## NIO 2.0

**01、基石：Path**

Path 既可以表示一个目录，也可以表示一个文件，就像 File 那样——当然了，Path 就是用来取代 File 的。

1）可以通过 Paths.get() 创建一个 Path 对象，此时 Path 并没有真正在物理磁盘上创建；参数既可以是一个文件名，也可以是一个目录名；绝对路径或者相对路径均可。

2）可以通过 Files.notExists() 确认 Path（目录或者文件） 是否已经存在。

3）可以通过 Files.createDirectory() 创建目录，此时目录已经在物理磁盘上创建成功，可通过资源管理器查看到。

4）可以通过 Files.createFile() 创建文件，此时文件已经在物理磁盘上创建成功，可通过资源管理器查看到。

5）可以通过 toAbsolutePath() 查看 Path 的绝对路径。

1. 可以通过 resolve() 将 Path 连接起来，参数可以是一个新的 Path 对象，也可以是对应的字符串。

**02、处理目录**

1）Files.newDirectoryStream(Path dir, String glob) 会返回一个过滤后的 DirectoryStream（ 目录流，），第一个参数为目录，第二个参数为 glob 表达式，比如\*.txt 表示所有 txt 后缀的文件。

2）由于 DirectoryStream 继承了 Closeable 接口，所以它可以配合 try-with-resources 语法写出更安全的代码，目录流会自动调用 close 方法关闭流，释放与流相关的资源，不需要再通过 finally 进行主动关闭。

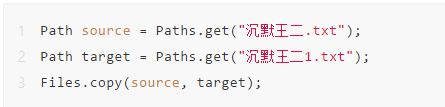
3）DirectoryStream 被称为目录流，允许方便地使用 for-each 结构来遍历目录。

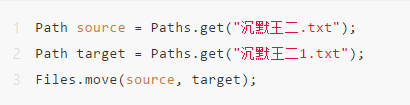
**03、处理目录树**



**04、文件的删除、复制、移动**

使用 Files.delete() 删除文件之前最好使用 Files.exists() 判断文件是否存在，否则会抛出 NoSuchFileException；Files.deleteIfExists() 则不用。





**05、快速地读写文件**

通过Files.newBufferedWriter() 获取一个文件缓冲输入流，并通过 write() 方法写入数据；然后通过 Files.newBufferedReader() 获取一个文件缓冲输出流，通过 readLine() 方法读出数据。



**06、重要：异步 I/O 操作**

异步 I/O 操作的核心概念是发起非阻塞方式的 I/O 操作，当 I/O 操作完成时通知。可以分为两种形式：Future 和 Callback。如果我们希望主线程发起 I/O 操作并轮循等待结果时，一般使用 Future 的形式；而 Callback 的基本思想是主线程派出一个侦查员（CompletionHandler）到独立的线程中执行 I/O 操作，操作完成后，会触发侦查员的 completed 或者 failed 方法。

1）Future



1. Callback

