# 第14章 I/O

I/O是输入/输出(Input/Output) 的缩写， I/O技术是非常实用的技术，如读/写文件，网络通讯等等。Java的IO支持通过java.io包下的类和接口来支持。

* I(Input输入)，从别的地方（文件、键盘、网络、内存等）读取数据到当前程序中
* O(Output输出)，从程序把数据写/输出到文件、屏幕（控制台）、网络、内存等中

## 14.1 java.io.File类

File类是java.io包下代表与平台无关的文件和目录，也就是说如果希望在程序中操作文件和目录都可以通过File类来完成，File类能新建、删除、重命名文件和目录。

在API中File的解释是文件和目录路径名的抽象表示形式，即File类是文件或目录的路径，而不是文件本身，因此File类不能直接访问文件内容本身，如果需要访问文件内容本身，则需要使用输入/输出流。

### 14.1.1 路径名

提示：File类代表磁盘或网络中某个文件或目录的路径名称，如：

/atguigu/javase/io/佟刚.jpg

但他不能直接通过File对象读取和写入数据，如果要操作数据，需要IO流。就好比“地址”不代表“水库”，要“存取”里面的水到你“家里”，需要“管道”。

File类可以使用文件路径字符串来创建File实例，该文件路径字符串既可以是绝对路径，也可以是相对路径，默认情况下，系统总是依据用户的工作路径来解释相对路径，这个路径由系统属性“user.dir”指定，通常也就是运行Java虚拟机时所作的路径。

* String getName()：返回此File对象的所表示的文件名（返回最后一级）
* String getPath()：返回此File对象所对应的路径名。
* String getAbsolutePath()：返回此File对象所对应的绝对路径名。
* File getAbsoluteFile()：返回此File对象所对应的绝对路径名所对应的File对象
* String getCanonicalPath()：返回此File对象所对应的规范路径名。所谓规范路径名，即对路径中的“..”等进行解析后的路径名。
* File getCanonicalFile()：返回此File对象所对应的规范路径名所对应的File对象
* String getParent()：返回此File对象的父目录名
* File getParentFile()：返回此File对象的父目录名所对应的File对象

|  |
| --- |
| @Test  **public** **void** test1() **throws** IOException{  File f1 = **new** File("d:\\atguigu\\javase\\HelloIO.java");  System.***out***.println("文件/目录的名称：" + f1.getName());  System.***out***.println("文件/目录的路径名：" + f1.getPath());  System.***out***.println("文件/目录的绝对路径名：" + f1.getAbsolutePath());  System.***out***.println("文件/目录的规范路径名：" + f1.getCanonicalPath());  System.***out***.println("文件/目录的父目录名：" + f1.getParent());  } |
| 文件/目录的名称：HelloIO.java  文件/目录的路径名：d:\atguigu\javase\HelloIO.java  文件/目录的绝对路径名：d:\atguigu\javase\HelloIO.java  文件/目录的规范路径名：D:\atguigu\javase\HelloIO.java  文件/目录的父目录名：d:\atguigu\javase |
| @Test  **public** **void** test2() **throws** IOException{  File f2 = **new** File("HelloIO.java");  System.out.println("user.dir =" + System.getProperty("user.dir"));  System.***out***.println("文件/目录的名称：" + f2.getName());  System.***out***.println("文件/目录的路径名：" + f2.getPath());  System.***out***.println("文件/目录的绝对路径名：" + f2.getAbsolutePath());  System.***out***.println("文件/目录的规范路径名：" + f2.getCanonicalPath());  System.***out***.println("文件/目录的父目录名：" + f2.getParent());  } |
| user.dir =D:\atguigu\temp\JDK18\test\_io  文件/目录的名称：HelloIO.java  文件/目录的路径名：HelloIO.java  文件/目录的绝对路径名：D:\atguigu\temp\JDK18\test\_io\HelloIO.java  文件/目录的规范路径名：D:\atguigu\temp\JDK18\test\_io\HelloIO.java  文件/目录的父目录名：null |
| @Test  **public** **void** test3() **throws** IOException{  File f3 = **new** File("../../HelloIO.java");  System.out.println("user.dir =" + System.getProperty("user.dir"));  System.***out***.println("文件/目录的名称：" + f3.getName());  System.***out***.println("文件/目录的路径名：" + f3.getPath());  System.***out***.println("文件/目录的绝对路径名：" + f3.getAbsolutePath());  System.***out***.println("文件/目录的规范路径名：" + f3.getCanonicalPath());  System.***out***.println("文件/目录的父目录名：" + f3.getParent());  } |
| user.dir =D:\atguigu\temp\JDK18\test\_io  文件/目录的名称：HelloIO.java  文件/目录的路径名：..\..\HelloIO.java  文件/目录的绝对路径名：D:\atguigu\temp\JDK18\test\_io\..\..\HelloIO.java  文件/目录的规范路径名：D:\atguigu\temp\HelloIO.java  文件/目录的父目录名：..\.. |

window的路径分隔符使用“\”，而Java程序中的“\”表示转义字符，所以在Windows中表示路径，需要用“\\”。或者直接使用“/”也可以，Java程序支持将“/”当成平台无关的路径分隔符。或者直接使用File.separator常量值表示。

路径中如果出现“..”表示上一级目录，路径名如果以“/”开头，表示从“根目录”下开始导航。

### 14.1.2 获取常规信息等方法

文件和目录检测方法：

* boolean exists()：判断File对象对应的文件或目录是否存在
* canRead()：判断File对象对应的文件或目录是否可读
* canWrite()：判断File对象对应的文件或目录是否可写
* isHidden()：判断File对象对应的文件或目录是否是否隐藏
* isFile()：判断File对象对应的是否是文件
* isDirectory()：判断File对象对应的是否是目录
* isAbsolute()：判断File对象对应的文件或目录是否是绝对路径

获取常规信息：

* long lastModified()：返回File对象对应的文件或目录的最后修改时间（毫秒值）
* long length()：返回File对象对应的文件的内容的长度（字节数），**如果File对象对应的是目录，则结果是不确定的**。

以上都是对应于File对象的属性信息，如果File对象对应的文件或目录不存在，返回的都是对应属性的默认值，例如：length()返回为0，isFile()和isDirectory()返回为false等。

|  |
| --- |
| @Test  **public** **void** test4(){  File file = **new** File("d:/atguigu/javase/HelloIO.java");  **long** time = file.lastModified();  SimpleDateFormat sf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss SSS");  String format = sf.format(**new** Date(time));  System.***out***.println("最后修改时间：" + format);  System.**out**.println("文件大小：" + file.length());  } |

### 14.1.3 操作文件

#### 1、创建文件

1. 创建文件：boolean createNewFile()throws IOException

如果指定的文件不存在并成功地创建，则返回 true；如果指定的文件已经存在，则返回 false。

代码示例：

|  |
| --- |
| @Test  **public** **void** test5() **throws** IOException{  File file = **new** File("d:/atguigu/javase/HelloIO.java");  file.createNewFile();  } |

（2）创建临时文件，通常会结合deleteOnExit()使用

* public static File createTempFile(String prefix,String suffix) throws IOException在默认临时文件目录中创建一个空文件，使用给定前缀和后缀生成其名称。调用此方法等同于调用 createTempFile(prefix, suffix, null)。
  + prefix - 用于生成文件名的前缀字符串；必须至少三个字符。
  + suffix - 用于生成文件名的后缀字符串；如果为 null，默认为 ".tmp"
* public static File createTempFile(String prefix,String suffix,File directory)throws IOException在指定目录中创建一个新的空文件，使用给定的前缀和后缀字符串生成其名称。
  + prefix - 用于生成文件名的前缀字符串；必须至少三个字符。
  + suffix - 用于生成文件名的后缀字符串；如果为 null，默认为 ".tmp"
  + directory - 将创建的文件所在的目录；如果使用默认临时文件目录，则该参数为 null

|  |
| --- |
| @Test  **public** **void** test6() **throws** IOException{  File tempFile = File.*createTempFile*("Hello", ".tmp");  System.***out***.println(tempFile.getAbsolutePath());  //C:\Users\Irene\AppData\Local\Temp\Hello2541030191749214481.tmp  } |

#### 2、删除文件

boolean delete() 当且仅当成功删除文件时，返回 true；否则返回 false。

public void deleteOnExit()：当退出JVM时，删除文件，一般用于删除临时文件

|  |
| --- |
| @Test  **public** **void** test7() {  File file = **new** File("d:/atguigu/javase/HelloIO.java");  file.delete();  } |
| @Test  **public** **void** test8() **throws** IOException{  File tempFile = File.*createTempFile*("Hello", ".tmp");  //....省略代码  tempFile.deleteOnExit();  } |

#### 3、文件重命名

boolean renameTo(File dest)：此方法行为的许多方面都是与平台有关的：重命名操作无法将一个文件从一个文件系统移动到另一个文件系统，该操作不是不可分的，如果已经存在具有目标抽象路径名的文件，那么该操作可能无法获得成功。应该始终检查返回值，以确保重命名操作成功。

|  |
| --- |
| @Test  **public** **void** test9(){  File src = **new** File("d:/atguigu/javase/HelloIO.java");  File dest = **new** File("d:/atguigu/javase/HelloFile.java");  src.renameTo(dest);  } |
| @Test  **public** **void** test10() {  File src = **new** File("d:/atguigu/javase/HelloIO.java");  File dest = **new** File("e:/HelloFile.java");  //D盘和E盘相同的文件系统可以成功，例如都是NTFS。  src.renameTo(dest);  } |

### 14.1.4 操作目录

#### 1、创建目录

boolean mkdir() ：必须确保父目录存在，否则创建失败

boolean mkdirs()：如果父目录链不存在，会一同创建父目录链

|  |
| --- |
| @Test  **public** **void** test1(){  File dir = **new** File("d:/atguigu/javase/io");  dir.mkdir();  //如果d:/atguigu/javase不存在不会报错，只是没创建成功  }  @Test  **public** **void** test2() {  File dir = **new** File("d:/atguigu/javase/io");  dir.mkdirs();  //如果d:/atguigu/javase不存在，会一并创建  } |

#### 2、列出目录内容

（1）public String[] list()列出当前目录的下级目录或文件的名称

（2）public File[] listFiles() 列出当前目录的下级的目录或文件对应的File对象

配合递归 可以列出下级的下级....

|  |
| --- |
| @Test  **public** **void** test3() {  File dir = **new** File("d:/atguigu");  listSubFiles(dir);  }  **public** **void** listSubFiles(File dir) {  **if** (dir != **null** && dir.isDirectory()) {  File[] listFiles = dir.listFiles();  **if** (listFiles != **null**) {  **for** (File sub : listFiles) {  listSubFiles(sub);  }  }  }  System.***out***.println(dir);  } |

代码示例：求目录总大小

|  |
| --- |
| @Test  **public** **void** test4() {  File dir = **new** File("D:/atguigu");  **long** length = getLength(dir);  System.***out***.println("大小：" + length);  }    **public** **long** getLength(File dir){  **if** (dir != **null** && dir.isDirectory()) {  File[] listFiles = dir.listFiles();  **if**(listFiles!=**null**){  **long** sum = 0;  **for** (File sub : listFiles) {  sum += getLength(sub);  }  **return** sum;  }  }**else** **if**(dir != **null** && dir.isFile()){  **return** dir.length();  }  **return** 0;  } |

1. public File[] listFiles(FileFilter filter)返回所有满足指定过滤器的文件和目录。如果给定 filter 为 null，则接受所有路径名。否则，当且仅当在路径名上调用过滤器的 FileFilter.accept(java.io.File) 方法返回 true 时，该路径名才满足过滤器。如果当前File对象不表示一个目录，或者发生 I/O 错误，则返回 null。
2. public File[] listFiles(FilenameFilter filter)返回所有满足指定过滤器的文件和目录。如果给定 filter 为 null，则接受所有路径名。否则，当且仅当在路径名上调用过滤器的 FilenameFilter.accept(java.io.File, java.lang.String) 方法返回 true 时，该路径名才满足过滤器。如果当前File对象不表示一个目录，或者发生 I/O 错误，则返回 null。
3. public static File[] listRoots()列出可用的文件系统根。

代码示例：列出“D:/atguigu/javaee/JavaSE20180808”该目录以及子目录下所有的.java文件

|  |
| --- |
| @Test  **public** **void** test5() {  File dir = **new** File("D:/atguigu/javaee/JavaSE20180808");  listByFileFilter(dir);  }    **public** **void** listByFileFilter(File file) {  **if** (file != **null** && file.isDirectory()) {  File[] listFiles = file.listFiles(**new** FilenameFilter() {  @Override  **public** **boolean** accept(File dir, String name) {  **return** name.endsWith(".java") || **new** File(dir,name).isDirectory();  }  });  **if** (listFiles != **null**) {  **for** (File sub : listFiles) {  **if**(sub.isFile()){  System.***out***.println(sub);  }  listByFileFilter(sub);  }  }  }  } |

#### 3、删除目录

boolean delete()：只能删除空目录。否则需要先将目录下的所有内容删除才能将该目录删除。

示例代码：删除"D:/atguigu/javase"目录，包括它的下一级

|  |
| --- |
| @Test  **public** **void** test6() {  File dir = **new** File("D:/atguigu/javase");  forceDeleteDir(dir);  }  **public** **void** forceDeleteDir(File dir) {  **if** (dir != **null** && dir.isDirectory()) {  File[] listFiles = dir.listFiles();  **if**(listFiles!=**null**){  **for** (File sub : listFiles) {  forceDeleteDir(sub);  }  }  }  dir.delete();  } |

#### 4、重命名目录

boolean renameTo(File dest)：如果是windows目录，只能在同一个盘下，不能从D盘移动到E盘。

示例代码：

|  |
| --- |
| @Test  **public** **void** test7() {  File dir = **new** File("D:/atguigu/javase");  File dest = **new** File("D:/atguigu/java代码");  dir.renameTo(dest);  } |
| @Test  **public** **void** test8() {  File dir = **new** File("D:/atguigu/javase");  File dest = **new** File("D:/temp");  dir.renameTo(dest);  } |

## 14.2 IO流的分类与设计

### 14.2.1 java.io包中的IO流

java.io包中类太多了，简直是......“琳琅满目”“眼花缭乱”。你第一次（第二次，第三次，。。。）看到这些API时，“哦”愁眉苦脸时，放心，你不是唯一受到惊吓和困扰的人。但是，如果你好好梳理他们的关系，并掌握几个经典代表的使用方式，其他的也就等于掌握了。

**第一：**其实不管有多少种IO流，最基本的只有四个。

IO的四个超级父类，抽象基类。

* 字节输入流：InputStream
* 字节输出流：OutputStream
* 字符输入流：Reader
* 字符输出流：Writer

**第二：**IO流类的设计选用了“装饰者”设计模式，即IO流分为两大类，“被装饰”的组件和“装饰”的组件。

例如：以InputStream为例

其中FileInputStream、ByteArrayInputStream等是“被装饰”的组件，依次用来连接和读取“文件”、“内存中的字节数组”的等。

而BufferedInputStream、DataInputStream、ObjectInputStream等是用来“装饰”的组件，依次是给其他InputStream的IO流提供装饰的辅助功能的，依次可以增加“提高效率的缓冲功能”、“按照Java数据类型读取数据的能力”、“读取并恢复Java对象的能力”等

你会发现OutputStream、Reader、Writer系列的流设计方式也是一样的。

**第三：**IO流的选取可以通过以下几个分类来简化选取过程。

按照IO流的方向：输入流和输出流

* I：代表Input
* O：代表Output

Java的IO流是单向的，只能从输入流（Input、Reader）中读取（read）数据，也只能往输出流（Output、Writer）中写（write、print）出数据。

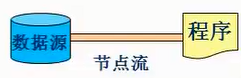
按照IO流的处理数据的基本单位分：

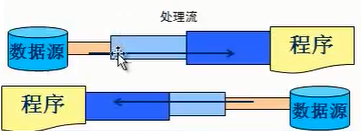
* 字节流（XxxStream）：直接处理二进制，一个字节一个字节处理，它适用于一切数据，包括纯文本、doc、xls、图片、音频、视频等等
* 字符流（XxxReader和XxxWriter）：一个字符一个字符处理，只能纯文本类的数据。

按照角色分：节点流、处理流

* 节点流：连接源头、目的地，即**被**装饰者IO流
* 处理流：增强功能，提高性能，即装饰者IO流

节点流处于IO操作的第一线，所有操作必须通过他们进行；处理流是通过包装节点流来完成功能的，处理流可以增加很多层。处理流必须依赖和包装节点流，而不能单独存在。





|  |
| --- |
| 提示：装饰模式（Decorator Pattern）也称为包装模式（Wrapper Pattern），其使用一种对客户端透明的方式来动态地扩展对象的功能，它是通过继承扩展功能的替代方案之一。在现实生活中你也有很多装饰者的例子，例如：人需要各种各样的衣着，不管你穿着怎样，但是，对于你个人本质来说是不变的，充其量只是在外面加上了一些装饰，有，“遮羞的”、“保暖的”、“好看的”、“防雨的”.... |

### 14.2.2 常见IO流

|  |  |  |
| --- | --- | --- |
| 文件IO流 | 从文件读取 | FileInputStream |
| FileReader |
| 写入到文件 | FileOutputStream |
| FileWriter |
| 缓冲IO流 | 从其他输入流中读取 | BufferedInputStream |
| BufferedReader |
| 写入到其他输出流中 | BufferedOutputStream |
| BufferedWriter |
| 转换流 | 从其他输入流读取，并解码 | InputStreamReader |
| 写入到其他输出流中，并编码 | OutputStreamWriter |
| 数据流 | 以与机器无关方式从底层输入流中读取基本 Java 数据类型 | DataInputStream |
| 以适当方式将基本 Java 数据类型写入输出流中 | DataOutputStream |
| 对象流 | 序列化 | ObjectOutputStream |
| 反序列化 | ObjectInputStream |
| 打印流 |  | PrintStream |
|  | PrintWriter |
| 其他 | ...... |  |

### 14.2.3 抽象基类的常用方法

#### 1、InputStream

（1）int read()

从输入流中读取数据的下一个字节。返回 0 到 255 范围内的 int 字节值。如果因为已经到达流末尾而没有可用的字节，则返回值 -1。

（2）int read(byte[] b)

从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中。如果因为已经到达流末尾而没有可用的字节，则返回值 -1。否则以整数形式返回实际读取的字节数。

（3）int read(byte[] b, int off,int len)

将输入流中最多 len 个数据字节读入 byte 数组。尝试读取 len 个字节，但读取的字节也可能小于该值。以整数形式返回实际读取的字节数。如果因为流位于文件末尾而没有可用的字节，则返回值 -1.

（4）public void close() throws IOException关闭此输入流并释放与该流关联的所有系统资源。

#### 2、OutputStream

（1）void write(int b)

将指定的字节写入此输出流。write 的常规协定是：向输出流写入一个字节。要写入的字节是参数 b 的八个低位。b 的 24 个高位将被忽略。 即写入0~255范围的

（2）void write(byte[] b)

将 b.length 个字节从指定的 byte 数组写入此输出流。write(b) 的常规协定是：应该与调用 write(b, 0, b.length) 的效果完全相同。

（3）void write(byte[] b,int off,int len)

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流。

1. public void flush()throws IOException 刷新此输出流并强制写出所有缓冲的输出字节，调用此方法指示应将这些字节立即写入它们预期的目标。
2. public void close() throws IOException关闭此输出流并释放与该流关联的所有系统资源。

#### 3、Reader

（1）int read()

读取单个字符。作为整数读取的字符，范围在 0 到 65535 之间 (0x00-0xffff)（2个字节的Unicode码），如果已到达流的末尾，则返回 -1

（2）int read(char[] cbuf)

将字符读入数组。如果已到达流的末尾，则返回 -1。否则返回本次读取的字符数。

（3）int read(char[] cbuf,int off,int len)

将字符读入数组的某一部分。存到数组cbuf中，从off处开始存储，最多读len个字符。如果已到达流的末尾，则返回 -1。否则返回本次读取的字符数。

（4）public void close() throws IOException关闭此输入流并释放与该流关联的所有系统资源。

#### 4、Writer

（1）void write(int c)、Writer append(char c)

写入单个字符。要写入的字符包含在给定整数值的 16 个低位中，16 高位被忽略。 即写入0 到 65535 之间的Unicode码。

（2）void write(char[] cbuf)、Writer append(CharSequence csq)

写入字符数组。

（3）void write(char[] cbuf,int off,int len)、Writer append(CharSequence csq, int start, int end)

写入字符数组的某一部分。从off开始，写入len个字符

（4）void write(String str)

写入字符串。

（5）void write(String str,int off,int len)

写入字符串的某一部分。

（6）void flush()

刷新该流的缓冲，则立即将它们写入预期目标。

（7）public void close() throws IOException关闭此输出流并释放与该流关联的所有系统资源。

### 14.2.4 IO流使用步骤

#### 1、读取/接收数据的步骤

第一步：选择IO流

1、选择节点流

明确你要从哪里读取数据，即数据源是什么？

如果从文件中，就选择FileInputStream、FileReader；

如果从内存的数组中，就选择ByteArrayInputStream、CharArrayReader；

如果从网络中，就选择InputStream；

2、选择字节流还是字符流

如果可选，那么看数据是否是纯文本，如果是纯文本并且文本的编码方式与当前平台的编码方式一致，那么选择字符流的效率更高，否则选择字节流。

3、是否需要增加额外的辅助功能

缓冲功能：BufferedReader、BufferedInputStream

是否需要按行读：BufferedReader、Scanner

是否需要以与机器无关方式从底层输入流中读取基本 Java 数据类型：DataInputStream

是否需要对象的反序列化：ObjectInputStream

是否需要对流中的数据进行解码的转换：InputStreamReader

第二步：循环读取数据

* read()
* readLine()
* readXxx()，例如：readInt()，readObject()，readUTF()等

第三步：关闭close()

#### 2、写/发送数据的步骤

第一步：选择IO流

1、选择节点流

明确你要把数据写到哪里去，即数据的目的地是哪里？

如果写到文件中，就选择FileOutputStream、FileWriter；

如果写到内存某数组中，就选择ByteArrayOutputStream、CharArrayWriter；

如果发送到网络中，就选择OutputStream；

2、选择字节流还是字符流

如果可选，那么看数据是否是纯文本，如果是纯文本并且文本的编码方式与当前平台的编码方式一致，那么选择字符流的效率更高，否则选择字节流。

3、是否需要增加额外的辅助功能

缓冲功能：BufferedOutputStream、BufferedWriter

是否需要按行写：PrintStream、PrintWriter

是否需要以以适当方式将基本 Java 数据类型写入输出流：DataOutputStream

是否需要将对象的序列化：ObjectOutputStream

是否需要对流中的数据进行编码的转换：OutputStreamWriter

第二步：循环写数据

* write(xx)
* print(xx)和println(xx)
* writeXxx：例如：writeInt(xx)、writeDouble(xx)、writeObject(xx)等

第三步：关闭IO流close()

## 14.3 案例分析

### 14.3.1 输出纯文本数据

案例：把用户的留言保存到message.txt文件中，从键盘输入留言

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.Scanner;  **public** **class** TestFileWriter {  **public** **static** **void** main(String[] args) {  Scanner input = **new** Scanner(System.***in***);  FileWriter fw = **null**;    **try** {  fw = **new** FileWriter("message.txt",true);//true表示追加模式，默认是覆盖模式    **while**(**true**){  System.***out***.println("请输入留言：");  String message = input.nextLine();    **if**("stop".equals(message)){  **break**;  }  fw.write(message);  System.***out***.println("结束留言，请输入stop");  }  } **catch** (IOException e) {  e.printStackTrace();  } **finally**{  input.close();  **try** {  fw.close();  } **catch** (IOException e) {  }  }  }  } |

### 14.3.2 读取纯文本数据

案例：读取用户留言信息message.txt文件中的内容

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.FileReader;  **import** java.io.IOException;  **public** **class** TestFileReader {  **public** **static** **void** main(String[] args) {  FileReader fr = **null**;  **try** {  fr = **new** FileReader("message.txt");    **char**[] data = **new** **char**[1024];  **int** len;  **while**((len = fr.read(data)) != -1){  System.***out***.println(**new** String(data,0,len));  }    } **catch** (IOException e) {  e.printStackTrace();  }**finally**{  **try** {  fr.close();  } **catch** (IOException e) {  }  }  }  } |

### 14.3.3 按行读取

案例：读取用户留言信息message.txt文件中的内容

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.BufferedReader;  **import** java.io.FileReader;  **import** java.io.IOException;  **public** **class** TestBufferedReader {  **public** **static** **void** main(String[] args) {  BufferedReader br = **null**;  **try** {  br = **new** BufferedReader(**new** FileReader("message.txt"));    String str;  **while**((str = br.readLine())!=**null**){  System.***out***.println(str);  }    } **catch** (IOException e) {  e.printStackTrace();  }**finally**{  **try** {  br.close();  } **catch** (IOException e) {  }  }  }  } |

### 14.3.4 复制文件基本版

文件IO流，和文件的数据传输：

（1）文件字节输入流：FileInputStream

（2）文件字节输出流：FileOutputStream

（3）文件字符输入流：FileReader，只能按照平台默认的编码方式读取数据

（4）文件字符输出流：FileWriter，只能按照平台默认的编码方式写入数据

注意：

在写入一个文件时，如果目录下有同名文件将被覆盖，除非是追加模式。

在读取文件时，必须保证该文件已存在，否则出异常。

案例：将文件”atguigu/file/1.jpg”复制”atguigu/img/1.jpg”

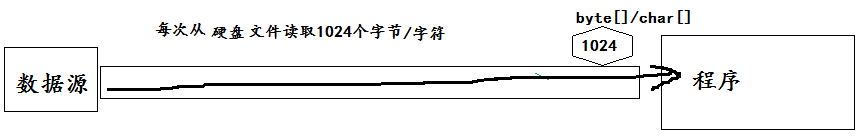
|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **public** **class** TestCopy {    **public** **static** **void** main(String[] args) {  **try** {  *copyFile*(**new** File("atguigu/file/1.jpg"), **new** File("atguigu/img/1.jpg"));  System.***out***.println("文件复制成功");  } **catch** (IOException e) {  System.***out***.println("文件复制失败");  }  }    **public** **static** **void** copyFile(File src, File dest) **throws** IOException{  **if**(!src.isFile()){  **throw** **new** RuntimeException(src.getPath() + "不存在");  }  FileInputStream fis = **new** FileInputStream(src);  FileOutputStream fos = **new** FileOutputStream(dest);    **byte**[] data = **new** **byte**[1024];  **int** len;  **while**((len = fis.read(data))!=-1){  fos.write(data, 0, len);  }    fos.close();  fis.close();  }  } |

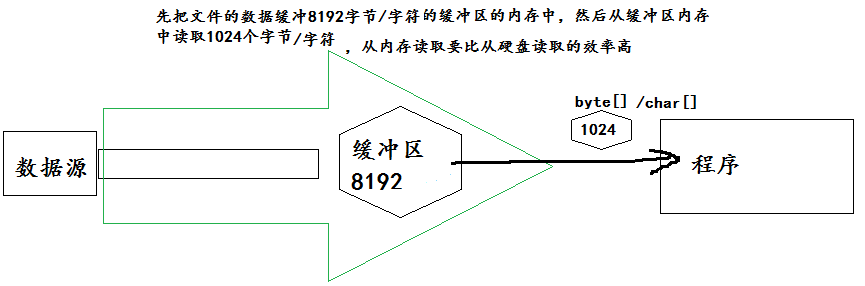
### 14.3.5 复制文件提升效率版

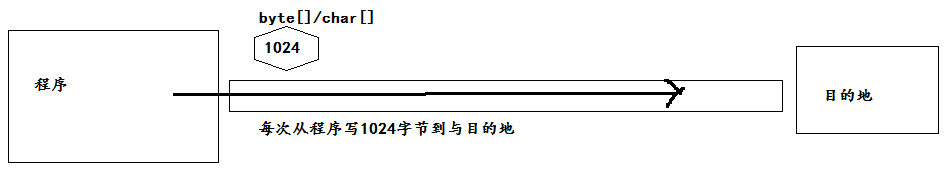
案例：将文件”atguigu/download/经典视频.zip”复制到”atguigu/share/值得看.zip”

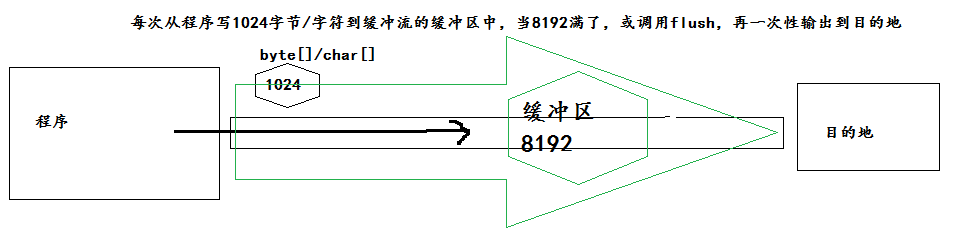
使用缓冲流，可以提高效率。缓冲流的默认缓冲区大小是8192字节/字符。

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.BufferedInputStream;  **import** java.io.BufferedOutputStream;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **public** **class** TestBufferedCopy{    **public** **static** **void** main(String[] args) {  **try** {  *copyFileInHighSpeed*(**new** File("atguigu/download/经典视频.zip"), **new** File("atguigu/share/值得看.zip"));  System.***out***.println("文件复制成功");  } **catch** (IOException e) {  System.***out***.println("文件复制失败");  }  }    **public** **static** **void** copyFileInHighSpeed(File src, File dest) **throws** IOException{  **if**(!src.isFile()){  **throw** **new** RuntimeException(src.getPath() + "不存在");  }  BufferedInputStream bis = **new** BufferedInputStream(**new** FileInputStream(src));  BufferedOutputStream bos = **new** BufferedOutputStream(**new** FileOutputStream(dest));    **byte**[] data = **new** **byte**[1024];  **int** len;  **while**((len = bis.read(data))!=-1){  bos.write(data, 0, len);  }    bis.close();  bos.close();  }  } |









### 14.3.6 读写纯文本数据的同时进行编码和解码

当要读取指定编码的纯文本数据时，可以使用InputStreamReader进行解码。所谓解码，即把二进制的字节序列按照指定字符编码解码为可以被正确识别的字符内容。

案例：当前系统平台的字符编码方式是：GBK，需要读取一个UTF-8的文件内容，到当前系统中。

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.BufferedReader;  **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **public** **class** TestDecode {  **public** **static** **void** main(String[] args) {  BufferedReader br = **null**;  **try** {  br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream("test\_utf8.txt")));    String str;  **while**((str = br.readLine())!=**null**){  System.***out***.println(str);  }  } **catch** (IOException e) {  e.printStackTrace();  } **finally**{  **try** {  br.close();  } **catch** (IOException e) {  }  }  }  } |

同理，要将纯文本数据，按照指定编码转成字节序列时，可以选择OutputStreamWriter。

案例：编写程序实现将一个编码为“GBK”的纯文本文件test.txt，复制为编码是“UTF-8”的文件other.txt。

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.OutputStreamWriter;  **public** **class** TestEncode {  **public** **static** **void** main(String[] args) **throws** IOException {  *copy*(**new** File("test.txt"),"GBK",**new** File("other.txt"),"UTF-8");  }    **public** **static** **void** copy(File src, String srcCharset, File dest, String destCharset) **throws** IOException{  **if**(!src.isFile()){  **throw** **new** RuntimeException(src.getPath() + "不存在");  }  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(src),srcCharset));  BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(dest),destCharset));    String str;  **while**((str = br.readLine())!=**null**){  bw.write(str);  bw.newLine();  }    bw.close();  br.close();  }  } |

### 14.3.7 操作Java各种数据类型的数据

如果要存储如下一组数据，到game.dat文件中，并在后面可以重写读取。

|  |
| --- |
| String name = “巫师”;  int age = 300;  char gender = ‘男’;  int energy = 5000;  double price = 75.5;  boolean relive = true; |

完成这个需求，可以使用DataOutputStream进行写，随后用DataInputStream进行读取，而且顺序要一致。

示例代码：

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.DataInputStream;  **import** java.io.DataOutputStream;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **public** **class** TestData {  **public** **void** save() **throws** IOException{  String name = "巫师";  **int** age = 300;  **char** gender = '男';  **int** energy = 5000;  **double** price = 75.5;  **boolean** relive = **true**;    DataOutputStream dos = **new** DataOutputStream(**new** FileOutputStream("game.dat"));  dos.writeUTF(name);  dos.writeInt(age);  dos.writeChar(gender);  dos.writeInt(energy);  dos.writeDouble(price);  dos.writeBoolean(relive);  dos.close();  }  **public** **void** reload()**throws** IOException{  DataInputStream dis = **new** DataInputStream(**new** FileInputStream("game.dat"));  String name = dis.readUTF();  **int** age = dis.readInt();  **char** gender = dis.readChar();  **int** energy = dis.readInt();  **double** price = dis.readDouble();  **boolean** relive = dis.readBoolean();    System.***out***.println(name+"," + age + "," + gender + "," + energy + "," + price + "," + relive);    dis.close();  }  } |

### 14.3.8 保存对象

如果你正在编写游戏，就得有存储和恢复游戏的功能。如果程序要存储状态，你可以选择上面的方式（DataOutputStream），对每一个对象逐个地把每一项变量的值写到特定格式的文件中。但是其实，Java还提供了面向对象的处理方式——直接存储对象。不过，这种方式保存的数据，必须是由Java程序来读取。如果在程序所储存的文件数据需要给某些非Java应用程序所读取时，就不能选用这种方式了。

#### 1、序列化与反序列化

想要输出对象，必须借助ObjectOutputStream，它有一个writeObject(obj)方法可以输出对象。

如果想要反序列化，必须借助ObjectInputStream，它有一个readObject()方法可以读取对象。

注意：

不是所有对象都可以序列化，必须实现java.io.Serializable接口的类或其子类的对象，而且如果该对象的属性也是引用数据类型，并且该属性也要序列化，那么该属性的类型或其父类也要实现java.io.Serializable接口，否则会报java.io.NotSerializableException异常。

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.ObjectOutputStream;  **import** com.atguigu.part2.io.bean.Account;  **public** **class** TestObjectOutputStream {  **public** **static** **void** main(String[] args) {  Account.*setInterestRate*(0.0024);  Account account = **new** Account("111000111", "尚硅谷", "123456", 1000.0, 2.4);    ObjectOutputStream oos = **null**;  **try** {  oos = **new** ObjectOutputStream(**new** FileOutputStream("account.dat"));  oos.writeObject(account);  } **catch** (IOException e) {  e.printStackTrace();  } **finally**{  **try** {  oos.close();  } **catch** (IOException e) {  }  }  }  } |
| **package** com.atguigu.part2.io.bean;  **import** java.io.Serializable;  **public** **class** Account **implements** Serializable{  **private** **static** **double** *interestRate*;  **private** String number;  **private** String name;  **private** String password;  **private** **double** balance;  **private** **double** interest;  **public** Account(String number, String name, String password, **double** balance, **double** interest) {  **super**();  **this**.number = number;  **this**.name = name;  **this**.password = password;  **this**.balance = balance;  **this**.interest = interest;  }  **public** **static** **double** getInterestRate() {  **return** *interestRate*;  }  **public** **static** **void** setInterestRate(**double** interestRate) {  Account.*interestRate* = interestRate;  }  **public** String getNumber() {  **return** number;  }  **public** **void** setNumber(String number) {  **this**.number = number;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  **public** **double** getBalance() {  **return** balance;  }  **public** **void** setBalance(**double** balance) {  **this**.balance = balance;  }  **public** **double** getInterest() {  **return** interest;  }  **public** **void** setInterest(**double** interest) {  **this**.interest = interest;  }  @Override  **public** String toString() {  **return** "Account [number=" + number + ", name=" + name + ", password=" + password + ", balance=" + balance  + ", interest=" + interest + "]";  }  } |

序列化的文件是很难让人阅读的，但它比纯文本文件、或一项一项数据保存的数据更容易让程序恢复对象的状态，也比较安全，因为一般人不会知道如何动手修改数据。

#### 2、哪些属性不序列化

首先static的属性不会被序列化，反序列化时恢复为当前类变量的值。因为static代表“一个类一个”，而不是“一个对象一个”，因此它不是对象的状态。

如果某个实例变量不能或不应该被序列化，就把它标记为transient（瞬时）的。为什么有些实例变量不能被实例化？如果某个属性的类型没有实现Serializable接口，而你又不能修改该类，那么只能把该属性标记为transient。或者动态数据只可以在执行时求出而不能或不必存储。

被标记为transient的属性，在反序列化时，被恢复为默认值。

例如：

|  |
| --- |
| **package** com.atguigu.part2.io.bean;  **import** java.io.Serializable;  **public** **class** Account **implements** Serializable{  **private** **static** **double** *interestRate*;  **private** String number;  **private** String name;  **private** String password;  **private** **double** balance;  **transient** **private** **double** interest;  //......此处省略构造器、get/set等其他成员方法  } |

#### 3、序列化版本ID

如果你将对象序列化，则必须要有该类才能还原和使用该对象。但若你同时又修改了类会发生什么事？

有些修改会严重违反Java的类型安全性和兼容性，例如：删除了某个实例变量、改变实例变量的类型、将类从可序列化修改为不可序列化，将实例变量修改静态的等等。

但是有些修改通常不会影响，例如：加入新的实例变量（还原时可以按默认值处理），将实例变量从瞬时修改为非瞬时的（可以使用默认值）等。

但是，现在所有对类的修改都导致原来的数据在反序列化时失败java.io.InvalidClassException。

解决这个问题的方法，就是在实现java.io.Serializable接口时，增加一个long类型的静态常量serialVersionUID。如果类没有显示定义这个静态变量，它的值是Java运行时环境根据类的内部细节自动生成的，若类的源代码作了修改，serialVersionUID 就会发生变化，从而导致“旧”数据反序列化失败。

如果对类做了会影响数据兼容性的操作时，要么修改serialVersionUID的值，使得原来的反序列化数据失败，要么你要对“旧”对象反序列化后引起的问题负责。

|  |
| --- |
| **package** com.atguigu.part2.io.bean;  **import** java.io.Serializable;  **public** **class** Account **implements** Serializable{  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** **static** **double** *interestRate*;  **private** String number;  **private** String name;  **private** String password;  **private** **double** balance;  **transient** **private** **double** interest;  //......此处省略构造器、get/set等其他成员方法  } |

### 14.3.9 按行输出文本内容

PrintStream和PrintWriter是两个打印流，可以实现将Java基本数据类型的数据格式转化为字符串输出，引用类型的数据自动调用toString()。这两个类提供了一系列重载的print()和println()方法，用于多种数据类型的输出，用于不会抛出 I/O 异常。

PrintStream和PrintWriter都可以指定编码方式。

与 PrintStream 类不同，如果启用了自动刷新，则只有在调用 println、printf 或 format 的其中一个方法时才可能完成此操作，而不是每当正好输出换行符时才完成。这些方法使用平台自有的行分隔符概念，而不是换行符。

案例：

从键盘输入消息，按行写入文件

|  |
| --- |
| **package** com.atguigu.part2.io;  **import** java.io.IOException;  **import** java.io.PrintWriter;  **import** java.util.Scanner;  **public** **class** TestPrintWriter {  **public** **static** **void** main(String[] args) {  Scanner input = **new** Scanner(System.***in***);  PrintWriter pw = **null**;    **try** {  pw = **new** PrintWriter("message.txt");    **while**(**true**){  System.***out***.println("请输入留言：");  String message = input.nextLine();    **if**("stop".equals(message)){  **break**;  }  pw.println(message);  System.***out***.println("结束留言，请输入stop");  }  } **catch** (IOException e) {  e.printStackTrace();  } **finally**{  input.close();  pw.close();  }  }  } |

### 14.3.10 拷贝或剪切整个目录

案例：

将”atguigu/download”文件夹，复制到”atguigu/temp”目录中去

|  |
| --- |
| **package** com.atguigu.part3.io;  **import** java.io.File;  **import** java.io.IOException;  **public** **class** TestFileUtils {  **public** **static** **void** main(String[] args) **throws** IOException {  FileUtils.*copyDirectory*(**new** File("atguigu/download"), **new** File("atguigu/temp"));  }  } |
| **package** com.atguigu.part3.io;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **public** **class** FileUtils {  //复制文件  **public** **static** **void** copyFile(File src, File dest) **throws** IOException{  **if**(!src.isFile()){  **throw** **new** RuntimeException(src.getPath() + "不存在");  }  FileInputStream fis = **new** FileInputStream(src);  FileOutputStream fos = **new** FileOutputStream(dest);    **byte**[] data = **new** **byte**[1024];  **int** len;  **while**((len = fis.read(data))!=-1){  fos.write(data, 0, len);  }    fos.close();  fis.close();  }  //复制文件夹  **public** **static** **void** copyDirectory(File srcDir,File destDir)**throws** IOException{  //父目录不能拷贝到子目录中  **if**(destDir.getAbsolutePath().contains(srcDir.getAbsolutePath())){  **throw** **new** IOException("父目录不能拷贝到子目录中");  }  **if**(!destDir.isDirectory()){  **throw** **new** IOException("目标对象不是目录");  }  File dest = **new** File(destDir,srcDir.getName());  //如果srcDir是个文件，那么直接复制文件到destDir中  **if**(srcDir.isFile()){  *copyFile*(srcDir, dest);  }**else**{  dest.mkdirs();//先在destPath目录中创建srcDir对应的文件夹  File[] list = srcDir.listFiles();//准备复制srcDir的子目录或子文件  **for**(File f: list){  *copyDirectory*(f,dest);  }  }  }  } |

思考：那么剪切呢？

|  |
| --- |
| **package** com.atguigu.part3.io;  **import** java.io.File;  **import** java.io.IOException;  **public** **class** TestFileUtils {  **public** **static** **void** main(String[] args) **throws** IOException {  FileUtils.cutDirectory(**new** File("atguigu/download"), **new** File("atguigu/temp"));  }  } |
| //剪切文件夹  **public** **static** **void** cutDirectory(File srcDir,File destDir)**throws** IOException{  **if**(destDir.getAbsolutePath().contains(srcDir.getAbsolutePath())){  **throw** **new** IOException("父目录不能剪切到子目录中");  }  **if**(!destDir.isDirectory()){  **throw** **new** IOException("目标对象不是目录");  }  File dest = **new** File(destDir,srcDir.getName());  **if**(srcDir.isFile()){  *copyFile*(srcDir, dest);  }**else**{  dest.mkdirs();  File[] list = srcDir.listFiles();  **for**(File f: list){  *cutDirectory*(f,dest);  }  }  srcDir.delete();  } |

## 14.4 System中的IO流

System.in和System.out分别代表了系统标准的输入和输出设备

* 默认输入设备是键盘，输出设备是显示器
* System.in的类型是InputStream
* System.out的类型是PrintStream
* System.err的类型是PrintStream，按照惯例，此输出流用于显示错误消息，或者显示那些即使用户输出流（变量 out 的值）已经重定向到通常不被连续监视的某一文件或其他目标，也应该立刻引起用户注意的其他信息。

重定向：

* public static void setIn(InputStream in)
* public static void setOut(PrintStream out)
* public static void setErr(PrintStream err)

控制台是个文件，查看System类中的initializeSystemClass()方法

* FileInputStream fdIn = new FileInputStream(FileDescriptor.in);
* FileOutputStream fdOut = new FileOutputStream(FileDescriptor.out);
* FileOutputStream fdErr = new FileOutputStream(FileDescriptor.err);

### 14.3.10.1 重定向System.in和System.out

|  |
| --- |
| **package** com.atguigu.part4.io;  **import** java.io.FileDescriptor;  **import** java.io.FileInputStream;  **import** java.util.Scanner;  **public** **class** TestSystemIn {  **public** **static** **void** main(String[] args) **throws** Exception{  //重定向从文件输入  System.*setIn*(**new** FileInputStream("myjava\\info.txt"));  Scanner input = **new** Scanner(System.***in***);  **while**(input.hasNext()){  String str = input.nextLine();  System.***out***.println(str);  }  input.close();  //重定向回键盘输入  System.*setIn*(**new** FileInputStream(FileDescriptor.***in***));  }  } |
| **package** com.atguigu.part4.io;  **import** java.io.FileDescriptor;  **import** java.io.FileOutputStream;  **import** java.io.PrintStream;  **public** **class** TestSystemOut {  **public** **static** **void** main(String[] args) **throws** Exception{  System.***out***.println("hello");  //重定向输出到文件  System.*setOut*(**new** PrintStream("myjava\\print.txt"));  System.***out***.println("world");  //重定向回控制台  System.*setOut*(**new** PrintStream(**new** FileOutputStream(FileDescriptor.***out***)));  System.***out***.println("java");  }  } |

### 14.3.10.2 为Scanner指定输入流

java.util.Scanner是一个可以使用正则表达式来解析基本类型和字符串的简单文本扫描器。

示例代码：从键盘输入数据

|  |
| --- |
| **package** com.atguigu.part4.io;  **import** java.util.Scanner;  **public** **class** TestScannerSystemIn {  **public** **static** **void** main(String[] args) {  Scanner input = **new** Scanner(System.***in***);    System.***out***.print("姓名：");  String name = input.nextLine();    System.***out***.print("性别：");  **char** gender = input.next().charAt(0);    System.***out***.print("年龄：");  **int** age = input.nextInt();    System.***out***.print("电话：");  String phone = input.next();    System.***out***.print("邮箱：");  String email = input.next();    System.***out***.println("姓名：" + name);  System.***out***.println("性别：" + gender);  System.***out***.println("年龄：" + age);  System.***out***.println("电话：" + phone);  System.***out***.println("邮箱：" + email);    input.close();  }  } |

示例代码：从文件扫描数据

|  |
| --- |
| **package** com.atguigu.part4.io;  **import** java.io.FileInputStream;  **import** java.util.Scanner;  **public** **class** TestScanner {  **public** **static** **void** main(String[] args) **throws** Exception{  Scanner input = **new** Scanner(**new** FileInputStream("myjava\\info.txt"));  **while**(input.hasNext()){  String str = input.nextLine();  System.***out***.println(str);  }  input.close();  }  } |

## 14.5 JDK1.7新特性：try...with...resources

JDK1.7新特性，try...with...resources

格式：

|  |
| --- |
| try(声明需要关闭的资源对象){  逻辑代码  }catch(异常类型 e){  异常处理代码  } |

不管{}部分是否发生异常，都会关闭资源对象

|  |
| --- |
| **package** com.atguigu.part5;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **public** **class** TestTryWithResource {  **public** **static** **void** copy(File src, File dest) **throws** IOException{  **try**(  InputStream input = **new** FileInputStream(src);  OutputStream output = **new** FileOutputStream(dest);  ){  **byte**[] data = **new** **byte**[1024];  **int** len;  **while** ((len = input.read(data)) != -1) {  output.write(data, 0, len);  }  }  }    **public** **static** **void** copy2(File src, File dest) {  **try**(  InputStream input = **new** FileInputStream(src);  OutputStream output = **new** FileOutputStream(dest);  ){  **byte**[] data = **new** **byte**[1024];  **int** len;  **while** ((len = input.read(data)) != -1) {  output.write(data, 0, len);  }  }**catch**(FileNotFoundException e){  e.printStackTrace();  }**catch**(IOException e){  e.printStackTrace();  }  }  } |

## 14.6 NIO

早期的java只提供了一个File类来访问文件系统，但File类的功能比较有限，所提供的方法性能也不高。而且，大多数方法在出错时仅返回失败，并不会提供异常信息。

NIO. 2为了弥补这种不足，引入了Path接口，代表一个平台无关的平台路径，描述了目录结构中文件的位置。Path可以看成是File类的升级版本，实际引用的资源也可以不存在。

在以前IO操作都是这样写的:

import java.io.File;

File file = new File("index.html");

但在Java7 中，我们可以这样写：

import java.nio.file.Path;

import java.nio.file.Paths;

Path path = Paths.get("index.html");

同时，NIO.2还提供了Files、Paths工具类，Files包含了大量静态的工具方法来操作文件；Paths则包含了两个返回Path的静态工厂方法。

static Path get(String first, String … more) : 用于将多个字符串串连成路径

static Path get(URI uri): 返回指定uri对应的Path路径

|  |
| --- |
| Path path = Paths.get("io", "java", "1.txt");  System.out.println(path); |

### 14.6.1 Path接口

* String toString() ： 返回调用 Path 对象的字符串表示形式
* boolean startsWith(String path) : 判断是否以 path 路径开始
* boolean endsWith(String path) : 判断是否以 path 路径结束
* boolean isAbsolute() : 判断是否是绝对路径
* Path getParent() ：返回Path对象包含整个路径，不包含 Path 对象指定的文件路径
* Path getRoot() ：返回调用 Path 对象的根路径
* Path getFileName() : 返回与调用 Path 对象关联的文件名
* Path getName(int idx) : 返回指定索引位置 idx 的路径名称
* int getNameCount() : 返回Path 根目录后面元素的数量
* Path toAbsolutePath() : 作为绝对路径返回调用 Path 对象
* Path resolve(Path p) :合并两个路径，返回合并后的路径对应的Path对象
* File toFile(): 将Path转化为File类的对象

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // Path path = Paths.get("1.txt");  // Path path = Paths.get("io", "java");  // Path path = Paths.get("d:","atguigu");  Path path = Paths.*get*("io", "java", "1.txt");  System.***out***.println("toString:"+path);    System.***out***.println("startsWith:"+path.startsWith("io"));  System.***out***.println("endsWith:"+path.endsWith("1.txt"));    System.***out***.println("parent:"+path.getParent());//io/java  System.***out***.println("root:"+path.getRoot());//null 只有绝对路径才有根  System.***out***.println("fileName："+path.getFileName());  System.***out***.println("FileSystem.separator:"+path.getFileSystem().getSeparator());    System.***out***.println("nameCount:"+path.getNameCount());  **for** (**int** i = 0; i < path.getNameCount(); i++) {  System.***out***.println("name("+(i+1)+"):"+path.getName(i));  }    System.***out***.println("isAbsolutePath:"+path.isAbsolute());  System.***out***.println("toAbsolutePath:"+path.toAbsolutePath());  } |

### 14.6.2 Files工具类

用于操作文件或目录的工具类：

* + Path createDirectory(Path path, FileAttribute<?> … attr) : 创建一个目录
  + Path createDirectories(Path dir, FileAttribute<?>... attrs) 创建一个目录，通过创建所有不存在的父目录。
  + Path createFile(Path path, FileAttribute<?> … arr) : 创建一个文件
  + Path createLink(Path link, Path existing) 创建一个新的链接（目录项）为现有的文件（可选操作）。
  + void delete(Path path) : 删除一个文件，如果不存在，执行报错
  + void deleteIfExists(Path path) : Path对应的文件如果存在，执行删除，如果不存在，就什么也不干
  + Path copy(Path src, Path dest, CopyOption … how) : 文件的复制
  + Path move(Path src, Path dest, CopyOption…how) : 将 src 移动到 dest 位置
  + long size(Path path) : 返回 path 指定文件的大小

|  |
| --- |
| public static void main(String[] args) throws IOException {  **import** java.io.IOException;  **import** java.nio.file.Files;  **import** java.nio.file.Path;  **import** java.nio.file.Paths;  **import** java.nio.file.StandardCopyOption;  **import** org.junit.Test;  **public** **class** TestFiles {  @Test  **public** **void** test1()**throws** IOException{  Path dir = Paths.*get*("first", "second","io");  // Files类的：Path createDirectory(Path path, FileAttribute<?> … attr) : 创建一个目录  // Files类的：Path createDirectories(Path dir, FileAttribute<?>... attrs)  // Files.createDirectory(dir);//类似于mkdir()父目录如果不存在，会报错  Files.*createDirectories*(dir);//类似于 mkdirs()  }    @Test  **public** **void** test2()**throws** IOException{  Path file = Paths.*get*("first", "second","io", "1.txt");  // Path createFile(Path path, FileAttribute<?> … arr) : 创建一个文件，如果父目录不存在会报错，如果文件已存在也会报错  Files.*createFile*(file);  }      @Test  **public** **void** test3()**throws** IOException{  Path dir = Paths.*get*("first", "second","io");  Path file = Paths.*get*("first", "second","io", "1.txt");  // void delete(Path path) : 删除一个文件，如果不存在，执行报错，也可以删除空目录，如果目录非空，会报错  // void deleteIfExists(Path path) : Path对应的文件如果存在，执行删除，如果不存在，就什么也不干  Files.*deleteIfExists*(dir);  Files.*deleteIfExists*(file);  }    @Test  **public** **void** test4()**throws** IOException{  // Path copy(Path src, Path dest, CopyOption … how) : 文件的复制，目标目录不存在则报错  Path file = Paths.*get*("first", "second","io", "1.txt");  Path dest = Paths.*get*("first", "second","io", "2.txt");  Files.*copy*(file, dest, StandardCopyOption.***REPLACE\_EXISTING***);  }    @Test  **public** **void** test5()**throws** IOException{  // Path move(Path src, Path dest, CopyOption…how) : 将 src 移动到 dest 位置  Path file = Paths.*get*("first", "second","io", "1.txt");  Path target = Paths.*get*("io", "2.txt");  Files.*move*(file, target, StandardCopyOption.***REPLACE\_EXISTING***,StandardCopyOption.***ATOMIC\_MOVE***);  // StandardCopyOption.ATOMIC\_MOVE要么一起成功，要么不移动  }    @Test  **public** **void** test6()**throws** IOException{  // long size(Path path) : 返回 path 指定文件的大小  Path file = Paths.*get*("first", "second","io", "1.txt");  **long** size = Files.*size*(file);  System.***out***.println(size);  }  } |

Files常用方法：用于判断

* boolean exists(Path path, LinkOption … opts) : 判断文件是否存在
* boolean notExists(Path path, LinkOption … opts) : 判断文件是否不存在
* boolean isDirectory(Path path, LinkOption … opts) : 判断是否是目录
* boolean isRegularFile(Path path, LinkOption … opts) : 判断是否是文件
* boolean isHidden(Path path) : 判断是否是隐藏文件
* boolean isReadable(Path path) : 判断文件是否可读
* boolean isWritable(Path path) : 判断文件是否可写

Files常用方法：用于操作内容

* + InputStream newInputStream(Path path, OpenOption…how):获取 InputStream 对象
  + OutputStream newOutputStream(Path path, OpenOption…how) : 获取 OutputStream 对象

Files其他方法：public static List<String> readAllLines(Path path,Charset cs)

|  |
| --- |
| package com.atguigu.files;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStreamReader;  import java.nio.charset.Charset;  import java.nio.file.Files;  import java.nio.file.Paths;  import java.util.ArrayList;  import java.util.List;  /\*  \* 需求：读取一个文本文件的内容，以行读取，把文件的内容封装到一个集合中  \*/  public class TestFiles {  public static void main(String[] args) {  // ArrayList<String> readFile = readFile("1.txt","GBK");  // for (String string : readFile) {  // System.out.println(string);  // }    try {  List<String> readAllLines = Files.readAllLines(Paths.get("1.txt"),Charset.forName("GBK"));    } catch (IOException e) {  e.printStackTrace();  }  }  public static ArrayList<String> readFile(String filename,String charset){  File file = new File(filename);  FileInputStream fr = null;  InputStreamReader isr = null;  BufferedReader br = null;  ArrayList<String> list = new ArrayList<String>();    try {  fr = new FileInputStream(file);  isr = new InputStreamReader(fr,charset);  br = new BufferedReader(isr);    String str;  while((str=br.readLine())!=null){  list.add(str);  }    } catch (FileNotFoundException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  }finally{  try {  if(br!=null){  br.close();  }  } catch (IOException e) {  e.printStackTrace();  }  try {  if(isr!=null){  isr.close();  }  } catch (IOException e1) {  e1.printStackTrace();  }  try {  if(fr!=null){  fr.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  return list;  }  } |

## 14.7 本章小结

通过本章的学习，你的技能得到更大的提升了，你可以做到把数据持久化到文件中，而不是仅仅存在与内存中了，也为在网络中传输数据奠定了良好的基础。

|  |
| --- |
| 问：那JDK1.4引入的NIO相关类呢？  答：java.nio这批API带来了重大的效能提升并可以充分利用执行程序的机器上的原始容量。NIO的一项关键能力是你可以直接控制buffer。另一项能力是non-blocking的输入和输出，它能让你的输入/输出程序代码在没有东西可以读取或写入时不必等在那里。某些现有的类（包括FileInputStream和FileOutputStream）会利用到其中的一部分功能。NIO使用起来更为复杂，除非你真的很需要新功能，不然的话，使用本章介绍的功能方法会简单的多。此外，如果没有很好的设计，NIO可能会引发效能损失。非NIO的输入/输出适合九成以上的应用，特别在你还是新手的时候更是如此。 |