# day17

## 字符流

### 字节流读取中文乱码问题

1. 问题 : 通过字节输入流读取一个带有中文文本文件
2. 按照单个字节进行读取, 读取到结果乱码(实际结果与预期结果不符), 原因 : 默认的GBK编码中, 一个中文占有2个字节, 通过单个字节读取文件, 就将中文拆分了, 拆分之后转换成对应的字符, 导致乱码
3. 按照字节数组读取, 设置大小为2字节数组, 如果文件中是中英文混合, 仍然有可能将中文拆分开读取, 结果还是乱码

结论 : 不要使用字节流读取带有中文的文本文件

1. 解决中文文本读取乱码问题? 按照字符读取,就可以解决中文被拆分的问题, 因为一个字母, 一个符号, 或者是一个中文, 一个数字, 统统都是一个字符

代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileInputStream;  import java.io.IOException;  public class Demo01\_FileInputStreamReadChinese {  public static void main(String[] args) throws IOException{  // 1. 绑定一个数据源  FileInputStream fis = new FileInputStream("chinese.txt");  // 2. 使用单个字节进行文件内容读取  // len表示每次读取到的字节对应的数据结果  /\* int len;  while((len = fis.read()) != -1) {  System.out.print((char)len);  }\*/    // 3. 使用字节数组读取文件内容  byte[] b = new byte[2];  // len表示每次读取到的字节的个数  int len;  while((len = fis.read(b)) != -1) {  System.out.print(new String(b,0,len));  }  fis.close();  }  } |

### 字符输入流

1. Reader : 字符输入流抽象父类, 来自于java.io包
2. 抽象类不能实例化对象, 因此需要子类 FileReader, 来自于java.io包, 作用就是通过字符读取文件内容, 注意 : FileReader主要读取文本文件(txt文件)
3. 字符输入流FileReader读取原理:

所有文件都是字节组成, 在默认的GBK编码表中, 一个数字, 一个符号, 一个大小写字母, 都占有一个字节, 来自于ASCII编码表中, 都是正数(0-127); 一个中文占有2个字节, 第一个字节为负数;

字符输入流会先从文件中读取出一个字节, 判断如果这个字节为正数, 证明读取到了ASCII编码表中的一个字符, 直接参考编码表转字符; 如果读取到的第一个字节是负数, 那么动态再向下读取出另外一个字节, 将两个字节一起参考编码表转换成字符

1. FileReader构造方法:

a : [FileReader](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/FileReader.html" \l "FileReader(java.lang.String))([String](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/lang/String.html" \o "java.lang 中的类) fileName)

b: [FileReader](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/FileReader.html" \l "FileReader(java.io.File))([File](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/File.html" \o "java.io 中的类) file)

1. 读取文件的方式:
2. read() : 每次读取一个字符, 返回值类型int, 返回的就是读取到的字符在编码表中的整数结果, 如果返回-1,证明读取完毕
3. read(char[] ch) : 每次最多读取出ch.length个字符, 将读取到的字符数据同步到参数ch数组中, 查看ch数组相当于在查看文件内容; 返回值类型int, 表示本次读取到的字符的个数, 如果读取到-1,证明文件读取完毕

代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileReader;  import java.io.IOException;  public class Demo02\_FileReader {  public static void main(String[] args) throws IOException{  // 1. 创建出一个字符输入流, 绑定一个数据源  FileReader fr = new FileReader("chinese.txt");  // 2. 使用单个字符读取文件内容  // len表示读取到的字符在编码表中对应的整数结果  /\*int len;  while((len = fr.read()) != -1 ) {  System.out.print((char)len);  }\*/    // 3. 使用字符数组读取文件内容  char[] ch = new char[3];  // len表示读取到的字符的个数  int len;  while((len = fr.read(ch)) != -1) {  System.out.print(new String(ch, 0, len));  }  fr.close();  }  } |

### 字符输出流

1. Writer : 字符输出流的抽象父类, 来自于java.io包
2. 抽象类不能实例化对象, 因此需要子类FileWriter, 来自于java.io包. 主要功能就是以字符方式向文件中写入数据
3. FileWriter构造方法:
4. FileWriter(String fileName)
5. FileWriter(File file)
6. 向文件中写入数据方式: write方法返回值类型 void
7. write(int c) : 表示每次写个一个字符
8. write(char[] ch) : 表示每次可以将ch数组中的所有字符内容写入到文件中
9. write(char[] ch , int index , int length) : 表示每次将ch数组中的一部分写入到文件中; 从index索引位置开始写, 写入length个字符
10. write(String s): 将字符串s写入到文件中
11. write(String s, int index, int length) : 表示将字符串s从index索引位置开始, 写入length个字符到文件中

代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.File;  import java.io.FileWriter;  import java.io.IOException;  public class Demo03\_FileWriter {  public static void main(String[] args) throws IOException{  File f = new File("abc.txt");  if(!f.exists()) {  f.createNewFile();  }    // 1. 创建出一个字符输出流, 绑定一个数据目的  FileWriter fw = new FileWriter(f);  //A1a&中a&我爱我家,天气真好我家,天  // 2. 向文件中写入数据  // 写入单个字符  fw.write('A');  char[] ch = {'1','a','&','中'};  // 写入字符数组  fw.write(ch);  // 写入字符数组一部分  fw.write(ch, 1, 2);  String s = "我爱我家,天气真好";  // 写入字符串  fw.write(s);  // 写入字符串一部分  fw.write(s, 2, 4);  // 3. 一定要关闭资源  fw.close();  }  } |

### flush和close方法的比较

IO流资源中的数组缓冲 : 带有数组缓冲流资源在读写效能上更高, 但是举例 : 向文件中写入内容; 写的数据都是自动存储在底层数组缓冲区域中, 如果缓冲区写满, 自动同步到文件中; 或者通过flush或者close方法让缓冲区中的数据同步到文件中

1. flush() : 表示刷新, 将当前正在使用的IO流资源在底层缓冲区中的数据同步到文件中, flush刷新之后的流资源还可以继续使用
2. close() : 表示关闭资源, 当调用close方法关闭资源之前, 先调用flush功能, 将缓冲区中数据同步到文件中, 同步数据结束, 再关闭流资源和相关系统资源; close关闭资源之后,流资源不能再继续使用

注意 : 使用IO流, 一定必须要关闭资源

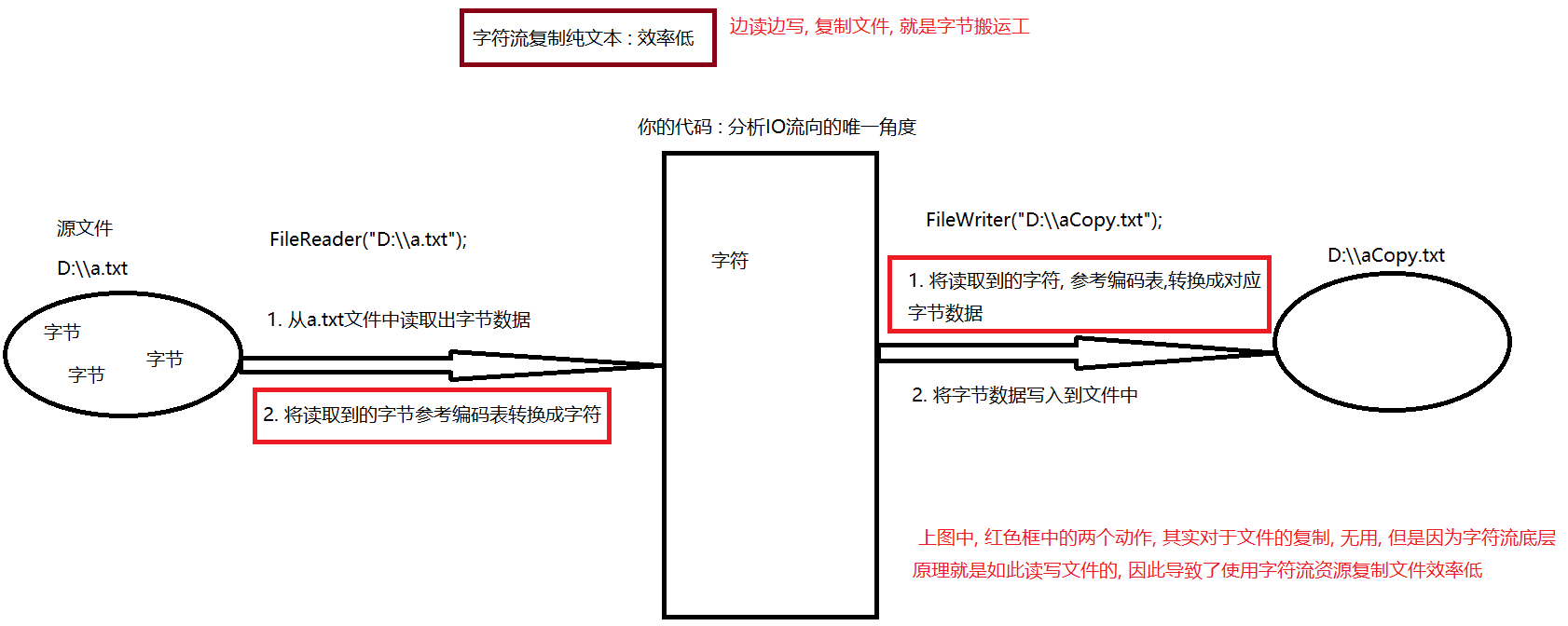
代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.File;  import java.io.FileWriter;  import java.io.IOException;  public class Demo03\_FileWriter {  public static void main(String[] args) throws IOException{  File f = new File("abc.txt");  if(!f.exists()) {  f.createNewFile();  }    // 1. 创建出一个字符输出流, 绑定一个数据目的  FileWriter fw = new FileWriter(f);  //A1a&中a&我爱我家,天气真好我家,天  // 2. 向文件中写入数据  // 写入单个字符  fw.write('A');  char[] ch = {'1','a','&','中'};  // 写入字符数组  fw.write(ch);  // 写入字符数组一部分  fw.write(ch, 1, 2);  // 刷新数组缓冲区中数据到文件中  fw.flush();  String s = "我爱我家,天气真好";  // 写入字符串  fw.write(s);  // 写入字符串一部分  fw.write(s, 2, 4);  // 3. 一定要关闭资源  fw.close();  // 关闭流资源之后, 流不能再继续使用, 否则代码会报错 : IOException  // fw.write("helloworld");  }  } |

### **字符流复制纯文本文件**

结论 :

1. 如果使用字符流复制文件, 只能复制纯文本文件(使用txt记事本形式可以打开, 还能读懂的文件, 称为纯文本文件)
2. 复制过程效率不高(效率低)

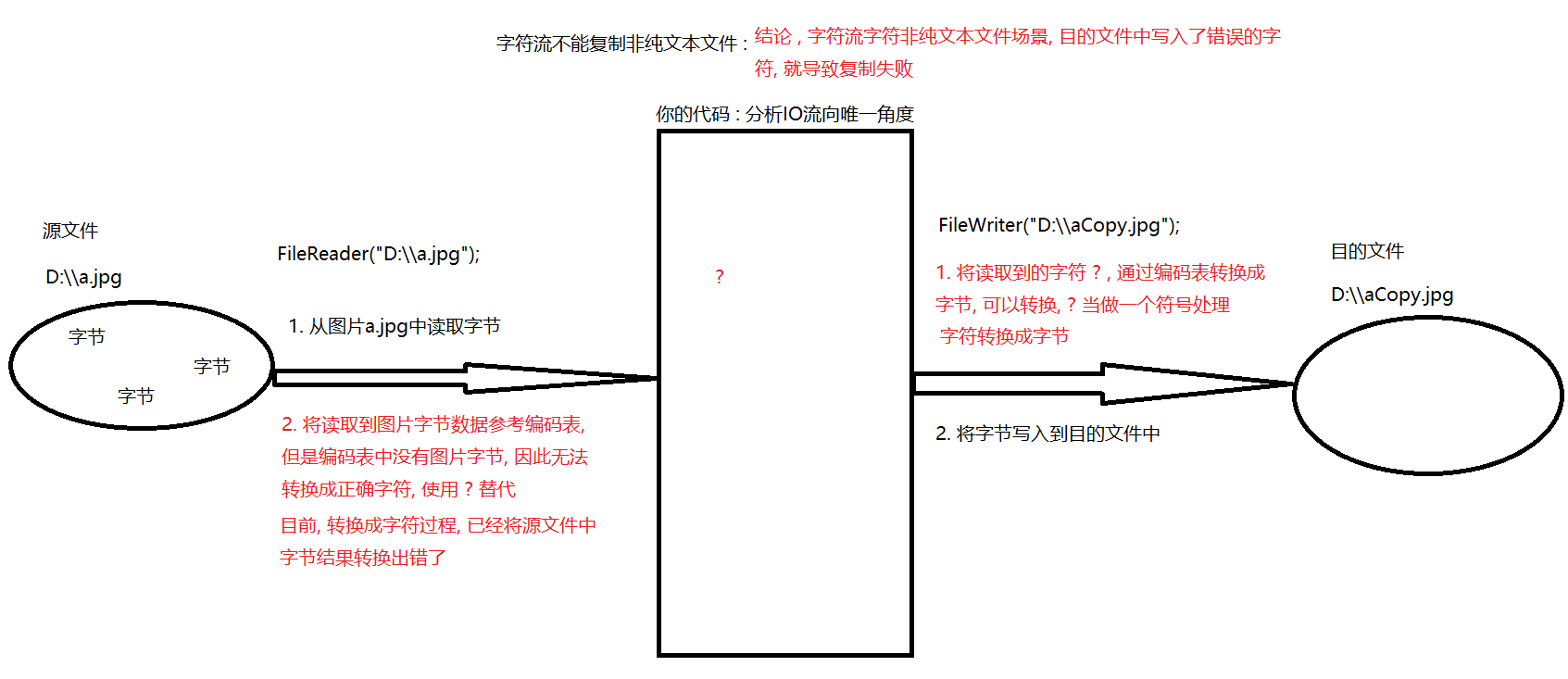


代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class Demo04\_字符串Copy {  public static void main(String[] args) throws IOException{  // 1. 创建出一个字符输入流, 读取文件 , 绑定一个数据源  FileReader fr = new FileReader("abc.txt");  // 2. 创建出一个字符输出流, 写入文件, 绑定一个数据目的  FileWriter fw = new FileWriter("abcCopy.txt");  // len表示每次读取到的字符的个数  int len;  char[] ch = new char[10];  while((len = fr.read(ch)) != -1) {  fw.write(ch, 0, len);  }    fw.close();  fr.close();  }  } |

### **字符流不能复制非纯文本文件**

**非纯文本文件中字节, 例如 : 图片,音频,视频...在编码表中不存在, 也没有办法转换成正确字符, 导致文件复制过程中, 字节转换成字符的过程出错; 将错误的字符转换成字节, 写入到目的文件中, 目的文件中字节与源文件已经不同, 导致复制失败**



代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class Demo05\_字符流复制非纯文本失败 {  public static void main(String[] args) throws IOException{  FileReader fr = new FileReader("D:\\1012系统班\\day06\\图解\\两个对象在内存中理解.png");  FileWriter fw = new FileWriter("D:\\对象Copy.png");    int len;  char[] ch = new char[1024];  while((len = fr.read(ch)) != -1) {  fw.write(ch, 0, len);  }  fw.close();  fr.close();  }  } |

### 高效缓冲字符流

1. 高效缓冲字符流 : 作用就是可以提高字符流读写效率, 将字符数组作为读写的缓冲, 从而减少与磁盘交互, 进而提高读写效率
2. BufferedReader: 是Reader的子类, 称为高效缓冲字符输入流, 本身是一个包装类, 将一个最普通字符输入流,包装成一个高效字符输入流

从构造方法封装可以看出包装性 : **[BufferedReader](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/BufferedReader.html" \l "BufferedReader(java.io.Reader))**([Reader](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/Reader.html" \o "java.io 中的类) in)

1. BufferedWriter: 是Writer的子类, 称为高效缓冲字符输出流, 本身是一个包装类, 将一个最普通字符输出流,包装成一个高效字符输出流

从构造方法封装可以看出包装性 : **[BufferedWriter](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/BufferedWriter.html" \l "BufferedWriter(java.io.Writer))**([Writer](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/Writer.html" \o "java.io 中的类) out)

1. 简单描述高效缓冲流的高效原理:
2. BufferedReader, 当创建出一个BufferedReader类型流资源对象时, 代码底层自动创建出一个大小为8192(1024\*8)的字符数组, 每次调用read方法读取文件, 都最多读取出8129个字符放置到底层字符数组中, 以后读取就从底层数组中读取, 如果8192读取完毕,继续从文件中最多读8192个,以此类推, 直到读取到-1,文件结束为止; 如此减少与磁盘的交互, 提高读取效率
3. BufferedWriter, 当创建出一个BufferedWriter流资源对象时, 代码底层自动创建出一个大小为8192(1024\*8)的字符数组, 每次向文件中写入数据, 都是先写入到底层数组中, 如果8192写满了, 自动将数组中的内容同步到文件中, 如果8192没有写满, 可以使用flush或者close方法将缓冲区中数据同步到文件中; 如此减少与磁盘的交互, 提高读取效率

代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class Demo06\_BufferedStream {  public static void main(String[] args) throws IOException{  BufferedReader br = new BufferedReader(  new FileReader("chinese.txt"));    BufferedWriter bw = new BufferedWriter(  new FileWriter("chineseCopy.txt"));    int len;  while((len = br.read()) != -1) {  bw.write(len);  }    br.close();  bw.close();  }  } |

### **高效缓冲字符流特有方法**

1. BufferedReader : 中有特殊方法功能

readLine() : 每次从文件中读取出一行文字, 返回值类型String类型; 如果返回的结果为null, 证明文件读取完毕

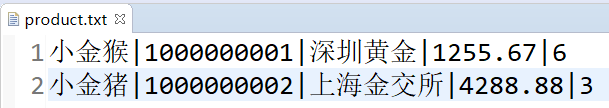
1. BufferedWriter : 中有特殊方法功能

newLine() : 在文件中写入一个换行符, 不受操作系统限制

案例1 : 复制具有指定格式的代码文件

|  |
| --- |
| package com.ujiuye.io;  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class Demo07\_字符缓冲流特有方法 {  public static void main(String[] args) throws IOException{  BufferedReader br = new BufferedReader(  new FileReader("javaCode.txt"));    BufferedWriter bw = new BufferedWriter(  new FileWriter("javaCodeCopy.txt"));    // s表示每次读取到的一行内容  String s;  while((s = br.readLine()) != null) {  bw.write(s);  // 向文件中写入回车换行  bw.newLine();  }  bw.close();  br.close();  }  } |

案例2 : 文件内容如下:



读取文件内容, 获取出每一个产品的每一部分数据信息

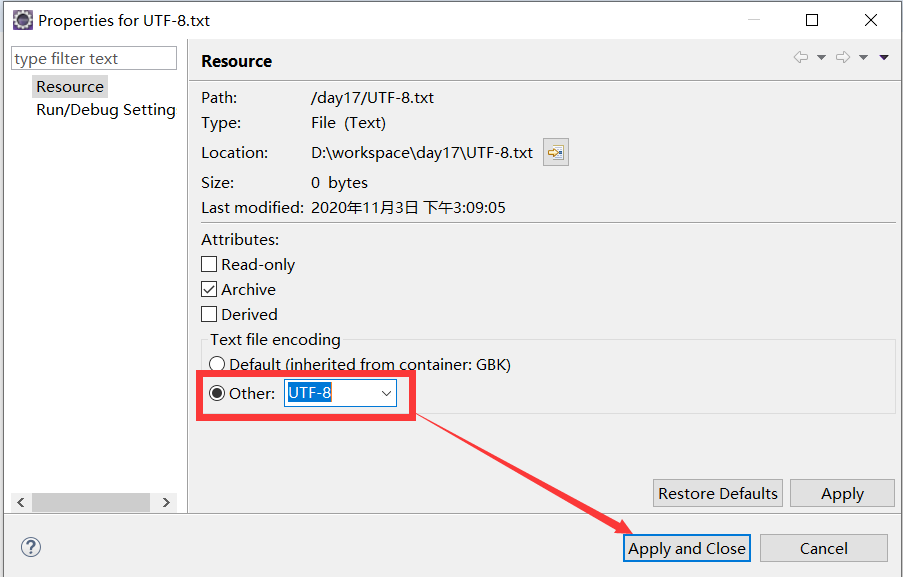
代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.util.Arrays;  public class Demo08\_高效缓冲字符流案例 {  public static void main(String[] args) throws IOException{  BufferedReader br = new BufferedReader(  new FileReader("product.txt"));    String s;  while((s = br.readLine()) != null) {  String[] sPro = s.split("\\|");  System.out.println(Arrays.toString(sPro));  }  br.close();  }  } |

## 转换流

1. 编码表 :
2. GBK : 中国标准码表, 底层兼容ASCII编码, 一个中文占有2字节大小
3. UTF-8 : 万国码表, 底层兼容ASCII编码, 一个中文占有3字节大小
4. 读写带有中文文本文件时, 发生乱码情况一般有两种:
5. 流资源选择不正确, 例如字节流读取中文, 可能将中文拆开读取, 导致乱码, 改成字符流读取
6. 读写文件时, 因为文件的编码集不同, 也会导致读写文件是发生乱码问题, 使用转换流解决不同编码集文件读写乱码问题
7. 将文件编码集进行修改 : 如下图

点击指定需要修改编码集的文件--->鼠标右键--->properties(属性)



1. 转换流 : 主要功能就是可以根据文件指定的编码集读写文件
2. InputStreamReader : 字节到字符的桥梁, 父类Reader, 字符输入流

构造方法 : **[InputStreamReader](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/InputStreamReader.html" \l "InputStreamReader(java.io.InputStream, java.lang.String))**([InputStream](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/InputStream.html" \o "java.io 中的类) in, [String](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/lang/String.html" \o "java.lang 中的类) charsetName)

a : 第一个参数 in , 表示一个字节输入流, 从文件中读取出字节数据

b : 将读取到的字节通过第二个参数charsetName编码集, 将字节通过指定编码集转换成字符

1. OutputStreamWriter : 字符到字节桥梁, 父类Writer, 字符输出流

构造方法 :

**[OutputStreamWriter](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/OutputStreamWriter.html" \l "OutputStreamWriter(java.io.OutputStream, java.lang.String))**([OutputStream](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/OutputStream.html" \o "java.io 中的类) out, [String](mk:@MSITStore:C:\\Users\\96243\\Desktop\\JDK_API_1_6_zh_CN.CHM::/java/io/../../java/lang/String.html" \o "java.lang 中的类) charsetName)

a : 将文件中读取到的字符, 参考第二个参数charsetName, 参考指定编码表转换成字节数据

b : 将字节使用第一个参数out写入到文件中

注意 : 使用转换流时, 给出编码集与要操作的文件的编码集一致

举例 : 例如需要读取的文件编码是UTF-8, 转换流中给出的编码集就是UTF-8

例如需要写入的文件的编码是GBK, 转换流中给出的编码集就是GBK

代码

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.InputStreamReader;  import java.io.OutputStreamWriter;  public class Demo10\_转换流 {  public static void main(String[] args) throws IOException{  // 1. 创建出一个输入流, 读取文件  InputStreamReader isr = new InputStreamReader(  new FileInputStream("UTF-8.txt"), "UTF-8");    // 2. 创建出一个输出流, 向文件中写入数据  OutputStreamWriter osw = new OutputStreamWriter(  new FileOutputStream("GBK.txt"), "GBK");    int len;  while((len = isr.read()) != -1) {  osw.write(len);  }    osw.close();  isr.close();  }  } |

## IO流的标准异常处理方式(扩展)

1. 标准IO流异常处理方式很繁琐,但是很安全!
2. 从JDK1.7版本开始, 为IO流资源使用提供新的异常处理语法结构:

try(

IO流资源创建写入在try的小括号中;

){

// 可能会发生异常的代码, 主要IO流资源的读写使用

}catch(异常类型 变量名){

// 对于异常处理方式

}

1. try代码块中的小括号作用 : 会在流资源使用结束后, 自动关闭资源; 因此新的语法结构中, IO流资源不需要手动关闭

JDK7之前, IO流标准异常处理方式

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  public class Demo11\_IO流标准处理方式 {  public static void main(String[] args) {  FileInputStream fis = null;  FileOutputStream fos = null;  try {  fis = new FileInputStream("abc.txt");  fos = new FileOutputStream("abcCopy.txt");    int len;  while((len = fis.read()) != -1) {  fos.write(len);  }  }catch(IOException ex) {  ex.printStackTrace();  }finally {  try {  if(fos != null) {  fos.close();  }  }catch(IOException ex) {  ex.printStackTrace();  }finally {  try {  if(fis != null) {  fis.close();  }  }catch (IOException e) {  e.printStackTrace();  }  }  }  }  } |

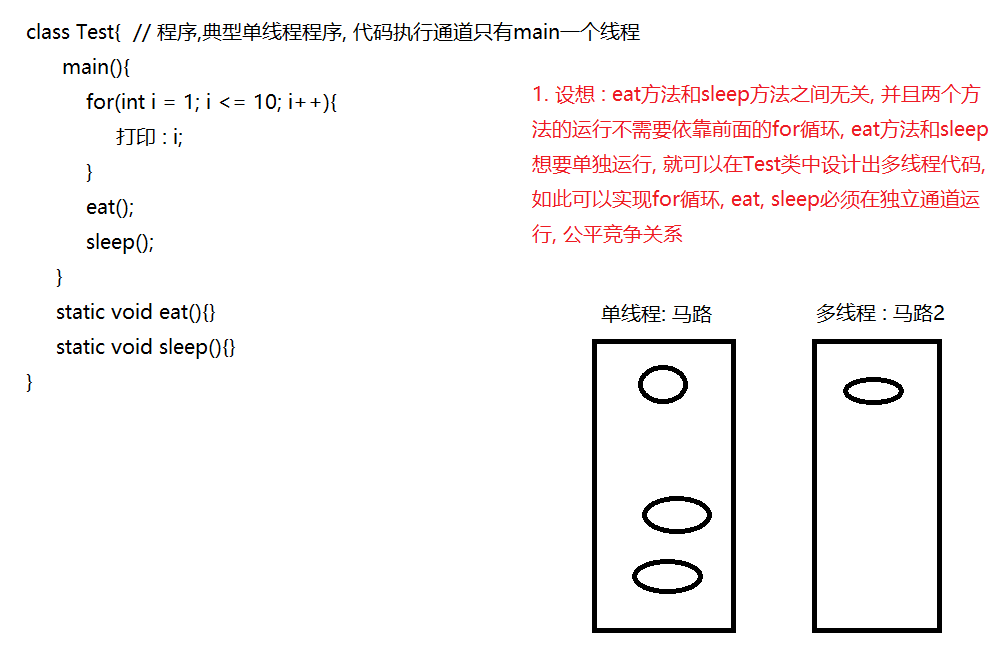
JDK7对于IO流新异常处理方式

|  |
| --- |
| package com.ujiuye.io;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  public class Demo12\_JDK7IO流资源新异常处理 {  public static void main(String[] args) {  try(  FileInputStream fis = new FileInputStream("abc.txt");  FileOutputStream fos = new FileOutputStream("abcCopy.txt");  ){  int len;  while((len = fis.read()) != -1) {  fos.write(len);  }    }catch(IOException ex) {  ex.printStackTrace();  }  }  } |

## 多线程

### 多线程的相关概念

1. 程序 : 表示固定逻辑和数据的集合, 举例 : 每一天写出的.java文件就是程序代码
2. 进程 : 在内存中正在运行的程序, 称为进程
3. 线程 : 进程中代码的执行通道, 称为线程; 如果代码中只有一条代码执行通道, 那么称程序为单线程程序; 如果代码中有多条代码执行通道, 那么程序称为多线程程序; 线程通道之间彼此独立运行



1. 并行 : 同一个时间, 多个程序都要求运行, 要求设备硬件很好, 例如 : 有多个CPU, 多核...真的可以做到让多个程序同时都在运行

举例 : 饭店, 需要炒出5盘菜, 于是5个厨师, 每一个厨师炒一道菜

1. 并发 : 同一个时间, 多个程序都要求运行,只有一个CPU, 于是CPU在多个程序之间来回切换执行, 但是CPU处理速度非常快, 基本感觉不到多个程序在轮流执行

举例 : 饭店, 需要炒出5盘菜, 但是只有一个厨师, 1和人炒5道菜, 在5道菜 之间来回切换处理

注意 : 多线程程序都是并发运行机制, 因此最终哪个线程运行的多, 优先结束, 都不一定, 因为并发执行具有很大随机性

### 多线程的第一种实现方式

1. 继承Thread类 : Thread类表示一个线程, 就是一个线程类



1. 多线程实现步骤:
2. 自定义出一个类, 成为Thread类的子类, 于是自定义类属于线程类
3. 在自定义线程类中, 重写从父类Thread中继承到的run方法功能, 将需要独立运行的代码逻辑写到run方法中
4. 创建出一个自定义的线程类对象, 表示准备有一个独立代码执行通道
5. 使用线程类对象, 调用start方法, 开启一个线程 :



代码

|  |
| --- |
| package com.ujiuye.thread;  // 1. 自定义出一个类, 称为Thread类的子类, 于是自定义类属于线程类  public class MyThread extends Thread {  // 2) 在自定义线程类中, 重写从父类Thread中继承到来的run方法功能,  // 将需要独立运行的代码逻辑写到run方法中  @Override  public void run() {  for(int i = 1; i <= 10; i++) {  System.out.println("run----" + i);  }  }  } |

|  |
| --- |
| package com.ujiuye.thread;  public class TestThread {  public static void main(String[] args) {  // 1. main方法本身就是一个线程  // 2. 自定出一个额外线程通道, 执行独立代码  // 3)创建出一个自定义的线程类对象, 表示准备有一个独立代码执行通道  MyThread my = new MyThread();  my.start();    for(int i = 1; i <= 10; i++) {  System.out.println("main---" + i);  }  }  } |