• NETTY

精尽 Netty 源码解析 —— ChannelHandler（一）之简介

1. 概述

在《精尽 Netty 源码分析 —— Netty 简介（二）之核心组件》中，对 ChannelHandler 做了定义，我们再来回顾下：

> ChannelHandler ，连接通道处理器，我们使用 Netty 中最常用的组件。ChannelHandler 主要用来处理各种事件，这里的事件很广泛，比如可以是连接、数据接收、异常、数据转换等。

文章目录

ChannelHandler 的身影，已经是熟悉的老朋友了。当然，我们还是会在这个
认识 ChannelHandler 。

annel 处理器接口。代码如下：

```
annelHandler} was added to the actual context and it's ready to
 *
 * ChannelHandler 已经成功被添加到 ChannelPipeline 中，可以进行处理事件。
 *
 * 该方法，一般用于 ChannelHandler 的初始化的逻辑
 */
void handlerAdded(ChannelHandlerContext ctx) throws Exception;

/**
 * Gets called after the {@link ChannelHandler} was removed from the actual context and it doesn't
 * anymore.
 *
 * ChannelHandler 已经成功从 ChannelPipeline 中被移除，不再进行处理事件。
 *
 * 该方法，一般用于 ChannelHandler 的销毁的逻辑
 */
void handlerRemoved(ChannelHandlerContext ctx) throws Exception;

/**
 * Gets called if a {@link Throwable} was thrown.
```

```
     *
     * 抓取到异常。目前被废弃，移到 ChannelInboundHandler 接口中，作为对 Exception Inbound 事件的处理
     *
     * @deprecated is part of {@link ChannelInboundHandler}
     */
    @Deprecated
    void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception;

    /**
     * Indicates that the same instance of the annotated {@link ChannelHandler}
     * can be added to one or more {@link ChannelPipeline}s multiple times
     * without a race condition.
     * <p>
     * If this annotation is not specified, you have to create a new handler
     * instance every time you add it to a pipeline because it has unshared
     * state such as member variables.
     * <p>
     * This annotation is provided for documentation purpose, just like
     * <a href="http://www.javaconcurrencyinpractice.com/annotations/doc/">the JCIP annotations</a>.
     */
    @Inherited
    @Documented
    @Target(ElementType.TYPE)
                                              E)
```

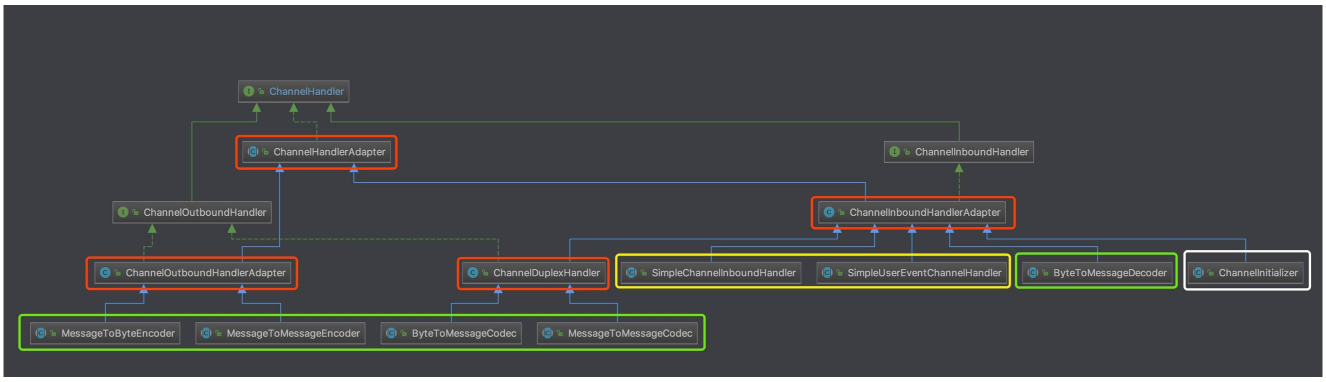
文章目录

emoved(...) 、 #exceptionCaught(...) 方法，胖友看方法上的注

共享，即是否可以被**多次**添加。在《精尽 Netty 源码解析 —— ChannelPipeline（二）之添加 ChannelHandler》的「3. checkMultiplicity」小节，已经有详细解析。

2. 核心类

ChannelHandler 的**核心类**的类图如下图：



- ChannelInboundHandler，在《精尽 Netty 源码解析 —— ChannelPipeline（五）之 Inbound 事件的传播》有详细解析。

- ChannelOutboundHandler ，在 《精尽 Netty 源码解析 —— ChannelPipeline（六）之 Outbound 事件的传播》 有详细解析。
- **红框**部分，ChannelHandler Adaptive 实现类，提供默认的骨架(Skeleton)实现。
- **绿框**部分，用于编解码消息的 ChannelHandler 实现类。关于这部分，我们会在 《Codec》专属的章节，而不是在《ChannelHandler》章节。
- **黄框**部分
  - SimpleChannelInboundHandler ，抽象类，处理**指定类型**的消息。应用程序中，我们可以实现 SimpleChannelInboundHandler 后，实现对**指定类型**的消息的自定义处理。
  - Simple**UserEvent**ChannelHandler ，和 SimpleChannelInboundHandler 基本一致，差别在于将指定类型的消息，改成了制定类型的事件。
  - 详细解析，见 《精尽 Netty 源码解析 —— ChannelHandler（三）之 SimpleChannelInboundHandler》 。
- ChannelInitializer ，一个**特殊**的 ChannelHandler ，用于 Channel 注册到 EventLoop 后，**执行自定义的初始化操作**。一般情况下，初始化自定义的 ChannelHandler 到 Channel 中。详细解析，见 《精尽 Netty 源码解析 —— ChannelHandler（二）之 ChannelInitializer》 。

3. ChannelHandlerAdaptive

在看看 ChannelHandlerAdaptive 的具体代码实现之前，我们先一起了解 ChannelHandlerAdaptive 的设计思想。在《Netty 权威指南》如是说：

> ler 会选择性地拦截和处理某个或者某些事件，其他
> annelHandler 进行拦截和处理。这就会导致一个问
> 必须要实现 ChannelHandler 的所有接口，包括它不
> 这会导致用户代码的冗余和臃肿，代码的可维护性也
>
> 供了ChannelHandlerAdapter基类，它的所有接口
> 户ChannelHandler关心某个事件，只需要覆盖
> 应的方法即可，对于不关心的，可以直接继承使用父
> 类的方法，这样子类的代码就会非常简洁和清晰。

文章目录

😈 下面，我们看到的其它 Adaptive 实现类，也是这样的设计思想。

`io.netty.channel.ChannelHandlerAdapter` ，实现 ChannelHandler 接口，ChannelHandler Adapter 抽象类。

3.1 isSharable

```
// Not using volatile because it's used only for a sanity check.
/**
 * 是否已经初始化
 */
boolean added;

/**
 * Throws {@link IllegalStateException} if {@link ChannelHandlerAdapter#isSharable()} returns {@code t
 */
```

```
protected void ensureNotSharable() {
    if (isSharable()) {
        throw new IllegalStateException("ChannelHandler " + getClass().getName() + " is not allowed to
    }
}

/**
 * Return {@code true} if the implementation is {@link Sharable} and so can be added
 * to different {@link ChannelPipeline}s.
 */
public boolean isSharable() {
    /**
     * Cache the result of {@link Sharable} annotation detection to workaround a condition. We use a
     * {@link ThreadLocal} and {@link WeakHashMap} to eliminate the volatile write/reads. Using differ
     * {@link WeakHashMap} instances per {@link Thread} is good enough for us and the number of
     * {@link Thread}s are quite limited anyway.
     *
     * See <a href="https://github.com/netty/netty/issues/2289">#2289</a>.
     */
    Class<?> clazz = getClass();
    Map<Class<?>, Boolean> cache = InternalThreadLocalMap.get().handlerSharableCache();
    Boolean sharable = cache.get(clazz);
    if (sharable == null) {
                                        Present(Sharable.class);
```

文章目录

《精尽 Netty 源码解析 —— ChannelPipeline（二）之添加
》小节，已经有详细解析。

```
/**
 * Do nothing by default, sub-classes may override this method.
 */
@Override
public void handlerAdded(ChannelHandlerContext ctx) throws Exception {
    // NOOP
}

/**
 * Do nothing by default, sub-classes may override this method.
 */
@Override
public void handlerRemoved(ChannelHandlerContext ctx) throws Exception {
    // NOOP
}

/**
 * Calls {@link ChannelHandlerContext#fireExceptionCaught(Throwable)} to forward
 * to the next {@link ChannelHandler} in the {@link ChannelPipeline}.
 *
 * Sub-classes may override this method to change behavior.
 */
@Override
public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
    ctx.fireExceptionCaught(cause);
}
```

- 对于 `#handlerAdded(ChannelHandlerContext ctx)` 和 `#handlerRemoved(ChannelHandlerContext ctx)` 方法，默认无任何逻辑。子类如果有自定义的逻辑，可以进行覆写对应的方法。
- `#exceptionCaught(ChannelHandlerContext ctx, Throwable cause)` 方法，直接转发到下一个节点，**实际上**也是默认无任何逻辑。子类如果有自定义的逻辑，可以进行覆写对应的方法。

4. ChannelOutboundHandlerAdapter

`io.netty.channel.ChannelOutboundHandlerAdapter` ，实现 ChannelOutboundHandler 接口，继承 ChannelHandlerAdapter 抽象类，ChannelOutboundHandler Adapter 实现类。代码如下：

```
public class ChannelOutboundHandlerAdapter extends ChannelHandlerAdapter implements ChannelOutboundHan
```

文章目录

```
ext#bind(SocketAddress, ChannelPromise)} to forward
oundHandler} in the {@link ChannelPipeline}.

method to change behavior.

ntext ctx, SocketAddress localAddress, ChannelPromise promise) th
);

    /**
     * Calls {@link ChannelHandlerContext#connect(SocketAddress, SocketAddress, ChannelPromise)} to fo
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void connect(ChannelHandlerContext ctx, SocketAddress remoteAddress, SocketAddress localAdd
        ctx.connect(remoteAddress, localAddress, promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#disconnect(ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void disconnect(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        ctx.disconnect(promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#close(ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void close(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        ctx.close(promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#deregister(ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void deregister(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        ctx.deregister(promise);
    }
```

文章目录

```
                                              ext#read()} to forward
                                              oundHandler} in the {@link ChannelPipeline}.

                                              ethod to change behavior.

                                              text ctx) throws Exception {



    /**
     * Calls {@link ChannelHandlerContext#write(Object, ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception
        ctx.write(msg, promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#flush()} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void flush(ChannelHandlerContext ctx) throws Exception {
        ctx.flush();
    }

}
```

- 每个实现方法，直接转发到下一个节点，**实际上**也是默认无任何逻辑。子类如果有自定义的逻辑，可以进行覆写对应的方法。

5. ChannelInboundHandlerAdapter

`io.netty.channel.ChannelInboundHandlerAdapter` ，实现 ChannelInboundHandler 接口，继承 ChannelHandlerAdapter 抽象类，ChannelInboundHandler Adapter 实现类。代码如下：

```
public class ChannelInboundHandlerAdapter extends ChannelHandlerAdapter implements ChannelInboundHandl

    /**
     * Calls {@link ChannelHandlerContext#fireChannelRegistered()} to forward
     * to the next {@link ChannelInboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.

                                                 nelHandlerContext ctx) throws Exception {




                                                 ext#fireChannelUnregistered()} to forward
                                                 ndHandler} in the {@link ChannelPipeline}.

                                                 ethod to change behavior.
```

文章目录

```
    @Override
    public void channelUnregistered(ChannelHandlerContext ctx) throws Exception {
        ctx.fireChannelUnregistered();
    }

    /**
     * Calls {@link ChannelHandlerContext#fireChannelActive()} to forward
     * to the next {@link ChannelInboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
        ctx.fireChannelActive();
    }

    /**
     * Calls {@link ChannelHandlerContext#fireChannelInactive()} to forward
     * to the next {@link ChannelInboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void channelInactive(ChannelHandlerContext ctx) throws Exception {
        ctx.fireChannelInactive();
    }

    /**
     * Calls {@link ChannelHandlerContext#fireChannelRead(Object)} to forward
     * to the next {@link ChannelInboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        ctx.fireChannelRead(msg);
    }

    /**
     * Calls {@link ChannelHandlerContext#fireChannelReadComplete()} to forward
     * to the next {@link ChannelInboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
```

文章目录

```
                                                annelHandlerContext ctx) throws Exception {
                                                ;



                                                ext#fireUserEventTriggered(Object)} to forward
                                                ndHandler} in the {@link ChannelPipeline}.

                                                ethod to change behavior.

    @Override
    public void userEventTriggered(ChannelHandlerContext ctx, Object evt) throws Exception {
        ctx.fireUserEventTriggered(evt);
    }

    /**
     * Calls {@link ChannelHandlerContext#fireChannelWritabilityChanged()} to forward
     * to the next {@link ChannelInboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void channelWritabilityChanged(ChannelHandlerContext ctx) throws Exception {
        ctx.fireChannelWritabilityChanged();
    }

    /**
     * Calls {@link ChannelHandlerContext#fireExceptionCaught(Throwable)} to forward
     * to the next {@link ChannelHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
        ctx.fireExceptionCaught(cause);
    }

}
```

- 每个实现方法，直接转发到下一个节点，**实际上**也是默认无任何逻辑。子类如果有自定义的逻辑，可以进行覆写对应的方法。

6. ChannelDuplexHandler

`io.netty.channel.ChannelDuplexHandler` ，实现 ChannelOutboundHandler 接口，继承 ChannelInboundHandlerAdapter 抽象类，Channel Duplex Handler 实现类，支持对 Inbound 和 Outbound 事件的 Adaptive 处理，所以命名上带有“**Duplex**”(双重)。代码如下：

```
public class ChannelDuplexHandler extends ChannelInboundHandlerAdapter implements ChannelOutboundHandl
```

```
                                                    text#bind(SocketAddress, ChannelPromise)} to forward
                                                    oundHandler} in the {@link ChannelPipeline}.

                                                    method to change behavior.


                                                    ntext ctx, SocketAddress localAddress, ChannelPromise promise) th
                                                    e);


    /**
     * Calls {@link ChannelHandlerContext#connect(SocketAddress, SocketAddress, ChannelPromise)} to fo
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void connect(ChannelHandlerContext ctx, SocketAddress remoteAddress, SocketAddress localAdd
        ctx.connect(remoteAddress, localAddress, promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#disconnect(ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void disconnect(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        ctx.disconnect(promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#close(ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void close(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        ctx.close(promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#close(ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void deregister(ChannelHandlerContext ctx, ChannelPromise promise) throws Exception {
        ctx.deregister(promise);
    }
```

文章目录

```
                                   text#read()} to forward
                                   undHandler} in the {@link ChannelPipeline}.

                                   ethod to change behavior.

                                   ntext ctx) throws Exception {
```

```
    /**
     * Calls {@link ChannelHandlerContext#write(Object, ChannelPromise)} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception
        ctx.write(msg, promise);
    }

    /**
     * Calls {@link ChannelHandlerContext#flush()} to forward
     * to the next {@link ChannelOutboundHandler} in the {@link ChannelPipeline}.
     *
     * Sub-classes may override this method to change behavior.
     */
    @Override
    public void flush(ChannelHandlerContext ctx) throws Exception {
        ctx.flush();
    }

}
```

- 实现代码上，和 「4. ChannelOutboundHandlerAdapter」 是一致的。因为 Java 不支持**多继承**的特性，所以不得又重新实现一遍。

👿 大多数情况下，我们会实现 ChannelDuplexHandler 类，覆写部分方法，处理对应的事件。

666. 彩蛋

小小水文一篇，主要帮胖友梳理下，对 ChannelHandler 有整体的认识。在后续的文章中，我们会看具体的一个一个 ChannelHandler 的带有“业务”的实现类。

推荐阅读如下文章：

- Hypercube 《自顶向下深入分析Netty（八）–ChannelHandler》

文章目录