I/O流

一、文件操作

File类

作用：表示一个目录或一个文件，可以作为节点流参数，表示目录。

1、java中文件路径的表示方法

windows中表示： c:\sun.txt

java中表示： c:\\sun.txt

c:/sun.txt

2、File的构造方法

File类中没有无参构造

构造方法定义	构造方法描述
`File(File parent, String child)`	从父抽象路径名和子路径名字符串创建新的 `File` 实例。
File(String path)	根据字符串类型的path创建一个File对象
File(String parentPath,String childPath)	根据父路径字符串和子路径字符串创建一个File对象

案例：

```
    public static void main(String[] args) {
//        File(String path)
//        根据字符串创建一个File对象
        File f1 = new File("E:/files/file");
//        File(String parentPath,String childPath)
//        根据父路径和子路径共同创建一个file对象，最终结果表示 E:/files/file
        File f2 = new File("E:/files","file");
//        File(File parent, String child)
//
//        在参数f1表示的是 E:\files\file
//        最终文件表示路径 E:\files\file\hello
        File f3 = new File(f1,"hello");
    }
```

3.File类中常用的方法

方法名	作用	返回值
createNewFile()	创建一个文件，如果文件已存在不会覆盖	boolean
mkdir()	创建一个目录（文件夹）	boolean
delete()	删除一个文件或空目录	boolean
exists()	判断File对象表示的文件（目录）是否存在	boolean
getAbsolutePath()	获取文件的绝对路径	String
getName()	获取文件名	String
getParent()	获取文件所在的目录	String
listFiles()	如果File对象为目录，返回目录中所有的File对象	File[]
list()	返回当前目录下所有的文件，子目录	String[]
listFiles(FileFilter f)	获取满足文件过滤器条件的文件	File[]

案例：file类中常用方法

```
public class Test2 {
    public static void main(String[] args) throws IOException {
        File f1 = new File("E:/files/file/a.txt");
//          根据文件对象，在磁盘中创建指定的文件
        f1.createNewFile();
        File f2 = new File("E:/files1");
//       mkdirs(): 根据文件对象创建目录
        f2.mkdirs();
//
        File f3 = new File("abc/bcd","hello/a.txt");
        f3.createNewFile();
//          getPath():获取文件路径
        System.out.println(f3.getPath());
//          getParent():获取文件的父路径
        System.out.println(f3.getParent());
//          getName():获取文件的名字
        System.out.println(f3.getName());
//          getAbsolutePath():获取文件的绝对路径
        System.out.println(f3.getAbsolutePath());
//      isFile():表示是否是文件
        System.out.println(f3.isFile());
//          isDirectory():判断是否是目录
        System.out.println(f2.isDirectory());
    }
}
```

3.递归删除文件

```
public class Test3 {
    public static void main(String[] args) {
//          创建一个文件对象（目录）
        File f = new File("E:\\files\\dirs");
        deleteDir(f);
```

```
    }
//    递归删除文件
    public static void deleteDir(File dir){
//        1.获取该目录下的所有文件或子目录
        File[] files = dir.listFiles();
        for(File f:files){
//            判断当前file是否是一个目录
            if(f.isDirectory()){
                deleteDir(f);
            }
            f.delete();
        }
//        删除空目录
        dir.delete();

    }
```

4.根据文件过滤器，查找文件

案例：使用文件过滤器，筛选匹配的文件

```
public class Test4 {
    public static void main(String[] args) {
//        创建一个File对象 ，根据该File对象查找指定的文件
        File file = new File("E:\\fileTest");
        findFiles(file);
    }
//    查找指定目录下所有的.java文件
    public static void findFiles(File file){
        File[] files = file.listFiles(
//                FileFilter:文件过滤器
                new FileFilter() {
            @Override
            public boolean accept(File pathname) {
//                判断是否是文件夹，如果是返回值true
                if(pathname.isDirectory()){
                    return true;
                }
//                如果文件的后缀名是.java则直接返回
                return pathname.getName().endsWith(".java");
            }
        });
//        遍历返回的结果
        for(File f:files){
//            如果是指定的文件，则输出
            if(f.isFile()){
                System.out.println(f.getAbsolutePath());
            }
//            如果返回的是文件夹，则进行递归查找
            if(f.isDirectory()){
                findFiles(f);
            }
        }
    }
}
```

二、I/O流的分类和介绍

（一）I/O流

作用：将程序中的数据发送到外部，或将外部的数据读取到程序内部，使用IO流可以在不同的程序或容器中进行数据的交换。

位置：java.io包中

（二）相关概念：

1.I(Input) 输入：站在程序（代码）的角度,将外部的数据读取到程序内部，称之为输入流。

例如：java.io.InputStream

2.O(Output) 输出：站在程序的角度，将程序内部的数据输出到外部（磁盘文件，第三方程序，网络）

例如：java.io.OutputStream

3.流：用来传输数据的管道，类似与生活中水管，电线。

计算机数据的两大存储媒介

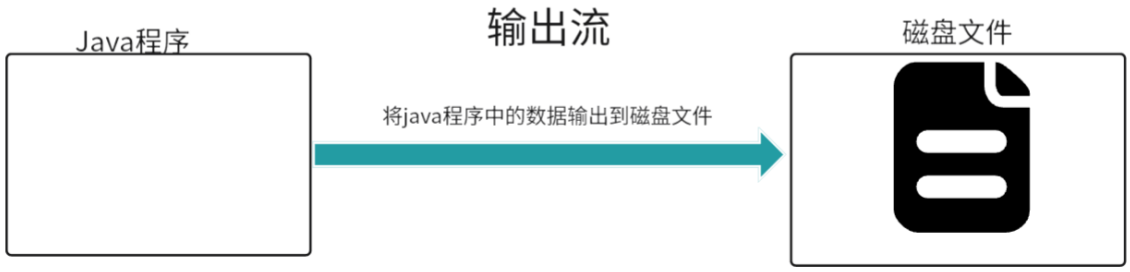
1.硬盘：硬盘是电脑中用来存放文件的数据存储设备，特点存储容量大，不受断电影响，可以持久化存储数据，但是数据传输速度慢。
2.内存：内存是计算机硬盘数据和CPU数据交换的中转站，存储程序中经常使用或运算修改的数据，特点：存储容量小，电子读写，断电后内存中的数据就会被清空，只能存储临时的文件或数据，数据传输速度极快，是硬盘传输速度的好几倍。

（三）流的分类

（1） 按照方向区分

1.输入流：将程序外部的数据读入到程序内部。 比如：Scanner;

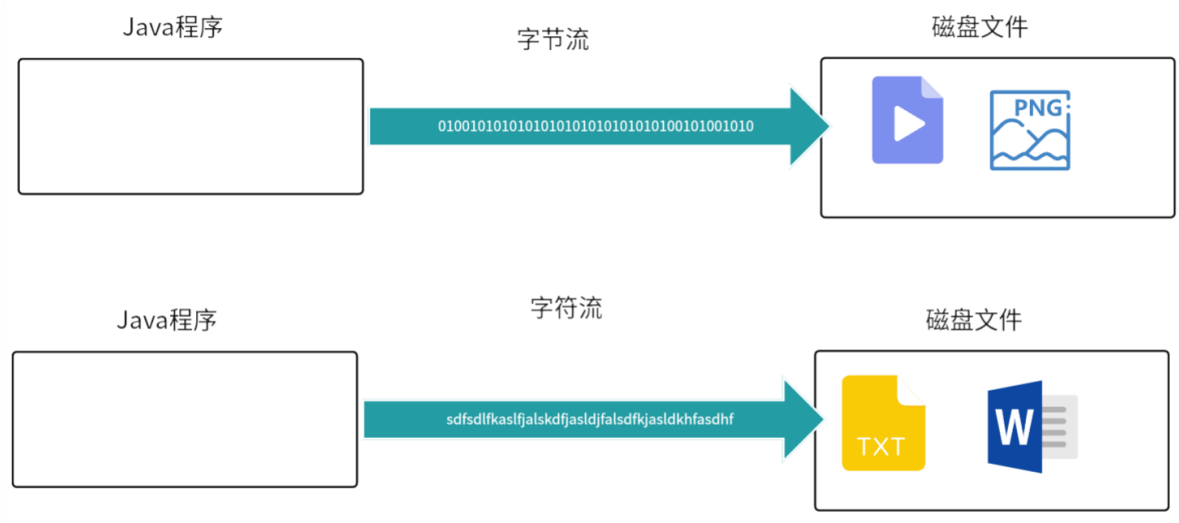
2.输出流：将程序内部的数据输出到外部。 比如.System.out.println();



（2）按照流的数据单位进行区分

1.字节流：以字节为单位进行处理和传输数据，可以处理所有类型的文件，比如文本，图片，音频，视频等等。

2.字符流：以字符为单位进行处理数据和传输数据，只可以处理文本类型的文件，比如，txt文件，.java



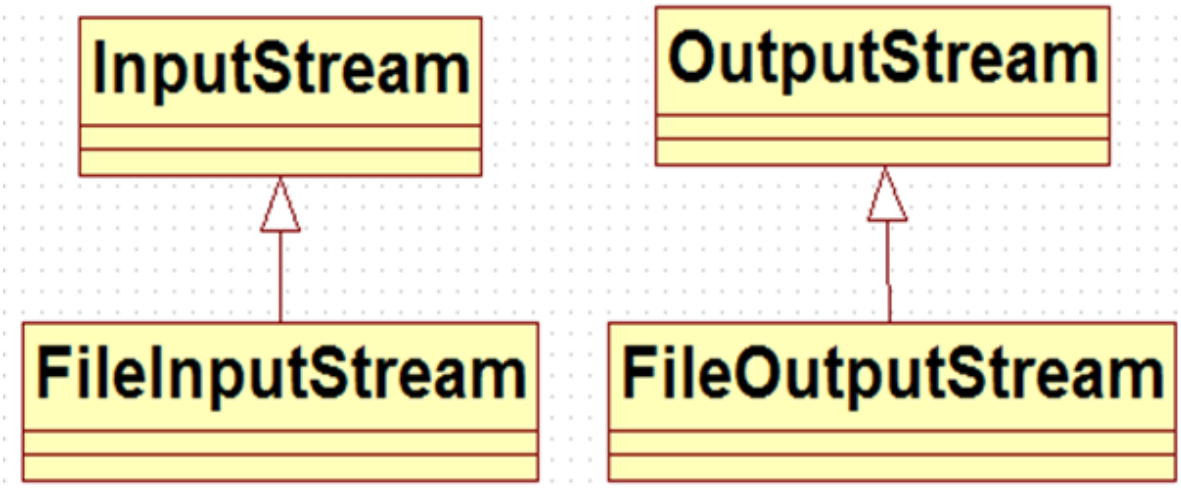
（3）按照流的功能

1.节点流：负责读写数据的流

2.过滤流：在字节流的基础上增加额外的功能，需要依赖节点流

三、字节流

特点：以字节为单位处理数据，可以处理所有类型的文件，图片，音频，视频，文本等



（一）OutputStream

特点：字节输出流父类（抽象类），可以将程序中数据，输出到程序外部（磁盘文件）

常用方法

方法	描述	返回值
write(int b)	写出一个字节的数据	void
writer(byte[] bs)	写出多个字节的数据	void
writer(byte[] bs,int off,int len)	写出多个字节的数据，从数组的off下标开始，往后len个字节	void
close()	释放资源，关闭流	void

例：向test.txt文件中写一个字符串Hello

```
public class OutPutStreamDemo1 {
    public static void main(String[] args) throws IOException {
        String hello = "hello world";
//        getBytes():将字符串转为对应的byte类型的数组
        byte[] bs = hello.getBytes();
//        创建一个字节输出流对象
        OutputStream out = new FileOutputStream("test.txt");
//        通过输出流对象.writer()向test.txt文件写入内容
        out.write(bs);
//        向文件写入内容后，释放资源，关闭流
        out.close();
    }
}
```

案例：向文件中追加内容

```
public class OutPutStreamDemo1 {
    public static void main(String[] args) throws IOException {
        String hello = " hello world";
//        getBytes():将字符串转为对应的byte类型的数组
        byte[] bs = hello.getBytes();
//        创建一个字节输出流对象,在字节输出流构造方法中添加一个参数，true,表示该文件是可以追加内容的
        OutputStream out = new FileOutputStream("test.txt",true);
//        通过输出流对象.writer()向test.txt文件写入内容
        out.write(bs);
//        向文件写入内容后，释放资源，关闭流
        out.close();
    }
}
```

（二）InputStream

特点：字节输入流父类（抽象类），可以将外部的数据读入到程序内部

常用方法：

方法	作用	返回值
read()	读取一个字节的数据并返回，如果数据读取完毕返回-1	int
read(byte[] bs)	从该输入流读取最多 `byte.length` 个字节的数据到字节数组。	int
`close()`	关闭流	void

例：从test.txt文件读取一个字节的数据

```
public class InputStreamDemo1 {
//    从test.txt文件中读取一个字节
    public static void main(String[] args) throws IOException {
//        创建一个输入流
        InputStream input = new FileInputStream("test.txt");
//        读取一个字节
        System.out.println(input.read());
        System.out.println(input.read());
        System.out.println(input.read());
```

```
        System.out.println(input.read());
        System.out.println(input.read());
        System.out.println(input.read());
        System.out.println(input.read());
        System.out.println(input.read());
        System.out.println(input.read());   //返回-1
//        关闭流
        input.close();
    }
}
```

例：读取test.txt中所有字节

```
//
    public static void main(String[] args) throws IOException {
//        创建一个输入流
        InputStream input = new FileInputStream("test.txt");
//        读取test.text中所有的字节
        int len = 0;
        while((len=input.read())!=-1){
            char c = (char)len;
            System.out.println(c);
        }
//        关闭流
        input.close();
    }
```

为了提高输入流的读取速度可以一次读取固定长度的字节

案例：

```
    public static void main(String[] args) throws IOException {
//        创建一个输入流
        InputStream input = new FileInputStream("test.txt");
//        读取test.text中所有的字节
//        创建一个长度为20的字节容器，一次取20个字节
        byte[] bs = new byte[20];

        int len = 0;
        while((len=input.read(bs))!=-1){
            for(int i=0;i<len;i++){
                System.out.print((char)bs[i]);
            }
            System.out.println();
        }
//        关闭流
        input.close();
    }
```

字节输入流输出流综合案例

案例：将test.txt文件中的内容复制到b.txt文件中

```
//    将test.txt文件中的内容复制到b.txt文件中
    public static void main(String[] args) throws IOException {
//        1.创建输入流，读取test.txt文件
```

```
        InputStream input = new FileInputStream("test.txt");
//        2.创建输出流，写文件
        OutputStream out = new FileOutputStream("b.txt");
        byte[] bs = new byte[10];
        int len = 0;
//        每次从输入流中读取指定长度的字节，将值复制给数组bs
        while((len = input.read(bs))!=-1){
//            将读取的内容，输出到指定文件中
            out.write(bs);
        }
//        关闭流
        input.close();
        out.close();
    }
```

（三）IO流中的异常处理

输入流异常处理

```
public class TestInputStreamDemo1 {
    public static void main(String[] args) {
        InputStream input = null;
        try{
//            创建输入流对象
            input  = new FileInputStream("test.txt");
//            进行读取操作
            int len = 0;
            byte[] bs = new byte[10];
            while((len = input.read(bs))!=-1){
                for(int i=0;i<bs.length;i++){
                    System.out.print((char)bs[i]);
                }
                System.out.println();
            }
        }catch (IOException e){
            e.printStackTrace();
        }finally {
//            关闭流
            try {
                input.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

输出流异常处理

例：

```
public class TestOutputStreamDemo {
    public static void main(String[] args) {
//
```

```
        OutputStream out = null;
        try{
//              1.创建流对象
            out = new FileOutputStream("b.txt");
//              2.进行写操作
            String str = "hello china";
            out.write(str.getBytes());

        }catch (IOException e){
            e.printStackTrace();
        }finally {
//              关闭流
            try {
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

IO流异常处理总结

```
1.在try语句之前创建流引用
2.在try语句块中，创建具体的流对象
3.进行读写操作
4.异常处理
5.在finally语句块中关闭流
```

（四）字节节点流

字节节点流：最纯粹的字节输入或输出流对象

（1）FileOutputStream

作用：如果系统允许，可以将数据写入到系统中的某个文件

构造方法

构造方法	描述
FileOutputStream(String name)	创建文件输出流，输出到指定的文件中（name）
FileOutputStream(String name,boolean append)	创建文件输出流，输出到指定的文件中（name）,可以拼接的输出流对象

（2）FileInputStream

作用：读取某个文件中的内容

构造方法

构造方法	描述
`FileInputStream(String name)`	从name所述的文件中读取数据，如果文件不存在则抛出文件未找到异常

（五）字节过滤流

（1）ObjectOutputStream 对象输出流

作用：如果系统允许，可以将数据写到系统中的某个文件中，支持8中基本数据类型和对象类型的数据

构造方法

构造方法	描述
`ObjectOutputStream(OutputStream out)`	构建Object过滤流对象，需要传入字节输出流对象，节点对象将数据写到目标位置

常用方法

方法名	作用	返回值
`write(int val)`	写入一个字节	void
`writeBoolean(boolean val)`	写入一个boolean类型的值	void
`writeChars(String str)`	写入一个字符串	void
`writeDouble(double val)`	写入一个double类型的值	void
`writeInt(int val)`	写入一个int类型的值	void

过滤流的开发步骤

```
1.创建节点流
2.基于节点流创建过滤流
3.进行读写操作
4.关闭流
```

例：使用字节过滤流向b.txt文件写入内容

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

        OutputStream out = null;
        ObjectOutputStream oout = null;
        try{
            //        1.创建节点流
            out = new FileOutputStream("b.txt",true);
            //        2.根据节点流创建过滤流
            oout = new ObjectOutputStream(out);
            //        3.进行读写操作
            List<String> list = new ArrayList<>();
            list.add("zhangsan");
```

```
            list.add("lisi");
            list.add("wangwu");
            list.add("zhaoliu");
//              将list对象写入文件中
            oout.writeObject(list);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            //        4.关闭流
            try {
                oout.close();
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }
```

(2) ObjectInputStream 对象输入流

作用：读取某个文件中的数据，读入的程序中，支持八种基本数据类型的，而且支持对象类型的数据

构造方法：

构造方法	描述
`ObjectInputStream(InputStream in)`	通过字节输入流，构建一个对象输入流

常用方法：

方法	描述	返回值
readShort()	读取一个short类型的数据	short
readInt()	读取一个int类型的数据	int
readDouble()	读取一个double类型的数据	double
readBoolean	读取一个boolean类型的数据	boolean
readObject()	读取一个对象类型的数据	Object

案例：从b.dat文件中读取多种数据类型的数据

```
public class TestObjectInputStreamDemo1 {
    public static void main(String[] args) {

        InputStream input = null;
        ObjectInputStream oinput = null;
        try{
            //        1.创建节点流
            input = new FileInputStream("b.dat");
            //        2.根据节点流创建过滤流
            oinput = new ObjectInputStream(input);
            //        3.进行读写操作
            double d1 = oinput.readDouble();
```

```
            boolean flag = oinput.readBoolean();
            int i = oinput.readInt();
            List<String> list = (List<String>) oinput.readObject();
            System.out.println(d1);
            System.out.println(flag);
            System.out.println(i);
            for(String str:list){
                System.out.println(str);
            }
        }catch (IOException | ClassNotFoundException e){
            e.printStackTrace();
        }finally {
//          4.关闭流
            try {
                oinput.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                input.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

(3) 字节缓冲流(了解)

BufferedInputStream 字节缓冲输入流

BufferedOutputStream 字节缓冲输出流

案例:

```
public class BufferedStreamDemo {
    public static void main(String[] args) {
//        创建节点流
        OutputStream out = null;
        BufferedOutputStream bout = null;
        try{
            out = new FileOutputStream("b.txt");
//            根据节点流封装过滤器
            bout = new BufferedOutputStream(out);
            out.write("hello BufferOutputSteam".getBytes());
        }catch (IOException e){
            e.printStackTrace();
        }finally {
//            关闭流
            try {
                bout.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                out.close();
            } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    }
}
```

四、对象的序列化【重点】

（一）相关概念

对象的序列化：将Java中的对象转为二进制字节的形式，保持到文件中。

对象的反序列化：将程序外部的文件，以字节流的方法读取到程序中，并且将字节流转为相关的对象。

（二）序列化授权

概念：在java中为了处于安全的考虑，并不是所有的类都可以在流中进行传输和共享，如果该对象需要以流的方式进行传输和共享，则需要进行序列化的授权。

授权方式：实现Serializable接口，此接口的源码如下，是一种表示型接口。

```
public interface Serializable{}
```

授权的作用：告知JVM此类对象可以在流中进行传输

未授权的结果：产生不可序列化的异常

例：将Student对象写入student.dat文件中，然后再读取出来

```
//对象序列化
public class StudentOutputStreamDemo {
//    将学生对象通过对象的序列化，写入到student.dat文件中
    public static void main(String[] args) {
        Student stu1 = new Student(1001,"张宁波","男",20,"河南");
```

```
//          创建节点流
        OutputStream out = null;
        ObjectOutputStream oout = null;
        try{
            out = new FileOutputStream("student.dat");
            //          根据节点流创建过滤器
            oout = new ObjectOutputStream(out);
//              进行对象的序列化操作
            oout.writeObject(stu1);
        }catch (IOException e){
            e.printStackTrace();
        }finally {
            try {
                oout.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    //对象的反序列化
    public class StudentInputStreamDemo {
//      反序列化：将student.dat中的数据恢复为java对象
    public static void main(String[] args) {
//          1.创建节点流
        InputStream input = null;
        ObjectInputStream oinput = null;
        try{
            input = new FileInputStream("student.dat");
            //          2.根据节点流创建过滤流
            oinput = new ObjectInputStream(input);
//              进行反序列化操作
            Student stu =(Student) oinput.readObject();
            System.out.println(stu);
        }catch (Exception e){
            e.printStackTrace();
        }finally {
            //          4.关闭流
            try {
                oinput.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                input.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
}
```

（三）对象序列化需要注意的

```
1.jdk中很多的类默认实现了Serializable接口，比如八种封装类，String
2.序列化对象时，必须要保证所有的属性均可序列化，如果该属性的类型为自定义类型，必须保证该属性对应的类一定要实现Serializable接口，否则会出现NotSerializableException异常
3.如果序列化对象为集合时，必须保证集合中的所有元素必须时可序列化的
4.使用transient修饰的属性可以让其不参与序列化过程，不会拆分为字节再流中进行传输。
5.使用static修饰的属性不能被序列化
```

五、try-with-resources处理异常

作用：自动的关闭需要关闭的流。

语法：

```
try(
    //创建需要关闭资源的对象，再次声明的流对象，可以自动进行关闭操作
){
    //进行读写操作
}catch(Exception e){
    //异常处理
}
```

案例：

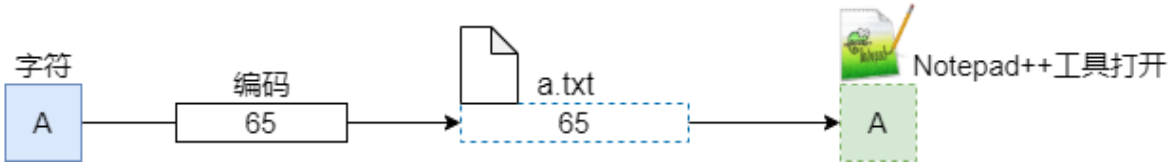
```
    public static void main(String[] args) {
        Address address = new Address("中国","河南","开封杞县","苏木乡花湖寨村","51000");
        Student stu1 = new Student(1001,"张宁波","男",20,address);
//        创建节点流
        try(
//                声明需要关闭的资源，这些资源再使用完毕之后会自动关闭
                OutputStream out = new FileOutputStream("student.dat");
                ObjectOutputStream oout = new ObjectOutputStream(out);
                ){
//            进行对象的序列化操作
            oout.writeObject(stu1);
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

注意：不是所有的资源都可以使用try-with-resoures机制，可以使用的资源必须要实现AutoCloseable接口

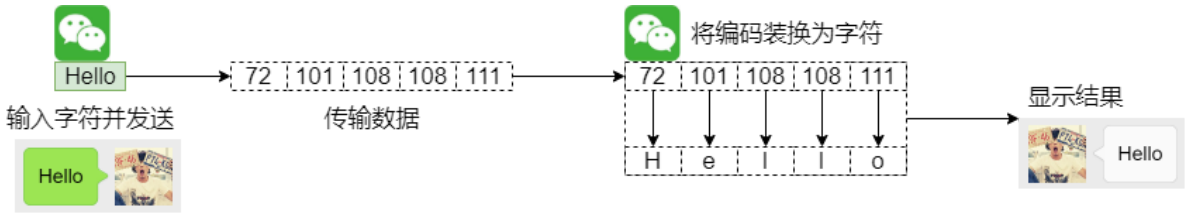
六、字符与乱码

（一）字符的存储形式和传输形式

在计算机中，任何数据都是以字节的方式存储在，字符也不例外，当我们在文件中存储一个字符时，实际存储时一个一段“编码”，该编码会映射（对应）的字符，在打开文件时，由打开文件的工具负责将此编码转化为字符。



计算机在传输字符时，依然以编码的形式进行传输，最后由目标在根据编码进行解析，将编码转为字符。



编码：将字符按照特定的字符编码方案，解析为对应的数字，

解码：将编码数字参照对应的字符编码方案，将数组解析为字符的过程

（二）字符集和编码方案

作用：明确字符和编码数字的对应关系与存储形式，一般情况下字符集只有一套存储形式，所以字符集和编码方式可以理解为同义词。

常用的字符集和编码方案

字符集	编码方式	字符容量	所占空间	所含字符
ASCII	单字节编码	128	1b	英文字母+标点符号+控制字符
GB2312	-	7445	2b	兼容ASCII+简体中文+中文符号
GBK	-	21886	1~2b	兼容ASCII+简体中文+中文符号
Unicode	UTF-8 UTF-16 UTF-32	144697	1~4b 2~4b 4b	兼容ASCII+所有管家的文字和符号

（三）乱码

> 字符在进行编码和解码的过程中出现的问题，无法将字节转化为原来的字符。

原因1：编码和解码对应的方案不同， 比如Linux默认使用UTF-8，Windows默认使用GBK,传输数据时就容易处理乱码的问题

原因2：字符的完整性被破坏，计算机是以字节为单位进行存储，由于字符可能占用多个字节，如果还是单字节的形式处理字符，那么一个字符的完整性就得不到保证，例：使用字节流向文件中输出一个中文字符，粗暴的只写一个char类型的字节。

乱码的案例：

```
    public static void main(String[] args) throws UnsupportedEncodingException {
        String oname = "测试";
//        编码：将字符串“测试”按照GBK编码方案进行编码
        byte[] bs = oname.getBytes("GBK");
//        解码：将字节数组 bs 按照 GBK解析解码
        String dname = new String(bs,"GBK");
        System.out.println(dname);

//        将字符串测试按照GBK进行编码
        byte[] bs1 = oname.getBytes("GBK");
//         解码：将bs1字节数组按照另外一种编码方法进行解码
        String dname1 = new String(bs1,"utf-8");
//        编码和解码的方案不一致，导致乱码
        System.out.println(dname1);
    }
```

如何解决乱码

```
//    乱码解决成功
    public static void main(String[] args) throws UnsupportedEncodingException {
        String str = "测试";
//        编码
        byte[] bs = str.getBytes("GBK");
//        使用错误的字符编码进行解码
        String errorStr = new String(bs,"ISO-8859-1");
        System.out.println(errorStr);

//        将乱码的字符按照错误的编码方案进行编码，还原原有的字节数组
        byte[] bs1 = errorStr.getBytes("ISO-8859-1");
//        将还原的字节数组，解码为原来的GBK
        String rigthStr = new String(bs1,"GBK");
        System.out.println(rigthStr);
    }

//    乱码解决失败案例：
    public static void main(String[] args) throws UnsupportedEncodingException {
        String str = "测试";
//        使用GBK进行编码
        byte[] bs = str.getBytes("GBK");
//        使用错误的字符编码方案进行解码
        String errorStr = new String(bs,"UTF-8");
        System.out.println(errorStr);

//        使用错误的字符编码方案进行还原byte[]
        byte[] bs1 = errorStr.getBytes("UTF-8");
//        在使用正确的字符编码进行解码，这里无法解决乱码问题，原因，utf-8采用的可边长的字符编码
        String newStr = new String(bs1,"GBK");
        System.out.println(newStr);
    }
```

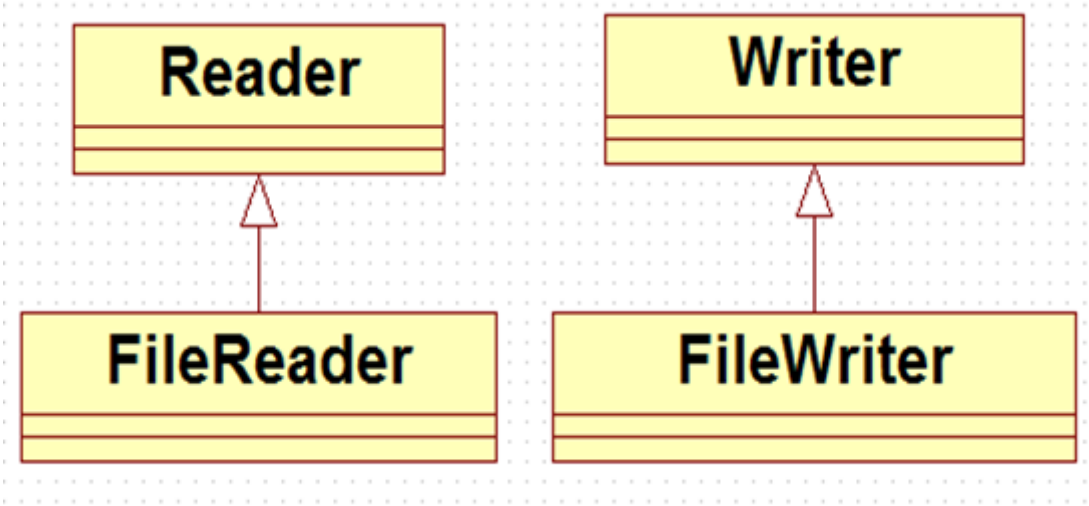
如何解决乱码

1.保证字符的完整性

2.保证编码和解码的方案一致性。

七、字符流

在流的内部保证一个字符的完整性，流的内容是一个个完整的字符。



（一）Writer(字符输出流)

用于写入字符流的抽象类，

常用方法：

方法名	作用	返回值
writer(char[] cs)	将char类型的数组，输出到指定文件	void
writer(String str)	将String字符串，传输到指定的文件	void

案例：

```
    public static void main(String[] args) throws IOException {
//        1.创建字符流对象
        Writer writer = new FileWriter("b.txt");
//        2.向文件写如字符
        writer.write("危楼高百尺");
        writer.write("手可摘星辰");
//        3.关闭流
        writer.close();
    }
```

（二）Reader（字符输入流）

常用方法：

方法名	描述	作用
read()	读一个字符	int
read(char[] c)	读多个字符，保持到字符数组c中	int
read(char[] c,int off,int leng)	读取多个字符，从开始下标到往后len个长度的字符	int

案例：

```
//        创建字符输入流
        Reader r = new FileReader("b.txt");
//          开始读
        int len = 0;
        while((len=r.read())!=-1){
            System.out.println((char)len);
        }
//          关闭流
        r.close();
    }
```

（三）字符流节点流

FileWriter:字符文件输出流

FileReader:字符文件输入流

（四）字符过滤流

字符流节点流在流中往文件写入内容时，不方便进行换行操作。同理，也无法从文件中进行一行一行的读取数据。

（1） BufferedWriter

概念：带缓冲8192的字符输出流，内部提供了一个newLine()方法，可以根据平台进行不同的换行。

（2）BufferedReader

概念：带缓冲8192的字符输入流，内部提供一个readLine(),可以一次读取一行的内容，读取完毕返回null

（3）字符过滤流的开发步骤

```
1.创建字符节点流
2.根据字符节点流创建字符过滤流
3.进行读写操作
4.关闭流
```

案例：

```
//通过字符过滤流向文件中进行一行一行的写操作
public static void main(String[] args) throws IOException {
//        1.创建字符节点流
```

```
        Writer writer = new FileWriter("b.txt");
//        2.根据字符节点流，创建字符过滤流
        BufferedWriter bw = new BufferedWriter(writer);
//        3.进行读写操作
        bw.write("静夜思");
        bw.newLine();
        bw.write("床前明月光");
        bw.newLine();
        bw.write("疑是地上霜");
        bw.newLine();
        bw.write("举头望明月");
        bw.newLine();
        bw.write("低头思故乡");
//        4.关闭流
        bw.close();
    }
//通过字符过滤流从文件中进行一行一行的读的操作
    public static void main(String[] args) throws IOException {
//        1.创建字符节点流
        Reader reader = new FileReader("b.txt");
//        2.根据字符节点流创建字符过滤流
        BufferedReader br = new BufferedReader(reader);
//        3.进行读写操作
        while(true){
            String str = br.readLine();      //每次从字符输入流中读取一行字符
            if(str==null){       //读取到文件末尾，返回null
                break;
            }
            System.out.println(str);
        }
//        4.关闭流
        br.close();
        reader.close();
    }
```

(4) PrintWriter

作用：带缓冲的可以将所有数据类型转换字符输出，支持换行，本身也是一个节点流

案例：

```
  public static void main(String[] args) throws FileNotFoundException {
//        使用字符打印流，往文件c.txt中打印一首诗
        PrintWriter pw = new PrintWriter("c.txt");
//        println:向文件中打印一行内容，自带换行
        pw.println("悯农");
        pw.println("锄禾日当午");
        pw.println("汗滴禾下土");
        pw.println("谁知盘中餐");
        pw.println("粒粒皆辛苦");

        pw.close();
    }
```

(5) 桥转换

OutputStreamWriter/InputStreamReader

作用：将字节流转为字符流，在转换的过程中可以自己定义编码方案的，默认的字符流使用的编码方案和操作系统有关，windows系统默认字符编码方案 GBK,

Bridge: 读取文件--- 读取字节 ----转换字符 ----自动分配编码，手动分配

使用场景：按照指定编码读取文本文件

```
 public static void main(String[] args) throws IOException {
//         字符流读取文件默认的编码方案：windows --- GBK
//         创建一个节点流
        InputStream input = new FileInputStream("c.txt");    //字节流
//         桥转换流  将字节流转为字符流，转换时可以指定对应的字符编码方案
        InputStreamReader isr = new InputStreamReader(input,"UTF-8");    //将读取的字节转为字符
//         将字符流包装为字符过滤流
        BufferedReader br = new BufferedReader(isr);
        while(true){
            String str = br.readLine();
            if(str==null){
                break;
            }
            System.out.println(str);
        }
//         关闭流
        br.close();
        isr.close();
        input.close();
    }
```

场景2：按照指定编码向文件写入文本

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

//         1.创建字节节点流
        OutputStream out = new FileOutputStream("d.txt");
//         2.根据字节节点流包装字符流(桥转换)
        OutputStreamWriter ow = new OutputStreamWriter(out,"UTF-8");
//         3.根据字符流包装字符过滤流
        BufferedWriter bw = new BufferedWriter(ow);
//         4.使用字符过滤流进行读写操作
        bw.write("望庐山瀑布");
        bw.newLine();
        bw.write("日照香炉生紫烟");
        bw.newLine();
        bw.write("遥看瀑布挂前川");
        bw.newLine();
        bw.write("飞流直下三千尺");
        bw.newLine();
        bw.write("疑是银河落九天");
//         5.关闭流
        bw.close();
        ow.close();
        out.close();
```

```
    }
```

桥转换的步骤总结

```
1.创建一个节点流读取文件
2.根据节点流，包装字符流（桥转换），可以指定对应的字符编码方案
3.根据字符流，包装字符过滤流，
4.使用字符过滤流进行读写操作
5.关闭流
```

八、总结

类名	作用
InputStream	字节输入流的父类
OutputStream	字节输出流的父类
FileInputStream	文件输入流（字节输入节点流）
FileOutputStream	文件输出流（字节输出节点流）
BufferedInputStream	字节输入缓冲过滤流
BufferedOutputStream	字节输出缓冲过滤流
ObjectInputStream	对象输入过滤流（带缓冲，可读8种基本数据类型和对象）
ObjectOutputStream	对象输出过滤流（带缓冲，可写8种基本数据类型和对象）
PrintStream	字节打印流（以字节为单位将八种基本数据类型和对象类型转发字符串输出流目标路径）
Reader	字符输入流父类
Writer	字符输出流的父类
FileReader	字符输入节点流
FileWriter	字符输出节点流
BufferedReader	字符输入缓冲过滤流（内部提供了一个readLine方法，可以一次读一行文字）
BufferedWriter	字符输出缓冲过滤流（内部提供了一个newLine()方法，可以进行换行）
PrintWriter	字符打印流（以字符为单位，将任意类型转为字符串并输出到文件）
InputStreamReader	字符桥转换输入流（可以将字节输入流转换为对应字符输入流，可以自定义编码格式）
OutputStreamWriter	字符前转换输出流（可以将字节输出流转为字符输出流，可以自定义编码方案）
File	文件类，（表示磁盘一个文件，可以File类对象操作磁盘文件，可以作为流对象的参数）

IO流常规操作

1. 创建节点流

```
InputStream is = new FileInputStream("a.dat");       //创建字节节点流
```

2. 封装过滤流

```
ObjectInputStream ois = new ObjectInputStream(is);
```

3. 读写操作

```
int i = ois.readInt();       //读取一个整数
double d = ois.readDouble();     //读取一个double类型的值
Student stu =(Student) ois.readObject();         //读取一个对象
```

4. 关闭流

```
ois.close();
```