## 17. Declarative REST Client: Feign

[Feign](https://github.com/Netflix/feign)是一个声明式的Web服务客户端。它让使用Web服务更简单。要使用Feign，需要创建一个接口并添加注解。它提供插拔式的Feign注解和JAX-RS注解支持。Feign还支持可插拔编码器和解码器。Spring Cloud增加了对Spring MVC注解的支持，以及在Spring Web项目中默认使用同样的HttpMessageConverters。Spring Cloud集成Ribbon和Eureka以在使用Feign时提供负载均衡的http客户端。

## 17.1 How to Include Feign

要在您的项目中使用Feign，需要引入group org.springframework.cloud artifact ID spring-cloud-starter-feign。

*@Configuration*

*@ComponentScan*

*@EnableAutoConfiguration*

*@EnableFeignClients*

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

**StoreClient.java.**

*@FeignClient("stores")*

**public** **interface** StoreClient {

*@RequestMapping(method = RequestMethod.GET, value = "/stores")*

List<Store> getStores();

*@RequestMapping(method = RequestMethod.POST, value = "/stores/{storeId}", consumes = "application/json")*

Store update(*@PathVariable("storeId")* Long storeId, Store store);

}

@FeignClient注解的value（上文中的“stores”）是客户端名字（微服务application name），这个名字被用于创建Ribbon负载平衡器（[有关Ribbon支持的详细信息，](https://springcloud.cc/spring-cloud-dalston.html" \l "spring-cloud-ribbon)请参阅[下文](https://springcloud.cc/spring-cloud-dalston.html#spring-cloud-ribbon)））。您还可以使用注解的url属性（值可为绝对路径或主机名 http://ip:port/api）指定URL。这些接口在Application context中的实例化的bean名称是该接口的完全限定名称。要指定您自己的别名值，您可以使用@FeignClient注解的qualifier属性。

Ribbon客户端将会寻找“stores”服务的物理地址。如果您的应用程序是Eureka客户端，那么它将在Eureka服务注册中解析该服务。如果您不想使用Eureka，你可以配置 stores的服务列表（细节参考16.6）。

## 17.2 Overriding Feign Defaults

Spring Cloud的Feign支持的核心概念是声明式客户端。每个feign客户端都是feign全体组件的一部分，它们一起工作以根据需要联系远程服务器，并且该集合有一个名字用在@FeignClient注解中。Spring cloud 每一个使用 FeignClientsConfiguration 生成的声明式客户端创建一个application context。这里面包含 一个 feign.Decoder, 一个 feign.Encoder 和一个 feign.Contract.

Spring cloud提供通过@FeignClient.添加添额外的配置的方法让你完全控制feign client。例如

*@FeignClient(name = "stores", configuration = FooConfiguration.class)*

**public** **interface** StoreClient {

*//..*

}

在这种情况下，client由FeignClientsConfiguration中的组件和FooConfiguration中的组件共同组成（后者会覆盖先者）。

* 警告：FooConfiguration不需要使用@Configuration注解。如果加上了，需要将它从@ComponentScan注解中排除否则它将会作为的 feign.Decoder, feign.Encoder, feign.Contract 等等组件的默认来源，如果加上了@Configuration注解，你可以将它放在一个分离的，非重叠性的 @ComponentScan 注解或者@SpringBootApplication 注解扫描包中，或者在@ComponentScan中显示的排除掉

注意：该 serviceId 已经过时，建议使用 name 属性

警告：以前，使用 url 属性，则 name 不是必须的，但现在是必须的.

name 和 url 属性都支持占位符。

*@FeignClient(name = "${feign.name}", url = "${feign.url}")*

**public** **interface** StoreClient {

*//..*

}

Spring Cloud Netfix 默认给 feign 提供下列的beans（BeanType beanName: ClassName）

Decoder feignDecoder: ResponseEntityDecoder (包装了SpringDecoder)  
*Encoder feignEncoder: SpringEncoder*  
Logger feignLogger: Slf4jLogger  
*Contrac feignContract: SpringMvcContract //因此可以试用spring mvc注解*  
Feign.Builder feignBuilder: HystrixFeign.Builder  
*Client feignClient: 如果开启了Ribbon使用LoadBalancerFeignClient, 否则使用默认的 feign Client.*

可以通过将feign.okhttp.enabled或feign.httpclient.enabled设置为true，并将它们放在类路径上来使用OkHttpClient和ApacheHttpClient feign客户端。你可以提供一个ClosableHttpClient 来自定义http client，当你使用apache or ok时。

Spring Cloud Netfix 默认没有给feign装配下列的beans，但是在创建feign 客户端时会在application context中查找并使用这些类型的bean:

* Logger.Level
* Retryer
* ErrorDecoder
* Request.Options
* Collection<RequestInterceptor>
* SetterFactory

创建这些类型的一个bean并放在@FeignClient配置中(如上FooConfiguration),允许你覆盖所描述的每一个bean. 例子:

*@Configuration*

**public** **class** FooConfiguration {

*@Bean*

**public** Contract feignContract() {

**return** **new** feign.Contract.Default();

}

*@Bean*

**public** BasicAuthRequestInterceptor basicAuthRequestInterceptor() {

**return** **new** BasicAuthRequestInterceptor("user", "password");

}

}

这将SpringMvcContract替换为feign.Contract.Default，并将一个RequestInterceptor添加到RequestInterceptor的集合中。

@FeignClient 也可以在配置文件中设置

application.yml

feign:

client:

config:

feignName:

connectTimeout: 5000

readTimeout: 5000

loggerLevel: full

errorDecoder: com.example.SimpleErrorDecoder

retryer: com.example.SimpleRetryer

requestInterceptors:

- com.example.FooRequestInterceptor

- com.example.BarRequestInterceptor

decode404: **false**

可以在@EnableFeignClients属性defaultConfiguration中以与上述相似的方式指定默认配置。不同之处在于，此配置将适用于feign客户端。

如果你选择使用配置文件设置所有的@FeignClient，你可以使用 “default” feign name（可以针对某一个feignclient设置）

application.yml

feign:

client:

config:

default:

connectTimeout: 5000

readTimeout: 5000

loggerLevel: basic

如果你同时创建了@Configuration bean和配置文件，配置文件优先级搞，它将会覆盖@Configuration bean属性。如果你要改变优先级策略，你可以设置feign.client.default-to-properties为 false.

注意: 如果你需要在 RequestInterceptor 中使用 ThreadLocal 去绑定变量，你需要设置对应的hystrix线程隔离策略为“SEMAPHORE”或取消Hystrix在Feign中的使用.

application.yml

*# To disable Hystrix in Feign*

feign:

hystrix:

enabled: **false**

*# To set thread isolation to SEMAPHORE*

hystrix:

command:

default:

execution:

isolation:

strategy: SEMAPHORE

## 17.3 Creating Feign Clients Manually

在一些情况下可能需要自定义Feign clients但是不能用以上的方法。所以你可以使用[Feign Builder API](https://github.com/OpenFeign/feign/#basics)创建clients。下面是一个例子，创建了两个相同接口的client但是用配置了不同的拦截器。

*@Import(FeignClientsConfiguration.class)*

**class** FooController {

**private** FooClient fooClient;

**private** FooClient adminClient;

*@Autowired*

**public** FooController(

Decoder decoder, Encoder encoder, Client client) {

**this**.fooClient = Feign.builder().client(client)

.encoder(encoder)

.decoder(decoder)

.requestInterceptor(**new** BasicAuthRequestInterceptor("user", "user"))

.target(FooClient.**class**, "http://PROD-SVC");

**this**.adminClient = Feign.builder().client(client)

.encoder(encoder)

.decoder(decoder)

.requestInterceptor(**new** BasicAuthRequestInterceptor("admin", "admin"))

.target(FooClient.**class**, "http://PROD-SVC");

}

}

|  |  |
| --- | --- |
| 注意 | 在上面的例子中，FeignClientsConfiguration.class是Spring Cloud Netflix提供的默认配置。 |
| 注意 | PROD-SVC是客户端将要求的服务的名称。 |

## 17.4 Feign Hystrix Support

如果Hystrix在classpath中并设置feign.hystrix.enabled=true, Feign用熔断器包装所有方法。返回一个 com.netflix.hystrix.HystrixCommand。这允许你以相应模式使用（使用.toObservable()或者.observer()）或者 异步调用（.queue()）。

要在一个客户端上禁用 Hystrix 支持，创建一个 Feign.Builder 并将scope 设置为”prototype”,例如:

*@Configuration*

**public** **class** FooConfiguration {

*@Bean*

*@Scope("prototype")*

**public** Feign.Builder feignBuilder() {

**return** Feign.builder();

}

}

注意:在Spring Cloud Dalston版本之前，只要Hystrix 存在你项目的classpath里面，Feign默认会给所有的方法都加上熔断器(circuit breaker),这个默认的行为在Spring Cloud Dalston版本中变为可选择的方法

## 17.5 Feign Hystrix Fallbacks

Hystrix支持服务降级：当执行方法出错或者断点为开启状态时一段默认路径的代码将会被执行，可以通过在@FeignClient注解上设置fallback属性为对应的实现类上来开启回调功能，需要声明这个实现类为spring bean.

*@FeignClient(name = "hello", fallback = HystrixClientFallback.class)*

**protected** **interface** HystrixClient {

*@RequestMapping(method = RequestMethod.GET, value = "/hello")*

Hello iFailSometimes();

}

**static** **class** HystrixClientFallback **implements** HystrixClient {

*@Override*

**public** Hello iFailSometimes() {

**return** **new** Hello("fallback");

}

}

如果你想获取造成服务降级的原因，你可以使用@FeignClient注解的fallbackFactory属性：

*@FeignClient(name = "hello", fallbackFactory = HystrixClientFallbackFactory.class)*

**protected** **interface** HystrixClient {

*@RequestMapping(method = RequestMethod.GET, value = "/hello")*

Hello iFailSometimes();

}

*@Component*

**static** **class** HystrixClientFallbackFactory **implements** FallbackFactory<HystrixClient> {

*@Override*

**public** HystrixClient create(Throwable cause) {

**return** **new** HystrixClient() {

*@Override*

**public** Hello iFailSometimes() {

**return** **new** Hello("fallback; reason was: " + cause.getMessage());

}

};

}

}

注意: Feign fallbacks有一个限制。Fallbacks目前不支持返回 com.netflix.hystrix.HystrixCommand 和 rx.Observable 类的方法

## 17.6 Feign and @Primary

当使用Feign与Hystrix服务降级时，在同一类型的ApplicationContext中有多个bean。这将导致@Autowired无法起作用，因为不只有一个bean，或者有多个bean却没有标记其中一个为主。要解决这个问题，Spring Cloud Netflix将所有Feign实例标记为@Primary，所以Spring Framework将知道要注入哪个bean。在某些情况下并不需要。要关闭此行为，将@FeignClient的primary属性设置为false。

*@FeignClient(name = "hello", primary = false)*

**public** **interface** HelloClient {

*// methods here*

}