IO - Day02

今日学习内容：

- 文件拷贝操作
- 缓冲流的基本操作
- 缓冲流的优势
- 对象序列化操作
- 打印流
- 标准IO

今日学习目标：

- 重点熟练使用字符流完成文件拷贝
- 重点熟练使用缓冲流完成文件拷贝
- 了解缓冲流的优势
- 了解对象序列化的整个过程
- 了解标准IO
- 了解打印流

1、文件拷贝操作

需求：把copy_before.txt文件中的数据拷贝到copy_after.txt文件中

```
private static void copy() throws Exception {
    //1):创建源或者目标对象
    File src = new File("file/copy_before.txt");
    File dest = new File("file/copy_after.txt");

    //2):创建IO流对象
    FileReader reader = new FileReader(src);
    FileWriter writer = new FileWriter(dest);

    //3):具体的IO操作
    int len = -1;//记录以及读取了多个字符
    char[] cbuf = new char[1024];//每次可以读取1024个字符
    while( (len = reader.read(cbuf)) != -1 ) {
        //边读边写
        out.write(buff, 0, len);
    }

    //4):关闭资源(勿忘)
    writer.close();
    reader.close();
}
```

如何，正确处理异常：

```
private static void copy2() {
    //1):创建源或者目标对象
    File src = new File("file/copy_before.txt");
```

```
    File dest = new File("file/copy_after.txt");

    //把需要关闭的资源，声明在try之外
    FileReader in = null;
    FileWriter out = null;
    try {
        //可能出现异常的代码
        //2):创建IO流对象
        in = new FileReader(src);
        out = new FileWriter(dest);

        //3):具体的IO操作
        int len = -1;//记录以及读取了多个字符
        char[] cbuf = new char[1024];//每次可以读取1024个字符
        while( (len = reader.read(cbuf)) != -1 ) {
            //边读边写
            out.write(buff, 0, len);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        //4):关闭资源(勿忘)
        try {
            if (out != null) {
                out.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

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

此时关闭资源的代码，又臭又长，在后续的学习中为了方便就直接使用throws抛出IO异常了，在实际开发中需要更高级处理方式。

2、缓冲（Buffer）流

节点流的功能都比较单一，性能较低。

处理流，也称之为**包装流**，相对于节点流更高级，这里存在一个设计模式——装饰设计模式，此时撇开不谈。

包装流如何区分？写代码的时候，发现创建流对象时，需要传递另一个流对象，类似：

```
new  流类B(  new 流类A(..) ) ;
```

那么流B就属于包装流，当然A可能属于节点流也可能属于包装流。

有了包装流之后，我们只关心包装流的操作即可，比如只需要关闭包装流即可，无需在关闭节点流。

非常重要的包装流——缓冲流，根据四大基流都有各自的包装流：

```
BufferedInputStream / BufferedOutputStream 字节缓存流
BufferedReader / BufferedWriter 字符缓存流
```

缓冲流内置了一个默认大小为8192个字节或字符的缓存区，缓冲区的作用用来减少磁盘的IO操作，拿字节缓冲流举例，比如一次性读取8192个字节到内存中，或者存满8192个字节再输出到磁盘中。

操作数据量比较大的流，都建议使用上对应的缓存流。

2.1、字节缓存流

需求：把郭德纲-报菜名.mp3文件中的数据拷贝到郭德纲-报菜名2.mp3文件中

```
private static void copy3() throws Exception {

        File srcFile = new File("郭德纲-报菜名.mp3");
        File toFile = new File("郭德纲-报菜名2.mp3");

        // 文件输入流管道
        FileInputStream in = new FileInputStream(srcFile);
        BufferedInputStream bis = new BufferedInputStream(in,8192);

        // 文件输出流管道
        FileOutputStream out = new FileOutputStream(toFile);
        BufferedOutputStream bos = new BufferedOutputStream(out,8192);

        byte[] buf = new byte[1024];     // 每次可以读取1024个字节
        int len = -1;                    // 记录以及读取了多个字节
        while ( (len = bis.read(buf)) != -1 ) {
            // 利用缓冲流向输出流中写出去时，不会立即写到文件中，直到缓冲流中缓冲区达到8192字节后，才会自动写入。
            bos.write(buf,0,len);
        }

        bos.close();
        bis.close();
}
```

2.2、字符缓存流

BufferedReader 继承于Reader，实现**对文本型文件进行高效(一次读取一行)的读取。**在BufferedReader 维护了一个缓冲区，一次可以存储文件中的一个行。提供了读取更强大的方法：

- readLine() : 一次读取一行，读到末尾返回null

BufferedWriter 继承Writer,实现对文本型文件进行高效的写入，提供了特有方法

- newLine() : 写入一个换行。

需求: 按行读取一个文本文件

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

```
        // step 1: 确定数据源
        File file = new java.io.File("G:\\test\\g.txt");

        // step 2 建立管道
        FileReader in = new FileReader(file);
        BufferedReader reader = new BufferedReader(in);

        // step 3 操作IO
        // 一次读取一行
        /*
        String line = null;
        line = reader.readLine();
        line = reader.readLine();
        line = reader.readLine();
        line = reader.readLine();

        line = reader.readLine(); // 如果没有可读的行，返回null
        System.out.println(line);
        */

        // 循环读取
        String line;
        while ((line = br.readLine()) != null) {
          System.out.println(line);
        }

        // step 4: 关闭流通道
        reader.close();
    }
}
```

需求2：把程序中的一首诗按行写入到f.txt文件中

```
public class Test02 {
    public static void main(String[] args) throws IOException {
       // step 1: 确定目的地
        File file = new File("G:\\test\\f.txt");
        // step 2: 建立管道
        FileWriter writer = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter(writer);

        // step 3: 写入一个字符串后再写入一个换行
        bw.write("床前明月光");
        bw.newLine();
        bw.write("疑似地上霜");
        bw.newLine();
        bw.write("举头望明月");
        bw.newLine();
        bw.write("低头思故乡");
        bw.newLine();

        // step 4: 关闭通道
        bw.close();
    }
}
```

3、对象序列化（了解）

序列化：指把Java堆内存中的对象数据，通过某种方式把对象数据存储到磁盘文件中或者传递给给网络上传输。序列化在分布式系统在应用非常广泛。

反序列化：把磁盘文件中的对象的数据或者把网络节点上的对象数据恢复成Java对象的过程。

需要做序列化的类必须实现序列化接口：java.io.Serializable（这是标志接口[没有抽象方法]）

可以通过IO中的**对象流**来做序列化和反序列化操作。

- ObjectOutputStream：通过writeObject方法做序列化操作的
- ObjectInputStream：通过readObject方法做反序列化操作的

如果字段使用transient 修饰则不会被序列化。

```
public class User implements Serializable {
    private String name;
    private transient String password;
    private int age;

    public User(String name, String password, int age) {
        this.name = name;
        this.password = password;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public String getPassword() {
        return password;
    }
    public int getAge() {
        return age;
    }
    public String toString() {
        return "User [name=" + name + ", password=" + password + ", age=" + age
+ "]";
    }
}
```

测试代码

```
public class ObjectStreamDemo {
    public static void main(String[] args) throws Exception {
        String file = "file/obj.txt";
        ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(file));
        User u = new User("Will", "1111", 17);
        out.writeObject(u);
        out.close();

        //-------------------------------------
        ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
```

```
        Object obj = in.readObject();
        in.close();
        System.out.println(obj);
    }
}
```

obj.txt文件

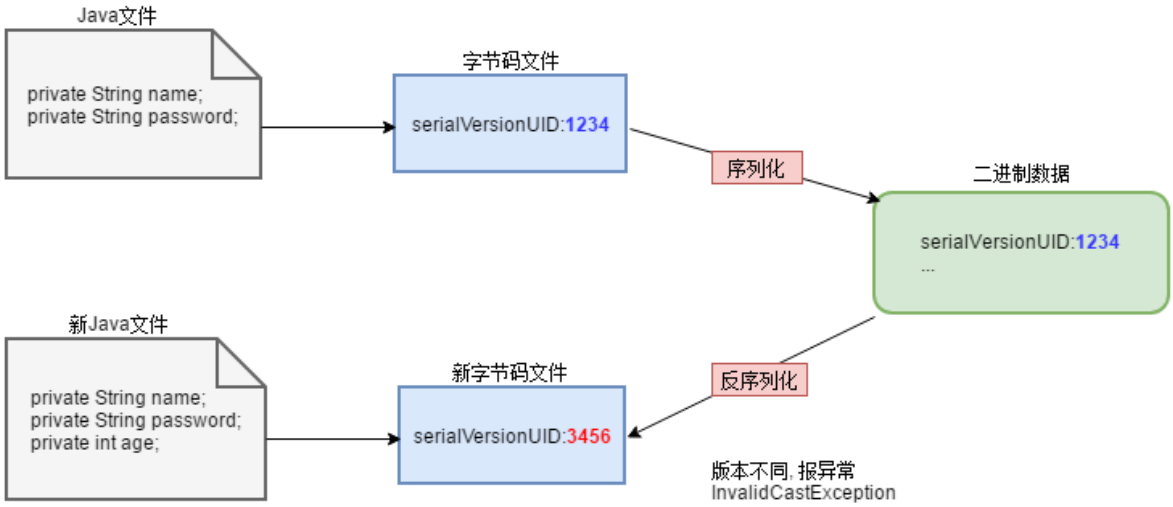
obj.txt

```
1 ??srio.file.Usero+?L?IageLnametLjava/lang/String;xpt?Will
```

序列化的版本问题

当类实现Serializable接口后，在编译的时候就会根据字段生成一个缺省的serialVersionUID值，并在序列化操作时，写到序列化数据文件中。

但随着项目的升级系统的class文件也会升级(增加一个字段/删除一个字段)，此时再重新编译，对象的serialVersionUID值又会改变。那么在反序列化时，JVM会把对象数据数据中的serialVersionUID与本地字节码中的serialVersionUID进行比较，如果值不相同（意味着类的版本不同），那么报异常InvalidClassException，即：类版本不对应，不能进行反序列化。如果版本号相同，则可以进行反序列化。



为了避免代码版本升级而造成反序列化因版本不兼容而失败的问题，在开发中我们可以故意在类中提供一个固定的serialVersionUID值。

```
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
     // TODO
}
```

4、查漏补缺（了解）

4.1、标准IO (了解)

标准输入流：通过键盘录入数据给程序。键盘 ---> 程序

标准输出流：程序数据显示在屏幕上。程序 ---> 显示器

在System类中有两个常量in和out分别就表示了标准流:

```
InputStream in  = System.in;    // 标准输入流是字节输入流
PrintStream out = System.out;   // 标准输出流是字节输出流
```

需求:控制台输入一个字符，显示器原样输出

```
public class Test01InOut {
    public static void main(String[] args) throws IOException {

        // 需求:控制台输入一个字符，显示器原样输出
        // 确定数据源 => 键盘 => 并创建管道
        InputStream in = System.in;
        OutputStream out = System.out;

        // 读进来
        int b = in.read();
        // 写出去
        out.write(97);
        out.flush();

        out.close(); // 标准输入输出流不需要关闭

        // 读取多个字节
        /*
        byte[] buf = new byte[10];
        int len = in.read(buf);
        System.out.println(Arrays.toString(buf));
        String str = new String(buf,0,3,"UTF-8");
        System.out.println("str = " + str);
        */
    }
}
```

4.2、打印流 (了解)

打印流是一种特殊是字节输出流，可以**输出任意类型的数据**，比一般的输出流更好用。可以作为处理流包装一个平台的节点流使用，平时我们使用的System.out.println其实就是使用的打印流。

- PrintStream ：字节打印流
- PrintWriter ：字符打印流

打印流中的方法：

- 提供了print方法：打印不换行
- 提供了println方法：先打印，再换行

```
private static void test5() throws Exception {
    // 需求:使用标准输出流在显示器上打印数字的97
    PrintStream out = System.out;
    /*
    out.write(97);  // 输出a
    out.flush();
    */

    // PrintStream实际上可以看成包装流，包装了字节流，提供更强大的功能：直接打印任意类型的数据。
    out.println("will");
    out.println(97);
    out.println("众里寻他千百度，蓦然回首，那人却在，灯火阑珊处。");
}
```

IO流小结

在开发中使用比较多的还是字节和字符流的读写操作，务必要非常熟练，再体会一下六字箴言（读进来，写出去），到底有何深意。

综合练习题：做一个统计代码行数的程序，扫描一个目录能统计出该目录中包括所有子目录中所有Java文件的行数，不统计空行。

```
private static void doWork(File dir) {
    // TODO
}
```

- 到这，Java就结束了吗?
- 不，往往故事的结束是为了新的开始，做完这道题目，让我们一起来开启大神之门。