# 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
- 掌握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。该类包含了文件的创建、删除、重命名、判断是否存在等方法。

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
- 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);
        }
    }
}
```

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 TestO5File {

    public static void printDir(File file) {

        // 1> 输出file对象表示的目录
        System.out.println(file.getPath());

        // 2> 获取file的子目录和子文件
        File[] subFileArr = file.listFiles();
        for( File subFile:subFileArr ) {
            if( subFile.isFile() ) {    // 文件
                System.out.println(subFile.getPath());
            }else {                     // 目录
                printDir(subFile);
            }
        }
    }

    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");
        printDir(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，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高位是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、常见的字符编码和操作（了解）

常见的字符集

- **ASCII：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：

存储字母和数字无论是什么字符集都占1个字节.

存储汉字，GBK家族占两个字节，UTF-8家族占3个字节。

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（掌握）

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

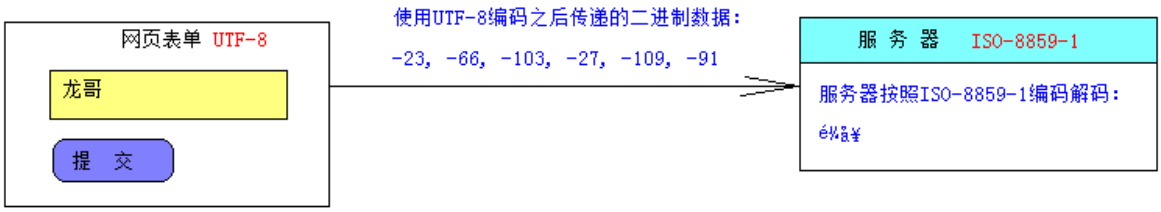
**注意：一定要保证编码和解码的字符集相同，才能正确解码出信息。**

经典案例：在表单中填写中文，为什么在服务端看到的是乱码问题。

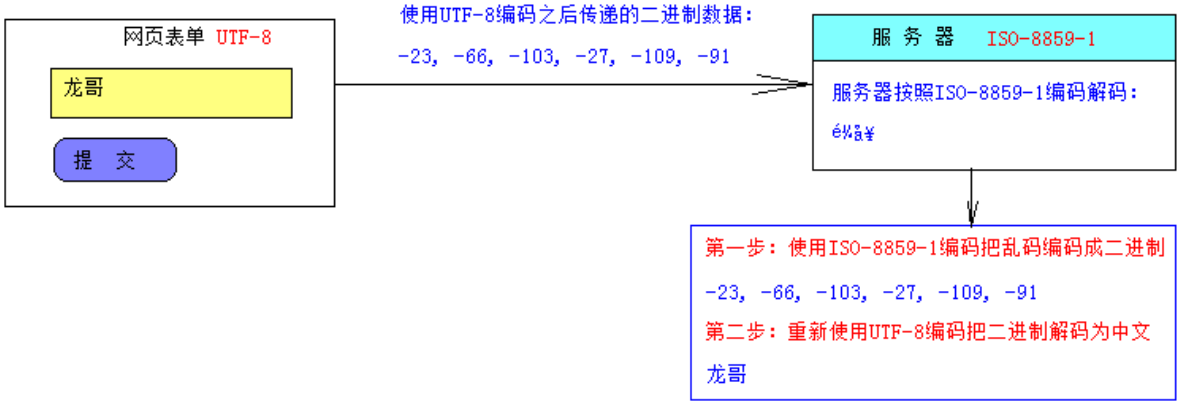
情景分析，比如浏览器使用UTF-8编码，服务器使用ISO-8859-1解码。

此时编码和解码的字符类型不同，那么乱码就出现了。

先来分析乱码产生的原因：




乱码的解决方案：




```
public class CodeDemo {
    public static void main(String[] args) throws Exception {
        String input = "龙哥";//模拟用户输入的中文数据

        // 编码操作：String -> byte[]
        byte[] data = input.getBytes("UTF-8");
        System.out.println(Arrays.toString(data));//[-23, -66, -103, -27, -109,
-91]

        // 解码操作：byte[] -> String
        // 因为服务器时老外写的,老外在解码的时候使用ISO-8859-1,此时就乱码了
        String ret = new String(data, "ISO-8859-1");
        System.out.println(ret);//输出：é¾•å•¥

        //-----------------------------------------
        // 解决方案：重新对乱码编码回到byte[],重新按照UTF-8解码
        data = ret.getBytes("ISO-8859-1");
        System.out.println(Arrays.toString(data));//[-23, -66, -103, -27, -109,
-91]
        ret = new String(data,"UTF-8");
        //-----------------------------------------
        System.out.println(ret);//输出：龙哥
    }
}
```

## 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：麦克风、扫描器、键盘、鼠标等
- 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）。

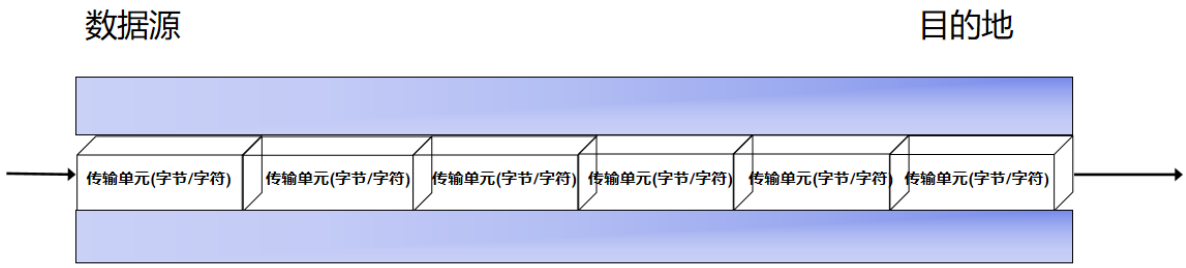
- 此时需要把游戏中的数据存储起来，数据只能存储在文件中。

案例2：打游戏操作，需求读取之前游戏的记录信息，数据存储在一个文件中的。

- 此时游戏程序需要去读取文件中的数据，并显示在游戏中。

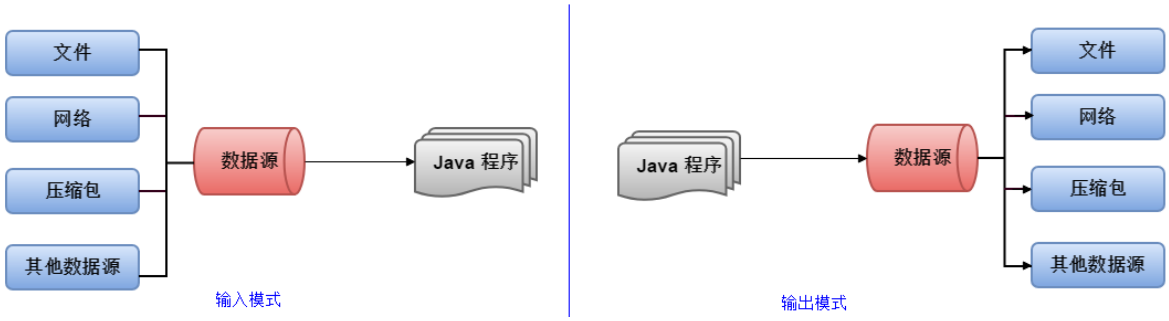
**流的概念**

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。


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？？？
- 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**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):创建源或者目标对象(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。<br>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。<br>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的超类。

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，并没有定义源数据源和目的地，java.io包中的类是按照源数据源和目的地进行划分的。

java.io包中的类命名规则：数据源/目的地 + 数据传输单元（流向）

### 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 Test02FileInputStream {
    public static void main(String[] args) throws IOException {

        // 需求：给定一个UTF-8编码的文本型文件(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
        FileInputStream in = new FileInputStream(file);
        // 3> 读取操作(一次读取一个)
        int c = 0;
        c = in.read();
        c = in.read();
        c = in.read();
        c = in.read();
        c = in.read();

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

        // 4> 关闭操作
        in.close();

    }
}
-------------------------------------------------------------------------------
 public class Test03FileInputStream {
    public static void main(String[] args) throws IOException {
        // 需求：给定一个UTF-8编码的文本型文件(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
        FileInputStream in = new FileInputStream(file);
        // 3> 读取操作
        int c = 0;
        StringBuilder sb = new StringBuilder();
        while( (c = in.read())  != -1 ){
            sb.append((char)c);
        }
        System.out.println(sb.toString());

        // 4> 关闭操作
        in.close();
    }
}
-------------------------------------------------------------------------------
 public class Test04FileInputStream {
    public static void main(String[] args) throws IOException {

        // 需求：给定一个UTF-8编码的文本型文件(a.txt),读取文件的内容到程序中。
        File file = new File("G:\\test\\a.txt");
```

```
        FileInputStream in = new FileInputStream(file);

        // 一次读取1k(1k = 1024byte)
        byte[] buf = new byte[1024];
        int len = 0;
        StringBuilder sb = new StringBuilder();
        while( (len = in.read(buf) ) != -1 ){
            String str = new String(buf,0,len,"UTF-8");
            sb.append(str);
        }
        System.out.println(sb.toString());

        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，操作英文**

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

        // 需求：把程序中的数据(中国你好)以GBK编码写入到 G:\\test\\b.txt;
        // 1> 确定目的地
        File file = new File("G:\\test\\b.txt");

        // 2> 建立管道
        FileOutputStream out = new FileOutputStream(file);

        // 3> 写出操作
        // out.write('a');

        String str = "中国你好";
        // 以指定的GBK字符编码写入文件
        byte[] buf = str.getBytes("GBK");
        out.write(buf);

        // 4> flush
        out.flush();

        // 5> 关闭流
        out.close();

    }
}
```