SpringCloud之三

负载均衡ribbon

在微服务架构中，业务都会被拆分成一个独立的服务，服务与服务的通讯是基于http restful的。Spring cloud有两种服务调用方式，一种是ribbon+restTemplate，另一种是feign。在这一篇文章首先讲解下基于ribbon+rest。

一、ribbon简介

官方介绍如下图：

官网说明如下：

Ribbon

Ribbon is a client side IPC library that is battle-tested in cloud. It provides the following features

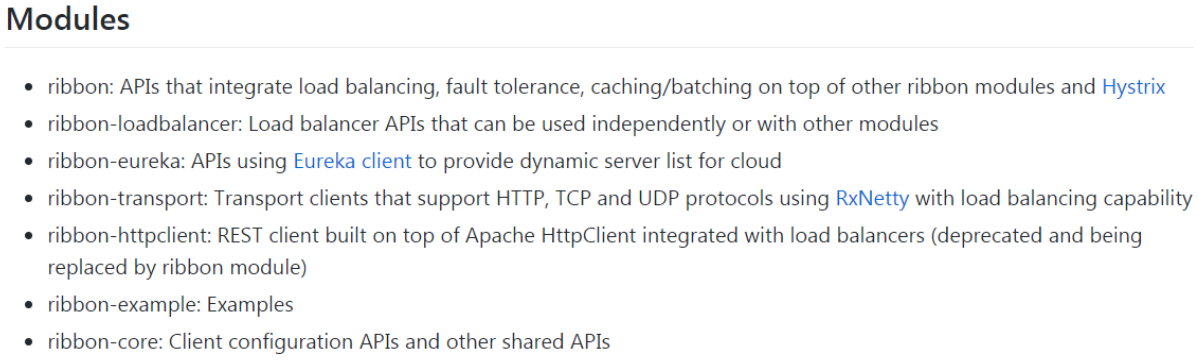
- Load balancing
- Fault tolerance
- Multiple protocol (HTTP, TCP, UDP) support in an asynchronous and reactive model
- Caching and batching

To get ribbon binaries, go to maven central. Here is an example to add dependency in Maven:

Ribbon是一个客户端IPC库，在云中经过实战测试。它提供以下功能

- 负载均衡
- 容错
- 异步和反应模型中的多协议（HTTP，TCP，UDP）支持
- 缓存和批处理

包含模块如下图



模块

ribbon：在其他功能区模块和Hystrix之上集成负载平衡，容错，缓存/批处理的API

ribbon-loadbalancer：负载均衡器API，可以单独使用，也可以与其他模块一起使用

ribbon-eureka：使用Eureka客户端为云提供动态服务器列表的API

ribbon-transport：使用具有负载平衡功能的RxNetty传输支持HTTP，TCP和UDP协议的客户端

ribbon-httpclient：构建在与负载均衡器集成的Apache HttpClient之上的REST客户端（不建议使用并由功能块模块替换）

ribbon-example：示例

ribbon-core：客户端配置API和其他共享API

更多说明可以查阅官网。https://github.com/Netflix/ribbon

ribbon 已经默认实现了这些配置bean：

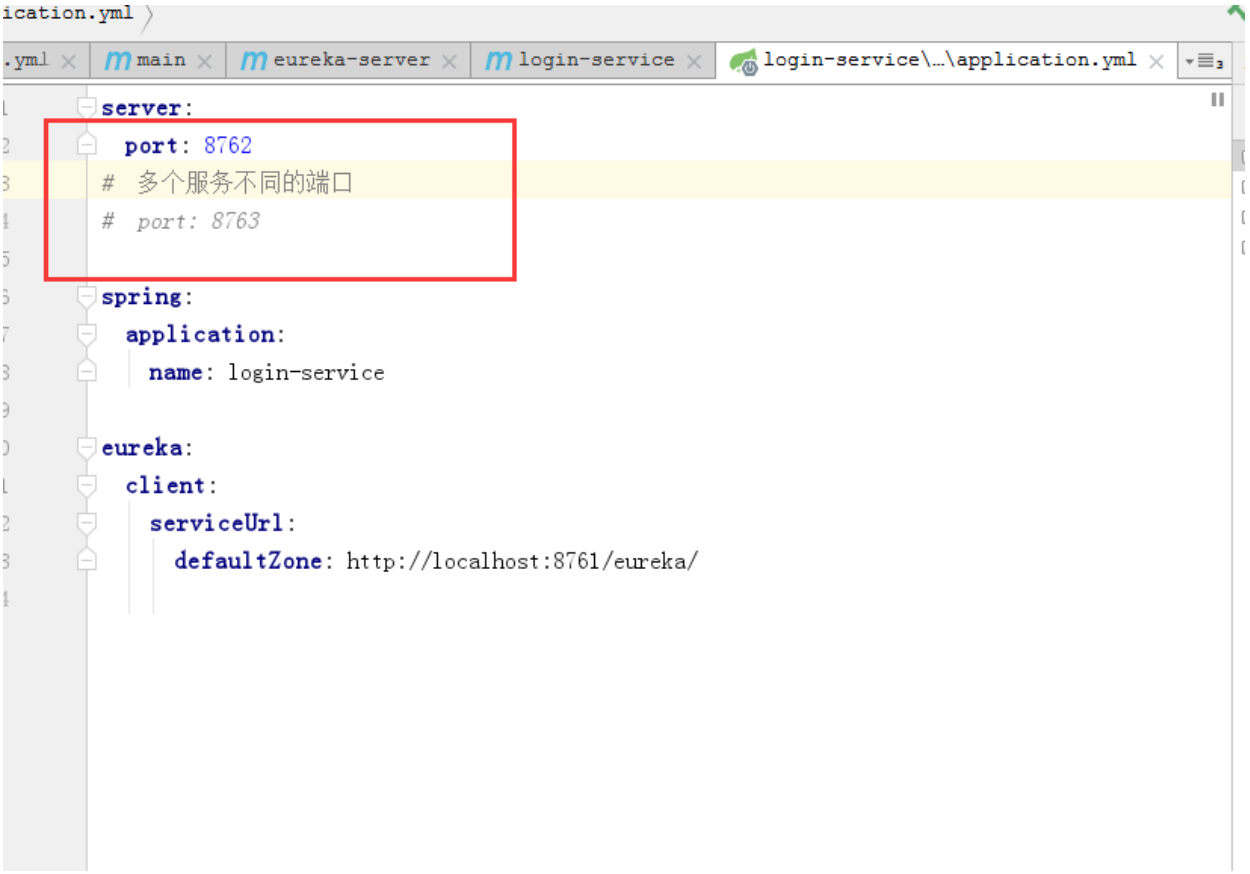
- **IClientConfig ribbonClientConfig:** DefaultClientConfigImpl
- **IRule ribbonRule:** ZoneAvoidanceRule
- **IPing ribbonPing:** NoOpPing
- **ServerList ribbonServerList:** ConfigurationBasedServerList
- **ServerListFilter ribbonServerListFilter:** ZonePreferenceServerListFilter
- **ILoadBalancer ribbonLoadBalancer:** ZoneAwareLoadBalancer

好了，废话不多说，直接上手操作一番。

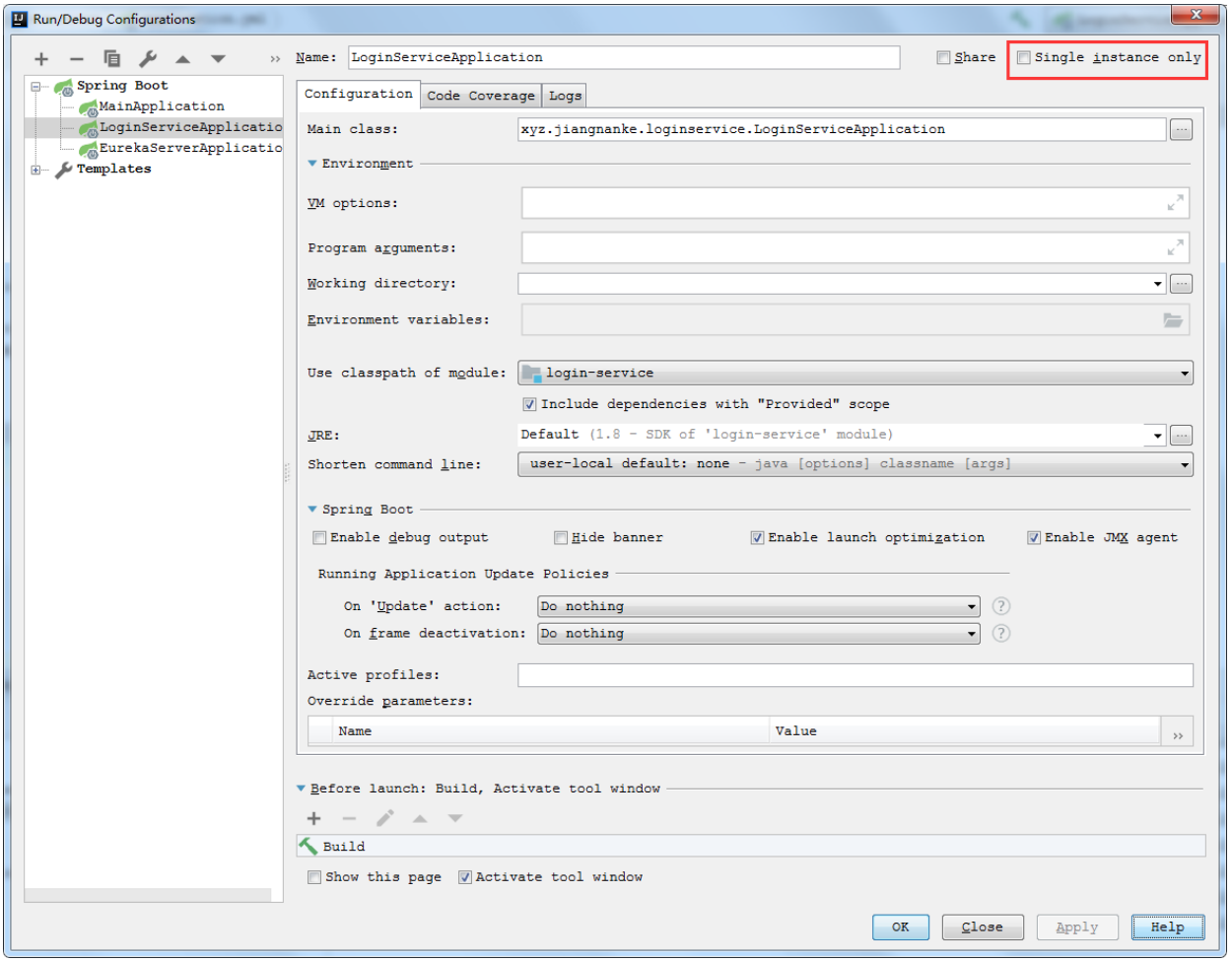
二、LoginService的负载均衡处理

1、准备工作

启动多个login-service实例，当然由于是单机的运行，所以需要端口号不一致，修改如下

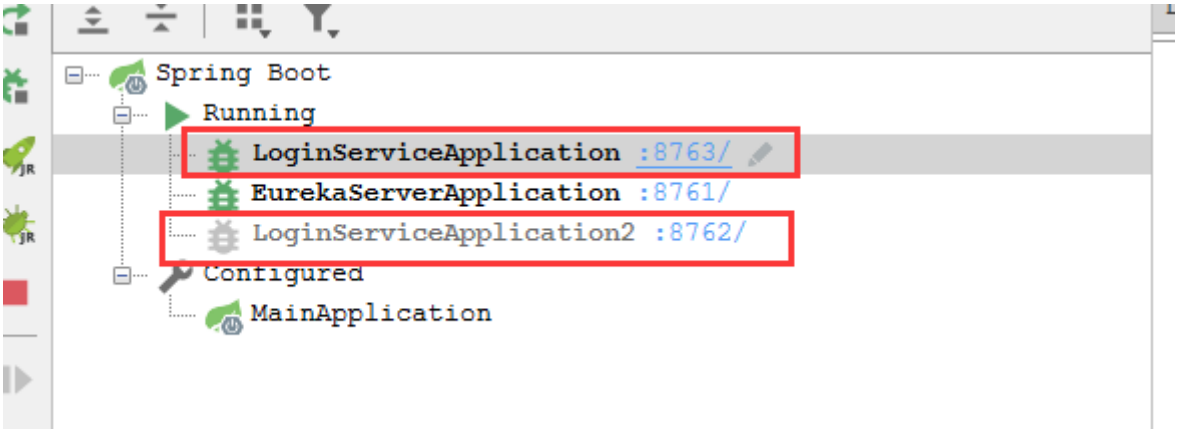


启动前也要把启动器里面的实例模式改成非单例的情况，如下：

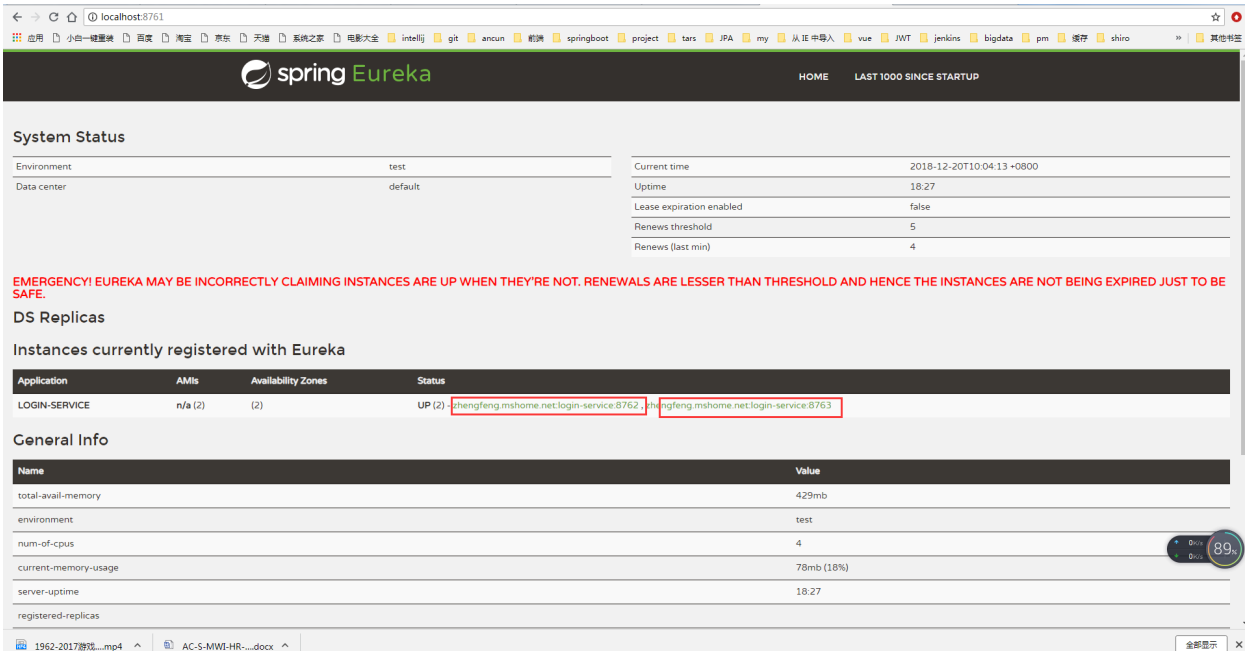


启动后变化

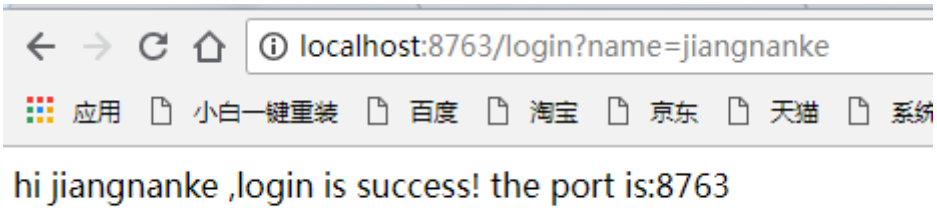
可以看到我已经启动了两个loginservice实例了



eureka里面也注册了

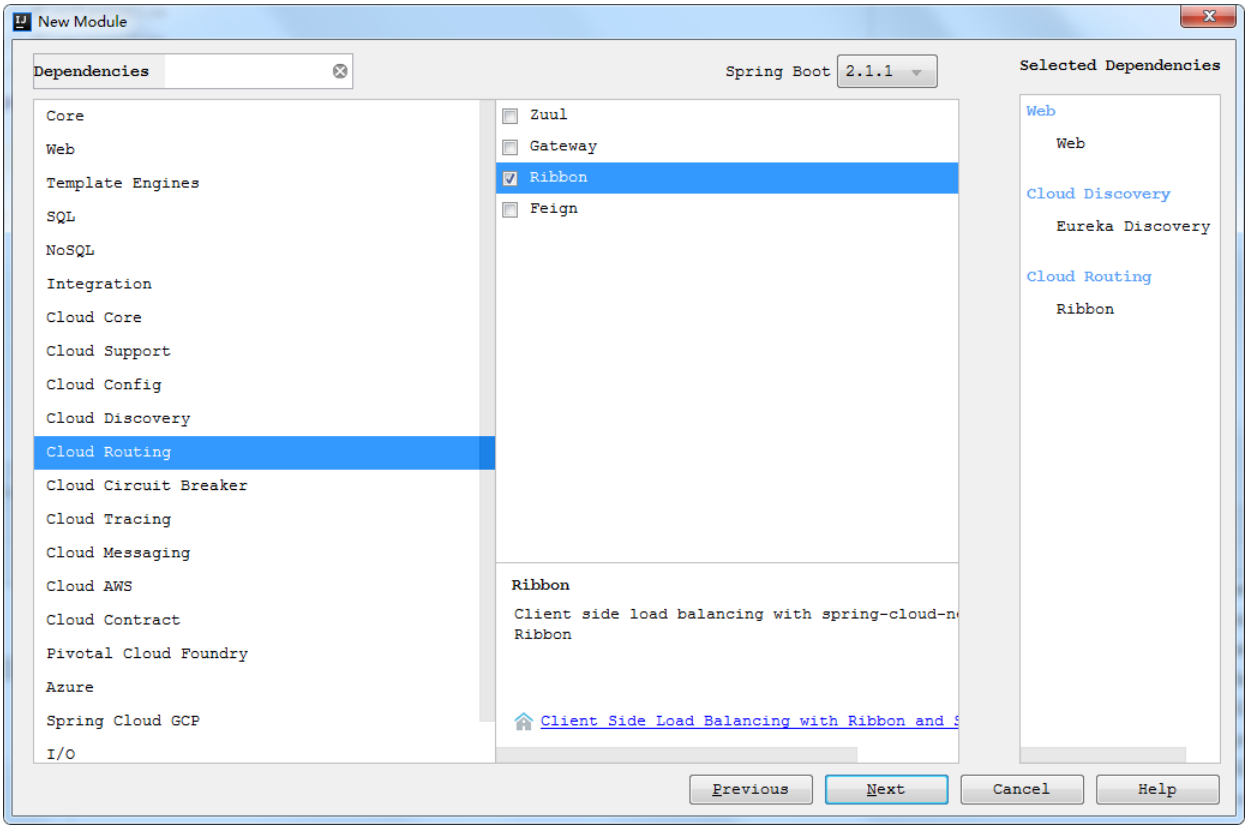


也可以访问成功



2、创建ribbon项目

重新新建一个spring-boot工程，取名为：ribbon-service;



在它的pom.xml继承了父pom文件，并引入了以下依赖：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>ribbon-service</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>ribbon-service</name>
 <description>Demo project for Spring Boot</description>

  <parent>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>main</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 </parent>

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

</project>
```

main里面也要加上ribbon-service的模块

```

<modules>
  <module>../eureka-server</module>
  <module>../login-service</module>
```

```
  <module>../ribbon-service</module>
</modules>
```

3、配置项目

3.1、配置application.yml文件

```
server:
 port: 8764

spring:
 application:
 name: ribbon-service

eureka:
 client:
  serviceUrl:
  defaultZone: http://localhost:8761/eureka/
```

3.2、配置启动类

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。

```
package xyz.jiangnanke.ribbonservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
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;

@EnableEurekaClient
@SpringBootApplication
@EnableDiscoveryClient
public class RibbonServiceApplication {

 public static void main(String[] args) {
 SpringApplication.run(RibbonServiceApplication.class, args);
```

```
18   }
19
20   @Bean
21   @LoadBalanced
22   RestTemplate restTemplate() {
23   return new RestTemplate();
24   }
25
26  }
27
```

3.3、编写测试类

写一个测试类HelloService，通过之前注入ioc容器的restTemplate来消费service-hi服务的“/hi”接口，在这里我们直接用的程序名替代了具体的url地址，在ribbon中它会根据服务名来选择具体的服务实例，根据服务实例在请求的时候会用具体的url替换掉服务名。

LoginService.java 代码如下：

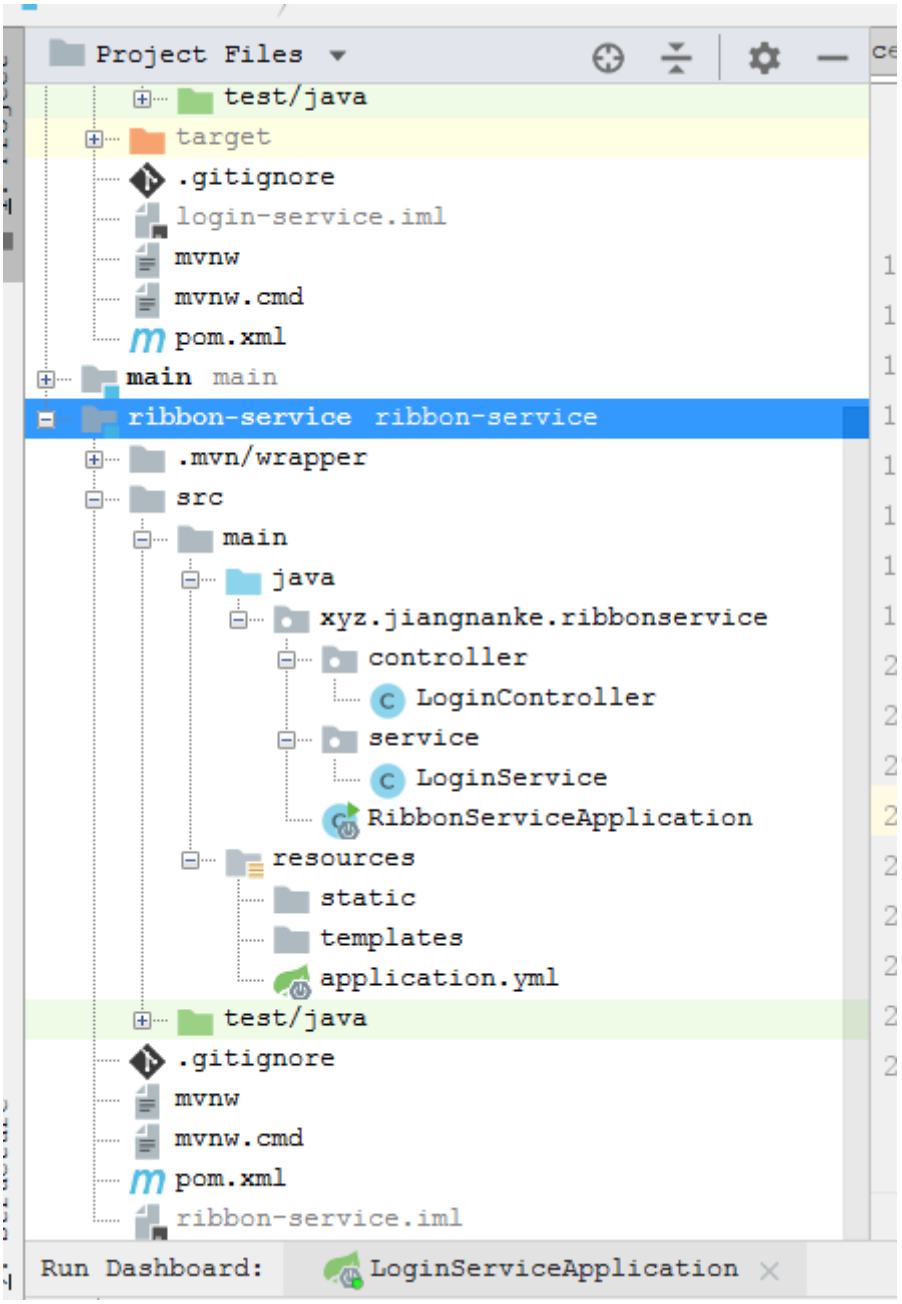
```
1  package xyz.jiangnanke.ribbonservice.service;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Service;
5  import org.springframework.web.client.RestTemplate;
6
7  /**
8   * @Auther: zhengfeng
9   * @Date: 2018\12\20 0020 10:18
10   * @Description:
11   */
12 @Service
13 public class LoginService {
14
15   @Autowired
16   RestTemplate restTemplate;
17
18   public String hiService(String name) {
19   return restTemplate.getForObject("http://LOGIN-SERVICE/login?name="+nam
e,String.class);
20   }
21 }
```

LoginController.java代码如下：

```
1  package xyz.jiangnanke.ribbonservice.controller;
2
```

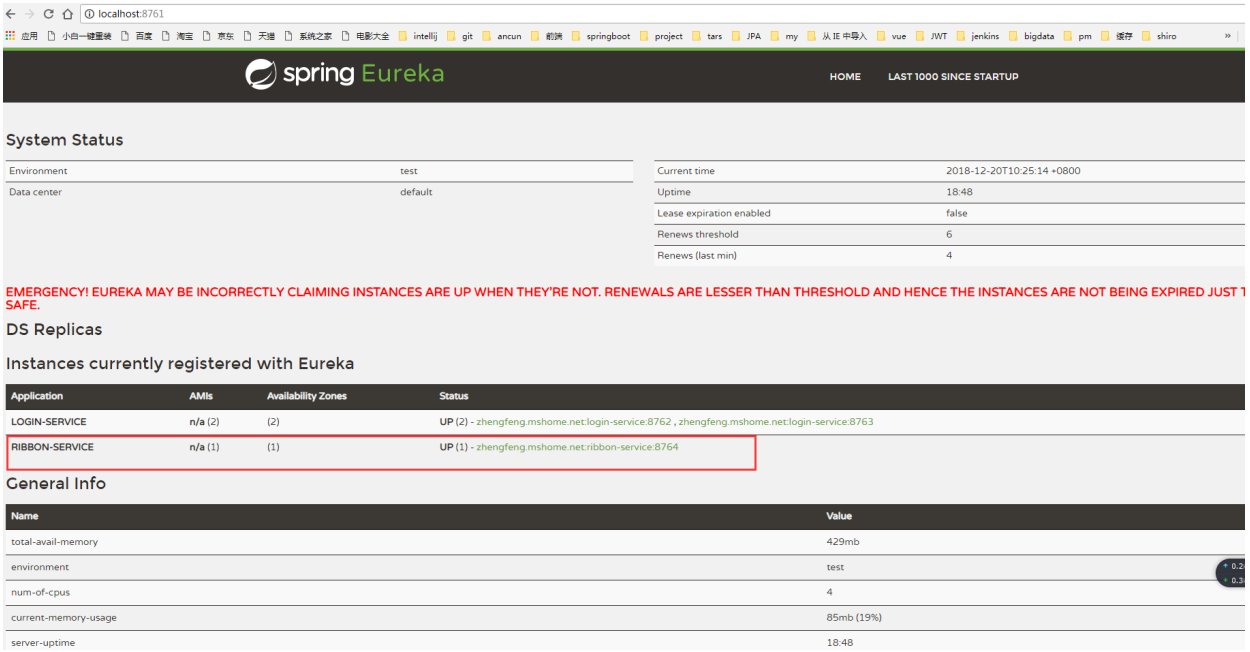
```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import xyz.jiangnanke.ribbonservice.service.LoginService;

/**
 * @Auther: zhengfeng
 * @Date: 2018\12\20 0020 10:20
 * @Description:
 */
@RestController
public class LoginController {
 @Autowired
 LoginService loginService;

 @GetMapping(value = "/login")
 public String login(@RequestParam String name){
 return loginService.login(name);
 }
}
```

最终项目结构如图：

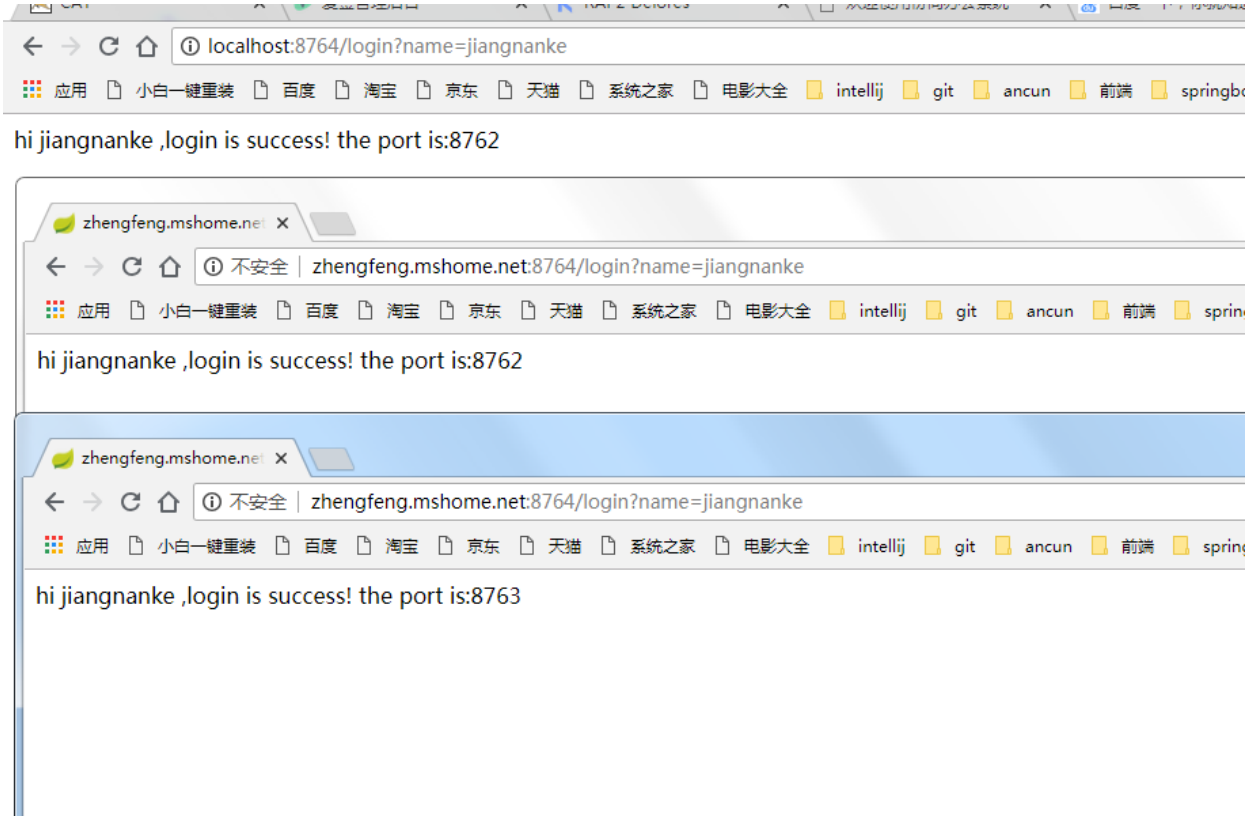


3.4、运行测试

启动项目，查看eureka的注册情况如下：



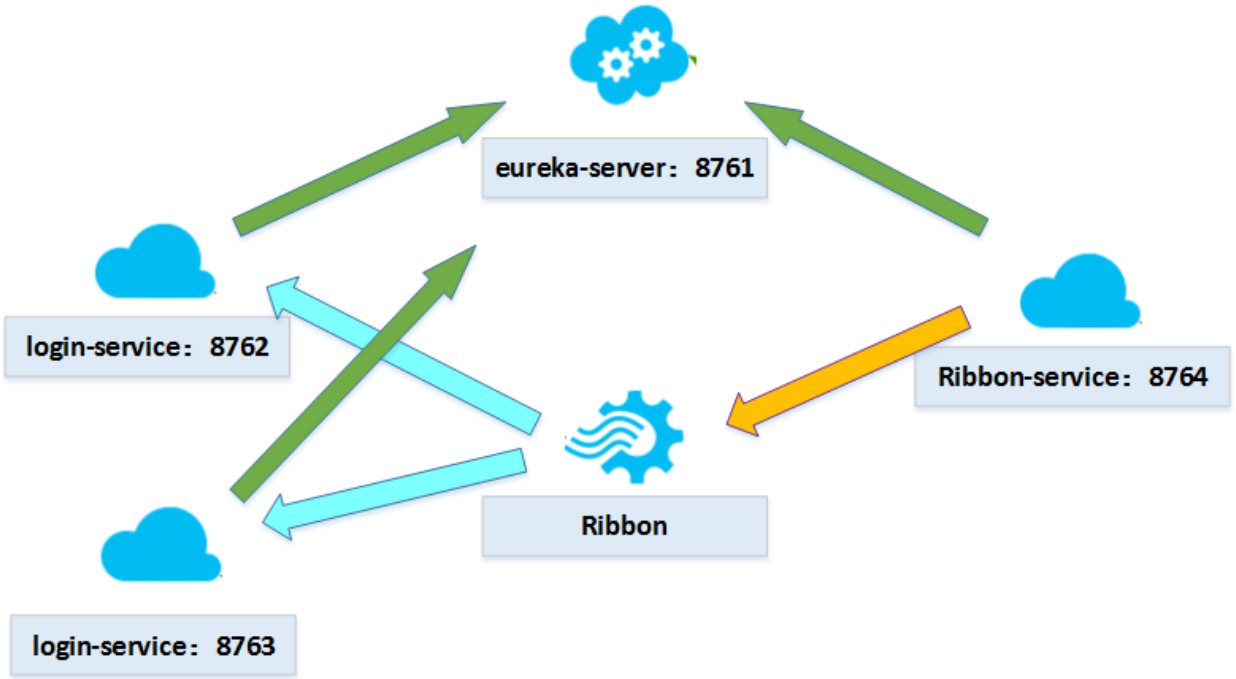
多次访问如下：



这说明当我们通过调用`restTemplate.getForObject("http://LOGIN-SERVICE/login?name="+name,String.class);`方法时，已经做了负载均衡，访问了不同的端口的服务实例。

四、说在后面

此时我的架构是：



- 一个服务注册中心，eureka server,端口为8761
- login-service工程跑了两个实例，端口分别为8762,8763，分别向服务注册中心注册
- ribbon-sercvice端口为8764,向服务注册中心注册
- 当ribbon-sercvice通过restTemplate调用login-service 的 login 接口时，因为用ribbon进行了负载均衡，会轮流的调用login-service：8762和8763 两个端口的 login 接口；

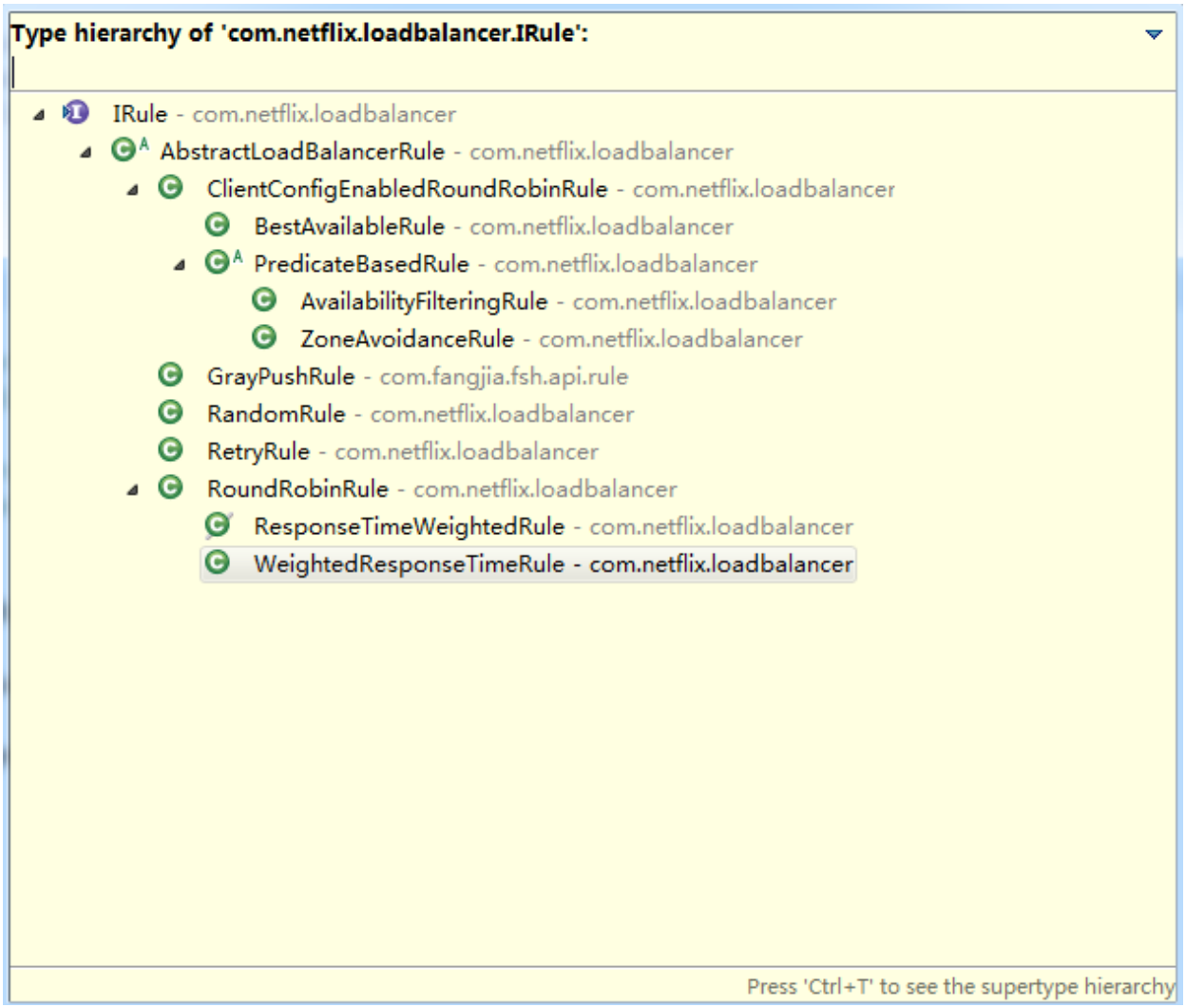
目前主流的负载方案分为两种，**一种是集中式负载均衡**，在消费者和服务提供方中间使用独立的代理方式进行负载，有硬件的，比如F5，也有软件的，比如Nginx。

另一种则是客户端自己做负载均衡，根据自己的请求情况做负载，Ribbon就是属于客户端自己做负载的。

一句话介绍那就是Ribbon是Netflix开源的一款用于客户端负载均衡的工具软件。GitHub地址：https://github.com/Netflix/ribbon。

Ribbon默认的策略是轮询，我们可以自定义负载策略来覆盖默认的，当然也可以通过配置指定使用哪些策略。

Ribbon支持的策略如下图，GrayPushRule请忽略，那是我自定义的灰度发布的Rule：



- BestAvailabl：选择一个最小的并发请求的Server，逐个考察Server，如果Server被tripped了，则跳过。
- AvailabilityFilteringRule：过滤掉那些一直连接失败的被标记为circuit tripped的后端Server，并过滤掉那些高并发的的后端Server或者使用一个AvailabilityPredicate来包含过滤server的逻辑，其实就就是检查status里记录的各个Server的运行状态。
- ZoneAvoidanceRule：复合判断Server所在区域的性能和Server的可用性选择Server。
- RandomRule：随机选择一个Server。
- RoundRobinRule：轮询选择， 轮询index，选择index对应位置的Server。
- RetryRule：对选定的负载均衡策略机上重试机制，在一个配置时间段内当选择Server不成功，则一直尝试使用subRule的方式选择一个可用的server。
- ResponseTimeWeightedRule：作用同WeightedResponseTimeRule，二者作用是一样的，ResponseTimeWeightedRule后来改名为WeightedResponseTimeRule。

- WeightedResponseTimeRule：根据响应时间分配一个weight(权重)，响应时间越长，weight越小，被选中的可能性越低。

使用指定的规则只需要加上下面的配置即可：

```
# 配置负载均衡策略
fsh-house.ribbon.NFLoadBalancerRuleClassName=com.netflix.loadbalancer.RandomRule
```

参考资料：

git上ribbon源码： https://github.com/Netflix/ribbon

https://blog.csdn.net/forezp/article/details/81040946

http://cloud.spring.io/spring-cloud-static/Finchley.RELEASE/single/spring-cloud.html

http://cxytiandi.com/blog/detail/13598

http://cxytiandi.com/blog/detail/12507