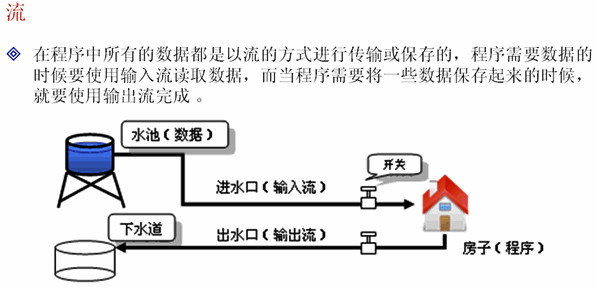
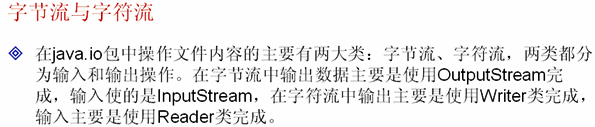
**IO流**

**1、流的概念**

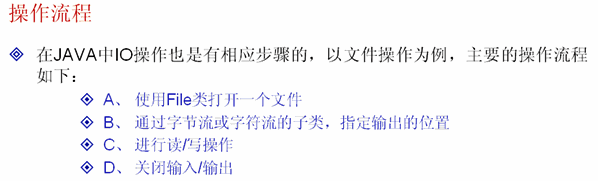


**2、字节流与字符流**



内容操作就四个类：OutputStream、InputStream、writer、Rader





使用 File 类操作的时候一定要有路径的问题，注意分隔符

实际上四个操作类都是抽象类

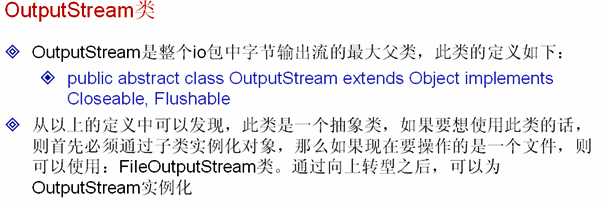
IO操作属于资源操作，对于资源操作，操作的最后必须关闭，否则就有可能出现未知错误。

**3.字节流**



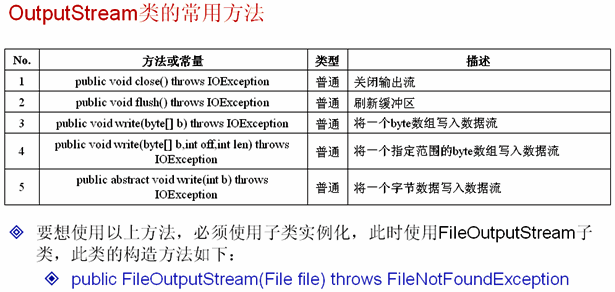
Byte 是字节，肯定使用字节流操作。所有的数基本都可以直接使用 byte 数组表示出来。

**字节输出流： OutputStream 类**



Clonseable： 表示可以关闭的操作，因为程序运行到最后肯定要关闭。

Fluashable： 表示刷新，清空内在中的数据



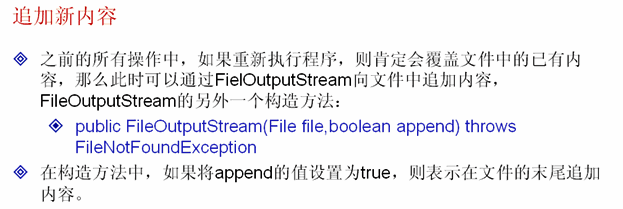
1. **import** java.io.File ;
2. **import** java.io.OutputStream ;
3. **import** java.io.FileOutputStream ;
4. **public** **class** OutputStreamDemo01{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. OutputStream out = **null** ;   // 准备好一个输出的对象
10. out = **new** FileOutputStream(f)  ;    // 通过对象多态性，进行实例化
11. // 第3步、进行写操作
12. String str = "Hello World!!!" ;     // 准备一个字符串
13. **byte** b[] = str.getBytes() ;         // 只能输出byte数组，所以将字符串变为byte数组
14. out.write(b) ;                      // 将内容输出，保存文件
15. // 第4步、关闭输出流
16. out.close() ;                       // 关闭输出流
17. }
18. };

在操作的时候，如果文件本身不存在，则会为用户自动创建新文件。

          在操作输出流的时候，也可以使用 write(int i) 的方法写出数据

1. **import** java.io.File ;
2. **import** java.io.OutputStream ;
3. **import** java.io.FileOutputStream ;
4. **public** **class** OutputStreamDemo02{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. OutputStream out = **null** ;   // 准备好一个输出的对象
10. out = **new** FileOutputStream(f)  ;    // 通过对象多态性，进行实例化
11. // 第3步、进行写操作
12. String str = "Hello World!!!" ;     // 准备一个字符串
13. **byte** b[] = str.getBytes() ;         // 只能输出byte数组，所以将字符串变为byte数组
14. **for**(**int** i=0;i<b.length;i++){     // 采用循环方式写入
15. out.write(b[i]) ;   // 每次只写入一个内容
16. }
17. // 第4步、关闭输出流
18. out.close() ;                       // 关闭输出流
19. }
20. };

以上的操作中在写入数据之后，文件之前的内容已经不存在了，因为在IO操作中默认的情况是将其进行覆盖的，那么如果现在要想执行追加的功能，则必须设置追加的操作，找到FileOutputStream 类；



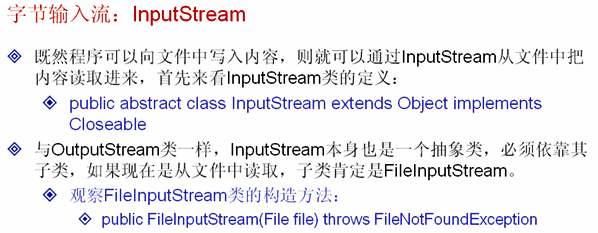
1. **import** java.io.File ;
2. **import** java.io.OutputStream ;
3. **import** java.io.FileOutputStream ;
4. **public** **class** OutputStreamDemo03{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. OutputStream out = **null** ;   // 准备好一个输出的对象
10. out = **new** FileOutputStream(f,**true**)  ;   // 此处表示在文件末尾追加内容
11. // 第3步、进行写操作
12. String str = "Hello World!!!" ;     // 准备一个字符串
13. **byte** b[] = str.getBytes() ;         // 只能输出byte数组，所以将字符串变为byte数组
14. **for**(**int** i=0;i<b.length;i++){     // 采用循环方式写入
15. out.write(b[i]) ;   // 每次只写入一个内容
16. }
17. // 第4步、关闭输出流
18. out.close() ;                       // 关闭输出流
19. }
20. };

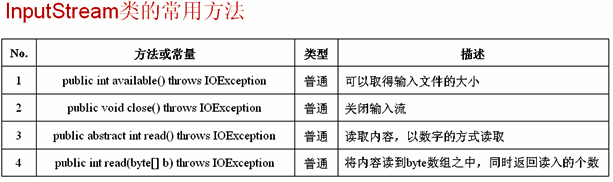
程序本身是可以追加内容了，但是没有换行，是直接在末尾追加的。

如果在文件操作中想换行的话，使用  “\r\n” 完成。（注意：在window系统中，换行用\r\n表示，在**[Linux](http://lib.csdn.net/base/linux)**中用\n ）

1. **import** java.io.File ;
2. **import** java.io.OutputStream ;
3. **import** java.io.FileOutputStream ;
4. **public** **class** OutputStreamDemo04{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. OutputStream out = **null** ;   // 准备好一个输出的对象
10. out = **new** FileOutputStream(f,**true**)  ;   // 此处表示在文件末尾追加内容
11. // 第3步、进行写操作
12. String str = "\r\nHello World!!!" ;     // 准备一个字符串
13. **byte** b[] = str.getBytes() ;         // 只能输出byte数组，所以将字符串变为byte数组
14. **for**(**int** i=0;i<b.length;i++){     // 采用循环方式写入
15. out.write(b[i]) ;   // 每次只写入一个内容
16. }
17. // 第4步、关闭输出流
18. out.close() ;                       // 关闭输出流
19. }
20. };

**字节输入流：InputStream**





1. **import** java.io.File ;
2. **import** java.io.InputStream ;
3. **import** java.io.FileInputStream ;
4. **public** **class** InputStreamDemo01{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. InputStream input = **null** ;  // 准备好一个输入的对象
10. input = **new** FileInputStream(f)  ;   // 通过对象多态性，进行实例化
11. // 第3步、进行读操作
12. **byte** b[] = **new** **byte**[1024] ;     // 所有的内容都读到此数组之中
13. input.read(b) ;     // 读取内容
14. // 第4步、关闭输出流
15. input.close() ;                     // 关闭输出流
16. System.out.println("内容为：" + **new** String(b)) ;    // 把byte数组变为字符串输出
17. }
18. };

此时，内容确实已经读取进来了，但是可以发现存在问题。

1. **import** java.io.File ;
2. **import** java.io.InputStream ;
3. **import** java.io.FileInputStream ;
4. **public** **class** InputStreamDemo02{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. InputStream input = **null** ;  // 准备好一个输入的对象
10. input = **new** FileInputStream(f)  ;   // 通过对象多态性，进行实例化
11. // 第3步、进行读操作
12. **byte** b[] = **new** **byte**[1024] ;     // 所有的内容都读到此数组之中
13. **int** len = input.read(b) ;       // 读取内容
14. // 第4步、关闭输出流
15. input.close() ;                     // 关闭输出流\
16. System.out.println("读入数据的长度：" + len) ;
17. System.out.println("内容为：" + **new** String(b,0,len)) ;  // 把byte数组变为字符串输出
18. }
19. };

 这个代码还存在问题，现在文件没有这么大，但是开辟了这么大的数组空间，这样肯定很浪费，能不能根据文件大小来开辟数组空间呢？

如果要想知道文件大小，直接使用File 类即可。

1. **import** java.io.File ;
2. **import** java.io.InputStream ;
3. **import** java.io.FileInputStream ;
4. **public** **class** InputStreamDemo03{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. InputStream input = **null** ;  // 准备好一个输入的对象
10. input = **new** FileInputStream(f)  ;   // 通过对象多态性，进行实例化
11. // 第3步、进行读操作
12. **byte** b[] = **new** **byte**[(**int**)f.length()] ;      // 数组大小由文件决定
13. **int** len = input.read(b) ;       // 读取内容
14. // 第4步、关闭输出流
15. input.close() ;                     // 关闭输出流\
16. System.out.println("读入数据的长度：" + len) ;
17. System.out.println("内容为：" + **new** String(b)) ;    // 把byte数组变为字符串输出
18. }
19. };

以上直接使用 byte 数组的方式完成的。

现在使用 public abstract int read() throws IOException 读取内容。

1. **import** java.io.File ;
2. **import** java.io.InputStream ;
3. **import** java.io.FileInputStream ;
4. **public** **class** InputStreamDemo04{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. InputStream input = **null** ;  // 准备好一个输入的对象
10. input = **new** FileInputStream(f)  ;   // 通过对象多态性，进行实例化
11. // 第3步、进行读操作
12. **byte** b[] = **new** **byte**[(**int**)f.length()] ;      // 数组大小由文件决定
13. **for**(**int** i=0;i<b.length;i++){
14. b[i] = (**byte**)input.read() ;     // 读取内容
15. }
16. // 第4步、关闭输出流
17. input.close() ;                     // 关闭输出流\
18. System.out.println("内容为：" + **new** String(b)) ;    // 把byte数组变为字符串输出
19. }
20. };

以上的操作，只适合于知道输入流大小的时候，如果现在不知道大小呢？

1. **import** java.io.File ;
2. **import** java.io.InputStream ;
3. **import** java.io.FileInputStream ;
4. **public** **class** InputStreamDemo05{
5. **public** **static** **void** main(String args[]) **throws** Exception{    // 异常抛出，不处理
6. // 第1步、使用File类找到一个文件
7. File f= **new** File("d:" + File.separator + "test.txt") ;  // 声明File对象
8. // 第2步、通过子类实例化父类对象
9. InputStream input = **null** ;  // 准备好一个输入的对象
10. input = **new** FileInputStream(f)  ;   // 通过对象多态性，进行实例化
11. // 第3步、进行读操作
12. **byte** b[] = **new** **byte**[1024] ;     // 数组大小由文件决定
13. **int** len = 0 ;
14. **int** temp = 0 ;          // 接收每一个读取进来的数据
15. **while**((temp=input.read())!=-1){
16. // 表示还有内容，文件没有读完
17. b[len] = (**byte**)temp ;
18. len++ ;
19. }
20. // 第4步、关闭输出流
21. input.close() ;                     // 关闭输出流\
22. System.out.println("内容为：" + **new** String(b,0,len)) ;  // 把byte数组变为字符串输出
23. }
24. };

当不知道读取内容有多在的时候，就只能以读取的数据是否为-1为读完的标志。