Spring Cloud

-with Eureka, Ribbon, Feign

Table of Contents

Preface

Maven dependencies could be different in different Spring cloud version, please check Spring Cloud versions from [1] https://spring.io/projects/spring-cloud, if there are any compiling issues.

In Maven pom.xml file, giving <spring-cloud.version> property, skip the dependencies version; Spring cloud will figure out the dependencies versions.

Using Ribbon as load balancer

Architecture

Netflix Ribbon is a Client Side Load Balancer. It serves as a proxy to user clients. During initialization, it registers a list of services instances. User requests send to Ribbon, where it decides which service will be used based on different algorithms, the default is round robin.

One potential issue is when a service instance is down, Ribbon may not know unless you override the ribbon configuration to detect services down issue. By default, when a service instance is down, and the request is forwarded to this service instance via Rest template, you will get service unavailable response. Use Feign will not get this issue.

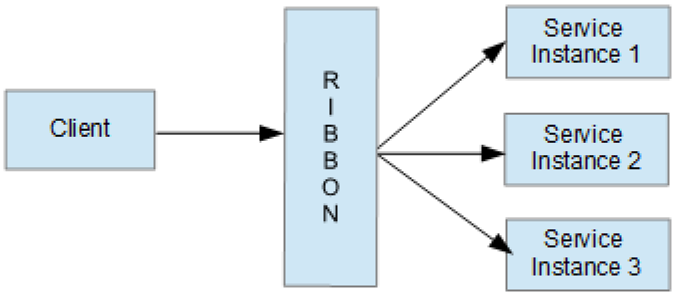


Figure 1 Ribbon Load Balancer

Below will show a demo how to implement Ribbon load balancer. We will show how to use RestTemplate, and Feign to redirect requests. Code can found at:

https://github.com/dsong99/Notes/tree/main/Java/spring-cloud/my-cloud-ribbon-feign_1

Create services

Create a typical Spring Boot service, specify the name property in pom.xml, such as:

```
<artifactId>services</artifactId>
 <name>services</name>.
```

Create Service Proxy (Ribbon Load Balancer)

1. Maven pom.xml:

```xml
<spring-cloud.version>Finchley.RC2</spring-cloud.version>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

   Note that, different Spring cloud version, the dependencies artifact id could be different. Please check Spring Cloud page [1] when having compile issues.

2. In application.proerties, disable Eureka, provide a list of services, and this proxy port number:

```
services.ribbon.eureka.enabled=false
services.ribbon.listOfServers=localhost:8081,localhost:8082,localhost:8083
services.ribbon.ServerListRefreshInterval=2000
server.port=9090
```

3. In application or controller, Register Ribbon client with a name (the same as the service name specified in the service pom.xml, artifactid or name property), so this name can be used to redirect requests to services via Rest Template or Feign (enable Feign Client).

```java
@RibbonClient(name = "services")
```

   You can provide a customized Ribbon configuration in the above line. We use the default configuration here.

4. Feign Client proxy
   Using Feign Client proxy is simpler than using Rest Template. It can also detect failed services and forward request to available services. In the same scenario, rest template will return service not available in case of using default configuration.

```java
@FeignClient(name = "services")
public interface FeignClientProxy {

  @RequestMapping("simple-services/hello")
```

```
    public String hello();
}
```

Feign proxy client simply given a service name, and redirect URL for each method.

Using Eureka, Ribbon

Architecture

Eureka is a service that provides services discovery. It contains Ribbon as it's default load balancer. Each service and the Ribbon load balancer register them self to Eureka during initialization. Eureka will maintain a list of available services and detect failed service using heartbeat. During runtime, when Ribbon receives a request, it will ask Eureka for an available service and redirect request to that service.

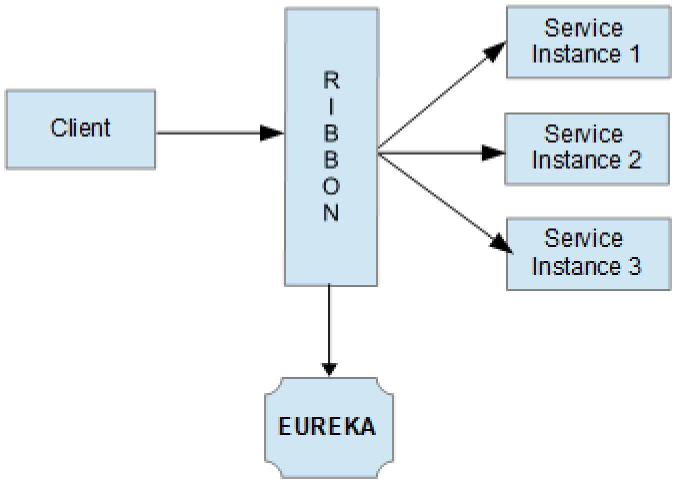


Figure 2 Eureka, Ribbon, Services

The demo code can be found at:

https://github.com/dsong99/Notes/tree/main/Java/spring-cloud/my-cloud-ribbon-eureka_2

Create Eureka Naming Service

1. In spring boot application, add `@EnableEurekaServer`
2. In application property file, specify port number, application name, and disable eureka client (to prevent runtime errors during starting service)

Maven dependency:

```
<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
```

Start the service, open Eureka server page at: http://localhost:8761/

Create services

1. In spring boot application, add `@EnableEurekaClient`
2. In application property file,
   - specify port number, application name, and enable eureka client
   - specify Eureka server url and enable client registry to true.

Maven dependency:

```
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

Run the services on different port numbers, such as 8081, 8082, 8083.

Service Console message:

```
DiscoveryClient_SERVICES/DESKTOP-F20CDKS.myfiosgateway.com:services:8081: registering service...
```

Eureka console message:

```
Registered instance SERVICES/DESKTOP-F20CDKS.myfiosgateway.com:services:8081 with status UP
```

Check Eureka server page, we can see services are registered in Eureka server.

Test: http://localhost:8082/simple-services/hello

Create Service Proxy

Eureka default load balancer is Ribbon, so we don't need to specify any ribbon related code or configurations.

The configuration is the same as in Services, @EnableEurekaClient and specify Eureka server URL.

Maven dependency:

```
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

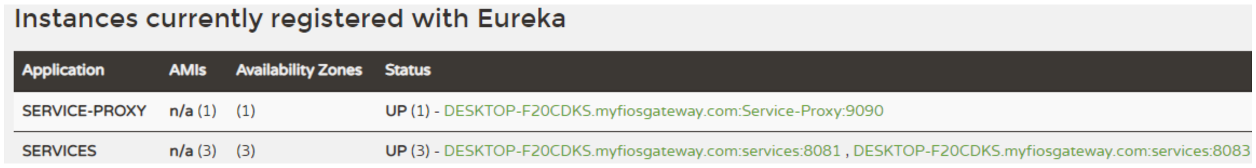
Console message:

DiscoveryClient_SERVICE-PROXY/DESKTOP-F20CDKS.myfiosgateway.com:Service-Proxy:9090: registering service...

Eureka console message:

Registered instance SERVICE-PROXY/DESKTOP-F20CDKS.myfiosgateway.com:Service-Proxy:9090 with status UP

Check Eureka server page, we can see services are registered in Eureka server:



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SERVICE-PROXY	n/a (1)	(1)	UP (1) - DESKTOP-F20CDKS.myfiosgateway.com:Service-Proxy:9090
SERVICES	n/a (3)	(3)	UP (3) - DESKTOP-F20CDKS.myfiosgateway.com:services:8081 , DESKTOP-F20CDKS.myfiosgateway.com:services:8083

Run:

http://localhost:9090//simple-services/hello

http://localhost:9090//simple-services/hello-feign

We can see the messages show on the page with a different port number for each refresh the page, 8081, 8082, …

References

1. https://spring.io/projects/spring-cloud
2. https://spring.io/projects/spring-cloud-netflix
3.