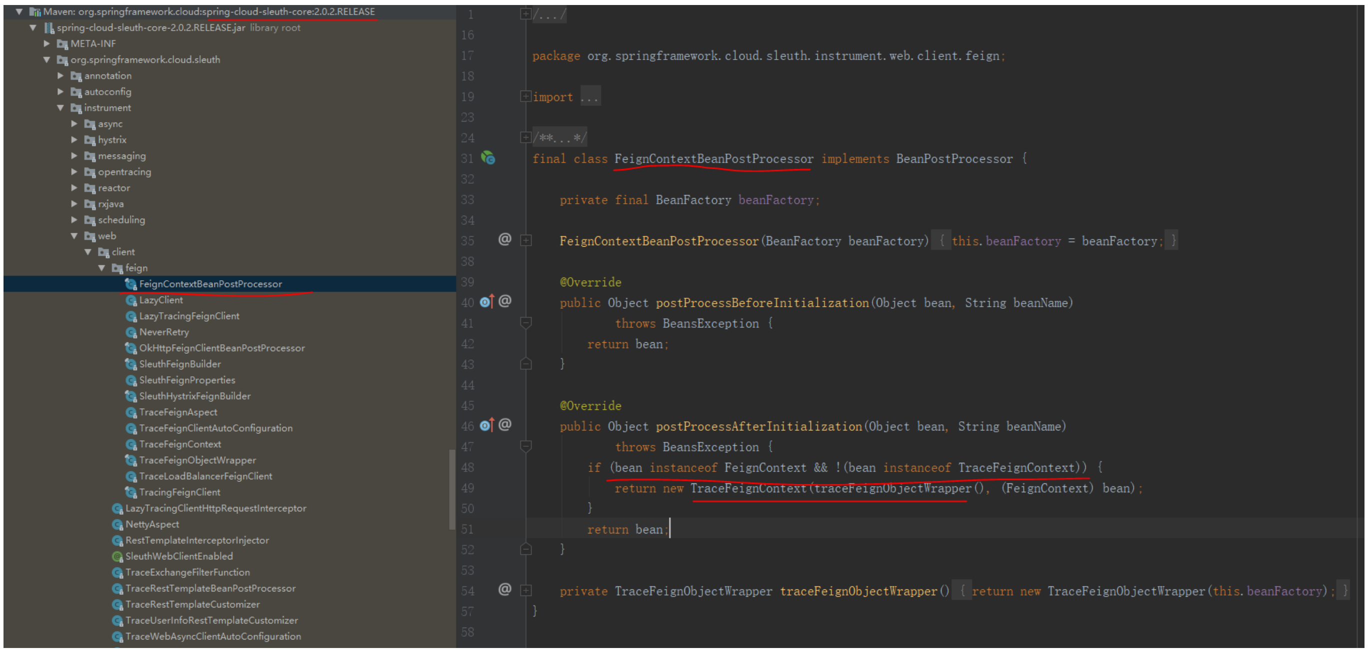
基础概念

核心原理

fegin 核心是客户端 Client，核心流程是通过 factorybean.getcontext() 获取到 fegincontext， 详见 feign源码解析 sleuth 核心流程时 替换feign默认的客户端， TracingFeignClient

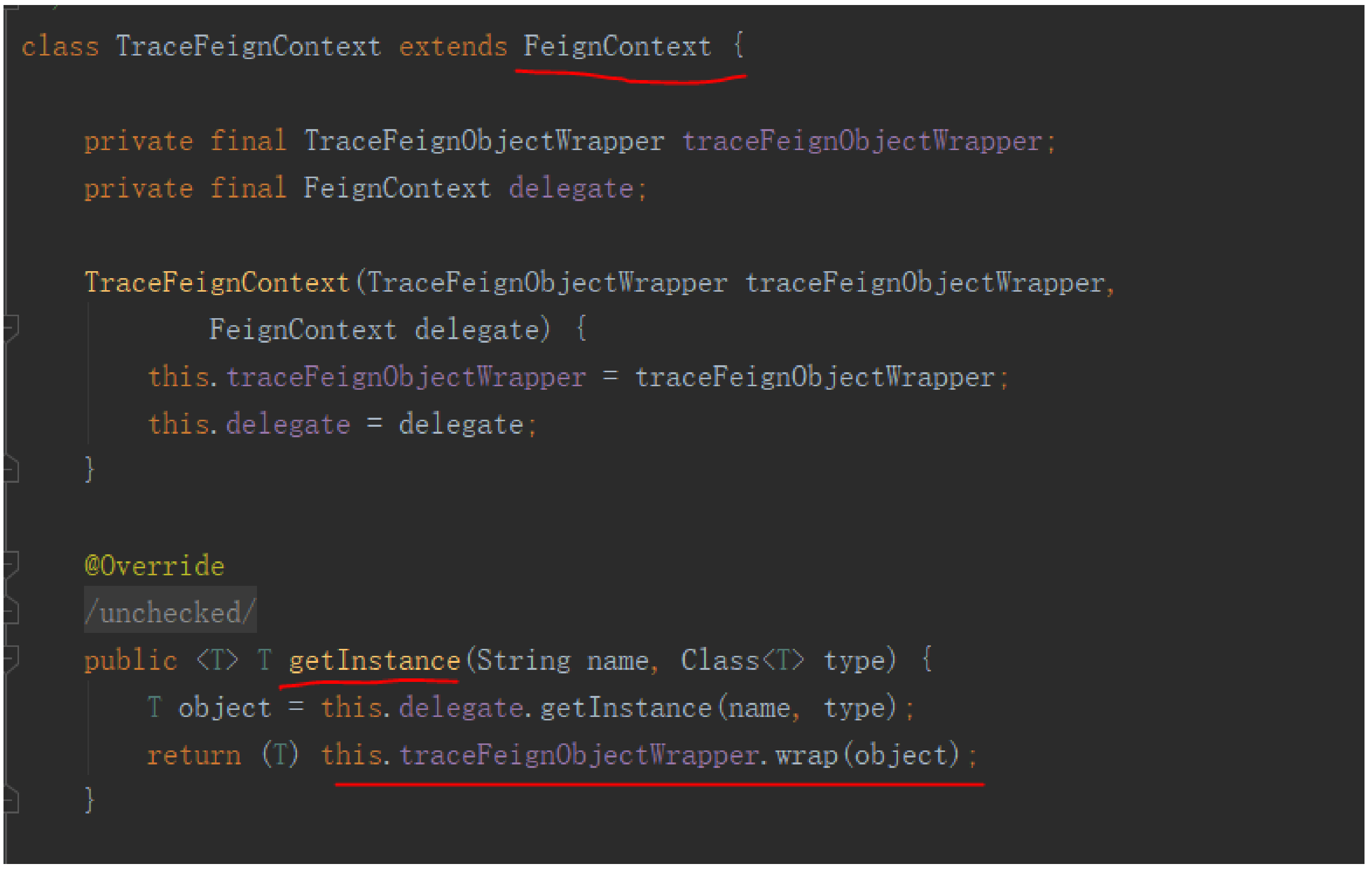
如何替换？ 不同版本是不一样的， 早期版本是在 TraceFeignClientAutoConfiguration 中声明 TracingFeignClient 未primary 今天分析的是另一个版本

FeignContextBeanPostProcessor 通过 容器的接口 BeanPostProcessor 对 feginclient 进行封装

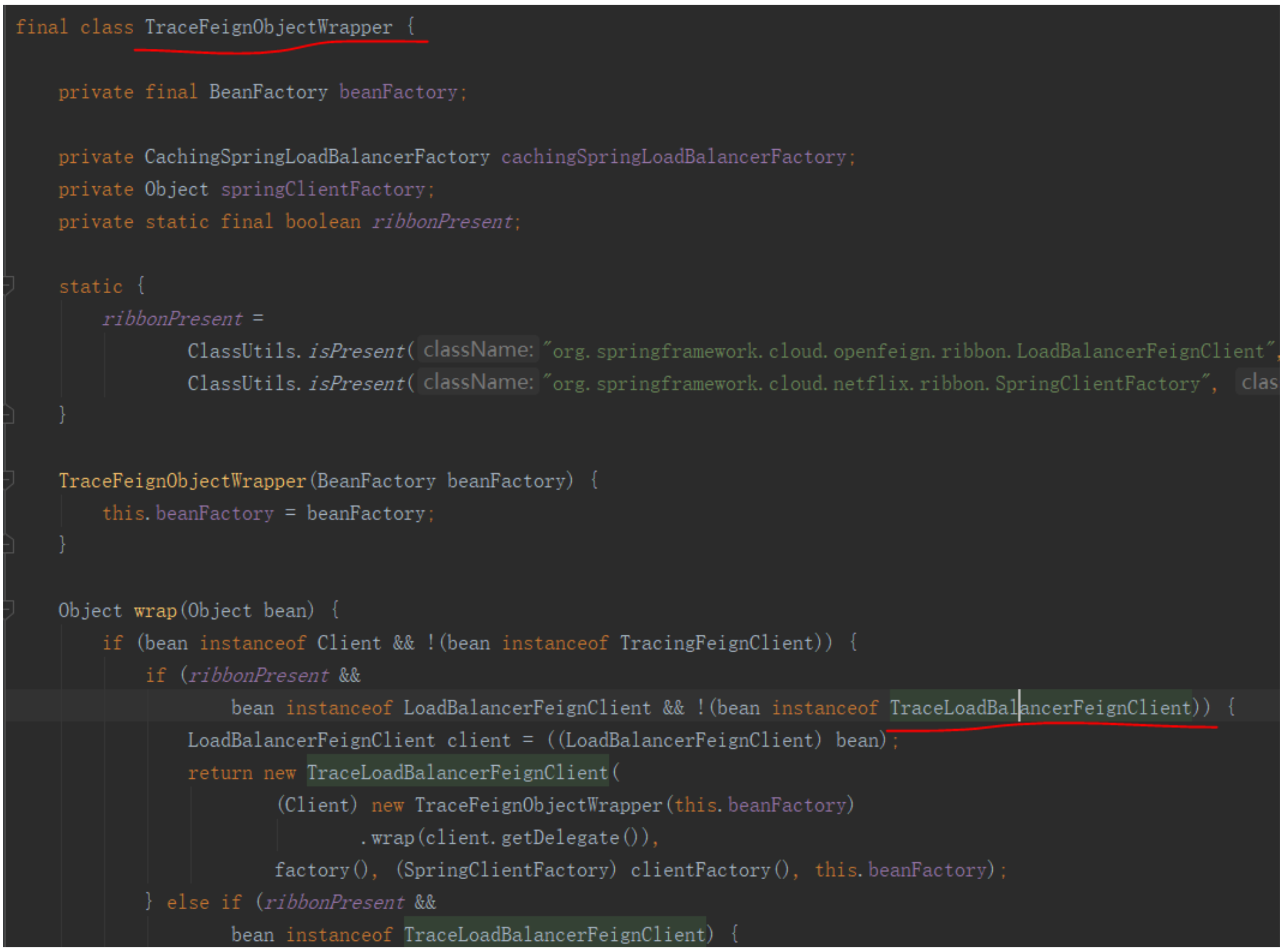


```
▼ Maven: org.springframework.cloud:spring-cloud-sleuth-core:2.0.2.RELEASE
  ▼ spring-cloud-sleuth-core-2.0.2.RELEASE.jar library root
    ▶ META-INF
    ▼ org.springframework.cloud.sleuth
      ▶ annotation
      ▶ autoconfig
      ▼ instrument
        ▶ async
        ▶ hystrix
        ▶ messaging
        ▶ opentracing
        ▶ reactor
        ▶ rxjava
        ▶ scheduling
        ▼ web
          ▼ client
            ▼ feign
                FeignContextBeanPostProcessor
                LazyClient
                LazyTracingFeignClient
                NeverRetry
                OkHttpFeignClientBeanPostProcessor
                SleuthFeignBuilder
                SleuthFeignProperties
                SleuthHystrixFeignBuilder
                TraceFeignAspect
                TraceFeignClientAutoConfiguration
                TraceFeignContext
                TraceFeignObjectWrapper
                TraceLoadBalancerFeignClient
                TracingFeignClient
              LazyTracingClientHttpRequestInterceptor
              NettyAspect
              RestTemplateInterceptorInjector
              SleuthWebClientEnabled
              TraceExchangeFilterFunction
              TraceRestTemplateBeanPostProcessor
              TraceRestTemplateCustomizer
              TraceUserInfoRestTemplateCustomizer
              TraceWebAsyncClientAutoConfiguration
```

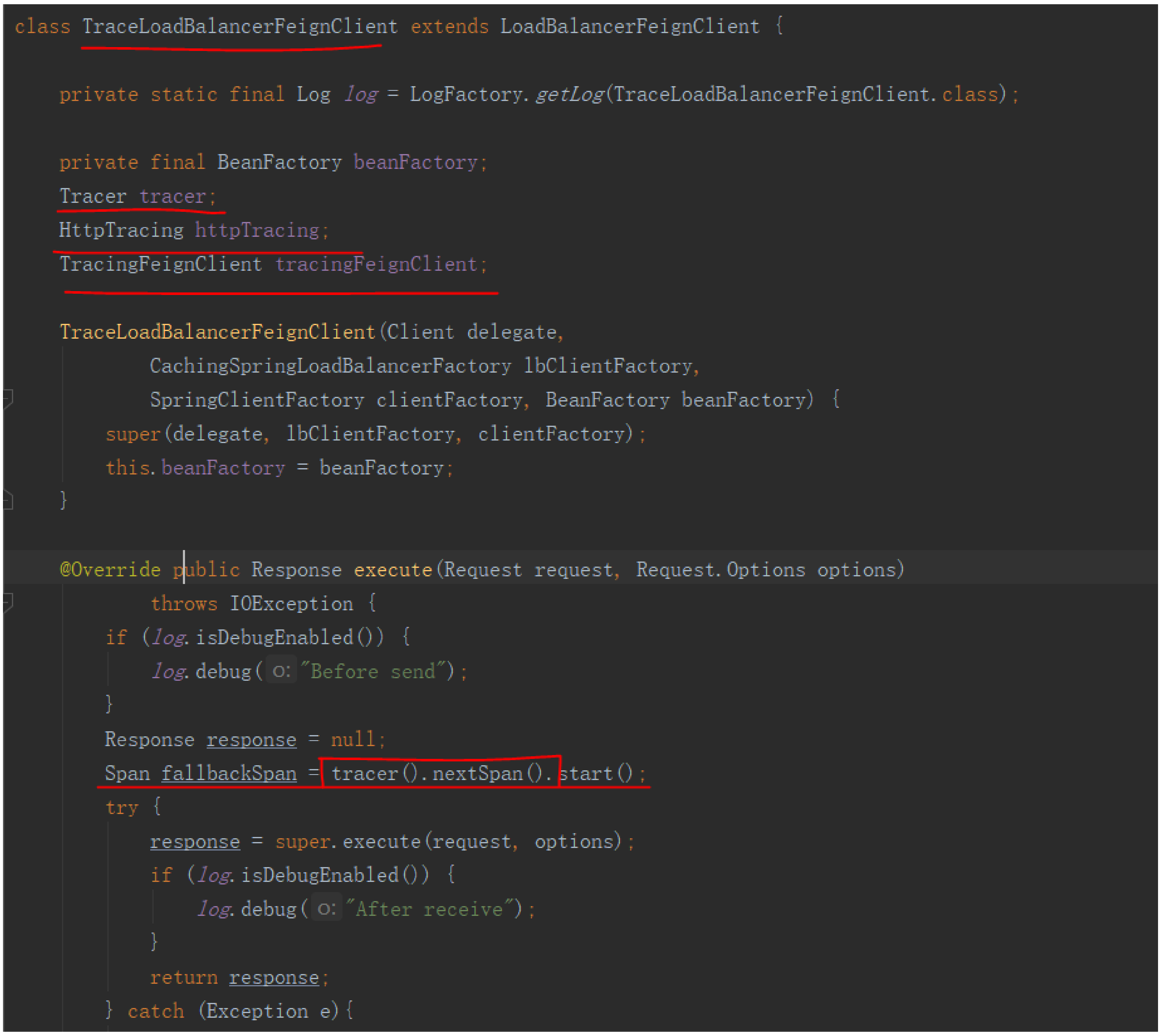
```java
/.../

package org.springframework.cloud.sleuth.instrument.web.client.feign;

import ...

/**...*/
final class FeignContextBeanPostProcessor implements BeanPostProcessor {

    private final BeanFactory beanFactory;

    FeignContextBeanPostProcessor(BeanFactory beanFactory) { this.beanFactory = beanFactory; }

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName)
            throws BeansException {
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName)
            throws BeansException {
        if (bean instanceof FeignContext && !(bean instanceof TraceFeignContext)) {
            return new TraceFeignContext(traceFeignObjectWrapper(), (FeignContext) bean);
        }
        return bean;
    }

    private TraceFeignObjectWrapper traceFeignObjectWrapper() { return new TraceFeignObjectWrapper(this.beanFactory); }
}
```



```java
class TraceFeignContext extends FeignContext {

    private final TraceFeignObjectWrapper traceFeignObjectWrapper;
    private final FeignContext delegate;

    TraceFeignContext(TraceFeignObjectWrapper traceFeignObjectWrapper,
            FeignContext delegate) {
        this.traceFeignObjectWrapper = traceFeignObjectWrapper;
        this.delegate = delegate;
    }

    @Override
    /unchecked/
    public <T> T getInstance(String name, Class<T> type) {
        T object = this.delegate.getInstance(name, type);
        return (T) this.traceFeignObjectWrapper.wrap(object);
    }
```

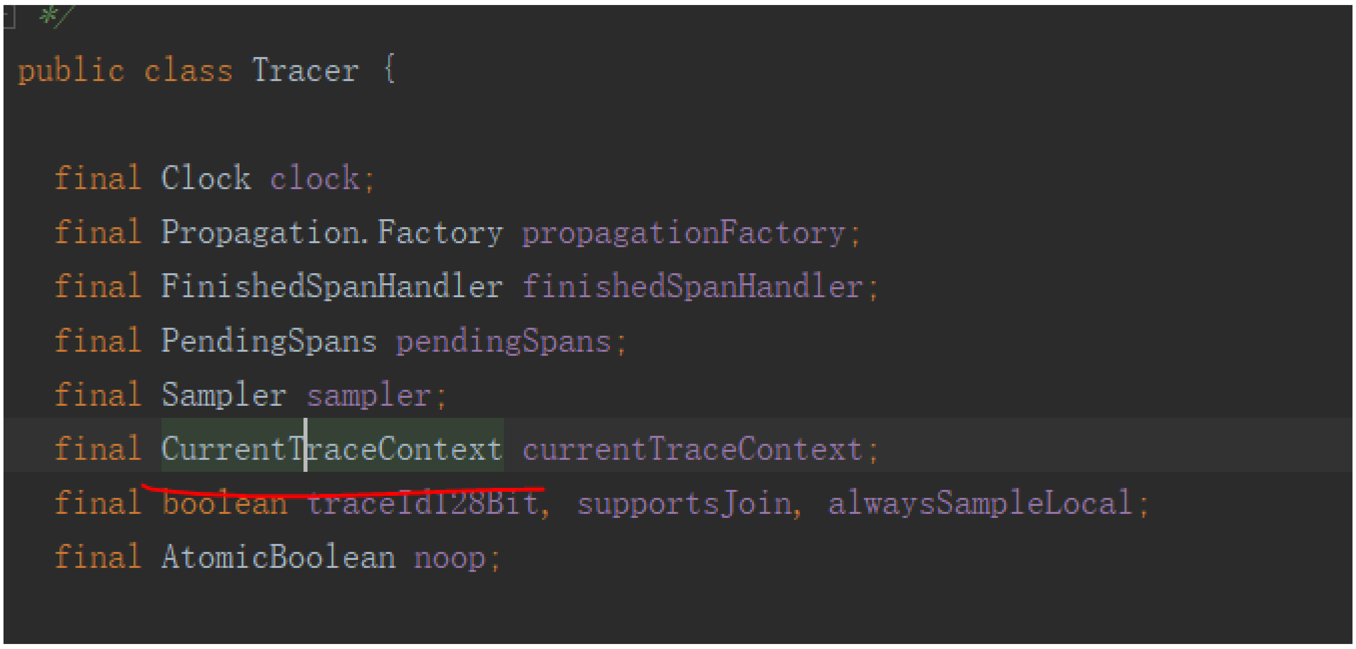


```
final class TraceFeignObjectWrapper {

    private final BeanFactory beanFactory;

    private CachingSpringLoadBalancerFactory cachingSpringLoadBalancerFactory;
    private Object springClientFactory;
    private static final boolean ribbonPresent;

    static {
        ribbonPresent =
                ClassUtils.isPresent( className: "org.springframework.cloud.openfeign.ribbon.LoadBalancerFeignClient",
                ClassUtils.isPresent( className: "org.springframework.cloud.netflix.ribbon.SpringClientFactory",  clas
    }

    TraceFeignObjectWrapper(BeanFactory beanFactory) {
        this.beanFactory = beanFactory;
    }

    Object wrap(Object bean) {
        if (bean instanceof Client && !(bean instanceof TracingFeignClient)) {
            if (ribbonPresent &&
                    bean instanceof LoadBalancerFeignClient && !(bean instanceof TraceLoadBalancerFeignClient)) {
                LoadBalancerFeignClient client = ((LoadBalancerFeignClient) bean);
                return new TraceLoadBalancerFeignClient(
                        (Client) new TraceFeignObjectWrapper(this.beanFactory)
                                .wrap(client.getDelegate()),
                        factory(), (SpringClientFactory) clientFactory(), this.beanFactory);
            } else if (ribbonPresent &&
                    bean instanceof TraceLoadBalancerFeignClient) {
```



```java
class TraceLoadBalancerFeignClient extends LoadBalancerFeignClient {

    private static final Log log = LogFactory.getLog(TraceLoadBalancerFeignClient.class);

    private final BeanFactory beanFactory;
    Tracer tracer;
    HttpTracing httpTracing;
    TracingFeignClient tracingFeignClient;

    TraceLoadBalancerFeignClient(Client delegate,
            CachingSpringLoadBalancerFactory lbClientFactory,
            SpringClientFactory clientFactory, BeanFactory beanFactory) {
        super(delegate, lbClientFactory, clientFactory);
        this.beanFactory = beanFactory;
    }

    @Override public Response execute(Request request, Request.Options options)
            throws IOException {
        if (log.isDebugEnabled()) {
            log.debug( o: "Before send");
        }
        Response response = null;
        Span fallbackSpan = tracer().nextSpan().start();
        try {
            response = super.execute(request, options);
            if (log.isDebugEnabled()) {
                log.debug( o: "After receive");
            }
            return response;
        } catch (Exception e){
```

剩余进入 sluth 的 Tracer 分析



```java
*/
public class Tracer {

  final Clock clock;
  final Propagation.Factory propagationFactory;
  final FinishedSpanHandler finishedSpanHandler;
  final PendingSpans pendingSpans;
  final Sampler sampler;
  final CurrentTraceContext currentTraceContext;
  final boolean traceId128Bit, supportsJoin, alwaysSampleLocal;
  final AtomicBoolean noop;
```

org.springframework.cloud.sleuth.log.Slf4jCurrentTraceContext - 设置 MDC 变量