

一、 java NIO 简介

nio 是 java New IO 的简称，在 jdk1.4 里提供的新 api。Sun 官方标榜的特性如下：

λ 为所有的原始类型提供(Buffer)缓存支持。

λ 字符集编码解码解决方案。

λ Channel: 一个新的原始 I/O 抽象。

λ 支持锁和内存映射文件的文件访问接口。

λ 提供多路(non-blocking)非阻塞式的高伸缩性网络 I/O。


关于 java NIO 的实现部分不是本文讨论的重点，有兴趣的朋友可以访问 [JAVA 夜无眠的博主 JAVA NIO 实例](#)。

二、 NIO 框架简介

在 Java 社区，最知名的开源 Java NIO 框架要属 Mina 和 Netty。实际上，Netty 的作者原来就是 Mina 作者之一，所以可以想到，Netty 和 Mina 在设计理念上会有很多共同点。而本文主要介绍的是使用 netty 搭建简单的游戏服务器，对于 netty 与 mina 的比较以及简单 netty 应用教程，将在其他文章中有所提及，敬请关注！

三、 netty 游戏框架搭建

a) ServerBootstrap——netty 框架的总入口


Java 代码 

```
1. /**
2.  * 作者:chenpeng
3.  * E-mail:46731706@qq.com
4.  * 创建时间: 2012-7-12 下午 12:22:53
5.  * 类说明 netty game
6.  */
7. public class ServerTest {
8.
9.
10. public static void main(String[] args) {
11.     DOMConfigurator.configureAndWatch("config/log4j.xml");
12.     ApplicationContext factory = new FileSystemXmlApplicationContext(
13.         new String[] { "config/propholder.xml" });
14.
15.     ServerBootstrap bootstrap = new ServerBootstrap(
16.         new NioServerSocketChannelFactory(
17.             Executors.newCachedThreadPool(),
18.             Executors.newCachedThreadPool()));
19.     ServerPipelineFactory httpServerPipelineFactory=(ServerPipelineFactory)
20.         factory.getBean("serverPipelineFactory");
21.     bootstrap.setPipelineFactory(httpServerPipelineFactory);
22.     //启动端口 8888
23.     bootstrap.bind(new InetSocketAddress(8888));
24.     System.out.print("8888 server is starting.....");
25.
26. }
```

```
27.  
28. }
```

b) ChannelPipeline

channelPipeline 是一系列 channelHandler 的集合，他参照 J2ee 中的 Intercepting Filter 模式来实现的，让用户完全掌握如果在一个 handler 中处理事件，同时让 pipeline 里面的多个 handler 可以相互交互。

Java 代码 

```
1. import org.jboss.netty.channel.ChannelPipeline;  
2. import org.jboss.netty.channel.ChannelPipelineFactory;  
3. import org.jboss.netty.channel.Channels;  
4.  
5. import com.cp.netty.coder.Decoder;  
6. import com.cp.netty.coder.Encoder;  
7.  
8. /**  
9. * 作者:chenpeng  
10. * E-mail:46731706@qq.com  
11. * 创建时间: 2012-7-12 上午 11:28:56  
12. * channelPipeline 是一系列 channelHandler 的集合，他参照 J2ee 中的 Interceptin  
13. * g Filter 模式来实现的，  
14. * 让用户完全掌握如果在一个 handler 中处理事件，同时让 pipeline 里面的多个 handler  
15. * 可以相互交互  
16. */  
17. public class ServerPipelineFactory implements ChannelPipelineFactory {  
18.     public ServerHandler serverHandler;  
19.  
20.     public ChannelPipeline getPipeline() throws Exception {  
21.         ChannelPipeline pipeLine = Channels.pipeline();  
22.         pipeLine.addLast("decoder", new Decoder(Integer.MAX_VALUE, 0, 4));  
23.         pipeLine.addLast("encoder", new Encoder(4));  
24.         pipeLine.addLast("handler", serverHandler);  
25.         return pipeLine;  
26.     }  
27.  
28.     public ServerHandler getServerHandler() {  
29.         return serverHandler;  
30.     }  
31.  
32.     public void setServerHandler(ServerHandler serverHandler) {  
33.         this.serverHandler = serverHandler;  
34.     }  
35. }
```

```
34. }
```

c) Decoder——消息解码器

Decoder 解码器继承于 FrameDecoder，FrameDecoder 是 Netty codec 包中的辅助类，它是个 ChannelUpstreamHandler，decode 方法是 FrameDecoder 子类需要实现的。在本程序采用的是 LengthFieldBasedFrameDecoder。LengthFieldBasedFrameDecoder 是基于长度字段的解码器。如果协议格式类似“内容长度”+内容、“固定头”+“内容长度”+动态内容这样的格式，就可以使用该解码器。至于其他类型的解码器，这里不再一一介绍。

Java 代码

```
1. import org.jboss.netty.buffer.ChannelBuffer;
2. import org.jboss.netty.channel.Channel;
3. import org.jboss.netty.channel.ChannelHandlerContext;
4. import org.jboss.netty.handler.codec.frame.LengthFieldBasedFrameDecoder;
5.
6.
7. /**
8.  * 作者:chenpeng
9.  * E-mail:46731706@qq.com
10. * 创建时间: 2012-7-12 上午 11:22:14
11. * 协议解码器
12. */
13. public class Decoder extends LengthFieldBasedFrameDecoder {
14.     // 第一个参数为信息最大长度，超过这个长度回报异常，
15.     // 第二参数为长度属性的起始（偏移）位，我们的协议中长度是 0 到第 3 个字节，所以这
    里写 0，
16.     // 第三个参数为“长度属性”的长度，我们是 4 个字节，所以写 4，
17.     // 第四个参数为长度调节值，在总长被定义为包含包头长度时，修正信息长度，
18.     // 第五个参数为跳过的字节数，根据需要我们跳过前 4 个字节，以便接收端直接接受到不
    含“长度属性”的内容。
19.
20.     public Decoder(int maxFrameLength, int lengthFieldOffset,
21.         int lengthFieldLength) {
22.         super(maxFrameLength, lengthFieldOffset, lengthFieldLength);
23.     }
24.
25.     @Override
26.     protected Object decode(ChannelHandlerContext ctx, Channel channel,
27.         ChannelBuffer buffer) throws Exception {
28.         ChannelBuffer bufs = (ChannelBuffer)super.decode(ctx, channel, buff
    er);
29.         return bufs;
30.     }
31. }
```

d) ServerHandler——消息分发器

在介绍这个类之前，先对几个概念进行简要说明：

1. Channel: channel 是负责数据读、写的对象。channel 是双向的，既可以 write 也可以 read。而且在 NIO 中用户不应该直接从 channel 中读写数据，而是应该通过 buffer，通过 buffer 再将数据读写到 channel 中。

一个 channel 可以提供给用户下面几个信息

(1)channel 的当前状态，比如 open 还是 closed

(2)ChannelConfig 对象，表示 channel 的一些参数，比如 bufferSize

(3)channel 支持的所有 i/o 操作（比如 read,write,connect.bind）

2. channelEvent: ChannelEvent 广义的认为 Channel 相关的事件处理。他分为 Upstream events 和 downstream events 两大块。如果以服务器端为主体，那么 client 到 server 的数据传输过程是 Upstream，而 server 到 client 的数据传输过程则是 downstream；以客户端为主体的过程正好相反。一下主要介绍以服务器端为主体的开发。

3. 常用的 Upstream events 包括

a) messageReceived: 信息被接受时 ---MessageEvent

b) exceptionCaught: 产生异常时 ---ExceptionEvent

c) channelOpen: channel 被开启时 ---ChannelStateEvent

d) channelClosed: channel 被关闭时 ---ChannelStateEvent

e) channelBound: channel 被开启并准备去连接但还未连接上的时候 ---ChannelStateEvent

f) channelUnbound: channel 被开启不准备去连接时候 ---ChannelStateEvent

g) channelConnected: channel 被连接上的时候 ---ChannelStateEvent

h) channelDisconnected:channel 连接断开的时候 ---ChannelStateEvent

在本游戏架构里，ServerHandler 扮演着创建线程、验证消息、分发消息的重要角色，程序如下：

Java 代码 

```
1. import java.util.concurrent.ConcurrentLinkedQueue;
2.
3. import org.apache.log4j.Logger;
4. import org.jboss.netty.buffer.ChannelBuffer;
5. import org.jboss.netty.channel.ChannelHandlerContext;
6. import org.jboss.netty.channel.ChannelStateEvent;
7. import org.jboss.netty.channel.ExceptionEvent;
8. import org.jboss.netty.channel.MessageEvent;
9. import org.jboss.netty.channel.SimpleChannelUpstreamHandler;
10.
11. import com.cp.game.HandlerDispatcher;
12. import com.cp.game.domain.MessageQueue;
13. import com.cp.netty.domain.GameRequest;
14.
15. /**
16. * 作者:chenpeng
```

```

17. *   E-mail:46731706@qq.com
18. *   创建时间: 2012-7-12 下午 12:02:52
19. *   游戏协议接收分发器
20. */
21. public class ServerHandler extends SimpleChannelUpstreamHandler {
22.     public Logger log = Logger.getLogger(this.getClass());
23.     public static HandlerDispatcher handlerDispatcher;
24.
25.
26.     public void init() {
27.         new Thread(handlerDispatcher).start();
28.     }
29.
30.
31.
32.     /* (non-Javadoc)
33.      * @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#channelConn
34.      *   建立一个新 channel
35.      */
36.     @Override
37.     public void channelConnected(ChannelHandlerContext ctx, ChannelStateEven
38.         t e)
39.         throws Exception {
40.         log.debug("进来一个 channel: " + ctx.getChannel().getId());
41.         MessageQueue messageQueue = new MessageQueue(
42.             new ConcurrentLinkedQueue<GameRequest>());
43.         handlerDispatcher.addMessageQueue(ctx.getChannel().getId(), messageQ
44.             ueue);
45.
46.     }
47.
48.     /* (non-Javadoc)
49.      * @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#channelDisc
50.      *   玩家主动关闭 channel
51.      */
52.     @Override
53.     public void channelDisconnected(ChannelHandlerContext ctx,
54.         ChannelStateEvent e) throws Exception {
55.         log.error("关掉一个 channel: " + ctx.getChannel().getId());

```

```

54.         handlerDispatcher.removeMessageQueue(e.getChannel().getId().toString
        ());
55.         e.getChannel().close();
56.     }
57.
58.     /* (non-Javadoc)
59.      * @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#exceptionCa
        ught(org.jboss.netty.channel.ChannelHandlerContext, org.jboss.netty.channel.
        ExceptionEvent)
60.      * 玩家被动关闭 channel
61.      */
62.     @Override
63.     public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEven
        t e)
        throws Exception {
64.         log.error("出异常啦....." + ctx.getChannel().getId());
65.         e.getCause().printStackTrace();
66.         handlerDispatcher.removeMessageQueue(e.getChannel().getId().toString
        ());
67.         e.getChannel().close();
68.     }
69.
70.
71.     /* (non-Javadoc)
72.      * @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#messageRece
        ived(org.jboss.netty.channel.ChannelHandlerContext, org.jboss.netty.channel.
        MessageEvent)
73.      * 消息接收处理器
74.      */
75.     @Override
76.     public void messageReceived(ChannelHandlerContext ctx, MessageEvent e)
        throws Exception {
77.         ChannelBuffer buffs = (ChannelBuffer) e.getMessage();
78.         buffs.skipBytes(4); // 越过 dataLength 的字节
79.         byte[] decoded = new byte[buffs.readableBytes()];
80.         buffs.readBytes(decoded);
81.         GameRequest gameRequest = new GameRequest(e.getChannel(), decode
        d);
82.
83.
84.         // 通知回调协议
85.         handlerDispatcher.addMessage(gameRequest);
86.     }
87.
88.     public HandlerDispatcher getHandlerDispatcher() {
89.         return handlerDispatcher;

```

```

90.     }
91.
92.     public void setHandlerDispatcher(HandlerDispatcher handlerDispatche
    r) {
93.         ServerHandler.handlerDispatcher = handlerDispatcher;
94.     }
95.
96. }

```

需要注意的是：**HandlerDispatcher** 是一个多线程处理器，用于处理游戏逻辑请求。这部分功能可根据用户的不同需求进行定制。

e) Encoder——消息编码器

消息编码器主要完成的是对游戏逻辑处理器返回的数据进行编码，组合成符合客户端规范的消息格式并发送。

Java 代码 ☆

```

1. import org.jboss.netty.buffer.ChannelBuffer;
2. import org.jboss.netty.buffer.ChannelBuffers;
3. import org.jboss.netty.channel.Channel;
4. import org.jboss.netty.channel.ChannelHandlerContext;
5. import org.jboss.netty.handler.codec.frame.LengthFieldPrepender;
6.
7. import com.cp.netty.domain.GameResponse;
8.
9. /**
10. * 作者:chenpeng
11. * E-mail:46731706@qq.com
12. * 创建时间: 2012-7-12 上午 11:43:11
13. * 类说明
14. */
15. public class Encoder extends LengthFieldPrepender {
16.
17.     public Encoder(int lengthFieldLength) {
18.         super(lengthFieldLength);
19.     }
20.
21.     @Override
22.     protected Object encode(ChannelHandlerContext cxt, Channel channel,
23.         Object msg) throws Exception {
24.
25.         ChannelBuffer buffer = ChannelBuffers.dynamicBuffer(channel.getConfig().getBufferFactory());
26.         GameResponse response = (GameResponse) msg;
27.         buffer.writeInt(response.getRtMessage().length+20);
28.         buffer.writeInt(response.getCommondId());

```

```

29.         buffer.writeInt(response.getPlayerId());
30.         buffer.writeInt(response.getCommandType());
31.         buffer.writeLong(response.getTime());
32.         System.out.println("send message "+response.getCommondId());
33.         buffer.writeBytes(response.getRtObj().getBytesM());
34.         return buffer;
35.
36.     }
37.
38. }

```

Java 代碼

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:aop="http://www.springframework.org/schema/aop"
5.     xmlns:tx="http://www.springframework.org/schema/tx"
6.     xsi:schemaLocation="
7.         http://www.springframework.org/schema/beans http://www.springfra
mework.org/schema/beans/spring-beans-2.5.xsd
8.         http://www.springframework.org/schema/aop http://www.springframe
work.org/schema/aop/spring-aop-2.5.xsd
9.         http://www.springframework.org/schema/tx http://www.springframew
ork.org/schema/tx/spring-tx-2.5.xsd">
10.
11.     <bean id="serverPipelineFactory"
12.         class="com.cp.netty.ServerPipelineFactory"
13.         scope="prototype">
14.         <property name="serverHandler" ref="appHandler"></property>
15.     </bean>
16.
17.     <bean id="appHandler" class="com.cp.netty.ServerHandler"
18.         init-method="init">
19.         <property name="handlerDispatcher" ref="handlerDispatcher" />
20.     </bean>
21.     <bean id="handlerDispatcher"
22.         class="com.cp.game.HandlerDispatcher" init-method="init"
23.         destroy-method="stop">
24.         <property name="messageExecutor">
25.             <bean class="com.cp.netty.domain.FiexThreadPoolExecutor"
26.                 destroy-method="shutdown">
27.                 <constructor-arg
28.                     value="${app.dispatcher.pool.corePoolSize}" />
29.                 <constructor-arg

```



```
30.         value="${app.dispatcher.pool.maximumPoolSize}" />
31.         <constructor-arg
32.             value="${app.dispatcher.pool.keepAliveSecond}" />
33.     </bean>
34. </property>
35. <property name="sleepTime" value="${app.dispatcher.sleepTime}" />
36. <property name="handlerMap" ref="serverHandlerMapping" />
37. </bean>
38.
39. <bean id="serverMainController" class="com.cp.game.ServerMainHandler"
40.     abstract="true">
41. </bean>
42.
43. <bean id="serverHandlerMapping" class="java.util.HashMap">
44.     <constructor-arg>
45.         <map>
46.             <!-- 测试协议 -->
47.             <entry key="1000">
48.                 <bean
49.                     class="com.cp.game.handler.common.InitHandler"
50.                     parent="serverMainController">
51.                 </bean>
52.             </entry>
53.         </map>
54.     </constructor-arg>
55. </bean>
56.
57. </beans>
```