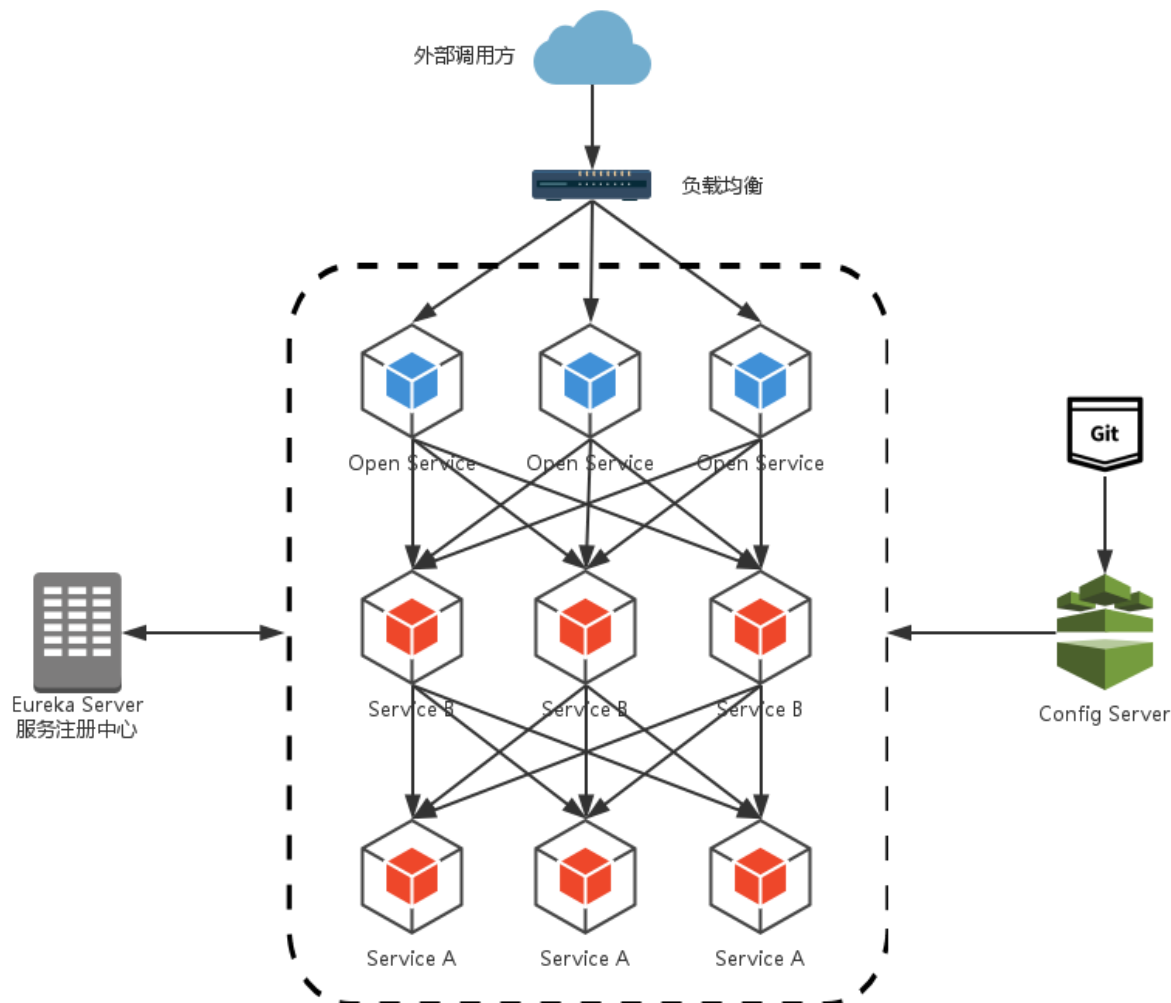


通过之前几篇Spring Cloud中几个核心组件的介绍，我们已经可以构建一个简略的（不够完善）微服务架构了，如下图所示：



alt

我们使用Spring Cloud Netflix中的Eureka实现了服务注册中心以及服务注册与发现；而服务间通过Ribbon或Feign实现服务的消费以及均衡负载；通过Spring Cloud Config实现了应用多环境的外部化配置以及版本管理。为了使得服务集群更为健壮，使用Hystrix的熔断机制来避免在微服务架构中个别服务出现异常时引起的故障蔓延。

在该架构中，我们的服务集群包含：内部服务Service A和Service B，他们都会注册与订阅服务至Eureka Server，而Open Service是一个对外的服务，通过均衡负载公开至服务调用方。本文我们把焦点聚集在对外服务这块，这样的实现是否合理，或者是否有更好的实现方式呢？

先来说说这样架构需要做的一些事儿以及存在的不足：

- 首先，破坏了服务无状态特点。为了保证对外服务的安全性，我们需要实现对服务访问的权限控制，而开放服务的权限控制机制将会贯穿并污染整个开放服务的业务逻辑，这会带来的最直接问题是，破坏了服务集群中REST API无状态的特点。从具

体开发和测试的角度来说，在工作中除了要考虑实际的业务逻辑之外，还需要额外持续对接口访问的控制处理。

- 其次，无法直接复用既有接口。当我们需要对一个即有的集群内访问接口，实现外部服务访问时，我们不得不通过在原有接口上增加校验逻辑，或增加一个代理调用来实现权限控制，无法直接复用原有的接口。

为了解决上面这些问题，我们需要将权限控制这样的东西从我们的服务单元中抽离出去，而最适合这些逻辑的地方就是处于对外访问最前端的地方，我们需要一个更强大一些的均衡负载器，它就是本文将来介绍的：服务网关。

服务网关是微服务架构中一个不可或缺的部分。通过服务网关统一向外系统提供REST API的过程中，除了具备服务路由、均衡负载功能之外，它还具备了权限控制等功能。Spring Cloud Netflix中的Zuul就担任了这样的角色，为微服务架构提供了前门保护的作用，同时将权限控制这些较重的非业务逻辑内容迁移到服务路由层面，使得服务集群主体能够具备更高的可复用性和可测试性。

下面我们通过实例例子来使用一下Zuul来作为服务的路有功能。

准备工作

在构建服务网关之前，我们先准备一下网关内部的微服务，可以直接使用之前的工程：

- eureka-server
- eureka-client
- eureka-consumer

在启动了eureka-client和eureka-consumer的实例之后，所有的准备工作就以就绪，下面我们来试试使用Spring Cloud Zuul来实现服务网关的功能。

构建服务网关

使用Spring Cloud Zuul来构建服务网关的基础步骤非常简单，只需要下面几步：

- 创建一个基础的Spring Boot项目，命名为：api-gateway。并在pom.xml中引入依赖：

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

- 创建应用主类，并使用@EnableZuulProxy注解开启Zuul的功能。

@EnableZuulProxy

@SpringCloudApplication

```
public class ApiGatewayApplication {
    public static void main(String[] args) {
        new
        SpringApplication.Builder(ApiGatewayApplication.class).web(true).run(args);
    }
}
```

- 创建配置文件`application.yaml`，并加入服务名、端口号、eureka注册中心的地址：

```
spring.application.name=api-gateway
```

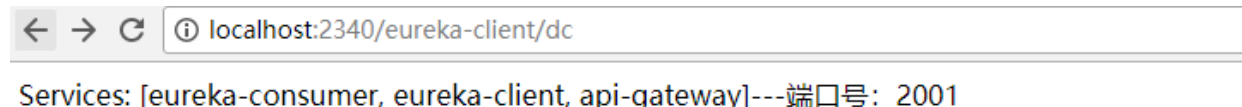
```
server.port=2340
```

```
eureka.client.serviceUrl.defaultZone=http://localhost:1001/eureka/
```

到这里，一个基于Spring Cloud Zuul服务网关就已经构建完毕。启动该应用，一个默认的服务网关就构建完毕了。由于Spring Cloud Zuul在整合了Eureka之后，具备默认的服务路由功能，即：当我们这里构建的`api-gateway`应用启动并注册到eureka之后，服务网关会发现上面我们启动的两个服务`eureka-client`和`eureka-consumer`，这时候Zuul就会创建两个路由规则。每个路由规则都包含两部分，一部分是外部请求的匹配规则，另一部分是路由的服务ID。针对当前示例的情况，Zuul会创建下面的两个路由规则：

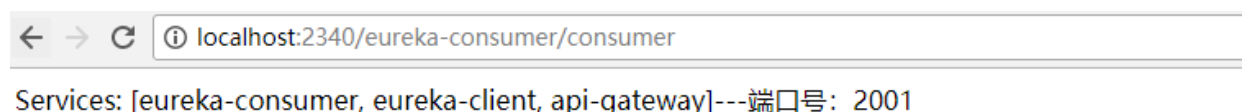
- 转发到`eureka-client`服务的请求规则为：`/eureka-client/**`
- 转发到`eureka-consumer`服务的请求规则为：`/eureka-consumer/**`

最后，我们可以通过访问2340端口的服务网关来验证上述路由的正确性：



← → ↻ ⓘ localhost:2340/eureka-client/dc

Services: [eureka-consumer, eureka-client, api-gateway]---端口号: 2001



← → ↻ ⓘ localhost:2340/eureka-consumer/consumer

Services: [eureka-consumer, eureka-client, api-gateway]---端口号: 2001

- 比如访问：<http://localhost:1101/eureka-client/dc>，该请求将最终被路由到`eureka-client`的`/dc`接口上。

本节介绍了构建服务网关的基础。通过上面的构建内容，我们已经为所有内部服务提供了一个统一的对外入口，同时对于服务的路由都是自动创建了，减少了传统方式大量的运维配置工作。