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-bloking)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夜无眠的博客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  [收藏代码](javascript:void())**

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 {

10. **public** **static** **void** main(String[] args) {
11. DOMConfigurator.configureAndWatch("config/log4j.xml");
12. ApplicationContext factory = **new** FileSystemXmlApplicationContext(
13. **new** String[] { "config/propholder.xml" });
15. ServerBootstrap bootstrap = **new** ServerBootstrap(
16. **new** NioServerSocketChannelFactory(
17. Executors.newCachedThreadPool(),
18. Executors.newCachedThreadPool()));
19. ServerPipelineFactory httpServerPipelineFactory=(ServerPipelineFactory)factory.getBean("serverPipelineFactory");
20. bootstrap.setPipelineFactory(httpServerPipelineFactory);
21. //启动端口 8888
22. bootstrap.bind(**new** InetSocketAddress(8888));
23. System.out.print("8888  server is starting……");

26. }
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  [收藏代码](javascript:void())**

1. **import** org.jboss.netty.channel.ChannelPipeline;
2. **import** org.jboss.netty.channel.ChannelPipelineFactory;
3. **import** org.jboss.netty.channel.Channels;
5. **import** com.cp.netty.coder.Decoder;
6. **import** com.cp.netty.coder.Encoder;
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Intercepting Filter模式来实现的，
13. \*   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
14. \*/
15. **public** **class** ServerPipelineFactory **implements** ChannelPipelineFactory {
16. **public** ServerHandler serverHandler;
18. **public** ChannelPipeline getPipeline() **throws** Exception {
19. ChannelPipeline pipeLine = Channels.pipeline();
20. pipeLine.addLast("decoder", **new** Decoder(Integer.MAX\_VALUE, 0, 4));
21. pipeLine.addLast("encoder", **new** Encoder(4));
22. pipeLine.addLast("handler", serverHandler);
23. **return** pipeLine;
24. }
26. **public** ServerHandler getServerHandler() {
27. **return** serverHandler;
28. }
30. **public** **void** setServerHandler(ServerHandler serverHandler) {
31. **this**.serverHandler = serverHandler;
32. }
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  [收藏代码](javascript:void())**

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;

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。
20. **public** Decoder(**int** maxFrameLength, **int** lengthFieldOffset,
21. **int** lengthFieldLength) {
22. **super**(maxFrameLength, lengthFieldOffset, lengthFieldLength);
23. }
25. @Override
26. **protected** Object decode(ChannelHandlerContext ctx, Channel channel,
27. ChannelBuffer buffer) **throws** Exception {
28. ChannelBuffer buffs = (ChannelBuffer)**super**.decode(ctx, channel, buffer);
29. **return** buffs;
30. }
32. }

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  [收藏代码](javascript:void())**

1. **import** java.util.concurrent.ConcurrentLinkedQueue;
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;
11. **import** com.cp.game.HandlerDispatcher;
12. **import** com.cp.game.domain.MessageQueue;
13. **import** com.cp.netty.domain.GameRequest;
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;

26. **public** **void** init() {
27. **new** Thread(handlerDispatcher).start();
28. }


32. /\* (non-Javadoc)
33. \* @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#channelConnected(org.jboss.netty.channel.ChannelHandlerContext, org.jboss.netty.channel.ChannelStateEvent)
34. \*  建立一个新channel
35. \*/
36. @Override
37. **public** **void** channelConnected(ChannelHandlerContext ctx, ChannelStateEvent e)
38. **throws** Exception {
39. log.debug("进来一个channel：" + ctx.getChannel().getId());
40. MessageQueue messageQueue = **new** MessageQueue(
41. **new** ConcurrentLinkedQueue<GameRequest>());
42. handlerDispatcher.addMessageQueue(ctx.getChannel().getId(), messageQueue);
44. }
46. /\* (non-Javadoc)
47. \* @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#channelDisconnected(org.jboss.netty.channel.ChannelHandlerContext, org.jboss.netty.channel.ChannelStateEvent)
48. \*  玩家主动关闭channel
49. \*/
50. @Override
51. **public** **void** channelDisconnected(ChannelHandlerContext ctx,
52. ChannelStateEvent e) **throws** Exception {
53. log.error("关掉一个channel：" + ctx.getChannel().getId());
54. handlerDispatcher.removeMessageQueue(e.getChannel().getId().toString());
55. e.getChannel().close();
56. }
58. /\* (non-Javadoc)
59. \* @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#exceptionCaught(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, ExceptionEvent e)
64. **throws** Exception {
65. log.error("出异常啦……" + ctx.getChannel().getId());
66. e.getCause().printStackTrace();
67. handlerDispatcher.removeMessageQueue(e.getChannel().getId().toString());
68. e.getChannel().close();
69. }
71. /\* (non-Javadoc)
72. \* @see org.jboss.netty.channel.SimpleChannelUpstreamHandler#messageReceived(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)
77. **throws** Exception {
78. ChannelBuffer buffs = (ChannelBuffer) e.getMessage();
79. buffs.skipBytes(4);// 越过dataLength的字节
80. **byte**[] decoded = **new** **byte**[buffs.readableBytes()];
81. buffs.readBytes(decoded);
82. GameRequest gameRequest = **new** GameRequest(e.getChannel(), decoded);
84. // 通知回调协议
85. handlerDispatcher.addMessage(gameRequest);
86. }
88. **public** HandlerDispatcher getHandlerDispatcher() {
89. **return** handlerDispatcher;
90. }
92. **public** **void** setHandlerDispatcher(HandlerDispatcher handlerDispatcher) {
93. ServerHandler.handlerDispatcher = handlerDispatcher;
94. }
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  [收藏代码](javascript:void())**

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;
7. **import** com.cp.netty.domain.GameResponse;
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 {
17. **public** Encoder(**int** lengthFieldLength) {
18. **super**(lengthFieldLength);
19. }
21. @Override
22. **protected** Object encode(ChannelHandlerContext cxt, Channel channel,
23. Object msg) **throws** Exception {
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;
36. }
38. }

**Java代码  [收藏代码](javascript:void())**

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.springframework.org/schema/beans/spring-beans-2.5.xsd
8. http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
9. http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
11. <bean id="serverPipelineFactory"
12. **class**="com.cp.netty.ServerPipelineFactory"
13. scope="prototype">
14. <property name="serverHandler" ref="appHandler"></property>
15. </bean>
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>
39. <bean id="serverMainController" **class**="com.cp.game.ServerMainHandler"
40. **abstract**="true">
41. </bean>
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>
57. </beans>