005.使用NIO实现网络通信

上一门的课程《IO模型和NIO、AIO入门》把NIO里的大部分知识都讲的很清楚了：

1)Channel

2)Buffer

3)Selector

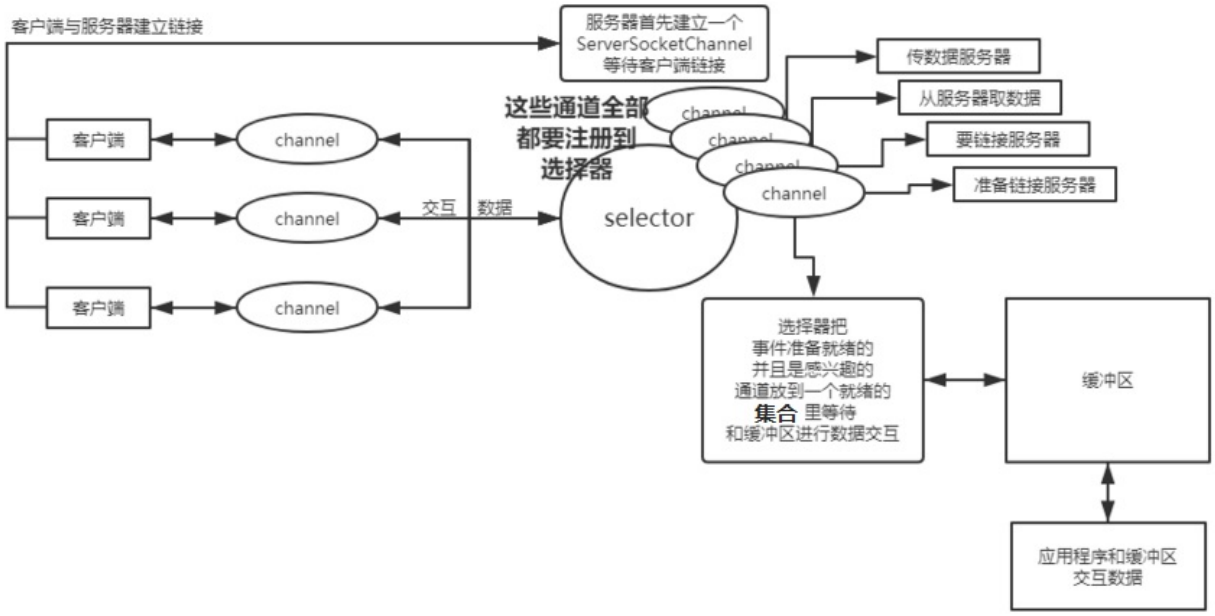
涉及到网络编程的部分，前面是跳过的，我们这里把补充一下：

ServerSocketChannel 监听新进来的TCP连接的通道，对应前面第四单元里讲的代表服务器端ServerSocket

SocketChannel 连接到TCP网络套接字的通道，对应前面第四单元里讲的代表客户端Socket

DatagramChannel 连接到UDP包的通道，后面第七单元讲

回顾一下上一门课里讲的NIO的网络编程流程：



ServerSocketChannel类

NIO中的 ServerSocketChannel 是一个可以监听新进来的TCP连接的通道，就像标准IO中的ServerSocket一样。

通过调用 ServerSocketChannel.open() 方法来获取ServerSocketChannel的对象实例. 如：

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
```

通过调用ServerSocketChannel.close() 方法来关闭ServerSocketChannel. 如：

```
serverSocketChannel.close();
```

监听新进来的连接

通过 ServerSocketChannel.accept() 方法监听新进来的连接。

当 accept()方法返回的时候,它返回一个包含新进来的连接的 SocketChannel。

因此, accept()方法会一直阻塞到有新连接到达。

通常不会仅仅只监听一个连接,在while循环中调用 accept()方法. 如下面的例子:

```
while(true){
    SocketChannel socketChannel =
            serverSocketChannel.accept();
    //do something with socketChannel...
}
```

当然,也可以在while循环中使用除了true以外的其它退出准则。

非阻塞模式

ServerSocketChannel可以设置成非阻塞模式。

在非阻塞模式下，accept() 方法会立刻返回，如果还没有新进来的连接,返回的将是null。

因此，需要检查返回的SocketChannel是否是null.如：

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.socket().bind(new InetSocketAddress(9999));
serverSocketChannel.configureBlocking(false);
while(true){
    SocketChannel socketChannel =
            serverSocketChannel.accept();
    if(socketChannel != null){
        //do something with socketChannel...
    }
}
```

SocketChannel类

NIO中的SocketChannel是一个连接到TCP网络套接字的通道。可以通过以下2种方式创建SocketChannel：

1. SocketChannel的open方法。
2. 一个新连接到达ServerSocketChannel时，会创建并返回一个SocketChannel。

打开 SocketChannel

```
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("http://baidu.com", 80));
```

关闭 SocketChannel

当用完SocketChannel之后调用SocketChannel.close()关闭SocketChannel：

```
socketChannel.close();
```

从 SocketChannel 读取数据

要从SocketChannel中读取数据，调用一个read()的方法之一。以下是例子：

```
ByteBuffer buf = ByteBuffer.allocate(48);
int bytesRead = socketChannel.read(buf);
```

首先，分配一个Buffer。从SocketChannel读取到的数据将会放到这个Buffer中。

然后，调用SocketChannel.read()。该方法将数据从SocketChannel 读到Buffer中。

read()方法返回的int值表示读了多少字节进Buffer里。

如果返回的是-1，表示已经读到了流的末尾（连接关闭了）。

写入 SocketChannel

写数据到SocketChannel用的是SocketChannel.write()方法，该方法以一个Buffer作为参数。示例如下：

```
String newData = "New String to write to file..." +
System.currentTimeMillis();
ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());
buf.flip();
while(buf.hasRemaining()) {
    channel.write(buf);
}
```

注意SocketChannel.write()方法的调用是在一个while循环中的。Write()方法无法保证能写多少字节到SocketChannel。

所以，我们重复调用write()直到Buffer没有要写的字节为止。

非阻塞模式

可以设置 SocketChannel 为非阻塞模式（non-blocking mode）.

设置之后，就可以在异步模式下调用connect(), read() 和write()了。

connect()

如果SocketChannel在非阻塞模式下，此时调用connect()，该方法可能在连接建立之前就返回了。

为了确定连接是否建立，可以调用finishConnect()的方法。像这样：

```
socketChannel.configureBlocking(false);
socketChannel.connect(new InetSocketAddress("http://jenkov.com", 80));
while(! socketChannel.finishConnect() ){
    //wait, or do something else...
}
```

write()

非阻塞模式下，write()方法在尚未写出任何内容时可能就返回了。

所以需要在循环中调用write()。前面已经有例子了，这里就不赘述了。

read()

非阻塞模式下,read()方法在尚未读取到任何数据时可能就返回了。

所以需要关注它的int返回值，它会告诉你读取了多少字节。

非阻塞模式与选择器

非阻塞模式与选择器搭配会工作的更好，通过将一或多个SocketChannel注册到Selector，可以询问选择器哪个通道已经准备好了读取，写入等。Selector与SocketChannel的搭配使用会在上一门课程里面讲了的。

在了解上面的基本情况下，把上一章中的聊天室案例升级一下：

服务端思路：

定义一个 Charset 字符集用于解析数据

定义两个 ByteBuffered 缓冲区用于收发数据

定义一个 ChatRoomMap<String,Channel> 用于存放客户端集合

定义一个 Selector 用于监听通道事件

通过 ServerSocketChannel 的 open 方法打开一个 ServerSocketChannel 通道

设置为非阻塞模式，并传递一个 InetSocketAddress 绑定端口

注册 ServerSocketChannel 的接收就绪事件通道到 Selector

循环监听事件

- 调用 Selector 的 select 方法阻塞直到有准备就绪的通道
- 调用 Selector 的 selectedKeys 获取准备就绪的通道集合
- 针对事件处理通道

- **接收就绪：**
    - 代表有客户端要连接，通过 SelectionKey 对象获取 ServerSocketChannel
    - 调用 accept 方法接收请求客户端通道 SocketChannel
    - 为 SocketChannel 通道注册可读数据事件到 Selector
- **可读就绪**:
    - 代表客户端向服务器端发送了消息，通过 SelectionKey 对象获取 SocketChannel
    - 调用 read 方法将数据读入通道，通过 Charset 的 decode 将数据解码
    - 根据信息的情况调用login方法实现登录，调用 dispatch 方法将消息转发给各个客户端，或者调用 sendMsgToUser方法将信息发送给指定客户端

处理完事件后要清空 SelectionKey 集合，当下次该通道变成就绪时，Selector 才会再次将其放入 SelectionKey 中

客户端思路：
定义一个 Charset 字符集用于解析数据
定义两个 ByteBuffered 缓冲区用于收发数据
定义一个 Selector 用于监听通道事件
通过 SocketChannel 的 open() 打开一个 SocketChannel 实例
设置 SocketChannel 实例为非阻塞模式，并调用 connect() 连接到服务端
注册 SocketChannel 的连接就绪事件到 Selector
循环监听事件

- 调用 Selector 的 select 方法阻塞直到有准备就绪的通道
- 调用 Selector 的 selectedKeys 获取准备就绪的通道集合
- 针对事件处理通道
    - **连接就绪：**
        - 调用 SocketChannel 的 isConnectionPending 判断否正在连接服务器端，如果是则调用 finishConnect 完成

连接

- 新建一个线程，通过 Scanner 接收用户输入并通过 SocketChannel 的 write 写入到通道，和基于BIO实现的案例一样我们要在信息上进行协议字符的包装。
- 设置监听 SocketChannel 的可读数据事件

- **可读就绪：**
  - 有从服务器端发送过来的信息，读取输出到屏幕上

处理完事件后要清空 SelectionKey 集合，当下次该通道变成就绪时，Selector 才会再次将其放入 SelectionKey 中