

IO流概述

2018年8月24日 14:05

1. IO流概念

InputStream - 输入流

OutputStream - 输出流

Java程序输入输出数据的方式

2. IO流的分类

根据方向可以分为 输入流 输出流

根据操作的内容的不同 分为 字节流 和 字符流

两两相乘就得到了四大基本流：

	输入流	输出流
字符流	Reader	Writer
字节流	InputStream	OutputStream

这四大基本流都是抽象的，使用时通常使用这些抽象类的具体实现类

字符流 - 字符输入流

2018年8月24日 14:12

1. 字符输入流 Reader - FileReader

构造方法：

构造方法摘要

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。

重要方法：

int	<u>read</u> ()	读取单个字符。
int	<u>read</u> (char[] cbuf)	将字符读入数组。
abstract int	<u>read</u> (char[] cbuf, int off, int len)	将字符读入数组的某一部分。
abstract void	<u>close</u> ()	关闭该流并释放与之关联的所有资

****为什么read方法返回的不是char而是Int**

因为read如果返回的是char，则无法用任意的char表示到达了文件的结尾，所以此处不返回char而是返回int，正常情况下返回的是正数，强转为char即可得到对应字符，而当读取到文件结尾返回-1表示

案例：编写一个流 来读取外部文件中的字符数据

```
package cn.tedu.io;

import java.io.FileReader;
import java.io.Reader;

/**
 * 案例：编写一个流 来读取外部文件中的字符数据
 */
public class Demo01 {
    public static void main(String[] args) throws
        Exception {
        //1.创建文件字符输入流链接到 1.txt上
        Reader reader = new FileReader("1.txt");
        //2.通过流读取文件中的数据
        int i = 0;
        while((i=reader.read())!=-1){
            System.out.println((char)i);
        }
        //3.关闭流
        reader.close();
    }
}
```

字符流-字符输出流

2018年8月24日 14:17

1. 字符流-字符输出流-Writer-FileWriter

构造方法

构造方法摘要

FileWriter(**File** file)

根据给定的 File 对象构造一个 FileWriter 对

FileWriter(**String** fileName)

根据给定的文件名构造一个 FileWriter 对

重要方法

void	<u>write</u> (char[] cbuf) 写入字符数组。
abstract void	<u>write</u> (char[] cbuf, int off, int len) 写入字符数组的某一部分。
void	<u>write</u> (int c) 写入单个字符。
abstract void	<u>close</u> () 关闭此流，但要先刷新它。
abstract void	<u>flush</u> () 刷新该流的缓冲。

****在输出数据时，有部分数据可能会被缓冲在流的内部，通过调用flush()可以强制刷新流，将缓存在流内部的数据刷出出去，所以在writer()之后 最好做一次flush()**

****调用close()方法时，close()方法内部会隐含的做一次flush()防止在关流时有数据死在缓冲区内**

案例：编写一个流 来讲指定的字符写出到外部文件中

```
package cn.tedu.io;

import java.io.FileWriter;
import java.io.Writer;

/**
 * 案例：编写一个流 来将指定的字符写出到外部文件中
 */
public class Demo02 {
    public static void main(String[] args) throws Exception {
        //1.创建文件字符输出流
        Writer writer = new FileWriter("2.txt");
        //2.通过字符输出流输出数据
        writer.write((int)'a');
        writer.write((int)'b');
        writer.write((int)'c');
        writer.write((int)'d');
        writer.write((int)'e');
        //3.刷新流
        //writer.flush();
        //--关闭流，关闭流过程中会隐含的刷新一次流
        writer.close();
    }
}
```

字符流-关闭流过程

2018年8月24日 14:21

1. 关闭流的过程

IO流用过之后必须关闭，但是IO流的代码中往往存在大量的异常，为了保证关流的操作一定会被执行，所以通过关流都在finally代码块中进行。

而为了保证finally中可以看到流对象，通常都要外置定义流对象。

又由于close方法本身有异常，需要再次捕获异常。

而在finally里通过将引用置为null 使其成为垃圾 可以被回收。

```
finally {  
    //3.关闭流  
    if(reader!=null){  
        try {  
            reader.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally{  
            reader = null;  
        }  
    }  
    if(writer!=null){  
        try {  
            writer.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally{  
            writer = null;  
        }  
    }  
}
```

字符流 - 自定义缓冲实现效率的提升

2018年8月24日 14:22

案例：编写流来拷贝文件

```
package cn.tedu.io;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.io.Writer;

/**
 * 案例：利用字符流实现字符文件的拷贝 1.txt -> 3.txt
 */
public class Demo03 {
    public static void main(String[] args) {
        Reader reader = null;
        Writer writer = null;
        try {
            //1.创建字符输入流 连接1.txt 创建字符输出流 连接3.txt
            reader= new FileReader("1.txt");
            writer = new FileWriter("3.txt");
            //2.从1.txt读取数据
            int i = 0;
            while((i = reader.read())!=-1){
                writer.write(i);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //3.关闭流
            if(reader!=null){
                try {
```

} }

****由于一次拷贝一个字节 效率非常低**

```
package cn.tedu.io;
```

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.io.Writer;
```

```
/**
 * 通过字符流 拷贝字符文件 讨论效率问题
 */
public class Demo04 {
    public static void main(String[] args){
        //--开始时间
```



```

long begin = System.currentTimeMillis();

Reader reader = null;
Writer writer = null;
try {
    //1.创建流
    reader = new FileReader("4.txt");
    writer = new FileWriter("5.txt");
    //2.拷贝数据
    //int count = 0;
    int i = 0;
    char [] buf = new char [1024];
    while ((i = reader.read(buf)) != -1) {
        //System.out.println("读取了"++count+"次");
        writer.write(buf,0,i);
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    //3.关闭流
    if(reader!=null){
        try {
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        } finally{
            reader = null;
        }
    }
    if(writer!=null){
        try {
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            writer = null;
        }
    }
}

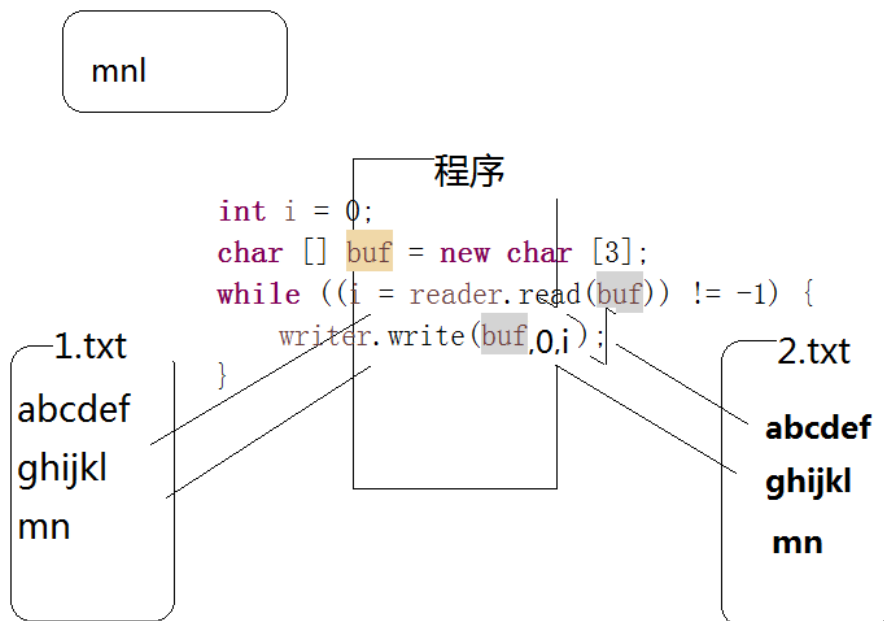
```

```

    }
}
//--结束时间
long end = System.currentTimeMillis();
System.out.println("===共耗时："+(end-
begin)+"ms===");
}
}

```

通过自定义char[],一次拷贝一个数组提升性能



字符流 - 缓冲流 - BufferedReader BufferedWriter

2018年8月24日 14:24

1. 缓冲流概述

java提供了自带缓冲区的流 `BufferedReader`
`bufferedWriter` , 内部自带缓冲区

功能 :

装饰其他流 提升读写性能
装饰起来流 提供额外方法

2. 重要API

构造方法 :

构造方法摘要

[`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。

[`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。

重要方法 :

[`String`](#)

[`readLine\(\)`](#)

读取一个文本行。

构造方法 :

构造方法摘要

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。

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。

重要方法：

void	<u>newLine</u> () 写入一个行分隔符。
------	--------------------------------

案例：利用缓冲流 包装普通的文件流 复制文件

```
package cn.tedu.io;

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

/**
 * BufferedReader和BufferedWriter提供的新方法
 *
 * * BufferedReader
 * *   readLine()
 * *
 * * BufferedWriter
 * *   newLine()
 */
public class Demo06 {
    public static void main(String[] args) throws
        Exception {
```

//1.创建流 并用缓冲流包装

```
BufferedReader reader = new  
BufferedReader(new FileReader("1.txt"));  
BufferedWriter writer = new  
BufferedWriter(new FileWriter("2.txt"));
```

//2.对接流 拷贝文件

```
String line = null;  
while((line = reader.readLine())!=null){  
    writer.write(line);  
    writer.newLine();  
}
```

//3.关闭流

```
reader.close();  
writer.close();
```

```
    }  
}
```

装饰设计模式

2018年8月24日 14:28

1. 增强类的功能的三种方式

a. 继承

可以通过继承父类，在子类中增强能力，缺点是对已经创建的父类对象无效

b. 装饰设计模式

是java的23中设计模式之一，可以用来增强对象的能力
可以将已有的对象装饰

c. 动态代理

暂时不讲

2. 装饰设计模式概述

所谓的设计模式，其实就是前人总结的写代码的套路。
java中共有23种设计模式。
装饰设计模式是其中的一种。

装饰设计模式的主要的功能，就是在原有的被装饰者的基础上增加其他能力。

3. 装饰设计模式的实现过程

- a. 写一个装饰类 并在装饰类的内部提供一个构造方法 接受参数 允许将被装饰者传入 并保存在类的内部
- b. 提供和Man相同的方法，为了能够具有相同的方法，通

常会选择和被装饰者实现相同的接口或继承相同的父类，在这些方法中调用在构造方法中传入的被装饰者相同的方法

- c. 需要增强的功能，在原有的功能上增强，或提供新的方法

案例：为Person类通过装饰设计模式 使其能够飞

```
interface Ani{
    public void eat();
    public void sleep();
}

class Man implements Ani{
    protected String name;
    protected int age;

    public Man() {
    }

    public Man(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void eat(){
        System.out.println(name+"吃。。。");
    }
    public void sleep(){
        System.out.println(name+"睡。。。");
    }
}

/**
```

```

* 装饰设计模式
*/
class SuperMan2 implements Ani{
    private Man man = null;
    public SuperMan2(Man man) {
        this.man = man;
    }
    @Override
    public void eat() {
        System.out.println("绕地球飞一圈。。");
        man.eat();
        System.out.println("到太平洋里游一圈。。");
    }
    @Override
    public void sleep() {
        man.sleep();
    }

    public void fly(){
        System.out.println(man.name + "飞。。。");
    }
}

public static void main(String[] args) {
    //--装饰
    Man man = new Man("ls",20);
    man.eat();
    man.sleep();

    SuperMan2 sman = new SuperMan2(man);
    sman.eat();
    sman.sleep();
    sman.fly();
}
}

```

4. 缓冲流也是装饰设计模式的实现

BufferedReader BufferedWriter就是使用了装饰设计模式实现的

实验：翻阅 [BufferedReader BufferedWriter 源代码](#)

字符流 - 其他字符流 - StringReader

2018年8月24日 15:16

1. 其他字符流 - StringReader

数据来源为字符串的字符输入流

构造方法：

构造方法摘要

StringReader(String s)

创建一个新字符串 reader。

重要方法：

int	<u>read</u> ()	读取单个字符。
int	<u>read</u> (char[] cbuf, int off, int len)	将字符读入数组的某一部分。
void	<u>close</u> ()	关闭该流并释放与之关联的所有系统资源。

案例：利用StringWriter以一段字符串为数据来源 对接流 输出到文件中

```
package cn.tedu.io;
```

```
import java.io.FileWriter;  
import java.io.StringReader;
```

```
/**
```

```
 * StringReader的用法
```

```
 */
```

```
public class Demo08 {
```

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

```
        //1.创建StringReader
```

```
String str = "hello java~ hello world~ 这是一段中文~";
StringReader reader = new StringReader(str);
//2.创建FileWriter
FileWriter writer = new FileWriter("7.txt");
//2.对接流输出
int i = 0;
char [] cs = new char[1024];
while((i = reader.read(cs))!=-1){
    writer.write(cs,0,i);
}
//3.关闭流
reader.close();
writer.close();
    }
}
```

字节流 - 字节输入流

2018年8月24日 15:25

1. 字节流-字节输入流-InputStream-FileInputStream

构造方法：

构造方法摘要

[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](#) name)

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。

重要方法：

int	<u>read</u> ()	从此输入流中读取一个数据字节。
int	<u>read</u> (byte[] b)	从此输入流中将最多 <code>b.length</code> 个字节的数据读入一个 <code>byte</code> 数组中。
int	<u>read</u> (byte[] b, int off, int len)	从此输入流中将最多 <code>len</code> 个字节的数据读入一个 <code>byte</code> 数组中。
void	<u>close</u> ()	关闭此文件输入流并释放与此流有关的所有系统资源。

****其中read()方法读取字节信息，但返回的是一个int，这是因为如果返回byte，则无论返回什么都无法表示读取到文件结尾的状态，所以read()方法 返回的是int，正常读取到数据时都是正数，直接强转就可以得到对应byte，而当读取到文件结尾时，返回一个-1，作为特殊状态值。**

案例：利用字节流复制文件

```
package cn.tedu.io2;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
```

```

/**
 * 利用字节流实现文件的复制
 */
public class Demo01 {
    public static void main(String[] args) throws Exception {
        //1.创建流
        InputStream in = new FileInputStream("1.wmv");
        OutputStream out = new FileOutputStream("2.wmv");
        //2.对接流 实现复制
        int i = 0;
        while((i = in.read())!=-1){
            out.write(i);
        }
        //3.关闭流
        in.close();
        out.close();
    }
}

```

案例：利用字节流复制文件 - 使用自定义的缓冲区

```

package cn.tedu.io2;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * 利用字节流实现文件的复制 - 使用缓冲区
 */
public class Demo02 {
    public static void main(String[] args) throws Exception {
        //1.创建流
        InputStream in = new FileInputStream("1.wmv");
        OutputStream out = new FileOutputStream("2.wmv");
        //2.对接流 实现复制
        int i = 0;
        byte [] tmp = new byte[1024];
        while((i = in.read(tmp))!=-1){
            out.write(tmp,0,i);
        }
        //3.关闭流
        in.close();
        out.close();
    }
}

```

}

字节流 - 字节输出流

2018年8月24日 15:30

1. 字节流-字节输出流-OutputStream-FileOutputStream

构造方法：

**通过boolean的append参数，可以指定数据是否追加，如果传入false（默认就是false）则，会产生新的文件覆盖旧的文件，如果传入true，则在原有文件的基础上进行追加

构造方法摘要

[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](#) name)

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。

普通方法：

void [write](#)(byte[] b)

将 b.length 个字节从指定 byte 数组写入此文件输出流中。

void	<u>write</u> (byte[] b, int off, int len) 将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此文件输出流。
void	<u>write</u> (int b) 将指定字节写入此文件输出流。
void	<u>flush</u> () 刷新此输出流并强制写出所有缓冲的输出
void	<u>close</u> () 关闭此文件输出流并释放与此流有关的所有系统资源。

****在利用输出流输出数据的过程中，流的底层具有缓冲机制提升效率，但同时也有可能造成部分数据堆积在底层流的缓冲区中，一时无法写出，此时可以调用flush()方法，手动的将流中缓冲的数据写出**

****close()方法关闭流，在关闭流的过程中，会隐含的调用一次flush() 保证不会有数据死在缓冲区里。**

案例：通过字节流 实现文件复制

字节流和字符流的桥梁 - 转换流

2018年8月24日 16:21

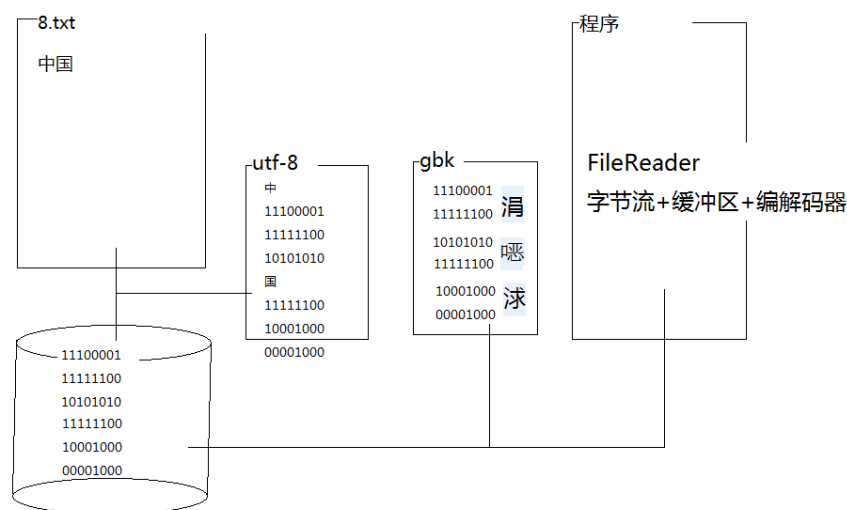
1. 转换流概述

字符流的底层也是字节流，只不过在字节流的基础上增加了缓冲区和编解码器。

字符流内置的编解码器默认采用的编码集是系统码，并且无法修改。

这在使用字符流读写非系统码的字符数据时就会造成乱码。

此时无法通过字符流解决，java提供了转换流，可以自己编写字节流读取数据，再通过转换流转换为字符流，并且在这个过程中自己手动指定码表，从而实现采用指定码表的自定义字符流。



2. 转换流API

java.io

类 InputStreamReader

构造方法：

构造方法摘要

[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。

java.io

类 OutputStreamWriter

构造方法：

构造方法摘要

[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。

案例：拷贝一个utf-8编码集 包含中文的文本文件 要求产生的文件也是utf-8编码

```
package cn.tedu.io2;

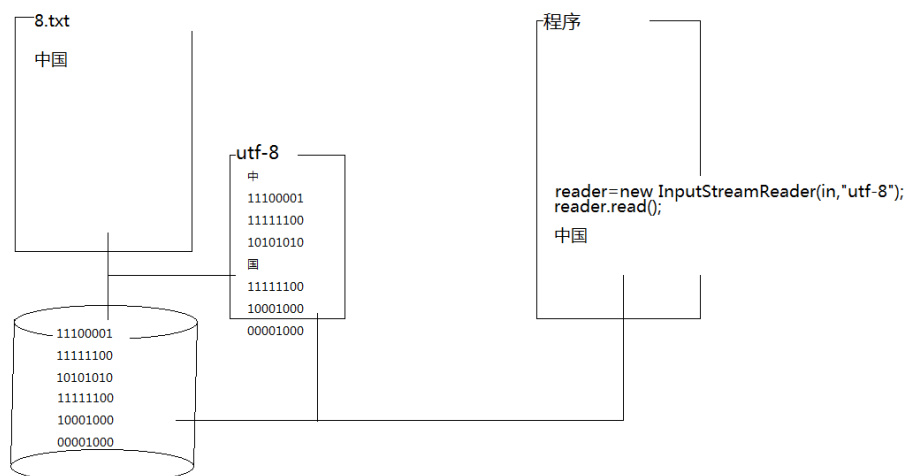
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;

/**
 * 案例：通过转换流 生成自定义码表的字符流 复制文件 解决乱码
 */
public class Demo04 {
    public static void main(String[] args) throws Exception {
        //1.创建字节流
        InputStream in = new FileInputStream("d://8.txt");
        OutputStream out = new FileOutputStream("d://9.txt");

        //2.创建转换流 将字节流转换为字符流 并显式指定码表为utf-8
        InputStreamReader reader = new InputStreamReader(in,"utf-8");
        OutputStreamWriter writer = new OutputStreamWriter(out,"utf-8");

        //3.对接流 复制文件
        char [] data = new char[1024];
        int i = 0;
        while((i=reader.read(data))!=-1){
            writer.write(data,0,i);
        }

        //4.关闭流
        reader.close();
        writer.close();
    }
}
```



案例：拷贝一个utf-8编码集 包含中文的文本文件 要求产生的文件是gbk编码

```
package cn.tedu.io2;
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
```

```
/**
```

```
 * 案例：通过转换流 生成自定义码表的字符流 复制文件 将utf-8文件转换为gbk格
式的文件
```

```
 */
```

```
public class Demo05 {
```

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

```
        //1.创建字节流
```

```
        InputStream in = new FileInputStream("d://8.txt");
```

```
        OutputStream out = new FileOutputStream("d://9.txt");
```

```
        //2.创建转换流 将字节流转换为字符流 并显式指定码表为utf-8
```

```
        InputStreamReader reader = new InputStreamReader(in,"utf-8");
```

```
        OutputStreamWriter writer = new OutputStreamWriter(out,"gbk");
```

```
        //3.对接流 复制文件
```

```
        char [] data = new char[1024];
```

```
        int i = 0;
```

```
        while((i=reader.read(data))!=-1){
```

```
            writer.write(data,0,i);
```

```
        }
```

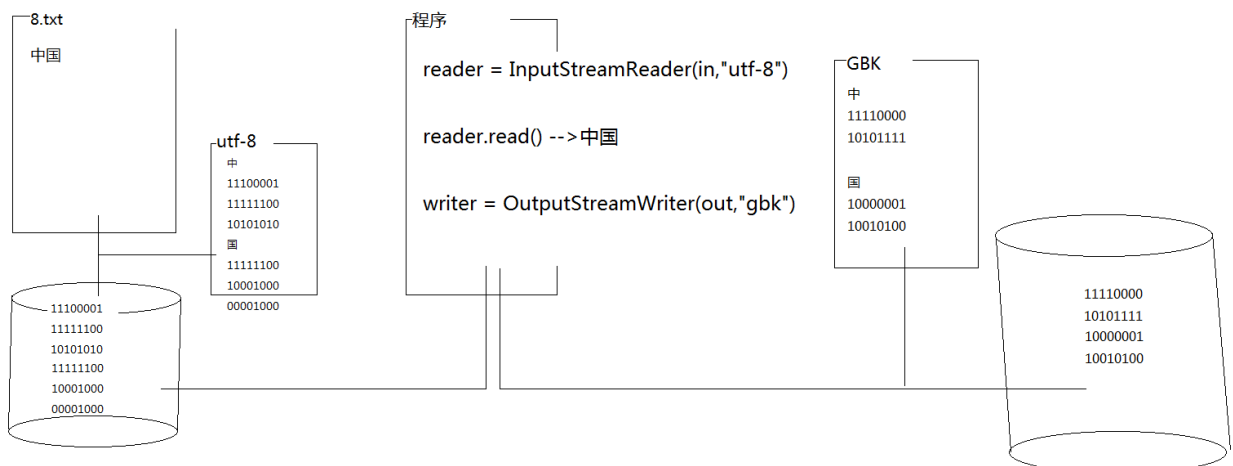
```
        //4.关闭流
```

```
        reader.close();
```

```
        writer.close();
```

```
    }
```

```
}
```



字节流-其他流 - 系统流

2018年8月24日 16:45

1. 系统流概述

在java中有一个重要的类，叫做System，代表当前系统，在其中提供了大量的和当前系统相关的静态属性和静态方法，可以实现一些系统级别的操作，其中包含了如下三个重要的属性：

java.lang

类 System

字段摘要	
static PrintStream	<u>err</u> “标准” 错误输出流。
static InputStream	<u>in</u> “标准” 输入流。
static PrintStream	<u>out</u> “标准” 输出流。

这三个属性，代表的就是系统流

所谓的系统流，就是当前系统提供给我们使用的流，这些流不需要创建，也不可以关闭，需要的时候拿过来用即可。

系统流默认连接到当前程序的控制台，从而可以实现从控制台中读写数据的操作。

另外，这三个系统流连接到哪里是可以改的。

2. 系统输出流

指向系统默认输出位置的输出流 如果不修改默认是控制台

```
System.out.println("abc");  
System.err.println("xyz");
```

****唯一的区别在于err打印的数据是红色的，out打印的是黑色的**

3. 修改标系统输出流的输出位置

```
static void setOut(PrintStream out)
```

	重新分配 “标准” 输出流。
static void	<code>setErr(PrintStream err)</code> 重新分配 “标准” 错误输出流。

4. 系统输入流

指向系统默认输入位置的输入流 如果不指定默认是控制台

```
InputStream in = System.in;
System.out.println((char)in.read());
```

或

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
String line = reader.readLine();
System.out.println(line);
```

5. 修改标准系统输入流的输入位置

static void	<code>setIn(InputStream in)</code> 重新分配 “标准” 输入流。
-------------	--