**Java笔记**

# eclipse 快捷键

使用eclipse的好处：

1．不需要我们手动的编译，只要你保存java文件，那么就会自动编译。

2．可以精准的报错，会自动修复。

**快捷键：**

ctrl+ f11 ：运行

**alt + / ：快速补全**

**shift + ctrl + o ：快速导包**

**ctrl + 2放手后+ l ：快速声明**

ctrl + 1 ：自动修复

ctrl + shift + f ：代码格式化

ctrl + shift + x ：转大写

ctrl + shift + y ：转小写

ctrl + shift + / ：添加多行注释

ctrl + shift + \ ：取消多行注释

ctrl + / ：添加取消单行注释。

ctrl + d ：删除当前行代码

alt + 上下方向键 ：代码移动

**debug调试：**

可以让程序运行到断点的时候停止运行，然后查看当前变量或者是对象的数据。 方便我们分析错误。

step over(F6) ：跳过本本条语句，进入下一个语句

step into(f5) ：进入到指定的方法内部查看.

step return ：结束当前方法。

# 常见的dos命令：

盘: ：进入指定的盘的根目录。

dir ：列出当前控制台所在的路径下的所有文件以及文件夹。

cd 路径 ：进入到指定的路径下。

cd / ：返回当前路径下的根目录。

cd .． ：返回上一级目录。

md 文件夹的名称 ：创建一个文件夹。只会创建一个文件夹。不会创建文件的。

rd 文件夹的名称 ：删除一个指定名字的文件夹。不能删除非空的文件夹。

echo 数据>文件 ：创建一个文件并且写入数据。

type 文件名 ：查看指定文件的数据。

del 文件名 ：删除指定文件名的文件。文件夹名就删除该文件夹中所有文件。

\* ：通配符，通配符可以匹配任何的文件名。

Cls ：清屏（清除以前执行过的命令）

Tab ：内容补全。

上下方向键 ：找回之前敲过的命令。

Exit ：关闭当前窗口。

# jdk1.5的新特性

## 静态导入

静态导入的作用：简化书写。

静态导入仅对于一个类的静态成员起作用、

静态导入的格式：

**import static 包名.类名.静态成员;**

**如：import static java.util.Collections.max;**

**这样在使用时就不用写Collections.max(list)这么长了。写max(list)就可使用。**

**import static java.util.Collections.\*; \*表示通配符**

静态导入要注意的实现：

1．如果静态导入的成员与本类的静态成员同名了，那么在本类中优先使用本类的静态成员， 如果要使用静态导入的方法，那么需要指定类名进行调用。

## 增强for循环

增强for循环的主要作用是： 简化迭代器的书写格式。

**增强for循环的格式：**

**for(元素的数据类型 变量名 : 遍历的目标){ }**

**如：**

**int[] arr = {19,17,18,30};**

**for(int item :arr){**

**System.out.print(item+",");**

**}**

**使用增强for循环的前提： 只有实现了Iterable接口的对象或者是数组对象才能使用。**

增强for循环要注意的细节：

1．增强for循环 的底层还是通过获取迭代器的方式进行遍历的，只不过不需要我们手动获取迭代器而已，所以在使用增强for循环的遍历元素 的过程中不准使用集合

对象改变集合对象中的元素个数。

2．增强for循环与迭代器遍历的区别： 增强for循环只能用于遍历集合的元素，不能增删元素 。 而迭代器的遍历方式可以使用迭代器的方法增删元素。

3．增强for循环使用的时候必须要存在一个遍历的目标对象， 而普通的for循环可以不存在遍历的目标对象。

## 可变参数。

**可变参数的格式：**

**数据类型..． 变量名**

**如：public static void sum(int..．arr){** **}**

**这样不管是多少个int型的调用都可使用。**

可变参数要 注意的事项：

1．如果一个函数的形参使用了可变参数，那么调用该函数的时候可以传递参数也可以不传递参数。

2．可变参数实际上就是一个数组.

3．可变参数必须要位于形参列表中的最后一个位置。

4．一个方法中最多只能使用一个可变参数。

## 自动装箱与自动拆箱

java是面向对象的语言， 任何的事物都使用了一个类进行了描述。 基本类型数据也是一类事物，所以java对于基本类型的数据也使用了对应的类进行了描述，我们把描述基本类型变量的类称作为----- 包装类。

基本数据类型 包装类型

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

基本数据类型设计出对应的包装类型的好处： 可以让基本数据类型也使用上方法，让功能变得更加强大。

String str = "12";

//功能一：可以把字符串转成数字

int num = Integer.parseInt(str);

System.out.println(num+1);

//功能2： 可以把数字转成指定进制格式的字符串。

int num2 = 10;

System.out.println("二进制："+ Integer.toBinaryString(num2));

System.out.println("八进制："+ Integer.toOctalString(num2));

System.out.println("十六进制："+ Integer.toHexString(num2));\*/

//自动装箱： 自动把基本数据类型转成引用数据类型数据，

/\*ArrayList<Integer> list = new ArrayList<Integer>();

list.add(1); // new Integer(数据)

list.add(2);

\*/

//自动拆箱： 自动把引用数据类型数据转成 基本数据类型.

Integer i = new Integer(10); //引用数据类型数据

System.out.println(2+i); // intValue() 获取基本类型的数据

Integer类内部维护了一个静态的数组，该数组内部存储了-128~127这些数据，当我们创建一个Integer对象的时候，如果数据是落入到了-128~127这个范围之内，那么会直接从数组中获取，而不会创建新的Integer对象。 ----------> 把一些常用的数据事先创建好，然后贡献给大家一起使用 ： 享元模式。

集合： 集合是存储任意对象类型的集合容器。

# 枚举

枚举类的应用场景：一些方法在运行时，它需要的数据不能是任意的，而必须是一定范围内的值 , 这时候我们就可以使用枚举类进行限制。

比如： 一年四季、星期几、方向、性别..．

**定义枚举类的格式：**

**enum 类名{**

**}**

**如： //季节**

**enum Season{**

**SPRING,SUMMER,AUTUMN,WINTER; //枚举值**

**}**

**switch (Season.SUMMER) {**

**case SPRING:**

**System.out.println("春天来了..");break;**

**case SUMMER:**

**System.out.println("夏天到了..");break;**

**}**

**在switch中要写枚举类.枚举值，在case中只需写枚举值。**

枚举类要注意的事项：

1．枚举类是一个特殊的类。

2．枚举值默认的修饰符是： public static final 。

3．枚举值其实就是一个当前枚举类的对象。

4．枚举值必须是枚举类中的第一个语句。

5．枚举类的构造方法默认的修饰符是private，修饰符也必须是private.

6．枚举类允许出现抽象的方法，但是一旦出现了抽象的方法，那么枚举值就必须要实现抽象方法。

如：

enum Direction{

UP("上"){

public void test() {

System.out.println("哈哈，爽吗?");

}

},DOWN("下"){

@Override

public void test() {

System.out.println("呵呵，是向下！！");

}

}; //枚举值

String value; //成员变量

private Direction(String value){

this.value = value;

}

//成员函数

public void print(){

System.out.println("value:"+value);

}

public abstract void test();

}

# IO流

**如果操作的是文本数据时，这时候就应该使用字符流。**

**如果操作的是非文本数据时，这时候则应该使用字节流。**

IO流技术： 解决设备与设备之间的数据传输问题。

比如： 内存 ----- 硬盘 硬盘------> 内存 键盘-----> 内存

## IO技术的应用场景：

导出报表 ，上传大头照、 播放音频文件 、 切水果 、

如果数据想永久性的保存起来，那么数据一般会保存在硬盘上，硬盘的数据一般以文件形式存在。

sun也使用了一个类描述了文件与文件夹————File

## File的构造函数：

**File(String pathname) 指定文件或者文件夹的路径，创建一个File对象**

**File(File parent, String child) 指定父路径与子路径构建一个File对象**

**File(String parent, String child) 指定父路径与子路径构建一个File对象**

## 索引路径

**绝对路径： 指定文件的完整路径创建一个File对象，绝对路径一般以盘符开头。**

**相对路径： 资源文件相对于对当前路径。**

**． 代表是当前路径**

**.． 代表是上一级路径**

**注意：**

**1．如果当前路径与资源文件不是在同一个盘符下，没法写相对路径的。**

**2．在windows操作系统下, 可以使用"\" 与 "/" 作为目录分隔符 , 但是在Unix/Linux的操作系统下只能使用"/"作为目录分隔符。**

## File的常用方法

通过File类常用的方法我们就可以获取以及修改文件 的属性数据。

### 创建：

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

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。

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。

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。

**如：**

**System.out.println("创建一个空文件："+file.createNewFile());**

**System.out.println("创建一个单级文件夹："+ dir.mkdir());**

**System.out.println("创建一个多级文件夹："+ dir.mkdirs());**

#### renameTo注意事项：

**操作文件：**如果源文件与目标文件在同一级路径下，那么renameTo方法的作用是重命名，如果源文件与目标文件不在同一级目录下，那么renameTo的作用就是剪切。

**操作文件夹：**如果源文件夹与目标文件夹在同一级路径下，那么renameTo方法的作用是重命名, 如果源文件夹与目标文件夹不在同一级目录下,那么renameTo不起作用（不能用于剪切文件夹）。

### 删除：

**delete()**

删除文件或一个空文件夹，如果是文件夹且不为空，则不能删除，成功返回true，失败返回false。**执行就马上删除**

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。**当jvm退出的时候执行删除动作。**

### 判断：

exists() 判断文件或文件夹是否存在。

isFile() 判断是否是一个文件。如果不存在，则始终为false。

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

isHidden() 判断是否是一个隐藏的文件或是否是隐藏的目录。

isAbsolute() 测试此抽象路径名是否为绝对路径名。

### 获取：

getName() **获取文件或文件夹的名称**，不包含上级路径。

getPath() 返回绝对路径，可以是相对路径，但是目录要指定

getAbsolutePath() 获取文件的绝对路径，与文件是否存在没关系

length() 获取文件的大小（字节数），如果文件不存在则返回0L（零L），如果是文件夹也返回0L。

getParent() 返回此抽象路径名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。

lastModified() 获取最后一次被修改的时间。

### 文件夹相关：

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

**File[] files = File.listRoots(); //列出所有的盘符**

list() 返回目录下的文件或者目录名，包含隐藏文件。对于文件这样操作会返回null。

**String[] fileNames = file.list(); //获取当前路径下面的所有子文件名与子文件夹名。**

listFiles() 返回目录下的文件或者目录对象（File类实例），包含隐藏文件。对于文件这样操作会返回null。

**File[] files = file.listFiles(); //把子文件与子目录存储到一个数组中返回。**

**File[] files = dir.listFiles(new JavaFileFilter()); //返回所有符合条件的子文件与子目录**

**JavaFileFilter()是使用的FilenameFilter接口的比较器**

## IO流类别：

### 1．流向划分 ：

输入流

输出流

什么时候使用输入流什么时候使用输出流？

**以当前程序作为参照物，数据流入 则使用输入流， 数据流出则使用输出流。**

### 2．处理的单位：

**字节流：**字节流就是用于 读取文件的字节数据的，读取到的数据不会经过任何的处理

**字符流：**读取到的字节数据还会帮你转换成你看得懂的字符数据，读取的是以字符作单位的数据。 字符流 = 字节流+ 解码

## IO异常处理:

**try{**

**异常代码**

**}catch(IOException e){**

**throw new RuntimeException(e);**

**// 把真正的异常原因包装到RuntimeException中然后再抛出。 （糖衣炮弹）**

**}finally{**

**try{ //关闭资源()**

**if(fileInputStream!=null){**

**fileInputStream.close(); } //关闭资源成功**

**}catch(IOException e){**

**throw new RuntimeException(e); //关闭资源失败**

**}**

**}**

## 字节流

字节流： 读取的数据是以字节为单位的，由8位组成。

### 输入字节流

----------| InputStream 抽象类 所有输入字节流的基类

--------------| FileInputStream 读取文件数据的输入字节流

--------------| BufferedInputStream 缓冲输入字节流

#### 输入字节流FileInputStream：

使用FileInputStream 读取文件数据：

1．找到目标文件

2．建立数据的输入通道

3．读取文件的数据

常用方式：

//使用循环配合缓冲 数组读取

public static void read() throws IOException{

//找到目标文件

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

//建立数据的通道

FileInputStream fileInputStream= new FileInputStream(file);

//读取数据

byte[] buf = new byte[1024]; //缓冲字节数组的长度一般都是1024的倍数。

int length = 0 ; //记录本次读取的自己个数。

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

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

}

//关闭资源(释放资源文件)

fileInputStream.close();

}

#### 缓冲输入字节流BufferedInputStream

该类的本质其实只是在内部维护了一个8kb的字节数组而已。主要是为了提高我们的读取文件的效率。

缓冲输入字节流的作用： 提高我们读取文件数据的效率。

**凡是缓冲流都没有读写文件的能力，所以要先建立文件与程序的输入通道。**

步骤：

//第一步：找到目标文件

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

//第二步：建立文件与程序的输入通道

FileInputStream fileInputStream = new FileInputStream(file);

//第三部：建立缓冲输入字节流

BufferedInputStream bufferedInputStream = new BufferedInputStream(fileInputStream);

//读取文件数据

BuffereInputStream注意的事项：

1．BuffereInputStream 的close方法实际上关闭的就是你传递进去的FileInputStream对象。

### 输出字节流

--------| OutputStream 抽象类 所有输出字节流的父类。

------------| FileOutputStream 向文件输出数据的输出字节流。

------------| BufferedOutputStream 缓冲输出字节流，为了提高写文件数据的效率。

#### 输出字节流FileOutputStream：

使用FileOutputStream步骤：

1．找到目标文件

2．建立数据的输出通道

public static void write() throws IOException{

//找到目标文件

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

//建立数据的输出通道

FileOutputStream fileOutputStream = new FileOutputStream(file,true);

//第二个参数为true时，写入文件数据就是以追加的形式写入的

//准备数据， 把数据写出

String str = "\r\nhello world";

//把字符串转成字节数组

byte[] buf = str.getBytes();

//把字节数组写出

fileOutputStream.write(buf);

//关闭资源

fileOutputStream.close();

}

FileOutputStream要注意的细节：

1．new FileOutputStream 的时候，**如果目标文件不存在，那么会先创建目标文件**，然后再写入。

2．**new FileOutputStream(file) 如果目标文件已经存在，那么会先清空目标文件的数据，然后再写入新的数据.**

3．写入数据的时候如果**需要以追加的形式写入，那么需要使用new FileOutputStream(file,true) 这个构造函数**。

4．使用write(int b)方法的时候，虽然参数接受的一个int类型的数据，但是实际上只会把数据的低八位写出，其他24位丢弃。

**write(byte[] b, int off, int len) // 指定开始的索引值与字节个数写出。**

#### 缓冲输出字节流BufferedOutputStream

**BufferedOutputStream 需要注意的事项：**

1．使用BufferedOutputStream的write方法时候，数据其实是写入了BufferedOutputStream内部维护的字节数组中，只有你**调用 BufferedOutputStream的close方法或者是flush方法数据才会真正的写到硬盘上去**或者内部维护的字节数组已经存储满数据了，这时候数据也会写到硬盘上去。

2． BufferedOutputStream 的close方法实际上关闭的就是你传入的OutputStream对象的close方法。

## 字符流

字符流： 读取的数据是以字符为单位的，会把读取到字节数据转换成我们看得懂的字符。

**字符流 = 字节流 + 编码（解码）**

### 输入字符流

-----------| Reader 抽象类 所有输入字符流的基类。

----------------| FileReader 读取文件数据的输入字符流。。

----------------| BufferedReader 缓冲输入字符流。

#### 输入字符流FileReader

FileReader的使用步骤：

1. 找到目标文件。
2. 建立数据的输入通道
3. 读取文件数据
4. 关闭资源

如：

**用法一：**

//找到目标对象

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

//建立数据的输入通道

FileReader fileReader = new FileReader(file);

//读取文件数据

int content= 0;

while((content=fileReader.read())!=-1){

**// FileReader的read()方法每次读取一个字符的数据，如果读到了文件末尾返回-1表示。**

System.out.print((char)content);

}

//关闭资源

fileReader.close();

**用法二：**

//找到目标文件

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

//建立数据的输入通道

FileReader fileReader = new FileReader(file);

//建立缓冲字符数组，读取文件的数据

char[] buf = new char[1024];

int length = 0;

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

// read(char[] buf) 读取到的字符数组存储到了字符数组中，返回了本次读取到的字符个数。

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

}

//关闭资源

fileReader.close();

#### 缓冲输入字符流BufferedReader

该类出现的目的：

提高读取文件字符数据的效率，对FileReader的功能进行了拓展---**readLine()**。

**readLine() //读一行字符**

使用：

//找到目标文件

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

//建立文件的输入通道。

FileReader fileReader = new FileReader(file);

//缓冲输入字符流

BufferedReader bufferedReader = new BufferedReader(fileReader);

//读取文件的数据

String line = null;

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

System.out.println(Arrays.toString(line.getBytes()));

}

//关闭资源

bufferedReader.close();

### 输出字符流

-----------| Writer 抽象类 输出字符流的基类。

----------------| FileWriter 向文件写出数据输出字符流。

----------------| BufferedWriter 缓冲输出字符流。

#### 输出字符流FileWriter

FileWriter 使用步骤：

1．找到目标文件

2．建立数据的输出通道

3．写出数据

4．关闭资源

**FileWriter 要注意的事项：**

1．new FileWriter（file）的时候 ，如果目标文件不存在，那么会创建目标文件对象， 如果目标文件已经存在了，那么则不再重新创建。

2．使用new FileWriter（file） 这个构造方法的时候，默认是会先清空文本的数据，然后再写入新的数据。如果需要追加数据则需要使用 new FileWriter(file,true)这个构造方法。

3．**使用FileWriter的write方法的时候，数据是写入了FileWriter内部维护的字符数组中，如果需要把数据真正的写到硬盘上去，需要调用flush方法或者 是close方法 或者是内部维护的字符数组已经满了，这时候也会写到硬盘上。**

#### 缓冲输出字符流BufferedWriter

目的： 为了提高写文件数据的效率以及拓展FileWriter的功能---- **newLine()**

**newLine() //添加一个回车换行 等价于bufferedWriter.write("\r\n");**

使用：

//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);

//建立缓冲输出字符流

BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

//准备数据，把数据写出

bufferedWriter.newLine(); // bufferedWriter.write("\r\n");

bufferedWriter.write("马上你们就要做练习了喔！");

//刷新一个缓冲区

bufferedWriter.close();

## 装饰者设计模式:

**装饰者设计模式： 增强一个类的功能。**

### 装饰者设计模式的步骤：

1．在增强类的内部维护一个被增强类的引用。

2．让增强与被增强类有一个共同父类或者是父接口.

### 装设者设计模式的：

好处：利用了多态达到了类与类之间可以互相的装饰，比较灵活。

缺点： 代码结构不清晰，难以理解。

### 实例：

需求1 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有行号。

需求2 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有分号。

需求3 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有双引号。

需求4 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有行号+ 分号。

需求5 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有行号+ 双引号。

需求6 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有分号+双引号。

需求7 ：编写一个类对BufferedReader的功能进行增强， 增强其readLine方法，返回数据带有行号+ 分号+双引号。

//带行号的缓冲类 增强类

class BufferedLineNum extends BufferedReader{

BufferedReader bufferedReader; // 内部维护一个bufferedReader类，用于指向子类实现多态

int count = 1;

public BufferedLineNum(BufferedReader bufferedReader){

// BufferedReader bufferedReader = new BufferedSemi(); 指向子类

super(bufferedReader);

//该语句是没有任何作用的语句，只不过为了让编译不报错而已。因为父类没有无参的构造方法。

this.bufferedReader = bufferedReader;

}

public String readLine() throws IOException {

String line = bufferedReader.readLine();

// 解决方案：如果这里的readLine方法是BufferedSemi的readline方法那么该问题解决了。

if(line==null){

return null;

}

line = count+":"+line;

count++;

return line;

}

}

//带分号的缓冲输入字符流

class BufferedSemi extends BufferedReader {

//为什么要继承BufferedReader ? 继承是为了让该类的对象可以传递给BuffereLineNum的构造方法。

BufferedReader bufferedReader ; //在内部维护一个需要被增强的类的引用

public BufferedSemi(BufferedReader bufferedReader){

super(bufferedReader);

this.bufferedReader = bufferedReader;

}

public String readLine() throws IOException {

String line = bufferedReader.readLine();

if(line==null){

return null;

}

line = line+";";

return line;

}

}

//带双引号 缓冲类

class BufferedQuto extends BufferedReader{

//在内部维护一个需要被增强的类的引用

BufferedReader bufferedReader ;

public BufferedQuto(BufferedReader bufferedReader){

super(bufferedReader);

this.bufferedReader = bufferedReader;

}

public String readLine() throws IOException {

String line = bufferedReader.readLine();

if(line==null){

return null;

}

line = "\""+line +"\"";

return line;

}

}

public class Demo{

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

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

//建立缓冲输入流对象

FileReader fileReader = new FileReader(file);

//创建一个缓冲输入字符流对象

BufferedReader bufferedReader = new BufferedReader(fileReader);

//创建一个带行号 的缓冲输入字符流

BufferedLineNum bufferedLineNum = new BufferedLineNum(bufferedReader);

//带分号

BufferedSemi bufferedSemi = new BufferedSemi(bufferedLineNum);

//创建一个带双引号的缓冲类对象

BufferedQuto bufferedQuto = new BufferedQuto(bufferedSemi);

String line = null;

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

System.out.println(line);

}

}

}

## SequenceInputStream (序列流)

可以把多个输入流串联起来读取，先从第一个开始i读取，然后再下一个....

注：序列流是没有读取文件的能力。

### 构造方法

#### [SequenceInputStream](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/io/SequenceInputStream.html#SequenceInputStream(java.util.Enumeration))([Enumeration](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/util/Enumeration.html)<? extends [InputStream](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/io/InputStream.html)> e)

记住参数来初始化新创建的 SequenceInputStream，该参数必须是生成运行时类型为 InputStream 对象的 Enumeration 型参数。

#### [SequenceInputStream](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/io/SequenceInputStream.html#SequenceInputStream(java.io.InputStream, java.io.InputStream))([InputStream](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/io/InputStream.html) s1, [InputStream](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/io/InputStream.html) s2)

通过记住这两个参数来初始化新创建的 SequenceInputStream（将按顺序读取这两个参数，先读取 s1，然后读取 s2），以提供从此 SequenceInputStream 读取的字节。

### 使用方法

#### 通过Vector得到一个迭代器

1．找到目标文件

File inFile1 = new File("F:\\day01.txt");

…

2．建立数据的输入输出通道

FileOutputStream fileOutputStream = new FileOutputStream(outFile);

…

3．通过Vector得到一个迭代器

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

v.add(fileInputStream1);

v.add(fileInputStream2);

v.add(fileInputStream3);

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

5．创建一个序列流对象

SequenceInputStream inputStream = new SequenceInputStream(e);

6．创建缓冲数组，边读边写

byte[] buf = new byte[1024];

int length = 0;

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

fileOutputStream.write(buf,0,length);

}

7．关闭资源

inputStream.close();

fileOutputStream.close();

#### 通过ArrayList得到一个迭代器

…

3．通过ArrayList得到一个迭代器

ArrayList<FileInputStream> list = new ArrayList<FileInputStream>();

list.add(fileInputStream1);

list.add(fileInputStream2);

list.add(fileInputStream3);

**final Iterator<FileInputStream> it = list.iterator();**

//创建一个序列流对象

SequenceInputStream inputStream = new SequenceInputStream(

**new Enumeration<FileInputStream>() {**

**public boolean hasMoreElements() {**

**return it.hasNext();**

**}**

**public FileInputStream nextElement() {**

**return it.next();**

**}**

**}**);

**创建一个Enumeration将Iterator的it.hasNext();和it.next();传入**

## 对象输入输出流

ObjectOutputStream (对象的输出流类) : 该类主要是用于把对象数据写出到文件上的。

writeObject(Objec); //读方法

ObjectInputStrea (对象的输入流类) : 把硬盘中的对象数据读取回来。

readObject(); //写方法

**private static final long serialVersionUID = 1L; 自定义class版本号**

**transient 透明化，将不会存入文件**

### 实例：

//地址类

class Address implements Serializable{

String country;

String city;

public Address(String country, String city) {

super();

this.country = country;

this.city = city;

}

}

//用户类

class User implements Serializable{

**private static final long serialVersionUID = 1L; //自定义class版本号**

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

String userName;

int password;

**transient int age; //透明化，将不会存入文件**

public User(String userName, int password, int age) {

this.userName = userName;

this.password = password;

this.age =age;

}

public String toString() {

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

}

}

public class Demo {

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

writeObj();

readObj();

}

//对象的反序列化-----> 读取硬盘中的对象到内存中。

public static void readObj() throws Exception{

//找到目标文件

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

//建立数据的输入流对象

FileInputStream fileInputStream = new FileInputStream(file);

//建立对象的输入流

ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

//读取对象的数据

User user = (User) objectInputStream.readObject();

//反序列化的时候需要创建对象， 创建对象需要依赖什么？ Class文件

System.out.println("对象的信息："+ user);

//关闭资源

objectInputStream close();

}

//把对象写到文件上------>对象的序列化。

public static void writeObj() throws IOException{

User user = new User("admin",123,18);

//找到目标

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

//建立数据的输出通道

FileOutputStream fileOutputStream = new FileOutputStream(file);

//建立对象的输出流对象

ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);

//把对象写出到输出流中

objectOutputStream.writeObject(user);

//关闭资源

objectOutputStream.close();

}

}

#### 对象的输入输出流要注意的实现：

1．使用ObjectOutputStream的writeObject方法时候，只能写出实现了Serializable接口的对象。Serializable 接口没有任何的方法，这种接口我们称作为标识接口。

2．对象反序列化的时候创建对象是不会调用构造方法的。

3．我们把对象写到文件上的时候，文件除了记录对象的一些信息以外，还记录了class的版本号（serialVersionUID）， 这个版本号

是通过一个类的类名、 包名、 工程名、成员一起算出的一个id号。

4．在反序列化的时候，jvm会使用本地class文件算出一个id号与文件记录的id号进行对比，如果不一致，反序列化失败。

**5．如果一个类的成员可能在后期会发生改动，那么可以在序列化之前就指定一个serialVersionUID , 如果一个类一家指定了一个serialVersionUID那么java虚拟机则不会再计算该class文件的serialVersionUID了。**

**6 ．如果一个类的某些成员不想被序列化到硬盘上，可以使用关键字transient修饰。**

**7．如果一个类的内部维护了另外一个类对象，那么另外一个类也必须要实现Serializable接口。**

## 配置文件类Properties

**Properties ----> 配置文件类 属于Map集合体系的。**

### Properties的作用：

#### 1．生成配置文件。

setProperty (key,value) //将指定的键和值添加到Properties对象中

store(FileWriter, "描述文字") //利用Properties生成一个配置文件

#### 2．读取配置。

[load](mk:@MSITStore:D:\教程\说明手册\JAVA%20API【中文】.CHM::/java/util/Properties.html#load(java.io.InputStream))([InputStream](mk:@MSITStore:D:\\教程\\说明手册\\JAVA%20API【中文】.CHM::/java/io/InputStream.html" \o "java.io 中的类) inStream) //加载配置文件到Properties

getProperty(key) //获取指定键的值

setProperty(key,value) //用添加去修改数据

properties.entrySet() //使用从类Hashtable继承的方法去便利数据

### Properties要注意的事项：

1．**往Properties添加数据的时候，千万不要添加非字符串类型的数据**，如果添加了非字符串类型的数据，那么properties的处理方式就是进行强制类型转换，强转报错。

2．如果properties的数据出现了中文字符，那么使用store方法时，千万不要使用字节流，如果使用了字节流，那么默认使用iso8859-1码表进行保存，如果出了

中文数据建议使用字符流。

3．如果修改了properties里面的数据，一定要重新生成一个配置文件。

### 实例：

**//创建一个配置文件，使用setProperty()方法添加可强制添加字符串类型**

public static void createProperties() throws IOException{

//创建一个Properties对象

Properties properties = new Properties();

properties.setProperty("狗娃", "123");

properties.setProperty("狗剩", "234");

properties.setProperty("铁蛋", "456");

**//字节流写入使用FileOutputStream创建**

//FileOutputStream fileOutputStream = new FileOutputStream("f:\\users.properties");

**//字符流写入使用FileWriter创建**

FileWriter fileWriter = new FileWriter("f:\\users.properties");

**//利用Properties生成一个配置文件。**

properties.store(fileWriter, "hello"); //"hello"是一段对参数列表进行描述的文字。

fileWriter.close();

}

**//读取配置文件 ---- 加载配置文件到Properties是使用load方法。**

public static void readProperties() throws IOException{

//创建一个Properties

Properties properties = new Properties();

//建立输入字符流对象

FileReader fileReader = new FileReader("f:\\users.properties");

//加载配置文件的数据到Properties是使用load方法。

properties.load(fileReader);

**//遍历元素**

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

**for(Entry<Object, Object> entry: set){**

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

**}**

//修改狗娃...

properties.setProperty("狗娃", "110");

//重新生成一个配置文件

properties.store(new FileWriter("f:\\users.properties"), "hehe");

fileReader.close();

}

## 打印流 PrintStream

### 打印流的好处：

1．打印流可以打印任意类型的数据。

2．打印流打印任意类型数据之前，会先把数据转成字符串然后再打印出去。

我们使用字节流或者字符流写出int类型数据的时候会比较麻烦，因为我们需要把这些数据线转换成字符串然后我们才能写出去。

### 使用实例：

**PrintStream printStream = new PrintStream(file);**

#### 作用１：可以为我们打印任意类型的数据，而且打印数据之前会先转换成字符串然后再打印。

//找到目标文件

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

//建立打印流对象

PrintStream printStream = new PrintStream(file);

//打印数据

printStream.println(97);

printStream.println(true);

printStream.println(3.14);

//关闭资源

printStream.close();

#### 作用2： 收集日志信息

FileOutputStream fileOutputStream = new FileOutputStream(new File("F:\\20150519.log"), true);

PrintStream printStream = new PrintStream(fileOutputStream);

try{

String str=null;

System.out.println("字符个数："+ str.length());

int result = 4/0;

}catch(Exception e){

e.printStackTrace(printStream);

}

**小知识：标准的输出流默认是指控制台，可以进行修改到文件。**

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

//修改标准的输出流

System.setOut(printStream);

System.out.println("hello"); //这样hello就输入到F:\\b.txt中了

## 编码与解码

### 码表：

ASCII：英文码表，每个字符占1个字节。A是65，a是97

GB2312：兼容ASCII，包含中文，每个英文占1个字节（正数），中文占2个字节（2个负数）。

GBK：兼容GB2312，包含更多中文，每个英文占1个字节（正数），中文占2个字节（第一个负数、第二个可正可负）。

Unicode：国际码表，每个字符占2个字节。Java中存储字符类型就是使用的Unicode编码。

UTF-8：国际码表，英文占1个字节，中文占3个字节。

### 注意事项：

1．我们一般都会让编码与解码使用同样的码表，这样子可以避免出现乱码问题。

2．英文在每个码表中都是兼用的。

3．编码的时候如果使用了unicode码表， 那么默认就会使用utf-16码表。

4．中国的电脑默认是使用GBK码表。

## 转换流

**输入字节流的转换流： InputStreamReader**

**输出字节流的转换流： OutputStreamWriter**

### 转换流的作用：

1．可以把字节流转换成字符流。

2．可以指定任意的码表进行读写数据。

### 使用场景：

因为有些特殊的情况下得到的是字节流所以属于使用到转换流。

### 实例：

#### 把输入字节流转换成了输入字符流

//因InputStream只能获取标准的输入字节流

InputStream in = System.in;

//把字节流转换成字符流，建立缓冲输出字符流。

InputStreamReader inputStreamReader = new InputStreamReader(in);

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

**通常会把上面三步这样写**

**BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));**

String line = null;

while((line = bufferedReader.readLine())!=null){ //一次读取一行的功能

System.out.println(line);

}

#### 把输出字节流转换成输出字符流

FileOutputStream fileOutputStream = new FileOutputStream("f:\\a.txt");

String data = "hello world";

//需求：要把输出字节流转换成输出字符流.

//字节流向文件输出数据的时候需要借助String类的getbyte功能，我想使用字符流.

OutputStreamWriter writer = new OutputStreamWriter(fileOutputStream);

//写出数据

writer.write(data);

//关闭资源

writer.close();

#### 指定码表进行写数据

// FileWriter 默认使用的码表是gbk码表，而且不能指定码表写。

FileOutputStream fileOutputStream = new FileOutputStream("f:\\a.txt");

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

fileWriter.write("中国");

fileWriter.close();

#### 指定码表读取数据

//找到文件

FileInputStream fileInputStream = new FileInputStream("F:\\a.txt");

//指定码表读取

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));

}

inputStreamReader.close();

# 多线程

## 概况

**进程： 正在运行的程序称作为一个进程。 进程负责了内存空间的划分。**

**线程： 线程负责了代码的执行， 线程是进程中的一个代码执行路径。**

**多线程 ： 在一个进程中有多个线程在执行不同的任务代码。**

**疑问：windows号称是多任务的操作系统 , widnows真的在同时运行多个应用程序吗？**

宏观角度，windows确实在同时运行多个应用程序。

微观角度， cpu在做一个快速的切换动作，由于切换的速度比较快，所以我们没有感觉到而已.

**疑问： 以前我们没有学过线程啊，为什么代码还是可以执行呢？**

任何一个java程序在运行的时候，jvm都会为该应用程序创建一个主线程， 主线程的任务就是把main方法的代码执行完毕。

**笔试题目： 一个java应用程序在运行的时候至少有几个线程？**

**2个线程，分别为主线线和圾回收器线程。**

### 多线程的优缺点：

#### 优点：

1．解决在一个进程中可以同时执行多个任务代码的问题。

2．提高了资源利用率。

#### 缺点：

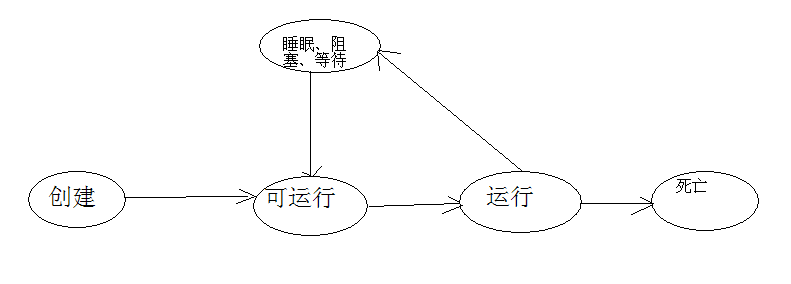
1．增加了cpu的负担。

2．降低了一个进程中线程的执行概率.

3．引发了线程安全问题。

4．引发了死锁现象.

### 线程的状态



创建：新创建了一个线程对象。

## 如何自定义线程：

**推荐使用： 推荐使用第二种。 因为java是单继承的。**

### 方式一：

1．自定义一个类继承Thread.

2．子类重写run方法，把自定义线程的任务定义在run方法上。

3．创建thread子类的对象，并且调用start方法开启线程。

### 方式二：

1．自定义一个类去实现Runnable接口。

2．实现了Runnable接口的run方法， 把自定义线程的任务定义在run方法上。

3．创建Runnable实现类的对象。

4．创建Thread对象，并且把Runnable实现类对象作为参数传递进去。

**如：Thread thread = new Thread(new Runnable1(),"狗娃");**

5．调用thread对象的start方法开启线程。

**疑问1： 重写run方法的目的是什么？**

每个线程都有自己的任务代码， main线程的任务代码是main方法里面的所有代码， 而自定义线程的任务代码就是run方法中的所有代码。

**疑问2： Runnable实现类对象是线程对象吗？**

runnable实现类的对象并不是一个线程对象，只不过是实现了Runnable接口的对象而已。

**疑问3： 为什么要把Runnable实现类的对象作为参数传递给thread对象呢？作用是什么？**

作用： 是把Runnable实现类的对象的run方法作为了任务代码去执行了。

**注意： run方法千万不能直接调用，直接调用run方法相当于调用了一个普通的方法而已，并没有开启一个新的线程。**

### 线程常用的方法：

Thread(String name) 初始化线程的名字

setName(String name) 设置线程对象名

getName() 返回线程的名字

static sleep() 那个线程执行了sleep的代码 ，那么该线程就会睡眠指定毫秒数。

currentThread() 返回当前执行该方法的线程对象引用。

getPriority() 返回当前线程对象的优先级 默认线程的优先级是5

setPriority(int newPriority) 设置线程的优先级。虽然设置了线程的优先级，但是具体的实现取决于底层的操作系统的实现（最大的优先级是10 ，最小的1 ， 默认是5）。

### 实例1：常用方法实现

public class Demo extends Thread {

**public Demo(String name){**

**super(name); //指定调用Thread类一个参数的构造方法。给线程初始化名字。**

**}**

public void run() {

for(int i = 0 ; i<100 ; i++){

System.out.println(Thread.currentThread().getName()+":"+i);

}

}

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

//创建一个自定义的线程对象

**Demo d = new Demo("狗娃"); //调用Demo类一个参数的构造方法**

d.setPriority(1); //设置优先数

d.start();

System.out.println("自定义线程的优先级："+ d.getPriority());

Thread.sleep(1000); //指定线程睡眠的毫秒数，**出现在那个线程中就睡眠那个**

Thread mainThread = Thread.currentThread() ; //返回当前线程的对象引用

System.out.println("主线程的优先级："+ mainThread.getPriority()); //默认的优先级是5 .

mainThread.setPriority(10);

System.out.println("主线程的名字："+ mainThread.getName());

for(int i = 0 ; i<100 ; i++){

System.out.println(mainThread.getName()+":"+i);

}

}

}

### 实例2：使用线程的第二种创建方式实现买票

class SaleTickets implements Runnable{

int num = 50; //非静态成员变量

public void run() { //this

while(true){

synchronized ("锁") {

if(num>0){

System.out.println(Thread.currentThread().getName()+"卖出了第"+ num+"号票");

num--;

}else{

System.out.println("售罄了...");

break;

}

}

}

}

}

public class Demo {

public static void main(String[] args) {

**SaleTickets saleTickets = new SaleTickets(); //创建Runnable实现类的对象**

**//创建三个线程对象**

Thread t1 = new Thread(saleTickets,"窗口1");

Thread t2 = new Thread(saleTickets,"窗口2");

Thread t3 = new Thread(saleTickets,"窗口3");

**//调用start方法开启线程**

t1.start();

t2.start();

t3.start();

}

}

## 线程安全

**线程安全问题出现的根本原因：**

1．存在着两个或者两个以上的线程。

2．多个线程共享了着一个资源， 而且操作资源的代码有多句。

### 线程安全问题的解决方案：

**推荐使用： 同步代码块**

**推荐的原因：**

1．同步代码块的锁对象可以由我们自己指定，同步函数的锁对象是固定 的。

2．同步代码块可以随意指定那个范围需要被同步，而同步函数必须是整个函数都同步， 代码不灵活。

#### 1．使用同步代码块

格式：

synchronized(锁对象){

需要被同步的代码；

}

**同步代码块要注意的细节：**

1．锁对象可以是任意的一个对象。

2．锁对象必须是多个线程共享的对象（锁对象必须是唯一）。

3．线程调用了sleep方法是不会释放锁对象的。

4．如果不存在着线程安全问题，不要使用同步代码块或者是同步函数，会降低效率的。

#### 2．同步函数

使用synchronized修饰该函数则称作为同步函数。

**同步函数要注意的事项：**

1．非静态同步函数的锁对象是this对象，静态函数的锁对象是当前所属类的class文件对象。

2．同步函数的锁对象是固定的，无法更改。

### 实例：

class SaleTickets extends Thread{

static int num = 50; //非静态成员变量。 非静态成员变量在每个对象中都维护了一份数据。

Static Object o = new Object(); //定义锁对象

public SaleTickets(String name){

super(name); //调用父类一个参数的构造函数，初始化线程的名字。

}

//线程的任务代码...

public void run() {

while(true){

synchronized (o) { //最简单的锁对象是"锁"（也就是一个String类的对象）

if(num>0){

try {

Thread.sleep(100);

System.out.println(Thread.currentThread().getName()+"卖出了"+num+"号票");

num--;

} catch (InterruptedException e) {

e.printStackTrace();

}

}else{

System.out.println("售空了...");

break;

}

}

}

}

}

public class Demo {

public static void main(String[] args) {

//创建线程对象

SaleTickets thread1 = new SaleTickets("窗口1");

SaleTickets thread2 = new SaleTickets("窗口2");

SaleTickets thread3 = new SaleTickets("窗口3");

//开启线程

thread1.start();

thread2.start();

thread3.start();

}

}

### 死锁现象

java同步机制解决了线程安全问题，但是同时也引发了死锁现象。

**死锁现象如何解决呢： 没法解决。 只能尽量的避免死锁现象。**

**死锁现象出现的根本原因：**

1．存在两个或者两个以上的线程存在。

2．多个线程必须共享两个或者两个以上的资源。

## 线程特殊方法

### 守护线程(后台线程)：setDaemon方法

**当前一个java应用只剩下守护线程的时候，那么守护线程马上结束。**

**守护线程应用场景：**

1．新的软件版本下载。

**守护线程要注意的事项：**

1．所有的线程默认都不是守护线程。

#### 使用方法

**线程对象.setDaemon(true) 设置一个线程为守护线程。**

**线程对象.isDaemon 判断一个线程是否为守护线程。**

### 线程让步：join方法

**如果当前线程执行了join方法，那么当前线程就会让步给新加入的线程先完成任务，然后当前线程才继续的执行自己的任务。**

#### 使用方法：

public void run() { //一个线程代码中

…

Son s = new Son(); //另一个线程

s.start();

try {

**s.join(); //当前线程执行了join方法，就会让步给新加入的线程先完成任务。**

} catch (InterruptedException e) {

e.printStackTrace();

}

…

}

## 线程的通讯

当一个线程完成了一个任务的时候，要通知另外一个线程去处理其他的事情。

### 线程通讯的方法：

**wait()** 执行了wait方法的线程，会让该线程进入以锁对象建立的线程池中等待。

**notify()** 唤醒以锁对象建立的线程池中等待线程中的一个。

**notifyAll()** 唤醒以锁对象建立的线程池中等待线程中的所有的线程。

### 线程通讯要注意的事项：

1．wait、notify、notifyAll方法都是属于Object对象的方法。

2．wait、notify方法必须要在同步代码块或者是同步函数中调用。

3．wait、notify方法必须由锁对象调用，否则报错。

4．一个线程执行了wait方法会释放锁对象的。

### 实例：生产者消费者关系

//产品类

class Product{

String name;

int price;

boolean flag ; //产品是否生成完毕的标识。false为还没有生成完毕，true生成完毕。

}

//生产者类

class Producer extends Thread{

//维护一个产品

Product p;

public Producer(Product p){

this.p = p;

}

public void run() {

int i = 0;

while(true){

synchronized (p) {

if(p.flag==false){

if(i%2==0){

p.name = "摩托车";

p.price= 4000;

}else{

p.name = "自行车";

p.price = 300;

}

System.out.println("生产了"+ p.name+" 价格："+ p.price);

i++;

//生成完毕 --- 改标识

p.flag = true;

//唤醒消费者去消费

p.notifyAll();;

}else{

//如果产品已经生产完毕，应该等待消费者先消费

try {

p.wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

}

//消费者

class Customer extends Thread{

//产品

Product p;

public Customer(Product p){

this.p = p;

}

public void run() {

while(true){

synchronized (p) {

if(p.flag==true){

System.out.println("消费者消费了："+ p.name+" 价格："+ p.price);

//改标识

p.flag = false;

p.notifyAll();

}else{

//如果产品已经被消费完毕,应该唤醒生产者去生成

try {

p.wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

}

public class Demo {

public static void main(String[] args) {

//创建一个产品对象

Product p = new Product();

//创建线程对象

Producer producer = new Producer(p);

Customer customer = new Customer(p);

//启动线程

producer.start();

customer.start();

}

}

## 停止线程： interrupt方法

**停止线程要注意的实现：**

1．我们停止一个线程一般都会配合一个变量去控制。

2．如果我们停止的是一个等待状态下的线程，那么需要配合interrupt方法去使用。

**注：interrupt() 无法停止一个线程，配合一个变量去控制在用interrupt结束线程等待。**

**线程对象.interrupt(); //强制清除指定线程的wait、 sleep状态。**

### 实例：

public class Demo extends Thread {

**boolean flag = true; //控制变量，用于结束线程**

public Demo (String name){

super(name);

}

public synchronized void run() {

int i = 0;

while(flag){

try {

this.wait(); //等待...

} catch (InterruptedException e) {

System.out.println("**父级无错误，子级只能接收错误**");

}

System.out.println(Thread.currentThread().getName()+":"+ i);

i++;

}

}

public static void main(String[] args) {

//创建线程对象

Demo d = new Demo("狗娃");

d.start();

//当主线程的i到80的时候，停止狗娃线程。

for(int i = 0 ; i<100 ; i++){

**if(i==80){**

**d.flag = false; //改变控制变量**

**d.interrupt(); //强制清除wait状态。使线程判断flag== false结束线程**

**}**

System.out.println(Thread.currentThread().getName()+":"+i);

}

}

}

# 网络编程

## 概况

**网络编程**： 不需要依赖浏览器进行通讯． 比如：QQ 、 飞Q 、 红蜘蛛。

**网页编程**： 以html作为基础实现浏览器与服务器之间的通讯。

**计算机网络**： 分布在不同地域的计算机通过外部设备把计算机链接起来，达到了数据输出、共享资源的目的就称作为一个计算机网络。

**计算机与计算机的通讯三要素：**

**1．IP地址**

**2．端口号**

**3．通讯协议**

## IP地址

IP地址的本质是有32个二进制位数组成的， 为了方便人类记录，所以把一个ip地址分成了4段， 每段8个二进制数据。 2^8 = 0~255

IP地址　＝　网络号 + 主机号

**IP的类别：**

A类地址： 一个网络号 + 三个主机号

B类地址： 两个网络号 + 两个主机号

C类地址： 三个网络号 + 一个主机号

## 端口号

端口号的范围：0－65535

公认端口：从0到1023，它们紧密绑定（binding）于一些服务。

注册端口：从1024到49151。它们松散地绑定于一些服务。

动态和/或私有端口：从49152到65535。

**使用1024到65535即可。**

**常用端口**

21 FTP

80 HTTP

443 HTTPS

## IP地址类： InetAddress

java是面向对象的语言，所以java使用了一个类描述了IP地址。

### InetAddress需要掌握的方法

getLocalHost() 返回本机的IP地址对象。

getByName(String host) 指定字符串形式的IP地址或者是主机名创建一个IP地址对象。

getAllByName(String host) 指定字符串形式的IP地址或者是主机名创建一个IP地址对象数组。

getHostAddress() 返回IP地址字符串的表示形式。

getHostName() 返回计算机的名字。

### 使用：

InetAddress address = InetAddress.getLocalHost(); //获取到本机的IP地址对象

InetAddress address = InetAddress.getByName("192.168.10.15"); //获取指定IP的IP地址对象

System.out.println("IP ： "+ address.getHostAddress()); //输出指定IP的IP地址

System.out.println("计算机的名字："+ address.getHostName()); //输出指定IP的计算机名

InetAddress[] addresses = InetAddress.getAllByName("http://www.baidu.com");

//由域名得到百度的服务器IP地址对象数组

## 网络通讯协议

在java中不管是用哪种协议通讯，计算机与计算机之间的通讯我们都统称为Socket(插座)通讯．通讯的两端计算机都必须要安装上Socket。在不同的协议下就应该有不同的插座。

### UDP 用户数据包协议

**特点：**

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,数据包会丢失。
4. 效率高。

**常用领域**：物管的对讲机、视频会议、飞Q通讯游戏行业。

### UDP通讯类： UDP分为发送端、接收端

**基于字节数据发送接收，所以需要把数据转成字节使用。**

**DatagramSocket (udp协议的服务类)**

**DatagramPacket (数据包类)**

**DatagramPacket(byte[] buf, int length, InetAddress address, int port)**

**buf: 当前数据的字节数组表示形式。**

**length : 字节数组的长度。**

**address : 发送的IP地址。**

**port: 端口号。**

### 实例：

#### UDP的发送端

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

**//第一步： 建立udp的服务**

DatagramSocket socket = new DatagramSocket();

**//第二步： 准备数据，把数据封装到数据包中**

String data = "这个是我的第一个udp的例子!!";

byte[] buf = data.getBytes();

**DatagramPacket packet = new DatagramPacket(buf, buf.length, InetAddress.getByName("192.168.1.68"),9090);**

**//第三步：调用udp的服务发送数据**

socket.send(packet);

**//第四步：关闭资源(释放端口号)**

socket.close();

}

#### UDP的接收端

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

**//第一步： 建立UDP的服务,并且要监听一个端口**

DatagramSocket socket = new DatagramSocket(9090);

**//第二步： 准备一个空的数据包**

byte[] buf = new byte[1024];

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

**//第三步： 调用up的服务接受数据包，数据包是依赖于字节数据存储东西的。**

socket.receive(packet); **// receive()阻塞型的方法， 没有接受数据会一直等待下去。**

System.out.println(packet.getAddress().getHostAddress()+ "接收到的数据："+ **new String(buf,0,packet.getLength())**);

**// getLength() 获取本次接收到的字节个数。**

**// getAddress() 获取对方的IP地址对象 ,**

**//第四步：关闭资源**

socket.close();

}

#### 局域网飞Q群聊

##### 发送端

public class ChatSender extends Thread {

public void run() {

//第一步： 建立udp的服务

DatagramSocket socket = null;

try {

socket = new DatagramSocket();

//第二步：准备数据， 把数据封装到数据包中发送。

**//数据是来自于键盘**

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

String line = null;

DatagramPacket packet = null;

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

//键盘录入的数据已经封装到了数据包中了。

packet = new **DatagramPacket(line.getBytes(), line.getBytes().length,**

**InetAddress.getByName("192.168.1.255"), 9090)**;

//调用udp的服务发送数据

socket.send(packet);

}

} catch (Exception e) {

e.printStackTrace();

}finally{

//关闭资源

socket.close();

}

}

}

##### 接收端

public class ChatReceive extends Thread {

public void run() {

//第一步： 建立dup的服务

try {

DatagramSocket socket = new DatagramSocket(9090);

//第二步：准备空的数据包，接收数据

byte[] buf = new byte[1024];

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

//调用udp的服务不断的接受数据包。

while(true){

socket.receive(packet);

System.out.println(packet.getAddress().getHostAddress()+"说："+

new String(buf,0,packet.getLength()));

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

##### 主线程

public class ChatMain {

public static void main(String[] args) {

//创建了接收端与发送端的线程对象

ChatReceive receive = new ChatReceive();

ChatSender sender = new ChatSender();

//启动线程接收与发送数据

receive.start();

sender.start();

}

}

## TCP 传输控制协议

**特点：**

1．发送数据都是需要把数据封装到数据包中再发送 的，面向无连接。

2．数据包大小不能超过64kb。

3．因为UDP协议是面向无连接的， 所以会出现数据包丢失的情况。

4．因为面向无连接，所以速度快。

5．UDP协议是不分客户端与服务端，只分发送端与接收端。

**常用领域**： 打电话、飞Q文件传输。

**数据包在什么情况下会丢失呢：**

1．带宽不足的时候。

2．cpu处理能力不足的时候。

### TCP通讯类： TCP分为客户端、服务端

**基于IO流实现，所以在建立还通讯通道后就像IO流一样的读写了。**

**Socket (客户端类)**

**ServerSocket (服务端类）**

**Socket socket = new Socket(IP地址, 端口号);**

**注意：服务端每获取到一个Socket就会由生成一个通道专门负责这个Socket的操作。**

### 实例：

#### 客户端 Socket

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

**//第一步： 建立tcp的客户端服务**

**Socket socket = new Socket(“192.168.1.64”, 9090);**

**//第二步： 准备数据 , 获取对应的流通道。**

String data = "这个是我第一个tcp的例子";

OutputStream out = socket.getOutputStream();

**//第三步： 可以吧数据写出**

out.write(data.getBytes());

**//客户端要接收服务端回送的数据**

**//获取socket的输入流**

InputStream inputStream = socket.getInputStream();

byte[] buf = new byte[1024];

int length =inputStream.read(buf);

System.out.println("客户端接收到的内容是："+ new String(buf,0,length));

**//第四步：关闭资源**

socket.close();

}

#### 服务端 ServerSocket

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

**//第一步： 建立tcp的服务端。**

ServerSocket serverSocket = new ServerSocket(9090);

**//第二步：接受客户端的连接**

Socket socket = serverSocket.accept(); //accept() 是一个阻塞型的方法

**//第三步： 获取socket对应的流通道**

InputStream inputStream = socket.getInputStream();

**//第四步： 通过输入流通道读取数据**

byte[] buf = new byte[1024];

int length = inputStream.read(buf);

System.out.println("服务端接收到的数据："+ new String(buf,0,length));

**//服务端给客户端回送数据客户端**

**//获取socket的输出流**

OutputStream out = socket.getOutputStream();

out.write("客户端你辛苦啦！！".getBytes());

**//第五步： 关闭资源**

serverSocket.close();

}

#### 一个服务端可以与多个客户端进行连接

##### 发送图片服务端

public class ImageServer extends Thread {

Socket socket;

static HashSet<String> ips = new HashSet<String>(); //该集合是用户存储客户端的ip地址的。

public ImageServer(Socket socket) {

this.socket = socket;

}

public void run() {

try {

//获取socket输出字节流

OutputStream socketOut = socket.getOutputStream();

//获取图片的输入流，读取图片的数据，把图片数据写出给客户端

FileInputStream fileInputStream = new FileInputStream("F:\\美女\\1.jpg");

byte[] buf = new byte[1024];

int length = 0 ;

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

socketOut.write(buf,0,length);

}

String ip = socket.getInetAddress().getHostAddress(); //获取到对方的ip地址

if(ips.add(ip)){ //如果可以存储到集合中，那么就意味着这个是一个新的IP地址。

System.out.println("恭喜："+ ip+"同学下载图片成功！！");

System.out.println("当前下载的人数："+ ips.size());

}

//第五步：关闭资源

fileInputStream.close();

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

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

//第一步： 建立tcp服务端的服务

ServerSocket serverSocket = new ServerSocket(9090);

while(true){

Socket socket = serverSocket.accept(); //不断的接受用户的请求连接

new ImageServer(socket).start();

//如果产生了一个Socket，那么就意味着有一个用户与服务端连接了，那么马上开启一线程为其服务。

}

}

}

##### 获取图片的客户端

public class ImageClient {

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

//第一步： 建立tcp客户端服务。

Socket socket= new Socket(InetAddress.getByName("192.168.1.68"),9090);

//获取socket的输入流对象

InputStream inputStream = socket.getInputStream();

//建立一个文件的输出流对象

FileOutputStream fileOutputStream = new FileOutputStream("f:\\美女.jpg");

//边读边写

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();

socket.close();

}

}

#### 多用户登录注册。

要求：

1．用户注册的数据必须要保存到服务端的文件上面。

2．客户端登录的时候，客户端输入的数据需要与服务端的文件校验，如果存在该用户名与密码，才允许登录成功。

使用字符流写出数据的时候,数据是写入了其内部维护的缓冲字符数组中，需要调用flush或者是close方法数据才会真正的写出去。 或者字符数组已经存储满了，数据也一样会写出去。

##### 客户端

public class LoginClient {

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

Scanner scanner = new Scanner(System.in);

//建立tcp客户端的服务

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

//获取socket的输入、输出通道。

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

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

//用户选择的功能

while(true){

System.out.println("请选择功能： A(注册) B(登录)");

String option = scanner.next();

if("a".equalsIgnoreCase(option)){

System.out.println("你选择了注册功能");

//注册

getInfo(scanner, socketReader, socketWriter, option);

}else if("b".equalsIgnoreCase(option)){

//登录

System.out.println("你选择了登录功能");

getInfo(scanner, socketReader, socketWriter, option);

}else{

System.out.println("你的选择有误,请重新选择!!");

}

}

}

public static void getInfo(Scanner scanner, BufferedReader socketReader, OutputStreamWriter socketWriter,

String option) throws IOException {

System.out.println("请输入用户名：");

String userName = scanner.next();

System.out.println("请选择密码:");

String password = scanner.next();

//把用户名与密码发送给服务端

String line =option+" "+userName+" "+password+"\r\n";

socketWriter.write(line);

//刷新缓冲流

socketWriter.flush();

//读取服务端的反馈信息

line = socketReader.readLine();

System.out.println(line);

}

}

##### 服务端

public class LoginServer extends Thread {

Socket socket ;

public LoginServer(Socket socket){

this.socket = socket;

}

@Override

public void run() {

try {

//第一步：获取socket的输入、输出流对象

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

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

//第二步： 读取客户端发送的消息。

while(true){

String line = socketReader.readLine();

String[] datas = line.split(" ");

String option = datas[0]; //客户端选择的功能

String userName = datas[1]; //用户名

String password = datas[2]; //密码

Properties properties = new Properties();

//先把配置文件的信息先加载到properties文件上。

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

if("a".equalsIgnoreCase(option)){

//注册

//把用户名与密码保存到Properties中

if(!properties.containsKey(userName)){

//如果该用户名不存在，我就把用户的信息添加上去。

properties.setProperty(userName,password);

// 把用户的注册信息保存到文件上。

properties.store(new FileWriter("F:\\users.txt"),"users info");

socketWriter.write("恭喜你,注册成功\r\n");

}else{

//如果用户名已经存在了，那么提示客户端用户名已经存在，请重新注册

socketWriter.write("该用户名已经存在，请重新输入！！\r\n");

}

}else if("b".equalsIgnoreCase(option)){

//登录

String tempPassword = properties.getProperty(userName);

if(tempPassword==null){

socketWriter.write("没有该用户名存在，请重新登录!\r\n");

}else{

if(tempPassword.equals(password)){

socketWriter.write("欢迎"+userName+"登录成功!\r\n");

}else{

socketWriter.write("你的密码有误，请重新输入!\r\n");

}

}

}

socketWriter.flush();

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

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

//建立tcp的服务端， 并且要监听一个端口

ServerSocket serverSocke = new ServerSocket(9090);

//不断的接受用户的请求连接

while(true){

Socket socket = serverSocke.accept();

//每个用户都应该开启一个线程为其服务

new LoginServer(socket).start();

}

}

}

# 图形化界面

## 概况

**软件的交互方式：**

1．dos命令的交互方式。

2．图形化界面的交互方式

**在java中图形化界面编程中把所有的图形类都称作为组件类。**

**所有图形组件类都是位于： java.awt 和 javax.swing 包。**

**在awt包中存在图形类，在swing一样会存在，区别在于swing的图形都是以J开头的。**

**java.awt包的图形类与javax.swing 包的图形类的区别：**

1．java.awt包中的所有图形类的图形都是依赖系统的图形库的。javax.swing中的图形类的图形都是sun自己去实现的。

**所有的图形类都称作为组件：容器组件、非容器组件**

## 图形化界面的工具类：用于生成居中窗体

public class FrameUtil {

//设置窗体出现在中间位置

public static void initFrame(JFrame frame,int width,int height){

**//获取默认系统工具包**

**Toolkit toolkit = Toolkit.getDefaultToolkit();**

**//获取屏幕的分辨率。**

**Dimension d = toolkit.getScreenSize();**

int x = (int) d.getWidth();

int y = (int) d.getHeight();

**frame.setBounds((x-width)/2, (y-height)/2, width, height); //x值,y值,窗体宽，窗体高**

//设置窗体的可见性

frame.setVisible(true);

**//设置窗体关闭时间**

**frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

}

}

## 容器组件

### 对话框

#### 构造方法

**Dialog(对话框)**

Dialog(Dialog owner, String title, boolean modal)

owner ： 所有者

title ： 对话框的标题

modal ： 模式 true: 对话框没有关闭的时候，不准操作所有者。 false: 对话框即使没有关闭，还是可以操作所有者

**JOptionPane(对话框)**

//消息、 警告、 错误

showMessageDialog(Component parentComponent, Object message, String title, int messageType)

parentComponent: 所有者

message ： 显示的消息

title ： 对话框的标题

messageType ： 指定对话框的类型 （消息、 警告、 错误）

#### 实例：

public static void main(String[] args) {

//创建一个窗体

JFrame frame = new JFrame("窗体");

//创建一个对话框

/\*

JDialog dialog = new JDialog(frame,"对话框",false);

FrameUtil.initFrame(frame, 500,400); //调用自定义工具类initFrame，居中

dialog.setBounds(580, 250, 200, 200); //对话框位置及大小

dialog.setVisible(true); //设置对话框展示出来

\*/

FrameUtil.initFrame(frame, 500,400);

/\*

**消息对话框INFORMATION\_MESSAGE**

JOptionPane.showMessageDialog(frame, "消息","通知:",JOptionPane.INFORMATION\_MESSAGE);

**警告对话框WARNING\_MESSAGE**

JOptionPane.showMessageDialog(frame, "警告","警告:",JOptionPane.WARNING\_MESSAGE);

**错误对话框ERROR\_MESSAGE**

JOptionPane.showMessageDialog(frame, "错误","出局:",JOptionPane.ERROR\_MESSAGE);

**确认对话框JOptionPane.showConfirmDialog**

int num = JOptionPane.showConfirmDialog(frame, "软件继续安装吗?");

System.out.println("num:"+ num);

\*/

**//输入对话框JOptionPane.showInputDialog**

String money = JOptionPane.showInputDialog(frame,"请输入你要取的金额：");

System.out.println("money："+ money);

}

### 文件对话框：FileDialog

#### 构造方法：

FileDialog(Dialog parent, String title, int mode)

parent : 所有者

title : 标题

mode : FileDialog.LOAD(加载) 或 FileDialog.SAVE(保存)

#### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("窗体");

FileDialog dialog = new FileDialog(frame, "请打开文件", FileDialog.SAVE);

//设置居中显示dialog

FrameUtil.initFrame(frame, 500, 400);

dialog.setVisible(true);

System.out.println("文件的路径：" + dialog.getDirectory()); //**getDirectory()获取文件的路径**

System.out.println("文件名："+ dialog.getFile()); //**getFile() 获取文件名**

}

}

### 面板：Panel

#### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("窗体");

//面板

JPanel panel = new JPanel();

panel.setBackground(Color.BLUE); //面板背景色

//把面板添加到窗体上。

frame.add(panel); 将面板加入窗体

FrameUtil.initFrame(frame, 500, 400);

}

### 画图：Graphics

**如果需要画图，要使用到JPanel，需要类继承JPanel，重写paint方法。**

**设置画笔的颜色 g.setColor(颜色常量);**

**使用画笔画矩形 g.fill3DRect(x坐标, y坐标, 宽, 高, 是否要立体感);**

**设置字的属性 g.setFont(new Font("字体", 加粗斜体之类的属性常量, 字的大小));**

**使用画笔写字 g.drawString("要写的文本", x坐标, y坐标);**

#### 实例：

public class Demo extends JPanel {

@Override

public void paint(Graphics g) { // Graphics 是一个画笔, 该画笔可以画图形，也可以写文字。

**//设置画笔的颜色**

**g.setColor(Color.GRAY);**

**//使用画笔画矩形**

**g.fill3DRect(0, 0, 20, 20, true);**

**g.fill3DRect(20, 0, 20, 20, true);**

**//设置画笔的颜色**

**g.setColor(Color.red);**

**//设置字的属性**

**g.setFont(new Font("宋体", Font.BOLD, 36));**

**//使用画笔写字**

**g.drawString("GAME OVER!", 200, 250);**

}

public static void main(String[] args) {

JFrame frame = new JFrame("画笔");

Demo d = new Demo();

frame.add(d);

FrameUtil.initFrame(frame, 600, 600);

}

}

## 非容器组件

### 组件方法

**标签对象 JLabel**

**输入框 JTextField**

**密码框 JPasswordField**

**单选按钮 JRadioButton**

**单选分组 ButtonGroup**

**下拉框 JComboBox**

**复选框 JCheckBox**

**文本框 JTextArea**

### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("注册");

//创建一个面板

JPanel panel = new JPanel();

**//创建一个标签对象 JLabel**

**JLabel nameLabel = new JLabel("用户名:");**

**//输入框 JTextField**

**JTextField nameField = new JTextField(12);**

//把组件添加面板

panel.add(nameLabel);

panel.add(nameField);

//密码

JLabel passLabel = new JLabel("密码:");

**//密码框 JPasswordField**

**JPasswordField passField = new JPasswordField(12);**

panel.add(passLabel);

panel.add(passField);

//性别： 单选框标签

JLabel sexLabel = new JLabel("性别:");

**//单选按钮 JRadioButton**

**// 注意：单选框一定要进行分组，在同一组的单选框中只能选择其中的一个。**

**JRadioButton man = new JRadioButton("男", true);**

**JRadioButton woman = new JRadioButton("女");**

**//单选分组 ButtonGroup**

**ButtonGroup group = new ButtonGroup();**

**group.add(man);**

**group.add(woman);**

panel.add(sexLabel);

panel.add(man);

panel.add(woman);

**//城市 --- 下拉框 JComboBox(Object[])**

JLabel cityLabel = new JLabel("来自的城市");

**Object[] citys = {"北京","上海","广州","深圳"};**

**JComboBox cityBox = new JComboBox(citys);**

panel.add(cityLabel);

panel.add(cityBox);

**//爱好---复选框 JCheckBox**

JLabel hobitLabel = new JLabel("爱好");

**JCheckBox java = new JCheckBox("java");**

**JCheckBox javascript = new JCheckBox("javascript");**

**JCheckBox write = new JCheckBox("敲java");**

panel.add(hobitLabel);

panel.add(java);

panel.add(javascript);

panel.add(write);

**//自我简介--文本框 JTextArea**

JLabel introLabel = new JLabel("自我简介:");

**JTextArea area = new JTextArea(15, 15);**

panel.add(introLabel);

panel.add(area);

frame.add(panel);

FrameUtil.initFrame(frame, 830, 500);

}

}

### 菜单组件

**菜单组件：**

**菜单条 JMenuBar**

**菜单 JMenu**

**菜单项 JMenuItem**

**关系： 菜单条添加菜单 ， 菜单添加菜单项**

**复选菜单： 菜单添加菜单， 然后菜单再添加菜单项。**

#### 实例：

public class Notepad {

//窗体

JFrame frame = new JFrame("记事本");

**//菜单条**

**JMenuBar bar = new JMenuBar();**

**//菜单**

**JMenu fileMenu = new JMenu("文件");**

**JMenu editMenu = new JMenu("编辑");**

**JMenu helpMenu = new JMenu("帮助");**

**//菜单项**

**JMenuItem open = new JMenuItem("打开");**

**JMenuItem save = new JMenuItem("保存");**

**JMenuItem copy = new JMenuItem("拷贝");**

**//用于制作帮助菜单的复选**

JMenuItem about = new JMenuItem("关于");

JMenuItem version = new JMenuItem("升级");

//文本域

JTextArea area = new JTextArea(20,20);

public void init(){

**//把菜单添加到菜单条上**

bar.add(fileMenu);

bar.add(editMenu);

**//把菜单项添加到菜单**

fileMenu.add(open);

fileMenu.add(save);

editMenu.add(copy);

**//复选菜单，菜单添加到菜单上。**

**editMenu.add(helpMenu);**

**//菜单添加复选菜单项**

helpMenu.add(about);

helpMenu.add(version);

//把菜单条添加到窗体上

frame.add(bar,BorderLayout.NORTH); //添加位置为北

frame.add(area);

FrameUtil.initFrame(frame, 500, 600);

}

public static void main(String[] args) {

new Notepad().init();

}

}

## 布局管理器

布局管理器的作用就是用于摆放组件。

每种不同的布局管理器都有不同的风格。

### 边框布局管理器：BorderLayout

#### 注意事项：

1．如果一个容器使用了BorderLayout布局管理器，那么往该容器添加组件的时候如果没有指定具体的方位，那么默认在中间。

2．Frame默认使用的就是BorderLayout布局管理器.

3．东南西北那个组件是缺少的，那么中间的组件都会占据其空缺位置。

#### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("边框布局管理器");

**//创建边框布局管理器**

**BorderLayout borderLayout = new BorderLayout();**

**//让窗体使用边框布局管理器**

**frame.setLayout(borderLayout);**

**// JFrame默认使用了BorderLayout，不添加也可使用**

**frame.add(new JButton("北"),BorderLayout.NORTH);**

**frame.add(new JButton("南"),BorderLayout.SOUTH);**

**frame.add(new JButton("西"),BorderLayout.WEST);**

**frame.add(new JButton("东"),BorderLayout.EAST);**

**frame.add(new JButton("中"),BorderLayout.CENTER);**

//初始化窗体

FrameUtil.initFrame(frame, 300, 300);

}

}

### 流式布局管理器：FlowLayout

**FlowLayout(对齐方向,左右间距,上下间距)**

#### 注意事项：

1．使用FlowLayout的时候默认是居中对齐的。

2．panel默认使用的布局管理器就是FlowLayout．

#### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("窗体");

//面板

**JPanel panel = new JPanel();**

**//创建一个流式布局管理器**

**FlowLayout flowLayout = new FlowLayout(FlowLayout.LEFT,10,0);**

**//让面板使用流式布局管理器**

**panel.setLayout(flowLayout);**

**// JPanel默认使用了FlowLayout，不添加也可使用**

frame.add(panel);

panel.add(new JButton("one"));

panel.add(new JButton("two"));

panel.add(new JButton("three"));

panel.add(new JButton("four"));

FrameUtil.initFrame(frame, 300, 300);

}

}

### 表格布局管理器：GridLayout

GridLayout(行数, 列数)

#### 注意事项：

1．如果添加的组件超过了表格的个数，那么会添加多一列处理。

#### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("计算器");

**//创建一个表格布局管理器**

**GridLayout gridLayout = new GridLayout(4, 4);**

**//让窗体使用表格布局管理器**

**frame.setLayout(gridLayout);**

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

frame.add(new JButton(i+""));

}

frame.add(new JButton("+"));

frame.add(new JButton("-"));

frame.add(new JButton("\*"));

frame.add(new JButton("/"));

frame.add(new JButton("="));

frame.add(new JButton("."));

FrameUtil.initFrame(frame, 300, 300);

}

}

## 事件

**当某个组件发生了指定的动作事，会有相应的处理方案。**

**事件分为：事件源、监听器、事件、处理方式**

### 添加监听器：addActionListener

#### 动作监听器：ActionListener(){需写方法}

**动作监听器对于鼠标点击、和空格键都是起作用的。**

##### 需写的方法

**actionPerformed 发生鼠标点击、按下空格键**

##### 实例：

public class Demo {

public static void main(String[] args) {

//创建窗体

JFrame frame = new JFrame("窗体");

//创建一个按钮

JButton button = new JButton("点我啊");

**//给按钮添加一个监听器**

**button.addActionListener(new ActionListener() {**

**//如果发生鼠标点击、按下空格键就会调用actionPerformed方法**

**@Override**

**public void actionPerformed(ActionEvent e) { // ActionEvent 事件..**

**JButton button = (JButton) e.getSource(); //获取事件源对象**

**String content = button.getText(); //获取对象中的文本**

**if("点我啊".equals(content))**

**button.setText("点他吧！"); //修改对象中的文本**

**}else{**

**button.setText("点我啊");**

**}**

**}**

**});**

frame.add(button);

FrameUtil.initFrame(frame, 200,200);

}

}

#### 鼠标监听器：MouseListener (){需写方法}

##### 需写的方法

**mouseReleased 鼠标松开**

**mousePressed 鼠标按下**

**mouseExited 鼠标离开**

**mouseEntered 鼠标进入**

**mouseClicked 鼠标单击**

##### 鼠标监听配置器

**new MouseAdapter() {**

**@Override**

**public void mouseClicked(MouseEvent e) {**

**if(e.getClickCount()==2){ // getClickCount()短时间点击次数**

**System.out.println("双击...");**

**}**

**}**

**}**

##### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("窗体");

//创建一个按钮

JButton button = new JButton("点我啊");

/\*

**//给按钮添加一个鼠标监听器**

**button.addMouseListener(new MouseListener() {**

**@Override**

**public void mouseReleased(MouseEvent e) {**

**System.out.println("mouseReleased鼠标松开..");**

**}**

**@Override**

**public void mousePressed(MouseEvent e) {**

**System.out.println("mousePressed鼠标按下...");**

**}**

**@Override**

**public void mouseExited(MouseEvent e) {**

**System.out.println("****mouseExited鼠标离开...");**

**}**

**@Override**

**public void mouseEntered(MouseEvent e) {**

**System.out.println("mouseEntered鼠标进入....");**

**}**

**@Override**

**public void mouseClicked(MouseEvent e) {**

**System.out.println("mouseClicked鼠标单击....");**

**}**

**})**

;\*/

**//MouseAdapter适配器----该类实现了MouseListener接口，但实现的方法全部都是空实现。**

**button.addMouseListener(new MouseAdapter() {**

**@Override**

**public void mouseClicked(MouseEvent e) {**

**if(e.getClickCount()==2){**

**System.out.println("双击了...");**

**}**

**}**

**});**

frame.add(button);

FrameUtil.initFrame(frame, 200, 200);

}

}

#### 键盘监听器KeyListener (){需写方法}

##### 需写的方法

**keyTyped 键入某个键**

**keyReleased 释放某个键**

**keyPressed 按下某个键**

#### 实用的方法

**e.getKeyChar 获取按下键的字符**

**e.getKeyCode 获取按下键的值**

**注：每个键都有自己的值，不会重复。**

##### 键盘监听配置器

**new KeyAdapter() {**

**@Override**

**public void keyPressed(KeyEvent e) {**

**System.out.println("按下键的字符："+ e.getKeyChar()+" 键的code:"+ e.getKeyCode());**

**}**

**}**

##### 实例：

public class Demo {

public static void main(String[] args) {

JFrame frame = new JFrame("窗体");

//创建一个按钮

JButton button = new JButton("点我啊");

/\*

**//给按钮添加一个键盘监听器**

**button.addKeyListener(new KeyListener() {**

**@Override**

**public void keyTyped(KeyEvent e) {**

**System.out.println(" 键入某个键...");**

**}**

**@Override**

**public void keyReleased(KeyEvent e) {**

**System.out.println("释放键 ...");**

**}**

**@Override**

**public void keyPressed(KeyEvent e) {**

**System.out.println("按下某个键...");**

**}**

**});**

\*/

**button.addKeyListener(new KeyAdapter() {**

**@Override**

**public void keyPressed(KeyEvent e) {**

**System.out.println("按下的键："+ e.getKeyChar()+" 键的code:"+ e.getKeyCode());**

**}**

**})**;

frame.add(button);

FrameUtil.initFrame(frame, 200, 200);

}

}

## 笔记本打开保存监听代码：

### 打开事件监听器

open.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

**FileDialog fileDialog = new FileDialog(frame, "选择需要打开的文件", FileDialog.LOAD);**

fileDialog.setVisible(true);

**String path = fileDialog.getDirectory(); //获取保存文件的路径**

**String fileName = fileDialog.getFile(); //获取保存文件的名字**

File file = new File(path,fileName);

try {

**//将文件内容写入到area文本中**

**FileReader fileReader = new FileReader(file);**

**BufferedReader bufferedReader =new BufferedReader(fileReader);**

**String line = null;**

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

**area.append(line+"\r\n");**

**}**

**bufferedReader.close();**

} catch (IOException e2) {

// TODO 自动生成的 catch 块

e2.printStackTrace();

}

}

});

### 保存事件监听器

save.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

**FileDialog fileDialog = new FileDialog(frame, "请输入保存的文件名", FileDialog.SAVE);**

fileDialog.setVisible(true);

**String path = fileDialog.getDirectory(); //获取保存文件的路径**

**String fileName = fileDialog.getFile(); //获取保存文件的名字**

File file = new File(path,fileName);

try {

//获取文本域的内容

**String content = area.getText().replaceAll("\n","\r\n"); //将所以的\n替换成\r\n**

**// JTextArea的getText方法遇到\r\n的时候只要了\n,把\r给去除。**

**FileWriter fileWriter = new FileWriter(file);**

**fileWriter.write(content);**

**fileWriter.close();**

} catch (IOException e1) {

e1.printStackTrace();

}

}

});

# 贪吃蛇实现

public class RetroSnaker extends JPanel {

private static final long serialVersionUID = 703945880410489582L;

// 宽

public static final int WIDTH = 30;

// 高

public static final int HEIGHT = 30;

// 每个单元格的宽

public static final int CELLWITH = 20;

// 每个单元格的高

public static final int CELLHEIGHT = 20;

// 地图

char [][] background = new char [HEIGHT][WIDTH];

// 蛇

private LinkedList<Point> snakes = new LinkedList<Point>();

//游戏是否结束

private boolean isGameOver = false; //默认没有结束

// 四个常量代表了四个方向

public static final int DIRECTION\_UP = 1; // 上

public static final int DIRECTION\_DOWN = -1; // 下

public static final int DIRECTION\_LEFT = 2; // 左

public static final int DIRECTION\_RIGHT = -2; // 右

// 当前的方向

private int currentDirecion = -2;

## // 游戏结束判断

public void isGameOver(){

//撞墙

Point head = snakes.getFirst();

if(background[head.y][head.x] == '\*'){

isGameOver = true;

}

//咬到了自己

for(int i = 1; i<snakes.size() ; i++){

Point body = snakes.get(i);

if(body.equals(head)){

isGameOver = true;

}

}

}

## // 蛇的描述

// 初始化蛇

public void initSnake() {

int x = WIDTH / 2;

int y = HEIGHT / 2;

snakes.addFirst(new Point(x - 1, y));

snakes.addFirst(new Point(x, y));

snakes.addFirst(new Point(x + 1, y));

}

// 移动

public void move() {

// 获取原始头的位置

Point head = snakes.getFirst();

// 添加新头

switch (currentDirecion) {

case DIRECTION\_UP:

snakes.addFirst(new Point(head.x, head.y - 1));

break;

case DIRECTION\_DOWN:

snakes.addFirst(new Point(head.x, head.y + 1));

break;

case DIRECTION\_LEFT:

snakes.addFirst(new Point(head.x - 1, head.y));

break;

case DIRECTION\_RIGHT:

snakes.addFirst(new Point(head.x + 1, head.y));

break;

}

if(eatFood()){

createFood();

initBackground();

showFood();

}else{

//删除蛇尾

snakes.removeLast();

}

repaint();

}

// 改变蛇方向的方法

public void changeDirection(int newDirection) {

if (newDirection + currentDirecion != 0) {

// 不是反方向

this.currentDirecion = newDirection;

}

}

## // 地图的描述

// 初始化地图

public void initBackground() {

for (int rows = 0; rows < HEIGHT; rows++) {

for (int cols = 0; cols < WIDTH; cols++) {

// 第一行、最后一行、 第一列、最后一列

if (rows == 0 || rows == (HEIGHT - 1) || cols == 0 || cols == (WIDTH - 1)) {

background[rows][cols] = '\*';

} else {

background[rows][cols] = ' ';

}

}

}

}

## // 食物的描述

//食物

private Point food;

//生成食物

public void createFood(){

Random random = new Random();

while(true){

int x = random.nextInt(WIDTH);

int y = random.nextInt(HEIGHT);

if(background[y][x]!='\*'){

food = new Point(x, y);

break;

}

}

}

//显示食物--- 把食物的坐标反馈到地图上，在地图上画上相应的字符

public void showFood(){

background[food.y][food.x] = '@';

}

//吃食物

public boolean eatFood(){

//获取蛇头

Point head = snakes.getFirst();

if(head.equals(food)){

showFood();

return true;

}else{

return false;

}

}

## // 画图

@Override

public void paint(Graphics g) {

// 地图和食物

for (int rows = 0; rows < HEIGHT; rows++) {

for (int cols = 0; cols < WIDTH; cols++) {

if (background[rows][cols]=='\*') {

g.setColor(Color.GRAY);

} else if(background[rows][cols]=='@'){

g.setColor(Color.YELLOW);

} else if(background[rows][cols]==' '){

g.setColor(Color.WHITE);

}

g.fill3DRect(cols \* CELLWITH, rows \* CELLHEIGHT, CELLWITH, CELLHEIGHT, true);

}

}

// 画蛇

// 画蛇头

Point head = snakes.getFirst();

g.setColor(Color.RED);

g.fill3DRect(head.x \* CELLWITH, head.y \* CELLHEIGHT, CELLWITH, CELLHEIGHT, true);

// 画蛇身

g.setColor(Color.GREEN);

for (int i = 1; i < snakes.size(); i++) {

Point body = snakes.get(i);

g.fill3DRect(body.x \* CELLWITH, body.y \* CELLHEIGHT, CELLWITH, CELLHEIGHT, true);

}

}

//主方法

public static void main(String[] args) {

JFrame frame = new JFrame("贪吃蛇");

final RetroSnaker snakeView = new RetroSnaker();

snakeView.initBackground();

snakeView.initSnake();

snakeView.createFood();

snakeView.showFood();

frame.add(snakeView);

//调用[图形化界面的工具类](#_图形化界面的工具类：用于生成居中窗体)：用于生成居中窗体

FrameUtil.initFrame(frame, CELLWITH \* WIDTH + 30, CELLHEIGHT \* HEIGHT + 50);

while(true) {

snakeView.move();

snakeView.isGameOver(); //每行走一步都应该判断蛇是已经死亡

if(snakeView.isGameOver==true){

System.out.println("游戏已经结束~88");

System.exit(0);

}

snakeView.repaint();

try {

Thread.sleep(300);

} catch (InterruptedException e1) {

// TODO 自动生成的 catch 块

e1.printStackTrace();

}

frame.addKeyListener(new KeyAdapter() {

@Override

public void keyPressed(KeyEvent e) {

int keyCode = e.getKeyCode();

switch (keyCode) {

case KeyEvent.VK\_UP:

snakeView.changeDirection(DIRECTION\_UP);

break;

case KeyEvent.VK\_DOWN:

snakeView.changeDirection(DIRECTION\_DOWN);

break;

case KeyEvent.VK\_LEFT:

snakeView.changeDirection(DIRECTION\_LEFT);

break;

case KeyEvent.VK\_RIGHT:

snakeView.changeDirection(DIRECTION\_RIGHT);

break;

default:

break;

}

}

});

}

}

}

# 正则表达式

正则表达式就是用于操作字符串的一些规则， 在正则表达式中使用了一些特殊的符号代表了字符串的规则。

## 预定义字符

. 任何字符（与行结束符可能匹配也可能不匹配）

\d 数字：[0-9]

\D 非数字： [^0-9]

\s 空白字符：[ \t\n\x0B\f\r]

\S 非空白字符：[^\s]

\w 单词字符：[a-zA-Z\_0-9] a-z A-Z 0-9 \_

\W 非单词字符：[^\w]

注意： 一个预定义字符只能匹配一个字符，除非预定义字符串配合了数量词使用。

## 数量词

X? 一次或一次也没有

X\* 零次或多次

X+ 一次或多次

X{n} 恰好 n 次

X{n,} 至少 n 次

X{n,m} 至少 n 次，但是不超过 m 次

## 范围词

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a 到 z 或 A 到 Z，两头的字母包括在内（范围）

注意：一个范围词没有配合数量词使用也只能匹配一个字符而已。

## 正则对字符串常见的操作：

**正则表达式就是用于操作字符串的规则。**

### 匹配：matches()

/\*匹配一个手机号

手机号码的规则：

1．首位必须是1.

2．第二位： 3 4 、5 、 7、 8、

3．长度 必须是11位

\*/

public static void matchPhone(String phone){

String reg = "1[34578]\\d{9}";

System.out.println(phone.matches(reg)?"合法手机号":"非法手机号");

}

### 切割：split()

//需求： 根据空格切割。

public static void splitTest1(){

String str = "明 天 学 i o";

String reg = " +";

String[] datas = str.split(reg);

System.out.println("数组的元素："+ Arrays.toString(datas));

}

### 替换：replaceAll(String regex, String replacement)

//需求： 把重叠词替换成单个字符。

public static void repalce2(){

String str = "我我要要要成成为为高富富富富帅帅帅帅帅帅";

str = str.replaceAll("(.)\\1+", "$1");

**//注意引用正则的内容如果不是在一个正则表达式内部，那么需要使用："$组号" 进行引用**

System.out.println(str);

}

### 正则查找：

如果正则表达式需要查找内容需要使用以下两个对象：

Pattern(正则对象)

Matcher（匹配器对象）

指定为字符串的正则表达式必须首先被编译为此类Pattern的实例。然后，Pattern对象可以与任意字符序列匹配 ，匹配字符串之后就可以得到一个 Matcher 对象，

所涉及与字符串匹配的状态都驻留在匹配器中 ．

**典型的调用顺序是**

**Pattern p = Pattern.compile("正则表达式");**

**Matcher m = p.matcher("需查找的字符串");**

**boolean b = m.matches();**

**如：**

**//找出三个字符构成的单词。**

**public static void main(String[] args) {**

**String data = "hou tian you ke yi fang jia liao ,da jia shuang ma";**

**String reg = "\\b[a-z]{3}\\b";**

**//第一步：先要把字符串的正则编译成Pattern对象**

**Pattern p = Pattern.compile(reg);**

**//第二步： 使用正则对象去匹配字符串，得到一个matcher 对象。**

**Matcher m = p.matcher(data);**

**while(m.find()){**

**System.out.println(m.group());**

**}**

**}**

单词边界匹配器: 单词边界匹配器不匹配 任何 的字符，只是代表了一个单词的开始或者结束部分。

**\b 单词边界**

**(\\.[a-z]{2,3}){1,2} ()组后也可以加数量词，用于表示多次出现**

#### 匹配器常用的方法：

**find()** 通知匹配器去查找符合该正则的字符串。如果存在符合规则的字符串返回true，否则返回false．

**group()** 获取符合规则的字符串。

**注意：使用匹配器的方法时候，要先调用find方法才能调用group方法。 否则匹配器没有去查找合适的内容，报错。**

# 泛型

泛型是jdk1.5出现的新特性。

## 泛型的好处:

1．可以把运行时出现的问题提前至编译时。

2．避免了无谓的强制类型转换。

## 使用泛型要注意的事项：

**1．泛型中没有多态的概念,左右两边的数据类型必须要一致，或者是只写一边的泛型。**

推荐使用： 两边都写上同样数据类型的泛型。

## 自定义泛型

自定义泛型可以理解为是一个数据类型的变量或者是一个数据类型的占位符。

### 函数自定义泛型的格式：

修饰符 <声明自定义泛型> 返回值类型 函数名(形参列表 ...){ }

### 函数自定义泛型需注意的细节：

1．函数上自定义泛型的具体数据类型是在调用该函数的时候，传递实参数据的时候确定具体的数据类型的。

2．自定义泛型使用的标识符可以自定义的，只要符合标识符的命名规则即可。但是一般自定义泛型使用的标识符都是单个字母而已。

**泛型中是不能使用基本数据类型数据的，如果需要使用基本数据类型数据，那么就要使用基本数据类型对应的包装类型**

int -------->Integer

float ------->Float

double ---- >Double

char -------> Character

boolean ----> Boolean

byte ----->Byte

short ---->Short

long -----> Long

## 自定义泛型类

### 自定义泛型类的格式：

class 类名<自定义的泛型>{ }

### 泛型类要注意的事项：

1．类上声明的自定义泛型的具体数据类型是在使用该类创建对象的时候确定的。

2．如果一个类已经声明了自定义泛型，该类在创建对象的时候没有指定自定义泛型的具体数据类型，那么默认则为Object类型。

3．静态的方法不能使用类上声明的自定义泛型，如果需要使用自定义泛型只能在自己方法上声明。

## 泛型接口

### 泛型接口的定义格式：

interface 接口名<声明自定义的泛型>{ }

### 泛型接口要注意的细节:

1．接口上自定义泛型的具体数据类型是在实现该接口的时候确定的。

2．如果一个接口已经自定义了泛型，在实现该接口的时候没有指定自定义泛型的具体数据类型，那么默认为Object类型。

泛型的上下限

**泛型的通配符： ？**

**? super** Integer **泛型的下限** 只能用于Integer或者是Integer的父类类型数据

**? extends** Number **泛型的上限** 只能用于Number或者是Number的子类类型数据。

需求1： 定义一个方法可以接受任意类型的集合对象， 接收的集合对象只能存储Integer以及Integer父类类型的数据。

? super Integer 泛型的下限 只能用于Integer或者是Integer的父类类型数据

需求2： 定义一个方法可以接受任意类型的集合对象， 接收的集合对象只能存储Number或者Number子类类型的数据。

? extends Number 泛型的上限 只能用于Number或者是Number的子类类型数据。

# 工具类

## Arrays数组工具类：

### 数组排序

sort(int[])

sort(char[])……

### 二分查找,数组需要有序

binarySearch(int[])

binarySearch(double[])

### 将数组变成字符串。

toString(int[])

### 复制数组

copyOf(源数组，新数组长度);

copyOfRange(源数组，起始位置，结束位置);

### 比较两个数组是否相同。

equals(int[],int[]);

### 将数组变成集合。

List asList(T[]);

这样可以通过集合的操作来操作数组中元素，

但是不可以使用增删方法，add，remove。因为数组长度是固定的，会出现

UnsupportOperationExcetion。

可以使用的方法：contains，indexOf。。。

如果数组中存入的基本数据类型，那么asList会将数组实体作为集合中的元素。

如果数组中的存入的引用数据类型，那么asList会将数组中的元素作为集合中

的元素。

## Collections集合工具类：

### 1.对list集合进行排序。

**sort(list);**

//对list进行排序,其实使用的事list容器中的对象的compareTo方法

**sort(list,comaprator);** 按照指定比较器进行排序

### 2.对list进行二分查找：前提该集合一定要有序。

**int binarySearch(list,key);**

//必须根据元素自然顺序对列表进行升级排序

//要求list 集合中的元素都是Comparable 的子类。

**int binarySearch(list,key,Comparator);**

### 3.对集合取最大值或者最小值。

max(Collection)

max(Collection,comparator)

min(Collection)

min(Collection,comparator)

### 4.对list集合进行反转。

reverse(list);

### 5.对list集合中的元素进行位置的置换。

swap(list,x,y);

### 6.对list集合进行元素的替换。如果被替换的元素不存在，那么原集合不变。

replaceAll(list,old,new);

### 7.可以将不同步的集合变成同步的集合。

Set synchronizedSet(Set<T> s)

Map synchronizedMap(Map<K,V> m)

List synchronizedList(List<T> list)

**如果想要将集合变数组：**

**可以使用Collection 中的toArray 方法。注意：是Collection不是Collections工具类**

**传入指定的类型数组即可，该数组的长度最好为集合的size。**

## Math 数学类

### 常用的方法：

abs(double a) 绝对值

ceil(double a) 向大取整

floor(double a) 向小取整

round(double a) 四舍五入

random() 生成随机数范围0.0 ~ 1.0 不包括1.0

System.out.println("绝对值：" + Math.abs(-3.14));

System.out.println("向上取整：" + Math.ceil(-3.14));

System.out.println("向下取整：" + Math.floor(3.14));

System.out.println("四舍五入："+ Math.round(3.55));

System.out.println("随机数："+ Math.random());

## Random 随机数类

int [nextInt](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/util/Random.html#nextInt(int))(int n) 返回一个伪随机数，随机数范围是0-n

**Random** random = new Random();

System.out.println("随机数："+ random.nextInt(11)); // 随机数范围是0-10

题目需求： 实现一个四位的验证码。

public class Demo6 {

public static void main(String[] args) {

char[] arr = {'a','A','中','雨','共','W','O','1','9','4'};

Random random = new Random();

//创建一个字符串缓冲区类

StringBuilder sb = new StringBuilder();

//产生四个随即的索引值。

for(int i = 0 ; i < 4 ; i++){

int index = random.nextInt(arr.length);

char temp = arr[index];

sb.append(temp);

}

System.out.println("验证码："+ sb);

}

}

## Runtime类

代表了当前程序的运行环境。

Runtime对象需要掌握方法：

Process exec(String command) 执行指定路径下的可执行文件。返回子进程子进程对象

long maxMemory() 返回 Java 虚拟机试图使用的最大内存量。

long totalMemory () 返回 Java 虚拟机中当前的内存总量。

long freeMemory () 返回 Java 虚拟机中的空闲内存量。

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

Runtime runtime = Runtime.getRuntime(); //获取Runtime对象

**Process p = runtime.exec("C:\\Windows\\notepad.exe");**

Thread.sleep(3000); //让当前程序暂停3秒钟

**p.destroy(); //杀死进程。**

System.out.println("试图使用的最大内存量:"+ runtime.maxMemory());

System.out.println("Java 虚拟机中的内存总量："+ runtime.totalMemory());

System.out.println("当前空闲的内存："+ runtime.freeMemory());

## 日期类

### Date 日期类（已过时但仍可用）

Date date = new Date(); 获取当前系统时间的对象。

……详情见JAVA API

### Calendar 日期类（）

Calendar calendar = Calendar.getInstance(); 获取了当前的系统时间

System.out.println("年份："+ calendar.get(Calendar.YEAR)); 年

System.out.println("月份："+ (calendar.get(Calendar.MONTH)+1)); 月（西方是0-11）

System.out.println("日："+ calendar.get(Calendar.DATE)); 日

……详情见JAVA API

### SimpleDateFormat 日期格式化类：

作用：

1．可以将时间对象转成指定格式的字符串。 format();

2．可以把字符串转成日期对象 parse();

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

//创建日期格式化类对象。 y、M、d、H、m、s都是日期格式符。详情见JAVA API 中的SimpleDateFormat

//把日期对象转换指定格式的字符串的方法 **format**

String text = dateFormat.format(new Date());

System.out.println("当前系统时间："+ text);\*/

**字符串--> 时间对象 字符串的格式必须要与SimpleDateformat指定的模式要一致，否则报错。**

String text = "1990年09月08日 07:01:00";

Date date = dateFormat.parse(text);

System.out.println(date);

## Object 类:

Object类是所有类的终极父类。

Object类常用的方法：

toString() 返回的字符串是用于描述该对象的。(返回：包名+@+内存地址)

equals(Object obj) 判断两个对象是否为同一个对象。（默认比较内存地址）

hashCode() 返回一个对象的哈希码值（内存地址）。（返回：内存地址）

java规范: 一般我们在重写equals方法的时候，我们都会重写hashCode方法。

为了整体性，equals判断为同一对象时，hashCode调用两个相同的对象时应该要是相同的。

查看源代码的方式：

方式一： 按住ctrl+ 鼠标单击

方式二： 按下f3

## System系统类

主要的作用是用于获取系统的一个参数。

### System类需要掌握的方法：

void arraycopy(Object src, int srcPos, Object dest, int destPos, int length) 拷贝数组

src 源数组。

srcPos 源数组中的起始位置。

dest 目标数组。

destPos 目标数据中的起始位置。

length 要复制的数组元素的数量。

long currentTimeMillis() 获取当前的系统时间

void exit(int status) 退出jvm，0表示正常退出jvm，非0 表示异常退出

void gc() 建议jvm尽快的启动垃圾回收器回收垃圾

getenv(String name) 获取环境变量

[Properties](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/util/Properties.html) getProperties() 获取系统属性

String getProperty(String key) 获取指定键指示的系统属性

## Integer类

int [parseInt](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/lang/Integer.html#parseInt(java.lang.String))([String](mk:@MSITStore:C:\Users\Guo\Desktop\JAVA%20API【中文】.CHM::/java/lang/String.html) s) 将字符串参数转换成十进制整数。

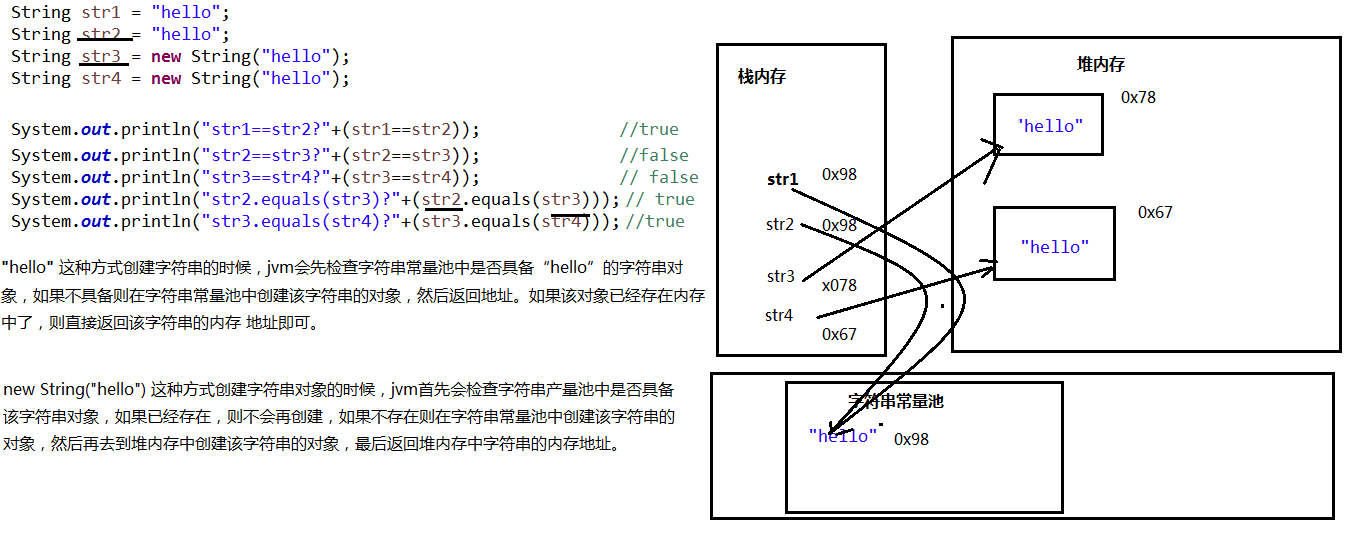
## String 类

### 字符串的比较规则：

如果两个字符串可以找到对应位置上不同的字符，那么就比较第一个不同的字符。

如果找不到对应位置上不同的字符，那么则对比两个字符串的长度。

### 字符串创建的方式



在String中比较字符串需要用equals去比较。因为在String中equals被重写成了比较字符串的每个字符。

### String类的构造方法：

String() 创建了一个空字符串对象

String("字符串") 构建一个指定的字符串

String(byte[] bytes) 使用byte数组构建字符串（ byte是字符对应的码值，97=’A’）

String(char[] value) 使用字符数组构建字符串

**String(byte[] bytes, int offset, int length)**

**使用byte字节数组构建字符串对象，指定开始使用的位置索引值与使用的长度**

String(char[] value, int offset, int count)

使用字符数组构建字符串对象，并且指定开始使用的位置索引值与使用的字符个数

**总结： 使用字节数组与字符数组是可以构建一个字符串对象的。**

### String类的获取方法

int length() 获取字符的个数

char charAt(int index) 指定索引值查找字符(不可越界)

int indexOf(String str) 查找指定字符串第一次出现的索引值

int lastIndexOf(int ch) 查找指定字符串最后一次出现的索引值

### String类的判断方法

boolean endsWith(String str) 是否以指定的字符或串结束

boolean isEmpty() 是否长度为0，指针指NULL时会报错

boolean contains(CharSequences) 判断字符串是否包含指定的字符或串

boolean equals(Object anObject) 判断的内容是否一致

boolean equalsIgnoreCase(String anotherString) 忽略大小写比较的内容是否一致

### String类的转换方法

char[] toCharArray() 将字符串转换为字符数组

**byte[] getBytes() 把字符串转换字节数组、**

### String类的其他方法

String replace(char oldChar, char newChar) 替换字符或串

String[] split(String regex) 指定分隔符进行切割

String substring(int beginIndex) 指定开始的索引值截取字符串

String substring(int beginIndex, int endIndex) 指定开始与结束的索引值截取字符串

注意 : 截取的位置是从beginIndex开始，末尾是endIndex-1(包头不包尾)

String toUpperCase() 转大写

String toLowerCase() 转小写

String trim() 去除空格

## StringBuffer（字符串缓冲区类）

如果一个字符串的内容需要频繁修改的时候，这时候我们则需要使用字符串缓冲区类。

下例方法没写返回值的返回值为StringBuffer

### StringBuffer类的增加方法

StringBuffer("jack") 在创建对象的时候赋值

append() 在缓冲区的尾部添加新的文本对象

insert() 在指定的下标位置添加新的文本对象

### StringBuffer类的删除方法

delete(int start, int end) 指定开始与结束的索引值删除字符串缓冲区的内容

deleteCharAt(int index) 指定索引值删除字符

### StringBuffer类的查找方法

String toString() 返回这个容器的字符串

int indexOf(String str) 返回指定子字符串第一次出现在该字符串中的索引

String substring(int start) 从开始的位置开始截取字符串

### StringBuffer类的修改方法

replace(int start，int end，String str)

使用给定字符串str替换此序列的子字符串start ~end-1中的字符。

void setCharAt(int index char ch) 指定索引位置替换一个字符

**笔试题： 使用Stringbuffer无参的构造方法默认的初始化容量是多少？ 当容量不够用时会自动增长多少？**

StringBuffer的底层是维护了一个字符数组，存储字符的时候实际上是往该字符数组中存储的，而字符数组的初始化容量是16， 当容量不够使用时，自动增长1倍（1倍+2）。

## StringBuilder（字符串缓冲区类）

操作同StringBuffer一样是存储字符的一个集合容器，字符串缓冲区类。

推荐使用： StringBuilder

**笔试题目： StringBuffer 与 StringBuilder的区别 ？**

*相同点：*

都是字符串缓冲类，底层都是维护了一个字符数组用于存储数据的。

*不同点：*

1．StringBuffer 是线程安全的，操作效率低。 StringBuilder是线程非安全的，操作效率高。

2．StringBuffer 是jdk1.0的时候出现的， StringBuilder是jdk1.5的时候出现。

# 集合

集合： 集合存储任意对象数据的容器。

## 集合的特点：

1．集合可以存储任意类型的对象数据

2．集合的长度是会发生变化的。

## Collection集合体系：

---| Collection 单列集合的根接口

------| List 如果是实现了List接口的集合类具备的。 特点：有序，元素可重复。

---------| ArrayList ArrayList的底层是维护了一个Object数组去实现的。

---------| LinkedList LinkedList的底层是使用了链表数据结构实现的。

---------| Vector (了解) Vector的底层也是使用一个Object数组去实现的。

------| Set 如果是实现了Set接口的集合类具备的。 特点：无序，元素不可重复。

---------| HashSet 底层是使用了哈希表支持的。 特点：存取的速度快。

---------| TreeSet 底层是使用了二叉树数据结构实现。 特点：可以对元素进行排序存储。

**有序：在集合中所谓的“有序”不是指自然顺序，而是指添加进去的顺序与存储的顺序一致。**

**笔试题目：使用 ArrayList无参的构造方法是默认的容量是多少？ 当容量不够使用时，自动回增长多少？**

ArrayList的底层是使用了一个Object数组去实现的，往ArrayList存储数据的时候，数据实际上是存储到了Object数组中， 使用无参构造函数是，Object数组的初始化容量是10， 当容量不够使用时会自动自增原来的0.5倍。

**笔试题目： Vector 与 ArrayList的区别：**

相同点： Vector与ArrayList底层都是使用Object数组去实现的。

不同点：

1．Vector是线程安全的，操作效率低， ArrayList是线程非安全的，操作效率高。

2．Vector是jdk1.0出现 的， ArrayList是jdk1.2出现的。

### ArrayList

ArrayList的底层是维护了一个Object数组去实现的。特点： 查询速度快，增删慢。

应用场景：数据的查询多，增删少时使用。如：图书馆

### LinkedList

LinkedList的底层是使用了链表数据结构实现的。特点： 查询速度慢， 增删快。

#### LinkedList特有的方法：

1：方法介绍

addFirst(E e) 把元素添加到集合的首位置

addLast(E e) 把元素添加到集合的末尾处。

getFirst() 获取集合的首位置元素

getLast() 获取集合的末尾元素

removeFirst() 删除集合的首元素

removeLast() 删除集合的末尾元素

2：数据结构

1：栈 （1.6）

先进后出

push() 入栈

pop() 出栈

2：队列（双端队列1.5）

先进先出

offer() 入队

poll() 出队

3：返回逆序的迭代器对象

descendingIterator() 返回逆序的迭代器对象

### Vector

Vector的底层也是使用一个Object数组去实现的。但是Vector线程安全的，操作效率低。

### hashSet要注意的事项：

往hashSet添加元素的时候，首先会调用元素的hashCode方法得到元素的哈希码值，通过哈希码值就可以算出该元素在哈希表中的存储位置 。

情况1： 如果根据元素的哈希码算出的位置目前没有任何元素存储在该位置上，那么该元素可以直接添加到哈希表中。

情况2： 如果根据元素的哈希码算出的位置目前已经有其他元素存储在了该位置上，那么还会调用元素的equals方法再与这个位置上的元素再比较一次，如果equals方法返回的是true，则视为重复元素，不允许添加到哈希表中，如果equals方法返回的是false，该元素则允许添加到哈希表中。

### TreeSet 要注意的事项：

1．如果往TreeSet添加元素的时候，如果元素本身具备自然顺序的特性，那么treeSet就会按照元素自然顺序的特性进行排序存储。

2．如果往TreeSet添加元素的时候， 如果元素本身不具备自然顺序的特性, 那么元素所属的类就必须要实现Comparable接口，把元素的比较规则定义在compareTo(T o)方法上。

3．在TreeSet中如果比较的方法返回的是0，该元素则被视为重复元素，不允许添加。

4．往TreeSet添加元素的时候，如果元素本身不具备自然顺序的特性，而且元素所属的类没有实现Comparable接口，那么在创建TreeSet对象 的时候就必须要传入一个比较器对象。

**推荐使用： 比较器。**

**比较器的创建格式：**

**class 类名 implements Comparator{ }**

## Colllection要掌握方法：

### 增加:

add(E e) 加一个元素

addAll(Collection c) 把一个集合的元素添加到另外一个集合容器上。

### 删除：

clear() 清除集合的元素

remove(Object o) 删除指定的元素

removeAll(Collection c) 删除两个集合中交集元素

retainAll(Collection c) 保留两个集合中的交集元素，其他的元素删除。

### 查看：

contains(Object o) 如果有指定元素，则返回 true，否则返回false.

containsAll(c) 如果包含指定集合中的所有元素，则返回 true，否则返回false.

isEmpty() 如果Collection不包含任何的元素，则返回true，否则返回false.

size() 查看集合中的元素个数

### 迭代 ：

toArray() 把集合中的元素存储到一个Object数组中返回。

iterator() 获取集合中的迭代器 。 迭代器的作用就是用于抓取集合中的元素。

iterator() **迭代器常用的方法：**

hasNext() 当前游标指向的位置是否有元素

next() 获取当前游标指向的元素，然后游标向下移动一个单位。

remove() 移除迭代器最后一次返回的元素。

## List接口下面特有的方法：

### 增加

add(int index, E element) 指定索引值添加元素

addAll(int index, Collection c) 指定索引值把集合c添加到调用集合中

### 删除

remove(int index) 指定索引值删除元素

### 修改

set(int index, E element) 指定索引值修改元素

### 获取

get(int index) 根据索引值获取元素

indexOf(Object o) 查找返回指定元素第一次出现的索引值, 如果没有则返回-1

lastIndexOf(Object o) 查找返回指定元素最后一次出现的索引值，如果没有则返回-1

subList(int fromIndex, int toIndex) 返回指定开始和结束的索引值截取集合中的元素

### 迭代listIterator()

#### listIterator特有的方法：

next() 先获取当前游标指向的元素，然后游标向下移动一个单位。

hasPrevious() 判断是否有上一个元素

previous() 游标先向上移动一个单位，然后获取当前游标指向的元素。

add(E e) 把元素添加到当前游标指向的位置上。

set(E e) 使用指定的元素替代迭代器最后一次返回的元素。工具类

**迭代器在迭代的过程中要注意的事项：**

1．迭代器在迭代的过程中不准使用集合对象改变集合的元素个数。 否则会报错：ConcurrentModificationException

2．在迭代过程中如果需要改变集合中的元素个数，只能使用迭代器的方法去改变。

# 双列集合

实现了Map接口的集合类具备的特点：存储的数据都是以键值对的形式存在的， 键(key )不能重复，值（value）可以重复

## 体系

---| Map 中的元素是两个对象，键不可以重复，但是值可以重复

------| HashMap 底层是哈希表数据结构。 特点：查询快，增删慢

------| TreeMap 底层是二叉树数据结构。 特点：查询慢，增删快

------| Hashtable(了解)

## 注意：

1.如果往TreeMap添加元素的时候， 如果键key本身不具备自然顺序的特性, 那么key所属的类就必须要实现Comparable接口。

2.往TreeMap添加元素的时候，如果键key本身不具备自然顺序的特性，而且键key所属的类没有实现Comparable接口，那么在创建TreeMap对象 的时候就必须要传入一个比较器对象。

**总体来说和单列集合是一致的**

## Map接口的方法:

### 添加

V put(K key, V value) （可以相同的key值，但是添加的value值会覆

盖前面的，返回值是前一个，如果没有就返回null）

putAll(Map<? extends K,? extends V> m) 从指定映射中将所有映射关

系复制到此映射中（可选操作）。

### 删除

remove(value) 删除关联对象，指定key对象

clear() 清空集合对象

### 获取

value get(key) 根据键key获取对应的值value

int size（） 查看map集合的元素个数

### 判断

boolean isEmpty() 长度为0返回true否则false

boolean containsKey(Object key) 判断集合中是否包含指定的key

boolean containsValue(Object value) 判断集合中是否包含指定的value

## 迭代

### keySet()

**返回所有的key对象的Set集合再通过get方法获取键对应的值。**

**遍历方式一： 可以使用keySet方法遍历。 keySet方法返回的集合只是包含了所有的键，没有包含值。**

Set<String> set = map.keySet(); //把map集合中所有key的数据存储到Set集合中返回。

Iterator<String> it = set.iterator();

while(it.hasNext()){

String key = it.next();

System.out.println("键："+ key+" 值："+ map.get(key));

}

### values()

**获取所有的值**

**遍历的方式二：可以使用values方法进行遍历。 values方法返回的集合对象只是包含了map中的所有值，没有键的数据。**

Collection<String> list = map.values(); // values 把map集合中的所有值存储到一个Collection集合返回。

Iterator<String> it= list.iterator();

while(it.hasNext()){

String value = it.next();

System.out.println("值："+ value);

}

### entrySet()

**将map 集合中的键值映射关系打包成一个对象Map.Entry对象通过Map.Entry 对象的getKey，getValue获取其键和值。**

**遍历的方式三： 使用 entrySet方式遍历。 推荐使用。**

Set<Map.Entry<String,String>> entrys = map.entrySet();

Iterator<Map.Entry<String,String>> it = entrys.iterator();

while(it.hasNext()){

Map.Entry<String,String> entry = it.next();

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

}

# 关键字

this 关键字代表本类对应的引用。

super 关键字代表父类存储空间的标识。

final（最终） 常量

final关键字要注意的细节 ：

1．final关键字修饰一个基本类型变量时， 该变量不能重新赋值。

2．final关键字修饰一个引用类型变量时，其引用重新指向新的对象。

3．final修饰一个方法时，该方法不能被重写。

4．final修饰一个类的时候，该类不能被继承。

常量的修饰符: public static final．

常量命名规范： 所有字母大写，单词与单词之间使用下划线分隔 。

## **extends** 继承

使用格式：class 类名1 extends 类名2{ }

继承要注意的细节：

1． 子类可以继承父类的成员,但是千万不要为了减少重复代码而且继承，只有 真正存在继承关系的情况下才去继承。

2．父类私有的成员是不能被继承的。

3．父类的构造方法也是不能被继承的。

4．创建子类对象时默认会先调用父类的无参的构造函数。

## abstract 抽象

使用格式：

abstract class 类名{

public abstract void 方法名();

}

抽象类的应用场景：

描述一类事物的时候，发现该事物确实存在着某种行为，但是目前该行为是不具体的，

那么这时候我们应该抽取该方法的声明，不去实现该方法，这时候我们应该使用抽象类。

abstract不能与以下关键字配合使用：

1．abstract 不能与private配合使用。

2．abstract 不能与final关键字配合使用。

3．abstract 不能与static关键字配合使用。

## Instanceof 判断

instanceof关键字： 判断一个对象是否属于指定的类型。

使用前提：判断的对象与指定的类型必须存在继承的关系，或者是实现的关系。

instanceof关键字的使用格式：

对象 instanceof 类名

## Interface 接口

接口的定义格式：

interface 接口名{ }

实现接口的格式：

class 类名 implements 接口{ }

接口要注意的细节：

1．接口其实是一个特殊的类。

2．接口中的成员变量都是属于常量，默认的修饰符 public static final.

3．接口中的方法都是抽象 的方法， 默认的修饰符： public abstract 。

4．接口是不能用于创建对象。

5．接口是没有构造方法的，因为接口中的变量都是常量，接口中方法都是抽象的方法。

6．一个非抽象类实现一个接口时，必须实现接口中的所有方法。

# 异常处理

----| Throwable

---------| Error 错误一般都是由于jvm或者是硬件引发的问题，一般不会通过代码去处理。

---------| Exception 异常 如果程序出现了异常，那么一般就需要通过代码去处理了。

Throwable常用的方法：

1．toString() 返回的是用于描述该异常情况的类的完整类名。 包名+类名 = 完整类名。

2．getMessage() 返回创建Throwable对象的时候传入的消息字符串的,

3．printStackTrace() 打印异常的栈信息。

## 捕获处理

捕获处理的格式：

try{

可能会发生异常的代码

}catch(异常的类型 变量名){

异常处理代码；

}

## 抛出异常

throw和throws的用法：

throw 定义在函数内，用于抛出异常对象。

throws 定义在函数上，用于抛出异常类，可以抛出多个用逗号隔开。

## finally 块

使用前提：必须配合try块使用，不能单独使用。

finally块在任何情况下都可以执行，只有的java虚拟机退出时无法执行。

使用格式：

try{

可能会发生异常的代码

释放资源；（因为有异常，所以后面的释放资源没有执行）

}catch(异常的类型 变量名){

异常处理代码；

} finally{

释放资源；（帮助释放资源）

}

# 包

目前存在的问题：

1．如果两个java文件的类名一致，那么后编译的class文件就会覆盖 前面编译的class文件。

包：java中的包相当于windows文件夹。

包的好处：

1．解决类名重复class文件覆盖的问题。

2．方便软件版本的发布。

## package 建包

包语句的格式：

package 包名;

存在的问题: 有了包语句之后，每次编译成功我们都需要新建一个文件夹，把对应的class文件拖进去。

解决方案： 只需要在编译的时候加上-d 参数即可。

格式： javac -d 存放class文件的路径 java源文件

包要注意的事项：

1．一个java文件如果有了包语句，那么该类文件的完整类名是"包名+ 类名":

2． 一个java文件只能有一个package语句。

3．package语句必须位于java文件中的第一个语句。

## import 导包

导包语句的作用是简化书写。

导包语句的格式：

import 包名.类名;

导包语句要注意的细节：

1．一个java文件可以出现多个import语句。

2．导包的时候可以使用通配符"\*"来匹配类名。"\*"可以匹配任何的类名。

3．import bb.\*; 该语句不会作用于bb包下面的子包。

推荐不要使用通配符导包，因为结构不清晰。

# 打jar包

如何将class文件打包成jar文件呢？

需要使用到jdk的开发工具jar.exe

## jar工具的使用格式：

jar cvf jar包名 class文件名|包名

## jar要注意的事项：

1．打jar包 的时候必须要指定jar包的入口类，入口类的指定是在清单文件上去指定的，

格式： Main-Class: 包名.类名

2．jar双击运行仅对于图形化界面程序起作用，对于控制台 程序不起作用。

3．如果使用jar包中的类文件，一定要先设置好classpath路径。

## jar的作用：

1．可以将一个项目的class文件打包，方便用户运行项目

2．可以将工具类打包给jar提供给其他人去使用。

# bat处理

jar文件双击运行仅对于图形化界面的程序起作用，对于控制台程序是不起作用。

bat处理文件： bat处理文件就是指一次性可以执行多个命令的文件。

## bat处理文件如何编写 ：

新建一个记事本，然后把后缀名改成bat即可，然后把要执行的命令写在bat文件上。

## bat处理文件常用的命令：

echo 向控制台输出指定的语句.

title 设置控制台标题。

color 设置控制台的颜色

echo off 一个bat处理文件如果执行了echo off命令， 那么echo off以下的命令都会隐藏起来，只显示结果。

@ 隐藏当前行的命令。

rem 表示此命令后的字符为解释行（注释），不执行，只是给自己今后参考用的（相当于程序中的注释） 或者%注释的内容%

pause; 让控制台停留下来。

%[1-9] 表示参数，变量。

# 反射技术

**反射这门技术在做一些比较通用的工具类的开发的时候，就非常有用了。**

**反射： 当一个class文件被加载到内存中的时候，那么jvm就会马上对该class文件的成员进行解剖，然后将class文件的成员数据封装到一个Class对象中，我们如果可以获取到Class对象，那么我们就可以通过该Class对象来操作该类的所有成员。**

**注意：在反射技术中一个类的任何成员都使用了一个类来描述。**

## Class对象的获取方式：

**推荐使用方式1：通过Class.forName获取Class对象.**

### 实例：

**//方式1：通过Class.forName获取Class对象.**

Class clazz1 = Class.forName("cn.itcast.reflect.Person");

System.out.println("clazz1:"+clazz1.getSimpleName());

**//方式2：可以通过类名获取Class对象。**

Class clazz2 = Person.class;

System.out.println("是同一个对象吗？"+(clazz1==clazz2));

**//方式3：可以通过对象获取Class对象**

Class clazz3 = new Person().getClass();

System.out.println("是同一个对象吗："+ (clazz2 == clazz3));

## 通过Class对象获取构造方法

**Constructor类： 该类是用于表述一个构造方法的。**

**getConstructors() 获取了公共的构造方法**

**getDeclaredConstructors() 获取所有构造方法，包括私有的构造方法。**

**getConstructor() 获取单个构造方法**

**getDeclaredConstructor() 获取单个私有的构造方法**

**setAccessible (true) 设置构造方法的访问权限**

**newInstance(对象残数) 创建对象**

### 实例：

public class Demo {

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

Class clazz = Class.forName("cn.itcast.reflect.Person");

//通过class对象找到所有的构造方法。

/\*

**//只获取了公共的构造方法。**

**Constructor[] constructors = clazz1.getConstructors();**

**//获取一个类的所有构造方法，包括私有的构造方法。**

**Constructor[] constructors = clazz1.getDeclaredConstructors();**

for(Constructor constructor : constructors){ //输出显示所有构造方法

System.out.println(constructor);

}

**//获取单个构造方法**

**Constructor constructor = clazz.getConstructor(String.class,int.class);**

**//通过Constructor对象创建对象。**

**Person p = (Person) constructor.newInstance("狗娃",12);**

**System.out.println(p); //输出显示所有构造方法**

\*/

**// 获取私有的构造函数**

**Constructor constructor = clazz.getDeclaredConstructor(null);**

**//设置构造方法的访问权限（暴力反射）**

**constructor.setAccessible(true);**

Person p = (Person) constructor.newInstance(null);

System.out.println(p);

}

}

## 通过Class对象获取成员函数

**Method类： 该类是用于表述一个成员函数的。**

**getMethods 获取所有的方法，获取所有公共的方法，包括继承下来的方法。**

**getDeclaredMethods 获取所有的方法，包括私有的，但是不包括继承下来的方法。**

**获取单个方法**

**getMethod("eat", int.class,String[].class) 获取公共的方法**

**第一个参数是方法名， 第二参数是形参列表的数据类型。**

**getDeclaredMethod("study", null) 获取私有的方法**

**第一个参数是方法名， 第二参数是形参列表的数据类型。**

**执行方法**

**invoke(p, 3,new String[]{"aa","bb"})**

**第一个参数：方法的调用者对象， 后面参数： 方法执行所需要的参数。**

### 实例：

public class Demo {

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

Class clazz = Class.forName("cn.itcast.reflect.Person");

/\*

**//获取所有的方法，获取所有公共的方法，包括继承下来的方法。**

**Method[] methods = clazz.getMethods();**

**//获取所有的方法，包括私有的，但是不包括继承下来的方法。**

**Method[] methods = clazz.getDeclaredMethods();**

for(Method m : methods){

System.out.println(m);

}

\*/

Person p = new Person("狗娃",12);

**//获取单个方法**

**//第一个参数是方法名， 第二参数是形参列表的数据类型。**

**Method m = clazz.getMethod("eat", int.class,String[].class) ;**

**//执行一个方法**

**Person p = new Person("狗娃",12);**

**//第一个参数：方法的调用者对象， 后面参数： 方法执行所需要的参数。**

**m.invoke(p, 3,new String[]{"aa","bb"});**

Method m = clazz.getMethod("sleep", int.class);

m.invoke(null, 23);

**//私有的方法**

**Method m = clazz.getDeclaredMethod("study", null);**

**//设置方法的访问权限是可以访问**

**m.setAccessible(true);**

**m.invoke(p, null); //执行p方法**

}

}

## 通过Class对象获取一个类的成员变量

**Field类： 该类是用于表述一个成员变量的。**

**getDeclaredFields() 获取该类的所有成员变量**

**getDeclaredField(成员变量名) 获取指定成员变量**

**set(p, "铁蛋"); 设置成员变量第一个参数： 对象， 第二参数：成员变量的值。**

### 实例：

public class Demo {

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

Class clazz = Class.forName("cn.itcast.reflect.Person");

**//获取该类的所有成员变量**

**Field[] fields = clazz.getDeclaredFields();**

for(Field field : fields){

System.out.println(field);

}

Person p = new Person("狗剩",12);

**Field nameField = clazz.getDeclaredField("name");**

**nameField.set(p, "铁蛋"); // 第一个参数：对象 第二参数：成员变量的值。**

System.out.println(p);

}

}

# 设计模式

## 单设计模式

**用于保证一个类在内存中只有一个对象。**

**推荐使用：饿汉单例设计模式。懒汉单例设计模式存在线程安全问题。**

### 饿汉单例设计模式

饿汉单例设计模式的步骤:

1．私有化构造函数。

2．声明本类的引用类型变量，并且创建本类的对象。

3．提供一个公共的方法获取本类的对象。

#### 实例：

class Single{

//创建本类的对象。

private static Single s = new Single();

//私有化构造函数

private Single(){}

//提供一个公共的方法获取本类的对象。

public static Single getInstance(){

return s;

}

}

### 懒汉单例设计模式

懒汉单例设计模式：

1．私有化构造函数。

2．声明变量的引用类型变量，但是先不要创建 本类的对象。

3．提供一个公共静态的方法获取本类的对象，获取之前先判断是否已经创建了本类的对象，如果没有创建，那么先创建本类的对象，然后再返回。否则直接返回即可。

#### 实例：

class Single2{

//声明变量的引用类型变量，但是先不要创建 本类的对象。

private static Single2 s;

//私有化构造函数

private Single2(){}

//提供一个公共静态的方法获取本类的对象，获取之前先判断是否已经创建了本类的对象，如果没有创建，那么先创建本类的对象，然后再返回。否则直接返回即可。

public static Single2 getInstance(){

if(s==null){

s =new Single2();

}

return s;

}

}

### 懒汉单例线程安全解决

class Single{

//声明本类的引用类型变量，但是不创建对象。

private static Single s = null;

//私有化构造函数

private Single(){}

public static Single getInstance(){

if(s==null){

synchronized ("锁") {

if(s==null){

s = new Single();

}

}

}

return s;

}

}

## 观察者设计模式

**当一个事物发生了指定的动作的时候，要 通知另外一个事物做出相应的处理。**

需求：编写一个气象站、气象站要不断的更新天气， 人要根据要根据当前的天气作用相应的处理。

问题1： 天气更新了多次，然后人才做出一次的反应。

问题2：目前气象站只能为一个人服务 ？

问题3： 在现实生活中，出了员工要关注天气以外，其他的群体也要关注天气的？

**观察者设计模式的实现步骤：**

**把要通知对方的行为抽取出来定义在一个接口上，然后在本类中维护该接口的成员。**

### 实例：

#### // 气象站

public class WeatherStation {

String[] weathers = {"暴雨","台风","霜冻","晴天"};

String weather;//当前的天气

Random random = new Random();

**// 程序的解耦。 程序设计讲究低耦合, 类与类之间不要过分依赖。**

**ArrayList<Weather> list = new ArrayList<Weather>();**

public void addListener(Weather e){ //

list.add(e);

}

//工作

public void startWork(){

new Thread(){

@Override

public void run() {

while(true){

updateWeather();

//每隔1~1.5秒更新一次天气

for(Weather e:list){

e.notifyWeather(weather);

}

**int millis = random.nextInt(501)+1000;**

try {

Thread.sleep(millis);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}.start();

}

//更新天气的方法

public void updateWeather(){

int index = random.nextInt(weathers.length);

weather = weathers[index];

System.out.println("当前的天气："+ weather);

}

}

#### // 设置接口

**// 如果要订阅天气的群体必须也要实现Weather接口。**

public interface Weather {

public void notifyWeather(String weather);

}

#### // 使用对象1

public class Emp implements Weather{ //**实现了Weather接口**

String name;

public Emp(String name) {

super();

this.name = name;

}

//人要根据天气做出相应的处理方案

public void notifyWeather(String weather){

// "暴雨" ,"台风","霜冻","晴天"

if("暴雨".equals(weather)){

System.out.println(name+"带着雨伞上班！！");

}else if("台风".equals(weather)){

System.out.println(name+"拖着一块大石头上班！！");

}else if("霜冻".equals(weather)){

System.out.println(name+"带着棉被上班！！");

}else if("晴天".equals(weather)){

System.out.println(name+"开开心心上班！！");

}

}

}

#### // 使用对象2

public class Student implements Weather { //**实现了Weather接口**

String name;

public Student(String name) {

this.name = name;

}

public void notifyWeather(String weather){

if("暴雨".equals(weather)){

System.out.println(name+"在宿舍休息！！");

}else if("台风".equals(weather)){

System.out.println(name+"在宿舍休息！！");

}else if("霜冻".equals(weather)){

System.out.println(name+"盖棉被休息！！");

}else if("晴天".equals(weather)){

System.out.println(name+"去教室休息！！");

}

}

}

#### // 实现

public class WeatherMain {

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

**// 群体1**

Emp e = new Emp("狗娃");

Emp e2 = new Emp("如花");

**// 群体2**

Student s = new Student("小明");

Student s2 = new Student("小黑");

**// 气象站**

WeatherStation station = new WeatherStation();

**// 由于两个群体都实现了Weather接口，所以WeatherStation可以添加两个群体的对象**

station.addListener(e);

station.addListener(e2);

station.addListener(s);

station.addListener(s2);

**// 启动气象站**

station.startWork();

}

}

## 工厂设计模式

工厂设计模式就是专门用于产生对象的方法。

需求： 定义一个工厂方法可以产生任何类型的对象，产生的对象是根据配置文件而定。

### 实例：

public class Demo {

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

Person p = (Person) newInstance();

System.out.println("人对象："+ p);

}

public static Object newInstance() throws Exception{

BufferedReader bufferedReader = new BufferedReader(new FileReader("obj.txt"));

**//从文件中读取到完整的类名**

String className = bufferedReader.readLine();

**//运用反射技术根据类名获取Class对象**

Class clazz = Class.forName(className);

**//获取无参的构造函数**

Constructor constructor = clazz.getConstructor(null);

**//创建对象。**

Object o = constructor.newInstance(null);

**//读取配置文件，然后对象的属性数据封装到对象中。**

String line = null;

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

String[] datas = line.split("=");

Field field = clazz.getDeclaredField(datas[0]);

field.setAccessible(true);

if(field.getType()==int.class){

field.set(o, Integer.parseInt(datas[1]));

}else{

field.set(o, datas[1]);

}

}

return o;

}

}