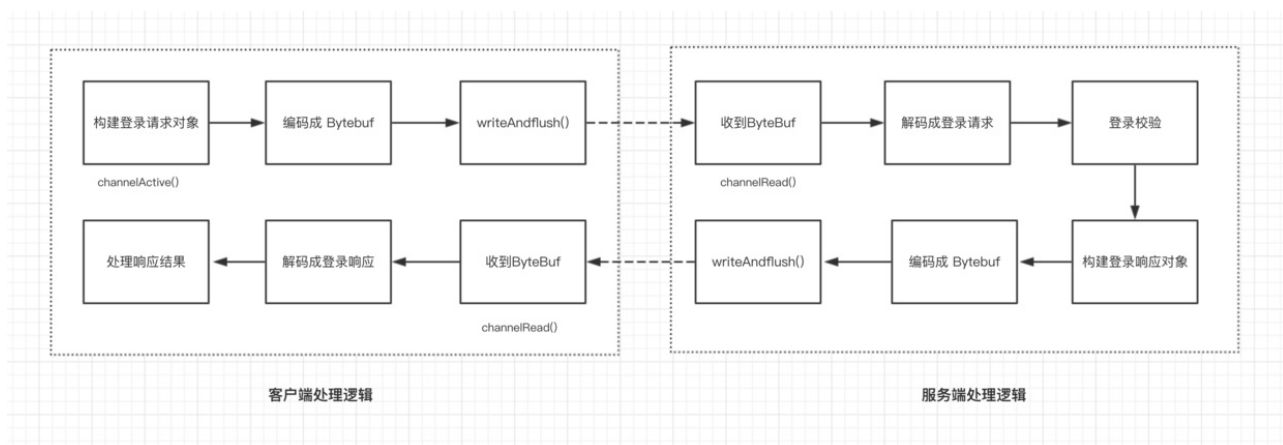


实战：Netty 实现客户端登录

本小节，我们来实现客户端登录到服务端的过程

登录流程



从上图中我们可以看到，客户端连接上服务端之后

1. 客户端会构建一个登录请求对象，然后通过编码把请求对象编码为 ByteBuf，写到服务端。
2. 服务端接收到 ByteBuf 之后，首先通过解码把 ByteBuf 解码为登录请求响应，然后进行校验。
3. 服务端校验通过之后，构造一个登录响应对象，依然经过编码，然后再写回到客户端。
4. 客户端接收到服务端的之后，解码 ByteBuf，拿到登录响应响应，判断是否登陆成功

逻辑处理器

接下来，我们分别实现一下上述四个过程，开始之前，我们先来回顾一下客户端与服务端的启动流程，客户端启动的时候，我们会在引导类 Bootstrap 中配置客户端的处理逻辑，本小节中，我们给客户端配置的逻辑处理器叫做 ClientHandler

```
public class ClientHandler extends  
ChannelInboundHandlerAdapter {  
}
```

然后，客户端启动的时候，我们给 Bootstrap 配置上这个逻辑处理器

```
bootstrap.handler(new  
ChannelInitializer<SocketChannel>() {  
    @Override  
    public void initChannel(SocketChannel  
ch) {  
        ch.pipeline().addLast(new  
ClientHandler());  
    }  
});
```

这样，在客户端侧，Netty 中 IO 事件相关的回调就能够回调到我们的 ClientHandler。

同样，我们给服务端引导类 ServerBootstrap 也配置一个逻辑处理器 ServerHandler

```
public class ServerHandler extends
ChannelInboundHandlerAdapter {
}

serverBootstrap.childHandler(new
ChannelInitializer<NioSocketChannel>() {
    protected void
initChannel(NioSocketChannel ch) {
        ch.pipeline().addLast(new
ServerHandler());
    }
}
```

这样，在服务端侧，Netty 中 IO 事件相关的回调就能够回调到我们的 ServerHandler。

接下来，我们就围绕这两个 Handler 来编写我们的处理逻辑。

客户端发送登录请求

客户端处理登录请求

我们实现在客户端连接上服务端之后，立即登录。在连接上服务端之后，Netty 会回调到 ClientHandler 的 channelActive() 方法，我们在这个方法体里面编写相应的逻辑

```
ClientHandler.java
```

```
public void channelActive(ChannelHandlerContext
ctx) {
    System.out.println(new Date() + ": 客户端开始登
录");

    // 创建登录对象
    LoginRequestPacket loginRequestPacket = new
LoginRequestPacket();

loginRequestPacket.setUserId(UUID.randomUUID().to
String());
    loginRequestPacket.setUsername("flash");
    loginRequestPacket.setPassword("pwd");

    // 编码
    ByteBuf buffer =
PacketCodecC.INSTANCE.encode(ctx.alloc(),
loginRequestPacket);

    // 写数据
    ctx.channel().writeAndFlush(buffer);
}
```

这里，我们按照前面所描述的三个步骤来分别实现，在编码的环节，我们把 PacketCodecC 变成单例模式，然后把 ByteBuf 分配器抽取出一个参数，这里第一个实参 ctx.alloc() 获取的就是与当前连接相关的 ByteBuf 分配器，建议这样来使用。

写数据的时候，我们通过 ctx.channel() 获取到当前连接（Netty 对连接的抽象为 Channel，后面小节会分析），然后调用 writeAndFlush() 就能把二进制数据写到服务端。这样，客户端发送登录请求的逻辑就完成了，接下来，我们来看一下，服务端接受到这个数据之后是如何来处理的。

服务端处理登录请求

ServerHandler.java

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

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

    // 判断是否是登录请求数据包
    if (packet instanceof LoginRequestPacket) {
        LoginRequestPacket loginRequestPacket =
(LoginRequestPacket) packet;

        // 登录校验
        if (valid(loginRequestPacket)) {
            // 校验成功
        } else {
            // 校验失败
        }
    }
}

private boolean valid(LoginRequestPacket
loginRequestPacket) {
    return true;
}
```

我们向服务端引导类 `ServerBootstrap` 中添加了逻辑处理器 `ServerHandler` 之后，Netty 在收到数据之后，会回调 `channelRead()` 方法，这里的第二个参数 `msg`，在我们这个场景中，可以直接强转为 `ByteBuf`，为什么 Netty 不直接把这个参数类型定义为 `ByteBuf`？我们在后续的小节会分析到。

拿到 `ByteBuf` 之后，首先要做的事情就是解码，解码出 java 数据包对象，然后判断如果是登录请求数据包 `LoginRequestPacket`，就进行登录逻辑的处理，这里，我们假设所有的登录都是成功的，`valid()` 方法返回 `true`。服务端校验通过之后，接下来就需要向客户端发送登录响应，我们继续编写服务端的逻辑。

服务端发送登录响应

服务端处理登录响应

```
ServerHandler.java
```

```
LoginResponsePacket loginResponsePacket = new
LoginResponsePacket();
loginResponsePacket.setVersion(packet.getVersion(
));
if (valid(loginRequestPacket)) {
    loginResponsePacket.setSuccess(true);
} else {
    loginResponsePacket.setReason("账号密码校验失
败");
    loginResponsePacket.setSuccess(false);
}
// 编码
ByteBuf responseByteBuf =
PacketCodec.INSTANCE.encode(ctx.alloc(),
loginResponsePacket);
ctx.channel().writeAndFlush(responseByteBuf);
```

这段逻辑仍然是在服务端逻辑处理器 `ServerHandler` 的 `channelRead()` 方法里，我们构造一个登录响应包 `LoginResponsePacket`，然后在校验成功和失败的时候分别设置标志位，接下来，调用编码器把 Java 对象编码成 `ByteBuf`，调用 `writeAndFlush()` 写到客户端，至此，服务端的登录逻辑编写完成，接下来，我们还有最后一步，客户端处理登录响应。

客户端处理登录响应

ClientHandler.java

客户端接收服务端数据的处理逻辑也是在 `ClientHandler` 的 `channelRead()` 方法

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

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

    if (packet instanceof LoginResponsePacket) {
        LoginResponsePacket loginResponsePacket =
(LoginResponsePacket) packet;

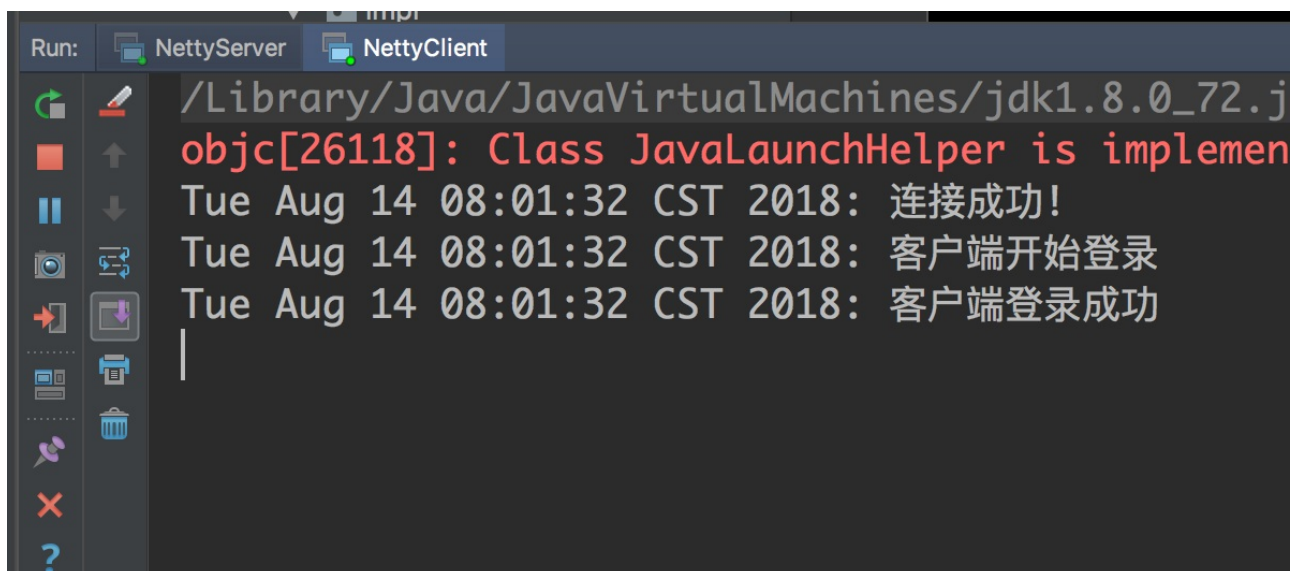
        if (loginResponsePacket.isSuccess()) {
            System.out.println(new Date() + ": 客
户端登录成功");
        } else {
            System.out.println(new Date() + ": 客
户端登录失败, 原因: " +
loginResponsePacket.getReason());
        }
    }
}
```

客户端拿到数据之后，调用 PacketCodecC 进行解码操作，如果类型是登录响应数据包，我们这里逻辑比较简单，在控制台打印出一条消息。

至此，客户端整个登录流程到这里就结束了，这里为了给大家演示，我们的客户端和服务端的处理逻辑较为简单，但是相信大家应该已经掌握了使用 Netty 来做服务端与客户端交互的基本思路，基于这个思路，再运用到实际项目中，并不是难事。

服务端

客户端



```
Run: NettyServer NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_72.j
objc[26118]: Class JavaLaunchHelper is implemen
Tue Aug 14 08:01:32 CST 2018: 连接成功!
Tue Aug 14 08:01:32 CST 2018: 客户端开始登录
Tue Aug 14 08:01:32 CST 2018: 客户端登录成功
|
```

总结

本小节，我们梳理了一下客户端登录的基本流程，然后结合上一小节的编解码逻辑，我们使用 Netty 实现了完整的客户端登录流程。

思考

客户端登录成功或者失败之后，如果把成功或者失败的标识绑定在客户端的连接上？服务端又是如何高效避免客户端重新登录？欢迎留言讨论。