课时5

声明式服务调用-Feign

1. Feign基础知识
2. Feign的使用
3. Feign的最佳使用技巧
4. Feign源码分析

常用的API调用方式

HttpURLConnection

HttpClient

Okhttp

RestTemplate

Feign简介

Feign

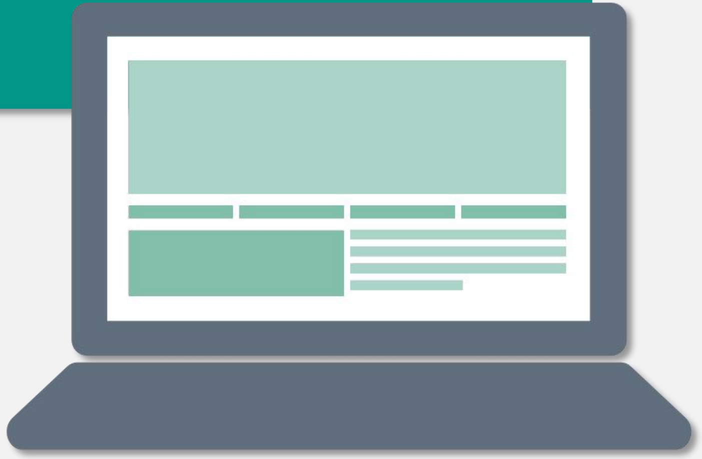
是一个声明式的REST客户端，它的目的就是让REST调用更加简单

Feign提供了HTTP请求的模板，通过编写简单的接口和插入注解，就可以定义好HTTP请求的参数、格式、地址等信息

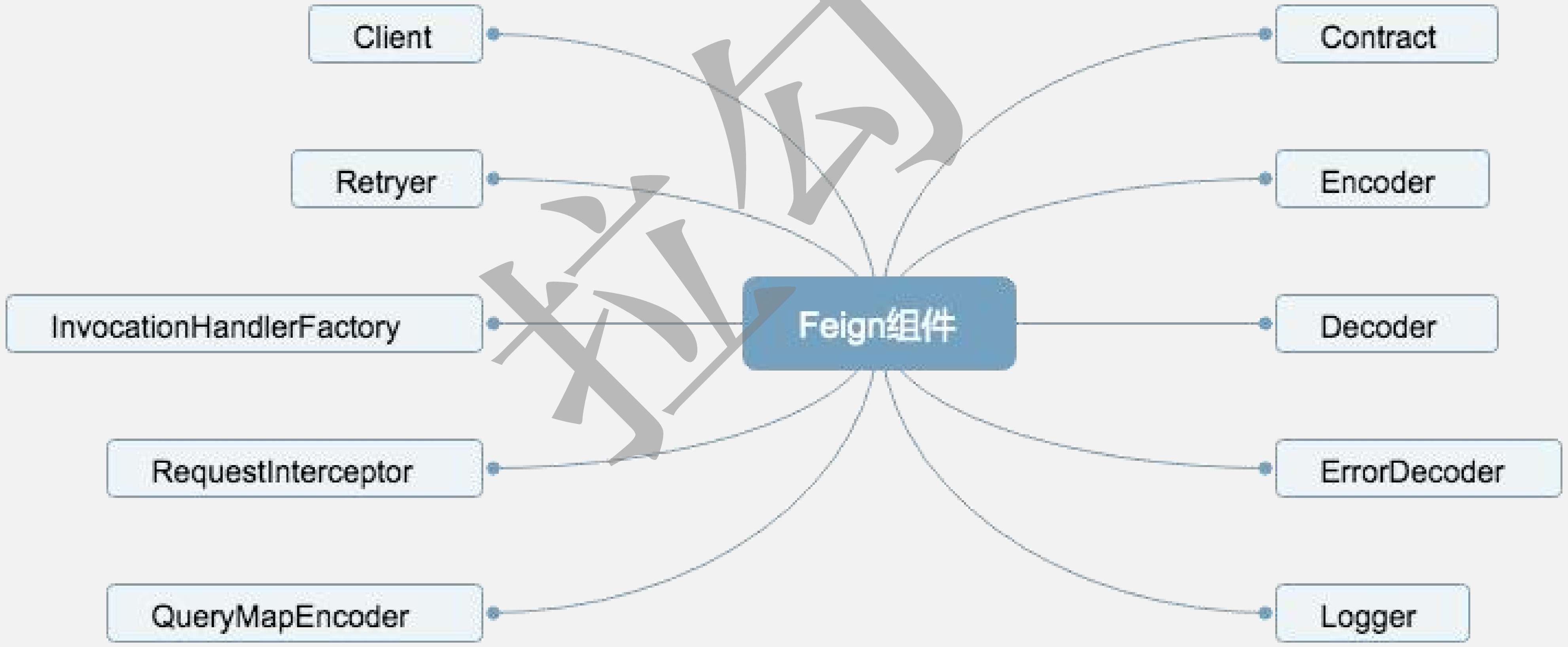
而Feign则会完全代理HTTP请求，我们只需要像调用方法一样调用它就可以完成服务请求及相关处理

Spring Cloud对Feign进行了封装，使其支持SpringMVC标准注解和HttpMessageConverters

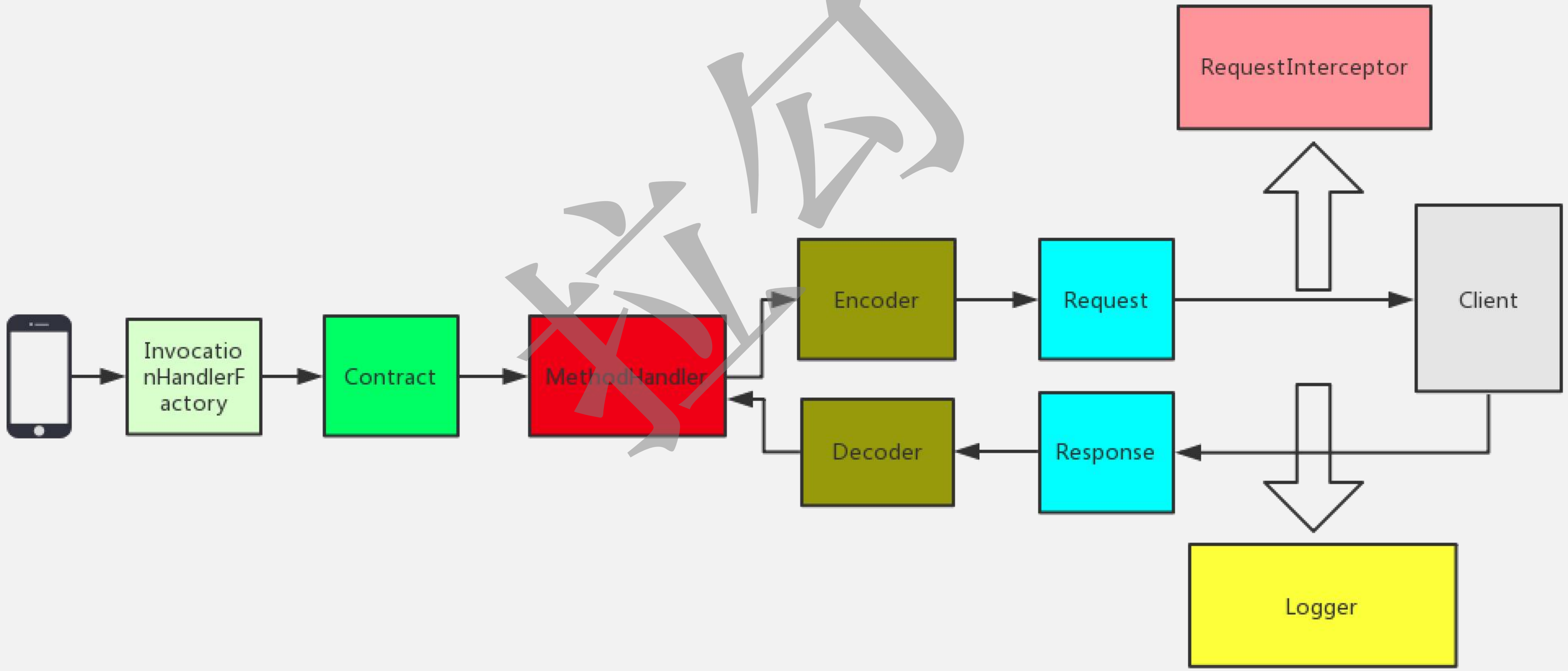
Feign可以与Eureka和Ribbon组合使用以支持负载均衡



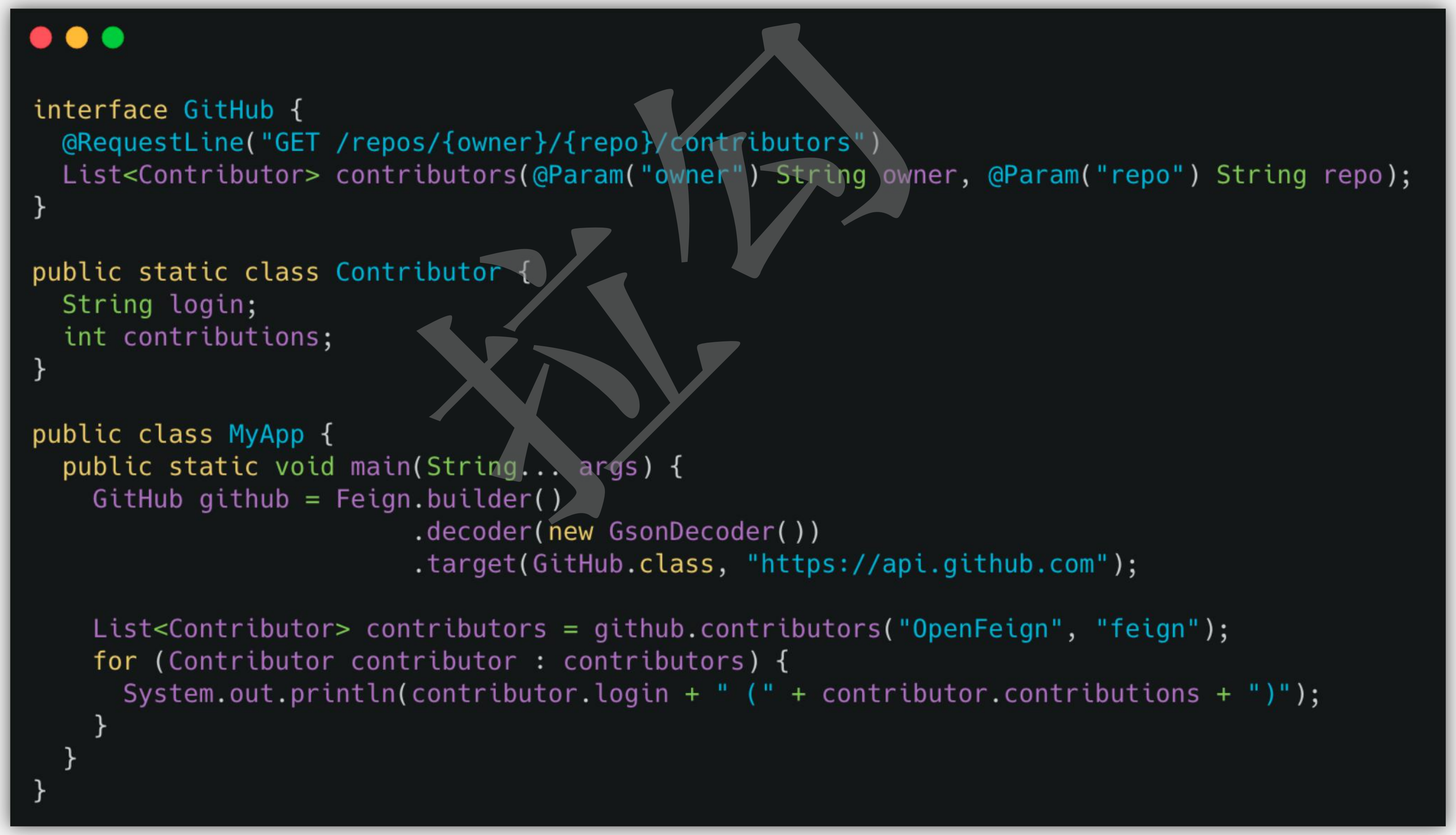
Feign中的重要组件



Feign执行过程


InvocationHandlerFactory
Contract
MethodHandler
Encoder
Request
RequestInterceptor
Client
Decoder
Response
Logger

Feign的使用-原生API



```
interface GitHub {
  @RequestLine("GET /repos/{owner}/{repo}/contributors")
  List<Contributor> contributors(@Param("owner") String owner, @Param("repo") String repo);
}

public static class Contributor {
  String login;
  int contributions;
}

public class MyApp {
  public static void main(String... args) {
    GitHub github = Feign.builder()
                         .decoder(new GsonDecoder())
                         .target(GitHub.class, "https://api.github.com");

    List<Contributor> contributors = github.contributors("OpenFeign", "feign");
    for (Contributor contributor : contributors) {
      System.out.println(contributor.login + " (" + contributor.contributions + ")");
    }
  }
}
```

Feign自带注解

注解	作用
@RequestLine	定义请求类型和URI
@Param	参数映射
@Headers	请求头的映射
@QueryMap	多参数封装成实体类接收
@HeaderMap	Http Header映射成Map
@Body	定义一个数据模板，通过@Param解析对应的表达式

Feign的使用-Spring Cloud

1. 启动类加@EnableFeignClients启用Feign

2. @FeignClient定义客户端

3. 通过客户端访问接口

Feign整合Hystrix

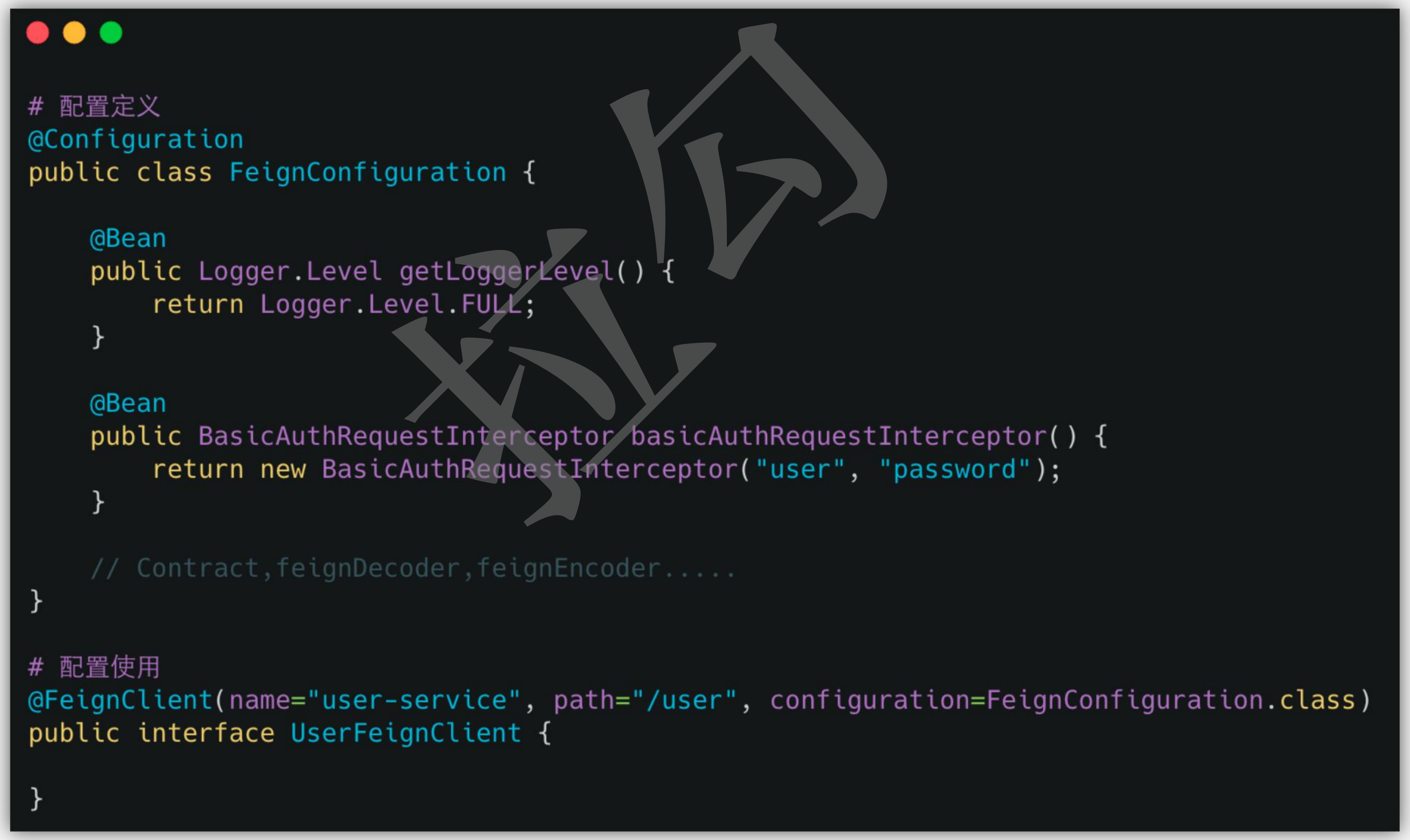
增加依赖

spring-cloud-starter-netflix-hystrix

开启Feign中的Hystrix

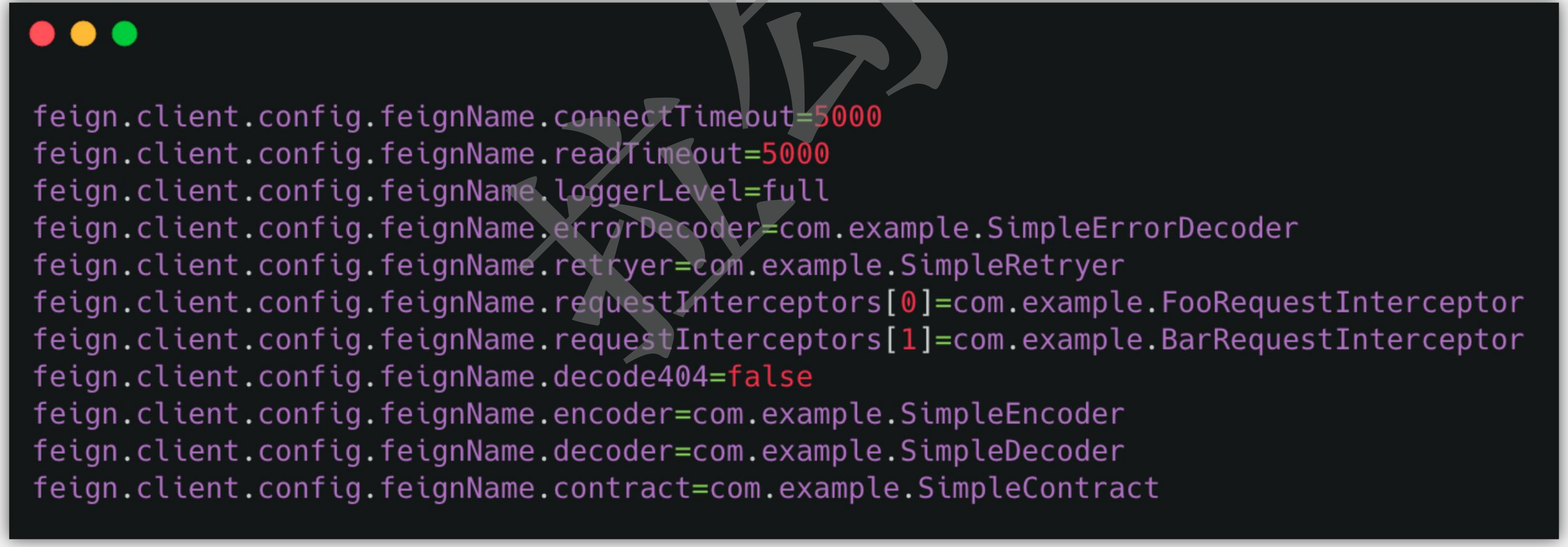
feign.hystrix.enabled=true

Feign的配置-代码方式



```
# 配置定义
@Configuration
public class FeignConfiguration {

    @Bean
    public Logger.Level getLoggerLevel() {
        return Logger.Level.FULL;
    }

    @Bean
    public BasicAuthRequestInterceptor basicAuthRequestInterceptor() {
        return new BasicAuthRequestInterceptor("user", "password");
    }

    // Contract,feignDecoder,feignEncoder.....
}

# 配置使用
@FeignClient(name="user-service", path="/user", configuration=FeignConfiguration.class)
public interface UserFeignClient {

}
```

Feign的配置-配置文件方式



```
feign.client.config.feignName.connectTimeout=5000
feign.client.config.feignName.readTimeout=5000
feign.client.config.feignName.loggerLevel=full
feign.client.config.feignName.errorDecoder=com.example.SimpleErrorDecoder
feign.client.config.feignName.retryer=com.example.SimpleRetryer
feign.client.config.feignName.requestInterceptors[0]=com.example.FooRequestInterceptor
feign.client.config.feignName.requestInterceptors[1]=com.example.BarRequestInterceptor
feign.client.config.feignName.decode404=false
feign.client.config.feignName.encoder=com.example.SimpleEncoder
feign.client.config.feignName.decoder=com.example.SimpleDecoder
feign.client.config.feignName.contract=com.example.SimpleContract
```

Feign的最佳使用技巧

继承特性

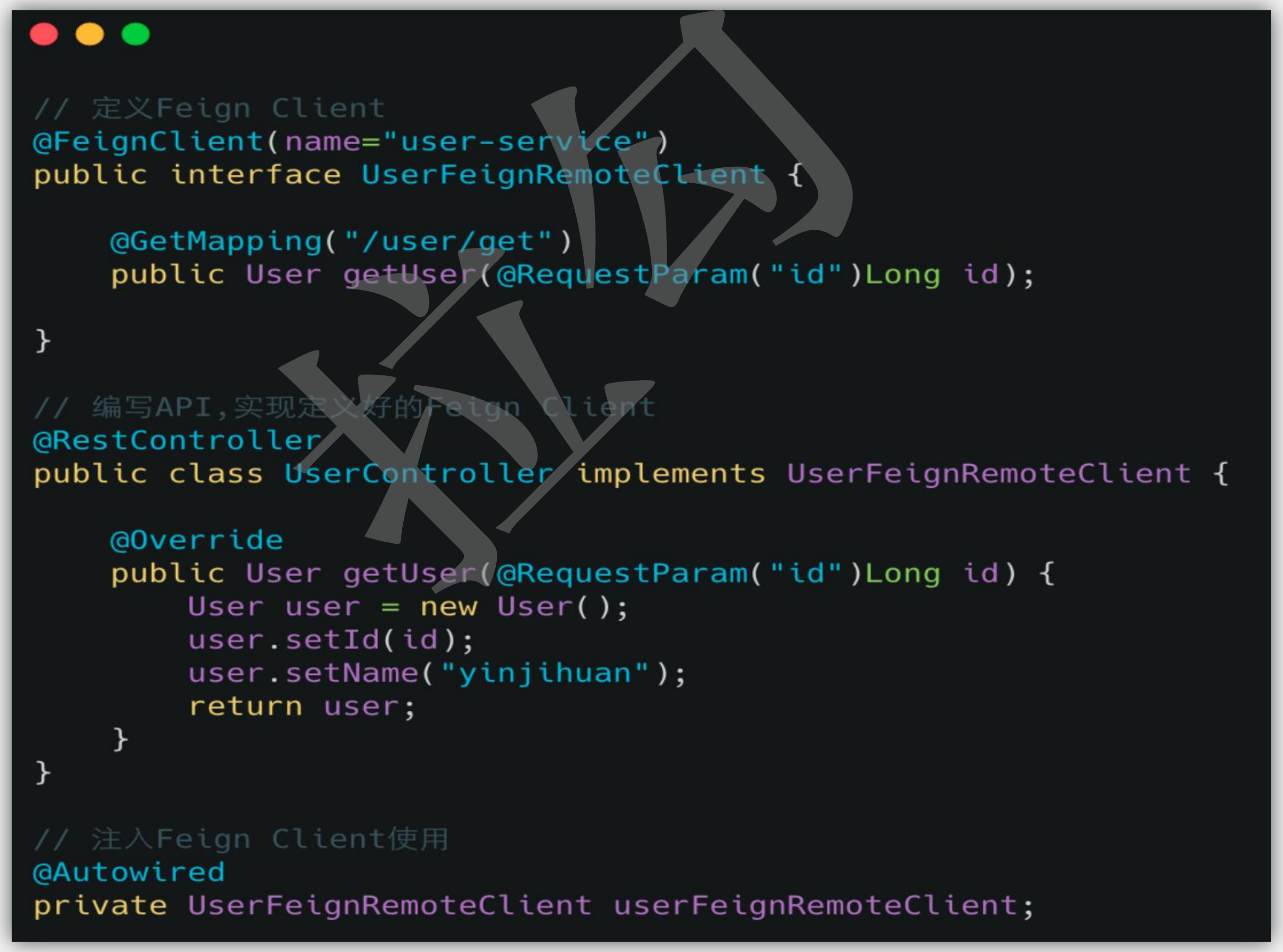
拦截器

GET请求多参数传递

日志配置

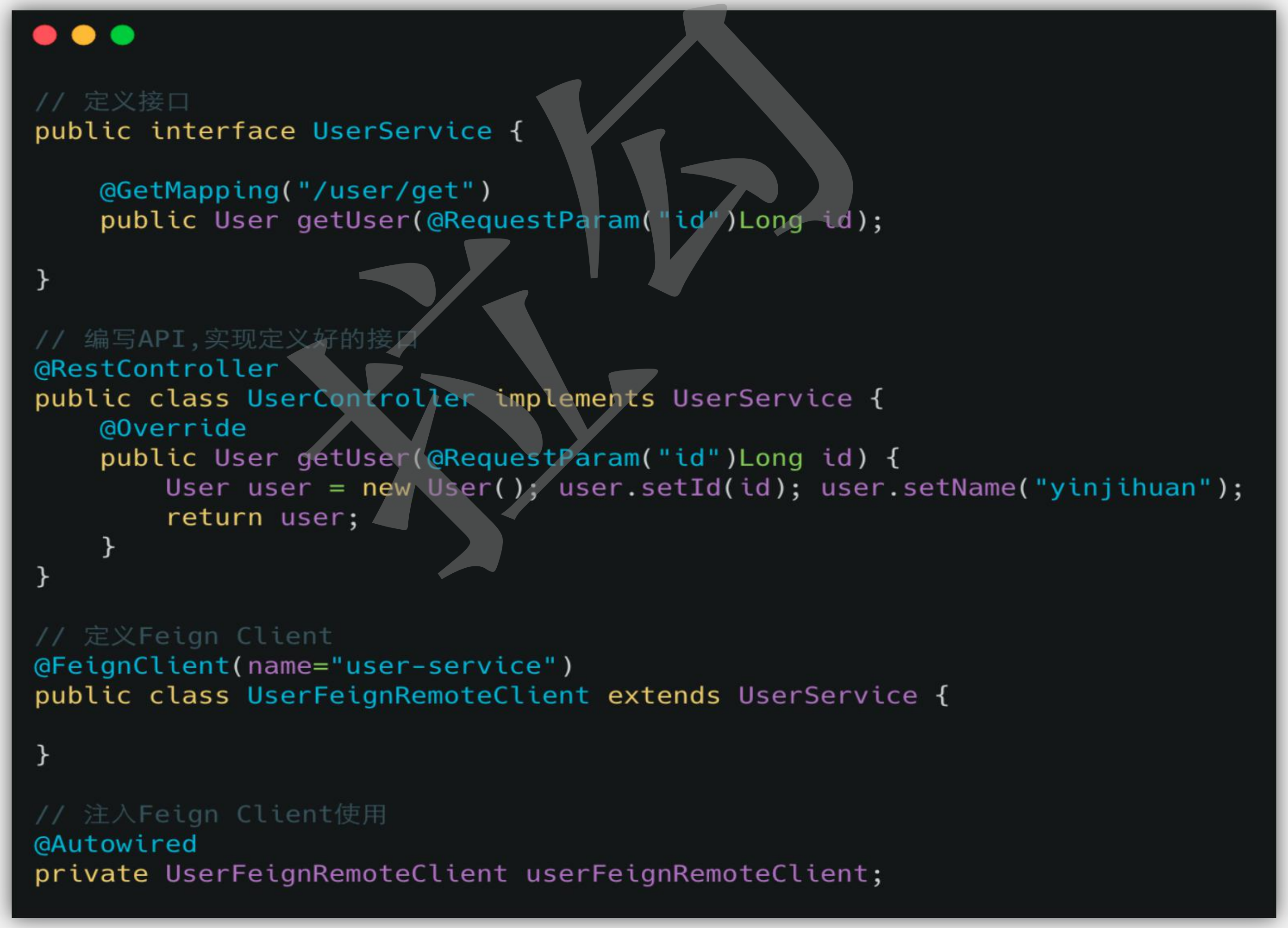
异常传递

继承特性-方式一



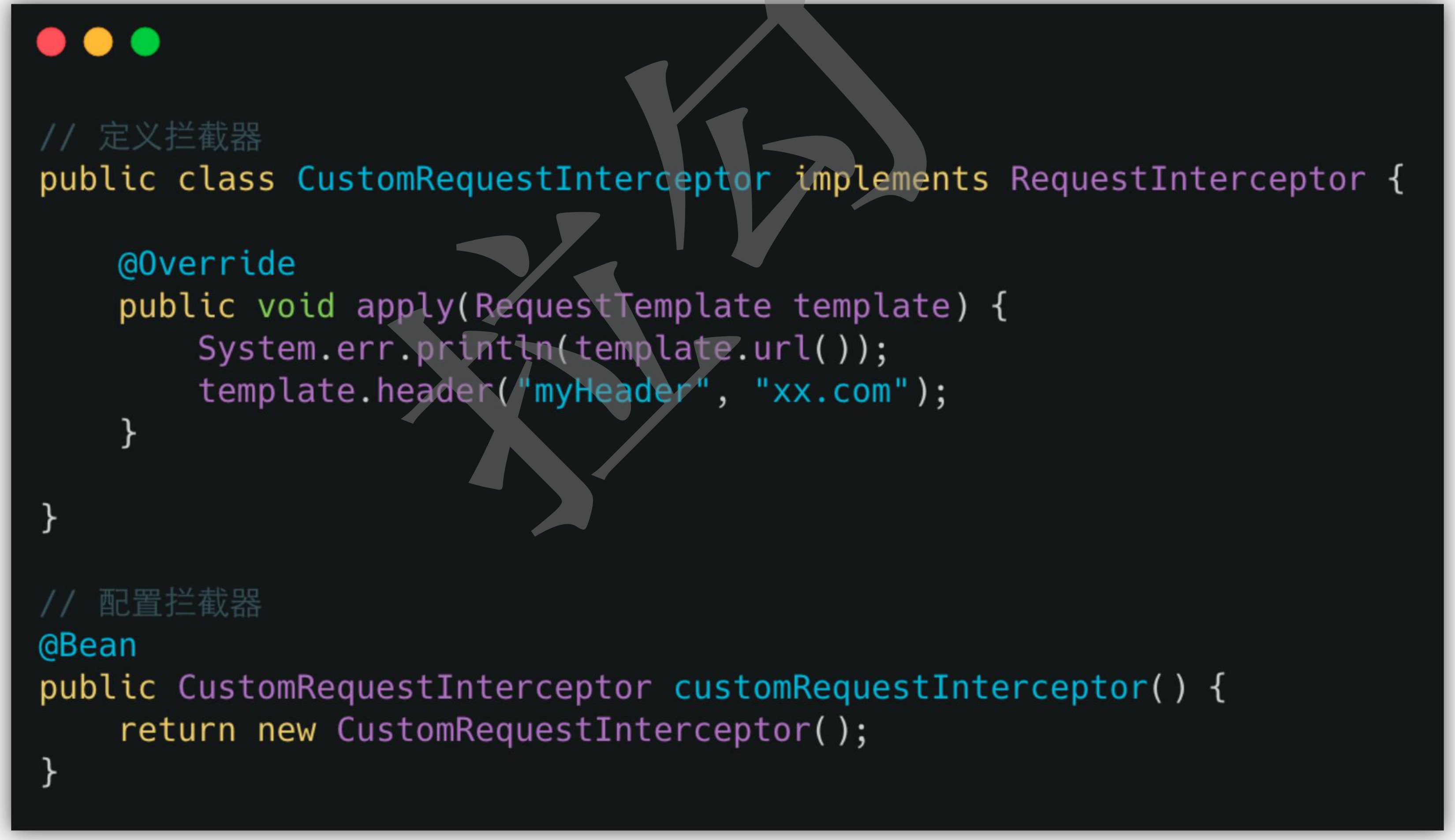
```
// 定义Feign Client
@FeignClient(name="user-service")
public interface UserFeignRemoteClient {

    @GetMapping("/user/get")
    public User getUser(@RequestParam("id")Long id);

}

// 编写API,实现定义好的Feign Client
@RestController
public class UserController implements UserFeignRemoteClient {

    @Override
    public User getUser(@RequestParam("id")Long id) {
        User user = new User();
        user.setId(id);
        user.setName("yinjihuan");
        return user;
    }
}

// 注入Feign Client使用
@Autowired
private UserFeignRemoteClient userFeignRemoteClient;
```

继承特性-方式二



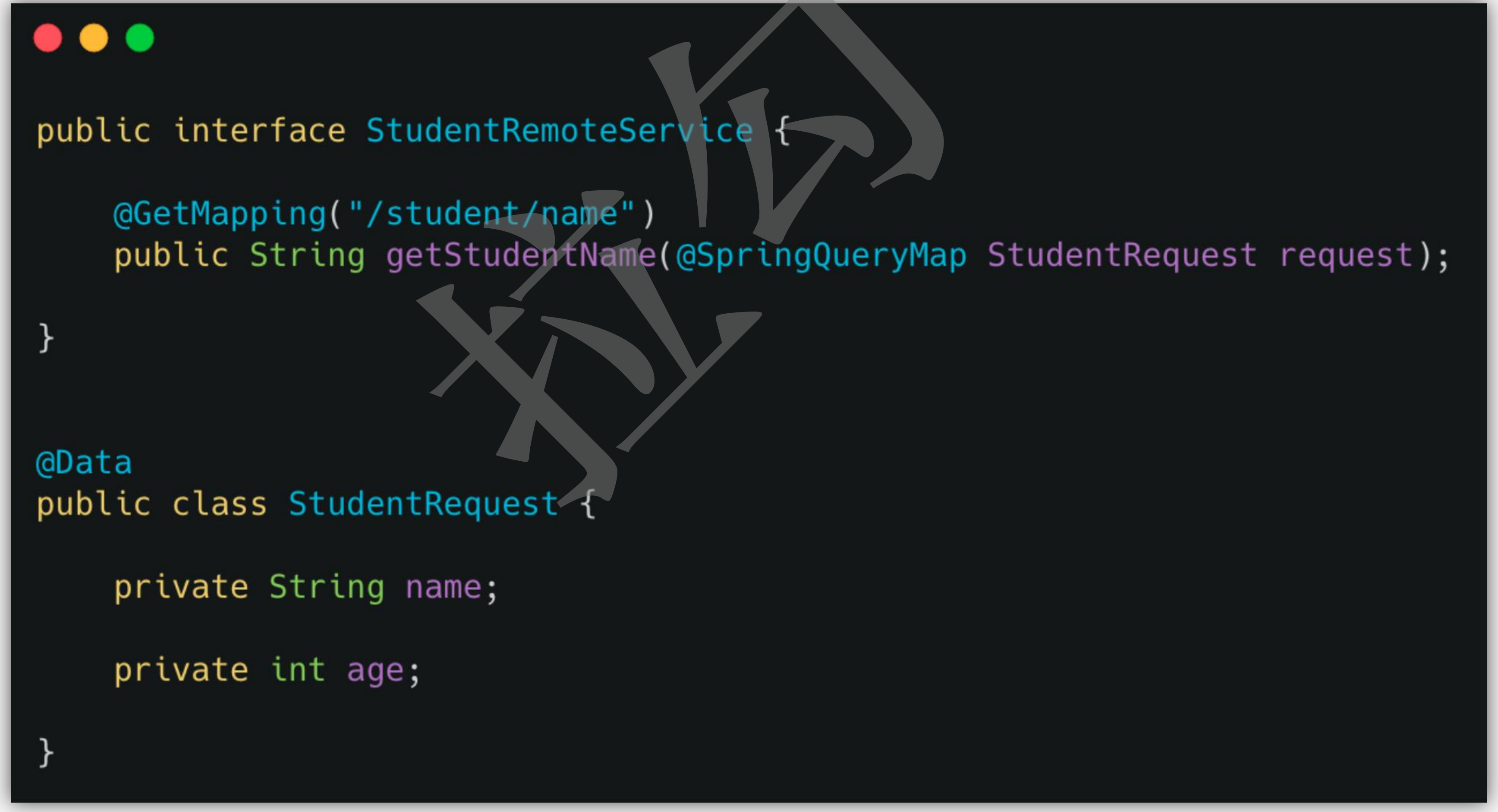
```
// 定义接口
public interface UserService {

    @GetMapping("/user/get")
    public User getUser(@RequestParam("id")Long id);

}

// 编写API,实现定义好的接口
@RestController
public class UserController implements UserService {
    @Override
    public User getUser(@RequestParam("id")Long id) {
        User user = new User(); user.setId(id); user.setName("yinjihuan");
        return user;
    }
}

// 定义Feign Client
@FeignClient(name="user-service")
public class UserFeignRemoteClient extends UserService {

}

// 注入Feign Client使用
@Autowired
private UserFeignRemoteClient userFeignRemoteClient;
```

拦截器



```java
// 定义拦截器
public class CustomRequestInterceptor implements RequestInterceptor {

    @Override
    public void apply(RequestTemplate template) {
        System.err.println(template.url());
        template.header("myHeader", "xx.com");
    }

}

// 配置拦截器
@Bean
public CustomRequestInterceptor customRequestInterceptor() {
    return new CustomRequestInterceptor();
}
```

GET请求多参数传递



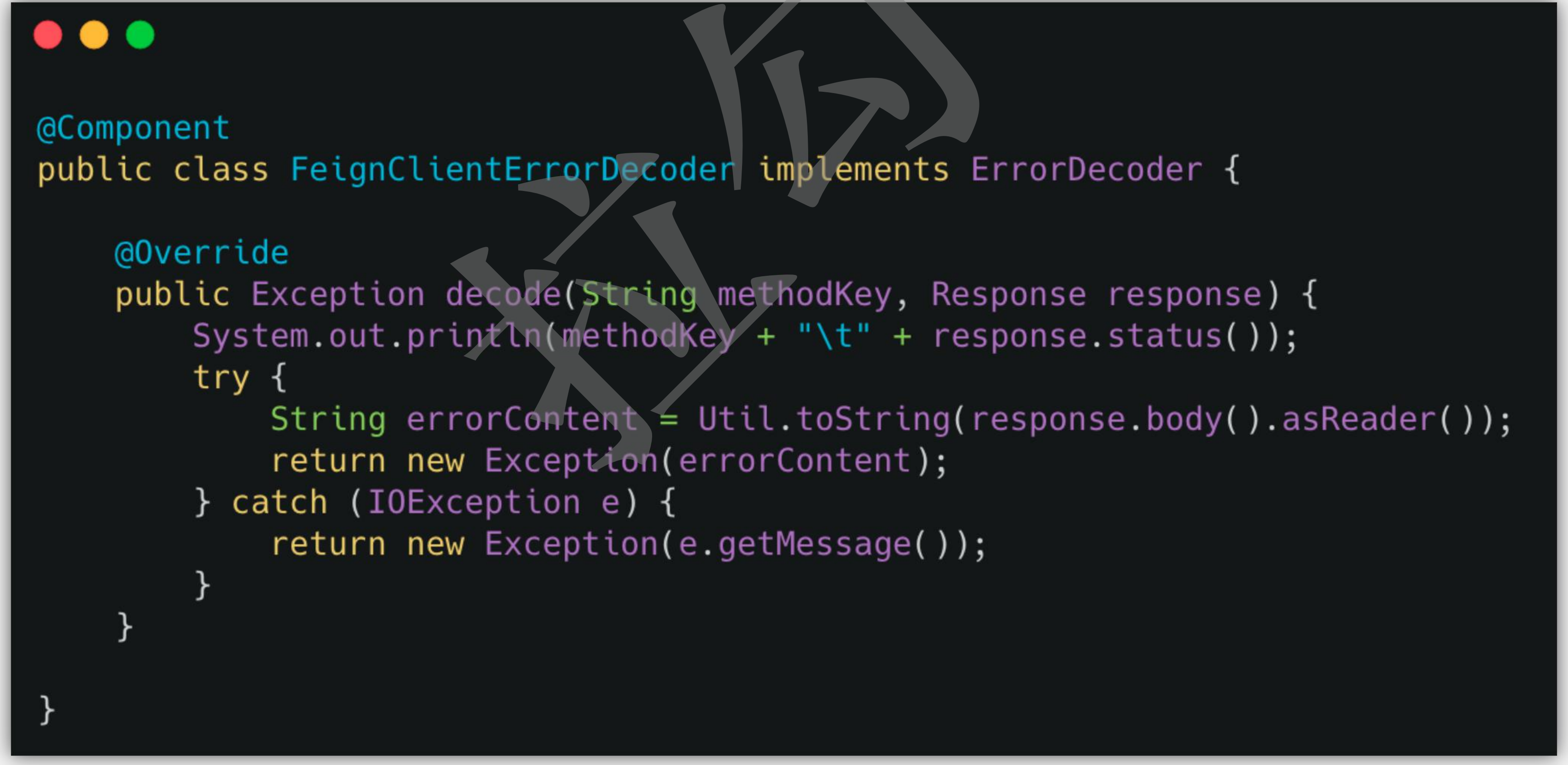
```
public interface StudentRemoteService {

    @GetMapping("/student/name")
    public String getStudentName(@SpringQueryMap StudentRequest request);

}


@Data
public class StudentRequest {

    private String name;

    private int age;

}
```

日志配置

- # 配置Feign Client名称为user-service的Feign日志级别为Full
- feign.client.config.user-service.loggerLevel=full
- # 指定日志级别为DEBUG
- logging.level.com.example.feign.client.UserRemoteClient=DEBUG

- - NONE
- 不输出日志
- - BASIC
- 只输出请求的方法的URL和响应的状态码和执行的时间
- - HEADERS
- 将BASIC信息和请求头信息输出
- - FULL
- 输出全部完整的请求信息

异常解码器


```java
@Component
public class FeignClientErrorDecoder implements ErrorDecoder {

    @Override
    public Exception decode(String methodKey, Response response) {
        System.out.println(methodKey + "\t" + response.status());
        try {
            String errorContent = Util.toString(response.body().asReader());
            return new Exception(errorContent);
        } catch (IOException e) {
            return new Exception(e.getMessage());
        }
    }

}
```

Next：课时6《API 网关服务-Zuul》