# 1. File类

在程序运行时，变量中存储的值都是在内存中的，数据是临时的，程序一旦关闭，数据就消失。如果想永久保留数据，获得上次程序结束时的数据，那么就需要用到文件。因此掌握程序操作文件非常重要。

首先学习File类。File类用于描述文件或文件夹，提供了一些对文件或目录的基本操作。

## 1.1 体验File类

下面的程序在File的一个构造中传入了文件的真实路径，获得了一个File对象，此时File对象就是该物理文件的抽象表现形式，可以调用File对象的方法来获得文件信息。注意File对象只是抽象表现形式，只要传递合法的路径，就能得到File对象，不管硬盘上是否真的有该文件或目录。

调用的方法判断了此文件是否存在、是否是目录和是否是文件。

|  |
| --- |
| **package** com.zhang.test;  **import** java.io.File;  **public class** Demo {  **public static void** main(String[] args) {  *// Windows中，各级路径间用反斜杠 \ 分隔  // 由于 \ 是转义字符，因此再加上一个 \ 转义* File file = **new** File(**"d:\\1.txt"**);  *// Linux中路径间用 / 分隔。  // 实际上Java中也支持在Windows上用斜杠 / ，这里也可以写成 d:/1.txt* System.***out***.println(file.exists()); *// 判断文件是否存在* System.***out***.println(file.isFile()); *// 判断是否是文件* System.***out***.println(file.isDirectory()); *// 判断是否是目录   // 由于该文件不存在，因此上述的输出都是false* } } |

## 1.2 File类的构造方法

File类常用构造方法有3个：

（1）public File(String pathname)；根据文件或目录路径创建File对象。

（2）public File(String parent, String child)；根据一个路径和另一个路径得到File对象。效果就是以parent和child的混合路径创建一个对象。所以一般parent是目录，child是该目录下的文件或目录。

（3）public File(File parent, String child)；根据一个父File对象和一个子文件/目录得到File对象。此方法和上一方法类似，只不过参数1是File对象。

需要注意的是，File类并没有无参数的构造。

使用File类需要导包，即java.io.File。在操作文件中用到的所有类基本都要导入java.io包。

## 1.3 路径问题

首先，File类的对象，既可以代表文件也可以代表文件夹，因此传入的路径既能是目录，也能是文件。

在Windows中，各级目录分隔符为“\”，而在Linux中分隔符为“/”。注意字符“\”是主要转义的。为了跨平台方便，更专业的做法是不手动写目录分隔符，而是用系统提供的变量：File.separatorChar代替。该值会根据不同的操作系统得到相应的目录分隔符。例子：

|  |
| --- |
| File file = **new** File(**"d:"** + File.***separatorChar*** + **"1.txt"**); |

绝对路径和相对路径：找到一个文件既能通过绝对路径，也能通过相对路径。

绝对路径就是从根目录开始的路径，Linux系统中根目录就是“/”，比如“/home/zhang/1.txt”就是一个绝对路径。而Windows的绝对路径是从某个盘符开始的，比如“c:\1.txt”就是一个绝对路径。

相对路径是相对于当前目录的路径，即从当前目录开始找到所需文件。如果当前目录是“d:\code”，那么我们在此目录中，直接用“exam\1.txt”就能找到文件“d:\code\exam\1.txt”。在相对路径中，“..”表示上级目录，“.”表示本目录，所以用“.\exam\1.txt”和“exam\1.txt”的效果一致。“..\file1\2.txt”就进入目录“d:\file1\2.txt”。如果是Linux，那就是斜杠/，建议学习一下Linux常用操作。

执行Java程序时，当前目录为执行java命令时所在的目录。在IDE中，Java命令在src目录外执行，因此如果程序中写相对路径比如“1.txt”，那么表示此文件位置在src目录外的项目文件夹中。但是在程序中写“src/1.txt”这样也不好，因为IDE编译时，会把src下源码编译到一个专门的文件夹中，而把其他类型文件复制到该文件夹中，特点是会保持源码和本项目需要的其他文件的位置关系。至于这应该怎么使用，以后讲。

## 1.4 File类中常用方法

### 1.4.1 创建方法

（1）public boolean createNewFile()：创建该文件，若成功返回true。如果已经存在该文件，则不会再创建，并返回false。

注意，要想成功创建文件，此文件的上面各级目录必须已经存在，并不能直接创建好各级目录并创建文件，否则会抛出IOException异常，说明找不到文件（此方法本身就要处理IOException异常）。下面的方法就是用于创建目录的。

（2）public boolean mkdir()：创建文件夹。如果存在这样的文件夹，就不创建了。同样，该方法需要存在父级目录。

（3）public boolean mkdirs()：创建文件夹，如果各级父文件夹不存在，会帮你创建出来。这是终极方法，会创建路径中所有不存在的目录。

要清楚创建的是文件还是文件夹，不要调错了方法和写错了路径。

例子：

|  |
| --- |
| **package** com.zhang.test;  **import** java.io.File; **import** java.io.IOException;  **public class** Demo {  *// 需要抛出异常* **public static void** main(String[] args) **throws** IOException {  File directory = **new** File(**"d:"** + File.***separatorChar*** + **"mycode"**);  *// 创建目录* directory.mkdirs();  File file = **new** File(directory, **"1.txt"**);  *// 创建目录下的文件* file.createNewFile();  *// 现在应该存在了* System.***out***.println(file.exists());  } } |

### 1.4.2 删除和重命名（剪切）功能

（1）public boolean delete()：删除一个文件，若删除成功返回true。如果要删除一个文件夹，则此文件夹必须为空，否则不能删除返回false。程序中的删除是不经过回收站，直接从硬盘中清除。

（2）public boolean renameTo(File dest)：将该文件重命名成指定的File对象。如果dest与该文件父路径相同，就是重命名；如果他们父路径不同，就是剪切并且重命名。重命名的情况下可操作非空文件夹，若是剪切的情况，不能操作非空文件夹。

### 1.4.3 判断功能