前言

Feign是一个声明式的web service客户端。Spring Cloud中的Open Feign在Feign的基础上支持Spring MVC注解、 JAX-RS注解，同时集成了Ribbon、Eureka（对于负载均衡，Spring Cloud LoadBalancer也同样支持）。

> Feign is a declarative web service client. It makes writing web service clients easier. To use Feign create an interface and annotate it. It has pluggable annotation support including Feign annotations and JAX-RS annotations. Feign also supports pluggable encoders and decoders. Spring Cloud adds support for Spring MVC annotations and for using the same HttpMessageConverters used by default in Spring Web. Spring Cloud integrates Ribbon and Eureka, as well as Spring Cloud LoadBalancer to provide a load-balanced http client when using Feign.

基于接口生成动态代理，进行远程通信。

原理概述

对@EnableFeignClients注解进行解析。

针对指定的Feign Client生成动态代理，以及对它的方法描述进行解析。

组装成一个Request对象，发起请求。

代码样例

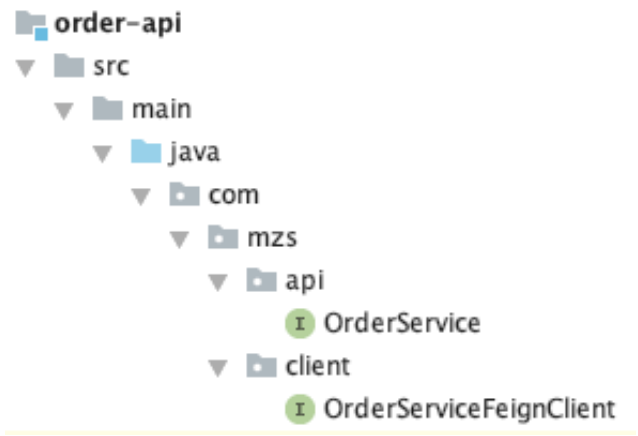
方式一

在服务提供者模块定义Feign Client接口

1、order-api模块

```java
package com.mzs.api;

@RequestMapping("/order")
public interface OrderService {

  @GetMapping
  String getAllOrders();
}
```

```java
package com.mzs.client;

@FeignClient("order-service")
public interface OrderServiceFeignClient extends OrderService {

}
```



然后对order-api模块执行`mvn clean install`。

2、order-service模块

需要在pom.xml文件中引入order-api的jar包。

```properties
# application.properties
spring.application.name=order-service
server.port=8762
```

```java
package com.mzs.order_service.controller;

@RestController
public class OrderServiceImpl implements OrderService {

  @Override
  public String getAllOrders() {
    return "success";
  }
}
```

3、user-service模块

需要在pom.xml文件中引入order-api的jar包。

```java
package com.mzs.user_service.controller;

@RestController
@RequestMapping("/user")
public class UserController {

  private final OrderService orderService;
  // private final OrderServiceFeignClient feignClient;

  @Auwotired
  public UserController(OrderService orderService) {
    this.orderService = orderService;
  }

  @GetMapping("/orders")
  public String getAllOrders() {
    return orderService.getAllOrders();
  }
}
```

```
package com.mzs.user_service;

@SpringBootApplication
// 指定OrderServiceFeignClient所在包路径
@EnableFeignClients("com.mzs.client")
public class UserSpringApplication {

  public static void main(String[] args) {
    SpringApplication.run(UserSpringApplication.class, args);
  }
}
```

方式二

在服务消费者模块定义Feign Client接口

user-service模块

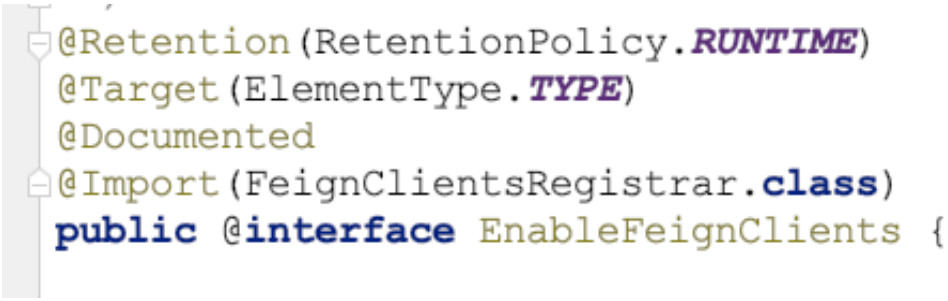
```
package com.mzs.user_service.clients;

@RequestMapping("/order")
public interface OrderServiceFeignClient {

  @GetMapping
  String getAllOrders();
}
```

```
package com.mzs.user_service.controller;

@RestController
@RequestMapping("/user")
public class UserController {

  private final OrderServiceFeignClient feignClient;

  @Auwotired
  public UserController(OrderServiceFeignClient feignClient) {
    this.feignClient = feignClient;
  }

  @GetMapping("/orders")
  public String getAllOrders() {
```

```
    return feignClient.getAllOrders();
  }
}
```

源码分析

@EnableFeignClients



作用：扫描所有标注@FeignClient注解的接口。

FeignClientRegistrar

ImportBeanDefinitionRegistrar接口的实现类

```
@Override
public void registerBeanDefinitions(AnnotationMetadata metadata,
                                    BeanDefinitionRegistry registry) {
  registerDefaultConfiguration(metadata, registry);
  registerFeignClients(metadata, registry);
}
```

```
private void registerDefaultConfiguration(AnnotationMetadata metadata,
                                          BeanDefinitionRegistry registry) {
  // 获取@EnableFeignClients注解的属性以及属性值
  Map<String, Object> defaultAttrs = metadata
    .getAnnotationAttributes(EnableFeignClients.class.getName(), true);
```

```
  // 如果有指定defaultConfiguration属性
  if (defaultAttrs != null && defaultAttrs.containsKey("defaultConfiguration"))
{
    String name;
    if (metadata.hasEnclosingClass()) {
      name = "default." + metadata.getEnclosingClassName();
    }
    else {
      name = "default." + metadata.getClassName();
    }
    // 注册@EnableFeignClients注解指定的配置类（将配置类封装到
FeignClientSpecification中进行注册）
    registerClientConfiguration(registry, name,
                                defaultAttrs.get("defaultConfiguration"));
  }
}
```

```
public void registerFeignClients(AnnotationMetadata metadata,
                                 BeanDefinitionRegistry registry) {

  LinkedHashSet<BeanDefinition> candidateComponents = new LinkedHashSet<>();
  Map<String, Object> attrs = metadata
    .getAnnotationAttributes(EnableFeignClients.class.getName());
  AnnotationTypeFilter annotationTypeFilter = new AnnotationTypeFilter(
    FeignClient.class);
  final Class<?>[] clients = attrs == null ? null
    : (Class<?>[]) attrs.get("clients");
  if (clients == null || clients.length == 0) {
    ClassPathScanningCandidateComponentProvider scanner = getScanner();
    scanner.setResourceLoader(this.resourceLoader);
    scanner.addIncludeFilter(new AnnotationTypeFilter(FeignClient.class));
    Set<String> basePackages = getBasePackages(metadata);
    for (String basePackage : basePackages) {
      // 扫描指定路径的所有标注@FeignClient注解的接口
      candidateComponents.addAll(scanner.findCandidateComponents(basePackage));
    }
  }
  else {
    for (Class<?> clazz : clients) {
      candidateComponents.add(new AnnotatedGenericBeanDefinition(clazz));
    }
  }

  for (BeanDefinition candidateComponent : candidateComponents) {
```

```
    if (candidateComponent instanceof AnnotatedBeanDefinition) {
      // verify annotated class is an interface
      AnnotatedBeanDefinition beanDefinition = (AnnotatedBeanDefinition)
candidateComponent;
      AnnotationMetadata annotationMetadata = beanDefinition.getMetadata();
      Assert.isTrue(annotationMetadata.isInterface(),
                    "@FeignClient can only be specified on an interface");

      // 获取@FeignClient注解的属性以及属性值
      Map<String, Object> attributes = annotationMetadata
        .getAnnotationAttributes(FeignClient.class.getCanonicalName());

      String name = getClientName(attributes);
      // 注册@FeignClient注解指定的配置类（将配置类封装到FeignClientSpecification中进
行注册）
      registerClientConfiguration(registry, name,
                                  attributes.get("configuration"));

      // 注册FeignClientFactoryBean
      registerFeignClient(registry, annotationMetadata, attributes);
    }
  }
}
```

FeignClientFactoryBean

FactoryBean接口的实现类

```
@Override
public Object getObject() throws Exception {
  return getTarget();
}
```

```
<T> T getTarget() {
  // 获取FeignContext
  FeignContext context = applicationContext.getBean(FeignContext.class);
  Feign.Builder builder = feign(context);

  if (!StringUtils.hasText(url)) {
    if (!name.startsWith("http")) {
      url = "http://" + name;
```

```
    }
    else {
      url = name;
    }
    url += cleanPath();
    // 负载均衡
    return (T) loadBalance(builder, context,
                           new HardCodedTarget<>(type, name, url));
  }
  if (StringUtils.hasText(url) && !url.startsWith("http")) {
    url = "http://" + url;
  }
  String url = this.url + cleanPath();
  Client client = getOptional(context, Client.class);
  if (client != null) {
    if (client instanceof LoadBalancerFeignClient) {
      // not load balancing because we have a url,
      // but ribbon is on the classpath, so unwrap
      client = ((LoadBalancerFeignClient) client).getDelegate();
    }
    if (client instanceof FeignBlockingLoadBalancerClient) {
      // not load balancing because we have a url,
      // but Spring Cloud LoadBalancer is on the classpath, so unwrap
      client = ((FeignBlockingLoadBalancerClient) client).getDelegate();
    }
    builder.client(client);
  }
  Targeter targeter = get(context, Targeter.class);
  return (T) targeter.target(this, builder, context,
                             new HardCodedTarget<>(type, name, url));
}
```

```
protected <T> T loadBalance(Feign.Builder builder, FeignContext context,
                            HardCodedTarget<T> target) {
  // LoadBalancerFeignClient
  Client client = getOptional(context, Client.class);
  if (client != null) {
    builder.client(client);
    // DefaultTargeter
    Targeter targeter = get(context, Targeter.class);
    return targeter.target(this, builder, context, target);
  }

  throw new IllegalStateException(
    "No Feign Client for loadBalancing defined. Did you forget to include
spring-cloud-starter-netflix-ribbon?");
```

```
}
```

DefaultTargeter

```
class DefaultTargeter implements Targeter {

  @Override
  public <T> T target(FeignClientFactoryBean factory, Feign.Builder feign,
      FeignContext context, Target.HardCodedTarget<T> target) {
    return feign.target(target);
  }

}
```

ReflectiveFeign

```
public <T> T target(Target<T> target) {
  // 生成ReflectiveFeign代理实例
  return build().newInstance(target);
}
```

```
@Override
public <T> T newInstance(Target<T> target) {
  // 针对Feign Client的方法描述进行解析
  Map<String, MethodHandler> nameToHandler =
targetToHandlersByName.apply(target);
  Map<Method, MethodHandler> methodToHandler = new LinkedHashMap<Method,
MethodHandler>();
  List<DefaultMethodHandler> defaultMethodHandlers = new
LinkedList<DefaultMethodHandler>();

  for (Method method : target.type().getMethods()) {
    if (method.getDeclaringClass() == Object.class) {
      continue;
```

```
    } else if (Util.isDefault(method)) {
      DefaultMethodHandler handler = new DefaultMethodHandler(method);
      defaultMethodHandlers.add(handler);
      methodToHandler.put(method, handler);
    } else {
      methodToHandler.put(method,
nameToHandler.get(Feign.configKey(target.type(), method)));
    }
  }
  // FeignInvocationHandler
  InvocationHandler handler = factory.create(target, methodToHandler);
  // 针对指定的FeignClient生成动态代理类
  T proxy = (T) Proxy.newProxyInstance(target.type().getClassLoader(),
                                       new Class<?>[] {target.type()},
handler);

  for (DefaultMethodHandler defaultMethodHandler : defaultMethodHandlers) {
    defaultMethodHandler.bindTo(proxy);
  }
  return proxy;
}
```

SpringMvcContract

```
@Override
public MethodMetadata parseAndValidateMetadata(Class<?> targetType, Method
method) {
  processedMethods.put(Feign.configKey(targetType, method), method);
  // 解析、验证元数据。将相关信息封装到MethodMetadata。
  MethodMetadata md = super.parseAndValidateMetadata(targetType, method);

  RequestMapping classAnnotation = findMergedAnnotation(targetType,
                                                        RequestMapping.class);
  if (classAnnotation != null) {
    // produces - use from class annotation only if method has not specified
this
    if (!md.template().headers().containsKey(ACCEPT)) {
      // 根据解析出来的元数据，构建RequestTemplate
      parseProduces(md, method, classAnnotation);
    }
```

```
    // consumes -- use from class annotation only if method has not specified
this
    if (!md.template().headers().containsKey(CONTENT_TYPE)) {
      // 构建RequestTemplate
      parseConsumes(md, method, classAnnotation);
    }


    // 构建RequestTemplate
    parseHeaders(md, method, classAnnotation);
  }
  return md;
}
```

FeignInvocationHandler

Feign Client方法的调用会触发InvocationHandler的调用。

```
@Override
public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
  if ("equals".equals(method.getName())) {
    try {
      Object otherHandler =
        args.length > 0 && args[0] != null ?
Proxy.getInvocationHandler(args[0]) : null;
      return equals(otherHandler);
    } catch (IllegalArgumentException e) {
      return false;
    }
  } else if ("hashCode".equals(method.getName())) {
    return hashCode();
  } else if ("toString".equals(method.getName())) {
    return toString();
  }

  // dispatch：Map<Method, MethodHandler>
  return dispatch.get(method).invoke(args);
}
```

SynchronousMethodHandler

```
@Override
public Object invoke(Object[] argv) throws Throwable {
  // 构建RequestTemplate
  RequestTemplate template = buildTemplateFromArgs.create(argv);
  Options options = findOptions(argv);
  Retryer retryer = this.retryer.clone();
  while (true) {
    try {
      return executeAndDecode(template, options);
    } catch (RetryableException e) {
      try {
        retryer.continueOrPropagate(e);
      } catch (RetryableException th) {
        Throwable cause = th.getCause();
        if (propagationPolicy == UNWRAP && cause != null) {
          throw cause;
        } else {
          throw th;
        }
      }
      if (logLevel != Logger.Level.NONE) {
        logger.logRetry(metadata.configKey(), logLevel);
      }
      continue;
    }
  }
}
```

```
@Override
public RequestTemplate create(Object[] argv) {
  RequestTemplate mutable = RequestTemplate.from(metadata.template());
  mutable.feignTarget(target);
  if (metadata.urlIndex() != null) {
    int urlIndex = metadata.urlIndex();
    checkArgument(argv[urlIndex] != null, "URI parameter %s was null",
urlIndex);
    mutable.target(String.valueOf(argv[urlIndex]));
  }
  Map<String, Object> varBuilder = new LinkedHashMap<String, Object>();
  for (Entry<Integer, Collection<String>> entry :
metadata.indexToName().entrySet()) {
    int i = entry.getKey();
    Object value = argv[entry.getKey()];
    if (value != null) { // Null values are skipped.
```

```
      if (indexToExpander.containsKey(i)) {
        value = expandElements(indexToExpander.get(i), value);
      }
      for (String name : entry.getValue()) {
        varBuilder.put(name, value);
      }
    }
  }

  // 编码，并解析RequestTemplate
  RequestTemplate template = resolve(argv, mutable, varBuilder);
  if (metadata.queryMapIndex() != null) {
    // add query map parameters after initial resolve so that they take
    // precedence over any predefined values
    Object value = argv[metadata.queryMapIndex()];
    Map<String, Object> queryMap = toQueryMap(value);
    template = addQueryMapQueryParameters(queryMap, template);
  }

  if (metadata.headerMapIndex() != null) {
    template =
      addHeaderMapHeaders((Map<String, Object>)
argv[metadata.headerMapIndex()], template);
  }

  return template;
}
```

```
Object executeAndDecode(RequestTemplate template, Options options) throws
Throwable {
  // 组装Request对象
  Request request = targetRequest(template);

  if (logLevel != Logger.Level.NONE) {
    logger.logRequest(metadata.configKey(), logLevel, request);
  }

  Response response;
  long start = System.nanoTime();
  try {
    // 发送请求 - LoadBalancerFeignClient
    response = client.execute(request, options);
    // ensure the request is set. TODO: remove in Feign 12
    response = response.toBuilder()
      .request(request)
      .requestTemplate(template)
```

```
        .build();
    } catch (IOException e) {
      if (logLevel != Logger.Level.NONE) {
        logger.logIOException(metadata.configKey(), logLevel, e,
elapsedTime(start));
      }
      throw errorExecuting(request, e);
    }
    long elapsedTime = TimeUnit.NANOSECONDS.toMillis(System.nanoTime() - start);

    // 解码
    if (decoder != null)
      return decoder.decode(response, metadata.returnType());

    CompletableFuture<Object> resultFuture = new CompletableFuture<>();
    asyncResponseHandler.handleResponse(resultFuture, metadata.configKey(),
response,
                                        metadata.returnType(),
                                        elapsedTime);

    try {
      if (!resultFuture.isDone())
        throw new IllegalStateException("Response handling not done");

      return resultFuture.join();
    } catch (CompletionException e) {
      Throwable cause = e.getCause();
      if (cause != null)
        throw cause;
      throw e;
    }
}
```

LoadBalancerFeignClient

```
@Override
public Response execute(Request request, Request.Options options) throws
IOException {
    try {
      URI asUri = URI.create(request.url());
      String clientName = asUri.getHost();
      URI uriWithoutHost = cleanUrl(request.url(), clientName);
```

```
    FeignLoadBalancer.RibbonRequest ribbonRequest = new
FeignLoadBalancer.RibbonRequest(
      this.delegate, request, uriWithoutHost);

    IClientConfig requestConfig = getClientConfig(options, clientName);
    // FeignLoadBalancer#executeWithLoadBalancer
    return lbClient(clientName)
      // 发送请求
      .executeWithLoadBalancer(ribbonRequest, requestConfig).toResponse();
  }
  catch (ClientException e) {
    IOException io = findIOException(e);
    if (io != null) {
      throw io;
    }
    throw new RuntimeException(e);
  }
}
```