

知识点列表

编号	名称	描述	级别
1	java.io.File 的用途	理解 java 中的 File 类, 请查阅 API 帮助文档	*
2	java.io.File 常用 API 方法	掌握如何创建 java 中的 File 类, 以及常用的操作 File 的 API 方法	**
3	回调模式和 FileFilter	理解回调模式, 掌握 FileFilter 的运用方式, 练习并回顾匿名内部类	**
4	RandomAccessFile 类	重点理解在 Java 中如何写入字符、数字, 理解 RandomAccessFile 的功能, 初步了解编码问题	*
5	序列化与基本类型序列化	通过案例理解什么是序列化和反序列化, 对 RandomAccessFile 类基本方法进行练习	*
6	IO 流 (InputStream, OutputStream)	了解 InputStream 和 OutputStream 两个抽象类, 对操作 IO 流的 API 方法建立初步认识	*
7	FileInputStream	理解 FileInputStream 的使用, 重点练习并掌握工具类 IOUtils.java	***
8	FileOutputStream	理解如何向文件中写入	*
9	DataOutputStream	理解过滤流 DataOutputStream	**
10	DataInputStream	理解过滤流 DataOutputStream	**
11	BufferedInputStream && BufferedOutputStream	熟练掌握 BufferedInputStream && BufferedOutputStream 的使用, 较常用	**
12	文件复制实现与优化	掌握文件复制, 重点掌握优化的文件复制方法	**

注: "*"理解级别 "***"掌握级别 "****"应用级别

目录

1. Java 文件系统管理

1.1. File 类的用途 *

java.io.File 用于表示文件（目录），也就是说程序员可以通过 File 类在程序中操作硬盘上的文件和目录。

File 类只用于表示文件（目录）的信息（名称、大小等），不能对文件的内容进行访问。

1.2. java.io.File 基本 API **

File 代表文件系统中对文件/目录的管理操作（增删改查，CRUD）

常用 API 方法：

- | | |
|-----------------------------|-----------|
| ■ File(String) | 指定文件名的构造器 |
| ■ long length() | 文件的长度 |
| ■ long lastModified() | |
| ■ String getName() | |
| ■ String getPath() | |
| ■ boolean exists() | |
| ■ boolean dir.isFile() | |
| ■ boolean dir.isDirectory() | |
| ■ boolean mkdir() | |

- boolean mkdirs()
- boolean delete();
- boolean createNewFile() throw IOException
- File[] listFile()

【案例】File API 方法演示

```

1 package corejava.day08.ch04;
2 import java.io.File;
3
4 /**
5  * 任务：
6  * A 检查当前文件夹中是否包含目录 demo
7  * B 如果没有demo，就创建文件夹demo
8  * C 在demo 中 创建文件 test.txt
9  * D 显示demo 文件夹的内容。
10  * E 显示test.txt 的绝对路径名
11  * F 显示test.txt 的文件长度和创建时间
12  */
13
14 public class FileDemo {
15     public static void main(String[] args)
16         throws IOException{
17         File dir = new File(".");
18         //显示路径
19         System.out.println(dir.getCanonicalPath()); //pwd
20         //A
21         File demo = new File(dir, "demo"); //不是在磁盘上创建
22         if(! demo.exists()){
23             //B
24             demo.mkdir(); //mkdir demo
25         }
26         //C
27         File test = new File(demo, "test.txt");
28         if(! test.exists()){
29             //空文件，长度为0
30             test.createNewFile(); //touch
31         }
32         //D 显示目录内容
33         File[] files = demo.listFiles(); //ls demo
34         System.out.println(Arrays.toString(files));
35         //E
36         System.out.println(test.getCanonicalPath());
37     }
38 }

```

```

38      //F
39      System.out.println(test.length());
40      System.out.println(new Date(test.lastModified()));
41      //删除文件和目录
42      test.delete();//rm
43      demo.delete();
44  }
45 }

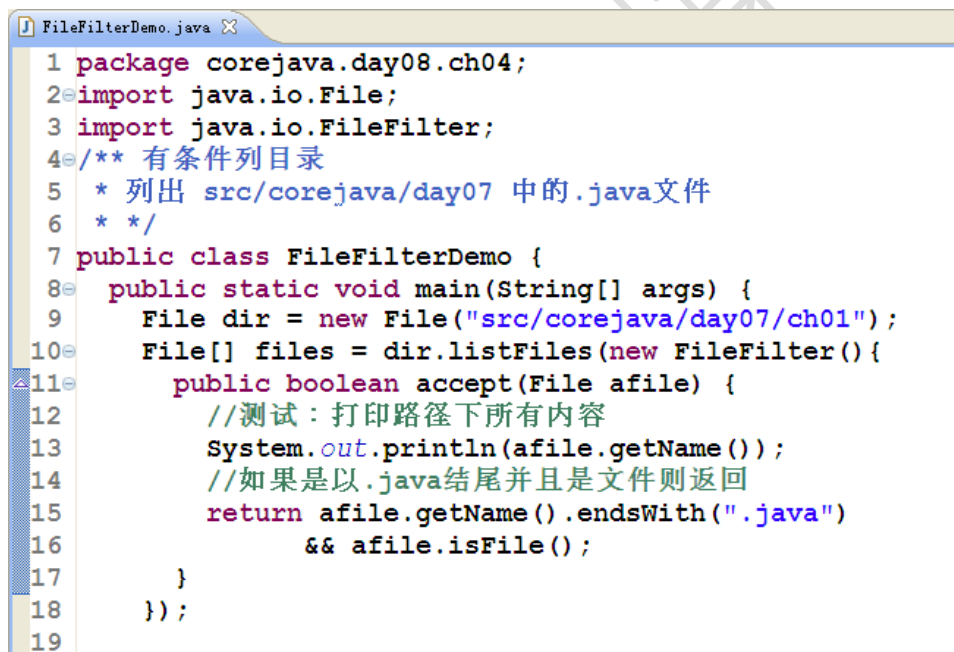
```

1.3. 回调模式和 FileFilter **

- FileFilter 类是对操作文件的过滤，相当于命令：`ls|grep patten`

✓ API 方法：`File[] listFile(FileFilter)`

【案例】列出指定目录下所有的.java 文件



```

FileFilterDemo.java X
1 package corejava.day08.ch04;
2 import java.io.File;
3 import java.io.FileFilter;
4 /** 有条件列目录
5  * 列出 src/corejava/day07 中的.java文件
6  * */
7 public class FileFilterDemo {
8     public static void main(String[] args) {
9         File dir = new File("src/corejava/day07/ch01");
10        File[] files = dir.listFiles(new FileFilter(){
11            public boolean accept(File afile) {
12                //测试：打印路径下所有内容
13                System.out.println(afile.getName());
14                //如果是以.java结尾并且是文件则返回
15                return afile.getName().endsWith(".java")
16                    && afile.isFile();
17            }
18        });
19    }

```

```

20    //2.1 for循环输出文件名
21    //for(int i=0; i<files.length; i++){
22    //    File file = files[i];
23    //    System.out.println(file.getName());
24    //}
25    //2.2 java5 提供的简化版的foreach迭代
26    for(File file:files){
27        System.out.println(file.getName());
28    }
29 }
30 }

```

注：

- ✓ **listFiles()方法**会将 dir 中每个文件交给 **accept()方法**检测，如果返回 true，就作为方法的返回结果元素
- ✓ **增强循环（for each 循环）**：JDK5 提供的简化版 for 循环
 - 实现原理基本相同，表现形式更简洁
- ✓ **回调模式**
 - **accept()方法**的调用属于回调模式

2. RandomAccessFile **

RandomAccessFile 类是 Java 提供的功能丰富的文件内容访问类，它提供了众多方法来访问文件内容，既可以读取文件内容，也可以向文件输出数据，RandomAccessFile 支持“随机访问”方式，可以访问文件的任意位置。

1) Java 文件模型

在硬盘上文件是 byte by byte 存储的，是数据的集合

2) 打开文件

有两种模式 "rw"（读写）、"r"（只读）

```
RandomAccessFile raf = new RandomAccessFile(file, "rw");
```

打开文件时候默认文件指针在开头 pointer=0

3) 写入方法

raf.write(int)可以将整数的“低八位”写入到文件中，同时指针自动移动到下一个位置，准备再次写入

- ✓ 注意：文件名的**扩展名**要明确指定，没有默认扩展名现象！

```
RandomAccessFile raf = new RandomAccessFile("Hello.java", "rw");
```

4) 读取文件

int b = raf.read() 从文件中读取一个 byte(8 位) 填充到 int 的低八位，高 24 位为 0，返回值

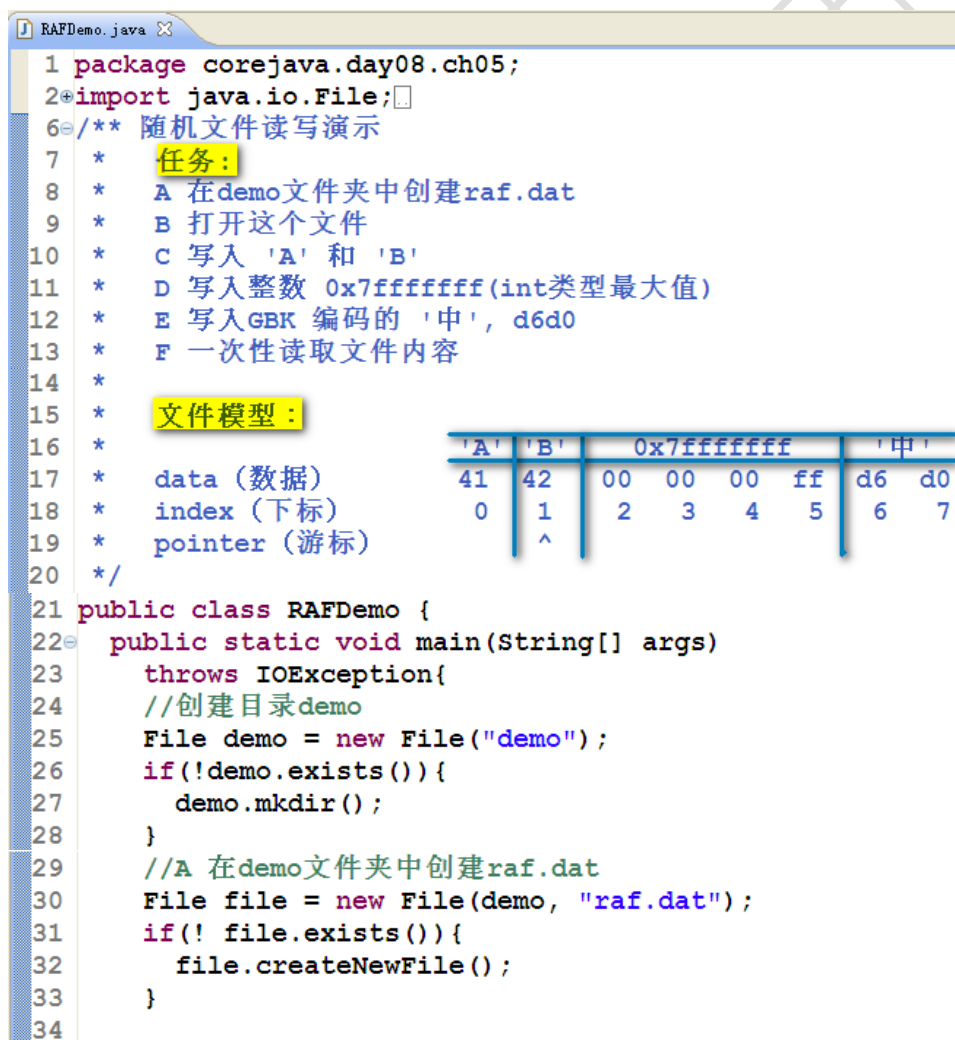
范围正数: 0~255, 如果返回-1 表示读取到了文件末尾! 每次读取后自动移动文件指针, 准备下次读取。

5) 文件读写完成以后一定关闭文件

Sun 官方说明, 如果不关闭, 可能遇到一些意想不到的错误, 根据具体操作平台不同会有不同。在使用过程中, 切忌文件读写完成后要关闭文件。

2.1. 写入方法

【案例 1】RandomAccessFile 演示_写入文件



```

1 package corejava.day08.ch05;
2 import java.io.*;
3
4 /** 随机文件读写演示
5  *
6  * 任务:
7  *   A 在demo文件夹中创建raf.dat
8  *   B 打开这个文件
9  *   C 写入 'A' 和 'B'
10  *   D 写入整数 0x7fffffff(int类型最大值)
11  *   E 写入GBK 编码的 '中', d6d0
12  *   F 一次性读取文件内容
13  *
14  *
15  * 文件模型:
16  *
17  *   data (数据)
18  *   index (下标)
19  *   pointer (游标)
20  */
21 public class RAFDemo {
22     public static void main(String[] args)
23         throws IOException{
24         //创建目录demo
25         File demo = new File("demo");
26         if(!demo.exists()){
27             demo.mkdir();
28         }
29         //A 在demo文件夹中创建raf.dat
30         File file = new File(demo, "raf.dat");
31         if(!file.exists()){
32             file.createNewFile();
33         }
34     }
35 }

```

'A'	'B'	0x7fffffff				'中'	
41	42	00	00	00	ff	d6	d0
0	1	2	3	4	5	6	7
	^						

```

35 //B 打开这个文件, 进行随机读写
36 RandomAccessFile raf =
37     new RandomAccessFile(file, "rw");
38 //输出默认的"游标"位置
39 System.out.println(raf.getFilePointer()); //0
40
41 //C 写入字符 'A' 和 'B'
42 raf.write('A');
43 System.out.println(raf.getFilePointer()); //1
44 raf.write('B');
45
46 //D 写入整数0x7fffffff
47 //D-1 写入int数据的底层写法
48 int i = 0x7fffffff;
49 raf.write(i>>>24); // i>>>24 00 00 00 7f
50 raf.write(i>>>16); // i>>>16 00 00 7f ff
51 raf.write(i>>>8); // i>>>8 00 7f ff ff
52 raf.write(i); // i 7f ff ff ff
53 //D-2 写入int数据的封装方法
54 raf.writeInt(i);
55
56 //E 写入GBK 编码的字符'中', GBK编码为d6d0
57 String s = "中"; //默认系统编码为4e2d
58 //E-1 得到"中"的gbk编码形式
59 byte[] gbk = s.getBytes("gbk"); //gbk = {d6, d0}
60 raf.write(gbk);
61 System.out.println(raf.length()); //8
62 System.out.println(raf.getFilePointer()); //8文件尾
63
64 //移动文件游标到 "头"
65 raf.seek(0);
66 //F 一次性读取全部内容到buf中
67 byte[] buf = new byte[(int)raf.length()];
68 //F-1 从文件中读取内容到buf数组, 尽可能填满
69 raf.read(buf);
70 //F-2 输出byte数组 (默认按10进制打印)
71 System.out.println(Arrays.toString(buf));
72 //F-3 输出16进制形式
73 for (byte b : buf) {
74     System.out.print(Integer.toHexString(b & 0xff) + " ");
75 }
76
77 //关闭
78 raf.close();
79 }
80 }

```

注:

✓ raf.write('A')的写入过程:

首先，字符 A 在内存中是 16 位无符号整数 0000 0000 0000 0041

其次，自动类型转换，转为 int 类型 0000 0000 0000 0000 0000 0000 0000 0041

最后，截取高 8 位，将低 8 位的数据写入“流”中 0000 0041

2.2. 读取文件

【案例 2】RandomAccessFile 演示_读取文件

```

1 package corejava.day08.ch05;
2 import java.io.IOException;
3
4 /** 文件读取操作
5  * 任务:
6  *     A 只读打开文件，移动到int数据位置
7  *     B 连续读取4个byte，拼接为int（反序列化）
8  *
9  * 文件模型:
10 *     data    :  41 42 7f ff ff ff d6 d0 ...
11 *     index   :   0  1  2  3  4  5  6  7  8
12 *     pointer:           ^
13 */
14 public class RAFReadDemo {
15     public static void main(String[] args)
16         throws IOException{
17         //A-1 只读打开文件
18         RandomAccessFile raf =
19             new RandomAccessFile("demo/raf.dat", "r");
20         int i = 0;
21         //A-2 移动到int位置
22         raf.seek(2);
23
24         //B-1.1 读取第1个byte
25         int b = raf.read(); //7f
26         System.out.println(raf.getFilePointer()); //3
27         //B-1.2 开始拼接为int
28         i = i | (b << 24);    // 7f 00 00 00
29         //B-2.1 读取第2个byte
30         b = raf.read();      // 00 00 00 ff
31         //B-2.2 拼接
32         i = i | (b << 16);    // 7f ff 00 00
    }
}

```



```

33 //B-3.1 读取第3个byte
34 b = raf.read(); // 00 00 00 ff
35 //B-3.2 拼接
36 i = i | (b << 8); // 7f ff ff 00
37 //B-4.1 读取第4个byte
38 b = raf.read(); // 00 00 00 ff
39 //B-4.2 拼接
40 i = i | b; // 7f ff ff ff
41 System.out.println(Integer.toHexString(i));
42
43 /*Java API提供的封装方法*/
44
45 //A 移动到int位置
46 raf.seek(2);
47 //B 连续读取4个byte, 拼接为int (反序列化)
48 i = raf.readInt();
49 System.out.println(Integer.toHexString(i));
50
51 raf.close();
52 }
53 }

```

3. 序列化与基本类型序列化 *

- 1) 将类型 int 转换为 4 byte, 或将其它数据类型 (如 long -> 8 byte) 的过程, 即将数据转换为 n 个 byte 序列叫**序列化** (数据 -> n byte)
如: 0x7fffffff -> [7f , ff , ff , ff]
- 2) **反序列化**, 将 n byte 转换为一个数据的过程 (n byte -> 数据)
如: [7f , ff , ff , ff] -> 0x7fffffff
- 3) RandomAccessFile 提供基本类型的读写方法, 可以将基本类型数据序列化到文件或者将文件内容反序列化为数据

【案例】序列化与反序列化

```

Demo.java
1 package corejava.day08.ch05;
2 import java.io.*;
5 /** 序列化和反序列化 */
6 public class Demo {
7     public static void main(String[] args)
8         throws IOException{

```

```

9      //创建目录demo
10     File demo = new File("demo");
11     if(!demo.exists()){ demo.mkdir(); }
12     //在demo文件夹中创建raf.dat
13     File file = new File(demo, "raf.dat");
14     if(! file.exists()){ file.createNewFile(); }
15     //打开这个文件，进行随机读写
16     RandomAccessFile raf =
17         new RandomAccessFile(file, "rw");
18
19     /*序列化*/
20     int i = 0x7fffffff;
21     raf.write(i>>>24); // i>>>24 00 00 00 7f
22     raf.write(i>>>16); // i>>>16 00 00 7f ff
23     raf.write(i>>>8); // i>>>8 00 7f ff ff
24     raf.write(i); // i 7f ff ff ff
25     System.out.println(raf.getFilePointer());
26
27     /*反序列化*/
28     raf.seek(0);
29     //1.1 读取第1个byte
30     int b = raf.read(); //7f
31     //1.2 开始拼接为int
32     i = i | (b<<24); // 7f 00 00 00
33     //2.1 读取第2个byte
34     b = raf.read(); // 00 00 00 ff
35     //2.2 拼接
36     i = i | (b<<16); // 7f ff 00 00
37     //3.1 读取第3个byte
38     b = raf.read(); // 00 00 00 ff
39     //3.2 拼接
40     i = i | (b<<8); // 7f ff ff 00
41     //4.1 读取第4个byte
42     b = raf.read(); // 00 00 00 ff
43     //4.2 拼接
44     i = i | b; // 7f ff ff ff
45     System.out.println(Integer.toHexString(i));
46 }
47 }

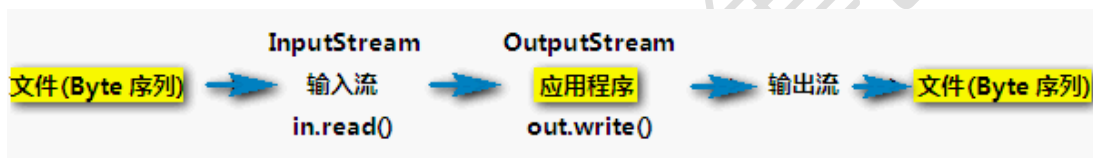
```

4. IO 流 (InputStream, OutputStream) *

Java 中的 IO 流是实现输入/输出的基础。

- 1) InputStream、OutputStream 都是抽象类
 - InputStream 抽象了应用程序读取数据的方式
 - OutputStream 抽象了应用程序写出数据的方式

- 2) **EOF** = End of File = -1
- 3) 输入流基本方法:
 - `int b = in.read()` 读取一个 byte 无符号填充到 int 低八位,-1 是 EOF
 - `in.read(byte[] buf)` 读取数据填充到 buf 中
 - `in.read(byte[] buf, int start, int size)` 读取数据填充到 buf 中
 - `in.skip(long n)`
 - `in.close();`
- 4) 输出流的基本方法:
 - `out.write(int b)` 写出一个 byte 到流 b 的低八位写出
 - `out.write(byte[] buf)` 将缓冲区 buf 都写入到流
 - `out.write(byte[] buf, int start, int size)` 将 buf 的一部分写到流中
 - `out.flush()` 清理缓冲
 - `out.close();`



注：

- ✓ 输入流、输出流是相对应用程序而言的，应用程序是参照物。

5. FileInputStream **

文件输入流 `FileInputStream` 继承了 `InputStream`，`FileInputStream` 具体实现了在文件上读取数据。

【案例 1】IO 工具类 IOUtils.java

案例描述

`printHex()`方法实现功能：读取文件并且按照 HEX 输出，每 10 byte 为一行。

- ✓ 版本 01
 - 自定义 IO 工具类，`printHex (String fileName)` 方法实现功能：读取指定文件内容，按照 16 进制输出到控制台
- ✓ 版本 02
 - `printHex (String fileName)` 方法中添加业务逻辑：单位数前边补 0，如 8->08
 - 定义 `printHex(String fileName)` 的重载方法：`printHex(File file)` 和 `printHex(InputStream in)`

✓ 版本 03

- printHex (String fileName) 方法中添加业务逻辑：每输出 10byte，则换行

执行结果

```

1c c7 56 4d a3 30 78 54 26 31
83 b5 73 a3 4c 77 ef 23 1d 44
51 60 dd 87 69 83 b7 c6 61 a1
7f cd e9 b8 32 b0 39 6e 3d 8f
46 75 5b be a1 03 89 93 e5 47
06 ba b8 28 0e 29 67 b5 87 dc
06 69 0a 35 75 4c 0c f4 d3 25
d7 20 c2 66 44 23 49 6c 4e d2
    
```

注：

- ✓ 查看文件不要太大，否则读取速度很慢

【版本 01】

● IOUtils.java

```

1 package corejava.day09.ch01;
2 import java.io.FileInputStream;
3
4 /** IO 工具类 */
5 public class IOUtils_01 {
6
7     /**
8      * 读取文件并且按照16进制输出，每10 byte为一行
9      * @param fileName 输入文件名
10      * @throws IOException 文件读取异常
11      */
12     public static void printHex(String fileName)
13         throws IOException{
    
```

```

14 //1. 创建文件输入流对象
15 FileInputStream in =
16     new FileInputStream(fileName);
17 int b;
18 //2. 循环读取byte
19 while((b=in.read())!=-1){//每次读取一个byte,并检查EOF
20     if(b<=0xf){//单位数前面补0, 如: 8 -> 08
21         System.out.print("0");
22     }
23     System.out.print(Integer.toHexString(b) + " ");
24 }
25 in.close();
26 }
27 }

```

● Demo.java 测试类

```

1 package corejava.day09.ch01;
2 import java.io.IOException;
3 /** 测试类 */
4 public class Demo {
5     public static void main(String[] args)
6         throws IOException{
7         IOUtils_01.printHex("demo/raf.dat");
8     }
9 }
10

```

Console

<terminated> Demo (4) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 9, 2011 10:58:)

41 42 7f ff ff ff 7f ff ff ff d6 d0

'A' 'B' 0x7fffffff 0x7fffffff '中'

注：

- ✓ 参数 fileName 可以是相对路径，也可以是绝对路径，如/home/soft01/abc.txt

【版本 02】

● IOUtils.java

```

1 package corejava.day09.ch02;
2 import java.io.File;
3
4 /** IO 工具类 */
5
6 public class IOUtils_02 {
7     /**
8      * 读取文件并且按照16进制输出, 每10 byte为一行
9      * @param fileName 输入文件名
10      * @throws IOException 文件读取异常
11      */
12
13     public static void printHex(String fileName)
14         throws IOException{
15         //1. 创建文件输入流对象
16         FileInputStream in =
17             new FileInputStream(fileName);
18         int b;
19         //2. 循环读取byte
20         while((b=in.read())!=-1){ //每次读取一个byte, 并检查EOF
21             if(b<=0xf){ //单位数前面补0, 如: 8 -> 08
22                 System.out.print("0");
23             }
24             System.out.print(Integer.toHexString(b) + " ");
25         }
26         in.close();
27     }
28     /**printHex(String fileName)的重载方法 */
29     public static void printHex(File file)
30         throws IOException{
31         //1. 创建文件输入流对象
32         FileInputStream in =
33             new FileInputStream(file);
34         int b;
35         //2. 循环读取byte
36         while((b=in.read())!=-1){
37             if(b<=0xf){
38                 System.out.print("0");
39             }
40             System.out.print(Integer.toHexString(b) + " ");
41         }
42         in.close();
43     }
44
45     /**printHex(String fileName)的重载方法 */
46     public static void printHex(InputStream in)
47         throws IOException{
48         int b;
49         while((b=in.read())!=-1){
50             if(b<=0xf){
51                 System.out.print("0");

```

```

52     }
53     System.out.print(Integer.toHexString(b) + " ");
54 }
55 in.close();
56 }
57 }

```

注：

- ✓ 重载的方法之间有很多重复代码，可以更精简

【版本 03】

● IOUtils.java 代码片段

```

13 public static void printHex(String fileName)
14     throws IOException{
15     //1. 创建文件输入流对象
16     FileInputStream in =
17         new FileInputStream(fileName);
18     int b;
19     int i = 1;
20     if(b<=0xf){//单位数前面补0，如：8 -> 08
21         System.out.print("0");
22     }
23     System.out.print(Integer.toHexString(b) + " ");
24     if(i++%10==0){//每10行换行
25         System.out.println();
26     }
27 }
28 System.out.println();
29 in.close();
30 }
31 }

```

【案例 2】IO 工具类 IOUtils.java

案例描述

IO 工具类中增加读取文件内容的方法 **read(String filename)**及重载方法 **read(File file)**，该方法适用于读取小文件。

参考代码

● IOUtils.java

```

IOUtils.java X Demo.java
1 package corejava.day09.ch02;
2 import java.io.File;
5 /** IO 工具类 */
6 public class IOUtils {
7     /**
8      * 读取文件内容到byte数组中, 适用于小文件
9      */
10    public static byte[] read(String filename)
11        throws IOException{
12        File file = new File(filename);
13        //1. 按照文件长度创建byte数组
14        byte[] buf = new byte[(int)file.length()];
15        //2. 打开文件
16        FileInputStream in = new FileInputStream(file);
17        //3. 读取文件, read方法 尽可能多的读取流中数据, 填充到buf
18        //返回值size是读取的数量
19        int size = in.read(buf); //一次读取全部!
20        in.close();
21        return buf;
22    }
23
24    /** 重载方法 */
25    public static byte[] read(File file) throws IOException{
26        //1. 按照文件长度创建byte数组
27        byte[] buf = new byte[(int)file.length()];
28        //2. 打开文件
29        FileInputStream in = new FileInputStream(file);
30        //3. 读取文件, read方法 尽可能多的读取流中数据, 填充到buf
31        //返回值size是读取的数量
32        int size = in.read(buf); //一次读取全部!
33        in.close();
34        return buf;
35    }
36 }

```

● Demo.java 测试类

```

Demo.java X
1 package corejava.day09.ch02;
2 import java.io.IOException;
3 import java.util.Arrays;
4 /** 测试类 */
5 public class Demo {
6     public static void main(String[] args)
7         throws IOException{
8         byte[] b = IOUtils.read("demo/raf.dat");
9         System.out.println(Arrays.toString(b));

```



```
10 }  
11 }  
12 |
```

Console

<terminated> Demo (5) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 9, 2011 2:04:20 PM)
[65, 66, 127, -1, -1, -1, 127, -1, -1, -1, -42, -48]

6. FileOutputStream *

文件输出流 `FileOutputStream` 继承了 `OutputStream`，`FileOutputStream` 实现了向文件中写出 `byte` 数据的方法。

【案例】FileOutputStream 演示

案例描述

✓ 版本 01

完成任务

A 在 demo 文件夹中创建 `out.dat`

B 打开这个文件

C 写入 'A' 和 'B'

D 写入整数 255 占用 4 个 `byte`

E 写入 GBK 编码的 '中', `d6d0`

✓ 版本 02

演示 `out.wirte(byte[] b, int off, int len)` 方法

注意，`out.flush()` 方法表示清理缓存，在写代码中尽量加上，能保障可靠写

【版本 01】

```

FileOutDemo.java
1 package corejava.day09.ch03;
2 import java.io.FileOutputStream;
3 import java.io.IOException;
4 /** 文件输出流演示
5  * 任务:
6  * A 在demo文件夹中创建out.dat
7  * B 打开这个文件
8  * C 写入 'A' 和 'B'
9  * D 写入整数 -3 (占用4个byte)
10 * E 写入GBK 编码的"中国", d6 d0 b9 fa
11 */
12 public class FileOutDemo {
13     public static void main(String[] args)
14         throws IOException{
15         //A FileOutputStream有文件则覆盖, 无则创建
16         //B
17         FileOutputStream out =
18             new FileOutputStream("demo/out.dat");
19         //C
20         out.write('A'); // 写出'A'的低八位
21         out.write('B'); // 写出'B'的低八位
22         //D
23         int a = -3; // 0xffffffff
24         out.write(a>>>24); // 写出a的 ff
25         out.write(a>>>16); // 写出a的 ff
26         out.write(a>>>8); // 写出a的 ff
27         out.write(a); // 写出a的 fd
28         //E
29         //E-1 获得"中国"的GBK编码
30         byte[] gbk = "中国".getBytes("gbk");
31         //E-2 写出byte[]数组全部内容
32         out.write(gbk);
33
34         out.close();
35         IOUtils.printHex("demo/out.dat");
36     }
37 }

```

```

Console
<terminated> FileOutDemo [Java Application] C:\Program Files\Java\jdk1.8.0_06\bin\javaw.exe (Nov 9, 2011 3:47
41 42 ff ff ff fd d6 d0 b9 fa

```

注：

- ✓ 文件输出流 (`FileOutputStream`) 的构造器，如果没有文件，会自动的创建文件！
- ✓ 输出时默认是覆盖这个文件内容，如果需要追加内容，需要使用新的构造器：
`boolean append = true;`
`new FileOutputStream(file, append);`

【版本 02】

```

FileOutDemo02.java
1 package corejava.day09.ch03;
2 import java.io.FileOutputStream;
3
4 /** 文件输出流演示
5  * out.write(byte[] b) 方法的重载方法
6  * out.write(byte[] b, int off, int len)
7  * 参数: off 起始位置
8  * 参数: len 长度
9  */
10 public class FileOutDemo02 {
11     public static void main(String[] args)
12         throws IOException{
13         FileOutputStream out =
14             new FileOutputStream("demo/out.dat");
15
16         byte[] gbk = "中国".getBytes("gbk");
17         //写出byte[]缓冲区中的部分内容(前2个)
18         out.write(gbk, 0, 2);
19         out.flush(); //清理缓冲区(尽可能写)
20         out.close(); //先flush(), 然后关闭文件
21         IOUtils.printHex("demo/out.dat");
22     }
23 }

```

Console

```

<terminated> FileOutDemo02 [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 9, 2011 3:4
d6 d0

```

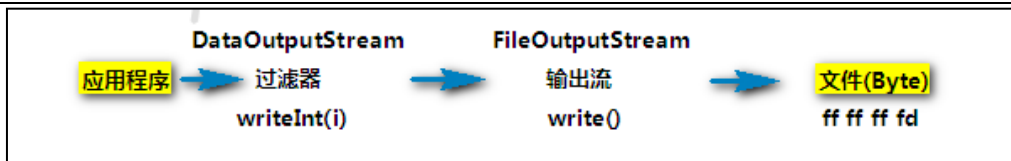
7. DataOutputStream *

DataOutputStream 和 DataInputStream 是对“流”功能的扩展，可以更方便的读取 int、long、字符等类型数据。

DataOutputStream 对基本的输出流功能扩展，提供了基本数据类型的输出方法，也就是基本类型是序列化方法：

- ✓ writeInt()
- ✓ writeDouble()
- ✓ writeUTF()

DataOutputStream 可以被理解为“过滤器”



【案例】DataOutputStream 演示

```

DOSDemo.java
1 package corejava.day09.ch04;
2 import java.io.DataOutputStream;
3 /**
4  * DataOutputStream 演示
5  * 任务: 输出 int 5
6  *       输出 -5
7  *       输出 long -5
8  *       ...
9  */
10
11 public class DOSDemo {
12     public static void main(String[] args)
13         throws IOException{
14         String file = "demo/dos.dat";
15         FileOutputStream fos = new FileOutputStream(file);
16         DataOutputStream out = new DataOutputStream(fos);
17         //DataOutputStream 扩展了基本流fos的功能
18
19         out.writeInt(5);           //00 00 00 05
20         out.writeInt(-5);          //ff ff ff fb
21         out.writeLong(-5L);         //ff ff ff ff ff ff ff fb
22         out.writeDouble(5.0);       //40 14 00 00 00 00 00 00
23         //采用UTF-8 编码写出
24         out.writeUTF("中国");       //e4 b8 ad e5 9b bd
25         //采用 UTF-16BE 编码写出
26         out.writeChars("中国");     //4e 2d 56 fd
27
28         out.close();
29         IOUtils.printHex(file);
30     }
31 }
32
Console
<terminated> DOSDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 9, 2011 4:08:24)
00 00 00 05 ff ff ff fb ff ff
ff ff ff ff ff fb 40 14 00 00
00 00 00 00 00 06 e4 b8 ad e5
9b bd 4e 2d 56 fd
  
```

注：

- ✓ out.writeDouble()的输出结果 40 14 00 00 00 00 00 00 是浮点数规则，此处不用过多关注
- ✓ out.writeUTF()方法：采用 UTF-8 编码
 - 规则是 **3 个 byte** 表示 1 个字符，如 ‘中’ 为 e4 b8 ad ； ‘国’ 为 e5 9b bd
- ✓ out.writeChars()方法：采用 UTF-16BE 编码
- ✓ 00 06 e4 b8 ad 中 e4 b8 ad 表示 ‘中’，**00 06 是标识符**，此处不用过多关注

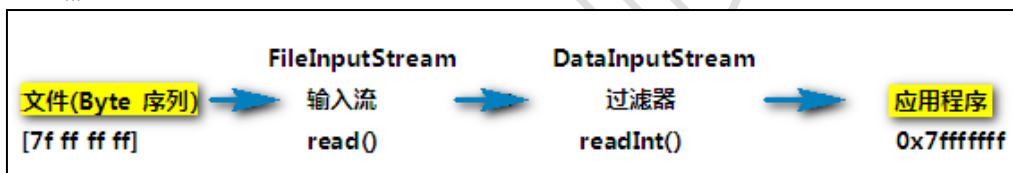
8. DataInputStream *

DataInputStream 是对基本输入流（InputStream）功能的扩展，它提供基本类型的输入方法，就是基本类型的反序列化。

DataInputStream 是**过滤器**，只是功能扩展，不能直接读取文件。

DataInputStream 提供的方法有

- ✓ readInt()
- ✓ readDouble()
- ✓ ...



【案例 1】DataInputStream 演示

```

1 DISDemo.java X
2
3 4 public class DISDemo {
4     public static void main(String[] args)
5         throws IOException{
6         String file = "demo/dos.dat";
7         IOUtils.printHex(file);
8         FileInputStream fis = new FileInputStream(file);
9         DataInputStream in = new DataInputStream(fis);
10
11
12         int i = in.readInt();
13         System.out.println(i); // 5 = 00 00 00 05
14         i = in.readInt();
15         System.out.println(i); // -5 = ff ff ff fb
16         long l = in.readLong();
17         System.out.println(l); // -5 = ff ff ff ff ff ff ff fb
18         double d = in.readDouble();
19         System.out.println(d); // 5.0 = 40 14 00 00 00 00 00 00
    
```

```

20     String s = in.readUTF();
21     System.out.println(s); //中国 = e4 b8 ad e5 9b bd
22
23     in.close();
24 }
25 }

```

【案例 2】将洗好的一副扑克牌写入文件，再读出来

- Card.java

(略) 请参考 day04 代码

- IOUtils.java

(略)

- CardIODemo.java

```

CardsIODemo.java
1 package corejava.day09.ch05;
2 import java.io.DataInputStream;
11 /** 读写一副扑克牌 */
12 public class CardsIODemo {
13     public static void main(String[] args)
14         throws IOException{
15         //构造一副扑克牌
16         List<Card> cards = new ArrayList<Card>();
17         for(int rank=Card.THREE;rank<=Card.DEUCE;rank++){
18             cards.add(new Card(Card.DIAMOND, rank));
19             cards.add(new Card(Card.CLUB, rank));
20             cards.add(new Card(Card.HEART, rank));
21             cards.add(new Card(Card.SPADE, rank));
22         }
23         cards.add(new Card(Card.JOKER, Card.BLACK));
24         cards.add(new Card(Card.JOKER, Card.COLOR));
25         //洗牌
26         Collections.shuffle(cards);
27
28         //1. 写出扑克牌集合到文件 cards.dat
29         String file = "demo/cards.dat";
30         //1.1 打开文件流
31         FileOutputStream fos = new FileOutputStream(file);

```

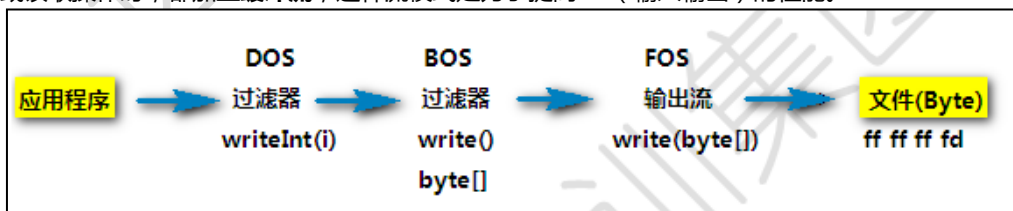
```

32 //1.2 用过滤流进行过滤
33 DataOutputStream out = new DataOutputStream(fos);
34 for(Card c: cards){
35     out.writeInt(c.getSuit()); //花色
36     out.writeInt(c.getRank()); //点数
37 }
38 out.close();
39 IOUtils.printHex(file); //查看文件内容
40 System.out.println(cards);
41
42
43 //2. 读取cards.dat 到 others 集合
44 List<Card> others = new ArrayList<Card>();
45 //2.1 打开文件流
46 FileInputStream fis = new FileInputStream(file);
47 //2.2 用过滤流进行过滤
48 DataInputStream in = new DataInputStream(fis);
49 try{
50     while(true){
51         int suit = in.readInt();
52         int rank = in.readInt();
53         others.add(new Card(suit, rank));
54     }
55 }catch(EOFException e){ //应该处理的、可以预知的异常
56     System.out.println("读完了!");
57 }
58 System.out.println(others);
59 }
60 }

```

9. BufferedInputStream && BufferedOutputStream **

BufferedInputStream && BufferedOutputStream 为 IO 操作提供了缓冲区，一般打开文件进行写入或读取操作时，都加上缓冲流，这种流模式是为了提高 IO（输入输出）的性能。



如图所示，我们想从应用程序中把数据放入文件，相当于将一缸水倒入另一个缸中：

- ✓ 仅使用 FOS 的 write()方法，相当于一滴水一滴水的“转移”
- ✓ 使用 DOS 的 writeXxx()方法方便些，相当于一瓢一瓢的“转移”
- ✓ 使用 BOS 的 writeXxx()方法更方便，相当于从 DOS 一瓢一瓢放入桶(BOS)中，再从桶(BOS)

中倒入另一个缸，性能提高了

【案例】将洗好的一副扑克牌写入文件，再读出来

```

14 /** 读写一副扑克牌 */
15 public class CardsIODemo {
16     public static void main(String[] args)
17         throws IOException{
18         //构造一副扑克牌
19         List<Card> cards = new ArrayList<Card>();
20         for(int rank=Card.THREE;rank<=Card.DEUCE; rank++){
21             cards.add(new Card(Card.DIAMOND, rank));
22             cards.add(new Card(Card.CLUB, rank));
23             cards.add(new Card(Card.HEART, rank));
24             cards.add(new Card(Card.SPADE, rank));
25         }
26         cards.add(new Card(Card.JOKER, Card.BLACK));
27         cards.add(new Card(Card.JOKER, Card.COLOR));
28         //洗牌
29         Collections.shuffle(cards);
30
31         //1. 写出扑克牌集合到文件 cards.dat
32         String file = "demo/cards.dat";
33         //1.1 打开文件流
34         FileOutputStream fos = new FileOutputStream(file);
35         //1.2 放入缓冲流
36         BufferedOutputStream bos =
37             new BufferedOutputStream(fos);
38         //1.3 用过滤流进行过滤
39         DataOutputStream out = new DataOutputStream(bos);
40         for(Card c: cards){
41             out.writeInt(c.getSuit()); //花色
42             out.writeInt(c.getRank()); //点数
43         }
44         out.close();
45         IOUtils.printHex(file); //查看文件内容
46         System.out.println(cards);
47
48         //2. 读取cards.dat 到 others 集合
49         List<Card> others = new ArrayList<Card>();
50         //2.1 打开文件流
51         FileInputStream fis = new FileInputStream(file);
52         //2.2 放入缓冲流
53         BufferedInputStream bis =
54             new BufferedInputStream(fis);

```



```

56 //2.3 用过滤流进行过滤
57 DataInputStream in = new DataInputStream(bis);
58 try{
59     while(true){
60         int suit = in.readInt();
61         int rank = in.readInt();
62         others.add(new Card(suit, rank));
63     }
64 }catch(EOFException e){//应该处理的、可以预知的异常
65     System.out.println("读完了!");
66 }
67 System.out.println(others);
68 }
69 }

```

Console X

<terminated> CardsIODemo (1) [Java Application] C:\Program Fil... 0.06\bin\javaw.exe (Nov 9, 2011 5:42:...

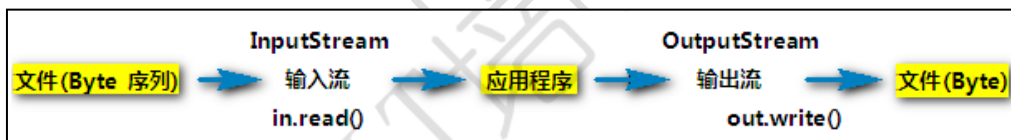
```

00 00 00 03 00 00 00 08 00 00 00 00 00 02
00 00 00 01 00 00 00 09 00 00 00 00 00 05
00 00 00 01 00 00 00 05 00 00 00 00 00 0b
00 00 00 03 00 00 00 07 00 00 00 01 00 04

```

黑桃J

10. 文件复制实现与优化 **



【案例】工具类 IOUtils.java 增加文件拷贝功能

案例描述

- ✓ 版本 01
效率低，实现小文件的拷贝功能，拷贝大文件时会非常慢
- ✓ 版本 02
效率稍高，可拷贝大文件，拷贝文件时使用了“缓冲流”
- ✓ 版本 03（常用写法）
效率最高，可拷贝大文件，使用了 byte[] 数组作为一个“缓冲区”

【版本 01】

- IOUtils.java

```

1 package corejava.day09.ch07;
2 import java.io.FileInputStream;
7 /** IO 工具类 */
8 public class IOUtils01 {
9
10     /**
11      * 复制文件，从输入流读取写出到输出流
12      * @param src 源文件名
13      * @param dest 目标文件名
14      */
15     public static void copy(String src, String dest)
16         throws IOException{
17         InputStream in = new FileInputStream(src);
18         OutputStream out = new FileOutputStream(dest);
19         int b;
20         while((b=in.read())!=-1){
21             out.write(b);
22         }
23         in.close();
24         out.close();
25     }

```

● CopyDemo.java 测试类

```

1 package corejava.day09.ch07;
2 import java.io.IOException;
3 /** 复制文件 */
4 public class CopyDemo {
5     public static void main(String[] args) {
6         try {
7             long start = System.currentTimeMillis();
8             IOUtils01.copy("d:/uninstall.sh", "d:/bak.sh");
9             long end = System.currentTimeMillis();
10            System.out.println("复制了一遍!" + (end-start));
11            //IOUtils.printHex("d:/uninstall.sh");
12            //IOUtils.printHex("d:/bak.sh");
13        } catch (IOException e) {
14            System.out.println(
15                "文件复制失败:" + e.getMessage());
16        }
17    }
18 }

```

测试文件大小 1,646KB

Console [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 10, 2011 10:13)
复制了一遍! 27797

【版本 02】

● IOUtils.java

```

1 package corejava.day09.ch07;
2 import java.io.BufferedReader;
9 /** IO 工具类 */
10 public class IOUtils02 {
11     /**
12      * 复制文件，从输入流读取写出到输出流
13      * @param src 源文件名
14      * @param dest 目标文件名
15      */
16     public static void copy(String src, String dest)
17         throws IOException{
18         InputStream in = new BufferedInputStream(
19             new FileInputStream(src));
20         OutputStream out = new BufferedOutputStream(
21             new FileOutputStream(dest));
22         int b;
23         while((b=in.read())!=-1){
24             out.write(b);
25         }
26         in.close();
27         out.close();
28     }

```

● CopyDemo.java 测试类

```

1 package corejava.day09.ch07;
2 import java.io.IOException;
3 /** 复制文件 */
4 public class CopyDemo {
5     public static void main(String[] args) {
6         try {
7             long start = System.currentTimeMillis();
8             IOUtils02.copy("d:/uninstall.sh", "d:/bak.sh");
9             long end = System.currentTimeMillis();
10            System.out.println("复制了一遍!"+(end-start));
11            //IOUtils.printHex("d:/uninstall.sh");
12            //IOUtils.printHex("d:/bak.sh");

```

```

13     } catch (IOException e) {
14         System.out.println(
15             "文件复制失败:" + e.getMessage());
16     }
17 }
18 }
19

```

Console

<terminated> CopyDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 10, 2011 10:41)
复制了一遍! 172

【版本 03】

- IOUtils.java

```

1 package corejava.day09.ch07;
2 import java.io.FileInputStream;
3 /** IO 工具类 */
4 public class IOUtils {
5
6     /**
7      * 复制文件，从输入流读取写出到输出流
8      * @param src 源文件名
9      * @param dest 目标文件名
10     */
11     public static void copy(String src, String dest)
12         throws IOException{
13         InputStream in = new FileInputStream(src);
14         OutputStream out = new FileOutputStream(dest);
15         byte[] buf = new byte[1024*512]; //512K
16         int count;
17         while((count=in.read(buf))!=-1){
18             //System.out.println(count);
19             out.write(buf, 0, count);
20         }
21         in.close();
22         out.close();
23     }
24 }

```

注：

- ✓ byte[] buf = new byte[1024*512]和 byte[] buf = new byte[524288]效率相同
因为在 1024*512 运算在编译期执行

- CopyDemo.java 测试类

```
CopyDemo.java X
1 package corejava.day09.ch07;
2 import java.io.IOException;
3 /** 复制文件 */
4 public class CopyDemo {
5     public static void main(String[] args) {
6         try {
7             long start = System.currentTimeMillis();
8             IOUtils.copy("d:/uninstall.sh", "d:/bak.sh");
9             long end = System.currentTimeMillis();
10            System.out.println("复制了一遍!" + (end - start));
11            //IOUtils.printHex("d:/uninstall.sh");
12            //IOUtils.printHex("d:/bak.sh");
13        } catch (IOException e) {
14            System.out.println(
15                "文件复制失败:" + e.getMessage());
16        }
17    }
18 }

Console X
<terminated> CopyDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Nov 10, 2011 10:23)
复制了一遍! 31
```