- **文件字节流**

FileInputStream通过字节的方式读取文件，适合读取所有类型的文件(图像、视频、文本文件等)。Java也提供了FileReader专门读取文本文件。

- 1.创建流对象
- 2.创建一个缓存字节的容器数组
- 3.定义一个变量，保存实际读取的字节数
- 4.循环读取数据
- 5.操作保存数据的数组
- 6.关闭流

将字符串/字节数组的内容写入到文件中

- 1.选择流：创建流对象
- 2.准备数据源，把数据源转换成字节数组类型
- 3.通过流向文件当中写入数据
- 4.刷新流
- 5.关闭流

```
1 2
3 4
5 6
7 8
9 10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

```
import java.io.FileOutputStream; import java.io.IOException;
public class TestFileOutputStream {     public static void main(String[] args
        FileOutputStream fos = null;         String string = "北京尚学堂欢迎您！
        try {             // true表示内容会追加到文件末尾；false表示重写整个文件内容。
            fos = new FileOutputStream("d:/a.txt", true);             //该方法
节数组写入文件中；而write(int n)是写入一个字节             fos.write(string.getByte
        } catch (Exception e) {             e.printStackTrace();         } fi
            try {                 if (fos != null) {                     fos.
                }             } catch (IOException e) {
                e.printStackTrace();             }         }     } }
```

void write(byte[ ] b)，该方法不再一个字节一个字节地写入，而是直接写入一个字节数组;另外其还有一个重载的方法：

void write(byte[ ] b, int off, int length)

输入流中可以被读的剩下的bytes字节的估计值：fileinputStream.available()

利用文件流实现文件的复制

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class TestFileCopy {
    public static void main(String[] args) {
        //将a.txt内容拷贝到b.txt
        copyFile("d:/a.txt", "d:/b.txt");
    }

    /**
     * 将src文件的内容拷贝到dec文件
     * @param src 源文件
     * @param dec 目标文件
     */
    static void copyFile(String src, String dec) {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        //为了提高效率，设置缓存数组！（读取的字节数据会暂存放到该字节数组中）
        byte[] buffer = new byte[1024];
        int temp = 0;
        try {
            fis = new FileInputStream(src);
            fos = new FileOutputStream(dec);
            //边读边写
            //temp指的是本次读取的真实长度，temp等于-1时表示读取结束
            while ((temp = fis.read(buffer)) != -1) {
                /*将缓存数组中的数据写入文件中，注意：写入的是读取的真实长度；
                 *如果使用fos.write(buffer)方法，那么写入的长度将会是1024，即缓存
                 *数组的长度*/
                fos.write(buffer, 0, temp);
```

```
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //两个流需要分别关闭
            try {
                if (fos != null) {
                    fos.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                if (fis != null) {
                    fis.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

1. 为了减少对硬盘的读写次数，提高效率，通常设置缓存数组。相应地，读取时使用的方法为：read(byte[] b);写入时的方法为：write(byte[ ] b, int off, int length)。

2. 程序中如果遇到多个流，每个流都要单独关闭，防止其中一个流出现异常后导致其他流无法关闭的情况。

- **文件字符流**

字节流不能很好的处理Unicode字符，经常会出现“乱码”现象。

使用FileReader与FileWriter实现文本文件的复制

```
import java.io.FileNotFoundException;
import java.io.FileReader;
```

```java
import java.io.FileWriter;
import java.io.IOException;
public class TestFileCopy2 {
    public static void main(String[] args) {
        // 写法和使用Stream基本一样。只不过，读取时是读取的字符。
        FileReader fr = null;
        FileWriter fw = null;
        int len = 0;
        try {
            fr = new FileReader("d:/a.txt");
            fw = new FileWriter("d:/d.txt");
            //为了提高效率，创建缓冲用的字符数组
            char[] buffer = new char[1024];
            //边读边写
            while ((len = fr.read(buffer)) != -1) {
                fw.write(buffer, 0, len);
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (fw != null) {
                    fw.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                if (fr != null) {
                    fr.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
        }
    }
}
```

- **缓冲字节流**

缓冲流是先将数据缓存起来，然后当缓存区存满后或者手动刷新时再一次性的读取到程序或写入目的地。

使用缓冲流实现文件的高效率复制

```
import  java.io.BufferedInputStream;
import  java.io.BufferedOutputStream;
import  java.io.FileInputStream;
import  java.io.FileOutputStream;
import  java.io.IOException;

public  class  TestBufferedFileCopy1 {

    public  static  void  main(String[] args) {
        // 使用缓冲字节流实现复制
        long  time1 = System.currentTimeMillis();
        copyFile1("D:/电影/华语/大陆/尚学堂传奇.mp4", "D:/电影/华语/大陆/尚学堂越
                "+"来越传奇.mp4");
        long  time2 = System.currentTimeMillis();
        System.out.println("缓冲字节流花费的时间为："  + (time2 - time1));

        // 使用普通字节流实现复制
        long  time3 = System.currentTimeMillis();
        copyFile2("D:/电影/华语/大陆/尚学堂传奇.mp4", "D:/电影/华语/大陆/尚学堂越
                "+"来越传奇2.mp4");
        long  time4 = System.currentTimeMillis();
        System.out.println("普通字节流花费的时间为："  + (time4 - time3));
    }
    /**缓冲字节流实现的文件复制的方法*/
```

```java
static void copyFile1(String src, String dec) {
    FileInputStream fis = null;
    BufferedInputStream bis = null;
    FileOutputStream fos = null;
    BufferedOutputStream bos = null;
    int temp = 0;
    try {
        fis = new FileInputStream(src);
        fos = new FileOutputStream(dec);
        //使用缓冲字节流包装文件字节流，增加缓冲功能，提高效率
        //缓存区的大小（缓存数组的长度）默认是8192，也可以自己指定大小
        bis = new BufferedInputStream(fis);
        bos = new BufferedOutputStream(fos);
        while ((temp = bis.read()) != -1) {
            bos.write(temp);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        //注意：增加处理流后，注意流的关闭顺序！"后开的先关闭！"
        try {
            if (bos != null) {
                bos.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            if (bis != null) {
                bis.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            if (fos != null) {
                fos.close();
```

```
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            if (fis != null) {
                fis.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
/**普通节流实现的文件复制的方法*/
static void copyFile2(String src, String dec) {
    FileInputStream fis = null;
    FileOutputStream fos = null;
    int temp = 0;
    try {
        fis = new FileInputStream(src);
        fos = new FileOutputStream(dec);
        while ((temp = fis.read()) != -1) {
            fos.write(temp);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (fos != null) {
                fos.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            if (fis != null) {
                fis.close();
```

```
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- **缓冲字符流**

使用BufferedReader与BufferedWriter实现文本文件的复制

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class TestBufferedFileCopy2 {
    public static void main(String[] args) {
        // 注：处理文本文件时，实际开发中可以用如下写法，简单高效！！
        FileReader fr = null;
        FileWriter fw = null;
        BufferedReader br = null;
        BufferedWriter bw = null;
        String tempString = "";
        try {
            fr = new FileReader("d:/a.txt");
            fw = new FileWriter("d:/d.txt");
            //使用缓冲字符流进行包装
            br = new BufferedReader(fr);
            bw = new BufferedWriter(fw);
            //BufferedReader提供了更方便的readLine()方法，直接按行读取文本
            //br.readLine()方法的返回值是一个字符串对象，即文本中的一行内容
            while ((tempString = br.readLine()) != null) {
```

```
                //将读取的一行字符串写入文件中
                bw.write(tempString);
                //下次写入之前先换行，否则会在上一行后边继续追加，而不是另起一行
                bw.newLine();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (bw != null) {
                    bw.close();
                }
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            try {
                if (fw != null) {
                    fw.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                if (fr != null) {
                    fr.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
        }
    }
}
```

- **字节数组流**

FileInputStream是把文件当做数据源。ByteArrayInputStream则是把内存中的”某个字节数组对象”（缓冲区）当做数据源。

ByteArrayInputStream 的使用

```
import java.io.ByteArrayInputStream;
import java.io.IOException;

public class TestByteArray {
    public static void main(String[] args) {
        //将字符串转变成字节数组
        byte[] b = "abcdefg".getBytes();
        test(b);
    }
    public static void test(byte[] b) {
        ByteArrayInputStream bais = null;
        StringBuilder sb = new StringBuilder();
        int temp = 0;
        //用于保存读取的字节数
        int num = 0;
        try {
            //该构造方法的参数是一个字节数组，这个字节数组就是数据源
            bais = new ByteArrayInputStream(b);
            while ((temp = bais.read()) != -1) {
                sb.append((char) temp);
                num++;
            }
            System.out.println(sb);
            System.out.println("读取的字节数：" + num);
        } finally {
```

```
            try {
                if (bais != null) {
                    bais.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```