我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro

- NETTY

精尽 Netty 源码解析 —— ChannelPipeline（三）之移除 ChannelHandler

文章目录

elHandler 的代码具体实现。

eline（一）之初始化》 中，我们看到 ChannelPipeline 定义了一大堆**移除**

```
ChannelPipeline remove(ChannelHandler handler);
ChannelHandler remove(String name);
<T extends ChannelHandler> T remove(Class<T> handlerType);
ChannelHandler removeFirst();
ChannelHandler removeLast();
```

- 本文仅分享 `#remove(ChannelHandler handler)` 方法，从 pipeline **移除**指定的 ChannelHandler 对象。

2. remove

`#remove(ChannelHandler handler)` 方法，从 pipeline **移除**指定的 ChannelHandler 对象。代码如下：

```
@Override
public final ChannelPipeline remove(ChannelHandler handler) {
    remove(getContextOrDie(handler));
    return this;
}
```

- 调用 `#getContextOrDie(ChannelHandler handler)` 方法，获得对应的 AbstractChannelHandlerContext 节点。代码如下：

```
private AbstractChannelHandlerContext getContextOrDie(ChannelHandler handler) {
    AbstractChannelHandlerContext ctx = (AbstractChannelHandlerContext) context(handler);
    if (ctx == null) { // die
        throw new NoSuchElementException(handler.getClass().getName());
    } else {
        return ctx;
    }
}
```

```
@Override
public final ChannelHandlerContext context(ChannelHandler handler) {
    if (handler == null) {
        throw new NullPointerException("handler");
    }
    AbstractChannelHandlerContext ctx = head.next;
    // 循环，获得指定 ChannelHandler 对象的节点
    for (;;) {
        if (ctx == null) {
            return null;
        }
        if (ctx.handler() == handler) { // ChannelHandler 相等
            return ctx;
```

文章目录

点的情况下，抛出 NoSuchElementException 异常。

lerContext ctx) 方法，移除指定 AbstractChannelHandlerContext 节点。

#remove(AbstractChannelHandlerContext ctx) 方法，移除指定 AbstractChannelHandlerContext 节点。代码如下：

> 代码的整体结构，和 #addLast(EventExecutorGroup group, String name, ChannelHandler handler) 方法是一致的。

```
1: private AbstractChannelHandlerContext remove(final AbstractChannelHandlerContext ctx) {
2:     assert ctx != head && ctx != tail;
3:
4:     synchronized (this) { // 同步，为了防止多线程并发操作 pipeline 底层的双向链表
5:         // 移除节点
6:         remove0(ctx);
7:
8:         // pipeline 暂未注册，添加回调。再注册完成后，执行回调。详细解析，见 {@link #callHandlerCallbac
9:         // If the registered is false it means that the channel was not registered on an eventloop
10:        // In this case we remove the context from the pipeline and add a task that will call
11:        // ChannelHandler.handlerRemoved(...) once the channel is registered.
12:        if (!registered) {
13:            callHandlerCallbackLater(ctx, false);
14:            return ctx;
15:        }
16:
17:        // 不在 EventLoop 的线程中，提交 EventLoop 中，执行回调用户方法
18:        EventExecutor executor = ctx.executor();
19:        if (!executor.inEventLoop()) {
20:            // 提交 EventLoop 中，执行回调 ChannelHandler removed 事件
21:            executor.execute(new Runnable() {
22:                @Override
23:                public void run() {
```

```
24:                     callHandlerRemoved0(ctx);
25:                 }
26:             });
27:             return ctx;
28:         }
29:     }
30:
31:     // 回调 ChannelHandler removed 事件
32:     callHandlerRemoved0(ctx);
33:     return ctx;
34: }
```

文章目录

止多线程并发操作 pipeline 底层的双向链表。

nnelHandlerContext ctx) 方法，从 pipeline **移除**指定的

详细解析，见「3. remove0」。

=====

kLater(AbstractChannelHandlerContext, added) 方法，添加

Channel 注册完成后，执行该回调。详细解析，见「8.

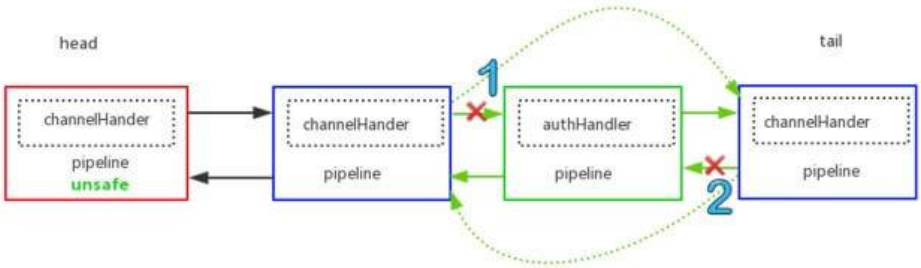
- <2>
- 第 19 行：不在 EventLoop 的线程中。
- 第 20 至 26 行：提交 EventLoop 中，调用 #callHandlerRemoved0(AbstractChannelHandlerContext) 方法，执行回调 ChannelHandler 移除完成(removed)事件。详细解析，见「4. callHandlerRemoved0」。
- <3>
- 这种情况，是 <2> 在 EventLoop 的线程中的版本。也因为此，已经确认在 EventLoop 的线程中，所以不需要在 synchronized 中。
- 第 32 行：和【第 24 行】的代码一样，调用 #callHandlerRemoved0(AbstractChannelHandlerContext) 方法，执行回调 ChannelHandler 移除完成(removed)事件。

3. remove0

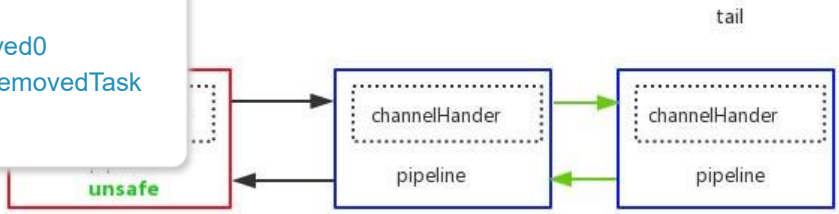
#remove0(AbstractChannelHandlerContext ctx) 方法，从 pipeline **移除**指定的 AbstractChannelHandlerContext 节点。代码如下：

```
private static void remove0(AbstractChannelHandlerContext ctx) {
    // 获得移除节点的前后节点
    AbstractChannelHandlerContext prev = ctx.prev;
    AbstractChannelHandlerContext next = ctx.next;
    // 前后节点互相指向
    prev.next = next;
    next.prev = prev;
}
```

> FROM 闪电侠 《netty 源码分析之 pipeline(一)》
>
> - 经历的过程要比添加节点要简单，可以用下面一幅图来表示



文章目录



结合这两幅图，可以很清晰地了解移除 Handler 的过程，另外，被删除的节点因为没有对象引用到，果过段时间就会被 gc 自动回收。

4. callHandlerRemoved0

`#callHandlerRemoved0(AbstractChannelHandlerContext)` 方法，执行回调 ChannelHandler 移除完成(removed)事件。代码如下：

```
1: private void callHandlerRemoved0(final AbstractChannelHandlerContext ctx) {
2:     // Notify the complete removal.
3:     try {
4:         try {
5:             // 回调 ChannelHandler 移除完成( removed )事件
6:             ctx.handler().handlerRemoved(ctx);
7:         } finally {
8:             // 设置 AbstractChannelHandlerContext 已移除
9:             ctx.setRemoved();
10:        }
11:    } catch (Throwable t) {
12:        // 触发异常的传播
13:        fireExceptionCaught(new ChannelPipelineException(
14:                ctx.handler().getClass().getName() + ".handlerRemoved() has thrown an exception.",
15:    }
16: }
```

- 第 6 行：调用 `ChannelHandler#handlerRemoved(AbstractChannelHandlerContext)` 方法，回调 ChannelHandler 移除完成(removed)事件。一般来说，通过这个方法，来释放 ChannelHandler 占用的资源。**注意**，因为这个方法的执行在 EventLoop 的线程中，所以要尽量避免执行时间过长。
- 第 9 行：调用 `AbstractChannelHandlerContext#setRemoved()` 方法，设置 AbstractChannelHandlerContext 已移除。
- 第 11 至 15 行：发生异常，触发异常的传播。详细解析，见《精尽 Netty 源码解析 —— ChannelPipeline（六）之异常事件的传播》 。

5. PendingHandlerRemovedTask

PendingHandlerRemovedTask 实现 PendingHandlerCallback 抽象类，用于回调移除 ChannelHandler 节点。代码如下：


```
RemovedTask extends PendingHandlerCallback {

ractChannelHandlerContext ctx) {

    }

    @Override
    void execute() {
        EventExecutor executor = ctx.executor();
        // 在 EventLoop 的线程中，回调 ChannelHandler removed 事件
        if (executor.inEventLoop()) {
            callHandlerRemoved0(ctx);
        } else {
            // 提交 EventLoop 中，执行回调 ChannelHandler removed 事件
            try {
                executor.execute(this); // <1>
            } catch (RejectedExecutionException e) {
                if (logger.isWarnEnabled()) {
                    logger.warn(
                            "Can't invoke handlerRemoved() as the EventExecutor {} rejected it," +
                                    " removing handler {}.", executor, ctx.name(), e);
                }
                // 标记 AbstractChannelHandlerContext 为已移除
                // remove0(...) was call before so just call AbstractChannelHandlerContext.setRemoved(
                ctx.setRemoved();
            }
        }
    }

}
```

- 在 `#execute()` 实现方法中，我们可以看到，和 `#remove((AbstractChannelHandlerContext ctx)` 方法的【第 17 至 32 行】的代码比较类似，目的是，在 EventLoop 的线程中，执行 `#callHandlerRemoved0(AbstractChannelHandlerContext)` 方法，回调 ChannelHandler 移除完成(removed)事件。

- `<1>` 处，为什么 PendingHandlerRemovedTask 可以直接提交到 EventLoop 中呢？因为 PendingHandlerRemovedTask 是个 Runnable ， 这也就是为什么 PendingHandlerCallback 实现 Runnable 接口的原因。

666. 彩蛋

水文一小篇。推荐阅读文章：

- 闪电侠 《Netty 源码分析之 pipeline(一)》

量 次