Java IO类和方法的使用

#### 创建一个文件：使用File.createNewFile()方法创建。

#### 构建文件路径：

we will show you three Java examples to construct a file path :

1. File.separator or System.getProperty(“file.separator”) (Recommended)
2. File file = new File(workingDir, filename); (Recommended)
3. Create the file separator manually. (Not recommend, just for fun)

#### 如何在Java中设置文件权限（set the file permission ）

**Check if the file permission allow** :

1. file.canExecute(); – return true, file is executable; false is not.
2. file.canWrite(); – return true, file is writable; false is not.
3. file.canRead(); – return true, file is readable; false is not.

**Set the file permission** :

1. file.setExecutable(boolean); – true, allow execute operations; false to disallow it.
2. file.setReadable(boolean); – true, allow read operations; false to disallow it.
3. file.setWritable(boolean); – true, allow write operations; false to disallow it.

#### 读取文件——BufferedInputStream（字节）、FileInputStream、DataInputStream

The readLine() from the type DataInputStream is deprecated. Sun officially announced this method can not convert property from bytes to characters. It’s advised to use BufferedReader.

#### 读取文件——BufferedReader（字符）、FileReader

**Note**  
There are many ways to read a file, but this BufferedReader is the simplest and most common-used method.

#### 写入文件——File、FileOutputStream

In Java, [FileOutputStream](http://docs.oracle.com/javase/1.4.2/docs/api/java/io/FileOutputStream.html) is a bytes stream class that’s used to handle raw binary data. To write the data to file, you have to convert the data into bytes and save it to file

#### 写入文件——FileWriter、BufferedWriter

**Note**  
The BufferedWriter is a character stream class to handle the character data. Unlike byte stream (convert data into bytes), you can just write the strings, arrays or character data directly to a file.

#### 追加文件内容

In Java, you can use FileWriter(file,true) to append new content to the end of a file.

#### 删除文件——File.delete()

No nonsense, just issue the **File.delete()** to delete a file, it will return a boolean value to indicate the delete operation status; true if the file is deleted; false if failed.

#### 删除特定后缀的文件——FilenameFilter

In Java, you can implements the [FilenameFilter](http://java.sun.com/j2se/1.4.2/docs/api/java/io/FilenameFilter.html), override the accept(File dir, String name) method, to perform the file filtering function.

In this example, we show you how to use FilenameFilter to list out all files that are end with “**.txt**” extension in folder “**c:\\folder**“, and then delete it.

#### 查找特定后缀的文件——FilenameFilter

#### 使用递归方法删除文件夹及文件夹下的文件

public static void deleteAllFilesOfDir(File path) {

    if (!path.exists())

        return;

    if (path.isFile()) {

        path.delete();

        return;

    }

    File[] files = path.listFiles();  //列出文件夹中的所以文件，并逐一删除

    for (int i = 0; i < files.length; i++) {

        deleteAllFilesOfDir(files[i]);

    }

    path.delete();

}

#### 文件重命名——renameTo()

#### 复制文件

Java didn’t comes with any ready make file copy function, you have to manual create the file copy process. To copy file, just convert the file into a bytes stream with **FileInputStream** and write the bytes into another file with **FileOutputStream**.

#### 移动文件到另一个目录

**Java.io.File** does not contains any ready make move file method, but you can workaround with the following two alternatives :

File.renameTo().

Copy to new file and delete the original file.

public static void main(String[] args) {

File file = new File("test.txt");

if(file.renameTo(new File("test\\"+file.getName()))){

System.out.println("文件移动成功");

}else{

System.out.println("移动文件失败");

}

}

#### 获取文件的创建日期

There are no official way to get the file creation date in Java. However, you can use the following workaround to get the file creation date in Windows platform.

**How it work**

In Windows command prompt, type the command to list the file creation date.

C:\>cmd /c dir c:\logfile.log /tc

Volume in drive C has no label.

Volume Serial Number is 0410-1EC3

Directory of c:\

31/05/2010 08:05 14 logfile.log

1 File(s) 14 bytes

0 Dir(s) 35,389,460,480 bytes free

The “**31/05/2010 08:05**” is what you need. The idea is use the Java “**Runtime.getRuntime().exec**” to execute the above command, hold the output, and parse it by lines until you get the date and time.

#### 获取文件的最后修改时间——File.lastModified()

In Java, you can use the **File.lastModified()** to get the file’s last modified timestamps. This method will returns the time in milliseconds (long value), you may to format it with **SimpleDateFormat** to make it a human readable format.

#### 获取文件大小

In Java, you can use the **File.length()** method to get the file size in bytes.

#### 获取一个文件的完整路径（绝对路径，包括文件名）——File.getAbsolutePath()

The **File.getAbsolutePath()** will give you the full complete path name (filepath + filename) of a file.

For example：

File file = File("C:\\abcfolder\\textfile.txt");

System.out.println("Path : " + file.getAbsolutePath());

Java IO知识点记录

#### 字符到字节的转换：InputStreamReader

对于IO操作，不管是磁盘还是网络，最终都是对字节的操作，而我们平时写的程序都是字符形式的，所以在传输的过程中需要进行转换。在字符到字节的转换过程中，我们需要用到一个类：InputStreamReader。

#### BufferedWriter和PrintWriter的区别

BufferedWriter和PrintWriter都可以写入数据，但BufferedWriter没有相关的格式化的方法，而PrintWriter可以在写入同时对写入的数据进行格式化。

#### JAVA获取当前系统时间System.currentTimeMillis()

System.currentTimeMillis()产生一个当前的毫秒，这个毫秒其实就是自1970年1月1日0时起的毫秒数，Date()其实就是相当于Date(System.currentTimeMillis());

#### ScriptEngineManager类用于支持脚本语言

JavaScript eval() 函数 ：eval() 函数可计算某个字符串，并执行其中的的 JavaScript 代码。

用法示例如：

**private** **final** **static** ScriptEngine ***jse*** = **new** ScriptEngineManager().getEngineByName("JavaScript");

**public** **static** Object **cal**(String expression) **throws** ScriptException{

**return** ***jse***.eval(expression); //

}

Java BIO知识点记录

#### ServerSocket

ServerSocket：Java中一个专门用来建立Socket服务器的类,可以用服务器需要使用的端口号作为参数来创建服务器对象。

Model：ServerSocket serverSocket=new ServerSocket(port);  
socket=serverSocket.accept();// 监听,等待连接,一旦有client端连接便创建socket实例，然后通过socket交互数据.

#### serverSocket.accept();

**serverSocket.accept()**;//serverSocket.accept();的这一方法可以说是阻塞式的,没有client端连接就一直监听着,等待连接.直到有client端连接进来才通过socket实例与client端进行交互,一个server端可以被多个client端连接,每连接一次都会创建一个socket实例,派发服务线程.  
如果把监听写进while()循环里便可实现不断的监听.

 Java NIO知识点记录

#### NIO简介：

NIO我们一般认为是New I/O（也是官方的叫法），因为它是相对于老的I/O类库新增的（其实在JDK 1.4中就已经被引入了，但这个名词还会继续用很久，即使它们在现在看来已经是“旧”的了，所以也提示我们在命名时，需要好好考虑），做了很大的改变。但民间跟多人称之为Non-block I/O，即非阻塞I/O，因为这样叫，更能体现它的特点。

#### SocketChannel和ServerSocketChannel

NIO提供了与传统BIO模型中的Socket和ServerSocket相对应的SocketChannel和ServerSocketChannel两种不同的套接字通道实现。

新增的着两种通道都支持阻塞和非阻塞两种模式。

对于低负载、低并发的应用程序，可以使用同步阻塞I/O来提升开发速率和更好的维护性；对于高负载、高并发的（网络）应用，应使用NIO的非阻塞模式来开发。

#### 缓冲区Buffer

Buffer是一个对象，包含一些要写入或者读出的数据。

在ＮIO库中，所有数据都是用缓冲区处理的。在读取数据时，它是直接读到缓冲区中的；在写入数据时，也是写入到缓冲区中。任何时候访问NIO中的数据，都是通过缓冲区进行操作。

缓冲区实际上是一个数组，并提供了对数据结构化访问以及维护读写位置等信息。

具体的缓存区有这些：ByteBuffer、CharBuffer、ShortBuffer、IntBuffer、LongBuffer、FloatBuffer、DoubleBuffer。他们实现了相同的接口：Buffer。

#### 通道Channel

我们对数据的读取和写入要通过Channel，它就像水管一样，是一个通道。通道不同于流的地方就是通道是双向的，可以用于读、写和同时读写操作。

底层的操作系统的通道一般都是全双工的，所以全双工的Channel比流能更好地映射底层操作系统的API。

Channel主要分成两大类：

SelectableChannel：用户网络读写。 FileChannel：用于文件操作。

ServerSocketChannel和SocketChannel都是SelectableChannel的子类

#### 多路复用器Selector

Selector是Java NIO 编程的基础。

Selector提供选择已经就绪的任务的能力：Selector会不断轮询注册在其上的Channel，如果某个Channel上面发生读或者写事件，这个Channel就处于就绪状态，会被Selector轮询出来，然后通过SelectionKey可以获取就绪Channel的集合，进行后续的I/O操作。

一个Selector可以同时轮询多个Channel，因为JDK使用了epoll()代替传统的select实现，所以没有最大连接句柄1024/2048的限制。所以，只需要一个线程负责Selector的轮询，就可以接入成千上万的客户端。

#### Select、Poll与Epoll比较

**select / poll / epoll 从本质上说都是非阻塞同步IO**，select会收到IO就绪的状态，然后通知用户去处理IO，实际的IO操作还需要用户等待内核复制操作。

##### 1 Select、Poll与Epoll简介

|  |  |
| --- | --- |
| Select | select本质上是通过设置或者检查存放fd标志位的[**数据结构**](http://lib.csdn.net/base/datastructure)来进行下一步处理。这样所带来的缺点是：  1 单个进程可监视的fd数量被限制  2 需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大  3 对socket进行扫描时是线性扫描 |
| Poll | poll本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态，如果设备就绪则在设备等待队列中加入一项并继续遍历，如果遍历完所有fd后没有发现就绪设备，则挂起当前进程，直到设备就绪或者主动超时，被唤醒后它又要再次遍历fd。这个过程经历了多次无谓的遍历。  它没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点：大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。  poll还有一个特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。 |
| Epoll | epoll支持水平触发和边缘触发，最大的特点在于边缘触发，它只告诉进程哪些fd刚刚变为就需态，并且只会通知一次。  在前面说到的复制问题上，epoll使用mmap减少复制开销。  还有一个特点是，epoll使用“事件”的就绪通知方式，通过epoll\_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，epoll\_wait便可以收到通知 |

注：水平触发（level-triggered）——只要满足条件，就触发一个事件(只要有数据没有被获取，内核就不断通知你)；边缘触发（edge-triggered）——每当状态变化时，触发一个事件。

##### 2 Select、Poll与Epoll区别

|  |  |  |  |
| --- | --- | --- | --- |
|  | Select | Poll | Epoll |
| 支持最大连接数 | 1024（x86） or 2048（x64） | 无上限 | 无上限 |
| IO效率 | 每次调用进行线性遍历，时间复杂度为O（N） | 每次调用进行线性遍历，时间复杂度为O（N） | 使用“事件”通知方式，每当fd就绪，系统注册的回调函数就会被调用，将就绪fd放到rdllist里面，这样epoll\_wait返回的时候我们就拿到了就绪的fd。时间发复杂度O（1） |
| fd拷贝 | 每次select都拷贝 | 每次poll都拷贝 | 调用epoll\_ctl时拷贝进内核并由内核保存，之后每次epoll\_wait不拷贝 |

#### [SelectionKey理解（总结）](http://www.cnblogs.com/burgeen/p/3618059.html)

SelectKey注册了写事件，不在合适的时间去除掉，会一直触发写事件，因为写事件是代码触发的

client.register(selector, SelectionKey.OP\_WRITE);

或者sk.interestOps(SelectionKey.OP\_WRITE)

 执行了这以上任一代码都会无限触发写事件，跟读事件不同，一定注意

nio的select()的时候，只要数据通道允许写，每次select()返回的OP\_WRITE都是true。所以在nio的写数据里面，我们在每次需要写数据之前把数据放到缓冲区，并且注册OP\_WRITE，对selector进行wakeup()，这样这一轮select()发现有OP\_WRITE之后，将缓冲区数据写入channel，清空缓冲区，并且反注册OP\_WRITE，写数据完成。

这里面需要注意的是，每个SocketChannel只对应一个SelectionKey，也就是说，在上述的注册和反注册OP\_WRITE的时候，不是通过channel.register()和key.cancel()做到的，而是通过key.interestOps()做到的。

#### [java.nio.ByteBuffer中flip、rewind、clear方法的区别](http://www.blogjava.net/sdjxsgb/archive/2013/06/18/400703.html)

对缓冲区的读写操作首先要知道缓冲区的下限、上限和当前位置。下面这些变量的值对Buffer类中的某些操作有着至关重要的作用：

1. limit：所有对Buffer读写操作都会以limit变量的值作为上限。
2. position：代表对缓冲区进行读写时，当前游标的位置。
3. capacity：代表缓冲区的最大容量（一般新建一个缓冲区的时候，limit的值和capacity的值默认是相等的）。

flip、rewind、clear这三个方法便是用来设置这些值的。

##### clear方法: clear方法将缓冲区清空，一般是在重新写缓冲区时调用。

##### flip方法:反转缓冲区。首先将限制设置为当前位置，然后将位置设置为 0。如果已定义了标记，则丢弃该标记。 常与compact方法一起使用。通常情况下，在准备从缓冲区中读取数据时调用flip方法。

Java AIO知识点记录

#### AIO简介

NIO 2.0引入了新的异步通道的概念，并提供了异步文件通道和异步套接字通道的实现。

异步的套接字通道是真正的异步非阻塞I/O，对应于UNIX网络编程中的事件驱动I/O（AIO）。他不需要过多的Selector对注册的通道进行轮询即可实现异步读写，从而简化了NIO的编程模型。

#### CountDownLatch

[CountDownLatch](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CountDownLatch.html)是一个同步工具类，它允许一个或多个线程一直等待，直到其他线程的操作执行完后再执行。

#### InetSocketAddress

netSocketAddress是SocketAddress的实现子类。

此类实现 IP 套接字地址（IP 地址 + 端口号），不依赖任何协议。  
在使用Socket来连接服务器时最简单的方式就是直接使用IP和端口，但Socket类中的connect方法并未提供这种方式，而是使用SocketAddress类来向connect方法传递服务器的IP和端口。

#### [AsynchronousServerSocketChannel](http://blog.csdn.net/tony168hongweigan/article/details/8532764)

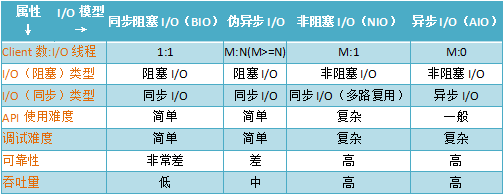
异步套接字通道

详细参考：<http://www.ibm.com/developerworks/cn/java/j-nio2-1/>

Java IO总结

#### 各种I/O的对比

先以一张表来直观的对比一下：



具体选择什么样的模型或者NIO框架，完全基于业务的实际应用场景和性能需求，如果客户端很少，服务器负荷不重，就没有必要选择开发起来相对不那么简单的NIO做服务端；相反，就应考虑使用NIO或者相关的框架了。