IO流 input输入 output输出

Java IO操作主要是指使用Java程序完成输入输出操作，输入：将硬盘中的文件以数据流的形式读取到Java程序中。

输出：用Java程序将数据流写入到硬盘的文件中。

File类

IO流是通过Java程序对文件进行读写的，首先需要明白Java如何来表示文件，Java会将文件抽象成对象，有对象就一定有类，文件对象如何创建？通过java.io.File类来创建。

File类的实例化对象就是在描述一个物理资源/文件。

File的常用方法

public File(String pathname) 根据路径创建对象

public String getName() 获取文件名

public String getParent() 获取文件所在的目录

public File getParentFile() 获取文件所在的目录对应的File对象

public String getPath() 获取文件路径

public boolean exists() 判断对象是否存在

public boolean isDirectory() 判断对象是否为目录

public boolean isFile() 判断对象是否为文件

public long length() 获取文件的大小，以字节为单位

public boolean createNewFile() 根据当前对象创建新文件

public boolean delete() 删除对象

public boolean mkdir() 根据当前对象创建新目录

public boolean renameTo(File dest) 为已存在的对象重命名

在UTF-8编码规则下，一个字母，空格，回车，标点符号... 占一个字节 byte

一个汉字占3个字节

abc：3

你好：6

createNewFile：如果该file对象不存在，则根据其路径创建该文件，返回true

如果file对象已经存在，则创建失败，返回false

renameTo：给文件重命名，如果原文件和目标文件在不同的路径下，会按照目标文件的路径重新设置该文件的路径，相当于把原文件进行移动。

```java
package com.southwind.io;

import java.io.File;
import java.io.IOException;

public class FileTest {
    public static void main(String[] args) {
//      String pathname = "/Users/southwind/Desktop/iotest/test.txt";
//      File file = new File(pathname);
//      if(file.exists()) {
//          String name = file.getName();
//          System.out.println("文件名: "+name);
//          String parentPath = file.getParent();
//          System.out.println("文件所在的目录: "+parentPath);
//          File parentFile = file.getParentFile();
//          test(parentFile);
//          test(file);
//          System.out.println(file+"的长度: "+file.length());
//      }
        String path = "/Users/southwind/Desktop/iotest/测试.txt";
        File file = new File(path);
        try {
//          System.out.println(file.mkdir());
            File file2 = new File("/Users/southwind/Desktop/test.txt");
            System.out.println(file.renameTo(file2));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void test(File file) {
        boolean flag = file.isDirectory();
        if(flag) {
            System.out.println(file+"是一个目录");
        }else {
            System.out.println(file+"不是一个目录");
        }
        boolean flag2 = file.isFile();
        if(flag2) {
            System.out.println(file+"是一个文件");
        }else {
            System.out.println(file+"不是一个文件");
        }
    }
}
```

IO

I:input 输入流

O:output 输出流

流就是一组有序的数据序列，以先进先出的方式发送数据的通道。

Java中的流有很多种不同的分类：

1.按照方向分，可以分为输入流和输出流，如何判断输入还是输出，站在Java程序角度，看数据是从外界来到Java中呢还是从Java中输出到外界，外届-Java 就是输入流，Java-外界 就是输出流。

2.按照单位分，可以分为字节流和字符流，字节流指每次处理数据都是以字节为单位，字符流是指每次处理数据都是以字符为单位。

3.按照功能分，可以分为节点流和处理流。

字节流按照方向可以分为输入字节流（InputStream）和输出字节流（OutputStream）

输入字节流 InputStream，通过该流可以将文件中的数据读入到Java程序中。

InputStream是java.io包中的顶层父类，实现了Closeable接口，该接口的作用是每次操作结束之后释放资源，InputStream常用的方法。

public abstract int read() 以字节为单位读取文件中的数据

public int read(byte b[]) 将数据存入byte类型的数组中，并返回数据的长度

public int read(byte b[],int off,int len) 将数据存入byte类型的数组的指定区间中，并返回数据的长度

public byte[] readAllBytes() 将数据存入byte类型的数组中，并返回该数组

public int available() 返回当前数据中未读取的数据个数

public void close() 关闭数据流

实际开发中不能直接实例化InputStream，因为它是一个抽象类，应该实例化其实现了抽象方法的子类，FileInputStream

FileInputStream的实例化就相当于一根连接到某个文件的管道，通过该管道就可以把文件中的数据取出读取到Java程序中，FileInputStream只能连接文件，不能连接目录，否则会抛出FileNotFoundException。

read()读取数据，每一此read()返回值就是一个字节的数据，如果取出的数据为-1，则表示该文件的所有的数据已经全部取出。

```
package com.southwind.io;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public class IOTest {
```

```
    public static void main(String[] args) {
        try {
//          File file = new File("/Users/southwind/Desktop/test.txt");
//          InputStream inputStream = new FileInputStream(file);
//          long length = file.length();
//          for(int i = 0; i < length; i++) {
//              int temp = inputStream.read();
//              System.out.println(temp);
//          }

//          int temp = inputStream.read();
//          System.out.println(temp);
//          while(temp != -1) {
//              temp = inputStream.read();
//              if(temp == -1) {
//                  break;
//              }
//              System.out.println(temp);
//          }
            InputStream inputStream = new
FileInputStream("/Users/southwind/Desktop/test.txt");
//          System.out.println("当前还有"+inputStream.available()+"byte的数据
未读");
//          int temp = 0;
//          while((temp = inputStream.read())!=-1) {
//              System.out.println("当前还有"+inputStream.available()+"byte
的数据未读");
//              System.out.println(temp);
//          }
//          inputStream.close();

//          byte[] bytes = new byte[2];
//          int length = inputStream.read(bytes);
//          System.out.println(length);
//          for (byte b : bytes) {
//              System.out.println(b);
//          }

//          byte[] bytes = new byte[10];
//          int length = inputStream.read(bytes,3,2);
//          System.out.println(length);
//          for (byte b : bytes) {
//              System.out.println(b);
//          }

            byte[] bytes = inputStream.readAllBytes();
            for (byte b : bytes) {
                System.out.println(b);
            }
```

```
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

OutputStream跟InputStream类似，也是一个抽象父类，

OutputStream常用方法

public abstract void write(int b) 以字节为单位写数据

public void write(byte b[]) 将byte类型数组中的数据写出

public void write(byte b[],int off,int len) 将byte类型数组中指定区间的数据写出

public void flush() 可以强制将缓冲区的数据同步到输出流中

public void close() 关闭数据流

实际开发中不能直接实例化OutputStream的对象，因为它是一个抽象类，我们应该实例化其实现了抽象方法的字了FileOutputStream。

FileOutputStream(File file) file如果不存在，则自动创建该文件。

```
package com.southwind.io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class OutputStreamTest {
    public static void main(String[] args) {
        try {
            OutputStream outputStream = new
FileOutputStream("/Users/southwind/Desktop/test.txt");
//          outputStream.write(111);
//          byte[] bytes = {97,98,99};
//          outputStream.write(bytes);
//          byte[] bytes = {(byte)228,(byte)189,(byte)160,(byte)229,
(byte)165,(byte)189};
```

```
//          byte[] bytes = {(byte)228,(byte)189,(byte)160,(byte)229,
(byte)165,(byte)189};
//          outputStream.write(bytes,3,3);
            outputStream.flush();
            outputStream.close();

//          InputStream inputStream = new
FileInputStream("/Users/southwind/Desktop/test.txt");
//          int temp = 0;
//          while((temp = inputStream.read())!=-1) {
//              System.out.println(temp);
//          }
//          inputStream.close();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
}
```

字符流

字符流是有别于字节流的另外一种数据流，两者的区别在于每次处理数据的单位不同，字节流是以字节为单位，字符流是以字符为单位。

字符流按照方向可以分为输入字符流（Reader）和输出字符流（Writer）

Reader是一个抽象类，实现了Readable和Colseable两个接口，Closeable接口表示该类的实例化对象可以手动释放资源，Readable接口的作用是可以将字符读入指定的字符缓冲区。

Reader常用方法

public int read() 以字符为单位读取数据

public int read(char cbuf[]) 将数据读入char类型的数组，并返回数据长度

public int read(char cbuf[],int off,int len) 将数据读入char类型数组的指定区间，并返回数据长度

public abstarct void close() 关闭数据流

public long transferTo(Writer out) 将数据直接读入字符输出流

我们在使用Reader进行读入操作时，不能直接实例化Reader的对象，因为Reader是一个抽象类，我们应该实例化其实现了抽象方法的子类FileReader。

在UTF-8编码规范下，一个英文字母对应一个字节（byte），对应一个字符（char）

一个汉字对应三个字节（byte），对应一个字符（char）

```
package com.southwind.io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.Reader;
import java.io.Writer;

public class PngTest {
    public static void main(String[] args) {
//      try {
//          InputStream inputStream = new
FileInputStream("/Users/southwind/Desktop/1.png");
//          int temp = 0;
//          OutputStream outputStream = new
FileOutputStream("/Users/southwind/Desktop/2.png");
//          while((temp = inputStream.read())!=-1) {
//              outputStream.write(temp);
//          }
//          inputStream.close();
//          outputStream.close();
//      } catch (FileNotFoundException e) {
//          // TODO Auto-generated catch block
//          e.printStackTrace();
//      } catch (IOException e) {
//          // TODO Auto-generated catch block
//          e.printStackTrace();
//      }

        try {
            Reader reader = new
FileReader("/Users/southwind/Desktop/1.png");
            int temp = 0;
            Writer writer = new
FileWriter("/Users/southwind/Desktop/2.png");
            while((temp = reader.read())!=-1) {
                writer.write(temp);
            }
            reader.close();
            writer.flush();
```

```
            writer.close();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

用IO进行文件的读写时，输入流和输出流的类型必须一致，即要么都是字节流，要么都是字符流，不能用字节流读取，然后用字符流输出，也不能用字符流然后用字节流输出，否则会导致文件的数据错误，因为打乱了数据的内部排列结构。

如果是文本类型的数据，txt，word既可以用字节流完成读取，也可以用字符流完成读取，但是非文本类型的，如图片，视频，音频等，只能用字节流读取，用字符流读取同样会导致文件的数据错误，从而导致文件无法打开。

Writer输出字符流，和Reader一样，它也是一个抽象类。

Appendable接口的实现类的对象能够添加char序列和值。

Writer常用方法

public void write(int c) 以字符为单位写数据

public void write(char cbuf[]) 将插入类型数组中的数据写出

public abstract void write(char cbuf[],int off,int len) 将插入类型数组中指定区间的数据写出

public void write(String str) 将String类型的数据写出

public void write(String str,int off,int len) 将String类型指定区间的数据写出

public abstract void flush() 可以强制将缓冲区的数据同步到输出流中

public abstract void close() 关闭数据流

```
package com.southwind.io;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

public class WriterTest {
    public static void main(String[] args) {
        try {
```

```
            Writer writer = new
FileWriter("/Users/southwind/Desktop/test.txt");
//          writer.write(20320);
//          writer.write(22909);
//          char[] chars = {'你','好','世','界'};
//          writer.write(chars);
            String str = "Hello World";
//          writer.write(chars,2,2);
//          writer.write(str);
            writer.write(str, 6, 3);
            writer.flush();
            writer.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
}
```