

一、ribbon负载均衡

1.ribbon简介

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套 **客户端** **负载均衡的工具**。

简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法，将Netflix的中间层服务连接在一起。Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。

LB（负载均衡）

LB，即负载均衡(Load Balance)，在微服务或分布式集群中经常用的一种应用。
负载均衡简单的说就是将用户的请求平摊的分配到多个服务上，从而达到系统的HA。
常见的负载均衡有软件Nginx，LVS，硬件 F5等。
相应的在中间件，例如：dubbo和SpringCloud中均给我们提供了负载均衡，**SpringCloud的负载均衡算法可以自定义。**

进程内LB

将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。
Ribbon就属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

2. 基本配置

修改consumer 80项目，因为ribbon是客户端负载均衡，所以我们修改consumer项目。

1. POM依赖

consumer 80项目添加如下依赖

```
1 <!-- Ribbon相关 , 负载均衡-->
2     <dependency>
3
4     <groupId>org.springframework.cloud</groupId>
5     <artifactId>spring-cloud-starter-
6     eureka</artifactId>
7     </dependency>
8     <dependency>
9
10    <groupId>org.springframework.cloud</groupId>
11    <artifactId>spring-cloud-starter-
12    ribbon</artifactId>
13    </dependency>
14    <dependency>
15
16    <groupId>org.springframework.cloud</groupId>
17    <artifactId>spring-cloud-starter-
18    config</artifactId>
19    </dependency>
```

2. yaml文件

添加配置

```
1 eureka:
2   client:
3     register-with-eureka: false #自己不能注册
4     service-url:
5       defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7
002.com:7002/eureka/,http://eureka7003.com:7003/e
ureka/
```

3.修改ConfigBean

getRestTemplate方法上添加@LoadBalanced注解,这个注解模式使用轮询算法

```
1 @Configuration //@Configuration配置    ConfigBean =
applicationContext.xml
2 public class ConfigBean {
3     @Bean
4     @LoadBalanced//Spring Cloud Ribbon是基于
Netflix Ribbon实现的一套客户端      负载均衡的工
具。
5     public RestTemplate getRestTemplate() {
6         return new RestTemplate();
7     }
8 }
```

4. 修改主启动类

添加@EnableEurekaClient注解

```
1 @SpringBootApplication
2 @EnableEurekaClient//ribbon做负载均衡时添加的
3 public class DeptConsumer80_App {
4
5     public static void main(String[] args) {
6
7         SpringApplication.run(DepartmentConsumer80_App.class,
8             args);
9     }
10 }
```

5.修改访问客户端

使用微服务名称进行远程调用

```
1 @RestController
2 public class DeptController_Consumer {
3     // private static final String REST_URL_PREFIX
4     // = "http://localhost:8001";
5     private static final String REST_URL_PREFIX
6     = "http://springcloud-dept";
7
8     @Autowired
9     private RestTemplate restTemplate;
10
11     @SuppressWarnings("unchecked")
```

```
9      @RequestMapping(value =  
    "/consumer/dept/list")  
10     public List<Dept> list() {  
11  
12         return  
    template.getForObject(REST_URL_PREFIX +  
    "/dept/list", List.class);  
13     }  
14 }
```

6.测试

首先启动7001,7002,7003三个服务端项目，然后启动8001项目。最后启动80 项目。测试地址<http://localhost/consumer/dept/list>。

实现通过微服务名称访问微服务

3. 负载均衡配置

1.仿照8001项目创建两个provider

8002项目yaml配置

```
1 server:  
2   port: 8002      # 项目的端口号  
3  
4 mybatis:
```

```
5   config-location:
    classpath:mybatis/mybatis.cfg.xml          #
    mybatis配置文件所在路径
6   type-aliases-package:
    com.atguigu.springcloud.entities          # 所有Entity
    别名类所在包
7   mapper-locations:
8   - classpath:mybatis/mapper/**/*.xml
      # mapper映射文件
9
10  spring:
11    profiles:
12      active:
13      - dev
14    application:
15      name: springcloud-dept  # 当前微服务向外暴露的
    微服务名称
16    datasource:
17      type: com.alibaba.druid.pool.DruidDataSource
      # 当前数据源操作类型
18      driver-class-name: org.gjt.mm.mysql.Driver
      # mysql驱动包
19      url: jdbc:mysql://localhost:3306/cloudDB02
      # 数据库名称
20      username: root
21      password: 123456
22      dbcp2:
```

```
23     min-idle: 5
        # 数据库连接池的最小维持连接数
24     initial-size: 5
        # 初始化连接数
25     max-total: 5
        # 最大连接数
26     max-wait-millis: 200
        # 等待连接获取的最大超时时间
27 eureka:
28     client: #客户端注册进eureka服务列表内
29         service-url:
30 #         defaultZone: http://localhost:7001/eureka
        #单机版
31         defaultZone:
            http://eureka7001.com:7001/eureka/,http://eureka
            7002.com:7002/eureka/,http://eureka7003.com:7003
            /eureka/
32     instance:
33         instance-id: deptService8002        #对当前服务
        起的别名
34         prefer-ip-address: true        #我们在eureka服务
        端查看服务名称的时候：访问路径可以显示IP地址
35     info:
36         app.name: Deptservicecloud
37         company.name: www.haoge.com
38         build.artifactId: $project.artifactId$
39         build.version: $project.version$
40
```

8003项目yaml配置

```
1 server:
2   port: 8003      # 项目的端口号
3
4 mybatis:
5   config-location:
6     classpath:mybatis/mybatis.cfg.xml      #
7     mybatis配置文件所在路径
8   type-aliases-package:
9     com.atguigu.springcloud.entities      # 所有Entity
10    别名类所在包
11   mapper-locations:
12     - classpath:mybatis/mapper/**/*.xml
13       # mapper映射文件
14
15 spring:
16   profiles:
17     active:
18     - dev
19   application:
20     name: springcloud-dept # 当前微服务向外暴露的
21     微服务名称
22   datasource:
23     type: com.alibaba.druid.pool.DruidDataSource
24       # 当前数据源操作类型
25     driver-class-name: org.gjt.mm.mysql.Driver
26       # mysql驱动包
```



```
19     url: jdbc:mysql://localhost:3306/cloudDB03
        # 数据库名称
20     username: root
21     password: 123456
22     dbcp2:
23         min-idle: 5
        # 数据库连接池的最小维持连接数
24         initial-size: 5
        # 初始化连接数
25         max-total: 5
        # 最大连接数
26         max-wait-millis: 200
        # 等待连接获取的最大超时时间
27 eureka:
28     client: #客户端注册进eureka服务列表内
29         service-url:
30 #         defaultZone: http://localhost:7001/eureka
        #单机版
31         defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka
7002.com:7002/eureka/,http://eureka7003.com:7003
/eureka/
32     instance:
33         instance-id: deptService8003        #对当前服务
起的别名
34         prefer-ip-address: true        #我们在eureka服务
端查看服务名称的时候：访问路径可以显示IP地址
35     info:
```

```
36  app.name: Deptservicecloud
37  company.name: www.haoge.com
38  build.artifactId: $project.artifactId$
39  build.version: $project.version$
40
```

2.测试

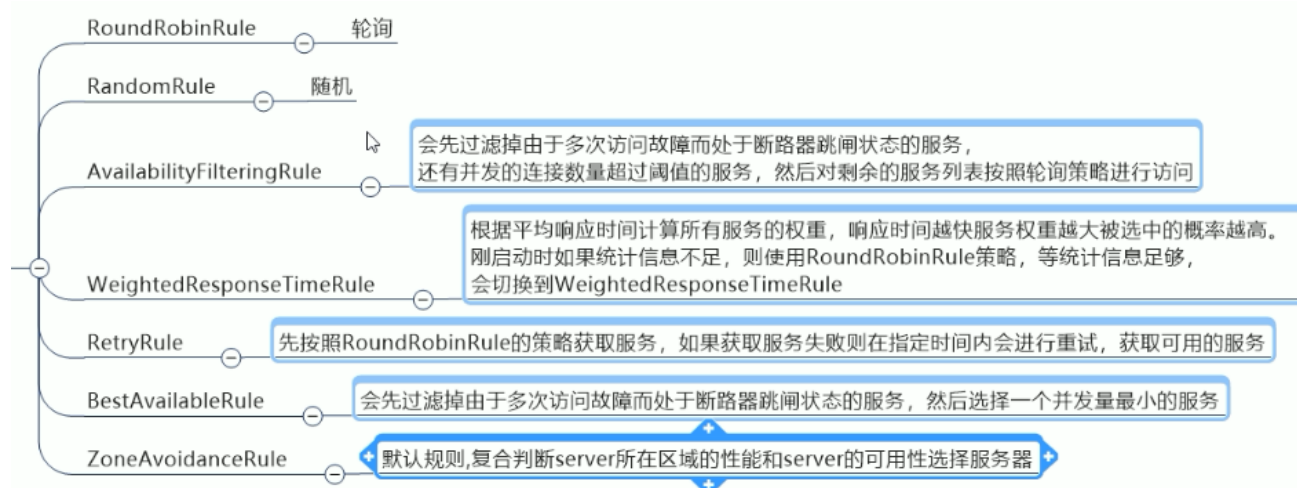
分别启动7001,7002,7003,8001,8002,8003,80 项目

链接：<http://localhost/consumer/dept/list>

效果：ribbon默认使用轮询算法。我们看到依次访问的是8001,8002,8003提供的微服务

4. 切换负载均衡策略

1. ribbon默认提供的负载均衡策略



2. 代码示例

切换ribbon默认的负载均衡策略.ribbon默认使用轮询算法，同时提供了上述7种策略。如果我们需要切换默认负载均衡算法。显式的申明我们相用的算法，并且使用@Bean注解将其加在容器中即可。如下的myRule

```
1  @Configuration // @Configuration配置    ConfigBean
   = applicationContext.xml
2  public class ConfigBean {
3      @Bean
4      @LoadBalanced//Spring Cloud Ribbon是基于
Netflix Ribbon实现的一套客户端      负载均衡的工具。
5      // @LoadBalanced内置7种不同的负载均衡的算法，如
果我们不显示的申明我们想要的算法，就使用默认的轮训算
法。
6      //如果我们显示的声明我们需要的算法，则会替代默认
的轮训算法
7      public RestTemplate getRestTemplate() {
8          return new RestTemplate();
9      }
10 //如果我们要显式的指定自己想要的算法，则改变返回算法
的名字即可。例子如下
11     @Bean
12     public IRule myRule(){
13         //return new RetryRule();//如果服务提供者
全部可用，则和轮训算法一样。当某一个服务不可用的时候
```

```
14 //查询该服务不可用  
    几次之后，自动的不会再次查找该服务。在剩下的服务中进行轮训  
15  
16     return new RandomRule();//随机算法。  
    达到的目的，用我们重新选择的随机算法替代默认的轮询。  
17 }  
18 }
```

5. 自定义负责均衡算法

1. @RibbonClient

主启动类加@RibbonClient(name="springcloud-dept",configuration=MySelfRule.class)注解。表示对springcloud-dept的微服务使用MySelfRule定义的负载均衡算法

```
1 @RibbonClient(name="springcloud-  
  dept",configuration=MySelfRule.class)  
2 public class DeptConsumer80_App {  
3  
4     public static void main(String[] args) {  
5  
6         SpringApplication.run(DeptConsumer80_App.class,  
7         args);  
8     }  
9 }
```

2.MySelfRule.java

```
1 @Configuration  
2 public class MySelfRule {  
3     @Bean  
4     public IRule mySelfRuler() {  
5         return new RandomRule_ldh();  
6     }  
7 }
```

3. RandomRule_ldh

效果：当前服务器调用五次之后重新随机一个服务器再次调用五次

```
1 public class RandomRule_ldh extends
AbstractLoadBalancerRule
2 {
3     Random rand=new Random();
4     // total = 0 // 当total==5以后，我们指针才能往
    下走，
5     // index = 0 // 当前对外提供服务的服务器地址，
6     // total需要重新置为零，但是已经达到过一个5次，
    我们的index = 1
7     // 分析：我们5次，但是微服务只有8001 8002 8003
    三台，OK？
8     //
9     private int total = 0;           // 总共被调用
    的次数，目前要求每台被调用5次
10    private int currentIndex = 0;    // 当前提供服
    务的机器号
11
12    public Server choose(ILoadBalancer lb,
Object key)
13    {
14        if (lb == null) {
15            return null;
16        }
17        Server server = null;
18
19        while (server == null) {
20            if (Thread.interrupted()) {
21                return null;
```

```
22         }
23         //获得可用的服务器列表
24         List<Server> upList =
lb.getReachableServers();
25         //获得所有的服务器列表
26         List<Server> allList =
lb.getAllServers();
27
28         int serverCount = allList.size();
29         if (serverCount == 0) {
30             /*
31              * No servers. End regardless of
pass, because subsequent passes only get more
32              * restrictive.
33              */
34             return null;
35         }
36
37         //         int index =
rand.nextInt(serverCount);//
java.util.Random().nextInt(3);
38         //         server = upList.get(index);
39
40
41         //         private int total = 0;           //
总共被调用的次数，目前要求每台被调用4次
42         //         private int currentIndex = 0;    //
当前提供服务的机器号
```

```
43         if(total < 4)//如果total小于4，则继续
访问这个服务器。否则重新进行随机
44         {
45             //          currentIndex =
rand.nextInt(serverCount);//
java.util.Random().nextInt(3);
46             server =
upList.get(currentIndex);//获取将返回的服务器
47             total++;
48         }else {
49             total = 0;
50             currentIndex =
rand.nextInt(serverCount);//重新随机
51             server =
upList.get(currentIndex);//获取将返回的服务器
52             //          if(currentIndex >=
upList.size())
53             //          {
54             //              currentIndex = 0;
55             //          }
56         }
57
58
59         if (server == null) {
60             /*
61             * The only time this should
happen is if the server list were somehow
trimmed.
```



```
62         * This is a transient
condition. Retry after yielding.
63         */
64         Thread.yield();
65         continue;
66     }
67
68     if (server.isAlive()) {//如果服务器可
用，返回服务器地址
69         return (server);
70     }
71
72     // Shouldn't actually happen.. but
must be transient or a bug.
73     server = null;
74     Thread.yield();
75 }
76
77 return server;
78
79 }
80
81 @Override
82 public Server choose(Object key)
83 {
84     return choose(getLoadBalancer(), key);
85 }
86 @Override
```

```
87     public void initWithNiwsConfig(IClientConfig
    clientConfig)
88     {
89         // TODO Auto-generated method stub
90
91     }
92 }
```