P2 ChatServer

1 实验要求

实现一个多客户端的纯文本聊天服务器，能同时接受多个客户端的连接，并将任意一个客户端发送的文本向所有客户端（包括发送方）转发。

按照压力测试结果打分。

2 实验环境

操作系统：Windows 10(64 bits)

CPU：Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz　1.50 GHz

Java版本：

java 18.0.2 2022-07-19
Java(TM) SE Runtime Environment (build 18.0.2+9-61)
Java HotSpot(TM) 64-Bit Server VM (build 18.0.2+9-61, mixed mode, sharing)

IDE：VSCode

编译及运行：推荐使用VSCode，在VSCode中下载 `Extension Pack for Java` 拓展，此拓展会自动识别Java工程，识别完成后点击导出到Jar文件即可产生可执行的jar文件。在命令行中输入 `java -jar ./*.jar` 便可运行。

3 设计与实现

客户端

由于本实验重点在于服务器的设计，故仅实现了一个相对基础的客户端。客户端需要同时接收来自用户和服务器的消息且使用了 bio，所以要新开一个 Receiver 线程来接收服务器的消息。由于发送消息的函数 `out.println(msg)` 可能会受服务器状态影响而阻塞(服务器未及时接收消息并返回确认信号)，故每次要发送消息时新开一个 Sender 线程，使主线程能够及时回去监听用户消息。

Receiver

接收线程，主要运行如下循环：

```
while (true) {
    String msg = in.readLine();
    System.out.println("Received: " + msg);
}
```

即接收服务器消息并打印在终端。

Sender

发送线程，`run` 函数如下：

```
private PrintWriter out;

public void run() {
    out.println(msg);
}
```

即简单地替主线程将消息发给服务器。

Client

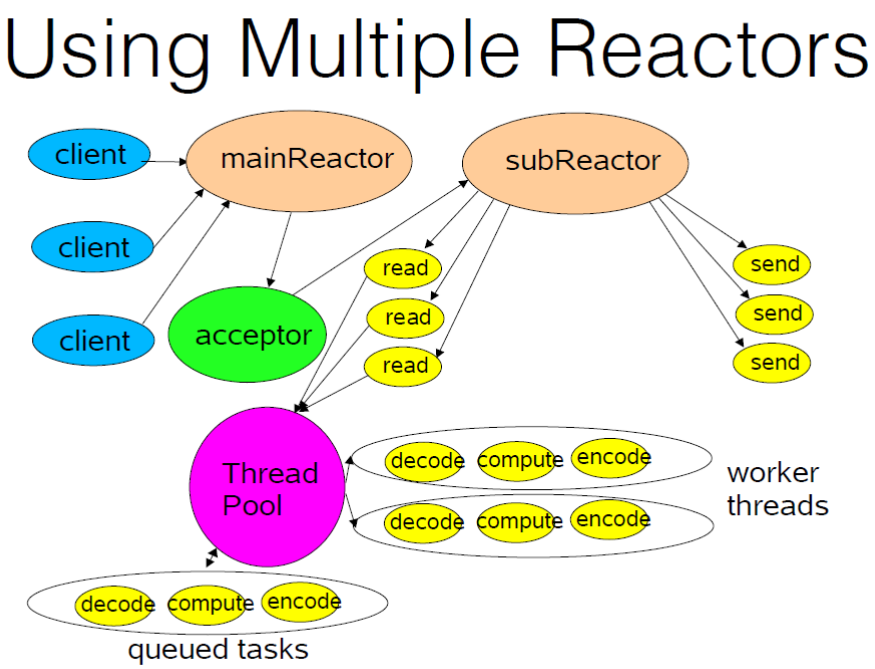
客户端主类，包含了 socket 的相关设置和与客户的交互，如连接服务器、获取输入输出流、新建辅助线程、读取并处理用户输入等。其中初始化部分在构造函数内完成，而具体的处理流程在 go 函数中：

```
private ExecutorService executer = Executors.newCachedThreadPool();

void go() {
    executer.submit(new Receiver(socket, in));
    Scanner scanner = new Scanner(System.in);
    try {
        while (!socket.isClosed()) {
            String msg = scanner.nextLine();
            executer.submit(new Sender(out, msg));
            if (msg.equals("END")) {
                socket.close();
                break;
            }
        }
    } catch (IOException e) {
        System.out.println(e);
    } finally {
        executer.shutdown();
        scanner.close();
    }
}
```

为了便于管理线程，使用了线程池。如果用户输入了 END，就通知服务器(即把 END 发给服务器)并关闭客户端。

服务端

本实验重点在于设计具备高并发能力的服务端。整体使用了 Reactor Pattern，用事件触发代替原来的阻塞等待，即现在一个 reactor 线程可以非阻塞地监听多个客户 channel，当客户发送连接请求或者发送消息触发 IO 事件时，代表对应的 channel 准备好了 IO 操作，此时 reactor 就响应式地去处理这些 IO 事件。



利用 java.nio 实现该设计，以下是 ChatGPT 对 java.nio 的理解：

Java 的 NIO 提供了一系列的接口和类，用于处理不同类型的 IO 操作。常见的 NIO 接口包括：

- Channel 接口：Channel 接口是 NIO 中最基本的接口，它表示一个可以进行 IO 操作的对象。Channel 接口提供了对 IO 操作的抽象，并且提供了向 Channel 中写入数据和从 Channel 中读取数据的方法。常用的 Channel 实现类包括 FileChannel、**SocketChannel** 和 **ServerSocketChannel** 等。
- Buffer 接口：Buffer 接口表示一个可以存储数据的容器。Buffer 接口提供了存储、读取数据的方法，并且提供了对数据的管理功能，如维护读写位置和容量等。Buffer 接口有多个实现类，每个实现类都针对特定类型的数据，如 **ByteBuffer**、CharBuffer 和 IntBuffer 等。
- Selector 接口：Selector 接口表示一个选择器，它可以管理多个 Channel，并**监听它们的IO事件**。Selector 接口提供了选择通道、注册通道和查询通道状态的方法。使用 Selector 可以实现**单线程管理多个通道**，提高程序的效率。**监听IO事件，然后分发到指定的handler。**

下面按照接受客户 -> 接收消息 -> 转发消息 -> 客户退出的顺序简单讲解框架和代码。

接受客户

由 mainReactor 负责，以 Server 类作为 MainReactor：

```
private Server() { // MainReactor
    try {
        ...
        SelectionKey sk = serverSocket.register(selector,
SelectionKey.OP_ACCEPT);
        sk.attach(new Acceptor());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

初始化时将接受事件注册在 selector 中，并将负责接收的对象 Acceptor attach 到该 sk，这样一旦监听到 OP_ACCEPT，对应的 SelectionKey 就会被加入到 Ready Set 中。

然后为准备好 IO 操作的 channel 分派(dispatch)相应的任务：

```
while (true) {
    selector.select();
    Set<SelectionKey> selected = selector.selectedKeys();
    Iterator<SelectionKey> it = selected.iterator();
    while (it.hasNext()) {
        dispatch((SelectionKey)(it.next()));
    }
    selected.clear();
}
```

所谓分派，即调用之前 attach 的 run 函数：

```
void dispatch(SelectionKey k) {
    Runnable r = (Runnable)(k.attachment());
    if (r != null) {
        r.run();
    }
}
```

即调用 Acceptor 的 run：

```
public void run() {
    try {
        SocketChannel sc = serverSocket.accept();
        if (sc != null) {
            Client c = new Client(subSelectors[next], sc);
            if (++next == subSelectors.length) next = 0;
            c.addMsgListener(theServer);
            synchronized (theServer) {
                lstClient.add(c);
            }
        }
    } catch(IOException ex) {}
}
```

将客户交给子选择器线程去监听和处理后续事件，主线程只负责接受新客户。

接收消息

即对应 Channel 准备好了 IO 操作，subSelector 为其分派之前 attach 的对象(即 Client)的 run 函数：

```
public void run() {
  try {
    ByteBuffer in = ByteBuffer.allocate(MAXIN);
    socket.read(in);
    String str = new String(in.array());
    notifyMsgRcvd(str);
    if (str.equals("END")) {
      System.out.println("closing...");
```

```
      quit();
    }
  } catch (IOException e) {
    quit();
  }
}
```

即读取客户的消息，并调用 `notifyMsgRcvd(str)` 通知 Server 将消息加入消息队列，并唤醒 queueRunner 线程去转发消息：

```
synchronized void sendMsg(String msg) {
    msgQueue.add(msg);
    this.notify();
}
```

转发消息

消息队列中增加了 msg，queueRunner 被唤醒，并遍历消息队列，将 msg 转发给当前客户列表中的所有客户，同理，将发送消息的任务交给线程池处理。

```
public void run() {
    while (true) {
      synchronized (this) {
        try {
          while (msgQueue.isEmpty()) {
            this.wait();
          }
          for (String msg : msgQueue) {
            for (Client c : lstClient) {
              executer.submit(new Sender(c, msg));
            }
            msgQueue.poll(); // right?
          }
        } catch (InterruptedException e) {}
      }
    }
}
```

客户退出

退出时需要通知 Server，便于将此客户从客户列表中移除，同时要取消其在 sk 中的注册记录。

```
void quit() {
  notifyQuit();
  sk.cancel();
  try {
    socket.close();
  } catch (IOException e1) {}
}
```

更多细节详见代码。

4 测试结果

个人未做压力测试，故仅展示多个客户端连接服务器并发送消息时的现象。

启动服务器

```
PS D:\浙大学习事项\大三上\Java应用技术\ChatServer> java -jar .\ChatServer.jar
Started: sun.nio.ch.ServerSocketChannelImpl[/[0:0:0:0:0:0:0:0]:3936]
```

启动 5 个客户端

```
PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52955]

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52960]

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52961]

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52962]

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52964]
```

客户端 1 发送消息

```
PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52993]
hello, here is 1
Received: hello, here is 1

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52995]
Received: hello, here is 1

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52996]
Received: hello, here is 1

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52997]
Received: hello, here is 1

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52998]
Received: hello, here is 1
```

可见转发成功。

客户端 2 发送消息

```
PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52993]
hello, here is 1
Received: hello, here is 1
Received: hello, here is 2

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52995]
Received: hello, here is 1
hello, here is 2
Received: hello, here is 2

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52996]
Received: hello, here is 1
Received: hello, here is 2

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52997]
Received: hello, here is 1
Received: hello, here is 2

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52998]
Received: hello, here is 1
Received: hello, here is 2
```

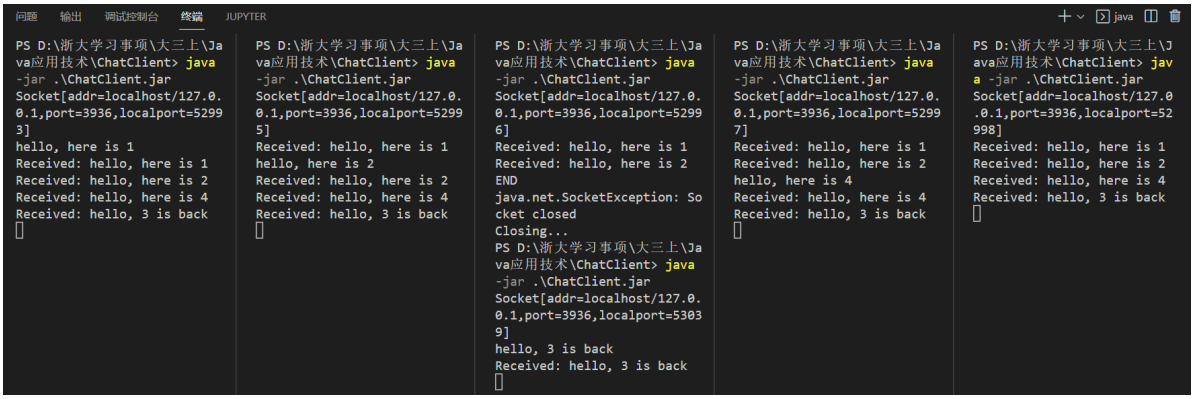
转发成功。

客户端 3 退出后 客户端 4 发送消息

```
PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52993]
hello, here is 1
Received: hello, here is 1
Received: hello, here is 2
Received: hello, here is 4

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52995]
Received: hello, here is 1
hello, here is 2
Received: hello, here is 2
Received: hello, here is 4

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52996]
Received: hello, here is 1
Received: hello, here is 2
END
java.net.SocketException: Socket closed
Closing...
PS D:\浙大学习事项\大三上\Java应用技术\ChatClient>

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52997]
Received: hello, here is 1
Received: hello, here is 2
hello, here is 4
Received: hello, here is 4

PS D:\浙大学习事项\大三上\Java应用技术\ChatClient> java -jar .\ChatClient.jar
Socket[addr=localhost/127.0.0.1,port=3936,localport=52998]
Received: hello, here is 1
Received: hello, here is 2
Received: hello, here is 4
```

可见其他已存在的连接不受影响。

客户端 3 再次连接并发送消息



可见之前的退出行为不会影响新连接的建立和后续通信。

5 讨论与心得

之前在计网课程中已经使用 C++ 实现过 Socket Server，不过当时是以功能为主，而此次是以性能为主，学到了更多技术层面的知识。老师上课时已经讲解了模块间的解耦(让我想起了 MVVM 框架)、类的设计、消息队列、客户端退出等传统层面的设计和优化。在实现了基本的优化后，我又学习了 Doug Lea 大师的 nio 课件，学习了 Reactor Pattern 和 java.nio，并加以实现。由于期末缺少设计压力测试的时间，故没能测试 Reactor Pattern 下的性能，不确定在实践中能否明显优于原来的设计。总而言之，收获还是比较大的。