第 23 天 IO 流

今日内容介绍

- 字节流
- 字符流

第 1 章 字节流

在前面的学习过程中，我们一直都是在操作文件或者文件夹，并没有给文件中写任何数据。现在我们就要开始给文件中写数据，或者读取文件中的数据。

1.1字节输出流 OutputStream

OutputStream 此抽象类，是表示输出字节流的所有类的超类。操作的数据都是字节，定义了输出字节流的基本共性功能方法。

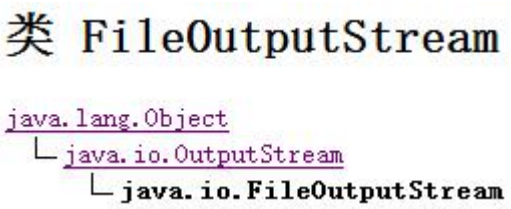
输出流中定义都是写 write 方法，如下图:

void	**close**() 关闭此输出流并释放与此流有关的所有系统资源。
void	**flush**() 刷新此输出流并强制写出所有缓冲的输出字节。
void	**write**(byte[] b) 将 b.length 个字节从指定的 byte 数组写入此输出流。
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。
abstract void	**write**(int b) 将指定的字节写入此输出流。

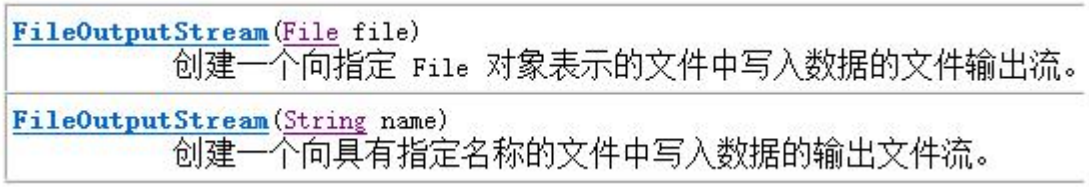
1.1.1 FileOutputStream 类

OutputStream 有很多子类，其中子类 FileOutputStream 可用来写入数据到文件。

FileOutputStream 类，即文件输出流，是用于将数据写入 File 的输出流。



- 构造方法



1.1.2 FileOutputStream 类写入数据到文件中

- 将数据写到文件中，代码演示：

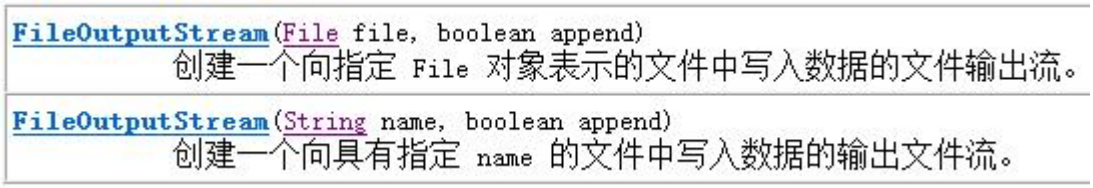
```
public class FileOutputStreamDemo {
    public static void main(String[] args) throws IOException {
        //需求：将数据写入到文件中。
        //创建存储数据的文件。
        File file = new File("c:\\file.txt");
        //创建一个用于操作文件的字节输出流对象。一创建就必须明确数据存储目的地。
        //输出流目的是文件，会自动创建。如果文件存在，则覆盖。
        FileOutputStream fos = new FileOutputStream(file);
        //调用父类中的 write 方法。
        byte[] data = "abcde".getBytes();
        fos.write(data);
        //关闭流资源。
        fos.close();
    }
}
```

1.1.3 给文件中续写和换行

我们直接 **new** FileOutputStream(file)这样创建对象，写入数据，会覆盖原有的文件，那么我们想在原有的文件中续写内容怎么办呢？

继续查阅 FileOutputStream 的 API。发现在 FileOutputStream 的构造函数中，可以接受一个 boolean 类型的值，如果值 true，就会在文件末位继续添加。

- 构造方法



- 给文件中续写数据和换行，代码演示：

```
public class FileOutputStreamDemo2 {
    public static void main(String[] args) throws Exception {
        File file = new File("c:\\file.txt");
        FileOutputStream fos = new FileOutputStream(file, true);
        String str = "\r\n"+"itcast";
        fos.write(str.getBytes());
        fos.close();
    }
}
```

1.1.4 IO 异常的处理

在前面编写代码中都发生了 IO 的异常。我们在实际开发中，对异常时如何处理的，我们来演示一下。

```
public class FileOutputStreamDemo3 {
    public static void main(String[] args) {
        File file = new File("c:\\file.txt");
        //定义 FileOutputStream 的引用
        FileOutputStream fos = null;
        try {
            //创建 FileOutputStream 对象
            fos = new FileOutputStream(file);
```

```
                //写出数据
                fos.write("abcde".getBytes());
            } catch (IOException e) {
                System.out.println(e.toString() + "----");
            } finally {
                //一定要判断 fos 是否为 null，只有不为 null 时，才可以关闭资源
                if (fos != null) {
                    try {
                        fos.close();
                    } catch (IOException e) {
                        throw new RuntimeException("");
                    }
                }
            }
        }
    }
```

1.2字节输入流 InputStream

通过前面的学习，我们可以把内存中的数据写出到文件中，那如何想把内存中的数据读到内存中，我们通过 InputStream 可以实现。InputStream 此抽象类，是表示字节输入流的所有类的超类。，定义了字节输入流的基本共性功能方法。

abstract int	read() 从输入流中读取数据的下一个字节。
int	read(byte[] b) 从输入流中读取一定数量的字节，并将其存储在缓冲区数组 b 中。

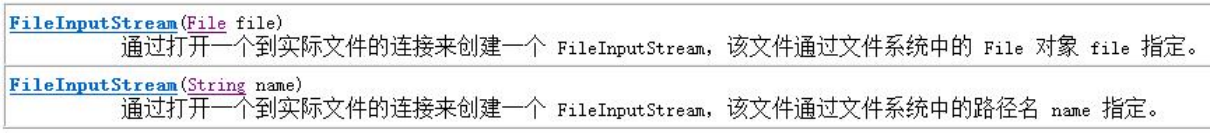
- int read():读取一个字节并返回，没有字节返回-1.
- int read(byte[]): 读取一定量的字节数，并存储到字节数组中，返回读取到的字节数。

1.2.1 FileInputStream 类

InputStream 有很多子类，其中子类 FileInputStream 可用来读取文件内容。

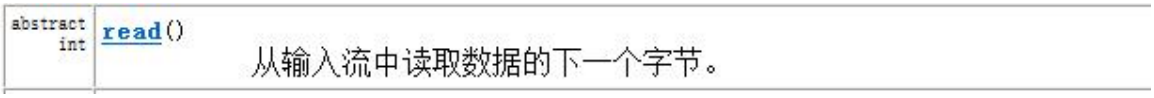
FileInputStream 从文件系统中的某个文件中获得输入字节。

- 构造方法



1.2.2 FileInputStream 类读取数据 read 方法

在读取文件中的数据时，调用 read 方法，实现从文件中读取数据



- 从文件中读取数据，代码演示：

```
public class FileInputStreamDemo {
    public static void main(String[] args) throws IOException {
        File file = new File("c:\\file.txt");
        //创建一个字节输入流对象,必须明确数据源，其实就是创建字节读取流和数据源相关联。
        FileInputStream fis = new FileInputStream(file);
        //读取数据。使用 read();一次读一个字节。
        int ch = 0;
        while((ch=fis.read())!=-1){
            System.out.pr          }intln("ch="+(char)ch);

        // 关闭资源。
        fis.close();
    }
}
```

1.2.3 读取数据 read(byte[])方法

在读取文件中的数据时，调用 read 方法，每次只能读取一个，太麻烦了，于是我们可以定义

数组作为临时的存储容器，这时可以调用重载的 read 方法，一次可以读取多个字符。

int	read(byte[] b) 从输入流中读取一定数量的字节，并将其存储在缓冲区数组 b 中。

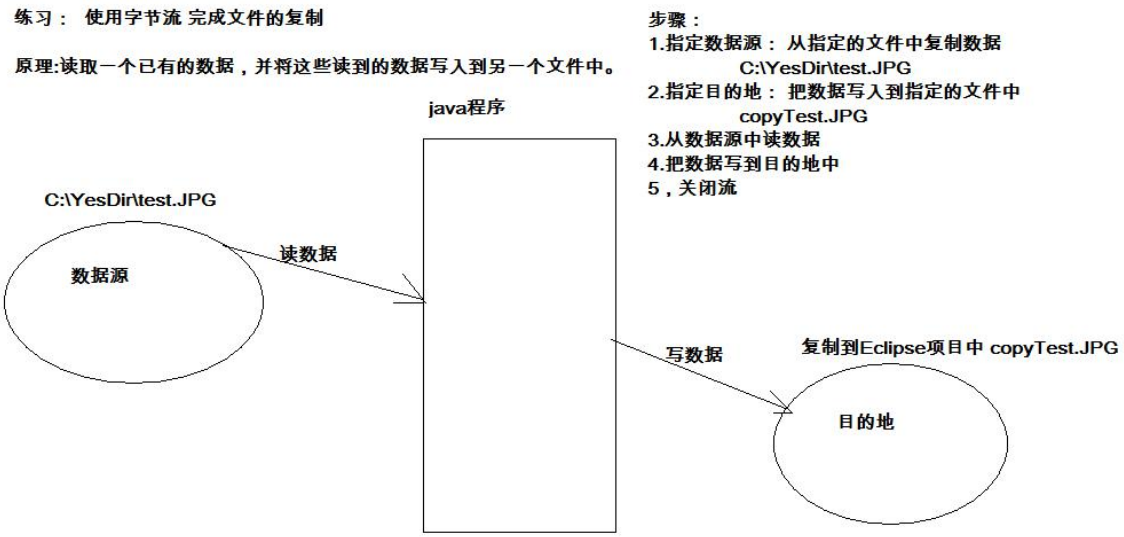
```
public class FileInputStreamDemo2 {
    public static void main(String[] args) throws IOException {
        /*
         * 演示第二个读取方法， read(byte[]);
         */
        File file = new File("c:\\file.txt");
        // 创建一个字节输入流对象,必须明确数据源，其实就是创建字节读取流和数据源相关联。
        FileInputStream fis = new FileInputStream(file);
        //创建一个字节数组。
        byte[] buf = new byte[1024];//长度可以定义成 1024 的整数倍。
        int len = 0;
        while((len=fis.read(buf))!=-1){
            System.out.println(new String(buf,0,len));
        }
        fis.close();
    }
}
```

1.3字节流练习

既然会了文件的读和写操作了，那么我们就要在这个基础上进行更为复杂的操作。使用读写操作完成文件的复制。

1.3.1 复制文件

原理；读取一个已有的数据，并将这些读到的数据写入到另一个文件中。




```
public class CopyFileTest {
    public static void main(String[] args) throws IOException {
        //1,明确源和目的。
        File srcFile = new File("c:\\YesDir\test.JPG");
        File destFile = new File("copyTest.JPG");

        //2,明确字节流 输入流和源相关联，输出流和目的关联。
        FileInputStream fis = new FileInputStream(srcFile);
        FileOutputStream fos = new FileOutputStream(destFile);

        //3，使用输入流的读取方法读取字节，并将字节写入到目的中。
        int ch = 0;
        while((ch=fis.read())!=-1){
            fos.write(ch);
        }
        //4,关闭资源。
        fos.close();
        fis.close();
    }
}
```

上述代码输入流和输出流之间是通过 ch 这个变量进行数据交换的。

上述复制文件有个问题，每次都从源文件读取一个，然后在写到指定文件，接着再读取一个字符，然后再写一个，一直这样下去。效率极低。

1.3.2 缓冲数组方式复制文件

上述代码复制文件效率太低了，并且频繁的从文件读数据，和写数据，能不能一次多把文件中多个数据都读进内容中，然后在一次写出去，这样的速度一定会比前面代码速度快。

```
public class CopyFileByBufferTest {
    public static void main(String[] args) throws IOException {
        File srcFile = new File("c:\\YesDir\test.JPG");
        File destFile = new File("copyTest.JPG");
        // 明确字节流 输入流和源相关联，输出流和目的关联。
        FileInputStream fis = new FileInputStream(srcFile);
        FileOutputStream fos = new FileOutputStream(destFile);
        //定义一个缓冲区。
        byte[] buf = new byte[1024];
        int len = 0;
        while ((len = fis.read(buf)) != -1) {
            fos.write(buf, 0, len);// 将数组中的指定长度的数据写入到输出流中。
        }
        // 关闭资源。
        fos.close();
        fis.close();
    }
}
```

第 2 章 字符流

经过前面的学习，我们基本掌握的文件的读写操作，在操作过程中字节流可以操作所有数据，可是当我们操作的文件中有中文字符，并且需要对中文字符做出处理时怎么办呢？

2.1字节流读取字符的问题

通过以下程序读取带有中文件的文件。

```
public class CharStreamDemo {
    public static void main(String[] args) throws IOException {
        //给文件中写中文
        writeCNText();
        //读取文件中的中文
```

```
            readCNText();
        }
        //读取中文
        public static void readCNText() throws IOException {
            FileInputStream fis = new FileInputStream("c:\\cn.txt");
            int ch = 0;
            while((ch = fis.read())!=-1){
                System.out.println(ch);
            }
        }
        //写中文
        public static void writeCNText() throws IOException {
            FileOutputStream fos = new FileOutputStream("c:\\cn.txt");
            fos.write("a 传智播客欢迎你".getBytes());
            fos.close();
        }
    }
```

上面程序在读取含有中文的文件时，我们并没有看到具体的中文，而是看到一些数字，这是什么原因呢？既然看不到中文，那么我们如何对其中的中文做处理呢？要解决这个问题，我们必须研究下字符的编码过程。

2.2字符编码表

我们知道计算机底层数据存储的都是二进制数据，而我们生活中的各种各样的数据，如何才能和计算机中存储的二进制数据对应起来呢？

这时老美他们就把每一个字符和一个整数对应起来，就形成了一张编码表，老美他们的编码表就是 ASCII 表。其中就是各种英文字符对应的编码。

编码表：其实就是生活中字符和计算机二进制的对应关系表。

1、ascii： 一个字节中的 7 位就可以表示。对应的字节都是正数。0-xxxxxxx

2、iso-8859-1:拉丁码表 latin，用了一个字节用的 8 位。1-xxxxxxx 负数。

3、GB2312:简体中文码表。包含 6000-7000 中文和符号。用两个字节表示。两个字节第一个字节是负数,第二个字节可能是正数

GBK:目前最常用的中文码表，2 万的中文和符号。用两个字节表示，其中的一部分文字，第一个字节开头是 1，第二字节开头是 0

GB18030：最新的中文码表，目前还没有正式使用。

1、unicode：国际标准码表:无论是什么文字，都用两个字节存储。

- Java 中的 char 类型用的就是这个码表。char c = 'a';占两个字节。
- Java 中的字符串是按照系统默认码表来解析的。简体中文版 字符串默认的码表是 GBK。

5、UTF-8:基于 unicode，一个字节就可以存储数据，不要用两个字节存储，而且这个码表更加的标准化，在每一个字节头加入了编码信息(后期到 api 中查找)。

能识别中文的码表：GBK、UTF-8；正因为识别中文码表不唯一，涉及到了编码解码问题。

对于我们开发而言；常见的编码 GBK UTF-8 ISO-8859-1

文字--->(数字) ：编码。 “abc”.getBytes() byte[]

(数字)--->文字 ： 解码。 byte[] b={97,98,99} new String(b)

2.3字符输入流 Reader

上述程序中我们读取拥有中文的文件时，使用的字节流在读取，那么我们读取到的都是一个一个字节。只要把这些字节去查阅对应的编码表，就能够得到与之对应的字符。API 中是否给我们已经提供了读取相应字符的功能流对象，Reader，读取字符流的抽象超类。

int	read() 读取单个字符。
int	read(char[] cbuf) 将字符读入数组。

- read():读取单个字符并返回
- read(char[]):将数据读取到数组中，并返回读取的个数。

2.3.1 FileReader 类

查阅 FileInputStream 的 API，发现 FileInputStream 用于读取诸如图像数据之类的原始字节流。要读取字符流，请考虑使用 FileReader。

打开 FileReader 的 API 介绍。用来读取字符文件的便捷类。此类的构造方法假定默认字符编码和默认字节缓冲区大小都是适当的

- 构造方法

2.3.2 FileReader 读取包含中文的文件

- 使用 FileReader 读取包含中文的文件

```
public class CharStreamDemo {
    public static void main(String[] args) throws IOException {
        //给文件中写中文
        writeCNText();
        //读取文件中的中文
        readCNText();
    }
    //读取中文
    public static void readCNText() throws IOException {
        FileReader fr = new FileReader("D:\\test\\cn.txt");
        int ch = 0;
        while((ch = fr.read())!=-1){
            //输出的字符对应的编码值
            System.out.println(ch);
            //输出字符本身
            System.out.println((char)ch);
        }
    }
    //写中文
    public static void writeCNText() throws IOException {
        FileOutputStream fos = new FileOutputStream("D:\\test\\cn.txt");
```

```
            fos.write("a 传智播客欢迎你".getBytes());
            fos.close();
        }
    }
```

2.4字符输出流 Writer

既然有专门用于读取字符的流对象，那么肯定也有写的字符流对象，查阅 API，发现有一个 Writer 类，Writer 是写入字符流的抽象类。其中描述了相应的写的动作。

void	write(char[] cbuf) 写入字符数组。
abstract void	write(char[] cbuf, int off, int len) 写入字符数组的某一部分。
void	write(int c) 写入单个字符。
void	write(String str) 写入字符串。
void	write(String str, int off, int len) 写入字符串的某一部分。

2.4.1 FileWriter 类

查阅 FileOutputStream 的 API ,发现 FileOutputStream 用于写入诸如图像数据之类的原始字节的流。要写入字符流，请考虑使用 FileWriter。

打开 FileWriter 的 API 介绍。用来写入字符文件的便捷类。此类的构造方法假定默认字符编码和默认字节缓冲区大小都是可接受的。

- 构造方法

FileWriter(File file) 根据给定的 File 对象构造一个 FileWriter 对象。
FileWriter(File file, boolean append) 根据给定的 File 对象构造一个 FileWriter 对象。
FileWriter(String fileName) 根据给定的文件名构造一个 FileWriter 对象。
FileWriter(String fileName, boolean append) 根据给定的文件名以及指示是否附加写入数据的 boolean 值来构造 FileWriter 对象。

2.4.2 FileWriter 写入中文到文件中

- 写入字符到文件中，先进行流的刷新，再进行流的关闭。

```
public class FileWriterDemo {
    public static void main(String[] args) throws IOException {
        //演示 FileWriter 用于操作文件的便捷类。
        FileWriter fw = new FileWriter("d：\\text\\fw.txt");
        fw.write("你好谢谢再见");//这些文字都要先编码。都写入到了流的缓冲区中。
        fw.flush();
        fw.close();
    }
}
```

2.5flush()和 close()的区别？

abstract void	close() 关闭此流，但要先刷新它。
abstract void	flush() 刷新该流的缓冲。

flush():将流中的缓冲区缓冲的数据刷新到目的地中，刷新后，流还可以继续使用。

close():关闭资源，但在关闭前会将缓冲区中的数据先刷新到目的地，否则丢失数据，然后在关闭流。流不可以使用。如果写入数据多，一定要一边写一边刷新，最后一次可以不刷新，由 close 完成刷新并关闭。

2.6字符流练习

2.6.1 复制文本文件

练习：复制文本文件。

思路：

1，既然是文本涉及编码表。需要用字符流。

2，操作的是文件。涉及硬盘。

3，有指定码表吗？没有，默认就行。

操作的是文件，使用的 默认码表。使用哪个字符流对象。直接使用字符流操作文件的便捷类。FileReader　FileWriter

```
public class CopyTextFileTest {
	public static void main(String[] args) throws IOException {
		copyTextFile();
	}
	public static void copyTextFile() throws IOException {
		//1,明确源和目的。
		FileReader fr = new FileReader("c:\\cn.txt");
		FileWriter fw = new FileWriter("c:\\copy.txt");
		//2,为了提高效率。自定义缓冲区数组。字符数组。
		char[] buf = new char[1024];
		int len = 0;
		while((len=fr.read(buf))!=-1){
			fw.write(buf,0,len);
		}
		/*2,循环读写操作。效率低。
		int ch = 0;
		while((ch=fr.read())!=-1){
			fw.write(ch);
		}
		*/
		//3,关闭资源。
		fw.close();
		fr.close();
	}
}
```

第 3 章　总结

3.1知识点总结

- IO 流的分类

  |- 字节流

- 字节输入流 InputStream 抽象类
  - FileInputStream 操作文件的字节输入流
- 字节输出流 OuputStream 抽象类
  - FileOutputStream 操作文件的字节输出流

- 字符流
  - 字符输入流 Reader 抽象类
    - InputStreamReader 输入操作的转换流
      - FileReader 用来操作文件的字符输入流（简便的流）
  - 字符输出流 Writer 抽象类
    - OutputStreamWriter 输出操作的转换流
      - FileWriter 用来操作文件的字符输出流（简便的流）