链滴

我的 Netty 学习之路

作者：xjtushilei

原文链接：https://ld246.com/article/1527824544136

来源网站：链滴

前言

在博客的_drafts里看到了一篇2017-02-17-我的java存在深入浅出.md,那个时候在完成了一定量的编之后，感觉还是想提高自己，于是就开始了阅读。可能以为阅读完几本书就够了，什么JVM，并发，程宝典之类的。现在看起来一年多前的自己还是比较年轻的，即便现在也是很年轻。

今天偶然看到了akka这么一个很厉害的高并发分布式框架，于是想学习一下，发现他和netty有千丝缕的联系。Akka针对IO操作有一个抽象，这和netty是一样的。使用Akka可以用来创建计算集群,Acto在不同的机器之间传递消息。从这个角度来看,Akka相对于Netty来说，是一个更高层次的抽象。但aka用Scala写的，不太熟，估计会看晕。

然而，发现自己的netty属于半知半解的状态，没看过源码，没了解架构，没写过深层的代码，所以篇博客才来了。

学习过程

学习过程主要围绕 gitbookEssential Netty in Action 《Netty 实战(精髓)》来进行。下面仅仅是记一些笔记，方便记忆和回忆。同时提供一些总结性的东西，或者提供一些原书没有的东西。

1. echo server 的编码

整体流程

1. 写一个服务端
2. 写一个客户端
3. 启动服务端，客户端进行调用

服务端任务

1. 创建 ServerBootstrap 实例来引导服务器并随后绑定
2. 创建并分配一个 NioEventLoopGroup 实例来处理事件的处理，如接受新的连接和读/写数据。
3. 指定本地 InetSocketAddress 给服务器绑定
4. 通过 EchoServerHandler 实例给每一个新的 Channel 初始化
5. 最后调用 ServerBootstrap.bind() 绑定服务器

客户端任务

1. 连接服务器
2. 发送信息
3. 发送的每个信息，等待和接收从服务器返回的同样的信息
4. 关闭连接

具体来说就是

1. 一个 Bootstrap 被创建来初始化客户端

2. 一个 NioEventLoopGroup 实例被分配给处理该事件的处理，这包括创建新的连接和处理入站和站数据

3. 一个 InetSocketAddress 为连接到服务器而创建

4. 一个 EchoClientHandler 将被安装在 pipeline 当连接完成时

5. 之后 Bootstrap.connect（）被调用连接到远程的 - 本例就是 echo(回声)服务器。

代码放在https://github.com/xjtushilei/netty-demo/tree/master/src/main/java/echo，还是比好理解的。首先写一个netty的基本通讯流程来理解过程，然后下面的章节来理解为什么。

2. 主要概念学习

- BOOTSTRAP
- CHANNEL
- CHANNELHANDLER
- CHANNELPIPELINE
- EVENTLOOP
- CHANNELFUTURE

详细介绍简书传送门，有图片加深理解与记忆。

下面将其他的一些核心的组件。

Buffer API

主要包括

- ByteBuf
- ByteBufHolder

Netty 使用reference-counting(**引用计数**)来判断何时可以释放 ByteBuf 或 ByteBufHolder 和其他关资源，从而可以利用池和其他技巧来提高性能和降低内存的消耗。这一点上不需要开发人员做任何情，但是在开发 Netty 应用程序时，尤其是使用 ByteBuf 和 ByteBufHolder 时，你应该尽可能早地放池资源。 Netty 缓冲 API 提供了几个优势：

- 可以自定义缓冲类型
- 通过一个内置的复合缓冲类型实现零拷贝
- 扩展性好，比如 StringBuilder
- 不需要调用 flip() 来切换读/写模式
- 读取和写入索引分开
- 方法链
- 引用计数
- Pooling(池)

Channel

Channel 这一章主要有，Channel中大量用了buffer API，所以刚才了解buffer很重要。虽然有些细仅仅是知其意而看不懂实现。

- Channel
- ChannelHandler
- ChannePipeline
- ChannelHandlerContext

codec

- 编码器
- 解码器
- 编解码器

第一次接触到TCP黏包、拆包的问题。针对该问题主要有以下方法：

- （1）消息长度固定：累计读取到固定长度为LENGTH之后就认为读取到了一个完整的消息。然后计数器复位，重新开始读下一个数据报文。
- （2）将特殊的分隔符作为消息的结束标志，回车换行符就是一种特殊的结束分隔符。
- （3）通过在消息头中定义长度字段来标示消息的总长度。

我们代码里采用的DelimiterBasedFrameDecoder就是（3）里面的。以 lineDelimiter，即换行符（linux和windows的换行符都可以）。关于这部分内容，我想专门开一篇博客来进行讲解：Netty之TC黏包与拆包

SSL

关于ssl这块，netty基于ssl的handler来实现的，并提供了默认的实现。但是我个人不推荐在netty层现ssl。当然了实现没问题，但是不推荐这么用。我浅显的认为，https等直接在网关层进行处理即可不需要再应用层进行ssl的处理。一旦连接从网关进入了内网，就是http进行传输了。这样应用层不用系加解密逻辑，只需要关心业务逻辑。

netty还提供了http的encoder和decoder：

- client: 添加 HttpResponseDecoder 用于处理来自 server 响应
- client: 添加 HttpRequestEncoder 用于发送请求到 server
- server: 添加 HttpRequestDecoder 用于接收来自 client 的请求
- server: 添加 HttpResponseEncoder 用来发送响应给 client

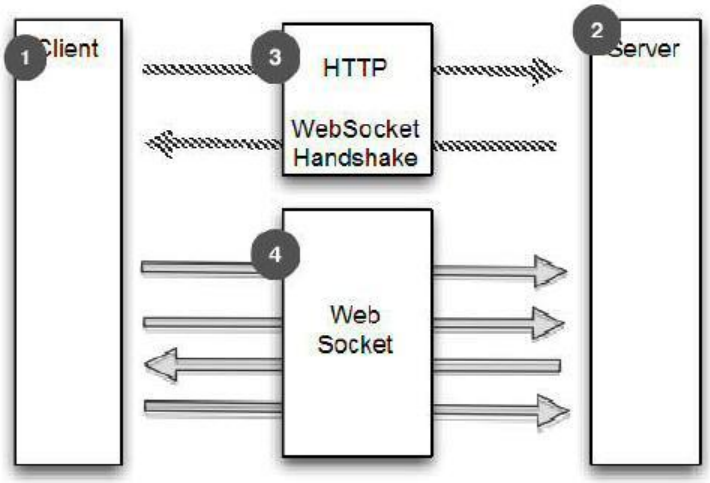
WebSocket

HTTP 是不错的协议，但是如果需要实时发布信息怎么做？有个做法就是客户端一直轮询请求服务器这种方式虽然可以达到目的，但是其缺点很多，也不是优秀的解决方案，为了解决这个问题，便出现了WebSocket。

WebSocket 允许数据双向传输，而不需要请求-响应模式。早期的WebSocket 只能发送文本数据，后现在不仅可以发送文本数据，也可以发送二进制数据，这使得可以使用 WebSocket 构建你想要的

序。下图是WebSocket 的通信示例图：

1. Client (HTTP) 与 Server 通讯
2. Server (HTTP) 与 Client 通讯
3. Client 通过 HTTP(s) 来进行 WebSocket 握手,并等待确认
4. 连接协议升级至 WebSocket

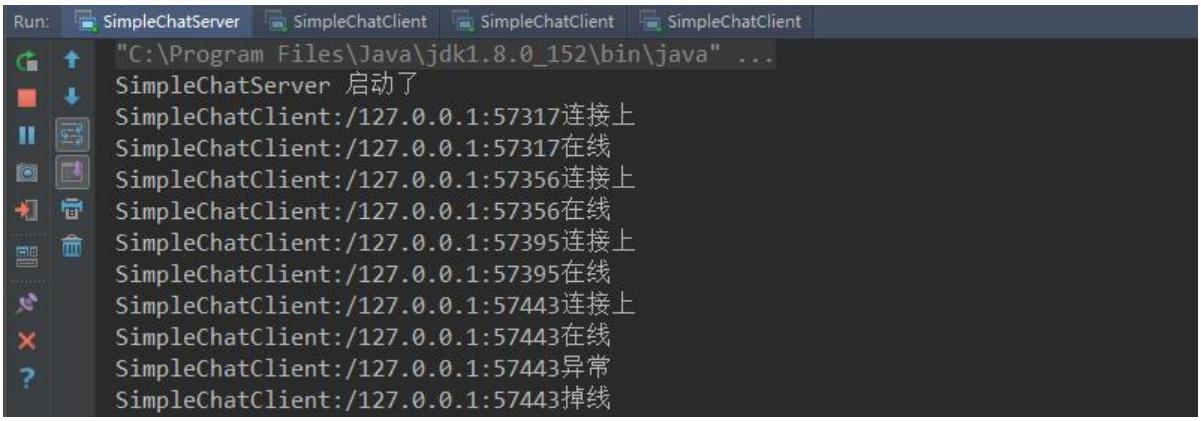


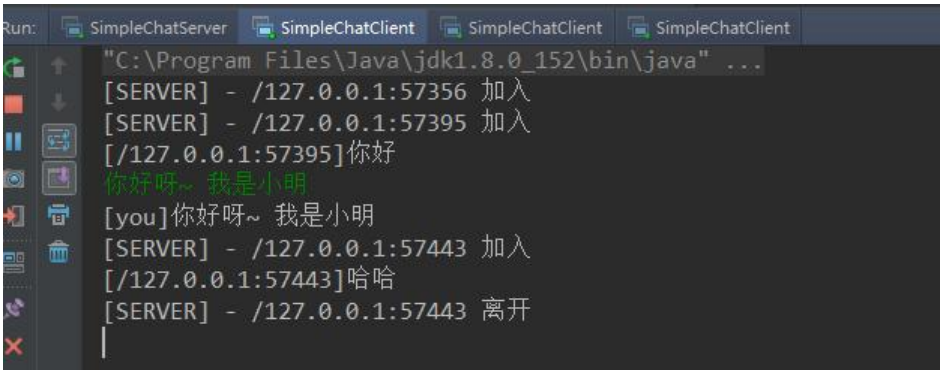
3. 简单多人聊天室

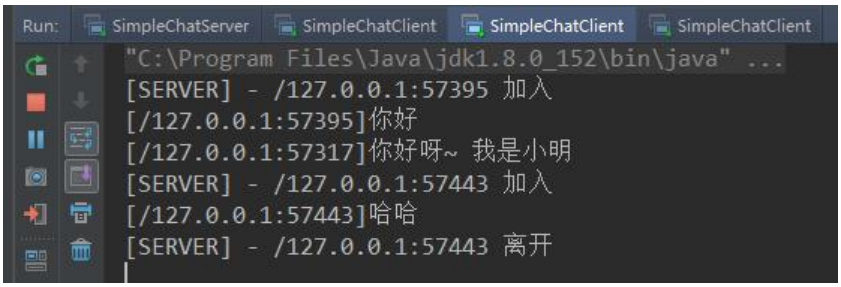
然后参考代码,整了一个simple聊天，感觉用netty写一个聊天室感觉很有意思。

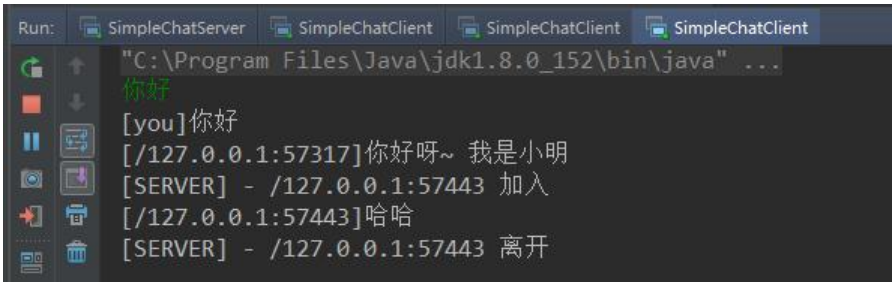
主要就是通过记录所有连接到服务器的channel，维护了这样的一个channel的list，每当有新用户连到服务器或者关闭库客户端，都动态的更新channel list。每当接收到一个channel的消息后，触发handler，转发消息给所有其他在线的handler。

效果图：









代码见：https://github.com/xjtushilei/netty-demo/tree/master/src/main/java/simplechart

其中需要注意的几个解释如下：

- sync()同步等待
- SO_BACKLOG服务端处理客户端连接请求是顺序处理的，所以同一时间只能处理一个客户端连接多个客户端来的时候，服务端将不能处理的客户端连接请求放在队列中等待处理，backlog参数指定队列的大小
- SO_KEEPALIVE参数对应于套接字选项中的SO_KEEPALIVE，该参数用于设置TCP连接，当设置该项以后，连接会测试链接的状态，这个选项用于可能长时间没有数据交流的连接。当设置该选项以后如果在两小时内没有数据的通信时，TCP会自动发送一个活动探测数据报文。

其他的ChannelOption参见：Netty之ChannelOption

- NioServerSocketChannel.class我们最常用的就是socket，除了tcp外，还有sctp协议的channel。

接下来

最近3天的零散时间，把netty基本的东西都了解了下，缺少实践。有以下几点可考虑，用netty写简单eb框架，用netty写简单rpc框架。有打算，很难付诸行动。

接下来准备看点实习中可能会用到的的吧，如果有机会还能跟大家一起分享。