

实战：实现客户端与服务端收发消息

这一小节，我们来实现客户端与服务端收发消息，我们要实现的具体功能是：在控制台输入一条消息之后按回车，校验完客户端的登录状态之后，把消息发送到服务端，服务端收到消息之后打印并且向客户端发送一条消息，客户端收到之后打印。

收发消息对象

首先，我们来定义一下客户端与服务端的收发消息对象，我们把客户端发送至服务端的消息对象定义为 `MessageRequestPacket`。

```
@Data
public class MessageRequestPacket extends Packet
{
    private String message;

    @Override
    public Byte getCommand() {
        return MESSAGE_REQUEST;
    }
}
```

指令为 `MESSAGE_REQUEST = 3`

我们把服务端发送至客户端的消息对象定义为
MessageResponsePacket

```
@Data
public class MessageResponsePacket extends Packet
{
    private String message;

    @Override
    public Byte getCommand() {
        return MESSAGE_RESPONSE;
    }
}
```

指令为 MESSAGE_RESPONSE = 4

至此，我们的指令已经有如下四种

```
public interface Command {

    Byte LOGIN_REQUEST = 1;

    Byte LOGIN_RESPONSE = 2;

    Byte MESSAGE_REQUEST = 3;

    Byte MESSAGE_RESPONSE = 4;
}
```

判断客户端是否登录成功

在[前面一小节](#)

(<https://juejin.im/book/5b4bc28bf265da0f60130116/section>

我们在文末出了一道思考题：如何判断客户端是否已经登录？

在[客户端启动流程](#)

(<https://juejin.im/book/5b4bc28bf265da0f60130116/section>

这一章节，我们有提到可以给客户端连接，也就是 Channel 绑定属性，通过 `channel.attr(xxx).set(xx)` 的方式，那么我们是否可以在登录成功之后，给 Channel 绑定一个登录成功的标志位，然后判断是否登录成功的时候取出这个标志位就可以了呢？答案是肯定的

我们先来定义一下是否登录成功的标志位

```
public interface Attributes {  
    AttributeKey<Boolean> LOGIN =  
    AttributeKey.newInstance("login");  
}
```

然后，我们在客户端登录成功之后，给客户端绑定登录成功的标志位

ClientHandler.java

```

public void channelRead(ChannelHandlerContext
ctx, Object msg) {
    // ...
    if (loginResponsePacket.isSuccess()) {
        LoginUtil.markAsLogin(ctx.channel());
        System.out.println(new Date() + ": 客
户端登录成功");
    } else {
        System.out.println(new Date() + ": 客
户端登录失败, 原因: " +
loginResponsePacket.getReason());
    }
    // ...
}

```

这里，我们省去了非关键代码部分

```

public class LoginUtil {
    public static void markAsLogin(Channel
channel) {
        channel.attr(Attributes.LOGIN).set(true);
    }

    public static boolean hasLogin(Channel
channel) {
        Attribute<Boolean> loginAttr =
channel.attr(Attributes.LOGIN);

        return loginAttr.get() != null;
    }
}

```

如上所示，我们抽取出 LoginUtil 用于设置登录标志位以及判断是否有标志位，如果有标志位，不管标志位的值是什么，都表示已经成功登录过，接下来，我们来实现控制台输入消息并发送至服务端。

控制台输入消息并发送

在[客户端启动](#)

(<https://juejin.im/book/5b4bc28bf265da0f60130116/section>

这小节中，我们已经学到了客户端的启动流程，现在，我们在客户端连接上服务端之后启动控制台线程，从控制台获取消息，然后发送至服务端

NettyClient.java

```
private static void connect(Bootstrap bootstrap,
String host, int port, int retry) {
    bootstrap.connect(host,
port).addListener(future -> {
        if (future.isSuccess()) {
            Channel channel = ((ChannelFuture)
future).channel();
            // 连接成功之后，启动控制台线程
            startConsoleThread(channel);
        }
        // ...
    });
}

private static void startConsoleThread(Channel
channel) {
```

```

        new Thread(() -> {
            while (!Thread.interrupted()) {
                if (LoginUtil.hasLogin(channel)) {
                    System.out.println("输入消息发送至服
务端: ");
                    Scanner sc = new
Scanner(System.in);
                    String line = sc.nextLine();

                    MessageRequestPacket packet = new
MessageRequestPacket();
                    packet.setMessage(line);
                    ByteBuf byteBuf =
PacketCodeC.INSTANCE.encode(channel.alloc(),
packet);
                    channel.writeAndFlush(byteBuf);
                }
            }
        }).start();
    }
}

```

这里，我们省略了非关键代码，连接成功之后，我们调用 `startConsoleThread()` 开始启动控制台线程，然后在控制台线程中，判断只要当前 `channel` 是登录状态，就允许控制台输入消息。

从控制台获取消息之后，将消息封装成消息对象，然后将消息编码成 `ByteBuf`，最后通过 `writeAndFlush()` 将消息写到服务端，这个过程相信大家在学习了上小节的内容之后，应该不会太陌生。接下来，我们来看一下服务端收到消息之后是如何来处理的。

服务端收发消息处理

ServerHandler.java

```
public void channelRead(ChannelHandlerContext
ctx, Object msg) {
    ByteBuf requestByteBuf = (ByteBuf) msg;

    Packet packet =
PacketCodecC.INSTANCE.decode(requestByteBuf);

    if (packet instanceof LoginRequestPacket) {
        // 处理登录..
    } else if (packet instanceof
MessageRequestPacket) {
        // 处理消息
        MessageRequestPacket messageRequestPacket
= ((MessageRequestPacket) packet);
        System.out.println(new Date() + ": 收到客户
端消息: " + messageRequestPacket.getMessage());

        MessageResponsePacket
messageResponsePacket = new
MessageResponsePacket();
        messageResponsePacket.setMessage("服务端回
复【" + messageRequestPacket.getMessage() + "】");
        ByteBuf responseByteBuf =
PacketCodecC.INSTANCE.encode(ctx.alloc(),
messageResponsePacket);

ctx.channel().writeAndFlush(responseByteBuf);
    }
}
```

服务端在收到消息之后，仍然是回调到 `channelRead()` 方法，解码之后用一个 `else` 分支进入消息处理的流程。

首先，服务端将收到的消息打印到控制台，然后封装一个消息响应对象 `MessageResponsePacket`，接下来还是老样子，先编码成 `ByteBuf`，然后调用 `writeAndFlush()` 将数据写到客户端，最后，我们再来看一下客户端收到消息的逻辑。

客户端收消息处理

ClientHandler.java

```
public void channelRead(ChannelHandlerContext
ctx, Object msg) {
    ByteBuf byteBuf = (ByteBuf) msg;

    Packet packet =
PacketCodeC.INSTANCE.decode(byteBuf);

    if (packet instanceof LoginResponsePacket) {
        // 登录逻辑...
    } else if (packet instanceof
MessageResponsePacket) {
        MessageResponsePacket
messageResponsePacket = (MessageResponsePacket)
packet;
        System.out.println(new Date() + ": 收到服务
端的消息: " + messageResponsePacket.getMessage());
    }
}
```


客户端在收到消息之后，回调到 `channelRead()` 方法，仍然用一个 `else` 逻辑进入到消息处理的逻辑，这里我们仅仅是简单地打印出消息，最后，我们再来看一下服务端和客户端的运行效果

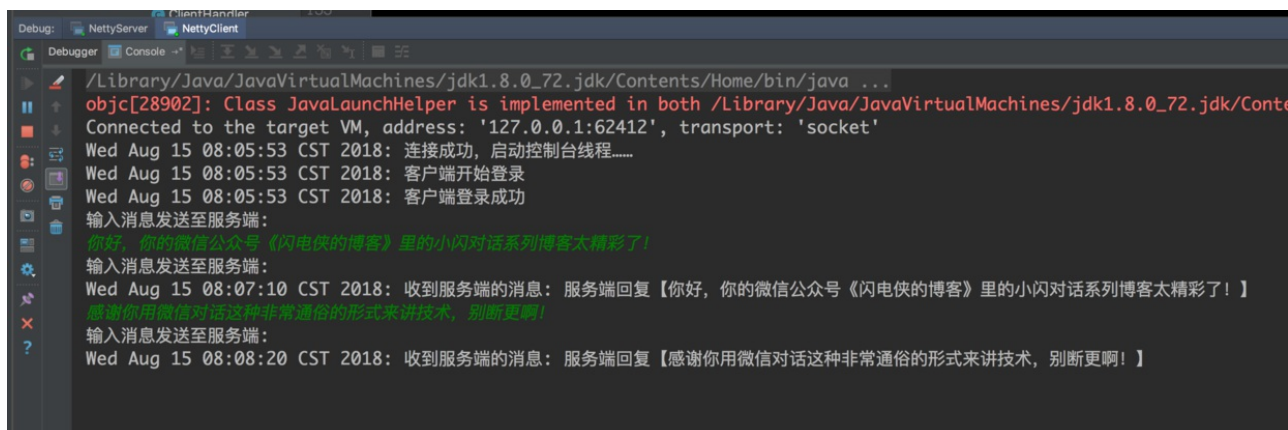
完整的代码参考 [github](#)

<https://github.com/lightningMan/flash-netty/tree/%E5%AE%9E%E7%8E%B0%E5%AE%A2%E6%88%B7>

分别启动 `NettyServer.java` 与 `NettyClient.java` 即可看到效果。

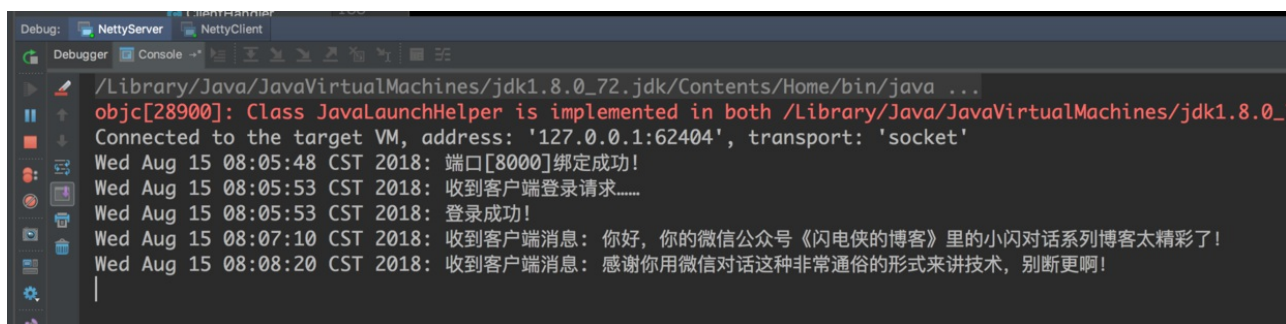
控制台输出

客户端



```
Debug: NettyServer NettyClient
Debugger Console
/Library/Java/JavaVirtualMachines/jdk1.8.0_72.jdk/Contents/Home/bin/java ...
objc[28902]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_72.jdk/Conte
Connected to the target VM, address: '127.0.0.1:62412', transport: 'socket'
Wed Aug 15 08:05:53 CST 2018: 连接成功, 启动控制台线程.....
Wed Aug 15 08:05:53 CST 2018: 客户端开始登录
Wed Aug 15 08:05:53 CST 2018: 客户端登录成功
输入消息发送至服务端:
你好, 你的微信公众号《闪电侠的博客》里的小闪对话系列博客太精彩了!
输入消息发送至服务端:
Wed Aug 15 08:07:10 CST 2018: 收到服务端的消息: 服务端回复【你好, 你的微信公众号《闪电侠的博客》里的小闪对话系列博客太精彩了!】
感谢你用微信对话这种非常通俗的形式来讲技术, 别断更啊!
输入消息发送至服务端:
Wed Aug 15 08:08:20 CST 2018: 收到服务端的消息: 服务端回复【感谢你用微信对话这种非常通俗的形式来讲技术, 别断更啊!】
```

服务端



```
Debug: NettyServer NettyClient
Debugger Console
/Library/Java/JavaVirtualMachines/jdk1.8.0_72.jdk/Contents/Home/bin/java ...
objc[28900]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_72.jdk/Contents/Home/bin/java ...
Connected to the target VM, address: '127.0.0.1:62404', transport: 'socket'
Wed Aug 15 08:05:48 CST 2018: 端口[8000]绑定成功!
Wed Aug 15 08:05:53 CST 2018: 收到客户端登录请求.....
Wed Aug 15 08:05:53 CST 2018: 登录成功!
Wed Aug 15 08:07:10 CST 2018: 收到客户端消息: 你好, 你的微信公众号《闪电侠的博客》里的小闪对话系列博客太精彩了!
Wed Aug 15 08:08:20 CST 2018: 收到客户端消息: 感谢你用微信对话这种非常通俗的形式来讲技术, 别断更啊!
```

总结

在本小节中

1. 我们定义了收发消息的 Java 对象进行消息的收发。
2. 然后我们学到了 channel 的 attr() 的实际用法: 可以通过给 channel 绑定属性来设置某些状态, 获取某些状态, 不需要额外的 map 来维持。
3. 接着, 我们学习了如何在控制台获取消息并且发送至服务端。
4. 最后, 我们实现了服务端回消息, 客户端响应的逻辑, 可以看到, 这里的部分实际上和前面一小节的登录流程有点类似。

思考

随着我们实现的指令越来越多, 如何避免 channelRead() 中对指令处理的 if else 泛滥? 欢迎留言讨论。