

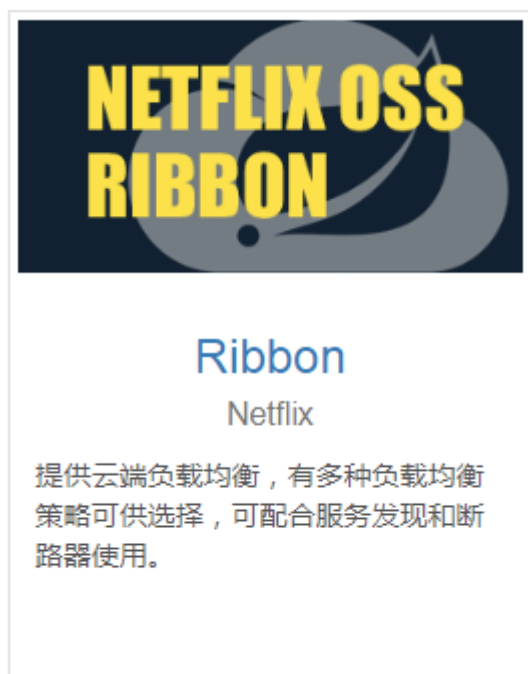
SpringCloud之三

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

Modules

- ribbon: APIs that integrate load balancing, fault tolerance, caching/batching on top of other ribbon modules and [Hystrix](#)
- ribbon-loadbalancer: Load balancer APIs that can be used independently or with other modules
- ribbon-eureka: APIs using [Eureka client](#) to provide dynamic server list for cloud
- ribbon-transport: Transport clients that support HTTP, TCP and UDP protocols using [RxNetty](#) with load balancing capability
- ribbon-httpclient: REST client built on top of Apache HttpClient integrated with load balancers (deprecated and being replaced by ribbon module)
- ribbon-example: Examples
- ribbon-core: Client configuration APIs and other shared APIs

模块

ribbon：在其他功能区模块和Hystrix之上集成负载均衡，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：使用具有负载均衡功能的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：

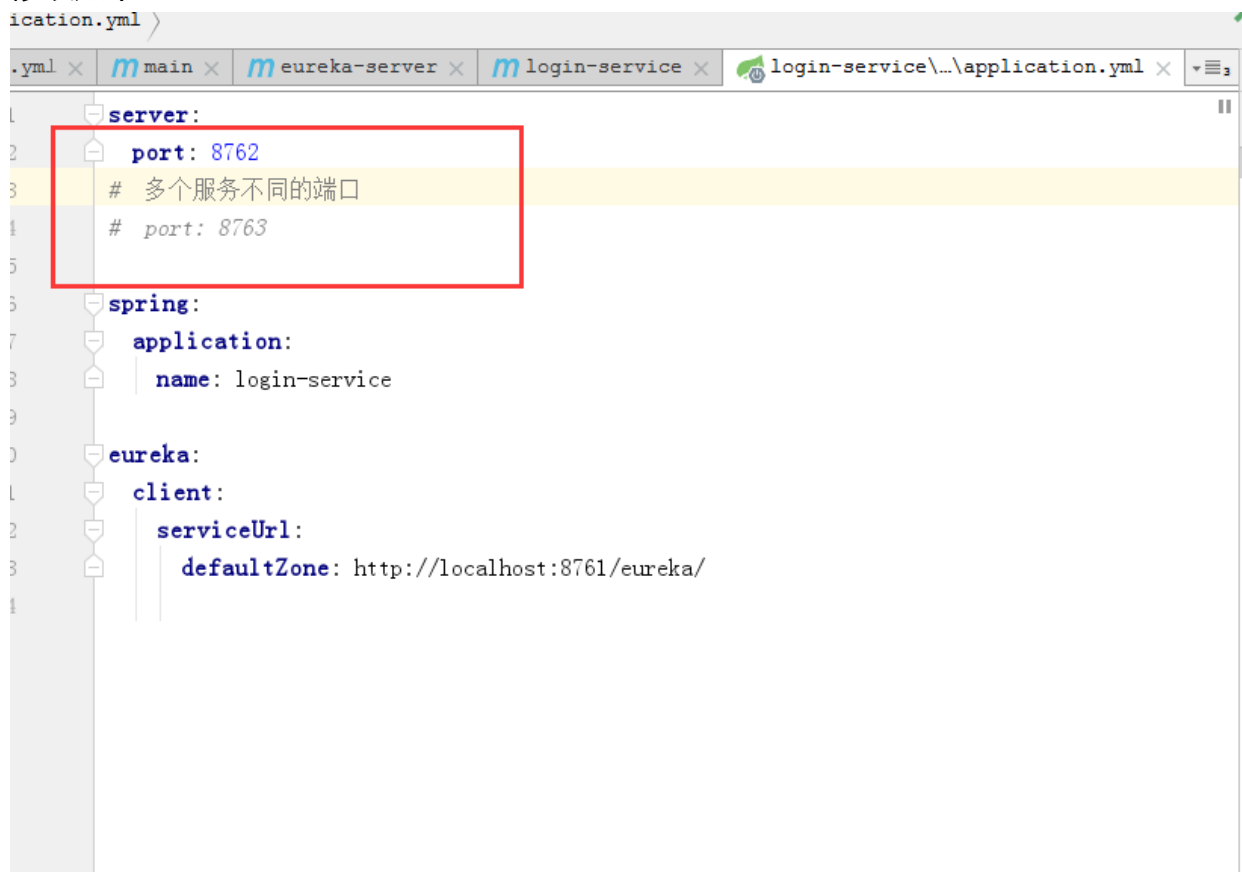
- **IClientConfig ribbonClientConfig:** DefaultClientConfigImpl
- **IRule ribbonRule:** ZoneAvoidanceRule
- **IPing ribbonPing:** NoOpPing
- **ServerList ribbonServerList:** ConfigurationBasedServerList
- **ServerListFilter ribbonServerListFilter:** ZonePreferenceServerListFilter
- **ILoadBalancer ribbonLoadBalancer:**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

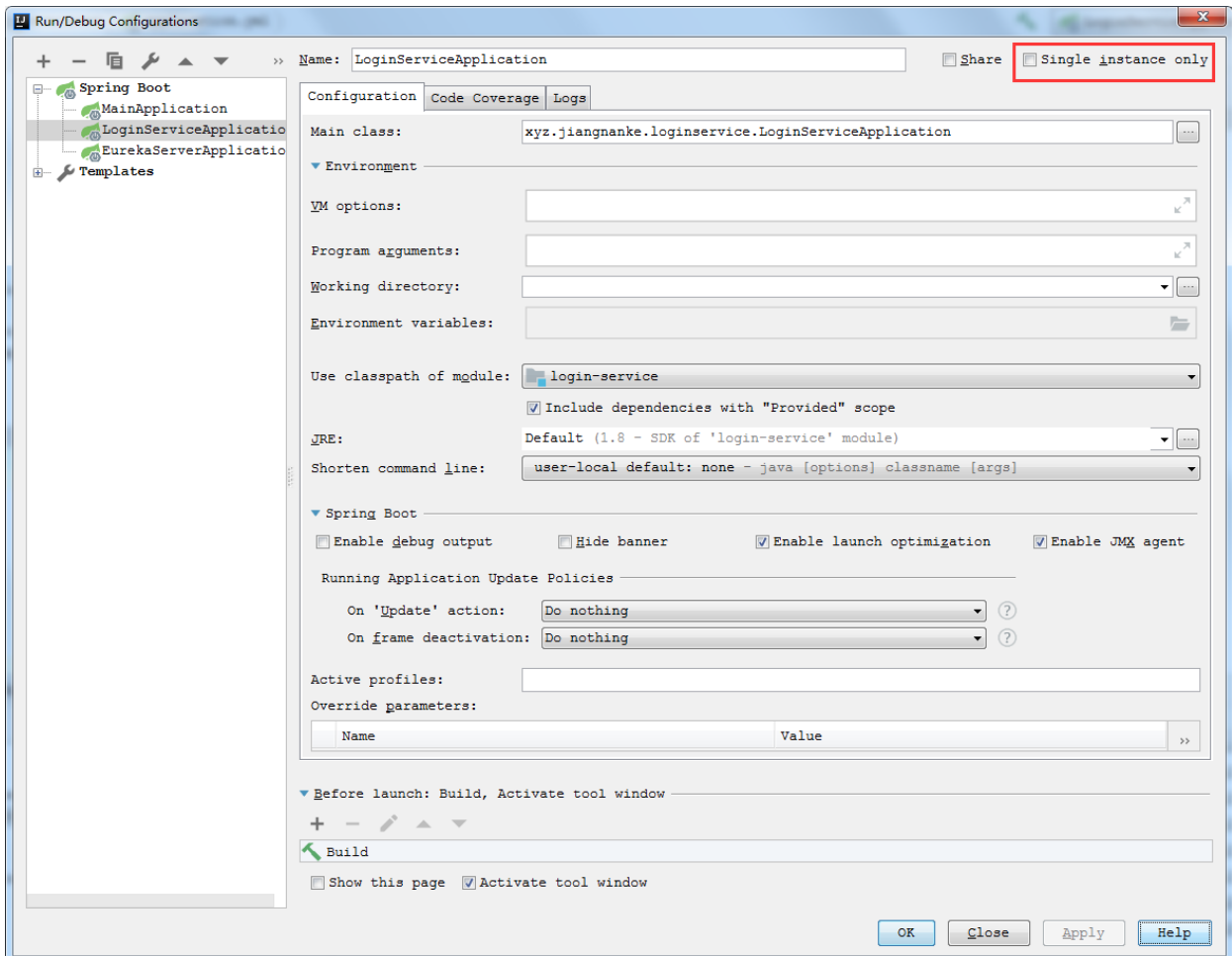


```
server:
  port: 8762
  # 多个服务不同的端口
  # port: 8763

spring:
  application:
    name: login-service

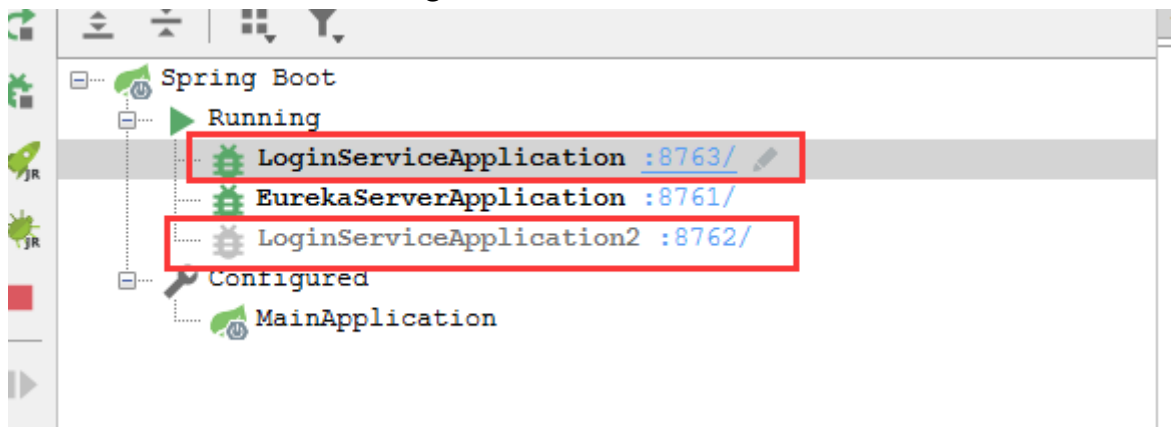
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

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：

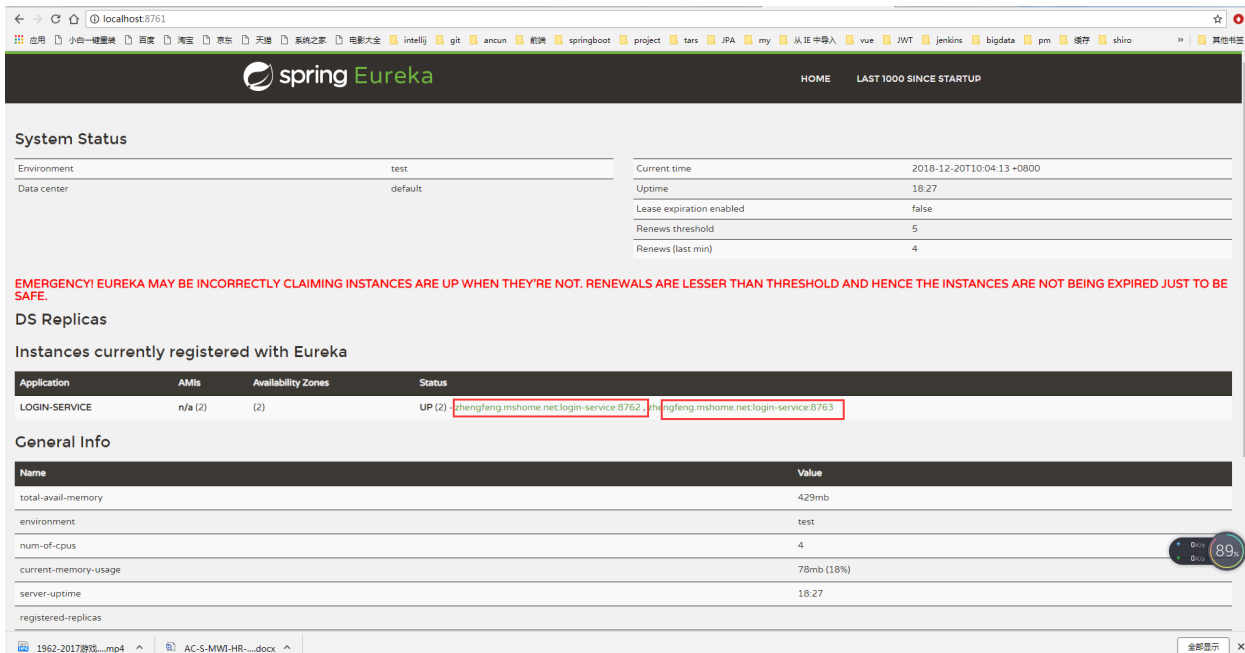


启动后变化

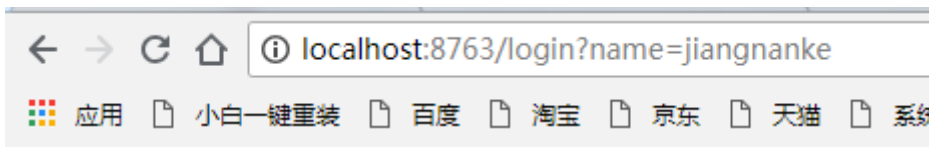
可以看到我已经启动了两个loginservice实例了



eureka里面也注册了



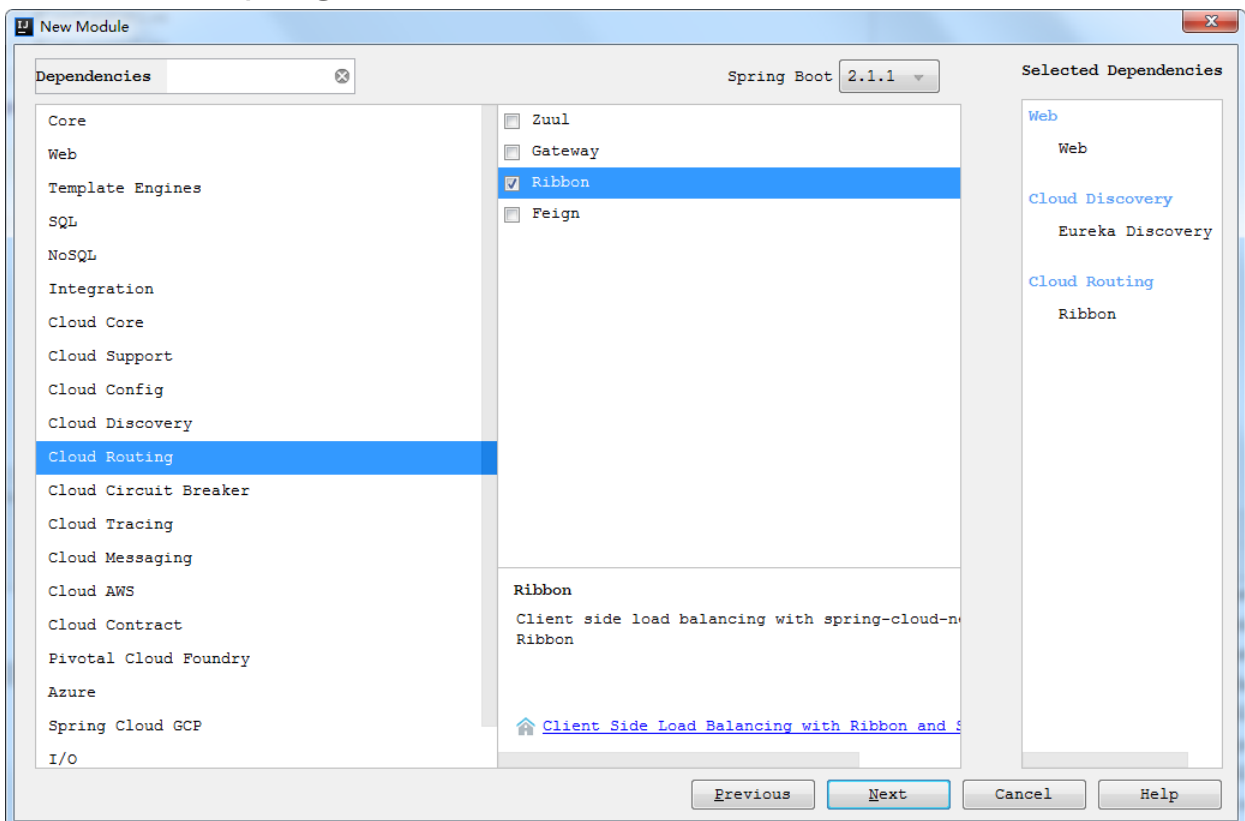
也可以访问成功



hi jiangnanke ,login is success! the port is:8763

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：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
  w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
    e.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>xyz.jiangnanke</groupId>
6   <artifactId>ribbon-service</artifactId>
7   <version>0.0.1-SNAPSHOT</version>
8   <name>ribbon-service</name>
9   <description>Demo project for Spring Boot</description>
10
11   <parent>
12     <groupId>xyz.jiangnanke</groupId>
13     <artifactId>main</artifactId>
14     <version>0.0.1-SNAPSHOT</version>
15   </parent>
16
17   <dependencies>
18     <dependency>
19       <groupId>org.springframework.boot</groupId>
20       <artifactId>spring-boot-starter-web</artifactId>
21     </dependency>
22     <dependency>
23       <groupId>org.springframework.cloud</groupId>
24       <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
25     </dependency>
26     <dependency>
27       <groupId>org.springframework.cloud</groupId>
28       <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
29     </dependency>
30   </dependencies>
31
32 </project>
```

main里面也要加上ribbon-service的模块

```
1
2 <modules>
3   <module>../eureka-server</module>
4   <module>../login-service</module>
```

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

3、配置项目

3.1、配置application.yml文件

```
1 server:
2   port: 8764
3
4 spring:
5   application:
6     name: ribbon-service
7
8 eureka:
9   client:
10    serviceUrl:
11    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。

```
1 package xyz.jiangnanke.ribbon-service;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6 import org.springframework.cloud.client.loadbalancer.LoadBalanced;
7 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
8 import org.springframework.context.annotation.Bean;
9 import org.springframework.web.client.RestTemplate;
10
11 @EnableEurekaClient
12 @SpringBootApplication
13 @EnableDiscoveryClient
14 public class RibbonServiceApplication {
15
16     public static void main(String[] args) {
17         SpringApplication.run(RibbonServiceApplication.class, args);
18     }
19 }
```

```

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.ribbon.service.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   * @Author: 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="+name,String.class);
20      }
21  }

```

LoginController.java代码如下：

```

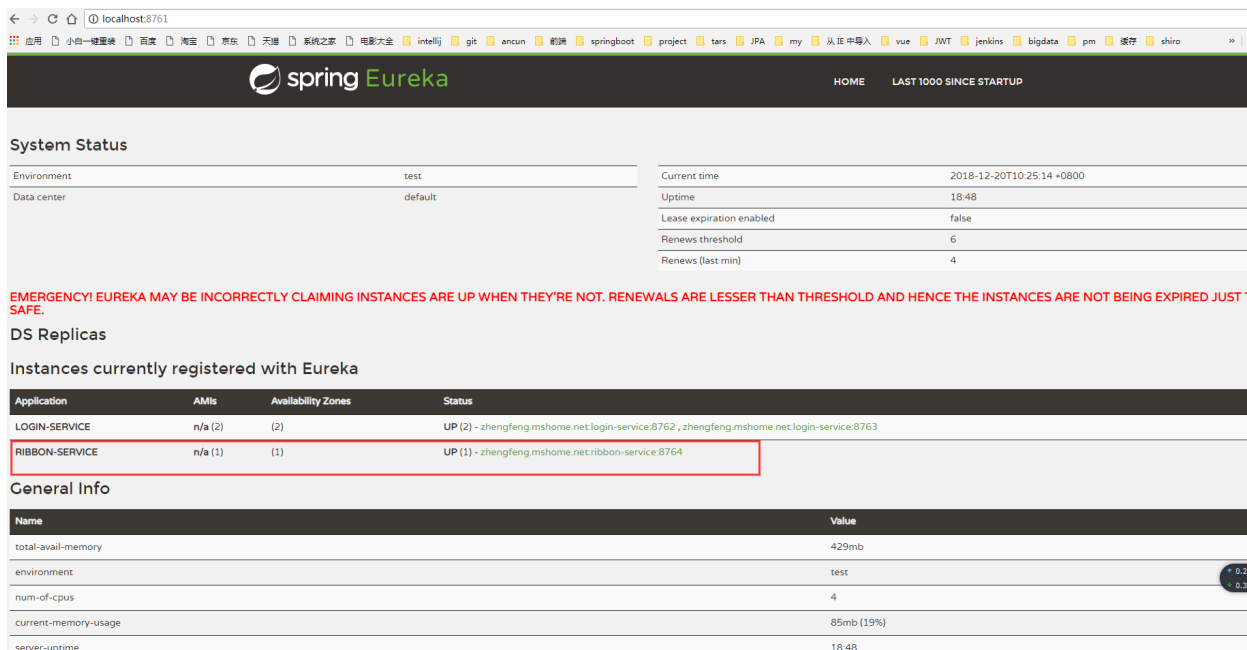
1  package xyz.jiangnanke.ribbon.service.controller;
2

```



```
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestParam;
6 import org.springframework.web.bind.annotation.RestController;
7 import xyz.jiangnanke.ribbon.service.service.LoginService;
8
9 /**
10  * @Auther: zhengfeng
11  * @Date: 2018\12\20 0020 10:20
12  * @Description:
13  */
14 @RestController
15 public class LoginController {
16     @Autowired
17     LoginService loginService;
18
19     @GetMapping(value = "/login")
20     public String login(@RequestParam String name){
21         return loginService.login(name);
22     }
23 }
```

最终项目结构如图：



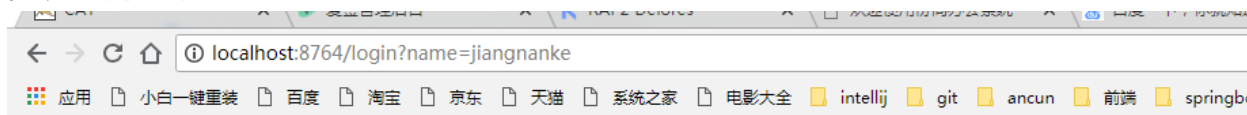
The screenshot shows the Spring Eureka dashboard. At the top, there's a navigation bar with 'HOME' and 'LAST 1000 SINCE STARTUP'. Below this is the 'System Status' section, which includes a table with system metrics like Environment (test), Data center (default), Current time (2018-12-20T10:25:14+0800), Uptime (18:48), Lease expiration enabled (false), Renewals threshold (6), and Renewals (last min) (4). A red warning message states: 'EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST YET. NOT SAFE.' Below the warning is the 'DS Replicas' section, showing 'Instances currently registered with Eureka'. This is followed by a table of registered instances:

Application	AMIs	Availability Zones	Status
LOGIN-SERVICE	n/a (2)	(2)	UP (2) - zhengfeng.mshome.net:login-service:8762, zhengfeng.mshome.net:login-service:8763
RIBBON-SERVICE	n/a (1)	(1)	UP (1) - zhengfeng.mshome.net:ribbon-service:8764

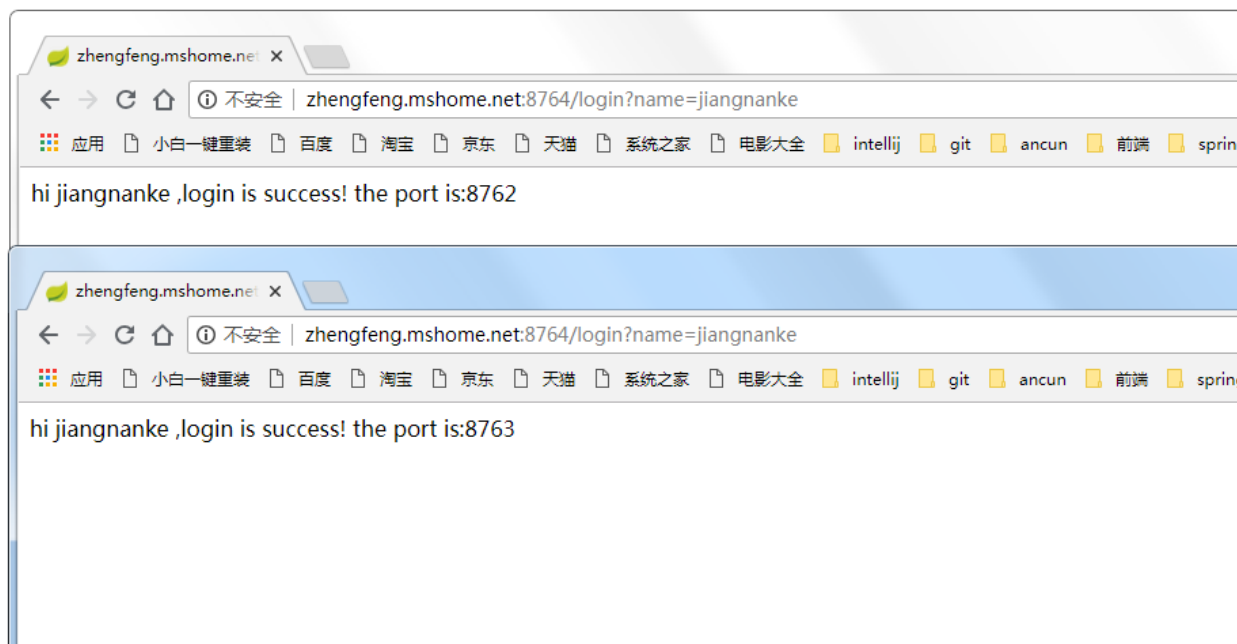
Below the instances table is the 'General Info' section, which contains a table of system metrics:

Name	Value
total-avail-memory	429mb
environment	test
num-of-cpus	4
current-memory-usage	85mb (19%)
server-uptime	18:48

多次访问如下：



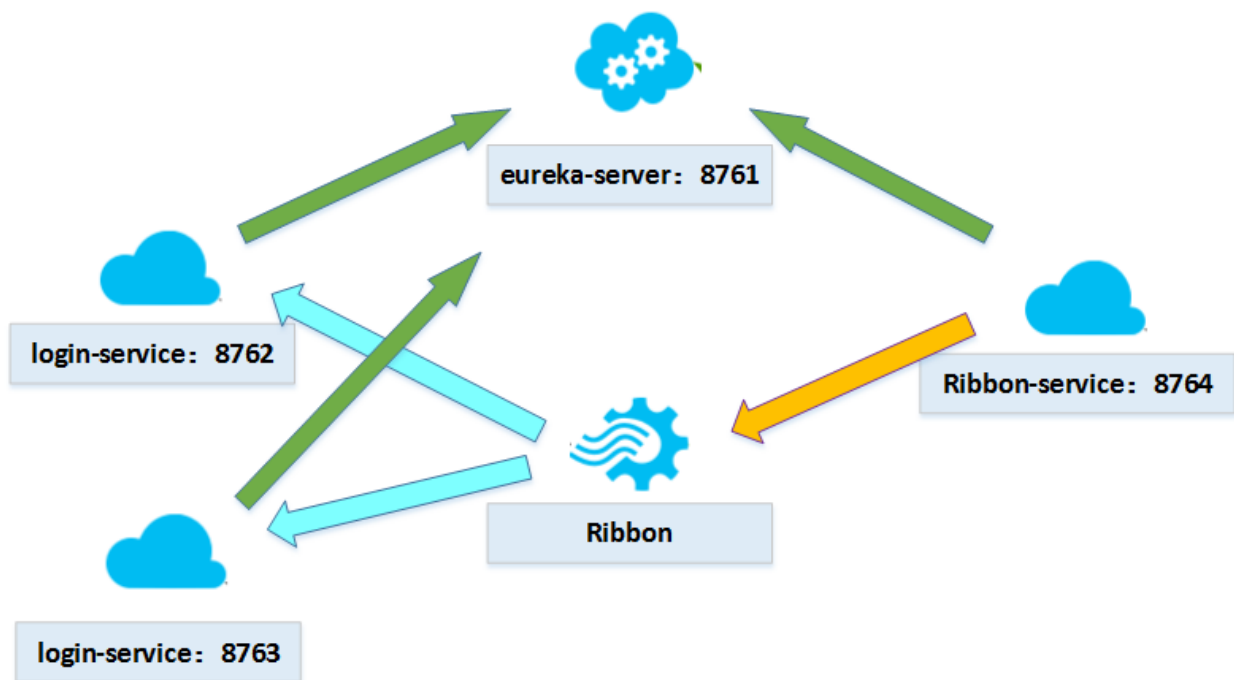
hi jiangnanke ,login is success! the port is:8762



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-sercvic端口为8764,向服务注册中心注册
- 当ribbon-sercvic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；

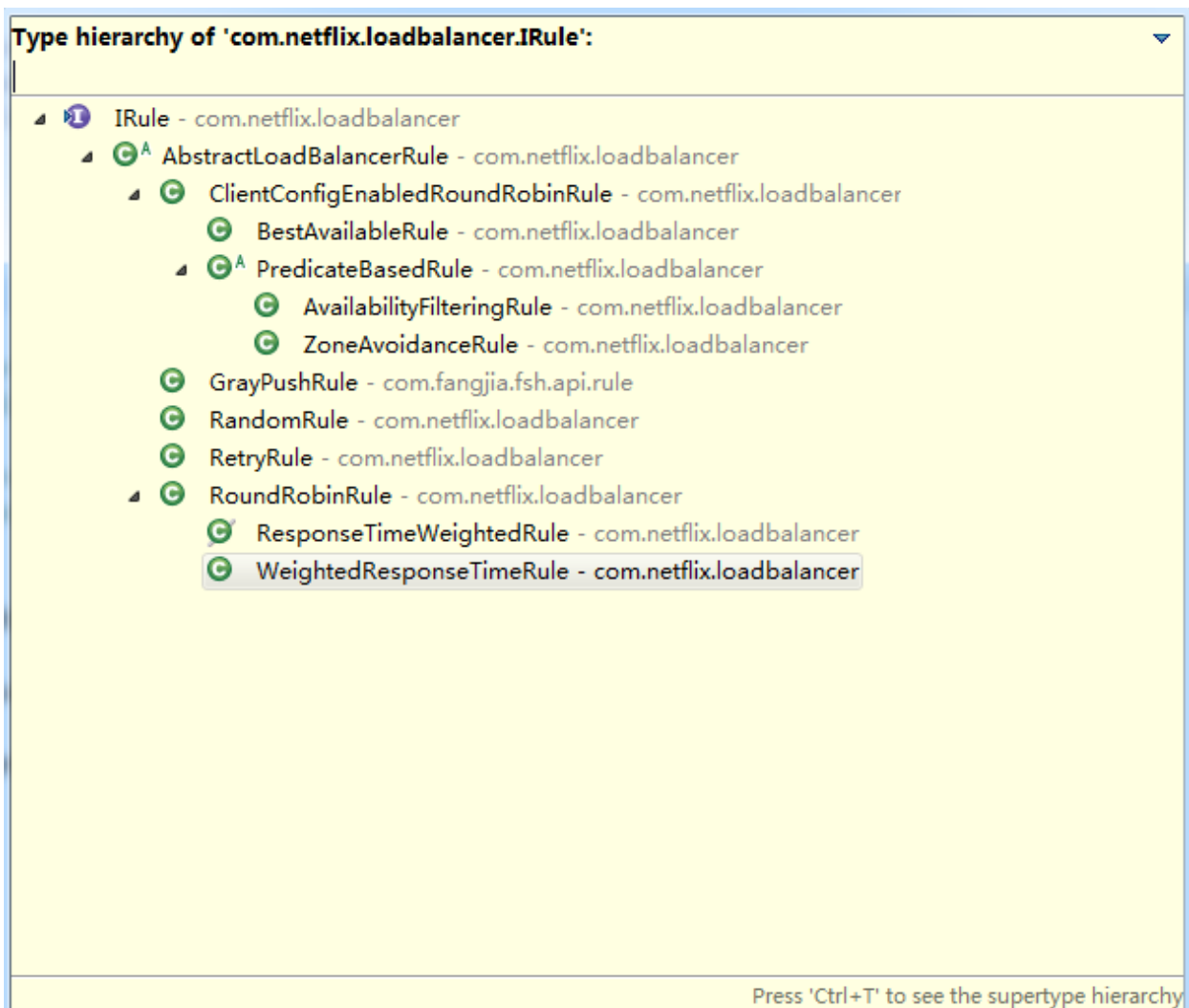
目前主流的负载方案分为两种，**一种是集中式负载均衡**，在消费者和服务提供方中间使用独立的代理方式进行负载，有硬件的，比如F5，也有软件的，比如Nginx。

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：



- BestAvailableRule : 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：

```
1 # 配置负载均衡策略
2 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>