

一，打印流：

//打印刘：字节流printstream，字符流printwriter

```
FileOutputStream fos = null;
try {
    fos = new FileOutputStream(new File("print.txt"));

} catch (Exception e) {
    // TODO: handle exception
    e.printStackTrace();
}
```

//创建打印输出流，设置为自动刷新模式，写入换行符或字节‘\n’时都会刷新输出缓冲区

```
PrintStream pStream = new PrintStream(fos, true);
if (pStream != null) {
    //把标准输出流（控制台输出）改成文件输出
    //输出到一个新的打印台的位置
    System.setOut(pStream);
}
for (int i = 0; i <= 255; i++) {
    //输出ASCII字符
    System.out.print((char)i);
    if (i % 30 == 0) {
        //每30个数据换一行
        System.out.println();
    }
}
pStream.close();
```

二，数据流：

处理流之五：数据流（了解）

- 为了方便地操作Java语言的基本数据类型的数据，可以使用数据流。
- 数据流有两个类：(用于读取和写出基本数据类型的数据)
 - DataInputStream 和 DataOutputStream
 - 分别“套接”在 InputStream 和 OutputStream 节点流上
- DataInputStream中的方法

<u>boolean readBoolean()</u>	<u>byte readByte()</u>
<u>char readChar()</u>	<u>float readFloat()</u>
<u>double readDouble()</u>	<u>short readShort()</u>
<u>long readLong()</u>	<u>int readInt()</u>
<u>String readUTF()</u>	<u>void readFully(byte[] b)</u>
- DataOutputStream中的方法
 - 将上述的方法的read改为相应的write即可。

三，对象流

1，对象的序列化

- ①对象的序列化机制：允许把内存中的java对象转换成为与平台无关的二进制流，从而允许把这种二进制流持久的保存在磁盘上，或者通过网络将这种二进制流传输到另一个网络节点，当其他程序获取了这种二进制流，就可以恢复成为原来的java对象
- ②序列化的好处在于可将任何实现了serializable接口的对象转化为字节数据，使其在保存和传输的时候可以被还原
- ③序列化是rmi（remote method invoke——远程方法调用）过程的参数和返回值都必须实现的机制，而rmi是javaee的基础，因此序列化机制是javaee平台的基础
- ④如果需要对某一个对象支持序列化机制，则必须让其类是可序列化的，为了让某个类是可序列化的，该类必须实现如下俩个接口之一：serializable和externalizable

处理流之六：对象流

● ObjectInputStream和ObjectOutputStream

- 用于存储和读取对象的处理流。它的强大之处就是可以把Java中的对象写入到数据源中，也能把对象从数据源中还原回来。
- 序列化(Serialize): 用ObjectOutputStream类将一个Java对象写入IO流中
- 反序列化(Deserialize): 用ObjectInputStream类从IO流中恢复该Java对象
- ObjectOutputStream和ObjectInputStream不能序列化static和transient修饰的成员变量

要求实现序列化的类:

- ①要求此类是可序列化的，实现serializable接口
- ②要求类的属性同样的要实现serializable接口
- ③提供一个版本号，private static final long serialVersionUID
- ④使用static或transient修饰的属性，不可实现序列化

四，随机访问流

RandomAccessFile 类

- RandomAccessFile 类支持“随机访问”的方式，程序可以直接跳到文件的任意地方来读、写文件
 - 支持只访问文件的部分内容
 - 可以向已存在的文件后追加内容
- RandomAccessFile 对象包含一个记录指针，用以标示当前读写处的位置。RandomAccessFile 类对象可以自由移动记录指针：
 - **long getFilePointer():** 获取文件记录指针的当前位置
 - **void seek(long pos):** 将文件记录指针定位到 pos 位置

RandomAccessFile 类

构造器

- public **RandomAccessFile**(File file, String mode)
- public **RandomAccessFile**(String name, String mode)

创建 **RandomAccessFile** 类实例需要指定一个 mode 参数，该参数指定 **RandomAccessFile** 的访问模式：

- **r**: 以只读方式打开
- **rw**: 打开以便读取和写入
- **rwd**: 打开以便读取和写入；同步文件内容的更新
- **rws**: 打开以便读取和写入；同步文件内容和元数据的更新

alt+向上键或者向下键表示向上或者向下移动一行

* RandomAccessFile: 支持随机访问

- * 1, 既可以充当一个输入流，又可以充当一个输出流
- * 2, 支持从文件的开头读取和写入
- * 3, 支持从任意位置的读取和写入

//进行文件的读和写

```
RandomAccessFile randomAccessFile=null;
RandomAccessFile rand =null;
try {
    randomAccessFile = new RandomAccessFile(new
File("C://Users//Anly//Desktop//55.txt"), "r");
    rand = new RandomAccessFile (new
File("C://Users//Anly//Desktop//55.txt"), "rw");
    byte []b = new byte[100];
    int len;
    try {
        while((len = randomAccessFile.read(b)) != -1){
            rand.write(b, 0, len);
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
```

```

        e.printStackTrace();
    }
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
finally {
    if (rand!=null) {
        try {
            rand.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    if (randomAccessFile != null) {
        try {
            randomAccessFile.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
}

```