1. API Gateway简介

⑴API Gateway是一个服务器。它是一个系统的单一入口点。API Gateway封装了内部系统架构。它提供了针对每个客户端的API。它还具有其他职责，如身份验证、监视、负载平衡、缓存、请求形成和管理以及静态响应处理。

API Gateway还负责请求路由、组合和协议转换。客户机发出的所有请求都要经过API Gateway。之后，API Gateway将请求路由到适当的微服务。

API Gateway可以为每个客户端提供一个定制的API。它还可以在两个协议之间进行转换，比如内部使用的HTTP、WebSockets和对web不友好的协议。

⑵API Gateway处理请求的两种方式：

①它将请求路由或代理到适当的服务。

②将请求散布（传播）到多个服务。

2. API Gateway的组成

Security

Caching

API composition and processing

Managing access quotas

API health monitoring

Versioning

Routing

3. API Gateway的优点

PI Gateway最重要的优点是它封装了应用程序的内部结构。

客户机直接与API网关通信，而不是调用特定的服务。

它减少了客户机和应用程序之间的往返次数。

它简化了客户机代码。

它减少了编码工作，提高了应用程序的效率，同时减少了错误。

它为每种客户机提供特定的API。

4. 微服务中API Gateway的作用

在微服务中，我们通过一个API路由所有请求。我们可以在API网关中实现认证、授权、安全、路由、服务聚合、速率限制、审计和日志记录等常见功能。

5. Zuul简介

Zuul Server是一个API Gateway应用程序。它处理所有请求并执行微服务应用程序的动态路由。它可以作为所有请求的前门。它也被称为边缘服务器。

构建Zuul是为了启用动态路由、监视、弹性和安全性。它还可以将请求路由到多个Amazon Auto Scaling Groups。

Zuul提供了一系列不同类型的过滤器，允许我们快速灵活地将功能应用到边缘服务。过滤器执行以下功能：

Authentication and Security: 它为每个资源提供身份验证需求。

Insights and Monitoring: 它跟踪有意义的数据和统计数据，使我们对生产有一个准确的看法。

Dynamic Routing: 它根据需要动态地将请求路由到不同的支持集群。

Stress Testing: 为了测试性能，它增加了到集群的流量。

Load Shedding(减载): 它为每种类型的请求分配容量，并删除超出限制的请求。

Static Response Handling: 它直接在边缘构建一些响应，而不是将它们转发到内部集群。

Multi-region Resiliency(多区弹性): 它跨AWS区域路由请求，以使我们的ELB使用多样化。

6. Zuul组件

Zuul 2.x 组件：

zuul-core：它是一个包含Zuul 2.0核心功能的库。

zuul-sample：这是Zuul 2.0的一个示例驱动程序。

Zuul 1.x components：

zuul-core：它定义了核心功能。

zuul-simple-webapp：一个web应用程序，展示了如何使用zuul-core构建应用程序的简单示例。

zuul-netflix：它是一个向Zuul添加其他NetflixOSS组件的库。

zuul-netflix-webapp：它是一个将zuul-core和zuul-netflix打包在一起的网络应用程序。

7. 创建Zuul API Gateway服务器步骤

设置Zuul API Gateway的过程分为三个步骤：

为Zuul API Gateway创建一个组件；

确定Zuul API Gateway应做的事情；

所有重要的请求都配置为通过Zuul API Gateway传递。

⑴使用SpringBoot Initializr创建SpringBoot应用程序，选择依赖Zuul，Eureka Client和DevTools。

⑵在应用程序中使用@EnableZuulProxy来开启Zuul代理。

8. 创建Zuul Logging Filter

继承ZuulFilter，实现它的四个抽象方法：

shouldFilter()：shouldFilter()方法检查请求并决定是否执行过滤器。

run()：如果！isFilterDisabled()和shouldFilter()方法都返回true，则调用run()方法。

filterType()：filterType()方法按类型对过滤器进行分类。 Zuul中有四种类型的标准过滤器：用于路由选择前过滤的pre，用于路由到源的路由，用于路由选择后过滤器的post和用于错误处理的错误。 Zuul还支持用于静态响应的静态类型。 可以通过调用方法runFilters(type)来创建或添加和运行任何过滤器类型。

filterOrder()：必须为过滤器定义过滤器顺序。如果优先级对过滤器不重要，则过滤器可能具有相同的筛选顺序。过滤器的顺序不需要是连续的。

package com.longmao.zuul.filter;

import com.netflix.zuul.ZuulFilter;

import com.netflix.zuul.context.RequestContext;

import com.netflix.zuul.exception.ZuulException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;

@Component

public class ZuulLoggingFilter extends ZuulFilter {

private Logger logger = LoggerFactory.getLogger(this.getClass());

@Override

public String filterType() {

return "pre";

}

@Override

public int filterOrder() {

return 1;

}

@Override

public boolean shouldFilter() {

return true;

}

@Override

public Object run() throws ZuulException {

8. 使用Zuul API Gateway Server执行请求

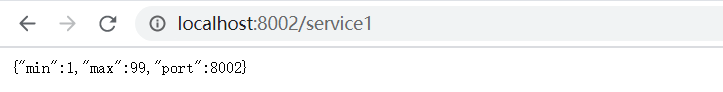
HttpServletRequest httpServletRequest = RequestContext.getCurrentContext().getRequest();

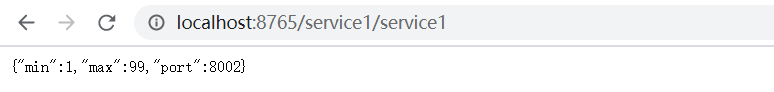
logger.info("request -> {} request uri -> {}", httpServletRequest, httpServletRequest.getRequestURI());

return null;

}

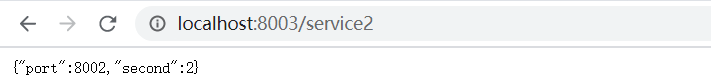
}

调用http://localhost:8002/service1，得到如下结果。

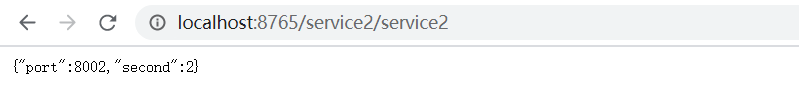
接下来我们将尝试使用Zuul API Gateway来调用请求，使用如下http://localhost:8765/{application-name}/{uri}。在这个例子中，我们的application-name为service1，uri为service1，所以URL为http://locahost:8765/service1/service1。

9. 使用Zuul API Gateway Server来进行微服务之间相互调用

⑴修改FeignClient的name属性，改为Zuul应用程序名，如@FeignClient(name = "zuul-server")。

⑵定义调用的微服务的地址，/{application-name}/{uri}，如@GetMapping("/service1/service1")

Zuul控制台输出：

上面例子只实现了微服务内部调用通过Zuul API Gateway，若与微服务的通信完全通过Zuul API Gateway，则客户端请求也满足{zuul\_server}/{application-name}/{uri}，如http://localhost:8765/service2/service2。

Zuul控制台输出：