# IO流

IO技术主要的作用是解决设备与设备之间 的数据传输问题。 比如：

硬盘--->内存，把键盘的数据------->内存中，内存的数据---->硬盘上，

引用场景：导出报表 ， 上传大头照 、 下载 、 解释xml文件 ...

## File

File类可以描述一个文件或者一个文件夹

构造方法

1、File(String pathname) 指定文件或者文件夹的路径创建一个File文件。

2、File(File parent, String child) 根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。

用途：如在window系统下的盘符当父File：

File file = new File("d:\\"); //复盘

File file2 = new File(file,"a.txt"); //就不用指定d盘了

if(!file2.exists()) {

file2.createNewFile();

}

3、File(String parent, String child)

一般用2种构造方法：父路径可以先做预处理动作：如先删除d盘没用的，第3种不怎么用

参数要注意问题

细节一：目录分隔符 此类有静态域：File.separator

使用File.separator：在windows机器上的目录分隔符是 \ ,在linux机器上的目录分隔符是/

注意：在windows上面\ 与 / 都可以使用作为目录分隔符。 而且，如果写/ 的时候只需要写一个即可。

细节二：路径

绝对路径: 该文件在硬盘上 的完整路径。绝对路径一般都是以盘符开头的。

相对路径: 相对路径就是资源文件相对于当前程序所在的路径。

如果不知道相对路径的所在路径：老师教了一个方法：

//第一步：在此参数中放入一个点

File file = new File(".");

//第二步使用：file.getAbsolutePath()打印出来

System.out.println("当前路径是："+ file.getAbsolutePath());打印：I: \eclipsev4.7\eclipse \project\MyJavaCode\_1.7

### 创建

共4个方法

1、createNewFile() 在指定位置创建一个空文件，成功就返回true，如果已存在就不创建然后返回false

File file = new File("F:\\aa");

System.out.println("创建成功了吗？"+file.createNewFile());

2、mkdir() 在指定位置创建目录，这只会创建最后一级目录，如果上级目录不存在就抛异常。

//使用 mkdir 创建一个单级文件夹，注意只能创建单级文件夹

File dir = new File("F:\\a.txt");

System.out.println("创建文件夹成功吗？"+dir.mkdir());

3、mkdirs() 在指定位置创建目录，这会创建路径中所有不存在的目录。

//使用 mkdirs 创建多级文件夹，下面demo会创建aa文件夹和bb文件夹

dir = new File("F:\\aa\\bb");

System.out.println("创建多级文件夹："+ dir.mkdirs());

4、renameTo(File dest) 重命名文件或文件夹，也可以操作非空的文件夹，文件不同时相当于文件的剪切,剪切时候不能操作非空的文件夹。移动/重命名成功则返回true，失败则返回false。

//重命名文件名，也可以重命名文件夹名：路径相同

File file = new File("d:\\a.txt");

File file2 = new File("d:\\cccc");

file.renameTo(file2);//要注意需要被重命名的file进行调用方法

//只能移动(剪切)文件不能移动文件夹：路径不相同

File file = new File("d:\\a.txt");

File file2 = new File("e:\\cccc");

file.renameTo(file2);//要注意需要被重命名的file进行调用方法

### 判断

共5个方法

1、exists() 文件或文件夹是否存在。存在返回true

File file = new File("..\\..\\a.txt");

System.out.println("存在吗？"+ file.exists());

2、isFile() 是否是一个文件，如果不存在，则始终为false。

3、isDirectory() 是否是一个目录，如果不存在，则始终为false。

4、isHidden() 是否是一个隐藏的文件或是否是隐藏的目录。

5、isAbsolute() 测试此抽象路径名是否为绝对路径名。

### 删除

共2个方法

1、delete() 删除文件或一个空文件夹，如果是文件夹不为空，不能删除，成功true，失败false。

2、deleteOnExit() 在虚拟机终止时，请求删除此抽象路径名表示的文件或目录，保证程序异常时创建的临时文件也可以被删除

File file = new File("F:\\a.txt");

file.deleteOnExit(); //jvm退出的时候删除文件。 一般用于删除临时 文件。

### 获取

共6个方法

1、getName() 获取文件或文件夹的名称，不包含上级路径。

2、getPath() 返回绝对路径，可以是相对路径，但是目录要指定

3、getAbsolutePath() 获取文件的绝对路径

4、length() 获取文件的大小（字节数），如果文件不存在则返回0L，如果是文件夹也返回0L。

5、getParent() 返回父目录的路径名字符串；如果此路径名没有指定父目录，则返回null。

6、lastModified() 获取最后一次被修改的时间。可用来判断此文件是否被修改

long lastModified = file.lastModified();

Date date = new Date(lastModified);

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");

System.out.println("获取最后一次的修改时间(毫秒值)："+ dateFormat.format(date) );

7、staic File[] listRoots() 列出所有的根目录（Window中就是所有系统的盘符）

File[] roots = File.listRoots(); //列出所有的根目录

for(File file : roots){

System.out.println(file);

}

8、list() 返回目录下的文件或者目录名，包含隐藏文件。指定到文件不是目录会返回null。

//把 当前文件夹下面的所有子文件名与子文件夹名 存储到一个String类型 的数组中 返回。

File file = new File("F:\\ ");

String[] fileNames = file.list();

for(String fileName : fileNames){

System.out.println(fileName);

}

9、listFiles() 返回目录下的文件或者目录对象，包含隐藏文件。指定到文件不是目录会返回null。

// 返回file对象：存储到file数组中

File file = new File("F:\\ ");

File[] files = file.listFiles();

for(File fileItem : files){

System.out.println("文件名："+ fileItem.getName());

}

10、listFiles(FilenameFilter filter) 目录中符合过滤条件的子文件或子目录。指定文件不是目录会返回null。

//获取a目录中的后缀名为java的文件

File dir = new File("d:\\a");

File[] files = dir.listFiles(new FilenameFilter() { //第一步 使用匿名内部类

public boolean accept(File dir, String name) { //第二步 接口中需要被实现的方法

//System.out.println("文件夹:"+dir+" 文件名："+ name);

return name.endsWith(".java"); //第三步 使用字符串的方法判断是否为java

}

});

for(File file : files){ //第四步 fareace 得到a目录下的所有java文件

System.out.println(file.getName());

}

11、list(fileNameFilter filter) 目录中符合过滤条件的子文件或子目录。指定到文件不是目录会返回null。

### 例子

例子1：指定一个文件夹，获取该文件夹下面所有java文件。

这边以根目录开头操作(当然你可以直接指定到文件夹目录下操作更方便)：

File file = new File("d:\\");

File[] files = file.listFiles();第一步 获取盘目录下所有的子文件和目录

for (File f : files) {

if(f.getName().equals("a")) { 第二步：判断目录名称是否为a(建议还要判断是否为目录)

File[] listFiles = f.listFiles(); 第三步 获取a目录下的所有子文件和目录

for (File f2 : listFiles) {

if(f2.getName().endsWith("java")) { 第四步 使用字符串方法判断后缀名为java的文件

System.out.println(f2.getName());

}

}

}

}

需求2： 指定一个文件夹，然后列出文件夹下面的所有子文件与文件夹，但是格式要如下:

文件：

文件名1

文件名2

文件名3

..

文件夹：

文件夹名1

文件夹名2

文件夹名3

....

File dir = new File("d:\\");

File[] files = dir.listFiles();//获取到所有的子文件

System.out.println("文件：");

for(File fileItem : files){

if(fileItem.isFile()){

System.out.println("\t"+fileItem.getName());

}

}

System.out.println("文件夹：");

for(File fileItem : files){

if(fileItem.isDirectory()){

System.out.println("\t"+fileItem.getName());

}

}

}

## 字节流

字节流: 字节流读取得都是文件中二进制数据，读取到二进制数据不会经过任何的处理。

FileInputStream 输入流

InputStream 所有输入字节流的父类(基类) 它是抽象类

FileInputStream 读取文件数据的输入字节流

FileOutputStream 输出流

OutputStream 是所有输出字节流 的父类。 抽象类

FileOutStream 向文件输出数据的输出字节流。

拷贝一张图片

第一种 使用普通方式

//第一步 找到目标文件

File inFile = new File("F:\\美女\\1.jpg");

File destFile = new File("E:\\1.jpg");

//第二步 建立数据的输入输出通道

FileInputStream input = new FileInputStream(inFile);

FileOutputStream out = new FileOutputStream(destFile); //追加数据....

//每新创建一个FileOutputStream的时候，默认情况下FileOutputStream 的指针是指向了文件的开始的位置。 每写出一次，指向都会出现相应移动。

//第三步 建立缓冲数据，边读边写

byte[] buf = new byte[1024];

int length = 0 ; //获取数组的个数：初始为0，如第一次循环是存满的就是1024，最后一次824就是824

while((length = input.read(buf))!=-1){ //最后一次只剩下了824个字节

//参数：字节数组，数组开始位置，数组结束位置

out.write(buf,0,length); //写出很多次数据，所以就必须要追加。

}

//第四步 关闭资源 原则： 先开后关，后开先关。

input.close();

out.close();

第二种 使用缓冲流方式

//第一步 找到目标文件

File inFile = new File("F:\\美女\\1.jpg");

File outFile = new File("E:\\1.jpg");

//第二步 建立数据输入输出通道

FileInputStream fileInputStream = new FileInputStream(inFile);

FileOutputStream fileOutputStream = new FileOutputStream(outFile);

//第三步 建立缓冲输入输出流：内部都维护了8kb的字节数组

BufferedInputStream bufferedInput = new BufferedInputStream(fileInputStream);

BufferedOutputStream bufferedOutput = new BufferedOutputStream(fileOutputStream);

int content = 0;

//第四步 循环边读边写：最后返回值为-1证明已经读完

while((content = bufferedInput.read())!=-1){ // 如果使用read方法中无参数(没传入缓冲数组)，那么返回值是读取到的内容。

bufferedOutput.write(content);

}

//第5步：关闭资源

bufferedInput.close();

bufferedOutput.close();//先flush后关闭，所有循环中不用flush，不然效率很低

}

注意细节(如果忘记仔细看上面两种方法)：

read(byte): 内容是存储到缓冲数组中，返回值是存储到缓冲数组中的字节个数。

read();如果使用read方法中无参数(没传入缓冲数组)，那么返回值是读取到的内容.所以缓冲：直接读内容

记住：缓冲流都不具备读写文件的能力构造方法中需要传入FileInputStream

## 字符流

前提引申：使用字节流也可以读取txt内容

字节流之所以能够写中文是因为借助了字符串的getBytes方法对字符串进行了编码（字符---->数字）

使用字节流出现乱码的原因： 一个中文在gbk码表中默认是占两个字节，它只是一个字节一个字节的读所以不是一个完整的中文。

使用字符流的应用场景： 如果是读写字符数据的时候则使用字符流。

字符流会把读取到的二进制的数据进行对应 的编码与解码工作。 字符流 = 字节流 + 编码(解码)

FileReader 输入字符流 一个字符的读

reader 输入字符流的基类。 抽象类

FileReader 向文件数据数据的输入字符流

//找到目标文件

File file = new File("F:\\1208project\\day21\\src\\day21\\Demo1.java");

// 建立数据的输入通道

FileReader fileReader = new FileReader(file);

//建立缓冲字符数组读取文件数据

char[] buf = new char[1024];

int length = 0 ;

while((length = fileReader.read(buf))!=-1){

System.out.print(new String(buf,0,length));

}

FileWriter 输出字符流

reader 输出字符流的基类。 抽象类

FileReader 向文件数据数据的输出字符流

要注意的事项：

1. 使用FileWriter写数据的时候，它爷爷writer内部是维护了一个1024个字符数组的，写数据的时候会先写入到它内部维护的字符数组中，如果需要把数据真正写到硬盘上：需要调用flush或者是close方法或者是填满了内部的字符数组。

2. 使用FileWriter的时候，如果目标文件不存在，那么会自动创建目标文件。

3．使用FileWriter的时候， 如果目标文件已经存在了，那么默认情况会先清空文件中的数据，然后再写入数据 ， 如果需要在原来的基础上追加数据，需要使用“new FileWriter(File , boolean)”的构造方法，第二参数为true。

//找到目标文件

File file = new File("F:\\a.txt");

//建立数据输出通道

FileWriter fileWriter = new FileWriter(file,true);

//准备数据，把数据写出

String data = "今天天气非常好！！";

fileWriter.write(data); //字符流具备解码的功能。

//刷新字符流

// fileWriter.flush();

//关闭资源

fileWriter.close();

拷贝txt文件

第一种 使用缓冲字符流操作 要注意”\r\n”问题:\r:13,\n:10

1. 输入输出内部维护了8196Kb的char数组
2. 缓冲输入流：读一行字符数据readLine();方法
3. 缓冲输出流：写入分隔符”\r\n”的功能newLine()方法

//第一步 创建缓冲字符输出、输入流

BufferedReader bufferedReader = new BufferedReader(new FileReader("F:\\Test.txt"));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("E:\\Test.txt"));

//第二步 使用String接收内容，因为内部维护了char数组， String默认值是null

String line=null;

while((line = bufferedReader.readLine())!=null){ //读取一行

bufferedWriter.write(line);

}

//第三步 关闭流

bufferedWriter.close();

bufferedReader.close();

第二种 普通方式( 一般都不会用：和字节流普通方式差不多)

//找到目标文件

File file = new File("F:\\1208project\\day21\\src\\day21\\Demo1.java");

File file2 = new File("F:\\1208project\\day21\\src\\day21\\Demo1.java");

// 建立数据的输入通道

FileReader fileReader = new FileReader(file);

FileWrite fileWrite= new FileWrite (file2);

//建立缓冲字符数组读取文件数据

char[] buf = new char[1024];

int length = 0 ;

while((length = fileReader.read(buf))!=-1){

fileWrite .write(buf,0,length));

}

注意问题：

记住：缓冲流都不具备读写文件的能力

拓展FileReader和FileWriter

装饰者设计模式：增强一个类的功能，而且还可以让这些装饰类互相装饰。

使用步骤：继承BufferedReader缓冲类，重写readLine的方法

需求1： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有行号。

class BufferedLineNum extends BufferedReader{

//第一步 默认行号为1

int count = 1 ;

//第二步 创建带参构造函数

public BufferedLineNum(Reader in) {

super(in);

}

第二步 重写readLine()

public String readLine() throws IOException {

String line = super.readLine(); //首先获得父类的方法返回内容

if(line ==null){

return null;

}

line = count+" "+ line; //在内容最前面加上行号即可

count++;

return line;

}

}

调用：

File file = new File("F:\\Demo1.java");

//建立数据的输入通道

FileReader fileReader = new FileReader(file);

//建立带行号的缓冲输入字符流

BufferedLineNum bufferedLineNum = new BufferedLineNum(fileReader);

String line = null;

while((line = bufferedQuto.readLine())!=null){

System.out.println(line);

}

这里只写一个列子：详细请看2015基础33期第21天的课程

## 打印流

打印流（printStream）打印任意类型的数据，而且打印数据之前都会先把数据转换成字符串再进行打印。

指定标准的输出流显示

//打印流可以打印任何类型的数据，而且打印数据之前都会先把数据转换成字符串再进行打印。

File file = new File("F:\\a.txt");

//第一步 创建一个打印流

PrintStream printStream = new PrintStream(file);

//第二步 重新设置了标准的输出流对象(默认标准的输出流就是向控制台输出的)

System.setOut(printStream);

System.out.println("哈哈，猜猜我在哪里！！");

收集异常日志信息

File logFile = new File("F:\\2015年1月8日.log");

//第一步 创建一个打印流：追加内容需要用到字节流的构造方法设为true

PrintStream logPrintStream = new PrintStream( new FileOutputStream(logFile,true) );

try{

int c = 4/0;

System.out.println("c="+c);

int[] arr = null;

System.out.println(arr.length);

}catch(Exception e){

e.printStackTrace(logPrintStream); //第二步 把异常信息发送到打印流中

}

## 转换流

输入字节流的转换流：InputStreamReader 是字节流通向字符流的桥

输出字节流的转换流： OutputStreamWriter可以把输出字节流转换成输出字符流 。

转换流的作用：

1.获取到的是一个字节流需要转换字符流使用，这时候就可以使用转换流。 字节流----> 字符流

(1) 使用字符缓冲流接收字节输入流来写入数据的操作

InputStream in = System.in; //获取了标准的输入流。

//System.out.println("读到 的字符："+ (char)in.read()); //read()一次只能读取一个字节。

//第一步 需要把字节流转换成字符流。

InputStreamReader inputStreamReader = new InputStreamReader(in);

//第二步 使用字符流的缓冲类

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String line = null;

while((line = bufferedReader.readLine())!=null){

System.out.println("内容："+ line);

}

2. 使用转换流可以指定编码表进行读写文件。

(1)使用输入字节流的转换流指定码表进行读取文件数据

public static void readTest2() throws IOException{

File file = new File("F:\\a.txt");

FileInputStream fileInputStream = new FileInputStream(file);

//创建字节流的转换流并且指定码表进行读取

InputStreamReader inputStreamReader = new InputStreamReader(fileInputStream,"utf-8");

char[] buf = new char[1024];

int length = 0;

while((length = inputStreamReader.read(buf))!=-1){

System.out.println(new String(buf,0,length));

}

}

(2)使用输出字节流的转换流指定码表写出数据

public static void writeTest2() throws IOException{

File file = new File("F:\\a.txt");

//建立数据的输出通道

FileOutputStream fileOutputStream = new FileOutputStream(file);

//把输出字节流转换成字符流并且指定编码表。

OutputStreamWriter outputStreamWriter = new OutputStreamWriter(fileOutputStream, "utf-8");

outputStreamWriter.write("新中国好啊");

//关闭资源

outputStreamWriter.close();

}

## 对象流

对象的输入输出流 : 对象的输入输出流 主要的作用是用于写对象的信息与读取对象的信息。 对象信息一旦写到文件上那么对象的信息就可以做到持久化了

对象的输出流： ObjectOutputStream .

对象的输入流: ObjectInputStream

对象输入输出流要注意的细节：

1. 如果对象需要被写出到文件上，那么对象所属的类必须要实现Serializable接口。

2. 对象的反序列化创建对象的时候并不会调用到构造方法、

3. serialVersionUID 是用于记录class文件的版本信息的，serialVersionUID这个数字是通过一个类的类名、成员、包名、工程名算出的一个数字。

4. 使用ObjectInputStream反序列化的时候，ObjeectInputStream会先读取文件中的serialVersionUID，然后与本地的class文件的serialVersionUID

进行对比，如果这两个id不一致，那么反序列化就失败。

5. 如果序列化与反序列化的时候可能会修改类的成员，那么最好一开始就给这个类指定一个serialVersionUID，如果一类已经指定的serialVersionUID，然后在序列化与反序列化的时候，jvm都不会再自己算这个 class的serialVersionUID了。

6. 如果一个对象某个数据不想被序列化到硬盘上，可以使用关键字transient修饰。

7. 如果一个类维护了另外一个类的引用，那么另外一个类也需要实现Serializable接口。

使用步骤：

类实现Serializable接口

class User implements Serializable{

private static final long serialVersionUID = 1L;

String userName ;

String password;

transient int age; // transient 透明 不被系列化到硬盘上使用此关键字修饰

Address address ;

public User(String userName , String passwrod) {

this.userName = userName;

this.password = passwrod;

}

public User(String userName , String passwrod,int age,Address address) {

this.userName = userName;

this.password = passwrod;

this.age = age;

this.address = address;

}

@Override

public String toString() {

return "用户名："+this.userName+ " 密码："+ this.password+" 年龄："+this.age+" 地址："+this.address.city;

}

}

反系列化和系列化操作

public class Demo3 {

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

writeObj();

// readObj();

}

//把文件中的对象信息读取出来-------->对象的反序列化

public static void readObj() throws IOException, ClassNotFoundException{

//第一步 找到目标文件

File file = new File("F:\\obj.txt");

//第二步 建立数据的输入通道

FileInputStream fileInputStream = new FileInputStream(file);

//第三步 建立对象的输入流对象

ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

// 第四步 读取对象信息

User user = (User) objectInputStream.readObject(); //创建对象肯定要依赖对象所属 的class文件。

System.out.println("对象的信息："+ user);

}

//定义方法把对象的信息写到硬盘上------>对象的序列化。

public static void writeObj() throws IOException{

//第一步 把user对象的信息持久化存储。

Address address = new Address("中国","广州");

User user = new User("admin","123",15,address);

//第二步 找到目标文件

File file = new File("F:\\obj.txt");

//第三步 建立数据输出流对象

FileOutputStream fileOutputStream = new FileOutputStream(file);

//第四步 建立对象的输出流对象

ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);

//第五步 把对象写出

objectOutputStream.writeObject(user);

//第六步 关闭资源

objectOutputStream.close();

}

## properties

Properties（配置文件类）: 主要用于生产配置文件与读取配置文件的信息。

Properties要注意的细节：

1. 如果配置文件的信息一旦使用了中文，那么在使用store方法生成配置文件的时候只能使用字符流解决，如果使用字节流生成配置文件的话，默认使用的是iso8859-1码表进行编码存储，这时候会出现乱码。

2. 如果Properties中的内容发生了变化，一定要重新使用Properties生成配置文件，否则配置文件信息不会发生变化。

读取和修改

//第一步 创建Properties对象

Properties properties = new Properties();

//第二步 加载配置文件信息到Properties中

properties.load(new FileReader("F:\\persons.properties"));

//第三步 获取所有的键值信息：遍历

Set<Entry<Object, Object>> entrys = properties.entrySet();

for(Entry<Object, Object> entry :entrys){

System.out.println("键："+ entry.getKey() +" 值："+ entry.getValue());

}

//修改密码需要重新生成配置文件

//把修改后的Properties再生成一个配置文件

properties.setProperty("狗娃", "007");

properties.store(new FileWriter("F:\\persons.properties"), "hehe");

保存

//保存配置文件文件的信息。

public static void creatProperties() throws IOException{

//创建Properties

Properties properties = new Properties();

properties.setProperty("狗娃", "123");

properties.setProperty("狗剩","234");

properties.setProperty("铁蛋","345");

//使用Properties生产配置文件。

//第一个参数是一个输出流对象，第二参数是使用一个字符串描述这个配置文件的信息。

properties.store(new FileWriter("F:\\persons.properties"), "hehe");

## 系列流(合并流)

序列流，对多个流进行合并。SequenceInputStream

把三个文件合并成一个文件：需要用到Vector集合

public static void merge3() throws IOException{

//第一步 找到目标文件和被合并的位置

File file1 = new File("F:\\a.txt");

File file2 = new File("F:\\b.txt");

File file3 = new File("F:\\c.txt");

File file4 = new File("F:\\d.txt"); //被合并最后的路径名

//第二步 建立对应的输出流对象

FileOutputStream fileOutputStream = new FileOutputStream(file4);

//第三步 建立对应的输入流对象

FileInputStream fileInputStream1 = new FileInputStream(file1);

FileInputStream fileInputStream2 = new FileInputStream(file2);

FileInputStream fileInputStream3 = new FileInputStream(file3);

//第四步 创建序列流对象：Vector相当于arrayList，区别它是同步安全的，效率低

Vector<FileInputStream> vector = new Vector<FileInputStream>();

vector.add(fileInputStream1);

vector.add(fileInputStream2);

vector.add(fileInputStream3);

Enumeration<FileInputStream> e = vector.elements(); //java早期迭代器

SequenceInputStream sequenceInputStream = new SequenceInputStream(e);//把Vector放进合并流中

//第五步：读取文件数据

byte[] buf = new byte[1024];

int length = 0;

while((length = sequenceInputStream.read(buf))!=-1){ //使用合并流进行写数据

fileOutputStream.write(buf,0,length); //输出流把数据写到指定文件

}

//关闭资源

sequenceInputStream.close();

fileOutputStream.close();

}

合并只有两个文件的操作：直接使用构造方法带两个参数的

public static void merge2() throws IOException{

//找到目标文件

File inFile1 = new File("F:\\a.txt");

File inFile2 = new File("F:\\b.txt");

File outFile = new File("F:\\c.txt");

//建立数据的输入输出通道

FileOutputStream fileOutputStream = new FileOutputStream(outFile);

FileInputStream fileInputStream1 = new FileInputStream(inFile1);

FileInputStream fileInputStream2 = new FileInputStream(inFile2);

//建立序列流对象

SequenceInputStream inputStream = new SequenceInputStream(fileInputStream1,fileInputStream2);

byte[] buf = new byte[1024];

int length = 0 ;

while((length = inputStream.read(buf))!=-1){

fileOutputStream.write(buf,0,length);

}

//关闭资源

inputStream.close();

fileOutputStream.close();

}

### 切割MP3

public static void cutFile() throws IOException{

File file = new File("F:\\美女\\1.mp3");

//目标文件夹

File dir = new File("F:\\music");

//建立数据的输入通道

FileInputStream fileInputStream = new FileInputStream(file);

//建立缓冲数组读取

byte[] buf = new byte[1024\*1024];

int length = 0;

for(int i = 0 ; (length = fileInputStream.read(buf))!=-1 ; i++){

FileOutputStream fileOutputStream = new FileOutputStream(new File(dir,"part"+i+".mp3"));

fileOutputStream.write(buf,0,length);

fileOutputStream.close();

}

//关闭资源

fileInputStream.close();

}

### 合并MP3

public static void mergeFlile() throws IOException{

//找到目标文件

File dir = new File("F:\\music");

//通过目标文件夹找到所有的MP3文件，然后把所有的MP3文件添加到vector中。

Vector<FileInputStream> vector = new Vector<FileInputStream>();

File[] files = dir.listFiles();

for(File file : files){

if(file.getName().endsWith(".mp3")){

vector.add(new FileInputStream(file));

}

}

//通过Vector获取迭代器

Enumeration<FileInputStream> e = vector.elements();

//创建序列流

SequenceInputStream inputStream = new SequenceInputStream(e);

//建立文件的输出通道

FileOutputStream fileOutputStream = new FileOutputStream("F:\\合并.mp3");

//建立缓冲数组读取文件

byte[] buf = new byte[1024];

int length = 0 ;

while((length = inputStream.read(buf))!=-1){

fileOutputStream.write(buf,0,length);

}

//关闭资源

fileOutputStream.close();

inputStream.close();

}

# 线程

成员变量

如果一个变量是成员变量，那么多个线程对同一个对象的成员变量进行操作时，它们对该成员变量是彼此影响的，也就是说一个线程对成员变量的改变会影响到另一个线程。

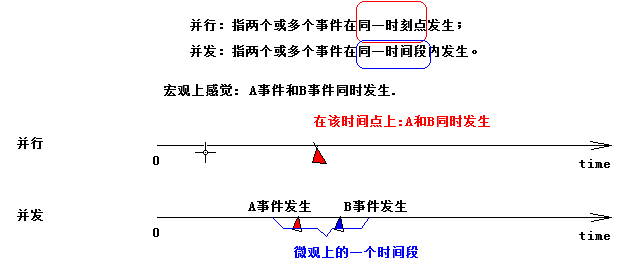
局部变量

如果一个变量是局部变量，那么每个线程都会有一个该局部变量的拷贝（即便是同一个对象中的方法的局部变量，也会对每一个线程有一个拷贝），一个线程对该局部变量的改变不会影响到其他线程。

并行：指两个或多个事件在同一时刻点发生；(即同时发生)

并发：指两个或多个事件在同一时间段内发生;(如：0.1秒至0.5秒这就是同一时间段内)

其实两个宏观上都觉的是同时发生：开发更多都是指并发



## 守护线程

守护线程(后台线程)：设置为守护线程的线程的线程跟着主线程的结束而结束

用途：如坦克大战：碉堡没了，自然而然就游戏结束了

public static void main(String[] args) {

Thread t1 = new Thread() {

public void run() {

for(int i = 0; i < 2; i++) {

System.out.println(getName() + "...aaaaaaaaaaaaaaaaaaaa");

}

}

};

Thread t2 = new Thread() {

public void run() {

for(int i = 0; i < 50; i++) {

System.out.println(getName() + "...bb");

}

}

};

t2.setDaemon(true); //设置为守护线程

t1.start();

t2.start();

}

}

## 多线程操作

### join()的使用

当前线程暂停, 等待指定的线程执行结束后, 当前线程再继续

public static void main(String[] args) {

final Thread t1 = new Thread() {

public void run() {

for(int i = 0; i < 10; i++) {

System.out.println(getName() + "...aaaaaaaaaaaaa");

}

}

};

Thread t2 = new Thread() {

public void run() {

for(int i = 0; i < 10; i++) {

if(i == 2) {

try {

//t1.join();//不指定时间，t1插队执行完再执行t2线程

t1.join(1); //插队时间1毫秒,过了指定时间后,两条线程交替执行

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println(getName() + "...bb");

}

}

};

t1.start();

t2.start();

}

### 两个线程顺序执行

①使用Object中的方法②JDK1.5新特性的ReentrantLock两种方案

1、使用Object中的方法开启两个线程顺序执行A类中的两个方法

this.wait()：让此线程等待

this.notify()：唤醒一条等待中的线程

注意：①这两个方法都必须在同步锁中进行②锁对象必须要一致

public class ThreadDemo {

//主方法开启两个线程执行A类中的两个方法(这两个方法都进行了同步锁)

public static void main(String[] args) {

final A a= new A();

new Thread() { //开启一条线程执行A类中的a方法

public void run() {

while(true) {

a.a(); //锁不应该在run方法中，所以需要抽取出来

}

}

}.start();

new Thread() { //开启第二条条线程执行A类中的b方法

public void run() {

while(true) {

a.b();

}

}

}.start();

//----------------------------------A类---------------------------------------------------

public class A {

int i=1;

public void a(){

synchronized (this) { //开启同步锁，非static方法对象都是this

if(i !=1) {

try {

this.wait(); //让此线程等待

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

System.out.println("A"+"\r\n");

i=2;

this.notify(); //唤醒等待中的随机一条线程

}

}

public void b(){

synchronized (this) {

if(i !=2) {

try {

this.wait();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

System.out.println("B"+"\r\n");

i=1;

this.notify();

}

}

2、用到对象：JDK1.5版本ReentrantLock对象和Condition

r.lock();：获取锁：开始

r.unlock();释放锁：结束

c1.await():指定c1等待 谁调用谁等待

c1.signal();指定c1的线程唤醒 谁调用谁唤醒

注意：①await；signal这两个方法都必须ReentrantLock创建的锁中使用，对象也要一致

-------------------------------------------------------主线程开始--------------------------------------------------------------

public static void main(String[] args) {

final A a= new A();

new Thread() { //开启第一条线程执行A类中的a方法

public void run() {

while(true) {

a.a();

}

}

}.start();

new Thread() {//开启第二条线程执行A类中的b方法

public void run() {

while(true) {

a.b();

}

}

}.start();

}

}

-------------------------------------------A类开始------------------------------------------------------

class A {

private ReentrantLock r = new ReentrantLock(); //锁对象：r.lock()上锁，r.unlock();释放锁

private Condition c1 = r.newCondition();//await() 等待 ：谁调用谁等待；c2.signal();唤醒：谁调用谁唤醒

private Condition c2 = r.newCondition();

int i=1;

public void a(){

//synchronized (this) { 被r.lock()代替：从这里开始上锁

r.lock(); //开始上锁

if(i !=1) {

try {

//this.wait(); 被：c1.await()代替

c1.await(); //让c1的线程等待

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

System.out.println("A"+"\r\n");

i=2;

//this.notify();被c1.signal()代替 随机唤醒别的等待线程

c2.signal(); //指定c2的锁释放

//} r.unlock()

r.unlock();//：从这里结束：释放锁

}

public void b(){

r.lock();

if(i !=2) {

try {

//this.wait(); 被：c2.await()代替 //让此线程等待

c2.await();//指定c2的上锁

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

System.out.println("B"+"\r\n");

i=1;

//this.notify();被c2.signal()代替

c1.signal();//指定c1的解锁

r.unlock();

}

### 线程之间数据传送

使用到管道流:

字节：PipedInputStream输入，PipedOutputStream输出

字符：PipedReader输入, PipedWriter输出

步骤：

第一步 创建两个类分别实现Runnable

第二步 发送类：创建对象PipedInputStream对象,创建方法返回PipedInputStream对象，发送数据

第三步 接收类:创建对象PipedOutputStream,创建方法返回PipedOutputStream对象，需接收管道输入对象：两个方式：构造传入或使用方法connect接收管道输入

第四步 在main方法中创建线程对象开启start线程，

发送方Sender类

public class Sender implements Runnable{

private PipedOutputStream out = new PipedOutputStream(); //创建管道流输出对象

public PipedOutputStream getOutputStream(){ //创建返回管道流方法

return out;

}

public void run(){ //run方法中使用：字节输出流一样的操作

String str = "Receiver, 你好!";

try{

out.write(str.getBytes()); // 向管道流中写入数据（发送）

out.close();

}

catch (IOException e) {

e.printStackTrace();

}

}

}

接收方Receiver ：和发送方类差不多

public class Receiver implements Runnable{

private PipedInputStream in = new PipedInputStream(); //创建管道流输入对象

public PipedInputStream getInputStream(){ //创建返回管道流输入对象方法：在main方法要用到

return in;

}

public void run(){ //run方法中使用：字节输出流一样的操作

String s = null;

byte b0[] = new byte[1024];

try {

int length = in.read(b0);

if (-1 != length) {

s = new String(b0, 0 , length);

System.out.println("收到了以下信息：" + s);

}

in.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

main方法调用

try{

Sender sender = new Sender(); //创建发送对象

Receiver receiver = new Receiver(); //创建接收对象

Thread senderThread = new Thread(sender); //构造发送线程

Thread receiverThread = new Thread(receiver); //构造接收线程

PipedOutputStream out = sender.getOutputStream(); // 获取管道输出流的对象

PipedInputStream in = receiver.getInputStream(); //获取管道输入流的对象

out.connect(in); // 将输出发送到输入

senderThread.start();

receiverThread.start();

} catch (IOException e) {

e.printStackTrace();

}

# 网络编程

网络通讯的三要素： 1. IP；2. 端口号；3. 协议.

IP(host):对应java的类：InetAddress

IP地址

Internet上的每台主机(Host)都有一个唯一的IP地址。IP地址的长度为32位，分为4段，每段8位，用十进制数字表示，每段数字范围为0～255，段与段之间用句点隔开。例如159.226.1.1。（四个字节）

IP地址类别

IP地址是由网络号和主机号组成

A类地址：8位网络位，24位主机位 2^24 政府单位

B类地址： 16位网络位，16位主机位 2^16 事业单位（学校、银行..）

C类地址： 24位网络位，8位主机位 2^8 私人使用

特殊的IP地址

127.0.0.1 表示 本机回环地址

端口号(port):没有对应java的类

Port地址

如果把IP地址比作一间房子 ，端口就是出入这间房子的门。真正的房子只有几个门，但是一个IP地址的端口 可以有65536（即：2^16）个之多！端口是通过端口号来标记的，端口号只有整数，范围是从0 到65535（2^16-1）。

Port分类(0~65535)

公认端口（WellKnownPorts）：从0到1023，它们紧密绑定（binding）于一些服务。

注册端口（RegisteredPorts）：从1024到49151。它们松散地绑定于一些服务。

动态和/或私有端口（Dynamicand/orPrivatePorts）：从49152到65535。

常用端口

21 FTP

80 HTTP

443 HTTPS

协议：UDP、TCP两个协议

TCP：Transmission Control Protocol 传输控制协议TCP是一种面向连接（连接导向）的、可靠的、基于字节流的运输层（Transport layer）通信协议。

特点：

面向连接的协议，数据传输必须要建立连接，所以在TCP中需要连接时间。

传输数据大小限制，一旦连接建立，双方可以按统一的格式传输大的数据。

一个可靠的协议，确保接收方完全正确地获取发送方所发送的全部数据。

UDP： User Datagram Protocol的简称， 中文名是用户数据包协议，是 OSI 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。

特点：

每个数据报中都给出了完整的地址信息，因此无需要建立发送方和接收方的连接。

UDP传输数据时是有大小限制的，每个被传输的数据报必须限定在64KB之内。

UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

## InetAddress

InetAddress: 常用的方法

1、getLocalHost(); 获取本机的IP地址

2、getByName("IP或者主机名") 根据一个IP地址的字符串形式或者是一个主机名生成一个IP地址对象。 (用于获取别人的IP地址对象)

3、getHostAddress() 返回一个IP地址的字符串表示形式。

4、getHostName() 返回计算机的主机名。

---------------------------------------------------------------获取本地机器信息--------------------------------------------------------------

InetAddress in = InetAddress.getLocalHost();

System.out.println(in.toString());

System.out.println(in.getHostAddress());

System.out.println(in.getHostName());

----------------------------------------------------------------获取任意一台主机的IP信息------------------------------------------------

InetAddress in = InetAddress.getByName("www.baidu.com");

System.out.println(in.toString());

System.out.println(in.getHostAddress());

System.out.println(in.getHostName());

----------------------------------------------------------------获取百度对应的多台主机----------------------------------------------------

InetAddress [] ins = InetAddress.getAllByName("www.baidu.com");

for(InetAddress in:ins){

System.out.println(in.toString());

System.out.println(in.getHostAddress());

System.out.println(in.getHostName());

}

运行结果：

www.baidu.com/220.181.111.148

220.181.111.148

www.baidu.com

www.baidu.com/220.181.112.143

220.181.112.143

www.baidu.com

## 协议

### UDP

1、将数据极其源和目的封装为数据包，不需要建立连接。

2、每个数据包大小限制在64K中

3、因为无连接，所以不可靠

4、因为不需要建立连接，所以速度快

5.udp 通讯是不分服务端与客户端的，只分发送端与接收端

运用场所： 物管的对讲机, 飞Q聊天、 游戏...

简单使用操作

发送端的使用步骤

1. 建立udp的服务。

2. 准备数据，把数据封装到数据包中发送。 发送端的数据包要带上ip地址与端口号。

3. 调用udp的服务，发送数据。

4. 关闭资源

//建立udp的服务

DatagramSocket datagramSocket = new DatagramSocket();

//准备数据，把数据封装到数据包中。

String data = "这个是我第一个udp的例子..";

//创建了一个数据包：参数type数组，长度，IP,端口

DatagramPacket packet = new DatagramPacket(data.getBytes(), data.getBytes().length,InetAddress.getLocalHost() , 9090);

//调用udp的服务发送数据包

datagramSocket.send(packet);

//关闭资源 ---实际上就是释放占用的端口号

datagramSocket.close();

接收端的使用步骤

接收端的使用步骤

1. 建立udp的服务

2. 准备空 的数据 包接收数据。

3. 调用udp的服务接收数据。

4. 关闭资源

//建立udp的服务 ，并且要监听一个端口。

DatagramSocket socket = new DatagramSocket(9090);

//准备空的数据包用于存放数据。

byte[] buf = new byte[1024];

DatagramPacket datagramPacket = new DatagramPacket(buf, buf.length); // 1024

//调用udp的服务接收数据，receive是一个阻塞型的方法，没有接收到数据包之前会一直等待。

socket.receive(datagramPacket); 数据实际上就是存储到了byte的自己数组中了。

System.out.println("接收到的数据："+ new String(buf,0,datagramPacket.getLength())); // getLength() 获取数据包存储了几个字节。

//关闭资源

socket.close();

群聊

接收端

public class ChatReceive extends Thread {

@Override

public void run() {

try {

//建立udp的服务,要监听一个端口

DatagramSocket socket = new DatagramSocket(9090);

//准备空的数据包存储数据

byte[] buf = new byte[1024];

DatagramPacket packet = new DatagramPacket(buf, buf.length);

boolean flag = true;

while(flag){

socket.receive(packet);

// packet.getAddress() 获取对方数据 包的IP地址对象。

System.out.println(packet.getAddress().getHostAddress()+"说:"+new String(buf,0,packet.getLength()));

}

//关闭资源

socket.close();

}catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

发送端

public class ChatSender extends Thread {

@Override

public void run() {

try {

//建立udp的服务

DatagramSocket socket = new DatagramSocket();

//准备数据，把数据封装到数据包中发送

BufferedReader keyReader = new BufferedReader(new InputStreamReader(System.in));

String line = null;

DatagramPacket packet = null;

while((line = keyReader.readLine())!=null){

//把数据封装 到数据数据包中，然后发送数据。

packet = new DatagramPacket(line.getBytes(), line.getBytes().length, InetAddress.getByName("192.168.15.255"), 9090);

//把数据发送出去

socket.send(packet);

}

//关闭 资源

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

### TCP

TCP通讯协议特点：

1. tcp是基于IO流进行数据 的传输 的，面向连接。

2. tcp进行数据传输的时候是没有大小限制的。

3. tcp是面向连接，通过三次握手的机制保证数据的完整性。 可靠协议。

4. tcp是面向连接的，所以速度慢。

5. tcp是区分客户端与服务端 的。

比如： 打电话、 QQ\feiQ的文件传输、 迅雷下载....

tcp的客户端使用步骤：

1. 建立tcp的客户端服务。

2. 获取到对应的流对象。

3.写出或读取数据

4. 关闭资源。

群聊

客户端

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

//建立tcp的客户端服务

Socket socket = new Socket(InetAddress.getLocalHost(),9090);

//获取socket的输出流对象。

OutputStreamWriter socketOut = new OutputStreamWriter(socket.getOutputStream());

//获取socket的输入流对象

BufferedReader socketReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

//获取键盘的输入流对象，读取数据

BufferedReader keyReader = new BufferedReader(new InputStreamReader(System.in));

String line = null;

//不断的读取键盘录入的数据，然后把数据写出

while((line = keyReader.readLine())!=null){

socketOut.write(line+"\r\n");

//刷新

socketOut.flush();

//读取服务端回送的数据

line = socketReader.readLine();

System.out.println("服务端回送的数据是："+line);

}

//关闭资源

socket.close();

}

服务端

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

//建立tcp的服务端

ServerSocket serverSocket = new ServerSocket(9090);

//接受客户端的连接，产生一个SOcket

Socket socket = serverSocket.accept();

//获取到Socket的输入流对象

BufferedReader socketReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

//获取到Socket输出流对象

OutputStreamWriter socketOut = new OutputStreamWriter(socket.getOutputStream());

//获取键盘的输入流对象

BufferedReader keyReader = new BufferedReader(new InputStreamReader(System.in));

//读取客户端的数据

String line = null;

while((line = socketReader.readLine())!=null){

System.out.println("服务端接收到的数据："+ line);

System.out.println("请输入回送给客户端的数据：");

line = keyReader.readLine();

socketOut.write(line+"\r\n");

socketOut.flush();

}

//关闭资源

serverSocket.close();

}

模拟Tomcat服务器

1. 开启多线程
2. 不断的接受客户端的请求
3. 可以获取到客户的ip：String address = socket.getInetAddress().getHostAddress();
4. 使用set集合统计访问次数
5. 访问：开启服务后直接在浏览器上访问栏通过ip访问

public class TomcatDemo extends Thread {

private Socket socket;

//集合保存访问者ip

static Set<String> set = new HashSet<String>();

public TomcatDemo(Socket socket){

this.socket = socket;

}

public void run() {

try {

//第四步 获取socket的输出流对象

OutputStream outputStream = socket.getOutputStream();

//第五步 把数据写到浏览器上

outputStream.write("<html><head><title>aaa</title></head><body>你好啊浏览器</body></html>".getBytes());

String address = socket.getInetAddress().getHostAddress(); //第六步 获取ip

if(set.add(address)) { //第七步 把访问者的ip存到set中

System.out.println("IP:"+address+"当前访问的次数是："+set.size());

}

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

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

//第一步：建立tcp的服务端

ServerSocket serverSocket = new ServerSocket(9090);

//第二步 不断的接受客户端的连接

while(true){

Socket socket = serverSocket.accept();

new TomcatDemo(socket).start();//第三步 开启多线程

}

}

}

# GUI

第一步 创建窗口

第二步 布局

\* BorderLayout（边界布局管理器）JFrame默认的布局管理器。

东，南，西，北，中

FlowLayout（流式布局管理器）Panel（面板）默认的布局管理器。

把Jframe设置为流式布局管理器

JFrame f = new JFrame("我的第一个窗口");

f.setLayout(new FlowLayout()); //设置布局管理器

行中间排列，一行放不下自动 跳到第二行进行中间排列

\* GridLayout（网格布局管理器）

规则的矩阵：

\* CardLayout（卡片布局管理器）

选项卡：就和easyui的选项卡一样

\* GridBagLayout（网格包布局管理器）

非规则的矩阵：如widows的计算器

第三步 事件

事件处理:

\* 事件: 用户的一个操作

\* 事件源: 被操作的组件

\* 监听器: 一个自定义类的对象, 实现了监听器接口, 包含事件处理方法,把监听器添加在事件源上, 当事件发生的时候虚拟机就会自动调用监听器中的事件处理方法

Frame f = new Frame("我的窗体");

//事件源是窗体,把监听器注册到事件源上

//事件对象传递给监听器

f.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

//退出虚拟机,关闭窗口

System.exit(0);

}

});

# 计时器

用到的类：Timer和它的子类TimerTask

Timer类用来执行

TimerTask类：继承了线程Runnable，需在run方法中写程序

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

Timer t = new Timer(); 第一步 创建对象Timer

//在指定时间安排指定任务

//第一个参数,是安排的任务,第二个参数是执行的时间,第三个参数是过多长时间再重复执行

t.schedule(new MyTimerTask(), new Date(188, 6, 1, 14, 22, 50),3000);第二步 使用方法调用子类和时间

while循环中没用，只是控制台看效果

while(true) {

Thread.sleep(1000);

System.out.println(new Date());

}

}

}

----------------------------------继承子类TimerTask开始----------------------------------------------

class MyTimerTask extends TimerTask {

@Override

public void run() {

System.out.println("起床背英语单词");

}

}