- NETTY

精尽 Netty 源码解析 —— Channel（六）之 writeAndFlush 操作

1. 概述

本文接《精尽 Netty 源码解析 —— Channel（五）之 flush 操作》，分享 Netty Channel 的 `#writeAndFlush(Object msg, ...)` 方法，write + flush 的组合，将数据写到内存队列后，立即刷新**内存队列**，又将其中的数据写入到对端。

😈 本来是不准备写这篇的，因为内容主要是《精尽 Netty 源码解析 —— Channel（四）之 write 操作》和《精尽 Netty 源码解析 —— Channel（五）之 flush 操作》的组合。但是，考虑到内容的完整性，于是乎就稍微水更下下。

文章目录

el

`ush(Object msg, ...)` 方法的实现，代码如下：

```
dFlush(Object msg) {
    return pipeline.writeAndFlush(msg);
}

@Override
public ChannelFuture writeAndFlush(Object msg, ChannelPromise promise) {
    return pipeline.writeAndFlush(msg, promise);
}
```

- 在方法内部，会调用对应的 `ChannelPipeline#write(Object msg, ...)` 方法，将 write 和 flush **两个**事件在 pipeline 上传播。详细解析，见「3. DefaultChannelPipeline」。
  - 最终会传播 write 事件到 `head` 节点，将数据写入到内存队列中。详细解析，见「5. HeadContext」。
  - 最终会传播 flush 事件到 `head` 节点，刷新**内存队列**，将其中的数据写入到对端。详细解析，见「5. HeadContext」。

3. DefaultChannelPipeline

`DefaultChannelPipeline#writeAndFlush(Object msg, ...)` 方法，代码如下：

```
@Override
public final ChannelFuture write(Object msg) {
    return tail.writeAndFlush(msg);
}
```

```
@Override
public final ChannelFuture write(Object msg, ChannelPromise promise) {
    return tail.writeAndFlush(msg, promise);
}
```

- 在方法内部，会调用 `TailContext#writeAndFlush(Object msg, ...)` 方法，将 write 和 flush **两个**事件在 pipeline 中，从尾节点向头节点传播。详细解析，见「4. TailContext」。

4. TailContext

TailContext 对 `TailContext#writeAndFlush(Object msg, ...)` 方法的实现，是从 AbstractChannelHandlerContext 抽象类继承，代码如下：

```
@Override
public ChannelFuture writeAndFlush(Object msg, ChannelPromise promise) {
    if (msg == null) {
        throw new NullPointerException("msg");
    }

                                    e 对象
                                    omise, true)) {
                                    关的资源
                                    elease(msg);

    // 写入消息( 数据 )到内存队列
    write(msg, true, promise); // <1>

    return promise;
}
```


- 这个方法，和我们在《精尽 Netty 源码解析 —— Channel（四）之 write 操作》的「4. TailContext」的小节，`TailContext#write(Object msg, ...)` 方法，基本类似，差异在于 `<1>` 处，调用 `#write(Object msg, boolean flush, ChannelPromise promise)` 方法，传入的 `flush = true` 方法参数，表示 write 操作的同时，**后续**需要执行 flush 操作。代码如下：

```
private void write(Object msg, boolean flush, ChannelPromise promise) {
    // 获得下一个 Outbound 节点
    AbstractChannelHandlerContext next = findContextOutbound();
    // 简化代码 😈
    // 执行 write + flush 操作
    next.invokeWriteAndFlush(m, promise);
}

private void invokeWriteAndFlush(Object msg, ChannelPromise promise) {
    if (invokeHandler()) {
        // 执行 write 事件到下一个节点
        invokeWrite0(msg, promise);
```

```
            // 执行 flush 事件到下一个节点
            invokeFlush0();
        } else {
            writeAndFlush(msg, promise);
        }
    }
```

- 在后面，就是《精尽 Netty 源码解析 —— Channel（四）之 write 操作》的「5. HeadContext」的小节及其后续的小节。
- 再在后面，就是《精尽 Netty 源码解析 —— Channel（五）之 flush 操作》。

666. 彩蛋

😈 真的是水更，哈哈哈哈。

推荐阅读文章：

- 闪电侠《netty 源码分析之 writeAndFlush 全解析》的「writeAndFlush: 写队列并刷新」小节。