Java的流及文件处理

Java中提供了两种方式作为输入/输出的API：流及文件。

读写数据又可以通过两种格式来进行：二进制格式和文本格式。

# 流

输入流：可以从其中读入一个字节序列的对象；

输出流：可以向其中写入一个字节序列的对象；

## 流家族

按照使用方法进行划分，形成了处理字节和字符的两个单独的层次结构。

### 处理字节

InputStream和OutputStream类可以读写单个字节或字节数组。子类用来读写字符串和数字。例如，DataInputStream和DataOutputStream可以以二进制格式读写所有的基本Java类型；ZipInputStream和ZipOutputStream可以以常见的Zip压缩格式读写文件。



#### InputStream接口

abstract int read();

从数据中读入一个字节，并返回该字节。碰到流结尾时返回-1。

int read(byte[] b);

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

读入一个字节数组，并返回实际读入的字节数，碰到流结尾时返回-1。

off：第一个读入字节应该被放置的位置在b中的偏移量；

len：读入字节的最大数量；

long skip(long n);

在输入流中跳过n个字节，返回实际跳过的字节数(如果碰到流的结尾，则可能小于n)；

int available();

返回在不阻塞的情况下课获取的字节数；

void close();

#### OutputStream接口

abstract void write(int n);

写出一个字节的数据；

void write(byte[] b);

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

写出数组b中所有字节或者某个范围的字节；

off：第一个写出字节在b中的偏移量；

len：写出字节的最大数量；

void close();

void flush();

### 处理字符

对于Unicode文本，使用Reader和Writer的子类进行读写。从其继承的类拥有的读入和写出操作都是基于两字节的Unicode码元的，而不是基于单字节的字符。



### 附加接口



#### Closeable

拥有方法：void close() throws IOException;

Closeable接口扩展了java.lang.AutoCloseable接口，因此，对任何Closeable进行操作时，都可以使用try-with-resource语句。

#### Flushable

拥有方法：void flush();

#### Readable

拥有方法：int read(CharBuffer cb);

尝试着向cb读入其可持有数量的char值。返回读入的char值的数量，或者当从这个Readable中无法再获得更多的值时返回-1。

CharBuffer类拥有按顺序和随机地进行读写访问的方法，它表示一个内存中的缓冲区或者一个内存映象的文件。

#### Appendable

拥有两个用于添加单个字符和字符序列的方法：

Appendable append(char c);

Appendable append(CharSequence s);

向这个Appenable中追加给定的码元或者给定的序列中的所有码元，返回this。

CharSequence接口描述了一个char值序列的基本属性。String、CharBuffer、StringBuilder和StringBuffer都实现了它。

### 组合流过滤器

某些流（例如FileInputStream和由URL类的openStream方法返回的输入流）可以从文件和其他更外部的位置上获取字节，而其他的流（例如DataInputStream和PrintWriter）可以将字节组装到更有用的数据类型中。

Java中通过嵌套过滤器来添加多重功能。FilterInputStream和FilterOutputStream的子类用于向原生字节流添加额外的功能。例如：

DataInputStream din = new DataInputStream(

new BufferedInputStream(

new FileInputStream(“employee.dat”)

)

);

## 文本输入输出

存储文本字符串时，需要考虑字符编码的方式。

OutputStreamWriter类将使用选定的字符编码方式，把Unicode字符流转换为字节流。而InputStreamReader类将包含字节（用某种字符编码方式表示的字符）的输入流转换为可以产生Unicode码元的读入器。可以在类的构造器中进行指定的方式来选择不同的编码方式。

### 输出

使用PrintWriter进行文本输出。这个类拥有以文本格式打印字符串和数字的方法。还有一个将PrintWriter链接到FileWriter的便捷方法。

PrintWriter out = new PrintWriter(“employee.bat”);

等同于

PrintWriter out = new PrintWriter(new FileWriter(“employee.bat”));

使用print、println和printf方法打印数字、字符、boolean值、字符串和对象。

### 输入

使用Scanner来读入文本输入。也可以使用BufferedReader类处理文本输入，但BufferedReader没有任何用于读入数字的方法，所以推荐使用Scanner。

### 字符集

Charset类统一了对字符集的转换。字符集建立了两字节Unicode码元序列与使用本地字符编码方式的字节序列之间的映射。为了兼容其他的命名惯例，每个字符集都可以拥有许多别名。字符集名字是大小写不敏感的。

本地编码方式模式不能表示所有的Unicode字符，如果某个字符不能表示，它将被转换成？。

Charset的接口为：

static SrotedMap availableCharsets();

获取这个虚拟机可用的所有字符集。返回一个映射表，它的键是字符集的名字，值是字符集。

static Charset forName(String name);

获取给定名字的字符集。

Set aliases();

返回这个字符集的别名集。

ByteBuffer encode(String str);

将给定的字符串编码为字节序列。

CharBuffer decode(ByteBuffer buffer);

解码给定的字节序列。无法识别的输入将被转换成Unicode的“替代字符”(‘\uFFFD’)。

## 二进制输入输出

### 写出

DataOutput接口定义了用于以二进制格式写数组、字符、boolean值和字符串的方法：

writeChars、writeByte、writeInt、writeShort、writeLong、writeFloat、writeDouble、writeChar、writeBoolean、writeUTF。

writeInt总是将一个整数写出为4字节的二进制数量值，而不管它有多少位；writeDouble总是将一个double值写出为8字节的二进制数量值。

DataOutputStream实现了DataOutput接口。

### 读入

DataInput接口定义了读回数据的接口：

readInt、readShort、readLong、readFloat、readDouble、readChar、readBoolean、readUTF。

DataInputStream类实现了DataInput接口。

### 随机访问文件

RandomAccessFile类可以在文件中的任何位置查找和写入数据。磁盘文件是随机访问的。

RandomAccessFile(String file, String mode);

RandomAccessFile(File file, String mode);

mode：”r”表示只读模式；”rw”表示读/写模式；”rws”表示每次更新时，都对数据和元数据的写磁盘操作进行同步的读/写模式；”rwd”表示每次更新时，只对数据的写磁盘操作进行同步的读/写模式。

long getFilePointer();

返回文件指针的当前位置。

void seek(long pos);

将文件指针设置到距文件开头pos个字节处。

long length();

返回文件按照字节来度量的长度。

# 文件

Path和Files类封住了在用户机器上处理文件系统所需的所有功能。

## Path

Path表示的是一个目录名序列，其后还可以跟着一个文件名。静态的Paths.get方法接受一个或多个字符串，并将它们用默认文件系统的路径分隔符连接起来，返回一个Path对象。如果表示的不是给定文件系统中的合法路径，抛出InvalidPathException异常。

Path resolve(Path other);

Path resolve(String other);

如果other是绝对路径，那么返回other；否则，返回通过连接this和other获得的路径。

Path resolveSibling(Path other);

Path resolveSibling(String other);

如果other是绝对路径，那么返回other；否则，返回通过连接this的父路径和other获得的路径。

Path relativize(Path other);

返回用this进行解析，相对于other的相对路径。

Path normalize();

移除诸如.和..等冗余的路径元素。

Path toAbsolutePath();

返回与该路径等价的绝对路径。

其他简单接口：getParent()、getFileName()、getRoot()、toFile()等。

## Files

### 读写文件

Files提供的读写方法适用于处理中等长度的文本文件，如果要处理的文件长度比较大，或者是二进制文件，就应该使用流或者读入器/写出器。

static byte[] readAllBytes(Path path);

static List<String> readAllLines(Path path, Charset charset);

读入文件的内容。

static Path write(Path path, byte[] contents, OpenOption… options);

static Path write(Path path, Iterable<? extends CharSequence> contents, OpenOption options);

将给定的内容写出到文件中，并返回path。

static InputStream newInputStream(Path path, OpenOption… options);

static OutputStream newOutputStream(Path path, OpenOption… options);

static BufferedReader newBufferedReader(Path path, Charset charset);

static BufferedWriter newBufferedWriter(Path path, Charset charset);

打开一个文件，用于读入或写出。

### 复制、移动和删除文件

static Path copy(Path from, Path to, CopyOption… options);

static Path move(Path from, Path to, CopyOption… options);

将from复制或移动到给定位置，并返回to。

如果目标路径已经存在，复制或移动将失败。如果想要覆盖已有的目标路径，可以使用REPLACE\_EXISTING选项。如果想要复制所有的文件属性，可以使用COPY\_ATTRIBUTES选项。也可以指定多个选项。

static void delete(Path path);

static boolean deleteIfExists(Path path);

删除给定文件或空目录。第一个方法在文件或目录不存在时抛出异常，第二个方法在这种情况下返回false。

### 创建文件和目录

static Path createFile(Path path, FileAttribute<?>… attrs);

static Path createDirectory(Path path, FileAttribute<?>… attrs);

static Path createDirectories(Path path, FileAttribute<?>… attrs);

创建一个文件或目录，createDirectories还会创建路径中所有的中间目录。

如果文件已存在，createFile会抛出异常。检查文件是否存在和创建文件是原子性的，如果文件不存在，那文件就会被创建，并且其他程序在此过程中是无法执行文件创建操作的。

static Path createTempFile(String prefix, String suffix, FileAttribute<?>… attrs);

static Path createTempFile(Path parentDir, String prefix, String suffix, FileAttribute<?>… attrs);

static Path createTempDirectory(String prefix, FileAttribute<?>… attrs);

static Path createTempDirectory(Path parentDir, String prefix, FileAttribute<?>… attrs);

在适合临时文件的位置，或者在给定的父目录中，创建一个临时文件或目录。返回所创建的文件或目录的路径。

prefix和suffix可以为null。

### 获取文件信息

检查由路径指定的文件的给定属性：exists()、isHidden()、isReadable()、isWriter()、isExecutable()、isRegularFile()、isDirectory()、isSymbolicLink(),这些函数都是静态，返回boolean。

static long size(Path path);

获取文件按字节数度量的尺寸。

A readAttributes(Path path, Class<A> type, LinkOption… options);

读取类型为A的文件属性。调用方法类似：

BasicFileAttributes attributes = files.readAttributes(path, BasicFileAttributes.class);

BasicFileAttributes接口包含基本的文件属性：

创建文件、最后一次访问和最后一次修改文件的时间，这些时间被表示为java.nio.file.attribute.FileTime。createionTime()、lastAccessTime()、lastModifiedTime();

文件是常规文件、目录还是符号链接，或者三者都不是。 isRegularFile()、isDirectory()、isSymbolicLink()；

文件尺寸。；

文件主键，这时某种类的对象，具体所属类与文件系统相关，有可能是文件的唯一标识符，也可能不是。filekey()。

### 迭代目录中的文件

#### 迭代目录中的文件

File类中的listFiles方法用来获取由一个目录中的所有文件构成的数组，但是当目录中包含大量的文件时，这个方法的性能非常低。因此，Files提供了newDirectoryStream()方法，它可以产生一个Iterable对象。

DirectoryStream<Path> newDirectoryStream(Path path);

DirectoryStream<Path> newDirectoryStream(Path path, String glob);

获取给定目录中可以遍历所有文件和目录的迭代器。第二个方法接受那些与给定的glob模式匹配的项。

Glob模式：

|  |  |  |
| --- | --- | --- |
| 模式 | 描述 | 示例 |
| \* | 匹配路径组成部分中0个或多个字符 | \*.java匹配当前目录中的所有java文件 |
| \*\* | 匹配跨目录边界的0个或多个字符 | \*\*.java匹配在所有子目录中的java文件 |
| ? | 匹配一个字符 | ????.java匹配所有四个字符的java文件 |
| […] | 匹配一个字符集合，可以使用连线符[0-9]和取反符[!0-9] | Test[0-9A-F].java匹配Testx.java，其中x是一个十六进制数字 |
| {…} | 匹配由逗号隔开的多个可选项之一 | \*.{java,class}匹配所有的Java文件和类class文件 |
| \ | 转义上述任意模式中的字符 | \*\\*\*匹配所有文件名中包含\*的文件 |

如果使用windows的glob语法，必须对反斜杠转义两次：一次为glob语法转义，一次为java字符串转义：Files.newDirectoryStream(dir, “C:\\\\”)。

#### 迭代目录中的所有子孙文件

调用wakFileTree方法，并向其传递一个FileVisitor类型的对象，这个对象会得到下列通知：

* 在遇到一个文件或目录时：FileVisitResult visitFile(T path, BasicFileAttributes attrs);
* 在一个目录被处理前：FileVisitResult preVisitDirectory(T dir, IOException ex);
* 在一个目录被处理后：FileVisitResult postVisitDirectory(T dir, IOException ex);
* 在试图访问文件或目录时发生错误，例如没有权限打开目录：FileVisitResult visitFileFailed(path, IOException);

对于上述每种情况，都可以指定是否希望执行下面的操作：

* 继续访问下一个文件：FileVisitResult.CONTINUE;
* 继续访问，但是不再访问这个目录下的任何项：FileVisitResult.\_SKIP\_SUBTREE;
* 继续访问，但是不再访问这个文件的兄弟文件：FileVisitResult.SKIP\_SIBLINGS;
* 终止访问：FileVisitResult.TERMINATE;
* 当有任何方法抛出异常时，就会终止访问；

SimpleFileVisitor实现了FileVisitor接口。它出visitFileFailed方法之外的所有方法不做任何处理而是直接继续访问，visitFileFailed方法会抛出由失败导致的异常，并进而终止访问。