第一章：初识JAVA

1：java环境搭建

下载JDK。JDK是用于开发java应用程序的开发包，它提供了编译、运行java程序所需的各种工具和资源。

配置环境变量。我的电脑—属性—高级系统设置—环境变量—（win10）找到path点击编辑再新建一个位置，将安装的JDK下的bin目录粘贴此处—在环境变量处新建java\_home，将JDK安装目录粘贴。

检查环境变量是否配置成功。Win+r键 打开命令行，输入cmd——再输入java -version 查看是否提示JDK版本号。也可以再输入java命令验证。

2：java程序执行过程

.java文件→编译器→生成.class字节码文件→JVM

第二章：数据类型和运算符

1：变量的命名

在java中，标识符用来为程序中的常量，变量，方法，类，接口和包命名。

标识符命名规则：

①：标识符由字母，数字，下划线(\_)，美元符($)组成

②：标识符的首字母有字母，下划线，美元符组成，不能以数字开头

③：标识符的命名不能与关键字，布尔值（false、true）和null相同

④：标识符区分大小写，没有长度限制，坚持见名知意原则

命名采用驼峰法（多单词组成时第一个单词首字母不大写第二个开始首字母大写）

2：java中的注释

①：单行注释 // 只能书写在一行的注释，当只有一行内容需要注释时，一般使用单行注释

②：/\*......\*/ 多行注释 ， 当有多行内容需要注释时，一般使用多行注释

③：/\*\*.......\*/ 文档注释 类似多行注释，但会生成一份HTML格式的API帮助文档。

3：数据类型

基本数据类型分为：

整数型：（byte,short,int,long)

字符型：（char）

基本数据类型：布尔型（boolean）

4:java中的基本数据类型取值范围

|  |  |  |  |
| --- | --- | --- | --- |
| 基本类型 | 大小 | 实例 | 取值范围 |
| Boolean | 1字节8位 | true | true,false |
| byte | 1字节8位有符号整数 | -12 | -128~+127 |
| short | 2字节16位有符号整数 | 100 | -32768~+32767 |
| int | 4字节32位有符号整数 | 12 | -2^31~+2^31-1 |
| long | 8字节64位有符号整数 | 10000 | -2^63~+2^63-1 |
| char | 2字节16位Unicode字符 | ‘a’ | 0~65535 |
| float | 4字节32位浮点数 | 3.4F | -3.4E38~3.4E38 |
| double | 8字节64位浮点数 | -2.4e3D | -1.7E308~1.7E308 |

5：常量

常量指的是在程序运行中值不能改变的量

比如布尔常量 只有 true和false

字符常量，比如：

\n 转义字符常量：表示换行

\b 转义字符常量：表示按backspace键

\t 转义字符常量：表示按TAB键

\\ 特殊字符常量：表示反斜杠

\’ 特殊字符常量：表示单引号

\” 特殊字符常量：表示双引号

6：变量

与常量对应的就是变量，变量是在程序运行时值可以改变的量

“变量类型”可以从数据类型中选择

“变量名”是定义的名称变量，要遵循标识符命名原则

中括号 中的内容为初始值，是可选项

7：数据类型转换

Byte→short→int→long→float→double

↑

char→↑

整数数据类型在计算时默认是int类型

存储位数越高。，类型的级别越高

强制类型转换：高级别到低级别，需要增加强制转换符

byte bt=10;

byte bb=(byte)(bt+bt);

8：常用运算符

①：赋值运算符

= int a=5；把5赋值给a a=5

long类型的数值，应该在后面加L/l；

**long** time=9999999999L;

Float类型的数值，应该在后面加F/f

**float** score=60.5F;

**char** sex='M';//单引号中写一个字符

## ②：算数运算符

+ - \* / %(取模) ++(自增1) --(自减1)

对于除法运算符"/",如果两个操作数为整数，结果会舍弃小数部分而保留整数部分。

如果操作数中有一个是浮点数，将会自动类型转换，结果会保留小数部分。

③：复合算术运算符

+= -= \*= /= %= 例如 a+=1 a-=1 //a=a+1 a=a-1

④：逻辑运算符

逻辑运算的结果为boolean类型

\* &&：两个操作数都为true，结果为true。其他情况都为false

\* ||: 两个操作数都为false，结果false。其他情况都为true

\* ! :操作数位true，结果为false。 操作数为false，结果为true

⑤：条件运算符

又称三目运算符/三元运算符

格式: 条件?表达式1:表达式2

首先判断条件，如果为true，则返回表达式1的值。如果为false，则返回表达式2的值。

例：int a;

a= 5<7?5:7

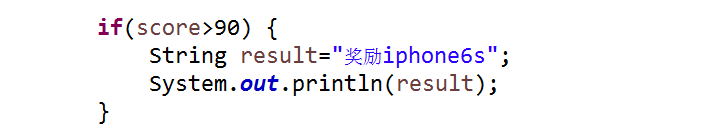
首先判断5<7是否成立（true或false）成立则取表达式1的值 此处是 5 不成立取表达式2的值 此处是7

第三章：选择结构和 循环结构

流程控制结构：1：顺序结构。指的是程序从上到下依次执行每条语句的结构

2：选择结构 根据条件的判断来选择执行不同的代码，选择结构可以分为单分支结构，双分支结构 和多分支结构

单分支：



双分支：

if的双分支结构：

\* 语法:

\* if(条件){

\* 代码块1

\* }else{

\* 代码2

\* }

\*

\* 当条件成立，执行代码块1；如果条件不成立，则执行代码块2

\* 注意:else子句不能单独使用，它是if语句的一部分。

多分支：

\* 语法:

\* if(条件1){

\* 代码块1

\* }else if(条件2){

\* 代码块2

\* }else if(条件3){

\* 代码块3;

\* }else if(....){

\* .....

\* }else{

\* 代码块n

\* }

\* 首先对if语句的条件1判断，为true，则执行代码块1。之后分支结构结束。

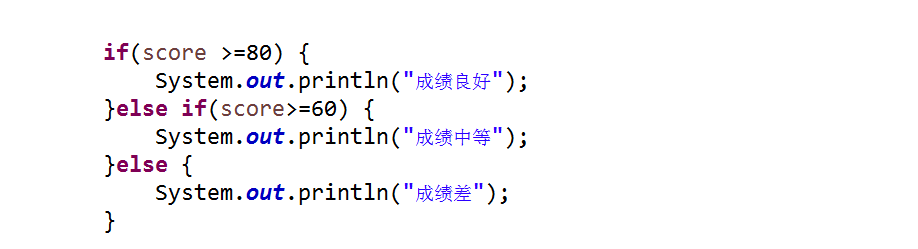
\* 若为false，则判断条件2，为true则执行代码块2。之后分支结束。

\* 依次如此...

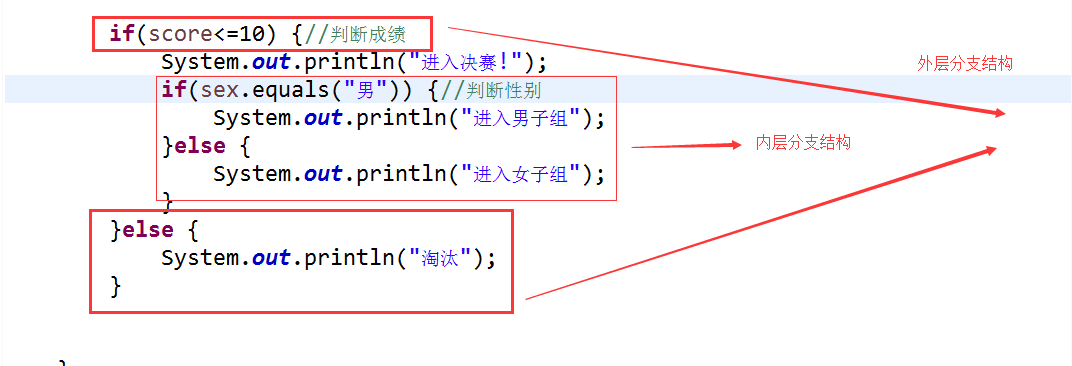
\* 最后，如果前面的条件都不成立，则执行else的代码块

\*

\* 注意:多分支if结构中，各个条件的顺序是不能任意排列的。



### 分支嵌套：



### Switch

Switch(表达式){

case 常量 1:

语句;

break;(注：非必输项)

case 常量 2:

语句;

break;

…

default:(注：非必选项)

语句;

}

常见错误

1. 如果每个case执行完后都需要跳出来，则不要忘记写break。
2. case后面的常量值不能重复。
3. default的位置顺序可以变动，但要符合逻辑顺序

Switch结构只能处理等值判断条件。针对于判断条件处在一个值得区间，则使用if结构或者三目运算符

3：循环结构。 循环结构

## While

while(循环条件，结果是boolean){

循环体

}

先判断循环条件，如果为ture，则执行循环体。

如果为false，结束循环

## do\_while

while(循环条件，结果是boolean){

循环体

}

先判断循环条件，如果为ture，则执行循环体。

如果为false，结束循环

While与do\_while的区别在于，do\_while不管循环条件是否成立，至少有一次执行循环体。

## For

for(表达式1;表达式2;表达式3){

循环体

}

首先执行表达1，一般进行变量的初始化。

然后执行表达式2，判断循环条件。

如果为true，则走循环体。

循环体执行完毕后，执行表达式3，改变循环变量的值。

再次执行表达式2，判断循环条件.....后续依次如此。

三种循环结构的使用：

①：循环次数确定的情况下，通常使用for循环

②：循环次数不确定的请款古关系啊，通常使用while 和do while

**break**

break的使用

break语句只能出现在switch和循环语句中。 表示结束switch，或循环

**continue**

continue的作用：跳过本次循环，而进行下一次循环 。

continue只能出现在循环语句中。

return

结束当前的方法

JVM内存分区

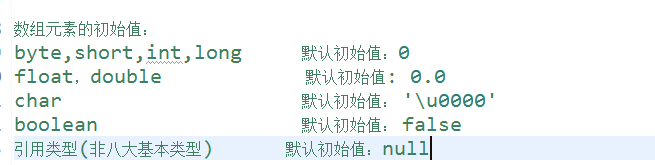
Jvm内存分为：栈，堆，方法区

# 数组

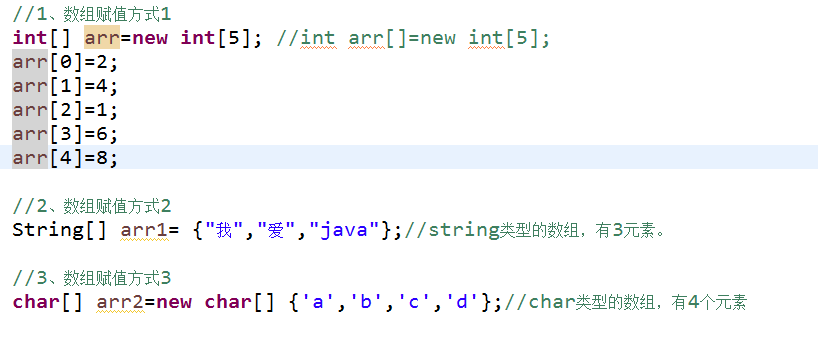
数组：数组是存储相同数据的一组数据

当数组初始化完毕后，java为数组在内存中，分配一段连续的空间，其在内存开辟的空间也将随之固定下来。此时数组长度就不在发生变化。即使数组没有保存任何数据，数组所占的空间依然存在(默认初始值)。

数组的默认初始值：



3种数组赋值方式：



注意：数组下标是从0开始，而不是1，如果访问的数组下标小于0或者大与数组长度，都将会出现数组下标越界的异常

2种数组遍历（循环）

普通for循环遍历

for(int i=0;i<score.length;i++) {

System.out.println(“score[i]”);

}

增强型for

for（int num : arr）{ //依次取出arr数组中的各个元素的值 并赋值给num

System.out.println(num);

}

变量 num的类型必须与数组arr元素的类型一致

计算数组长度：

数组名.length

例如 int []arr={1,2,3,4}; 数组长度为：arr.lenguth

# 二维数组

二维数组的声明和初始化：

二维数组实际是一个一维数组，他的每一个元素又是一个一维数组

Java中定义和操作多维数组的语法与一维数组类似

例如: 1、int[][] arr=new int[3][5]; 声明一个二维数组，分配了3元素。每个元素又是一个5个长度的一维数组

2、声明二维数组，有3个元素，但没有定义每个元素的数组长度

int[][] arr2=new int[3][];

arr[0]= new int[5];

arr[1]= new int[7];

arr[2]=new int [3];

1. int[][] arr3= {{2},{1,2},{3,2,1}};声明二维数组，有三个数组类型元素，元素的数组依次长度为1，2，3
2. int[][] arr4=new int[][] {{2},{1,2},{3,2,1}};声明二维数组，有三个数组类型元素，元素的数组依次长度为1，2，3

二维数组的遍历：

使用for循环遍历二维数组

例如：

定义了一个二维数组：int [][]arr ={{1},{1，2},{1，2，3}};有三个数组元素，每个元素的数组长度为1，2,3

For(int i=0 ;i<arr.length;i++){

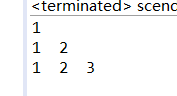
For(int j=0;j<arr[i].length;j++){

System.***out***.print(arr[i][j]+" ");

}

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

}



# Arrays类的使用

**①：**比较两个数组是否相等

Arrays类的equals()方法用于比较两个数组是否相等。只有当两个数组的长度相等，对应位置的元素也一一相等时，该方法返回true，否则返回false

例如：

Int[][]arr1 ={{1}，{1}，{1}}； Int[][]arr2 ={{1}，{1}，{1}}；

System.out.println(Arrays.equals(arr1,arr2));

此处完全相等 所以返回值为true

**②：**对数组的元素进行升序排列

Arrays类中的 sort()方法对数组的元素进行升序排列，即从小到大的顺序排列

例如：int []arr={4,3,2,1};

Arrays.sort(arr);

遍历后为：1,2,3,4

**③：**将数组转换成字符串

Arrays类中提供了专门输出数组内容的方法：toString()方法

使用方法：

Int []arr={1,2,3,4};

String num=Arrays.toString(arr);

System.out.println(num); 输出结果为字符串格式的【1,2,3,4】

④：将所有数组元素赋值为相同的值

int[] arr={10,50,40,30};

例如 将数组arr中的数值全部改成1

Arrays.fill(arr, 1); 遍历输出arr数组修改后的结果 为 1 1 1 1

⑤：将数组复制成一个长度为设定值的新数组

使用Arrays类中的copyOf()方法把数组复制成一个长度为设定值的新数组

Int [] arr1={10,20,30,40};

将数组复制成一个长度为3的新数组arr2

Int [] arr2=Arrays.copyOf(arr1,3); 输出结果为：10， 20, 30

System.arraycopy

System.arraycopy (可以选择从源数组的某个下标开始复制，比较灵活)

例如：int[] arr= {1,2,3,4,5,6,7};

int[] arr1=new int[7];

System.arraycopy(arr,1, arr1, 0, 6);//将arr中2,3,4,5,6,7复制到arr1数组中

System.out.println(Arrays.toString(arr1));输出结果为[2, 3, 4, 5, 6, 7, 0]

⑥：查询元素在数组中的下标

Arrays类中的binarySearch（）方法用于查询数组元素在数组中的下标，调用该方法需要先将数组中的元素进行升序排序排列，这样才能得到正确的结果

例如：

Int [] arr={2,652,13,14,6};

Arrays.sort(arr);升序排序

Int num=Arrays.binarySearch(arr,652);查找数组元素652在数组中的下标为多少， 输出结果为4