链滴

谁再悄咪咪的吃掉异常，我上去就是一 JIO

作者：9526xu
原文链接：https://ld246.com/article/1592870191560
来源网站：链滴

又到周末了，周更选手申请出站~

这次分享一下上个月碰到的离奇的问题。一个简单的问题，硬是因为异常被悄咪咪吃掉，过关难度直提升，导致小黑哥排查一个晚上。

这个美好的晚上，本想着开两把 LOL 无限火力，在召唤师峡谷来个五杀的~

哎，就这样没了啊！我知道，你们一定能理解这种五杀被抢的感觉~

下次，真的，谁再让我看到悄咪咪的吃掉异常，我真的要上去一 Jio 了！

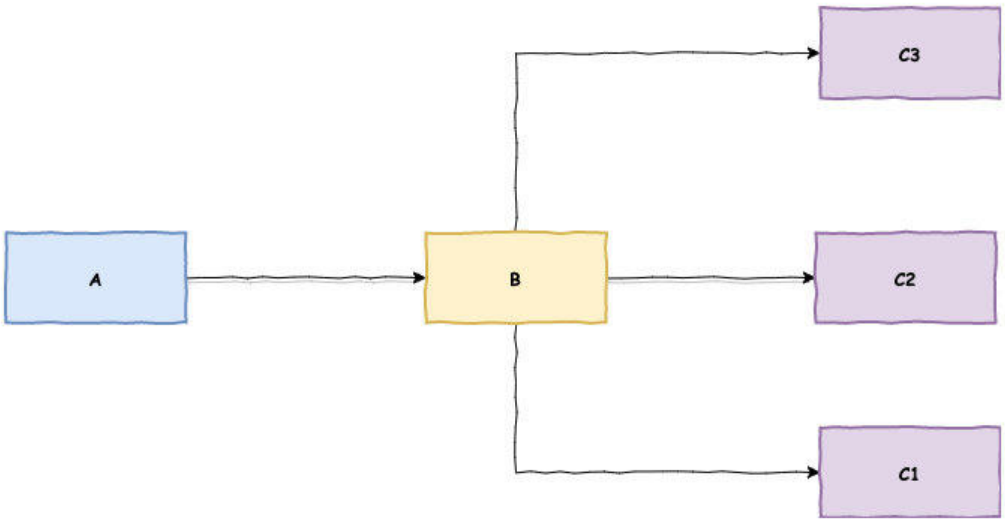
好了，本文可不是水文，看完本篇文章，你可以学到以下知识点：

- Arthas 排查技巧
- 啥是 NoClassDefFoundError
- Dubbo 异常内部处理方式

好了，同学们，打开小本子，准备记好知识点~

起因

我们有个业务系统，应用之间调用链如下所示：



A 应用是业务发生起始应用，在这个应用中将会根据一定规则选择最后的通讯渠道 C，然后将这个渠标识传递给 B 应用。

B 应用的功能类似网关，这个应用将会根据 A 应用传递过来的渠道标识，将会请求路由下发到具体的 应用，起到服务路由的功能。

C 应用是与外部应用交互的应用，我们将其称为渠道通讯机。

假设一次业务中，A 应用根据规则选择 C2 的渠道标识，然后传递给 B 应用。B 应用根据这个标识选使用 C2 进行通讯，最后 C2 调用外部应用完成一次业务调用。

上述所有应用都基于 **Dubbo** 进行远程通讯，B 应用实现原理在小黑哥之前文章「支付路由系统演进史中有写过，感兴趣的同学可以查看一下。

介绍完业务的基本情况，现在我们来看下到底发生了啥事。

一次业务需求中，需要改动 C2 应用，这次改动功能点真的很小，很快就完成了。小黑哥想着闲着也闲着，于是就把之前 C2 应用中打印的日志中一些没有**脱敏**的信息，进行脱敏处理。

由于之前日志框架脱敏处理存在一些问题，于是就将日志框架从 **Log4j** 升级为 **LogBack**。升级之后为了防止不同日志框架中之间的产生冲突，于是使用 **IDEA Maven Helper** 插件，统一将应用中所的 **Log4j** 相关依赖都给排除了。

改动完成之后，将 C2 应用发布到测试环境，再次从 A 应用发起测试， B 应用返回异常提示**未找到 C2 应用**。

B 应用业务代码类似如下：

```
public Response pay(Request req) {

    try {
        if (!isSupport(req.getChnlCode())) {
            return new Response("ERROR", "未找到相关渠道应用");
        }
        return doPay(req);
    } catch (Exception e) {
        return new Response("ERROR", "未找到相关渠道应用");
```

```
    }
}
```

正常情况下，若是配置存在问题，B 应用将会返回未找到具体渠道，请求也会在 B 应用结束，不会调到 C2 应用（也没办法调用）。

然而此次配置什么都没问题， 而且最诡异的是 C2 应用居然收到了请求，并且成功处理了业务请求。

排查问题

由于 B 应用异常处理时，将异常吃掉了，我们没办法得知这个过程到底发生了啥事，所以第一要紧的获取异常信息。

最简单的办法就是，将 B 应用改造一下，加入打印异常日志。不过当时比较懒，不想改造应用，就想取异常信息，于是想到使用 **Arthas**。

Arthas 排错技巧

Arthas 是Alibaba开源的Java诊断工具，这里就不再详细介绍这个工具，主要讲下这次排错用到的命-**watch**。

watch 命令可以方便观察指定方的调用情况，可以具体观察方法的 返回值、抛出异常、入参，另外可以通过 **OGNL**表达式查看对应的变量。

这里我们主要为了查看方法抛出的异常信息，执行命令如下：

```
watch com.dubbo.example.DemoService doPay -e -x 2 '{params,throwExp}'
```

上述命令将会在方法异常之后观察方法的入参以及异常信息。

注意，我们需要查看 doPay 方法，而不是 pay 方法。这是因为 pay方法中我们将异常捕获，不太可会抛出异常哦~

异常信息如下所示：

真正引起此次错误的异常信息为：

```
java.lang.NoClassDefFoundError: Could not initialize class xx.xxx.xx.GELogger
```

由于此次 B 应用不存在改动，所以推测这个异常实际发生在 C2 应用，于是在 C2 应用处再次使用 Art

as **watch** 命令，同样观察到相同的错误信息。

NoClassDefFoundError

NoClassDefFound，从名字上我们可以推测是因为类不存在，从而引发的这个错误。按照这个思路我们首先可以简单查看一下 B 应用中是否存在 GELogger 相关类。

查看 B 应用相关依赖包，从中发现了这个类文件，这说明这个类确实存在。

在 IDEA 反编译查看 GELogger类相关源码，从中发现了问题。

```
private static Logger logger;

static {
    System.out.println("static init");
    logger = Logger.getLogger(NoClassDefFoundErrorTestService.class);
    System.out.println("Logger init success");
}
```

GELogger存在一个静态代码块，用于初始化一个 org.apache.log4j.Logger日志类。

然后在上面改动中，全部的 Log4j依赖都被排除了，所以这里运行时应该会抛出另外一个找到 org.apache.log4j.Logger 错误。

执行以下代码，模拟抛错过程。

```
System.out.println("模拟第一次 Error");
try {
    NoClassDefFoundErrorTestService noClassDefFoundErrorTestService=new NoClassDefFoun
ErrorTestService();
} catch (Throwable e) {
    e.printStackTrace();
}
System.out.println("模拟第二次 Error");
try {
    NoClassDefFoundErrorTestService noClassDefFoundErrorTestService=new NoClassDefFoun
ErrorTestService();
} catch (Throwable e) {
    e.printStackTrace();
}
```

异常信息如下所示：

```
模拟第一次 Error
static init
java.lang.NoClassDefFoundError: org/apache/log4j/Logger
    at com.dubbo.example.NoClassDefFoundErrorTestService.<clinit>(NoClassDefFoundErrorTestService.java:15)
    at com.example.NoClassDefFoundErrorTest.test(NoClassDefFoundErrorTest.java:16) <22 internal calls>
Caused by: java.lang.ClassNotFoundException: org.apache.log4j.Logger <2 internal calls>
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:496)
    ... 24 more
模拟第二次 Error
java.lang.NoClassDefFoundError: Could not initialize class com.dubbo.example.NoClassDefFoundErrorTestService
    at com.example.NoClassDefFoundErrorTest.test(NoClassDefFoundErrorTest.java:22) <22 internal calls>
```

第一次创建 NoClassDefFoundErrorTestService实例时，Java 虚拟机读取加载时，将会初始化静态码块时。由于 org.apache.log4j.Logger类不存在，静态代码块执行异常，从而导致类加载失败。

第二次再创建 NoClassDefFoundErrorTestService 实例时，Java 虚拟机不会再次读取加载，所以直返回了以下异常。

java.lang.NoClassDefFoundError: Could not initialize class com.dubbo.example.NoClassDefFoundErrorTestService

找到问题真正原因，解决办法也很简单，直接排除 GELogger 所在依赖包。

Dubbo 内部异常处理

虽然问题到此解决了，但是这里还有一个疑问，为何 C2 应用发生了异常，却没有相关错误日志，并且 C2 业务逻辑也正常处理完成。

这就要说到 Dubbo 内部异常错误处理方式，上面 GELogger 其实作用在一个 Dubbo 自定义 Filter，用来记录结果，模拟代码如下：

```
@Activate(
        group = {"provider", "consumer"}
)
public class ErrorFilter implements Filter {


    @Override
    public Result invoke(Invoker<?> invoker, Invocation invocation) throws RpcException {

        Result result = invoker.invoke(invocation);
        NoClassDefFoundErrorTestService noClassDefFoundErrorTestService=new NoClassDefFoundErrorTestService();
        // 处理业务逻辑
        return result;
    }
}
```

这个自定义 Filter 中首先执行 invoker 方法，这个方法将会调用真正的业务方法，这就是为什么 C2 用逻辑是正常处理完成。

业务方法处理完成之后，然后执行后续逻辑。由于 NoClassDefFoundErrorTestService将会抛出 Error 最终这个 Error，将会在 HeaderExchangeHandler#handleRequest 被捕获，然后将会把相关异常息返回给调用 Dubbo 消费者。

```
Object msg = req.getData();
try {
    CompletionStage<Object> future = handler.reply(channel, msg);
    future.whenComplete((appResult, t) -> {
        try {
            if (t == null) {
                res.setStatus(Response.OK);
                res.setResult(appResult);
            } else {
                res.setStatus(Response.SERVICE_ERROR);
                res.setErrorMessage(StringUtils.toString(t));
            }
            channel.send(res);
        } catch (RemotingException e) {
            logger.warn( msg: "Send result to consumer failed, channel is " + channel + ", msg is " + e);
        } finally {
            // HeaderExchangeChannel.removeChannelIfDisconnected(channel);
        }
    });
} catch (Throwable e) {
    res.setStatus(Response.SERVICE_ERROR);
    res.setErrorMessage(StringUtils.toString(e));
    channel.send(res);
}
}
```

而在 Dubbo 消费者接受到服务提供者返回信息之后，将会在 DefaultFuture#doReceived转化成 RmotingException。

```
private void doReceived(Response res) {
    if (res == null) {
        throw new IllegalStateException("response cannot be null");
    }
    if (res.getStatus() == Response.OK) {
        this.complete(res.getResult());
    } else if (res.getStatus() == Response.CLIENT_TIMEOUT || res.getStatus() == Response.SERVER_TIMEOUT) {
        this.completeExceptionally(new TimeoutException(res.getStatus() == Response.SERVER_TIMEOUT, channel, res.getErrorMessag
    } else {
        this.completeExceptionally(new RemotingException(channel, res.getErrorMessage()));
    }
}
```

而 RemotingException 最终将会在 FailoverClusterInvoker#doInvoke 转换成 RpcException返回业务代码。

```
}
throw new RpcException(le ≠ null ? le.getCode() : 0, "Failed to invoke the method "
        + invocation.getMethodName() + " in the service " + getInterface().getName()
        + ". Tried " + len + " times of the providers " + providers
        + " (" + providers.size() + "/" + copyinvokers.size()
        + ") from the registry " + directory.getUrl().getAddress()
        + " on the consumer " + NetUtils.getLocalHost() + " using the dubbo version "
        + Version.getVersion() + ". Last error is: "
        + (le ≠ null ? le.getMessage() : ""), le ≠ null && le.getCause() ≠ null ? le.getCause() : le);
}
```

FailoverClusterInvoker#doInvoke

总结

好了，说了这么多，总结一下本文知识点

1. 异常捕获之后，一定要记得打印日志，并且要 **记得输出堆栈信息。**
2. 运行时类不存在，将会导致 NoClassDefFoundError，类加载过程失败，也会导致 NoClassDef

oundError。

3. 对外提供的二方包，最好不要依赖特定日志框架，如 Log4j,Logback 等，应该使用 Slf4j 框架。

帮助

1. 当Dubbo遇上Arthas：排查问题的实践
2. java.lang.NoClassDefFoundError 的解决方法一例
3. noclassdeffounderror-could-not-initialize-class-error

最后最后（IDEA 跳楼价，有需自取）

JetBrains 家的系列产品谁用都说好，但是唯一的缺点就是**贵，贵，贵**~

说实话，之前一直舍不得买，一直在用大学同学的学生账号，不过最近也快到期了~

刚好有个朋友给我推了个数码荔枝 618 年中大促的消息， 6 月 14 日起至 6 月 28 日，两款 JetBrains 产品**首年订阅**的个人用户可享受官网价格 **68 折**优惠。

说实话 JetBrains 官方很少搞活动，这个优惠的价格可能只能在国内代理商搞活动的时候才有。

有需要的同学可以复制下面的链接或者点击原文跳转购买~

IntelliJ IDEA Ultimate: JetBrains Java IDE

价格： ￥700.00

购买地址：https://store.lizhi.io/site/products/id/285?cid=6k78i9zu

JetBrains All Products Pack: 编程开发工具集（全家桶）

价格：￥1200.00

购买地址：https://store.lizhi.io/site/products/id/284?cid=6k78i9zu

本次活动价格为官网价 68 折，但由于汇率影响，可能造成折扣变化，请以实际支付金额为准