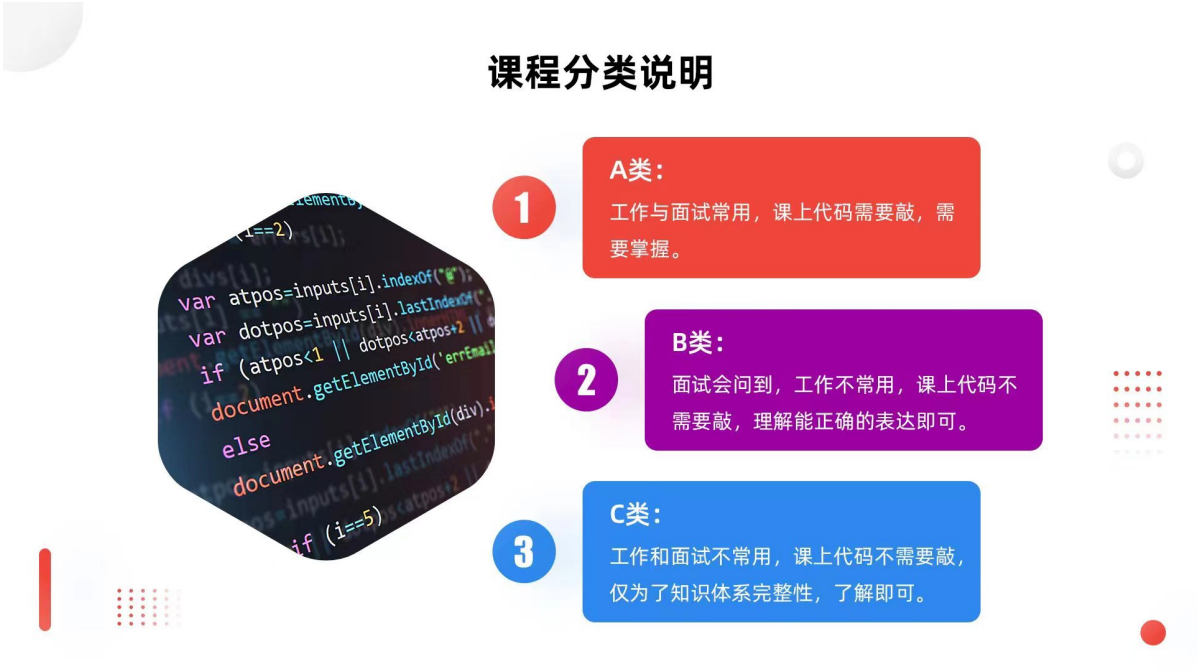
IO流章节说明



学习计划说明

总学时：5时32分，分为3天学习。

第一天总学时：1时45分钟

序号	课程名称	时长	分类
1	IO流介绍	11分15秒	A类
2	第一个简单的IO流程序	13分04秒	A类
3	IO流的经典写法	09分42秒	A类
4	IO流新语法经典写法	12分08秒	A类
5	Java中流的概念细分	11分26秒	A类
6	Java中IO流类的体系	08分39秒	A类
7	Java中IO的四大抽象类	06分02秒	A类
8	常用流详解-文件字节流-文件字节流的使用	15分07秒	A类
9	常用流详解-通过字节缓冲区提高读写效率	18分22秒	A类

第二天总学时：1时50分钟

序号	课程名称	时长	分类
10	常用流详解-文件字节流-缓冲字节流的使用	13分31秒	A类
11	常用流详解-文件字符流-文件字符流的使用	12分07秒	A类
12	常用流详解-文件字符流-缓冲字符流	16分37秒	A类
13	常用流详解-文件字符流-为文件中的内容添加行号	09分00秒	A类
14	常用流详解-转换流-通过转换流解决乱码	14分43秒	A类
15	常用流-转换流-通过字节流读取文本文件并添加行号	11分17秒	A类
16	常用流-转换流-通过转换流实现键盘输入屏幕输出	17分50秒	A类
17	常用流-字符输出流-字符输出流的使用	07分40秒	A类
18	常用流-字符输出流-通过字符输出流添加行号	08分09秒	A类

第三天总学时：1时55分钟

序号	课程名称	时长	分类
19	常用流-数据流-数据流的使用	14分13秒	C类
20	常用流-对象流-对象流的使用	14分30秒	A类
21	常用流-对象流-对象的序列化与反序列化介绍	08分25秒	A类
22	常用流-对象流-将对象序列化到文件	10分00秒	A类
23	常用流-对象流-将对象反序列化到内存中	07分38秒	A类
24	File类在IO中的作用	10分19秒	B类
25	装饰器模式构建IO流体系	11分29秒	C类
26	ApacheIO包-介绍	07分39秒	A类
27	ApacheIO包-FileUtils的使用一	08分59秒	A类
28	ApacheIO包-FileUtils的使用二	08分39秒	A类
29	ApacheIO包-IOUtils的使用	07分55秒	A类
30	本章总结	06分13秒	A类

实操说明

A类课程中的内容需要同学们跟着老师动手完成。

只要跟着课程一行一行代码照着敲，熟能生巧，一会能学的会！

IO流技术介绍

什么是IO

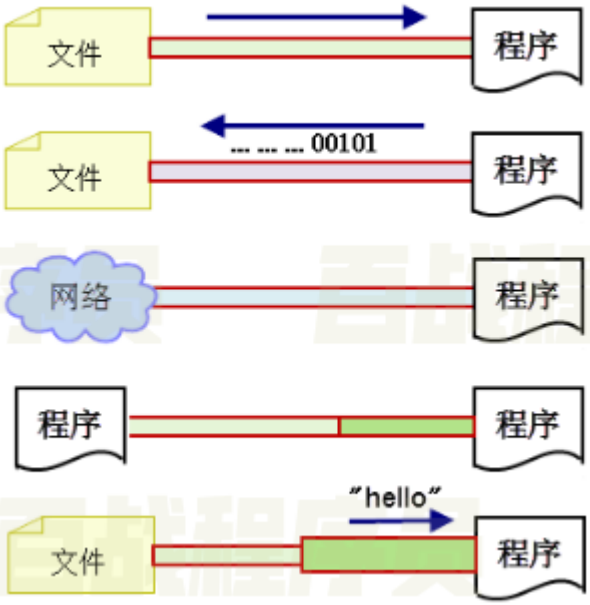
输入(Input)指的是：可以让程序从外部系统获得数据（核心含义是“读”，读取外部数据）。

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）。

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。

什么是数据源

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。如图所示。



数据源分为：源设备、目标设备。

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

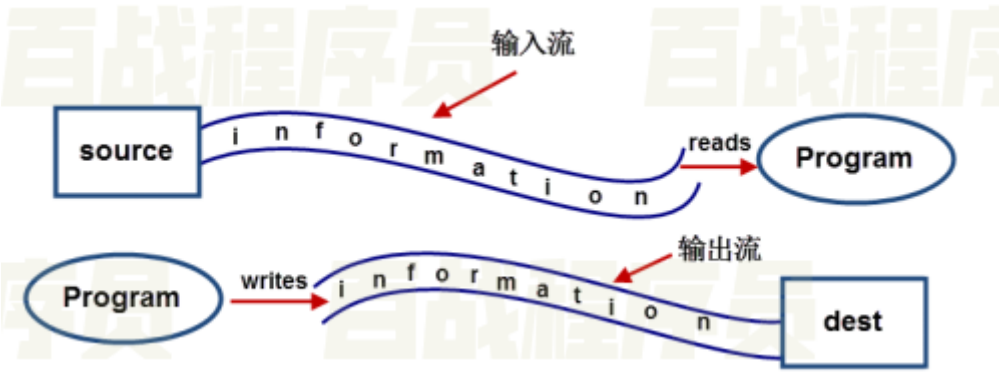
流的概念

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。

流与源数据源和目标数据源之间的关系:



> **Oldlu提示**
>
> 输入/输出流的划分是相对程序而言的，并不是相对数据源。

实时效果反馈

1.流的输入输出走向是站在哪一侧来看待的？

A 数据源一侧；

B 程序一侧；

C 数据源或程序任意一侧；

D 以上都不是；

答案

1=>B

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。

使用流读取文件内容(不规范的写法，仅用于测试)

```
import java.io.*;
public class TestI01 {
    public static void main(String[] args) {
        try {
            //创建输入流
            FileInputStream fis = new FileInputStream("d:/a.txt"); //
文件内容是: abc
            //一个字节一个字节的读取数据
            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，返回-1
            System.out.println(s1);
            System.out.println(s2);
            System.out.println(s3);
```

```
            System.out.println(s4);
            // 流对象使用完，必须关闭！不然，总占用系统资源，最终会造成系统崩溃!
            fis.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

以上案例我们要注意以下几点：

1. 我们读取的文件内容是已知的，因此可以使用固定次数的“**int** s= fis.read();”语句读取内容，但是在实际开发中通常我们根本不知道文件的内容，因此我们在读取的时候需要配合while循环使用。
2. 为了保证出现异常后流的正常关闭，通常要将流的关闭语句要放到finally语句块中，并且要判断流是不是null。

实时效果反馈

1.在Java的IO流中读取一个字节的方法是？

A input()；

B entry()；

C read()；

D rd()；

答案

1=>C

IO流的经典写法

使用流读取文件内容(经典代码，一定要掌握)

```java
import java.io.*;
public class Test2 {
    public static void main(String[] args) {
        FileInputStream fis = null;
        try {
            fis = new FileInputStream("d:/a.txt"); // 内容是：abc
            StringBuilder sb = new StringBuilder();
            int temp = 0;
            //当temp等于-1时，表示已经到了文件结尾，停止读取
            while ((temp = fis.read()) != -1) {
                sb.append((char) temp);
            }
            System.out.println(sb);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                //这种写法，保证了即使遇到异常情况，也会关闭流对象。
                if (fis != null) {
                    fis.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

```
27  }
28
```

实时效果反馈

1.在Java的IO流中，read()方法返回-1表示？

A 内容已读完时；

B 读取内容中有-1时；

C 读取内容失败时；

D 读取内容第一个字节时；

答案

1=>A

IO流新语法经典写法

在JDK7以及以后的版本中可以使用try-with-resource语法更优雅的关闭资源。

java.lang.AutoCloseable接口：

在java.lang.AutoCloseable接口中包含了一个close方法，该方法用于关闭资源。

只要是实现了java.lang.AutoCloseable接口的对象，都可以使用try-with-resource关闭资源。

使用最新的try-with-resource简化(经典代码，一定要掌握)

```java
public class Test3 {
  public static void main(String[] args) {
    //使用try-with-resource方式关闭资源。
    //在try中打开资源，不需要在代码中添加finally块关闭资源。
    try(FileInputStream fis = new FileInputStream("d:/a.txt");){
      StringBuilder sb = new StringBuilder();
      int temp=0;
      while((temp = fis.read()) != -1){
        sb.append((char) temp);


       }
      System.out.println(sb);
     }catch(Exception e){
      e.printStackTrace();
     }
   }

```

> **Oldlu建议**
>
> 如上代码是一段非常典型的IO流代码，其他流对象的使用也基本是同样的模式！

实时效果反馈

1.try-with-resource语法是在JDK的哪个版本中开始使用的？

A 1.5；

B 1.6；

C 1.7；

D 1.8；

2.try-with-resource可以对哪些对象关闭资源？

A 可以对所有需要做关闭资源的对象；

B 只能对特定的对象做资源关闭；

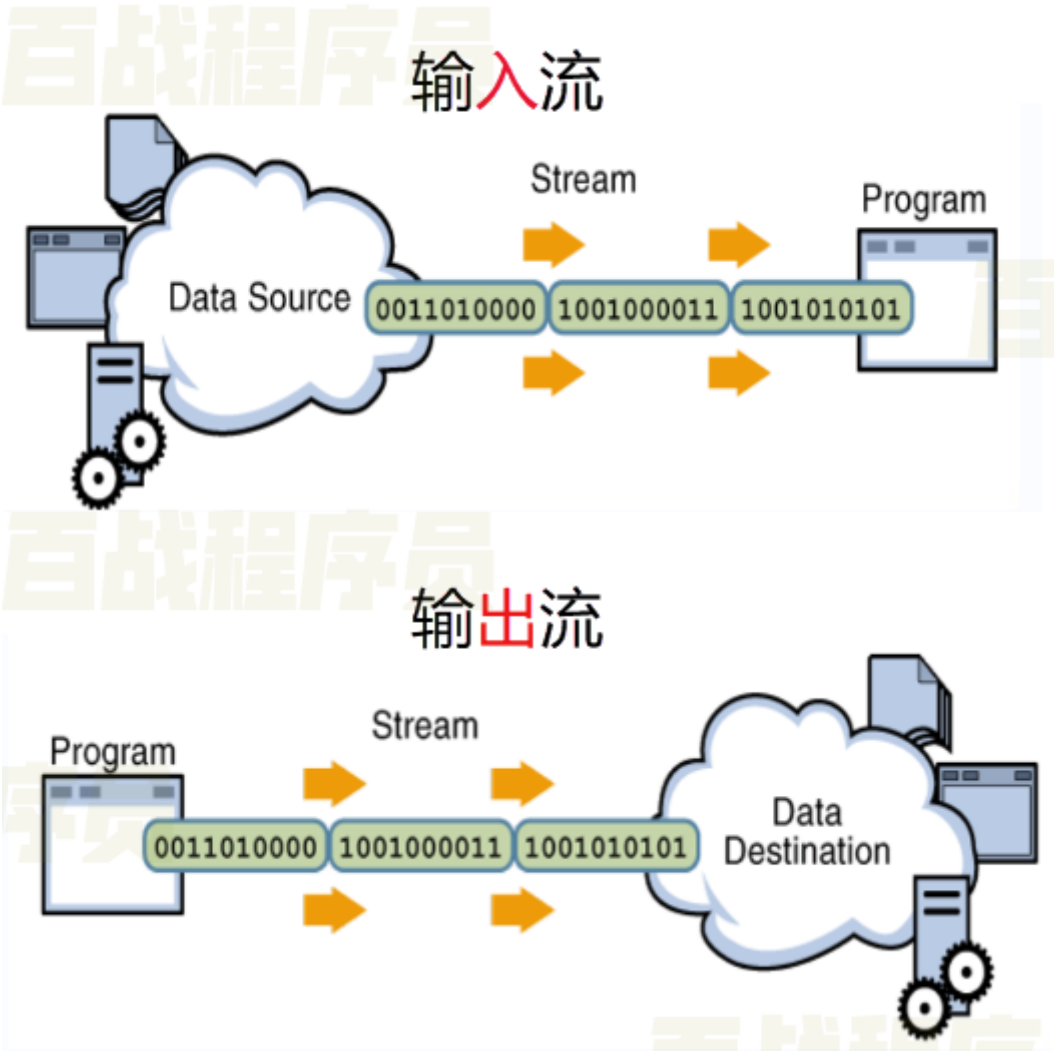
C 实现了java.lang.Closeable接口的对象

D 实现了java.lang.AutoCloseable接口的对象；

答案

1=>C 2=>D

Java中流的概念细分



按流的方向分类：

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

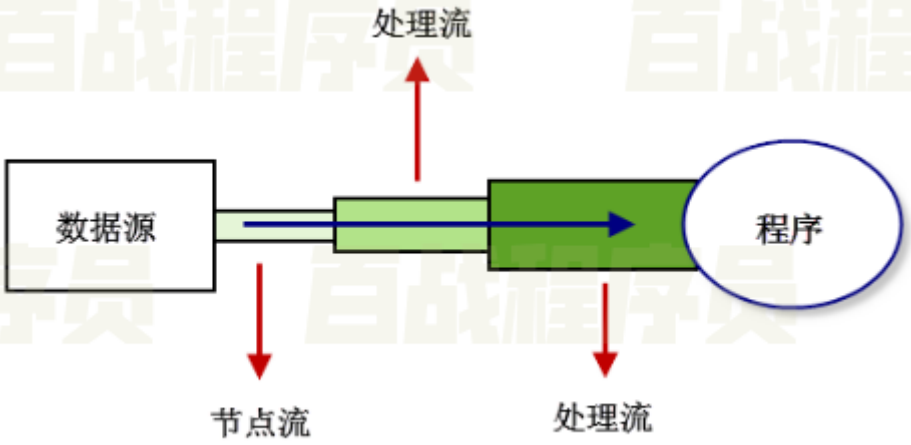
按处理的数据单元分类：

- 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。



实时效果反馈

1.Java的IO流，按处理的数据单元分为？

A 字节流与字符流；

B 节点流；

C 处理流

D 数据流；

2.Java的IO流，按处理对象不同分为？

A 字节流；

B 字符流；

C 数据流

D 节点流与处理流；

答案

1=>A 2=>D

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，如图所示，为Java中IO流类的体系。

> **注：**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。

实时效果反馈

1.FileInputStream是什么流？

A 文件的字符输入流；

B 文件的字符输出流；

C 文件的字节输入流；

D 文件的字节输出流；

2.FileReader是什么流？

A 文件的字符输入流；

B 文件的字符输出流；

C 文件的字节输入流；

D 文件的字节输出流；

答案

1=>C 2=>A

Java中IO的四大抽象类

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：

方法名	使用说明
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：

方法名	使用说明
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。

方法名	使用说明
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。

方法名	使用说明
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

实时效果反馈

1.InputStream/OutputStream是什么流？

A 字节流；

B 字符流；

C 随机访问流；

D 对象流；

2.Reader/Writer是什么流？

A 字节流；

B 字符流；

C 随机访问流；

D 对象流；

答案

1=>A 2=>B

常用流详解

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）。

FileOutputStream 通过字节的方式写数据到文件中，适合所有类型的文件（图像、视频、文本文件等）。

FileInputStream文件输入字节流

```
public class TestFileInputStream {
  public static void main(String[] args) {
    //使用try-with-resource方式关闭资源。
    //在try中打开资源，不需要在代码中添加finally块关闭资源。
    try(FileInputStream fis = new FileInputStream("d:/a.txt");){
      StringBuilder sb = new StringBuilder();
      int temp=0;
      while((temp = fis.read()) != -1){
        sb.append((char) temp);


       }
      System.out.println(sb);
     }catch(Exception e){
      e.printStackTrace();
     }
   }

```

FileOutputStream文件输出字节流

```
public class TestFileOutputStream {
  public static void main(String[] args) {
    String str = "Old Lu";
    // true表示内容会追加到文件末尾；false表示重写整个文件内容。
    try(FileOutputStream fos = new FileOutputStream("d:/a.txt",true))
{
      //将整个字节数组写入到文件中。
      fos.write(str.getBytes());
      //将数据从内存中写入到磁盘中。
      fos.flush();
     }catch (IOException e){
      e.printStackTrace();
     }
   }
}

```

实时效果反馈

1.FileInputStream是什么流？

A 文件字节输入流；

B 文件字符输入流；

C 文件随机访问流；

D 文件对象输入流；

2.FileOutputStream是什么流？

A 文件字节输出流；

B 文件字符输出流；

C 文件随机访问流；

D 文件对象输出流；

答案

1=>A 2=>A

通过字节缓冲区提高读写效率

有了缓冲区读写就是快！

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文件时的缓冲区定义。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。

```java
public class TestFileByteBuffer{
  public static void main(String[] args) {
    long time1 = System.currentTimeMillis();
    copyFile("d:/1.jpg", "d:/2.jpg");
    long time2 = System.currentTimeMillis();
    System.out.println(time2 - time1);
   }


  /**
   *
   * @param src 源文件
   * @param desc 目标文件
   */
  public static void copyFile(String src,String desc){
        //“后开的先关闭！”按照他们被创建顺序的逆序来关闭
    try(FileInputStream fis = new FileInputStream(src);
      FileOutputStream fos = new FileOutputStream(desc)){
      //创建一个缓冲区，提高读写效率
      byte[] buffer = new byte[1024];


      int temp = 0;
      while ((temp = fis.read(buffer)) != -1){
      //将缓存数组中的数据写入文件中，注意：写入的是读取的真实长度;
      fos.write(buffer,0,temp);


      }
      //将数据从内存中写入到磁盘中。
```

```
        fos.flush();
      }
      catch (IOException e) {
        e.printStackTrace();
       }
    }
}

```

注意 在使用字节缓冲区时，我们需要注意：

- 为了减少对硬盘的读写次数，提高效率，通常设置缓存数组。相应地，读取时使用的方法为：read(byte[] b)；写入时的方法为：write(byte[ ] b, int off, int length)
- 程序中如果遇到多个流，每个流都要单独关闭，防止其中一个流出现异常后导致其他流无法关闭的情况。

缓冲字节流

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）。

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。

使用缓冲流实现文件的高效率复制

下面我们通过两种方式（普通文件字节流与缓冲文件字节流）实现一个文件的复制，来体会一下缓冲流的好处。

```
public class TestFileBufferStream {
  public static void main(String[] args) {
    long time1 = System.currentTimeMillis();
    copyFile("d:/1.jpg","d:/2.jpg");
    long time2 = System.currentTimeMillis();
```

```
6       System.out.println(time2 - time1);
7     }
8
9
10    public static void copyFile(String source,String destination){
11        //实例化节点流
12      try(FileInputStream fis = new FileInputStream(source);
13        FileOutputStream fos = new FileOutputStream(destination);
14        //实例化处理流
15        BufferedInputStream bis = new BufferedInputStream(fis);
16        BufferedOutputStream bos = new BufferedOutputStream(fos)){
17
18
19        int temp = 0;
20        while ((temp = bis.read()) != -1){
21          bos.write(temp);
22         }
23        bos.flush();
24
25
26       }catch(IOException e){
27        e.printStackTrace();
28       }
29    }
30  }
31
```

> **注意**
>
> - 在关闭流时，应该先关闭最外层的包装流，即“后开的先关闭”。
> - 缓存区的大小默认是8192字节，也可以使用其它的构造方法自己指定大小。

实时效果反馈

1.缓冲字节流默认大小是多少字节？

A 1024字节；

B 2048字节；

C 4096字节；

D 8192字节；

答案

1=>D

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。

文件字符输入流

```
public class TestFileReader {
  public static void main(String[] args) {
    //创建文件字符输入流对象
    try(FileReader fr = new FileReader("d:/a.txt")){
      StringBuilder sb = new StringBuilder();
    //读取文件
      int temp = 0;
      while((temp = fr.read()) != -1){
        sb.append((char)temp);
      }
      System.out.println(sb);
     }catch (IOException e){
      e.printStackTrace();
     }
   }
}

```

文件字符输出流

```java
public class TestFileWriter {
  public static void main(String[] args) {
    //创建文件字符输出流对象
    try(FileWriter fw = new FileWriter("d:/aa.txt")){
      fw.write("您好尚学堂\r\n");
      fw.write("您好Old Lu\r\n");
      fw.flush();
     }catch (IOException e){
      e.printStackTrace();
     }
   }
}

```

实时效果反馈

1.FileReader是什么流？

A 文件字节输入流；

B 文件字符输入流；

C 文件随机访问流；

D 文件对象输入流；

2.FileWriter是什么流？

A 文件字节输出流；

B 文件字符输出流；

C 文件随机访问流；

D 文件对象输出流；

答案

1=>B 2=>B

缓冲字符流

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

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 TestBufferedReader {
  public static void main(String[] args) {
      //创建文件字符输入流对象
    try(FileReader fr = new FileReader("d:/aa.txt");
      //创建字符缓冲处理流。缓冲区默认大小为8192个字符。
      BufferedReader br = new BufferedReader(fr)){


      //操作流
      String temp = "";
      //readLine(): 读取一行文本。
      while((temp = br.readLine()) != null){
        System.out.println(temp);
       }


     }catch(IOException e){
      e.printStackTrace();
     }
   }
}

```

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 TestBufferedWriter {
  public static void main(String[] args) {
      //创建文件字符输出流对象
    try(FileWriter fw = new FileWriter("d:/sxt.txt");
      //创建字符输出缓冲流对象
      BufferedWriter bw = new BufferedWriter(fw)){
      //操作缓冲流
      bw.write("您好尚学堂");
      bw.write("您好Oldlu");
      //换行
      bw.newLine();
      bw.write("何以解忧");
      bw.newLine();
      bw.write("唯有尚学堂");
      bw.flush();
     }catch (IOException e){
      e.printStackTrace();
     }
   }
}

```

> 注意
>
> - readLine()方法是BufferedReader的方法，可以对文本文件进行更加方便的读取操作。
> - newLine()方法BufferedWriter的方法，可以使用newLine()方法换行。

实时效果反馈

1.在BufferedReader对象中，能够读取一行的方法是？

A read()；

B line()；

C rd()；

D readLine()；

2.在BufferedWriter对象中，能够换行的方法是？

A line()；

B insertLine()；

C newLine()；

D nLine()；

答案

1=>D 2=>C

为文件中的内容添加行号

```java
public class TestLineNumber {
  public static void main(String[] args) {
      //创建字符输入缓冲流与文件字符输入流
    try(BufferedReader br = new BufferedReader(new FileReader("d:/sxt.txt"));
      //创建字符输出缓冲流与文件字符输出流
      BufferedWriter bw = new BufferedWriter(new FileWriter("d:/sxt2.txt"))){


      String temp ="";
      //定义序号变量
      int i = 1;
      while((temp = br.readLine()) != null){
        //将读取到的内容添加序号，并输出到指定文件中。
        bw.write(i+","+temp);
        //换行处理
        bw.newLine();
        //序号变量累加
        i++;
       }
      //刷新
      bw.flush();
    }catch(IOException e){
      e.printStackTrace();
    }
  }
}

```

转换流

InputStreamReader/OutputStreamWriter用来实现将字节流转化成字符流。

通过转换流解决乱码

ANSI(American National Standards Institute)美国国家标准协会

```
public class TestInputStreamReader {
  public static void main(String[] args) {
    //创建文件字节输入流对象
    try(FileInputStream fis = new FileInputStream("d:/sxt.txt");
      //创建转换流(字节到字符的转换)流对象，并在该对象中指定编码。
      InputStreamReader isr = new InputStreamReader(fis,"gbk")){
      StringBuilder sb = new StringBuilder();
      //操作流对象
      int temp = 0;
      while((temp = isr.read()) != -1){
        sb.append((char) temp);
       }
      System.out.println(sb);
     }catch(IOException e){
      e.printStackTrace();
     }
   }
}

```

实时效果反馈

1.InputStreamReader流对象的作用是？

A 提供字符流到字节流之间的转换；

B 提供字符流到对象流之间的转换；

C 提供字节流到对象流之间的转换；

D 提供字节流到字符流之间的转换；

2.OutputStreamWriter流对象的作用是？

A 提供字符流到字节流之间的转换；

B 提供字符流到对象流之间的转换；

C 提供字节流到对象流之间的转换；

D 提供字节流到字符流之间的转换；

答案

1=>D 2=>A

通过字节流读取文本文件并添加行号

```
public class TestLineNumber2 {
  public static void main(String[] args) {
      //创建字符输入缓冲流、输入字节到字符转换流、文件字节输入流对象
    try(BufferedReader br = new BufferedReader(new
InputStreamReader(new FileInputStream("d:/sxt.txt")));
      //创建字符输出缓冲流、输出字符到字节转换流、文件字节输出流对象
      BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream("d:/sxt4.txt")))){


      //操作流
      String temp = "";
      //序号变量
      int i = 1;
      //按照行读取
      while((temp = br.readLine()) != null){
        bw.write(i+","+temp);
        //换行
        bw.newLine();
        //序号累加
        i++;
       }
      //刷新
      bw.flush();
    }catch(IOException e){
      e.printStackTrace();
    }
  }
}

```

通过转换流实现键盘输入屏幕输出

System.in是字节流对象，代表键盘的输入。

System.out是字节流对象，代表输出到屏幕。

```
public class TestKeyboardInput {
  public static void main(String[] args) {
      //创建键盘输入相关流对象
    try(BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      //创建向屏幕输出相关流对象
      BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(System.out))){


      while(true){
        bw.write("请输入: ");
        bw.flush();
        //获取键盘输入的字符串
        String input = br.readLine();
        //判断输入的内容是否含有退出关键字。
        if("exit".equals(input) || "quit".equals(input)){
          bw.write("Bye Bye !");
          bw.flush();
          break;
         }
        //将读取到键盘输入的字符串，输出到屏幕。
        bw.write("您输入的是: "+input);
        bw.newLine();
        bw.flush();
       }
     }catch(IOException e){
      e.printStackTrace();
     }
   }
}

```

通过转换流实现键盘输入屏幕输出

```
import java.io.*;


public class TestConvertStream {
    public static void main(String[] args) {
        // 创建字符输入和输出流:使用转换流将字节流转换成字符流
```

```
        BufferedReader br = null;
        BufferedWriter bw = null;
        try {
            br = new BufferedReader(new InputStreamReader(System.in));
            bw = new BufferedWriter(new OutputStreamWriter(System.out));
            // 使用字符输入和输出流
            String str = br.readLine();
            // 一直读取，直到用户输入了exit为止
            while (!"exit".equals(str)) {
                // 写到控制台
                bw.write(str);
                bw.newLine();// 写一行后换行
                bw.flush();// 手动刷新
                // 再读一行
                str = br.readLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 关闭字符输入和输出流
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (bw != null) {
                try {
                    bw.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

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 TestPrintWriter {
  public static void main(String[] args) {
    //创建字符输出流对象
    try(PrintWriter pw = new PrintWriter("d:/sxt5.txt")){
      //调用不带换行方法完成内容的输出
      pw.print("abc");
      pw.print("def");
      //调用带有自动换行方法完成内容的输出
      pw.println("Oldlu");
      pw.println("sxt");
      pw.flush();
     }catch(IOException e){
      e.printStackTrace();
     }
  }
}

```

实时效果反馈

1.PrintWriter是什么流对象？

A 字节输入流对象；

B 字节输出流对象；

C 字符输入流对象；

D 字符输出流对象；

答案

1=>D

通过字符输出流添加行号

```java
public class TestLineNumber3 {
  public static void main(String[] args) {
    //创建字符输入缓冲流对象与文件字符输入流对象
    try(BufferedReader br = new BufferedReader(new FileReader("d:/sxt.txt"));
      //创建字符输出流对象
      PrintWriter pw = new PrintWriter("d:/sxt6.txt")){
      //操作流
      String temp = "";
      //定义序号变量
      int i = 1;
      while((temp = br.readLine()) != null){
        pw.println(i+","+temp);
        //序号累加
        i++;
       }
      //刷新
      pw.flush();
    }catch(IOException e){
      e.printStackTrace();
    }
  }
}
```

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。

```
public class TestDataStream {
  public static void main(String[] args) {
    //创建数据输出流对象与文件字节输出流对象
    try(DataOutputStream dos = new DataOutputStream(new
FileOutputStream("d:/data"));
      //创建数据输入流对象与文件字节输入流对象
      DataInputStream dis = new DataInputStream(new
FileInputStream("d:/data"))){
      //将如下数据写入到文件中
      dos.writeChar('a');
      dos.writeInt(10);
      dos.writeDouble(Math.random());
      dos.writeBoolean(true);
      dos.writeUTF("北京尚学堂");
      //手动刷新缓冲区：将流中数据写入到文件中
      dos.flush();
      //直接读取数据：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(IOException e){
     e.printStackTrace();
    }
  }
}

```

Oldlu提示：

使用数据流时，读取的顺序一定要与写入的顺序一致，否则不能正确读取数据。

实时效果反馈

1.使用数据流时对于读写顺序的要求是？

A 没有任何顺序要求；

B 读取的顺序要与写入的顺序一致；

答案

1=>B

对象流

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，并不能读取对象（字符串除外），如果要对某个对象进行读写操作，我们需要学习一对新的处理流：ObjectInputStream/ObjectOutputStream。

处理基本数据类型数据

ObjectInputStream/ObjectOutputStream处理基本数据类型。

```
ipublic class TestObjectStreamBasicType {
  public static void main(String[] args) {
      //创建对象输出流对象与文件字节输出流对象
    try(ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("d:/data2"));
      //创建对象输入流对象与文件字节输入流对象
      ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("d:/data2"))){


      //将如下数据写入到文件中
      oos.writeInt(10);
      oos.writeDouble(Math.random());
      oos.writeChar('a');
      oos.writeBoolean(true);
      oos.writeUTF("你好Oldlu");
      oos.flush();


      //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());
```

```
23          System.out.println("String: "+ois.readUTF());
24        }catch(IOException e){
25          e.printStackTrace();
26        }
27      }
28  }
29
```

> 注意
>
> - 对象流不仅可以读写对象，还可以读写基本数据类型。
> - 读写基本数据类型时，读取的顺序一定要与写入的顺序一致，否则不能正确读取数据。

实时效果反馈

1.ObjectInputStream/ObjectOutputStream需要的节点流类型是？

A 字节流类型；

B 字符流类型；

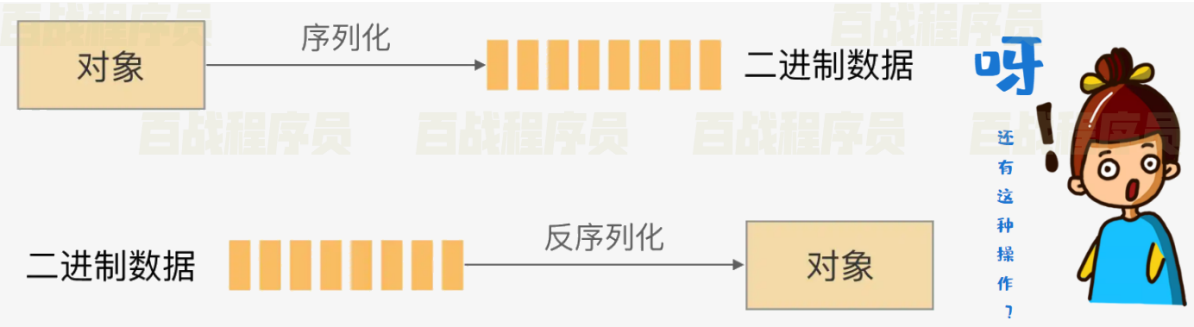
C 字节或字符流类型；

D 数据流类型；

答案

1=>A

Java对象的序列化和反序列化



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。

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。

实时效果反馈

1.Java对象的序列化是指？

A 将Java对象转换为字节序列；

B 将Java对象转换为字符序列；

C 将字节序列转换为Java对象；

D 将字符序列转换为Java对象；

2.Java对象的反序列化是指？

A 将Java对象转换为字节序列；

B 将Java对象转换为字符序列；

C 将字节序列转换为Java对象；

D 将字符序列转换为Java对象；

3.Java对象序列化时必须要实现哪个接口？

A Collection接口；

B Comparable接口；

C Comparator接口；

D Serializable接口；

答案

1=>A 2=>C 3=>D

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。

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;
   }


```

```
43    public void setUserage(String userage) {
44      this.userage = userage;
45     }
46
47
48    @Override
49    public String toString() {
50      return "Users{" +
51          "userid=" + userid +
52          ", username='" + username + '\'' +
53          ", userage='" + userage + '\'' +
54          '}';
55     }
56
```

序列化对象

```
1  public class TestObjectOutputStream {
2    public static void main(String[] args) {
3      //创建对象输出字节流与文件输出字节流对象
4      try(ObjectOutputStream oos = new ObjectOutputStream(new
   FileOutputStream("d:/data3"))){
5        //创建Users对象
6        Users users = new Users(1,"Oldlu","18");
7        //将对象序列化到文件中
8        oos.writeObject(users);
9        //刷新
10       oos.flush();
11     }catch(IOException e){
12       e.printStackTrace();
13     }
14   }
15 }
16
```

将对象反序列化到内存中

```java
public class TestObjectInputStream {
  public static void main(String[] args) {
    //创建对象输入字节流与文件字节输入流对象
    try(ObjectInputStream ois = new ObjectInputStream(new FileInputStream("d:/data3"))){
      //将对象反序列化到内存中
      Users users = (Users) ois.readObject();
      System.out.println(users);
     }catch(Exception e){
      e.printStackTrace();
     }
   }
}

```

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。

```java
public class TestFile {
  public static void main(String[] args) {
    //创建字符缓冲流与文件字符输入流对象
    try(BufferedReader br = new BufferedReader(new FileReader(new File("d:/sxt.txt")));
      //创建字符输出流对象
      PrintWriter pw = new PrintWriter(new File("d:/sxt8.txt"))){


      //操作流
```

```
        String temp = "";
        int i=1;
        while((temp = br.readLine()) != null){
          pw.println(i+","+temp);
          i++;
         }
        pw.flush();
       }catch(IOException e){
        e.printStackTrace();
       }
     }
  }

```

装饰器模式构建IO流体系

装饰器模式简介

装饰器模式是GOF23种设计模式中较为常用的一种模式。它可以实现对原有类的包装和装饰，使新的类具有更强的功能。

装饰器模式

```
class Iphone {
    private String name;
    public Iphone(String name) {
        this.name = name;
    }
    public void show() {
        System.out.println("我是" + name + ",可以在屏幕上显示");
    }
}


class TouyingPhone {
```

```
    public Iphone phone;
    public TouyingPhone(Iphone p) {
        this.phone = p;
    }
    // 功能更强的方法
    public void show() {
        phone.show();
        System.out.println("还可以投影，在墙壁上显示");
    }
}


public class TestDecoration {
    public static void main(String[] args) {
        Iphone phone = new Iphone("iphone30");
        phone.show();
        System.out.println("================装饰后");
        TouyingPhone typhone = new TouyingPhone(phone);
        typhone.show();
    }
}

```

IO流体系中的装饰器模式

IO流体系中大量使用了装饰器模式，让流具有更强的功能、更强的灵活性。比如：

```
FileInputStream fis = new FileInputStream(src);
BufferedInputStream bis = new BufferedInputStream(fis);

```

显然BufferedInputStream装饰了原有的FileInputStream，让普通的FileInputStream也具备了缓存功能，提高了效率。

实时效果反馈

1.在JavaIO的处理流中，使用了什么设计模式来处理节点流？

A 单例模式；

B 代理模式；

C 装饰器模式；

D 适配器模式；

答案

1=>C

Apache commons-io工具包的使用

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。

commons-io工具包

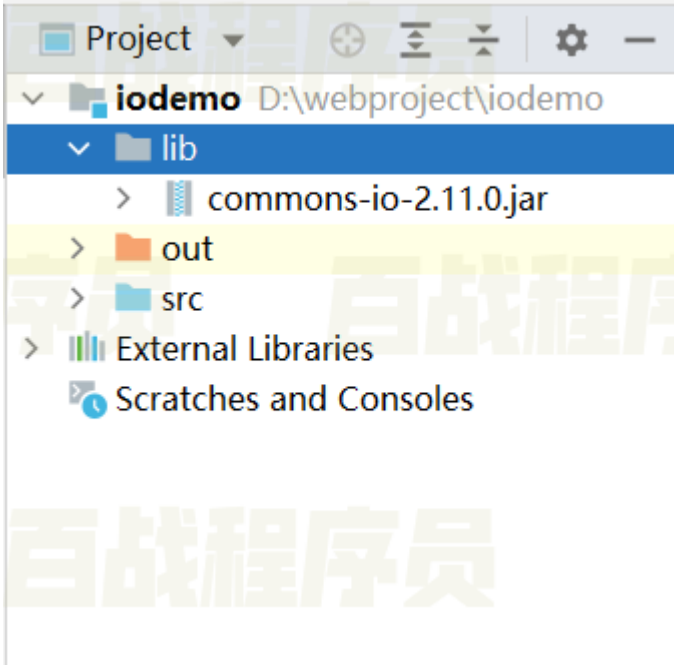
Apache的commons-io工具包中提供了IOUtils/FileUtils，为我们提供了更加简单、功能更加强大的文件操作和IO流操作功能。非常值得大家学习和使用。

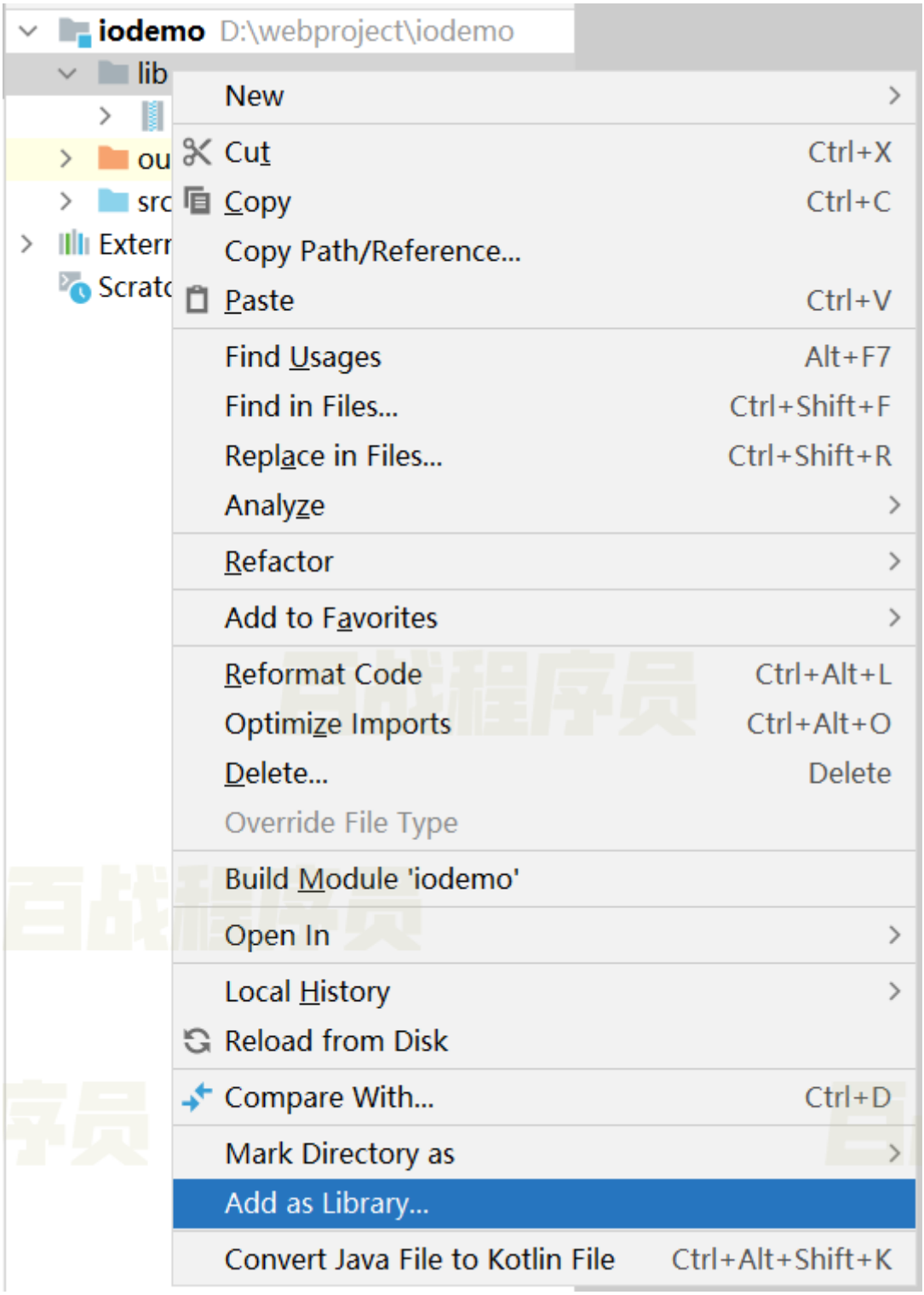
下载与添加commons-io包

下载地址

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

添加jar包


Project
iodemo D:\webproject\iodemo
lib
commons-io-2.11.0.jar
out
src
External Libraries
Scratches and Consoles



实时效果反馈

1.commons-io工具包的作用是？

A 处理IO相关操作；

B 处理容器相关操作；

C 处理字符串相关操作；

D 处理日期相关操作；

答案

1=>A

FileUtils类中常用方法的介绍

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：

方法名	使用说明
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

读取文件内容，并输出到控制台上（只需一行代码！）

```java
import java.io.File;
import org.apache.commons.io.FileUtils;
public class TestUtils1 {
    public static void main(String[] args) throws Exception {
        String content = FileUtils.readFileToString(new File("d:/a.txt"), "gbk");
        System.out.println(content);
    }
}

```

使用FileUtils工具类实现目录拷贝

我们可以使用FileUtils完成目录拷贝，在拷贝过程中可以通过文件过滤器(FileFilter)选择拷贝内容。

```java
import java.io.File;
import java.io.FileFilter;
import org.apache.commons.io.FileUtils;


public class TestFileUtilsDemo2 {
    public static void main(String[] args) throws Exception {
        FileUtils.copyDirectory(new File("d:/aaa"), new File("d:/bbb"), new FileFilter() {
            @Override
            public boolean accept(File pathname) {
                // 使用FileFilter过滤目录和以html结尾的文件
                if (pathname.isDirectory() || pathname.getName().endsWith("html")) {
                    return true;
                } else {
                    return false;
                }
            }
        });
    }
}

```

IOUtils的妙用

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：

方法名	使用说明
buffer	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	关闭流
contentEquals	比较两个流中的内容是否一致
copy	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	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	返回可以迭代每一行内容的迭代器
read	将输入流中的部分内容读入到字节数组中
readFully	将输入流中的所有内容读入到字节数组中
readLine	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	向流里面写入内容
writeLine	向流里面写入一行内容

我们没有必要对每个方法做测试，只是演示一下读入d:/sxt.txt文件内容到程序中，并转成String对象，打印出来。

IOUtils的使用

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

```

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
  - BufferedInputStream
  - DataInputStream
  - ObjectInputStream
- OutputStream的实现类：
  - FileOutputStream
  - BufferedOutputStream
  - DataOutputStream
  - ObjectOutputStream
- Reader的实现类
  - FileReader
  - BufferedReader
  - InputStreamReader
- Writer的实现类
  - FileWriter
  - BufferedWriter

- OutputStreamWriter
- PrintWriter

- 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。