006.使用AIO实现网络通信

I/O 与 NIO 一个比较重要的区别：

使用 I/O 的时候往往会引入多线程，每个连接使用一条单独的线程，

NIO 则是使用单线程或者只使用少量的多线程，很多连接共用一条线程。

NIO能使用单线程的同时又不阻塞就需要一直轮询，比较消耗系统资源，所以异步非阻塞模式 AIO 就诞生了。

阻塞、非阻塞、同步和异步，四个概念，我在上一门课《IO模型和NIO、AIO入门》一开始就给大家说明白了，

在这里，我在强调一下同步和异步的区别：

如果IO的实际操作是由操作系统完成，在将结果返回给程序，就是异步。

如果实际的IO需要应用程序本身去执行，会阻塞线程，就是同步。

在上一门的课程《IO模型和NIO、AIO入门》中，AIO的文件通道AsynchronousFileChannel，

读写数据方法都给大家讲清楚了，到现在网络编程的环境里，

对应的AsynchronousSocketChannel和AsynchronousServerSocketChannel，实际上，在数据的处理方式上，大同小异的！

AIO的核心类：除了上面三个以外，还有一个AsynchronousChannelGroup这个类。

- AsynchronousFileChannel类是异步的方式处理本地文件的文件通道。
- AsynchronousSocketChannel和AsynchronousServerSocketChannel两个个类是javaAIO为TCP通信提供的异步Channel。
- AsynchronousChannelGroup是异步Channel的分组管理器，它可以实现资源共享。

创建AsynchronousServerSocketChannel的代码如下：

```
AsynchronousServerSocketChannel serverChannel
          = AsynchronousServerSocketChannel.open().bind(new InetSocketAddress(PORT));
```

其中open()有一个重载方法，可以使用指定的AsynchronousChannelGroup来创建AsynchronousServerSocketChannel。

创建AsynchronousChannelGroup时，需要传入一个ExecutorService，

也就是绑定一个线程池，该线程池负责两个任务：处理IO事件和触发CompletionHandler回调接口。

```
AsynchronousServerSocketChannel serverSocketChannel = null;
        try {
            ExecutorService executorService = Executors.newFixedThreadPool(80);
            AsynchronousChannelGroup channelGroup =
AsynchronousChannelGroup.withThreadPool(executorService);
```

```
            serverSocketChannel = AsynchronousServerSocketChannel.open(channelGroup).bind(new
InetSocketAddress(9000));
        }catch (IOException ioe){
            ioe.printStackTrace();
        }
```

AsynchronousServerSocketChannel创建成功后，类似于ServerSocket，
也是调用accept()方法来接受来自客户端的连接，由于异步IO实际的IO操作是交给操作系统来做的，
用户进程只负责通知操作系统进行IO和接受操作系统IO完成的通知。

所以异步的ServerChannel调用accept()方法后，当前线程不会阻塞，
程序也不知道accept()方法什么时候能够接收到客户端请求并且操作系统完成网络IO，为解决这个问题，
AIO为accept方法提供两个版本：

Future<AsynchronousSocketChannel> accept() ：开始接收客户端请求，如果当前线程需要进行网络IO，
即获得AsynchronousSocketChannel，则应该调用该方法返回的Future对象的get()方法，但是get方法会阻塞该线程，所以这种方式是阻塞式的异步IO。

void accept(A attachment ,CompletionHandler<AsynchronousSocketChannel,? super A> handler)：
开始接受来自客户端请求，连接成功或失败都会触发CompletionHandler对象的相应方法。

CompletionHandler接口中定义了两个方法：

- completed(V result , A attachment)：当IO完成时触发该方法。
- faild(Throwable exc, A attachment)：当IO失败时触发该方法，第一个参数代表IO操作失败引发的异常或错误。

AsynchronousSocketChannel的的用法与Socket类似
有三个方法，但是不同的是每个方法又分为Future版本与CompletionHandler版本。

connect():用于连接到指定端口，指定IP地址的服务器
read()、write():完成读写。

注意！使用异步Channel时，accept()、connect()、read()、write()等方法都不会阻塞，
也就是说如果使用返回Future的这些方法，程序并不能知道什么时候成功IO，
必须要使用get方法，等get方法的阻塞结束后才能确保IO完成，继续执行下面的操作。

先看简单的应用代码：

```
public class SimpleAIOServer {
    public static void main(String[] args) {
        //创建AsynchronousServerSocketChannel对象
        try( AsynchronousServerSocketChannel serverChannel =
                     AsynchronousServerSocketChannel.open();) {
            //绑定ip和端口
            serverChannel.bind(new InetSocketAddress("127.0.0.1",5000));
            //循环接受客户的的链接
            while (true){
                Future<AsynchronousSocketChannel> future = serverChannel.accept();
                //拿到future后，通过get方法获取到客户端对应的socketChannel
                AsynchronousSocketChannel socketChannel = future.get();
                //拿到socketChannel就可以收发数据
                ByteBuffer wbuffer = ByteBuffer.allocate(1024);
                wbuffer.clear();
                wbuffer.put("这里是服务器端".getBytes());
                wbuffer.flip();
                socketChannel.write(wbuffer).get(); //异步模式，必须用get方法，把线程阻塞，不
然程序就结束了
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


public class SimpleAIOClient {
    public static void main(String[] args) {
        try(AsynchronousSocketChannel socketChannel =
                    AsynchronousSocketChannel.open();) {
            //读数据的缓冲
            ByteBuffer buffer = ByteBuffer.allocate(1024);
            //链接服务器
            socketChannel.connect(new InetSocketAddress("127.0.0.1",5000)).get();
            //读数据
            buffer.clear();
            socketChannel.read(buffer).get();
            buffer.flip();
            String content = Charset.forName("UTF-8").decode(buffer).toString();
            System.out.println("服务器发送来的信息：" + content);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

实现聊天室案例：

```
public class Server {
    private static final int SERVER_PORT = 31000;
    private static final String CHARSET = "UTF-8";
    private Charset charset = Charset.forName(CHARSET);
    //链接进来的客户端要保存到统一的集合
    public static ChatRoomMap<String,AsynchronousSocketChannel> clients = new ChatRoomMap<>();

    //对服务器进行初始化
    public void init(){
        try{
            //创建线程池
            ExecutorService executorService = Executors.newFixedThreadPool(20);
```

```
            //创建channelGroup
            AsynchronousChannelGroup channelGroup
                    = AsynchronousChannelGroup.withThreadPool(executorService);
            //拿到服务器的套接字通道
            AsynchronousServerSocketChannel serverSocketChannel
                    = AsynchronousServerSocketChannel.open(channelGroup);

            //绑定ip和端口
            serverSocketChannel.bind(new InetSocketAddress(SERVER_PORT));
            //循环接收客户端的链接
            serverSocketChannel.accept(null, new AcceptHandler(serverSocketChannel));
        }catch (Exception e){
            System.out.println("服务器启动失败，可能端口号被占用！");
        }
    }

    private class AcceptHandler implements CompletionHandler<AsynchronousSocketChannel,Object>{
        private AsynchronousServerSocketChannel serverSocketChannel;
        public AcceptHandler(AsynchronousServerSocketChannel serverSocketChannel){
            this.serverSocketChannel = serverSocketChannel;
        }

        //接收客户端的信息,发送信息的缓冲区
        private ByteBuffer rbuffer = ByteBuffer.allocate(1024);
        private ByteBuffer wbuffer = ByteBuffer.allocate(1024);

        /**
         * 操作系统完成了指定的io后，回调completed
         */
        @Override
        public void completed(AsynchronousSocketChannel clientSocketChannel, Object attachment) {
            //操作系统调用这里的completed方法，表示，服务器有客户端连进来了
            //作系统调用这里的completed方法，表示，服务器有客户端连进来了,递归，又让操作系统给我们准备接
下一个客户端
            serverSocketChannel.accept(null,this);
            clientSocketChannel.read(rbuffer, null, new CompletionHandler<Integer, Object>() {
                @Override
                public void completed(Integer hasRead, Object attachment) {
                    //读取数据成功
                    rbuffer.flip();
                    String content = charset.decode(rbuffer).toString();
//StandardCharsets.UTF_8.decode(rbuffer).toString();//Charset.forName("UTF-
8").decode(rbuffer).toString();//String.valueOf(Charset.forName("UTF-8").decode(rbuffer));//new
String(rbuffer.array(),0,result);
                    //服务器收到客户端的信息，有两类：客户端注册来的用户名；聊天的信息
                    if(content.startsWith(ChatRoomProtocol.USER_ROUND) &&
                            content.endsWith(ChatRoomProtocol.USER_ROUND)){
                        //信息是注册来的用户名
                        //要进行一系列的处理
                        login(clientSocketChannel,content);
                    }else if(content.startsWith(ChatRoomProtocol.PRIVATEMSG_ROUND) &&
content.endsWith(ChatRoomProtocol.PRIVATEMSG_ROUND)){
                        sendMsyToUser(clientSocketChannel,content);
                    }else if(content.startsWith(ChatRoomProtocol.PUBLICMSG_ROUND) &&
content.endsWith(ChatRoomProtocol.PUBLICMSG_ROUND)){
                        dispatch(clientSocketChannel,content);
                    }
                    rbuffer.clear();
                    //由来递归实现重复，循环读取数据
                    clientSocketChannel.read(rbuffer,null,this);
                }

                @Override
                public void failed(Throwable ex, Object attachment) {
                    System.out.println("数据读取失败：" + ex);
                    //方可能是客户端关闭了，要把失效的客户端从集合里移除
                    Server.clients.removeByValue(clientSocketChannel);
                }
```

```
            });
        }

        /**
         * 操作系统完成了指定的io过程中，出现异常，回调failed
         * @param exc
          * @param attachment
          */
        @Override
        public void failed(Throwable exc, Object attachment) {
            System.out.println("链接失败：" + exc);
        }

        /**
         * 服务器实现客户端登录功能
         * @param client
          * @param content
          */
        private void login(AsynchronousSocketChannel client,String content){
            System.out.println("登录来啦....");
            try{
                //接受到的是用户名称
                //拿到真正的用户名称
                String userName = getRealMsg(content);
                //判断用户不能重复
                if (Server.clients.map.containsKey(userName)) {
                    System.out.println("用户名重复了");
                    wbuffer.clear();
                    wbuffer.put(charset.encode(ChatRoomProtocol.USER_ROUND+ChatRoomProtocol.NAME_REP
                            +ChatRoomProtocol.USER_ROUND));
                    wbuffer.flip();
                    client.write(wbuffer).get();
                } else {
                    System.out.println("用户登录成功！");
                    wbuffer.clear();

wbuffer.put(charset.encode(ChatRoomProtocol.USER_ROUND+ChatRoomProtocol.LOGIN_SUCCESS
                            +ChatRoomProtocol.USER_ROUND));
                    wbuffer.flip();
                    client.write(wbuffer).get();
                    Server.clients.put(userName,client);
                }
            }catch (Exception e){
                e.printStackTrace();
            }
        }
        /**
         * 对私聊信息的转发
         * @param client
          * @param str
          */
        private void sendMsyToUser(AsynchronousSocketChannel client, String str){
            try{
                //客户端发送来的信息是私聊
                //拿到真正的信息,信息里包含了目标用户和消息
                String userAndMsg = getRealMsg(str);
                //上面的信息是用ChatRoomProtocol.SPLIT_SIGN来隔开的
                String targetUser = userAndMsg.split(ChatRoomProtocol.SPLIT_SIGN)[0];
                String privatemsg = userAndMsg.split(ChatRoomProtocol.SPLIT_SIGN)[1];

                //服务器就可以转发给指定的用户了三
                wbuffer.clear();
                wbuffer.put(charset.encode(Server.clients.getKeyByValue(client) +"悄悄地说：" +
privatemsg));
                wbuffer.flip();
                Server.clients.map.get(targetUser).write(wbuffer).get();
            }catch (Exception e){
                Server.clients.removeByValue(client);
```

```
            }
        }

        /**
         * 对公聊信息的广播
         * @param client
          * @param str
          */
        private void dispatch(AsynchronousSocketChannel client, String str){
            try{
                //拿到真正的信息
                String publicmsg = getRealMsg(str);
                Set<AsynchronousSocketChannel> valueSet = Server.clients.getValueSet();
                for(AsynchronousSocketChannel cli : valueSet){
                    wbuffer.clear();
                    wbuffer.put(charset.encode(Server.clients.getKeyByValue(client) +"说：" +
publicmsg));
                    wbuffer.flip();
                        cli.write(wbuffer).get();
                }
            }catch (Exception e){
                Server.clients.removeByValue(client);
            }
        }
        //去除协议字符的方法
        private String getRealMsg(String lines) {
            return lines.substring(ChatRoomProtocol.PROTOCOL_LEN, lines.length() -
ChatRoomProtocol.PROTOCOL_LEN);
        }
    }

    //服务器的程序执行入口
    public static void main(String[] args) {
        Server server = new Server();
        server.init();
        try {
            Thread.sleep(100000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Client {
    private static final int SERVER_PORT = 31000;
    private static final String CHARSET = "UTF-8";
    private Charset charset = Charset.forName(CHARSET);
    //与服务器链接的通道
    AsynchronousSocketChannel clientChannel;
    //定义主窗体
    JFrame mainWin = new JFrame("多人聊天室");
    //定义显示聊天内容的文本域
    JTextArea jta = new JTextArea(16,48);
    //定义输入聊天内容的文本框
    JTextField jtf = new JTextField(40);
    //定义发送聊天内容的按钮
    JButton sendBtn = new JButton("发送");
    //loing的tip
    String tip = "";
    //写缓冲器
    ByteBuffer wbuffer = ByteBuffer.allocate(1024);
    //读缓冲器
    ByteBuffer rbuffer = ByteBuffer.allocate(1024);


    //上面的皮肤的，整个客户端程序初始化
    public void init(){
```

```
        mainWin.setLayout(new BorderLayout());
        jta.setEditable(false);
        mainWin.add(new JScrollPane(jta),BorderLayout.CENTER);
        JPanel jp = new JPanel();
        jp.add(jtf);
        jp.add(sendBtn);
        //按钮我们要给他定义点击后的事件响应
        Action sendAction = new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String content = jtf.getText();
                content = content.trim();
                if(content.length() > 0){
                    try {
                        if (content.indexOf(":") > 0 && content.startsWith("//")) {
                            //私聊信息
                            content = content.substring(2);
                            wbuffer.clear();
                            wbuffer.put(charset.encode(ChatRoomProtocol.PRIVATEMSG_ROUND +
                                    content.split(":")[0] + ChatRoomProtocol.SPLIT_SIGN +
                                    content.split(":")[1] + ChatRoomProtocol.PRIVATEMSG_ROUND));
                            wbuffer.flip();
                            clientChannel.write(wbuffer).get();
                        } else {
                            //公聊信息
                            wbuffer.clear();
                            wbuffer.put(charset.encode(ChatRoomProtocol.PUBLICMSG_ROUND +
                                    content + ChatRoomProtocol.PUBLICMSG_ROUND));
                            wbuffer.flip();
                            clientChannel.write(wbuffer).get();
                        }
                    }catch (Exception ex){
                        System.out.println("发送数据异常！");
                    }
                }
                //把发送出去的信息，从文本框中清除
                jtf.setText("");
            }
        };
        //把自己定义的发送信息的事件响应和按钮本身关联
        sendBtn.addActionListener(sendAction);
        //还是定义一个Ctrl+Enter快捷键给发送信息
        jtf.getInputMap().put(KeyStroke.getKeyStroke('\n', InputEvent.CTRL_MASK),"send");
        //上面定义了一个快捷键，把快捷键和发送信息的事件响应关联起来
        jtf.getActionMap().put("send",sendAction);
        mainWin.add(jp,BorderLayout.SOUTH);
        mainWin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainWin.pack();//把窗体自动调整大小，根据包裹在里面的组件自动调整到合适的大小
        mainWin.setVisible(true);
    }
    //链接服务器
    public void connect(){
        try{
            //定义线程池
            ExecutorService executorService =
                    Executors.newFixedThreadPool(80);
            //定义channelGroup
            AsynchronousChannelGroup channelGroup =
                    AsynchronousChannelGroup.withThreadPool(executorService);
            //获取客户端的套接字通道,因为我们用了GUI，客户端套接字应该保存到上面的属性上去
            clientChannel =
                    AsynchronousSocketChannel.open(channelGroup);
            //链接服务器
            clientChannel.connect(new InetSocketAddress("127.0.0.1",SERVER_PORT)).get();
            System.out.println("客户端链接服务器成功！");
            //链接上服务器后，就要登录服务器
            login(clientChannel,tip);
```

```
            //接收服务器传来的信息
            rbuffer.clear();
            clientChannel.read(rbuffer, null, new CompletionHandler<Integer, Object>() {
                @Override
                public void completed(Integer result, Object attachment) {
                    //进到这里就表示读出来了
                    rbuffer.flip();
                    String content = charset.decode(rbuffer).toString();
                    //content也是有类型：登录后响应信息，聊天信息
                    if (content.startsWith(ChatRoomProtocol.USER_ROUND) &&
                            content.endsWith(ChatRoomProtocol.USER_ROUND)) {
                        //拿到真正的登录回复信息
                        String loginRes = getRealMsg(content);
                        if (loginRes.equals(ChatRoomProtocol.NAME_REP)) {
                            tip = "用户名重复，请重新";
                            login(clientChannel, tip);
                        } else if (loginRes.equals(ChatRoomProtocol.LOGIN_SUCCESS)) {
                            System.out.println("客户端登录成功！");
                        }
                    } else {
                        //回来的是聊天信息
                        jta.append(content + "\n");
                    }
                    rbuffer.clear();
                    clientChannel.read(rbuffer,null,this);
                }

                @Override
                public void failed(Throwable exc, Object attachment) {
                    System.out.println("读取数据失败" + exc);
                }
            });
        }catch (Exception e){
            e.printStackTrace();
        }
    }

    //去除协议字符的方法
    private String getRealMsg(String lines) {
        return lines.substring(ChatRoomProtocol.PROTOCOL_LEN, lines.length() -
ChatRoomProtocol.PROTOCOL_LEN);
    }


    private void login(AsynchronousSocketChannel client,String tip){
        try {
            //虽然我们还没见过到GUI，这里小小用一个gui里的弹出对话框
            String userName = JOptionPane.showInputDialog(tip + "输入用户名：");
            //把userName发送到服务器上去
            wbuffer.clear();
            wbuffer.put(charset.encode(ChatRoomProtocol.USER_ROUND + userName +
ChatRoomProtocol.USER_ROUND));
            wbuffer.flip();
            client.write(wbuffer).get();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //客户端的程序入口
    public static void main(String[] args) {
        Client client = new Client();
        client.init();
        client.connect();
    }

}
```