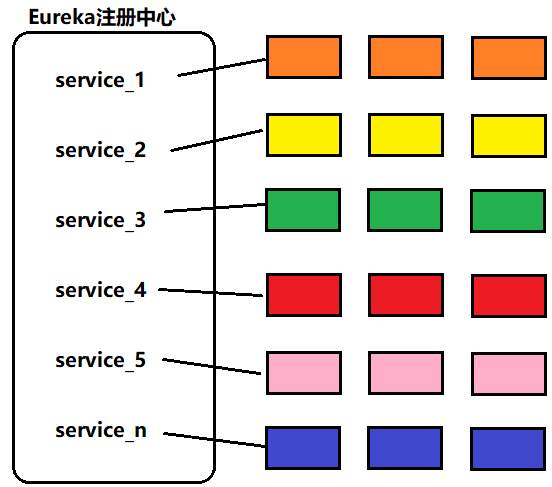


* **保护机制**

eureka注册中心.提供一种保护微服务集群注册服务信息的机制,防止由于网络波动导致大量的注册心跳检测发送未成功,造成的误剔除,给集群,给微服务带来的巨大伤害,所出现的一种机制--保护机制

当整个集群15%以上同一时间检测到超过90秒未发送心跳.将会开启保护机制,一旦开启,所有的服务注册信息,在eureka中心不会进行任何剔除;默认情况保护机制是开启的。

eureka.server.enable-self-preservation=false



## 2.6在测试案例的eureka-client-01中添加一些测试访问

* **功能需求:编写一个请求接受,返回当前客户端的端口号**

**package** cn.edu.scnu.controller;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** HelloCotroller {

//@value注解，读取配置文件的属性

@Value("${server.port}")

**private** String port;

@RequestMapping("hello")

**public** String sayHello(String name) {

**return** "hello " + name + ",i am from " + port;

}

}

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

* **将eureka-client-01拷贝成****eureka-client-02，eureka-client-03**

将eureka-client-01拷贝成eureka-client-02，eureka-client-03

修改启动类名称为02,03。修改端口号

将拷贝的工程启动查看eureka注册中心

# 3.RIBBON

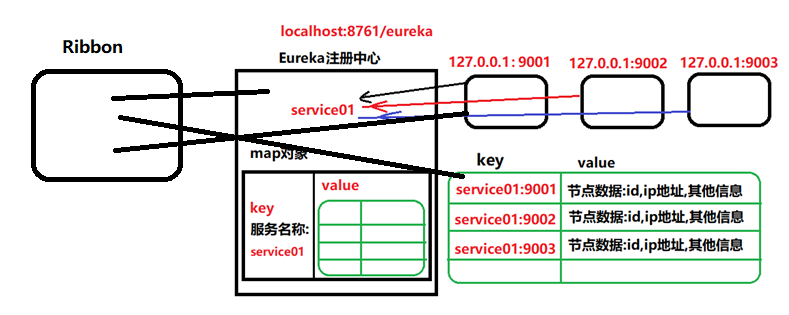
## 3.1介绍(实现内部服务调用另外一个服务的客户端)

在springcloud中,提供一个客户端组件,用来**负载均衡**的访问某个服务;

Ribbon启动之后,作为eureka的客户端到eureka注册中心实现抓取服务端的服务提供者列表list(每30秒重新抓取一次),这样就可以实现动态的对服务负载均衡访问。

绕过了nginx的静态文件配置.修改环境重启nginx;

这个客户端的访问服务的功能，需要依赖RestTemplate，发起http请求，请求的是服务名称)



## 3.2搭建一个ribbon客户端

* **quickstart**
* **pom文件(导入springcloud资源,依赖多了一个ribbon的组件,eureka客户端组件)**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.9.RELEASE</version>

</parent>

<!--Eureka相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<!--Ribbon相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-ribbon</artifactId>

</dependency>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* **properties配置文件**

server.port=9004

#eureka client config

spring.application.name=service02

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

* **启动类(实现eureka客户端功能.注册发现)**

加载一个能够实现ribbon负载均衡访问的对象RestTemplate(基于ribbon环境的)：

**package** cn.edu.scnu;

**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

@EnableEurekaClient

**public** **class** StarterRibbonClient {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(StarterRibbonClient.**class**, args);

}

// 想在Ribbon客户端发起一个服务的调用，则要创建一个RestTemplate对象

//如果光是new一个RestTemplate对象，

//则无法实现通过服务名称调用服务的功能，只能具体访问某个服务器，

//必须加上@LoadBalanced注解，生成一个具备ribbon拦截功能的restTemplate

@Bean

@LoadBalanced//加了该注解，就开启负载均衡的功能

//加了注解后，在创建RestTemplate后会对它实现监听,通过它发起请求做到一个拦截

// 会根据负载均衡策略对象(IRule接口的实现类)

// 实现访问服务的负载均衡方式 默认**roundRobin轮询**

**public** RestTemplate initRestTemplate() {

**return** **new** RestTemplate();

}

}

@LoadBalanced注解的作用：开启负载均衡，只需要在RestTemplate的bean上再添加一个@LoadBalanced注解。

* **测试类controller**

**package** cn.edu.scnu.controller;

**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** HelloController {

@Autowired

**private** RestTemplate restTemplate;

//接收当前系统的请求地址hello

@RequestMapping("hello")

**public** String sayHello(String name){

String respongestr=restTemplate.getForObject("http://service01/hello?name="+name, String.**class**);

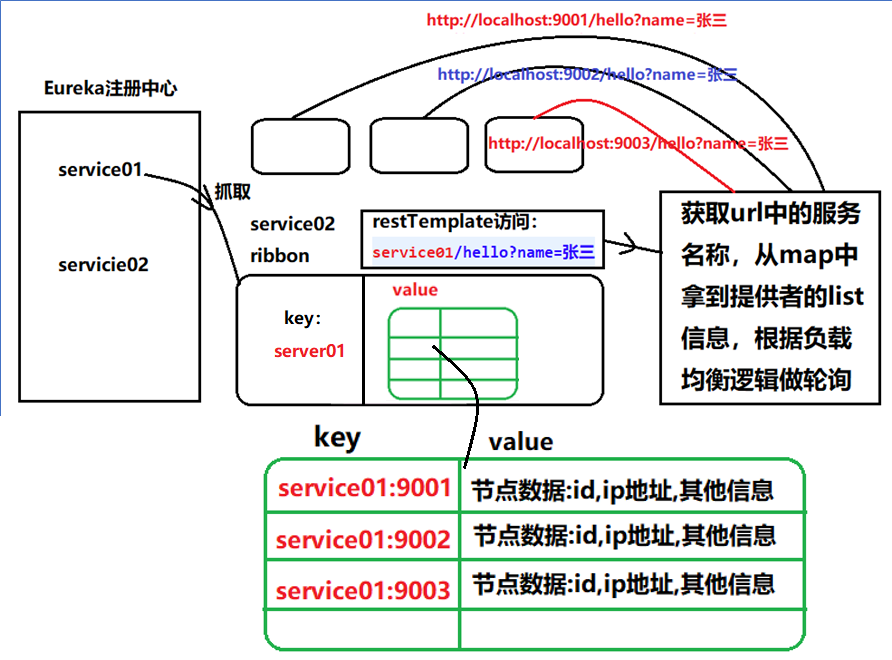
**return** "来自于Ribbon："+respongestr;

}

}

## 3.4Ribbon+RestTemplate实现负载均衡的原理

* ribbon启动 eureka客户端(注册,**发现**)
* ribbon客户端启动时,会根据eureka客户端角色到注册中心抓取所有的服务列表.将抓取的数据保存在内存的map
* restTemplate在发起请求之前做拦截.替换,根据请求地址中的服务名称,找到map中key,拿到提供者list,根据负载均衡逻辑(默认轮询)进行轮询的负载均衡访问



在LoadBalancerBuider做拦截

## 3.5其他负载均衡逻辑

* 其他负载均衡irule实现类

除了轮询的机制,还提供很多.比如随机.比如权重

|  |  |
| --- | --- |
| IRULE实现类 | 功能 |
| RamdomRule | 实现拦截拼接**服务提供者**信息时,采用**随机**的选取方式 |
| WeightedResponseTimeRule | 根据每个节点平均响应时间做动态的**权重值**变化,响应速度越快,权重值越高(负载访问占比越高) |

* 当前的环境是根据**生成的bean对象**来加载负载均衡策略(缺省值就是RoundRobin的对象)

只需要在配置类中（启动类就是配置类）.利用@Bean创建当前想使用的对象即可

// 创建一个自定义负载均衡策略

@Bean

**public** IRule myRule() {

**return** **new** RandomRule();

}

重启Ribbon项目，测试随机负载均衡。

## 3.6动态添加一个service01的项目

将eureka-client01,02,03其中之一复制,完成服务提供者集群的扩容(缩容)操作,实现服务动态维护.不需要修改任何内容

# 4.根据eureka和ribbon,修改order-user

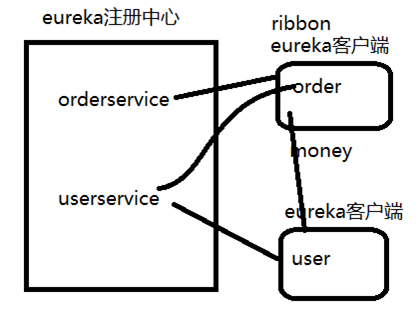
## 4.1角色结构

<http://orderservice/order/pay?orderId=a>

http://www.ssm.cn/order/pay?orderId=a

<http://userservice/user/query/point?userId=1>

http://www.ssm.cn/user/query/point?userId=1



## 4.2动手修改order系统

* **eureka注册中心使用已有的测试案例**
* **order系统添加springcloud相关依赖资源(ribbon,eureka客户端依赖)**
  + 添加导入的springcloud资源dependencyManagement

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* + eureka客户端角色,ribbon角色相关依赖

<!--Eureka相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<!--Ribbon相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-ribbon</artifactId>

</dependency>

* + 修改配置文件添加内容

服务名称=orderservice

注册中心的地址=eureka.client.serviceUrl.defaultZone

spring.application.name=orderservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

* + 启动类添加注解@EnableEurekaClient

**package** cn.edu.scnu;

**import** org.mybatis.spring.annotation.MapperScan;

**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

@MapperScan("cn.edu.scnu.mapper")

**@EnableEurekaClient**

**public** **class** StarterOrder {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(StarterOrder.**class**, args);

}

@Bean

**@LoadBalanced**

**public** RestTemplate intiRestTemplate(){

**return** **new** RestTemplate();

}

}

* + 修改service中发起请求的restTemplate代码的url参数地址

**try** {

Integer a = client.getForObject("http://userservice/user/updatePoint?o="

+order.getOrderMoney()+"&userId="+order.getUserId(), Integer.**class**);

} **catch** (Exception e) {

e.printStackTrace();

}

## 4.3动手修改user系统

* + 使用已有的eureka注册中心
  + pom文件添加注入的springcloud资源依赖

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* + eureka客户端角色

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

* + 添加eureka客户端相关配置properties
  1. 服务名称
  2. 注册中心访问接口地址

spring.application.name=userservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

* + 启动类添加@EnableEurekaClient
  + 补充知识点:可以通过,某个工程打包的jar在同一个服务器,不同的端口启动

java -jar \*\*.jar --server.port=8091

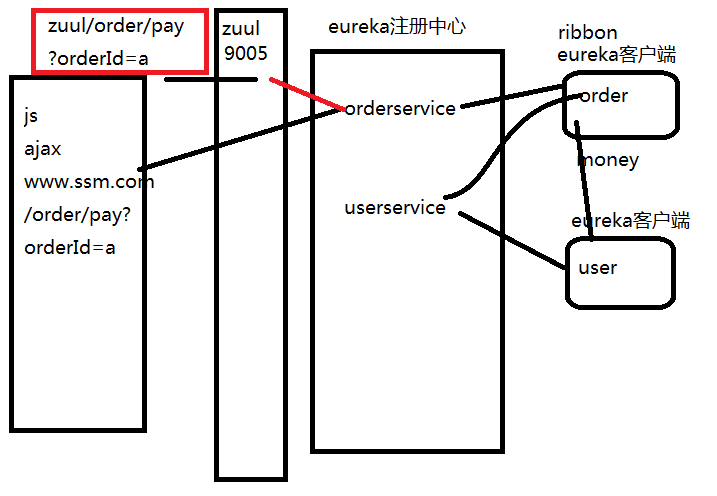
# 5.zuul网关(gateway)路由过滤

## 5.1介绍zuul网关

Zuul网关是整个微服务框架搭建完毕之后的集群中唯一**对外界提供接口访问**的角色,ribbon,eureka等等不能够将请求地址,接口地址对外暴露,一旦暴露就会出现被攻击的危险;

## 5.2路由功能入门案例

zuul底层实现了ribbon+restTemplate的编码逻辑的,可以在路由的模式下, 在zuul内部启动ribbon启动restTemplate,当对外暴露的接口地址被访问时.转发到一个真实的服务。



* **搭建一个zuul网关工程(没有页面)quickstart**

搭建之前，将前面扩容的eureka-client05删除

* **配置pom文件(eureka客户端,zuul网关插件)**
  1. Springboot

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.9.RELEASE</version>

</parent>

* 1. 引入依赖zuul和eureka客户端组件

<!--Eureka相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<!--zuul相关依赖 -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

* 1. 导入springcloud 爱德华资源

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

* **配置properties文件，定义路由规则(对外暴露的请求地址,访问ribbon测试工程)**

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

server.port=9005

spring.application.name=serviceZuul

zuul.routes.api-a.path=/api-a/\*\*

zuul.routes.api-a.serviceId=service01

zuul.routes.api-b.path=/api-b/\*\*

zuul.routes.api-b.serviceId=service02

zuul.routes.orders.path=/orders/\*\*

zuul.routes.orders.serviceId=orderservice

zuul.routes.users.path=/users/\*\*

zuul.routes.users.serviceId=userservice

定义了路由的规则;路由地址匹配逻辑

1)**zuul.routes.api-a.path**:定义一个与访问api-a相关服务配置的path的映射地址 /api-a/\*\*,请求的路径(url)中以/api-a开始的多级路径匹配。

例如：

localhost:9005/api-a/order/pay 可以匹配到

localhost:9005/api-h/order/pay 匹配不到

一旦当前请求和某一个路由匹配成功,将会根据zuul.routes.XX.path中的,寻找当前properties中一个zuul.routes.XX.serviceId=service01,从而找到对应的访问服务(ribbon+restTemplate访问服务获取响应)

例如：

localhost:9005/api-a/order/pay

寻找到：

localhost:9005/service01/order/pay

2）匹配规则:? 、\* 、\*\*

|  |  |
| --- | --- |
| 符号 | 匹配规则 |
| ? | 匹配一个字符,例如/api-a/? 只能匹配到类似/api-a/a、/api-a/b |
| \* | 匹配一个一级路径字符串,/api-a/\*,匹配到/api-a/a,而/api-a/a/haha,/api-a/a/b则匹配不到了 |
| \*\* | 只要是起始匹配成功.后续的多级,多字符串匹配都通配/api-a/\*\*, |

* **启动类(@EnableZuulProxy)**

**package** cn.edu.scnu;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication

**@EnableZuulProxy**

**public** **class** StarterZuul {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(StarterZuul.**class**, args);

}

}

* **路由规则,对应我们目前测试的环境**

spring.application.name=serviceZuul

zuul.routes.api-a.path=/api-a/\*\*

zuul.routes.api-a.serviceId=service01

zuul.routes.api-b.path=/api-b/\*\*

zuul.routes.api-b.serviceId=service02

* **路由的逻辑**

起始:http://localhost:9005/api-a/hello?name=张三

|

第二步:http://service01/hello?name=张三

|

第三步:http://localhost:9001(9002/9003)/hello?name=张三

起始:http://localhost:9005/**api-b**/hello?name=张三

|

第二步:http://service02/hello?name=张三

|

第三步:http://localhost:9004/hello?name=张三

|

第四步:http://service01/hello?name=张三

|

第五步:http://localhost:9001(9002/9003)/hello?name=张三

zuul对外暴露的接口文件

|  |  |
| --- | --- |
| 请求地址 | localhost:9005/api-a/hello?name=haha |
| 请求方式 | get |
| 请求参数 | String name |
| 返回数据 | String response (说hello) |

|  |  |
| --- | --- |
| 请求地址 | localhost:9005/api-b/hello?name=xulaoshi |
| 请求方式 | get |
| 请求参数 | String name |
| 返回数据 | String response (说hello,标识RIBBON) |

|  |  |
| --- | --- |
| 请求地址 | localhost:9005/orders/order/pay?orderId=a |
| 请求方式 | get |
| 请求参数 | String orderId |
| 返回数据 | 成功1,失败0,订单支付,调用用户服务做积分 |

|  |  |
| --- | --- |
| 请求地址 | localhost:9005/users/user/query/point?userId=1 |
| 请求方式 | get |
| 请求参数 | String userId |
| 返回数据 | user对象的String userJson 做ajax解析 |

将nginx配置成访问网关暴露接口的匹配地址,js,页面就可以springcloud搭建的微服务集群访问高可用的分布式系统了

server {

                 listen 80;

server\_name [www.ssm.com](http://www.ssm.com);

location /{

#相对nginx根目录

root orderuser;

index index.html;

}

location **/user** {

proxy\_pass <http://127.0.0.1:9005/users/user>;

add\_header 'Access-Control-Allow-Origin' '\*';

add\_header 'Access-Control-Allow-Credentials' 'true';

}

location /order {

proxy\_pass <http://127.0.0.1:9005/orders/order>;

add\_header 'Access-Control-Allow-Origin' '\*';

add\_header 'Access-Control-Allow-Credentials' 'true';

}

}

以我的积分查询为例,从浏览器发起请求,到最终响应过程

浏览器发起的js请求地址

<http://www.ssm.cn/user/query/poinst?userId=1>

|nginx判断端口80.域名www.ssm.com

|以/user起始,转向zuul关工程地址

<http://localhost:9005/users/user/query/point?userId=1>

|路由规则/users/\*\* 匹配上

|转向内部服务userservice

[http://userservice/user/query/point?userId=1](http://userdemoservice/user/query/point?userId=1)

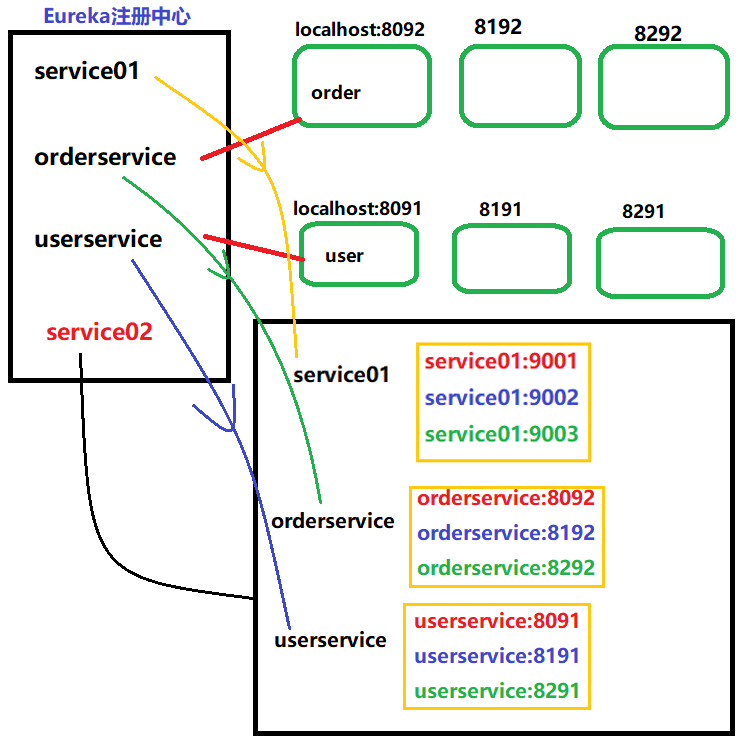
|ribbon的restTemplate发起请求,拦截

|userservice拦截拼接一个后端服务提供者的集群list做负载均衡

<http://localhost:8091/user/query/point?userId=1>

**问题:eureka注册中心高可用?**

两个eureka注册中心,相互注册,相互发现,同步信息;



# 6.eureka高可用集群

## 6.1.高可用集群的结构和同步

图示

描述已自动生成

## 6.2.搭建一个双机的eureka注册中心高可用

### 6.2.1搭建注册中心高可用

* 复制一个eureka注册中心(8762,8761)
* 修改properties配置内容
  + spring.application.name=eureka-server
  + 注册与发现开启,registry和fetch=true
  + 服务端互相注册发现必须配置 ip优先的内容

eureka.instance.preferIpAddress=true

* + 相互注册与发现,serviceUrl.defaultZone的值互相指向8761-->8762,8762-->8761

server.port=8761

#eureka config

spring.application.name=eureka-server

eureka.instance.hostname=localhost

eureka.client.registerWithEureka=true

eureka.client.fetchRegistry=true

eureka.instance.preferIpAddress=true

eureka.client.serviceUrl.defaultZone=http://localhost:8762/eureka

eureka.server.enable-seft-preservation=false

server.port=8762

#eureka config

spring.application.name=eureka-server

eureka.instance.hostname=localhost

eureka.client.registerWithEureka=true

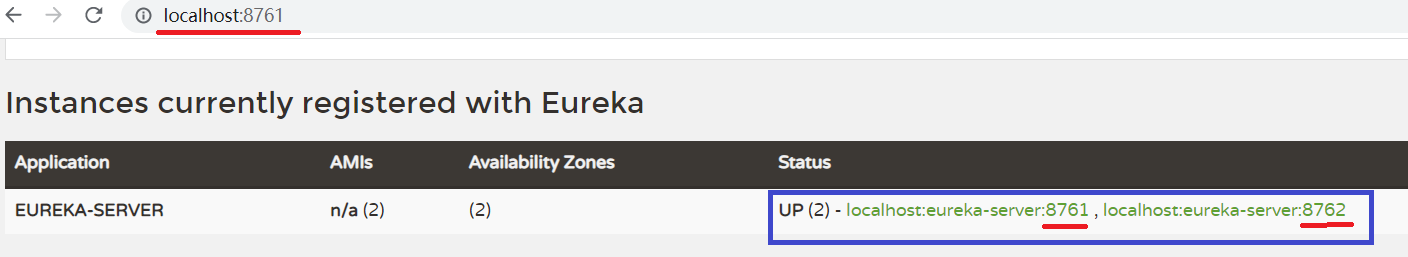
eureka.client.fetchRegistry=true

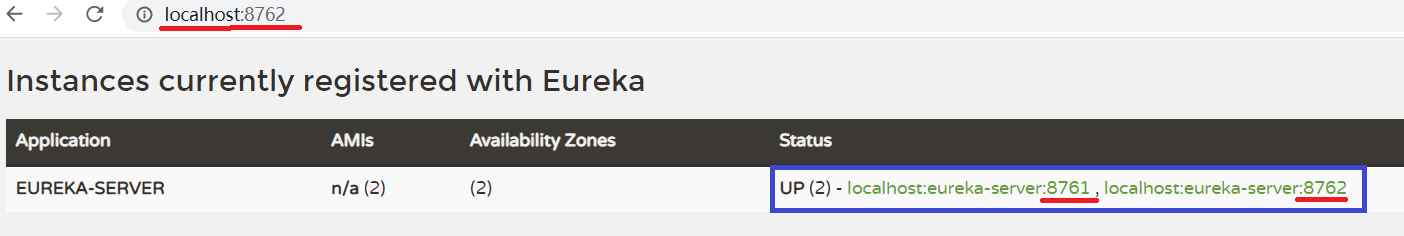
eureka.instance.preferIpAddress=true

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

eureka.server.enable-seft-preservation=false

* 观察页面

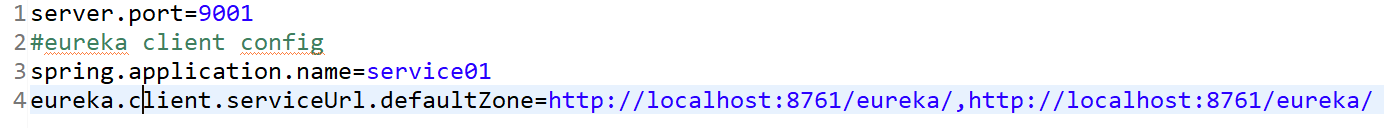




### 6.2.2eureka客户端配置

以任何一个测试案例的eureka客户端为例(EurekaClient)配置指向的eureka注册中心地址zone使用list结构(逗号隔开多个eureka服务端)

eureka.client.serviceUrl.defaultZone=<http://localhost:8761/eureka/,http://localhost:8762/eureka/>



8761：

图形用户界面, 文本, 应用程序

描述已自动生成

8762：

图形用户界面, 文本, 应用程序

描述已自动生成

全部项目都启动：

图形用户界面, 应用程序

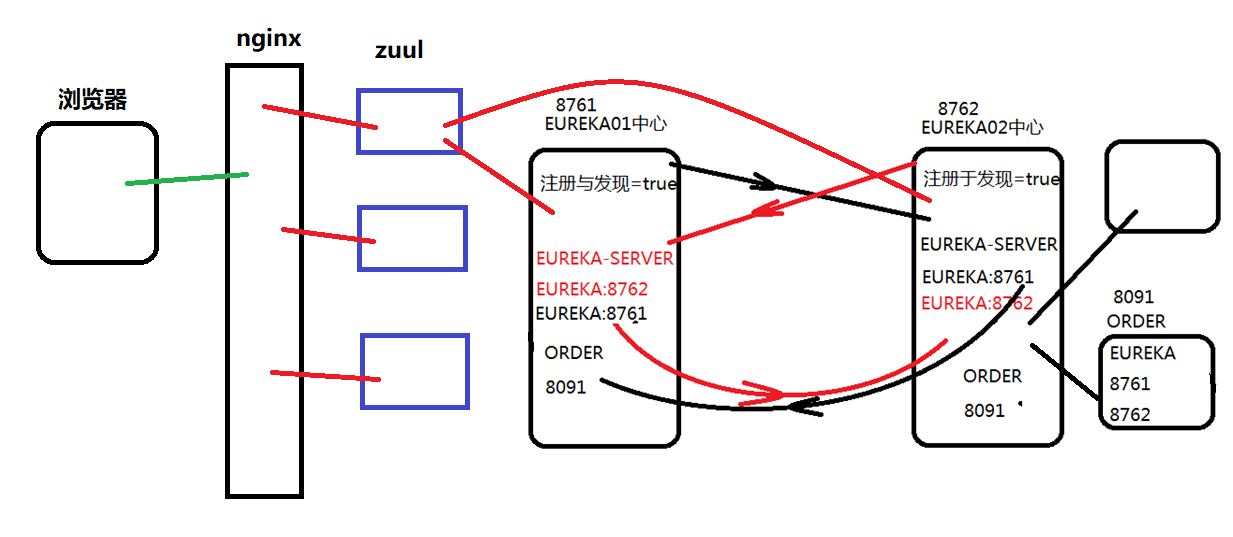
中度可信度描述已自动生成

图形用户界面, 应用程序

描述已自动生成

## 6.3 搭建zuul网关集群，实现zuul网关高可用

前面我们搭建了Eureka高可用集群，所有访问都要先经过zuul网关，所以zuul网关也要搭建集群。



6.3.1 添加springboot对maven插件的支持资源

springboot专门为maven提供了打包springboot的所有插件简化资源;

在在zuul网关工程的pom文件中添加所有插件简化资源

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

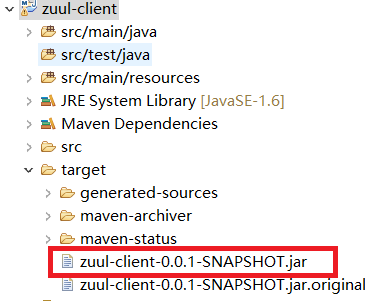
</plugins>

</build>

6.3.2 打jar包

选中zuul-client，先 “右键->run as -> Maven clean”，

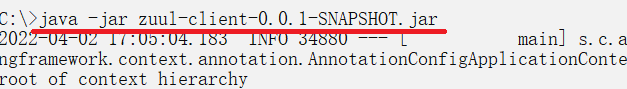
再“右键->run as -> Maven install”，生成jar包：



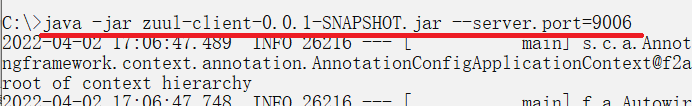
6.3.3 运行代码jar包

1.运行在properties配置的端口号（9005）

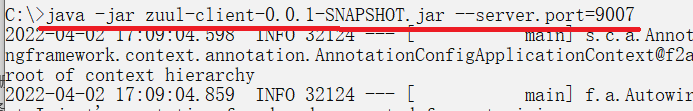
将jar包复制到c盘下，运行，不指定端口号表示使用properties



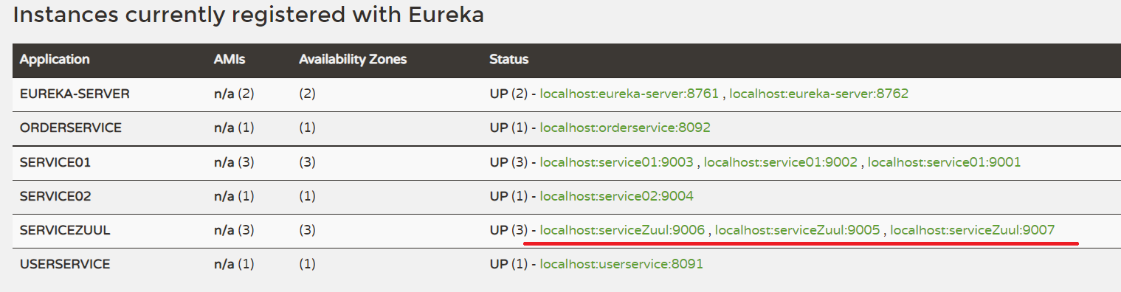
2.运行在 9006端口号



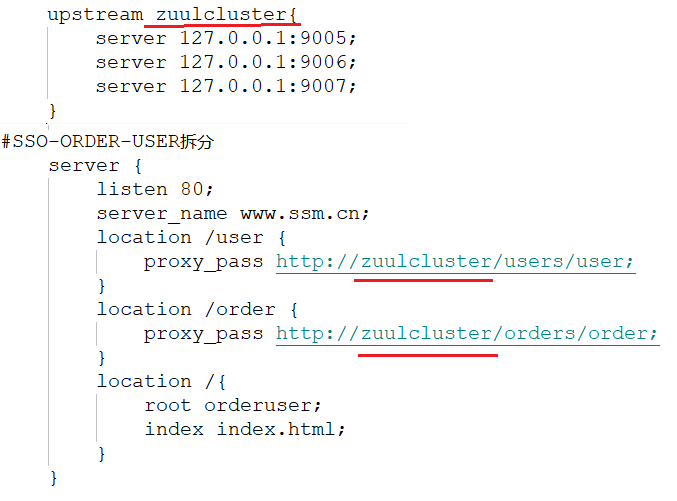
3. 运行在 9007端口号



查看eureka:



4.修改nginx配置



配置完毕，访问 www.ssm.cn