4 - Eureka Cluster

In the microservice architecture, a system is often composed of a dozen or even dozens of services. If all these services are registered in the same Eureka Server, it is very likely that the Eureka Server will crash due to overwhelm, and eventually the entire system will be paralyzed. . The most direct way to solve this problem is to deploy a Eureka Server cluster.

We know that there are three roles involved in implementing service registration and discovery in Eureka: Discover Server, service provider, and service consumer. These three roles have a clear division of labor and perform their duties. But in fact, in Eureka, all services are both service consumers and service providers, and the service registry Eureka Server is no exception.

When we built the service registry, we involved such a configuration in application.yml:

```
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

The reason for this setting is that micro-service-cloud-eureka-7001 itself is the Discover Server. The Discover Server cannot register itself with itself, but the Discover Server can register itself with other Discover Server as a service.

For example, there are two Eureka Servers named A and B. Although A cannot register itself with A, and B cannot register itself with B, A can register itself with B as a service. Similarly, B can also register itself to A.

In this way, a group of Eureka Server clusters registered with each other can be formed. When the service provider sends a registration request to the Eureka Server, the Eureka Server will forward the request to all the Eureka Servers connected to it in the cluster to achieve service synchronization between Eureka Servers.

Through service synchronization, service consumers can obtain services provided by service providers on any Eureka server in the cluster. In this way, even if a service registry in the cluster fails, the service consumer can still obtain service information from other Eureka servers in the cluster and call it without causing the overall system paralysis, which is the high availability of Eureka Server cluster.

3.1 Set up an Eureka Cluster

Refer to the building process of micro service cloud-eureka-7001, and create two additional Eureka Servers under the main project: micro service cloud-eureka-7002 and micro service cloud-eureka-7003. At this time, the three Eureka Servers are the same in terms of Maven dependency, code, and configuration.

Modify the configuration of application.yml in micro-service-cloud-eureka-7001, micro-service-cloud-eureka-7002, and micro-service-cloud-eureka-7003. The specific configuration is as follows.

The configuration of application.yml in micro-service-cloud-eureka-7001 is as follows

```yaml
server:
  port: 7001  #The port number of the Module
eureka:
  instance:
    hostname: eureka7001.com
  client:
    register-with-eureka: false #false means do not register itself with the registry.
    fetch-registry: false #False means that my end is the registration center, and my responsibility is to maintain the service instance, and there is no need to retrieve the service
    service-url:
      # defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/ #Stand-alone service registration center
      defaultZone: http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/ #cluster version. Register the current Eureka Server to 7003 and 7003 to form a group of Eureka Server clusters registered with each other
```

The configuration of application.yml in micro-service-cloud-eureka-7002 is as follows

```yaml
server:
  port: 7002
eureka:
  instance:
    hostname: eureka7002.com
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7003.com:7003/eureka/
```

The configuration of application.yml in micro-service-cloud-eureka-7003 is as follows

```yaml
server:
  port: 7003
eureka:
  instance:
    hostname: eureka7003.com
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/
```

Since we are building the Eureka Server cluster locally, we need to modify the local host file, add following to host file

```
127.0.0.1 eureka7001.com
```

```
127.0.0.1 eureka7002.com
127.0.0.1 eureka7003.com
```

Modify the value of eureka.client.service-url.defaultZone in the micro-service-cloud-provider-dept-8001 (service provider) configuration file application.yml, and register the service to the Eureka Server cluster. The specific configuration is as follows.

```
eureka:
  client: #Register the client into the eureka service list
    service-url:
      # defaultZone: http://localhost:7001/eureka  #This address is the registration address exposed by the 7001 registration center in application.yml (stand-alone version)
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/  #Register the service with the Eureka Server cluster
```

Run micro-service-cloud-eureka-7001, and use the browser to access“ http://eureka7001.com:7001/ ” From the web page, we can see that the services of the service provider (microservice-cloud-provider-dept-8001) have been registered with Eureka Server 7001, and the DS Replicas option also shows the other two Eureka servers in the cluster: Eureka Server 7002 and Eureka Server 7003. Run 7002 and 7003 and get the same result.

<rus result Pic>

Note: Now that we are visiting the service through Eureka. We have three eureka servers now. But by visiting http://eureka7001.com:7001/, we are making use of only one of them.
We should also visit http://eureka7002.com:7002/ or http://eureka7003.com:7003/, in this way, we can make use of all three eureka server. But it is not a good way for user. User should remember a list of eureka address. So we need a gateway, and a Load Balancer. we will discuss more about this later.