3

# 第 3 章

# 实例 1：用 Spring Cloud 实现一个微服务系统

组成微服务系统的组件非常多，各个组件之间相互协作，共同支撑微服务系统的稳定可靠的运行。本章通过一个实例来让读者了解什么是微服务系统，并引导读者实现一个比较简单的微服务系统。

## 3.1 本实例的架构和实现步骤

本实例将实现 1 个“服务中心”集群、1 个“服务提供者”集群和 1 个“服务消费者”，架构如图 3-1 所示。本实例通过“服务中心”Eureka 来进行服务治理，“服务消费者”调用“服务提供者”提供的服务。

从图 3-1 中可以看到，Service Provider1 和 Service Provider2 是微服务系统中的“服务提供者”，在本实例中它们被注册到“服务中心”组成“服务提供者”集群。

“服务消费者”在从“服务中心”获得“服务提供者”信息后调用此集群的功能。

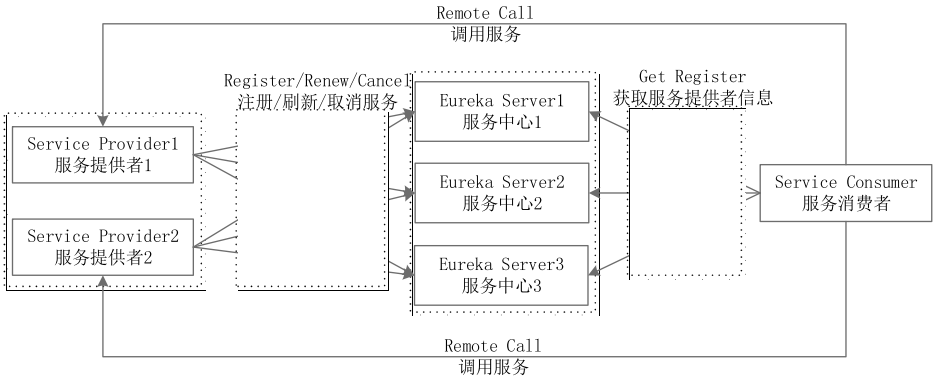


图 3-1　本实例的架构图

本实例的实现步骤为：

（1）用 Eureka 实现一个“服务中心”集群。

（2）通过 Eureka 的“服务注册”功能来注册“服务提供者”，以实现“服务提供者”集群。

（3）“服务消费者”不需要注册到“服务中心”中，它使用 Eureka 的“服务注册”功能根据“服务提供者”注册的名称来调用客户端 Feign 以消费“服务提供者”提供的接口。

（4）在完成所有的开发工作完成后，先启动“服务中心”集群、“服务提供者”集群，然后启用“服务消费者”客户端来测试和消费服务。

经过几年的快速、稳定发展，Eureka 的功能日趋完善，用户群体庞大。因此，本实例使用 Eureka 进行讲解，本实例基本上实现了以 Eureka 作为服务治理的大部分功能。但是，由于 Eureka 在 2018 年停止维护，前景难以预测，因此后面章节中的服务治理项目采用的是目前流行的 Consul。

## 3.2　创建 Spring Cloud 项目

这里使用 2.3.3 节安装好的 Spring Assistant 创建 Spring Cloud 项目。

（1）启动开发工具 IDEA。

（2）单击 IDEA 菜单栏中的“File→New→Project”命令，在弹出窗口中选择“Spring Assistant”。

（3）选中“Default”选项，单击“Next”按钮，在弹出的窗口中将“Project name”修改为“Eureka Server Demo”，然后单击“Next”按钮。

（4）在弹出的窗口中找到“Spring Boot version”（由于 Spring Cloud 基于 Spring Boot，所以这里显示的是 Spring Boot 的版本），然后在其右边的下拉框中选择要创建的 Spring Cloud 版本，这里选择的是“2.1.13”，如图 3-2 所示。

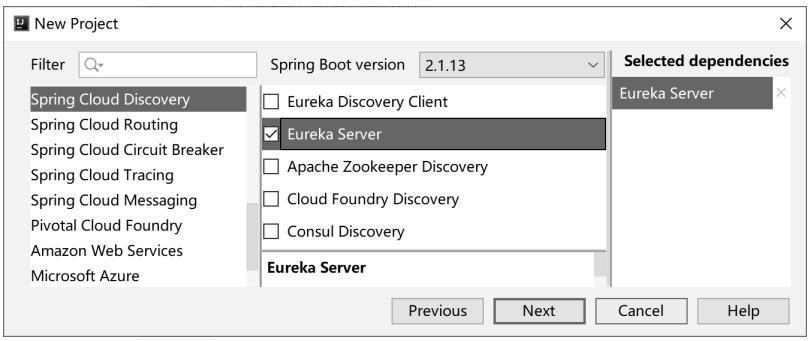


图 3-2　选择版本和依赖

如果要添加依赖，则在这个窗口中进行如下操作。

（1）单击左边框的“Spring Cloud Discovery”按钮，在窗口中间弹出的列表中勾选“Eureka Server”复选框，选中后 Spring Cloud 就会自动添加 Eureka Server 的依赖并进行下载。可以一次选择多个依赖，选择的内容将显示在右边的“Selected dependencies”下方。

（2）在自己的 IDEA 界面中单击“Next”按钮，在弹出的窗口中选择保存项目的目录（不要有中文路径），然后单击“Finish”按钮启动项目。

至此成功创建了 Spring Cloud 项目，并添加了 Eureka 的服务器端依赖。

还有另外一种创建 Spring Cloud 项目的方式（这种方式不太常用）：通过网络访问官方提供的在线项目创建器 Spring Initializr，在界面中根据自己的需要选择依赖、版本和配置，然后单击“创建”按钮生成工程包。在生成完成后，将工程包下载到本地电脑上并导入开发软件中即可使用。

## 3.3　用 Eureka 实现“服务中心”

一个完整的微服务系统需要用“服务中心”来统一治理服务。本节将实现一个高可用、可用于生产的“服务中心”集群。

代码　本实例的源码在本书配套资源的“/Chapter03/Eureka Server Demo”目录下。

### 3.3.1　添加配置

在 3.2 节创建的 Spring Cloud 项目中已经自动被添加了“Eureka Server”依赖，见以下代码：

```
<!--Spring Cloud 启动的依赖-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>
<!-- Eureka 服务器端的依赖-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

接下来需要配置“服务中心”的地址、端口号和应用名称，见以下代码：

```
#应用名称
spring.application.name=Eureka Server Demo
#服务器的端口号
server.port=8080
#是否注册到 Eureka Server，默认为 true
eureka.client.register-with-eureka=true
#是否从 Eureka Server 获取注册信息，默认为 true
eureka.client.fetch-registry=true
#设置与 Eureka Server 交互的地址。查询服务和注册服务都需要依赖这个地址。多个地址可使用“，”分隔
${server.port}代表配置的是 8080 端口
eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/
```

至此完成了单点的“服务中心”，它已经可以正常地提供注册和发现功能。

最好不要在生产环境中使用单点“服务中心”，因为：“服务中心”是非常关键的组件，如果采用的是单点“服务中心”，若“服务中心”遇到故障，则很可能导致整个微服务系统无法正常工作，甚至造成系统崩溃。为了维持系统的高可用性，必须使用“服务中心”集群。

用 Eureka 部署集群非常方便，只需要在“服务中心”（Eureka Server）中配置其他可用的“服务中心”地址即可实现高可用部署。下面介绍“服务中心”集群的实现。

### 3.3.2 实现“服务中心”集群（满足高可用）

因为大部分读者只有一台电脑，所以本节演示如何在一台电脑上实现“服务中心”集群。这里以模拟 3 台服务器来实现集群，具体规划见表 3-1。

表 3-1 单台电脑上的“服务中心”集群的具体规划

| 访问 URI | IP（本机） | 节点名称 | 端口号 |
| --- | --- | --- | --- |
| http://node1:8081 | 127.0.0.1 | node1 | 8081 |
| http://node2:8082 | 127.0.0.1 | node2 | 8082 |
| http://node3:8083 | 127.0.0.1 | node3 | 8083 |

**1. 配置虚拟地址**

在 C:\Windows\System32\drivers\etc 下的 hosts 文件中添加以下代码并保存（注意提供修改权限）：

```
#节点 node1
127.0.0.1 node1
#节点 node2
127.0.0.1 node2
#节点 node3
127.0.0.1 node3
```

DNS 的作用是将域名解析到对应的 IP 地址。比如，域名 www.baidu.com 对应的其中一个地址是 183.232.231.172（有可能有多个服务器），访问 www.baidu.com 实际上就是访问 IP 地址为 183.232.231.172 的服务器。

在做好上面配置后，访问 http://node1、http://node2、http://node3 都是访问 IP 地址为 127.0.0.1 的主机（即本机）。需要在域名地址后加上端口号，这样才能出现“服务中心”的管理界面。

**2. 多环境配置**

下面通过创建多个配置文件来模拟多个“服务中心”。

（1）创建节点 node1 的配置文件 application-node1.properties，并将“serviceUrl”指向节点 node2 和 node3。见以下代码：

```
#应用的名称
spring.application.name=Eureka Server Demo
#应用的端口号
server.port=8081
#节点的名称
eureka.instance.hostname=node1
#是否注册到 Eureka Server，默认为 true
eureka.client.register-with-eureka=true
#是否从 Eureka Server 获取注册信息，默认为 true
eureka.client.fetch-registry=true
#设置与 Eureka Server 进行交互的地址，查询服务和注册服务都需要使用它。多个地址可使用“,”分隔
eureka.client.serviceUrl.defaultZone=http://node2:8082/eureka/,http://node3:8083/eureka/
```

其中，通过配置项“eureka.client.serviceUrl.defaultZone”来配置多个“服务中心”。Eureka 的默认地址是 http://URL:8761/eureka。

（2）创建节点 node2 的配置文件 application-node2.properties，将“serviceUrl”指向节点 node1 和 node3。见以下代码：

```
#应用的名称
spring.application.name=Eureka Server Demo
#应用的端口号
server.port=8082
#节点的名称
eureka.instance.hostname=node2
#是否注册到 Eureka Server，默认为 true
eureka.client.register-with-eureka=true
#是否从 Eureka Server 获取注册信息，默认为 true
eureka.client.fetch-registry=true
#设置与 Eureka Server 进行交互的地址，查询服务和注册服务都需要使用它。多个地址可使用“,”分隔
eureka.client.serviceUrl.defaultZone=http://node1:8081/eureka/,http://node3:8083/eureka/
```

（3）创建节点 node3 的配置文件 application-node3.properties，并将“serviceUrl”指向节点 node1 和 node2。见以下代码：

```
#应用的名称
spring.application.name=Eureka Server Demo
#应用的端口号
server.port=8083
#节点的名称
eureka.instance.hostname=node3
#是否注册到 Eureka Server，默认为 true
eureka.client.register-with-eureka=true
#是否从 Eureka Server 获取注册信息，默认为 true
eureka.client.fetch-registry=true
#设置与 Eureka Server 进行交互的地址，查询服务和注册服务都需要使用它。多个地址可使用“,”分隔
eureka.client.serviceUrl.defaultZone=http://node1:8081/ eureka /,http ://node2: 8082/eureka/
```

### 3.3.3 打包和部署“服务中心”

#### 1. 打包成可执行的 JAR 包

在开发完成后，可以直接用 IDEA 将应用打包成 JAR（WAR）包，以便在多服务器多配置环境中运行。

下面以打包成 JAR 包为例介绍具体操作步骤。

（1）单击 IDEA 右侧上方的“Maven”按钮，在弹出的窗口中单击“lifeCycle→clean”命令，IDEA 就会运行“clean”命令。

（2）待控制台提示完成后，单击 IDEA 右侧上方的“Maven”按钮，在弹出的窗口中单击“lifeCycle→package”命令，此时控制台会出现执行情况的提示信息，然后提示执行完成：

```
[INFO] --- maven-jar-plugin:3.1.2:jar (default-jar) @ demo ---
[INFO] Building jar: I:\Spring Cloud\code\Eureka\Eureka Server\target\demo-0.0.1-SNAPSHOT.jar
```

```
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.7.RELEASE:repackage (repackage) @ demo ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

在上述信息中，“Building jar:”后就是 JAR 包的目录地址和名称。上述信息显示当前 JAR 包的位置是“I:\Spring Cloud\code\Eureka\Eureka Server\target\”，名称是“demo-0.0.1-SNAPSHOT.jar”。

**2. 部署“服务中心”集群**

接下来进入打包后的 JAR 包目录，分别以 node1、node2、node3 的配置参数启动“服务中心”Eureka。具体步骤如下。

（1）进入 JAR 目录，在地址栏输入“cmd”命令，按 3 次 Enter 键，即可打开 3 个 DOS 命令窗口。

（2）在 3 个 DOS 命令窗口中分别输入以下命令以启动服务中心。

命令 1：

```
#以 node1 的配置信息启动 Eureka 节点 1
java -jar demo-0.0.1-SNAPSHOT.jar --spring.profiles.active=node1
```

命令 2：

```
#以 node2 的配置信息启动 Eureka 节点 2
java -jar demo-0.0.1-SNAPSHOT.jar --spring.profiles.active=node2
```

命令 3：

```
#以 node3 的配置信息启动 Eureka 节点 3
java -jar demo-0.0.1-SNAPSHOT.jar --spring.profiles.active=node3
```

如果在启动 JAR 包前已经在 IDEA 中启动了工程，则一定要将工程停掉，否则它会抢占端口，导致启动报错。

（3）待各个节点依次启动完成后在浏览器中输入“http://node1:8081”，会显示如图 3-3 所示的“服务中心”管理界面。

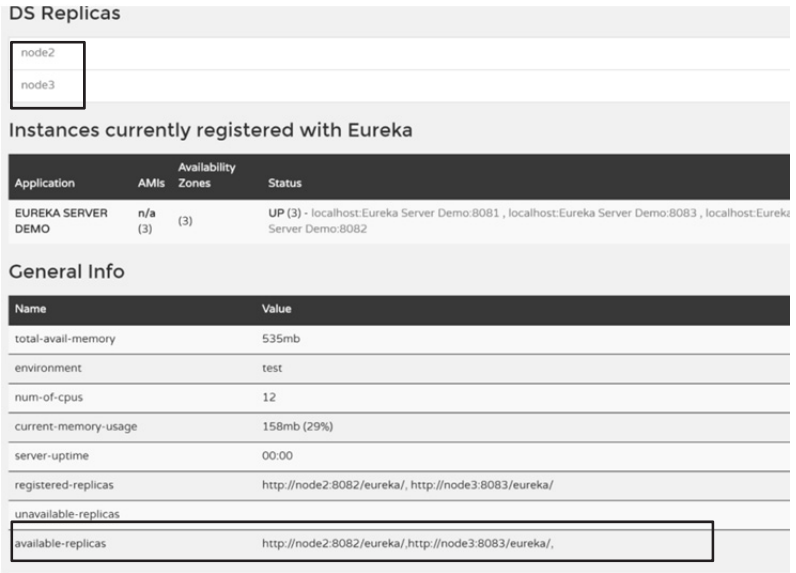


图 3-3 “服务中心”的管理界面

由图 3-3 可知，节点 node1 的“DS Replicas”中已经有了 node2、node3 的配置信息，并且在“available-replicas”中有 node1、node2、node3 的状态信息。如果停止 node2，则会发现 node2 被移动到“unavailable-replicas”一栏中，表示 node2 不可用。

至此多节点“服务中心”集群已经配置完成。

## 3.4 用 Eureka 实现“服务提供者”

在 3 .3 节已经实现了“服务中心”，接下来创建“服务提供者”并将其注册到“服务中心”以便“服务消费者”调用。

代码 本实例的源码在本书配套资源的“/Chapter03/Provider”目录下。

### 3.4.1 实现“服务提供者”的客户端

#### 1. 创建 Eureka 的客户端项目

参考 3.2 节中创建 Spring Cloud 项目的步骤创建 Eureka 的客户端项目。注意，依赖不是勾选“Eureka Server”，而是勾选“Eureka Discovery Client” 复选框。

#### 2. 添加依赖

在新创建好 Spring Cloud 项目后，需要在项目中的依赖文件中添加依赖。以下是添加依赖后的

代码：

```
<!-- Spring Boot 的 Web 依赖-->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--Eureka 客户端的依赖-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

### 3. 添加配置

在配置文件中配置应用的名称和端口号，以及“服务中心”的地址。“服务中心”的地址是在 3.3 节中完成的 3 个“服务中心”集群的地址，见以下代码：

```
#应用的名称
spring.application.name=provider
#应用的端口号
server.port=8000
provider.name=provider0
#“服务中心”的地址
eureka.client.serviceUrl.defaultZone=http://node1:8081/eureka/,http://node2:8082/eureka/,http://node3:8083/eureka/
```

### 4. 启用注册和发现

在启动类中添加注解@EnableDiscoveryClient，以启用服务的注册和发现功能，见以下代码：

```
//代表是 Spring Boot 应用程序的入口类
@SpringBootApplication
//启用客户端的服务注册和发现功能
@EnableDiscoveryClient
public class ProviderApplication {
        public static void main(String[] args) {
                SpringApplication.run(ProviderApplication.class, args);
        }
}
```

要用 Eureka 启用客户端服务的注册和发现功能，可以使用注解@EnableEurekaClient 和@EnableDiscoveryClient 来实现。如果是使用其他“服务中心”（如 Zookeeper、Consul），则只能使用注解@EnableDiscoveryClient（它是 Spring Cloud Common 自带的一个注解）来实现。注解@EnableEurekaClient 是 Eureka 的专用注解。

### 3.4.2 实现“服务提供者”的接口

该服务接口用于返回字符串信息，以供“服务消费者”调用，具体见以下代码：

```
/**服务接口*/
@RestController
public class HelloController {
    /*注入“服务提供者”的名称*/
    @Value("${provider.name}")
    private String name;
    /*注入“服务提供者”的端口号*/
    @Value("${server.port}")
    private String port;
    /*提供的接口，用于返回信息*/
    @RequestMapping("/hello")
    public String hello() {
        String str="provider:" + name + " port:" + port;
        //返回数据
        return str;
    }
}
```

其中的注解@Value 用于获取配置的信息，将会在 5.5 节详细讲解。

### 3.4.3 检查服务的有效性

（1）启动“服务中心”集群，并启动“服务提供者”应用。

（2）进入“服务中心”控制台 http://node1:8081/，查看“服务提供者”是否已经在“服务中心”注册过。如果已注册过，则在“Instances currently registered with Eureka”标题栏下方会出现“服务提供者”的应用名称。

（3）访问服务接口 http://localhost:8000/hello，可以看到返回如下消息：

```
provider:provider0 port:8000
```

这说明“服务提供者”的接口正在正常工作。

### 3.4.4 实现“服务提供者”集群

生产环境中的“服务提供者”也需要是集群，本节演示如何实现“服务提供者”集群。

#### 1. 多环境配置

创建配置文件 application-provider1.properties，加入以下代码：

```
# “服务提供者”的名称
```

```
spring.application.name=provider
#“服务提供者”的端口号
server.port=8001
#自定义的配置项
provider.name=provider1
#“服务中心”的地址
eureka.client.serviceUrl.defaultZone=http://node1:8081/eureka/,http://node2:8082/eureka/,http://node3:8083/eureka/
```

创建配置文件 application-provider2.properties，加入以下代码：

```
# “服务提供者”的名称
spring.application.name=provider
#“服务提供者”的端口号
server.port=8002
#自定义的配置项
provider.name=provider2
#“服务中心”的地址
eureka.client.serviceUrl.defaultZone=http://node1:8081/eureka/,http://node2:8082/eureka/,http://node3:8083/eureka/
```

**2. 打包启动**

分别以 provider1、provider2 的配置参数启动“服务提供者”。命令如下：

```
#打包
mvn clean package
#分别以 provider1 和 provider2 配置信息启动“服务提供者”
java -jar demo-0.0.1-SNAPSHOT.jar --spring.profiles.active=provider1
java -jar demo-0.0.1-SNAPSHOT.jar --spring.profiles.active=provider2
```

## 3.5　用 Feign 实现“服务消费者”

通过 3.3、3.4 节的操作已经创建好了“服务中心”和“服务提供者”，本节将实现“服务消费者”。

代码 本实例的源码在本书配套资源的“/Chapter03/Consumer”目录下。

### 3.5.1　用 Feign 实现“服务消费者”的客户端

**1. 创建 Eureka 的客户端项目**

参考 3.3 节的步骤创建 Eureka 的客户端项目（依赖不再勾选“Eureka Server”，而是和创建“服务提供者”一样勾选“Eureka Discovery Client”）。

**2. 添加依赖**

根据演示需要，添加 Web、Eureka 客户端和 Feign 客户端的依赖，见以下代码：

```
<!--Spring Boot 的 Web 依赖-->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--Eureka 客户端的依赖-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<!--Feign 客户端的依赖-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

**3. 添加配置**

如果“服务消费者”不对外提供服务接口，则不需要将其注册到“服务中心”。现在在“服务消费者”配置文件中配置“服务中心”的地址，见以下代码：

```
#“服务提供者”的名称
spring.application.name=consumer
#“服务提供者”的端口号
server.port=9000
#不注册到“服务中心”
eureka.client.register-with-eureka=false
#“服务中心”的地址
eureka.client.serviceUrl.defaultZone=node1:8081/eureka/,http://node2:8082/eureka/,http://node3:8083/eureka/
```

在默认情况下，如果添加了“服务中心”的依赖和地址，并启用了“服务发现”，则该“服务提供者”就会被注册到“服务中心”。如果该“服务提供者”不需要注册，则可以通过配置“eureka.client.register-with-eureka=false”来取消。

## 3.5.2 调用“服务提供者”的接口

**1. 启用客户端的发现和远程调用**

在启动类中，添加注解@EnableDiscoveryClient 以启用客户端的服务注册和发现功能，添加注解@EnableFeignClients 以启用 Feign 的远程服务调用。见以下代码：

```
//代表是 Spring Boot 应用程序的入口类
@SpringBootApplication
//启用客户端的服务注册和发现功能
@EnableDiscoveryClient
//启用 Feign 的远程服务调用
@EnableFeignClients
public class ConsumerApplication {
        public static void main(String[] args) {
                SpringApplication.run(ConsumerApplication.class, args);
        }
}
```

**2. 调用“服务提供者”接口**

Feign 是一个声明式 Web Service 客户端。使用 Feign 能让编写 Web Service 客户端更加简单。具体步骤为：

（1）添加 Feign 的依赖。

（2）在启动类中加入注解@EnableFeignClients。

（3）定义一个接口，在该接口中添加注解以使用它。接口的实现见以下代码：

```
//name：远程服务名，即“服务提供者”在“服务中心”中注册的名字
@FeignClient(name = "provider")
public interface MyFeignClient {
    @RequestMapping(value = "/hello")
    public String hello();
}
```

注解@FeignClient 中的 name 属性值是远程“服务提供者”的名字。

此接口中的方法需要和远程“服务提供者”中的方法名和参数保持一致。

Spring Cloud 对 Feign 进行了封装，使其支持 Spring MVC 标准注解。Feign 可以与 Eureka 和 Ribbon 组合使用以支持负载均衡。

**3. 实现客户端接口**

接下来将实现的MyFeignClient接口注入Controller层，以实现对该接口的调用，见以下代码。

```
@RestController
public class ConsumerController {
/*注入接口*/
```

```
    @Autowired
    MyFeignClient myFeignClient;
    @RequestMapping("/hello")
public String index() {
//返回内容
    return myFeignClient.hello();
    }
}
```

此接口将提供给最终的用户（设备）。

## 3.6 测试微服务系统

通过 3.3、3.4、3.5 节的代码实现了一个相对简单的微服务系统。下面来测试它的工作情况。

（1）启动“服务中心”集群、“服务提供者”集群、“服务消费者”。

（2）访问“服务消费者”接口 http://localhost:9000/hello，会返回如下信息：

```
provider:provider1 port:8001
```

因为 3.4 节实现了“服务提供者”负载均衡，所以，当再次访问 http://localhost:9000/hello 时，会返回如下信息：

```
provider:provider2 port:8002
```

如果多次访问 http://localhost:9000/hello，则会交替出现上面两个信息。这说明“服务提供者”集群已经正常工作。

（3）关闭其中任意一个“服务提供者”，再次访问 http://localhost:9000/hello，依然能正常返回信息。

（4）关闭其中任意两个“服务中心”，再次访问 http://localhost:9000/hello，依然能正常返回信息。

至此我们实现了一个相对简单的微服务系统，理清了“服务中心”“服务提供者”“服务消费者”三者的关系，以及集群的概念和用法。

但是，微服务系统的组件非常多，本实例只用到了一小部分。接下来会深入讲解微服务和 Spring Cloud 的基础知识。