Ribbon：Spring Cloud负载均衡与服务调用组件（非常详细）

< 上一节 下一节 >

Spring Cloud Ribbon 是一套基于 Netflix Ribbon 实现的客户端负载均衡和服务调用工具。

Netflix Ribbon 是 Netflix 公司发布的开源组件，其主要功能是提供客户端的负载均衡算法和服务调用。Spring Cloud 将其与 Netflix 中的其他开源服务组件（例如 Eureka、Feign 以及 Hystrix 等）一起整合进 Spring Cloud Netflix 模块中，整合后全称为 Spring Cloud Netflix Ribbon。

Ribbon 是 Spring Cloud Netflix 模块的子模块，它是 Spring Cloud 对 Netflix Ribbon 的二次封装。通过它，我们可以将面向服务的 REST 模板（RestTemplate）请求转换为客户端负载均衡的服务调用。

Ribbon 是 Spring Cloud 体系中最核心、最重要的组件之一。它虽然只是一个工具类型的框架，并不像 Eureka Server（服务注册中心）那样需要独立部署，但它几乎存在于每一个使用 Spring Cloud 构建的微服务中。

Spring Cloud 微服务之间的调用，API 网关的请求转发等内容，实际上都是通过 Spring Cloud Ribbon 来实现的，包括后续我们要介绍的 OpenFeign 也是基于它实现的。

负载均衡

在任何一个系统中，负载均衡都是一个十分重要且不得不去实施的内容，它是系统处理高并发、缓解网络压力和服务端扩容的重要手段之一。

负载均衡（Load Balance），简单点说就是将用户的请求平摊分配到多个服务器上运行，以达到扩展服务器带宽、增强数据处理能力、增加吞吐量、提高网络的可用性和灵活性的目的。

常见的负载均衡方式有两种：

- 服务端负载均衡
- 客户端负载均衡

服务端负载均衡

服务端负载均衡是最常见的负载均衡方式，其工作原理如下图。

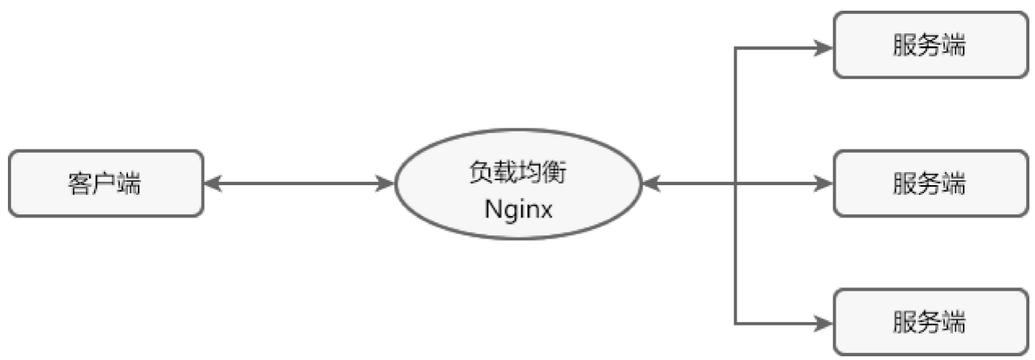

图1：服务端负载均衡工作原理

服务端负载均衡是在客户端和服务端之间建立一个独立的负载均衡服务器，该服务器既可以是硬件设备（例如 F5），也可以是软件（例如 Nginx）。这个负载均衡服务器维护了一份可用服务端清单，然后通过心跳机制来删除故障的服务端节点，以保证清单中的所有服务节点都是可以正常访问的。

当客户端发送请求时，该请求不会直接发送到服务端进行处理，而是全部交给负载均衡服务器，由负载均衡服务器按照某种算法（例如轮询、随机等），从其维护的可用服务清单中选择一个服务端，然后进行转发。

服务端负载均衡具有以下特点：

- 需要建立一个独立的负载均衡服务器。
- 负载均衡是在客户端发送请求后进行的，因此客户端并不知道到底是哪个服务端提供的服务。
- 可用服务端清单存储在负载均衡服务器上。

客户端负载均衡

相较于服务端负载均衡，客户端服务在均衡则是一个比较小众的概念。

客户端负载均衡的工作原理如下图。

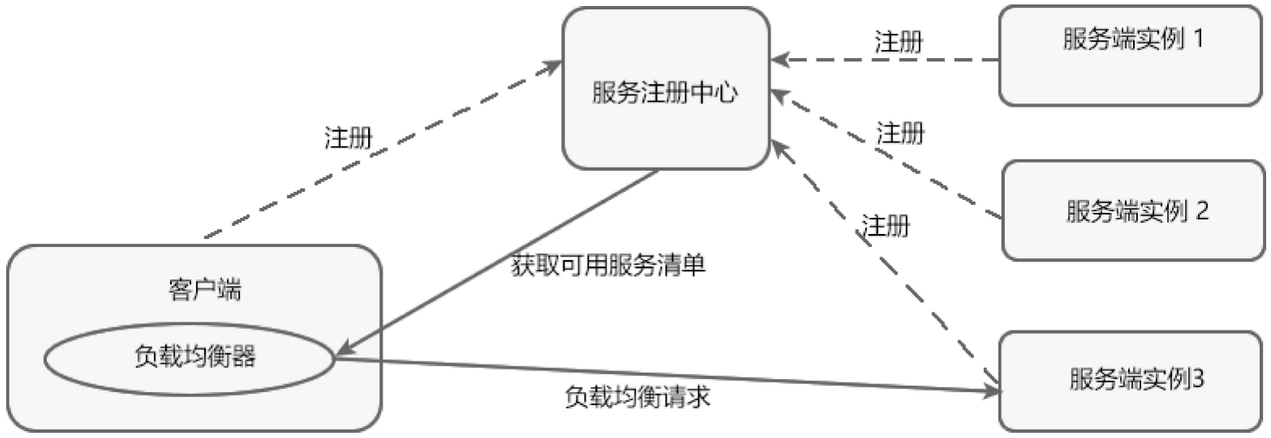

图2：客户端负载均衡工作原理

客户端负载均衡是将负载均衡逻辑以代码的形式封装到客户端上，即负载均衡器位于客户端。客户端通过服务注册中心（例如 Eureka Server）获取到一份服务端提供的可用服务清单。有了服务清单后，负载均衡器会在客户端发送请求前通过负载均衡算法选择一个服务端实例再进行访问，以达到负载均衡的目的；

客户端负载均衡也需要心跳机制去维护服务端清单的有效性，这个过程需要配合服务注册中心一起完成。

客户端负载均衡具有以下特点：

- 负载均衡器位于客户端，不需要单独搭建一个负载均衡服务器。
- 负载均衡是在客户端发送请求前进行的，因此客户端清楚地知道是哪个服务端提供的服务。
- 客户端都维护了一份可用服务清单，而这份清单都是从服务注册中心获取的。

Ribbon 就是一个基于 HTTP 和 TCP 的客户端负载均衡器，当我们将 Ribbon 和 Eureka 一起使用时，Ribbon 会从 Eureka Server（服务注册中心）中获取服务端列表，然后通过负载均衡策略将请求分摊给多个服务提供者，从而达到负载均衡的目的。

服务端负载均衡 VS 客户端负载均衡

下面我们就来对比下，服务端负载均衡和客户端负载均衡到底有什么区别，如下表。

不同点	服务端负载均衡	客户端负载均衡
是否需要建立负载均衡服务器	需要在客户端和服务端之间建立一个独立的负载均衡服务器。	将负载均衡的逻辑以代码的形式封装到客户端上，因此不需要单独建立负载均衡服务器。
是否需要服务注册中心	不需要服务注册中心。	需要服务注册中心。 在客户端负载均衡中，所有的客户端和服务端都需要将其提供的服务注册到服务注册中心上。
可用服务清单存储的位置	可用服务清单存储在位于客户端与服务器之间的负载均衡服务器上。	所有的客户端都维护了一份可用服务清单，这些清单都是从服务注册中心获取的。
负载均衡的时机	先将请求发送到负载均衡服务器，然后由负载均衡服务器通过负载均衡算法，在多个服务端之间选择一个进行访问；即在服务器端再进行负载均衡算法分配。 简单点说就是，先发送请求，再进行负载均衡。	在发送请求前，由位于客户端的服务负载均衡器（例如 Ribbon）通过负载均衡算法选择一个服务器，然后进行访问。 简单点说就是，先进行负载均衡，再发送请求。
客户端是否了解服务提供方信息	由于负载均衡是在客户端发送请求后进行的，因此客户端并不知道到底是哪个服务端提供的服务。	负载均衡是在客户端发送请求前进行的，因此客户端清楚的知道是哪个服务端提供的服务。

Ribbon 实现服务调用

Ribbon 可以与 RestTemplate（Rest 模板）配合使用，以实现微服务之间的调用。

RestTemplate 是 Spring 家族中的一个用于消费第三方 REST 服务的请求框架。RestTemplate 实现了对 HTTP 请求的封装，提供了一套模板化的服务调用方法。通过它，Spring 应用可以很方便地对各种类型的 HTTP 请求进行访问。

RestTemplate 针对各种类型的 HTTP 请求都提供了相应的方法进行处理，例如 HEAD、GET、POST、PUT、DELETE 等类型的 HTTP 请求，分别对应 RestTemplate 中的 headForHeaders()、getForObject()、postForObject()、put() 以及 delete() 方法。

下面我们通过一个简单的实例，来演示 Ribbon 是如何实现服务调用的。

1. 在主工程 spring-cloud-demo2 下，创建一个名为 micro-service-cloud-consumer-dept-80 的微服务，并在其 pom.xml 中引入所需的依赖，代码如下。

```
01.  <?xml version="1.0" encoding="UTF-8"?>
02.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
04.      <modelVersion>4.0.0</modelVersion>
05.      <!--父工程-->
06.      <parent>
07.          <artifactId>spring-cloud-demo2</artifactId>
```

```
        <groupId>net.biancheng.c</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <groupId>net.biancheng.c</groupId>
    <artifactId>micro-service-cloud-consumer-dept-80</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>micro-service-cloud-consumer-dept-80</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <!--公共子模块-->
        <dependency>
            <groupId>net.biancheng.c</groupId>
            <artifactId>micro-service-cloud-api</artifactId>
            <version>${project.version}</version>
        </dependency>
        <!--Spring Boot Web-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!--Spring Boot 测试-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--Spring Cloud Eureka 客户端依赖-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <!--Spring Cloud Ribbon 依赖-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2. 在类路径（即 /resource 目录）下，新建一个配置文件 application.yml，配置内容如下。

```
server:
  port: 80 #端口号

############################################## Spring Cloud Ribbon 负载均衡配置##########################
eureka:
  client:
    register-with-eureka: false #本微服务为服务消费者，不需要将自己注册到服务注册中心
    fetch-registry: true  #本微服务为服务消费者，需要到服务注册中心搜索服务
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/ #服务注册中心集群
```

3. 在 net.biancheng.c.config 包下，创建一个名为 ConfigBean 的配置类，将 RestTemplate 注入到容器中，代码如下。

```
package net.biancheng.c.config;

import com.netflix.loadbalancer.IRule;

import com.netflix.loadbalancer.RetryRule;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
```

```
// 配置类注解
@Configuration
public class ConfigBean {

    @Bean //将 RestTemplate 注入到容器中
    @LoadBalanced //在客户端使用 RestTemplate 请求服务端时，开启负载均衡（Ribbon）
    public RestTemplate getRestTemplate() {
        return new RestTemplate();
    }
}
```

4. 在 net.biancheng.c.controller 包下，创建一个名为 DeptController_Consumer 的 Controller，该 Controller 中定义的请求用于调用服务端提供的服务，代码如下。

```
package net.biancheng.c.controller;

import net.biancheng.c.entity.Dept;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@RestController
public class DeptController_Consumer {
    //private static final String REST_URL_PROVIDER_PREFIX = "http://localhost:8001/"; 这种方式是直调用服务方的方法，根本没有用到 Spring Cloud

    //面向微服务编程，即通过微服务的名称来获取调用地址
    private static final String REST_URL_PROVIDER_PREFIX = "http://MICROSERVICECLOUDPROVIDERDEPT"; // 使用注册到 Spring Cloud Eureka 服务注册中心中的服务，即 application.name

    @Autowired
    private RestTemplate restTemplate; //RestTemplate 是一种简单便捷的访问 restful 服务模板类，是 Spring 提供的用于访问 Rest 服务的客户端模板工具集，提供了多种便捷访问远程 HTTP 服务的方法

    //获取指定部门信息
    @RequestMapping(value = "/consumer/dept/get/{id}")
    public Dept get(@PathVariable("id") Integer id) {
        return restTemplate.getForObject(REST_URL_PROVIDER_PREFIX + "/dept/get/" + id, Dept.class);
    }
    //获取部门列表
    @RequestMapping(value = "/consumer/dept/list")
    public List<Dept> list() {
        return restTemplate.getForObject(REST_URL_PROVIDER_PREFIX + "/dept/list", List.class);
```

```
    }
}
```

5. 在 micro-service-cloud-consumer-dept-80 的主启动类上，使用 @EnableEurekaClient 注解来开启 Eureka 客户端功能，代码如下。

```
package net.biancheng.c;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class MicroServiceCloudConsumerDept80Application {

    public static void main(String[] args) {
        SpringApplication.run(MicroServiceCloudConsumerDept80Application.class, args);
    }
}
```

6. 依次启动服务注册中心 micro-service-cloud-eureka-7001、服务提供者 micro-service-cloud-provider-dept-8001 以及服务消费者 micro-service-cloud-consumer-dept-80。

7. 使用浏览器访问“http://eureka7001.com:80/consumer/dept/list”，结果如下图。

图3：Ribbon 实现服务调用

Ribbon 实现负载均衡

Ribbon 是一个客户端的负载均衡器，它可以与 Eureka 配合使用轻松地实现客户端的负载均衡。Ribbon 会先从 Eureka Server（服务注册中心）去获取服务端列表，然后通过负载均衡策略将请求分摊给多个服务端，从而达到负载均衡的目的。

Spring Cloud Ribbon 提供了一个 IRule 接口，该接口主要用来定义负载均衡策略，它有 7 个默认实现类，每一个实现类都是一种负载均衡策略。

序号	实现类	负载均衡策略
1	RoundRobinRule	按照线性轮询策略，即按照一定的顺序依次选取服务实例
2	RandomRule	随机选取一个服务实例
3	RetryRule	按照 RoundRobinRule（轮询）的策略来获取服务，如果获取的服务实例为 null 或已经失效，则在指定的时间之内不断地进行重试（重试时获取服务的策略还是 RoundRobinRule 中定义的策略），如果超过指定时间依然没获取到服务实例则返回 null 。
4	WeightedResponseTimeRule	WeightedResponseTimeRule 是 RoundRobinRule 的一个子类，它对 RoundRobinRule 的功能进行了扩展。 根据平均响应时间，来计算所有服务实例的权重，响应时间越短的服务实例权重越高，被选中的概率越大。刚启动时，如果统计信息不足，则使用线性轮询策略，等信息足够时，再切换到 WeightedResponseTimeRule。
5	BestAvailableRule	继承自 ClientConfigEnabledRoundRobinRule。先过滤点故障或失效的服务实例，然后再选择并发量最小的服务实例。
6	AvailabilityFilteringRule	先过滤掉故障或失效的服务实例，然后再选择并发量较小的服务实例。
7	ZoneAvoidanceRule	默认的负载均衡策略，综合判断服务所在区域（zone）的性能和服务（server）的可用性，来选择服务实例。在没有区域的环境下，该策略与轮询（RandomRule）策略类似。

下面我们就来通过一个实例来验证下，Ribbon 默认是使用什么策略选取服务实例的。

1. 在 MySQL 数据库中执行以下 SQL 语句，准备测试数据。

```
DROP DATABASE IF EXISTS spring_cloud_db2;

CREATE DATABASE spring_cloud_db2 CHARACTER SET UTF8;

USE spring_cloud_db2;

DROP TABLE IF EXISTS `dept`;
CREATE TABLE `dept` (
  `dept_no` int NOT NULL AUTO_INCREMENT,
  `dept_name` varchar(255) DEFAULT NULL,
  `db_source` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`dept_no`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `dept` VALUES ('1', '开发部', DATABASE());
INSERT INTO `dept` VALUES ('2', '人事部', DATABASE());
INSERT INTO `dept` VALUES ('3', '财务部', DATABASE());
INSERT INTO `dept` VALUES ('4', '市场部', DATABASE());
```

```
INSERT INTO `dept` VALUES ('5', '运维部', DATABASE());

#####################################################################################################
DROP DATABASE IF EXISTS spring_cloud_db3;

CREATE DATABASE spring_cloud_db3 CHARACTER SET UTF8;

USE spring_cloud_db3;

DROP TABLE IF EXISTS `dept`;
CREATE TABLE `dept` (
  `dept_no` int NOT NULL AUTO_INCREMENT,
  `dept_name` varchar(255) DEFAULT NULL,
  `db_source` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`dept_no`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `dept` VALUES ('1', '开发部', DATABASE());
INSERT INTO `dept` VALUES ('2', '人事部', DATABASE());
INSERT INTO `dept` VALUES ('3', '财务部', DATABASE());
INSERT INTO `dept` VALUES ('4', '市场部', DATABASE());
INSERT INTO `dept` VALUES ('5', '运维部', DATABASE());
```

2. 参考 micro-service-cloud-provider-dept-8001，再创建两个微服务 Moudle： micro-service-cloud-provider-dept-8002 和 micro-service-cloud-provider-dept-8003。

3. 在 micro-service-cloud-provider-dept-8002 中 application.yml 中，修改端口号、数据库连接信息以及自定义服务名称信息（eureka.instance.instance-id），修改的配置如下。

```
server:
  port: 8002  #端口号修改为 8002

spring:
  application:
    name: microServiceCloudProviderDept  #微服务名称，不做修改，与 micro-service-cloud-provider-dept-8001 的配置保持一致

  datasource:
    username: root        #数据库登陆用户名
    password: root        #数据库登陆密码
    url: jdbc:mysql://127.0.0.1:3306/spring_cloud_db2       #数据库url
    driver-class-name: com.mysql.jdbc.Driver                  #数据库驱动

eureka:
  client: #将客户端注册到 eureka 服务列表内
      service-url:
        #defaultZone: http://eureka7001:7001/eureka   #这个地址是 7001注册中心在 application.yml 中暴露出来额注册地址 （单机版）
```

```
18.             defaultZone:
    http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/   #将服务注册到 Eureka
    集群
19.     instance:
20.         instance-id: spring-cloud-provider-8002 #修改自定义的服务名称信息
21.         prefer-ip-address: true   #显示访问路径的 ip 地址
```

4. 在 micro-service-cloud-provider-dept-8003 中 application.yml 中，修改端口号以及数据库连接信息，修改的配置如下。

```
01.  server:
02.    port: 8003  #端口号修改为 8003
03.
04.  spring:
05.    application:
06.      name: microServiceCloudProviderDept  #微服务名称，不做修改，与 micro-service-cloud-provider-dept-8001 的配置保持一致
07.
08.    datasource:
09.      username: root        #数据库登陆用户名
10.      password: root        #数据库登陆密码
11.      url: jdbc:mysql://127.0.0.1:3306/spring_cloud_db3      #数据库url
12.      driver-class-name: com.mysql.jdbc.Driver                  #数据库驱动
13.
14.  eureka:
15.    client: #将客户端注册到 eureka 服务列表内
16.        service-url:
17.          #defaultZone: http://eureka7001:7001/eureka   #这个地址是 7001注册中心在 application.yml 中暴露出来额注册地址 （单机
    版）
18.            defaultZone:
    http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/   #将服务注册到 Eureka
    集群
19.    instance:
20.        instance-id: spring-cloud-provider-8003 #自定义服务名称信息
21.        prefer-ip-address: true   #显示访问路径的 ip 地址
```

5. 依次启动 micro-service-cloud-eureka-7001/7002/7003（服务注册中心集群）、micro-service-cloud-provider-dept-8001/8002/8003（服务提供者集群）以及 micro-service-cloud-consumer-dept-80（服务消费者）。

6. 使用浏览器连续访问“http://eureka7001.com/consumer/dept/get/1”,结果如下图。

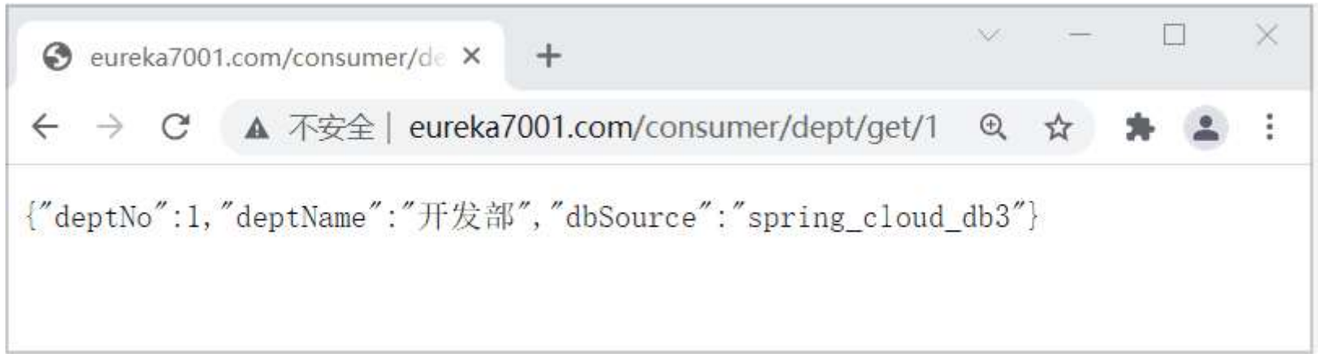

图4：Ribbon 默认负载均衡策略

通过图 4 中 dbSource 字段取值的变化可以看出，Spring Cloud Ribbon 默认使用轮询策略进行负载均衡。

切换负载均衡策略

Spring Cloud Ribbon 默认使用轮询策略选取服务实例，我们也可以根据自身的需求切换负载均衡策略。

切换负载均衡策略的方法很简单，我们只需要在服务消费者（客户端）的配置类中，将 IRule 的其他实现类注入到容器中即可。

下面我们就通过一个实例，来演示下如何切换负载均衡的策略。

1. 在 micro-service-cloud-consumer-dept-80 的配置类 ConfigBean 中添加以下代码，将负载均衡策略切换为 RandomRule（随机）。

```
01.  @Bean
02.  public IRule myRule() {
03.      // RandomRule 为随机策略
04.      return  new RandomRule();
05.  }
```

2. 重启 micro-service-cloud-consumer-dept-80，使用浏览器访问“http://eureka7001.com/consumer/dept/get/1”，结果如下图。

图5：切换负载均衡策略为随机

通过图 5 中 dbSource 字段取值的变化可以看出，我们已经将负载均衡策略切换为 RandomRule（随机）。

定制负载均衡策略

通常情况下，Ribbon 提供的这些默认负载均衡策略是可以满足我们的需求的，如果有特殊的要求，我们还可以根据自身需求定制负载均衡策略。

下面我们就来演示下如何定制负载均衡策略。

1. 在 micro-service-cloud-consumer-dept-80 中新建一个 net.biancheng.myrule 包，并在该包下创建一个名为 MyRandomRule 的类，代码如下。

```
01.  package net.biancheng.myrule;
02.
03.  import com.netflix.client.config.IClientConfig;
04.  import com.netflix.loadbalancer.AbstractLoadBalancerRule;
05.  import com.netflix.loadbalancer.ILoadBalancer;
06.  import com.netflix.loadbalancer.Server;
07.
08.  import java.util.List;
09.
10.  /**
11.  * 定制 Ribbon 负载均衡策略
12.  */
13.  public class MyRandomRule extends AbstractLoadBalancerRule {
14.      private int total = 0;            // 总共被调用的次数，目前要求每台被调用5次
15.      private int currentIndex = 0;    // 当前提供服务的机器号
16.
17.      public Server choose(ILoadBalancer lb, Object key) {
18.          if (lb == null) {
19.              return null;
20.          }
21.          Server server = null;
22.
23.          while (server == null) {
24.              if (Thread.interrupted()) {
25.                  return null;
26.              }
27.              //获取所有有效的服务实例列表
28.              List<Server> upList = lb.getReachableServers();
29.              //获取所有的服务实例的列表
30.              List<Server> allList = lb.getAllServers();
31.
32.              //如果没有任何的服务实例则返回 null
33.              int serverCount = allList.size();
34.              if (serverCount == 0) {
35.                  return null;
36.              }
37.              //与随机策略相似，但每个服务实例只有在调用 3 次之后，才会调用其他的服务实例
38.              if (total < 3) {
```

```
                server = upList.get(currentIndex);
                total++;
            } else {
                total = 0;
                currentIndex++;
                if (currentIndex >= upList.size()) {
                    currentIndex = 0;
                }
            }
            if (server == null) {
                Thread.yield();
                continue;
            }
            if (server.isAlive()) {
                return (server);
            }
            server = null;
            Thread.yield();
        }
        return server;
    }

    @Override
    public Server choose(Object key) {
        return choose(getLoadBalancer(), key);
    }

    @Override
    public void initWithNiwsConfig(IClientConfig clientConfig) {
        // TODO Auto-generated method stub
    }
}
```

2. 在 net.biancheng.myrule 包下创建一个名为 MySelfRibbonRuleConfig 的配置类，将我们定制的负载均衡策略实现类注入到容器中，代码如下。

```
package net.biancheng.myrule;

import com.netflix.loadbalancer.IRule;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
* 定制 Ribbon 负载均衡策略的配置类
* 该自定义 Ribbon 负载均衡策略配置类 不能在 net.biancheng.c 包及其子包下
* 否则所有的 Ribbon 客户端都会采用该策略，无法达到特殊化定制的目的
*/
```

```
@Configuration
public class MySelfRibbonRuleConfig {
    @Bean
    public IRule myRule() {
        //自定义 Ribbon 负载均衡策略
        return new MyRandomRule(); //自定义，随机选择某一个微服务，执行五次
    }
}
```

3. 修改位于 net.biancheng.c 包下的主启动类，在该类上使用 @RibbonClient 注解让我们定制的负载均衡策略生效，代码如下。

```
package net.biancheng.c;

import net.biancheng.myrule.MySelfRibbonRuleConfig;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;

@SpringBootApplication
@EnableEurekaClient
//自定义 Ribbon 负载均衡策略在主启动类上使用 RibbonClient 注解，在该微服务启动时，就能自动去加载我们自定义的 Ribbon 配置类，从而是配置生效
// name 为需要定制负载均衡策略的微服务名称（application name）
// configuration 为定制的负载均衡策略的配置类，
// 且官方文档中明确提出，该配置类不能在 ComponentScan 注解（SpringBootApplication 注解中包含了该注解）下的包或其子包中，即自定义负载均衡配置类不能在 net.biancheng.c 包及其子包下
@RibbonClient(name = "MICROSERVICECLOUDPROVIDERDEPT", configuration = MySelfRibbonRuleConfig.class)
public class MicroServiceCloudConsumerDept80Application {

    public static void main(String[] args) {
        SpringApplication.run(MicroServiceCloudConsumerDept80Application.class, args);
    }
}
```

4. 重启 micro-service-cloud-consumer-dept-80，使用浏览器访问“http://eureka7001.com/consumer/dept/get/1”，结果如下图。

图6：定制负载均衡策略

通过图 6 中 dbSource 字段取值的变化可以看出，我们定制的负载均衡策略已经生效。

< 上一节　　下一节 >

推荐阅读

C++函数模板（模板函数）详解
C++构造函数详解
编译型语言和解释型语言的区别
CSS box-sizing：改变盒子模型
C语言free()：释放堆内存
C语言中的文件是什么?
UE4新建项目
Redis HKEYS命令
JSON注释
《Web前端开发与应用教程（HTML5+CSS3+JavaScript）》PDF下载