# day16

## File类

### 1.1File类概述

a : File类,来自于java.io包, 表示一个文件或者文件夹的路径(就表示一个文件或者文件夹)

b : 路径, 表示文件或者文件夹存在位置的字符串, 举例 : “D:\\abc.txt”

1. 绝对路径 : 表示唯一的, 带有根路径的, 没有任何歧义的路径存在, 在windows操作系统中, 绝对路径是带有盘符的路径

D:\\1012系统班

1. 相对路径 : 表示路径存在是相对而言, 不是唯一的, 有一些歧义的

a\\123.txt

c : 在eclipse中, 如果给出的文件是一个相对路径, 那么文件的绝对路径就会拼接当前正在操作的项目工程根目录

### 1.2 File类中的构造方法

1. File(String path) : 将path所表示的文件或者文件夹路径封装到File类型中
2. File(String fu, String zi) : 将fu+zi拼接成一个完整路径, 路径封装到File类型中
3. File(File fu, String zi) : 将File类型的fu + zi拼接成一个完整路径, 路径封装到File类型中

代码

|  |
| --- |
| package com.ujiuye.file;  import java.io.File;  public class Demo01\_FileConstructor {  public static void main(String[] args) {  String s = "D:\\1012系统班";  File f = new File(s);  System.out.println(f);// D:\1012系统班    File f1 = new File(s,"day01");  System.out.println(f1);// D:\1012系统班\day01    File f2 = new File(f1, "笔记");  System.out.println(f2);// D:\1012系统班\day01\笔记  }  } |

### 1.3 File类新增功能

1. createNewFile() : 当File表示一个文件, 并且文件路径存在, 文件不存在的时候, 此方法新建出一个空的文件, 返回值类型boolean类型, 创建文件成功返回true

2. mkdir() : 每次只能创建出File类所表示的最后一级文件夹, 如果最后一级文件夹父路径不存在, 创建失败,返回值类型boolean类型, 创建成功返回true

3. mkdirs() : 每次可以创建一级或者多级文件夹路径, 返回值类型boolean类型, 创建成功返回true

代码

|  |
| --- |
| package com.ujiuye.file;  import java.io.File;  import java.io.IOException;  public class Demo02\_FileAddMethod {  public static void main(String[] args) throws IOException{    File f = new File("D:\\123.txt");  // createNewFile() : 当File表示的一个文件, 并且文件路径存在, 文件不存在时候, 此方法新建出一个空的文件,  // 返回值类型boolean类型, 创建成功返回true  boolean boo = f.createNewFile();  System.out.println(boo);    File f1 = new File("D:\\a");  // 每次只能创建出File类所表示的最后一级文件夹, 如果最后一级文件夹父路径不存在,创建失败,返回值类型boolean类型, 创建成功返回true  boolean boo1 = f1.mkdir();  System.out.println(boo1);    File f2 = new File("D:\\create\\file\\day01");  // mkdirs() : 每次可以创建一级或者多级文件夹路径, 返回值类型boolean类型, 创建成功返回true  boolean boo2 = f2.mkdirs();  System.out.println(boo2);  }  } |

### 1.4 File类的删除和重命名功能

1. delete() : 删除File所表示文件或者文件夹, 删除成功返回true. 删除失败返回false

注意 : 如果删除的是文件夹, 要求文件夹是空的才能删除, 这是安全保证; delete方法表示直接删除, 不走回收站

1. renameTo(File f): 将方法调用者的file所表示的文件进行重命名
2. 参数f的路径与方法调用者路径一致, 只表示修改文件名
3. 参数f的路径与方法调用者路径不一致, 先剪切到目标f路径下, 再重命名

代码

|  |
| --- |
| package com.ujiuye.file;  import java.io.File;  public class Demo03\_FileDeleteAndRename {  public static void main(String[] args) {  File f = new File("D:\\123.txt");  // 1. 直接删除文件, 如果文件真实存在, 删除成功  System.out.println(f.delete());    File f1 = new File("D:\\a");  // 2. 直接删除一个空的文件夹, 删除功能  System.out.println(f1.delete());    File f2 = new File("D:\\create");  // 3. 删除非空文件夹, 删除失败  System.out.println(f2.delete());    File f3 = new File("D:\\aa.txt");  // 4. f3的路径与参数路径一致, 只表示文件重命名  f3.renameTo(new File("D:\\aabbc.txt"));  File f4 = new File("D:\\bb.txt");  // 5. f3的路径与参数路径不一致 , 先将f3表示的文件剪切到目标参数路径下, 再重命名为change.txt  f4.renameTo(new File("D:\\1012系统班\\change.txt"));  }  } |

### 1.5 File类的判断功能

1. exists() : 验证File所表示文件或者文件夹是否真实存在, 如果存在返回true; 否则返回false;

2. isFile() : 验证File所表示的是否是一个文件, 如果是文件返回true; 否则返回false;

3. [isDirectory](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/File.html" \l "isDirectory())() : 验证File所表示的是否是一个文件夹, 如果是文件返回true; 否则返回false;

代码

|  |
| --- |
| package com.ujiuye.file;  import java.io.File;  public class Demo04\_FileCheckMethod {  public static void main(String[] args) {  File f = new File("D:\\1012系统班\\change.txt");  //1. exists() : 验证FIle所表示文件或者文件夹是否真实存在, 如果存在返回true; 否则返回false;  System.out.println(f.exists());// true  System.out.println(new File("D:\\hello.java").exists());// false    // 2. isFile() : 验证File所表示的是否是一个文件, 如果是文件返回true; 否则返回false;  System.out.println(f.isFile());  // 3. isDirectory() : 验证File所表示的是否是一个文件夹, 如果是文件返回true; 否则返回false;  System.out.println(f.isDirectory());// false  }  } |

### 1.6 File类的获取功能

1. [getAbsolutePath](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/File.html" \l "getAbsolutePath())() : 将File类型所表示的文件或者文件夹的绝对路径以字符串的形式进行返回

2. getPath() : 将File类型所封装的文件或者文件夹路径以字符串的形式进行返回

3. getName() : 将File类型所封装的文件或者文件夹路径最后一个部分的名字获取到, 返回String类型名字结果

4. length() : 表示获取File所表示的文件的长度(文件的大小), 以字节为丈量粒度, 返回值类型long类型(注意 : 不能丈量文件夹的大小, 文件夹数据不能保证正确)

实际场景使用 : 有时, 需要到指定路径下读取文件, 读取文件之前, 需要判断

File f = new File(“D:\\core.txt”);

if( f.exists() && f.length() > 0){

读取文件内容

}

1. [list](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/File.html" \l "list())() : 将File所表示的文件夹路径下, 所有的子级别(当前文件夹级别下)的文件和文件夹都获取到, 将获取到结果放置到String[]中
2. [listFiles](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/File.html" \l "listFiles())() : 将File所表示的文件夹路径下, 所有的子级别(当前文件夹级别下)的文件和文件夹都获取到, 将获取到结果放置到File[]中

代码

|  |
| --- |
| package com.ujiuye.file;  import java.io.File;  import java.util.Arrays;  public class Demo05\_FileGetMethod {  public static void main(String[] args) {  File f = new File("D:\\aabbc.txt");  // 1.getAbsolutePath() : 将File类型所表示的文件或者文件夹的绝对路径以字符串的形式进行返回  String path = f.getAbsolutePath();  System.out.println(path);// D:\aabbc.txt    File f1 = new File("123.txt");  System.out.println(f1.getAbsolutePath());// D:\workspace\day16\123.txt    // 2.getPath() : 将File类型所封装的文件或者文件夹路径以字符串的形式进行返回  System.out.println(f.getPath()); // D:\aabbc.txt  System.out.println(f1.getPath());// 123.txt    // 3.getName() : 将File类型所封装的文件或者文件夹路径最后一个部分的名字获取到, 返回String类型名字结果  System.out.println(f.getName());// aabbc.txt    File f2 = new File("D:\\1012系统班\\day01\\笔记");  System.out.println(f2.getName());//笔记    // 4.length() : 表示获取File所表示的文件的长度(文件的大小), 以字节为丈量粒度  System.out.println(f.length());// 10    // 5. list() : 将File所表示的文件夹路径下, 所有的子级别(当前文件夹级别下)的文件和文件夹都获取到,  // 将获取到结果放置到String[]中  String[] arr = f2.list();  System.out.println(Arrays.toString(arr));    // 6.listFiles() : 将File所表示的文件夹路径下, 所有的子级别(当前文件夹级别下)的文件和文件夹都获取到,  // 将获取到结果放置到File[]中  System.out.println(Arrays.toString(f2.listFiles()));    File f3 = new File("D:\\1012系统班");  System.out.println(Arrays.toString(f3.list()));  }  } |

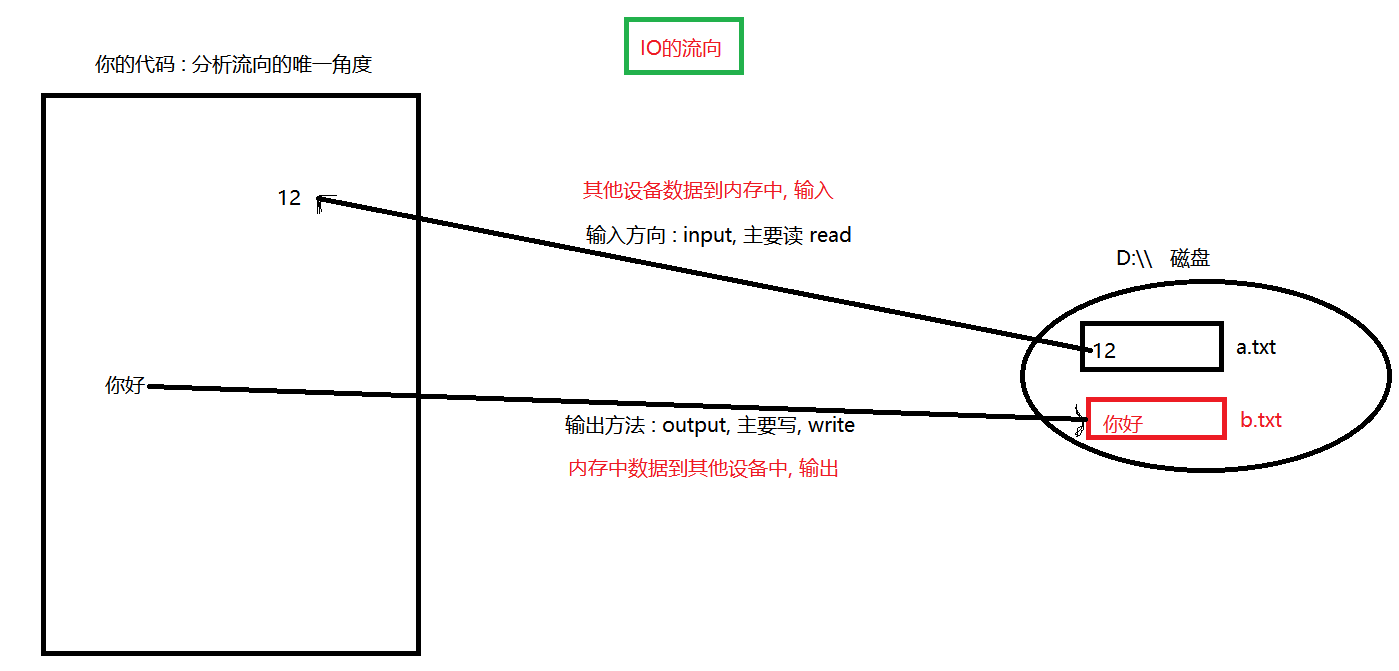
## **2.IO流介绍**

### 2.1 IO流概述

1. IO : I是i的大写字母, 表示input, 输入

O是o的大写字母, 表示output, 输出

1. IO流的主要作用就是进行数据传输, 不同的设备使用不同的IO流资源进行数据传输
2. IO的流向:



### 2.2 IO流的分类和使用步骤

1. IO流从功能性上分类:

字节流 : 以字节方式进行文件内容操作(读写)

输入流 : InputStream

输出流 : OutputStream

字符流 : 以字符方式进行文件内容操作(读写)

输入流 : Reader

输出流 : Writer

1. IO流使用的步骤:
2. 使用IO流资源之前, 需要导包, java.io
3. 使用IO流资源同时, 需要处理异常
4. 使用IO流资源结束之后, 必须关系资源, close()

## 字节流

1. 直接使用字节进行数据的操作
2. 从流向上进行分类:

输入流 : InputStream

输出流 : OutputStream

### 字节输入流

1. 字节输入流 : InputStream, 来自于java.io包, 是所有的字节输入流的父类;
2. 因为InputStream是抽象类, 不能实例化对象, 于是需要一个子类, FileInputStream, 来自于java.io包;
3. FileInputStream : 主要功能就是与磁盘上文件进行交互, 可以从磁盘上文件中读取出内容; 所有的文件都是以字节的形式组成, 因此FileInputStream 可以读取很多类型的文件内容, 例如 : txt, 图片, 视频...
4. FileInputStream的构造方法:
5. FileInputStream(String path) : 将path所表示的文件路径封装在一个字节输入流中, 以后字节输入流就可以从绑定的文件中读取出内容
6. FileInputStream(File file) : 将file所表示的文件路径封装在一个字节输入流中, 以后字节输入流就可以从绑定的文件中读取出内容

注意 : 构造方法中, 只能封装文件, 千万不能是文件夹, IO流资源只能对文件进行操作, 无法操作文件夹

1. 读取文件中内容:
2. read() : 表示每次从文件中读取出一个字节, 返回值int类型结果; 返回的就是读取到的字节所对应的整数结果, 如果读取到-1,证明文件读取完毕;

延伸 : 1个字节是8个连续的二进制位, 返回值类型int, 4字节大小, 因此读取到的字节需要补位成为一个4字节大小int, 因此前三个字节都补充0, 最高位相当于符号位, 0符号位就表示正数, read()方法读取到的结果全部都是正数

1. read(byte[] b) : 表示每次最多从文件中读取出b.length个字节, 读到的数据都同步到了参数b数组中, 通过查看数组b, 就相当于查看文件内容; 返回值类型int, 返回的是本次读取到的字节的个数, 如果读取到-1,证明文件读取完毕.

注意 : 按照数组进行文件内容读取比单个字节读取文件效率高很多

1. 读取文件的原理:



单个字节读取文件代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.File;  import java.io.FileInputStream;  import java.io.IOException;  public class Demo01\_FileInputStream {  // 可以随时查看类型之间的继承关系 : 把鼠标定位在类型上, ctrl + T可以查看这个类型的继承关系  public static void main(String[] args) throws IOException{  // 1. 创建出一个字节输入流, 相当于绑定一个数据源  FileInputStream fis = new FileInputStream("D:\\aabbc.txt");  // 注意 : IO流资源只能操作文件, 不能操作文件夹  // FileInputStream fis1 = new FileInputStream(new File("D:\\1012系统班"));  /\*int first = fis.read();  int second = fis.read();  int thrid = fis.read();  int four = fis.read();  int five = fis.read();  int six = fis.read();  System.out.println(first);// 97  System.out.println(second);// 98  System.out.println(thrid);// 99  System.out.println(four);// 33  System.out.println(five);// -1  System.out.println(six);// -1  \*/  // len表示每次读取到的字节数据  int len;  while((len = fis.read()) != -1) {  System.out.print((char)len);  }    // 一定要关闭资源  fis.close();  }  } |

字节数组读取文件代码:

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileInputStream;  import java.io.IOException;  public class Demo02\_FileInputStreamReadArr {  public static void main(String[] args) throws IOException{  FileInputStream fis = new FileInputStream("first.txt");    /\* byte[] b = new byte[4];  int i = fis.read(b);  System.out.println(i);// 4  System.out.println(new String(b));// hell  int i2 = fis.read(b);  System.out.println(i2);// 4  System.out.println(new String(b));// owor  int i3 = fis.read(b);  System.out.println(i3);// 2  System.out.println(new String(b));// 预想读到ld 但是读到 : ldor  int i4 = fis.read(b);  System.out.println(i4);// -1  System.out.println(new String(b));// ldor  \*/  byte[] b = new byte[12];  // len表示读取到的字节的个数  int len;  while((len = fis.read(b)) != -1) {  // b表示需要转换成字符串的数组  // 0表示读取的数组的起始索引位置  // len表示要从数组b中读取出几个字节转换成字符串  System.out.print(new String(b,0,len));  }  // 一定要关闭资源  fis.close();  }  } |

### 字节输出流

1. OutputStream : 是所有字节输出流的抽象父类, 来自于java.io包中
2. 因为OutputStream 是抽象类, 不能实例化对象, 因为需要一个子类 FileOutputStream
3. FileOutputStream : 是字节输出流, 主要功能就是与文件进行数据交互, 将字节数据写入到文件中, 因为所有文件都是由字节组成的, 因此FileOutputStream 可以向很多类型文件中写入数据, 例如 : txt, 图片, 视频...
4. FileOutputStream构造方法:
5. FileOutputStream(String path) : 将path所表示的文件路径封装在一个字节输出流中, 以后字节输出流就可以向目标文件中写入数据内容
6. FileOutputStream(File file) : 将file所表示的文件路径封装在一个字节输出流中, 以后字节输出流就可以向目标文件中写入数据内容

注意 : 构造方法中, 只能封装文件, 千万不能是文件夹, IO流资源只能对文件进行操作, 无法操作文件夹

1. 向文件中写入数据: write方法都没有返回值类型
2. write(int b) : 将字节数据b写入到文件中
3. write(byte[] b) : 将字节数组b中的所有内容写入到文件中
4. write(byte[] b, int index, int length) : 将数组b从index索引位置开始, 写入length个字节到文件中

注意 : 流资源使用完毕, 记得关闭资源; 按照字节数组写入数据比单个字节写入数据性能高很多

代码

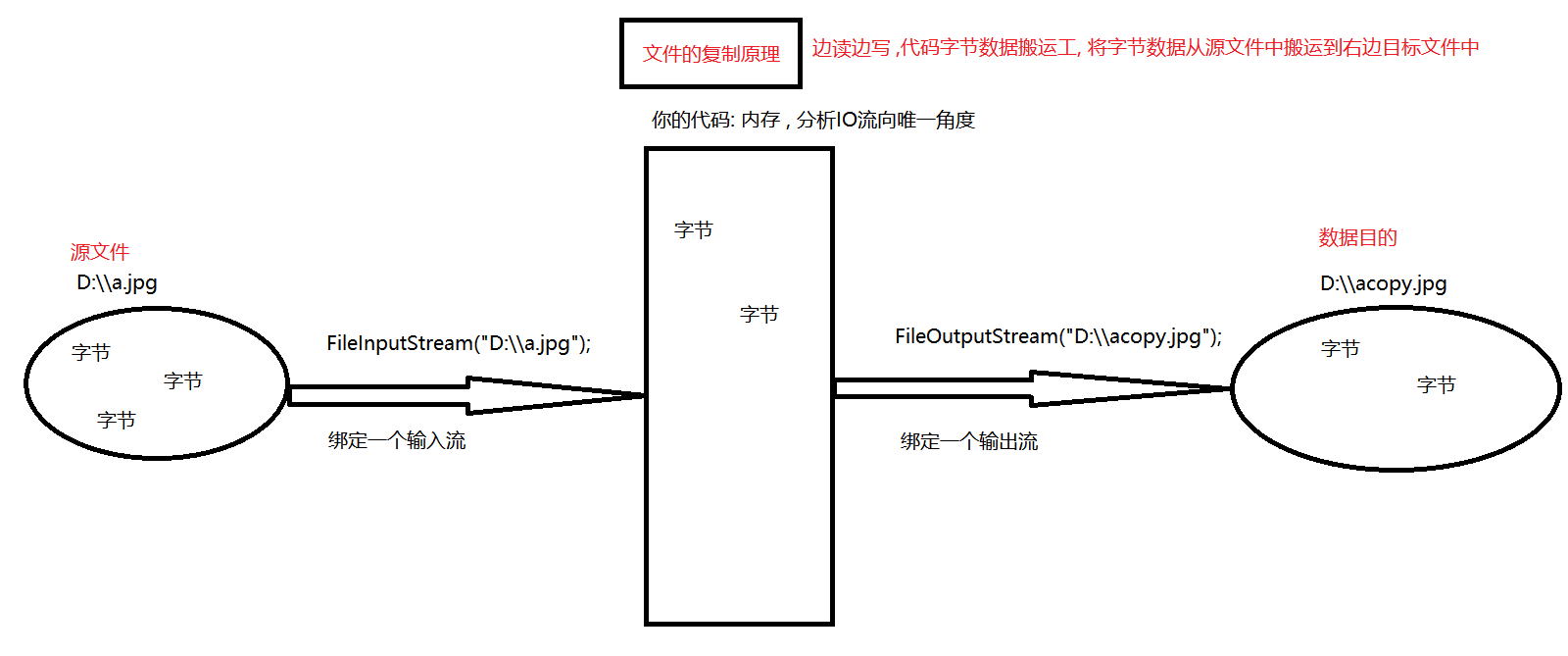
|  |
| --- |
| package com.ujiuye.io;  import java.io.File;  import java.io.FileOutputStream;  import java.io.IOException;  public class Demo03\_FileOutputStream {  public static void main(String[] args) throws IOException{  File f = new File("second.txt");  if(!f.exists()) {  f.createNewFile();  }  // 1. 创建出一个字节输出流, 相当于绑定一个数据目的  FileOutputStream fos = new FileOutputStream(f);  //aABCDECD  // 2. 向文件中写入数据  // 1)write(int b) : 将字节数据b写入到文件中  fos.write(97);// a  fos.write(65);// A    byte[] b = {66,67,68,69};  // 2)write(byte[] b) : 将字节数组b中的所有内容写入到文件中  fos.write(b);    // 3)write(byte[] b, int index, int length) : 将数组b从index索引位置开始,  // 写入length个字节到文件中  fos.write(b, 1, 2);// 67 68    // getBytes() : 将字符串转成字节数组  fos.write("你好".getBytes());    // 记得关闭资源  fos.close();  }  } |

### 文件的复制

文件复制 : 本质就是边读(数据源)边写(数据目的)

注意 : 实际开发中, 通过数组形式进行文件读写, 可以很大提高IO流的效能, 数组大小一般都是1024的倍数比较常见

最常见 : 1024 1024 \* 4 1024 \* 8 (1024\*8数组大小一般是JDK底层源代码比较倾向的数组大小设置)



代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  public class Demo04\_PictureCopy {  public static void main(String[] args) throws IOException{  // 1. 创建出一个字节输入流, 绑定一个数据源  FileInputStream fis = new FileInputStream("D:\\1012系统班\\day02\\图解\\变量定义和赋值过程.png");  // 2. 创建出一个字节输出流, 绑定一个数据目的  FileOutputStream fos = new FileOutputStream("D:\\变量定义copy2.png");    // 3. 单个字节复制  // len表示每次读取到的字节数据结果  /\*int len;  while((len = fis.read()) != -1) {  fos.write(len);  }\*/    // 4. 使用字节数组读写文件, 提高效率  byte[] b = new byte[1024\*4];  // len表示读取到的字节的个数  int len;  while((len = fis.read(b)) != -1) {  fos.write(b, 0, len);  }    fos.close();  fis.close();  }  } |

### 字节高效缓冲流

1. 字节高效缓冲流 : 作用就是提高普通字节流资源的读写效率, 缓冲流底层自动使用1024\*8字节数组进行文件的读写
2. BufferedInputStream : 字节高效缓冲输入流, 也是一个包装类, 功能将一个普通字节输入流,包装成一个高效字节缓冲流

通过构造方法可以看到包装过程 : **[BufferedInputStream](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/BufferedInputStream.html" \l "BufferedInputStream(java.io.InputStream))**([InputStream](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/InputStream.html" \o "java.io 中的类) in)

1. BufferedOutputStream : 字节高效缓冲输出流, 也是一个包装类, 功能将一个普通字节输出流,包装成一个高效字节缓冲流

通过构造方法可以看到包装过程 : **[BufferedOutputStream](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/BufferedOutputStream.html" \l "BufferedOutputStream(java.io.OutputStream))**([OutputStream](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/OutputStream.html" \o "java.io 中的类) out)

1. 高效原理:

BufferedInputStream : 当创建出一个BufferedInputStream 类型的输入流资源, 代码底层自动创建出一个大小为8192(1024 \* 8)的字节数组; 每次调用read方法读取数据, 都最多从文件中读取出8192个字节, 将读取到的字节数据放置到底层数组缓冲中, 以后读取数据,从底层数组缓冲中读取, 以此减少与磁盘交互, 从而提高读取效能; 直到读取到的结果为-1,读取完毕

BufferedOutputStream : 当创建出一个BufferedOutputStream 类型输出流资源,代码底层自动创建出一个大小为8192(1024 \* 8)字节数组;每次向文件中写入数据, 都是写入到了底层数组缓冲中, 等到将8192数组写满, 自动同步数据到文件中; 如果数组没有写满, 可以通过close方法, 关闭资源, 关闭资源之前, 会先将数组缓冲中的数据同步到文件中, 后关闭资源; 减少了与磁盘交互, 从而提高写入效能

代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.BufferedInputStream;  import java.io.BufferedOutputStream;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  public class Demo05\_BufferedStream {  public static void main(String[] args) throws IOException{  // 文件复制 : 使用高效缓冲流资源复制视频  // 1. 创建出一个字节高效缓冲输入流, 绑定一个数据源  BufferedInputStream bis = new BufferedInputStream(  new FileInputStream("D:\\1012系统班\\day01\\视频\\01.Java简介上.mp4"));    // 2. 创建出一个字节高效缓冲输出流, 绑定一个数据目的  BufferedOutputStream fos = new BufferedOutputStream(  new FileOutputStream("D:\\01.Java简介上Copy.mp4"));    long beginTime = System.currentTimeMillis();  // len表示每次读取到字节数据  int len;  while((len = bis.read()) != -1) {  fos.write(len);  }  long endTime = System.currentTimeMillis();  System.out.println(endTime-beginTime);  fos.close();  bis.close();  }  } |