IO

概述

IO操作是整个Java中最复杂的开发包，是一个难点也是一个重点，想学好IO，必须对面向对象的基本概念非常的熟悉，包括抽象类。

整个IO包最核心的就是5个类和1个接口：

File、InputStream、OutputStream、Writer、Reader、Serializable。

所有类和接口基本都在java.io包定义

File类

File类提供了与平台无关的操作磁盘上文件或目录的方法，每一个File对象表示磁盘上的一个文件或目录

File类只是对文件的属性进行操作，并没有提供方法来操作文件内容

文件、目录的创建和删除

创建或删除一个目录或文件

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

/**
 * 文件的创建、删除、创建父目录
 *
 * @author Kevin
 *
 */
public class FileDemo1 {

    public static void main(String[] args) {

        File file = new File("D:"+File.separator
                             +"demo"+File.separator
                             +"a"+File.separator
                             +"b"+File.separator
                             +"text.txt");
        try {
            //获取上级目录
            File parentFile = file.getParentFile();
            //判断上级目录是否存在
            if(!parentFile.exists()){
                //不存在，则创建（mkdirs()）
                System.out.println("创建目录："+parentFile.mkdirs());
            }
            //判断文件是否存在
            if(file.exists()){
                //存在-->删除
                System.out.println("删除文件："+file.delete());
            }else{
                //不存在-->创建
                System.out.println("创建文件："+file.createNewFile());;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

上述代码中，先判断文件的上级目录是否存在，如果不存在则创建上级目录，然后判断文件是否存在，存在则删除，不存在则创建。

常用方法

1. `public boolean exists()`: 判断文件或目录是否存在
2. `public boolean mkdirs()`:递归创建目录
3. `public boolean createNewFile() throws IOException`:创建文件
4. `public boolean delete()`: 删除文件

文件的操作

```java
import java.io.File;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * 获取文件的属性
 *
 * @author Kevin
 *
 */
public class FileDemo2 {

    public static void main(String[] args) {
        File file = new File("D:" + File.separator + "demo" + File.separator + "im.g0.jpg");
        String name = file.getName();
        System.out.println("文件名：" + name);
        System.out.println("文件后缀：" + name.substring(name.lastIndexOf(".") + 1));
        System.out.println(file.getParent());
        System.out.println(file.getParentFile());
        System.out.println(file.getAbsolutePath());
        System.out.println("是否是目录：" + file.isDirectory());
        System.out.println("是否是文件：" + file.isFile());
        long length = file.length();
        double size = (double)length / 1024;
        DecimalFormat format = new DecimalFormat("0.00");
        System.out.println(size);
        System.out.println(format.format(size));
        //文件最后修改时间
        long lastModified = file.lastModified();
        System.out.println(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss SSS").format(new
Date(lastModified)));
    }

}
```

常用方法

1. `public String getName()`：获取文件名
2. `public String getParent()`：获取上级目录的字符串没描述
3. `public File getParentFile()`：获取上级目录的文件描述
4. `public String getAbsolutePath()`：获取绝对路径
5. `public long length()` ：获取文件大小
6. `public long lastModified()`： 获取文件的最后修改时间

目录的操作

```java
import java.io.File;

/**
 * 目录的操作
 *
 * @author Kevin
 */
public class FileDemo3 {

    public static void main(String[] args) {
        File file = new File("D:" + File.separator);
        String[] list = file.list();
        for (String s : list) {
            System.out.println(s);
        }
        File[] listFiles = file.listFiles();
        int fileCount = 0;// 保存文件数量
        int dirCount = 0;// 保存目录数量
        for (File f : listFiles) {
            if (f.isFile()) {// 是文件
                fileCount++;
            } else if (f.isDirectory()) {// 是目录
                dirCount++;
            }
        }
        System.out.println("目录：" + dirCount + ",文件：" + fileCount);
    }

}
```

常用方法

1. `public String[] list()`：以字符串数组的形式返回当前目录下的目录和文件
2. `public File[] listFiles()`：以File数组的形式返回当前目录下的目录和文件
3. `public boolean isFile()`：是否是文件
4. `public boolean isDirectory()`：是否是目录

递归算法

程序调用自身的编程技巧叫做递归

获取目录下的全部文件和目录

```java
import java.io.File;

/**
 * 递归获取目录树
 *
 * @author Kevin
 *
 */
public class FileDemo4 {

    public static void main(String[] args) {
        File file = new File("D:"+File.separator
                +"ProgramFiles");
        print(file);
    }

    public static void print(File file) {
        //如果是目录
        if(file.isDirectory()){
            //获取目录下的子目录和文件
            File[] files = file.listFiles();
            //遍历每一个子目录或文件
            for (File f : files) {
                //调用自身
                print(f);
            }
        }
        System.out.println(file);
    }
}
```

字节流和字符流

File类本身可以对文件进行操作，但是不能操作内容，如果需要对文件内容进行读写，需要用到字节流和字符流
基本步骤：

1. 使用File类确定要操作的文件
2. 使用字节流或字符流的子类进行对象的实例化
3. 进行读或写操作
4. 关闭字节流或字符流

注意：流的操作属于资源操作，所有的资源操作都需要释放资源

字节流和字符流分为：

- 字节流：OutputStream、InputStream
- 字符流：Writer、Reader

字节输出流

字节输出流使用的是`OutputStream`，其类的定义如下：

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

可以发现，此类事抽象类需要用其子类进行实例化，以`FileOutputStream`为例

构造方法

1. `public FileOutputStream(File file) throws FileNotFoundException`：根据指定文件获取字节输出流对象，每次写入内容会覆盖
2. `public FileOutputStream(File file, boolean append)throws FileNotFoundException`：根据指定文件获取字节输出流，可以确定是覆盖写入还是追加

常用方法

1. `public void write(int b) throws IOException`：写入一个字节
2. `public void write(byte b[], int off, int len) throws IOException`：指定从数组的第几个字节开始写，往后写几个
3. `public void write(byte b[]) throws IOException`：将全部数组写入

例子

```java
import java.io.File;
import java.io.FileOutputStream;

/**
 * 字节输出流
 *
 * @author Kevin
 *
 */
public class OutputStreamDemo {

    public static void main(String[] args) {
        try {
            // 1.使用File类确定要操作的文件
            File file = new File("D:" + File.separator + "demo" + File.separator + "a.txt");
            if (!file.exists()) {
                file.createNewFile();
            }
            // 2.使用FileOutputStream实例化字节输出流对象
            FileOutputStream os = new FileOutputStream(file);
            // 2.1 准备要写入的数据
            String str = "你好世界";
            // 2.2 将要写入的字符串转换为byte数组
            byte[] bytes = str.getBytes();
//            for (int i = 0; i < bytes.length; i++) {
//                os.write(bytes[i]);
//            }
//            os.write(bytes, 2, 4);//从第1个字节往后写3个
            os.write(bytes);//将字节数组全部写入
            // 4.释放资源
            os.close();
            System.out.println("写入成功");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

以上代码执行之后，会在`a.txt`写入‘你好世界’，但是每次执行会覆盖原先内容。如果想要追加，需要使用两个参数的构造方法，将第二个参数设置为`true`，此时就可以以追加的形式输出内容。如果想要换行则需要添加`\r\n`

字节输入流

在程序中，可以使用OutputStream进行信息输出，也可以使用InputStream进行输入。

```java
public abstract class InputStream
extends Object
implements Closeable
```

此类事抽象类，需要使用其子类进行对象实例化，以`FileInputStream`为例

构造方法

`public FileInputStream(File file) throws FileNotFoundException`

常用方法

1. `public int read(byte b[]) throws IOException`：从字节输入流读取数据到指定byte数组
2. `public int read(byte b[], int off, int len) throws IOException`：从指定字节处开始读取，指定数量个字节到指定数组

例子

```
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

public class InputStreamDemo {

    public static void main(String[] args) {
        try {
            // 1.通过File类确定要读取的文件
            File file = new File("D:" + File.separator + "demo" + File.separator + "b.txt");
            // 2.实例化FileInputStream对象
            InputStream is = new FileInputStream(file);
            // 3.开始读取
            byte[] b = new byte[256];
            int len = 0;
            StringBuilder sb = new StringBuilder();
            while ((len = is.read(b)) != -1) {
                String str = new String(b, 0, len);
                sb.append(str);
            }
            // 4.释放资源
            is.close();
            System.out.println("[" + sb + "]");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

字符输出流

字符输出流值得是Writer，此类的定义如下：

```
public abstract class Writer implements Appendable, Closeable, Flushable
```

可以发现，此类是一个抽象类，需要使用其子类进行实例化，以`FileWriter`为例：

构造方法

1. `public FileWriter(File file) throws IOException`：通过指定的文件创建字符输出流，默认每次写入会覆盖之前内容
2. `public FileWriter(File file, boolean append) throws IOException`：通过指定文件创建字符输出流，可以指定四覆盖还是追加

常用方法

1. `public void write(String str) throws IOException`：直接写入字符串
2. `public void write(char cbuf[], int off, int len) throws IOException`：写入字符数组，从指定位置的字符开始写，往后写len个
3. `public void write(char cbuf[]) throws IOException`：写入全部字符数组

例子

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

public class WriterDemo {
    public static void main(String[] args) {
        try {
            // 1. 通过File类确定要操作的文件
            File file = new File("D:" + File.separator + "demo" + File.separator + "b.txt");
            // 2. 通过FileWriter进行实例化
            Writer writer = new FileWriter(file);
            // 3.开始写入
            String str = "你好世界";
            char[] cbuf = str.toCharArray();
            writer.write(cbuf, 1, 2);
            // 4.释放资源
            writer.close();
            System.out.println("写入成功");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

字符输入流

字符输入流指的是Reader，此类的定义如下：

```
public abstract class Readerextends Objectimplements Readable, Closeable
```

此类是一个抽象类，需要使用其子类进行实例化，以`FileReader`为例

构造方法

`public FileReader(File file) throws FileNotFoundException`

常用方法

`public int read(char cbuf[]) throws IOException`：读取数据到char类型数组

例子

```
import java.io.File;
import java.io.FileReader;
import java.io.Reader;

public class ReaderDemo {
    public static void main(String[] args) {
        try {
            // 1，指定要操作的文件
            File file = new File("D:" + File.separator + "demo" + File.separator + "b.txt");
            // 2. 实例化操作流
            Reader reader = new FileReader(file);
            // 3. 开始读取
            char[] cbuf = new char[32];
            int len = 0;//保存读到多少数据
            StringBuilder sb = new StringBuilder();
            while((len = reader.read(cbuf)) != -1){
                String str = new String(cbuf, 0, len);
                sb.append(str);
            }
            // 4.释放资源
            reader.close();
            System.out.println("[" + sb + "]");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

区别

类名	继承关系
FileWriter	`Object` -> `Writer` -> `OutputStreamWriter` -> `FileWriter`
FileReader	`Object` -> `Reader` -> `InputStreamReader` -> `FileReader`

1. 当把字节流操作和字符流操作的释放资源都取消的时候，发现，字节流可以正常写入，但是字符流并没有写入
2. 字符流操作，看起来一直在操作的是字符数据，其时在类的内部做了相应的转（char-->byte），转换好的字节数据存储在缓存区，如果没有调用close方法，则缓存区内的字节数据不会内输出。如果有需求不可以关闭字符流，可以调用flush()方法，强制清空缓冲区
3. 在磁盘上文件的存储都是以字节的形式，在开发中，字节数据使用比较多（比如：音频，视频，图片...），这时优先使用字节流。如果需要操作中文文本数据，优先使用字符流（字节流可能造成中文乱码）

练习-复制文件

复制D盘demo下的oracle.jpg到demo下的a目录，并改名为a.jpg

第一种方式

每次读一个字节，边读边写

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

public class CopyFileDemo {

    public static void main(String[] args) {
        try {
            File inFile = new File("D:"+File.separator+"demo"+File.separator+"oracle.jpg");
            File outFile = new
File("D:"+File.separator+"demo"+File.separator+"a"+File.separator+"a.jpg");
            if(!outFile.exists()){
                outFile.createNewFile();
            }
            InputStream is = new FileInputStream(inFile);
            OutputStream os = new FileOutputStream(outFile);
            //开始读
            System.out.println("开始复制。。。");
            long start = System.currentTimeMillis();
            int len = 0;
            while((len = is.read()) != -1){
                os.write(len);
            }
            long end = System.currentTimeMillis();
            System.out.println("复制成功，耗时："+(end-start)+"毫秒");
            is.close();
            os.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

此时我们发现，虽然可以复制成功，但是速度太慢，需要改进，也就是说，不应该每次读写都是一个字节，需要一个缓冲区

第二种

添加缓冲区（字节数组）每次读取和写入的大小就是数组的长度

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

public class CopyFileDemo {

    public static void main(String[] args) {
        try {
            File inFile = new File("D:" + File.separator + "demo" + File.separator +
"oracle.jpg");
            File outFile = new File("D:" + File.separator + "demo" + File.separator + "a" +
File.separator + "a.jpg");
            if (!outFile.exists()) {
                outFile.createNewFile();
            }
            InputStream is = new FileInputStream(inFile);
            OutputStream os = new FileOutputStream(outFile);
            // 开始读
            System.out.println("开始复制。。。");
            long start = System.currentTimeMillis();
            int len = 0;
            byte[] b = new byte[1024];
            while ((len = is.read(b)) != -1) {
                os.write(b, 0, len);
            }
            long end = System.currentTimeMillis();
            System.out.println("复制成功，耗时：" + (end - start) + "毫秒");
            is.close();
            os.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

可以发现，速度有了明显提升

转换流

有些时候，我们需要将字节流数据转换为字符流进行操作，这时就需要用到流的转换，需要以下两个类：

类名	构造方法	功能
InputStreamReader	`InputStreamReader(InputStream in)`	字节输入流->字符输入流
OutputStreamWriter	`OutputStreamWriter(OutputStream out)`	字节输出流->字符输出流

打印流

概述

> 有时候我们需要输出字符串类型，或者布尔类型，或者字符类型，这时如果使用OutputStream就很不方便了，因为OutputStream中只能操作字节数据，其他类型的数据很难操作。为了解决这个问题，在java.io包提供了两个类：

- PrintStream
- PrintWriter

通过API文档我们发现两个类的结构基本一样，只是PrintWriter的构造方法可以接收Writer对象

例子

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;

public class PrintStreamDemo {

    public static void main(String[] args) {
        try {
            File file = new File("D:" + File.separator + "demo" + File.separator + "c.txt");
            if (!file.exists()) {
                file.createNewFile();
            }
            FileOutputStream out = new FileOutputStream(file, true);
            PrintStream ps = new PrintStream(out);
            String name = "张三";
            int age = 25;
            float salary = 10.456f;
            ps.printf("姓名：%s,年龄：%d,工资：%1.2f", name, age, salary);
            ps.println("HELLOWORLD");
            ps.print("helloworld");
            ps.close();
            out.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

总结

在开发中打印流可以输出各种数据类型，比较好用。

在JDK1.5之后，为打印流添加了数据格式的支持：%s,%d,%f

System类对IO的支持

- 错误输出：`System.err`
- 屏幕输出：`System.out`

- 键盘输入：`System.in`

错误输出

```
String str = "这是一个错误信息";
System.out.println(str);
System.err.println(str);
```

System.err的用法和System,out一致，System.err在eclipse中输出显示为红色

屏幕输出

```
PrintStream out = System.out;
out.println("Hello");
```

上述代码等价于：

```
System.out.println("Hello");
```

键盘输入

```
InputStream is = System.in;
byte[] b = new byte[2];
System.out.print("请输入：");
int read = is.read(b);
String str = new String(b,0,read);
System.out.println("输入的是："+str);
```

此时，我们发现，可以正常获取到用户的键盘输入。但是此程序还存在问题，当用户输入的数量大于数组的容量时，就会有一部分输入内容丢失，还需要进一步改进：

```
InputStream is = System.in;
byte[] b = new byte[2];
StringBuilder sb = new StringBuilder();
System.out.print("请输入：");
int read = 0;
while ((read = is.read(b)) != -1) {
    sb.append(new String(b, 0, read));
    if(sb.charAt(sb.length()-1) == '\n'){
        break;
    }
}
System.out.println("输入的是：" + sb);
```