白话SpringCloud | 第九章：路由网关(Zuul)的使用

前言

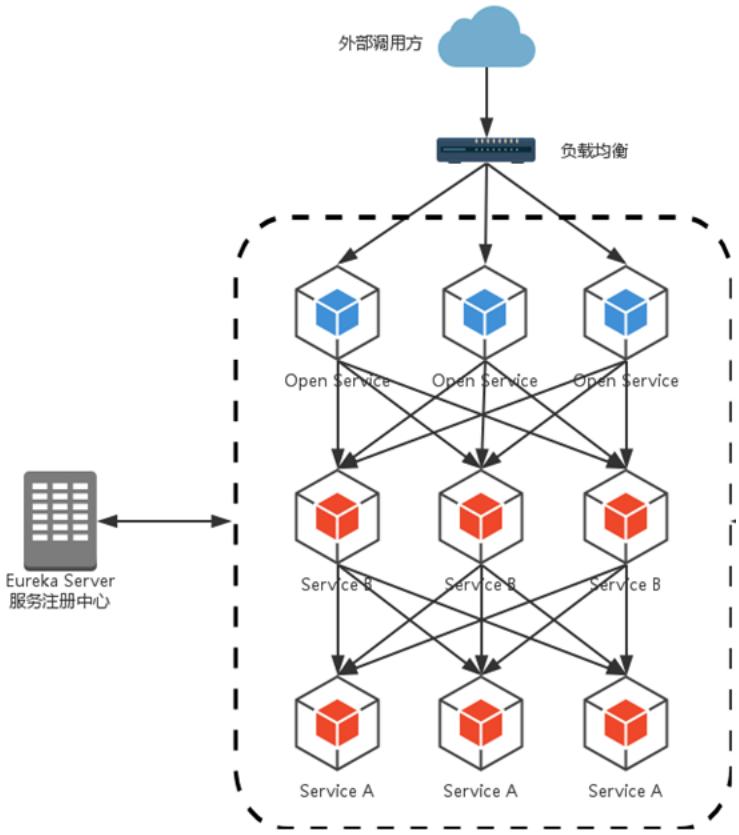
> 介绍完分布式配置中心，结合前面的文章。我们已经有了一个微服务的框架了，可以对外提供api接口服务了。但现在试想一下，在微服务框架中，每个对外服务都是独立部署的，对外的api或者服务地址都不是不尽相同的。对于内部而言，很简单，通过注册中心自动感知即可。但我们大部分情况下，服务都是提供给外部系统进行调用的，不可能同享一个注册中心。同时一般上内部的微服务都是在内网的，和外界是不连通的。而且，就算我们每个微服务对外开放，对于调用者而言，调用不同的服务的地址或者参数也是不尽相同的，这样就会造成消费者客户端的复杂性，同时想想，可能微服务可能是不同的技术栈实现的，有的是 `http` 、 `rpc` 或者 `websocket` 等等，也会进一步加大客户端的调用难度。所以，**一般上都有会有个api网关，根据请求的url不同，路由到不同的服务上去，同时入口统一了，还能进行统一的身份鉴权、日志记录、分流等操作。**接下来，我们就来了解今天要讲解的 `路由服务：zuul` 。

一点知识

为什么要使用微服务网关

简单来说，微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。

在未加入网关时，一般上会在服务外网架设一个负载均衡，如nginx等。此时，微服务的组成为：



此时，对于 `Open Service` 而言可能需要提供权限控制等和业务无关的能力，这样本身就破坏了微服务服务单一的原则。所以，一般上在 `Open Service` 之上，还有一层服务提供诸如通用的权限校验、参数校验等功能，此服务就是**网关**了。之后，对于内部微服务而言，只需要关系各自微服务提供的业务功能即可，无需去关心其他非业务相关的功能。

API网关是什么

> API网关可以提供一个单独且统一的API入口用于访问内部一个或多个API。简单来说嘛就是一个统一入口，比如现在的支付宝或者微信的相关api服务一样，都有一个统一的api地址，统一的请求参数，统一的鉴权。

客户端和服务端直连的弊端

- 客户端会对此请求不同的微服务，增加客户端复杂性
- 存在跨域请求时，需要进行额外处理
- 认证服务，每个服务需要独立认证
- UI端和微服务耦合

网关的优缺点

优点:

- 减少api请求次数
- 限流
- 缓存
- 统一认证
- 降低微服务的复杂度
- 支持混合通信协议(前端只和api通信，其他的由网关调用)
- ……

缺点:

- 网关需高可用，可能产生单点故障
- 管理复杂

网关的选择

现在市场上有很多的网关可供选择:

- Spring Cloud Zuul：本身基于 `Netflix` 开源的微服务网关,可以和 `Eureka` , `Ribbon` , `Hystrix` 等组件配合使用。

webservice(2)
hystrix(2)
eureka(2)
更多

随笔分类
Docker(8)
SpringBoot(39)
SpringCloud(12)
日常积累(5)

随笔档案
2019年5月 (1)
2019年3月 (2)
2018年12月 (1)
2018年11月 (4)
2018年10月 (6)
2018年9月 (10)
2018年8月 (19)
2018年7月 (21)

文章分类
SpringBoot
日常积累

最新评论
1. Re:SpringBoot | 第二十
bo的集成和使用

- Kong：基于OpenResty的 API 网关服务和网关服务管理层。
- Spring Cloud Gateway：是由 `spring` 官方基于 `Spring5.0` ，`Spring Boot2.0` ，`Project Reactor` 等技术开发的网关，提供了一个构建在 `Spring Ecosystem` 之上的API网关，旨在提供一种简单而有效的途径来发送API，并向他们提供交叉关注点，例如：安全性，监控/指标和弹性。目的是为了替换 `Spring Cloud Netfilx Zuul` 的。

在 `Spring cloud` 体系中，一般上选择 `zuul` 或者 `Gateway` 。当然，也可以综合自己的业务复杂性，自研一套或者改在一套符合自身业务发展的api网关的，最简单做法是做个 `聚合api服务` ，通过 `SpringBoot` 构建对外的api接口，实现统一鉴权、参数校验、权限控制等功能，说白了就是一个 `rest` 服务。

Zuul介绍和使用

何为Zuul

> `Zuul` 是 `Netflix` 开源的微服务网关，可以和 `Eureka` 、 `Ribbon` 、 `Hystrix` 等组件配合使用， `Spring Cloud` 对 `Zuul` 进行了整合与增强， `Zuul` 的主要功能是**路由转发**和**过滤器**。

18. Router and Filter: Zuul

Routing is an integral part of a microservice architecture. For example, `/` may be mapped to your web application, `/api/users` is mapped to the user service and `/api/shop` is mapped to the shop service. Zuul is a JVM-based router and server-side load balancer from Netflix.

Netflix uses Zuul for the following:

- Authentication
- Insights
- Stress Testing
- Canary Testing
- Dynamic Routing
- Service Migration
- Load Shedding
- Security
- Static Response handling
- Active/Active traffic management

Zuul's rule engine lets rules and filters be written in essentially any JVM language, with built-in support for Java and Groovy.

`Zuul` 基于JVM的路由器和服务器端负载均衡器。同时， `Zuul` 的规则引擎允许规则和过滤器基本上用任何JVM语言编写，内置支持 `Java` 和 `Groovy` 。这个功能，就可以实现动态路由的功能了。当需要添加某个新的对外服务时，一般上不停机更新是通过数据缓存配置或者使用 `Groovy` 进行动态路由的添加的。

`Zuul` 的核心一系列的过滤器：

- 身份认证与安全：识别每个资源的验证要求，并拒绝那些与要求不符的请求。
- 审查与监控：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：动态地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，并启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相应，从而避免其转发到内部集群。

加入 `Zuul` 网关后：

问下大佬， 如果springboot建了 那么dubbo admin是不啦？ 如果两个选其一应该是谢谢

--等待是一生

2. Re:SpringBoot | 第二十开发之异步调用

想问一下，异步请求和异步调个本质区别是什么？如果说异了快速释放服务端的请求压力完全也可以起一个线程去执行快速响应客户端，唯独感觉就是不会提前关闭res.....

--会烧

3. Re:白话SpringCloud | 第由网关(Zuul)：利用swagge文档

楼主,我下载了您的代码,运行制台一直无限刷错误,"com.nvery.shared.transport.Traption: Cannot exec.....

4. Re:白话SpringCloud | 第由网关(Zuul)：利用swagge文档

楼主,我按照您的方式试了一下没有成功,请问 cn.lqdev.leargcloud.zuul.swagger2 这个那个项目的控制层包路径吗?v.....

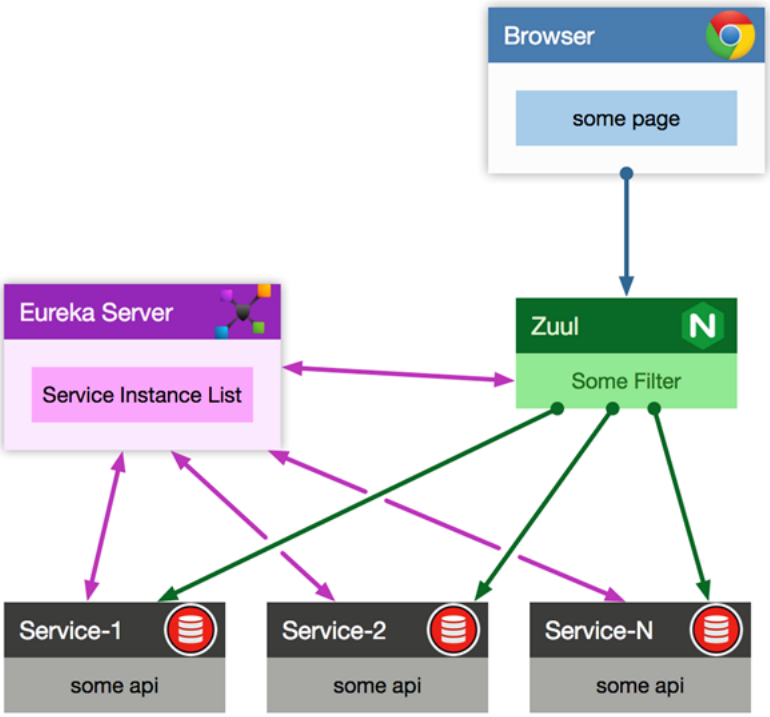
5. Re:SpringBoot | 第八章常、数据校验处理

666

阅读排行榜

1. Mybatis-Plus使用全解(1

2. SpringBoot | 第二十八章之Spring Boot Admin使用(



Zuul实践

创建工程： `spring-cloud-zuul` 。

这里直接加入了注册中心进行服务化模式。

0.加入pom依赖。

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

注意：这里的 `Eureka` 不是必须的。在没有注册中心的情况下，也是可以进行zuul使用的。

1.配置文件，配置注册中心相关信息、路由规则等。

```
spring.application.name=zuul-server
server.port=8888

# 注册中心地址 -此为单机模式
eureka.client.service-url.defaultZone=http://127.0.0.1:1000/eureka
# 启用ip配置 这样在注册中心列表中看见的是以ip+端口呈现的
eureka.instance.prefer-ip-address=true
# 实例名称  最后呈现地址：ip:15678
eureka.instance.instance-id=${spring.cloud.client.ip-address}:${server.port}

## 路由规则
## 传统路由配置：不依赖服务发现。
## 所有以myapi开头的url路由至http://127.0.0.1:2000/下
## 如http://127.0.0.1:8888/myapi/hello --> http://127.0.0.1:2000/hello
zuul.routes.myApi.path=/myapi/**
zuul.routes.myApi.url=http://127.0.0.1:2000

#forward模式 直接转发至zuul提供的rest服务
zuul.routes.myDemo.path=/myDemo/**
zuul.routes.myDemo.url=forward:/demo

## 服务发现模式
# 路由地址
zuul.routes.myEureka.path=/eureka/**
#为具体服务的名称
zuul.routes.myEureka.service-id=eureka-client
```

3. SpringBoot | 第十五章: an的RESTful接口测试(172
4. 关于@webFilter使用@O 题(1173)
5. SpringBoot | 第十八章: 发之WebJars使用(1134)

评论排行榜

1. SpringBoot | 第三十一章 B的集成和使用(5)
2. SpringBoot | 番外：使用 (3)
3. SpringBoot | 第二十一章 之异步调用(3)
4. SpringBoot | 第二十章: 异步请求(2)
5. Docker | 第三章：Docke (2)

推荐排行榜

1. SpringBoot | 番外：使用 (4)
2. SpringBoot | 第二十章: 异步请求(4)
3. SpringBoot | 第三十五章 的集成和使用(4)
4. Mybatis-Plus使用全解(3
5. SpringBoot | 第二十六章 (3)

友情提示：

默认情况下：Zuul代理所有注册到EurekaServer的微服务，路由规则：

`http://ZUUL_HOST:ZUUL_PORT/微服务实例名(serverId)/**`

转发至serviceId对应的微服务。

如：http://127.0.0.1:8888/eureka-client/hello?name=oKong
最后就是转发至：http://127.0.0.1:2000//hello?name=oKong

2.启动类加入 `@EnableZuulProxy` 注解，声明一个Zuul代理。

```
/**
 * zuul 示例
 *
 * @author oKong
 *
 */
@SpringBootApplication
@EnableZuulProxy
@EnableDiscoveryClient
@Slf4j
public class SpringCloudZuulApplication {
    public static void main(String[] args) throws Exception {
        SpringApplication.run(SpringCloudZuulApplication.class, args);
        log.info("spring-cloud-zuul启动!");
    }

}
```

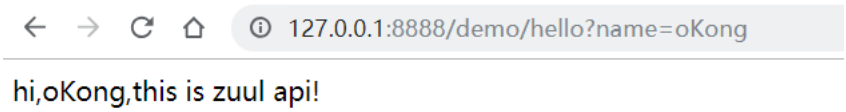
3.编写一个控制类，测试 `forward` 功能。

```
/**
 * zuul 内部提供对外服务示例
 * @author oKong
 *
 */
@RestController
@RequestMapping("/demo")
public class DemoController {

    @GetMapping("/hello")
    public String hello(String name) {
        return "hi," + name + ",this is zuul api! ";
    }
}
```

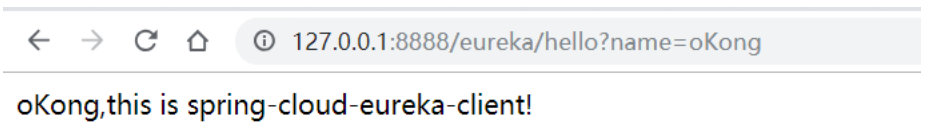
4.启动 `spring-cloud-eureka-server` 和 `spring-cloud-eureka-client` 服务，之后再启动 `spring-cloud-zuul` 服务。

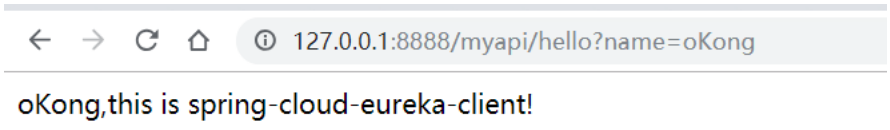
此时，我们来访问zuul内部服务：http://127.0.0.1:8888/demo/hello?name=oKong



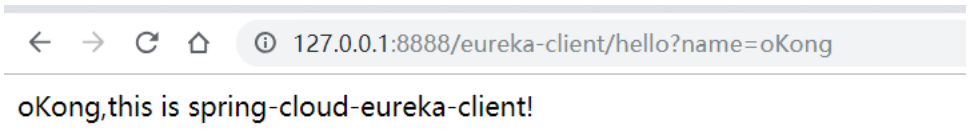
然后访问：http://127.0.0.1:8888/myDemo/hello?name=oKong 效果和上面直接方式是一样的，最后都是访问 `/demo/**` api地址的。

访问：http://127.0.0.1:8888/myapi/hello?name=oKong 和 http://127.0.0.1:8888/eureka/hello?name=oKong 最后效果都是一样的，都是访问http://127.0.0.1:2000/hello?name=oKong 。

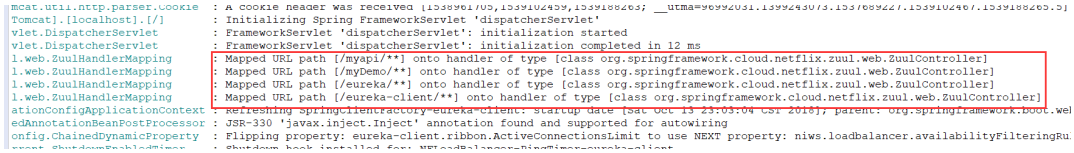




直接以默认形式访问:http://127.0.0.1:8888/eureka-client/hello?name=oKong 最后效果也是一样的。



其实，从控制台输出日志也是可以一窥究竟的:



路由规则说明

简单说明下关于路由规则的一些说明:

传统路由配置：不依赖服务发现，如nginx

- 单例实例配置：通过 `zuul.routes.<route>.path` 和 `zuul.routes.<route>.url` 参数对的方式来配置

```
# 传统路由配置
zuul.routes.server-provide.path=/server-provide/**
zuul.routes.server-provide.url=http://127.0.0.1:1001/
```

- 多实例配置：通过一组 `zuul.routes.<route>.path` 与 `zuul.routes.<route>.serviceId` 参数对的方式配置

```
# 多实例
zuul.routes.server-provide.path=/user-service/**
zuul.routes.server-provide.serviceId=user-service
ribbon.eureka.enabled=false
server-provide.ribbon.listOfServers=http://127.0.0.1:1001/,http://127.0.0.1:1001/
```

服务路由配置：依赖服务发现，结合Eureka

- 默认规则：http://ZUUL_HOST:ZUUL_PORT/微服务实例名(serverId)/** ，转发至serviceId对应的微服务。
- 自定义路由规则：通过一组 `zuul.routes.<route>.path` 与 `zuul.routes.<route>.serviceId` 参数对的方式配置

```
# 自定义路由
zuul.routes.server-provide.path=/server-api/**
zuul.routes.server-provide.serviceId=server-provide
```

比如:

```
zuul.routes.api-a.path=/api-a/**
zuul.routes.api-a.serviceId=service-ribbon
zuul.routes.api-b.path:=/api-b/**
zuul.routes.api-b.serviceId: service-feign
```

以/api-a/开头的请求都转发给service-ribbon服务
以/api-b/开头的请求都转发给service-feign服务

而且，要注意，这些过滤器是 `path` 进行最佳路径匹配的，所以，一般上在一些历史系统上，我们会在最后后面加上一个路径/的匹配规则，以保证历史api可以使用，做到最大兼容性,避免类似404的异常。**

```
zuul.routes.legacy.path=/**
```

参考资料

1. https://cloud.spring.io/spring-cloud-static/Finchley.SR1/single/spring-cloud.html#_router_and_filter_zuul

总结

> 本章节主要简单介绍了关于 `Zuul` 的一些简单使用以及一些路由规则的简单说明。开头也说过了， `Zuul` 的核心是一系列的过滤器。介于篇幅问题，关于**过滤器的介绍、自定义过滤器、异常处理、熔断降级**等放在下一章节来讲解。而相关的其他配置，大家可以去官网进行查看，或者自行搜索下，也可以查看下： `org.springframework.cloud.netflix.zuul.filters.ZuulProperties` 类，相关配置信息都在里面了。

最后

> 目前互联网上大佬都有分享 `SpringCloud` 系列教程，内容可能会类似，望多多包涵了。**原创不易，码字不易**，还希望大家多多支持。若文中有错误之处，还望提出，谢谢。

老生常谈

- 个人QQ： `499452441`
- 微信公众号： `lqdevOps`

个人博客：http://blog.lqdev.cn

源码示例：https://github.com/xie19900123/spring-cloud-learning

原文地址：http://blog.lqdev.cn/2018/10/14/SpringCloud/chapter-nine/

分类： SpringCloud

标签： springcloud， zuul

« 上一篇：白话SpringCloud | 第八章：分布式配置中心的服务化及动态刷新

» 下一篇：白话SpringCloud | 第十章：路由网关(Zuul)进阶：过滤器、异常处理

posted @ 2018-10-15 09:00 oKong_趔趄的猿 阅读(467) 评论(1) 编辑 收藏