[**#**](https://pig4cloud.com/#%E7%9B%AE%E7%9A%84) **目的**

先来看默认的feign service 是要求怎么做的。feign service 定义一个 factory 和 fallback 的类

@FeignClient(value = ServiceNameConstants.UMPS\_SERVICE, fallbackFactory = RemoteLogServiceFallbackFactory.class)

public interface RemoteLogService {}

在Spring Cloud 使用feign 的时候，需要明确指定fallback 策略，不然会提示错误。 但是我们大多数情况的feign 降级策略为了保证幂等都会很简单，输出错误日志即可。 类似如下代码，在企业中开发非常不方便

@Slf4j

@Component

public class RemoteLogServiceFallbackImpl implements RemoteLogService {

@Setter

private Throwable cause;

@Override

public R<Boolean> saveLog(SysLog sysLog, String from) {

log.error("feign 插入日志失败", cause);

return null;

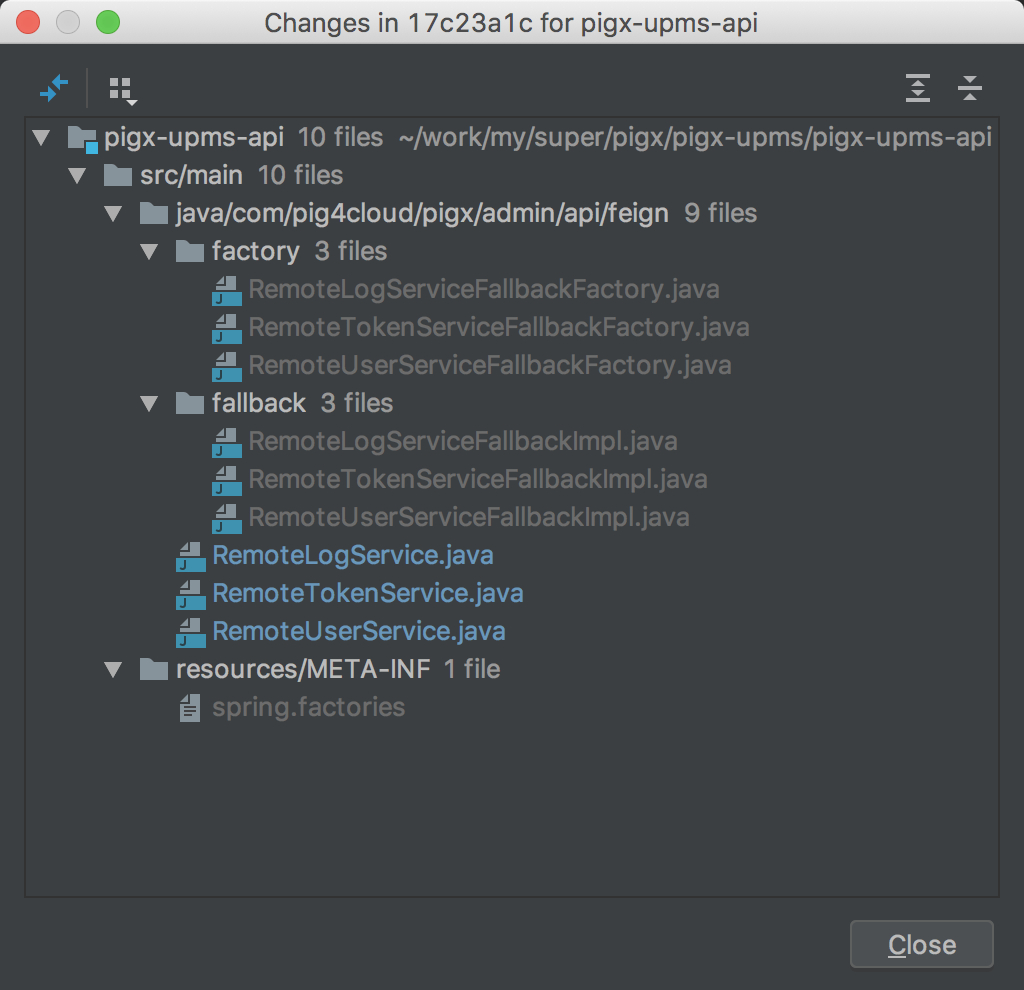
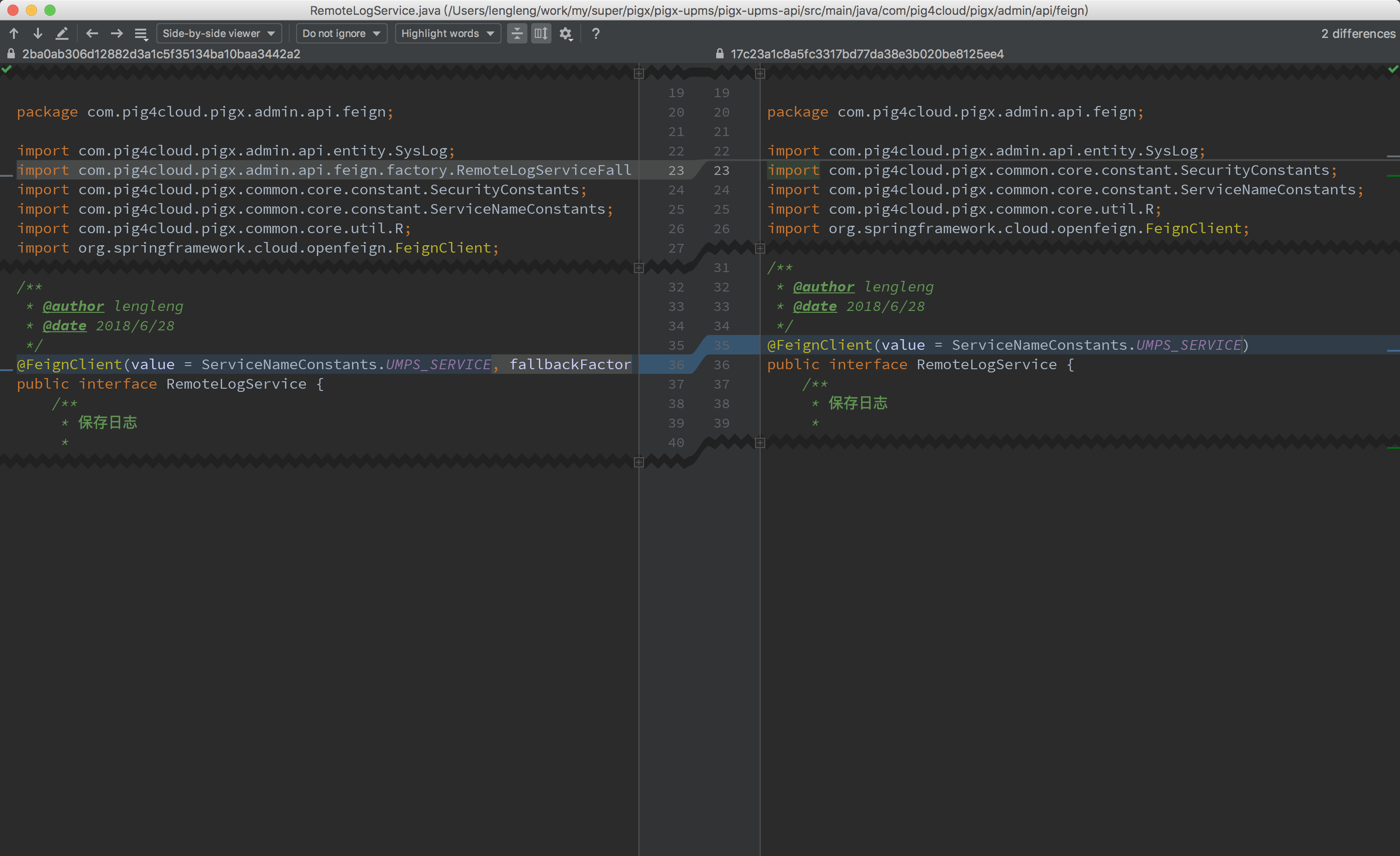
}

}

[**#**](https://pig4cloud.com/#%E8%87%AA%E5%AE%9A%E9%99%8D%E7%BA%A7%E6%95%88%E6%9E%9C) **自定降级效果**

@FeignClient(value = ServiceNameConstants.UMPS\_SERVICE)

public interface RemoteLogService {}

* Feign Service 完成同样的降级错误输出
* FeignClient 中无需定义无用的fallbackFactory
* FallbackFactory 也无需注册到Spring 容器中 

[**#**](https://pig4cloud.com/#%E6%A0%B8%E5%BF%83%E6%BA%90%E7%A0%81) **核心源码**

1. 注入我们个性化后的Feign

@Configuration

@ConditionalOnClass({HystrixCommand.class, HystrixFeign.class})

protected static class HystrixFeignConfiguration {

@Bean

@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

@ConditionalOnProperty("feign.hystrix.enabled")

public Feign.Builder feignHystrixBuilder(FeignContext feignContext) {

return PigxHystrixFeign.builder(feignContext)

.decode404()

.errorDecoder(new PigxFeignErrorDecoder());

}

}

1. PigxHystrixFeign.target 方法是根据@FeignClient 注解生成代理类的过程，注意注释

@Override

public <T> T target(Target<T> target) {

Class<T> targetType = target.type();

FeignClient feignClient = AnnotatedElementUtils.getMergedAnnotation(targetType, FeignClient.class);

String factoryName = feignClient.name();

SetterFactory setterFactoryBean = this.getOptional(factoryName, feignContext, SetterFactory.class);

if (setterFactoryBean != null) {

this.setterFactory(setterFactoryBean);

}

// 以下为获取降级策略代码，构建降级，这里去掉了降级非空的非空的校验

Class<?> fallback = feignClient.fallback();

if (fallback != void.class) {

return targetWithFallback(factoryName, feignContext, target, this, fallback);

}

Class<?> fallbackFactory = feignClient.fallbackFactory();

if (fallbackFactory != void.class) {

return targetWithFallbackFactory(factoryName, feignContext, target, this, fallbackFactory);

}

return build().newInstance(target);

}

1. 构建feign 客户端执行PigxHystrixInvocationHandler的增强

Feign build(@Nullable final FallbackFactory<?> nullableFallbackFactory) {

super.invocationHandlerFactory((target, dispatch) ->

new PigxHystrixInvocationHandler(target, dispatch, setterFactory, nullableFallbackFactory));

super.contract(new HystrixDelegatingContract(contract));

return super.build();

}

1. PigxHystrixInvocationHandler.getFallback() 获取降级策略

@Override

@Nullable

@SuppressWarnings("unchecked")

protected Object getFallback() {

// 如果 @FeignClient 没有配置降级策略，使用动态代理创建一个

if (fallbackFactory == null) {

fallback = PigxFeignFallbackFactory.INSTANCE.create(target.type(), getExecutionException());

} else {

// 如果 @FeignClient配置降级策略，使用配置的

fallback = fallbackFactory.create(getExecutionException());

}

}

1. PigxFeignFallbackFactory.create 动态代理逻辑

public T create(final Class<?> type, final Throwable cause) {

return (T) FALLBACK\_MAP.computeIfAbsent(type, key -> {

Enhancer enhancer = new Enhancer();

enhancer.setSuperclass(key);

enhancer.setCallback(new PigxFeignFallbackMethod(type, cause));

return enhancer.create();

});

}

1. PigxFeignFallbackMethod.intercept， 默认的降级逻辑，输出降级方法信息和错误信息，并且把错误格式

public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodProxy) {

log.error("Fallback class:[{}] method:[{}] message:[{}]",

type.getName(), method.getName(), cause.getMessage());

if (R.class == method.getReturnType()) {

final R result = cause instanceof PigxFeignException ?

((PigxFeignException) cause).getResult() : R.builder()

.code(CommonConstants.FAIL)

.msg(cause.getMessage()).build();

return result;

}

return null;

}