链滴

# Feign 使用 okhttp3 的正确姿势

作者：leejoker

原文链接：https://ld246.com/article/1617766120262

来源网站：链滴

# 首先来吐槽一波(ノ'□')ノ ⌒┻━┻

本来呢，我是想加一层feign的interceptor处理feign里request请求的返回信息的，比如只提取ResponseBody中的data啥的。后来想起来feign默认使用的是jdk的HttpURLConnection，而且feign本身支持替换okhttp的，于是打算搞起~

可是，百毒到的都是什么鬼啊，按照别人写的文档配好，各种问题，什么springboot注解不行啦，需使用feign默认注解，什么负载均衡失效啦，无语。(ノ'□')ノ ⌒┻┻

最后，还是自己操刀，从源码看吧。

遂有此文~

百毒，淦!

# 首先来写一个简单的interceptor

不需要这个的跳过看下一段吧~

```
//这个interceptor的目的是提取返回body中的data，并转化为json
@Component
public class OkHttpResponseInterceptor implements Interceptor {
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request request = chain.request();
        Response response = chain.proceed(request);

        if (HttpHeaders.hasBody(response)) {
            if (response.code() == 200) {
                ResponseBody responseBody = response.body();
                if (responseBody != null
                        && responseBody.contentLength() != 0
                        && Objects.requireNonNull(responseBody.contentType()).type()
                        .equals(MediaType.APPLICATION_JSON_VALUE)) {
                    String str = responseBody.string();
                    JSONObject json = JSONObject.parseObject(JSON.toJSON(str));
                    String data = json.getString("data");
                    if (StringUtils.isNotBlank(data)) {
                        ResponseBody body = ResponseBody.create(okhttp3.MediaType.get(MediaTy
e.APPLICATION_JSON_VALUE), data);
                        return response.newBuilder().body(body).build();
                    }
                }
            }
        }
        return response;
    }
}
```

嗯，这样，interceptor就定义好了，接下来就是配置feign了~

# 配置Feign

关于maven依赖修改啥的我就不说了，如果使用的spring-cloud-openfeign-dependencies，应该包含okhttp的依赖。

1. 修改application.yml

```
feign:
  hystrix:
    enabled: true
  okhttp:
    enabled: true
  httpclient:
    connectionTimeout: 30000
  client:
    config:
      default:
        readTimeout: 30000
```

2. 添加FeignOkHttpConfig.java

```
@Configuration
@ConditionalOnClass(Feign.class)
@AutoConfigureBefore(FeignLoadBalancerAutoConfiguration.class)
public class FeignOkHttpConfig {

    @Autowired
    private OkHttpResponseInterceptor okHttpResponseInterceptor;

    private okhttp3.OkHttpClient okHttpClient;

    @Bean
    @ConditionalOnMissingBean(ConnectionPool.class)
    public ConnectionPool httpClientConnectionPool(
            FeignHttpClientProperties httpClientProperties,
            OkHttpClientConnectionPoolFactory connectionPoolFactory) {
        Integer maxTotalConnections = httpClientProperties.getMaxConnections();
        Long timeToLive = httpClientProperties.getTimeToLive();
        TimeUnit ttlUnit = httpClientProperties.getTimeToLiveUnit();
        return connectionPoolFactory.create(maxTotalConnections, timeToLive, ttlUnit);
    }

    @Bean
    @ConditionalOnMissingBean(okhttp3.OkHttpClient.class)
    public okhttp3.OkHttpClient okHttpClient(OkHttpClientFactory httpClientFactory,
                                             ConnectionPool connectionPool,
                                             FeignClientProperties feignClientProperties,
                                             FeignHttpClientProperties feignHttpClientProperties) {
        FeignClientProperties.FeignClientConfiguration defaultConfig = feignClientProperties.ge
Config().get("default");
        int connectTimeout = feignHttpClientProperties.getConnectionTimeout();
        int readTimeout = defaultConfig.getReadTimeout();
        boolean disableSslValidation = feignHttpClientProperties.isDisableSslValidation();
        boolean followRedirects = feignHttpClientProperties.isFollowRedirects();
        this.okHttpClient = httpClientFactory.createBuilder(disableSslValidation)
                .readTimeout(readTimeout, TimeUnit.MILLISECONDS)
                .connectTimeout(connectTimeout, TimeUnit.MILLISECONDS)
```

```
                .followRedirects(followRedirects)
                .connectionPool(connectionPool)
                .addInterceptor(okHttpResponseInterceptor)
                .build();
        return this.okHttpClient;
    }

    @PreDestroy
    public void destroy() {
        if (this.okHttpClient != null) {
            this.okHttpClient.dispatcher().executorService().shutdown();
            this.okHttpClient.connectionPool().evictAll();
        }
    }
}
```

好的~ 这样基本上就配置完了。下面我来具体解释一下，这么配置就能生效的原因。（如果不需要负均衡，可以参考百毒到的配置方案，那个应该也是ok的）

# 解析

1. 什么条件下配置的okhttp才会生效

首先看org.springframework.cloud.openfeign.FeignAutoConfiguration

```
@Configuration(proxyBeanMethods = false)
@ConditionalOnClass(OkHttpClient.class)
@ConditionalOnMissingClass("com.netflix.loadbalancer.ILoadBalancer")
@ConditionalOnMissingBean(okhttp3.OkHttpClient.class)
@ConditionalOnProperty("feign.okhttp.enabled")
protected static class OkHttpFeignConfiguration {
...
}
```

只有当okhttp3.OkHttpClient这个Bean不存在时，才会启用OkHttpFeignConfiguration。

然而我们在配置中需要修改interceptor必然会手动创建这个Bean，因此我们需要手动添加其他的配。

但是，先不要急，因为如果使用负载均衡，这个类还不是关键。

2. feign负载均衡FeignLoadBalancerAutoConfiguration

```
@ConditionalOnClass(Feign.class)
@ConditionalOnBean(BlockingLoadBalancerClient.class)
@AutoConfigureBefore(FeignAutoConfiguration.class)
@AutoConfigureAfter(FeignRibbonClientAutoConfiguration.class)
@EnableConfigurationProperties(FeignHttpClientProperties.class)
@Configuration(proxyBeanMethods = false)
// Order is important here, last should be the default, first should be optional
// see
// https://github.com/spring-cloud/spring-cloud-netflix/issues/2086#issuecomment-3162816
3
@Import({ HttpClientFeignLoadBalancerConfiguration.class,
        OkHttpFeignLoadBalancerConfiguration.class,
        DefaultFeignLoadBalancerConfiguration.class })
```

```
public class FeignLoadBalancerAutoConfiguration {

}
```

不难发现，这个配置加载是在FeignAutoConfiguration之前的，因此，这个类对于我们而言更为关
。

```
@Import({ HttpClientFeignLoadBalancerConfiguration.class,
        OkHttpFeignLoadBalancerConfiguration.class,
        DefaultFeignLoadBalancerConfiguration.class })
```

通过Import注解的信息，我们得知需要查看OkHttpFeignLoadBalancerConfiguration。

3. 根据OkHttpFeignLoadBalancerConfiguration

```
@Configuration(proxyBeanMethods = false)
@ConditionalOnClass(OkHttpClient.class)
@ConditionalOnProperty("feign.okhttp.enabled")
@ConditionalOnBean(BlockingLoadBalancerClient.class)
@Import(OkHttpFeignConfiguration.class)
class OkHttpFeignLoadBalancerConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public Client feignClient(okhttp3.OkHttpClient okHttpClient,
            BlockingLoadBalancerClient loadBalancerClient) {
        OkHttpClient delegate = new OkHttpClient(okHttpClient);
        return new FeignBlockingLoadBalancerClient(delegate, loadBalancerClient);
    }

}
```

可以看出，这个配置类只生成了Client这个Bean，对于FeignAutoConfiguration中需要的剩下的Bea
显然是不够的，因此，剩下的内容应该都在

```
@Import(OkHttpFeignConfiguration.class)
```

这个Import的配置中，那么我们看看这个配置到底做了啥。

4. 最后一步了~

```
@Configuration(proxyBeanMethods = false)
@ConditionalOnMissingBean(okhttp3.OkHttpClient.class)
public class OkHttpFeignConfiguration {

    private okhttp3.OkHttpClient okHttpClient;

    @Bean
    @ConditionalOnMissingBean(ConnectionPool.class)
    public ConnectionPool httpClientConnectionPool(
            FeignHttpClientProperties httpClientProperties,
            OkHttpClientConnectionPoolFactory connectionPoolFactory) {
        Integer maxTotalConnections = httpClientProperties.getMaxConnections();
        Long timeToLive = httpClientProperties.getTimeToLive();
        TimeUnit ttlUnit = httpClientProperties.getTimeToLiveUnit();
        return connectionPoolFactory.create(maxTotalConnections, timeToLive, ttlUnit);
    }
```

```
    @Bean
    public okhttp3.OkHttpClient client(OkHttpClientFactory httpClientFactory,
            ConnectionPool connectionPool,
            FeignHttpClientProperties httpClientProperties) {
        Boolean followRedirects = httpClientProperties.isFollowRedirects();
        Integer connectTimeout = httpClientProperties.getConnectionTimeout();
        this.okHttpClient = httpClientFactory
                .createBuilder(httpClientProperties.isDisableSslValidation())
                .connectTimeout(connectTimeout, TimeUnit.MILLISECONDS)
                .followRedirects(followRedirects).connectionPool(connectionPool).build();
        return this.okHttpClient;
    }

    @PreDestroy
    public void destroy() {
        if (this.okHttpClient != null) {
            this.okHttpClient.dispatcher().executorService().shutdown();
            this.okHttpClient.connectionPool().evictAll();
        }
    }
}
```

很显然，这个类才是我们需要替换的，而且这个配置类的加载条件很简单

```
@ConditionalOnMissingBean(okhttp3.OkHttpClient.class)
```

我们只要自己创建这个Bean就可以了。

那么，我们要做的就是在FeignLoadBalancerAutoConfiguration配置类加载之前，生成这个Bean，根据OkHttpFeignConfiguration生成其他需要的Bean就可以了。具体参考 配置Feign 一节。