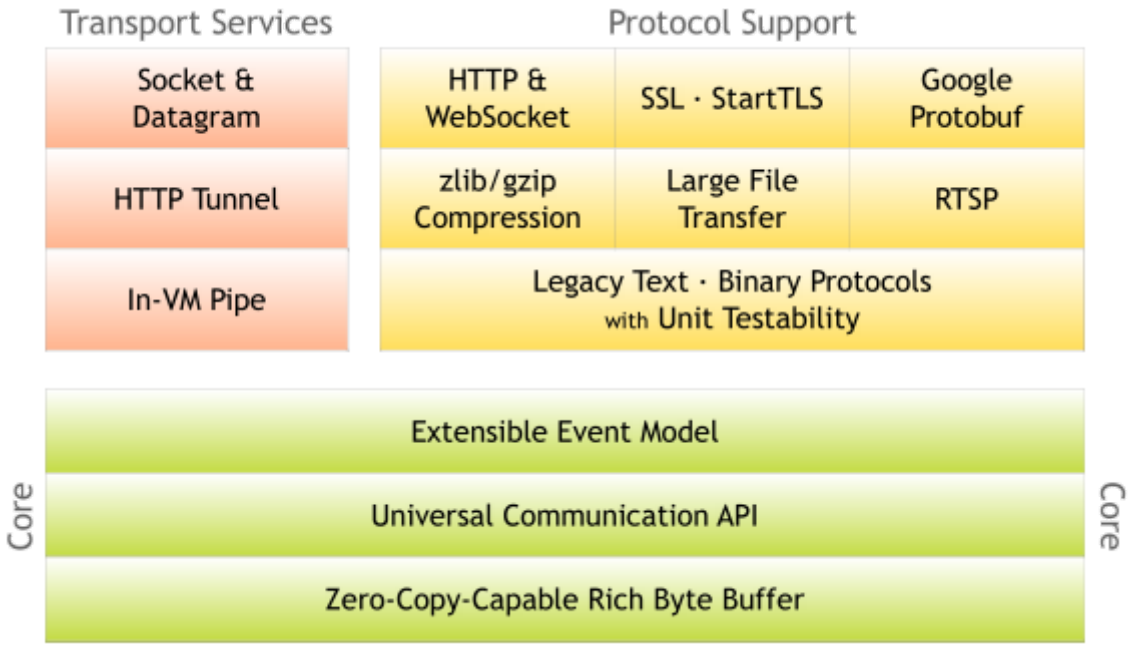
Netty初步理解和运用

netty是一个IO异步网络编程框架，因支持高并发连接、零拷贝技术，所以是一款高性能IO框架。

Netty架构图

附上官网的架构图：



从架构图看，大致分为三大块：Core、Transport Services、Protocol Support。

- Core：包含了零拷贝（性能高的一个方面）且有丰富操作的ByteBuffer、统一通讯API、可扩展事件模型；
- Transport Services：基于网络传输层的封装，此处利用操作系统的网络处理特性来实现高并发性能；
- Protocol Support：内置了流行的应用层协议的实现。

后续对netty的学习，主要是理解Core为主。

Netty逻辑原理

Netty的核心类`EventLoopGroup`，使用Reactive模式（响应器模式，或称为好莱坞法则），也因此具有异步的特性。

- 服务端：使用2个`EventLoopGroup`，第一个为`parent`(或称为`boss`)，负责注册客户端`Channel`（在nio里对应接受客户端连接）；第二个为`child`(或称为`worker`)，负责与客户端`Channel`的IO读写处理；有时候使用上会简化为一个，既处理网络连接又处理IO读写；
- 客户端：负责与服务端`Channel`的IO读写处理。

IO的读写，会交由一组`ChannelHandler`组成的`ChannelPipline`来处理。这些ChannelHandler分入站（`ChannelInboundHandler`）和出站（`ChannelOutboundHandler`），这有点类似防火墙流量的出入站行为。当网络中有可读数据时，会将网络数据传入第一个`ChannelInboundHandler`，然后依次逐个调用；同样的，当需要往网络中写入数据时，也会依次逐个调用`ChannelOutboundHandler`。这里需

要强调一点，所有出站之间是依次执行，有顺序性；同样的所有入站之间也是；但入站于出站之间，无顺序性，因为一次网络数据的处理，要么是读，要么是写。

以下摘抄自Netty源码文件ChannelPipeline.java。

```
 * <pre>
 *                                                 I/O Request
 *                                            via {@link Channel} or
 *                                        {@link ChannelHandlerContext}
 *                                                      |
 *  +---------------------------------------------------+---------------+
 *  |                           ChannelPipeline         |               |
 *  |                                                  \|/              |
 *  |    +---------------------+            +-----------+----------+    |
 *  |    | Inbound Handler  N  |            | Outbound Handler  1  |    |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |              /|\                                  |               |
 *  |               |                                  \|/              |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |    | Inbound Handler N-1 |            | Outbound Handler  2  |    |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |              /|\                                  .               |
 *  |               .                                   .               |
 *  | ChannelHandlerContext.fireIN_EVT() ChannelHandlerContext.OUT_EVT()|
 *  |        [ method call]                       [method call]         |
 *  |               .                                   .               |
 *  |               .                                  \|/              |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |    | Inbound Handler  2  |            | Outbound Handler M-1 |    |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |              /|\                                  |               |
 *  |               |                                  \|/              |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |    | Inbound Handler  1  |            | Outbound Handler  M  |    |
 *  |    +----------+----------+            +-----------+----------+    |
 *  |              /|\                                  |               |
 *  +---------------+-----------------------------------+---------------+
 *                  |                                  \|/
 *  +---------------+-----------------------------------+---------------+
 *  |               |                                   |               |
 *  |       [ Socket.read() ]                    [ Socket.write() ]     |
 *  |                                                                   |
 *  |  Netty Internal I/O Threads (Transport Implementation)            |
 *  +-------------------------------------------------------------------+
 * </pre>
```

使用初步认识

对netty的理解，从Core部分的统一通讯API开始。先看一段服务端代码实例，来有个初步的印象：

```
/*
 * 服务端Netty代码实例
 */
EventLoopGroup bossGroup  = new NioEventLoopGroup();/*Boss事件循环组*/
EventLoopGroup workerGroup  = new NioEventLoopGroup();/*Worker事件循环组*/
```

```
try {
    /*服务端启动必备*/
    ServerBootstrap b = new ServerBootstrap();
    b.group(bossGroup, workerGroup)
        .channel(NioServerSocketChannel.class)/*指定使用NIO的通信模式*/
        .localAddress(new InetSocketAddress(port))/*指定监听端口*/
        .childHandler(new ChannelInitializer<SocketChannel>() {
            @Override
            protected void initChannel(SocketChannel ch) throws Exception {
                ch.pipeline().addLast(new EchoServerHandler());
            }
        });
    ChannelFuture f = b.bind().sync();/*异步绑定到服务器，sync()会阻塞到完成*/
    f.channel().closeFuture().sync();/*阻塞当前线程，直到服务器的ServerChannel被关闭*/
} finally {
    bossGroup.shutdownGracefully().sync();
    workerGroup.shutdownGracefully().sync();
}
```