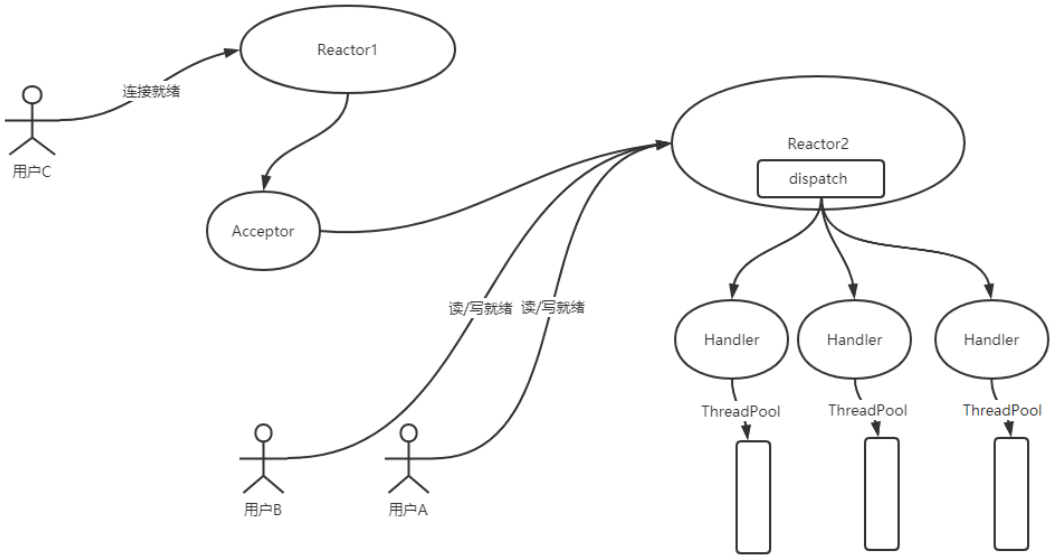
Java Project3 聊天服务器报告

设计理念：

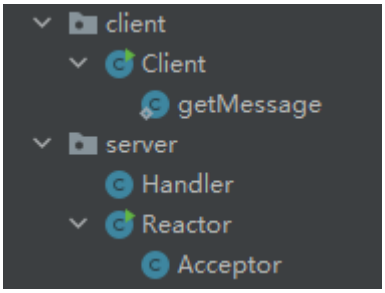
- 本程序由java实现
- 采用Reactor-Handler模式，利用nio的异步机制，提升服务器资源利用率
  - 客户端的连接就绪事件全部由一个selector处理，IO读就绪和写就绪事件由另一个selector处理
  - 利用线程池优化，减小了读就绪事件和写就绪事件的处理开销
  - 

功能描述：

- 服务器可以支持多个客户端同时发起连接；
- 仅支持纯文本消息
- 连接后，一个客户端发送的消息会被服务器传送给当前所有连接着的客户端

实现描述：

程序内部分为几个大类：



Reactor：

- 对IO事件做出反应，并且负责分发任务
  - 具体来说，是将所有的IO事件（连接建立、channel可读）状态更新的察觉者，针对不同的事件，它将事件分配给不同的对象处理

- 属性：

```
final Selector []selector=new Selector[2];
final ServerSocketChannel serverSocketChannel;
ReentrantReadWriteLock lock=new ReentrantReadWriteLock();
static final List<Handler> handlerList = new LinkedList<>();
static final ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();
```

  - selector为选择器，所有的selectionKey都注册到此对象上
    - 注册一个selectionKey到selector[0]上，关注socket连接就绪事件
    - 为每一个客户端注册1个selectionKey到selector[1]上，轮流关注读就绪和写就绪事件
  - serverSocketChannel为一个用于socket连接的处理通道，socket监听便被绑定到此channel上，异步。
  - handlerList：存储了各个客户端的Handler对象的索引，用于给各个客户端转发消息。
  - rwl：一个读写锁，用于线程之间的同步，针对handlerList

Acceptor：

- 连接就绪事件的处理类，实现了runnable接口
- run函数中，接受建立的新IOchannel，创建一个与之对应的Handler对象，并将Handler对象加入到handlerList中（同时调用了Handler的构造函数）

Handler：

- 用于表示一个客户端的连接
- 属性：

```
final SocketChannel socketChannel;
final SelectionKey sk;
static final int BUFFER_SIZE = 500;
final ExecutorService executor = Executors.newFixedThreadPool( nThreads: 1);
ByteBuffer input = ByteBuffer.allocate(BUFFER_SIZE);
ByteBuffer output = ByteBuffer.allocate(BUFFER_SIZE);
LinkedList<String> messageQueue = new LinkedList<>();
final ReentrantReadWriteLock rwl;
boolean flagStart = false;
static String hello = "hello";
```

  - socketChannel为Acceptor函数中accpet()函数返回的SocketChannel，构造时传入
  - sk为将这个channel注册到selector[1]上后返回的SelectionKey
  - executor：线程池，只需要保持一个一个线程或者就可以处理这个客户端
  - input与output：用于读写socketChannel对象
  - messageQueue：消息队列
  - rwl：消息队列的读写锁
  - hello：客户端和服务器初次连接后会交换一次hello变量代表的字符串。

Client：客户端

- 属性：

```
static int BUFFER_SIZE = 500;
static SocketChannel socketChannel;
static ByteBuffer writeBuffer;
static ByteBuffer readBuffer;
static String clientName;
static String IP = "127.0.0.1";
static String hello="hello";
```

  - BUFFER_SIZE：ByteBuffer的大小
  - readbuffer，writebuffer：缓冲区，和channel交互
  - clientName：用户在聊天室内显示的昵称
  - IP：默认为127.0.0.1，否则为用户输入的IP
- 运作方式：
  - 先索要服务端的IP地址
  - 索要用户的昵称（每次加入聊天室都可以更换
  - 建立连接
    - 交换hello信息
    - 建立新的线程用于阻塞接受服务端发送过来的消息
- 主线程继续读输入流，用户每输入一行字符串，主线程就把这一行字符串打包成一个消息发送到服务端去
  - 用户输入的字符串为bye时表示退出

使用方法：

服务端：直接运行jar文件即可，ctrl+c结束。

客户端：

- 运行jar
- 输入自己的昵称

```
Register by putting a nickname to represent yourself
```

- 输入要连接的IP地址，空行代表localhost

```
Register by putting a nickname to represent yourself
nihao
Input the destination IP, with a blank line meaning localhost:
```

- 连接成功

```
Input the destination IP, with a blank line meaning localhost:

Connected!
>>>
```

- 一有消息，客户端会直接显示在命令行上。
  -

```
Connected!
>>> 2020-12-25 19:41:52 User: nihao233
hello!
>>> hello?
>>> 2020-12-25 19:41:58 User: nihao123
hello?
>>>
```

```
Connected!
>>> hello!
>>> 2020-12-25 19:41:52 User: nihao233
hello!
>>> 2020-12-25 19:41:58 User: nihao123
hello?
>>>
```

- 用户通过输入消息发言，以回车结尾
- 用户输入bye时，断开连接。

```
>>> bye
Disconnected, bye!
```

仍然存在的问题：

以下均为服务端、客户端都在本机上跑时发现的问题：

1. 有时候某个客户端发了消息之后，它自己立刻接受到了消息。但是其它客户端没有接收到，需要发言一次之后才能接收到。而且这个问题时有时无。。
2. 有时候某个客户端断开连接后，服务端会正常反应：

```
/127.0.0.1:62447 disconnected
/127.0.0.1:62293 disconnected
/127.0.0.1:62453 disconnected
Exception in thread "pool-3-thread-1" java.nio.channels.CancelledKeyException Create breakpoint
    at java.base/sun.nio.ch.SelectionKeyImpl.ensureValid(SelectionKeyImpl.java:74)
    at java.base/sun.nio.ch.SelectionKeyImpl.interestOps(SelectionKeyImpl.java:99)
    at server.Handler.read(Handler.java:62)
    at server.Handler$processer.run(Handler.java:137) <3 internal calls>

进程已结束,退出代码-1
```

   但是有时候会抛出CancelledKeyException。

   经过debug明确了具体情境：

   当只有两个客户端连接，且其中一个客户端断开后，另一个客户端可能抛出此异常。断开的客户端会导致服务器将此客户端的key取消掉，并且把对应的channel关闭，但是为什么会影响另一个客户端，仍然无法解决。