# 5 - Ribbon with RestTemplate

Spring Cloud Ribbon is a set of client load balancing and service invocation tools based on Netflix Ribbon.

Netflix Ribbon is an open source component released by Netflix. Its main function is to provide load balancing algorithms and service calls for clients. Spring Cloud integrates it with other open source service components in Netflix (such as Eureka, Feign, Hytrix, etc.) into the Spring Cloud Netflix module, which is called the Spring Cloud Netflix Ribbon after integration.

The ribbon is a sub-module of the Spring Cloud Netflix module. It is the secondary encapsulation of the Netflix ribbon by Spring Cloud. Through it, we can transform the service oriented REST template request into a service call for client load balancing.

The ribbon is one of the most core and important components in the Spring Cloud system. Although it is only a tool type framework and does not need to be deployed independently like Eureka Server (service registry), it exists in almost every microservice built using Spring Cloud.

The call between the Spring Cloud microservices and the request forwarding of the API gateway are actually implemented through the Spring Cloud Ribbon, including the OpenFeign we will introduce later.

**Ribbon implements service invocation**

Ribbon can be used in conjunction with RestTemplate (RestTemplate) to realize the call between microservices.

RestTemplate is a request framework in the Spring family for consuming third-party REST services. RestTemplate implements the encapsulation of HTTP requests and provides a set of templated service invocation methods. Through it, Spring applications can easily access various types of HTTP requests.

The RestTemplate provides corresponding methods for processing various types of HTTP requests, such as HEAD, GET, POST, PUT, DELETE and other types of HTTP requests, which correspond to the headForHeader(), getForObject(), postForObject(), put() and delete() methods in the RestTemplate.

Let's use a simple example to demonstrate how the ribbon implements service invocation.

1. Under the main project DataEngineSwarm, create a microservice named micro-service-cloud-consumer-dept-80, and introduce the required dependencies in its pom.xml. The code is as follows.

```
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
```

```
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <!--parent project-->
    <parent>
      <groupId>com.luxbp</groupId>
      <artifactId>DataEngineSwarm</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </parent>
    <groupId>com.luxbp</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>
        <!--common module-->
        <dependency>
            <groupId>com.luxbp</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 test-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--Spring Cloud Eureka client depen -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <!--Spring Cloud Ribbon depen-->
        <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. In the classpath (i.e./resource directory), create a new configuration file, application.yml, with the following configuration contents.

```
server:
  port: 80 #port num
############################################## Spring Cloud Ribbon LB
config#########################
eureka:
  client:
    register-with-eureka: false
    fetch-registry: true
    service-url:
     defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:
7003/eureka/
```

3. Under the com.luxbp.config package, create a configuration class named ConfigBean and inject the RestTemplate into the container. The code is as follows.

```
package com.luxbp.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;

@Configuration
public class ConfigBean {

    @Bean //inject RestTemplate
    @LoadBalanced //open LB
    public RestTemplate getRestTemplate() {
```

```
        return new RestTemplate();
    }
}
```

4. Under the com.luxbp.controller package, create a DeptController_ The Controller of the Consumer. The request defined in the Controller is used to call the service provided by the server. The code is as follows.

```
package com.luxbp.controller;
import com.luxbp.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://MICROSERVICECLOUDPROVIDERDEPT";
    @Autowired
    private RestTemplate restTemplate;
    @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);
    }

    @RequestMapping(value = "/consumer/dept/list")
    public List<Dept> list() {
        return restTemplate.getForObject(REST_URL_PROVIDER_PREFIX + "/dept/list",
List.class);
    }
}
```

5. On the main startup class of micro-service-cloud-consumer-dept-80, use the @ EnableEurekaClient annotation to enable the Eureka client function. The code is as follows.

```
package com.luxbp;
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. Start the service registry micro-service-cloud-eureka-7001, the service provider micro-service-cloud-provider-dept-8001, and the service consumer micro-service-cloud-consumer-dept-800.

7. Now we should see result if visit http://eureka7001.com:80/consumer/dept/list

**Ribbon realizes load balancing**

Ribbon is a client-side load balancer, which can be used with Eureka to easily achieve client-side load balancing. Ribbon will first obtain the list of servers from Eureka Server (service registry), and then allocate requests to multiple servers through load balancing strategy to achieve load balancing.

The Spring Cloud Ribbon provides an IRule interface, which is mainly used to define load balancing policies. It has seven default implementation classes, each of which is a load balancing policy.

| | |
|---|---|
| RoundRobinRule | According to the linear polling strategy, that is, select service instances in a certain order |
| RandomRule | Select a service instance at random |
| RetryRule | The service is obtained according to the policy of RoundRobin Rule (polling). If the obtained service instance is null or has expired, it will be retried continuously within the specified time (the policy for obtaining the service during the retry is still the policy defined in RoundRobin Rule). If the service instance is not obtained after the specified time, null will be returned. |
| WeightedResponseTimeRule | WeightedResponseTimeRule is a subclass of RoundRobin Rule, which extends the function of RoundRobin Rule. The weight of all service instances is calculated according to the average response time. The shorter the response time, the higher the weight of the service instance, and the greater the probability of being selected. At the beginning, if the statistical information is insufficient, use the linear polling strategy. When the information is |

| | |
|---|---|
| | sufficient, switch to WeightedResponseTimeRule. |
| BestAvailableRule | Inherited from ClientConfigEnabledRoundRobinRule. First filter the service instances with point failures or failures, and then select the service instance with the smallest amount of concurrency. |
| AvailabilityFilteringRule | First filter out the failed or failed service instances, and then select the service instances with smaller concurrency. |
| ZoneAvoidanceRule | The default load balancing policy is used to comprehensively judge the performance of the service zone and the availability of the service to select the service instance. In the absence of regions, this policy is similar to the RandomRule policy. |

Next, we will use an instance to verify what policy the ribbon uses to select service instances by default.
1. Execute the following SQL statements in MySQL database to prepare test data.

```
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', 'dev', DATABASE());
INSERT INTO `dept` VALUES ('2', 'hr', DATABASE());
INSERT INTO `dept` VALUES ('3', 'fin', DATABASE());
INSERT INTO `dept` VALUES ('4', 'marketing', DATABASE());
INSERT INTO `dept` VALUES ('5', 'dev-ops', DATABASE());

```

```
21 ##############################################################################################
   #
22 DROP DATABASE IF EXISTS spring_cloud_db3;
23
24 CREATE DATABASE spring_cloud_db3 CHARACTER SET UTF8;
25
26 USE spring_cloud_db3;
27
28 DROP TABLE IF EXISTS `dept`;
29 CREATE TABLE `dept` (
30   `dept_no` int NOT NULL AUTO_INCREMENT,
31   `dept_name` varchar(255) DEFAULT NULL,
32   `db_source` varchar(255) DEFAULT NULL,
33   PRIMARY KEY (`dept_no`)
34 ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
35
36 INSERT INTO `dept` VALUES ('1', 'dev', DATABASE());
37 INSERT INTO `dept` VALUES ('2', 'hr', DATABASE());
38 INSERT INTO `dept` VALUES ('3', 'fin', DATABASE());
39 INSERT INTO `dept` VALUES ('4', 'marketing', DATABASE());
40 INSERT INTO `dept` VALUES ('5', 'dev-ops', DATABASE());
```

2. Refer to micro-service-cloud-provider-dept-8001, and create two microservices: micro-service-cloud-provider-dept-8002 and micro-service-cloud-provider-dept-8003. Modify the port number, database connection information and custom service name information (eureka. instance. instance id) in application.yml in micro-service-cloud-provider-dept-8002 and 8003

3. Start micro-service-cloud-eureka-7001/7002/7003 (service registry cluster), micro-service-cloud-provider-dept-8001/8002/8003 (service provider cluster) and micro-service-cloud-consumer-dept-800 (service consumer).

**Switch LB Strategy and modify specific LB**
**wait to be updated**