Java 方法 Java 异常处理

Java 流(Stream)、文件(File)和IO

Java.io 包几乎包含了所有操作输入、输出需要的类。所有这些流类代表了输入源和输出目标。

Java.io 包中的流支持很多种格式，比如：基本类型、对象、本地化字符集等等。

一个流可以理解为一个数据的序列。输入流表示从一个源读取数据，输出流表示向一个目标写数据。

Java 为 I/O 提供了强大的而灵活的支持，使其更广泛地应用到文件传输和网络编程中。

但本节讲述最基本的和流与 I/O 相关的功能。我们将通过一个个例子来学习这些功能。

读取控制台输入

Java 的控制台输入由 System.in 完成。

为了获得一个绑定到控制台的字符流，你可以把 System.in 包装在一个 BufferedReader 对象中来创建一个字符流。

下面是创建 BufferedReader 的基本语法：

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

BufferedReader 对象创建后，我们便可以使用 read() 方法从控制台读取一个字符，或者用 readLine() 方法读取一个字符串。

从控制台读取多字符输入

从 BufferedReader 对象读取一个字符要使用 read() 方法，它的语法如下：

```
int read( ) throws IOException
```

每次调用 read() 方法，它从输入流读取一个字符并把该字符作为整数值返回。 当流结束的时候返回 -1。该方法抛出 IOExcepti
on。

下面的程序示范了用 read() 方法从控制台不断读取字符直到用户输入 "q"。

BRRead.java 文件代码：

```
//使用 BufferedReader 在控制台读取字符
import java.io.*;
public class BRRead {
public static void main(String args[]) throws IOException {
char c;
// 使用 System.in 创建 BufferedReader
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("输入字符, 按下 'q' 键退出。");
// 读取字符
do {
c = (char) br.read();
System.out.println(c);
} while (c != 'q');
}
}
```

以上实例编译运行结果如下:

```
输入字符，按下 'q' 键退出。
runoob
r
u
n
o
o
b


q
q
```

从控制台读取字符串

从标准输入读取一个字符串需要使用 BufferedReader 的 readLine() 方法。

它的一般格式是：

```
String readLine( ) throws IOException
```

下面的程序读取和显示字符行直到你输入了单词"end"。

BRReadLines.java 文件代码：

```
//使用 BufferedReader 在控制台读取字符
import java.io.*;
public class BRReadLines {
public static void main(String args[]) throws IOException {
// 使用 System.in 创建 BufferedReader
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str;
System.out.println("Enter lines of text.");
System.out.println("Enter 'end' to quit.");
do {
str = br.readLine();
System.out.println(str);
} while (!str.equals("end"));
}
}
```

以上实例编译运行结果如下:

```
Enter lines of text.
Enter 'end' to quit.
This is line one
This is line one
This is line two
```

```
This is line two
end
end
```

JDK 5 后的版本我们也可以使用 Java Scanner 类来获取控制台的输入。

控制台输出

在此前已经介绍过，控制台的输出由 print( ) 和 println() 完成。这些方法都由类 PrintStream 定义，System.out 是该类对象的一个引用。

PrintStream 继承了 OutputStream类，并且实现了方法 write()。这样，write() 也可以用来往控制台写操作。

PrintStream 定义 write() 的最简单格式如下所示：

```
void write(int byteval)
```

该方法将 byteval 的低八位字节写到流中。

实例

下面的例子用 write() 把字符 "A" 和紧跟着的换行符输出到屏幕：

WriteDemo.java 文件代码：

```
import java.io.*;
//演示 System.out.write().
public class WriteDemo {
public static void main(String args[]) {
int b;
b = 'A';
System.out.write(b);
System.out.write('\n');
}
}
```

运行以上实例在输出窗口输出 "A" 字符

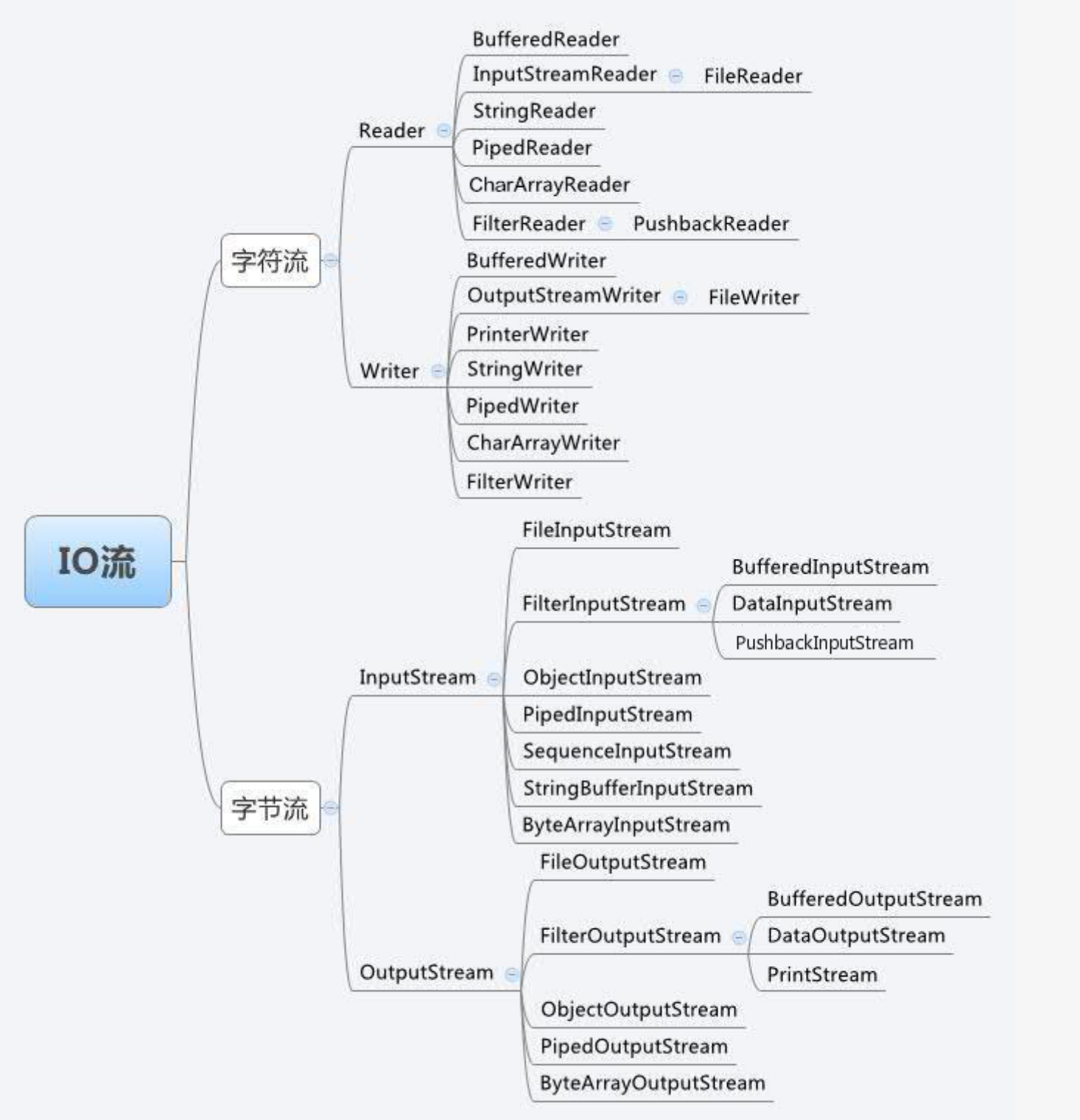
```
A
```

注意：write() 方法不经常使用，因为 print() 和 println() 方法用起来更为方便。

读写文件

如前所述，一个流被定义为一个数据序列。输入流用于从源读取数据，输出流用于向目标写数据。

下图是一个描述输入流和输出流的类层次图。



下面将要讨论的两个重要的流是 FileInputStream 和 FileOutputStream：

FileInputStream

该流用于从文件读取数据，它的对象可以用关键字 new 来创建。

有多种构造方法可用来创建对象。

可以使用字符串类型的文件名来创建一个输入流对象来读取文件：

```
InputStream f = new FileInputStream("C:/java/hello");
```

也可以使用一个文件对象来创建一个输入流对象来读取文件。我们首先得使用 File() 方法来创建一个文件对象：

```
File f = new File("C:/java/hello");
InputStream out = new FileInputStream(f);
```

创建了InputStream对象，就可以使用下面的方法来读取流或者进行其他的流操作。

序号	方法及描述
1	**public void close() throws IOException{}** 关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。
2	**protected void finalize()throws IOException {}** 这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。
3	**public int read(int r)throws IOException{}** 这个方法从 InputStream 对象读取指定字节的数据。返回为整数值。返回下一字节数据，如果已经到结尾则返回-1。
4	**public int read(byte[] r) throws IOException{}** 这个方法从输入流读取r.length长度的字节。返回读取的字节数。如果是文件结尾则返回-1。
5	**public int available() throws IOException{}** 返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取的字节数。返回一个整数值。

除了 InputStream 外，还有一些其他的输入流，更多的细节参考下面链接：

- ByteArrayInputStream
- DataInputStream

FileOutputStream

该类用来创建一个文件并向文件中写数据。

如果该流在打开文件进行输出前，目标文件不存在，那么该流会创建该文件。

有两个构造方法可以用来创建 FileOutputStream 对象。

使用字符串类型的文件名来创建一个输出流对象：

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

也可以使用一个文件对象来创建一个输出流来写文件。我们首先得使用File()方法来创建一个文件对象：

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

创建OutputStream 对象完成后，就可以使用下面的方法来写入流或者进行其他的流操作。

序号	方法及描述

1	**public void close() throws IOException{}** 关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。
2	**protected void finalize()throws IOException {}** 这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。
3	**public void write(int w)throws IOException{}** 这个方法把指定的字节写到输出流中。
4	**public void write(byte[] w)** 把指定数组中w.length长度的字节写到OutputStream中。

除了OutputStream外，还有一些其他的输出流，更多的细节参考下面链接：

- ByteArrayOutputStream
- DataOutputStream

实例

下面是一个演示 InputStream 和 OutputStream 用法的例子：

fileStreamTest.java 文件代码：

```
import java.io.*;
public class fileStreamTest {
public static void main(String args[]) {
try {
byte bWrite[] = { 11, 21, 3, 40, 5 };
OutputStream os = new FileOutputStream("test.txt");
for (int x = 0; x < bWrite.length; x++) {
os.write(bWrite[x]); // writes the bytes
}
os.close();
InputStream is = new FileInputStream("test.txt");
int size = is.available();
for (int i = 0; i < size; i++) {
System.out.print((char) is.read() + " ");
}
is.close();
} catch (IOException e) {
System.out.print("Exception");
}
}
}
```

上面的程序首先创建文件test.txt，并把给定的数字以二进制形式写进该文件，同时输出到控制台上。

以上代码由于是二进制写入，可能存在乱码，你可以使用以下代码实例来解决乱码问题：

fileStreamTest2.java 文件代码：

```
//文件名 :fileStreamTest2.java
import java.io.*;
public class fileStreamTest2 {
public static void main(String[] args) throws IOException {
File f = new File("a.txt");
FileOutputStream fop = new FileOutputStream(f);
// 构建FileOutputStream对象,文件不存在会自动新建
OutputStreamWriter writer = new OutputStreamWriter(fop, "UTF-8");
// 构建OutputStreamWriter对象,参数可以指定编码,默认为操作系统默认编码,windows上是gbk
writer.append("中文输入");
// 写入到缓冲区
writer.append("\r\n");
// 换行
writer.append("English");
// 刷新缓存冲,写入到文件,如果下面已经没有写入的内容了,直接close也会写入
writer.close();
// 关闭写入流,同时会把缓冲区内容写入文件,所以上面的注释掉
fop.close();
// 关闭输出流,释放系统资源
FileInputStream fip = new FileInputStream(f);
// 构建FileInputStream对象
InputStreamReader reader = new InputStreamReader(fip, "UTF-8");
// 构建InputStreamReader对象,编码与写入相同
StringBuffer sb = new StringBuffer();
while (reader.ready()) {
sb.append((char) reader.read());
// 转成char加到StringBuffer对象中
}
System.out.println(sb.toString());
reader.close();
// 关闭读取流
fip.close();
// 关闭输入流,释放系统资源
}
}
```

文件和I/O

还有一些关于文件和I/O的类，我们也需要知道：

- File Class(类)
- FileReader Class(类)
- FileWriter Class(类)

Java中的目录

创建目录：

File类中有两个方法可以用来创建文件夹：

- **mkdir( )**方法创建一个文件夹，成功则返回true，失败则返回false。失败表明File对象指定的路径已经存在，或者由于整个路径还不存在，该文件夹不能被创建。
- **mkdirs()**方法创建一个文件夹和它的所有父文件夹。

下面的例子创建 "/tmp/user/java/bin"文件夹：

CreateDir.java 文件代码：

```
import java.io.File;
public class CreateDir {
public static void main(String args[]) {
String dirname = "/tmp/user/java/bin";
File d = new File(dirname);
// 现在创建目录
d.mkdirs();
}
}
```

编译并执行上面代码来创建目录 "/tmp/user/java/bin"。

注意： Java 在 UNIX 和 Windows 自动按约定分辨文件路径分隔符。如果你在 Windows 版本的 Java 中使用分隔符 (/) ，路径依然能够被正确解析。

读取目录

一个目录其实就是一个 File 对象，它包含其他文件和文件夹。

如果创建一个 File 对象并且它是一个目录，那么调用 isDirectory() 方法会返回 true。

可以通过调用该对象上的 list() 方法，来提取它包含的文件和文件夹的列表。

下面展示的例子说明如何使用 list() 方法来检查一个文件夹中包含的内容：

DirList.java 文件代码：

```
import java.io.File;
public class DirList {
public static void main(String args[]) {
String dirname = "/tmp";
File f1 = new File(dirname);
if (f1.isDirectory()) {
System.out.println("目录 " + dirname);
String s[] = f1.list();
for (int i = 0; i < s.length; i++) {
File f = new File(dirname + "/" + s[i]);
if (f.isDirectory()) {
System.out.println(s[i] + " 是一个目录");
} else {
System.out.println(s[i] + " 是一个文件");
}
}
} else {
System.out.println(dirname + " 不是一个目录");
}
}
}
```

以上实例编译运行结果如下：

```
目录 /tmp
bin 是一个目录
lib 是一个目录
demo 是一个目录
test.txt 是一个文件
README 是一个文件
index.html 是一个文件
include 是一个目录
```

删除目录或文件

删除文件可以使用 **java.io.File.delete()** 方法。

以下代码会删除目录 **/tmp/java/**，需要注意的是当删除某一目录时，必须保证该目录下没有其他文件才能正确删除，否则将删除失败。

测试目录结构：

```
/tmp/java/
|-- 1.log
|-- test
```

DeleteFileDemo.java 文件代码：

```
import java.io.File;
public class DeleteFileDemo {
public static void main(String args[]) {
// 这里修改为自己的测试目录
File folder = new File("/tmp/java/");
deleteFolder(folder);
}
// 删除文件及目录
public static void deleteFolder(File folder) {
File[] files = folder.listFiles();
if (files != null) {
for (File f : files) {
if (f.isDirectory()) {
deleteFolder(f);
} else {
f.delete();
}
}
}
folder.delete();
}
}
```

Java 方法 Java 异常处理

4 篇笔记 **写笔记**