

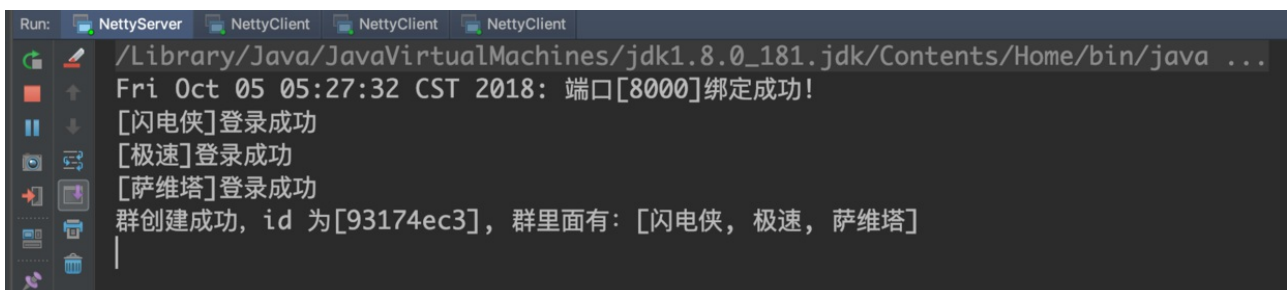
群聊的发起与通知

这小节，我们来学习一下如何创建一个群聊，并通知到群聊中的各位成员

我们依然是先来看一下最终的效果是什么样的。

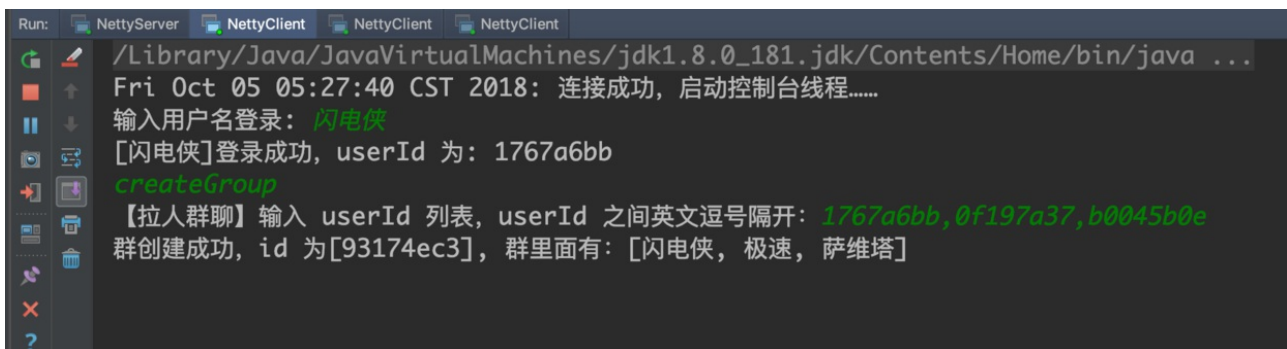
1. 最终效果

服务端



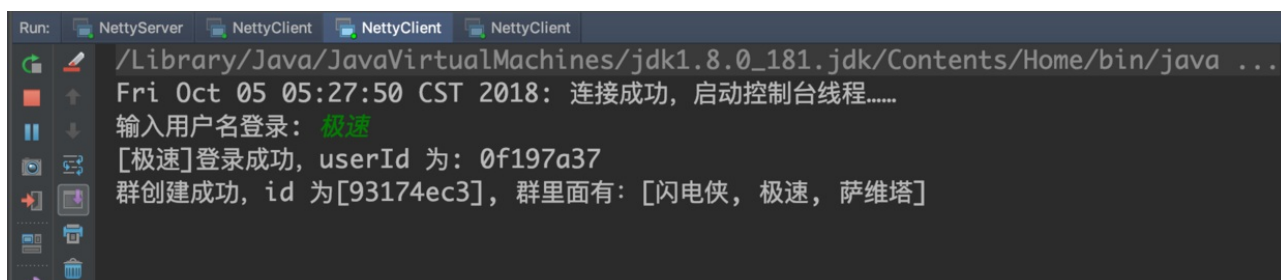
```
Run: NettyServer NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 05:27:32 CST 2018: 端口[8000]绑定成功!
[闪电侠]登录成功
[极速]登录成功
[萨维塔]登录成功
群创建成功, id 为[93174ec3], 群里面有: [闪电侠, 极速, 萨维塔]
```

创建群聊的客户端

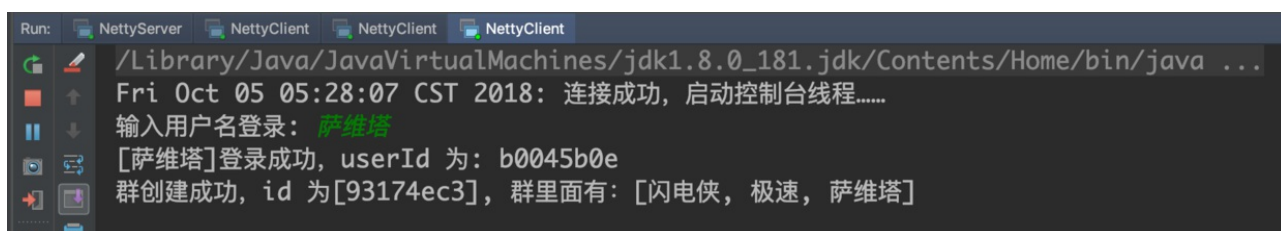


```
Run: NettyServer NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 05:27:40 CST 2018: 连接成功, 启动控制台线程.....
输入用户名登录: 闪电侠
[闪电侠]登录成功, userId 为: 1767a6bb
createGroup
【拉入群聊】输入 userId 列表, userId 之间英文逗号隔开: 1767a6bb,0f197a37,b0045b0e
群创建成功, id 为[93174ec3], 群里面有: [闪电侠, 极速, 萨维塔]
```

其他客户端



```
Run: NettyServer NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 05:27:50 CST 2018: 连接成功, 启动控制台线程.....
输入用户名登录: 极速
[极速]登录成功, userId 为: 0f197a37
群创建成功, id 为[93174ec3], 群里面有: [闪电侠, 极速, 萨维塔]
```



```
Run: NettyServer NettyClient NettyClient NettyClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Fri Oct 05 05:28:07 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: b0045b0e
群创建成功, id 为[93174ec3], 群里面有: [闪电侠, 极速, 萨维塔]
```

1. 首先，依然是三位用户依次登录到服务器，分别是闪电侠、极速、萨维塔。
2. 然后，我们在闪电侠的控制台输入 `createGroup` 指令，提示创建群聊需要输入 `userId` 列表，然后我们输入以英文逗号分隔的 `userId`。
3. 群聊创建成功之后，分别在服务端和三个客户端弹出提示信息，包括群的 ID 以及群里各位用户的昵称。

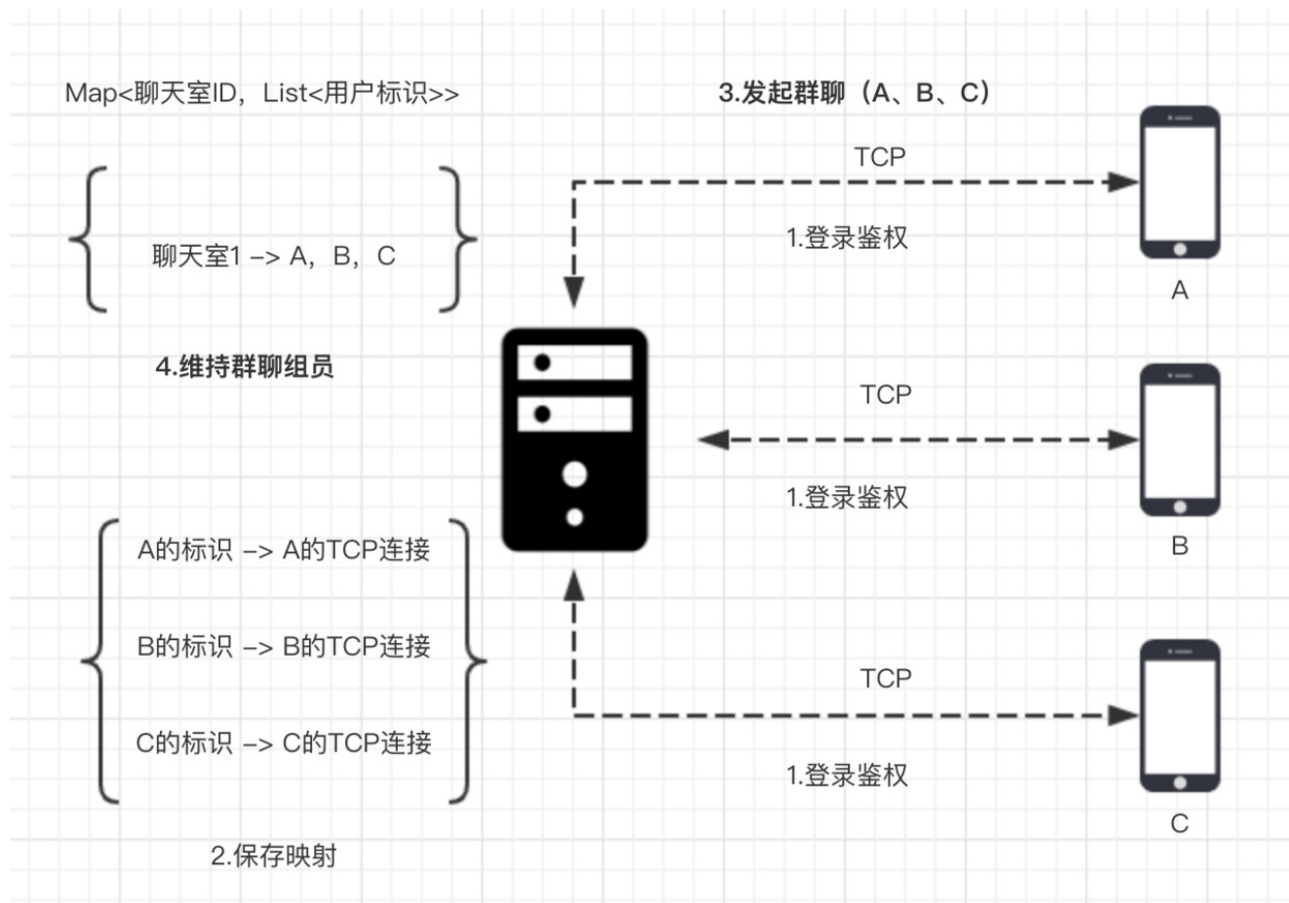
2. 群聊原理

群聊的原理我们在 [仿微信 IM 系统简介](#)

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

已经学习过，我们再来重温一下

群聊指的是一个组内多个用户之间的聊天，一个用户发到群组的消息会被组内任何一个成员接收，下面我们来看一下群聊的基本流程。



如上图，要实现群聊，其实和单聊类似

1. A, B, C 依然会经历登录流程，服务端保存用户标识对应的 TCP 连接
2. A 发起群聊的时候，将 A, B, C 的标识发送至服务端，服务端拿到之后建立一个群聊 ID，然后把这个 ID 与 A, B, C 的标识绑定
3. 群聊里面任意一方在群里聊天的时候，将群聊 ID 发送至服务端，服务端拿到群聊 ID 之后，取出对应的用户标识，遍历用户标识对应的 TCP 连接，就可以将消息发送至每一个群聊成员

这一小节，我们把重点放在创建一个群聊上，由于控制台输入的命令越来越多，因此在正式开始之前，我们先对我们的控制台程序稍作重构。

2. 控制台程序重构

2.1 创建控制台命令执行器

首先，我们把在控制台要执行的操作抽象出来，抽象出一个接口

ConsoleCommand.java

```
public interface ConsoleCommand {  
    void exec(Scanner scanner, Channel channel);  
}
```

2.2 管理控制台命令执行器

接着，我们创建一个管理类来对这些操作进行管理。

ConsoleCommandManager.java

```
public class ConsoleCommandManager implements  
ConsoleCommand {  
    private Map<String, ConsoleCommand>  
consoleCommandMap;  
  
    public ConsoleCommandManager() {  
        consoleCommandMap = new HashMap<>();  
        consoleCommandMap.put("sendToUser", new  
SendToUserConsoleCommand());  
        consoleCommandMap.put("logout", new  
LogoutConsoleCommand());  
        consoleCommandMap.put("createGroup", new  
CreateGroupConsoleCommand());  
    }  
}
```

```

@Override
public void exec(Scanner scanner, Channel
channel) {
    // 获取第一个指令
    String command = scanner.next();

    ConsoleCommand consoleCommand =
consoleCommandMap.get(command);

    if (consoleCommand != null) {
        consoleCommand.exec(scanner,
channel);
    } else {
        System.err.println("无法识别[" +
command + "]指令，请重新输入!");
    }
}
}

```

1. 我们在这个管理类中，把所有要管理的控制台指令都塞到一个 map 中。
2. 执行具体操作的时候，我们先获取控制台第一个输入的指令，这里以字符串代替，比较清晰（这里我们已经实现了上小节课后思考题中的登出操作），然后通过这个指令拿到对应的控制台命令执行器执行。

这里我们就拿创建群聊举个栗子：首先，我们在控制台输入 createGroup，然后我们按下回车，就会进入 CreateGroupConsoleCommand 这个类进行处理

CreateGroupConsoleCommand.java

```
public class CreateGroupConsoleCommand implements
ConsoleCommand {

    private static final String USER_ID_SPLITTER =
    ",";

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

        System.out.print("【拉人群聊】输入 userId 列
表, userId 之间英文逗号隔开: ");
        String userIds = scanner.next();

        createGroupRequestPacket.setUserIdList(Arrays.asL
ist(userIds.split(USER_ID_SPLITTER)));

        channel.writeAndFlush(createGroupRequestPacket);
    }
}
```

进入到 CreateGroupConsoleCommand 的逻辑之后，我们创建了一个群聊创建请求的数据包，然后提示输入以英文逗号分隔的 userId 的列表，填充完这个数据包之后，调用 writeAndFlush() 我们就可以发送一个创建群聊的指令到服务端。

最后，我们再来看一下经过我们的改造，客户端的控制台线程相关的代码。

```
NettyClient.java
```

```
private static void startConsoleThread(Channel
channel) {
    ConsoleCommandManager consoleCommandManager =
new ConsoleCommandManager();
    LoginConsoleCommand loginConsoleCommand = new
LoginConsoleCommand();
    Scanner scanner = new Scanner(System.in);

    new Thread(() -> {
        while (!Thread.interrupted()) {
            if (!SessionUtil.hasLogin(channel)) {
                loginConsoleCommand.exec(scanner,
channel);
            } else {
                consoleCommandManager.exec(scanner, channel);
            }
        }
    }).start();
}
```

抽取出控制台指令执行器之后，客户端控制台逻辑已经相对之前清晰很多了，可以非常方便地在控制台模拟各种在 IM 聊天窗口的操作，接下来，我们就来看一下如何创建群聊。

3. 创建群聊的实现

3.1 客户端发送创建群聊请求

通过我们前面讲述控制台逻辑的重构，我们已经了解到我们是发送一个 `CreateGroupRequestPacket` 数据包到服务端，这个数据包的格式为：

```
CreateGroupRequestPacket.java
```

```
public class CreateGroupRequestPacket extends
Packet {
    private List<String> userIdList;
}
```

它只包含了一个列表，这个列表就是需要拉取群聊的用户列表，接下来我们看下服务端如何处理的。

3.2 服务端处理创建群聊请求

我们依然是创建一个 handler 来处理新的指令。

```
NettyServer.java
```

```
.childHandler(new
ChannelInitializer<NioSocketChannel>() {
    protected void initChannel(NioSocketChannel
ch) {
        // ...
        // 添加一个 handler
        ch.pipeline().addLast(new
CreateGroupRequestHandler());
        // ...
    }
});
```


接下来，我们来看一下这个 handler 具体做哪些事情

CreateGroupRequestHandler.java

```
public class CreateGroupRequestHandler extends
SimpleChannelInboundHandler<CreateGroupRequestPacket> {
    @Override
    protected void
channelRead0(ChannelHandlerContext ctx,
CreateGroupRequestPacket
createGroupRequestPacket) {
        List<String> userIdList =
createGroupRequestPacket.getUserIdList();

        List<String> userNameList = new
ArrayList<>();
        // 1. 创建一个 channel 分组
        ChannelGroup channelGroup = new
DefaultChannelGroup(ctx.executor());

        // 2. 筛选出待入群聊的用户的 channel 和
userName
        for (String userId : userIdList) {
            Channel channel =
SessionUtil.getChannel(userId);
            if (channel != null) {
                channelGroup.add(channel);

            }
        }
        userNameList.add(SessionUtil.getSession(channel).
getUserName());
    }
}
```

```

    }

    // 3. 创建群聊创建结果的响应
    CreateGroupResponsePacket
createGroupResponsePacket = new
CreateGroupResponsePacket();

createGroupResponsePacket.setSuccess(true);

createGroupResponsePacket.setGroupId(IDUtil.rando
mId());

createGroupResponsePacket.setUserNameList(userNam
eList);

    // 4. 给每个客户端发送拉群通知

channelGroup.writeAndFlush(createGroupResponsePac
ket);

    System.out.print("群创建成功, id 为[" +
createGroupResponsePacket.getGroupId() + "], ");
    System.out.println("群里面有: " +
createGroupResponsePacket.getUserNameList());

    }
}

```

整个过程可以分为以下几个过程

1. 首先，我们这里创建一个 ChannelGroup。这里简单介绍一下 ChannelGroup：它可以把多个 chanel 的操作聚合在一起，可以往它里面添加删除 channel，可以进行 channel 的批量读

写，关闭等操作，详细的功能读者可以自行翻看这个接口的方法。这里我们一个群组其实就是一个 channel 的分组集合，使用 ChannelGroup 非常方便。

2. 接下来，我们遍历待入群聊的 userId，如果存在该用户，就把对应的 channel 添加到 ChannelGroup 中，用户昵称也添加到昵称列表中。
3. 然后，我们创建一个创建群聊响应的对象，其中 groupId 是随机生成的，群聊创建结果一共三个字段，这里就不展开对这个类进行说明了。
4. 最后，我们调用 ChannelGroup 的聚合发送功能，将拉群的通知批量地发送到客户端，接着在服务端控制台打印创建群聊成功的信息，至此，服务端处理创建群聊请求的逻辑结束。

我们接下来再来看一下客户端处理创建群聊响应。

3.3 客户端处理创建群聊响应

客户端依然也是创建一个 handler 来处理新的指令。

```
NettyClient.java
```

```
.handler(new ChannelInitializer<SocketChannel>()
{
    @Override
    public void initChannel(SocketChannel ch) {
        // ...
        // 添加一个新的 handler 来处理创建群聊成功响应
的指令
        ch.pipeline().addLast(new
CreateGroupResponseHandler());
        // ...
    }
});
```

然后，在我们的应用程序里面，我们仅仅是把创建群聊成功之后的具体信息打印出来。

CreateGroupResponseHandler.java

```
public class CreateGroupResponseHandler extends
SimpleChannelInboundHandler<CreateGroupResponsePa
cket> {

    @Override
    protected void
channelRead0(ChannelHandlerContext ctx,
CreateGroupResponsePacket
createGroupResponsePacket) {
        System.out.print("群创建成功, id 为[" +
createGroupResponsePacket.getGroupId() + "], ");
        System.out.println("群里面有: " +
createGroupResponsePacket.getUserNameList());
    }
}
```

在实际生产环境中，CreateGroupResponsePacket 对象里面可能有更多的信息，然后以上逻辑的处理也会更加复杂，不过我们这里已经能说明问题了。

到了这里，这小节的内容到这里就告一段落了，下小节，我们来学习群聊成员管理，包括添加删除成员，获取成员列表等等，最后，我们再对本小节内容做一下总结。

4. 总结

1. 群聊的原理和单聊类似，无非都是通过标识拿到 channel。
2. 本小节，我们重构了一下控制台的程序结构，在实际带有 UI 的 IM 应用中，我们输入的第一个指令其实就是对应我们点击 UI 的某些按钮或菜单的操作。
3. 通过 ChannelGroup，我们可以很方便地对一组 channel 进行批量操作。

5. 思考

如何实现在某个客户端拉取群聊成员的时候，不需要输入自己的用户 ID，并且展示创建群聊消息的时候，不显示自己的昵称？欢迎留言讨论。