基于 Java NIO 的非阻塞通信的研究与实现

Java NIO－based Non－blocking I/O and It's Implementation

封 玮　周世平　（烟台大学计算机学院　264005）

摘要：本文在研究和探讨 Java NIO 特性及其非阻塞通信工作机制的基础上，探索了实现非阻塞通信的方法、步骤。并针对一个网络应用实例，给出了非阻塞通信的 Java 实现。

关键词：非阻塞 NIO I/O、Java

1　引言

Java 平台传统的 I/O 系统基于 Byte(字节)和 Stream(数据流)。这种模式下的 I/O 操作以字节为单位——速度慢。而 JSR? 51[1]的面世，带给 Java 平台一组全新的 I/O APIs——NIO(New I/O)。NIO 系统的操作面向 Buffer(缓冲器)、Channel(通道)和 Selector(选择器)，不再是字节，这种模式利用了操作系统管理内存和文件的方式，并将一些耗时操作直接转嫁给操作系统，使 Java I/O 的速度得以提高。同时，NIO 提供了升级的套接字通道和文件 I/O 操作，提供了一个正则表达的包来支持模式匹配，提供了对字符集转换的编码器和解码器，以及优化过的文件系统支持(如文件锁定、内存映射等功能)。在 JSDK1.4[2]平台上，文件处理和网络服务器程序的性能都得到了明显的改善。但，NIO 并不是 Java.io 包的再实现或替代品，它的出现只是为了突破 Java 传统 I/O 包存在的限制，弥补 Java 传统 I/O 包的不足，满足新的应用需求。同时，它也并不是一个适于任何情况的工具。本文在研究和探讨 Java NIO 特性及其非阻塞通信工作机制的基础上，探索了实现非阻塞通信的方法、步骤。并针对一个网络应用实例，给出了非阻塞通信的 Java 实现。

2　NIO 包及其非阻塞特性

针对传统 I/O 系统工作模式的弊端，NIO 系统提出基于 Buffer(缓冲器)、Channel(通道)和 Selector(选择器)的新模式[3]。

2.1　NIO 包中三成员

Buffer(缓冲器)：Buffer 是抽象类，它及其派生出的“子类”，用以处理各种类型数据的读写以及相关的运算。每一缓冲器内部包含一个字节数组作为数据存储，实现数据的管理和运算。Java 为每一个原始数据类型定义了相应的缓冲器，它们是：ByteBuffer、CharBuffer、ShortBuffer、IntBuffer、LongBuffer、FloatBuffer 和 DoubleBuffer。NIO 还提供了一个特殊类 MappedByteBuffer 用于内存映射文件的 I/O 操作。在新模式下，NIO 支持“直接缓冲器”构造方式，利用操作系统特性和能力提高并改善 Java 传统 I/O 的性能。

Channel(通道)：Channel 是一个接口，功能类似传统 I/O 中的 Stream，但通道具有双向性，既可以读入，也可以写出。而 Stream 只能进行单向操作。Channel 共有 6 个子接口，如 ByteChannel、ReadableByteChannel、WritableByteChannel 等；NIO 中有 7 个类直接或间接实现了 Channel 接口。其中，实现了 Channel 接口的 Sockets 通道，FIlChannel 和 Pipe 的主要功能特性是：

- Sockets 通道：支持非阻塞通信、可选择通信、异步通信和套接字对等体通信；
- FileChannel：支持文件锁定、内存映射和交叉连接传输；
- Pipe：支持回送通道对，支持可选的 generic 通道。

Selector(选择器)：各类 Buffer 是数据的容器对象。各类.Channel 实现在各类 Buffer 与各类 I/O 服务间传输数据。而 Selector，负责监视着已注册的 Sockets 通道，并序列化服务器需要应答的请求。它提供各类 Channel 的状态信息，控制着各类 Channel 有效工作。

Buffers、Channels 和 Selectors 是组成 NIO 的三个重要部分，是三类新的抽象模型。它们提供了非阻塞套接字、就绪选择、文件锁定和内存映射文件等新的 I/O 能力。还提供了正则表达式、即插式字符集转换等新的非 I/O 方面的特性。得益于 Java NIO 的非阻塞 I/O 机制，我们可以非阻塞模式进行通信，可以单个线程管理大量的套接字通道，更好地利用资源。

那么，什么是阻塞通信？什么是非阻塞通信？通信应该是阻塞的还是非阻塞的？是应该首先明确的问题。

2.2　阻塞和非阻塞

Java 平台处理有关 I/O 操作的传统方式通常是：当一

个方法需要处理 I/O 有关的事务时，该方法立即被 Java 虚拟机设置成等待状态，直到有关的 I/O 操作完成。称这样的过程为阻塞 I/O。

Java NIO 中，提出一种新的 I/O 模式——非阻塞 I/O。非阻塞 I/O 的通信方式是当一个方法需要处理 I/O 有关的事务时，不要求方法等待 I/O 操作完成即可返回。从而减少了管理 I/O 连接导致的系统开销，大幅度提高了系统性能。

非阻塞通信方式，使 I/O 过程"为所欲为"。我们可以发送和读取任何能够发送/读取的信息。如果没有可以读取的东西，它就终止读操作，做其他的事情直到能够读取为止。发送数据时，该过程将试图发送所有的数据，但返回的实际发送出的内容可能是全部数据、部分数据或者根本没有发送数据。

但，阻塞与非阻塞相比也有自己的优势。比如：遇到错误控制问题的时候，可以得到更明确的标志错误的代码。

3 NIO 的非阻塞工作机制

NIO 的非阻塞 I/O 机制是围绕选择器和通道构建的。SelectableChannel 是能够实现非阻塞 I/O 和多通道 I/O 的基础类。只有 SelectableChannel 的子类（例如 ServerSocketChannel）才可以处于非阻塞模式下工作。而非 SelectableChannel 的子类（如 FileChannel），不能实现非阻塞 I/O 和多通道 I/O。非阻塞 I/O 和多通道 I/O 还必须通过 Selector 才能具体实现。SelectableChannel 及其子类的层次结构见图 1[4]。

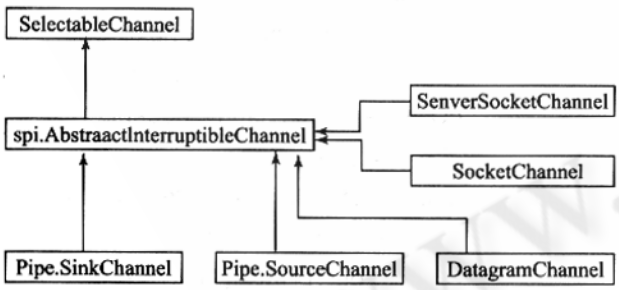

图 1　SelectableChannel 及其子类层次结构图

3.1 ServerSocketChannel 类

ServerSocketChannel 类等价于传统 I/O 中的 ServerSocket 类。常使用如下方法创建 ServerSocketChannel 类的对象：

```
ServerSocketChannel mySSChannel = ServerSocketChannel.open();
```

设置非阻塞的 I/O 工作方式：mySSChannel.configureBlocking(false)；

ServerSocketChannel 不能直接读写数据，数据的读写仍然通过 ServerSocket。调用 ServerSocketChannel.socket()方法可获得 ServerSocket。ServerSocketChannel.accept()方法返回新连接对象 SocketChannel。

3.2 SocketChannel 类

SocketChannel 类封装了用于读入用户请求和输出响应的 Socket 对象。

SocketChannel 有三种创建方式：

- new SocketChannel(SelectorProvider provider)：使用标准构造函数；
- SocketChannel.open()：使用系统缺省的 SelectorProvider，生成一无连接的 SocketChannel。须调用 connect(SocketAddress addr)来获得 Socket；
- SocketChannel.open(SocketAddress addr)：使用系统缺省的 SelectorProvider，生成一有连接的 SocketChannel，由参数 addr 指定连接。

Socket 和 SocketChannel 相互封装了对方的引用。调用 SocketChannel 类的 Socket()方法，可返回一个 Socket 对象。如有 Socket，调用 Socket 类的 getChannel()方法可返回一个 SocketChannel 对象。

3.3 Selector 类

Selector 类是 Java 实现非阻塞 I/O 和多通道 I/O 的核心。Selector 对象负责接听所有注册事件，如新用户连接，用户 Socket 读写数据。Selector 类中封装了三个维持必要信息的集合，它们是：

- key 集合：保存所有注册的通道，调用 keys()方法可获得该集合；
- 选中 key 集合：发现某通道有事件发生，此通道被选入该集合。调用 selectedKeys()可获得该集合；
- 取消选中 key 集合：当通道的事件被处理完后，此通道被置于该集合。该集合由系统管理。

我们调用 Selector 的 select()方法选出有新的事件发生的通道。

3.4 SelectionKey 类

SelectionKey 类封装了 SelectableChannel 对象在 Selector 中的注册信息。它用以区分哪一个通道何时需要何种操作。当一个或多个通道有事件生成时，每个通道相应的 SelectionKey 被放入"选中 key 集合"。集合中的元素是 SelectionKey 对象。遍历集合就可以处理每一个 SelectionKey 对象。

事件处理后，应将 SelectionKey 从 Selector Set 集合中清除，避免重复使用。

4 非阻塞通信的应用实现

非阻塞通信的应用实现关系到服务器端和客户端两方面非阻塞应用的实现。本例以乘法服务器为例演示了网络应用中,Java 非阻塞通信实现的方法和过程。

4.1 服务器端非阻塞的实现

乘法服务器端非阻塞应用的实现方法、过程如下:

```
import java.util.*; //引入 java.util 包
import java.nio.*; //引入 java.nio 包
import java.nio.channels.*; //引入 java.nio.channels 包
import java.net.*; //引入 java.net 包
import java.io.IOException; //引入 java.io.IOException 类
public class MultiplyServer{ //定义 MultiplyServer 类
    private ByteBuffer buf = ByteBuffer.allocate(8); //定义含有 8 个字节长度字节数组的缓冲器
    private IntBuffer myBuf = buf.asIntBuffer(); //将缓冲器作为整型缓冲器
    private SocketChannel cChannel = null; //声明通信用 Socket 通道
    private ServerSocketChannel sChannel = null; //声明服务器的套接字通道
    public MultiplyServer(){ //定义 MultiplyServer 类的构造函数
      try{
        openChannel(); //调用 openChannel()方法
        waitCont(); //调用 waitCont()方法
      }catch(IOException e){
        System.err.println(e.toString()); //例外处理
      }
    }
    private void openChannel()throws IOException { //定义 openChannel()方法
      sChannel = ServerSocketChannel.open(); //创建服务器的套接字通道对象
      sChannel.configureBlocking(false); //将服务器端设置成为非阻塞模式
      sChannel.socket().bind(new InetSocketAddress(10000)); //将连接绑定到主机端口 10000
      System.out.println("服务器通道已经打开");
    }
    private void waitCont() throws IOException{ //定义 waitCont()方法
      Selector mySelector = Selector.open(); //创建服务器端 Selector 的对象
      /*
      将该通道的操作类型注册给 mySelector。这就告诉 Selector,套接字要在 accept 操作发生 时被放在 ready 表上,因此,允许多元非阻塞 I/O 发生。
      */
      SelectionKey myKey = sChannel.register(mySelector, SelectionKey.OP_ACCEPT);
      int keysAdded = 0;
      while ((keysAdded = mySelector.select()) > 0) { //服务器循环等待事件发生
      //某客户已经准备好可以进行 I/O 操作了,获取其 ready 键集合
        Set readyKeys = mySelector.selectedKeys();
        Iterator i = readyKeys.iterator();
      //遍历 ready 键集合,并处理乘法请求
        while (i.hasNext()) {
          SelectionKey sk = (SelectionKey) i.next(); //移向下一个 SelectionKey
          i.remove(); //将 SelectionKey 从 Selector Set 集合中清除
          ServerSocketChannel nextReady = (ServerSocketChannel)sk.channel();
          cChannel = nextReady.accept(); //接受乘法请求并处理它
          Socket cSocket = cChannel.socket(); //创建 Socket 对象
          System.out.println("新的连接加入");
          processRequest(); //调用 processRequest()方法
          cSocket.close(); //关闭 Socket 对象
        }
      }
    }
    private void processRequest()throws IOException{ //定义 processRequest()方法
      buf.clear(); //清空缓冲器
      cChannel.read(buf); //将客户端传入的数据读
```

```
人缓冲器
        int result=myBuf.get(0)*myBuf.get(1); //分别取出缓冲器中的两个整数相乘后送 result
        buf.flip(); //重置缓冲器指针当前位置为 0
        buf.clear(); //清空缓冲器
        myBuf.put(0,result); //将 result 的值写入缓冲器 0 位置
        cChannel.write(buf); //将缓冲器内容送客户端
    }
    public static void main(String[] args){ //定义程序入口 main()方法
        new MultiplyServer(); //创建 MultiplyServer 类的实例对象
    }
}
```

4.2 客户端非阻塞的实现

客户端非阻塞应用的实现方法、过程与服务器端相似，为减少篇幅，略。

5 结语

实现非阻塞通信的优点主要有两方面：一，线程不再在通信时阻塞；二，Selector 能够处理多个连接，从而大幅降低了服务器应用程序开销。

但阻塞与非阻塞通信，哪种方式更好取决于我们应用的需求！以往我们在进行 Java 应用开发时没有过多地考虑这个问题，是因为我们只有一种选择——阻塞通信。现在，我们有了新的选择。如果我们进行同步通信，数据不必在读取任何数据之前处理的话，阻塞通信更好。而非阻塞通信则提供了处理任何已经读取的数据的机会。所以，异步通信，如 IRC 和聊天客户机则最好使用非阻塞通信，以避免冻结套接字。

参考资料

1 http://www.jcp.org/en/jsr/detail?id=051, JSR51 New I/O APIs for the JavaTM Platform, 2002 年 11 月。

2 http://java.sun.com/j2se/1.4.2/docs/api/, JSDK1.4. Sun Microsystems, 2003 年。

3 O'Reilly & Associates, Inc.. 《Java NIO》, 2002 年 8 月。

4 http://java.sun.com/j2se/1.4.2/docs/guide/nio/, New I/O APIs. Sun Microsystems, 2002 年。

5 Giuseppe? Naccarato. Introducing Nonblocking Sockets. ONJava.com, 2002 年 9 月。

6 Greg Travis. Getting started with new I/O (NIO). developerWorks, 2003 年 7 月。

7 Aruna Kalagnanam. Merlin brings nonblocking I/O to the Java platform. developerWorks, 2002 年 3 月。