39 如何快速排查生产问题

更新时间：2020-08-28 09:52:54

> 耐心是一切聪明才智的基础。——柏拉图

前言

你好，我是彤哥。

上一节，我们一起从监控的角度对实战项目做了增强，有了监控，不仅可以预防生产问题，还可以协助解决生产问题。

本节，我们就来谈谈如何快速排查生产问题。

好了， 进入今天的学习吧。

如何快速排查生产问题？

大家都遇到过哪些生产问题呢？

磁盘满了？

CPU 飙高？

内存溢出？

内存溢出又分好几种：堆内存溢出、元空间溢出、线程栈溢出、直接内存溢出。

在 Netty 中，最常见的当属直接内存溢出了，而内存溢出，往往又是内存泄漏导致的，所以，我们一般是排查内存泄漏。

那么，对于直接内存泄漏，我们该如何快速排查呢？

这里，我总结了几个步骤以供参考：

1. 查看监控，直接内存使用情况是否稳定；
2. 查看日志，是否有直接内存相关的报错；
3. 本地调试，关闭池化内存、添加内存泄漏检测、模拟大量请求、步步为营、耐心寻找泄漏点；

OK，我们以实战项目为例来描述一下各种步骤。

实战项目内存泄漏排查过程

伪造内存泄漏

要想复现内存泄漏，我们需要伪造一个内存泄漏的项目，我们还是以实战项目为例，添加一行代码即可：

```
public class BinaryWebSocketFrameEncoder extends MessageToMessageEncoder<ByteBuf> {
    @Override
    protected void encode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out) throws Exception {
        // 给msg多加一次引用
        ReferenceCountUtil.retain(msg);
        ReferenceCountUtil.retain(msg);
        BinaryWebSocketFrame binaryWebSocketFrame = new BinaryWebSocketFrame(msg);
        out.add(binaryWebSocketFrame);
    }
}
```

在前面的章节中，我们也分析过，retain () 这个方法是给 ByteBuf 加引用次数的，同时，对应的 release () 方法是给 ByteBuf 减引用次数的，每调用一次 Handler（Encoder/Decoder 本质也是 Handler），这个 ByteBuf 的引用次数都会自动减 1，这个自动减 1 的操作是在父类中自动完成的，当然，也不是所有的父类都会帮你完成这个动作，比如，ChannelInboundHandlerAdapter 就不会帮你把引用次数自动减 1。

所以，我们这里多次使用的时候需要显式地给 msg 加一次引用次数，但是，一不小心，我手抖了一下，多加了一次引用次数就发到生产上了，会出现什么样的情况呢？

首先，程序是肯定可以正常运行的。

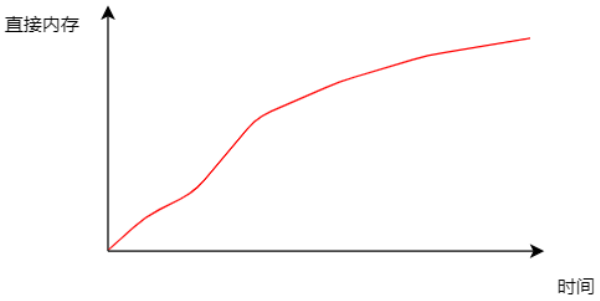
其次，内存溢出是一个非常缓慢的过程，可能十天半个月才会出现一次。

最后，导致的结果就是难以排查，无奈之下，只能重启，甚至，有的公司会写个定时任务，每天晚上系统没人用的时候偷偷地自动重启一次。

针对这种情况，我们来看看怎么排查。

查看监控

查看监控，会发现，直接内存的使用情况，呈现一条缓慢上升的曲线，类似于这样：



这个时间的长度可能是几个小时、一天、几天、十几天，都有可能，取决于系统的流量，随着时间的推移，最终达到了监控告警值，半夜被喊起来排查问题。

查看日志

遇到这种问题先不要慌，先去日志中查看是否有直接内存相关的报错，如果运气好是有相关日志的，拿出来分析一波，最坏的情况是完全没有，在完全没有日志可以参考的情况下，只能启用本地调试大法了。

本地调试

本地调试不是说拿到代码胡乱调试一通，也要讲究一些策略。

第一步，关闭池化内存，为什么要关闭池化内存？

因为，使用池化内存的时候，创建第一个 ByteBuf 的时候会分配 16M 的内存块，等这 16M 的内存块不够使用的时候才会创建下一个 16M 的内存块，而且，这 16M 使用完了之后并不会立即释放，而是交给 Netty 的内存来进行管理，方便后续复用，有内存池的干扰，很难看出内存泄漏导致的直接内存缓慢增长的过程。

下面是没有关闭池化内存的情况：

```
-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 16777223

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 16777223

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 16777223
```

长期维持在 16M，也看不出有没有内存泄漏。

下面是关闭池化内存的情况：

```
-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 0

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 1799

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 4359
```

可以看到，关闭了池化内存之后，是有一个缓慢增长的过程的。

那么，要怎么关闭池化内存呢？

这就要使用到前面章节讲过的一个参数了：`-Dio.netty.allocator.type=unpooled`，将这个参数的值设置为 unpooled 即可。

第二步，添加内存泄漏检测，如何添加内存泄漏检测？

这就要使用一个我们没讲过的参数了：`-Dio.netty.leakDetection.level=PARANOID`，这个参数有四个值：

- DISABLED，表示不检测内存泄漏；
- SIMPLE，表示会追踪小部分对象的内存使用情况，同时会消耗少量的资源；
- ADVANCED，表示会追踪大部分对象的内存使用情况，同时会消息大量的资源；
- PARANOID，表示会追踪所有对象的内存使用情况。

其中，默认值是 SIMPLE，四个值的强度为依次增加。

上面在查看日志的时候说的运气好，就是指这里使用 SIMPLE 的情况下有可能会有相关的日志。

在本地调试，直接开启最高级别 ——PARANOID。

第三步，模拟大量请求，为什么要模拟大量请求呢？

因为，上面的内存泄漏检测，只有在 gc 的时候才会打印相关的日志，平时，这些信息会保存在 Set 中，有兴趣的同学可以看看 `ResourceLeakDetector` 这个类，怎么才能快速 gc 呢？模拟大量的请求不失为一种好方法。

那么，怎么模拟大量的请求呢？

其实，也很简单，只需要在 MockClient 中加一个循环即可：

```
public class MockClient {
    public static void start(Channel channel) {
        // 发送hello消息
        for (int i = 0; i < 1000; i++) {
            HelloRequest helloRequest = HelloRequest.newBuilder()
                    .setName("彤哥")
                    .build();
            MessageUtils.sendRequest(helloRequest);
            try {
                TimeUnit.MILLISECONDS.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

OK，此时，分别启动服务端和客户端，观察服务端的日志，正常来说，就会出现错误信息了，我这里摘取一条：

```
07:40:04 [nioEventLoopGroup-3-2] ResourceLeakDetector: LEAK: ByteBuf.release() was not called before it's garbage-collected. See https://netty.io/wiki
/reference-counted-objects.html for more information.
Recent access records:
#1:
 io.netty.buffer.DefaultByteBufHolder.release(DefaultByteBufHolder.java:115)
 io.netty.util.ReferenceCountUtil.release(ReferenceCountUtil.java:88)
 io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:91)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite(AbstractChannelHandlerContext.java:707)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:790)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:700)
 io.netty.channel.ChannelOutboundHandlerAdapter.write(ChannelOutboundHandlerAdapter.java:113)
 io.netty.handler.codec.http.websocketx.WebSocketCloseFrameHandler.write(WebSocketCloseFrameHandler.java:73)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite(AbstractChannelHandlerContext.java:707)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:790)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:700)
 io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:112)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite(AbstractChannelHandlerContext.java:707)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:790)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:700)
 io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:112)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
 io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
 io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
 io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
 io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
 io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
 io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
 io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
 io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
 io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
 java.lang.Thread.run(Thread.java:748)
#2:
 io.netty.util.ReferenceCountUtil.release(ReferenceCountUtil.java:88)
 io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:91)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite(AbstractChannelHandlerContext.java:707)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:790)
 io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:700)
 io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:112)
 io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
 io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
 io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
 io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
 io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
 io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
```

```
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
#3:
io.netty.buffer.AdvancedLeakAwareByteBuf.retain(AdvancedLeakAwareByteBuf.java:36)
io.netty.util.ReferenceCountUtil.retain(ReferenceCountUtil.java:40)
com.imooc.netty.mahjong.common.codec.BinaryWebSocketFrameEncoder.encode(BinaryWebSocketFrameEncoder.java:17)
com.imooc.netty.mahjong.common.codec.BinaryWebSocketFrameEncoder.encode(BinaryWebSocketFrameEncoder.java:12)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite(AbstractChannelHandlerContext.java:707)
io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:790)
io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:700)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:112)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
#4:
io.netty.buffer.AdvancedLeakAwareByteBuf.retain(AdvancedLeakAwareByteBuf.java:36)
io.netty.util.ReferenceCountUtil.retain(ReferenceCountUtil.java:40)
com.imooc.netty.mahjong.common.codec.BinaryWebSocketFrameEncoder.encode(BinaryWebSocketFrameEncoder.java:16)
com.imooc.netty.mahjong.common.codec.BinaryWebSocketFrameEncoder.encode(BinaryWebSocketFrameEncoder.java:12)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite(AbstractChannelHandlerContext.java:707)
io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:790)
io.netty.channel.AbstractChannelHandlerContext.write(AbstractChannelHandlerContext.java:700)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:112)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
#5:
io.netty.buffer.AdvancedLeakAwareByteBuf.writeInt(AdvancedLeakAwareByteBuf.java:562)
com.imooc.netty.mahjong.common.protocol.MahjongProtocolHeader.encode(MahjongProtocolHeader.java:30)
com.imooc.netty.mahjong.common.protocol.MahjongProtocol.encode(MahjongProtocol.java:52)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:17)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:11)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
#6:
io.netty.buffer.AdvancedLeakAwareByteBuf.writeInt(AdvancedLeakAwareByteBuf.java:562)
```

```
com.imooc.netty.mahjong.common.protocol.MahjongProtocolHeader.encode(MahjongProtocolHeader.java:29)
com.imooc.netty.mahjong.common.protocol.MahjongProtocol.encode(MahjongProtocol.java:52)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:17)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:11)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
#7:
io.netty.buffer.AdvancedLeakAwareByteBuf.writeInt(AdvancedLeakAwareByteBuf.java:562)
com.imooc.netty.mahjong.common.protocol.MahjongProtocolHeader.encode(MahjongProtocolHeader.java:28)
com.imooc.netty.mahjong.common.protocol.MahjongProtocol.encode(MahjongProtocol.java:52)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:17)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:11)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
Created at:
io.netty.buffer.UnpooledByteBufAllocator.newDirectBuffer(UnpooledByteBufAllocator.java:96)
io.netty.buffer.AbstractByteBufAllocator.directBuffer(AbstractByteBufAllocator.java:187)
io.netty.buffer.AbstractByteBufAllocator.directBuffer(AbstractByteBufAllocator.java:173)
io.netty.buffer.AbstractByteBufAllocator.buffer(AbstractByteBufAllocator.java:107)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:16)
com.imooc.netty.mahjong.common.codec.MahjongProtocolEncoder.encode(MahjongProtocolEncoder.java:11)
io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
io.netty.channel.AbstractChannelHandlerContext.invokeWrite0(AbstractChannelHandlerContext.java:715)
io.netty.channel.AbstractChannelHandlerContext.invokeWriteAndFlush(AbstractChannelHandlerContext.java:762)
io.netty.channel.AbstractChannelHandlerContext$WriteTask.run(AbstractChannelHandlerContext.java:1089)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute$$$capture(AbstractEventExecutor.java:164)
io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java)
io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:472)
io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:497)
io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
java.lang.Thread.run(Thread.java:748)
: 8 leak records were discarded because the leak record count is targeted to 4. Use system property io.netty.leakDetection.targetRecords to increase th
e limit.
```

这是一条日志就有这么长，它记录了发生内存泄漏的那个 ByteBuf 所经过的路径，像这样的日志还有很多条躺在 Console 中，等待你去分析 ^^

第四步，步步为营，耐心寻找泄漏点。

这一步没有什么取巧的办法，只能慢慢分析上面的日志，或者，使用调试大法跟踪 ByteBuf 走过的路径，再慢慢缩小范围一步一步排查。

在上面日志的最后，已经明确告诉了你这个 ByteBuf 是在哪里创建的了，所以，使用调试大法的时候直接把断点打在那里一步一步往后面跟就可以了。

调试大法，前面我们已经用过太多次了，这里就不细说了，主要还是看日志，找到上面日志的 “#”3 处：

```
#3:
 io.netty.buffer.AdvancedLeakAwareByteBuf.retain(AdvancedLeakAwareByteBuf.java:36)
 io.netty.util.ReferenceCountUtil.retain(ReferenceCountUtil.java:40)
 com.imooc.netty.mahjong.common.codec.BinaryWebSocketFrameEncoder.encode(BinaryWebSocketFrameEncoder.java:17)
 com.imooc.netty.mahjong.common.codec.BinaryWebSocketFrameEncoder.encode(BinaryWebSocketFrameEncoder.java:12)
 io.netty.handler.codec.MessageToMessageEncoder.write(MessageToMessageEncoder.java:89)
```

这里有个类叫作 “BinaryWebSocketFrameEncoder”，比较可疑，在 IDEA 中点击类名即可跳到对应的类：

```
public class BinaryWebSocketFrameEncoder extends MessageToMessageEncoder<ByteBuf> {
    @Override
    protected void encode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out) throws Exception {
        ReferenceCountUtil.retain(msg);
        ReferenceCountUtil.retain(msg);
        BinaryWebSocketFrame binaryWebSocketFrame = new BinaryWebSocketFrame(msg);
        out.add(binaryWebSocketFrame);
    }
}
```

好吧，原来是这里多加了一次引用计数，导致这个 msg 回收不掉，最终导致内存泄漏。

把 retain () 去掉一行，重启服务端和客户端，观察服务端日志：

```
-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 7

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 78

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 14

-- Gauges ----------------------------------------------------------------------
maxDirectMemory
             value = 1883242496
usedDirectMemory
             value = 7
```

一切都正常了，直接内存即使偶而有升高，也会很快降下来。

好了，整个排查过程就介绍完了，那么，我们在平时的开发中有没有什么好的方法可以规避这种问题呢？

预防

要想预防这种直接内存泄漏的问题，方法还是有的，最简单有效的方法就是在测试环境设置这两个参数：

- -Dio.netty.allocator.type=unpooled
- -Dio.netty.leakDetection.level=PARANOID

在测试同学压测的时候，观察一下监控的情况，如果监控出现了直接内存缓慢增长的情况，那说明就有内存泄漏了，提前排查问题提前规避，当然，排查的过程还是一样要分析日志，或者耐心调试，这是少不了的步骤。

最后，在生产环境，记得把这两个参数去掉，或者恢复到默认值。

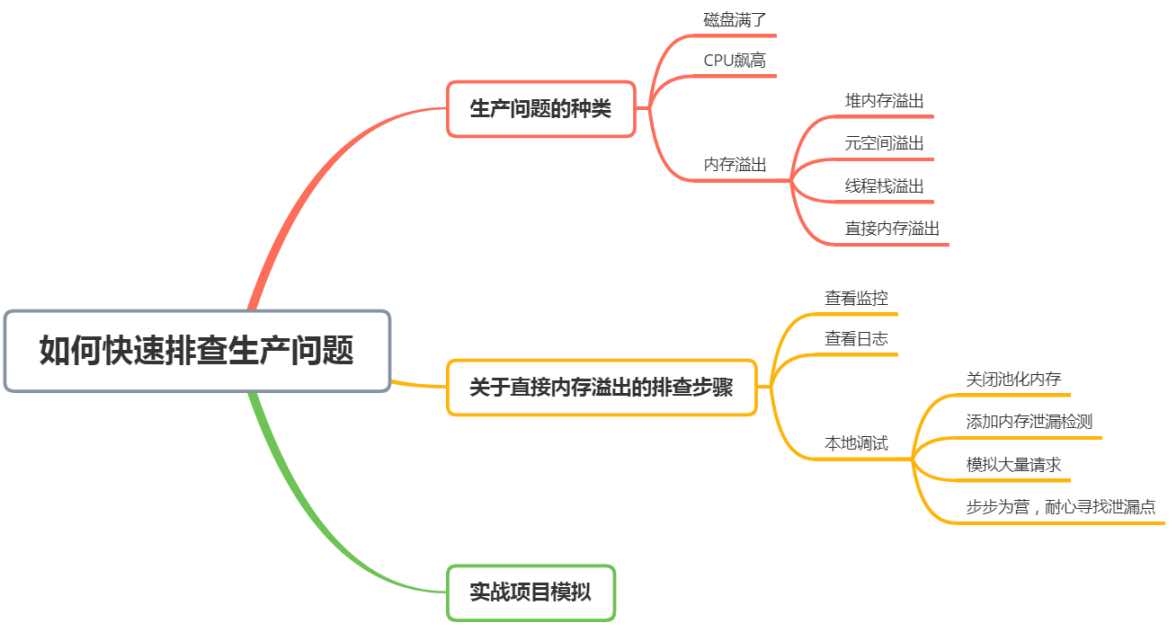
后记

本节，我们一起研究了如何快速排查生产问题中的直接内存泄漏问题，通过模拟实战项目的直接内存泄漏，可以给我们快速积累经验，即使生产环境真的出现了，我们也可以淡然面对，快速找出真凶。

在本节的附录部分，我准备了一个小案例，看看你能不能找出其中内存泄漏的真正原因。

到此，整个专栏就趋近于尾声了，下一节，我将对整个课程做一个总结与回顾，敬请期待。

思维导图



附录 —— 直接内存泄漏的小案例

点击链接直达：https://github.com/alan-tang-tt/springcloud-gateway-oom

}

38 如何增加监控

40 课程总结与回顾