07 如何优雅地编写Netty应用程序

更新时间：2020-09-04 11:46:35

> 人生的旅途，前途很远，也很暗。然而不要怕，不怕的人的面前才有路。—— 鲁 迅

前言

你好，我是彤哥。

上一节，我们一起从架构设计和模块设计两个方面分析了 Netty 的整体结构，有了上一节的学习，相信你一定知道从哪里“抄”代码了，并且能够“抄”出一份不错的 Netty 使用案例了。

不过，抄归抄，我们还是要大致了解一下编写 Netty 的常用步骤，笔者总结了一下，大概可以分为十大步骤。

所以，本节我们一起来学习下如何使用 Netty 编写简单的示例程序，并在文章的最后手写一个简单的群聊系统，同学们可以将 Netty 的群聊系统跟 Java NIO 的群聊系统进行对比。

好了，让我们开始今天的学习吧。

Netty 编码十步曲

我从 netty-example 工程下抄了一份 EchoServer 过来，并删减了部分代码，归纳出来一份编写 Netty 服务端程序的模板，我把它称为“Netty 编码十步曲”。

1. 声明线程池（必须）

一般来说，我们会声明两个 Group，一个是 bossGroup，用于处理 Accept 事件，一个是 workerGroup，用于处理消息的读写事件。其中，bossGroup 一般声明为一个线程。当然，如果声明一个 Group 也是可以的，只是不建议。

```
EventLoopGroup bossGroup = new NioEventLoopGroup(1);
EventLoopGroup workerGroup = new NioEventLoopGroup();
```

这就像是大型餐厅一般有接待生和服务员两种职位一样，接待生一般形象良好，专门负责接待客人，服务员形象稍差，专门负责上菜收碟，你要说不区分两种职位，混用行不行呢，当然也可以，只是不建议，专人干专事。

2. 创建服务端引导类（必须）

引导类，是用来集成所有配置，引导程序加载的，分成两种，一种是客户端引导类 Bootstrap（个人觉得叫 ClientBootstrap 可能更贴切），另一种是服务端引导类 ServerBootstrap，我们这里编写的是服务端程序，创建的当然是服务端引导类。
注意，Bootstrap 和 ServerBootstrap 之间并不是继承关系，他们是平等的，都继承了 AbstractBootstrap 抽象类。

```
ServerBootstrap serverBootstrap = new ServerBootstrap();
```

Bootstrap/ServerBootstrap 就像是店长，它负责统筹整个 Netty 程序的正常运行。

3. 设置线程池（必须）

把第一步声明的线程池设置到 ServerBootstrap 中，它说明了 Netty 应用程序以什么样的线程模型运行，正如前面所说 bossGroup 负责接受（Accept）连接，workerGroup 负责读写数据。

```
serverBootstrap.group(bossGroup, workerGroup);
```

4. 设置 ServerSocketChannel 类型（必须）

设置 Netty 程序以什么样的 IO 模型运行，我们这里介绍的是 NIO 编程，选择的当然是 NioServerSocketChannel。

```
serverBootstrap.channel(NioServerSocketChannel.class);
```

如果您需要使用阻塞型 IO 模型，直接把 Nio 改成 Oio 就可以了，即 OioServerSocketChannel，不过它已经废弃了，所以不建议。

另外，如果您的程序运行在 Linux 系统上，还可以使用一种更高效的方式，即 EpollServerSocketChannel，它使用的是 Linux 系统上的 epoll 模型，比 select 模型更高效，可见 Netty 把性能优化做到了极致。

5. 设置参数（可选）

设置 Netty 中可以使用的任何参数，这些参数都在 ChannelOption 及其子类中，后面我们会详细介绍各个参数的含义，不过，很遗憾地告诉你，大多数情况下并不需要修改 Netty 的默认参数，这就是 Netty 比较牛的地方。

```
serverBootstrap.option(ChannelOption.SO_BACKLOG, 100);
```

我们这里设置了一个 SO_BACKLOG 系统参数，它表示的是最大等待连接数量。

6. 设置 Handler（可选）

设置 ServerSocketChannel 对应的 Handler，注意只能设置一个，它会在 SocketChannel 建立起来之前执行，等我们看源码的时候会详细介绍它的执行时机。

```
serverBootstrap.handler(new LoggingHandler(LogLevel.INFO))
```

我们这里简单地设置一个打印日志的 Handler。

7. 编写并设置子 Handler（必须）

Netty 中的 Handler 分成两种，一种叫做 Inbound，一种叫做 Outbound。我们这里简单地写一个 Inbound 类型的 Handler，它接收到数据后立即写回客户端。

```
public class EchoServerHandler extends ChannelInboundHandlerAdapter {

 @Override
 public void channelRead(ChannelHandlerContext ctx, Object msg) {
  // 读取数据后写回客户端
  ctx.write(msg);
 }

 @Override
 public void channelReadComplete(ChannelHandlerContext ctx) {
  ctx.flush();
 }

 @Override
 public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
  cause.printStackTrace();
  ctx.close();
 }
}
```

设置子 Handler 设置的是 SocketChannel 对应的 Handler，注意也是只能设置一个，它用于处理 SocketChannel 的事件。

```
serverBootstrap.childHandler(new ChannelInitializer<SocketChannel>() {
 @Override
 public void initChannel(SocketChannel ch) throws Exception {
  ChannelPipeline p = ch.pipeline();
  // 可以添加多个子Handler
  p.addLast(new LoggingHandler(LogLevel.INFO));
  p.addLast(new EchoServerHandler());
 }
});
```

虽然只能设置一个，但是 Netty 的提供了一种可以设置多个 Handler 的途径，即使用 ChannelInitializer 方式，当然，第六步的设置 Handler 也可以使用这种方式设置多个 Handler。

这里，我们设置了一个打印的 Handler 和一个自定义的 EchoServerHandler。

8. 绑定端口（必须）

绑定端口，并启动服务端程序，sync () 会阻塞直到启动完成才执行后面的代码。

```
ChannelFuture f = serverBootstrap.bind(PORT).sync();
```

9. 等待服务端端口关闭（必须）

等待服务端监听端口关闭，sync () 会阻塞主线程，内部调用的是 Object 的 wait () 方法。

```
f.channel().closeFuture().sync();
```

10. 优雅地关闭线程池（建议）

最后，是在 finally 中调用 shutdownGracefully () 方法优雅地关闭线程池，优雅停机。

```
bossGroup.shutdownGracefully();
workerGroup.shutdownGracefully();
```

好了，Netty 编码十步曲介绍完毕了，不知道你有没有什么疑问呢？至少我是有的。

为什么需要设置 ServerSocketChannel 的类型，而不需要设置 SocketChannel 的类型呢？

那是因为 SocketChannel 是 ServerSocketChannel 在接受连接之后创建出来的，所以，并不需要单独再设置它的类型，比如，NioServerSocketChannel 创建出来的肯定是 NioSocketChannel，而 EpollServerSocketChannel 创建出来的肯定是 EpollSocketChannel。

你还有什么疑问呢？欢迎留言。

如何调试

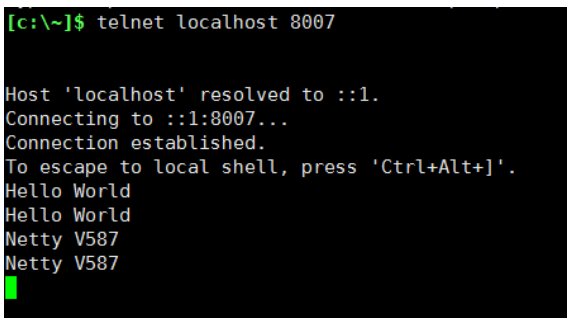
其实学 Netty 的 99% 都是服务端的同学，所以，我们的课程并不会刻意介绍如何编写客户端的 Netty 程序，但是我们怎么调试呢？

这里，我教给大家一个技巧，通过 XSHELL 这个工具调试，这个工具几乎是后端同学必备的一个工具，所以调试起来也是比较容易的。

比如，我上面启动了一个 Netty 服务端，它的端口是 8007，只要打开 XSHELL，不要连接任何服务器，输入以下代码即可连接到我们的 Netty 服务端：

```
telnet localhost 8007
```

然后，输入任何你想输入的内容，它都会照样返回，比如下面这样：

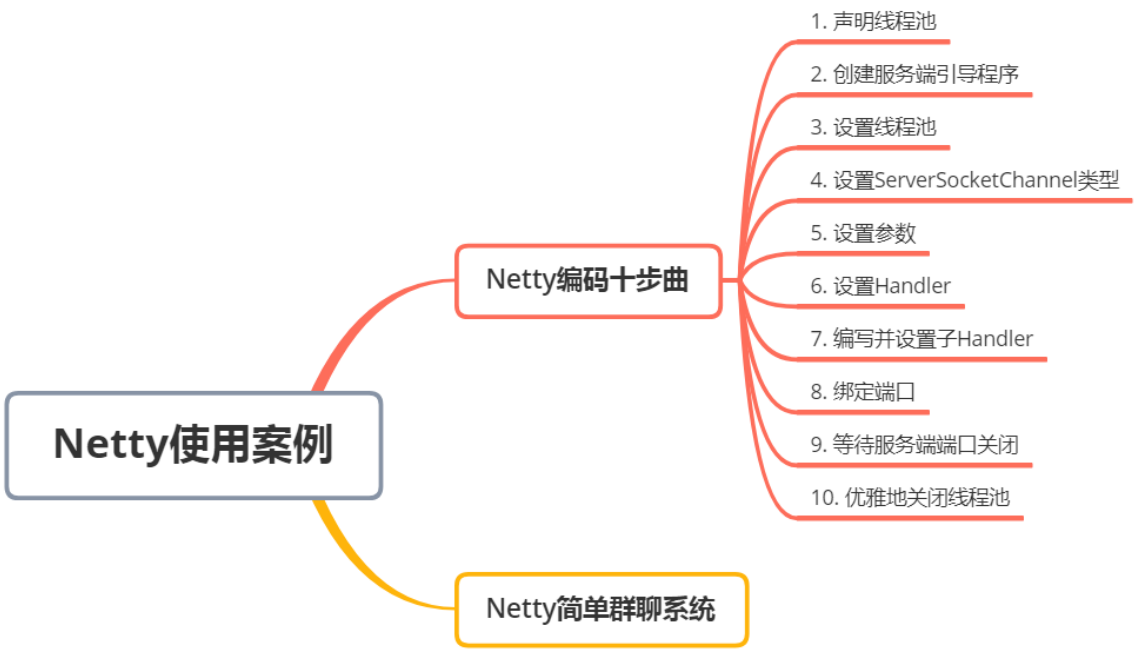


后记

本节，我们学习了 Netty 编码的十步曲，其中，有些步骤是必须的，有些步骤可选的，有些步骤是建议保留的，相信通过本节的学习，你一定可以写出十分健壮且优雅的 Netty 服务端程序了。

别急哦，本节还没有结束，在附录部分有一份简单的群聊系统，您也可以尝试按照本节介绍的十步曲自己尝试写一个简单的示例练练手。

思维导图



附录 —— 使用 Netty 实现简单群聊系统

代码

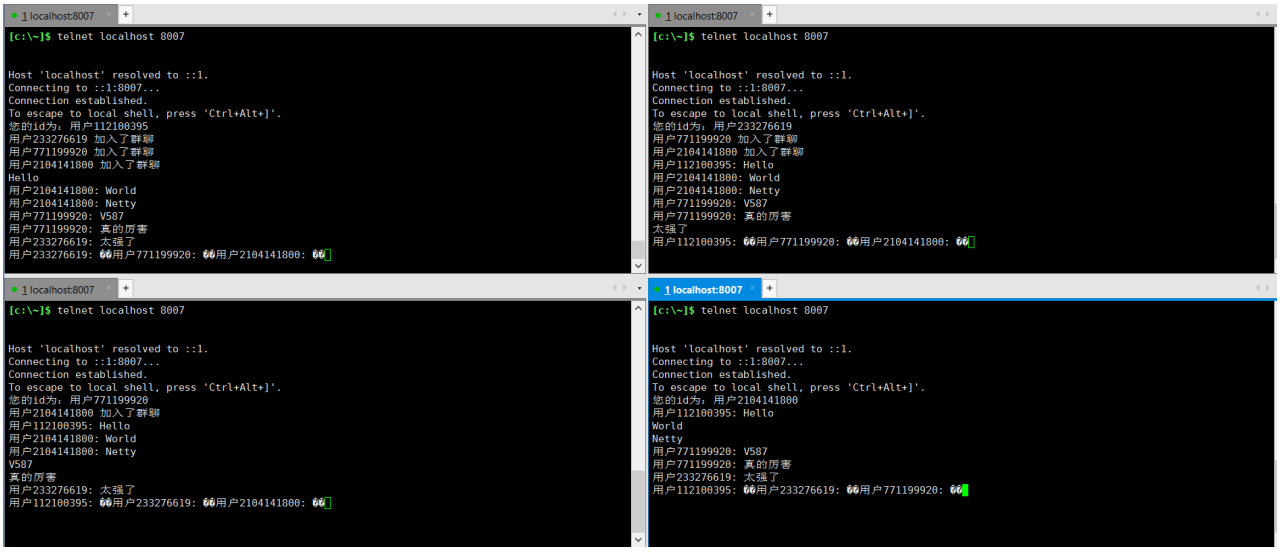
```
public final class NettyChatServer {

 static final int PORT = Integer.parseInt(System.getProperty("port", "8007"));

 public static void main(String[] args) throws Exception {
  // 1. 声明线程池
  EventLoopGroup bossGroup = new NioEventLoopGroup(1);
  EventLoopGroup workerGroup = new NioEventLoopGroup();
  try {
   // 2. 服务端引导器
   ServerBootstrap serverBootstrap = new ServerBootstrap();
   // 3. 设置线程池
   serverBootstrap.group(bossGroup, workerGroup)
     // 4. 设置ServerSocketChannel的类型
     .channel(NioServerSocketChannel.class)
     // 5. 设置参数
     .option(ChannelOption.SO_BACKLOG, 100)
     // 6. 设置ServerSocketChannel对应的Handler，只能设置一个
     .handler(new LoggingHandler(LogLevel.INFO))
     // 7. 设置SocketChannel对应的Handler
     .childHandler(new ChannelInitializer<SocketChannel>() {
      @Override
      public void initChannel(SocketChannel ch) throws Exception {
       ChannelPipeline p = ch.pipeline();
       // 可以添加多个子Handler
```

```
        p.addLast(new LoggingHandler(LogLevel.INFO));
        // 只需要改这一个地方就可以了
        p.addLast(new ChatNettyHandler());
       }
      });

     // 8. 绑定端口
     ChannelFuture f = serverBootstrap.bind(PORT).sync();
     // 9. 等待服务端监听端口关闭，这里会阻塞主线程
     f.channel().closeFuture().sync();
    } finally {
     // 10. 优雅地关闭两个线程池
     bossGroup.shutdownGracefully();
     workerGroup.shutdownGracefully();
    }
   }

   private static class ChatNettyHandler extends SimpleChannelInboundHandler<ByteBuf> {

    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
     System.out.println("one conn active: " + ctx.channel());
     // channel是在ServerBootstrapAcceptor中放到EventLoopGroup中的
     ChatHolder.join((SocketChannel) ctx.channel());
    }

    @Override
    protected void channelRead0(ChannelHandlerContext ctx, ByteBuf byteBuf) throws Exception {
     byte[] bytes = new byte[byteBuf.readableBytes()];
     byteBuf.readBytes(bytes);
     String content = new String(bytes, StandardCharsets.UTF_8);
     System.out.println(content);

     if (content.equals("quit\r\n")) {
      ctx.channel().close();
     } else {
      ChatHolder.propagate((SocketChannel) ctx.channel(), content);
     }
    }

    @Override
    public void channelInactive(ChannelHandlerContext ctx) throws Exception {
     System.out.println("one conn inactive: " + ctx.channel());
     ChatHolder.quit((SocketChannel) ctx.channel());
    }
   }

   private static class ChatHolder {
    static final Map<SocketChannel, String> USER_MAP = new ConcurrentHashMap<>();

    /**
     * 加入群聊
     *
     * @param socketChannel
     */
    static void join(SocketChannel socketChannel) {
     // 有人加入就给他分配一个id
     String userId = "用户" + ThreadLocalRandom.current().nextInt(Integer.MAX_VALUE);
     send(socketChannel, "您的id为：" + userId + "\n\r");

     for (SocketChannel channel : USER_MAP.keySet()) {
      send(channel, userId + " 加入了群聊" + "\n\r");
     }

     // 将当前用户加入到map中
     USER_MAP.put(socketChannel, userId);
    }

    /**
```

```
     * 退出群聊
     *
     * @param socketChannel
     */
    static void quit(SocketChannel socketChannel) {
     String userId = USER_MAP.get(socketChannel);
     send(socketChannel, "您退出了群聊" + "\n\r");
     USER_MAP.remove(socketChannel);

     for (SocketChannel channel : USER_MAP.keySet()) {
      if (channel != socketChannel) {
       send(channel, userId + " 退出了群聊" + "\n\r");
      }
     }
    }

    /**
     * 扩散说话的内容
     *
     * @param socketChannel
     * @param content
     */
    public static void propagate(SocketChannel socketChannel, String content) {
     String userId = USER_MAP.get(socketChannel);
     for (SocketChannel channel : USER_MAP.keySet()) {
      if (channel != socketChannel) {
       send(channel, userId + ": " + content);
      }
     }
    }

    /**
     * 发送消息
     *
     * @param socketChannel
     * @param msg
     */
    static void send(SocketChannel socketChannel, String msg) {
     try {
      ByteBufAllocator allocator = ByteBufAllocator.DEFAULT;
      ByteBuf writeBuffer = allocator.buffer(msg.getBytes().length);
      writeBuffer.writeCharSequence(msg, Charset.defaultCharset());
      socketChannel.writeAndFlush(writeBuffer);
     } catch (Exception e) {
      e.printStackTrace();
     }
    }
   }
```

可以发现，只需要改动设置子 Handler 里面的那一个地方就可以了，其它地方完全不需要修改，非常便捷。

模拟群聊



最后一行出现乱码，这个不用管，它是 XSHELL 本身发出的心跳，我们的代码中并没有处理 XSHELL 的心跳，所以显示了一些奇怪的字符。

问题

通过简单群聊系统，相信你一定被 Netty 的魅力所折服，然而，你可能会发现一个问题，上面的简单群聊系统只能支持所有人一起聊天，那么，我提出一个问题，你可以尝试完成，如何将上述简单群聊系统修改为真正的类似微信的聊天系统？

}

06 如何从整体上把握Netty的架构

08 Netty的核心组件有哪些