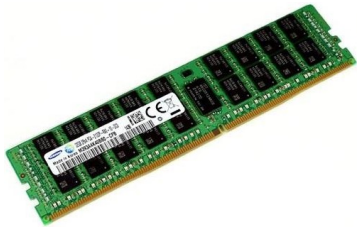
File类、IO流

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

1、我们目前的编程中，是怎么样存储数据的？弊端是什么？

```
int [] arr = new int[3];
List<String> list = new ArrayList<>();
```



- 在内存中存储的数据是不能长久保存的。

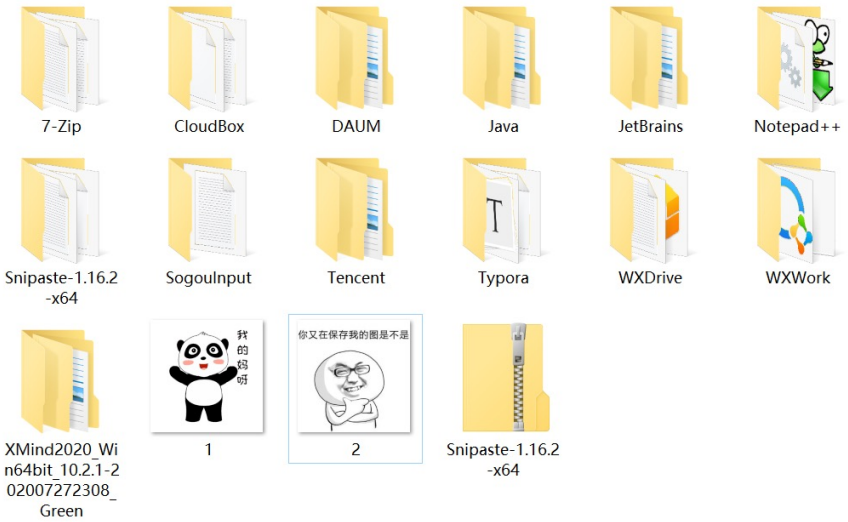
2、数据希望永久存储，怎么办？

- 使用文件的形式存储。
- 今日重点：1、围绕文件的操作　2、读写数据。

File　　IO流

File类概述

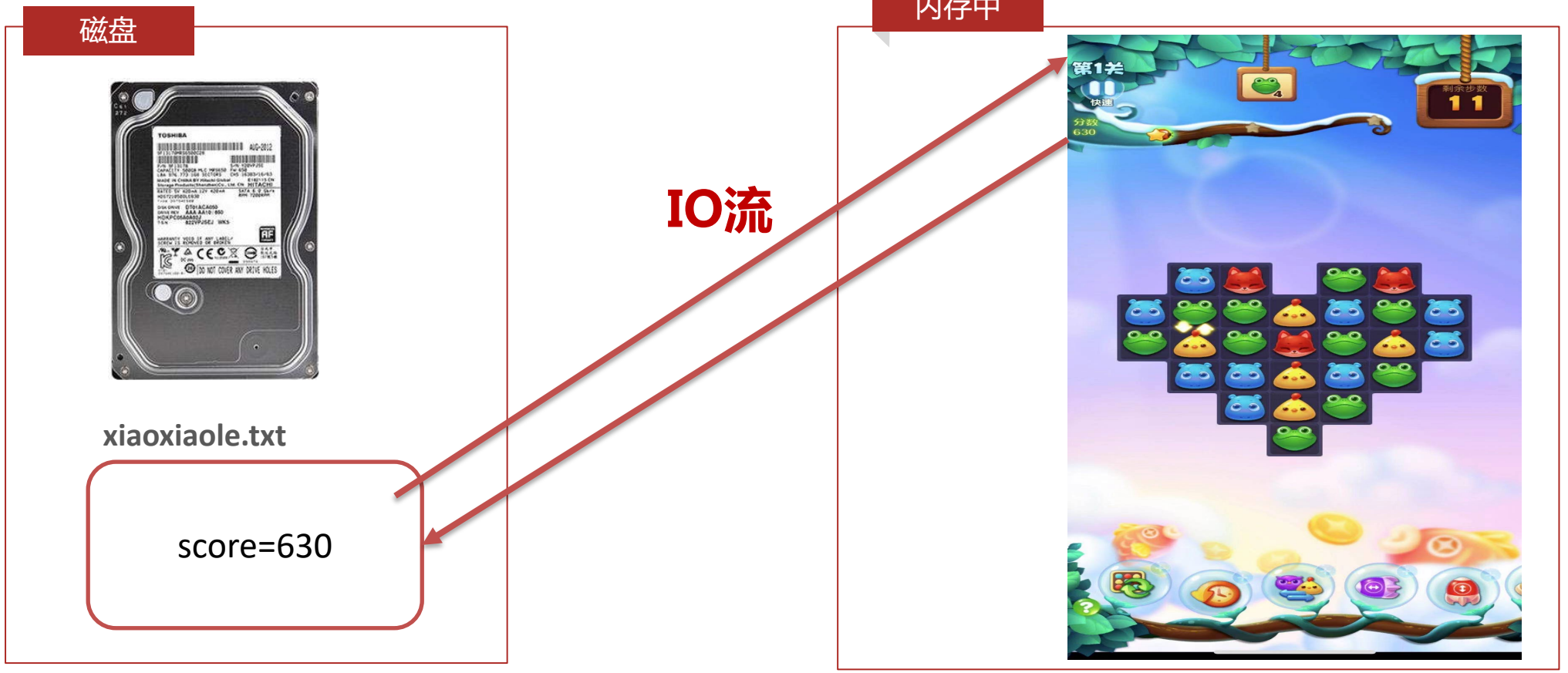
- File类的对象代表操作系统的文件（文件、文件夹），File类在java.io.File包下。



- File类提供了诸如：创建文件对象代表文件，获取文件信息（大小、修改时间）、删除文件、创建文件（文件夹）等功能。

IO流

- 用来读写数据的。



关于File、IO流，同学们需要学会什么

File类使用	方法递归	字符集	IO流的作用、分类	字节流
File类的对象代表操作系统的文件，File封装了对文件进行：删除、获取信息、创建文件\文件夹等操作。	递归是一种算法，可以解决一些问题（文件搜索）。需要知道递归是什么形式，具体如何使用。	要读写数据，必须先知道数据的底层形式。	IO流是用来读写文件数据的，IO流为了解决不同的业务场景下读写数据的需求，提供了不同特点的流。	按照字节的方式读写数据，可以进行文件数据的读取、文件拷贝等操作。

目录

Contents

File类创建对象

方法名称	说明
`public File (String pathname)`	根据文件路径创建文件对象
`public File (String parent, String child)`	根据父路径名字符串和子路径名字符串创建文件对象
`public File (File  parent, String child)`	根据父路径对应文件对象和子路径名字符串创建文件对象

注意

- File对象可以定位文件和文件夹
- File封装的对象仅仅是一个路径名，这个路径可以是存在的，也可以是不存在的。

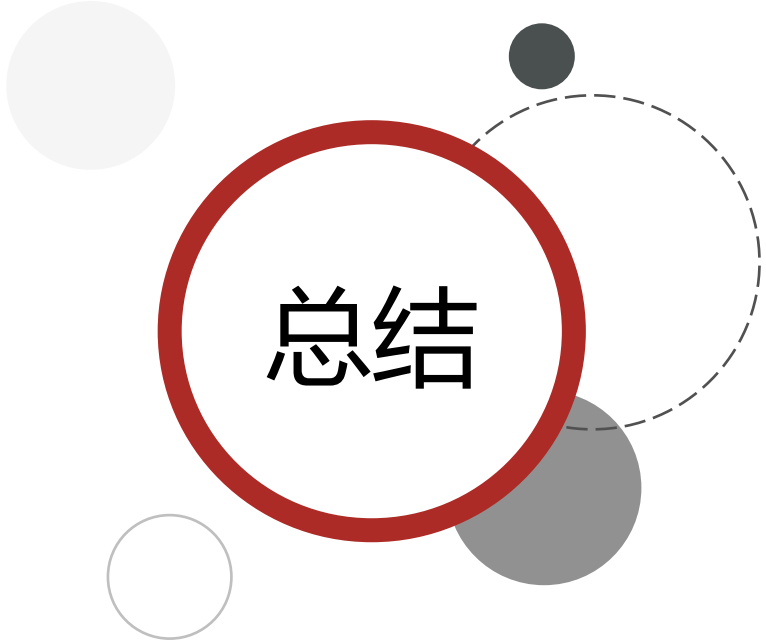
绝对路径和相对路径

- 绝对路径：从盘符开始

```
File file1 = new File("D:\\itheima\\a.txt");
```

- 相对路径：不带盘符，默认直接到当前工程下的目录寻找文件。

```
File file3 = new File("模块名\\a.txt");
```



1、File类的作用？

- **创建对象定位文件，可以删除、获取文件信息等。但是不能读写文件内容。**

2、File类构建对象的方式 ？

- **File file = new File(“文件/文件夹/绝对路径/相对路径”);**

3、绝对路径和相对路径是什么样的？

- **绝对路径是带盘符的。**
- **相对路径是不带盘符的，默认到当前工程下寻找文件。**

目录

Contents

File类的判断文件类型、获取文件信息功能

方法名称	说明
public boolean isDirectory()	判断此路径名表示的File是否为文件夹
public boolean isFile()	判断此路径名表示的File是否为文件
public boolean exists()	判断此路径名表示的File是否存在
public long length()	返回文件的大小（字节数量）
public String getAbsolutePath()	返回文件的绝对路径
public String getPath()	返回定义文件时使用的路径
public String getName()	返回文件的名称，带后缀
public long lastModified()	返回文件的最后修改时间（时间毫秒值）

目录

Contents

File类创建文件的功能

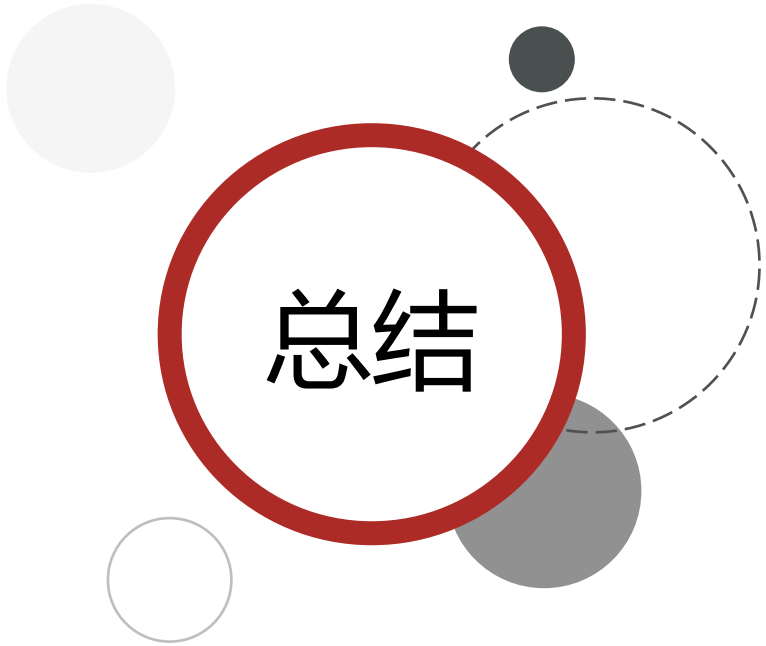
方法名称	说明
public boolean createNewFile()	创建一个新的空的文件
public boolean mkdir()	只能创建一级文件夹
public boolean mkdirs()	可以创建多级文件夹

File类删除文件的功能

方法名称	说明
public boolean delete ()	删除由此抽象路径名表示的文件或空文件夹

注意

- delete方法默认只能删除文件和空文件夹，delete方法直接删除不走回收站



1. 创建多级目录使用哪个方法？
    - **public boolean mkdirs()**
2. 删除文件需要注意什么？
    - **可以删除文件、空文件夹。**
    - **默认不能删除非空文件夹。**

目录

Contents

File类的遍历功能

方法名称	说明
public String[] list()	获取当前目录下所有的"一级文件名称"到一个字符串数组中去返回。
public File[] listFiles()(常用)	获取当前目录下所有的"一级文件对象"到一个文件对象数组中去返回（重点）

listFiles方法注意事项：

- 当文件不存在时或者代表文件时，返回null
- 当文件对象代表一个空文件夹时，返回一个长度为0的数组
- 当文件对象是一个有内容的文件夹时，将里面所有文件和文件夹的路径放在File数组中返回
- 当文件对象是一个有隐藏文件的文件夹时，将里面所有文件和文件夹的路径放在File数组中返回，包含隐藏文件
- 当没有权限访问该文件夹时，返回null

总结

1. 如何遍历文件夹下的文件对象，使用哪个API，有什么特点？

- **`public File[] listFiles()`(常用)。**
- **只能遍历当前文件夹对象下的一级文件对象。**

目录

Contents

什么是方法递归？

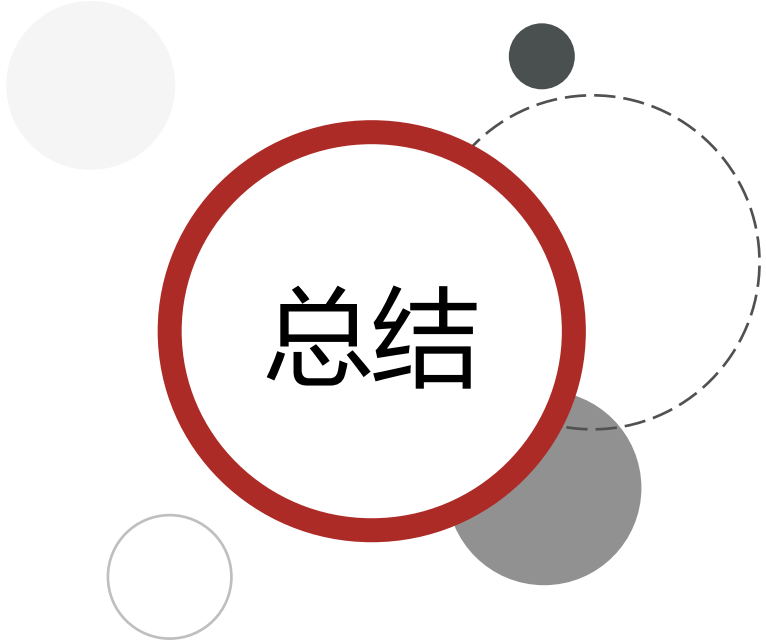
- 递归做为一种算法在程序设计语言中广泛应用。
- 方法调用自身的形式称为方法递归（ recursion ）。

递归的形式

- 直接递归：方法自己调用自己。
- 间接递归：方法调用其他方法，其他方法又回调方法自己。

方法递归存在的问题？

- 递归如果没有控制好终止，会出现递归死循环，导致栈内存溢出现象。



1. 什么是递归？
    - **方法调用自己的形式称为方法递归（ recursion ）。**
2. 什么是递归死循环？
    - **方法无限调用自己，无法终止，最终引起栈内存溢出。**

目录

Contents

- File类的使用
- 补充知识：方法递归
  - 方法递归的形式
  - 方法递归的应用、执行流程、递归算法的三个核心要素
  - 方法递归的经典案例
  - 其他形式的方法递归案例
- 补充知识：字符集
- IO流：概述
- IO流：字节流
- IO流：资源释放的方式

案例

递归案例导学-计算1-n的阶乘

需求：计算1-n的阶乘的结果，使用递归思想解决，我们先从数学思维上理解递归的应用和核心点。

分析

① 假如我们认为存在一个公式是 f(n) = 1*2*3*4*5*6*7*...(n-1)*n;

② 那么公式等价形式就是： f(n) = f(n-1) *n

③ 如果求的是 1-5的阶乘 的结果，我们手工应该应该如何应用上述公式计算。

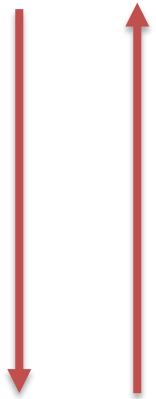
④ f(5) = f(4) * 5

f(4) = f(3) * 4

f(3) = f(2) * 3

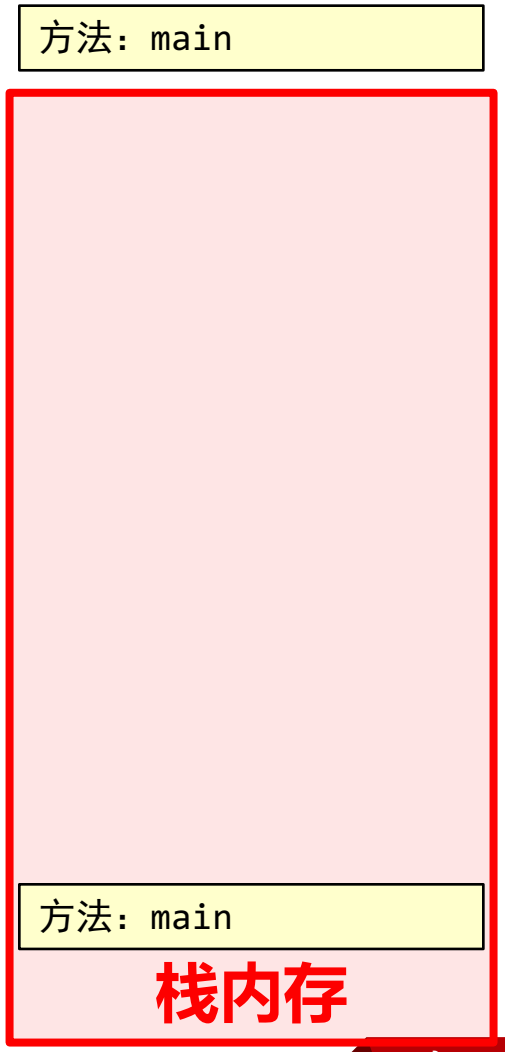
f(2) = f(1) * 2

f(1) = 1



案例：递归求阶乘的执行流程

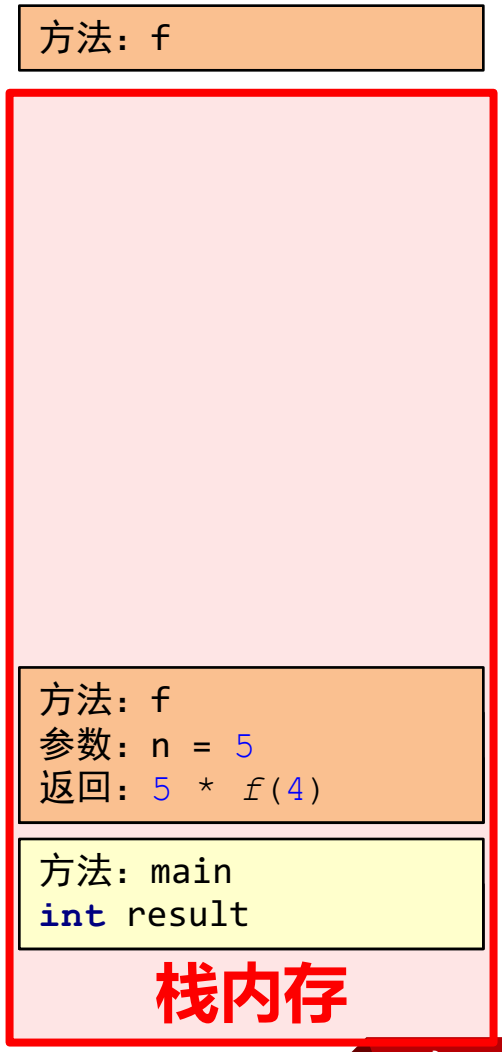
```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
    }
}
```



案例：递归求阶乘的执行流程

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
    }
}
```

```
public static int f(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * f(n - 1);
    }
}
```

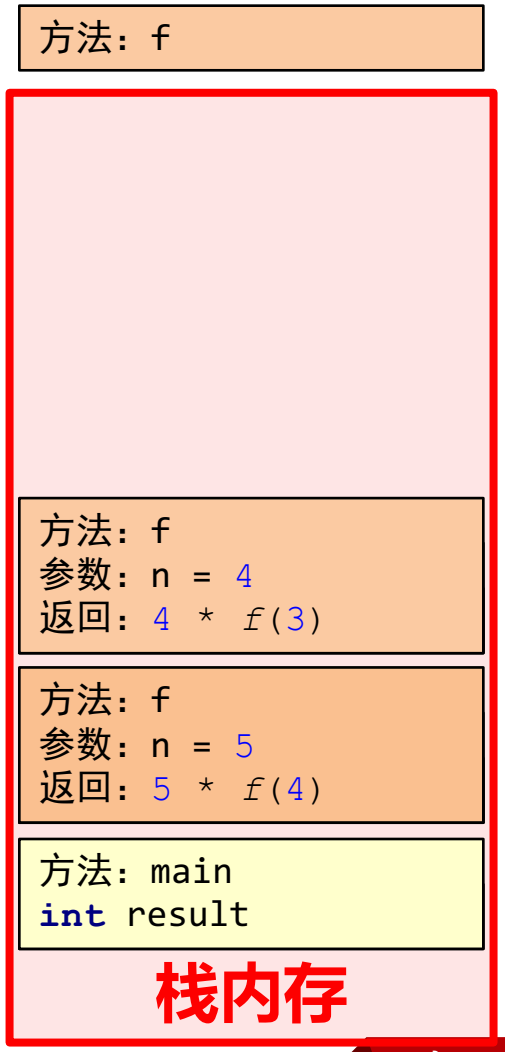


案例：递归求阶乘的执行流程

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
```

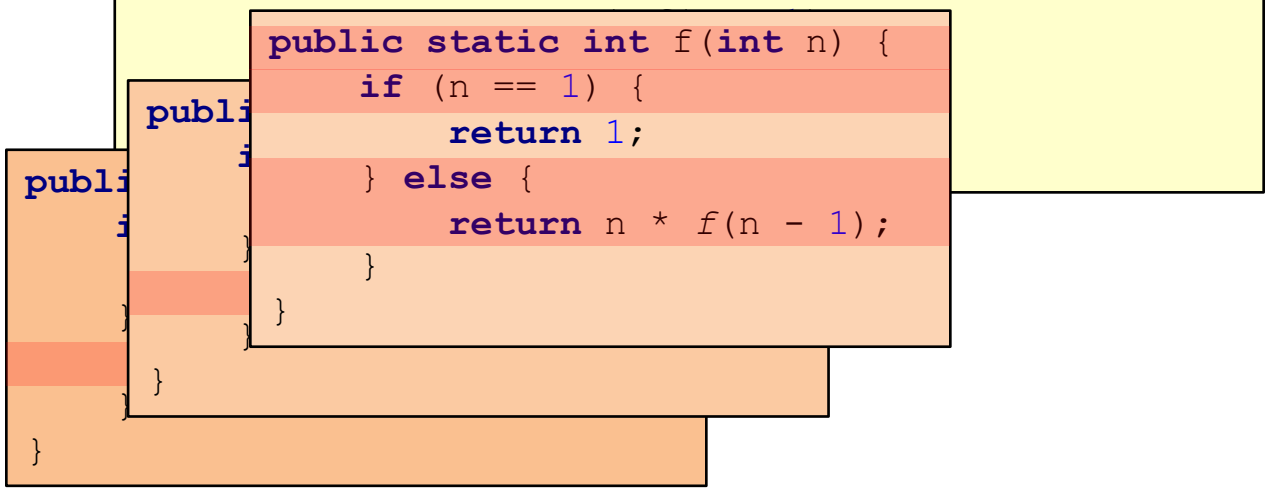
```
public static int f(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * f(n - 1);
    }
}
```

```
}
```



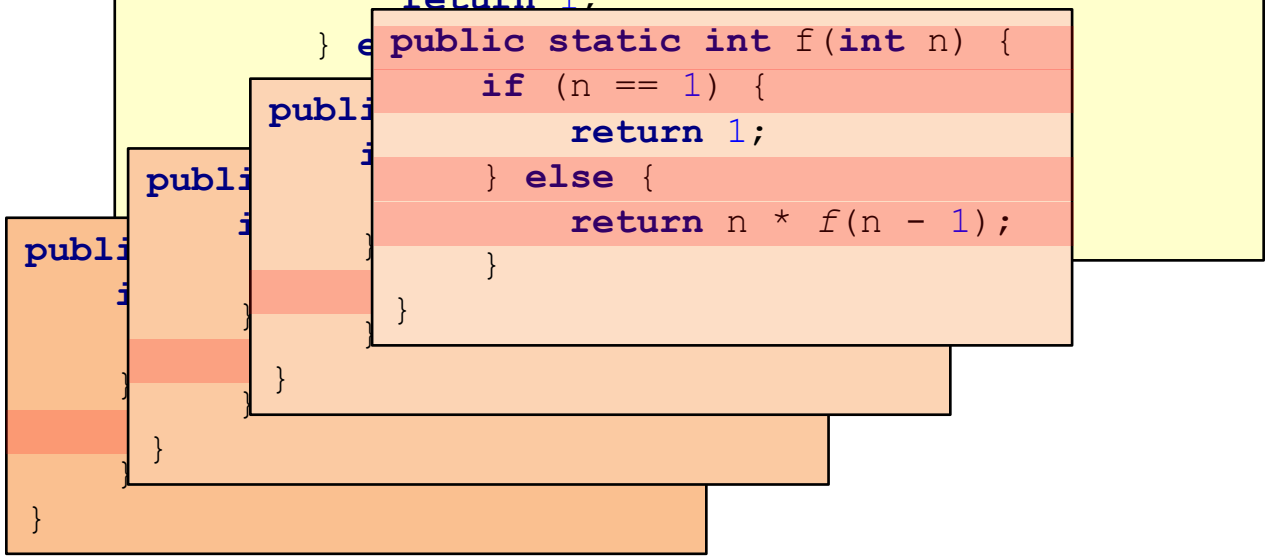
案例：递归求阶乘的执行流程

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
    }
}
```



案例：递归求阶乘的执行流程

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
    }
}
```



方法：f

方法：f
参数：n = 2
返回：2 * f(1)

方法：f
参数：n = 3
返回：3 * f(2)

方法：f
参数：n = 4
返回：4 * f(3)

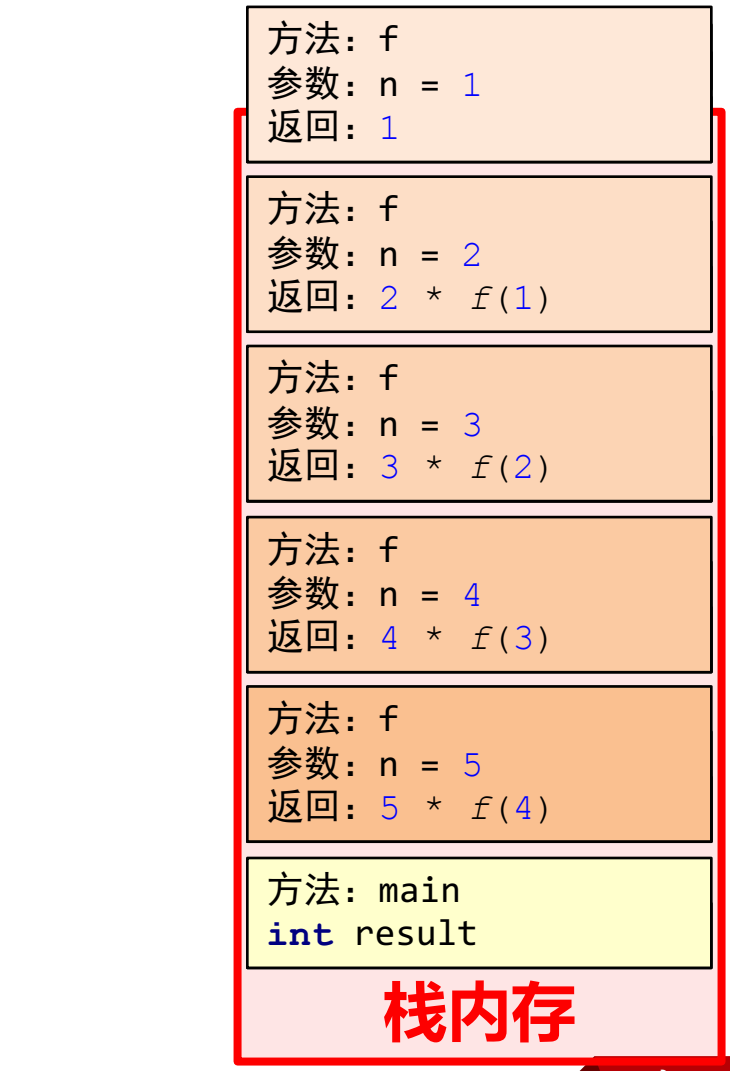
方法：f
参数：n = 5
返回：5 * f(4)

方法：main
int result

栈内存

案例：递归求阶乘的执行流程

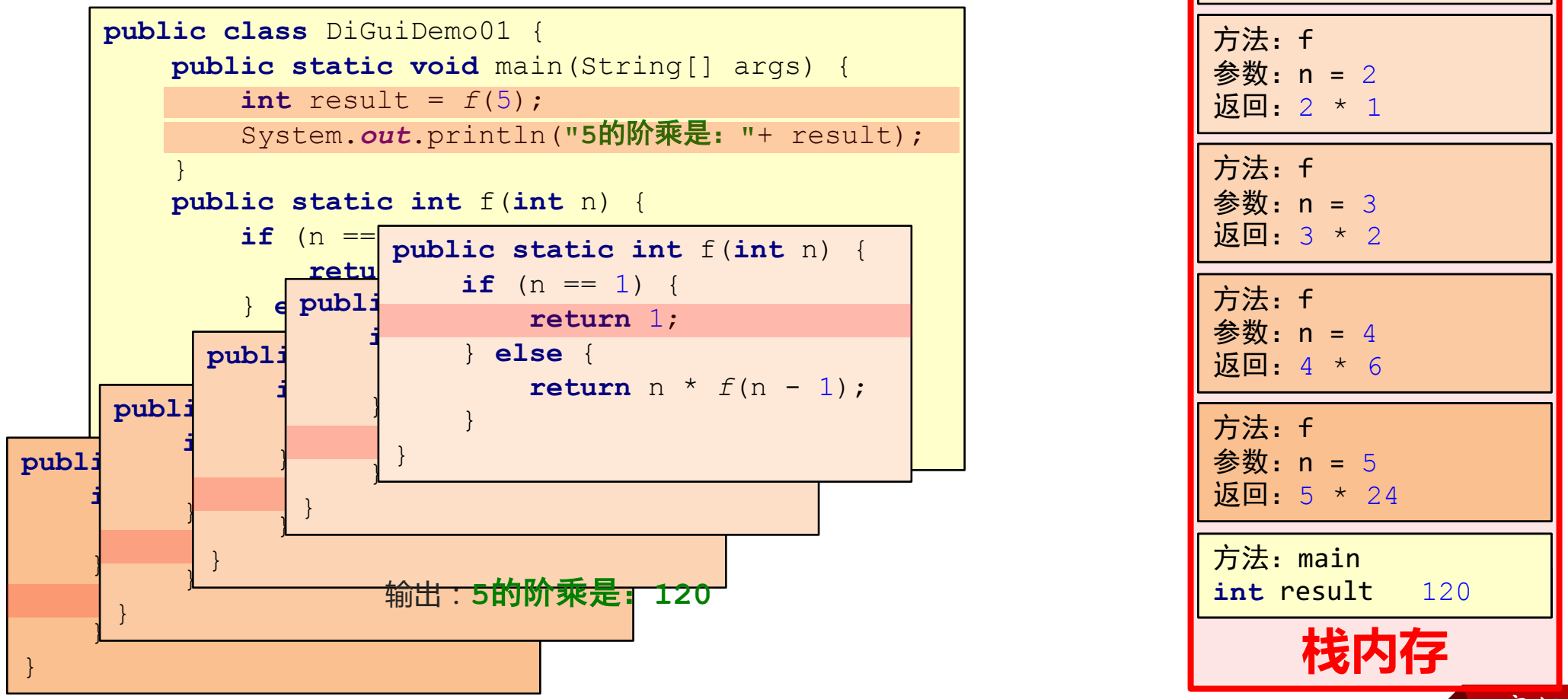
```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n ==
            retu
        } e
            publi
                i
                publi
                    i
                    publi
                        i
                        publi
                            i
                            public static int f(int n) {
                                if (n == 1) {
                                    return 1;
                                } else {
                                    return n * f(n - 1);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```



案例：递归求阶乘的执行流程

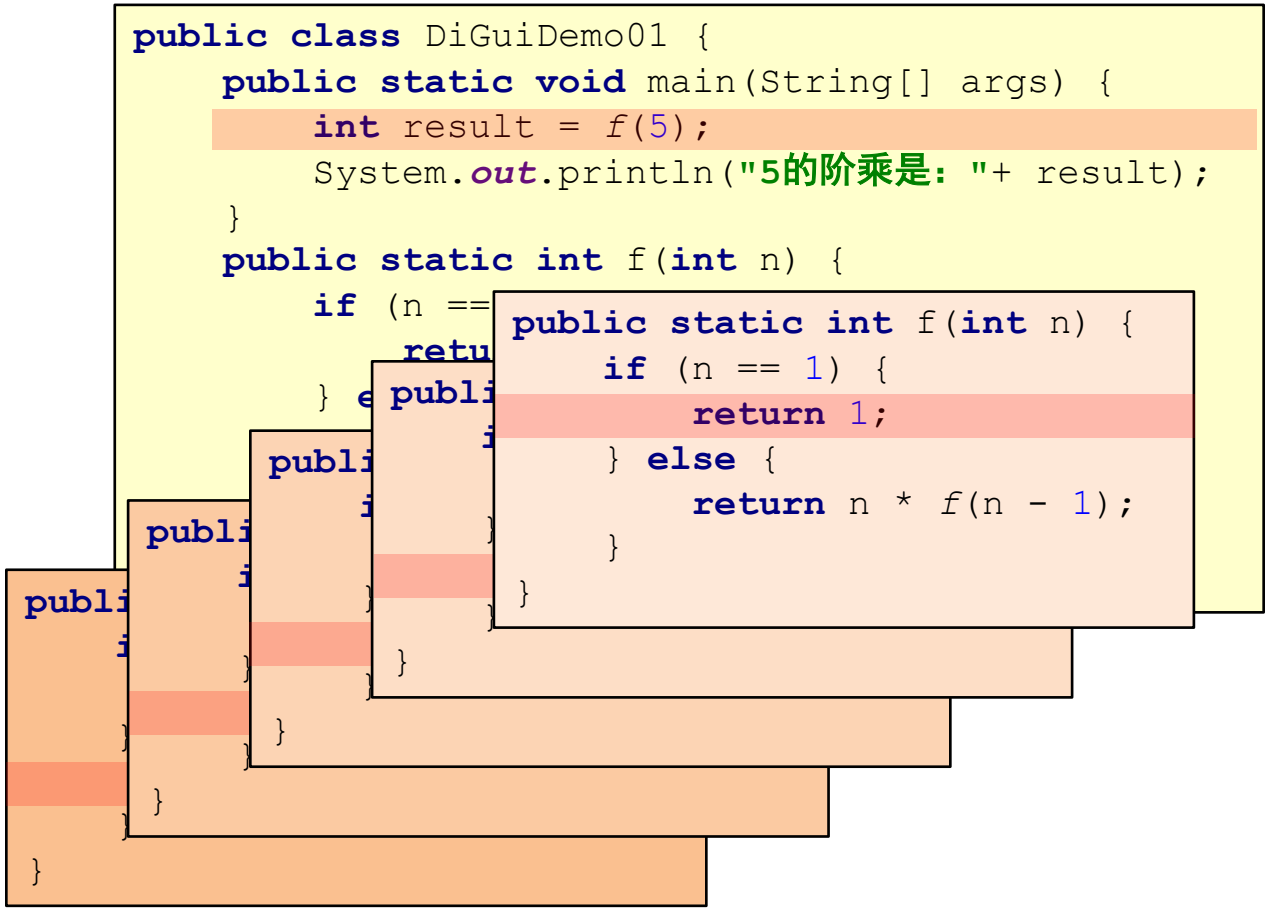
```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
    }
}
```

输出：5的阶乘是：120



案例：递归求阶乘的执行流程

```
public class DiGuiDemo01 {
    public static void main(String[] args) {
        int result = f(5);
        System.out.println("5的阶乘是："+ result);
    }
    public static int f(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * f(n - 1);
        }
    }
```



递归算法三要素大体可以总结为：

- **递归的公式： f(n) = f(n-1) * n;**
- **递归的终结点：f(1)**
- **递归的方向必须走向终结点：**

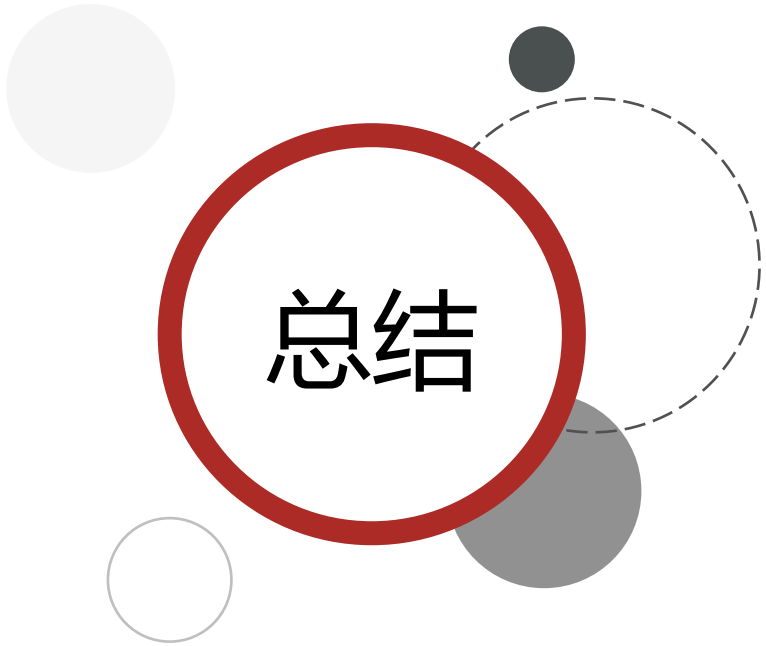
f(5) = f(4) * 5

f(4) = f(3) * 4

f(3) = f(2) * 3

f(2) = f(1) * 2

f(1) = 1



1. 递归算法三要素大体可以总结为：

- **递归的公式： f(n) = f(n-1) * n;**
- **递归的终结点：f(1)**
- **递归的方向必须走向终结点：**

目录

Contents

案例

递归求1-n的和

需求：计算1-n的和的结果，使用递归思想解决，我们先从数学思维上理解递归的流程和核心点。

分析

① 假如我们认为存在一个公式是 f(n) = 1 + 2 + 3 + 4 + 5 + 6 + 7 + ...(n-1) + n;

② 那么公式等价形式就是： f(n) = f(n-1) + n

③ 递归的终结点：f(1) = 1

④ 如果求的是 1-5的和 的结果，应该如何计算。

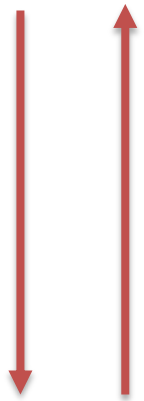
⑤ f(5) = f(4) + 5

f(4) = f(3) + 4

f(3) = f(2) + 3

f(2) = f(1) + 2

f(1) = 1



案例 猴子吃桃问题

- 猴子第一天摘下若干桃子，当即吃了一半，觉得好不过瘾，于是又多吃了一个

第二天又吃了前天剩余桃子数量的一半，觉得好不过瘾，于是又多吃了一个

以后每天都是吃前天剩余桃子数量的一半，觉得好不过瘾，又多吃了一个

等到第10天的时候发现桃子只有1个了。

需求：请问猴子第一天摘了多少个桃子？

分析：

① 整体来看，每一天都是做同一个事件，典型的规律化问题，考虑递归三要素：

② 递归公式：

③ 递归终结点：

④ 递归方向：

目录

Contents

- 在上述的案例中递归算法都是针对存在规律化的递归问题。
- 有很多问题是非规律化的递归问题，比如文件搜索。如何解决？

案例 文件搜索

需求：文件搜索、从C:盘中，搜索出某个文件名称并输出绝对路径。

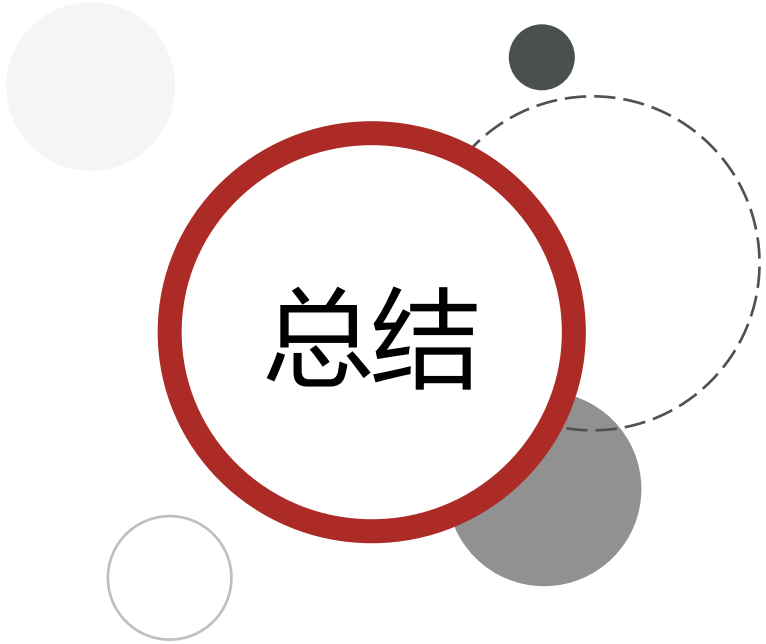
分析：

① 先定位出的应该是一级文件对象

② 遍历全部一级文件对象，判断是否是文件

③ 如果是文件，判断是否是自己想要的

④ 如果是文件夹，需要继续递归进去重复上述过程



1. 文件搜索用到了什么技术?
   - **递归，listFile只是搜索到了一级文件对象。**

案例 啤酒问题

需求：

啤酒2元1瓶，4个盖子可以换一瓶，2个空瓶可以换一瓶，

请问10元钱可以喝多少瓶酒，剩余多少空瓶和盖子。

答案：

15瓶 3盖子 1瓶子

练习

删除文件夹[拓展]

需求：删除非空文件夹

分析：

①：File默认不可以删除非空文件夹

②：我们需要遍历文件夹，先删除里面的内容，再删除自己。

目录

Contents

字符集基础知识：

- 字符集（Character Set）是多个字符的集合，字符集种类较多，每个字符集包含的字符个数不同，常见字符集有：
  - ASCII字符集
  - GBK字符集
  - Unicode（UTF-8）字符集等。

ASCII字符集：

- ASCII(American Standard Code for Information Interchange，美国信息交换标准代码)：包括了数字、英文、符号。
- ASCII使用1个字节存储一个字符，一个字节是8位，总共可以表示128个字符信息，对于表示英文、数字来说是够用的。

```
01100001  = 97  => a
01100010  = 98  => b
```

GBK：

- GBK是中国的码表，包含了几万个汉字等字符，同时也要兼容ASCII编码，
- GBK编码中一个中文字符一般以两个字节的形式存储。

Unicode字符集：

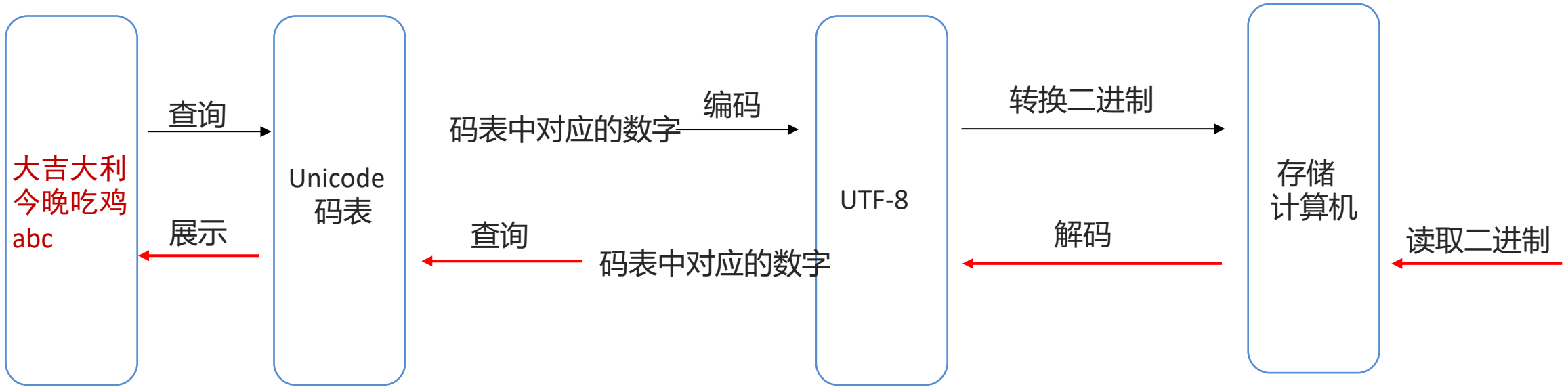
- 统一码，也叫万国码。是计算机科学领域里的一项业界标准。
- UTF-8是Unicode的一种常见编码方式。

注意

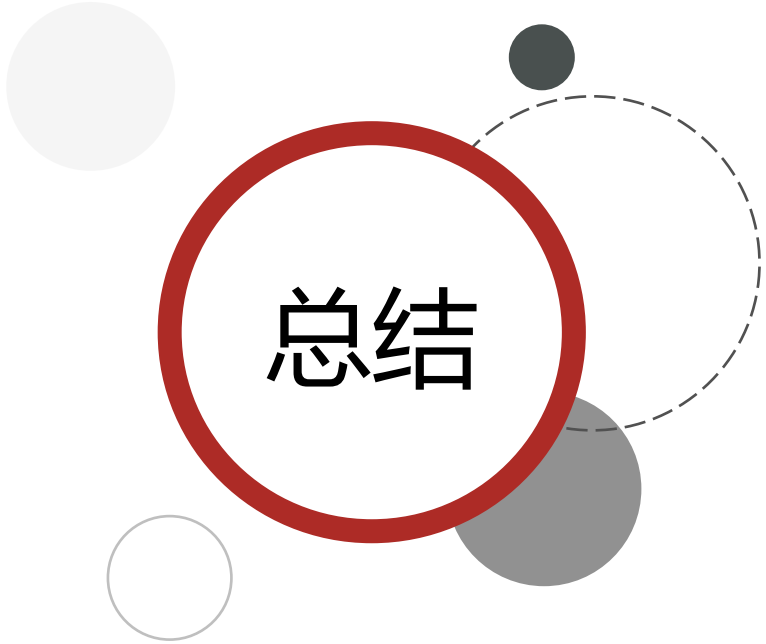
- UTF-8编码后一个中文一般以**三个字节**的形式存储，同时也要兼容ASCII编码表。
- 技术人员都应该使用UTF-8的字符集编码。

注意：**字符解码时使用的字符集和编码时使用的字符集必须一致，否则会出现乱码。**

汉字存储和展示过程解析



注意：**英文和数字在任何国家的编码中都不会乱码。**



1. 常见字符集底层字符的编码是什么样的？
   - **英文和数字等在任何国家的字符集中都占1个字节**
   - **GBK字符中一个中文字符占2个字节**
   - **UTF-8编码中一个中文1般占3个字节**
2. 编码前的字符集和解码时的字符集有什么要求？
   - **必须一致，否则会出现字符乱码**
   - **英文和数字不会乱码**

目录

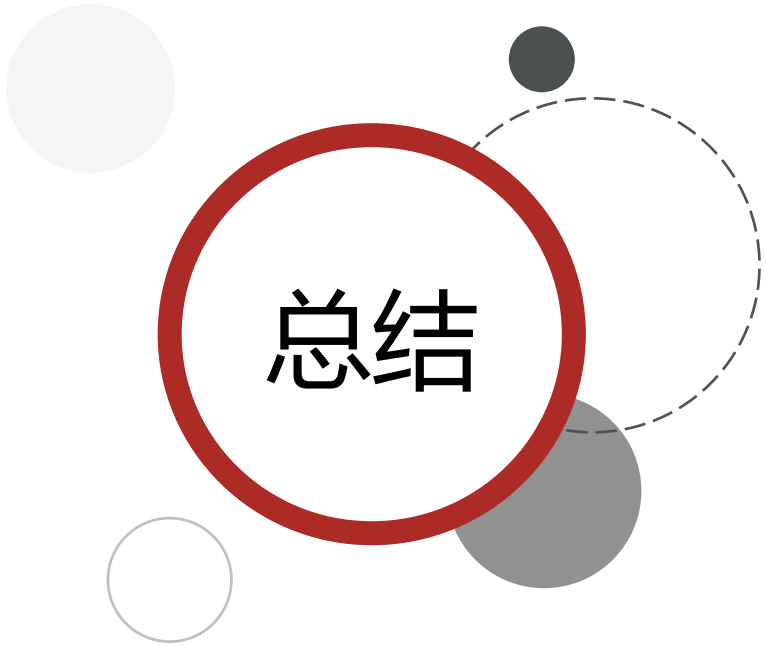
Contents

String编码

方法名称	说明
byte[] getBytes()	使用平台的默认字符集将该 String编码为一系列字节，将结果存储到新的字节数组中
byte[] getBytes(String charsetName)	使用指定的字符集将该 String编码为一系列字节，将结果存储到新的字节数组中

String解码

构造器	说明
`String (byte[] bytes)`	通过使用平台的默认字符集解码指定的字节数组来构造新的 String
`String (byte[] bytes, String charsetName)`	通过指定的字符集解码指定的字节数组来构造新的 String



1. 如何使用程序对字符进行编码？
   - **String类下的方法：**
   - **byte[] getBytes()：默认编码**
   - **byte[] getBytes(String charsetName)：指定编码**
2. 如何使用程序进行解码？
   - **String类的构造器：**
   - **`String (byte[] bytes)`：使用默认编码解码**
   - **`String (byte[] bytes, String charsetName))`：指定编码解码**

目录

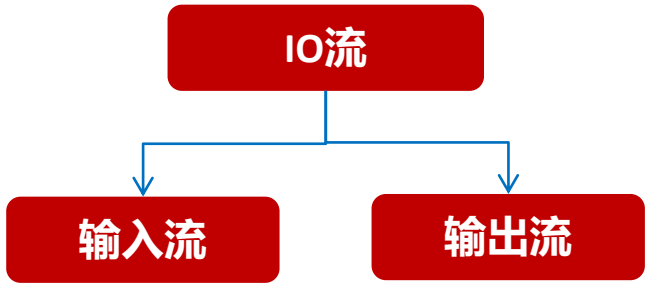
Contents

IO流概述

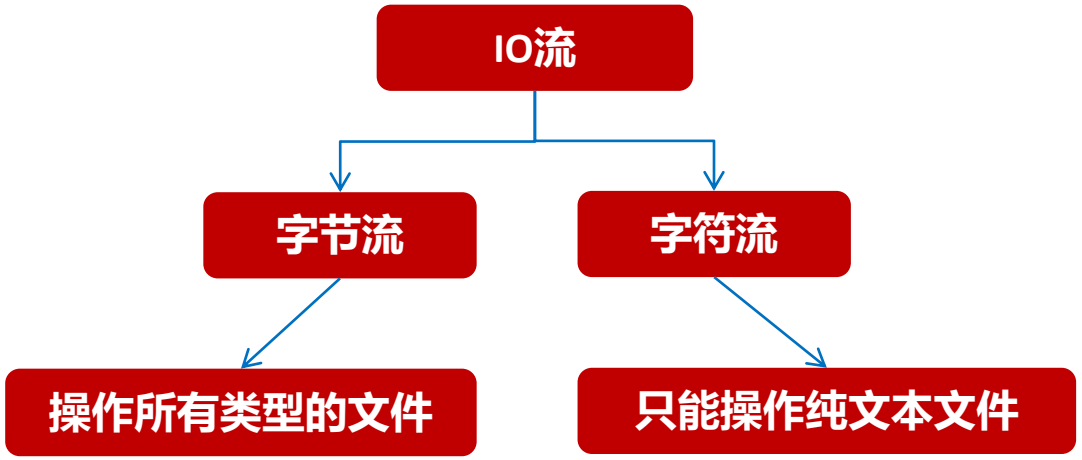
- I表示intput，把硬盘文件中的数据读入到内存的过程，称之输入，负责读。
- O表示output，把内存中的数据写出到硬盘文件的过程，称之输出，负责写。

IO流的分类

- 按流的方向分



- 按流中的数据最小单位分为

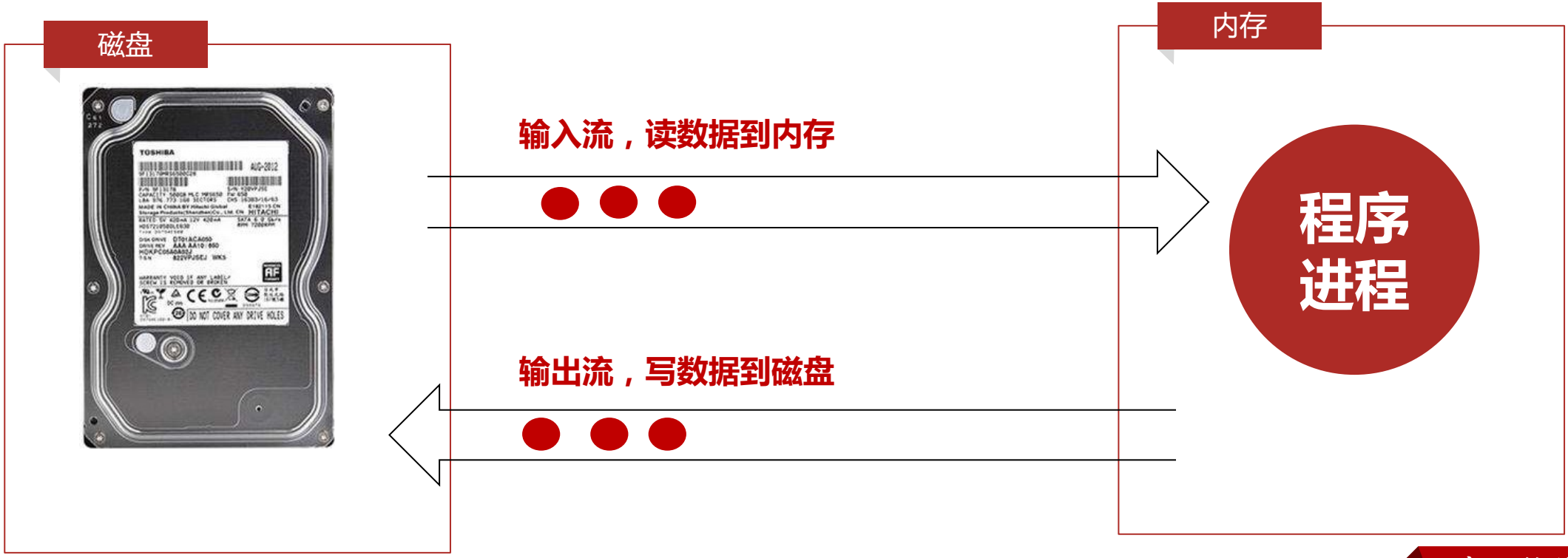


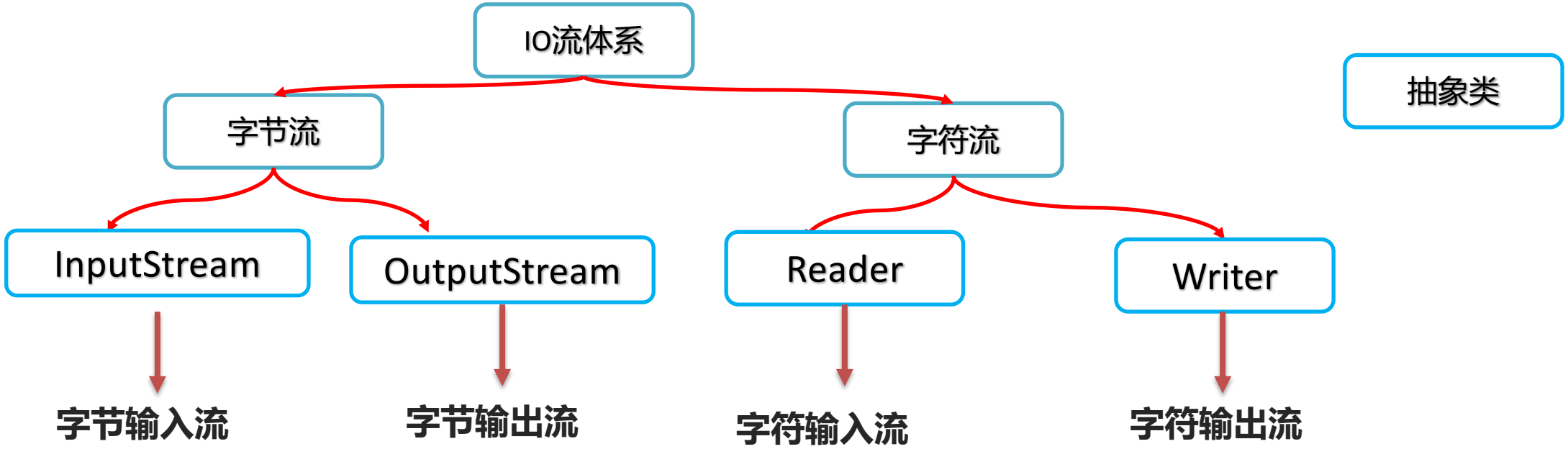
包括音频视频图片等

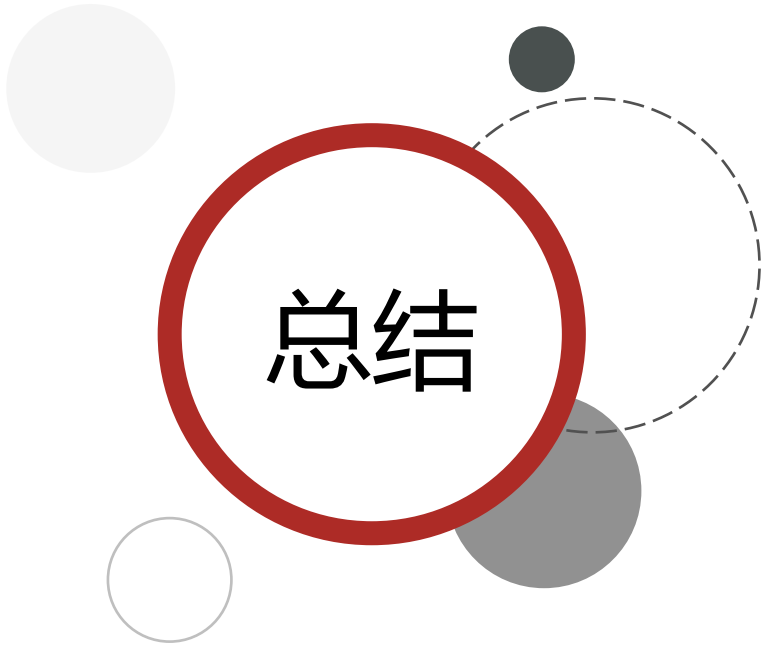
包括java文件，txt文件等

总结流的四大类:

- 字节输入流：以内存为基准，来自磁盘文件/网络中的数据以字节的形式读入到内存中去的流称为字节输入流。
- 字节输出流：以内存为基准，把内存中的数据以字节写出到磁盘文件或者网络中去的流称为字节输出流。
- 字符输入流：以内存为基准，来自磁盘文件/网络中的数据以字符的形式读入到内存中去的流称为字符输入流。
- 字符输出流：以内存为基准，把内存中的数据以字符写出到磁盘文件或者网络介质中去的流称为字符输出流。



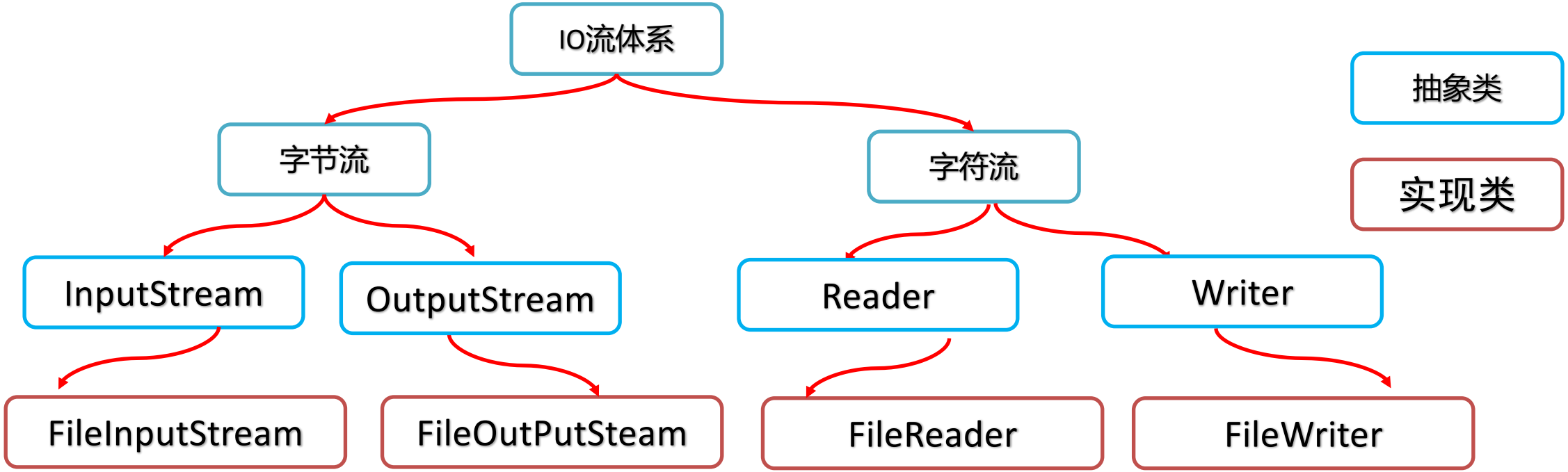

IO流体系
字节流
字符流
抽象类
InputStream
OutputStream
Reader
Writer
字节输入流
字节输出流
字符输入流
字符输出流



1. IO流的作用？
   - **读写文件数据的**
2. IO流是怎么划分的，大体分为几类，各自的作用？
   - **字节输入流 InputStream（读字节数据的）**
   - **字节输出流 OutoutStream（写字节数据出去的）**
   - **字符输入流 Reader（读字符数据的）**
   - **字符输出流 Writer（写字符数据出去的）**

目录

Contents


IO流体系
字节流
字符流
抽象类
实现类
InputStream
OutputStream
Reader
Writer
FileInputStream
FileOutPutSteam
FileReader
FileWriter

磁盘

文件字节输入流，一个一个字节读数据到内存

内存

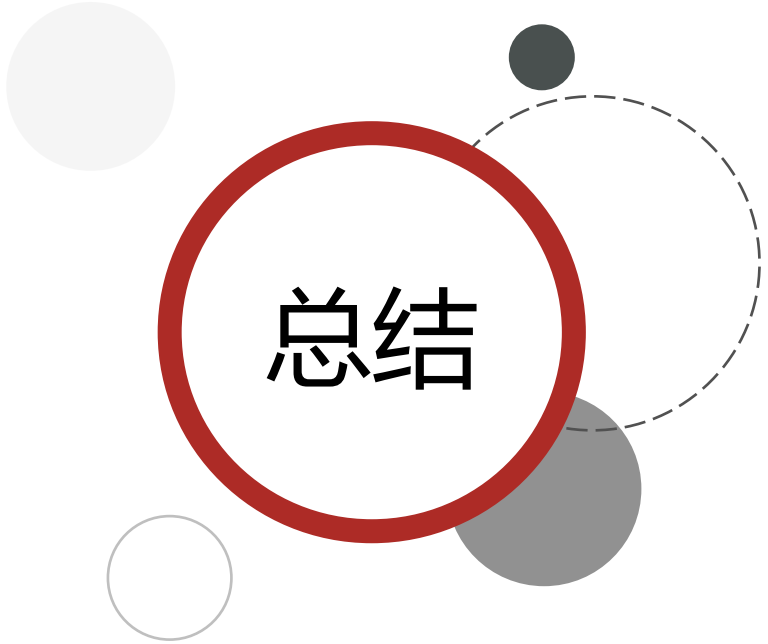
程序

文件字节输入流：FileInputStream

- 作用：以内存为基准，把磁盘文件中的数据以字节的形式读取到内存中去。

构造器	说明
public FileInputStream(File file)	创建字节输入流管道与源文件对象接通
public FileInputStream(String pathname)	创建字节输入流管道与源文件路径接通

方法名称	说明
public int read()	每次读取一个字节返回，如果字节已经没有可读的返回-1
public int read(byte[] buffer)	每次读取一个字节数组返回，如果字节已经没有可读的返回-1



1. 文件字节输入流，每次读取一个字节的api是哪个？

方法名称	说明
public int read()	每次读取一个字节返回，如果字节已经没有可读的返回-1

2. 每次读取一个字节存在什么问题？

- **性能较慢**
- **读取中文字符输出无法避免乱码问题。**

目录

Contents

磁盘

文件字节输入流，一个个字节数组读数据到内存

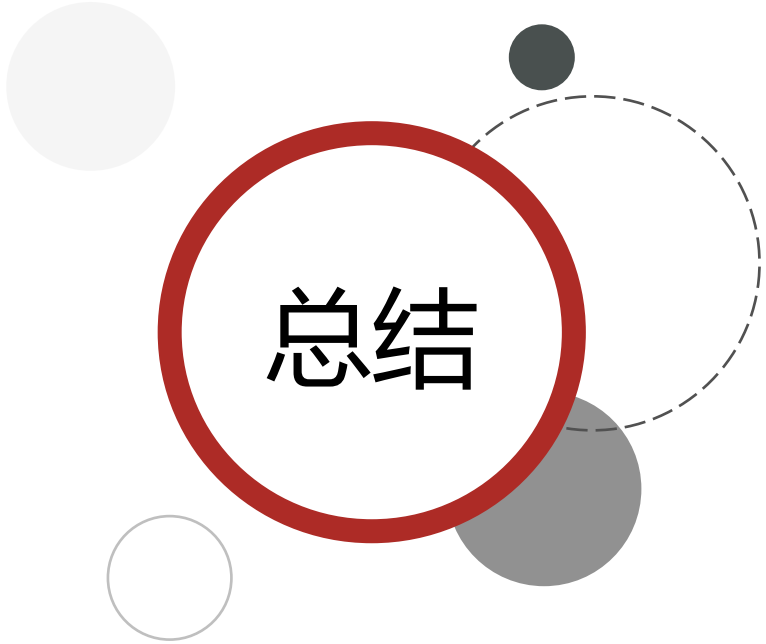
内存

程序
进程

文件字节输入流：FileInputStream

- 作用：以内存为基准，把磁盘文件中的数据以字节的形式读取到内存中去。

方法名称	说明
public int read()	每次读取一个字节返回，如果字节已经没有可读的返回-1
public int read(byte[] buffer)	每次使用字节数组来读取数据，返回读取的字节个数，如果没有可读返回-1



1. 文件字节输入流，每次读取一个字节数组的api是哪个？

方法名称	说明
public int read(byte[] buffer)	每次读取一个字节数组返回，如果字节已经没有可读的返回-1

2. 每次读取一个字节数组存在什么问题？

- **读取的性能得到了提升**
- **读取中文字符输出无法避免乱码问题。**

1、使用字节流读取中文输出乱码，如何使用字节输入流读取中文输出不乱码呢？

- **定义一个与文件一样大的字节数组，一次性读取完文件的全部字节。**

2、直接把文件数据全部读取到一个字节数组可以避免乱码，是否存在问题？

- **如果文件过大，字节数组可能引起内存溢出。**

目录

Contents

- File类的使用
- 补充知识：方法递归
- 补充知识：字符集
- IO流：概述
- IO流：字节流
  - 字节输入流：每次读取一个字节
  - 字节输入流：每次读取一个字节数组
  - 字节输入流：读取文件的全部字节
  - 字节输出流：写字节数据到文件
  - 文件拷贝
- IO流：资源释放的方式

方式一

- 自己定义一个字节数组与文件的大小一样大，然后使用读取字节数组的方法，一次性读取完成。

方法名称	说明
public int read(byte[] buffer)	每次读取一个字节数组返回，如果字节已经没有可读的返回-1

方式二

- 官方为字节输入流InputStream提供了如下API可以直接把文件的全部数据读取到一个字节数组中

方法名称	说明
public byte[] readAllBytes() throws IOException	直接将当前字节输入流对应的文件对象的字节数据装到一个字节数组返回

1、直接把文件数据全部读取到一个字节数组可以避免乱码，是否存在问题？

- **如果文件过大，字节数组可能引起内存溢出。**

总结

1. 如何使用字节输入流读取中文内容输出不乱码呢？
   - **一次性读取完全部字节。**
   - **可以定义与文件一样大的字节数组读取，也可以使用官方API.**
2. 直接把文件数据全部读取到一个字节数组可以避免乱码，是否存在问题？
   - **如果文件过大，定义的字节数组可能引起内存溢出。**

目录

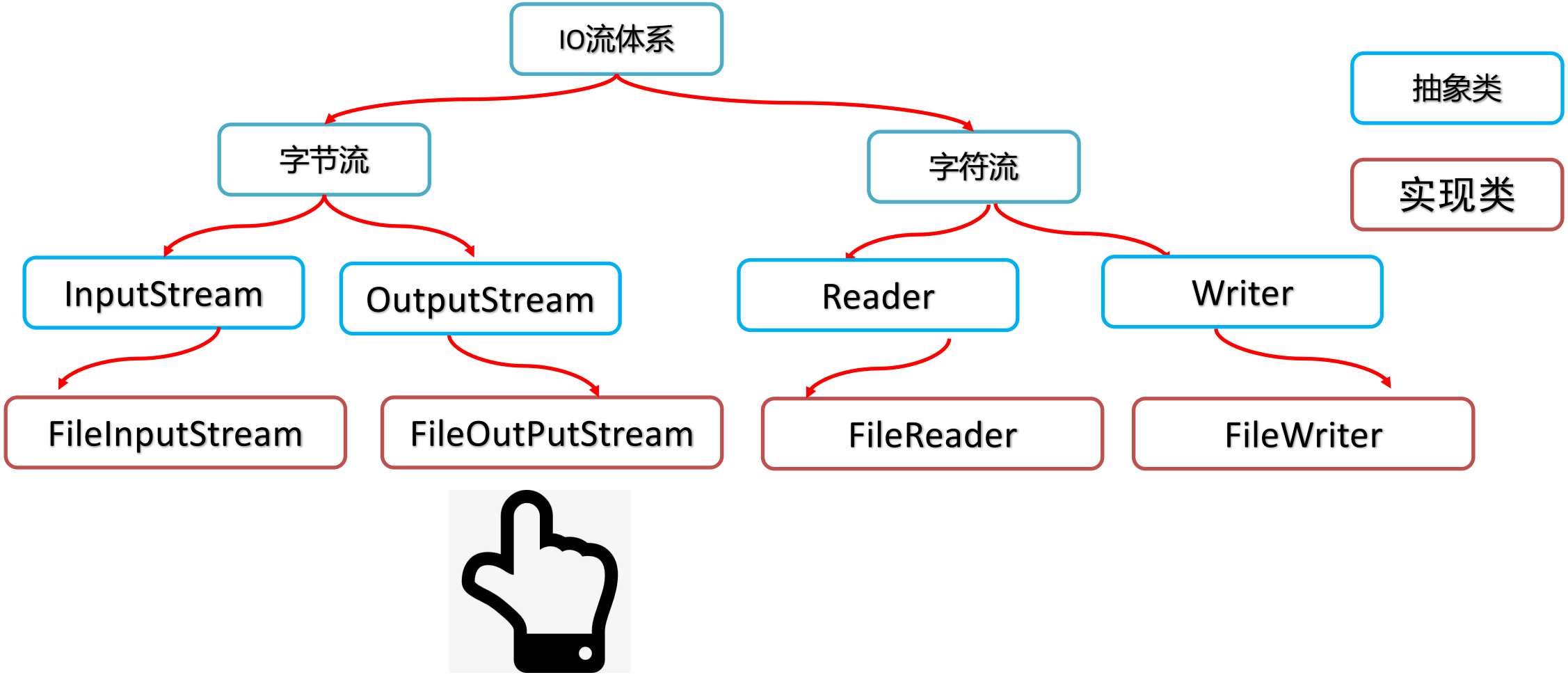
Contents

磁盘

文件字节输出流，写字节读数据到磁盘文件

内存

程序
进程


IO流体系
字节流
字符流
抽象类
实现类
InputStream
OutputStream
Reader
Writer
FileInputStream
FileOutPutStream
FileReader
FileWriter

文件字节输出流：FileOutputStream

- 作用：以内存为基准，把内存中的数据以字节的形式写出到磁盘文件中去的流。

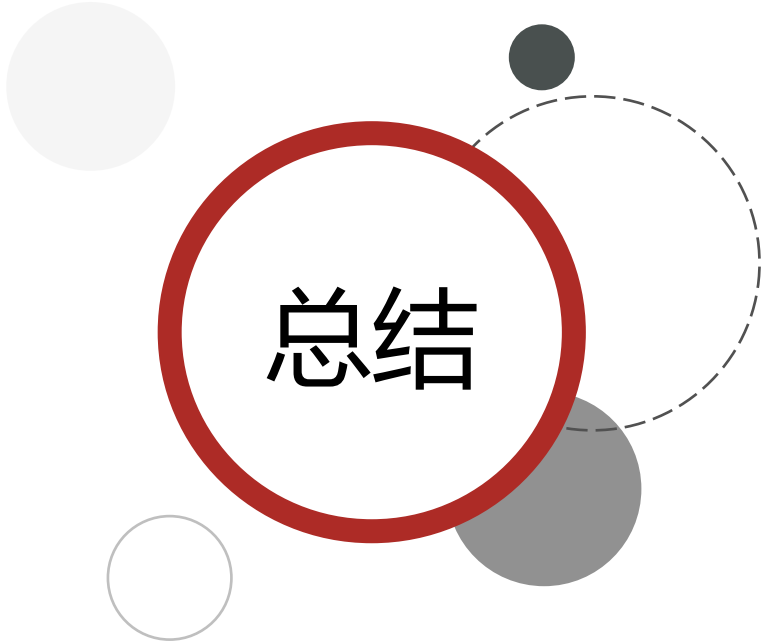
构造器	说明
`public FileOutputStream (File file)`	创建字节输出流管道与源文件对象接通
`public FileOutputStream (File file , boolean append)`	创建字节输出流管道与源文件对象接通，可追加数据
`public FileOutputStream (String filepath)`	创建字节输出流管道与源文件路径接通
`public FileOutputStream (String filepath , boolean append)`	创建字节输出流管道与源文件路径接通，可追加数据

文件字节输出流（FileOutputStream）写数据出去的API

方法名称	说明
`public void write(int a)`	写一个字节出去
`public void write(byte[] buffer)`	写一个字节数组出去
`public void write(byte[] buffer , int pos , int len)`	写一个字节数组的一部分出去。

流的关闭与刷新

方法	说明
flush()	刷新流，还可以继续写数据
close()	关闭流，释放资源，但是在关闭之前会先刷新流。一旦关闭，就不能再写数据

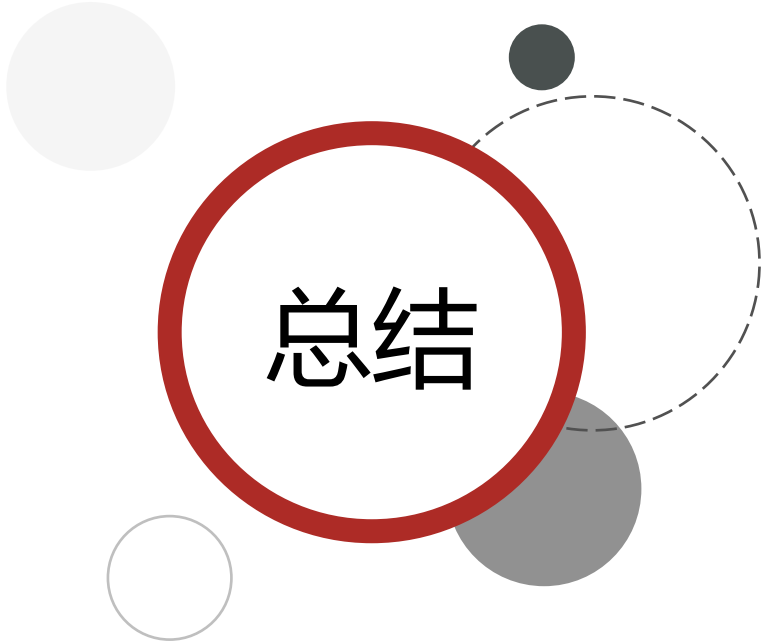


1. 字节输出流写数据的方法有哪些

方法名称	说明
public void write(int a)	写一个字节出去
public void write(byte[] buffer)	写一个字节数组出去
public void write(byte[] buffer , int pos , int len)	写一个字节数组的一部分出去。

2. 字节输出流如何实现数据追加

public FileOutputStream (String filepath , boolean append)	创建字节输出流管道与源文件路径接通，可追加数据



3. 字节输出流如何实现写出去的数据能换行

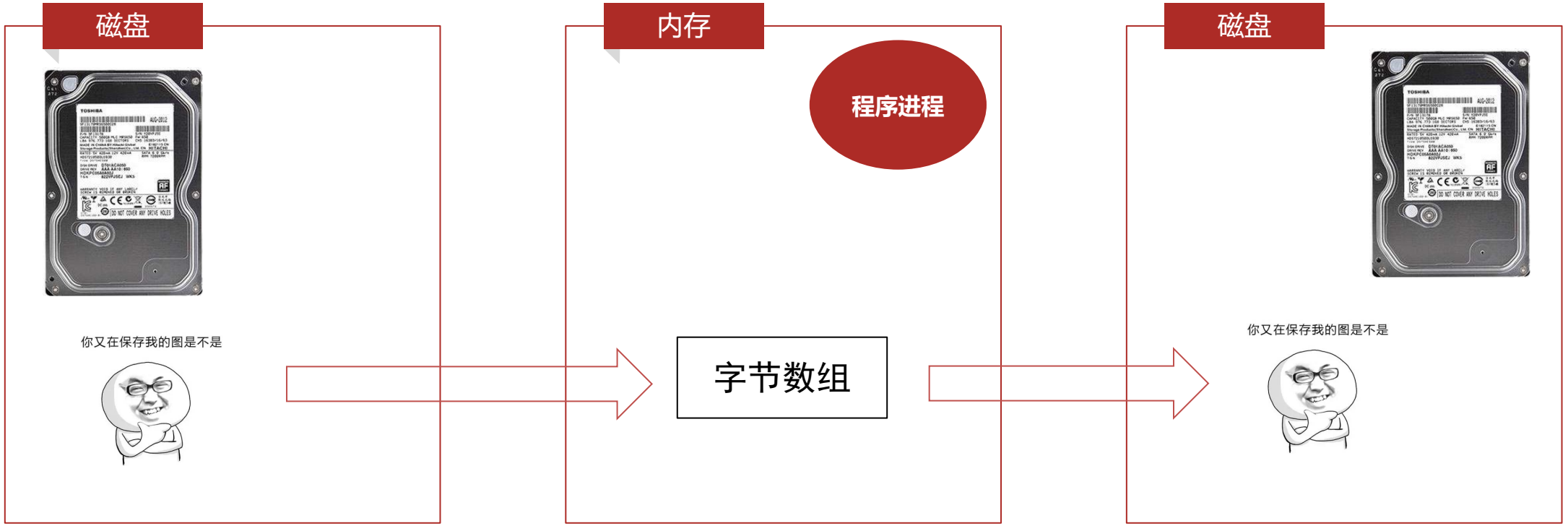
- **os.write(“\r\n”.getBytes())**

4. 如何让写出去的数据能成功生效？

- **flush()刷新数据**
- **close()方法是关闭流，关闭包含刷新，关闭后流不可以继续使用了。**

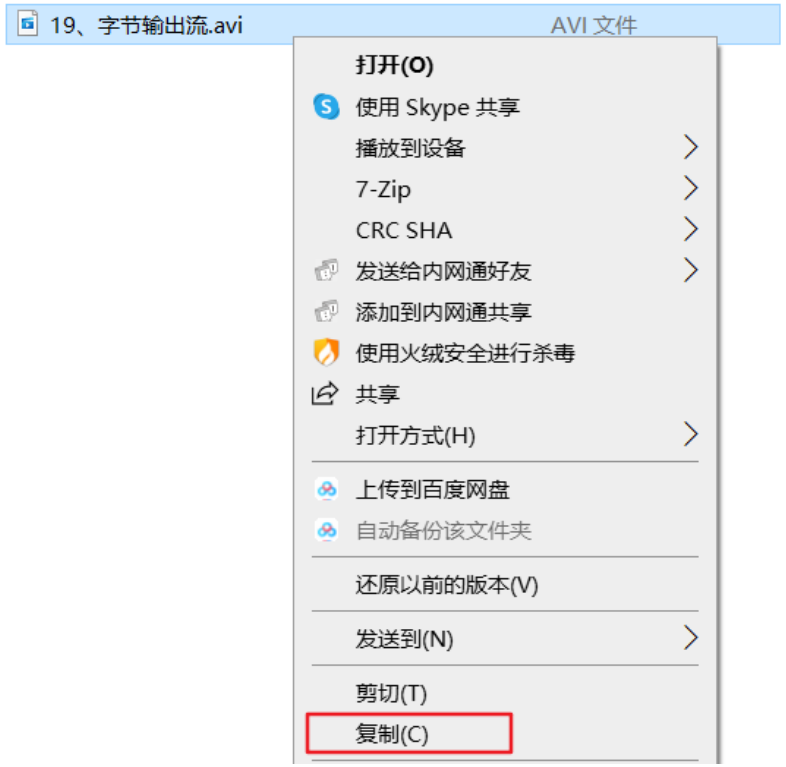
目录

Contents


磁盘
你又在保存我的图是不是
内存
程序进程
字节数组
磁盘
你又在保存我的图是不是

案例

文件拷贝



需求：

- 把某个视频复制到其他目录下的“b.avi”

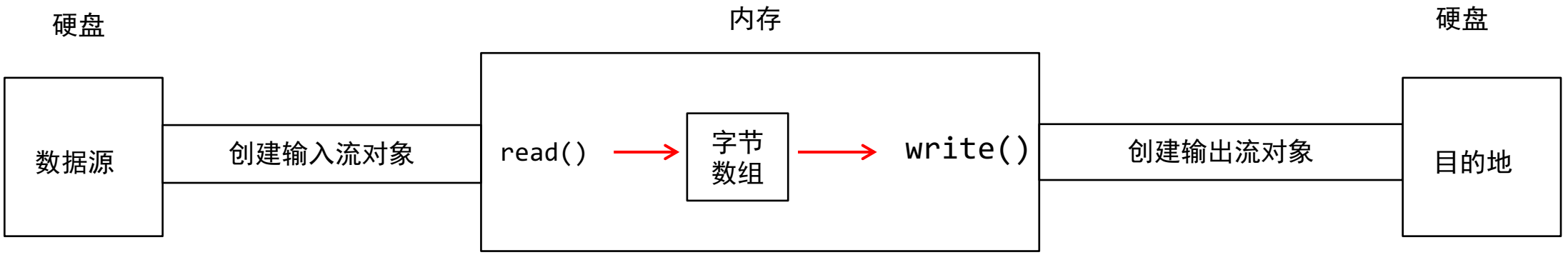
思路：

① 根据数据源创建字节输入流对象

② 根据目的地创建字节输出流对象

③ 读写数据，复制视频

④ 释放资源


硬盘
内存
硬盘
数据源
创建输入流对象
read()
字节
数组
write()
创建输出流对象
目的地

总结

1. 字节流适合做一切文件数据的拷贝吗？

   - **任何文件的底层都是字节，拷贝是一字不漏的转移字节，只要前后文件格式、编码一致没有任何问题。**

目录

Contents

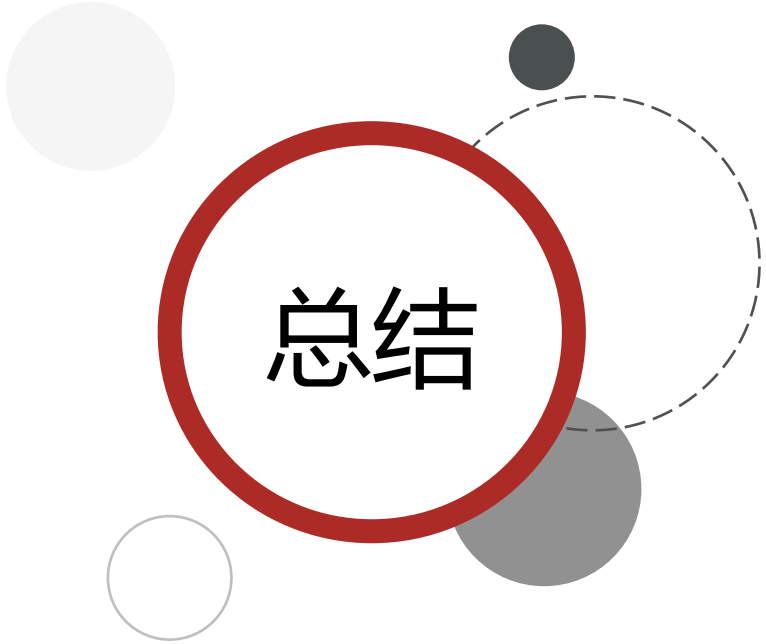
try-catch-finally

- **finally**：放在try-catch后面的，无论是正常执行还是异常执行代码，最后一定要执行，除非JVM退出。
- 作用：一般用于进行最后的资源释放操作（专业级做法）

try-catch-finally格式

```
try {
    FileOutputStream fos = new FileOutputStream("a.txt");
    fos.write(97);
    fos.close();
} catch (IOException e) {
    e.printStackTrace();
}  finally{

}
```



1. try-catch-finally的作用

- **finally代码块是最终一定要执行的，可以在代码执行完毕的最后用于释放资源。**

目录

Contents

1. finally虽然可以用于释放资源，但是释放资源的代码过于繁琐？
2. 有没有办法简化？

```
InputStream is = null ;
OutputStream os = null;
try{
    ...
}catch (Exception e){
    e.printStackTrace();
} finally {
    // 关闭资源!
    try {
        if(os != null) os.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        if(is != null) is.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

JDK 7和JDK9中都简化了资源释放操作

基本做法：

```
try{
    可能出现异常的代码;
}catch(异常类名 变量名){
    异常的处理代码;
}finally{
    执行所有资源释放操作;
}
```

手动释放资源

JDK7改进方案：

```
try(定义流对象){
    可能出现异常的代码;
}catch(异常类名 变量名){
    异常的处理代码;
}
```

资源用完最终自动释放

JDK9改进方案：

```
定义输入流对象;
定义输出流对象;
try(输入流对象；输出流对象){
    可能出现异常的代码;
}catch(异常类名 变量名){
    异常的处理代码;
}
```

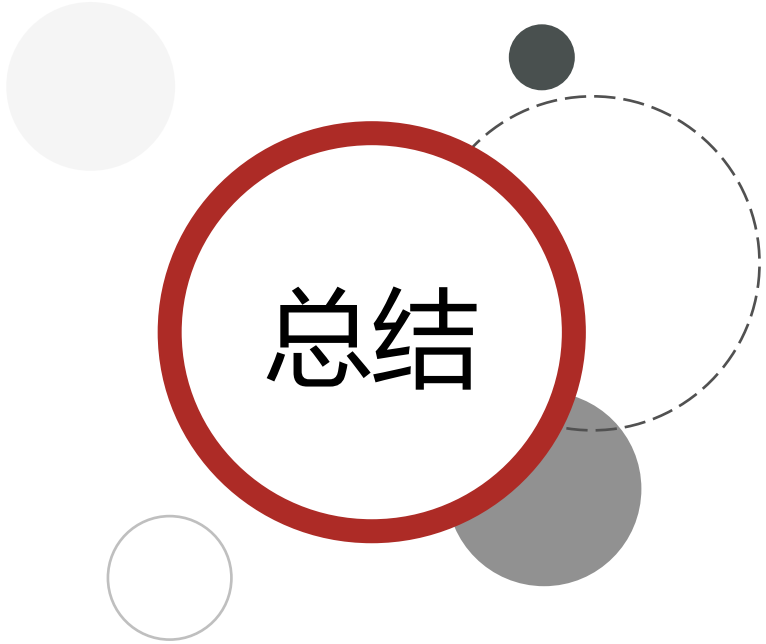
资源用完最终自动释放

注意

- JDK 7 以及 JDK 9的()中只能放置资源对象，否则报错
- 什么是资源呢？
- 资源都是实现了Closeable/AutoCloseable接口的类对象

```
public abstract class InputStream implements Closeable {}
```

```
public abstract class OutputStream implements Closeable, Flushable{}
```



1. try-catch-resource的作用
   - **自动释放资源、代码简洁**

练习

拷贝文件夹

需求：将某个磁盘的文件夹拷贝到另一个文件夹下去，包括文件夹中的全部信息

分析：

①：IO默认不可以拷贝文件夹

②：我们需要遍历文件夹，如果是文件则拷贝过去，如果是文件夹则要进行1-1创建，继续复制内容。

传智教育旗下高端IT教育品牌