# File对象

摘要: file对象是文件夹和文件的抽象表现形式, 并不是文件本身

File file = new File("E:\\demo\\aaa.txt");

当创建File对象后, 并未在相应的E盘目录下创建aaa.txt文件，File对象是文件夹和文件的抽象表现形式，并不是文件本身

API 简单说明

    // 描述file对象  
    File file = new File("E:\\demo");    
    File file1 = new File("E:\\demo\\test.txt");

    //创建文件夹    
    boolean mkdir = file.mkdir();    
    //创建文件    
    boolean createNewFile = file1.createNewFile();

    // 删除此抽象路径名表示的文件或目录    
    boolean delete = file1.delete();    
    boolean delete2 = file.delete();

    // 重新命名此抽象路径名表示的文件    
    File file2 = new File("E:\\demo\\sang.txt");    
    boolean renameTo = file1.renameTo(file2);

    // 判断功能  
    //测试此抽象路径名表示的文件是否是一个标准文件    
    boolean file3 = file.isFile();    
    //测试此抽象路径名表示的文件是否是一个目录    
    boolean directory = file.isDirectory();    
    //测试此抽象路径名表示的文件或目录是否存在    
    boolean exists = file.exists();    
    //测试应用程序是否可以读取此抽象路径名表示的文件    
    boolean canRead = file.canRead();    
    //测试应用程序是否可以修改此抽象路径名表示的文件    
    boolean canWrite = file.canWrite();    
    //测试此抽象路径名指定的文件是否是一个隐藏文件    
    boolean hidden = file.isHidden();

    //返回此抽象路径名的绝对路径名字符串    
    String absolutePath = file.getAbsolutePath();    
    //将此抽象路径名转换为一个路径名字符串    
    String path = file.getPath();    
    //返回由此抽象路径名表示的文件或目录的名称    
    String name = file.getName();    
    //返回由此抽象路径名表示的文件的长度    
    long length = file.length();    
    //返回此抽象路径名父目录的抽象路径名；如果此路径名没有指定父目录，则返回 null    
    File parentFile = file.getParentFile();

    //返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录    
    String[] list = file.list();    
    //返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中满足指定过滤器的文件和目录    
    String[] list2 = file.list(new FilenameFilter() {    
        public boolean accept(File dir, String name) {    
            return false;    
        }    
    });

    //返回一个抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件    
    File[] listFiles = file.listFiles();    
    //返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录    
    File[] listFiles2 = file.listFiles(new FilenameFilter() {    
        public boolean accept(File dir, String name) {    
            return false;    
        }    
    });

**一 : IO 流的分类**

      按照流向 ： 输入流 ， 输出流

      按照传输数据类型 ： 字节流 ， 字符流

      具体分为：字节输入流 InputStream，字节输出流 OutputStream，字符输入流 Reader，字符输出流 Writer

字节流可以处理所有类型数据格式的传输，例如图片和视频。字符流只可以处理文本格式数据的传输。先有字节流，后有字符流

**二 : 字节流**

**1, 字节输出流 OutputStream**

字节输出流OutputStream是抽象类 ，该类直接继承[Java](http://lib.csdn.net/base/java).lang.Object，是所有字节输出流的超类，其最常用的子类FileOutputStream

例子：使用字节输出流向文件中写入数据

//----------使用字节输出流向文件中写入数据----------------

//创建字节输出流对象

//执行new FileOutputStream("test.txt") 方法经过三个步骤

// 1,创建File文件对象 2,创建OutputStream对象 3,将输出流对象指向File文件对象

OutputStream out = new FileOutputStream("test.txt");

//-----------字节输出流 OutputStream的三种写数据方式------------

//一次写入一个字节

out.write('a');

//一次写入一个字节数组

out.write("你好啊，我的朋友".getBytes());

//一次写入一个字节数组的一部分,从某处开始,多少长度

//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

//此处为将byte数组中从0开始的3个字节写入此输出流

out.write("hellomyfriend".getBytes(), 0, 3);

//释放资源

out.close();

**2 , 字节输入流  InputStream**

InputStream是文件字节输入流的超类,直接继承自[Java](http://lib.csdn.net/base/java).lang.Object,其最常用的子类FileInputStream

小例子: 使用字节输入流读取文件中数据

/\*\*

\* 字节输入流读取文件中的数据

\*

\* 读取数据的方式 :

\* A : 一次读取一个字节 int read()

\* B : 一次读取一个字节数组 int read(byte[] b)

\*/

public static void main(String[] args) throws IOException {

//创建字节输入流对象

//该方法的主要执行了三件事

//1,创建File文件对象 2,创建字节输入流对象 3,将输入流对象指向File文件对象

InputStream in = new FileInputStream("test.txt"); // 文件内容: ABCDEFGHIJKLMN

//使用字节输入流读取文件中数据-----------

//第一种 : 一次读取一个字节--------------

int len = in.read();

//从输入流中读取数据的下一个字节,返回值为int类型,

//如果已经读到流的末尾而没有可用的字节则返回值为 -1

//通过循环的方法读取文件中所有的数据

while( len != -1 ){

System.out.print((char)len);

len = in.read();

}

//循环读取改进----

// int len1 = 0;

// // 将读取输入流动作在判断中执行

// while( (len1 = in.read()) != -1 ){

// System.out.print((char)len1);

// }

//第二种 : 一次读取一个字节数组------------

//定义一个用于存储的缓冲区数组

byte[] bys = new byte[1024];

//read(byte[] b)

//从输入流中读取一定数量的字节,字节的长度此处为1024个字节,

// 并将其存储在缓冲区数组 b 中,也就是存储到定义的 byte[] bys数组中

int len2 = in.read(bys);

String strContent = new String(bys);

System.out.println(strContent); //CDEFGHIJKLMN

//第三种 : 一次读取一个字节数组的一部分-----------

//定义一个用于存储缓冲区的数组

byte[] bys2 = new byte[1024];

//read(byte[] b, int off, int len)

//将输入流中最多 len 个数据字节读入 byte 数组,

// 从输入流中读取5个字节长度的数据存入定义好的 byte[] bys2 中

int len3 = in.read(bys2, 0, 5);

String content2 = new String(bys2);

System.out.println(content2); //ABCDE

//释放资源

in.close();

}

**三 , 文件复制 -- 流的对接**

文件包括：文件包括文本，图片，视频

主要分为四步：   
1，封装数据源  2，封装目的地 3，流的对接  4，释放资源

/\*\*

\* 使用字节流复制文件

\* @throws IOException

\*/

public static void main(String[] args) throws IOException {

// 1, 封装数据源

InputStream in = new FileInputStream("test.txt");

// 2, 封装目的地

OutputStream out = new FileOutputStream("aaa.txt");

// 3, 流的对接

// 方式一 ： 一次读取一个字节

// int len = 0;

//执行read()方法,将流中的数据写入到len中,如果读取到流的末尾则返回-1

//返回值len代表实际读取的字节个数

// while( (len = in.read()) != -1 ){

//一次写一个字节

// out.write(len);

// }

// 方式二 ： 一次读取一个字节数组

//定义一个用于存储读取数据的字节数组

byte[] bys = new byte[1024];

int len1 = 0;

//执行read()方法,将流中的数据写入到定义的数组bys中,如果读取到流的末尾则返回-1

//返回值len1代表实际读取的字节个数,不代表数组bys的长度

while( (len1 = in.read(bys)) != -1 ){

//一次写一个字节数组

// out.write(bys);

//一次写入一个字节数组的一部分

out.write(bys,0,len1);

}

// 4, 释放资源

in.close();

out.close();

}

**四 , 字节缓冲流**

     字节缓冲流  
     使用IO流复制文件时,字节流一次读写一个数组的速度明显比一次读写一个字节的速度快很多,这是因为加入了数组这样的缓冲效果,  
     java本身在设计的时候,也考虑到了这样的设计思想(装饰者模式), 所以提供了字节缓冲区流

     字节缓冲输入流 : BufferedInputStream

     字节缓冲输出流 : BufferedOutputStream

public static void main(String[] args) throws IOException {

//构造方法可以指定缓冲区的大小,不过一般默认的大小就可以了

//创建缓冲流时为什么不传递一个具体的文件，或者文件路径，而是传递 InputStream,OutputStream ?

//原因是字节缓冲区流仅仅提供缓冲区，为高效而设计，真正读写数据还得靠基本的流对象实现,

//所以，字节缓冲区流就像是水杯

BufferedInputStream in = new BufferedInputStream(new FileInputStream("aaa.txt"));

BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream("sang.txt"));

//方式1 ：一次读写一个字节

// int len = 0;

// while((len = in.read()) != -1){

// out.write(len);

// }

//方式2 ： 一次读写一个字节数组

byte[] bys = new byte[1024];

int len1 = 0;

while( (len1 = in.read(bys)) != -1 ){

//一次写入一个字节

out.write(bys);

//一次写入一个字节数组的一部分

out.write(bys, 0, len1);

}

out.close();

in.close();

}