Part 2c: Comparison of Designs

- Netty provides multiple event loop implementations. In a typical server channel setting, two event loop groups are created, with one typically called the boss group and the second worker group. What are they? How does Netty achieve synchronization among them?

```
boss group and worker group:
    两个从类 io.netty.channel.nio.NioEventLoopGroup 创建的 NioEventLoopGroup 对象。NioEventLoopGroup 是用来处理I/O操作的多线程事件循环器，‘boss’ 用来接收进来的连接，‘worker’用来处理已经被接收的连接，一旦‘boss’接收到连接，就会把连接信息注册到‘worker’上。
```

```
achieve synchronization:
    NioEventLoop 是NIO框架的 Reactor 线程，它聚合了一个多路复用器对象，它的 Selector 定义如下图。boss group 和 worker group 里面每个 NioEventLoop 都绑定一个多路复用器，不管是 boss group 同时接受多个连接，还是 worker group 里有多个已经被接受的连接，都由 NioEventLoop 的 Selector 来决定 Group 里面的连接如何映射到已经创建的 Channels 上。从而解决竞争问题，实现 synchronization。
```

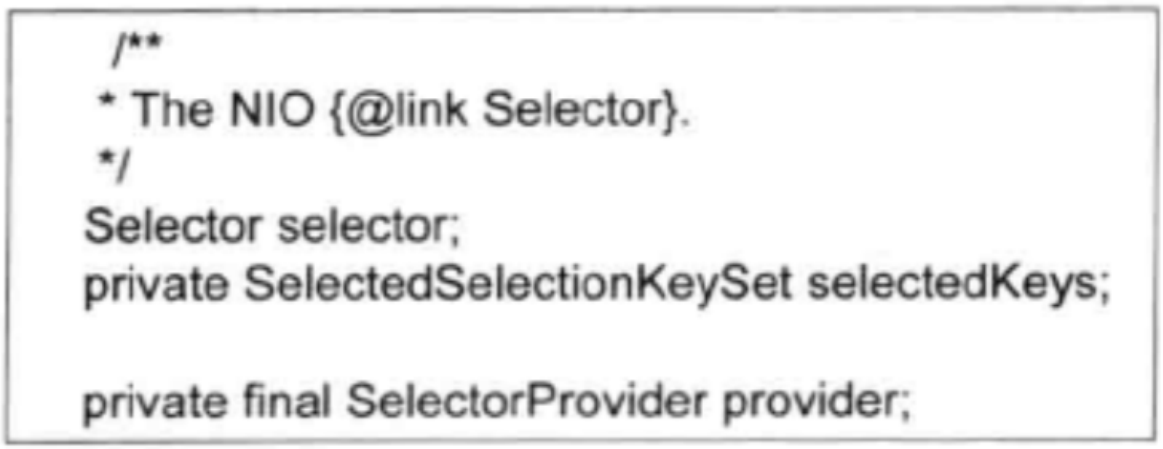

图 18-9　NioEventLoop 的 Selector 定义

参考自：

1. Netty user's guide

1. NioEventLoopGroup 是用来处理I/O操作的多线程事件循环器，Netty提供了许多不同的EventLoopGroup的实现用来处理不同传输协议。在这个例子中我们实现了一个服务端的应用，因此会有2个NioEventLoopGroup会被使用。第一个经常被叫做'boss'，用来接收进来的连接。第二个经常被叫做'worker'，用来处理已经被接收的连接，一旦'boss'接收到连接，就会把连接信息注册到'worker'上。如何知道多少个线程已经被使用，如何映射到已经创建的Channels上都需要依赖于EventLoopGroup的实现，并且可以通过构造函数来配置他们的关系。

2. CSDN https://blog.csdn.net/shenchaohao12321/article/details/89713518
3. 知乎 https://zhuanlan.zhihu.com/p/367218937

一个 NioEventLoop 可以处理多个 Channel 中的 IO 操作，而其只有一个线程。所以对于这个线程资源的使用，就存在了竞争。此时为每一个 NioEventLoop都绑定了一个多跑复用器 Selector，由 Selector 来决定当前 NioEventLoop 的线程为哪些 Channel 服务。

```
@Override
@Deprecated
List<Runnable> shutdownNow();
```

- Method calls such as bind return ChannelFuture. Please describe how one may implement the sync method of a future.

```
    ChannelFuture 实例提供IO操作的结果信息或状态，即将 Netty 异步计算的状态回调，sync（）使程序阻塞等待任务结束，如果任务失败，将“导致失败的异常”重新抛出来。
    sync() 方法一般用于需要等待异步非阻塞任务结束时，在 Future 和继承自 Future 的类中使用即可。Netty 官方相关使用如下，其中，第一个 sync() 用于等待 bind() 的完成执行，第二个用于等待 NioEventLoop 的正确关闭。
```

```
 // Bind and start to accept incoming connections.
    ChannelFuture f = b.bind(port).sync(); // (7)

// Wait until the server socket is closed.
// In this example, this does not happen, but you can do that to gracefully
// shut down your server.
    f.channel().closeFuture().sync();
} finally {
    workerGroup.shutdownGracefully();
    bossGroup.shutdownGracefully();
}
```

- Instead of using ByteBuffer, Netty introduces a data structure called ByteBuf. Please give one key difference between ByteBuffer and ByteBuf.

```
    ByteBuf 没有 flip() 方法,不会因为没有调用flip方法而引起没有数据或者错误数据被发送。因为 ByteBuf 有两个指针，一个对应读操作一个对应写操作，向ByteBuf里写入数据的时候写指针的索引就会增加，同时读指针的索引没有变化。而读指针索引和写指针索引分别代表了消息的开始和结束，即写完可以直接从读指针开始读取到写指针结束。
```

- A major novel, interesting feature of Netty which we did not cover in class is ChannelPipeline. A pipeline may consist of a list of ChannelHander. Compare HTTP Hello World Server and HTTP Snoop Server, what are the handlers that each includes?

```
HTTP Hello World Server 处理 handler 的代码:
@Override
33      public void initChannel(SocketChannel ch) {
34          ChannelPipeline p = ch.pipeline();
35          if (sslCtx != null) {
```

```
36              p.addLast(sslCtx.newHandler(ch.alloc()));
37          }
38          p.addLast(new HttpServerCodec());
39          p.addLast(new HttpHelloWorldServerHandler());
40      }
```

可以看出，使用了：
HttpServerCodec()：来自 io.netty.handler.codec.http.HttpServerCodec，A combination of HttpRequestDecoder and HttpResponseEncoder which enables easier server side HTTP implementation.
HttpHelloWorldServerHandler()：Server 内的一个类，里面含有一个方法 channelRead(ChannelHandlerContext ctx, Object msg) 用来处理 response。

HTTP Snoop Server 处理 handler 的代码：

```
33      @Override
34      public void initChannel(SocketChannel ch) {
35          ChannelPipeline p = ch.pipeline();
36          if (sslCtx != null) {
37              p.addLast(sslCtx.newHandler(ch.alloc()));
38          }
39          p.addLast(new HttpRequestDecoder());
40
41          //p.addLast(new HttpObjectAggregator(1048576));
42          p.addLast(new HttpResponseEncoder());
43
44          //p.addLast(new HttpContentCompressor());
45          p.addLast(new HttpSnoopServerHandler());
46      }
```

可以看出，使用了：
HttpRequestDecoder()，new HttpResponseEncoder()：与上面类似，处理请求和回复
HttpSnoopServerHandler()：同样为 Server 内的一个类，含有检测读取是否完成，编解码和检查结果等方法

Compare:
从 HTTP Snoop Server 的名字就可以看出来，它相比与 Hello World Server 多了监听 Client 状态的功能，在 handler 的 channelReadComplete(ChannelHandlerContext ctx) 等方法中也体现了这一特点。

- Please scan Netty implementation and give a high-level description of how ChannelPipeline is implemented.

忽略掉最初始的设计，ChannelPipeline 的实现从一个线程开始。当一个线程被分配时，可以对线程中的 pipeline 分配空间并添加 handler，代码如上一题。而当 pipeline 被 connect 时，里面的 handler 按照 List 的顺序执行从而实现信息处理。pipeline 处理结束后交由线程调节。boss group and worker group 进行同步处理。