**1 SpringCloud 中 Feign 核心原理**

如果不了解 SpringCloud 中 Feign 核心原理，不会真正的了解 SpringCloud 的性能优化和配置优化，也就不可能做到真正掌握 SpringCloud。

本章从Feign 远程调用的重要组件开始，图文并茂的介绍 Feigh 远程调用的执行流程、Feign 本地 JDK Proxy 实例的创建流程，彻底的为大家解读 SpringCloud 的核心知识。使得广大的工程师不光做到知其然，更能知其所以然。

## ****1.1**** 简介：Feign远程调用的基本流程

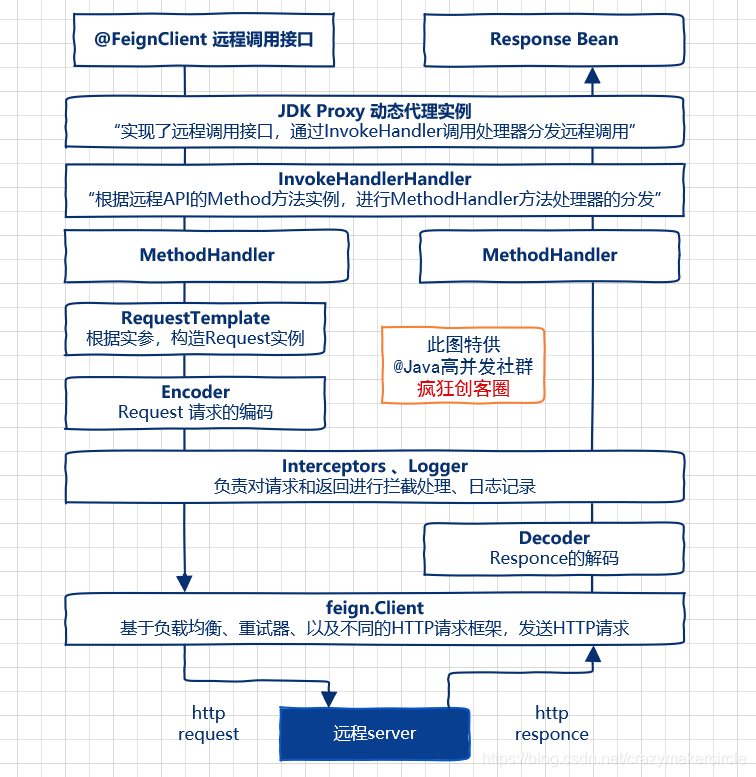
Feign远程调用，核心就是通过一系列的封装和处理，将以JAVA注解的方式定义的远程调用API接口，最终转换成HTTP的请求形式，然后将HTTP的请求的响应结果，解码成JAVA Bean，放回给调用者。Feign远程调用的基本流程，大致如下图所示。

图1 Feign远程调用的基本流程

从上图可以看到，Feign通过处理注解，将请求模板化，当实际调用的时候，传入参数，根据参数再应用到请求上，进而转化成真正的 Request 请求。通过Feign以及JAVA的动态代理机制，使得Java 开发人员，可以不用通过HTTP框架去封装HTTP请求报文的方式，完成远程服务的HTTP调用。

## ****1.2**** Feign 远程调用的重要组件

在微服务启动时，Feign会进行包扫描，对加@FeignClient注解的接口，按照注解的规则，创建远程接口的本地JDK Proxy代理实例。然后，将这些本地Proxy代理实例，注入到Spring IOC容器中。当远程接口的方法被调用，由Proxy代理实例去完成真正的远程访问，并且返回结果。

为了清晰的介绍SpringCloud中Feign运行机制和原理，在这里，首先为大家梳理一下Feign中几个重要组件。

### ****1.2.1**** 远程接口的本地JDK Proxy代理实例

**远程接口的本地JDK Proxy代理实例**，有以下特点：

（1）Proxy代理实例，实现了一个加 @FeignClient 注解的远程调用接口；

（2）Proxy代理实例，能在内部进行HTTP请求的封装，以及发送HTTP 请求；

（3）Proxy代理实例，能处理远程HTTP请求的响应，并且完成结果的解码，然后返回给调用者。

下面以一个简单的远程服务的调用接口 DemoClient 为例，具体介绍一下远程接口的本地JDK Proxy代理实例的创建过程。

DemoClient 接口，有两个非常简单的远程调用抽象方法：一个为hello() 抽象方法，用于完成远程URL “/api/demo/hello/v1”的HTTP请求；一个为 echo(…) 抽象方法，用于完成远程URL “/api/demo/echo/{word}/v1”的HTTP请求。具体如下图所示。

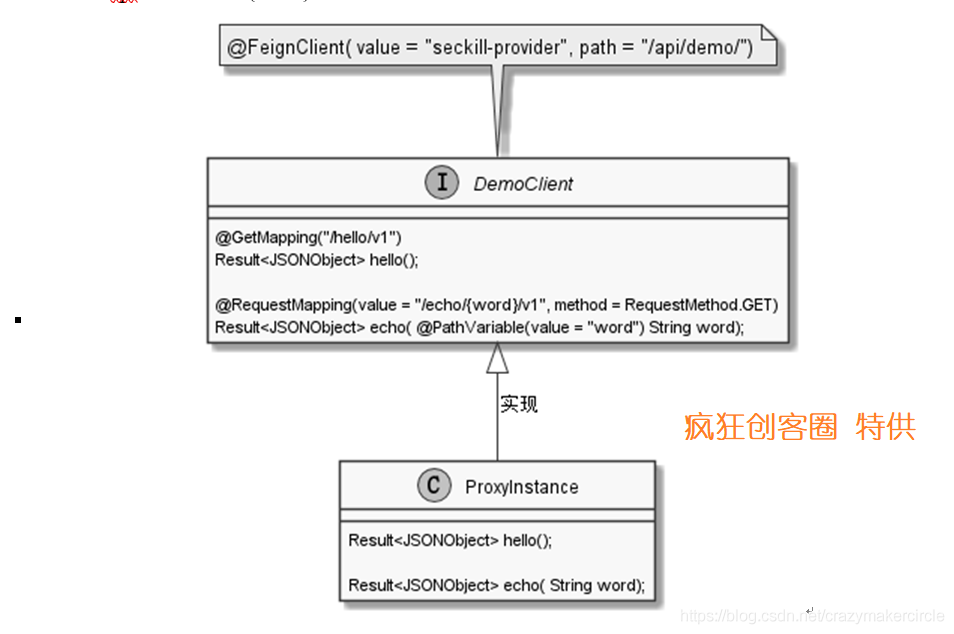


图2 远程接口的本地JDK Proxy代理实例示意图

DemoClient 接口代码如下：

package com.crazymaker.springcloud.demo.contract.client;

*//…省略import*

@FeignClient(

value = "seckill-provider", path = "/api/demo/",

fallback = DemoDefaultFallback.class)

public interface DemoClient {

*/\*\**

*\* 测试远程调用*

*\**

*\* @return hello*

*\*/*

@GetMapping("/hello/v1")

Result<JSONObject> hello();

*/\*\**

*\* 非常简单的一个 回显 接口，主要用于远程调用*

*\**

*\* @return echo 回显消息*

*\*/*

@RequestMapping(value = "/echo/{word}/v1", method = RequestMethod.GET)

Result<JSONObject> echo(

@PathVariable(value = "word") String word);

}

注意，上面的代码中，在DemoClient 接口上，加有@FeignClient 注解。也即是说，Feign在启动时，会为其创建一个本地JDK Proxy代理实例，并注册到Spring IOC容器。

如何使用呢？可以通过@Resource注解，按照类型匹配（这里的类型为DemoClient接口类型），从Spring IOC容器找到这个代理实例，并且装配给需要的成员变量。

DemoClient的 本地JDK Proxy 代理实例的使用的代码如下：

package com.crazymaker.springcloud.user.info.controller;

*//…省略import*

@Api(value = "用户信息、基础学习DEMO", tags = {"用户信息、基础学习DEMO"})

@RestController

@RequestMapping("/api/user")

public class UserController {

@Resource

DemoClient demoClient; *//装配 DemoClient 的本地代理实例*

@GetMapping("/say/hello/v1")

@ApiOperation(value = "测试远程调用速度")

public Result<JSONObject> hello() {

Result<JSONObject> result = demoClient.hello();

JSONObject data = new JSONObject();

data.put("others", result);

return Result.success(data).setMsg("操作成功");

}

*//…*

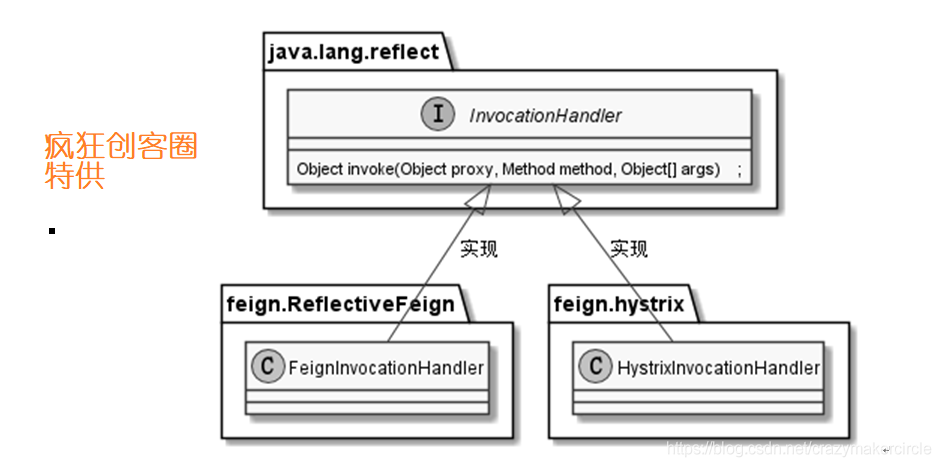
}

DemoClient的本地JDK Proxy代理实例的创建过程，比较复杂，稍后作为重点介绍。先来看另外两个重要的逻辑组件。

### ****1.2.2**** 调用处理器 InvocationHandler

大家知道，通过 JDK Proxy 生成动态代理类，核心步骤就是需要定制一个调用处理器，具体来说，就是实现JDK中位于java.lang.reflect 包中的 InvocationHandler 调用处理器接口，并且实现该接口的 invoke（…） 抽象方法。

为了创建Feign的远程接口的代理实现类，Feign提供了自己的一个默认的调用处理器，叫做 FeignInvocationHandler 类，该类处于 feign-core 核心jar包中。当然，调用处理器可以进行替换，如果Feign与Hystrix结合使用，则会替换成 HystrixInvocationHandler 调用处理器类，类处于 feign-hystrix 的jar包中。

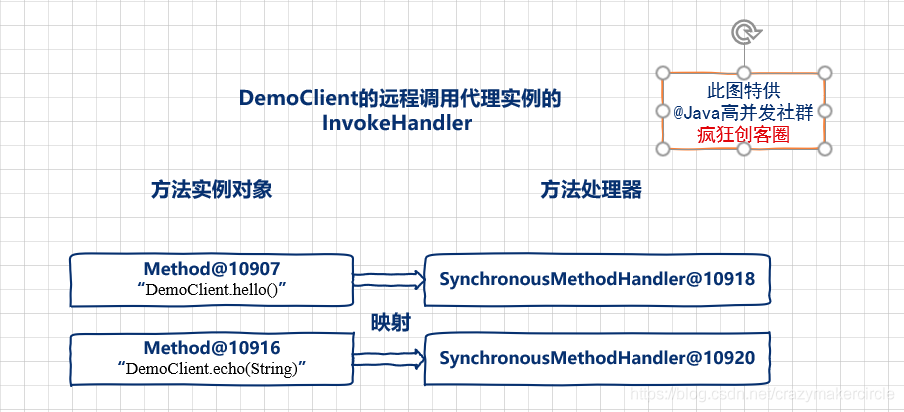


​ 图3 Feign中实现的 InvocationHandler 调用处理器

### ****1.2.1**** 默认的调用处理器 FeignInvocationHandler

默认的调用处理器 FeignInvocationHandler 是一个相对简单的类，有一个非常重要Map类型成员 dispatch 映射，保存着远程接口方法到MethodHandler方法处理器的映射。

以前面示例中DemoClient 接口为例，其代理实现类的调用处理器 FeignInvocationHandler 的dispatch 成员的内存结构图如图3所示。



**图4 DemoClient代理实例的调用处理器 FeignInvocationHandler的dispatch 成员**

为何在图3中的Map类型成员 dispatch 映射对象中，有两个Key-Value键值对呢？

原因是：默认的调用处理器 FeignInvocationHandle，在处理远程方法调用的时候，会根据Java反射的方法实例，在dispatch 映射对象中，找到对应的MethodHandler 方法处理器，然后交给MethodHandler 完成实际的HTTP请求和结果的处理。前面示例中的 DemoClient 远程调用接口，有两个远程调用方法，所以，其代理实现类的调用处理器 FeignInvocationHandler 的dispatch 成员，有两个有两个Key-Value键值对。

FeignInvocationHandler的关键源码，节选如下：

package feign;

*//...省略import*

public class ReflectiveFeign extends Feign {

*//...*

*//内部类：默认的Feign调用处理器 FeignInvocationHandler*

static class FeignInvocationHandler implements InvocationHandler {

private final Target target;

*//方法实例对象和方法处理器的映射*

private final Map<Method, MethodHandler> dispatch;

*//构造函数*

FeignInvocationHandler(Target target, Map<Method, MethodHandler> dispatch) {

this.target = checkNotNull(target, "target");

this.dispatch = checkNotNull(dispatch, "dispatch for %s", target);

}

*//默认Feign调用的处理*

@Override

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

*//...*

*//首先，根据方法实例，从方法实例对象和方法处理器的映射中，*

*//取得 方法处理器，然后，调用 方法处理器 的 invoke(...) 方法*

return dispatch.get(method).invoke(args);

}

*//...*

}

源码很简单，重点在于invoke(…)方法，虽然核心代码只有一行，但是其功能是复杂的：

（1）根据Java反射的方法实例，在dispatch 映射对象中，找到对应的MethodHandler 方法处理器；

（2）调用MethodHandler方法处理器的 invoke(...) 方法，完成实际的HTTP请求和结果的处理。

补充说明一下：MethodHandler 方法处理器，和JDK 动态代理机制中位于 java.lang.reflect 包的 InvocationHandler 调用处理器接口，没有任何的继承和实现关系。MethodHandler 仅仅是Feign自定义的，一个非常简单接口。

### ****1.2.2**** 方法处理器 MethodHandler

Feign的方法处理器 MethodHandler 是一个独立的接口，定义在 InvocationHandlerFactory 接口中，仅仅拥有一个invoke(…)方法，源码如下：

*//定义在InvocationHandlerFactory接口中*

public interface InvocationHandlerFactory {

*//…*

*//方法处理器接口，仅仅拥有一个invoke(…)方法*

interface MethodHandler {

*//完成远程URL请求*

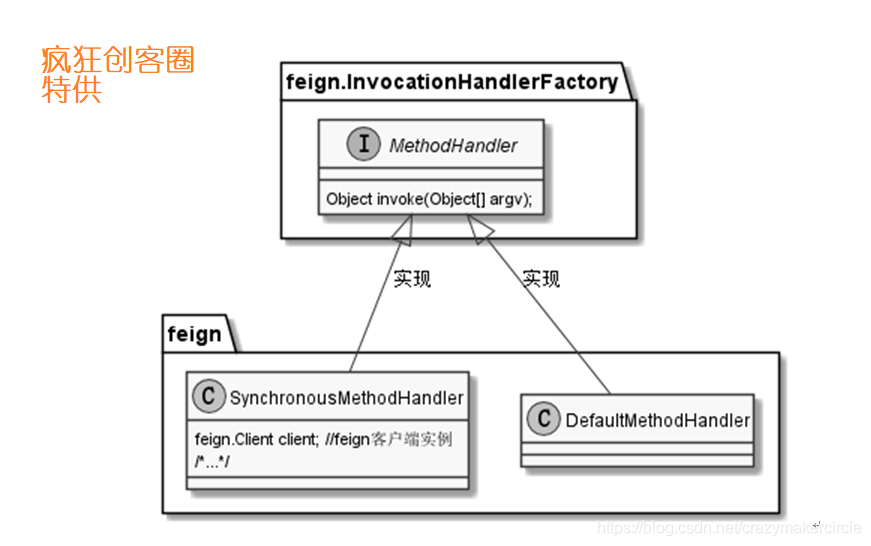
Object invoke(Object[] argv) throws Throwable;

}

*//...*

}

MethodHandler 的invoke(…)方法，主要职责是完成实际远程URL请求，然后返回解码后的远程URL的响应结果。Feign提供了默认的 SynchronousMethodHandler 实现类，提供了基本的远程URL的同步请求处理。有关 SynchronousMethodHandler类以及其与MethodHandler的关系，大致如图4所示。



​ **图5 Feign的MethodHandler方法处理器**

为了彻底了解方法处理器，来读一下 SynchronousMethodHandler 方法处理器的源码，大致如下：

package feign;

*//…..省略import*

final class SynchronousMethodHandler implements MethodHandler {

*//…*

*// 执行Handler 的处理*

public Object invoke(Object[] argv) throws Throwable {

RequestTemplate requestTemplate = this.buildTemplateFromArgs.create(argv);

Retryer retryer = this.retryer.clone();

while(true) {

try {

return this.executeAndDecode(requestTemplate);

} catch (RetryableException var5) {

*//…省略不相干代码*

}

}

}

*//执行请求，然后解码结果*

Object executeAndDecode(RequestTemplate template) throws Throwable {

Request request = this.targetRequest(template);

long start = System.nanoTime();

Response response;

try {

response = this.client.execute(request, this.options);

response.toBuilder().request(request).build();

}

}

}

SynchronousMethodHandler的invoke(…)方法，调用了自己的executeAndDecode(…) 请求执行和结果解码方法。该方法的工作步骤：

（1）首先通 RequestTemplate 请求模板实例，生成远程URL请求实例 request；

（2）然后用自己的 feign 客户端client成员，excecute(…) 执行请求，并且获取 response 响应；

（3）对response 响应进行结果解码。

### ****1.2.3**** Feign 客户端组件 feign.Client

客户端组件是Feign中一个非常重要的组件，负责端到端的执行URL请求。其核心的逻辑：发送request请求到服务器，并接收response响应后进行解码。

feign.Client 类，是代表客户端的顶层接口，只有一个抽象方法，源码如下：

package feign;

*/\*\*客户端接口*

*\* Submits HTTP {@link Request requests}.*

*Implementations are expected to be thread-safe.*

*\*/*

public interface Client {

*//提交HTTP请求，并且接收response响应后进行解码*

Response execute(Request request, Options options) throws IOException;

}

由于不同的feign.Client 实现类，内部完成HTTP请求的组件和技术不同，故，feign.Client 有多个不同的实现。这里举出几个例子：

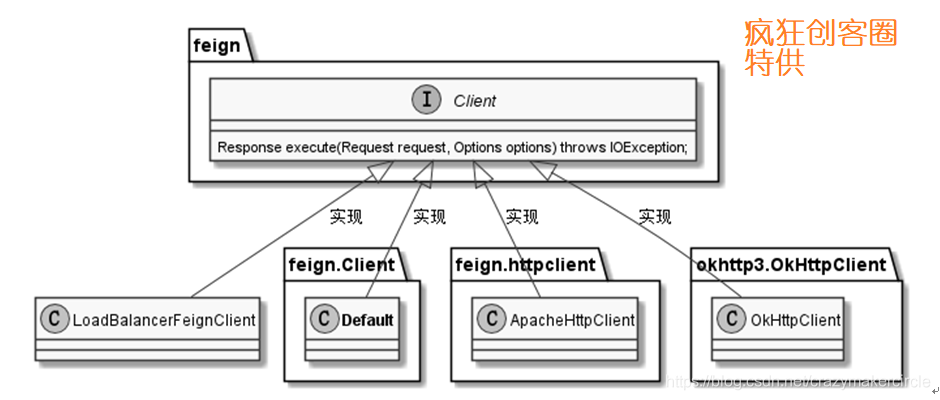
（1）Client.Default类：默认的feign.Client 客户端实现类，内部使用HttpURLConnnection 完成URL请求处理；

（2）ApacheHttpClient 类：内部使用 Apache httpclient 开源组件完成URL请求处理的feign.Client 客户端实现类；

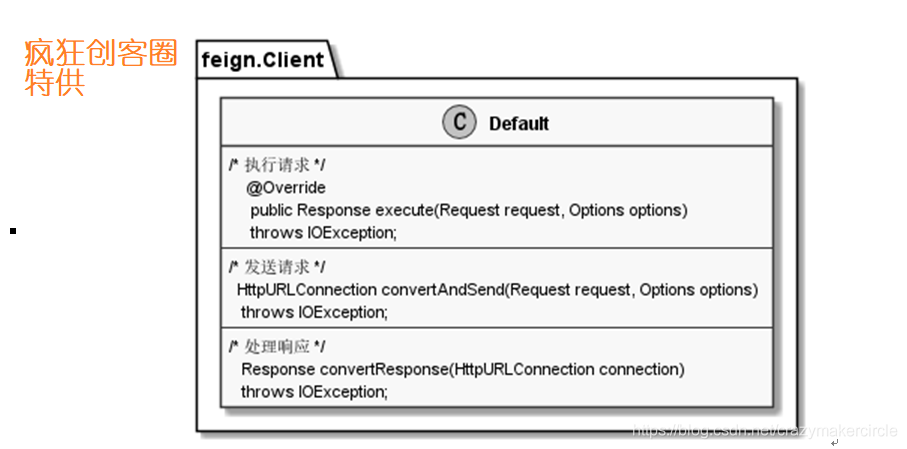
（3）OkHttpClient类：内部使用 OkHttp3 开源组件完成URL请求处理的feign.Client 客户端实现类。

（4）LoadBalancerFeignClient 类：内部使用 Ribben 负载均衡技术完成URL请求处理的feign.Client 客户端实现类。

此外，还有一些特殊场景使用的feign.Client客户端实现类，也可以定制自己的feign.Client实现类。下面对上面几个常见的客户端实现类，进行简要介绍。

图6 feign.Client客户端实现类

#### 一：Client.Default类：

作为默认的Client 接口的实现类，在Client.Default内部使用JDK自带的HttpURLConnnection类实现URL网络请求。图7 默认的Client 接口的客户端实现类

在JKD1.8中，虽然在HttpURLConnnection 底层，使用了非常简单的HTTP连接池技术，但是，其HTTP连接的复用能力，实际是非常弱的，性能当然也很低。具体的原因，参见后面的“SpringCloud与长连接的深入剖析”专题内容。

#### 二：ApacheHttpClient类

ApacheHttpClient 客户端类的内部，使用 Apache HttpClient开源组件完成URL请求的处理。

从代码开发的角度而言，Apache HttpClient相比传统JDK自带的URLConnection，增加了易用性和灵活性，它不仅使客户端发送Http请求变得容易，而且也方便开发人员测试接口。既提高了开发的效率，也方便提高代码的健壮性。

从性能的角度而言，Apache HttpClient带有连接池的功能，具备优秀的HTTP连接的复用能力。关于带有连接池Apache HttpClient的性能提升倍数，具体可以参见后面的对比试验。

ApacheHttpClient 类处于 feign-httpclient 的专门jar包中，如果使用，还需要通过Maven依赖或者其他的方式，倒入配套版本的专门jar包。

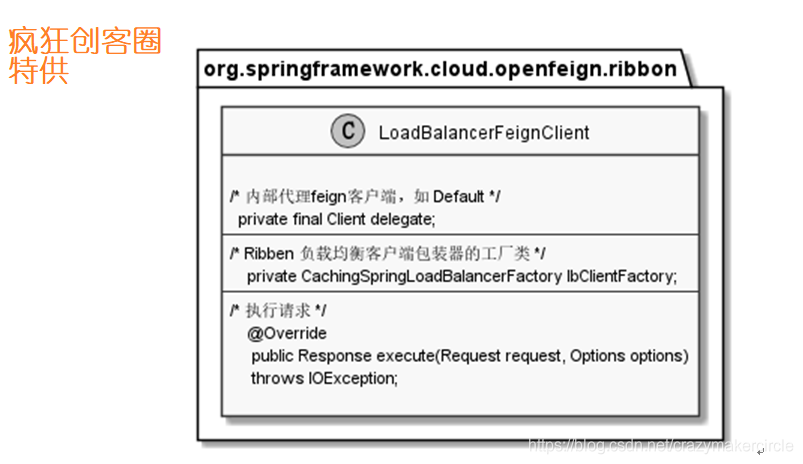
#### 三：OkHttpClient类

OkHttpClient 客户端类的内部，使用OkHttp3 开源组件完成URL请求处理。OkHttp3 开源组件由Square公司开发，用于替代HttpUrlConnection和Apache HttpClient。由于OkHttp3较好的支持 SPDY协议（SPDY是Google开发的基于TCP的传输层协议，用以最小化网络延迟，提升网络速度，优化用户的网络使用体验。），从Android4.4开始，google已经开始将Android源码中的 HttpURLConnection 请求类使用OkHttp进行了替换。也就是说，对于Android 移动端APP开发来说，OkHttp3 组件，是基础的开发组件之一。

#### 四：LoadBalancerFeignClient 类

LoadBalancerFeignClient 内部使用了 Ribben 客户端负载均衡技术完成URL请求处理。在原理上，简单的使用了delegate包装代理模式：Ribben负载均衡组件计算出合适的服务端server之后，由内部包装 delegate 代理客户端完成到服务端server的HTTP请求；所封装的 delegate 客户端代理实例的类型，可以是 Client.Default 默认客户端，也可以是 ApacheHttpClient 客户端类或OkHttpClient 高性能客户端类，还可以其他的定制的feign.Client 客户端实现类型。

LoadBalancerFeignClient 负载均衡客户端实现类，具体如下图所示。

图8 LoadBalancerFeignClient 负载均衡客户端实现类

## ****1.1**** Feigh 远程调用的执行流程

由于Feign远程调用接口的JDK Proxy实例的InvokeHandler调用处理器有多种，导致Feign远程调用的执行流程，也稍微有所区别，但是远程调用执行流程的主要步骤，是一致的。这里主要介绍两类JDK Proxy实例的InvokeHandler调用处理器相关的远程调用执行流程：

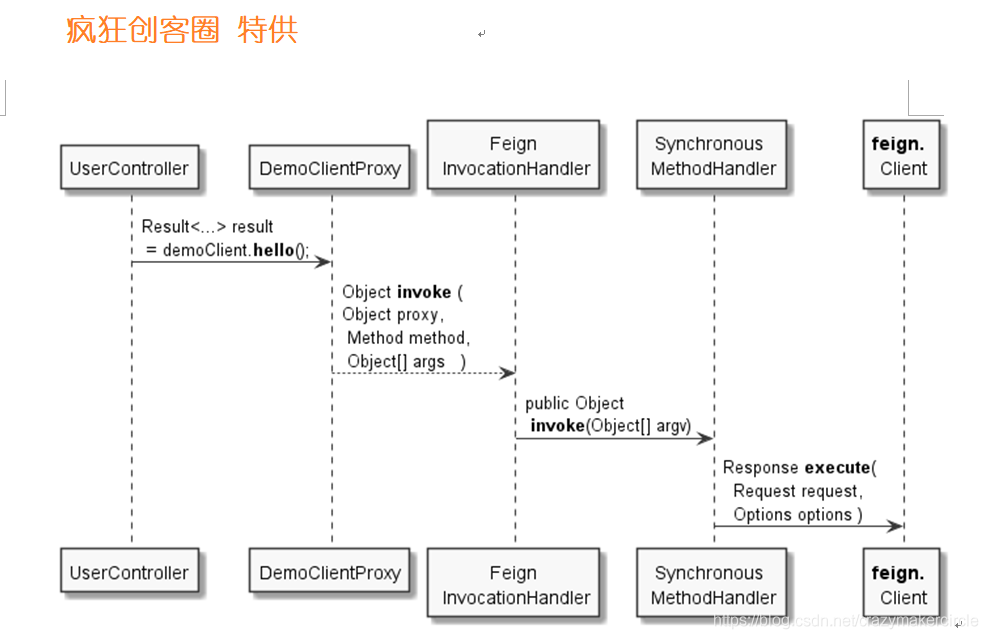
（1）与 默认的调用处理器 FeignInvocationHandler 相关的远程调用执行流程；

（2）与 Hystrix调用处理器 HystrixInvocationHandler 相关的远程调用执行流程。

介绍过程中，还是以前面的DemoClient的JDK Proxy远程动态代理实例的执行过程为例，演示分析Feigh远程调用的执行流程。

### ****1.1.1**** 与 FeignInvocationHandler 相关的远程调用执行流程

FeignInvocationHandler是默认的调用处理器，如果不对Feign做特殊的配置，则Feign将使用此调用处理器。结合前面的DemoClient的JDK Proxy远程动态代理实例的hello()远程调用执行过程，在这里，详细的介绍一下与 FeignInvocationHandler 相关的远程调用执行流程，大致如下图所示。

**图6 与 FeignInvocationHandler 相关的远程调用执行流程**

整体的远程调用执行流程，大致分为4步，具体如下：

**第1步：通过Spring IOC 容器实例，装配代理实例，然后进行远程调用。**

前文讲到，Feign在启动时，会为加上了@FeignClient注解的所有远程接口（包括 DemoClient 接口），创建一个本地JDK Proxy代理实例，并注册到Spring IOC容器。在这里，暂且将这个Proxy代理实例，叫做 DemoClientProxy，稍后，会详细介绍这个Proxy代理实例的具体创建过程。

然后，在本实例的UserController 调用代码中，通过@Resource注解，按照类型或者名称进行匹配（这里的类型为DemoClient接口类型），从Spring IOC容器找到这个代理实例，并且装配给@Resource注解所在的成员变量，本实例的成员变量的名称为 demoClient。

在需要代进行hello（）远程调用时，直接通过 demoClient 成员变量，调用JDK Proxy动态代理实例的hello（）方法。

**第2步：执行 InvokeHandler 调用处理器的invoke(…)方法**

前面讲到，JDK Proxy动态代理实例的真正的方法调用过程，具体是通过 InvokeHandler 调用处理器完成的。故，这里的DemoClientProxy代理实例，会调用到默认的FeignInvocationHandler 调用处理器实例的invoke(…)方法。

通过前面 FeignInvocationHandler 调用处理器的详细介绍，大家已经知道，默认的调用处理器 FeignInvocationHandle，内部保持了一个远程调用方法实例和方法处理器的一个Key-Value键值对Map映射。FeignInvocationHandle 在其invoke(…)方法中，会根据Java反射的方法实例，在dispatch 映射对象中，找到对应的 MethodHandler 方法处理器，然后由后者完成实际的HTTP请求和结果的处理。

所以在第2步中，FeignInvocationHandle 会从自己的 dispatch映射中，找到hello()方法所对应的MethodHandler 方法处理器，然后调用其 invoke(…)方法。

**第3步：执行 MethodHandler 方法处理器的invoke(…)方法**

通过前面关于 MethodHandler 方法处理器的非常详细的组件介绍，大家都知道，feign默认的方法处理器为 SynchronousMethodHandler，其invoke(…)方法主要是通过内部成员feign客户端成员 client，完成远程 URL 请求执行和获取远程结果。

feign.Client 客户端有多种类型，不同的类型，完成URL请求处理的具体方式不同。

**第4步：通过 feign.Client 客户端成员，完成远程 URL 请求执行和获取远程结果**

如果MethodHandler方法处理器实例中的client客户端，是默认的 feign.Client.Default 实现类性，则使用JDK自带的HttpURLConnnection类，完成远程 URL 请求执行和获取远程结果。

如果MethodHandler方法处理器实例中的client客户端，是 ApacheHttpClient 客户端实现类性，则使用 Apache httpclient 开源组件，完成远程 URL 请求执行和获取远程结果。

通过以上四步，应该可以清晰的了解到了 SpringCloud中的 feign 远程调用执行流程和运行机制。

实际上，为了简明扼要的介绍清楚默认的调用流程，上面的流程，实际上省略了一个步骤：第3步，实际可以分为两小步。为啥呢？ SynchronousMethodHandler 并不是直接完成远程URL的请求，而是通过负载均衡机制，定位到合适的远程server 服务器，然后再完成真正的远程URL请求。换句话说，SynchronousMethodHandler实例的client成员，其实际不是feign.Client.Default类型，而是 LoadBalancerFeignClient 客户端负载均衡类型。 因此，上面的第3步，如果进一步细分话，大致如下：（1）首先通过 SynchronousMethodHandler 内部的client实例，实质为负责客户端负载均衡 LoadBalancerFeignClient 实例，首先查找到远程的 server 服务端；（2） 然后再由LoadBalancerFeignClient 实例内部包装的feign.Client.Default 内部类实例，去请求server端服务器，完成URL请求处理。

最后，说明下，默认的与 FeignInvocationHandler 相关的远程调用执行流程，在运行机制以及调用性能上，满足不了生产环境的要求，为啥呢？ 大致原因有以下两点：

（1） 没有远程调用过程中的熔断监测和恢复机制；

（2） 也没有用到高性能的HTTP连接池技术。

接下来，将为大家介绍一下用到熔断监测和恢复机制 Hystrix 技术的远程调用执行流程，该流程中，远程接口的JDK Proxy动态代理实例所使用的调用处理器，叫做 HystrixInvocationHandler 调用处理器。