IO

面向对象程序设计

第 13 讲 OOP 编程 -IO

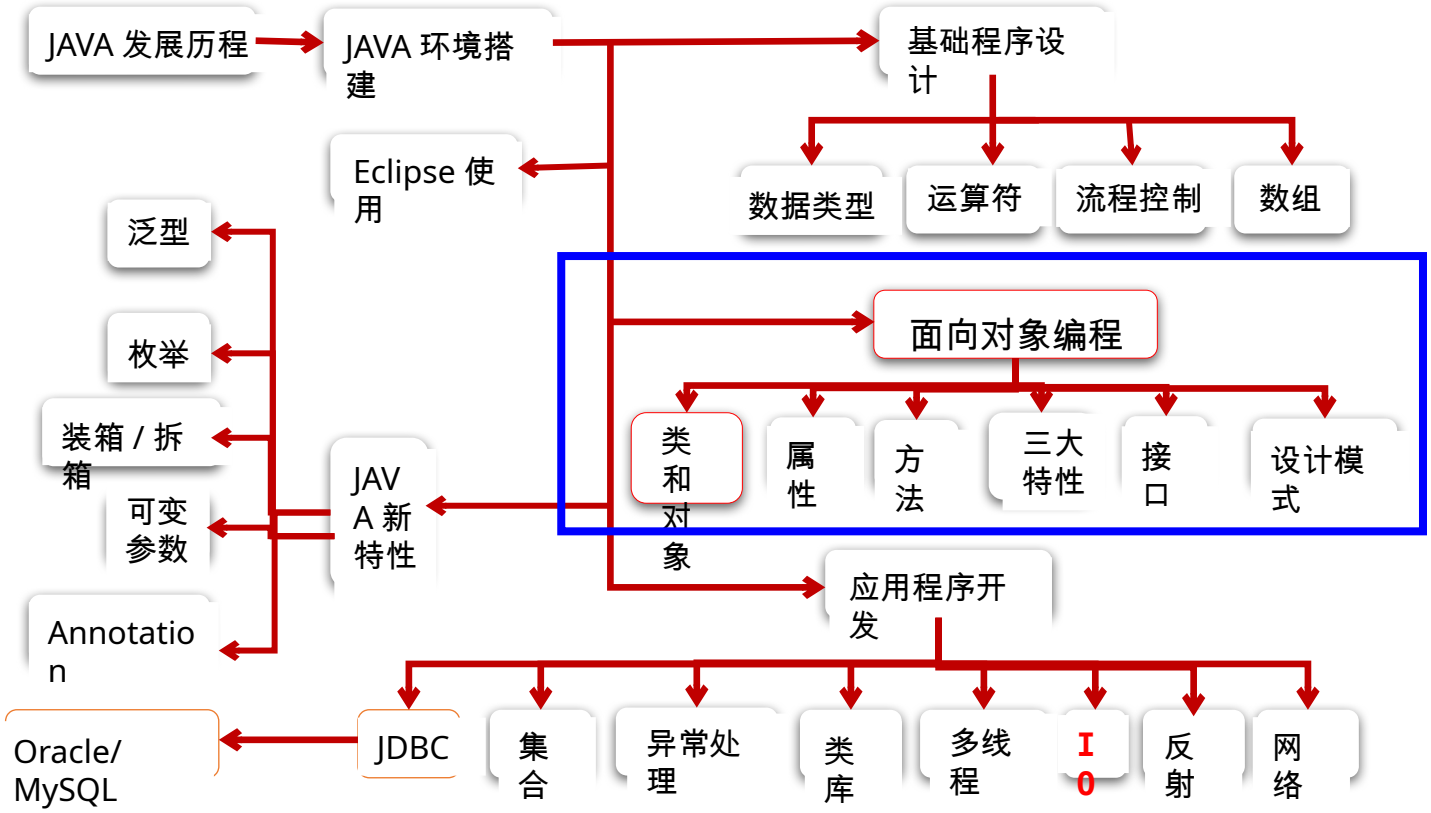
刘进

2230652597@qq.com

OOP 教辅 2022 秋季 QQ 群：

837966056

Java 基础知识图解


JAVA 发展历程
JAVA 环境搭建
基础程序设计
Eclipse 使用
数据类型
运算符
流程控制
数组
泛型
枚举
装箱 / 拆箱
可变参数
Annotation
JAVA 新特性
面向对象编程
类和对象
属性
方法
三大特性
接口
设计模式
应用程序开发
Oracle/ MySQL
JDBC
集合
异常处理
类库
多线程
IO
反射
网络

主要内容

1、 Java 流 (Stream) 、文件 (File) 和 IO

2、读取控制台输入

3、控制台输出

4、 FileInputStream 和 FileOutputStream

5、 Java 目录

6、转换流

7、打印流

8、数据流

9、对象流

内容

基本概念

常用频率 -1

常用频率 -2

泛化推广

1、Java 流(Stream)、文件(File)和 IO

Java IO 也称为 IO stream 流，它的核心就是对文件 File（抽象概念）的操作，对于字节、字符类型数据流的输入 Input 和输出流

基本概念

主体是“程序内存”

Output

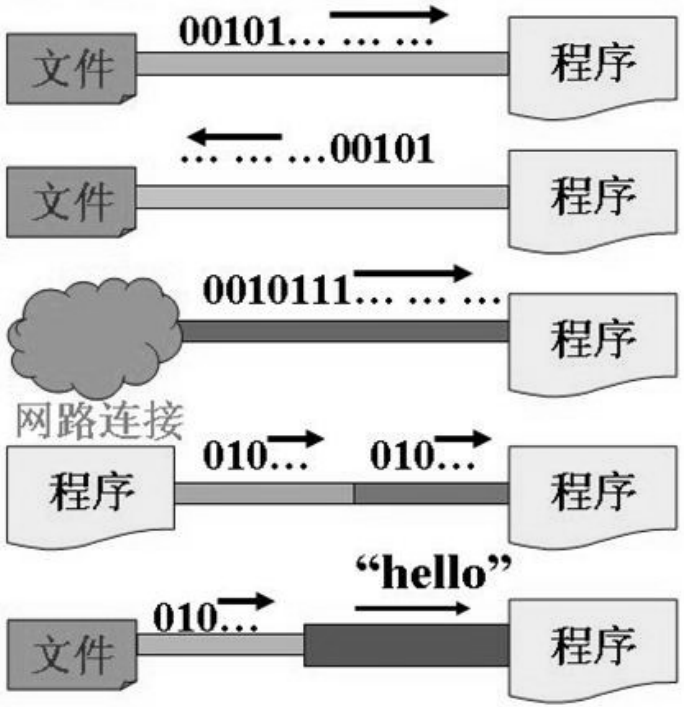
- **输入 input**：读取外部数据（磁盘、光盘等存储设备的数据）到程序（内存）中。

方向 + :（从数据集（File）注入到内存）

- **输出 output**：将程序（内存）数据输出到磁盘、光盘等存储设备中。

方向 - :（从内存注入到数据集（File））

通道过程（+ 有向 - ）

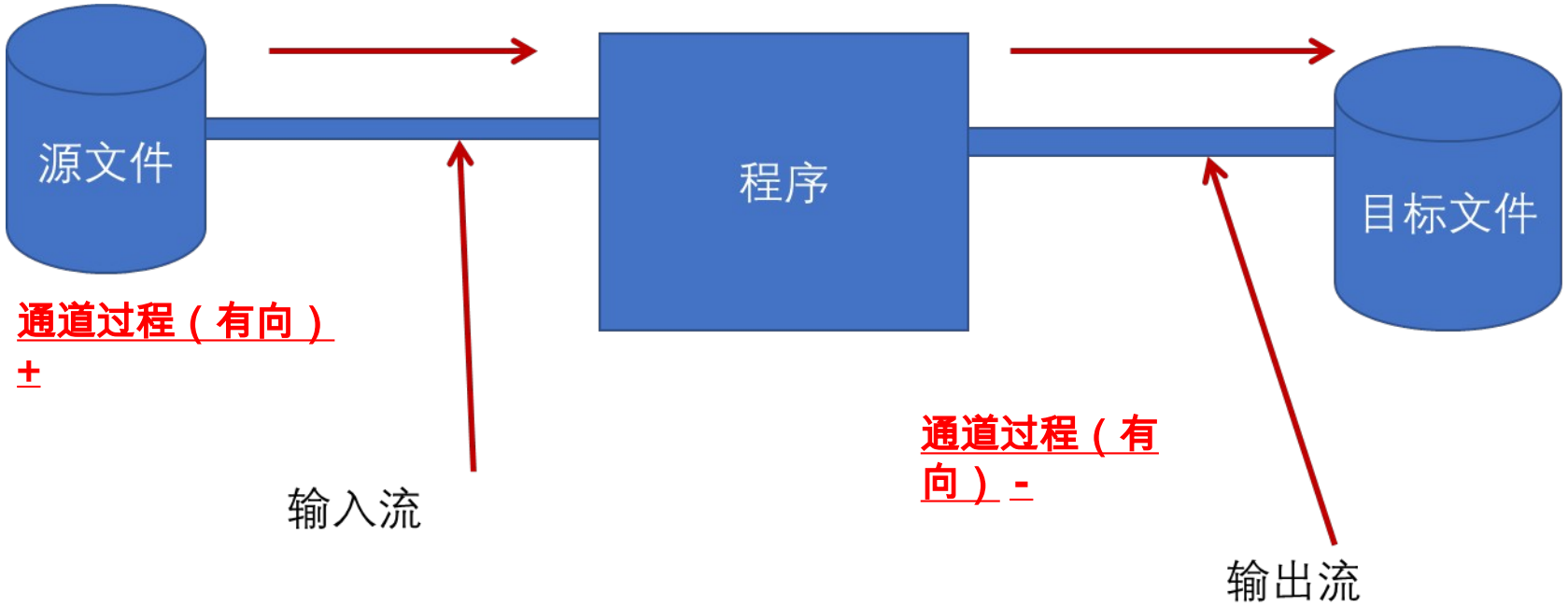


IO 流用来处理设备之间的数据传

Java IO

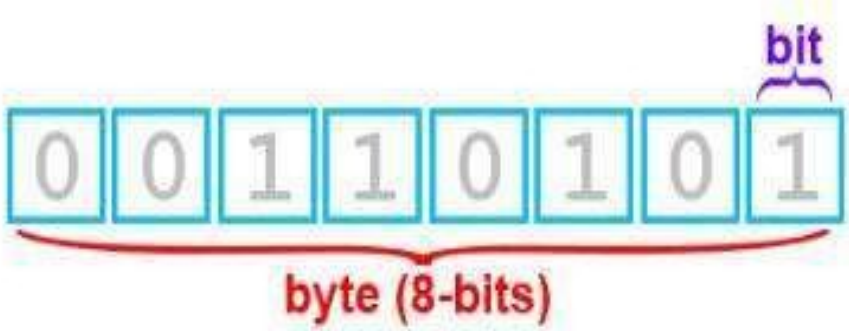
通道过程（有向）字节 / 字符

- IO 流主要分为两大类，字节流和字符流。**字节流可以处理任何类型的数据，如图片，视频等，字符流只能处理字符类型的数据。**
- IO 流的本质是数据传输，并且流是单向的。



Java IO

通道过程（有向）字节 / 字符　基本的砖块，更偏向“数据”的含义



字节 (Byte) 是计量单位，表示数据量多少，是计算机信息技术用于计量存储容量的一种计量单位，通常情况下一字节等于八位。

符号，有“意义的”数据

字符 (Character) 计算机中使用的字母、数字、字和符号，比
如 'A'、'B'、'$'、'&' 等。

一般在英文状态下一个字母或字符占用一个字节，一个汉字用两个字节表示。

ASCII 码中，一个英文字母（不分大小写）为一个字节，**一个中文汉字为两个字节**。
UTF-8 编码中，一个英文字为一个字节，**一个中文为三个字节**。
Unicode 编码中，一个英文为一个字节，**一个中文为两个字节**。
符号：英文标点为一个字节，**中文标点为两个字节**。例如：英文句号 . 占 1 个字节的大小，中文句号 。占 2 个字节的大小。
UTF-16 编码中，**一个英文字母字符或一个汉字字符存储都需要 2 个字节**（**Unicode 扩展区的一些汉字存储需要 4 个字节**）。
UTF-32 编码中，**世界上任何字符的存储都需要 4 个字节**。

Java 流操作的相关类或接口

File -- 文件类

RandomAccessFile -- 随机存储文件类

InputStream -- 字节输入流

OutputStream -- 字节输出流

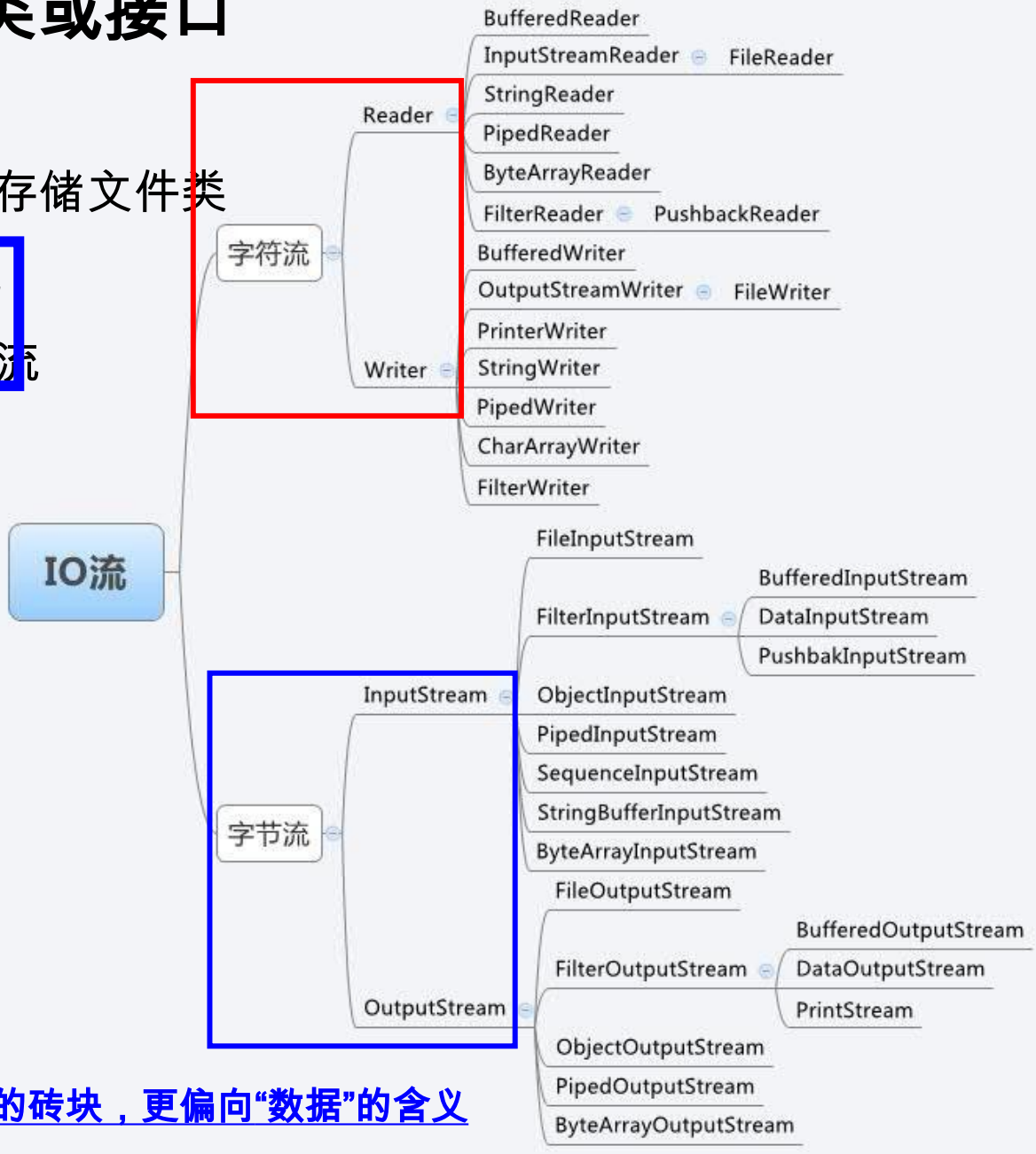
Reader -- 字符输入流

Writer -- 字符输出流

java.io 包下提供了各种“流”类和接口，用以获取不同种类的数据，并通过标准的方法输入或输出数据。

符号，有“意义的”数据

基本的砖块，更偏向“数据”的含义



Java IO 原理

- IO 流用来处理设备之间的数据传输。 “文件”是泛化的设备
- Java 程序中，对于数据的输入 / 输出操作以”流(stream)” 的方式进行。 数据流
- java.io 包下提供了各种“流”类和接口，用以获取不同种类的数据，并通过标准的**方法**输入或输出数据。

“流”的类型与标准化

流的概念和作用

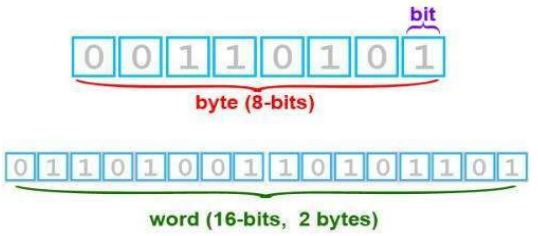
流

- 流是一组**有顺序的，有起点和终点的字节集合，是对数据传输的总称或抽象**。即数据在设备间传输称之为流。
- **流的本质是数据传输**，根据数据传输的特性将流区**分**为各种**类**，方便更直观的进行数据操作
- 根据数据处理的**不同类型**分为：**字节流**和**字符流**
- 根据数据**流向不同**分为：**输入流和输出流**

通道过程（+有向-）

字符流和字节流

- 字符流的由来：因为数据编码的不同，而有了对字符进行高效操作的流对象，本质上其实就是对于字节流的读取时，去查了指定的码表。 字符流 = 字节流 + 字符编码



- 字节流和字符流的区别：
  - 读写单位的不同：**字节流以字节（ 8bit ）为单位。字符流以字符为单位，根据码表映射字符，一次可能读多个字节**。
  - 处理对象不同：**字节流可以处理任何类型的数据**，如图片、avi 等，而**字符流只能处理字符类型的数据**。
- **程序从输入流中读取数据，向输出流中写入数据。**

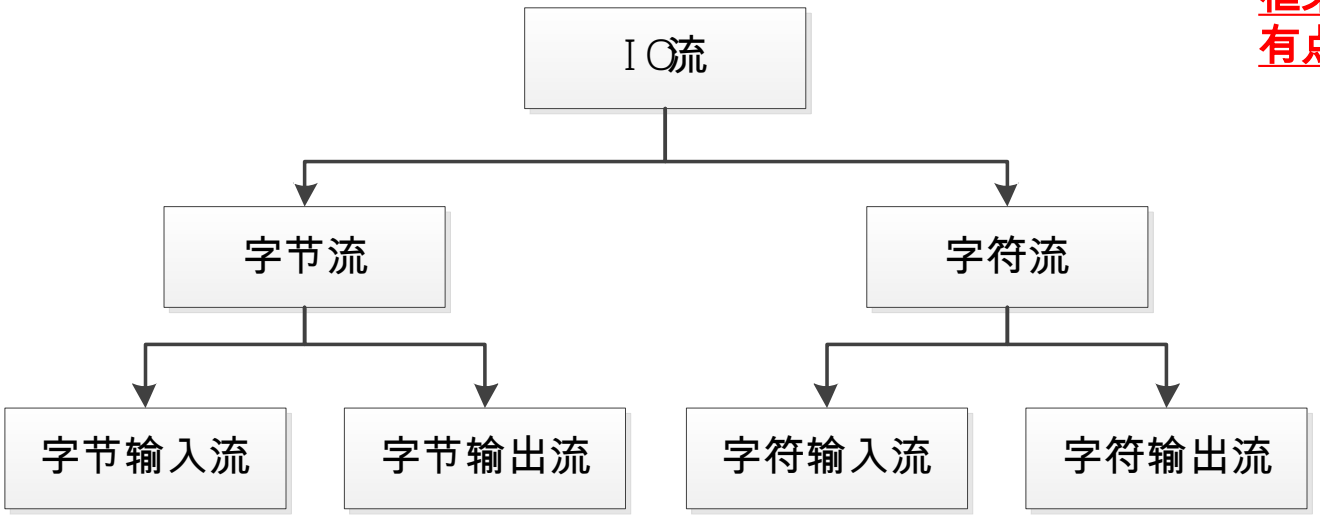
Java.io 包几乎包含了所有操作输入、输出需要的类。所有这些流类代表了输入源和输出目标。

Java.io 包中的流支持很多种格式，比如：基本类型、对象、本地化字符集等等。

一个流可以理解为一个数据的序列。输入流表示从一个源读取数据，输出流表示向一个目标写数据。

Java 为 I/O 提供了强大的而灵活的支持，使其更广泛地应用到文件传输和网络编程中

框架
有点像概念模型



字节流分别用 java.io.InputStream 和 java.io.OutputStream 表示，

字符流的输入输出流分别用 java.io.Reader 和 java.io.Writer 表示

- 按操作**数据单位**不同分为：**字节流 (8 bit) ，字符流 (16 bit)**
- 按数据流的**流向**不同分为：**输入流，输出流**
- 按流的**角色**的不同分为：**节点流，处理流**

（抽象基类）	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

1. Java 的 IO 流共涉及 40 多个类，实际上非常规则，都是从如下 4 个抽象基类派生的。
2. 由这四个类派生出来的子类名称都是以其父类名作为子类名后缀。

2、读取控制台输入

常用频率 - 控制台 ±

Java 的控制台输入由 System.in 完成。

为了获得一个绑定到控制台的字符流，可以将 System.in 包装在一个 BufferedReader 对象中来创建一个字符流。

创建 BufferedReader 的基本语法：

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

BufferedReader 对象创建后，可以使用 read() 方法从控制台读取一个字符，或者用 readLine() 方法读取一个字符串。每次调用 read() 方法从输入流读取一个字符并把该字符作为整数值返回。 当流结束的时候返回 -1 。该方法抛出 IOExcep-

从控制台读取多字符输入

举例±

BRRead.java

```
//使用 Buffer1edReader 在控制台读取字符
package ch013;
import java.io.*;

public class BRRead {
    public static void main(String args[]) throws IOException {
        char c;
        // 使用 System.in 创建 BufferedReader
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("输入字符，按下 'q' 键退出。");
        // 读取字符
        do {
            c = (char) br.read();
            System.out.println(c);
        } while (c != 'q');
    }
}
```

Console

```
<terminated> BRRead (1) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2018年12月
输入字符，按下 'q' 键退出。
123q
1
2
3
q
```

用 read() 方法从控制台不断读取字符直到用户输入 "q"

从控制台读取字符串

BRReadLines.java

```java
//使用 BufferedReader 在控制台读取字符
package ch013;
import java.io.*;

public class BRReadLines {
    public static void main(String args[]) throws IOException {
        // 使用 System.in 创建 BufferedReader
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str;
        System.out.println("Enter lines of text.");
        System.out.println("Enter 'end' to quit.");
        do {
            str = br.readLine();
            System.out.println(str);
        } while (!str.equals("end"));
    }
}
```

Console

```
<terminated> BRReadLines (1) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (2018
Enter lines of text.
Enter 'end' to quit.
line 1
line 1


line 2
line 2


end
end
```

举例

从标准输入读取一个字符串需要使用 BufferedReader 的 readLine() 方法。

3、控制台输出

常用频率 - 控制台 -

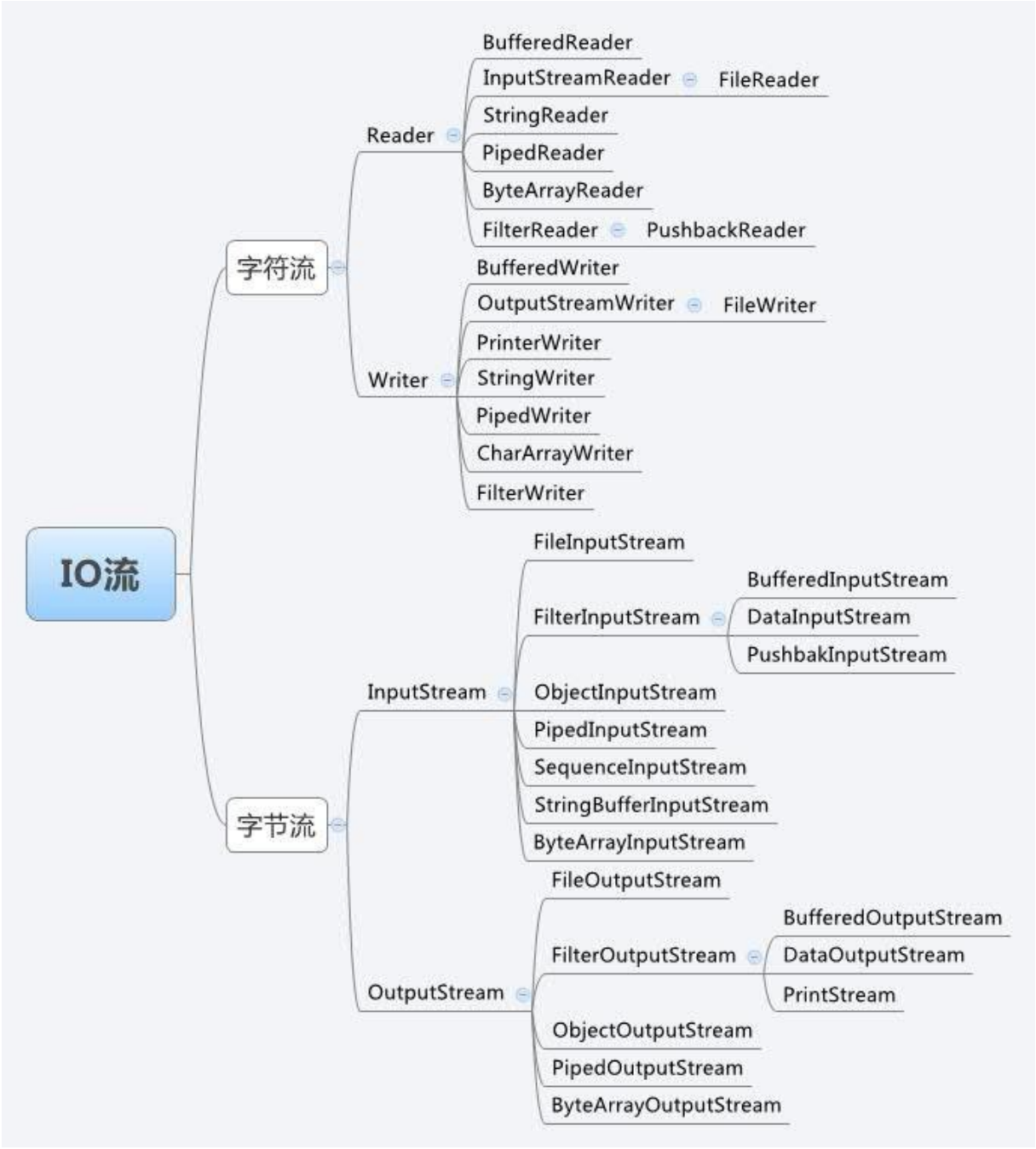
控制台的输出由 **print() 和 println()** 完成。这些方法都由类 PrintStream 定义，**System.out 是该类对象的一个引用**。PrintStream 继承了 OutputStream 类，并且实现了方法 write() 可以用来往控制台写操作。PrintStream 定义 write() 的最简单格式如下所示：**void write(int byteval)**，该方法将 byteval 的低八位字节写到流中。**write() 方法不经常使用，因为 print() 和 println() 方法用起来更为方便**。

从控制台读取多字符输入

常用频率 - 控制台 - write

用 write() 把字符 "A" 和紧跟着的换行符输出到屏幕

输入流和输出流的类层次



一个流被定义为一个数据序列。输入流用于从源读取数据，输出流用于向目标写数据。

4、 FileInputStream 和 FileOutputStream

常用频率 -2

FileOutputStream 该类用来**创建一个文件并向文件中写数据**。如果该流在打开文件进行输出前，目标文件不存在，那么该流会创建该文件。

有两个构造方法可以用来创建 FileOutputStream 对象：

1、使用字符串类型的文件名来创建一个输出流对象：

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

2、使用一个文件对象来创建一个输出流来写文件。我们首先得使用 File() 方法来创建一个文件对象：

```
File f = new File("C:/java/hello"); OutputStream f = new File-
OutputStream(f);
```

创建 OutputStream 对象完成后，可以使用下面的方法来写入流或者进行其他的流操作

序号	方法及描述
1	**public void close() throws IOException{}** 关闭此文件输入流并释放与此流有关的所有系统资源。抛出IOException异常。
2	**protected void finalize()throws IOException {}** 这个方法清除与该文件的连接。确保在不再引用文件输入流时调用其 close 方法。抛出IOException异常。
3	**public void write(int w)throws IOException{}** 这个方法把指定的字节写到输出流中。
4	**public void write(byte[] w)** 把指定数组中w.length长度的字节写到OutputStream中。

除了OutputStream外，还有一些其他的输出流，更多的细节参考下面链接：

- ByteArrayOutputStream
- DataOutputStream

InputStream 和 OutputStream 用法

BRReadLines.java | WriteDemo.java | fileStreamTest.java

```
import java.io.*;

public class fileStreamTest {
    public static void main(String args[]) {
        try {
            byte bWrite[] = { 11, 21, 3, 40, 5 };
            OutputStream os = new FileOutputStream("test.txt");
            for (int x = 0; x < bWrite.length; x++) {
                os.write(bWrite[x]); // writes the bytes
            }
            os.close();

            InputStream is = new FileInputStream("test.txt");
            int size = is.available();

            for (int i = 0; i < size; i++) {
                System.out.print((char) is.read() + "  ");
            }
            is.close();
        } catch (IOException e) {
            System.out.print("Exception");
        }
    }
}
```

Console

<terminated> fileStreamTest (1) [Java Application] C:\Program Files\Java\jdk1.8.0_1

□ □ □ (□

首先创建文件 test.txt ，并把给定的数字以二进制形式写进该文件，同时输出到控

名称

举例 + -

- .settings
- bin
- src
- .classpath
- .project
- test.txt

test.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

□□□(□

代码由于是二进制写入，存在乱码

改进的 InputStream 和 OutputStream 示例

BRReadLines.java | Wri...mo.java | fileStreamTest.java | fileStreamTest2.java

```
package ch013;
import java.io.*;

public class fileStreamTest2 {
    public static void main(String[] args) throws IOException {
        File f = new File("a.txt");
        FileOutputStream fop = new FileOutputStream(f);
        // 构建FileOutputStream对象,文件不存在会自动新建
        OutputStreamWriter writer = new OutputStreamWriter(fop, "UTF-8");
        // 构建OutputStreamWriter对象,参数可以指定编码,默认为操作系统默认编码,windows上是gbk
        writer.append("中文输入");
        // 写入到缓冲区
        writer.append("\r\n");
        // 换行
        writer.append("English");
        // 刷新缓存冲,写入到文件,如果下面已经没有写入的内容了,直接close也会写入
        writer.close();
        // 关闭写入流,同时会把缓冲区内容写入文件,所以上面的注释掉
        fop.close();
        // 关闭输出流,释放系统资源

        FileInputStream fip = new FileInputStream(f);
        // 构建FileInputStream对象
        InputStreamReader reader = new InputStreamReader(fip, "UTF-8");
        // 构建InputStreamReader对象,编码与写入相同
        StringBuffer sb = new StringBuffer();
        while (reader.ready()) {
            sb.append((char) reader.read());
            // 转成char加到StringBuffer对象中
        }
        System.out.println(sb.toString());
        reader.close();
        // 关闭读取流
        fip.close();
        // 关闭输入流,释放系统资源
    }
}
```

Console

<terminated> fileStreamTest2 (1) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.ex

```
中文输入
English
```

首先创建文件 a.txt ，并把给定的数字以字符串形式写进该文件，同时输出到控制台上。

a.txt - 记事本

文件(F) 编辑(E)

```
中文输入
English
```

- .settings
- bin
- src
- .classpath
- .project
- a.txt
- test.txt

举例 + -

文件和I/O

还有一些关于文件和I/O的类，我们也需要知道：

- File Class(类)
- FileReader Class(类)
- FileWriter Class(类)

5、Java 目录

泛化推广

创建目录：

File 类中有两个方法可以用来创建文件夹：

1、 **mkdir( ) 方法创建一个文件夹**，成功则返回 true ，失败则返回 false 。失败表明 File 对象指定的路径已经存在，或者由于整个路径还不存在，该文件夹不能被创建。

2、 mkdirs() 方法创建一个文件夹和它的所有父文件夹。

创建“ c:/abc" 文件夹

CreateDir.java | DirList.java

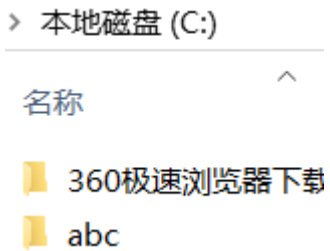
```
package ch013;
import java.io.File;

public class CreateDir {
    public static void main(String args[]) {
        String dirname = "c:/abc/tmp/user/java/bin";
        File d = new File(dirname);
        // 现在创建目录
        d.mkdirs();
    }
}
```

Console

```
<terminated> CreateDir [Java Application] C:\Program Files\Java\jdk1.
```

举例



Java 在 UNIX 和 Windows 自动按约定分辨文件路径分隔符。如果在 Windows 版本的 Java 中使用分隔符 (/) ，路径依然能够被正确解析。

读取目录或文件

一个目录其实就是一个 File 对象，它包含其他文件和文件夹。

如果**创建一个 File 对象**并且它是一个目录，那么调用 **isDirectory() 方法**会返回 true 。

可以通过**调用该对象上的 list() 方法，来提取它包含的文件和文件夹的列表**。

使用 list() 方法来检查一个文件夹中包含的内容

CreateDir.java | DirList.java

```
import java.io.File;

public class DirList {
    public static void main(String args[]) {
        String dirname = "c:/abc";
        File f1 = new File(dirname);
        if (f1.isDirectory()) {
            System.out.println("目录 " + dirname);
            String s[] = f1.list();
            for (int i = 0; i < s.length; i++) {
                File f = new File(dirname + "/" + s[i]);
                if (f.isDirectory()) {
                    System.out.println(s[i] + " 是一个目录");
                } else {
                    System.out.println(s[i] + " 是一个文件");
                }
            }
        } else {
            System.out.println(dirname + " 不是一个目录");
        }
    }
}
```

Console

```
<terminated> DirList (1) [Java Application] C:\Program Files\Java\jdk1.8.0_144\b
目录 c:/abc
tmp 是一个目录
```

举例

删除目录或文件

CreateDir.java | DeleteFileDemo.java

```
public class DeleteFileDemo {
    public static void main(String args[]) {
        // 这里修改为自己的测试目录
        File folder = new File("/tmp/java/");
        deleteFolder(folder);
    }

    // 删除文件及目录
    public static void deleteFolder(File folder) {
        File[] files = folder.listFiles();
        if (files != null) {
            for (File f : files) {
                if (f.isDirectory()) {
                    deleteFolder(f);
                } else {
                    f.delete();
                }
            }
        }
        folder.delete();
    }
}
```

Console

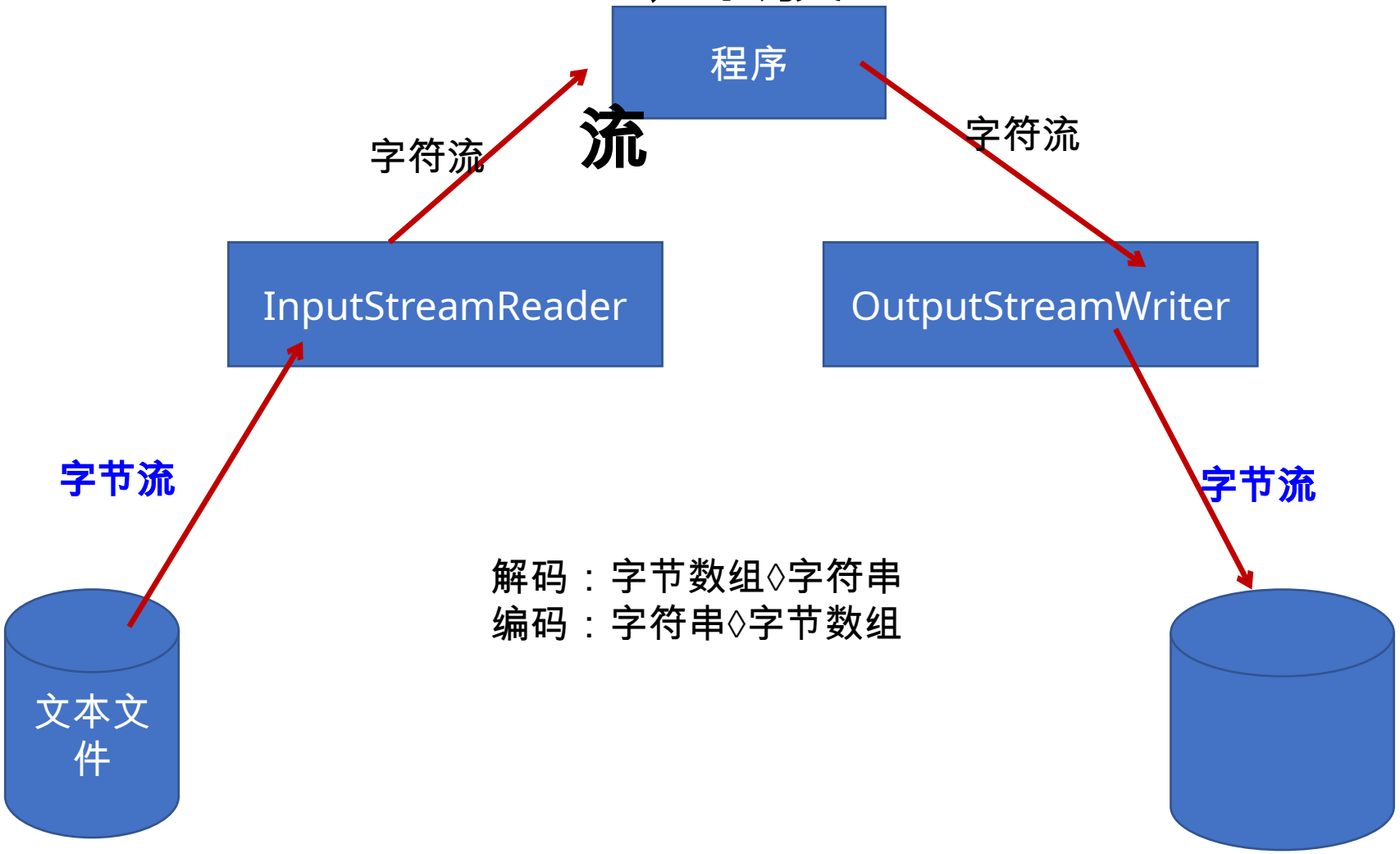
<terminated> DeleteFileDemo [Java Application] C:\Program Files\Jav

举例

删除文件可以使用 **java.io.File.delete()** 方法。

以下代码会删除目录 **/tmp/java/**，需要注意的是当删除某一目录时，必须保证该目录下没有其他文件才能正确删除，否则将删除失败。

6、转换流



解码：字节数组◊字符串
编码：字符串◊字节数组

```
public void testMyInput() throws Exception{
    FileInputStream fis = new FileInputStream("dbcp.txt");
    FileOutputStream fos = new
FileOutputStream("dbcp5.txt");

    InputStreamReader isr = new
InputStreamReader(fis,"GBK");
    OutputStreamWriter osw = new
OutputStreamWriter(fos,"GBK");

    BufferedReader br = new BufferedReader(isr);
    BufferedWriter bw = new BufferedWriter(osw);

    String str = null;
    while((str = br.readLine()) != null){
        bw.write(str);
        bw.newLine();
        bw.flush();
```

补充：字符编码

- **编码表的由来**

计算机只能识别二进制数据，早期由来是电信号。为了方便应用计算机，让它可以识别各个国家的文字。就将各个国家的文字用数字来表示，并一一对应，形成一张表。这就是编码表。

- **常见的编码表**

➢ ASCII ：美国标准信息交换码。
  ✓ 用一个字节的 7 位可以表示。

➢ ISO8859-1 ：拉丁码表。欧洲码表
  ✓ 用一个字节的 8 位表示。

➢ GB2312 ：中国的中文编码表。

➢ GBK ：中国的中文编码表升级，融合了更多的中文文字符号。

➢ Unicode ：国际标准码，融合了多种文字。
  ✓ 所有文字都用两个字节来表示 ,Java 语言使用的就是 unicode

➢ UTF-8 ：最多用三个字节来表示一个字符。

补充：字符编码

- 编码：字符串◊字节数组
- 解码：字节数组◊字符串

- **转换流的编码应用**
  - 可以将字符按指定编码格式存储。
  - 可以对文本数据按指定编码格式来解读。
  - 指定编码表的动作由构造器完成。

处理流之三：标准输入输出流

- System.in 和 System.out 分别代表了系统标准的输入和输出设备
- **默认输入设备是键盘，输出设备是显示器**
- System.in 的类型是 InputStream
- System.out 的类型是 PrintStream ，其是 OutputStream 的子类 FilterOutputStream 的子类
- 通过 System 类的 setIn ， setOut 方法对默认设备进行改变。
  - public static void **setIn**(InputStream in)
  - public static void **setOut**(PrintStream out)

例　题

举例

从键盘输入字符串，要求将读取到的整行字符串转成大写输出。然后继续进行输入操作，直至当输入“ e ”或者“ exit ”时，退出程序。

```
System.out.println(" 请输入信息 ( 退出输入 e 或 exit):");
// 把 " 标准 " 输入流 ( 键盘输入 ) 这个字节流包装成字符流 , 再包装成缓冲流
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String s = null;
try {
while ((s = br.readLine()) != null) {
// 读取用户输入的一行数据 --> 阻塞程序
if (s.equalsIgnoreCase("e") || s.equalsIgnoreCase("exit")) {
        System.out.println(" 安全退出 !!");
// 将读取到的整行字符串转成大写输出
System.out.println("-->:"+s.toUpperCase());
System.out.println(" 继续输入信息 ");
}
} catch (IOException e) {
        e.printStackTrace();
}
finally {
        try {
        if (br != null) {
        br.close();  // 关闭过滤流时 , 会自动关闭它包装的底层节点流
}
} catch (IOException e) {
        e.printStackTrace();
}
}
```

举例

7、打印

打印流

- 在整个 IO 包中，打印流是输出信息最方便的类。

流

- **PrintStream(字节打印流) 和 PrintWriter(字符打印流)**
  - 提供了一系列重载的 print 和 println 方法，用于多种数据类型的输出
  - PrintStream 和 PrintWriter 的输出不会抛出异常
  - PrintStream 和 PrintWriter 有自动 flush 功能

举例

```
FileOutputStream fos = null;
        try {
          fos = new FileOutputStream(new File("D:\\IO\
\text.txt"));
        } catch (FileNotFoundException e) {
                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 ps = new PrintStream(fos,true);
        if (ps != null) { //  把标准输出流 ( 控制台输出 ) 改成文件
                System.setOut(ps);}
        for (int i = 0; i <= 255; i++) {  // 输出 ASCII 字符
                System.out.print((char)i);
                if (i % 50 == 0) {   // 每 50 个数据一行
                        System.out.println(); //  换行
                } }
        ps.close();
```

8、数据流

- 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**

boolean readBoolean()　　byte readByte()
char readChar()　　float readFloat()
double readDouble()　　short readShort()
long readLong()　　int readInt()
String readUTF()　　void readFully(byte[] b)

- **DataOutputStream 中的方法**
  - 将上述的方法的 read 改为相应的 write 即可。

举例

```
DataOutputStream dos = null;
 try {      // 创建连接到指定文件的数据输出流对象
            dos = new DataOutputStream(new FileOutputStream(
                                        "d:\\IOTest\\destData.dat"));
                        dos.writeUTF("ab 中国 "); // 写 UTF 字符串
                        dos.writeBoolean(false); // 写入布尔值
                        dos.writeLong(1234567890L); // 写入长整数
                        System.out.println(" 写文件成功 !");
            } catch (IOException e) {
                        e.printStackTrace();
            } finally {// 关闭流对象
                        try {
                        if (dos != null) {
                        //  关闭过滤流时 , 会自动关闭它包装的底层节点流
                        dos.close();
                        }
            } catch (IOException e) {
                        e.printStackTrace();
            }
 }
```

对象流

9、对象流

- **ObjectInputStream 和 ObjectOutputSteam**
  - 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 VS 符号 - 向量化（Embedding）编码

- **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

反序列化 VS 符号 - 向量化（Decode）解码

  - ObjectOutputStream 和 ObjectInputStream 不能序列化 static 和 transient 修饰的成员变量

对象的序列化

- **对象序列化机制**允许**把内存中的 Java 对象转换成平台无关的二进制流**，从而允许把这种二进制流持久地保存在磁盘上，或通过网络将这种二进制流传输到另一个网络节点。当其它程序获取了这种二进制流，就可以恢复成原来的 Java 对象
- 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**，使其在保存和传输时可被还原
- **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**
- 如果需要让某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，该类必须实现如下两个接口之一：
  - **Serializable**
  - Externalizable

对象的序列化

- 凡是实现 Serializable 接口的类都有一个表示序列化版本标识符的静态变量：
  - **private static final long serialVersionUID;**
  - serialVersionUID 用来表明类的不同版本间的兼容性
  - 如果类没有显示定义这个静态变量，它的值是 Java 运行时环境根据类的内部细节自动生成的。若类的源代码作了修改， serialVersionUID 可能发生变化。故建议，显示声明
- 显示定义 serialVersionUID 的用途
  - 希望类的不同版本对序列化兼容，因此需确保类的不同版本具有相同的 serialVersionUID
  - 不希望类的不同版本对序列化兼容，因此需确保类的不同版本具有不同的 serialVersionUID

使用对象流序列化对象

- 若某个类实现了 Serializable 接口，该类的对象就是可序列化的：
  - **创建一个 ObjectOutputStream**
  - **调用 ObjectOutputStream 对象的 writeObject(对象) 方法输出可序列化对象。注意写出一次，操作 flush()**
- 反序列化
  - **创建一个 ObjectInputStream**
  - **调用 readObject() 方法读取流中的对象**
- **强调**：如果某个类的字段不是基本数据类型或 String 类型，而是另一个引用类型，那么这个引用类型必须是可序列化的，否则拥有该类型的 Field 的类也不能序列化

序列化：将对象写入到磁盘或者进行网络传输。 举例

要求对象必须实现序列化

```
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("test3.txt"));
Person p = new Person(" 韩梅梅 ",18," 中华大街 ",new
Pet());
oos.writeObject(p);
oos.flush();
oos.close();
// 反序列化：将磁盘中的对象数据源读出。
ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("test3.txt"));
Person p1 = (Person)ois.readObject();
System.out.println(p1.toString());
```

举例

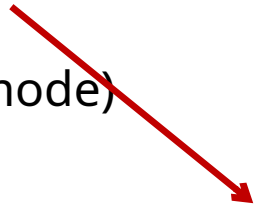
RandomAccessFile 类

- 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

举例

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**

举例

读取文件内容

```
RandomAccessFile raf = new RandomAccessFile(“test.txt”, “rw” ) ;
raf.seek(5);
byte [] b = new byte[1024];

int off = 0;
int len = 5;
raf.read(b, off, len);// read(byte[] b, int off 下标 , int len 长度 )

String str = new String(b, 0, len);
System.out.println(str);

raf.close();
```

举例

写入文件内容

```
RandomAccessFile raf = new RandomAccessFile("test.txt", "rw");
        raf.seek(5); // 将文件记录指针定位到 pos “ 5 ”的位置

        // 先读出来
        String temp = raf.readLine();

        raf.seek(5);
        raf.write("xyz".getBytes());          // write(byte[] b, int off, int len)
        raf.write(temp.getBytes());

        raf.close();
```

举例

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

public class MyRandomAccessFile {
   public static void main(String[] args) throws IOException {
      // 创建目录
      File dir = new File("demo");
      if (!dir.exists()) {
         dir.mkdir();
      }
      // 创建文件
      File file = new File(dir, "test.dat");
      if (!file.exists()) {
         file.createNewFile();
      }
      // 实例化 RandomAccessFile 对象
      RandomAccessFile raf = new RandomAccessFile(file, "rw");
      // 打开文件时指针位置在最前，即 0
      System.out.println(raf.getFilePointer());
      // 写入数据
      int[] num = {28, 14, 56, 23, 98};
      for (int i : num) {
         raf.writeInt(i);
      }
      // 读取文件，在读取前需要通过 seek() 方法把文件指针移到最前
      raf.seek(0);
      for (int i = 0; i*4 < raf.length(); i++) {
         System.out.print(raf.readInt()+" ");
      }
      // 操作结束后一定要关闭文件
      raf.close();
   }
}
```

举例

MyRandomAccessFile.java

```
package ch013;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;

public class MyRandomAccessFile {
    public static void main(String[] args) throws IOException {
        //创建目录
        File dir = new File("demo");
        if (!dir.exists()) {
            dir.mkdir();
        }
        //创建文件
        File file = new File(dir, "test.dat");
        if (!file.exists()) {
            file.createNewFile();
        }
        //实例化RandomAccessFile对象
        RandomAccessFile raf = new RandomAccessFile(file, "rw");
        //打开文件时指针位置在最前，即0
        System.out.println(raf.getFilePointer());  //打印0
        //写入数据
        int[] num = {28, 14, 56, 23, 98};
        for (int i : num) {
            raf.writeInt(i);
        }
        //读取文件，在读取前需要通过seek()方法把文件指针移到最前
        raf.seek(0);
        for (int i = 0; i*4 < raf.length(); i++) {
            System.out.print(raf.readInt()+" ");
        }
        //操作结束后一定要关闭文件
        raf.close();
    }
}
```

Console

```
<terminated> MyRandomAccessFile [Java Application] C:\Program Files\Java\jdk1.8.0_1
0
28 14 56 23 98
```

本章小节

- **流是用来处理数据的**。

- 处理数据时，一定要先明确**数据源**，与**数据目的地**
  - 数据源可以是文件，可以是键盘。
  - 数据目的地可以是文件、显示器或者其他设备。

- 而流只是在帮助数据进行传输，并对传输的数据进行处理，比如过滤处理、转换处理等。

- **字节流 - 缓冲流**
  - 输入流 InputStream-FileInputStream-BufferedInputStream
  - 输出流 OutputStream-FileOutputStream-BufferedOutputStream
- **字符流 - 缓冲流**
  - 输入流 Reader-FileReader-BufferedReader
  - 输出流 Writer-FileWriter-BufferedWriter
- **转换流**
  - InputSteamReader 和 OutputStreamWriter
- **对象流** ObjectInputStream 和 ObjectOutputStream
  - 序列化
  - 反序列化

本章课后作业

1. 第 13 讲 ppt 上的示例程序代码编写（与调试）一遍，代码发给 2230652597@qq.com