day18_IO流01

一 内容回顾（列举前一天重点难点内容）

1.1 教学重点:

```
1.掌握线程的基本概念
2.掌握线程的生命周期
3.掌握常见线程的方法
4.掌握线程同步的实现
5.掌握synchronized的使用
```

1.2 教学难点:

```
1.多线程的原理的理解
2.对单例实现线程同步
这里主要是咱们的课程中还没有讲解单例设计模式的内容,不过在单例中使用线程同步是他都一个重要应用,大家要掌握.
```

二 教学目标

```
1.掌握文件的基本操作
2.掌握IO流的分类体系
3.掌握字节流的读写过程实现
4.掌握字节流的读写原理
5.掌握字符流的读写过程实现
6.掌握字符流的读写原理
```

三 教学导读

3.1. 文件操作

顾名思义， 操作磁盘上的某一个文件或者某一个文件夹。 可以对他们进行创建、 删除、 移动、 属性获取、 属性设置等操作。 但是， 并不包含读取文件的内容、 拷贝文件。

在Java中， 使用 `java.io.File` 类描述一个文件， 或者是一个文件夹。

3.2. IO流

3.2.1. 什么是IO流

IO流： Input/Output Stream

流: 指的是一串流动的数据， 在数据在流中按照指定的方向进行流动。 实现数据的读取、写入的功能。

作用:实现两个设备之间数据的传递

3.2.2. IO流的使用场景

使用File类， 只能做关于文件的操作， 获取属性、 创建文件、 删除文件、 移动文件等操作， 但是不包含读取文件中的内容。 如果需要读取、修改文件中的内容， 此时就需要使用IO流来完成了。

使用场景: 对某一个文件进行读取或者写入操作。

> 注意事项:
>
> IO流是对一个文件进行读写的， 不是一个文件夹！ 在使用IO流的时候， 不要建立与一个文件夹的连接。

3.2.3. 设备

设备概念:能输出或者输入数据的都可以成为设备

设备:磁盘(硬盘),内存,键盘,文件,网络,控制台

网络:当前主机之外的网上资源

3.2.4. IO流的分类

按照不同的分类标准， 能够得到不同分类的IO流：

- **按照传输数据的单位：**
  - 字节流： 传输的是字节,是以字节为单位的。可以操作任意类型的数据 ------音频,视频,文件,图片等
  - 字符流: 传输的是字节,不同点是在传输过程中加入了编码的操作,让我们的操作更方便------文本
- **按照数据传输的方向：**

  以内存为参考
  - 输入流： 数据从其他设备传到内存
  - 输出流： 数据从内存传到其他设备

3.2.5. 基础的IO流类的简介

其实在 java.io 包中， 有很多很多的类， 都是来描述IO流的。 但是基本上所有的IO流的类， 都是直接或间接的继承自四大父类流。

字节流的两个父类:

- 字节输入流:InputStream
- 字节输出流:OutputStream

字符流的两个父类:

- 字符读入流:Reader
- 字符写出流:Writer

3.2.6. IO流使用的注意事项

- 四大父类流， 都是抽象类， 都不能实例化对象。 因此， 需要借助他们的子类实现数据的读写。
- 流对象一旦实例化完成， 将建立一个程序与文件之间的连接。 这个连接会持有这个文件。 如果这个连接不断， 此时这个文件就是一个被使用中的状态， 此时将无法对这个文件进行其他的操作， 例如删除。
- 一个流在使用完成之后， 切记！ 一定要进行流的关闭。

四 教学内容

4.1. 文件操作(会)

4.1.1. 绝对路径和相对路径

4.1.1.1. 相关概念

路径: 用来描述一个文件所在的地址， 用来定位一个文件的。 可以分为**绝对路径**和**相对路径**。

绝对路径： 从磁盘的根目录开始， 一层层的向下查找， 直到找到这个文件。

> 例如:
>
> C:\Users\luds\Desktop\集合\assets\map.png
>
> C:\shawn\documents\JavaPDF\IO流.pdf

相对路径: 找到一个参照物， 相对于这个参照物的路径。

> 例如:
>
> assets/collection.png

4.1.1.2. 对比

	优点	缺点
绝对路径	用来表示一个文件的地址， 只要这个使用方还在当前磁盘上， 一定可以找到这个文件。	一旦更换一个文件系统，此时这个路径表示的文件将无法找到。
相对路径	只要两者的相对位置不变， 无论在哪一个文件系统中， 都可以找到这个文件。	只要两者的相对位置发生了改变， 这个文件就无法被访问到。

4.1.2. 分隔符

4.1.2.1. 分隔符的简介

在描述路径的字符串中， 有两种分隔符， 是比较常见的： **目录分隔符** 和 **路径分隔符**。

目录分隔符

分隔开一个路径中的不同的文件夹， 用来描述层级关系、 包含关系。

在不同的操作系统中， 目录分隔符是不一样的。 在windows中， 使用 `\` 作为目录分隔符; 在非windows的操作系统中， 例如: Linux、Unix.. 使用的是 `/` 作为目录分隔符。

> 理论上来讲， 在windows中， 应该使用\作为目录分隔符。 但是在有些情况下， 使用/也是可以做目录分隔符的。

路径分隔符

分隔开一个字符串中的多个路径的。

在不同的操作系统中， 路径分隔符是不一样的。 在windows中， 使用 `;` 作为路径分隔符; 在非windows的操作系统中， 例如: Linux、Unix.. 使用的是 `:` 作为路径分隔符。

4.1.2.2. 分隔符的表示

如果你的程序只需要考虑部署在windows平台， 那么只需要按照windows的规范书写就可以； 如果你的程序只需要部署到linux上， 那么只需要按照linux的规范书写就可以。 但是， 如果你的程序需要考虑在不同的平台上部署运行， 此时就需要使用以下方法进行分隔符的获取。

方法	描述
File.**separator**();	获取一个目录分隔符。 会根据不同的操作系统， 返回一个指定的目录分隔符字符串。
File.**separatorChar**();	获取一个目录分隔符。 会根据不同的操作系统， 返回一个指定的目录分隔符字符。
File.**pathSeparator**();	获取一个路径分隔符。 会根据不同的操作系统， 返回一个指定的路径分隔符字符串。
File.**pathSeparatorChar**();	获取一个路径分隔符。 会根据不同的操作系统， 返回一个指定的路径分隔符字符。

4.1.3. File类

4.1.3.1. File类的简介

File是 **java.io** 包中的一个类。 是对磁盘上的某一个文件、文件夹（目录）的描述。 所谓的文件操作， 其实都是需要使用这个类来完成的。

4.1.3.2. File类的构造方法

参数	描述
String pathname	通过一个路径字符串， 描述一个指定路径下的文件。
String parent, String child	将parent和child拼接到一起， 成一个新的路径。 用来描述这个拼接好的路径。
File parent, String child	将parent所描述的路径与child拼接到一起， 成一个新的路径。 用来描述这个拼接好的路径。

```
import java.io.File;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description File类的构造方法
 */
public class Program {
    public static void main(String[] args) {
        // 1. File(String pathname)
        //   如果这个路径下的文件不存在，不影响File对象的实例化。
        File file = new File("C:\\Users\\luds\\Desktop\\dis_hash.png");
        System.out.println(file.exists());

        // 2. File(String parent, String child)
        //    在这个构造方法中，会将parent与child合并在一起
        File file1 = new File("C:\\Users\\luds\\Desktop", "dis_hash.png");
        System.out.println(file1);
        System.out.println(file1.exists());
```

```
20
21          // 3. File(File parent, String child)
22          //    在构造方法中，将parent的路径和child进行拼接，得到一个新的路径
23          File desktop = new File("C:\\Users\\luds\\Desktop");
24          File file2 = new File(desktop, "dis_hash.png");
25          System.out.println(file2);
26          System.out.println(file2.exists());
27      }
28  }
```

4.1.3.3. File类的常用方法

1) 文件属性获取方法

返回值	方法	描述
boolean	**exists()**	判断一个文件(目录)是否存在。
boolean	**isFile()**	判断一个路径指定的是否是一个文件。
boolean	**isDirectory()**	判断一个路径指定的是否是一个文件夹。
long	**length()**	获取一个文件的大小(注: 只能获取文件的大小)。
boolean	isHidden()	判断一个文件(目录)是否是隐藏的。
boolean	canRead()	判断一个文件(目录)是否是可读的。

boolean	canWrite()	判断一个文件(目录)是否是可写的。
boolean	canExecute()	判断一个文件(目录)是否是可执行的。
String	**getName()**	获取一个文件(目录)的目录。
String	**getPath()**	获取一个文件(目录)的路径(相对路径)。
String	**getAbsolutePath()**	获取一个文件(目录)的路径（绝对路径）。
String	**getParent()**	获取一个文件(目录)的父级路径(字符串)。
File	**getParentFile()**	获取一个文件(目录)的父级（File类)。
long	lastModified()	获取一个文件(目录)上次修改的时间。

```
1  import java.io.File;
2  import java.util.Date;
3
4  /**
5   * @Author 千锋大数据教学团队
6   * @Company 千锋好程序员大数据
7   * @Description File类中的常用的方法
8   */
9  public class FileOperation {
10     public static void main(String[] args) {
11         // 实例化一个对象
12         //1.直接通过绝对路径
```

```
13          //File file1 = new
File("D:\\BigData2005\\BigData2005N19\\src\\com\\qf\\te
st\\Demo1.java");
14  //      //2.通过父路径和子路径字符串形式的拼接
15          //File file2 = new
File("D:\\BigData2005\\BigData2005N19\\","src\\com\\qf\
\test\\Demo1.java");
16  //      //3.通过父路径对象和子路径对象(字符串)
17  //        File file3 = new
File("D:\\BigData2005\\BigData2005N19\\");
18  //        File file4 = new
File(file3,"src\\com\\qf\\test\\Demo1.java");
19
20
21        //实例演示
22          File file = new
File("src\\day24\\cFiles\\FileOperation.java");
23
24          // 1. 判断一个File对象指向的路径上是否有文件或者文件夹
25          System.out.println("exists = " +
file.exists());
26          // 2. 判断一个File对象指向的路径上是否是一个文件
27          System.out.println("isFile = " +
file.isFile());
28          // 3. 判断一个File对象指向的路径上是否是一个目录
29          System.out.println("isDir = " +
file.isDirectory());
30          // 4. 获取一个文件的大小（注：只能获取文件的大小，不能
获取文件夹的大小）
31          System.out.println("length = " +
file.length());
32          // 5. 判断一个文件（目录）是否是隐藏的
```

```
33          System.out.println("hidden = " +
file.isHidden());
34          // 6. 判断文件（目录）的权限
35          System.out.println("read = " + file.canRead());
// 可读权限
36          System.out.println("write = " +
file.canWrite());        // 可写权限
37          System.out.println("execute = " +
file.canExecute());    // 可执行权限
38          // 7. 获取文件的名字
39          System.out.println("name = " + file.getName());
40          // 8. 获取文件的路径
41          System.out.println("path = " + file.getPath());
// 相对路径
42          System.out.println("absolutePath = " +
file.getAbsolutePath()); // 绝对路径
43          // 9. 获取父级文件夹的路径(字符串)
44          System.out.println("parent = " +
file.getParent());
45          // 10. 获取父级文件夹(File对象)
46          System.out.println("parentFile = " +
file.getParentFile());
47          // 11. 获取文件上次修改的时间(时间戳)--是最后修改的时
间,不是查看的时间
48  //          long lastTime = file.lastModified();
49  //          System.out.println(lastTime);
50  //          Date date = new Date(lastTime);
51  //          SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
52  //          String strDate =
simpleDateFormat.format(date);
53  //          System.out.println(strDate);
54
```

```
55      }
56  }
```

2) 文件操作方法

返回值	方法	描述
boolean	createNewFile()	在指定的路径下创建一个文件。
boolean	mkdir()	在指定的路径下创建一级文件夹。
boolean	mkdirs()	在指定的路径下创建多级文件夹。
boolean	delete()	删除一个文件，或者删除一个空的文件夹。
boolean	renameTo(File dst)	将一个文件重命名为指定的文件， 也可以实现文件的移动。

```
1  import java.io.File;
2  import java.io.IOException;
3
4  /**
5   * @Author 千锋大数据教学团队
6   * @Company 千锋好程序员大数据
7   * @Description 文件的操作
8   */
9  public class FileOperation2 {
10      public static void main(String[] args) {
11          // 实例化一个File对象
12          File file = new
   File("C:\\Users\\luds\\Desktop\\abc\\a\\b\\c");
13
```

```
14          try {
15              // 1. 创建文件
16          /*
17           * 注意点:
18           * 1.必须保证文件以外的路径都是存在的
19           * 2.createNewFile只能用于创建文件
20           */
21              boolean flag = file.createNewFile();
22              System.out.println(flag);
23          } catch (IOException e) {
24              e.printStackTrace();
25          }
26
27          // 2. 创建文件夹(只能创建一级文件夹)
28          boolean flag = file.mkdir();
29          System.out.println(flag);
30
31          // 3. 创建文件夹(可以创建一级及以上的文件夹)
32          boolean flag = file.mkdirs();
33          System.out.println(flag);
34
35          // 4. 删除文件(目录)(可以删除文件，也可以删除空文件夹)
36          //    谨慎使用: 这个方法，可以将文件直接从磁盘删除，不经过回收站。没有撤销的余地。
37          boolean flag = file.delete();
38          System.out.println(flag);
39
40          // 5. 文件的重命名
41          File src = new File("C:\\Users\\luds\\Desktop\\abc");
42          File dst = new File("C:\\Users\\luds\\Desktop\\ABC");
43          System.out.println(src.renameTo(dst));
```

```
        // 5.1. 借助重命名，实现文件的移动
        File src1 = new
File("C:\\Users\\luds\\Desktop\\DIS_HASH.png");
        File dst1 = new
File("C:\\Users\\luds\\Desktop\\ABC\\dis_hash.png");
        System.out.println(src1.renameTo(dst1));
    }
}
```

3) 子文件获取

返回值	方法	描述
String[]	list()	获取一个目录下所有的子文件(夹)的名字。
String[]	list(FilenameFilter filter)	获取一个目录下所有的满足条件的子文件(夹)的名字。
File[]	**listFiles()**	获取一个目录下所有的子文件（夹）。
File[]	**listFiles(FileFilter filter)**	获取一个目录下所有的满足条件的子文件（夹）。
File[]	listFiles(FilenameFilter filter)	获取一个目录下所有的满足条件的子文件（夹）。

```
import java.io.File;

/**
 * @Author 千锋大数据教学团队
```

```
 * @Company 千锋好程序员大数据
 * @Description 获取某一个目录下的所有的内容
 */
public class FileOperation4 {
    public static void main(String[] args) {
        // 实例化一个File对象
        File file = new
File("C:\\Users\\luds\\Desktop");
        // 1. 列举一个目录下所有的子文件的名字
        String[] names = file.list();
        for (String name : names) {
            System.out.println(name);
        }

        // 2. 列举一个目录下所有的子文件的名字
        //    带有过滤信息的。
        String[] names1 = file.list((f, name) -> {
            // f: 父级文件夹的File对象
            // name: 子文件的名字
            // 返回值: 如果是true, 将会在结果的数组中展示
            return name.startsWith(".");
        });
        for (String s : names1) {
            System.out.println(s);
        }

        // 3. 列举一个目录下所有的子文件(以File对象的方式)
        File[] files = file.listFiles();
        for (File file1 : files) {
            System.out.println(file1);
        }

        // 4. 列举一个目录下所有的满足指定条件的子文件
```

```
        File[] files1 = file.listFiles(File::isHidden);
        for (File file1 : files1) {
            System.out.println(file1);
        }

        // 5. 举一个目录下所有的满足指定条件的子文件
        File[] files2 = file.listFiles((f, n) -> new File(f, n).isHidden());
    }
}
```

4.2. 基础的IO流(会)

4.2.1. 建立程序与文件的连接

其实， 就是建立了程序与文件之间连接的管道， 实现数据在这个管道之内进行流动。 管道分为不同的类型： 字节输入流、 字节输出流、 字符输入流、 字符输出流。 下面以字节输入流 InputStream 为例。

4.2.1.1. 标准流程

- try结构外面， 声明流对象， 为了在finally中使用。
- 在try结构里面， 实例化流对象， 并捕获异常。
- 在finally结构中， 对流进行关闭。 在关闭的时候， 需要考虑流对象是否是null， 以及要处理 IOException 异常。

```
import java.io.*;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
```

```
 * @Description 测试文件与程序的连接建立
 */
public class IO1 {
    public static void main(String[] args) {
        // 在外面声明变量
        InputStream inputStream = null;
        try {
            // 实例化一个FileInputStream对象，向上转型为InputStream类型。
            // 这个实例化如果完成，将会建立程序与文件之间的连接。
            // 建立好之后，数据就可以从文件中流动到程序中。
        // 注意：一:数据流动到程序中，并不意味着文件中没有数据了!
        //二:如果只写文件的相对路径,不写绝对路径,默认路径是当前的工程
        //FileNotFoundException:(系统找不到指定的路径。)
            inputStream = new FileInputStream("t");
            // 数据的读取操作
            // 在数据读取的过程中，也会出现 IOException 异常。一旦出现异常，后序的代码都不执行了，直接执行catch语句了
            // 流的关闭，不能放到try里面。需要放到finally中。
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } finally {
            // 流在使用结束之后，一定要进行关闭。
            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
```

```
34          }
35      }
36  }
```

4.2.1.2. try结构的特殊使用

在 JDK1.7 之后， 可以在try后面添加一对小括号。 将 AutoClosable 接口实现类的对象， 实例化放到小括号中完成。 此时， 在try结构执行结束的时候， 会自动的调用AutoClosable接口实现类中的close方法， 进行流的关闭。 这样写的流的建立比较简单， 也是后面我们最主要使用的方式。

```
1  import java.io.*;
2
3  /**
4   * @Author 千锋大数据教学团队
5   * @Company 千锋好程序员大数据
6   * @Description 常见的IO流的创建的方式
7   */
8  public class IO2 {
9      public static void main(String[] args) {
10         /**
11          * try结构的特殊语法：try ()
12          * 将 AutoClosable 接口的实现类对象的实例化放到小括号中。
13          * 此时，在离开了try结构的时候，会自动的对这个类进行close方法的调用
14          */
15         try (InputStream inputStream = new FileInputStream("file\\day25\\source")) {
16     // 数据的读取操作
17         } catch (FileNotFoundException e) {
18             e.printStackTrace();
```

```
19          } catch (IOException e) {
20              e.printStackTrace();
21          }
22          System.out.println(new
    File("file\\day25\\source").delete());
23      }
24  }
```

4.2.2. InputStream

4.2.2.1. InputStream简介

这是一个字节输入流。 从方向来说， 是一个输入流， 数据是从文件中流动到程序中(数据从其他设备到内存)， 是为了读取文件中的数据的。 从数据单位来说， 这个流中流动的数据直接是字节的形式。

4.2.2.2. 文件的读取

注意:

1.为了方便测试,我们可以先通过字节输出流或者手动在当前工程中创建一个test1.txt文件,写入字符串abcde

2.有三种读取数据的方式第一:一次读取一个字节;第二:一次读取多个字节;第三:一次读取全部字节

三种读取数据的方式中,推荐使用第二种,一次读取多个字节

1) 一次读取一个字节

read():一个字节一个字节的读,每次读出一个字节

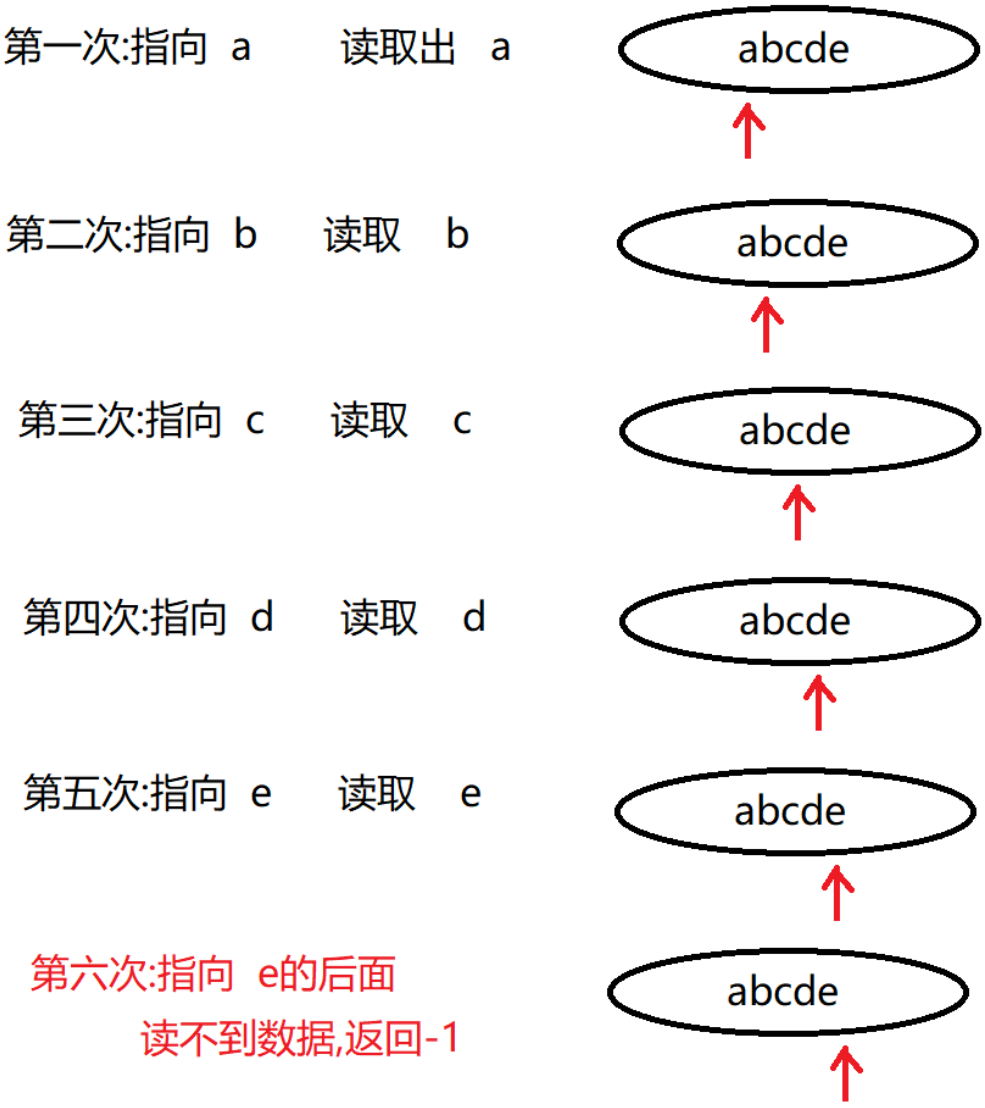
```
1      public static void read1() throws IOException {
2          // 1. 建立程序与文件之间的连接，用来读取这个文件
```

```
3           try(InputStream inputStream = new FileInputStream("test1.txt")){
4               //2.读取数据,先声明一个变量,read方法的返回值,就是读取的数据
5               //注意:返回-1,默认认为数据读完了
6               int num = 0;
7   //          num = inputStream.read();
8               //3.直接读取数据
9               //想显示字符需要自己强转
10  //          System.out.println((char)num);
11  //          num = inputStream.read();
12  //          System.out.println((char)num);
13  //          num = inputStream.read();
14  //          System.out.println((char)num);
15  //          num = inputStream.read();
16  //          System.out.println((char)num);
17  //          num = inputStream.read();
18  //          System.out.println((char)num);
19  //          num = inputStream.read();
20  //          System.out.println(num);//返回-1,默认认为数据读完了
21
22              //4.使用循环读取数据
23              while ((num = inputStream.read()) != -1) {
24                  System.out.print((char)num);
25              }
26          } catch (IOException e) {
27              e.printStackTrace();
28          }
29      }
```

图示:

read()　:每次读取一个字节

原理:控制磁头每次向后移动一个字节,依次读取,直到将所有的字节 读完.
当read()返回-1的时候,说读完了



2) 一次读取多个字节

read(数组):一次可以读出多个字节,数组的作用:每次会将读出的字节临时放到这个数组中

```
1    public static void read2() throws IOException {
2        // 1. 建立程序与文件之间的连接，用来读取这个文件
3        try (InputStream inputStream = new
FileInputStream("test1.txt")) {
```

```
4                 // 2. 读取字节流中的数据，需要有一个字节数组，用来读取数据
5                 //创建临时数组
6                 /* 数组是临时存放数据的地方,我们会将读到的字符放到临时数组中,数组的大小决定了我们一次可以读到的字符个数.
7                  * 一般这个数组的大小<=1kB
8                  * 返回值代表本次读到的真实的字符个数,如果返回值是-1代表读完了.
9                  */
10                byte[] arr = new byte[2];
11                // 3. 声明一个整型变量，用来记录每次读取了多少个字节的数据
12                int num = 0;
13                //4.直接读取数据
14  //            num = reader.read(arr);
15  //            System.out.println(new String(arr,0,num)+" num:"+num);
16  //            num = reader.read(arr);
17  //            System.out.println(new String(arr,0,num)+" num:"+num);
18  //            num = reader.read(arr);
19                //转部分字符到字符串,第二个参数是指定的下标,第三参数是字符数量
20                // 将读取到的字节数组中的数据，转成字符串输出
21                // 为了去除最后一次进行读取数据的时候，上次读取残留的问题
22                // 最后一次读取的数据，只有指定部分是我们需要的数据
23  //            System.out.println(new String(arr,0,num)+" num:"+num);
24  //            num = reader.read(arr);
25  //            System.out.println("    num:"+num);
26
27                //5.使用循环读取数据
```

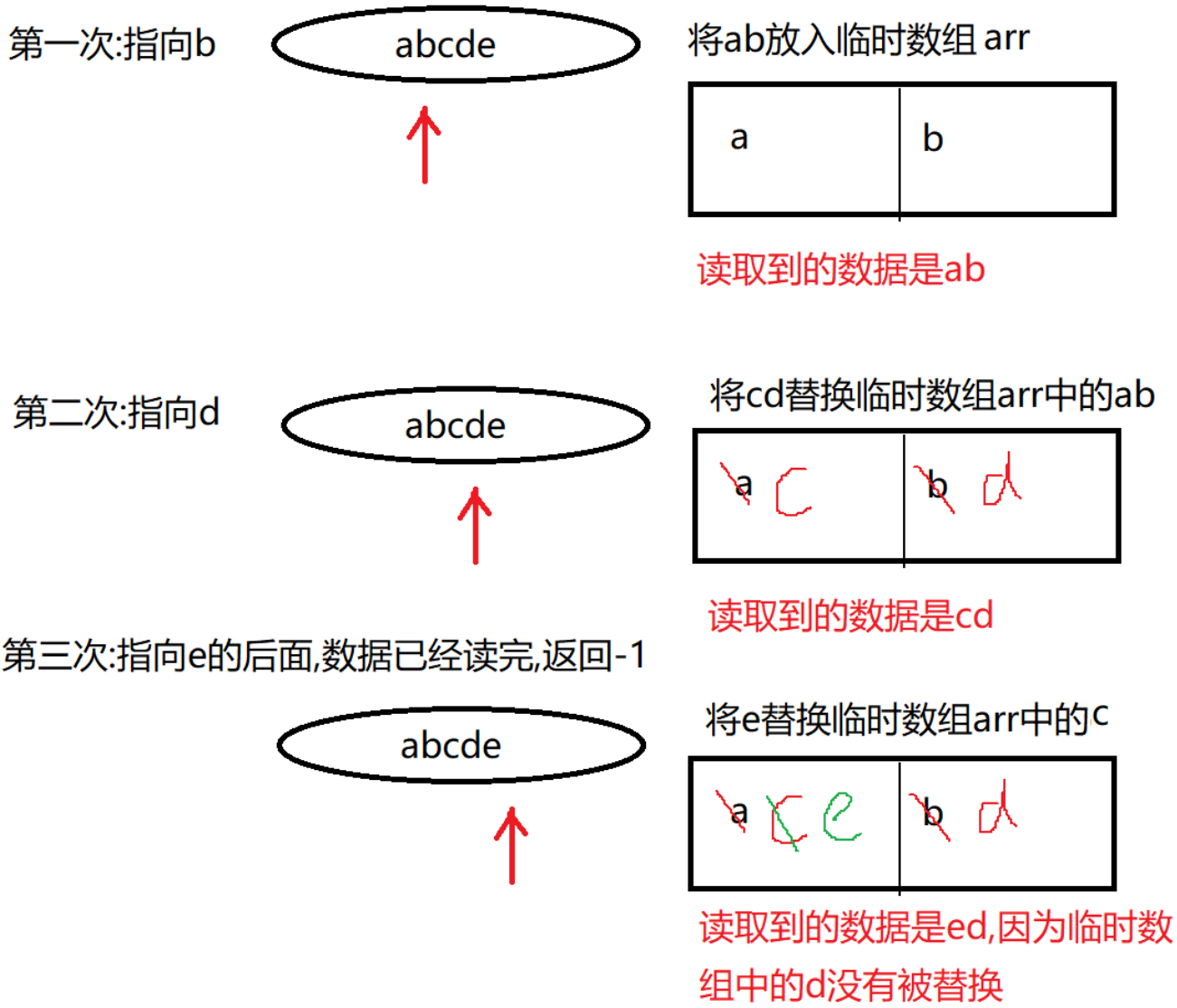
```
28              while ((num = inputStream.read(arr)) != -1)
    {
29                  System.out.println(new
    String(arr,0,num)+"    num:"+num);
30              }
31          } catch (IOException e) {
32              e.printStackTrace();
33          }
34      }
```

图示:

read(数组):一次读取多个字节,比一次读取一个效率高

注意:按照数组读,本质上也是一个一个的读

临时数组arr中只有两个元素,所以每次最多可以读取两个字节



3) 一次读取全部字节

可以通过available()方法获取全部字节个数

```
1  public static void read3() throws IOException {
2          // 1. 建立程序与文件之间的连接，用来读取这个文件
3          try (InputStream inputStream = new
   FileInputStream("test1.txt")) {
```

```
            // 2. 读取字节流中的数据，需要有一个字节数组，用来读取数据
            //获取文件的字节个数
            //注意:这种方式适合文件的字节数比较小的时候,大概是几kb之内.
            int num = inputStream.available();
            //2.读
            byte[] bytes = new byte[num];
            inputStream.read(bytes);
            System.out.println(new String(bytes));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

4.2.3. OutputStream

4.2.3.1 OutputStream简介

字节输出流。 从方向上来分， 是一个输出流， 数据从程序中流动到文件中(数据从内存到其他设备)， 实现文件的写操作。 从流中流动的数据单位来分， 是一个字节流， 流中流动的数据直接是字节的形式。

4.2.3.2. 文件的写

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

```

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 字节输出流，写文件
 */
public class OutputStreamTest {
    public static void main(String[] args) {
        // 1. 实例化一个管道，连接文件和程序。
        //    对于FileOutputStream来说，如果目标文件不存在，则会自动的创建,如果存在,会将原来的内容覆盖
        //    当无法创建这个文件的时候(父级目录不存在)，创建会失败，会触发 FileNotFoundException 。
        //OutputStream outputStream = new FileOutputStream("test1.txt")
        //文件的续写:FileWriter(String file,boolean value)
        //当value位true的时候,不会将原来的内容覆盖,会接着写
        try (OutputStream outputStream = new FileOutputStream("test1.txt",true)) {
            // 2. 准备需要写入到这个文件中的数据
            String message = "你好，师姐";
            // 3. 将数据写入到输出流中，由输出流写入到文件中
            //将字符串转成字节数组
            outputStream.write(message.getBytes());

            // 冲刷缓冲区，将缓冲区中的数据强制流动到文件中。
            // 在流关闭的时候，会自动的调用。
            //注意:当我们进行循环写入操作时,最好通过flush()方法刷新
            outputStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
33          }
34      }
35
```

4.2.4. 案例: 文件拷贝

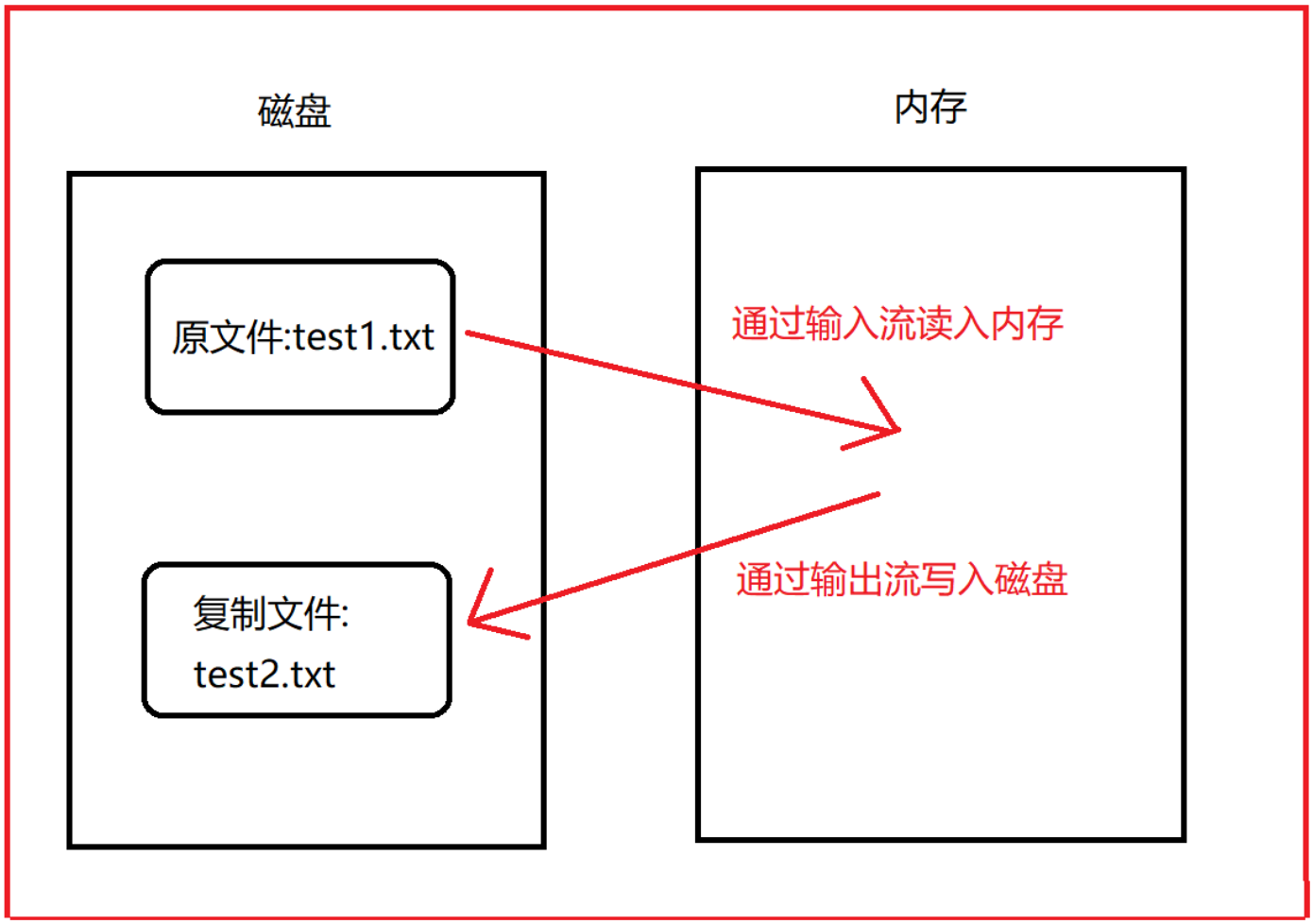
4.2.4.1. 需求分析

实现， 将一个文件拷贝到另外一个地方。 注意， 这个不是剪切， 拷贝完成之后， 原文件还在。

实现方式： 借助两个流来完成。

- 使用字节输入流输入， 循环读取原文件中的数据。
- 使用字节输出流， 将每次读取到的数据， 写入到目标文件中。

图示:



4.2.4.2. 示例代码

```
1  import java.io.*;
2
3  /**
4   * @Author 千锋大数据教学团队
5   * @Company 千锋好程序员大数据
6   * @Description 使用字节流实现文件的拷贝
7   */
8  public class FileCopy {
9      public static void main(String[] args) {
10         boolean ret =
   copy("C:\\Users\\luds\\Desktop\\src.mp4",
   "C:\\Users\\luds\\Desktop\\dst.mp4");
11         System.out.println(ret);
12     }
```

```
13
14      /**
15       * 实现功能：将源文件中的数据拷贝到目标文件
16       * @param srcPath 原文件路径
17       * @param dstPath 目标文件路径
18       * @return 拷贝的结果
19       */
20      private static boolean copy(String srcPath, String
dstPath) {
21          // 1. 判断目标路径上，是否有文件存在
22          File dst = new File(dstPath);
23          if (dst.exists()) {
24              return false;
25          }
26          // 2. 实现文件的拷贝
27          try (InputStream inputStream = new
FileInputStream(srcPath);
28               OutputStream outputStream = new
FileOutputStream(dst)) {
29              // 拷贝的过程
30              // 2.1. 实例化一个字节数组
31              byte[] array = new byte[1024];
32              // 2.2. 声明一个整型变量，用来记录每次读取到了多少
个字节的数据
33              int length = 0;
34              // 2.3. 循环读取数据
35              while ((length = inputStream.read(array))
!= -1) {
36                  // 2.4. 将读取到的数据，写入到输出流中
37                  outputStream.write(array, 0, length);
38              }
39              // 2.5. 冲刷缓冲区
40              outputStream.flush();
```

```
            return true;
        }
        catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

4.2.5. Reader

4.2.5.1. Reader的简介

这是一个字符输入流。 从方向来说， 是一个输入流， 数据是从文件中流动到程序中(数据从其他设备到内存)， 是为了读取文件中的数据的。 从数据单位来说， 这个流中流动的数据以字节为单位的,不同的是在传输过程中加入了编码的操作,让我们的操作更方便。

4.2.5.2. 读取文件

```
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 字符输入流读取数据
 */
public class ReaderTest {
    public static void main(String[] args) {

```

```
12          // 读取过程与字节输入流完全相同，只需要将使用到的类换一下即可。
13          try (Reader reader = new FileReader("file\\day25\\src")) {
14              // 1. 实例化一个字符数组
15              char[] array = new char[100];
16              // 2. 声明一个变量，用来记录每次读取到了多少个数据
17              int length = 0;
18              // 3. 循环读取数据
19              while ((length = reader.read(array)) != -1) {
20                  String str = new String(array, 0, length);
21                  System.out.print(str);
22              }
23          }
24          catch (IOException e) {
25              e.printStackTrace();
26          }
27      }
28  }
```

4.2.6. Writer

4.2.6.1. Writer的简介

字符输出流。 从方向上来分， 是一个输出流， 数据从程序中流动到文件中(数据从内存到其他设备)， 实现文件的写操作。 从流中流动的数据单位来分， 是一个字符流， 流中流动的数据是以字节为单位,不同的是在传输过程中加入了编码的操作,让我们的操作更方便。

4.2.6.2. 文件的写操作

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 使用字符流写数据
 */
public class WriterTest {
    public static void main(String[] args) {
        // 1. 实例化相关的类
        try (Writer writer = new
FileWriter("file\\day25\\target", true)) {
            // 2. 将数据写入到输出流中
            writer.write("hello, world");
            // 3. 冲刷缓冲区
            writer.flush();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4.2.7. 案例: 文件拷贝

4.2.7.1. 需求分析

实现， 将一个文件拷贝到另外一个地方。 注意， 这个不是剪切， 拷贝完成之后， 原文件还在。

实现方式： 借助两个流来完成。

- 使用字符输入流， 循环读取原文件中的数据。
- 使用字符输出流， 将每次读取到的数据， 写入到目标文件中。

4.2.7.2. 示例代码

```
/**
  * 使用字符流实现文件的拷贝
  * @param srcPath 原文件路径
  * @param dstPath 目标文件路径
  */
private static void fileCopy2(String srcPath, String dstPath) {
    // 2. 循环读取目标文件中的数据
    try (Reader reader = new FileReader(srcPath); Writer writer = new FileWriter(dstPath)) {
        // 3. 循环读取源文件中的数据
        char[] array = new char[100];
        int length = 0;
        while ((length = reader.read(array)) != -1) {
            // 4. 将读取到的数据写入到输出流
            writer.write(array, 0, length);
        }
        writer.flush();
        return true;
    }
```

```
19      catch (IOException e) {
20          e.printStackTrace();
21          return false;
22      }
23  }
```