# 一、Eureka

Eureka最重要的功能服务发现即Eureka服务端可以注册Eureka实例，Eureka客户端可以发现服务端并进行注册(生产消费模式中的生产者，对应的消费者就是调用Eureka客户端注册服务的应用)

Eureka服务端application.properties配置如下

*#本服务占用端口***server.port**=**8761***#下面防止自己注册自己start***eureka.client.registerWithEureka**=**false   
eureka.client.fetchRegistry**=**false***#上面防止自己注册自己end***eureka.server.waitTimeInMsWhenSyncEmpty**=**0  
spring.cloud.config.discovery.enabled**=**false**

服务端启动类代码如下，注意必须添加EurekaServer的注解，这个注解说明服务器开始Eureka服务端服务

@EnableEurekaServer  
@SpringBootApplication  
**public class** EurakaApplication {

**public static void** main(String[] args) {  
 *run*(EurakaApplication.**class**, args);  
 }  
  
}

Eueka客户端application.properties配置如下

**server.port**=**8762  
spring.application.name**=**eureka-client***#注册服务器的地址***eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/**

Eueka客户端启动类代码如下，注意必须添加EurekaClient的注解，这个注解说明服务器启动Eureka客户端服务

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

# 二、Eureka负载均衡

Ribbon用于客户端(本事EurekaClient端，但是他是一个消费者)负载均衡，通过从EurekaClient端(即生产者)获取的生产者列表(当然也可以不在生产者中获取，而在application.properties中配置)。

1. Ribbon（RestTemplate）+EurekaClient获取生产者的生产者列表实现负载均衡

注册服务器配置文件即启动类同上面一样，服务提供者(这个服务后文以s1代指)代码如下

@EnableEurekaClient  
@SpringBootApplication  
@RestController  
**public class** RibbonClientApplication {  
 @Value(**"${server.port}"**)  
 **private** String **port**;  
 @Value(**"${spring.application.name}"**)  
 **private** String **applicationName**;  
 @RequestMapping(**"/hi/{name}"**)  
 **public** String getHiName(@PathVariable(**"name"**)String name){  
  
 **return "applicationName:"**+**applicationName**+**",port:"**+**port**;  
 }  
 **public static void** main(String[] args) {  
  
 *run*(RibbonClientApplication.**class**, args);  
 }  
  
}

在这里面我们新增了测试的RestAPI，用于测试，由于系统要测试负载，所以同一个应用要启动多次，在代码中采用了多配置文件方式，通过添加启动参数来区分

application-pro1.properties

**server.port**=**8762  
spring.application.name**=**eureka-client-producer***#eureka.instance.instance-id=eureka-client-producer  
#注册服Ribbon务器的地址***eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/**

application-pro1.properties

**server.port**=**8763  
spring.application.name**=**eureka-client-producer***#eureka.instance.instance-id=eureka-client-producer  
#注册服Ribbon务器的地址***eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/**

注意这两个配置的除了服务端口不一致，其他都一样，这里一定要注意spring.application.name这个参数配置，两个采用相同的配置，在注册服务器中会以一个域名对应两个ip或者端口的形式出现，这样在消费者才可以利用spring.application.name这个参数值作为域名实现负载均衡。启动服务的时候可以添加以下参数

java -jar xxx.jar --spring.profiles.active=pro1

java -jar xxx.jar --spring.profiles.active=pro2

系统会分别按照application-pro1.properties和application-pro2.properties来启动服务

消费者代码如下：

@EnableEurekaClient  
@SpringBootApplication  
@RestController  
**public class** RibbonClientCustomApplication {  
 @Autowired  
 RibbonTestI **ribbonTestI**;  
 @RequestMapping(**"/hi/{name}"**)  
 **public** String getTest(@PathVariable(**"name"**)String name){  
 **return "你好："**+ name+**",您访问的地址地址信息如下"**+**ribbonTestI**.doStuff(**"/hi/"**+name);  
 }  
  
 **public static void** main(String[] args) {  
 *run*(RibbonClientCustomApplication.**class**, args);  
 }  
  
}

在消费者的启动类中增加RestApi作测试使用，在controller中调用了RibbonTestI（RibbonTestClass）接口，在这个接口实现了调用生产者的功能代码。

@Service  
**public class** RibbonTestClass **implements** RibbonTestI{  
 @Autowired  
 *//RestTemplate的@LoadBalanced注解不能同LoadBalancerClient一起使用，否则会失效或者产生负载均衡冲突  
 //当时LoadBalanced注解，RestTemplate会自动采用服务名转化ip的流程，所以会产生问题* **private** LoadBalancerClient **loadBalancer**;  
 @Autowired  
 **private** RestTemplate **restTemplate**;  
 **public** String doStuff(String path) {

*String result = restTemplate.getForObject("http://eureka-client-producer/"+path, String.class);* **return** result;  
 }  
  
}

@Configuration  
*//本配置累主要是为了控制Ribbon***public class** DefaultRibbonConfig {  
 @Bean  
@LoadBalanced**public** RestTemplate restTemplate(){  
 */\* loadBalance这个注解加上之后，这个注解有3件事情要处理。*

*第一件就是从负载均衡器中选一个对应的服务实例，那有的人就会问为什么从负载均衡器中挑选，原因很明显就是，所有的服务名实例都放在负载均衡器中的serverlist。*

*第二件事情就是从第一件事情挑选的实例中去请求内容。*

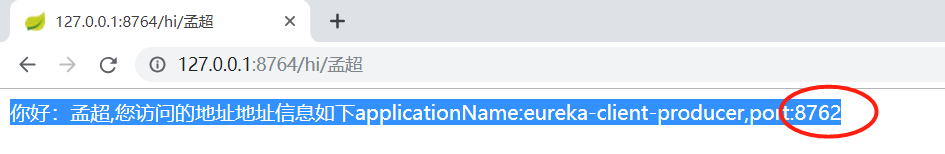
*第三件事情就是由服务名转为真正使用的ip地址\*/* **return new** RestTemplate();  
 }  
}

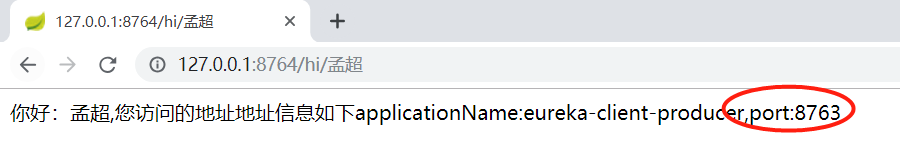
在代码中可以看出消费者通过RestTemplate调用了生产者提供的代码，这里要注意在getForObject中的url是通过域名的方式进行访问的，这个域名就是在消费者中spring.application.name配置的应用服务器名。还需要注意一点，在RestTemplate的bean上增加@LoadBalanced注解，添加这个主解之后，RestTemplate才能使用负载均衡。还要注意如果在RestTemplate使用这个注解之后，就不要通过LoadBalancerClient来获取生产者实例，用这个实例获取对应的ip和端口访问生产者，因为会产生冲突。

消费者的application.properties配置如下

**server.port**=**8764  
eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/  
spring.application.name**=**ribbonClientCustomApplication**

准备就绪，启动服务器后，访问127.0.0.1:8764/hi/孟超，可以看到消费者轮询调用生产者，这是由于系统默认的负责均衡策略是轮询策略。





1. Ribbon（RestTemplate）+ LoadBalancerClient+EurekaClient实现负载均衡

@Service  
**public class** RibbonTestClass **implements** RibbonTestI{  
 @Autowired  
 *//RestTemplate的@LoadBalanced注解不能同LoadBalancerClient一起使用，否则会失效或者产生负载均衡冲突  
 //当时LoadBalanced注解，RestTemplate会自动采用服务名转化ip的流程，所以会产生问题* **private** LoadBalancerClient **loadBalancer**;  
 @Autowired  
 **private** RestTemplate **restTemplate**;  
 **public** String doStuff(String path) { *//获取消费者内负载均衡的列表专用start* ServiceInstance instance = **loadBalancer**.choose(**"eureka-client-producer"**);  
 URI storesUri = URI.*create*(String.*format*(**"http://%s:%s"**, instance.getHost(), instance.getPort()));  
 System.***out***.println(storesUri.toString());  
 String service\_id = instance.getServiceId();  
 String result = **restTemplate**.getForObject(storesUri.toString()+**"/"**+path, String.**class**);  
 *//获取消费者内负载均衡的列表专用end* **return** result;  
 }  
  
}

@Configuration  
*//本配置累主要是为了控制Ribbon***public class** DefaultRibbonConfig {  
 @Bean**public** RestTemplate restTemplate(){  
 */\* loadBalance这个注解加上之后，这个注解有3件事情要处理。  
  
    第一件就是从负载均衡器中选一个对应的服务实例，那有的人就会问为什么从负载均衡器中挑选，原因很明显就是，所有的服务名实例都放在负载均衡器中的serverlist。  
  
    第二件事情就是从第一件事情挑选的实例中去请求内容。  
  
    第三件事情就是由服务名转为真正使用的ip地址\*/* **return new** RestTemplate();  
 }  
}

在使用LoadBalancerClient做负载均衡时，修改上面两个类即可，取消RestTemplate中的@LoadBalanced注解(为什么取消上面讲过原因)；利用LoadBalancer获取获取生产者实例（在调用请求的时候，也要注意到，采用的是ip+port的方式，而不再是域名方式）。LoadBalancer.choose方法的参数为生产者的spring.application.name属性。

1. Ribbon（RestTemplate）+ LoadBalancerClient+EurekaClient利用消费者本身配置的生产者列表实现负载均衡(不依赖eureka实现Ribbon负载均衡)

@Service  
**public class** RibbonTestClass **implements** RibbonTestI{  
 @Autowired  
 *//RestTemplate的@LoadBalanced注解不能同LoadBalancerClient一起使用，否则会失效或者产生负载均衡冲突  
 //当时LoadBalanced注解，RestTemplate会自动采用服务名转化ip的流程，所以会产生问题* **private** LoadBalancerClient **loadBalancer**;  
 @Autowired  
 **private** RestTemplate **restTemplate**;  
 **public** String doStuff(String path) {  
 *//本地缓存负载列表start* ServiceInstance instance = **loadBalancer**.choose(**"stores"**);  
 URI storesUri = URI.*create*(String.*format*(**"http://%s:%s"**, instance.getHost(), instance.getPort()));  
 System.***out***.println(storesUri.toString());  
 String service\_id = instance.getServiceId();  
 String result = **restTemplate**.getForObject(storesUri.toString()+**"/"**+path, String.**class**)  
 *//本地缓存负载列表end  
 //获取消费者内负载均衡的列表专用start  
 /\* ServiceInstance instance = loadBalancer.choose("eureka-client-producer");  
 URI storesUri = URI.create(String.format("http://%s:%s", instance.getHost(), instance.getPort()));  
 System.out.println(storesUri.toString());  
 String service\_id = instance.getServiceId();  
 String result = restTemplate.getForObject(storesUri.toString()+"/"+path, String.class)\*/*;  
 *//获取消费者内负载均衡的列表专用end  
// String result = restTemplate.getForObject("http://eureka-client-producer/"+path, String.class);* **return** result;  
 }  
  
}

配置文件如下：

**stores.ribbon.listOfServers**=**http://localhost:8762,http://localhost:8763***#禁止调用eurekaClient获取注册表,从stores中获取负载均衡的地址,此项为false时，需要同stores.ribbon.listOfServers配合存在***ribbon.eureka.enabled**=**false  
server.port**=**8764  
eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/  
spring.application.name**=**ribbonClientCustomApplication**

**ribbon.eureka.enabled**

ribbon.eureka.enabled=false参数配置使消费者不再从生产者获取生产者列表而是采用本地配置文件中的生产者列表，stores.ribbon.listOfServers配置的是生产者列表。

注意：在获取生产者实例的时候，loadBalancer.choose("stores")中参数stores是写死的，必须是这个，因为在配置文件中使用的是stores.ribbon.listOfServers=http://localhost:8762,http://localhost:8763，这是“<client>.ribbon.Ribboen参数”的形式，client已经写死了stores,所以获取的时候，也要用stores,如果在配置文件中将stores替换成eureka-client-producer的话,那么获取列表是必须为eureka-client-producer才可以。

注意:使用@RibbonClient(name="eureka-client-producer",configuration=RibbonClientConfig.class)相当于是针对特定的服务(spring.application.name=eureka-client-producer的应用)选择负载均衡。

1. 设置静态的服务列表，使用BaseLoadBalancer或其子类的BaseLoadBalancer.setServersList()方法。其中具体的列表可以在配置文件中配置，然使用${}的方式注入的对应的地方。

*// @Value("${stores.ribbon.listOfServers}")//在配置文件中获取* String **listOfServers** = **"http://localhost:8762,http://localhost:8763"**;  
  
 @Bean  
 **public** ServerList<Server> ribbonServerList() {  
 List<Server> list = Lists.*newArrayList*();  
 **if** (Strings.*isNotEmpty*(**listOfServers**)) {  
 **for** (String s: **listOfServers**.split(**","**)) {  
 list.add(**new** Server(s.trim()));  
 }  
 }  
 **return new** StaticServerList<Server>(list.get(0),list.get(1));  
 }

在使用的地方获取服务列表如下，需要注意的是在choose填写key的时候，无论填写那个key都可以获取到服务(需要查看代码才知道为什么会这样)。

@Service  
**public class** RibbonTestClass **implements** RibbonTestI{  
 @Autowired  
 *//RestTemplate的@LoadBalanced注解不能同LoadBalancerClient一起使用，否则会失效或者产生负载均衡冲突  
 //当时LoadBalanced注解，RestTemplate会自动采用服务名转化ip的流程，所以会产生问题* **private** LoadBalancerClient **loadBalancer**;@Autowired  
 **private** RestTemplate **restTemplate**;  
 **public** String doStuff(String path) {ServiceInstance instance = **loadBalancer**.choose(**"store"**);URI storesUri = URI.*create*(String.*format*(**"http://%s:%s"**, instance.getHost(), instance.getPort()));  
 System.***out***.println(storesUri.toString());  
 String service\_id = instance.getServiceId();  
 String result = **restTemplate**.getForObject(storesUri.toString()+**"/"**+path, String.**class**);

**return** result;  
 }  
  
}

1. 使用本地静态配置，实现负载均衡，主要是通过ConfigurationBasedServerList这个类实现，覆盖或者重写此类可以在配置文件或者使用@Ribbonclients两种方式实现

首先，继承ConfigurationBasedServerList类，

**public class** BazServiceList **extends** ConfigurationBasedServerList {  
 **public** BazServiceList() {  
 }  
 **public** BazServiceList(IClientConfig config) {  
 **super**.initWithNiwsConfig(config);  
 }  
}

其次，application.property配置如下**stores.ribbon.listOfServers**=**http://localhost:8762,http://localhost:8763***#stores.ribbon.NIWSServerListClassName=com.study.springcloud.BazServiceList***ribbon.eureka.enabled**=**false  
server.port**=**8764  
eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/  
spring.application.name**=**ribbonClientCustomApplication**

这里面配置注意两点，第一个stores.ribbon.listOfServers参数必须要存在，这个是配置的服务器列表，ribbon.eureka.enabled=false，防止消费者启用eureka从生产者获取服务器列表。

stores.ribbon.NIWSServerListClassName这个参数，配置系统技术服务器列表类，springboot启动的时候会获取这个参数值，并初始化对应的类，当然也可以不在这里配置，可以在@Ribbonclients引入的类中用下面的方式注入

@Bean*//本例用于***public** ServerList<Server> ribbonServerList(IClientConfig config) {  
 BazServiceList bl = **new** BazServiceList(config);  
 **return** bl;  
}

1. 从远程服务器获取服务器列表

首先，在@Ribbonclients引入的类中用下面的方式注

@Bean*//从远程服务器获取服务器列表，默认是采用DiscoveryEnabledNIWSServerList类***public** DiscoveryEnabledNIWSServerList discoveryEnabledNIWSServerList(IClientConfig config) {  
 DiyDiscoveryEnaledServerList bl = **new** DiyDiscoveryEnaledServerList();  
 **return** bl;  
}

DiyDiscoveryEnaledServerList这个类是自己写的，继承了从服务器获取服务提供者列表的默认类DiscoveryEnabledNIWSServerList  
**public class** DiyDiscoveryEnaledServerList **extends** DiscoveryEnabledNIWSServerList {  
}