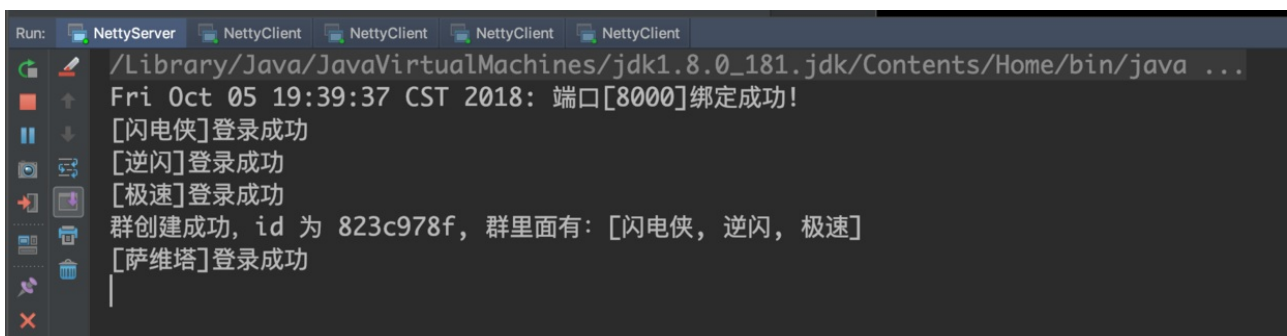


上一小节，我们已经学习了如何创建群聊并通知到群聊的各位成员。本小节，我们来实现群成员管理，包括群的加入退出，获取成员列表两大功能。有了前面两小节的基础，相信本小节的内容对你来说会比较简单。

在开始之前，我们依然是先来看一下最终的效果。

1. 最终效果

服务端



```
Run: NettyServer NettyClient NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 19:39:37 CST 2018: 端口[8000]绑定成功!
[闪电侠]登录成功
[逆闪]登录成功
[极速]登录成功
群创建成功, id 为 823c978f, 群里面有: [闪电侠, 逆闪, 极速]
[萨维塔]登录成功
```

从服务端可以看到，闪电侠、逆闪、极速先后登录到服务器，然后随后，闪电侠创建一个群聊，接下来，萨维塔也登录了。这里，客户端我们只展示闪电侠和萨维塔的控制台界面

客户端 - 闪电侠

```
Run: NettyServer NettyClient NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 19:39:43 CST 2018: 连接成功, 启动控制台线程.....
输入用户名登录: 闪电侠
[闪电侠]登录成功, userId 为: 57dabe0a
createGroup                                1.闪电侠邀请逆闪和极速加入群聊
【拉入群聊】输入 userId 列表, userId 之间英文逗号隔开: 57dabe0a,b73cc4e2,4221d994
群创建成功, id 为「823c978f」, 群里面有: 「闪电侠, 逆闪, 极速」
listGroupMembers                          3.萨维塔加入群聊之后群成员列表
输入 groupId, 获取群成员列表: 823c978f
群「823c978f」中的人包括: [4221d994:极速, f75e920f:萨维塔, 57dabe0a:闪电侠, b73cc4e2:逆闪]
listGroupMembers                          5.萨维塔离开群聊之后群成员列表
输入 groupId, 获取群成员列表: 823c978f
群「823c978f」中的人包括: [4221d994:极速, 57dabe0a:闪电侠, b73cc4e2:逆闪]
```

客户端 - 萨维塔

```
Run: NettyServer NettyClient NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 19:39:56 CST 2018: 连接成功, 启动控制台线程.....
输入用户名登录: 萨维塔
[萨维塔]登录成功, userId 为: f75e920f
joinGroup                                  2.萨维塔加入群聊
输入 groupId, 加入群聊: 823c978f
加入群「823c978f」成功!
quitGroup                                  4.萨维塔退出群聊
输入 groupId, 退出群聊: 823c978f
退出群聊「823c978f」成功!
```

我们可以看到最终效果是四位用户登录成功之后

1. 闪电侠先拉逆闪和极速加入了群聊，控制台输出群创建成功的信息。
2. 随后在萨维塔的控制台输入 "joinGroup" 之后再输入群聊的 id，加入群聊，控制台显示加入群成功。
3. 在闪电侠的控制台输入 "listGroupMembers" 之后再输入群聊的 id，展示了当前群聊成员包括了极速、萨维塔、闪电侠、逆闪。
4. 萨维塔的控制台输入 "quitGroup" 之后再输入群聊的 id，退出群聊，控制台显示退群成功。
5. 最后在闪电侠的控制台输入 "listGroupMembers" 之后再输入群聊的 ID，展示了当前群聊成员已无萨维塔。

接下来，我们就来实现加入群聊，退出群聊，获取成员列表三大功能。

2. 群的加入

2.1 控制台添加群加入命令处理器

JoinGroupConsoleCommand.java

```
public class JoinGroupConsoleCommand implements
ConsoleCommand {
    @Override
    public void exec(Scanner scanner, Channel
channel) {
        JoinGroupRequestPacket
joinGroupRequestPacket = new
JoinGroupRequestPacket();

        System.out.print("输入 groupId, 加入群
聊: ");
        String groupId = scanner.next();

        joinGroupRequestPacket.setGroupId(groupId);
        channel.writeAndFlush(joinGroupRequestPacket);
    }
}
```

按照前面两小节的套路，我们在控制台先添加群加入命令处理器 JoinGroupConsoleCommand，在这个处理器中，我们创建一个指令对象 JoinGroupRequestPacket，填上群 id 之后，将数据包发送至服务端。之后，我们将该控制台指令添加到 ConsoleCommandManager。

ConsoleCommandManager.java

```
public class ConsoleCommandManager implements
ConsoleCommand {

    public ConsoleCommandManager() {
        // ...
        consoleCommandMap.put("joinGroup", new
JoinGroupConsoleCommand());
        // ...
    }
}
```

接下来，就轮到服务端来处理加群请求了。

2.2 服务端处理群加入请求

服务端的 pipeline 中添加对应的 handler – JoinGroupRequestHandler

NettyServer.java

```
.childHandler(new
ChannelInitializer<NioSocketChannel>() {
    protected void initChannel(NioSocketChannel
ch) {
        // 添加加群请求处理器
        ch.pipeline().addLast(new
JoinGroupRequestHandler());
        // ..
    }
});
```

JoinGroupRequestHandler 的具体逻辑为

```
JoinGroupRequestHandler.java
```

```

public class JoinGroupRequestHandler extends
SimpleChannelInboundHandler<JoinGroupRequestPacket
> {
    @Override
    protected void
channelRead0(ChannelHandlerContext ctx,
JoinGroupRequestPacket requestPacket) {
        // 1. 获取群对应的 channelGroup, 然后将当前用
户的 channel 添加进去
        String groupId =
requestPacket.getGroupId();
        ChannelGroup channelGroup =
SessionUtil.getChannelGroup(groupId);
        channelGroup.add(ctx.channel());

        // 2. 构造加群响应发送给客户端
        JoinGroupResponsePacket responsePacket =
new JoinGroupResponsePacket();

        responsePacket.setSuccess(true);
        responsePacket.setGroupId(groupId);

ctx.channel().writeAndFlush(responsePacket);
    }
}

```

1. 首先，通过 groupId 拿到对应的 ChannelGroup，之后，只需要调用 ChannelGroup.add() 方法，将加入群聊的用户的 channel 添加进去，服务端即完成了加入群聊的逻辑。
2. 然后，构造一个加群响应，填入 groupId 之后，调用 writeAndFlush() 发送给加入群聊的客户端。

2.3 客户端处理群加入响应

我们在客户端的 pipeline 中添加对应的 handler –
JoinGroupResponseHandler 来处理加群之后的响应

NettyClient.java

```
.handler(new ChannelInitializer<SocketChannel>()  
{  
    @Override  
    public void initChannel(SocketChannel ch) {  
        // 添加加群响应处理器  
        ch.pipeline().addLast(new  
JoinGroupResponseHandler());  
        // ...  
    }  
});
```

JoinGroupResponseHandler 对应的逻辑为

JoinGroupResponseHandler.java

```
public class JoinGroupResponseHandler extends
SimpleChannelInboundHandler<JoinGroupResponsePack
et> {
    protected void
channelRead0(ChannelHandlerContext ctx,
JoinGroupResponsePacket responsePacket) {
        if (responsePacket.isSuccess()) {
            System.out.println("加入群[" +
responsePacket.getGroupId() + "]成功!");
        } else {
            System.err.println("加入群[" +
responsePacket.getGroupId() + "]失败, 原因为: " +
responsePacket.getReason());
        }
    }
}
```

该处理器的逻辑很简单，只是简单的将加群的结果输出到控制台，实际生产环境 IM 可能比这个要复杂，但是修改起来也是非常容易的。至此，加群相关的逻辑到这里就结束了。

3. 群的退出

关于群的退出和群的加入逻辑非常类似，这里展示一下关键代码，完整代码请参考 [github \(https://github.com/lightningMan/flash-netty\)](https://github.com/lightningMan/flash-netty) 对应本小节分支

服务端退群的核心逻辑为 QuitGroupRequestHandler

```
QuitGroupRequestHandler.java
```



```

public class QuitGroupRequestHandler extends
SimpleChannelInboundHandler<QuitGroupRequestPacket
> {
    @Override
    protected void
channelRead0(ChannelHandlerContext ctx,
QuitGroupRequestPacket requestPacket) {
        // 1. 获取群对应的 channelGroup, 然后将当前用
户的 channel 移除
        String groupId =
requestPacket.getGroupId();
        ChannelGroup channelGroup =
SessionUtil.getChannelGroup(groupId);
        channelGroup.remove(ctx.channel());

        // 2. 构造退群响应发送给客户端
        QuitGroupResponsePacket responsePacket =
new QuitGroupResponsePacket();

responsePacket.setGroupId(requestPacket.getGroupI
d());
        responsePacket.setSuccess(true);

ctx.channel().writeAndFlush(responsePacket);

    }
}

```

从上面代码其实可以看到, QuitGroupRequestHandler 和 JoinGroupRequestHandler 其实是一个逆向的过程

1. 首先, 通过 groupId 拿到对应的 ChannelGroup, 之后, 只

需要调用 `ChannelGroup.remove()` 方法，将当前用户的 `channel` 删除，服务端即完成了退群的逻辑。

2. 然后，构造一个退群响应，填入 `groupId` 之后，调用 `writeAndFlush()` 发送给退群的客户端。

至此，加群和退群的逻辑到这里就结束了，最后，我们来看一下获取成员列表的逻辑。

4. 获取成员列表

4.1 控制台添加获取群列表命令处理器

```
ListGroupMembersConsoleCommand.java
```

```
public class ListGroupMembersConsoleCommand
implements ConsoleCommand {

    @Override
    public void exec(Scanner scanner, Channel
channel) {
        ListGroupMembersRequestPacket
listGroupMembersRequestPacket = new
ListGroupMembersRequestPacket();

        System.out.print("输入 groupId, 获取群成员列
表: ");
        String groupId = scanner.next();

listGroupMembersRequestPacket.setGroupId(groupId)
;

channel.writeAndFlush(listGroupMembersRequestPack
et);
    }
}
```

依旧按照一定的套路，我们在控制台先添加获取群列表命令处理器 ListGroupMembersConsoleCommand，在这个处理器中，我们创建一个指令对象 ListGroupMembersRequestPacket，填上群 id 之后，将数据包发送至服务端。之后，我们将该控制台指令添加到 ConsoleCommandManager。

ConsoleCommandManager.java

```

public class ConsoleCommandManager implements
ConsoleCommand {

    public ConsoleCommandManager() {
        // ...
        consoleCommandMap.put("listGroupMembers",
new ListGroupMembersConsoleCommand());
        // ...
    }
}

```

接着，轮到服务端来处理获取成员列表请求。

4.2 服务端处理获取成员列表请求

服务端的 pipeline 中添加对应的 handler –
ListGroupMembersRequestHandler

NettyServer.java

```

.childHandler(new
ChannelInitializer<NioSocketChannel>() {
    protected void initChannel(NioSocketChannel
ch) {
        // 添加获取群成员请求处理器
        ch.pipeline().addLast(new
ListGroupMembersRequestHandler());
        // ..
    }
});

```

ListGroupMembersRequestHandler 的具体逻辑为

ListGroupMembersRequestHandler.java

```
public class ListGroupMembersRequestHandler
extends
SimpleChannelInboundHandler<ListGroupMembersRequestPacket> {
    protected void
channelRead0(ChannelHandlerContext ctx,
JoinGroupRequestPacket requestPacket) {
        // 1. 获取群的 ChannelGroup
        String groupId =
requestPacket.getGroupId();
        ChannelGroup channelGroup =
SessionUtil.getChannelGroup(groupId);

        // 2. 遍历群成员的 channel, 对应的 session,
构造群成员的信息
        List<Session> sessionList = new
ArrayList<>();
        for (Channel channel : channelGroup) {
            Session session =
SessionUtil.getSession(channel);
            sessionList.add(session);
        }

        // 3. 构建获取成员列表响应写回到客户端
        ListGroupMembersResponsePacket
responsePacket = new
ListGroupMembersResponsePacket();
```

```
        responsePacket.setGroupId(groupId);  
responsePacket.setSessionList(sessionList);  
ctx.channel().writeAndFlush(responsePacket);  
    }  
}
```

1. 首先，我们通过 `groupId` 拿到对应的 `ChannelGroup`。
2. 接着，我们创建一个 `sessionList` 用来装载群成员信息，我们遍历 `channel` 的每个 `session`，把对应的用户信息装到 `sessionList` 中，实际生产环境中，这里可能会构造另外一个对象来装载用户信息而非 `Session`，这里我们就简单粗暴点了，改造起来不难。
3. 最后，我们构造一个获取成员列表的响应指令数据包，填入 `groupId` 和群成员的信息之后，调用 `writeAndFlush()` 发送给发起获取成员列表的客户端。

最后，就剩下客户端来处理获取群成员列表的响应了。

4.3 客户端处理获取成员列表响应

套路和前面一样，我们在客户端的 `pipeline` 中添加一个 `handler - ListGroupMembersResponseHandler`

```
NettyClient.java
```

```
.handler(new ChannelInitializer<SocketChannel>()
{
    public void initChannel(SocketChannel ch) {
        // ...
        // 添加获取群成员响应处理器
        ch.pipeline().addLast(new
ListGroupMembersResponseHandler());
        // ...
    }
});
```

而我们这里 ListGroupMembersResponseHandler 的逻辑也只是在控制台展示一下群成员的信息

ListGroupMembersResponseHandler.java

```
public class ListGroupMembersResponseHandler
extends
SimpleChannelInboundHandler<ListGroupMembersRespo
nsePacket> {

    protected void
channelRead0(ChannelHandlerContext ctx,
ListGroupMembersResponsePacket responsePacket) {
        System.out.println("群[" +
responsePacket.getGroupId() + "]" + "中的人包括: " +
responsePacket.getSessionList());
    }
}
```

至此，群成员加入退出，获取群成员列表对应的逻辑到这里就全部实现了，其实从这小节和前面的一两个小节大家其实可以看到，我们添加一个新功能其实是有一定的套路的，我们在最后的总结给出这个套路。

5. 总结

添加一个服务端和客户端交互的新功能只需要遵循以下的步骤：

1. 创建控制台指令对应的 `ConsoleCommand` 并添加到 `ConsoleCommandManager`。
2. 控制台输入指令和数据之后填入协议对应的指令数据包 - `xxxRequestPacket`，将请求写到服务端。
3. 服务端创建对应的 `xxxRequestPacketHandler` 并添加到服务端的 `pipeline` 中，在 `xxxRequestPacketHandler` 处理完之后构造对应的 `xxxResponsePacket` 发送给客户端。
4. 客户端创建对应的 `xxxResponsePacketHandler` 并添加到客户端的 `pipeline` 中，最后在 `xxxResponsePacketHandler` 完成响应的处理。
5. 最后，最容易忽略的一点就是，新添加 `xxxPacket` 别忘了完善编解码器 `PacketCodec` 中的 `packetTypeMap`！

思考

1. 实现以下功能：客户端加入或者退出群聊，将加入群聊的消息也通知到群聊中的其他客户端，这个消息需要和发起群聊的客户端区分开，类似 "xxx 加入群聊 yyy" 的格式。
2. 实现当一个群的人数为 0 的时候，清理掉内存中群相关的信息。

欢迎留言讨论。