IO概述

什么是IO?

i: input 输入

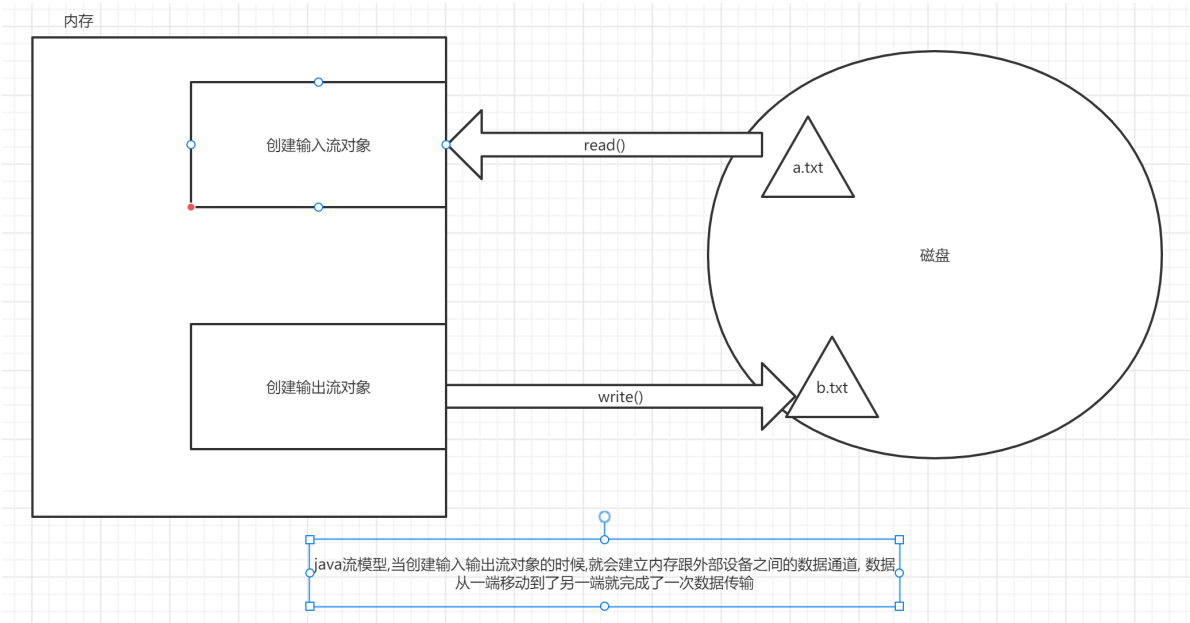
o: output 输出

为什么有IO?

我们之前学习File 不能对文件内容进行操作.想要对文件内容进行操作,就需要学习IO的知识.

java中如何实现IO功能

流模型



IO的分类

- 按照数据流向分(以内存为参照物)
  - 输入流 : 外部设备 -----> 内存
  - 输出流: 内存-----> 外部设备
- 按照数据类型分
  - 字节流: 逻辑单位是字节(1B= 0000 0000)
  - 字符流: 逻辑单位是字符(理解为一种文化符号,"abc", "你好", "の")

4个抽象基类

- 字节输出流: OutputStream
- 字节输入流: InputStream
- 字符输出流: Writer
- 字符输入流: Reader

由这4个抽象基类派生的子类,都是以其父类的名字作为后缀的

文件字节输出流 FileOutputStream

文件字节输入流 FileInputStream

文件字符输出流 FileWriter

FileReader

什么时候用什么流?

对于文本文件,一般使用字符流 .txt .java .cpp

其他文件,或者你不知道的文件格式,使用字节流(字节流是万能的) .mp3 .mp4 .jpg .png .exe .ppt .word

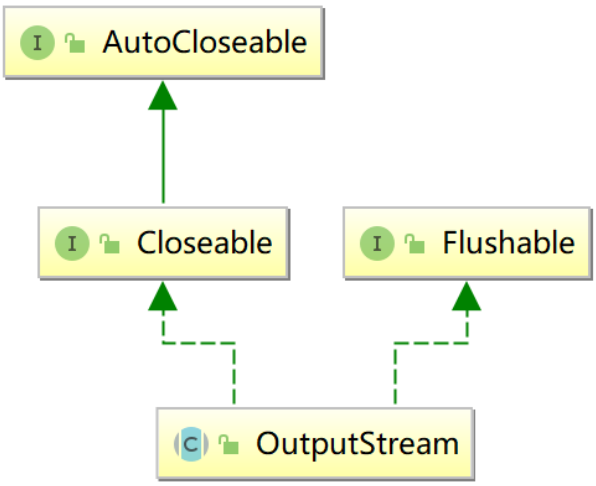
字节流

字节输出流

抽象基类OutputStream

此抽象类是表示输出字节流的所有类的超类

继承关系



成员方法

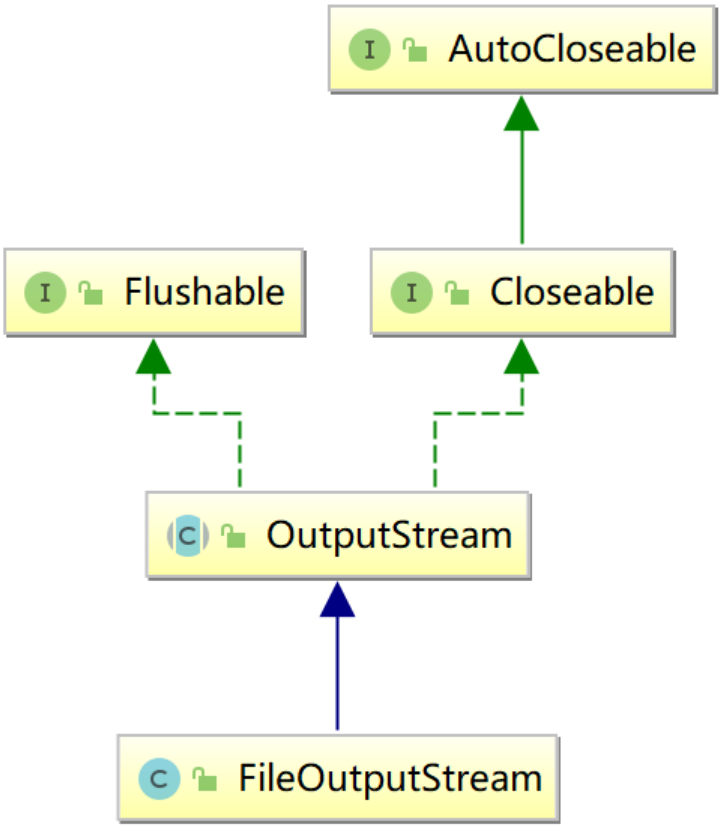
void	**close() 关闭此输出流并释放与此流有关的所有系统资源。**
void	flush() 刷新此输出流并强制写出所有缓冲的输出字节。
void	write(byte[] b) 将 b.length 个字节从指定的 byte 数组写入此输出流。
void	write(byte[] b, int off, int len) 将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流。
abstract void	write(int b) 将指定的字节写入此输出流。write 的常规协定是：向输出流写入一个字节。要写入的字节是参数 b 的八个低位。b 的 24 个高位将被忽略。

具体子类

FileOutputStream文件字节输出流

文件输出流是用于将数据写入 `File`

继承关系



构造方法

FileOutputStream(File file) 创建一个向指定 File 对象表示的文件中写入数据的文件输出流。
FileOutputStream(File file, boolean append) 创建一个向指定 File 对象表示的文件中写入数据的文件输出流。
FileOutputStream(String fileName) 创建一个向具有指定名称的文件中写入数据的输出文件流。
FileOutputStream(String name, boolean append) 创建一个向具有指定 name 的文件中写入数据的输出文件流。append - 如果为 true，则将字节写入文件末尾处，而不是写入文件开始处

成员方法

void	**close() 关闭此输出流并释放与此流有关的所有系统资源。**
void	flush() 刷新此输出流并强制写出所有缓冲的输出字节。
void	write(byte[] b) 将 b.length 个字节从指定的 byte 数组写入此输出流。
void	write(byte[] b, int off, int len) 将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流。
abstract void	write(int b) 将指定的字节写入此输出流。write 的常规协定是：向输出流写入一个字节。要写入的字节是参数 b 的八个低位。b 的 24 个高位将被忽略。

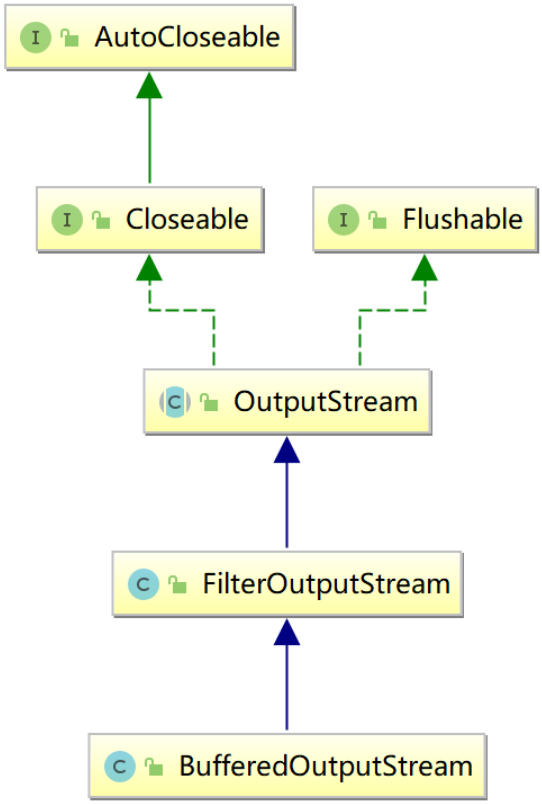
Demo:

```
        //创建输出流对象
        //FileOutputStream out = new FileOutputStream(new File("a.txt"));
        FileOutputStream out = new FileOutputStream("c.txt");
        //write写数据
        out.write(98);
        //释放资源 close
        out.close();
```

BufferedOutputStream缓冲字节输出流

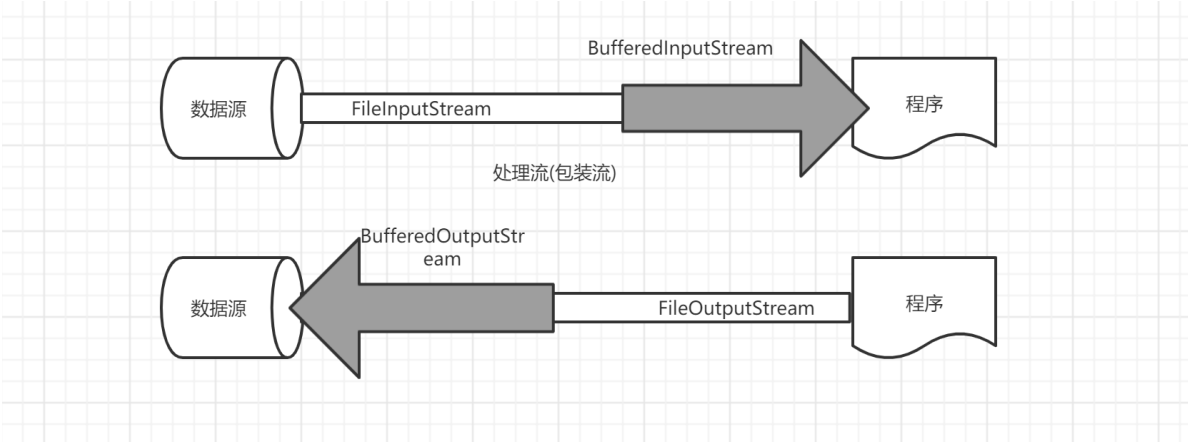
该类实现缓冲的输出流。通过设置这种输出流，应用程序就可以将各个字节写入底层输出流中，而不必针对每次字节写入调用底层系统。

继承关系



构造方法

BufferedOutputStream(OutputStream out) 创建一个新的缓冲输出流，以将数据写入指定的底层输出流。默认缓冲区大小是8KB
BufferedOutputStream(OutputStream out, int size) 创建一个新的缓冲输出流，以将具有指定缓冲区大小的数据写入指定的底层输出流。



成员方法

3个write方法

write(int b)

write(byte[] b)

write(byte[] b, int off ,int len)

Demo:

```java
package _18io.com.cskaoyan.bytestream._04buffer;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/23 11:13
 **/

/*
使用缓冲字节输出流写数据
 */
public class Demo {
    public static void main(String[] args) throws IOException {
        // 创建输出流对象
        BufferedOutputStream out =
                new BufferedOutputStream(new FileOutputStream("a.txt"));

        // write
        out.write("abc".getBytes());
        // flush
        out.flush();
        // close
        out.close();
    }
}
```

```java
/try/
public void close() throws IOException {
    try (OutputStream ostream = out) {
        flush();
    }
}
```

注意:

- 使用带缓冲的输出流的时候,不要忘记flush操作
- close会执行flush操作

注意事项

- 当创建输出流对象的时候,发生了什么?
    - 创建之前,jvm会向操作系统查看文件是否存在
    - 如果不存在,帮我们创建
    - 如果存在,覆盖重新写

- 怎么实现文件追加功能?
  - 利用带append的构造方法,设置为true
  - ```
    package _18io.com.cskaoyan.bytestream._01fileoutputstream;

    import java.io.FileOutputStream;
    import java.io.IOException;

    /**
     * @description:
     * @author: 景天
     * @date: 2022/7/22 16:45
     **/

    /*
    写数据的步骤
    1.创建输出流对象
    2.write写数据
    3.释放资源 close
     */
    public class Demo2 {
        public static void main(String[] args) throws IOException {
            //创建输出流对象
            FileOutputStream out = new FileOutputStream("c.txt",true);
            //write写数据
            out.write(97);
            out.write(98);
            out.write(99);
            //释放资源 close
            out.close();

        }
    }
    ```

- 怎么去换行功能?
  - 使用换行符
  - ```
    package _18io.com.cskaoyan.bytestream._01fileoutputstream;

    import java.io.FileOutputStream;
    import java.io.IOException;

    /**
     * @description:
     * @author: 景天
     * @date: 2022/7/22 16:45
     **/

    /*
    写数据的步骤
    1.创建输出流对象
    2.write写数据
    3.释放资源 close
    ```

```
实现换行
 */
public class Demo3 {
    public static void main(String[] args) throws IOException {
        //创建输出流对象
        FileOutputStream out = new FileOutputStream("a.txt");
        //write写数据
        // 使用换行符
        // "\r\n" "\r"  "\n"
        // 批量操作 使用字节数组
        String s = "abc";
        byte[] bytes = s.getBytes();
        out.write(bytes);
        out.write("\r\n".getBytes());
        out.write(bytes);
        out.write("\r".getBytes());
        out.write(bytes);
        out.write("\n".getBytes());
        out.write(bytes);

        // 系统默认换行符
        out.write(System.lineSeparator().getBytes());
        out.write(bytes);
        //释放资源 close
        out.close();

    }
}
```

- 关于异常处理
  - 传统的try-catch
    - ```
      package _18io.com.cskaoyan.bytestream._01fileoutputstream;

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

      /**
       * @description: try-catch异常处理
       * @author: 景天
       * @date: 2022/7/22 17:26
       **/

      public class Demo4 {
          public static void main(String[] args) {
              // 创建输出流对象
              FileOutputStream out = null;
              try {
                  //......
                  out = new FileOutputStream("a.txt");
                  // write
                  out.write("abc".getBytes());
      ```
      ```

```
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            // close
            try {
                // 做判断
                if (out != null) {
                    out.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
}
```

- try-with-resources
  - 语法

```
try(资源,实现了AutoCloseable接口的类){
// 可能出现异常的代码
// 当try中的代码执行完, 自动释放资源 执行close方法
}catch(){

}
```

  - ```
    package _18io.com.cskaoyan.bytestream._01fileoutputstream;

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

    /**
     * @description: try-with-resources
     * @author: 景天
     * @date: 2022/7/22 17:33
     **/

    public class Demo5 {
        public static void main(String[] args) {
            try(FileOutputStream out = new FileOutputStream("a.txt")) {
                // write
                out.write("abc".getBytes());
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
    ```
    ```

```
    }
}
```

- 验证自动释放
- ```
  package _18io.com.cskaoyan.bytestream._01fileoutputstream;

  /**
   * @description:
   * @author: 景天
   * @date: 2022/7/22 17:36
   **/

  public class Demo6 {
      public static void main(String[] args) {
          try (A a = new A()){
              // 调用func方法
              a.func();
          } catch (Exception e) {
              e.printStackTrace();
          }
      }
  }

  class A implements AutoCloseable{


      @Override
      public void close() throws Exception {
          System.out.println("close 方法执行了!");
      }

      public void func() {
          System.out.println("func 方法执行了!");
      }
  }
  ```

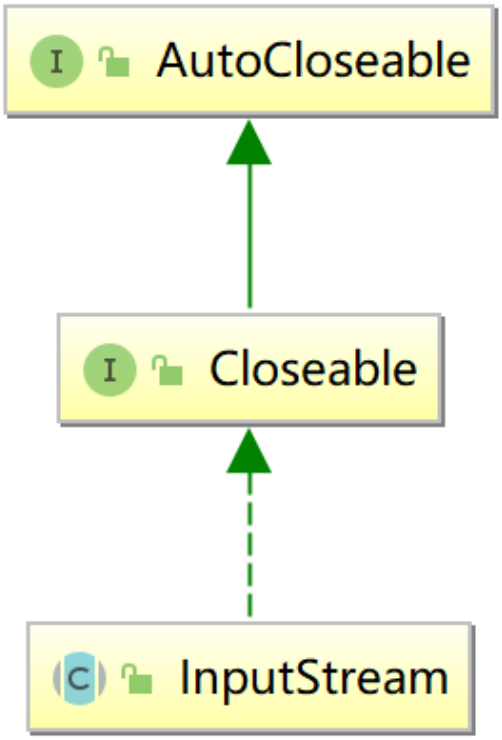
- 为什么要close?
  - 只要jvm用到了不属于jvm的资源, 不会进行垃圾回收. 必须显式的释放资源(close)

字节输入流
  ```

抽象基类InputStream

抽象类是表示字节输入流的所有类的超类。

继承关系



成员方法

abstract int	**read() 从输入流中读取数据的下一个字节。**
int	read(byte[] b) 从输入流中读取一定数量的字节，并将其存储在缓冲区数组 b 中。
int	read(byte[] b, int off, int len) 将输入流中最多 len 个数据字节读入 byte 数组。

具体子类

FileInputStream文件字节输入流

构造方法

FileInputStream(File file) 通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的 File 对象 file 指定。
FileInputStream(String fileName) 通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的路径名 name 指定。

成员方法

abstract int	**read() 从输入流中读取数据的下一个字节。返回 0 到 255 范围内的 int 字节值。如果因为已经到达流末尾而没有可用的字节，则返回值 -1 返回值代表的含义是读取到的字节值 readData**
int	read(byte[] b) 从输入流中读取一定数量的字节，并将其存储在缓冲区数组 b 中。读入缓冲区的总字节数；如果因为已经到达流末尾而不再有数据可用，则返回 -1。 返回值代表的含义是读取到的字节个数 readCount
int	read(byte[] b, int off, int len) 将输入流中最多 len 个数据字节读入 byte 数组。

Demo

```
package _18io.com.cskaoyan.bytestream._02fileinputstream;

import java.io.FileInputStream;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/23 9:45
 **/
/*
读取数据的步骤
1.创建输入流对象
2.read
3.close
 */
public class Demo {
    public static void main(String[] args) throws IOException {
        //1.创建输入流对象
        FileInputStream in = new
FileInputStream("D:\\workspace2\\java44th\\a.txt");
        //2.read
        // 单字节读取
        //readSingle(in);
        byte[] bytes = new byte[1024];
        //readMulti(in, bytes);

        // read(byte[] b, int off ,int len)
        int readCount = in.read(bytes, 2, 3);
```

```
        // 3.close
        in.close();
    }

    private static void readMulti(FileInputStream in, byte[] bytes) throws
IOException {
        // readCount 表示读取到的字节的个数
        int readCount = in.read(bytes);
        System.out.println(readCount);
        String s = new String(bytes);
        System.out.println(s);
    }

    private static void readSingle(FileInputStream in) throws IOException {
        // readData表示读取到的字节值
        int readData = in.read();
        System.out.println(((char) readData));
        int readData2 = in.read();
        System.out.println(((char) readData2));
        int readData3 = in.read();
        System.out.println(((char) readData3));
        int readData4 = in.read();
        System.out.println(readData4);
    }
}
```

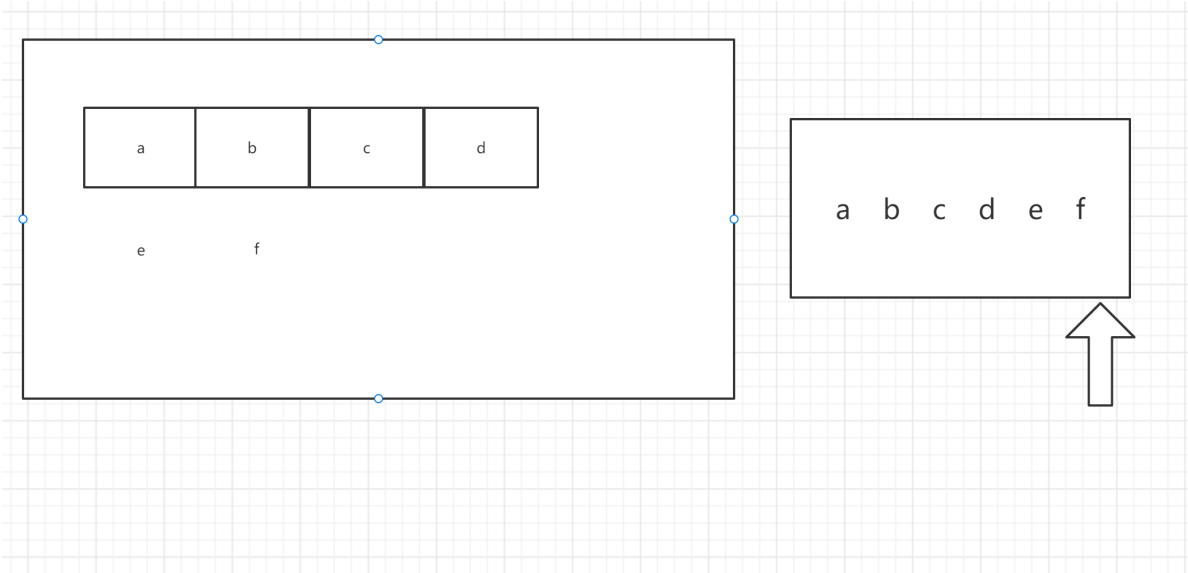
```
package _18io.com.cskaoyan.bytestream._02fileinputstream;

import java.io.FileInputStream;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/23 9:56
 **/

public class Demo2 {
    public static void main(String[] args) throws IOException {
        // 创建输入流对象
        FileInputStream in = new FileInputStream("a.txt");
        // read(byte [] b)
        byte[] bytes = new byte[4];
        int readCount = in.read(bytes);
        System.out.println(readCount);
        System.out.println(new String(bytes));
        int readCount2 = in.read(bytes);
        System.out.println(readCount2);
        System.out.println(new String(bytes,0,readCount2));
        // close
        in.close();
```

```
    }
}
```



循环读取数据

```
package _18io.com.cskaoyan.bytestream._02fileinputstream;

import java.io.FileInputStream;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/23 10:06
 **/
/*
循环读取
 */
public class Demo3 {
    public static void main(String[] args) throws IOException {
        // 创建输入流对象
        FileInputStream in = new FileInputStream("a.txt");
        // read
        // 方式一: 不用
        // readWhile1(in);

        // 方式二:
        // 单字节读取
        // 表示读取的字节值
        //readWhile2(in);

        // 批量读取
        byte[] bytes = new byte[1024];
        // 表示读取的字节的个数
```

```
        int readCount;
        while ((readCount = in.read(bytes)) != -1) {
            System.out.println(new String(bytes,0,readCount));
        }

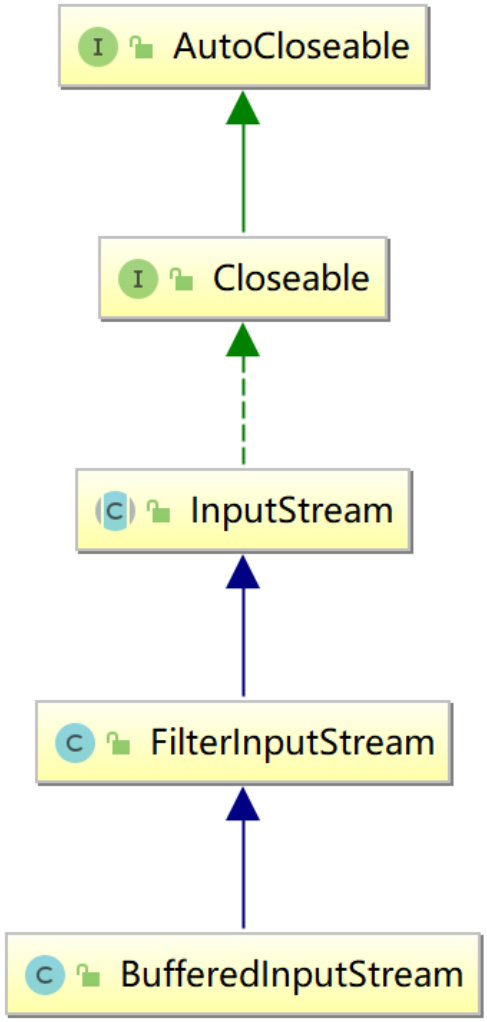
        // close
        in.close();
    }

    private static void readWhile2(FileInputStream in) throws IOException {
        int readData;
        while ((readData = in.read()) != -1) {
            System.out.println(readData);
        }
    }

    private static void readWhile1(FileInputStream in) throws IOException {
        while (true) {
            int readData = in.read();
            if (readData == -1) {
                break;
            }
            System.out.println(readData);
        }
    }
}
```

BufferedInputStream缓冲字节输入流

继承关系



构造方法

BufferedInputStream(InputStream in) 创建一个 BufferedInputStream 并保存其参数，即输入流 in，以便将来使用。默认是8KB
BufferedInputStream(InputStream in, int size) 创建具有指定缓冲区大小的 BufferedInputStream 并保存其参数，即输入流 in，以便将来使用。缓冲区大小就是size

成员方法

3个read

read() 读取单个字节

read(byte[] b) 读取数据,填充到字节数组

read(byte[] b,int off ,int len) 读取数据,填充到字节数组

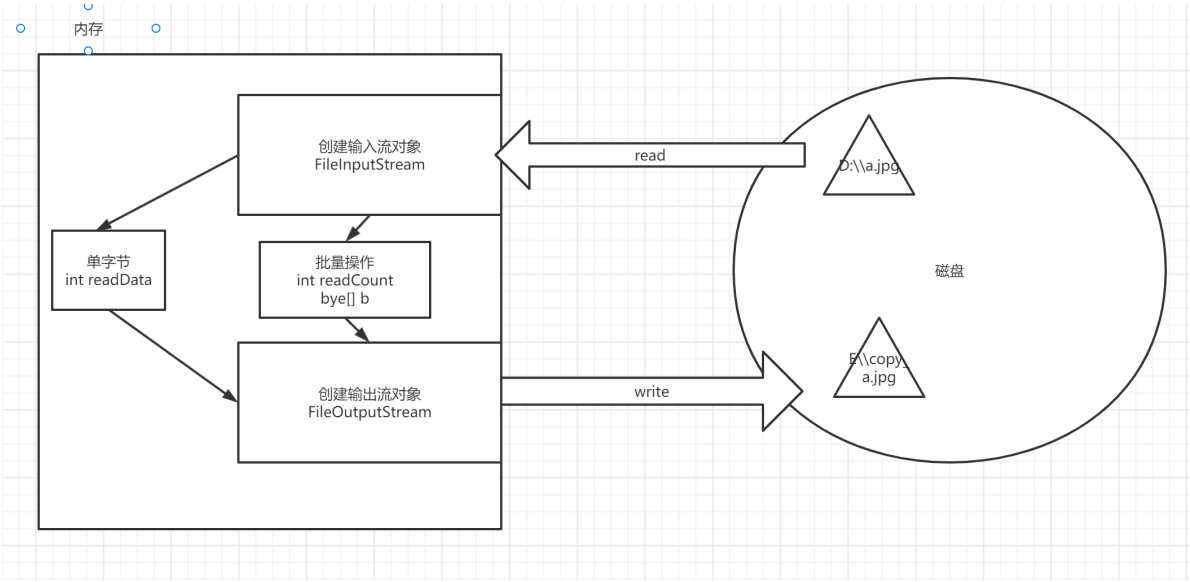
读取到末尾 返回-1

Demo

文件复制功能

思路:

- 把要复制的那个文件读取到内存
- 把数据写到新的文件中



文本文件 : 没有问题

图片文件: 没有问题

视频文件: 没有问题

单字节复制跟字节数组复制 , 哪个效率更高?

- 字节数组的效率高
- 字节数组的方式减少了跟操作系统交互

举例

我在京东买了5件商品, 快递小哥给我送快递.

单字节: 快递小哥一次送1个快递

字节数组: 东哥说了 , 大家都是兄弟 . 给每个兄弟配了一辆BMW.

字符流

为什么有字符流?

用字节流读取英文 数字 ===> 没有问题

用字节流读取中文 ==== 可能有问题

一个字符是如何存在计算机中的

基于某个编码表,每个编码表都对应一个整数值(编码值), 存储的就是编码值

编码表

ASCII：美国标准信息交换码。
用一个字节的7位可以表示。0000 0000- 0111 1111
ISO8859-1：拉丁码表。欧洲码表
用一个字节的8位表示。

GB2312：中国的中文编码表。
GBK：中国的中文编码表升级，融合了更多的中文文字符号。
GB18030：GBK的取代版本
BIG-5码 ： 通行于台湾、香港地区的一个繁体字编码方案，俗称“大五码”。

Unicode：国际标准码，融合了多种文字。

UTF-8：可变长度来表示一个字符。
UTF-8不同，它定义了一种“区间规则”，这种规则可以和ASCII编码保持最大程度的兼容：

它将Unicode编码为00000000-0000007F的字符，用单个字节来表示 0111 1111 = 7F
它将Unicode编码为00000080-000007FF的字符用两个字节表示
它将Unicode编码为00000800-0000FFFF的字符用3字节表示

1字节 0xxxxxxx
2字节 110xxxxx 10xxxxxx
3字节 1110xxxx 10xxxxxx 10xxxxxx

utf-16:

jvm使用的编码表，用2个字节来编解码

char : 2 字节

工作中常用的

- ASCII
- ISO8859-1
- GBK ： 2个字节表示一个中文字符
- UTF-8: 3个字节表示一个中文字符

默认编码表

- Win ：GBK(ANSI)
- idea: UTF-8

编解码

编码

- 基于某个编码表,把字符数据转为二进制数据,存储到计算机的过程 (把人看懂的东西 ---> 计算机看懂的东西)

解码

- 基于某个编码表,, 编码的逆过程.(把计算机看懂的东西 ----> 人看懂的东西)

中文编码表 你 0x0001

日文编码表 の 0x0001

举例:

类似于摩斯密码

java中的编解码

```
package _19io02.com.cskaoyan.charstream._02encoding;

import java.io.UnsupportedEncodingException;
import java.util.Arrays;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 10:20
 **/
/*
java中的编解码
 */
public class Demo {
    public static void main(String[] args) throws UnsupportedEncodingException {
        // 编码过程
        String s = "你好";
        //  byte[] getBytes()
        // 使用平台的默认字符集将此 String 编码为 byte 序列,
        // 并将结果存储到一个新的 byte 数组中
        //byte[] bytes = s.getBytes();
        //System.out.println(Arrays.toString(bytes));

        // byte[] getBytes(String charsetName)
        //  使用指定的字符集将此 String 编码为 byte 序列,
        // 并将结果存储到一个新的 byte 数组中。
        byte[] gBks = s.getBytes("GBk");
        System.out.println(Arrays.toString(gBks));

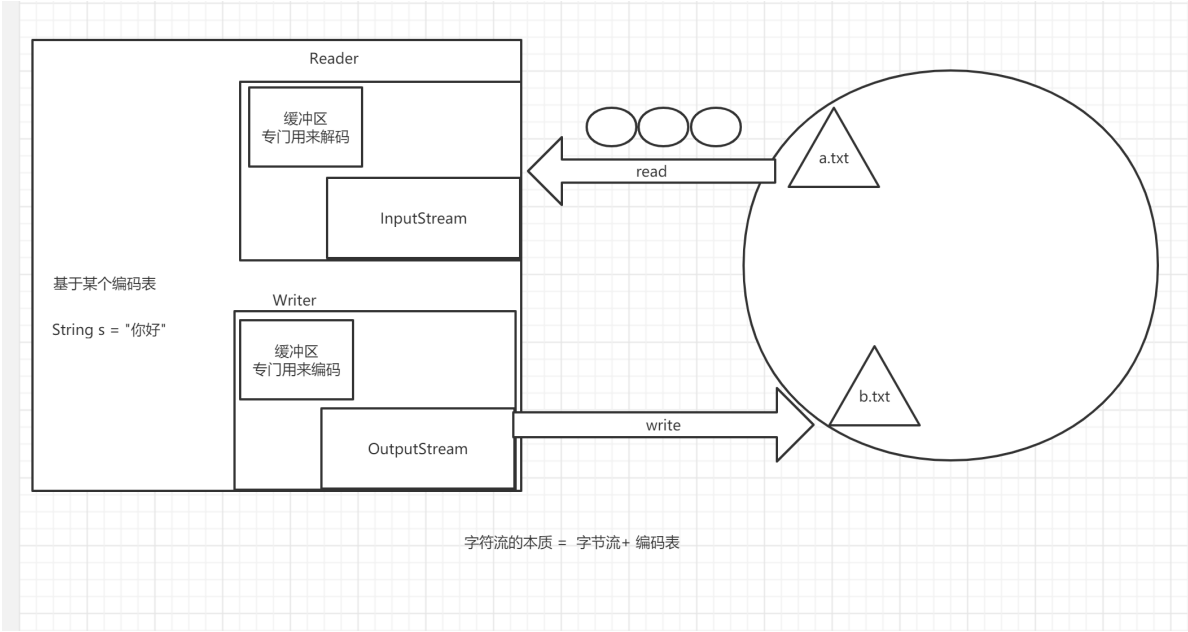
        // 解码过程
        // byte[] ---> String
        // 利用构造方法
        // String(byte[] bytes, String charsetName)
```

```
        // 通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。
        String s1 = new String(gBks,"GBK");
        System.out.println(s1);
    }
}
```

乱码问题

- 产生的原因: 编码解码不一致
- 解决方法: 使其一致

字符流的本质

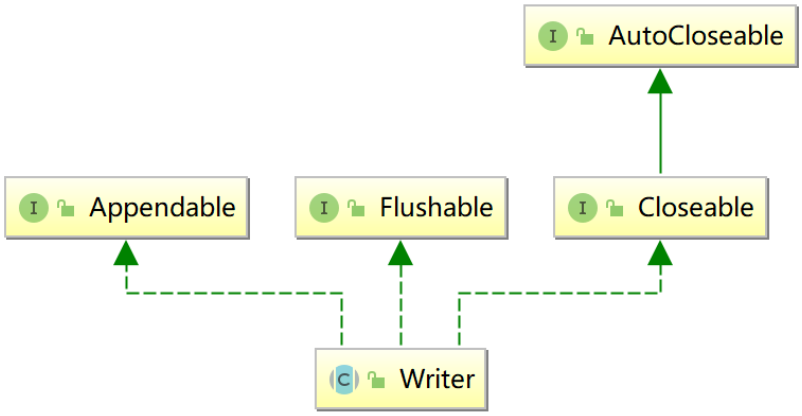


字符输出流

抽象基类Writer

写入字符流的抽象类

继承关系



成员方法

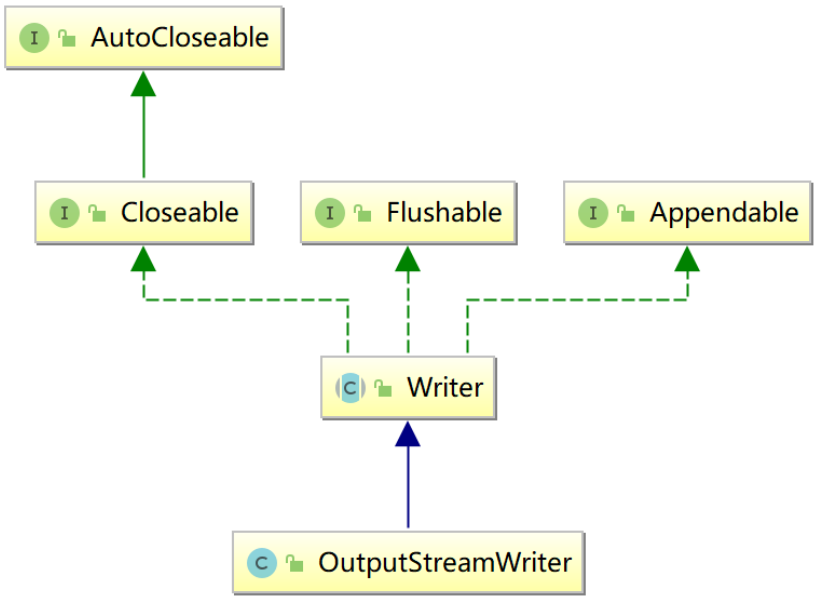
void	**write(char[] cbuf) 写入字符数组。**
abstract void	write(char[] cbuf, int off, int len) 写入字符数组的某一部分。
void	write(int c) 写入单个字符。要写入的字符包含在给定整数值的 16 个低位中，16 高位被忽略。
void	write(String str) 写入字符串。
void	write(String str, int off, int len) 写入字符串的某一部分。

具体子类

OutputStreamWriter转换流

OutputStreamWriter 是字符流通向字节流的桥梁：可使用指定的 `charset` 将要写入流中的字符编码成字节。它使用的字符集可以由名称指定或显式给定，否则将接受平台默认的字符集。

继承关系



构造方法

OutputStreamWriter(OutputStream out) 创建使用默认字符编码的 OutputStreamWriter。
OutputStreamWriter(OutputStream out, String charsetName) 创建使用指定字符集的 OutputStreamWriter。

成员方法

void	**write(char[] cbuf) 写入字符数组。**
abstract void	write(char[] cbuf, int off, int len) 写入字符数组的某一部分。
void	write(int c) 写入单个字符。要写入的字符包含在给定整数值的 16 个低位中，16 高位被忽略。
void	write(String str) 写入字符串。
void	write(String str, int off, int len) 写入字符串的某一部分。

Demo

```
package _19io02.com.cskaoyan.charstream._03transfer;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 11:16
```

```
 **/
/*
使用转换流写数据
 */
public class Demo {
    public static void main(String[] args) throws IOException {
        // 创建输出流对象
        OutputStreamWriter out =
                new OutputStreamWriter(new FileOutputStream("a.txt"));
        // write
        // write(int c) 写单个字符

        out.write(97);
        out.write(System.lineSeparator());

        // write(char[] c) 写字符数组
        String s = "宝,今天我输液了,什么夜,想你的夜";
        char[] chars = s.toCharArray();
        out.write(chars);

        // write(char[] c , int off ,int len)
        out.write(System.lineSeparator());
        out.write(chars, 2, 5);

        out.write(System.lineSeparator());
        // 字符流特有的方法 操作字符串
        // write(String s)
        out.write(s);

        out.write(System.lineSeparator());

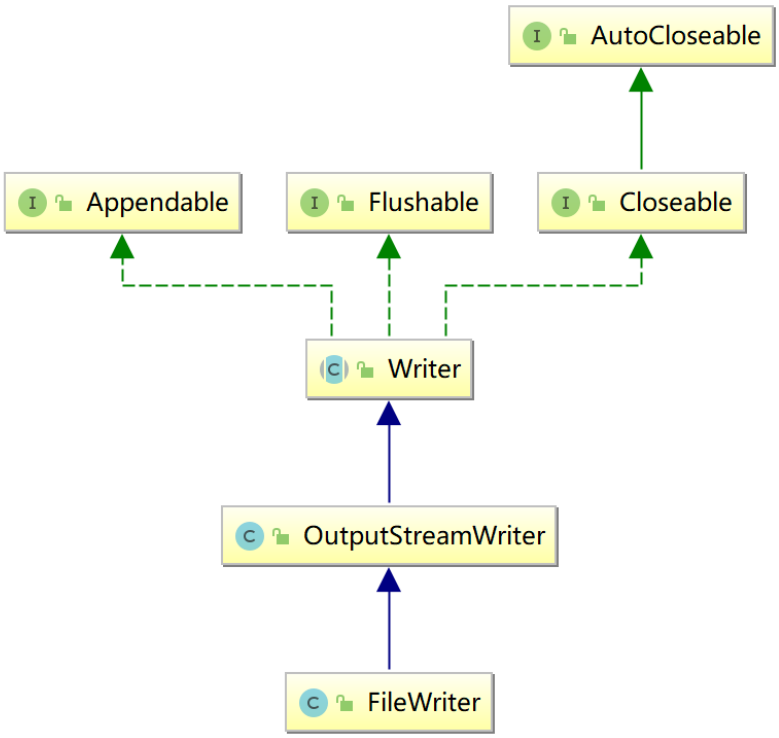
        // write(String s , int off, int len)
        out.write(s, 0, 1);


        // flush
        out.flush();
        // close
        out.close();
    }
}
```

FileWriter简化流

用来写入字符文件的便捷类

继承关系



构造方法

FileWriter(File file) 根据给定的 File 对象构造一个 FileWriter 对象。
FileWriter(File file, boolean append) 根据给定的 File 对象构造一个 FileWriter 对象。
FileWriter(String fileName) 根据给定的文件名构造一个 FileWriter 对象。
FileWriter(String fileName, boolean append) 根据给定的文件名以及指示是否附加写入数据的 boolean 值来构造 FileWriter 对象。

成员方法

3+2

3个跟字符相关的方法

2个跟String相关的方法

Demo

```
package _19io02.com.cskaoyan.charstream._05simple;

import java.io.FileWriter;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 14:42
```

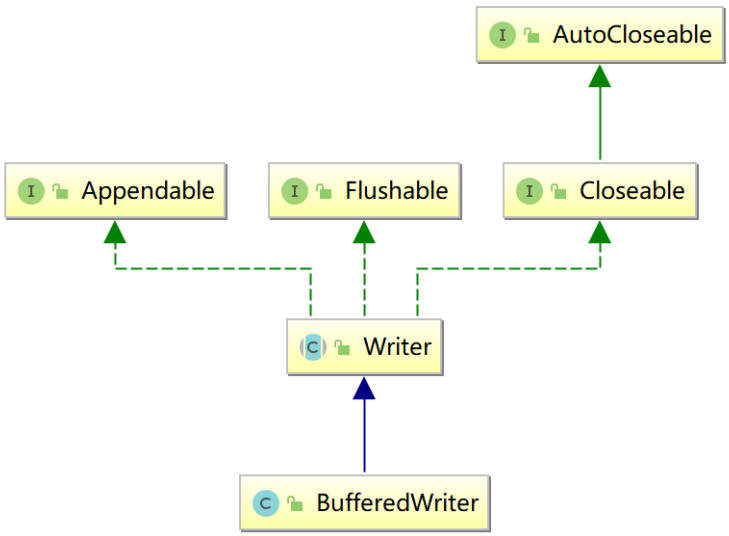
```
 **/
/*
使用简化流写数据
 */
public class Demo {
    public static void main(String[] args) throws IOException {
        // 创建输出流对象
        FileWriter fileWriter = new FileWriter("a.txt");
        // write
        fileWriter.write("大郎,来喝药");

        // flush
        fileWriter.flush();
        // close
        fileWriter.close();
    }
}
```

BufferedWriter缓冲流

将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入

继承关系



构造方法

BufferedWriter(Writer out) 创建一个使用默认大小输出缓冲区的缓冲字符输出流。16KB
BufferedWriter(Writer out, int sz) 创建一个使用给定大小输出缓冲区的新缓冲字符输出流。

成员方法

5+1

5个常规的write方法

- 3个跟字符相关的
- 2个跟String相关的

自己独有的方法

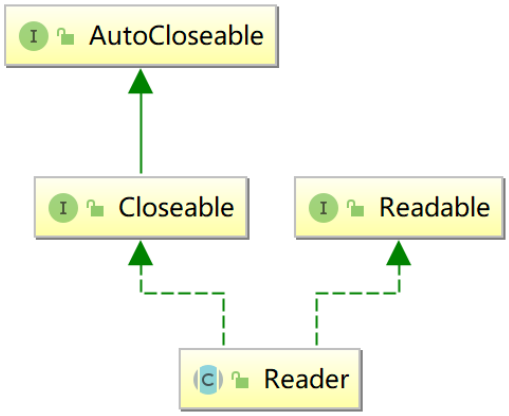
void	**newLine() 写入一个行分隔符。**

字符输入流

抽象基类Reader

用于读取字符流的抽象类

继承关系



成员方法

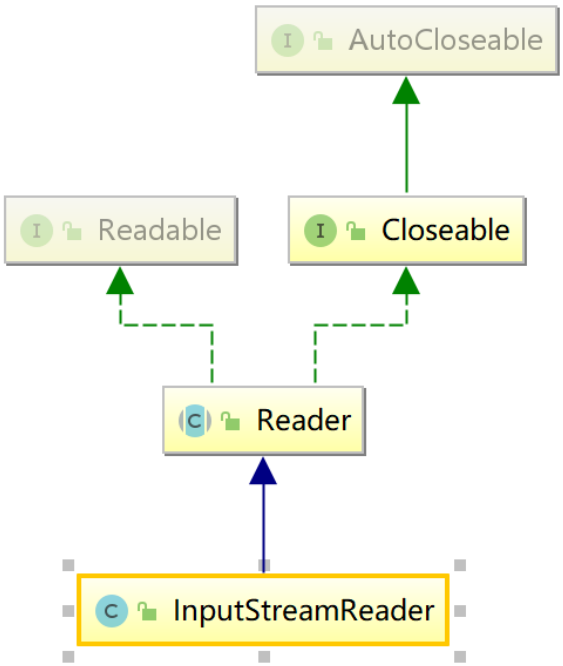
int	**read() 读取单个字符。 作为整数读取的字符，范围在 0 到 65535 之间 (0x00-0xffff)，如果已到达流的末尾，则返回 -1 readData**
int	read(char[] cbuf) 将字符读入数组。 读取的字符数，如果已到达流的末尾，则返回 -1 readCount表示
abstract int	read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

具体子类

InputStreamReader转换流

InputStreamReader 是字节流通向字符流的桥梁：它使用指定的 `charset` 读取字节并将其解码为字符。它使用的字符集可以由名称指定或显式给定，或者可以接受平台默认的字符集。

继承关系



构造方法

InputStreamReader(InputStream in) 创建一个使用默认字符集的 InputStreamReader。
InputStreamReader(InputStream in, String charsetName) 创建使用指定字符集的 InputStreamReader。

成员方法

int	**read() 读取单个字符。 作为整数读取的字符，范围在 0 到 65535 之间 (0x00-0xffff)，如果已到达流的末尾，则返回 -1 readData**
int	read(char[] cbuf) 将字符读入数组。 读取的字符数，如果已到达流的末尾，则返回 -1 readCount表示
abstract int	read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

```
package _19io02.com.cskaoyan.charstream._03transfer;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

/**
```

```
 * @description:
 * @author: 景天
 * @date: 2022/7/25 11:30
 **/

/*
使用转换流读取数据
 */
public class Demo2 {
    public static void main(String[] args) throws IOException {
        // 创建输入流对象
        InputStreamReader in =
                new InputStreamReader(new FileInputStream("b.txt"));
        // read
        // read() 读取单个字符
        // readData表示读取到的字符值
        int readData = in.read();
        System.out.println(((char) readData));

        char[] chars = new char[1024];
        // read(char[] c)
        // 表示读取到的字符的个数
        //int readCount = in.read(chars);
        //System.out.println(new String(chars,0,readCount));
        // read(char[] c, int off ,int len)
        int readCount = in.read(chars, 1, 10);
        System.out.println(new String(chars,1,readCount));


        // close
        in.close();
    }
}
```

循环读

```
package _19io02.com.cskaoyan.charstream._03transfer;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 11:37
 **/

/*
字符流循环读取
 */
public class Demo3 {
```

```
    public static void main(String[] args) throws IOException {
        // 创建输入流对象
        InputStreamReader in =
                new InputStreamReader(new FileInputStream("b.txt"));
        // read
        // 单字符
        //readSingle(in);


        // 批量
        char[] chars = new char[1024];
        int readCount;
        while ((readCount = in.read(chars)) != -1) {
            System.out.println(new String(chars,0,readCount));
        }

        // close
        in.close();
    }

    private static void readSingle(InputStreamReader in) throws IOException {
        int readData;
        while ((readData = in.read()) != -1) {
            System.out.print(((char) readData));
        }
    }
}
```

文件复制

文本文件 : 没有问题

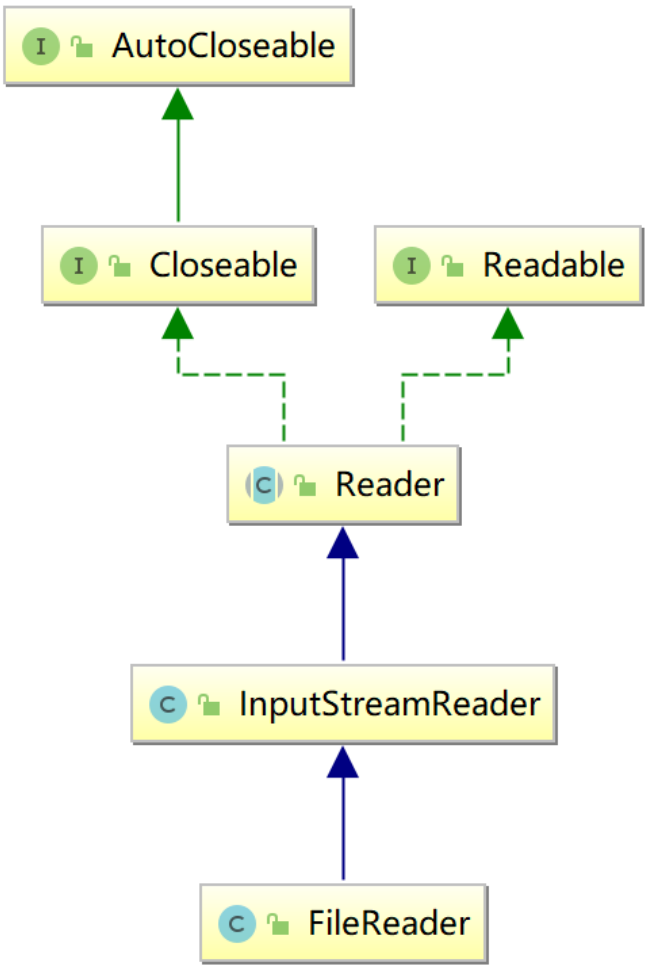
图片文件: 不行

视频文件: 不行

FileReader简化流

用来读取字符文件的便捷类

继承关系



构造方法

FileReader(File file) 在给定从中读取数据的 File 的情况下创建一个新 FileReader。
FileReader(String fileName) 在给定从中读取数据的文件名的情况下创建一个新 FileReader。

成员方法

3个read方法

read()

read(char[] c)

read(char[] c ,int off ,int len)

Demo

```
package _19io02.com.cskaoyan.charstream._05simple;

import java.io.FileReader;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 14:47
 **/
```

```java
/*
使用简化流读取数据
 */
public class Demo2 {
    public static void main(String[] args) throws IOException {
        // 创建输入流对象
        FileReader reader = new FileReader("a.txt");
        // read
        char[] chars = new char[1024];
        int readCount = reader.read(chars);
        System.out.println(new String(chars,0,readCount));
        // close
        reader.close();
    }
}
```

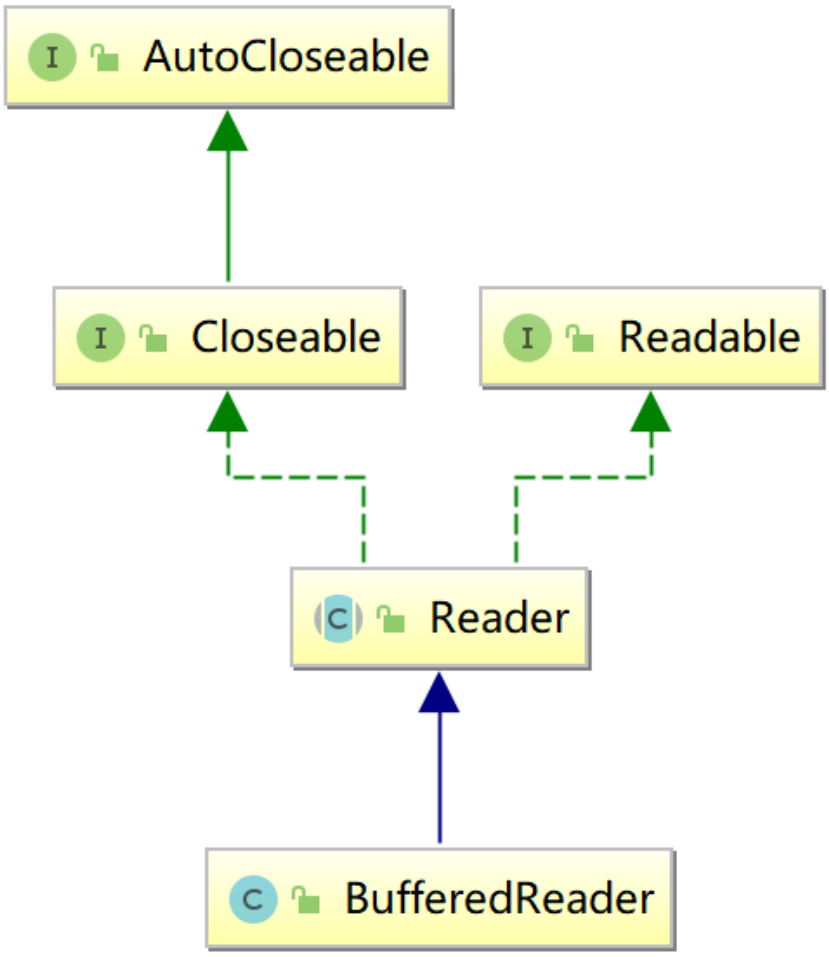
转化流 VS简化流

- 从使用角度看, 简化流简单, 转化流麻烦一点
- 从继承角度看, 转化流是父类,简化流是它的子类
- 核心区别, 转化流可以指定字符集,简化流不能指定字符集,使用默认的

BufferedReader缓冲流

从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取

继承关系



构造方法

BufferedReader(Reader in) 创建一个使用默认大小输入缓冲区的缓冲字符输入流。默认是16KB
BufferedReader(Reader in, int sz) 创建一个使用指定大小输入缓冲区的缓冲字符输入流。

成员方法

3+1

3个常规的read方法

- read()
- read(char[]c)
- read(char[] c ,int off ,int len)

自己独有的方法

String	**readLine() 读取一个文本行。**
	如果已到达流末尾，则返回 null

Demo

```java
package _19io02.com.cskaoyan.charstream._06buffer;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 15:09
 **/
/*
利用缓冲流读取数据
 */
public class Demo2 {
    public static void main(String[] args) throws IOException {
        // 创建输入流对象
        BufferedReader br = new BufferedReader(new FileReader("a.txt"));
        // readLine
        //read1(br);

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

        // close
        br.close();
    }

    private static void read1(BufferedReader br) throws IOException {
        String s = br.readLine();
        System.out.println(s);
        String s1 = br.readLine();
        System.out.println(s1);
        String s2 = br.readLine();
        System.out.println(s2);
    }
}
```

其他流

数据流

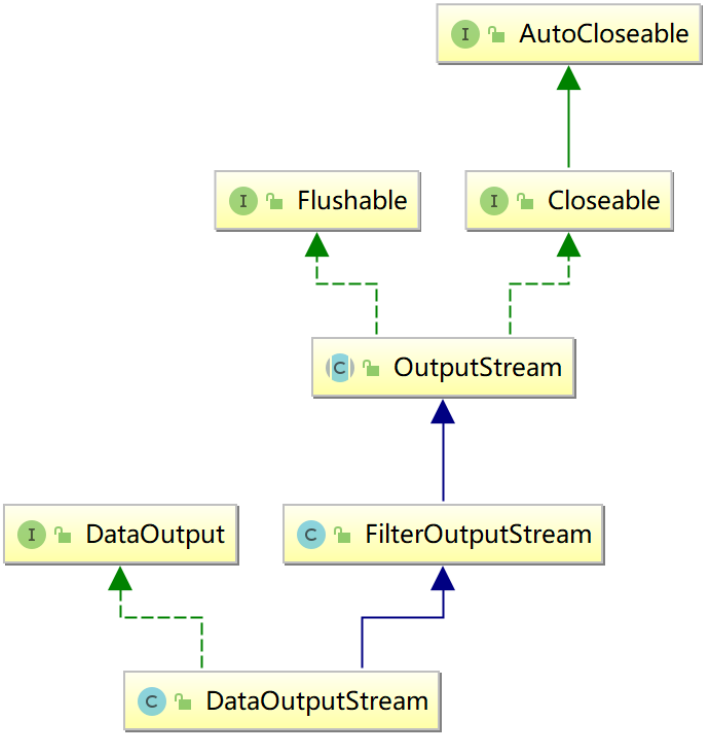
使用字节流向文件中写个整数1000 再写一个小数3.14

无法实现

DataOutputStream数据输出流

数据输出流允许应用程序以适当方式将基本 Java 数据类型写入输出流中。然后，应用程序可以使用数据输入流将数据读入

继承关系



构造方法

DataOutputStream(OutputStream out)　　创建一个新的数据输出流，将数据写入指定基础输出流。

成员方法

每个java基本数据类型都有1个相对应的write方法

比如

int　writeInt(int a)

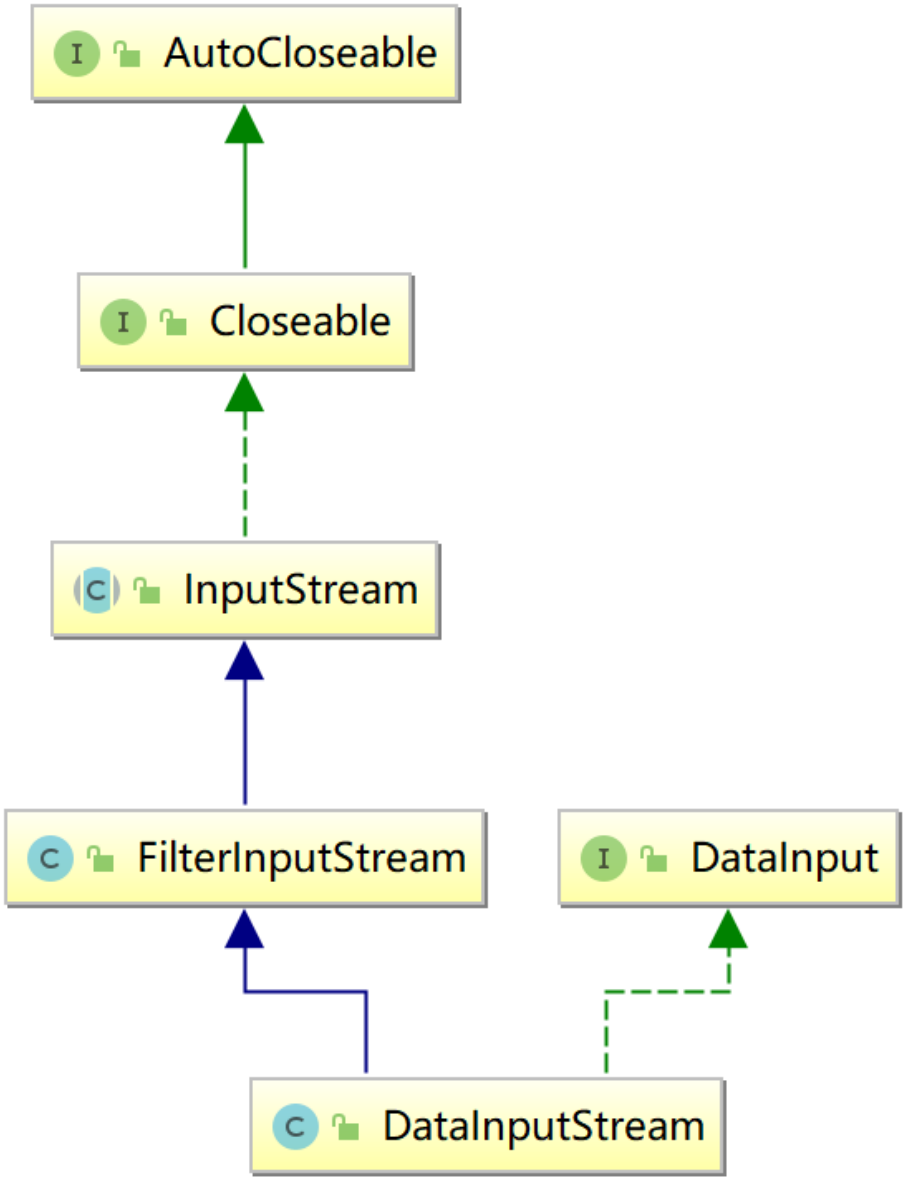
double　writeDouble(double a)

Demo

DataInputStream数据输入流

数据输入流允许应用程序以与机器无关方式从底层输入流中读取基本 Java 数据类型

继承关系



构造方法

DataInputStream(InputStream in)　　使用指定的底层 InputStream 创建一个 DataInputStream。

成员方法

每个java基本数据类型都有1个相对应的read方法

比如

int　readInt()

double　readDouble()

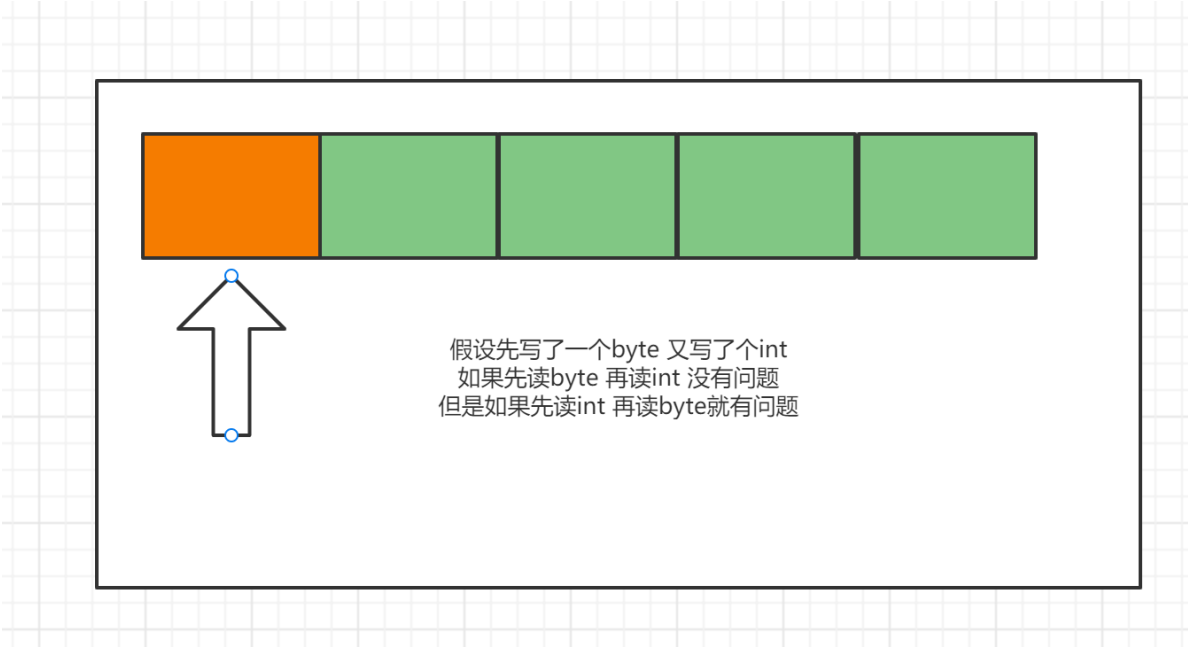
Demo

```
package _19io02.com.cskaoyan.ohterstream._01data;

import java.io.*;

/**
```

```
 * @description:
 * @author: 景天
 * @date: 2022/7/25 16:20
 **/
/*
利用数据流写java基本类型数据
 */
public class Demo2 {
    public static void main(String[] args) throws IOException {
        //writeData();

        // 读取基本数据类型
        // 创建输入流对象
        DataInputStream in = new DataInputStream(new FileInputStream("a.txt"));
        // readInt()
        int i = in.readInt();
        System.out.println(i);
        // readDouble()
        double v = in.readDouble();
        System.out.println(v);



        // close
        in.close();
    }

    private static void writeData() throws IOException {
        // 创建输出流对象
        DataOutputStream out = new DataOutputStream(new
FileOutputStream("a.txt"));
        // writeInt(int a)
        out.writeInt(1000);
        // writeDouble(double a)
        out.writeDouble(3.14);
        // close
        out.close();
    }
}
```

```
package _19io02.com.cskaoyan.ohterstream._01data;

import java.io.*;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 16:33
 **/

public class Demo3 {
    public static void main(String[] args) throws IOException {
        write();
        read();
```

```
    }

    private static void read() throws FileNotFoundException, IOException {
        DataInputStream dis = new DataInputStream(
                new FileInputStream("dos.txt"));
        byte b = dis.readByte();
        System.out.println(b);
        short s = dis.readShort();
        System.out.println(s);
        int i = dis.readInt();
        System.out.println(i);
        long l = dis.readLong();
        System.out.println(l);
        float f = dis.readFloat();
        System.out.println(f);
        double d = dis.readDouble();
        System.out.println(d);
        char ch = dis.readChar();
        System.out.println(ch);
        boolean bb = dis.readBoolean();
        System.out.println(bb);
        dis.close();
    }

    private static void write() throws IOException {
        DataOutputStream dos = new DataOutputStream(new FileOutputStream(
                "dos.txt"));
        dos.writeByte(1);
        dos.writeShort(20);
        dos.writeInt(300);
        dos.writeLong(4000);
        dos.writeFloat(12.34f);
        dos.writeDouble(12.56);
        dos.writeChar('a');
        dos.writeBoolean(true);
        dos.close();
    }



}
```



注意

- 写的顺序是什么,读取要按照相同的顺序

打印流

核心思想: 把不同类型的数据转为String 然后写的是字符串

练习题:

定义一个类Printer

提供一个成员变量OutputStream out

提供5个方法

向文件中写入int值 public void printInt(int a)

向文件中写入int值 并且换行 public void printIntLn(int a)

向文件中写入double值 public void printDouble(double a)

向文件中写入double值 并且换行 public void printDoubleLn(double a)

释放资源 public void close()

```
package _19io02.com.cskaoyan.ohterstream._02print;

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

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/25 16:43
 **/
/*
练习题:
```

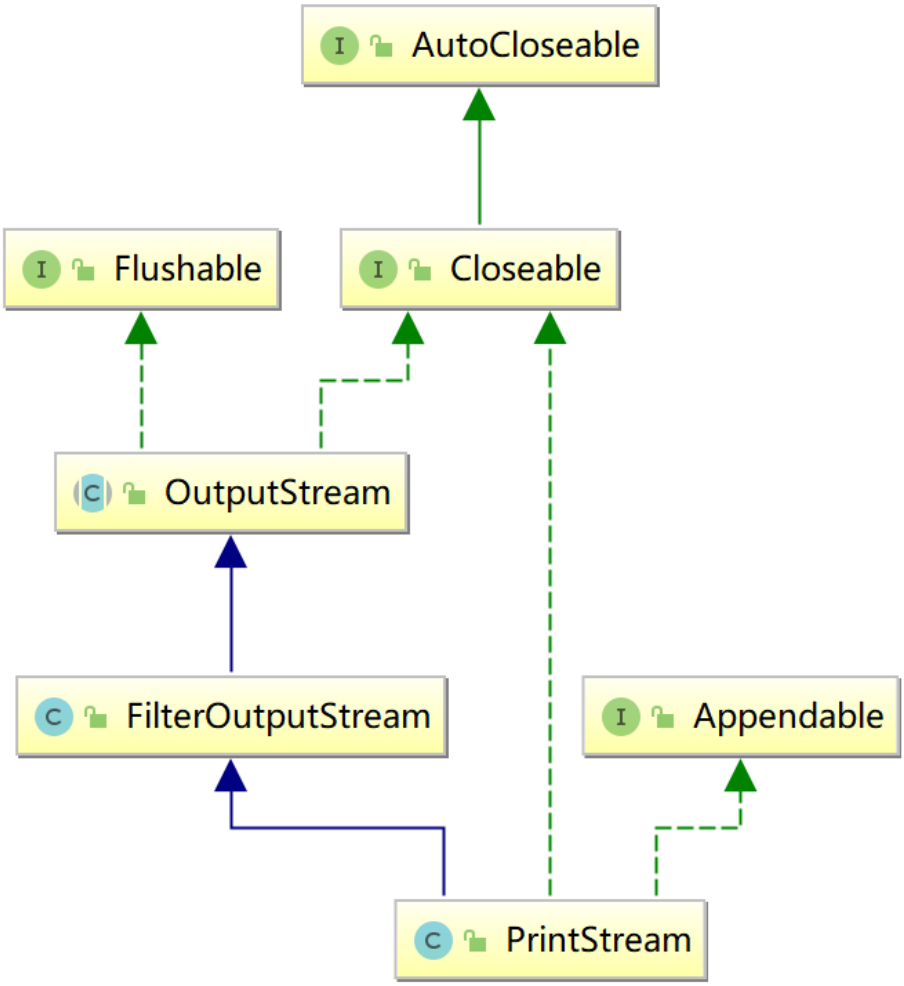
```
定义一个类Printer

提供一个成员变量OutputStream out

提供5个方法

向文件中写入int值 public void printInt(int a)

向文件中写入int值 并且换行 public void printIntLn(int a)

向文件中写入double值 public void printDouble(double a)

向文件中写入double值 并且换行 public void printDoubleLn(double a)

释放资源 public void close()
 */
public class Demo {
    public static void main(String[] args) throws IOException{
        // 创建Printer对象
        Printer printer = new Printer(new FileOutputStream("b.txt"));
        // 向文件中写1000
        printer.printIntLn(1000);

        // 3.14
        printer.printDouble(3.14);
        // close
        printer.close();
    }
}

class Printer{
    //提供一个成员变量OutputStream out
    OutputStream out;

    public Printer(OutputStream out) {
        this.out = out;
    }

    //提供5个方法
    //
    //向文件中写入int值 public void printInt(int a)
    public void printInt(int a) throws IOException {
        // int --->String
        String s = String.valueOf(a);
        out.write(s.getBytes());

    }
    //向文件中写入int值 并且换行 public void printIntLn(int a)
    public void printIntLn(int a) throws IOException {
        // int --->String
        String s = String.valueOf(a);
        out.write(s.getBytes());
        // 换行
        out.write(System.lineSeparator().getBytes());
```

```
    }
    //向文件中写入double值 public void printDouble(double a)
    public void printDouble(double a) throws IOException {
        // double --->String
        String s = String.valueOf(a);
        out.write(s.getBytes());

    }

    //向文件中写入double值 并且换行 public void printDoubleLn(double a)

    public void printDoubleLn(double a) throws IOException {
        // double --->String
        String s = String.valueOf(a);
        out.write(s.getBytes());
        out.write(System.lineSeparator().getBytes());

    }
    //释放资源 public void close()
    public void close() throws IOException {
        out.close();
    }

}
```

PrintStream字节打印流

`PrintStream` 为其他输出流添加了功能，使它们能够方便地打印各种数据值表示形式

继承关系



构造方法

PrintStream(File file) 创建具有指定文件且不带自动行刷新的新打印流。
PrintStream(OutputStream out) 创建新的打印流。
PrintStream(String fileName) 创建具有指定文件名称且不带自动行刷新的新打印流。

成员方法

每个java基本数据类型都有一个相对应的print方法

比如

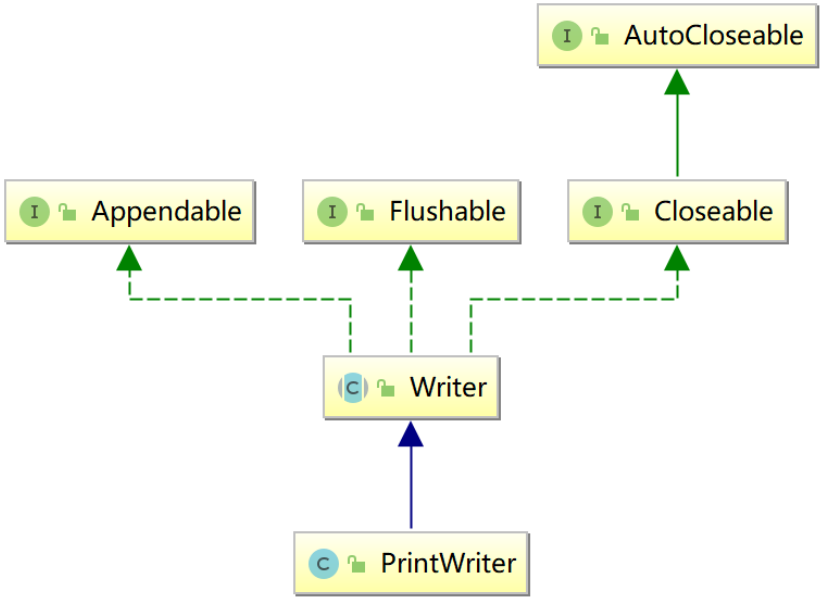
int print(int a)

double print(double a)

PrintWriter字符打印流

向文本输出流打印对象的格式化表示形式

继承关系



构造方法

PrintWriter(File file) 使用指定文件创建不具有自动行刷新的新 PrintWriter。
PrintWriter(OutputStream out) 根据现有的 OutputStream 创建不带自动行刷新的新 PrintWriter。
PrintWriter(OutputStream out, boolean autoFlush) 通过现有的 OutputStream 创建新的 PrintWriter。
PrintWriter(String fileName) 创建具有指定文件名称且不带自动行刷新的新 PrintWriter。
PrintWriter(Writer out) 创建不带自动行刷新的新 PrintWriter。
PrintWriter(Writer out, boolean autoFlush) 创建新 PrintWriter。

成员方法

每个java基本数据类型都有一个相对应的print方法

比如

int　print(int a)

double print(double a)

打印流特点

只能操作目的地，不能操作数据来源。

可以操作任意类型的数据。

- 把不同类型的数据转为字符串，操作字符串

如果启动了自动刷新，能够自动刷新。

- 但是有前提条件, 如果启用了自动刷新，则只有在调用 `println`、`printf` 或 `format` 的其中一个方法时才可能完成此操作

```
    private void newLine() {
        try {
            synchronized (lock) {
                ensureOpen();
                out.write(lineSeparator);
                if (autoFlush)
                    out.flush();
            }
        }
        catch (InterruptedIOException x) {
            Thread.currentThread().interrupt();
        }
        catch (IOException x) {
            trouble = true;
        }
    }
     */
    public PrintWriter format(Locale l, @NotNull String format, Object ...
        try {
            synchronized (lock) {
                ensureOpen();
                if ((formatter == null) || (formatter.locale() != l))
                    formatter = new Formatter( a: this, l);
                formatter.format(l, format, args);
                if (autoFlush)
                    out.flush();
            }
        } catch (InterruptedIOException x) {
            Thread.currentThread().interrupt();
        } catch (IOException x) {
            trouble = true;
        }
        return this;
    }
```

可以操作文件的流

- 构造方法可以直接传递File对象,或者String 文件名

标准输入输出流

标准输入流System.in

- 默认输入设备 键盘
- 本质:InputStream 普通字节输入流

标准输出流System.out

- 默认输入设备 显示器
- 本质: PrintStream 字节打印流

练习：利用System.in 完成Scanner的nextLine()的功能

BufferedReader

System.in ----> InputStream

对象流(序列化与反序列化流)

为什么有序列化

```
Student s = new Student("zs",20)
```

什么是序列化与反序列化?

- 序列化: 把对象数据转为二进制数据,进行持久化存储的过程
- 反序列化: 序列化的逆过程

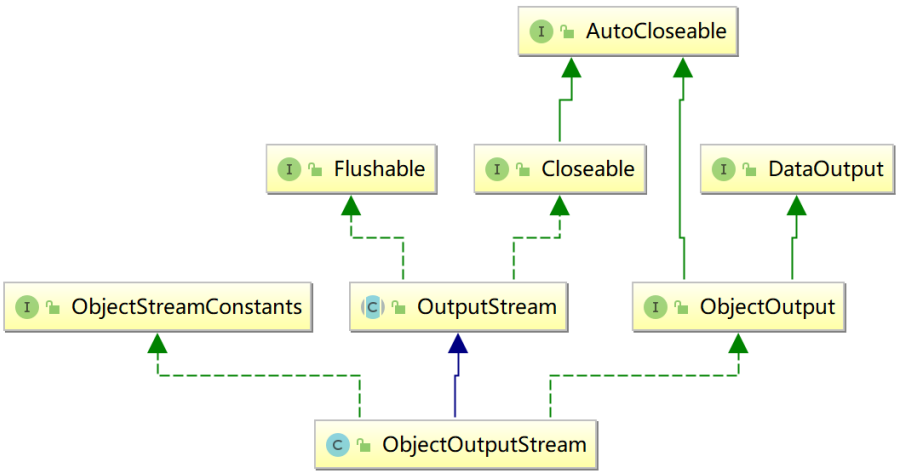
ObjectOutputStream序列化流

ObjectOutputStream 将 Java 对象的基本数据类型和图形写入 OutputStream。可以使用 ObjectInputStream 读取（重构）对象。通过在流中使用文件可以实现对象的持久存储。如果流是网络套接字流，则可以在另一台主机上或另一个进程中重构对象。

只能将支持 java.io.Serializable 接口的对象写入流中

Serializable接口是空接口 起到标记的作用

继承关系



构造方法

ObjectOutputStream(OutputStream out)　　创建写入指定 OutputStream 的 ObjectOutputStream。

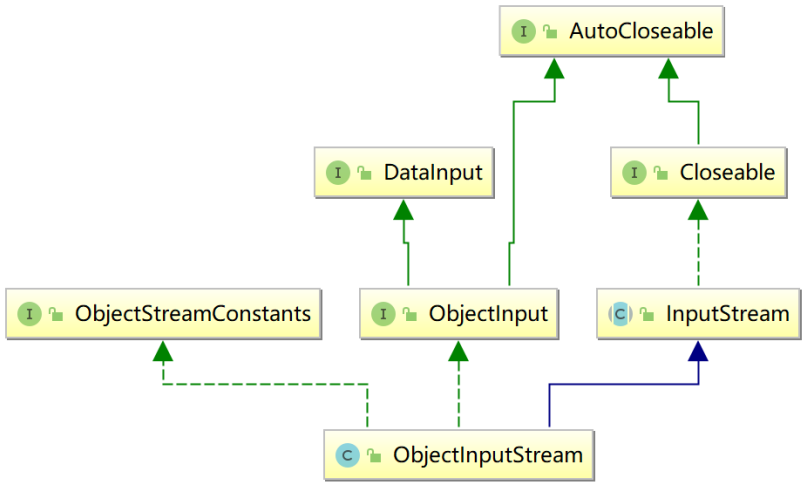
成员方法

void	writeObject(Object obj) 将指定的对象写入 ObjectOutputStream。

ObjectInputStream反序列化流

ObjectInputStream 对以前使用 ObjectOutputStream 写入的基本数据和对象进行反序列化

继承关系



构造方法

ObjectInputStream(InputStream in)　　创建从指定 InputStream 读取的 ObjectInputStream。

成员方法

Object	readObject() 从 ObjectInputStream 读取对象。

Demo

```
package _19io02.com.cskaoyan.ohterstream._03serialize;

import java.io.*;

/**
 * @description:
 * @author: 景天
 * @date: 2022/7/26 9:58
 **/
```

```
public class Demo {
    public static void main(String[] args) throws IOException,
ClassNotFoundException {
        // 序列化过程
        //serialize();

        // 反序列化操作
        // 创建输入流对象
        ObjectInputStream in = new ObjectInputStream(new
FileInputStream("a.txt"));
        // readObject()
        Object o = in.readObject();
        System.out.println(o);
        // close
        in.close();
    }

    private static void serialize() throws IOException {
        // 创建学生对象
        Student student = new Student("zs", 20);
        // 创建输出流对象
        ObjectOutputStream out =
                new ObjectOutputStream(new FileOutputStream("a.txt"));
        // writeObject(Object obj)
        out.writeObject(student);
        // close
        out.close();
    }
}
```

异常

- java.io.NotSerializableException　没有实现Serializable接口
- java.io.InvalidClassException: *19io02.com.cskaoyan.ohterstream.*03serialize.Student; local class incompatible: stream classdesc serialVersionUID = -7541313839409796339, local class serialVersionUID = 6795642774697880548　原因就是因为我们的类发生了变化 serialVersionUID不相等
- `transient关键字: 类中成员不想被序列化,使用该关键字进行修饰`

总结

类型	字节输出流	字节输入流	字符输出流	字符输入流
抽象基类	OutputStream	InputStream	Writer	Reader
文件相关的	FileOutputStream	FileInputStream	FileWriter	FileReader
缓冲相关的	BufferedOutputStream	BufferedInputStream	BufferedWriter	BufferedReader
转换相关			OutputStreamWriter	InputStreamReader
数据相关	DataOutputStream	DataInputStream		
打印相关	PrintStream		PrintWriter	
对象相关	ObjectOutputStream	ObjectInputStream		