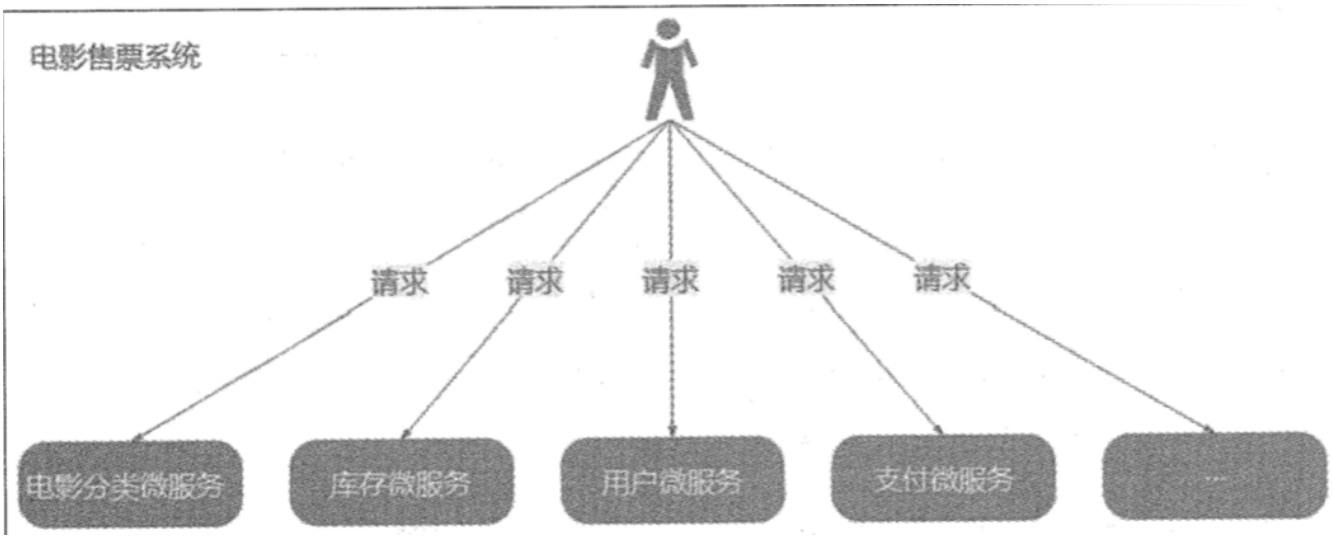
微服务之SpringCloud之Zuul

一、简介

1.1 微服务的问题

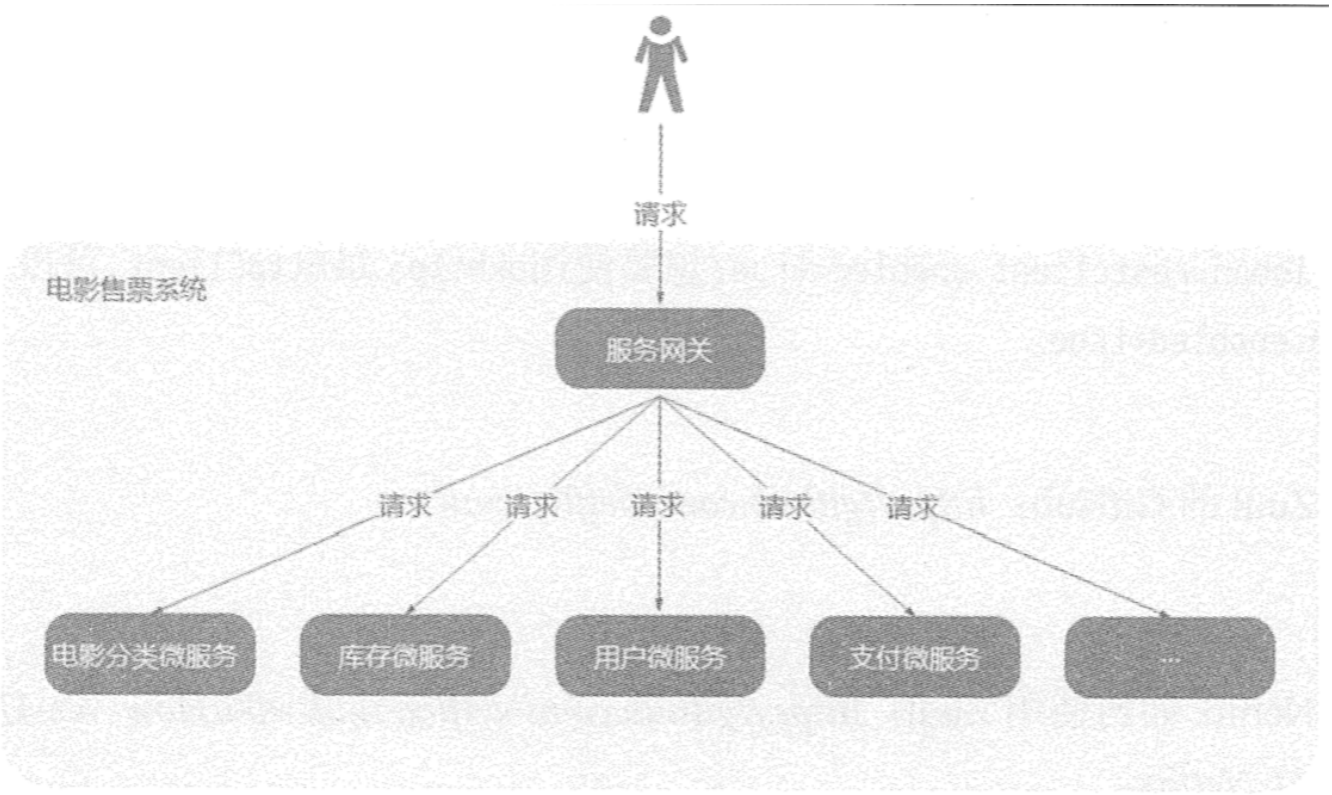
若让客户端直接与各个微服务通信，会有以下问题： 客户端会多次请求不同微服务，增加了客户端复杂性 存在跨域请求，处理相对复杂 认证复杂，每个服务都需要独立认证 难以重构，多个服务可能将会合并成一个或拆分成多个



微服务网关介于服务端与客户端的中间层，所有外部服务请求都会先经过微服务网关 客户只能跟微服务网关进行交互，无需调用特定微服务接口，使得开发得到简化 微服务网关还具备以下优点：

易于监控，微服务网关可收集监控数据并进行分析 易于认证，可在微服务网关上进行认证，然后在将请求转发给微服务，无须每个微服务都进行认证

减少客户端与微服务之间的交互次数



1.2 Zuul是什么

Zuul 是开源的微服务网关，可与 Eureka、Ribbon、Hystrix 等组件配合使用，

Zuul 它的核心是一系列过滤器

，这些过滤器可完成下面功能：

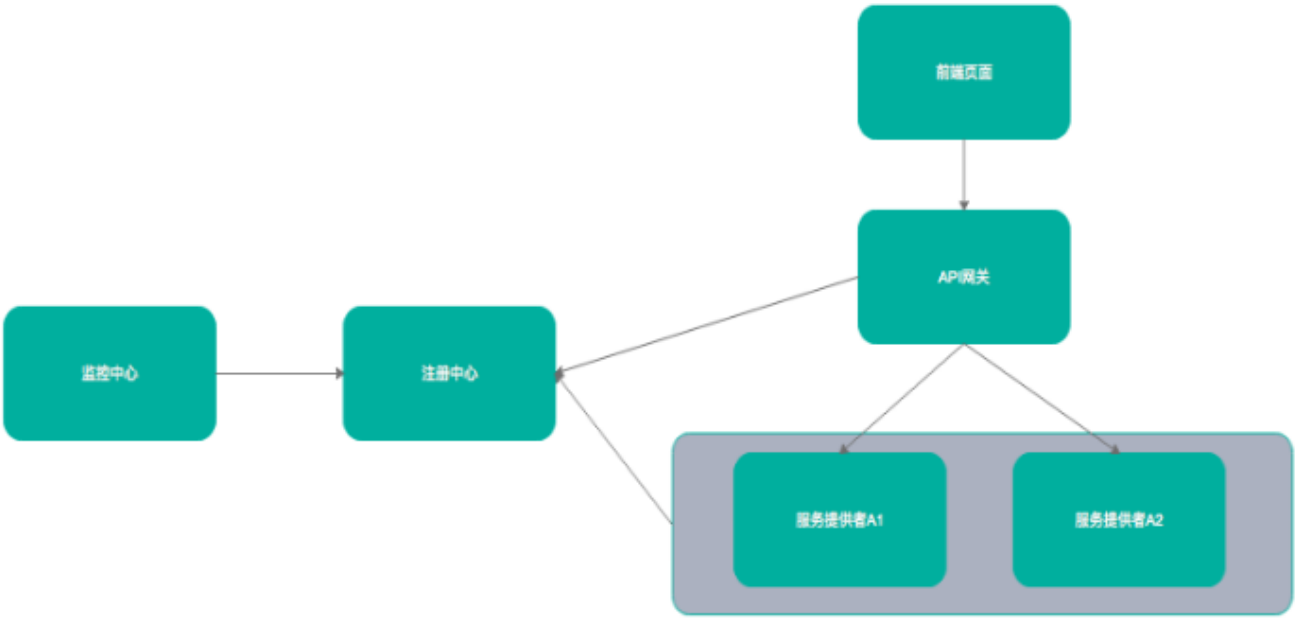
- 身份认证与安全：识别每个资源的验证要求，并拒绝那些要求不符合的请求
- 审核与监控：在边缘位置追踪有意义的数据和统计结果，从而带来精确的生产视图
- 动态路由：动态的将请求路由到不同的后端集群
- 压力测试：逐渐增加指向集群的流量，以了解性能
- 负载分配：为每一种负载类型分配对应容量，并弃用超出限定值的请求
- 静态响应处理：在边缘位置直接建立部分响应，从而避免转发到内部集群
- 多区域弹性：跨越 AWS Region 进行请求路由，实现 ELB 使用多样化，让系统边缘更贴近使用者
- Spring Cloud 对 Zuul 进行了整合和增强，Zuul 默认使用的 HTTP 客户端是 Apache Http Client，也可使用 RestClient 或 okHttpClient

若要使用 RestClient 可设置

```
ribbon.restclient.enabled = true
```

> 若要使用 okhttp3.OKHttpClient 可设置

```
ribbon.okhttp.enabled = true
```



二、初体验

实现步骤：

1、依赖jar

``

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

2、实现配置

``

```
eureka.client.serviceUrl.defaultZone= http://localhost:8761/eureka/
spring.application.name= Zuul
server.port=9903
zuul.routes.service1.path=/hello/**
zuul.routes.service1.service-id=HelloConsumer
```

zuul.routes.service1.path 外界请求路径名称

zuul.routes.service1.service-id 服务名称 一般指的是消费者的名称 也可以是提供者的名称 这个名称需要和注册中心一致

3、设置开关类

使用注解实现zuul代理

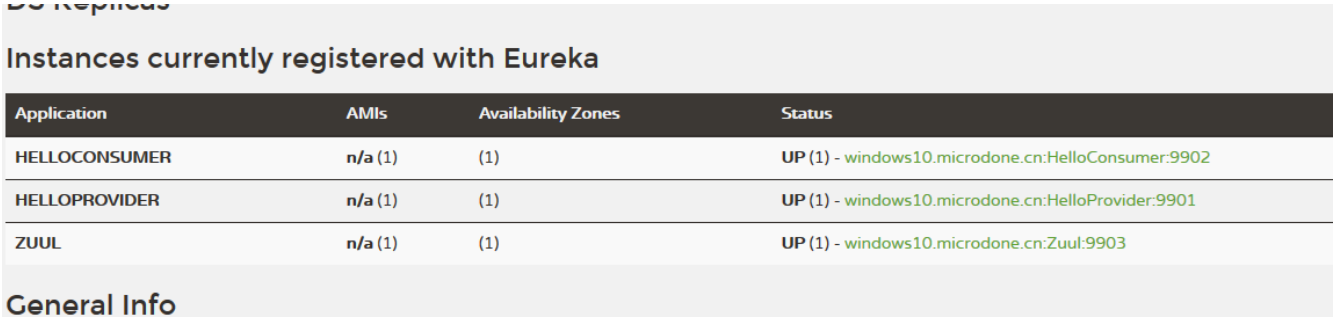
``

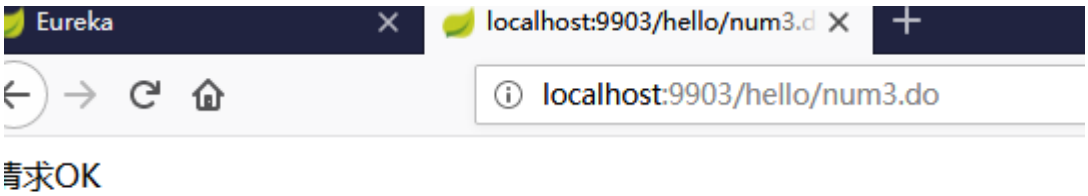
```
@SpringBootApplication
@EnableZuulProxy //启用Zuul代理
@EnableEurekaClient  //注册到注册中心
public class CloudZuulApplication {
    public static void main(String[] args) {
        SpringApplication.run(CloudZuulApplication.class, args);
    }
}
```

4、运行并测试

依次启动：注册中心、提供者、消费者、网关项目

Zuul可以指向提供者、也可以指向消费者





请求Zuul项目的path

示例：

http://localhost:9903/hello/num3.do

hello 对应的就是配置文件中的

zuul.routes.service1.path=/hello/**

num3.do 对应的其实就是消费者项目中接口名称

``

```
@GetMapping("num3.do")
public String num3(){
    return "请求OK";
}
```

三、核心技术

3.1 Zuul路由匹配规则

1.基本实现：

zuul.rotes.自定义名称.path 请求的匹配名称

zuul.rotes.自定义名称.service-id 对应的服务名称 在注册中心存在

zuul.rotes.自定义名称.url 对应的消费者项目的名称

注意：service-id 服务名称 url 服务路径

2.正则匹配：

PatternServiceRouteMapper

new PatternServiceRouteMapper("(?^.+)-(?v.+$)", "${version}/${name}")

根据请求的url进行正则匹配，参数说明：

1、匹配服务名称 检查是否服务注册的Service-Id 真正的服务名称

2、转换之后的路径名称 请求名称

PatternServiceRouteMapper ： 将请求的路径名称，按照正则匹配的结果生成对应的 服务名称

格式：http:主机ip:zuul所在项目的端口号/v1/服务名称

转换为：

服务名称-v1

服务名称：必须使用小写

http://10.8.165.98:9903/v1/helloconsumer/num3.do

v1/helloconsumer/num3.do 正则转换 helloconsumer-v1/num3.do

3.2 url过滤

请求链过滤

实现步骤：

1、定义类 继承ZuulFilter类

2、重写方法

1、filterType 类型

取值:

```
/**
 * 过滤器的类型
 * 取值:
 * 1、pre
 * 2、route
 * 3、post
 * 4、error
 * */
  @Override
    public String filterType() {
        return "pre";
    }
```

2、filterOrder 过滤器排序 值越小，优先级越高

3、shouldFilter 是否执行当前的过滤器 true执行 false不执行

4、run 执行过滤器 业务逻辑处理 对请求进行处理

``

```
/**
 * 你要做的事情
 * 过滤器的最终目的
 * */
@Override
public Object run() throws ZuulException {
    System.out.println("Url过滤器链 预处理");
    //获取请求对象
    RequestContext requestContext=RequestContext.getCurrentContext();
    HttpServletRequest request=requestContext.getRequest();
    String token=request.getHeader("usertoken");
    if(token==null){
        //如果没有Token,就跳转到登录
        requestContext.setResponseStatusCode(404);
        //拦截请求  true：放行   false：拦截请求
        requestContext.setSendZuulResponse(false);
        try {
            requestContext.getResponse().getWriter().print("No Login");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return null;
}
```

setSendZuulResponse 标记是否放行 true 放行 请求合法 false:拦截 请求非法

3.3 Zuul的过滤器类型

pre：预处理

route：过滤中

post：过滤之后

error：异常