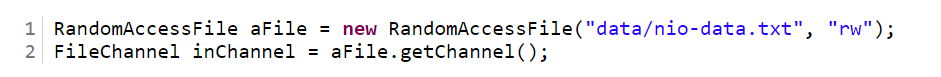
# FileChannel

## 基本使用

* 1. 简介

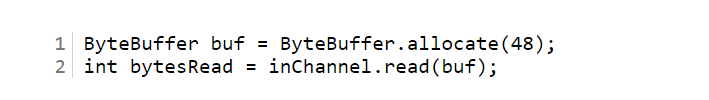
1. Java NIO 中的 FileChannel 是一个连接到文件的通道。可以通过文件通道读写文件。
2. FileChannel 无法设置为非阻塞模式，它总是运行在阻塞模式下。
   1. 打开FileChannel
3. 无法直接打开一个 FileChannel，需要通过使用一个 InputStream、OutputStream 或 RandomAccessFile 来获取一个 FileChannel 实例



* 1. 从FileChannel读取数据

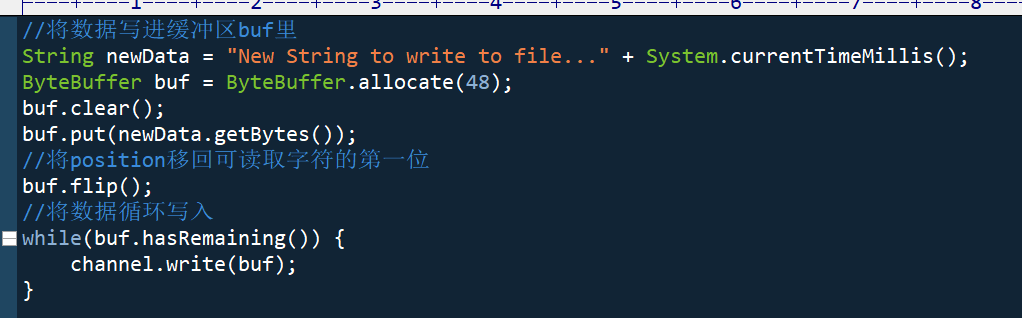
1. 首先，分配一个 Buffer。从 FileChannel 中读取的数据将被读到 Buffer 中。

然后，调用 FileChannel.read() 方法。该方法将数据从 FileChannel 读取到 Buffer 中。read() 方法返回的 int 值表示了有多少字节被读到了 Buffer 中。如果返回 - 1，表示到了文件末尾。



* 1. 向FileChannel写数据

1. 使用 FileChannel.write() 方法向 FileChannel 写数据，该方法的参数是一个 Buffer。



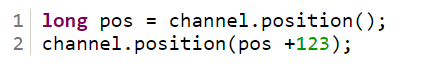
1. FileChannel.write() 是在 while 循环中调用的。**因为无法保证 write() 方法一次能向 FileChannel 写入多少字节**，因此需要重复调用 write() 方法，直到 Buffer 中已经没有尚未写入通道的字节。
   1. 关闭FileChannel

Channel.close()；

## 常用方法

* 1. position 方法

1. 有时可能需要在 FileChannel 的某个特定位置进行数据的读 / 写操作。可以通过调用 position() 方法获取 FileChannel 的当前位置。
2. 可以通过调用 position(long pos) 方法设置 FileChannel 的当前位置。



1. 如果将位置设置在文件结束符之后，然后试图从文件通道中读取数据，读方法将返回 - 1 —— 文件结束标志。
2. 如果将位置设置在文件结束符之后，然后向通道中写数据，文件将撑大到当前位置并写入数据。这可能导致 “文件空洞”，磁盘上物理文件中写入的数据间有空隙。
   1. size 方法
3. size() 方法将返回该实例所关联文件的大小

long fileSize = channel.size();

* 1. truncate 方法

1. 可以使用 FileChannel.truncate() 方法截取一个文件。截取文件时，文件将中指定长度后面的部分将被删除。

channel.truncate(1024);//截取文件的前1024字节

* 1. force 方法

1. FileChannel.force() 方法将通道里尚未写入磁盘的数据强制写到磁盘上。出于性能方面的考虑，操作系统会将数据缓存在内存中，所以无法保证写入到 FileChannel 里的数据一定会即时写到磁盘上。要保证这一点，需要调用 force() 方法。
2. force() 方法有一个 boolean 类型的参数，指明是否同时将文件元数据（权限信息等）写到磁盘上。

channel.force(true);

# 二、SocketChannel

## 简介

(1) Java NIO 中的 SocketChannel 是**一个连接到 TCP 网络套接字的通道**。

(2) 可以通过以下 2 种方式创建 SocketChannel：

打开一个 SocketChannel 并连接到互联网上的某台服务器。

一个新连接到达 ServerSocketChannel 时，会创建一个 SocketChannel。

## 基本使用

* 1. 打开 SocketChannel

1. SocketChannel socketChannel = SocketChannel.open();

socketChannel.connect(new InetSocketAddress("http://jenkov.com", 80));

直接打开一个socketChannel,并连接到某一台服务器

1. SocketChannel channel = serverSocketChannel.accept();

在服务端，一个新连接到达serverSocketChannel会创建一个socketChannel\

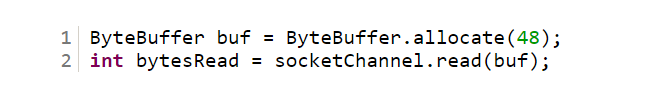
* 1. 关闭 SocketChannel

当用完 SocketChannel 之后调用 SocketChannel.close() 关闭 SocketChannel

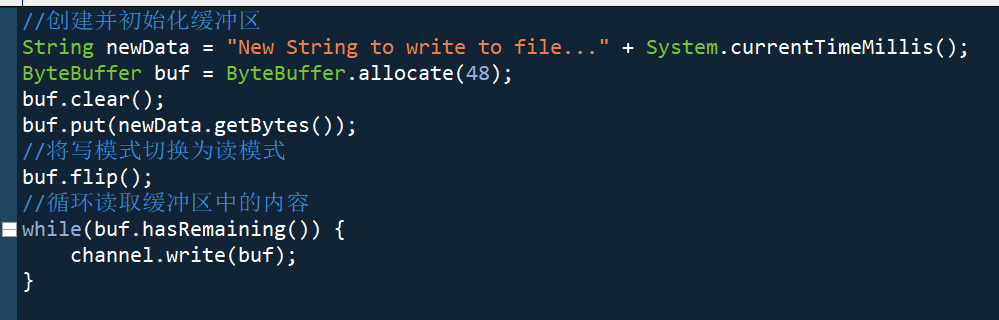
socketChannel.close()

* 1. 从 SocketChannel 读取数据

1. 从 SocketChannel 中读取数据，调用一个 read() 的方法



1. 分配一个 Buffer。从 SocketChannel 读取到的数据将会放到这个 Buffer 中。该方法将数据从 SocketChannel 读到 Buffer 中。read() 方法返回的 int 值表示读了多少字节进 Buffer 里。如果返回的是 - 1，表示已经读到了流的末尾（连接关闭了）。
   1. 写入 SocketChannel
2. 写数据到 SocketChannel 用的是 SocketChannel.write() 方法，该方法以一个 Buffer 作为参数。



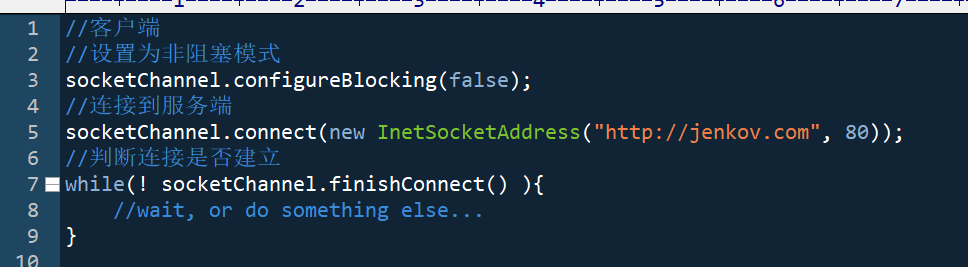
1. SocketChannel.write() 方法的调用是在一个 while 循环中的，因为Write() 方法无法保证能写多少字节到 SocketChannel。

## 非阻塞模式

可以设置 SocketChannel 为非阻塞模式（non-blocking mode）. 设置之后，就可以在异步模式下调用 connect(), read() 和 write() 了。

* 1. connect()

如果 SocketChannel 在非阻塞模式下，此时调用 connect()，该方法可能在连接建立之前就返回了。为了确定连接是否建立，可以调用 finishConnect() 的方法。



* 1. write()

非阻塞模式下，write() 方法在尚未写出任何内容时可能就返回了。所以需要在循环中调用 write()。

* 1. read()

非阻塞模式下, read() 方法在尚未读取到任何数据时可能就返回了。所以需要关注它的 int 返回值，它会告诉你读取了多少字节。

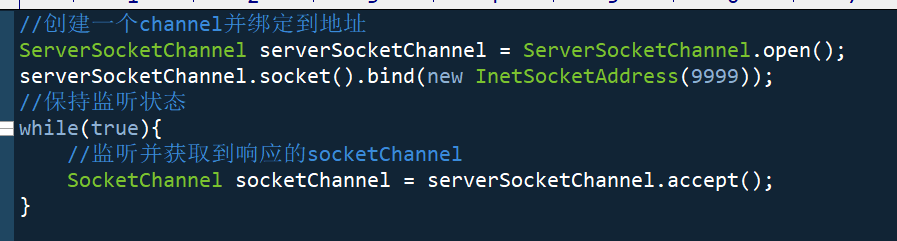
* 1. 非阻塞模式与选择器

非阻塞模式与选择器搭配会工作的更好，通过将一或多个 SocketChannel 注册到 Selector，可以询问选择器哪个通道已经准备好了读取，写入等。

# 三、ServerSocketChannel

## 简介

1. Java NIO 中的 ServerSocketChannel 是**一个可以监听新进来的 TCP 连接的通道**, 就像标准 IO 中的 ServerSocket 一样。**一般用在服务端。**
2. 实例代码



## 基本使用

* 1. 打开 ServerSocketChannel

1. 通过调用 ServerSocketChannel.open() 方法来打开 ServerSocketChannel.

ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();

* 1. 关闭 ServerSocketChannel

1. 通过调用 ServerSocketChannel.close() 方法来关闭 ServerSocketChannel.

serverSocketChannel.close();

* 1. 监听新进来的连接

1. 通过 ServerSocketChannel.accept() 方法监听新进来的连接。当 accept() 方法返回的时候, 它返回一个包含新进来的连接的 SocketChannel。因此, accept() 方法会一直阻塞到有新连接到达。



* 1. 非阻塞模式

1. ServerSocketChannel 可以设置成非阻塞模式。在非阻塞模式下，accept() 方法会立刻返回，如果还没有新进来的连接, 返回的将是 null。 因此，需要检查返回的 SocketChannel 是否是 null.
2. 一般来说，会配合选择器的阻塞方法select()进行使用



# 四、非阻塞式服务器

具体见文件 NIO非阻塞式服务器.java

# 五、DatagramChannel

## 简介

Java NIO 中的 DatagramChannel 是一个能收发 UDP 包的通道。因为 UDP 是无连接的网络协议，所以不能像其它通道那样读取和写入。它发送和接收的是数据包。

## 基本使用

* 1. 打开 DatagramChannel

1. 创建DatagramChannel并绑定到相应端口

DatagramChannel channel = DatagramChannel.open();

channel.socket().bind(new InetSocketAddress(9999));

* 1. 接收数据

1. 通过 receive() 方法从 DatagramChannel 接收数据，会将接收到的数据包内容复制到指定的 Buffer. 如果 Buffer 容不下收到的数据，多出的数据将被丢弃。

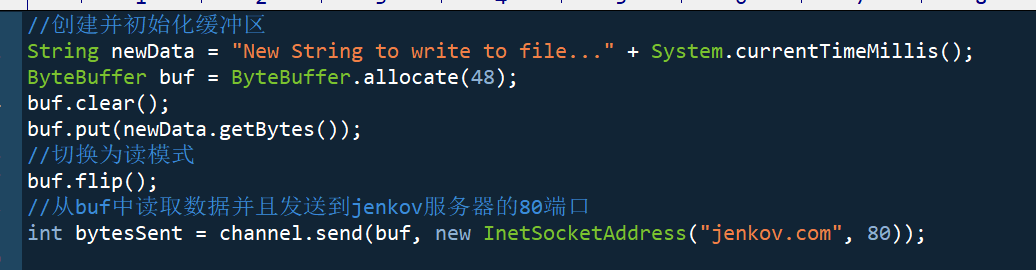
ByteBuffer buf = ByteBuffer.allocate(48);

buf.clear();

channel.receive(buf);

* 1. 发送数据

1. 通过 send() 方法从 DatagramChannel 发送数据，



* 1. 连接到特定的地址

1. 可以将 DatagramChannel“连接” 到网络中的特定地址的。由于 UDP 是无连接的，连接到特定地址并不会像 TCP 通道那样创建一个真正的连接。而是锁住 DatagramChannel ，让其只能从特定地址收发数据。
2. 像TCP连接一样可以收发数据，但是安全性没有保证。

channel.connect(new InetSocketAddress("jenkov.com", 80));

int bytesRead = channel.read(buf);

int bytesWritten = channel.write(but);