

IO - Day01

今日学习内容：

- File类和基本操作
- 常见的字符集，各自有什么特点
- 字符的编码和解码操作
- 什么是IO，IO有哪些分类
- IO操作示意图是怎么样的
- IO的四大基流有哪些
- 四个文件流的操作

今日学习目标：

- 掌握File类中常用方法的操作
- 了解字符集和其存储特点
- 了解字符编码和解码操作过程
- 了解IO操作示意图
- 了解IO的分类和操作注意
- 掌握IO的四大基流有哪些
- 熟练掌握使用文件字节流读取文件数据
- 熟练掌握使用文件字符流写入数据到文件

IO入门

1、File类（掌握）

File可以理解为文件和文件夹（目录），用于表示磁盘中某个文件或文件夹的路径。该类包含了文件或者文件夹的创建、删除、重命名、判断是否存在等方法。

注意：只能获取和设置文件本身的信息（文件大小，是否可读），不能设置和获取文件里面的内容。

- Unix/Linux/Mac: 严格区分大小写，使用“/”来表示路径分隔符。
- Windows: 默认情况下是不区分大小写的，使用“\”来分割目录路径。但是在Java中一个“\”表示转义，所以在Windows系统中就得使用两个“\\”。

操作File常见方法：

- String getName(): 获取文件名称
- String getPath(): 获取文件路径
- String getAbsolutePath(): 获取绝对路径
- File getParentFile(): 获取上级目录
- boolean exists(): 判断文件或者目录是否存在
- boolean isFile(): 是否是文件
- boolean isDirectory(): 判断是否是目录
- boolean delete(): 删除文件或者目录
- boolean mkdirs(): 创建当前目录和上级目录
- File[] listFiles(): 列出所有文件对象

```
public class FileDemo {  
    public static void main(String[] args) throws Exception {  
        File f = new File("C:/test/123.txt");  
        System.out.println(f.getName()); //123.txt  
    }  
}
```

```

        System.out.println(f.getPath()); //C:/test/123.txt
        System.out.println(f.getAbsolutePath()); //C:/test/123.txt
        System.out.println(f.getParentFile().getName()); //test
        System.out.println(f.exists()); //true
        System.out.println(f.isFile()); //true
        System.out.println(f.isDirectory()); //false

        //如果当前文件的父文件夹不存在,则创建
        if(!f.getParentFile().exists()) {
            f.getParentFile().mkdirs();
        }

        //列出当前文件夹中所有文件
        File[] fs = f.getParentFile().listFiles();
        for (File file : fs) {
            System.out.println(file);
        }
    }
}

```

需求：列出给定目录中的全部文件的路径，包括给定目录下面的所有子目录。

```

public class FileDemo2 {

    public static void list(File file) {

        if (file.isDirectory()) {
            System.out.println(file.getPath());
            // 如果是文件夹，则继续列出
            File[] fs = file.listFiles();
            if (fs != null) {
                for (File f : fs) {
                    list(f);
                }
            }
        }
        System.out.println(file);
    }

    public static void main(String[] args) {
        // 需求：给定一个目录(test),遍历其下的所有目录(包含子目录)和文件
        File file = new File("h:\\test");
        list(file);
    }
}

```

2、字符编码

2.1、字符编码发展历程（了解）

阶段一：计算机在美国等发达国家起步

计算机只认识数字(0和1)，在计算机里一切数据都是以数字来表示，因为英文符号有限，所以规定使用的字节的最位是0。每一个字节都是以0~127之间的数字来表示，比如A对应65，a对应97。此时把每一个字节按照顺序存放在一张表格中，这就是美国标准信息交换码——**ASCII编码表**。

阶段二：其他国家的普及

随着计算机在全球的普及，很多国家和地区都把自己的字符引入了计算机，比如汉字。此时发现一个字节（128个）能表示数字范围太小，而汉字太多，128个数字不能包含所有的中文汉字，那么此时就规定**使用两个字节一起来表示一个汉字**。

规定：原有的ASCII字符的编码保持不变，仍然使用一个字节表示，为了区别一个中文字符与两个ASCII码字符，中文字符的每个字节最高位（符号位）规定为1（中文的二进制是负数），该规范就是**GB2312编码表**。后来在GB2312码表的基础上增加了更多的中文汉字，也就出现了更强大的GBK码表。

阶段三：互联网发展

中国人是认识汉字的，现在需要和外国人通过网络交流，此时需要把把汉字信息传递给外国人，但外国的码表中没有收录汉字，此时就会把汉字显示为另一个符号甚至不能识别的乱码。为了解决各个国家因为本地化字符编码带来的影响，就干脆把全世界所有的符号统一收录进Unicode编码表。

如果使用Unicode码表，那么某一个字符在全世界任何地方都是固定的。比如'哥'这个字，在任何地方都是以十六进制的54E5来表示，因此说**Unicode是国际统一编码**。

Unicode字符集：<https://unicode-table.com/cn/blocks/>

2.2、常见的字符集（了解）

常见的字符集

- **ASCII**：每个字符占一个字节，只能包含128个符号。不能表示汉字。
- **ISO-8859-1**：也称之为latin-1，占一个字节，只收录西欧语言，不能表示汉字。
- **GB2312/GBK/GB18030**：占两个字节，支持中文。
- **ANSI**：占两个字节，在Win7简体中文的操作系统中ANSI 就指的是 GBK。
- **UTF-8**：是一种针对Unicode的可变长度字符编码，是Unicode的实现方式之一，支持中文。在开发中建议使用。
- **UTF-8 BOM**：是微软搞出来的一种编码，不要使用。

存储字母、数字、汉字的常识：

- (1) 存储字母和数字无论是什么字符集都占1个字节。
- (2) 存储汉字，GBK家族占两个字节，UTF-8家族占3个字节。
- (3) 不能使用单字节的字符集（ASCII、ISO-8859-1）来存储中文，否则会乱码。

2.3、字符的编码和解码操作（了解）

数据在网络上传输是以二进制的格式，二进制格式就是byte数组，此时需要把信息做编码和解码处理。

- 编码：把字符串转换为byte数组 String--->byte[]
- 解码：把byte数组转换为字符串 byte[]--->String

注意：一定要保证编码和解码的字符集相同，才能正确解码出信息。

IO流操作

1、IO流概述（了解）

1.1、IO概述 (了解)

什么是IO, Input和Output, 即输入和输出。

电脑相关的IO设备: 和电脑通信的设备, 此时要站在电脑的角度, 把信息传递给电脑叫输入设备, 把电脑信息传递出来的叫输出设备。

- 输入设备: 键盘、鼠标、麦克风、扫描器等
- 输出设备: 显示器、打印机、投影仪、耳机、音响等

为什么程序需要IO呢?

案例1: 打游戏操作, 需要存储游戏的信息 (保存副本)。

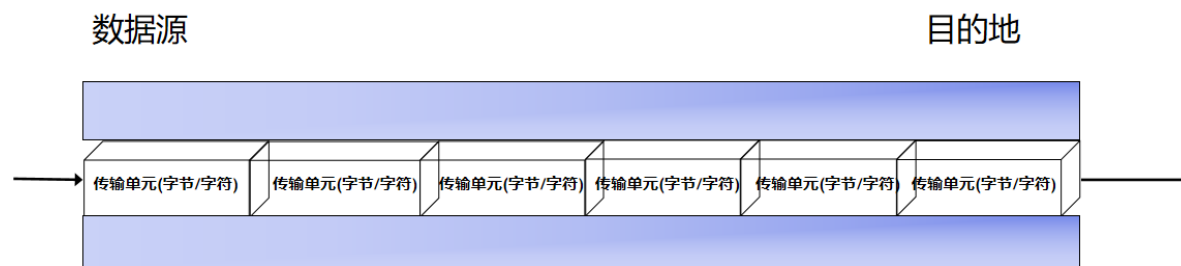
- 此时需要把游戏中的数据存储在文件中。

案例2: 打游戏操作, 需求读取之前游戏的记录信息, 数据存储在文件中的。

- 此时游戏程序需要去读取文件中的数据, 并显示在游戏中。

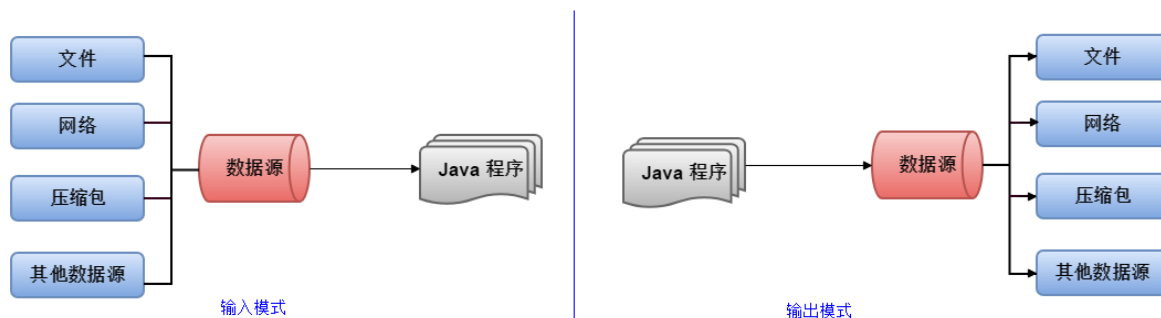
流的概念

流(stream)是指一连串流动的数据单元(字符、字节等),是以**先进先出**方式发送信息的通道。



IO流操作是一个相对的过程, 一般的, 我们在程序角度来思考 (程序的内存)。

- 程序需要读取数据: 文件——>程序, 输入操作
- 程序需要保存数据: 程序——>文件, 输出操作



1.2、IO操作示意图 (了解)

讲解IO知识点的时候, 习惯和生活中的水流联系起来, 一起来看看复古的水井和水缸。



水滴



此时站在水缸的角度，分析IO的操作方向：

输入操作：水井——>水缸

输出操作：水缸——>饭锅

注意：谁拥有数据，谁就是源，把数据流到哪里，哪里就是目标。那么，请问水缸是源还是目标。

1.3、流的分类（掌握）

根据流的不同特性，流的划分是不一样的，一般按照如下情况来考虑：

- 按**流动方向**：分为输入流和输出流
- 按**数据传输单位**：分为字节流和字符流，即每次传递一个字节（byte）或一个字符（char）
 - 字符和字节是什么关系？
 - 一个字符按照字符集可以编码成一个或者多个字节
- 按**功能上划分**：分为节点流和处理流，节点流功能单一，处理流功能更强（下节课）。

流的流向是相对的，我们一般**站在程序(内存)**的角度：

- 程序需要数据 → 把数据读进来 → 输入操作（read）：读进来
- 程序保存数据 → 把数据写出去 → 输出操作（write）：写出去

六字箴言：读进来，写出去（仔细揣摩这六个字有什么高深的含义）

2、四大基流（掌握）

| 流向 | 字节流(单位是字节) | 字符流(单位是字符) |
|-----|---------------------|---------------|
| 输入流 | InputStream（字节输入流） | Reader（字符输入流） |
| 输出流 | OutputStream（字节输出流） | Writer（字符输出流） |

操作IO流的模板：

- 1): 确定源或者目标对象(挖井)。
 - 输入操作：把文件中的数据流向到程序中,此时文件是源,程序是目标。
 - 输出操作：把程序中的数据流向到文件中,此时文件是目标,程序是源。
- 2): 创建IO流对象(水管)。
 - 输入操作：创建输入流对象。
 - 输出操作：创建输出流对象。
- 3): 具体的IO操作。

输入操作： 输入流对象的read方法。
输出操作： 输出流对象的write方法。

4):关闭资源(勿忘)。一旦资源关闭之后,就不能使用流对象了,否则报错。

输入操作： 输入流对象.close();
输出操作： 输出流对象.close();

注意:

- 四大抽象流是不能创建对象的,一般的我们根据不同的需求创建他们不同的子类对象,比如操作文件时就使用文件流。
- 不管是什么流,操作完毕都必须调用close方法,释放资源。

3、字节输入流/字节输出流

2.1、InputStream (字节输入流)

InputStream 表示字节输入流的所有类的抽象父类。

InputStream 常用方法

| 方法 | 方法作用 |
|-----------------------|---|
| abstract int read() | 从输入流中读取 一个字节 数据并返回该字节数据,如果到达流的末尾,则返回 -1。 |
| int read(byte[] buff) | 从输入流中读取 多个字节 数据,并存储在缓冲区数组 buff 中。返回已读取的字节数量,如果已到达流的末尾,则返回 -1 |
| void close() | 关闭此输入流并释放与该流关联的所有系统资源。 InputStream 的 close 方法不执行任何操作。 |

2.2、OutputStream (字节输出流)

OutputStream 表示字节输出流的所有类抽象父类。

OutputStream常用方法:

| 方法 | 方法作用 |
|----------------------------------|-----------------------------------|
| abstract void write(int b) | 将指定的 一个字节 数据b写入到输出流中。 |
| write(byte[] buff) | 把 数组buff中所有字节 数据写入到输出流中。 |
| write(byte[] b, int off,int len) | 把数组buff中从索引off 开始的len 个字节写入此输出流中。 |
| void close() | 关闭此输出流并释放与此流有关的所有系统资源。 |

InputStream/OutputStream 只定义了流的流向和流通道的数据单元,并没有定义数据源和目的地,java.io包中的类是按照源数据源和目的地进行划分的。

java.io包中的具体的流操作类命名规则: 数据源/目的地 + 数据传输单元 (流向)

例如 : File + InputStream / File + OutputStream

2.3、文件字节流（重点）

当程序需要读取文件中的数据或者把数据保存到文件中去，此时就得使用文件流，但是注意只能操作纯文本文件（txt格式），不要使用Word、Excel。文件流比较常用。

需求1：使用文件字节输入流，读取a.txt文件中的数据

```
public class FileInputStreamDemo1 {
    public static void main(String[] args) throws IOException {
        // 需求：使用文件字节输入流，a.txt文件中的数据读取到程序中
        /**
         * 分析：
         * 数据源：文件 File
         * 流向：输入流 Input
         * 传输单元：字节 Stream => FileInputStream
         */

        // step 1: 确定数据源
        File file = new File("G:\\test\\a.txt");
        // step 2: 建立管道
        FileInputStream in = new FileInputStream(file);
        // step 3: 读取操作(一次读取一个)
        int b = 0;
        b = in.read(); // 读取a字符
        b = in.read(); // 读取b字符
        b = in.read(); // 读取c字符
        System.out.println((char)c);

        // 已经读到末尾，返回-1
        b = in.read(); // -1
        System.out.println(c);

        // step 4: 关闭操作
        in.close();

    }
}
```

```
public class FileInputStreamDemo2 {
    public static void main(String[] args) throws IOException {
        // 需求：使用文件字节输入流，a.txt文件中的数据读取到程序中
        // step 1: 确定数据源
        File file = new File("G:\\test\\a.txt");
        // step 2: 建立输入流管道( 创建具体的IO流对象 )
        FileInputStream in = new FileInputStream(file);
        // step 3: 操作IO
        // 一次读取一个字节
        int b;
        while ( (b=in.read()) != -1 ) {
            System.out.println((char)b);
        }

        // step 4: 关闭管道
        in.close();

    }
}
```

```
public class FileInputStreamDemo3 {
```

```

public static void main(String[] args) throws IOException {
    // 需求：使用文件字节输入流，a.txt文件中的数据读取到程序中
    // step 1: 确定数据源
    File file = new File("G:\\test\\a.txt");
    // step 2: 建立输入流管道( 创建具体的IO流对象 )
    FileInputStream in = new FileInputStream(file);

    // step 3: 操作IO
    // 一次读取多个字节
    byte[] buf = new byte[2];    // 缓存数组
    int len;                      // 表示读取的有效字节的个数

    StringBuilder sb = new StringBuilder();
    while( (len = in.read(buf)) != -1 ) {
        String str = new String(buf,0,len);
        sb.append(str);
    }
    System.out.println(sb.toString());

    // step 4: 关闭管道
    in.close();
}
}

```

需求2: 使用文件字节输出流，把程序中数据保存到b.txt文件(先只操作英文)

```

public class Test01FileOutputStream {
    public static void main(String[] args) throws IOException {

        // step 1: 确定目的地 (b.txt)
        File file = new File("G:\\test\\b.txt");

        // step 2: 建立流管道 (通过具体的IO流对象创建管道)
        FileOutputStream out = new FileOutputStream(file);

        // step 3: 操作IO
        // 一次写入一个字节
        /*
        out.write(97);
        out.write(98);
        out.write(99);
        */

        // 一次写入多个字节
        String str = "hello";
        byte[] buf = str.getBytes("UTF-8");
        out.write(buf);

        // step 4: 关闭管道
        out.close();
    }
}

```


4、字符输入流/字符输出流

4.1、Reader（字符输入流）

Reader表示所有字符输入流的抽象父类。Reader常用方法：

| 方法 | 方法作用 |
|-----------------------|--|
| int read() | 从输入流中读取 一个字符 数据并返回该字符数据，如果到达流的末尾，则返回 -1。 |
| int read(char[] cbuf) | 从输入流中读取多个字符，并存储在缓冲区数组 cbuf 中。返回已读取的字符数，如果已到达流的末尾，则返回 -1。 |
| abstract void close() | 关闭此输入流并释放与该流关联的所有系统资源。 |

4.2、Writer（字符输出流）

Writer 表示所有字符输出流的抽象父类。Writer常用方法

| 方法 | 方法作用 |
|---|-----------------------------------|
| void write(int c) | 将指定的一个字符数据c写入到输出流中。 |
| void write(char[] cbuf) | 把数组cbuf中cbuf.length 个字符数据写入到输出流中。 |
| abstract void write(char[] cbuf, int off,int len) | 把数组cbuf中从索引off 开始的len 个字符写入此输出流中。 |
| void write(String str) | 将str字符串数据写入到输出流中。 |
| abstract void flush() | 刷新此输出流并强制写出所有缓冲的输出字符。 |
| abstract void close() | 关闭此输入流并释放与该流关联的所有系统资源。 |

关于字符流Reader/Writer，它们是抽象父类，只是规定了流的传输单元和流动方向，没有确定数据源和目的地，所以，如果需要具体的读写操作，一定要找其子类，其子类明确了数据源和目的地。

子类的命名规则：数据源/目的地 + 流的传输单元和流动方向。例如：File + Reader / File + Writer

4.3、文件字符流（重点）

需求1：使用文件字符输入流，读取result2.txt文件中的数据(包含中文)

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

        // 需求：给定一个文件(a.txt),读取文件的内容到程序中。
        // 1> 确定数据源
        File file = new File("G:\\test\\a.txt");

        // 2> 建立管道
        FileReader reader = new FileReader(file);

        // 3> 读取（一次读取一个字符）
        int c;
```

```

        StringBuilder sb = new StringBuilder();
        while ( (c = reader.read()) != -1){
            sb.append((char)c);
        }
        System.out.println(sb.toString());

        // 4> 关闭操作
        reader.close();
    }
}

-----

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

        // 需求: 给定一个文件(a.txt),读取文件的内容到程序中。
        // 1> 确定数据源
        File file = new File("G:\\test\\a.txt");

        // 2> 建立管道
        FileReader reader = new FileReader(file);

        // 3> 读取 (一次读取多个字符到字符缓冲区)
        char[] cbuf = new char[2];
        int len;
        StringBuilder sb = new StringBuilder();
        while( (len = reader.read(cbuf)) != -1 ){
            sb.append(cbuf,0,len);
        }
        System.out.println(sb.toString());

        // 4> 关闭操作
        reader.close();
    }
}

```

需求2: 使用文件字符输出流, 把程序中数据保存到result2.txt文件, 操作中文

```

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

        // step 1: 确定目的地
        File file = new File("G:\\test\\d.txt");
        // step 2: 建立管道
        FileWriter writer = new FileWriter(file);
        // step 3: 操作IO
        // 一次写一个字符
        /*writer.write('a');
        writer.write('b');
        writer.write('c');
        writer.write('中');*/

        // 一次写多个字符
        String str = "abc中";
        char[] cbuf = str.toCharArray();
        writer.write(cbuf);
    }
}

```

```
// writer.write(str);

// step 4: 刷新缓冲区, 把字符一次性写入文件中(可以省略)
writer.flush();

// step 5: 关闭流通道(关闭流通道时, 自动flush一次)
writer.close();

    }
}
```

注意: FileWriter 写入文件时, 以平台默认字符集写入文本型文件。

字节流和字符流选用问题

使用记事本打开某个文件, 如果可以看到内容的就是文本文件, 否则可以暂时认为是二进制格式的。

一般的, **操作二进制文件(图片、音频、视频等)必须使用字节流。操作文本文件使用字符流**, 尤其是操作带有中文的文件, 使用字符流容易导致乱码, 因为使用字节流可能出现读取半个汉字的尴尬(汉字由两个或三个字节组成)。当然, 如果不清楚属于哪一类型文件, 都可以使用字节流。