06 Eureka 服务注册中心的搭建

更新时间：2019-06-19 17:54:34

> 辛苦是获得一切的定律。
>
> ——牛顿

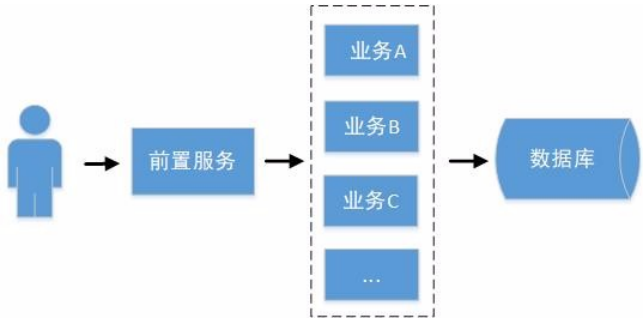
本节给大家介绍微服务架构中最为关键的一个组件：注册中心。注册中心可以说是微服务架构的交通枢纽中心，所有服务与服务之间的调用和被调用都需要注册中心。本节主要为大家介绍我们为什么要使用注册中心，以及如何搭建高可用的注册中心服务。

为什么要使用注册中心

我们可以从架构演进的角度来分析一下，为什么在微服务架构中需要注册中心。

在单体架构中，所有的服务都集中在一个项目中。当用户在前端发起请求的时候，直接由前端调用后端业务逻辑，再返回到界面完成系统响应。这个调用链是一条直线，由前端到后端，从后端响应到前端，因此不需要服务之间中转，也没有必要引入注册中心。

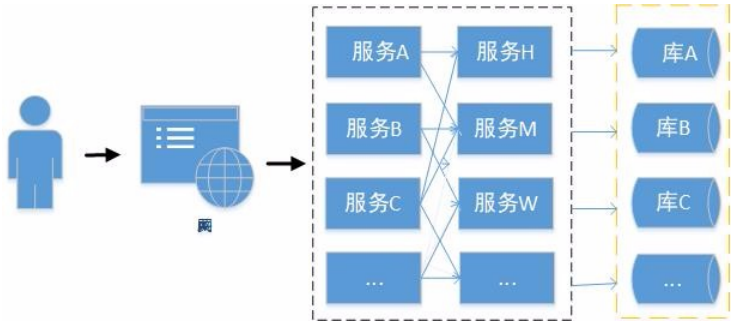
下面是单体架构的架构图：



随着公司产品规模越来越大，慢慢地会将服务进行拆分，比如在电商系统中，我们通常会拆分出：用户系统、订单系统、支付系统、库存系统等。

这样当用户在前端发起请求时，就需要后端各个项目相互配合，最后响应返回给用户。这个时候调用链就会相对复杂一些，前端用户发起请求，后端服务 A 接收到请求后调用服务 B，服务 B 可能需要调用服务 C、服务 D……经历过 N 个转发和响应之后，最后再由服务 A 将响应结果返回给用户。

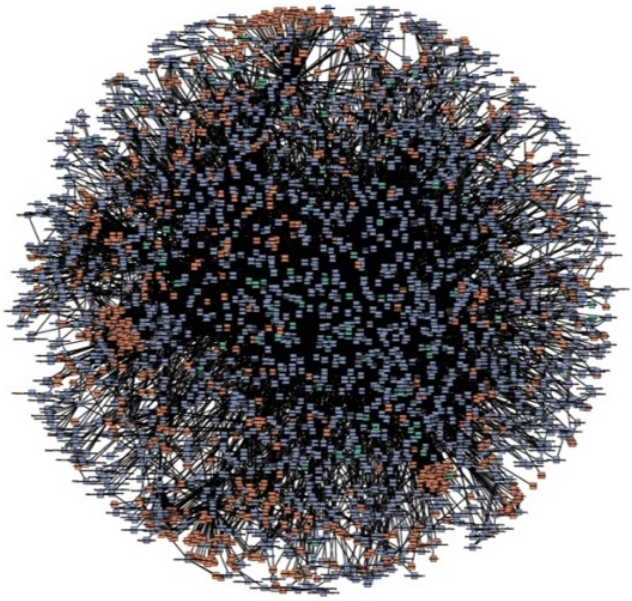
下面是分布式架构图：



当系统的规模继续扩大时，原有的拆分粒度可能已经不能满足业务的高速发展，于是决定将原来的服务进一步进行拆分，拆分出粒度更小的子服务，比如：短信服务、邮件服务、鉴权服务、积分服务、营销服务等等。服务拆分的粒度越小，服务之间的调用关系就越复杂，调用关系成指数级别的增长。

当到达一定规模的时候，就会出现一个问题，所有人都不知道项目与项目之间真正的调用关系是什么，当项目上线重启的时候很难去判断到底会影响到哪些相关的服务，这对后期项目的运维管理是一个巨大的灾难。

极端情况下就会出现下图所展示的这种情况：



所以在服务架构中，第一个需要解决的问题就是服务与服务之间的解耦，这时候 Eureka 就光荣地登上了历史舞台。

Eureka 介绍

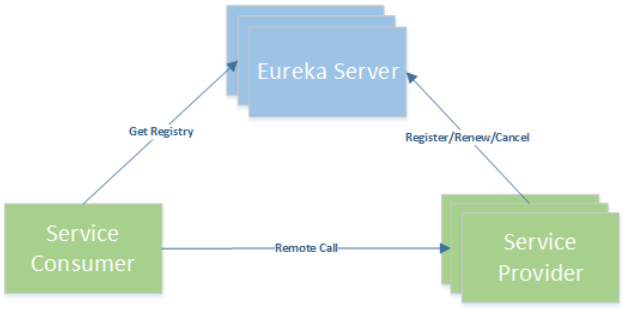
Eureka 是 Netflix 公司开源的产品，它是一种基于 REST（Representational State Transfer）的服务，主要用于 AWS 云。Eureka 提供了完整的 Service Registry 和 Service Discovery 实现，也是 Spring Cloud 体系中最重要最核心的组件之一。

简单来说，Eureka 就是 Netflix 开源的一款提供服务注册和发现的产品，并且提供了 Java 客户端。当然在 Spring Cloud 大力优化后的 Eureka，已经不仅仅只是用于 AWS 云，而是可以应用在任何需要使用注册中心的场景。

Spring Cloud 封装了 Netflix 公司开发的 Eureka 模块，在 Eureka 的基础上优化了一些配置，对一些不太合理的逻辑进行了优化，并提供了可视化界面，方便查看服务的运行状态。

Eureka 由两个组件组成：Eureka 服务端和 Eureka 客户端。Eureka 服务端就是注册中心。Eureka 客户端是一个 java 客户端，用来简化与服务端的交互、作为轮询负载均衡器，并提供服务的故障切换支持。

通过下图了解一下 Eureka 的使用场景：



从上图我们可以看出有三个角色：

- Eureka Server，担任注册中心的角色，提供了服务的注册和发现功能
- Service Provider，服务提供者，将自身服务注册到 Eureka Server，同时通过心跳来检查服务的运行状态
- Service Consumer，服务调用者，从 Eureka 获取注册服务列表，找到对应的服务地址再进行调用

搭建 Eureka Server

Eureka 是由 Java 语言开发而成，Spring Cloud 使用 Spring Boot 技术对 Eureka 进行了封装。因此它的部署方式非常简单，只需要项目引入 Eureka 对应的 Starter 包 `spring-cloud-starter-netflix-eureka-server`，然后像一个普通的 Spring Boot Web 项目一样，只需要一个启动命令即可。

添加依赖

将 pom 包中比较关键的几个点给大家截取出来解释一下。项目使用 Spring Boot 2.1.3 版本，以及 Spring Boot 体系内对应的依赖组件。

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.3.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

添加 Spring Cloud 的依赖版本信息。

```
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

properties 中定义了一些常用的变量，比如默认使用 Jdk 1.8 版本，Spring Cloud 的版本使用 Greenwich.SR1。dependencyManagement 设置项目使用 Spring Cloud SR1 版本。

Spring Cloud 体系内维护了各个组件之间的版本依赖关系，我们只需要在 pom 包中指明使用的具体 Spring Cloud 大版本即可。

接下来是比较重要的一块，添加 Eureka 依赖包，dependencies 模块中添加 `spring-cloud-starter-netflix-eureka-server`，项目即拥有了注册中心的功能。

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
```

在后面的项目中，其它内容不变化，只需要增删 dependencies 中的依赖组件即可。

报错

Jdk 版本 9 以上， 运行项目时会出现 `java.lang.TypeNotPresentException: Type javax.xml.bind.JAXBContext not present` 异常，这是因为 JAXB-API 是 Java EE 的一部分，在 Jdk 版本 9 以上里，没有在默认的类路径中，手动引入 JAXB 依赖包即可。

```
 <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>javax.activation</groupId>
    <artifactId>activation</artifactId>
    <version>1.1.1</version>
</dependency>
```

启动类

我们需要在启动类中添加注解，开启 Eureka Server 功能。

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class, args);
    }
}
```

添加`@EnableEurekaServer`注解，开启注册中心服务发现功能。

配置文件

我们需要添加一些 Eureka 的基础配置：

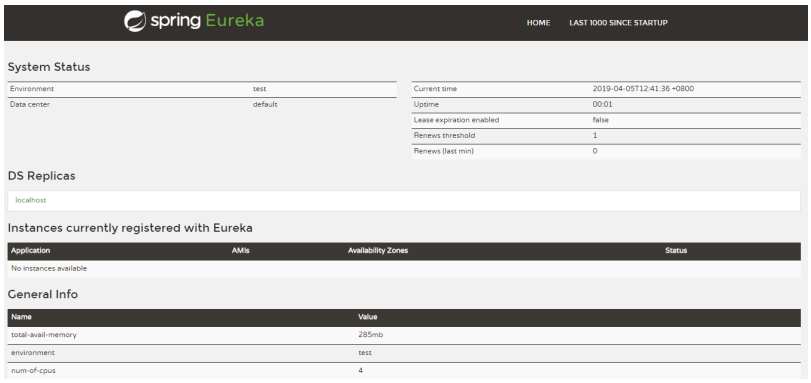
```
spring.application.name=eureka server
server.port=8000

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

- `spring.application.name` 项目名称
- `server.port` 项目启动的端口号
- `eureka.client.register-with-eureka` 表示是否将自己注册到 Eureka Server，默认为 true
- `eureka.client.fetch-registry` 表示是否从 Eureka Server 获取注册信息，默认为 true

测试

上述信息配置完成之后，Eureka 的服务端就搭建好了，在启动类上右键选择 run 启动项目。等待启动完成之后，在浏览器中访问地址：`http://localhost:8000/`，我们就可以看到 Eureka 的可视化界面，如下图：



可以发现后台页面被分为了五大块：

- System Status，主要展示系统状态，比如启动时间等
- DS Replicas，该服务从哪里同步数据
- Instances currently registered with Eureka，注册在 Eureka 的实例列表
- General Info，系统运行环境，比如内存、cpu 等
- Instance Info，本服务的基础信息，比如 ip 地址，状态等

注册中心高可用

理论上来讲，服务消费者本地缓存了服务提供者的地址。即使 Eureka Server 宕机，也不会影响服务之间的调用，但是一旦涉及到服务的上下线，本地的缓存信息将会出现偏差，从而影响到了整个微服务架构的稳定性，因此搭建 Eureka Server 集群来提高整个架构的高可用性，是非常有必要的。

下面我们进行高可用搭建。

搭建 Eureka 集群

开启 Eureka 集群配置后，服务启动时 Eureka Server 会将注册信息向其它 Eureka Server 进行同步，因此搭建高可用架构只需要将 Eureke Server 配置指向其它可用的 serviceUrl 即可。

我们在上面 Eureka 单个示例的基础上，复制出三份来分别命名为：eureka-a、eureka-b、eureka-c 三个示例项目，使用这三个示例项目搭建 Eureka Server 的集群。

接下来需要分别修改 eureka-a、eureka-b、eureka-c 的配置信息。

eureka-a 的配置信息如下：

```
spring.application.name=eureka server
server.port=8001

eureka.instance.hostname=eurekaA
eureka.client.serviceUrl.defaultZone=http://eurekaB:8002/eureka/,http://eurekaC:8003/eureka/

eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
```

- 项目端口改为 8001
- 增加`eureka.instance.hostname`配置，相当于给服务起个别名注册到注册中心
- `eureka.client.register-with-eureka`和`eureka.client.fetch-registry`配置为 true ，表示将自己注册到注册中心，并且从注册中心获取注册信息
- `eureka.client.serviceUrl.defaultZone`，指向其它两个注册中心，重点配置内容

eureka-b 和 eureka-c 项目的配置信息和 eureka-a 的配置内容大同小异，这里只贴出差异的部分，完整配置信息请查看课程示例代码：

```
eureka-b
--------------
spring.application.name=eureka server
server.port=8002

eureka.instance.hostname=eurekaB
eureka.client.serviceUrl.defaultZone=http://eurekaA:8001/eureka/,http://eurekaC:8003/eureka/

eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true


eureka-c
--------------
spring.application.name=eureka server
server.port=8003

eureka.instance.hostname=eurekaC
eureka.client.serviceUrl.defaultZone=http://eurekaA:8001/eureka/,http://eurekaB:8002/eureka/

eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
```

以上项目配置完成。

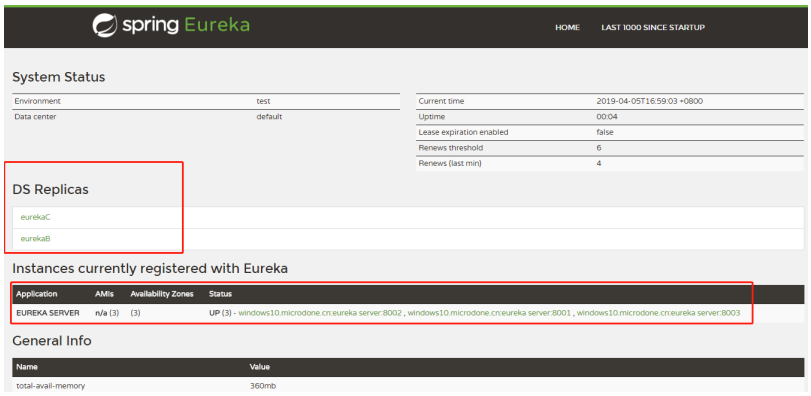
测试

本机直接访问 eurekaA/B/C 网络是不通的，需要本地配置 host 信息。在 Windows 的`C:\Windows\System32\drivers\etc\hosts`或者 Linux 的`/etc/hosts`文件末添加以下信息：

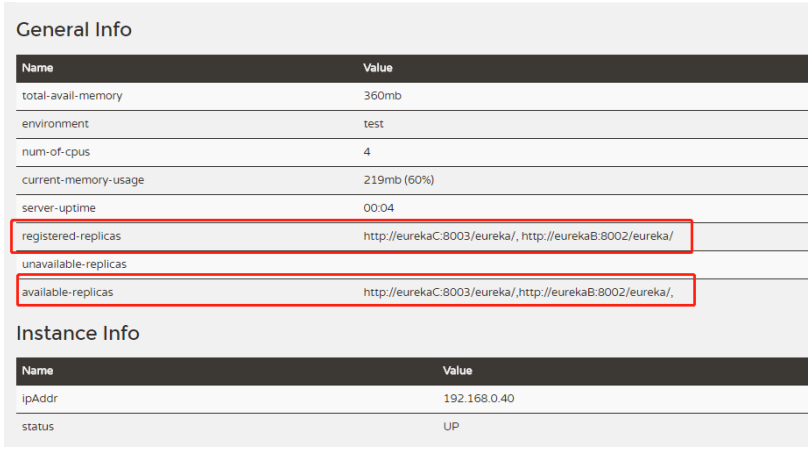
```
127.0.0.1 eurekaA eurekaB eurekaC
```

添加完成之后，依次启动 eureka-a、eureka-b、eureka-c 项目，等待全部启动完成之后，浏览器访问地址：`http://localhost:8001/`，即可看到页面的集群相关信息。

在 DS Replicas 模块中可以看到另外两台注册中心的别名；在 Instances currently registered with Eureka 模块中可以看到三个注册中心都以实例的形式注册到了注册中心。



在 General Info 模块中，可以看到 registered-replicas（已经注册到的副本）和 available-replicas（有效副本）都可以看到其它两个注册中心的信息。



证明 Eureka Server 高可用集群搭建成功。

常见问题

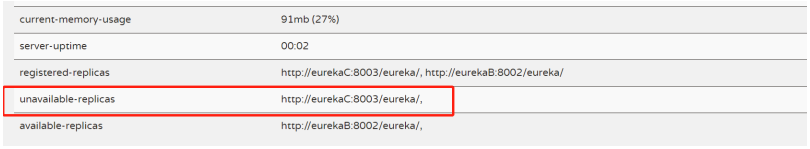
Eureka 的 UNKNOWN 问题

注册中心(Eureka Server)中的服务状态，常见的有 UP、DOWN，但有时会出现另外一种 UNKNOWN 状态，或者 UNKNOWN 服务名。

- UNKNOWN 服务名，是因为在项目中没有配置应用实例的名称导致，配置参数为`spring.application.name`或者`eureka.instance.appname`，如果这两个参数都不配置，则将会出现 UNKNOWN 服务名
- UNKNOWN 状态，出现的机率比较少，这是 Eureka Server 没有获取到客户端的心跳数据导致，如果将`eureka.client.healthcheck.enabled=true`配置到 bootstrap.yml 也会导致此问题，应该配置到 application.yml 中

注册中心副本不可用（unavailable-replicas）

有些同学在搭建 Eureka 集群时会出现，注册中心服务不可用的情况，其它 Eureka Server 的地址进入了 unavailable-replicas 中，此情况可能时由多种原因导致。



current-memory-usage	91mb (27%)
server-uptime	00:02
registered-replicas	http://eurekaC:8003/eureka/, http://eurekaB:8002/eureka/
unavailable-replicas	http://eurekaC:8003/eureka/,
available-replicas	http://eurekaB:8002/eureka/,

1、`eureka.client.serviceUrl.defaultZone`配置地址不以 localhost 地址来配置，应该按照文中示例配置。

2、没有开启相互注册

```
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
```

3、没有配置 host 信息。

出现类似问题一般都是配置信息错误或者少配置导致，根据课程内容多检查即可避免。

小结

本节我们学习了为什么需要使用注册中心，以及 Eureka 作为注册中心最关键的组件，都有哪些特点和优势，最后我们学习搭建了 Eureka Server 单台和集群的案例。

通过本节内容的学习，相信大家会对 Eureka 的使用有了更进一步地了解。

本文作者：纯洁的微笑、江南一点雨

05 Spring Cloud 和 Spring Boot 之间的密切关系

07 Eureka 工作原理