目 录

第一章　I/O 编程

1.1　字节流

`InputStream` 和 `OutputStream` 是所有字节输入输出流的父类，都为抽象类。

1.2　`FileInputStream` 和 `FileOutputStream`

文件字节流，是节点流。

1.2.1　`FileInputStream`

写入一个文件。

```
InputStream in = new FileInputStream("E://a.txt");//文件不存在，抛出异常
```

Tips

`InputStream`

1. `int read()`：读一个字节，返回值为读到的字节数据。每读一次，指针都会向后移动一个字节。如果该方法返回-1 表示读取结束。
2. `int read(bytes[] bs)`：从文件读取字节，把数组 `bs` 读满，返回值为读到的字节个数。返回-1 表示读取结束。
3. `int read(bytes[] bs, int start, int length)`：从文件读取字节，尝试读满数组中的一段（把数组的从 `start` 开始长 `length` 的区域读满），返回值为读到的字节个数。返回-1 表示读取结束。

1.2.2　`FileOutputStream`

`FileInputStream`：读取一个文件。

```
String filename = "d:\\a.txt";
OutputStream out = new FileOutputStream(filename);//要记得把文件名传入构造器，不写true表示覆盖原来的文件
OutputStream out = new FileOutputStream(filename,true);//要记得把文件名传入构造器，写true表示在原来的文件进行追加
out.write('A');//写入一个字节
out.close();
```

Tips

`OutputStream`

字符串转字节数组：`str.getBytes();`

1. `write(int a)`：往文件写出一个字节
2. `write(byte[] bs)`：将字节数组 `bs` 所有的数据写入文件
3. `write(byte[] bs, int start, int length)`：将字节数组 `bs` 从 `start` 开始往文件写入 `length` 个字节。

异常处理

```
String filename = "d:\\a.txt";
OutputStream out = null;
//OutputStream out = new FileOutputStream(filename,true);//要记得把文件名传入构造器，写true表示在原来的文件进行追加
try{
```

```
    out = new FileOutputStream(filename);//要记得把文件名传入构造器，不写true表示覆盖原来的文件
    out.write('A');//写入一个字节
}catch(Exception e){
    System.out.println(e.getMessage());
}finally{
    try{
        out.close();
    }catch(Exception e1){
        System.out.println(e1.getMessage());
    }
}
```

1.2.3 案例：文件拷贝

从 src 拷贝到 dst

版本 1

```
//文件拷贝
//long a = System.nanoTime();//纳秒
//long b = System.nanoTime();
//System.out.println((b-a)/1e9);
public static void fileCopy1(String src,String dst){
    InputStream fis = null;
    OutputStream fos = null;
    try {
        fis = new FileInputStream(src);
        fos = new FileOutputStream(dst);
        byte[] bs = new byte[1024*4];//一页4个kb
        while(true){
            int len = fis.read(bs);
            if(len == -1){
                break;
            }
            fos.write(bs,0,len);
        }
    }catch(Exception e1){
        System.out.println(e1.getMessage());
    }finally {
        try{
            fis.close();
        }catch(Exception e2){
            System.out.println(e2.getMessage());
        }
        try{
            fos.close();
        }catch(Exception e2){
            System.out.println(e2.getMessage());
        }
    }
```

```
}
```

1.2.4 过滤流

对于 int、long 这类的无法直接转为字节数组，需要使用过滤流来完成读写。

`DataOutputStream`，过滤流必须由节点流来构建，即：

```
OutputStream fos = null;
DataOutputStream dfos;
try {
    fos = new FileOutputStream("a.txt");
    dfos = new DataOutputStream(fos);
} catch (Exception e) {
    System.out.println(e.getMessage());
}finally{
    try{
        dfos.close();
        fos.close();
    }catch(Exception e1){
        System.out.println(e1.getMessage());
    }
}
```

Tips

`DataOutputStream`：往文件写入 8 种基本类型和 `String`

例如：写入 `long`， `dfos.writeLong(10l)`

1.2.5 I/O 操作流程

1. 创建节点流；
2. 包装过滤流；
3. 读写数据
4. 关闭流

1.2.6 缓冲流

缓冲流（`BufferedOutputStream/BufferedInputStream`）为过滤流。

作用：利用一个缓冲区，提高 I/O 效率，减少访问磁盘的次数。

`flush()` 方法是将缓存区文件内容写入文件中。写方法与文件输出流一样。

```
OutputStream fos = null;
BufferedOutputStream bos;
try {
    fos = new FileOutputStream("a.txt");
    bos = new BufferedOutputStream(fos);
    bos.writeLong(10l);//数据写入缓冲流
    bos.flush();//清空缓冲区，将缓存区文件内容写入文件中。
}
```

```
9 //catch和关闭流略
```

1.2.6.1 文件拷贝

使用缓冲流拷贝文件

```
1  //文件拷贝2
2  //加字节数组减少了I/O操作
3  //加缓冲流减少了访问磁盘操作
4  public static void fileCopy2(String src, String dst) {
5      InputStream fis = null;
6      BufferedInputStream bis = null;
7      OutputStream fos = null;
8      BufferedOutputStream bos = null;
9      try {
10         fis = new FileInputStream(src);
11         bis = new BufferedInputStream(fis);
12         fos = new FileOutputStream(dst);
13         bos = new BufferedOutputStream(fos);
14         byte[] bs = new byte[1024 * 4];
15         while (true) {
16             int len = bis.read(bs);
17             if (len == -1) {
18                 break;
19             }
20             bos.write(bs, 0, len);
21         }
22     } catch (Exception e1) {
23         System.out.println(e1.getMessage());
24     } finally {
25         try {
26             bos.close();
27             fos.close();
28             bis.close();
29             fis.close();
30         } catch (Exception e2) {
31             System.out.println(e2.getMessage());
32         }
33     }
34 }
```

1.2.7 ObjectOutPutStream/ObjectInputStream

过滤流，增强了缓冲区功能，增强了读写 8 种数据类型和字符串功能。

增强了读写对象的功能：`readObject()`从流中读取一个对象，`writeObject(Object obj)`向流中写入一个对象。

Tips

通过流传输对象：对象的序列化。

只有实现了`Serializable`接口的对象才能序列化。`Serializable`接口中没有方法。实现该接口的类不需要添加额外的方法。

transient 关键字为属性的修饰符，用该关键字修饰的属性属于临时属性，不参与对象的序列化。（往文件写时不会存该属性，从文件读取时该属性置为其对应的默认值。）

1.2.7.1 读取整个文件

ObjectInputStream 的读取方法 readObject() 返回值不是数，因此无法通过-1 来判断是否读完文件，文件读取完毕后，该方法会抛出一个 EFOException 异常，可以通过捕获该异常来结束读取文件。

```
FileInputStream fis = null;
ObjectInputStream ois = null;
try{
    fis = new FileInputStream("a.txt");
    ois = new ObjectInputStream(ois);
    try{
        while(true){
            System.out.println(ois.readObject());
        }
    }catch(EFOException e){
        System.out.println("文件读取结束！");
    }
}catch(Exception e1){
    System.out.println(e1.getMessage());
}finally{
    ois.close();
    fis.close();
}
```

fis.available(); 返回源文件包含的字节数。

1.2.7.2 反序列化

如果该对象的类实现了 Serializable 接口，反序列化时无需通过该类的构造方法。

如果该对象的类没有实现 Serializable 接口，反序列化时需通过该类的构造方法（必须是无参的构造方法，否则会抛出异常）。

1.2.8 自定义序列化规则

实现 Externalizable 接口，对该接口的两个方法进行重写。

```
class A implements Externalizable {
    int age;
    String name;

    public A() {
    }
    //Externalizable 方法
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeUTF(name);
```

```
        out.writeInt(age);
    }

    @Override
    public void readExternal(ObjectInput in) throws Exception {
        name = in.readUTF();
        age = in.readInt();
    }
}
```

注：`readExternal` 方法抛出的异常为 `throws IOException ClassNotFoundException`

1.2.9 重复序列化

如果对一个对象重复多次序列化后存入文件，则文件只会存入该对象第一次序列化时所有的信息，后边的都存的该对象的相关引用。

导致的问题，如果先存入该对象，但该对象某些属性改变后，想继续往文件写入该对象，存入的依旧是相关引用。

解决方法：创建一个与当前对象参数完全一样的对象，只改变需要改变的属性。

`clone()` 方法是 `Object` 类的，因此所有的类都包括该方法，但该方法是 `protected`，使用类的 `clone()` 方法需要在该类中对其进行重写。

对象克隆其对应的类还需实现 `Cloneable` 接口，该接口没有方法。

```
class A implements Cloneable{
    int age;
    String name;

    public A() {
    }

    //其他一些属性和方法
    //仅仅将权限改为public
    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

具体用法：（`clone` 方法抛出的异常为受查异常，需显示处理）

```
A a = new A();
Object o;
A b;
try {
    o = a.clone();
    b = (A)(o);
}catch(Exception e){
    System.out.println(e.getMessage());
```

```
}
```

该拷贝为浅拷贝，如果原型对象成员变量是值类型，将复制一份给克隆对象，如果原型对象的成员变量是引用类型，则将该引用复制一份给克隆对象。改变某一对象中的引用类型数据，另一个也会随之改变。（对于 String、Integer 这些不可变对象除外。）

深拷贝会创造一个一模一样的对象，新对象与原对象不共享内存。

实现方法：使用序列化先将对象写入流中，然后再从流中读入到新对象中。

1.2.10 ByteArrayOutputStream

该流不会写入文件中，而是在虚拟机上，也就是在内存上。

深拷贝

```
class B implements Cloneable, Serializable {
    //其他的属性和方法
    @Override
    public Object clone() throws CloneNotSupportedException {
        Object o = null;
        ByteArrayOutputStream baos = null;
        ObjectOutputStream oos = null;
        ByteArrayInputStream bais = null;
        ObjectInputStream ois = null;
        try {
            baos = new ByteArrayOutputStream();
            oos = new ObjectOutputStream(baos);
            oos.writeObject(this);//把文件写入baos
            byte[] bs = baos.toByteArray();//将其转为字节数组,放的是当前对象的全部
                字节

            bais = new ByteArrayInputStream(bs);
            ois = new ObjectInputStream(bais);
            o = ois.readObject();

        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                oos.close();
                baos.close();
                ois.close();
                bais.close();
            } catch (Exception e1) {
                System.out.println(e1.getMessage());
            }
        }
        return o;
    }
}
```

1.3 字符流

使用字符流可以解决字符的编码问题。字符的编码方式和解码方式不统一，可能会造成乱码问题。

把字符转为字节码称为编码，把字节码转为字符称为解码。

常见的编码方式：ASCII，ISO-8859-1（西欧），GB2312（简体中文，字符集较小），GBK（简体中文，字符集多于 GB2312，兼容 GB2312），Big5（繁体中文），Unicode（全球统一编码，变种 UTF-16 一个字符 16 位，两个字节；UTF-8（变长，最常见的））

无论采用何种编码，其都向上兼容 ASCII 码。

1.3.1 字符流的输入输出流

`Writer` 和 `Reader` 是字符流的父类，属于抽象类。

`FileReader` 和 `FileWriter` 是文件字符流。

`BufferedReader` 和 `BufferedWriter` 是过滤流。缓冲

Tips

`BufferedReader` 有一个方法 `readline()`，每次从文件中读取一行，读到换行符为止，返回值为字符串。当返回为 null 时，表示文件读取结束。

`PrinterWriter` 缓冲 过滤流。

Tips

`PrinterWriter` 有一个方法 `println()`，每次向文件中写入一行。

1.3.2 字节流转字符流

`OutputStreamWriter`（桥转换类）将字节流转换为字符流。`InputStreamWriter`

```
OutputStream fos = new FileOutputStream("a.txt");
Writer osw = new OutputStreamWriter(fos);//使用字节输出流构造出字符输出流
```

Tips

在使用桥转换功能时，可以指定编码解码方式。

```
OutputStream fos = new FileOutputStream("a.txt");
Writer osw = new OutputStreamWriter(fos,"GBK");//指定编码方式为GBK
```

1.4 `file` 类

IO 流：对文件中的内容进行操作

File 类：对文件自身进行操作，例如创建文件，删除文件，文件的重新命名，文件的权限设置等等。不能访问文件内容。

File 对象可以代表磁盘上的文件或目录。

```
File f = new File("a.txt");//只是创建了一个File对象，并没有在磁盘上创建文件
try{
    f.createNewFile();//把上边的File对象写入磁盘
}catch(Exception e){
```

```
    System.out.println(e.getMessage());
}
f.delete();//把文件从磁盘上删除（彻底删除，不会在回收站）

//在磁盘上创建目录
f.mkdir();
//删掉目录
f.delete();//如果目录不为空，则无法删掉，只能删掉空目录

//重命名
File f1 = new File("a1.txt");
f.renameTo(f1);//该方法的参数是File对象，成功返回true，失败返回false

//设置文件为只读
f.setReadOnly();

//获取文件名:含扩展名，文件夹名
f.getName();

//判断一个文件或目录是否存在
f.exists();

//获得绝对路径
f.getAbsolutePath();

//若f表示的是目录，则获取当前文件下所有的文件、文件夹，返回值类型为File[]
f.listFiles();

//判断File对象是否为文件，返回值为false是可能文件不存在，要注意这点
f.isFile();

//判断File对象是否为目录，返回值为false是可能目录不存在，要注意这点
f.isDirectory();
```

删除目录

```
//若某个目录下有子文件，需使用递归的方法删除
public static void delDir(File dir){
    File[] fs = dir.listFiles();
    for (File file: fs) {
        if (file.isFile()){//file是文件则直接删掉
            file.delete();
        }
        if (file.isDirectory()){//file是目录则删掉目录
            delDir(file);
        }
    }
    dir.delete();//将dir目录删除掉
}
```

1.4.1 文件过滤器

`public File[] listFiles(FileFilter filter);`

`FileFilter` 是一个接口，需要自己实现。

```
File dir = new File("E:/corejava");
File[] fs = dir.listFiles(new FileFilter(){
    @Override
    public boolean accept(File pathname){
        if(pathname.isFile()) return true;//是文件就返回true
        return false;//如果希望结果保存在fs中，返回true，否则返回false
    }
});
```

列出目录所有的.java 文件

```
public static void listJavaFiles(File dir){
    File[] ret = dir.listFiles(new FileFilter() {
        @Override
        public boolean accept(File pathname) {
            if (pathname.isDirectory()) return true;
            if (pathname.isFile()){
                String name = pathname.getName();
                if (name.endsWith(".java")){
                    return true;
                }
            }
            return false;
        }
    });
    for (File f: ret) {
        if (f.isFile()) System.out.println(f.getAbsolutePath());
        if (f.isDirectory()) listJavaFiles(f);
    }
}
```

第二章　NIO

`Buffer`：缓冲区

```
import java.nio.*;

ByteBuffer buffer = ByteBuffer.allocate(20);//创建了一个容量为20字节的Buffer缓冲对象

byte[] bs = new byte[20];
ByteBuffer buffer2 = ByteBuffer.wrap(bs);//创建方式2
```

`Channel`：通道

写数据

```
FileOutputStream fos = new FileOutputStream("a.txt");//文件输出流
FileChannel channel = fos.getChannel();//建立管道

ByteBuffer buffer = ByteBuffer.wrap("hello".getBytes());//将需要写的数据写入缓冲区
channel.write(buffer);//用管道把缓冲区写入文件
channel.close();
```

读数据

```
FileInputStream fis = new FileInputStream("a.txt");//文件输入流
FileChannel channel = fis.getChannel();//建立管道

ByteBuffer buffer = ByteBuffer.allocate(20);
while(true){
    int len = channel.read(buffer);//将需要读的数据读入缓冲区
    if(len == -1){
        break;
    }
    buffer.flip();//buffer中有三个指针，position,limits,capacity,该方法是将position置为0，即从写模式切换为读模式
    while(buffer.hasRemaining()){//如果buffer中还有元素未读，就继续读
        //从position读到limits
        //buffer.get();//还有其他get方法，例如getLong()
        System.out.println((char)(buffer.get()));
    }
    buffer.clear();//把limits，position置为0，即从读模式切换为写模式
}

buffer.close();
channel.close();
fis.close();
```

字节流转字符流

```
//方式1
ByteBuffer bb = ByteBuffer.wrap("你好".getBytes());
```

```
3
4  //方式2
5  Charset cs = Charset.forName("GBK");
6  ByteBuffer bb2 = cs.encode("你好");
7
8  CharBuffer cb = cs.decode(bb2);
```

2.0.1 文件拷贝

文件拷贝

```
1  //文件拷贝3
2  //该算法效率低于上边的拷贝2
3  public static void fileCopy3(String src, String dst) {
4      FileInputStream fis = null;
5      FileOutputStream fos = null;
6      ByteBuffer buffer = ByteBuffer.allocate(1024);
7      FileChannel channel1 = null;
8      FileChannel channel2 = null;
9      try {
10         fis = new FileInputStream(src);
11         fos = new FileOutputStream(dst);
12         channel1 = fis.getChannel();
13         channel2 = fos.getChannel();
14         while (true) {
15             int a = channel1.read(buffer);
16             if (a == -1) {
17                 break;
18             }
19             buffer.flip();//写模式变为读模式
20             channel2.write(buffer);
21             buffer.clear();//读模式变为写模式
22         }
23     } catch (Exception e) {
24         System.out.println(e.getMessage());
25     } finally {
26         try {
27             channel2.close();
28             channel1.close();
29             fos.close();
30             fis.close();
31         } catch (Exception e1) {
32             System.out.println(e1.getMessage());
33         }
34     }
35 }
```

```
1  //文件拷贝4
2  public static void fileCopy4(String src, String dst) {
3      FileInputStream fis = null;
```

```
    FileOutputStream fos = null;
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    FileChannel channel1 = null;
    FileChannel channel2 = null;
    try {
        fis = new FileInputStream(src);
        fos = new FileOutputStream(dst);
        channel1 = fis.getChannel();
        channel2 = fos.getChannel();
        MappedByteBuffer map = channel1.map(FileChannel.MapMode.READ_ONLY, 0,
            channel1.size());//虚拟内存映射
        channel2.write(buffer);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    } finally {
        try {
            channel2.close();
            channel1.close();
            fos.close();
            fis.close();
        } catch (Exception e1) {
            System.out.println(e1.getMessage());
        }
    }
}
```

```
//文件拷贝5
//性能最高
public static void fileCopy5(String src, String dst) {
    FileInputStream fis = null;
    FileOutputStream fos = null;
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    FileChannel channel1 = null;
    FileChannel channel2 = null;
    try {
        fis = new FileInputStream(src);
        fos = new FileOutputStream(dst);
        channel1 = fis.getChannel();
        channel2 = fos.getChannel();
        channel1.transferTo(0,channel1.size(),channel2);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    } finally {
        try {
            channel2.close();
            channel1.close();
            fos.close();
            fis.close();
        } catch (Exception e1) {
            System.out.println(e1.getMessage());
```

```
            }
        }
}
```