spring cloud Zuul

1. 网关概念提出的背景

首先，对于路由规则与服务实例的维护问题。Spring Cloud Zuul 通过与 Spring Cloud Eureka 进行整合，将自身注册为 Eureka 服务治理下的应用，同时从 Eureka 中获得了所有其他微服务的实例信息。这样的设计非常巧妙地将服务治理体系中维护的实例信息利用起来，使得将维护服务实例的工作交给了服务治理框架自动完成，不再需要人工介入。而对于路由规则的维护，Zuul 默认会将通过以服务名作为 ContextPath 的方式来创建路由映射，大部分情况下，这样的默认设置已经可以实现我们大部分的路由需求，除了一些特殊情况（比如兼容一些老的 URL）还需要做一些特别的配置。但是相比于之前架构下的运维工作量，通过引入 Spring Cloud Zuul 实现 API 网关后，已经能够大大减少了。

其次，对于类似签名校验、登录校验在微服务架构中的冗余问题。理论上来说，这些校验逻辑在本质上与微服务应用自身的业务并没有多大的关系，所以它们完全可以独立成一个单独的服务存在，只是它们被剥离和独立出来之后，并不是给各个微服务调用，而是在 API 网关服务上进行统一调用来对微服务接口做前置过滤，以实现对微服务接口的拦截和校验。Spring Cloud Zuul 提供了一套过滤器机制，它可以很好地支持这样的任务。开发者可以通过使用 Zuul 来创建各种校验过滤器，然后指定哪些规则的请求需要执行校验逻辑，只有通过校验的才会被路由到具体的微服务接口，不然就返回错误提示。通过这样的改造，各个业务层的微服务应用就不再需要非业务性质的校验逻辑了，这使得我们的微服务应用可以更专注于业务逻辑的开发，同时微服务的自动化测试也变得更容易实现。

微服务架构虽然可以将我们的开发单元拆分得更为细致，有效降低了开发难度，但是它所引出的各种问题如果处理不当会成为实施过程中的不稳定因素，甚至掩盖掉原本实施微服务带来的优势。所以，在微服务架构的实施方案中，API 网关服务的使用几乎成为了必然的选择。

2. 步骤

- 构建项目，加入依赖

网关会通过注册中心维护服务清单，通过服务中心来映射服务名与Localhost：port的关系从而实现面向服务的编程。

很显然，传统路由的配置方式对于我们来说并不友好，它同样需要运维人员花费大量的时间来维护各个路由 path 与 url 的关系。为了解决这个问题，Spring Cloud Zuul 实现了与 Spring Cloud Eureka 的无缝整合，我们可以让路由的 path 不是映射具体的 url，而是让它映射到某个具体的服务，而具体的 url 则交给 Eureka 的服务发现机制去自动维护，我们称这类路由为面向服务的路由。在 Zuul 中使用服务路由也同样简单，只需做下面这些配置。

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

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

- 配置文件

```
spring.application.name=api-gateway
server.port=5555
```

```
zuul.routes.api-a.path=/api-a/**
zuul.routes.api-a.serviceId=hello-service

zuul.routes.api-b.path=/api-b/**
zuul.routes.api-b.serviceId=feign-consumer

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```

可以看出ServiceId就是服务名，而其实Zuul默认的前缀就是服务名也就是说，如果不配置上述的红框里面的信息，则默认/hello-service/** 的前缀路由，实际上一般我们也是这样用的，配置特殊的路由时才会专门配置ServiceId 比如老的项目的路由。

- 问题汇总

实际上，上面的步骤已经实现了网关的作用，但是有几个问题需要明确：

1. 路由前缀

```
zuul.routes.api-a.path=/api/a/**
zuul.routes.api-a.serviceId=hello-service

zuul.routes.api-b.path=/api-b/**
zuul.routes.api-b.serviceId=hello-service
```

```
zuul.routes.api-c.path=/ccc/**
zuul.routes.api-c.serviceId=hello-service
```

以上三种路由都可以正常运作，如果这时候加上配置信息

```
zuul.prefix=/api
```

此时只有第三种能正确路由到服务，所以

上述实验基于 Brixton.SR7 和 Camden.SR3 测试均存在问题，所以在使用该版本或以下版本时候，务必避免让路由表达式的起始字符串与 zuul.prefix 参数相同。

2. 本地跳转

```
zuul.routes.api-a.path=/api-a/**
zuul.routes.api-a.url=http://localhost:8001/

zuul.routes.api-b.path=/api-b/**
zuul.routes.api-b.url=forward:/local
```

这里要注意，由于需要在 API 网关上实现本地跳转，所以相应的我们也需要为本地跳转实现对应的请求接口。按照上面的例子，在 API 网关上还需要增加一个/local/hello 的接口实现才能让 api-b 路由规则生效，比如下面的实现。否则 Zuul 在进行 forward 转发的时候会因为找不到该请求而返回 404 错误。

3. 过滤器

依赖以及主程序入口注解不再赘述，需要提一下过滤器的创建

过滤器的run（）

```
    public Object run() {
        RequestContext ctx = RequestContext.getCurrentContext();
        HttpServletRequest request = ctx.getRequest();

        log.info("send {} request to {}", request.getMethod(),
request.getRequestURL().toString());

        Object accessToken = request.getParameter("accessToken");
        if(accessToken == null) {
            log.warn("access token is empty");
            ctx.setSendZuulResponse(false);
            ctx.setResponseStatusCode(401);
            return null;
        }
        log.info("access token ok");
        return null;
    }
}
```

另外几个方法

- filterType：过滤器的类型，它决定过滤器在请求的哪个生命周期中执行。这里定义为pre，代表会在请求被路由之前执行。
- filterOrder：过滤器的执行顺序。当请求在一个阶段中存在多个过滤器时，需要根据该方法返回的值来依次执行。
- shouldFilter：判断该过滤器是否需要被执行。这里我们直接返回了true，因此该过滤器对所有请求都会生效。实际运用中我们可以利用该函数来指定过滤器的有效范围。
- run：过滤器的具体逻辑。这里我们通过ctx.setSendZuulResponse(false)令zuul过滤该请求，不对其进行路由，然后通过ctx.setResponseStatus-Code(401)设置了其返回的错误码，当然也可以进一步优化我们的返回，比如，通过ctx.setResponseBody(body)对返回的body内容进行编辑等。

之后需在主程序中加载bean

```
@EnableZuulProxy
@SpringCloudApplication
public class Application {

    public static void main(String[] args) {
        new SpringApplicationBuilder(Application.class).web(true).run(args);
    }

    @Bean
    public AccessFilter accessFilter() {
        return new AccessFilter();
    }
}
```