IO 流

主要内容

IO 简介

IO 流入门案例

File 类的使用

常用流对象

Apache IO 包

本章总结

学习目标

知识点	要求
IO 简介	了解
IO 流入门案例	了解
File 类的使用	掌握
常用流对象	掌握
Apache IO 包	掌握
本章总结	掌握

一、 IO 简介

1 什么是 IO

对于任何程序设计语言而言，输入输出(Input/Output)系统都是非常核心的功能。程序运行需要数据，数据的获取往往需要跟外部系统进行通信，外部系统可能是文件、数据库、其他程序、网络、IO 设备等等。外部系统比较复杂多变，那么我们有必要通过某种手段进

行抽象、屏蔽外部的差异，从而实现更加便捷的编程。

输入(Input)指的是 :可以让程序从外部系统获得数据(核心含义是“读” ,读取外部数据)。常见的应用：

- 读取硬盘上的文件内容到程序。例如：播放器打开一个视频文件、word 打开一个 doc 文件。
- 读取网络上某个位置内容到程序。例如：浏览器中输入网址后，打开该网址对应的网页内容；下载网络上某个网址的文件。
- 读取数据库系统的数据到程序。
- 读取某些硬件系统数据到程序。例如：车载电脑读取雷达扫描信息到程序；温控系统等。

输出(Output)指的是：程序输出数据给外部系统从而可以操作外部系统（ 核心含义是“写”，将数据写出到外部系统）。常见的应用有：

- 将数据写到硬盘中。例如：我们编辑完一个 word 文档后，将内容写到硬盘上进行保存。
- 将数据写到数据库系统中。例如：我们注册一个网站会员，实际就是后台程序向数据库中写入一条记录。
- 将数据写到某些硬件系统中。例如：导弹系统导航程序将新的路径输出到飞控子系统，飞控子系统根据数据修正飞行路径。

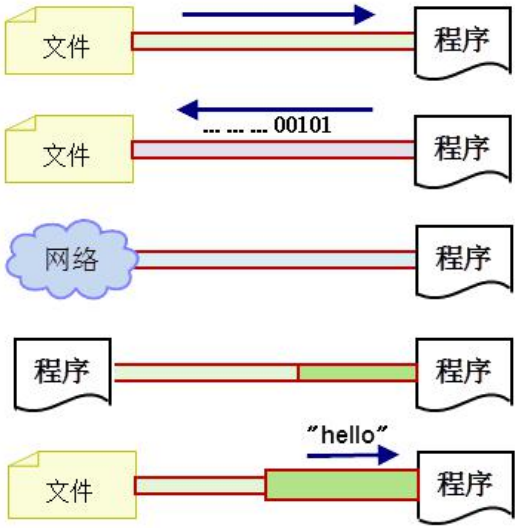
java.io 包为我们提供了相关的 API，实现了对所有外部系统的输入输出操作，这就是我们这章所要学习的技术。

2 数据源

数据源 Data Source，提供数据的原始媒介。常见的数据源有：数据库、文件、其他程序、内存、网络连接、IO 设备。

数据源分为：源设备、目标设备。

- 源设备：为程序提供数据，一般对应输入流。
- 目标设备：程序数据的目的地，一般对应输出流。

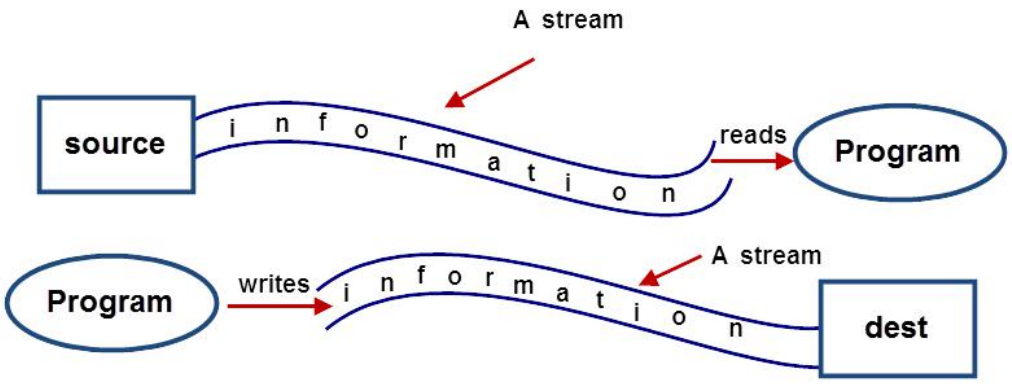


3 流的概念

流是一个抽象、动态的概念，是一连串连续动态的数据集合。

对于输入流而言，数据源就像水箱，流(Stream)就像水管中流动着的水流，程序就是我们最终的用户。我们通过流（A Stream）将数据源（Source）中的数据（information）输送到程序（Program）中。

对于输出流而言，目标数据源就是目的地（dest），我们通过流（A Stream）将程序（Program）中的数据（information）输送到目的数据源（dest）中。



菜鸟雷区

输入/输出流的划分是相对程序而言的，并不是相对数据源。

4 Java 中四大 IO 抽象类

InputStream/OutputStream 和 Reader/Writer 类是所有 IO 流类的抽象父类，我们有必要简单了解一下这个四个抽象类的作用。然后，通过它们具体的子类熟悉相关的用法。

InputStream

此抽象类是表示字节输入流的所有类的父类。InputSteam 是一个抽象类，它不可以实例化。 数据的读取需要由它的子类来实现。根据节点的不同，它派生了不同的节点流子类 。

继承自 InputSteam 的流都是用于向程序中输入数据，且数据的单位为字节（8 bit）。

常用方法：

int read() :读取一个字节的数据 ,并将字节的值作为 int 类型返回(0-255 之间的一个值)。如果未读出字节则返回-1（返回值为-1 表示读取结束）。

void close()：关闭输入流对象，释放相关系统资源。

OutputStream

此抽象类是表示字节输出流的所有类的父类。输出流接收输出字节并将这些字节发送到某个目的地。

常用方法：

void write(int n)：向目的地中写入一个字节。

void close()：关闭输出流对象，释放相关系统资源。

Reader

Reader 用于读取的字符流抽象类，数据单位为字符。

int read(): 读取一个字符的数据，并将字符的值作为 int 类型返回(0-65535 之间的一个值，即 Unicode 值)。如果未读出字符则返回-1（返回值为-1 表示读取结束）。

void close() ： 关闭流对象，释放相关系统资源。

Writer

Writer 用于输出的字符流抽象类，数据单位为字符。

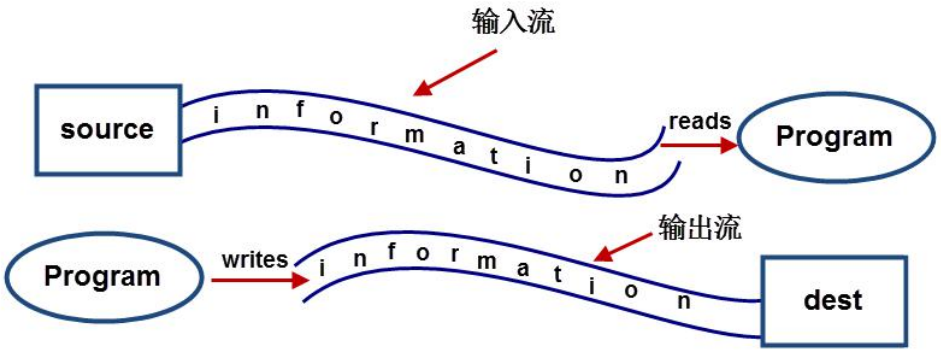
void write(int n)： 向输出流中写入一个字符。

void close() ： 关闭输出流对象，释放相关系统资源。

5 Java 中流的概念细分

按流的方向分类：

- 输入流：数据流从数据源到程序（以 InputStream、Reader 结尾的流）。
- 输出流：数据流从程序到目的地（以 OutPutStream、Writer 结尾的流）。



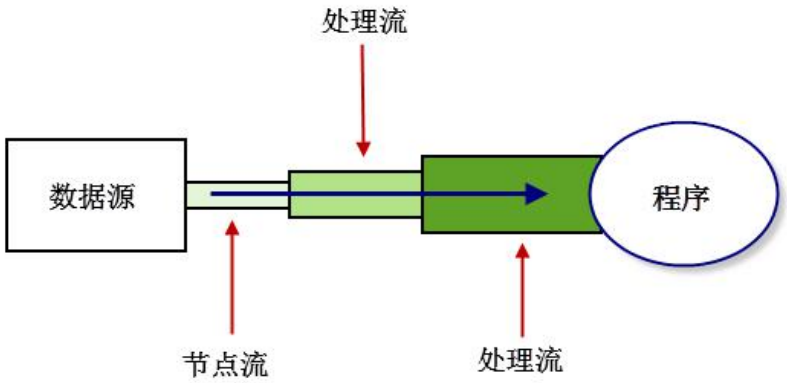
按处理的数据单元分类：

- 字节流：以字节为单位获取数据，命名上以 Stream 结尾的流一般是字节流，如 FileInputStream、 FileOutputStream。
- 字符流：以字符为单位获取数据，命名上以 Reader/Writer 结尾的流一般是字符流，如 FileReader、 FileWriter。

按处理对象不同分类：

- 节点流 :可以直接从数据源或目的地读写数据 ,如 FileInputStream、FileReader、DataInputStream 等。
- 处理流：不直接连接到数据源或目的地，是"处理流的流"。通过对其他流的处理提高程序的性能，如 BufferedInputStream、 BufferedReader 等。处理流也叫包装流。

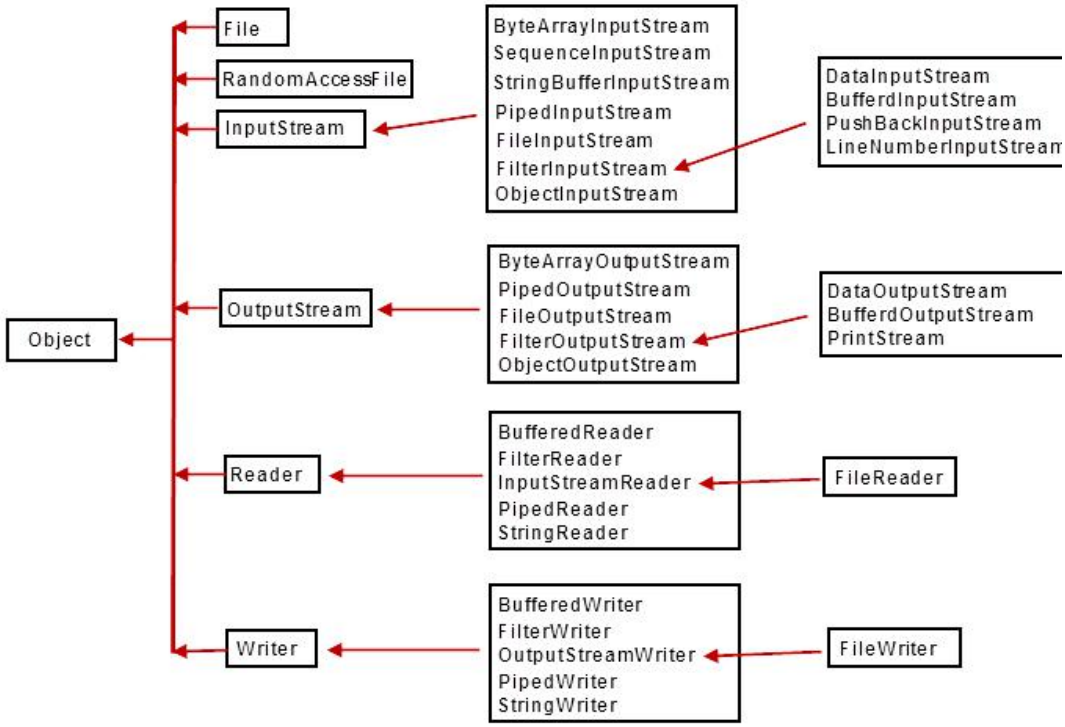
节点流处于 IO 操作的第一线，所有操作必须通过它们进行；处理流可以对节点流进行包装，提高性能或提高程序的灵活性。



6 Java 中 IO 流类的体系

Java 为我们提供了多种多样的 IO 流，我们可以根据不同的功能及性能要求挑选合适的 IO 流。

下图为 Java 中 IO 流类的体系（这里只列出常用的类，详情可以参考 JDK API 文档）



从上图发现，很多流都是成对出现的，比如：FileInputStream/FileOutputStream，显然是对文件做输入和输出操作的。我们下面简单做个总结：

1. InputStream/OutputStream

   字节流的抽象类。

2. Reader/Writer

   字符流的抽象类。

3. FileInputStream/FileOutputStream

   节点流：以字节为单位直接操作“文件”。

4. ByteArrayInputStream/ByteArrayOutputStream

   节点流：以字节为单位直接操作“字节数组对象”。

5. ObjectInputStream/ObjectOutputStream

处理流：以字节为单位直接操作“对象”。

6. DataInputStream/DataOutputStream

处理流：以字节为单位直接操作“基本数据类型与字符串类型”。

7. FileReader/FileWriter

节点流：以字符为单位直接操作“文本文件”（注意：只能读写文本文件）。

8. BufferedReader/BufferedWriter

处理流：将 Reader/Writer 对象进行包装，增加缓存功能，提高读写效率。

9. BufferedInputStream/BufferedOutputStream

处理流：将 InputStream/OutputStream 对象进行包装，增加缓存功能，提高读写效率。

10. InputStreamReader/OutputStreamWriter

处理流：将字节流对象转化成字符流对象。

11. PrintStream

处理流：将 OutputStream 进行包装，可以方便地输出字符，更加灵活。

Oldlu 建议

上面的解释，一句话就点中了流的核心作用。大家在后面学习的时候，用心体会。

二、 IO 流入门案例

1 第一个简单的 IO 流程序

当程序需要读取数据源的数据时，就会通过 IO 流对象开启一个通向数据源的流，通过这个 IO 流对象的相关方法可以顺序读取数据源中的数据。

```
public class FirstDemo {
    public static void main(String[] args) {

        FileInputStream fis = null;
        try{
```

```
            //创建字节输入流对象
            fis = new FileInputStream("d:/a.txt");
            int s1 = fis.read();//打印输入字符 a 对应的 ascii 码值 97
            int s2 = fis.read();//打印输入字符 b 对应的 ascii 码值 98
            int s3 = fis.read();//打印输入字符 c 对应的 ascii 码值 99
            int s4 = fis.read();//由于文件内容已经读取完毕，则返回-1
            System.out.println(s1);
            System.out.println(s2);
            System.out.println(s3);
            System.out.println(s4);
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            if(fis != null){
               try{
                   fis.close();
               }catch (Exception e){
                   e.printStackTrace();
               }
            }
        }
    }
}
```

2 改造入门案例

```
public class SecondDemo {
    public static void main(String[] args) {
        FileInputStream fis = null;
        try {
            //创建字节输入流对象
            fis = new FileInputStream("d:/a.txt");
            StringBuilder sb = new StringBuilder();
            int temp = 0;
            while((temp = fis.read()) != -1){
                System.out.println(temp);
```

```
                sb.append((char) temp);
            }
            System.out.println(sb.toString());
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            try{
                if(fis != null){
                    fis.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }

    }
}
```

三、 File 类的使用

1 File 类简介

1.1 File 类的作用

File 类是 Java 提供的针对磁盘中的文件或目录转换对象的包装类。一个 File 对象而可以代表一个文件或目录，File 对象可以实现获取文件和目录属性等功能，可以实现对文件和目录的创建，删除等功能。

1.2 File 类操作目录与文件的常用方法

1.2.1 针对文件操作的方法

createNewFile()//创建新文件。

delete()//直接从磁盘上删除

exists()//查询磁盘中的文件是否存在

getAbsolutePath()//获取绝对路径

getPath()//获取相对路径

getName()//获取文件名 相当于调用了一个 toString 方法。

isFile()//判断是否是文件

length()//查看文件中的字节数

isHidden()//测试文件是否被这个抽象路径名是一个隐藏文件。

1.2.2 针对目录操作的方法

exists()//查询目录是否存在

isDirectory()//判断当前路径是否为目录

mkdir()//创建目录

getParentFile()//获取当前目录的父级目录。

list()//返回一个字符串数组，包含目录中的文件和目录的路径名。

listFiles//返回一个 File 数组，表示用此抽象路径名表示的目录中的文件。

2 File 类的基本使用

2.1操作文件

```
public class FileDemo {
    public static void main(String[] args)throws Exception {

        //创建 File 对象

        File file = new File("d:/aa.txt");
        System.out.println(file.createNewFile());
        //System.out.println(file.delete());
        System.out.println(file.exists());
        System.out.println(file.getName());
        System.out.println(file.isFile());
        System.out.println(file.isHidden());
```

```
    }
}
```

2.2操作目录

```
public class DirectoryDemo {
    public static void main(String[] args) {

        //创建 File 对象

        File file = new File("d:/b/c");
        //System.out.println(file.mkdir());
        //System.out.println(file.mkdirs());
        //System.out.println(file.exists());
        //System.out.println(file.isDirectory());
        //System.out.println(file.getParent());
       // System.out.println(file.getParentFile().getName());
        File file2 = new File("d:/");
        String[] arr = file2.list();
        for(String temp:arr){
            System.out.println(temp);
        }
        System.out.println("------------------------");
        File[] arr2 = file2.listFiles();
        for(File temp :arr2){
            System.out.println(temp);
        }
    }
}
```

四、 常用流对象

1 文件字节流

FileInputStream 通过字节的方式读取文件，适合读取所有类型的文件（图像、视频、文本文件等）。Java 也提供了 FileReader 专门读取文本文件。

FileOutputStream 通过字节的方式写数据到文件中，适合所有类型的文件。Java 也提供了 FileWriter 专门写入文本文件。

1.1文件字节输入流

```
public class FileStreamDemo {
    public static void main(String[] args) {
        FileInputStream fis = null;
        try{
            //创建文件字节输入流对象
            fis = new FileInputStream("d:/sxt.jpg");
            int temp = 0;
            while((temp = fis.read()) != -1){
                System.out.println(temp);
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(fis != null){
                    fis.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

1.2文件字节输出流

```
public class FileStreamDemo {
    public static void main(String[] args) {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        try{
            //创建文件字节输入流对象
            fis = new FileInputStream("d:/sxt.jpg");
            //创建文件字节输出流对象
```

```
            fos = new FileOutputStream("d:/aa.jpg");
            int temp = 0;
            while((temp = fis.read()) != -1){
                fos.write(temp);
            }

            //将数据从内存中写入到磁盘中。

            fos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(fis != null){
                    fis.close();
                }
                if(fos != null){
                    fos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

1.3通过缓冲区提高读写效率

1.3.1 方式一

通过创建一个指定长度的字节数组作为缓冲区，以此来提高 IO 流的读写效率。该方式适用于读取较大图片时的缓冲区定义。注意：缓冲区的长度一定是 2 的整数幂。一般情况下 1024 长度较为合适。

```
public class FileStreamBuffedDemo {
    public static void main(String[] args) {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        try{

            //创建文件字节输入流对象
```

```
            fis = new FileInputStream("d:/itbz.jpg");
            //创建文件字节输出流对象
            fos = new FileOutputStream("d:/cc.jpg");
            //创建一个缓冲区，提高读写效率
            byte[] buff = new byte[1024];
            int temp = 0;
            while((temp = fis.read(buff)) != -1){
                fos.write(buff,0,temp);
            }
            //将数据从内存中写入到磁盘中。
            fos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(fis != null){
                    fis.close();
                }
                if(fos != null){
                    fos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

1.3.2 方式二

通过创建一个字节数组作为缓冲区，数组长度是通过输入流对象的 available()返回当前文件的预估长度来定义的。在读写文件时，是在一次读写操作中完成文件读写操作的。注意：如果文件过大，那么对内存的占用也是比较大的。所以大文件不建议使用该方法。

```
public class FileStreamBuffer2Demo {
    public static void main(String[] args) {
```

```
        FileInputStream fis = null;
        FileOutputStream fos = null;
        try{
            //创建文件字节输入流对象
            fis = new FileInputStream("d:/itbz.jpg");
            //创建文件字节输出流对象
            fos = new FileOutputStream("d:/cc.jpg");
            //创建一个缓冲区，提高读写效率
            byte[] buff = new byte[fis.available()];
            fis.read(buff);
            //将数据从内存中写入到磁盘中。
            fos.write(buff);
            fos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(fis != null){
                    fis.close();
                }
                if(fos != null){
                    fos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

1.4通过字节缓冲流提高读写效率

Java 缓冲流本身并不具有 IO 流的读取与写入功能，只是在别的流（节点流或其他处理流）上加上缓冲功能提高效率，就像是把别的流包装起来一样，因此缓冲流是一种处理流(包装流)。

当对文件或者其他数据源进行频繁的读写操作时，效率比较低，这时如果使用缓冲流就能够更高效的读写信息。因为缓冲流是先将数据缓存起来，然后当缓存区存满后或者手动刷新时再一次性的读取到程序或写入目的地。

因此，缓冲流还是很重要的，我们在 IO 操作时记得加上缓冲流来提升性能。

BufferedInputStream 和 BufferedOutputStream 这两个流是缓冲字节流，通过内部缓存数组来提高操作流的效率。

```
public class FileStreamBuffed3Demo {
    public static void main(String[] args) {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        BufferedInputStream bis = null;
        BufferedOutputStream bos = null;
        try{
            fis = new FileInputStream("d:/itbz.jpg");
            bis = new BufferedInputStream(fis);
            fos = new FileOutputStream("d:/ff.jpg");
            bos = new BufferedOutputStream(fos);

            //缓冲流中的 byte 数组长度默认是 8192

            int temp = 0;
            while((temp = bis.read()) != -1){
                bos.write(temp);
            }
            bos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{

            //注意：关闭流顺序："后开的先关闭"

            if(bis != null){
                bis.close();
            }
            if(fis != null){
                fis.close();
            }
            if(bos != null){
```

```
                bos.close();
            }
            if(fos !=null){
                fos.close();
            }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

1.5定义文件拷贝工具类

```
public class FileCopyTools {
    public static void main(String[] args) {
            copyFile("d:/itbz.jpg","d:/abc.jpg");
    }
    /**
     * 文件拷贝方法
     */
    public static void copyFile(String src,String des){
        FileInputStream fis = null;
        BufferedInputStream bis = null;
        FileOutputStream fos = null;
        BufferedOutputStream bos = null;

        try{
            bis = new BufferedInputStream(new FileInputStream(src));
            bos = new BufferedOutputStream(new FileOutputStream(des));
            int temp = 0;
            while((temp = bis.read()) != -1){
                bos.write(temp);
            }
            bos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(bis != null){
                    bis.close();
```

```
            }
            if(fis != null){
                fis.close();
            }
            if(bos != null){
                bos.close();
            }
            if(fos != null){
                fos.close();
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
}
```

2 文件字符流

前面介绍的文件字节流可以处理所有的文件，如果我们处理的是文本文件，也可以使用文件字符流，它以字符为单位进行操作。

2.1文件字符输入流

```
public class FileReaderDemo {
    public static void main(String[] args) {
        FileReader frd = null;
        try{
            //创建文件字符输入流对象
            frd = new FileReader("d:/a.txt");
            int temp = 0;
            while((temp = frd.read()) != -1){
                System.out.println((char) temp);
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            try{
                if(frd != null){
```

```
                frd.close();
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

2.2文件字符输出流

```
public class FileWriterDemo {
    public static void main(String[] args) {
        FileWriter fw = null;
        FileWriter fw2 = null;
        try{
            //创建字符输出流对象
            fw = new FileWriter("d:/sxt.txt");
            fw.write("你好尚学堂\r\n");
            fw.write("你好 Oldlu\r\n");
            fw.flush();
            fw2 = new FileWriter("d:/sxt.txt",true);
            fw2.write("何以解忧\r\n 唯有尚学堂");
            fw2.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(fw != null){
                    fw.close();
                }
                if(fw2 != null){
                    fw2.close();
                }
            }catch(Exception e){
                e.printStackTrace();
```

```
            }
        }
    }
}
```

2.3使用字符流实现文本文件的拷贝处理

```
public class FileCopyTools2 {
    public static void main(String[] args) {
        FileReader fr = null;
        FileWriter fw = null;
        try{
            fr = new FileReader("d:/2.txt");
            fw = new FileWriter("d:/3.txt");
            char[] buffer = new char[1024];
            int temp = 0;
            while((temp = fr.read(buffer)) != -1){
                fw.write(buffer,0,temp);
            }
            fw.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(fr != null){
                    fr.close();
                }
                if(fw != null){
                    fw.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

3 字符缓冲流

BufferedReader/BufferedWriter 增加了缓存机制，大大提高了读写文本文件的效率。

3.1字符输入缓冲流

BufferedReader 是针对字符输入流的缓冲流对象，提供了更方便的按行读取的方法：readLine()；在使用字符流读取文本文件时，我们可以使用该方法以行为单位进行读取。

```
public class BufferedReaderDemo {
    public static void main(String[] args) {
        FileReader fr = null;
        BufferedReader br = null;
        try{
            fr = new FileReader("d:/sxt.txt");
            br = new BufferedReader(fr);
            String temp = "";
            while((temp = br.readLine()) != null){
                System.out.println(temp);
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            try{
                if(br != null){
                    br.close();
                }
                if(fr != null){
                    fr.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

3.2字符输出缓冲流

BufferedWriter 是针对字符输出流的缓冲流对象，在字符输出缓冲流中可以使用newLine()；方法实现换行处理。

```
public class BufferedWriterDemo {
```

```
    public static void main(String[] args) {
        FileWriter fw = null;
        BufferedWriter bw = null;
        try{
            fw = new FileWriter("d:/sxt2.txt");
            bw = new BufferedWriter(fw);

            bw.write("你好尚学堂");


            bw.write("你好 Oldlu");

            bw.newLine();

            bw.write("何以解忧");

            bw.newLine();

            bw.write("唯有尚学堂");

            bw.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(bw != null){
                    bw.close();
                }
                if(fw != null){
                    fw.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

3.3通过字符缓冲流实现文本文件的拷贝

```
public class FileCopyTools3 {
    public static void main(String[] args) {
            copyFile("d:/2.txt","d:/22.txt");
    }
    /**
```

```
    * 基于字符流缓冲流实现文件拷贝
    */
   public static void copyFile(String src,String des){
       BufferedReader br = null;
       BufferedWriter bw = null;
       try{
              br = new BufferedReader(new FileReader(src));
              bw = new BufferedWriter(new FileWriter(des));
              String temp = "";
              while((temp = br.readLine()) != null){
                  bw.write(temp);
                  bw.newLine();
              }
              bw.flush();
       }catch(Exception e){
          e.printStackTrace();
       }finally {
          try{
              if(br != null){
                  br.close();
              }
              if(bw != null){
                  bw.close();
              }
          }catch(Exception e){
              e.printStackTrace();
          }
       }

   }
}
```

3.4通过字符缓冲流为文件中的内容添加行号

```
public class LineNumberDemo {
   public static void main(String[] args) {
       BufferedReader br = null;
       BufferedWriter bw = null;
       try{
          br = new BufferedReader(new FileReader("d:/sxt2.txt"));
          bw = new BufferedWriter(new FileWriter("d:/sxt3.txt"));
```

```
            String temp = "";
            int i = 1;
            while((temp = br.readLine()) != null){
                bw.write(i+","+temp);
                bw.newLine();
                i++;
            }
            bw.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(br != null){
                    br.close();
                }
                if(bw != null){
                    bw.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

4 转换流

InputStreamReader/OutputStreamWriter 用来实现将字节流转化成字符流。比如，如下场景：

System.in 是字节流对象，代表键盘的输入，如果我们想按行接收用户的输入时，就必须用到缓冲字符流 BufferedReader 特有的方法 readLine()，但是经过观察会发现在创建 BufferedReader 的构造方法的参数必须是一个 Reader 对象，这时候我们的转换流 InputStreamReader 就派上用场了。

而 System.out 也是字节流对象，代表输出到显示器，按行读取用户的输入后，并且要将

读取的一行字符串直接显示到控制台，就需要用到字符流的 write(String str)方法，所以我们要使用 OutputStreamWriter 将字节流转化为字符流。

4.1通过转换流实现键盘输入屏幕输出

```
public class ConvertStream {
    public static void main(String[] args) {
        BufferedReader br = null;
        BufferedWriter bw = null;
      try{
          br = new BufferedReader(new InputStreamReader(System.in));
          bw = new BufferedWriter(new
OutputStreamWriter(System.out));
         while(true){

            bw.write("请输入：");

            bw.flush();
            String input = br.readLine();
            if("exit".equals(input)){
                break;
            }

            bw.write("你输入的是："+input);

            bw.newLine();
            bw.flush();
         }
      }catch(Exception e){
          e.printStackTrace();
      }finally{
          try{
              if(bw != null){
                  bw.close();
              }
              if(br != null){
                  br.close();
              }

          }catch(Exception e){
              e.printStackTrace();
          }
      }
```

```
    }
}
```

4.2通过字节流读取文本文件并添加行号

```
public class LineNumberDemo2 {
    public static void main(String[] args) {
        BufferedReader br = null;
        BufferedWriter bw = null;
        try{
            br = new BufferedReader(new InputStreamReader(new
FileInputStream("d:/sxt.txt")));
            bw = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream("d:/sxt3.txt")));
            String temp = "";
            int i = 1;
            while((temp = br.readLine()) != null){
                bw.write(i+","+temp);
                bw.newLine();
                i++;
            }
            bw.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(br != null){
                    br.close();
                }
                if(bw != null ){
                    bw.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

5 字符输出流

在 Java 的 IO 流中专门提供了用于字符输出的流对象 PrintWriter。该对象具有自动行刷新缓冲字符输出流，特点是可以按行写出字符串，并且可通过 println();方法实现自动换行。

```
public class LineNumberDemo3 {
    public static void main(String[] args) {
        BufferedReader br = null;
        PrintWriter pw = null;
        try{
            br = new BufferedReader(new InputStreamReader(new
FileInputStream("d:/sxt.txt")));
            pw = new PrintWriter("d:/sxt4.txt");
            String temp ="";
            int i = 1;
            while((temp = br.readLine()) != null){
                pw.println(i+","+temp);
                i++;
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            try{
                if(br != null){
                    br.close();
                }
                if(pw != null){
                    pw.close();
                }
            }catch (Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

6 字节数组流

ByteArrayInputStream 和 ByteArrayOutputStream 经常用在需要流和数组之间转化的情

况！

6.1字节数组输入流

说白了，FileInputStream 是把文件当做数据源。ByteArrayInputStream 则是把内存中的”字节数组对象”当做数据源。

```
public class ByteArrayInputDemo {
    public static void main(String[] args) {
        byte[] arr = "abcdefg".getBytes();
        ByteArrayInputStream bis = null;
        StringBuilder sb = new StringBuilder();
        try{
            //该构造方法的参数是一个字节数组，这个字节数组就是数据源
            bis = new ByteArrayInputStream(arr);
            int temp = 0;
            while ((temp = bis.read()) != -1){
                sb.append((char)temp);
            }
            System.out.println(sb.toString());
        }finally{
            try{
                bis.close();
            }catch (Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

6.2字节数组输出流

ByteArrayOutputStream 流对象是将流中的数据写入到字节数组中。

```
public class ByteArrayOutputDemo {
    public static void main(String[] args) {
        ByteArrayOutputStream bos = null;
        try{
```

```
            StringBuilder sb = new StringBuilder();
                bos = new ByteArrayOutputStream();
                bos.write('a');
                bos.write('b');
                bos.write('c');
                byte[] arr = bos.toByteArray();
                for(int i=0;i<arr.length;i++){
                    sb.append((char)arr[i]);
                }
            System.out.println(sb.toString());
        }finally {
            try{
                if(bos != null){
                    bos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

7 数据流

数据流将“基本数据类型与字符串类型”作为数据源，从而允许程序以与机器无关的方式从底层输入输出流中操作 Java 基本数据类型与字符串类型。

DataInputStream 和 DataOutputStream 提供了可以存取与机器无关的所有 Java 基础类型数据（如：int、double、String 等）的方法。

7.1数据输出流

```
public class DataOutputDemo {
    public static void main(String[] args) {
        DataOutputStream dos = null;
        try{
          dos = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream("d:/data.txt")));
            dos.writeChar('a');
```

```
            dos.writeInt(10);
            dos.writeDouble(Math.random());
            dos.writeBoolean(true);

            dos.writeUTF("你好尚学堂");

            dos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(dos != null){
                    dos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

7.2数据输入流

```
public class DataInputDemo {
    public static void main(String[] args) {
        DataInputStream dis = null;
        try{
            dis = new DataInputStream(new BufferedInputStream(new
FileInputStream("d:/data.txt")));

            //直接读取数据，注意：读取的顺序要与写入的顺序一致，否则不能正确读

取数据。

            System.out.println("char: "+dis.readChar());
            System.out.println("int: "+dis.readInt());
            System.out.println("double: "+dis.readDouble());
            System.out.println("boolean: "+dis.readBoolean());
            System.out.println("String: "+dis.readUTF());
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(dis != null ){
```

```
                dis.close();
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

8 对象流

对象的本质是用来组织和存储数据的，对象本身也是数据。那么，能不能将对象存储到硬盘上的文件中呢？能不能将对象通过网络传输到另一个电脑呢？我们可以通过序列化和反序列化来实现这些需求。

8.1 Java 对象的序列化和反序列化

8.1.1 序列化和反序列化是什么

当两个进程远程通信时，彼此可以发送各种类型的数据。 无论是何种类型的数据，都会以二进制序列的形式在网络上传送。比如，我们可以通过 http 协议发送字符串信息；我们也可以在网络上直接发送 Java 对象。发送方需要把这个 Java 对象转换为字节序列，才能在网络上传送；接收方则需要把字节序列再恢复为 Java 对象才能正常读取。

把 Java 对象转换为字节序列的过程称为**对象的序列化**。把字节序列恢复为 Java 对象的过程称为**对象的反序列化**。

对象序列化的作用有如下两种：

- **持久化：** 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中。
- **网络通信：**在网络上传送对象的字节序列。比如：服务器之间的数据通信、对象传递。

8.1.2 序列化涉及的类和接口

ObjectOutputStream 代表对象输出流，它的 writeObject(Object obj)方法可对参数指定的 obj 对象进行序列化，把得到的字节序列写到一个目标输出流中。

ObjectInputStream 代表对象输入流，它的 readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。

只有实现了 Serializable 接口的类的对象才能被序列化。 Serializable 接口是一个空接口，只起到标记作用。

8.2操作基本数据类型

我们前边学到的数据流只能实现对基本数据类型和字符串类型的读写，并不能对 Java 对象进行读写操作（字符串除外），但是在对象流中除了能实现对基本数据类型进行读写操作以外，还可以对 Java 对象进行读写操作。

8.2.1 写出基本数据类型数据

```
public class ObjectOutputStreamBasicTypeDemo {
    public static void main(String[] args) {
        ObjectOutputStream oos = null;
        try{
            oos = new ObjectOutputStream(new BufferedOutputStream(new
FileOutputStream("d:/sxt5.txt")));
            oos.writeInt(10);
            oos.writeDouble(Math.random());
            oos.writeChar('a');
            oos.writeBoolean(true);

            oos.writeUTF("你好 Oldlu");

            oos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally {
```

```
            try{
                if(oos != null){
                    oos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

8.2.2 读取基本数据类型数据

```
public class ObjectInputStreamBasicTypeDemo {
    public static void main(String[] args) {
        ObjectInputStream ois = null;
        try{
            ois = new ObjectInputStream(new BufferedInputStream(new
FileInputStream("d:/sxt5.txt")));

            //必须要按照写入的顺序读取数据

            System.out.println("int: "+ois.readInt());
            System.out.println("double: "+ois.readDouble());
            System.out.println("char: "+ois.readChar());
            System.out.println("boolean: "+ois.readBoolean());
            System.out.println("String: "+ois.readUTF());
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(ois != null){
                    ois.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

8.3操作对象

8.3.1 将对象序列化到文件

ObjectOutputStream 可以将一个内存中的 Java 对象通过序列化的方式写入到磁盘的文件中。被序列化的对象必须要实现 Serializable 序列化接口，否则会抛出异常。

8.3.1.1 创建对象

```
public class Users implements Serializable {
    private int userid;
    private String username;
    private String userage;

    public Users(int userid, String username, String userage) {
        this.userid = userid;
        this.username = username;
        this.userage = userage;
    }

    public Users() {
    }

    public int getUserid() {
        return userid;
    }

    public void setUserid(int userid) {
        this.userid = userid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUserage() {
```

```
        return userage;
    }

    public void setUserage(String userage) {
        this.userage = userage;
    }
}
```

8.3.1.2 序列化对象

```
public class ObjectOutputStreamObjectTypeDemo {
    public static void main(String[] args) {
        ObjectOutputStream oos = null;
        try{
            oos = new ObjectOutputStream(new
FileOutputStream("d:/sxt6.txt"));
            Users users = new Users(1,"Oldlu","18");
            oos.writeObject(users);
            oos.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(oos != null){
                    oos.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

8.3.2 将对象反序列化到内存中

```
public class ObjectInputStreamObjectTypeDemo {
    public static void main(String[] args) {
        ObjectInputStream ois = null;
        try{
```

```
            ois = new ObjectInputStream(new
FileInputStream("d:/sxt6.txt"));
            Users users =  (Users)ois.readObject();

System.out.println(users.getUserid()+"\t"+users.getUsername()+"\t
"+users.getUserage());
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(ois != null){
                    ois.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

9 随机访问流

RandomAccessFile 可以实现两个作用：

1. 实现对一个文件做读和写的操作。
2. 可以访问文件的任意位置。不像其他流只能按照先后顺序读取。

在开发某些客户端软件时，经常用到这个功能强大的可以”任意操作文件内容”的类。比如，软件的使用次数和使用日期，可以通过本类访问文件中保存次数和日期的地方进行比对和修改。 Java 很少开发客户端软件，所以在 Java 开发中这个类用的相对较少。

学习这个流我们需掌握三个核心方法：

1. RandomAccessFile(String name, String mode) name 用来确定文件； mode 取 r(读)或 rw(可读写)，通过 mode 可以确定流对文件的访问权限。
2. seek(long a) 用来定位流对象读写文件的位置，a 确定读写位置距离文件开头

的字节个数。

3. getFilePointer()　获得流的当前读写位置。

```
public class RandomAccessFileDemo {
    public static void main(String[] args) {
        RandomAccessFile raf = null;
        try{
            raf = new RandomAccessFile("d:/sxt7.txt","rw");

            //将若干数据写入到文件中

            int[] arr = new int[]{10,20,30,40,50,60,70,80,90,100};
            for(int i=0;i<arr.length;i++){
                raf.writeInt(arr[i]);
            }
            raf.seek(4);
            System.out.println(raf.readInt());


            //隔一个读一个数据

            for(int i=0;i<10; i+=2){
                raf.seek(i*4);
                System.out.print(raf.readInt()+"\t");
            }
            System.out.println();

            //在第 8 个字节位置插入一个新的数据 45，替换之前的数据 30

            raf.seek(8);
            raf.writeInt(45);
            for(int i=0;i<10; i+=2){
                raf.seek(i*4);
                System.out.print(raf.readInt()+"\t");
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            try{
                if(raf != null){
                    raf.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
```

```
        }
    }
}
```

10 File 类在 IO 中的作用

当以文件作为数据源或目标时，除了可以使用字符串作为文件以及位置的指定以外，我们也可以使用 File 类指定。

```
public class FileInIODemo {
    public static void main(String[] args) {
        BufferedReader br = null;
        BufferedWriter bw = null;
        try{
            br = new BufferedReader(new FileReader(new
File("d:/sxt.txt")));
            bw = new BufferedWriter(new FileWriter(new
File("d:/sxt8.txt")));
            String temp = "";
            int i =1;
            while((temp = br.readLine()) != null){
                    bw.write(i+","+temp);
                    bw.newLine();
                    i++;
            }
            bw.flush();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                if(br != null){
                    br.close();
                }
                if(bw != null){
                    bw.close();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
```

```
    }
}
```

五、 Apache IO 包

JDK 中提供的文件操作相关的类，但是功能都非常基础，进行复杂操作时需要做大量编程工作。实际开发中，往往需要你自己动手编写相关的代码，尤其在遍历目录文件时，经常用到递归，非常繁琐。 Apache-commons 工具包中提供了 IOUtils/FileUtils，可以让我们非常方便的对文件和目录进行操作。 本文就是让大家对 IOUtils/FileUtils 类有一个全面的认识，便于大家以后开发与文件和目录相关的功能。

Apache IOUtils 和 FileUtils 类库为我们提供了更加简单、功能更加强大的文件操作和 IO 流操作功能。非常值得大家学习和使用。

1 Apache 基金会介绍

Apache 软件基金会（也就是 Apache Software Foundation，简称为 ASF），是专门为支持开源软件项目而办的一个非盈利性组织。在它所支持的 Apache 项目与子项目中，所发行的软件产品都遵循 Apache 许可证（Apache License）。 官方网址为：www.apache.org 。

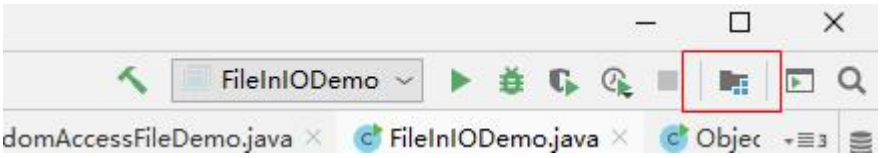
很多著名的 Java 开源项目都来源于这个组织。比如：commons、kafka、lucene、maven、shiro、 struts 等技术，以及大数据技术中的：hadoop（大数据第一技术）、 hbase、 spark、 storm、 mahout 等。

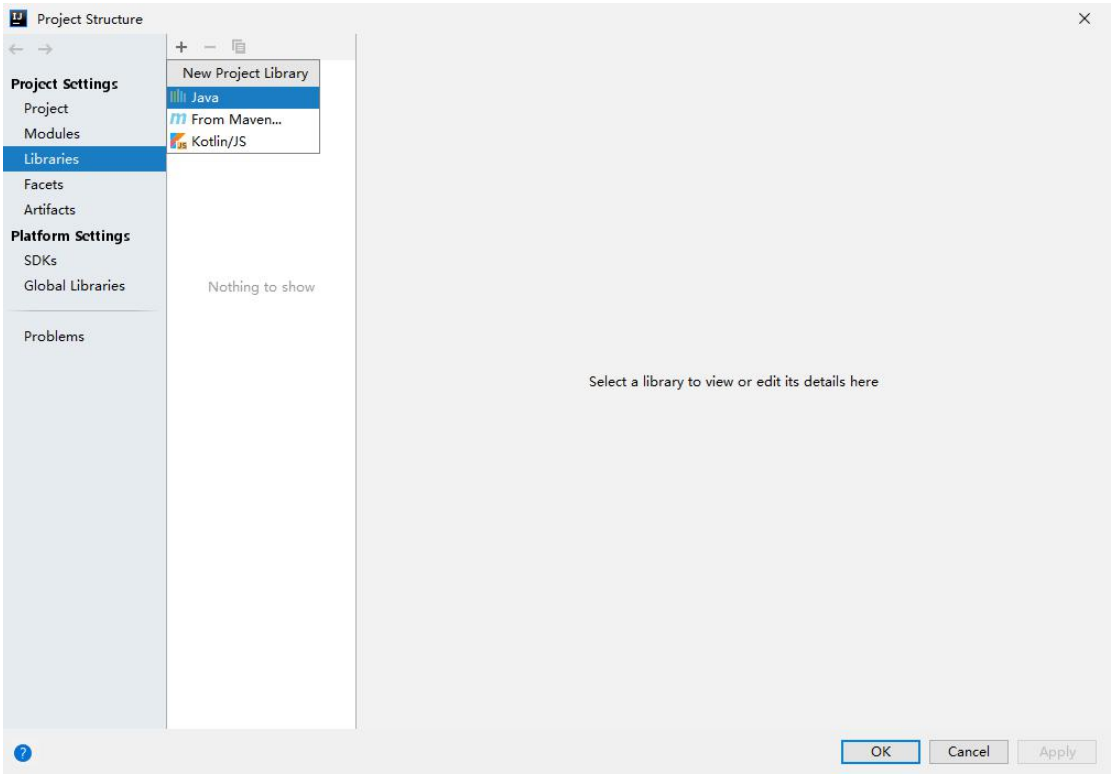
2 下载与添加 commons-io 包

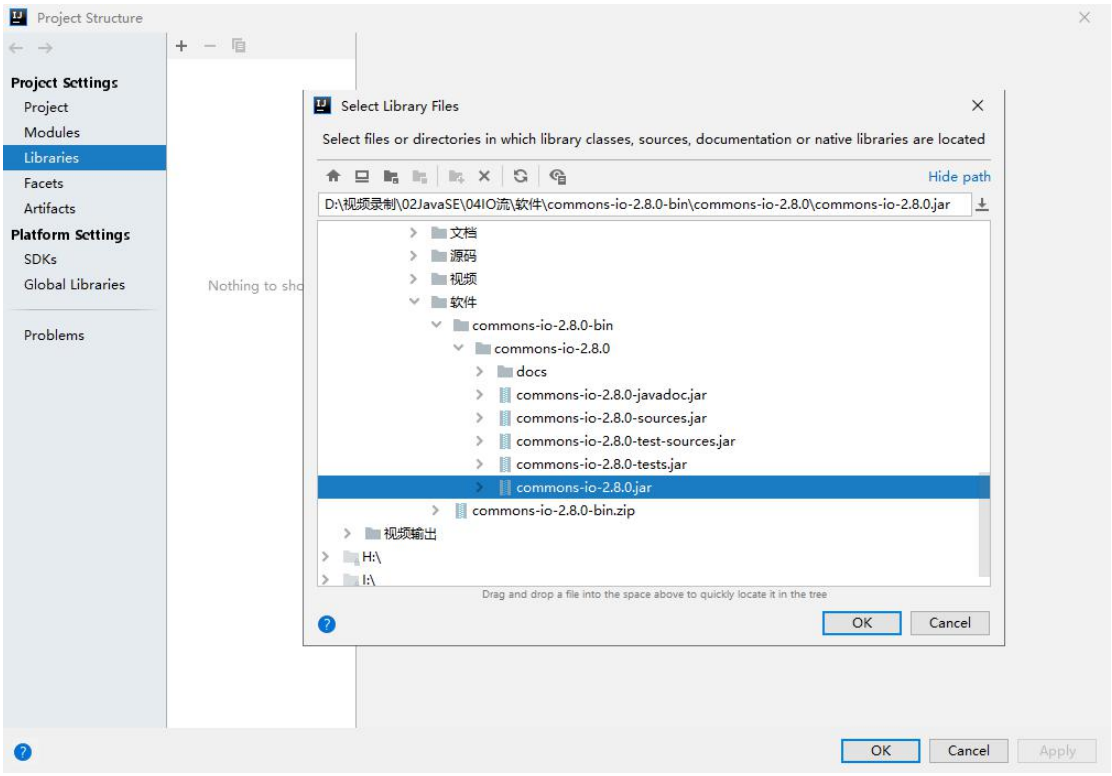
2.1 下载地址

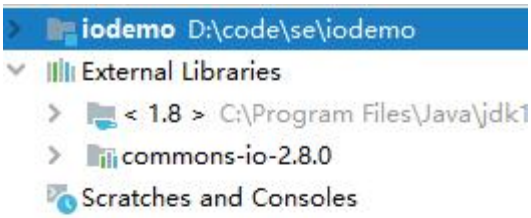
https://commons.apache.org/proper/commons-io/download_io.cgi

2.2 添加 jar 包






Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
Nothing to sho
Select Library Files
Select files or directories in which library classes, sources, documentation or native libraries are located
Hide path
D:\视频录制\02JavaSE\04IO流\软件\commons-io-2.8.0-bin\commons-io-2.8.0\commons-io-2.8.0.jar
文档
源码
视频
软件
commons-io-2.8.0-bin
commons-io-2.8.0
docs
commons-io-2.8.0-javadoc.jar
commons-io-2.8.0-sources.jar
commons-io-2.8.0-test-sources.jar
commons-io-2.8.0-tests.jar
commons-io-2.8.0.jar
commons-io-2.8.0-bin.zip
视频输出
H:\
I:\
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel
OK
Cancel
Apply


iodemo D:\code\se\iodemo
External Libraries
< 1.8 > C:\Program Files\Java\jdk1
commons-io-2.8.0
Scratches and Consoles

3 FileUtils 的使用

FileUtils 类中常用方法：

打开 FileUtils 的 api 文档，我们抽出一些工作中比较常用的方法，进行总结和讲解。总结如下：

cleanDirectory：清空目录，但不删除目录。

contentEquals：比较两个文件的内容是否相同。

copyDirectory：将一个目录内容拷贝到另一个目录。可以通过 FileFilter 过滤需要拷贝的文件。

copyFile：将一个文件拷贝到一个新的地址。

copyFileToDirectory：将一个文件拷贝到某个目录下。

copyInputStreamToFile：将一个输入流中的内容拷贝到某个文件。

deleteDirectory：删除目录。

deleteQuietly：删除文件。

listFiles：列出指定目录下的所有文件。

openInputSteam：打开指定文件的输入流。

readFileToString：将文件内容作为字符串返回。

readLines：将文件内容按行返回到一个字符串数组中。

size：返回文件或目录的大小。

write：将字符串内容直接写到文件中。

writeByteArrayToFile:将字节数组内容写到文件中。

writeLines：将容器中的元素的 toString 方法返回的内容依次写入文件中。

writeStringToFile：将字符串内容写到文件中。

3.1 FileUtils 的使用一

```
public class FileUtilsDemo1 {
    public static void main(String[] args)throws Exception {
       String content =  FileUtils.readFileToString(new
File("d:/sxt.txt"),"utf-8");
        System.out.println(content);
    }
}
```

3.2 FileUtils 的使用二

```
public class FileUtilsDemo2 {
    public static void main(String[] args)throws Exception {
        FileUtils.copyDirectory(new File("d:/a"), new File("c:/a"),
new FileFilter() {

            //在文件拷贝时的过滤条件

            @Override
            public boolean accept(File pathname) {
                if(pathname.isDirectory() ||
pathname.getName().endsWith("html")){
                    return true;
                }
                return false;
            }
        });
    }
}
```

4 IOUtils 的使用

打开 IOUtils 的 api 文档，我们发现它的方法大部分都是重载的。所以，我们理解它的方法并不是难事。因此，对于方法的用法总结如下：

buffer 方法：将传入的流进行包装，变成缓冲流。并可以通过参数指定缓冲大小。

closeQueitly 方法：关闭流。

contentEquals 方法：比较两个流中的内容是否一致。

copy 方法：将输入流中的内容拷贝到输出流中，并可以指定字符编码。

copyLarge 方法：将输入流中的内容拷贝到输出流中，适合大于 2G 内容的拷贝。

lineIterator 方法：返回可以迭代每一行内容的迭代器。

read 方法：将输入流中的部分内容读入到字节数组中。

readFully 方法：将输入流中的所有内容读入到字节数组中。

readLine 方法：读入输入流内容中的一行。

toBufferedInputStream，toBufferedReader：将输入转为带缓存的输入流。

toByteArray，toCharArray：将输入流的内容转为字节数组、字符数组。

toString：将输入流或数组中的内容转化为字符串。

write 方法：向流里面写入内容。

writeLine 方法：向流里面写入一行内容。

```
public class IOUtilsDemo {
    public static void main(String[] args)throws Exception {
        String content = IOUtils.toString(new
FileInputStream("d:/sxt.txt"),"utf-8");
        System.out.println(content);
    }
}
```

六、 本章总结

- 按流的方向分类：

- 输入流：数据源到程序(InputStream、 Reader 读进来)。
- 输出流：程序到目的地(OutPutStream、 Writer 写出去)。

❑ 按流的处理数据单元分类：

- 字节流：按照字节读取数据(InputStream、 OutputStream)。
- 字符流：按照字符读取数据(Reader、 Writer)。

❑ 按流的功能分类：

- 节点流：可以直接从数据源或目的地读写数据。
- 处理流：不直接连接到数据源或目的地，是处理流的流。通过对其他流的处理提高程序的性能。

❑ IO 的四个基本抽象类：InputStream、 OutputStream、 Reader、 Writer

❑ InputStream 的实现类：

- FileInputStream
- ByteArrayInutStream
- BufferedInputStream
- DataInputStream
- ObjectInputStream

❑ OutputStream 的实现类：

- FileOutputStream
- ByteArrayOutputStream
- BufferedOutputStream
- DataOutputStream
- ObjectOutputStream
- PrintStream

❑ Reader 的实现类

- FileReader
- BufferedReader
- InputStreamReader

❑ Writer 的实现类

- FileWriter

- BufferedWriter
- OutputStreamWriter

- 把 Java 对象转换为字节序列的过程称为对象的序列化。

- 把字节序列恢复为 Java 对象的过程称为对象的反序列化。