基于springboot、spring-cloud实现网关系统相关功能。大致原理是filter过滤请求，在自定义filter中实现相应功能。

比较关键的类

DispatcherServlet

HandlerAdapter

Controller接口

ZuulController

ZuulServlet

ZuulFilter各种实现，自定义的，预定义的

ZuulProperties

请求网关的地址：http://wangguan-address(ip:port)/serviceid/url

Route是根据注册中心获取的所有服务，服务id和服务实例的真是地址，zuul会根据route中服务id来获取真实的服务地址（id端口），这其中就有负载均衡的功能。利用这一点可以做一些拦截处理，指定某些服务的请求才暴露到网关中。

下面分析一下整个过程。

Springboot通过DispatpatcherServlet拦截所有的请求/，经过HandlerMapping、HandlerAdapter分发到ZuulController， ZuulController.handleRequest， 然后调用ZuulServlet的service，调用各filter的实现。

ZuulController 调用ZuulServlet，在ServletWrappingController实例化ZuulServlet

this.servletInstance = (Servlet)ReflectionUtils.accessibleConstructor(this.servletClass, new Class[0]).newInstance();

下面对一些关键代码说明一下

DiscoveryClientRouteLocator 下面有CompositeDiscoveryClient,其下是一个DiscoveryClient列表，其中包括ConsulDiscoveryClient、SimpleDiscoveryClient（本地服务的地址）

DiscoveryClientRouteLocator 实例化的地方。

ZuulProxyAutoConfiguration

@Bean

@ConditionalOnMissingBean(DiscoveryClientRouteLocator.class)

public DiscoveryClientRouteLocator discoveryRouteLocator() {

return new DiscoveryClientRouteLocator(this.server.getServlet().getServletPrefix(), this.discovery, this.zuulProperties,

this.serviceRouteMapper, this.registration);

}

ZuulServerAutoConfiguration 这个配置类，很多zuul相关的filter都是在这里实例化的

@Bean

public ZuulController zuulController() {

return new ZuulController();

}

ConsulDiscoveryClient 实例化的地方。

ConsulDiscoveryClientConfiguration

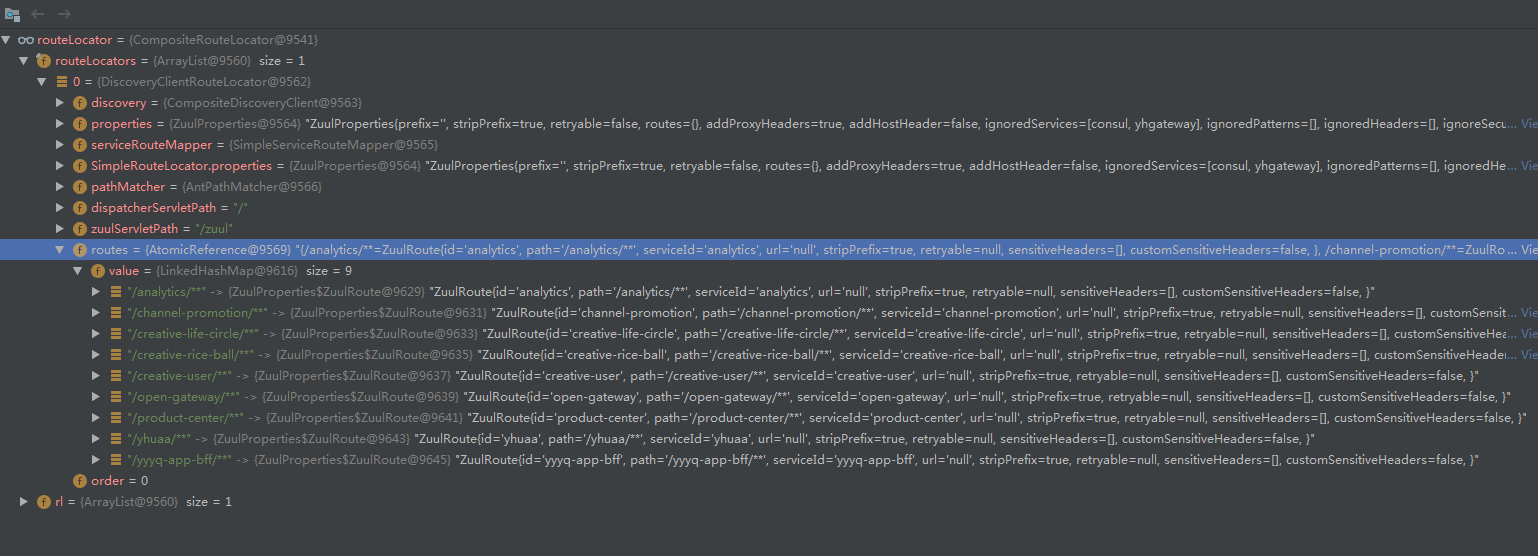
@Bean

@ConditionalOnMissingBean

public ConsulDiscoveryClient consulDiscoveryClient(ConsulDiscoveryProperties discoveryProperties) {

return new ConsulDiscoveryClient(consulClient, discoveryProperties);

}



这里的routes信息是从注册中心consul获取到的。将其转换成了Zuul请求path的pattern，例如：/product-center/\*\*

ZuulFilter 这个接口主要是过滤请求，做一些校验、检查、修改路径。实现它可以做很多事情，ZuulFilter有很多实现类，分为4类pre、route、post、error。这些filter只要实例化后，就可以实现相应的filter功能。

Springcontext中的ZuulFilter就是在这里注入到filters中的。

@Configuration

protected static class ZuulFilterConfiguration {

@Autowired

private Map<String, ZuulFilter> filters;

@Bean

public ZuulFilterInitializer zuulFilterInitializer(

CounterFactory counterFactory, TracerFactory tracerFactory) {

FilterLoader filterLoader = FilterLoader.getInstance();

FilterRegistry filterRegistry = FilterRegistry.instance();

return new ZuulFilterInitializer(this.filters, counterFactory, tracerFactory, filterLoader, filterRegistry);

}

}

ZuulRunner调用FilterProcessor相应的方法，FilterProcessor是单例的，用简单的静态变量实现的。

FilterProcessor 是调用各种filter的，pre、route、post、error等等。

这个Processor在ZuulServletFilter、ZuulServlet中实例化然后调用。

ZuulServletFilter中doFilter调用zuulRunner中pre、route、post、error 4中类型的过滤器。

ZuulServlet中service方法调用zuulRunner中pre、route、post、error 4中类型的过滤器。

他们最终实现相应的filter功能。

从FilterLoader获取pre、route、post、error 4中相应的filter的run方法，然后执行processZuulFilter(zuulFilter)

其中FilterLoader是分4类管理ZuulFilter

SendForwardFilter 这个filter是转发请求的地方。在这个filter中的run方法中

@Override

public Object run() {

try {

RequestContext ctx = RequestContext.getCurrentContext();

String path = (String) ctx.get(FORWARD\_TO\_KEY);

RequestDispatcher dispatcher = ctx.getRequest().getRequestDispatcher(path);

if (dispatcher != null) {

ctx.set(SEND\_FORWARD\_FILTER\_RAN, true);

if (!ctx.getResponse().isCommitted()) {

dispatcher.forward(ctx.getRequest(), ctx.getResponse());

ctx.getResponse().flushBuffer();

}

}

}

catch (Exception ex) {

ReflectionUtils.rethrowRuntimeException(ex);

}

return null;

}

修改请求路径

Object originalRequestPath = ctx.get(FilterConstants.REQUEST\_URI\_KEY);

String modifiedRequestPath = originalRequestPath.toString().replace("/openapi", "/api");

ctx.put(FilterConstants.REQUEST\_URI\_KEY, modifiedRequestPath);

FilterConstants 这个常量类中有很多有用的常量。

PreDecorationFilter，这个filter，根据route设置requestcontext。

RequestContext ctx = RequestContext.getCurrentContext();

ctx.setSendZuulResponse(true);// 对该请求进行路由

ctx.setSendZuulResponse(false);// 过滤该请求，不对其进行路由

ctx.set("isSuccess", true);// 设值，让下一个Filter看到上一个Filter的状态

filterType：返回一个字符串代表过滤器的类型，在zuul中定义了四种不同生命周期的过滤器类型，具体如下：

pre：可以在请求被路由之前调用

route：在路由请求时候被调用

post：在route和error过滤器之后被调用

error：处理请求时发生错误时被调用

RestTemplate是Spring提供的用于访问Rest服务的客户端，RestTemplate提供了多种便捷访问远程Http服务的方法,能够大大提高客户端的编写效率。

SimpleControllerHandlerAdapter：

表示所有实现了org.springframework.web.servlet.mvc.Controller接口的Bean可以作为Spring Web MVC中的处理器。如果需要其他类型的处理器可以通过实现HadlerAdapter来解决。

在SimpleControllerHandlerAdapter中调用Controller的。

@Nullable

public ModelAndView handle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

return ((Controller)handler).handleRequest(request, response);

}

ZuulProperties中的这个方法规范化配置的服务

public String getServletPattern() {

String path = this.servletPath;

if (!path.startsWith("/")) {

path = "/" + path;

}

if (!path.contains("\*")) {

path = path.endsWith("/") ? (path + "\*") : (path + "/\*");

}

return path;

}

类似/producet-center/\*

LocationRewriteFilter中将serviceid转换成对象对应的ip+端口的真是路径

参考文档：

Springboot mvc处理分发请求

<https://blog.csdn.net/bigtree_3721/article/details/74331208>

<https://blog.csdn.net/LeiXiaoTao_Java/article/details/83542907>

<https://www.cnblogs.com/baiduligang/p/4247164.html>

Zuul网关相关

<https://www.cnblogs.com/liangzs/p/8946740.html>

<https://www.cnblogs.com/AWSL/p/10608802.html> SpringCloud学习记录——网关（Zuul）

<https://blog.csdn.net/u012930316/article/details/80846553> springcloud zuul 过滤器修改转发的url

<https://www.cnblogs.com/a8457013/p/8352349.html>

spring cloud-zuul的Filter详解

<https://blog.csdn.net/u014091123/article/details/75433656>

Zuul动态路由及动态Filter实现

<https://my.oschina.net/10000000000/blog/1592370>

springcloud zuul 网关 持久化 动态加载路由的思路分析

<https://blog.csdn.net/u013815546/article/details/68944039>

springcloud----Zuul动态路由

<https://www.cnblogs.com/a8457013/p/8352349.html>

spring cloud-zuul的Filter详解

<https://www.cnblogs.com/duanxz/p/7527765.html>

服务网关zuul之一：入门介绍

<https://www.cnblogs.com/huangjuncong/p/9060984.html>

SpringCloud实战6-Zuul网关服务