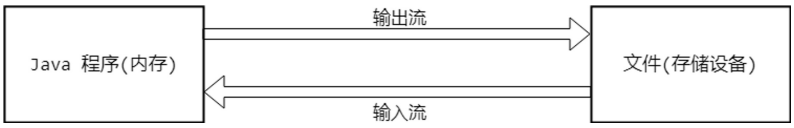
day09_字节流、字符流

java.io 包的 File 类用于目录和文件的创建、删除、遍历等操作，但不能用于文件的读写。Java 程序对于文件数据的读写是以 IO 流的形式来进行的：当 Java 程序与硬盘上的文件进行数据传输时，相当于在程序和文件之间建立了一个管道，在管道内进行二者之间的数据传输，这种数据传输可以看做是一种数据的流动，即 IO 流。以 Java 程序为基准，分为输入流(input)、输出流(output)，即流出 Java 程序的是输出流，流向 Java 程序的是输入流。



其实 IO 流不仅仅限于对文件的读写操作，在 Java 程序中，对于所有外部数据的输入、输出操作都是以 IO 流的形式来进行的，而这个外部数据的来源可以是键盘、外部设备上的文件、网络资源等等。所以 IO 流是实现 Java 程序进行数据输入、输出的基础，所有数据的输入、输出都是依靠 IO 流来完成的。

根据数据传输的基本单位，IO 流可以分为以下两种：

- 字节流：以字节为数据传输的基本单位，读写数据的流。即一次读入、读出是8个 bit 位。
- 字符流：以字符为数据传输的基本单位，读写数据的流。即一次读入、读出是16个 bit 位。

一.字节流

任何类型的数据(文本、图片、视频等)都是以二进制数字的形式存储的，并且使用一个一个的字节(8 bit 位)为基本存储单位。那么使用字节流传输数据时亦是如此，使用字节流来传输数据，是将数据底层存储的二进制数字以字节为基本单位，传入到字节流中进行传输。所以字节流可以传输任意类型的数据，但是在操作流的时候，我们要时刻明确，无论使用什么样的 IO 流对象，底层传输的始终是二进制数据。

1.字节输出流 OutputStream

java.io.OutputStream 抽象类是表示字节输出流的所有类的超类，它定义了字节输出流的基本共性功能方法，这些共性的方法会被其子类覆盖重写。字节输出流可以把 "写入到字节输出流中的数据" 输出到目的地。

- `public void close()`：关闭此输出流并释放与此流相关联的任何系统资源(当完成 IO 流的操作时，必须调用此方法释放系统资源)
- `public void flush()`：刷新此输出流并强制任何缓冲的输出字节被写出
- `public void write(int b)`：将 "int 类型整数" 以字节为基本单位依次写入到字节输出流中，再由字节输出流写入到目的地
- `public void write(byte[] b)`：将 "数组中的 byte 类型整数" 以字节为基本单位依次写入到字节输出流中，再由字节输出流写入到目的地
- `public void write(byte[] b, int off, int len)`：从字节数组的 off 索引开始，依次写入 len 个数组元素到字节输出流中

(1).FileOutputStream 类

(2).数据的追加续写、写出换行

2.字节输入流 InputStream

java.io.InputStream 抽象类是表示字节输入流的所有类的超类，它定义了字节输入流的基本共性功能方法，这些共性的方法会被其子类覆盖重写。字节输入流可以把 "读取到字节输入流中的数据" 输入到程序中。

- `public void close()`：关闭此输入流并释放与此流相关联的任何系统资源(当完成 IO 流的操作时，必须调用此方法释放系统资源)
- `public int read()`：从输入流读取数据的下一个字节。返回读取到的字节对应的 int 类型整数，若没有下一个字节(数据末尾)，则返回 -1
- `public int read(byte[] b)`：从输入流中读取一定数量的字节，并将它们存储到缓冲区字节数组 b 中。返回读取到的有效字节个数，若没有下一个字节(数据末尾)，则返回 -1

(1).FileInputStream 类

(2).字节流练习：图片文件复制

二.字符流

使用字符流来传输数据时，是将数据底层存储的二进制数字以字符为基本单位，传入到字符流中进行传输。字符流可以读取、写入单个字符，只能用来处理文本文件，不能操作图片、视频等非文本文件。

1.字符输出流 Writer

java.io.Writer 抽象类是表示字符输出流的所有类的超类，它定义了字符输出流的基本共性功能方法，这些共性的方法会被其子类覆盖重写。字符输出流可以把 "写入到字符输出流中的数据" 输出到目的地。

- `public void close()`：先刷新此输出流、然后关闭此输出流并释放与此流相关联的任何系统资源
- `public void flush()`：刷新此输出流并强制任何缓冲区的输出字符被写出
- `public void write(int c)`：写入单个字符到此输出流的内存缓冲区中
- `public void write(char[] cbuf)`：写入字符数组到此输出流的内存缓冲区中
- `public void write(char[] cbuf, int off, int len)`：写入字符数组的一部分到此输出流的内存缓冲区中(off 为开始索引、len 为个数)
- `public void write(String str)`：写入字符串到此输出流的内存缓冲区中
- `public void write(String str, int off, int len)`：写入字符串的一部分到此输出流的内存缓冲区中(off 为开始索引、len 为个数)

字符输出流 Writer 类中所有写入数据的 write( ) 方法，都是把数据先写入到内存缓冲区(将字符数据转换成字节数据，缓存到底层的 bytebuffer[ ] 数组的过程)，而不是直接写入到目的地。当该缓冲区数组满了，才会将数据自动写入目的地。我们也可以调用 flush( ) 方法把内存缓冲区中的数据，手动刷新到目的地，然后再调用 close( ) 方法关闭此输出流。其实 close( ) 方法在关闭输出流之前，也会先刷新缓冲区中的数据到目的地，然后再关闭输出流释放系统资源，所以只需要使用 close( ) 方法即可。

- flush( ) 方法：刷新缓冲区数据到文件中，流对象可以继续使用。
- close( ) 方法：先刷新缓冲区数据到文件中，然后关闭流释放系统资源，流对象不可以再被使用了。

而字节输出流 OutputStream 类中所有写入数据的 write( ) 方法，会直接把数据写入到目的地，所以不需要使用 flush( ) 方法。

(1).FileWriter 类

(2).数据的追加续写、写出换行

2.字符输入流 Reader

java.io.Reader 抽象类是表示字符输入流的所有类的超类，它定义了字符输入流的基本共性功能方法，这些共性的方法会被其子类覆盖重写。字

符输入流可以把 "读取到字符输入流中的数据" 输入到程序中。

- `public void close()`：关闭此输入流并释放与此流相关联的任何系统资源
- `public int read()`：从输入流中读取下一个字符并返回。返回读取到的字符对应的 int 类型整数，若没有下一个字符(数据末尾)，则返回 -1
- `public int read(char[] cbuf)`：从输入流中读取一定数量的字符，并将它们存储到缓冲区字符数组 cbuf 中。返回读取到的有效字符个数，若没有下一个字符(数据末尾)，则返回 -1

(1).FileReader 类

三.IO 异常的处理

之前都是使用 throws 来处理 IO 流异常，一般我们会使用 try...catch 来处理 IO 流中的异常。在 JDK7 之前处理 IO 流异常的方式如下：

```
public static void main(String[] args) {
    try {
        FileWriter fw = new FileWriter( fileName: "day09_code\\a.txt");
        fw.write( c: 97);
        fw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

一旦构造方法、write( ) 方法中产生了异常，close( ) 方法就不会被执行，所以 close( ) 方法要放在 finally 代码块中，保证无论上述代码是否发生异常都会调用 close( ) 方法关闭流释放系统资源。如下：

```
public static void main(String[] args) {
    try {
        FileWriter fw = new FileWriter( fileName: "day09_code\\a.txt");
        fw.write( c: 97);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        fw.close();
    }
}
```

但是流对象 fw 是 try 代码块中的局部变量，在 finally 代码块中无法使用，所以流对象要定义在外部。如下：

```
public static void main(String[] args) {
    FileWriter fw = null;
    try {
        fw = new FileWriter( fileName: "w:\\day09_code\\a.txt");
        fw.write( c: 97);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        // 如果fw不是null值，才需要关闭流释放系统资源
        if (fw != null) {
            try {
                fw.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

此时 close( ) 方法也会产生编译期异常，继续使用 try...catch 来处理。如下：

```
public static void main(String[] args) {
    FileWriter fw = null;
    try {
        new FileWriter( fileName: "day09_code\\a.txt");
        fw.write( c: 97);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

如果在 new 一个流对象时，给定的系统路径不存在(如：w:\day09_code\a.txt)，则会抛出 FileNotFoundException 异常，并且导致创建流对象失败。此时流对象 fw 的值仍为 null ，在执行 finally 代码块时，fw 调用 close( ) 方法会抛出 NullPointerException 异常，所以需要对程序进行进一步优化。

```
public static void main(String[] args) {
    FileWriter fw = null;
    try {
        fw = new FileWriter( fileName: "w:\\day09_code\\a.txt");
        fw.write( c: 97);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        // 如果fw不是null值，才需要关闭流释放系统资源
        if (fw != null) {
            try {
                fw.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

1.JDK7 之后对 IO 流异常的处理——JDK7 新特性

2.JDK9 之后对 IO 流异常的处理——JDK9 新特性

四.属性集

java.util.Properties 集合 extends Hashtable<K, V> 集合 implements Map<K, V>集合。Properties 集合类是 Hashtable 集合的子类，所以 Properties 集合类是一个可以存储 "键/值对" 型数据的双列 Map 集合，并且 Properties 集合类的两个泛型 Key(键)、Value(值) 默认都是 String 类型，即该集合存储的元素的键、值都是字符串。

1.Properties 集合类的方法

2.Properties 集合与 IO 流