day04

File类

删除目录

delete()方法可以删除一个目录，但是只能删除空目录。

```
package file;

import java.io.File;

/**
 * 删除一个目录
 */
public class DeleteDirDemo {
    public static void main(String[] args)
{
        //将当前目录下的demo目录删除
        File dir = new File("demo");
//        File dir = new File("a");
        if(dir.exists()){
            dir.delete();//delete方法删除目录
时只能删除空目录
            System.out.println("目录已删
除!");
        }else{
```

```
            System.out.println("目录不存
在!");
        }
    }
}
```

访问一个目录中的所有子项

listFiles方法可以访问一个目录中的所有子项

```
package file;

import java.io.File;

/**
 * 访问一个目录中的所有子项
 */
public class ListFilesDemo1 {
    public static void main(String[] args)
{
        //获取当前目录中的所有子项
        File dir = new File(".");
        /*
            boolean isFile()
            判断当前File表示的是否为一个文件
            boolean isDirectory()
            判断当前File表示的是否为一个目录
         */
        if(dir.isDirectory()){
            /*
                File[] listFiles()
```

```
                        将当前目录中的所有子项返回。返回
的数组中每个File实例表示其中的一个子项
                 */
                File[] subs = dir.listFiles();
                System.out.println("当前目录包
含"+subs.length+"个子项");
                for(int i=0;i<subs.length;i++){
                    File sub = subs[i];

 System.out.println(sub.getName());
                }
            }
        }
}
```

获取目录中符合特定条件的子项

重载的listFiles方法:File[] listFiles(FileFilter)

该方法要求传入一个文件过滤器，并仅将满足该过滤器要求的子项返回。

```
package file;

import java.io.File;
import java.io.FileFilter;

/**
 * 重载的listFiles方法，允许我们传入一个文件过滤器
从而可以有条件的获取一个目录
 * 中的子项。
```

```
 */
public class ListFilesDemo2 {
    public static void main(String[] args)
{
        /*
            需求:获取当前目录中所有名字以"."开始
的子项
         */
        File dir = new File(".");
        if(dir.isDirectory()){
//            FileFilter filter = new
FileFilter(){//匿名内部类创建过滤器
//                public boolean
accept(File file) {
//                    String name =
file.getName();
//                    boolean starts =
name.startsWith(".");//名字是否以"."开始
//                    System.out.println("过
滤器过滤:"+name+",是否符合要求:"+starts);
//                    return starts;
//                }
//            };
//            File[] subs =
dir.listFiles(filter);//方法内部会调用accept方
法


            File[] subs = dir.listFiles(new
FileFilter(){
```

```
                public boolean accept(File
file) {
                    return
file.getName().startsWith(".");
                }
            });

 System.out.println(subs.length);
        }
    }
}
```

使用递归操作删除一个目录

循环是重复执行某个步骤，而递归是重复整个过程。

```
package file;

import java.io.File;

/**
 * 编写一个程序，要求实现1+2+3+4+....100并输出结
果。
 * 代码中不能出现for，while关键字
 *
 * 编写程序计算:
 * 一个人买汽水，1块钱1瓶汽水。3个瓶盖可以换一瓶汽
水，2个空瓶可以换一瓶汽水。不考虑赊账问题
 * 问20块钱可以最终得到多少瓶汽水。
```

```
 *
 * 删除一个目录
 */
public class Test {
    public static void main(String[] args)
{
        File dir = new File("./a");
        delete(dir);
    }

    /**
     * 将给定的File对象表示的文件或目录删除
     * @param file
     */
    public static void delete(File file){
        if(file.isDirectory()) {
            //清空目录
            File[] subs = file.listFiles();
            for (int i = 0; i <
subs.length; i++) {
                File sub = subs[i];//从目录中
获取一个子项
                //将该子项删除
                delete(sub);//递归调用
            }
        }
        file.delete();
    }
}
```

Lambda表达式

JDK8之后,java支持了lambda表达式这个特性.

- lambda可以用更精简的代码创建匿名内部类.但是该匿名内部类实现的接口只能有一个抽象方法,否则无法使用!
- lambda表达式是编译器认可的,最终会将其改为内部类编译到class文件中

```
package lambda;

import java.io.File;
import java.io.FileFilter;

/**
 * JDK8之后java支持了lambda表达式这个特性
 * lambda表达式可以用更精简的语法创建匿名内部类，但是实现的接口只能有一个抽象
 * 方法，否则无法使用。
 * lambda表达式是编译器认可的，最终会被改为内部类形式编译到class文件中。
 *
 * 语法:
 * (参数列表)->{
 *      方法体
 * }
 */
public class LambdaDemo {

```

```
    public static void main(String[] args)
{
        //匿名内部类形式创建FileFilter
        FileFilter filter = new
FileFilter() {
            public boolean accept(File
file) {
                return
file.getName().startsWith(".");
            }
        };

        FileFilter filter2 = (File file)->{
            return
file.getName().startsWith(".");
        };

        //lambda表达式中参数的类型可以忽略不写
        FileFilter filter3 = (file)->{
            return
file.getName().startsWith(".");
        };

        /*
            lambda表达式方法体中若只有一句代码，
则{}可以省略
            如果这句话有return关键字，那么
return也要一并省略!
         */
```

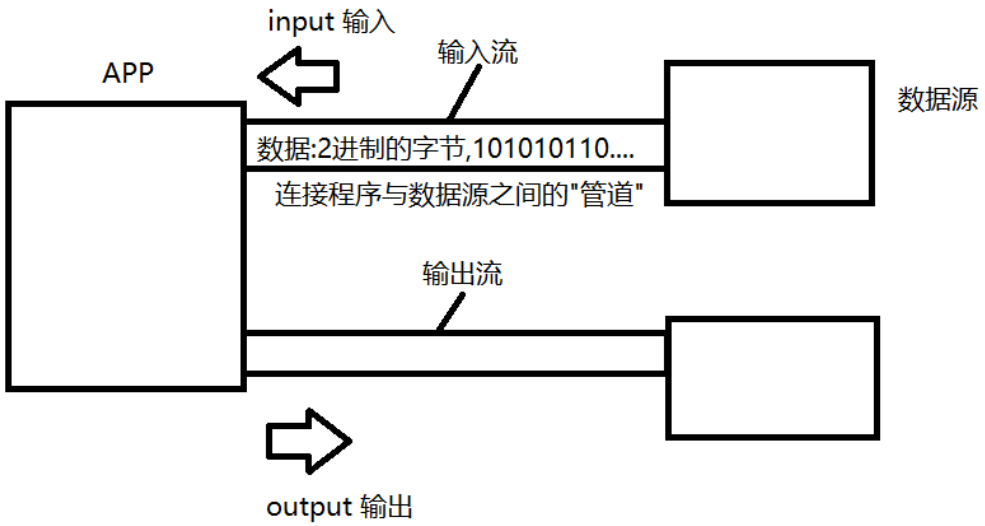
```
        FileFilter filter4 = (file)-
>file.getName().startsWith(".");
    }
}
```

JAVA IO

- java io可以让我们用标准的读写操作来完成对不同设备的读写数据工作.
- java将IO按照方向划分为输入与输出,参照点是我们写的程序.
- 输入:用来读取数据的,是从外界到程序的方向,用于获取数据.
- 输出:用来写出数据的,是从程序到外界的方向,用于发送数据.

  java将IO比喻为"流",即:stream. 就像生活中的"电流","水流"一样,它是以同一个方向顺序移动的过程.只不过这里流动的是字节(2进制数据).所以在IO中有输入流和输出流之分,我们理解他们是连接程序与另一端的"管道",用于获取或发送数据到另一端.

java将输入和输出命名为"流",英文stream



Java定义了两个超类(抽象类):

- java.io.InputStream:所有字节输入流的超类,其中定义了读取数据的方法.因此将来不管读取的是什么设备(连接该设备的流)都有这些读取的方法,因此我们可以用相同的方法读取不同设备中的数据
- java.io.OutputStream:所有字节输出流的超类,其中定义了写出数据的方法.

java将流分为两类:节点流与处理流:

- 节点流:也称为低级流.节点流的另一端是明确的,是实际读写数据的流,读写一定是建立在节点流基础上进行的.
- 处理流:也称为高级流.处理流不能独立存在,必须连接在其他流上,目的是当数据流经当前流时对数据进行加工处理来简化我们对数据的该操作.

实际应用中,我们可以通过串联一组高级流到某个低级流上以流水线式的加工处理对某设备的数据进行读写,这个过程也成为流的连接,这也是IO的精髓所在.

文件流

文件流是一对低级流,用于读写文件数据的流.用于连接程序与文件(硬盘)的"管道".负责读写文件数据.

文件输出流:java.io.FileOutputStream

```
package io;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * JAVA IO  Input&Output  输入和输出
 * java程序与外界交换数据是基于IO完成的，这里输入与输出一个负责读一个负责写
 * 输入:是从外界到我们写的程序的方向，是用来从外界获取信息的。因此是"读"操作
 * 输出:是从我们写的程序到外界的方向，是用来向外界发送信息的。因此是"写"操作
 *
 * java将IO比喻为"流",可以理解为是程序与外界连接的"管道",内部流动的是字节，并且
```

```
 * 字节是顺着同一侧方向顺序移动的。
 *
 * java.io.InputStream 输入流，这个类是所有字节输入流的超类，规定了所有字节输入
 * 流读取数据的相关方法。
 * java.io.OutputStream 输出流，这个类是所有字节输出流的超类，规定了所有字节输出
 * 流写出数据的相关方法。
 *
 * 实际应用中，我们连接不同的设备，java都专门提供了用于连接它的输入流与输出流，而
 * 这些负责实际连接设备的流称为节点流，也叫低级流。是真实负责读写数据的流。
 * 与之对应的还有高级流，高级流可以连接其他的流，目的是当数据流经它们时，对数据做某
 * 种加工处理，用来简化我们的操作。
 *
 *
 * 文件流
 * java.io.FileInputStream和FileOutputStream
 * 这是一对低级流，继承自InputStream和OutputStream。用于读写硬盘上文件的流
 *
 */
public class FOSDemo {
    public static void main(String[] args) throws IOException {

        //向当前目录下的demo.dat文件中写入数据
        /*
```

```
            FileOutputStream提供的常用构造器
            FileOutputStream(String path)
            FileOutputStream(File file)
         */
        //文件流创建时，如果该文件不存在会自动将其
创建(前提是该文件所在目录必须存在!)
        FileOutputStream fos = new
FileOutputStream("./demo.dat");
        /*
            void write(int d)
            向文件中写入1个字节，写入的内容是给定
的int值对应的2进制的"低八位"

            int值 1:
vvvvvvvv
            二进制:00000000 00000000
00000000 00000001
            demo.dat文件内容:
            00000000

         */
        fos.write(1);
        /*

vvvvvvvv
            00000000 00000000 00000000
00000010

            demo.dat文件内容
            00000001 00000010
```

```
         */
        fos.write(2);



        fos.close();
        System.out.println("执行完了!");


    }
}
```

文件输入流

```
package io;

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

/**
 * end 结尾
 * read 读
 *
 *
 * 文件字节输入流，用于从文件中读取字节
 */
public class FISDemo {
    public static void main(String[] args)
throws IOException {
```

```
        /*
            fos.dat文件内容
            00000001 00000011
         */
        FileInputStream fis = new
FileInputStream("fos.dat");
        /*
            int read()
            读取一个字节，并一int型返回。返回的整
数中读取的字节部分在该整数2进制的最后8位上
            如果返回值为整数-1,则表示流读取到了末
尾。对于读取文件而言就是EOF(end of file
            文件末尾)

            第一次调用read():
            int d = fis.read();
            fos.dat文件内容
            00000001 00000011
            ^^^^^^^^
            读取该字节

            返回int值时，2进制样子:
            00000000 00000000 00000000
00000001

^^^^^^^^
            |-----补充24个0(3字节)-----| 读取
的字节

            返回的int值d就是上述内容
```

```
         */
        int d = fis.read();
        System.out.println(d);

        /*
            第二次调用read()
            d = fis.read();
            fos.dat文件内容
            00000001 00000011
                     ^^^^^^^^
                     读取该字节

            返回int值时，2进制样子:
            00000000 00000000 00000000
00000011

^^^^^^^^
            |-----补充24个0(3字节)-----| 读取
的字节

            返回的int值d就是上述内容
         */
        d = fis.read();
        System.out.println(d);

       /*
            第三次调用read()
            d = fis.read();
            fos.dat文件内容
            00000001 00000011
```

```
                                        ^^^^^^^^
                                        文件末尾了

                返回int值时，2进制样子:
                11111111 11111111 11111111
11111111

^^^^^^^^
                |-----补充32个1(4字节，来表示-1)--
---|

                返回的int值d就是上述内容
             */
            d = fis.read();
            System.out.println(d);

            fis.close();
        }
}
```

文件复制

```
package io;

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

/**
```

```
 * 文件的复制
 */
public class CopyDemo {
    public static void main(String[] args)
throws IOException {
        //创建文件输入流读取原文件
        FileInputStream fis = new
FileInputStream("image.jpg");
        //创建文件输出流写入复制文件
        FileOutputStream fos = new
FileOutputStream("image_cp.jpg");

        int d;//保存每次读取到的字节
        /*
            原文件数据:
            11000011 10101010 00001111
11001100 00110011 ...
                     ^^^^^^^^
            d = fis.read();
            d:00000000 00000000 00000000
10101010
            fos.write(d);
            复制文件的数据:
            11000011 10101010
         */
        long start =
System.currentTimeMillis();//获取当前系统时间
的毫秒值(UTC时间)
        while((d = fis.read()) != -1) {
            fos.write(d);
```

```
        }

        long end =
System.currentTimeMillis();//获取当前系统时间
的毫秒值(UTC时间)
        System.out.println("复制完毕!耗时:"+
(end-start)+"ms");
        fis.close();
        fos.close();
    }
}
```