Java 基础语法

天津卓讯科技有限公司

Java 基础语法

- 2.1 Java程序基本结构
- 2.2 变量
- 2.3 数据类型 float double char boolean byte int short long
- 2.4 运算符
- 2.5 语句
- 2.6 方法
- 2.7 数组
- 2.8 常用算法

2.1 Java程序基本结构

- 代码框架及注释
- 标识符
- 关键字
- 常量

代码框架

- 代码基本框架

```
public class 类名{
    public static void main(String[] args){//入口方法
        ……
    }
}
```

注释

- 用于说明解释程序的文字。提高了代码的可阅读性。
- Java中的注释格式：
  - 单行注释： //注释文字
  - 多行注释： /* 注释文字 */
  - 文档注释：/** 注释文字 */
- 对于单行和多行注释，被编译器编译成字节码时就被忽略了。
- 文档注释是java特有的注释，其中注释内容可以被JDK提供的工具javadoc.exe所解析，生成一套以网页文件形式体现的该程序的说明文档。

标识符

- Java对包、类、方法、参数和变量等要素命名时使用的字符序列称为标识符。
- Java标识符命名规则：
  - 由英文字母、数字、下划线（_）和美元符号（$）组成。
  - 不能以数字开头。
  - 不能是Java中的保留关键字。
  - 区分大小写。
  - 长度无限制。
- 标识符命名习惯：见名知意。示例：
  - HelloWorld、username2、user_name、_userName、$abc_123
  - 2UserName、user#Name、Hello World

Java中的名称规范

- 包名：多单词组成时所有字母都小写。
  - aaa.bbb.ccc　com.computer.java　com.computer.html
- 类名、接口名：多单词组成时，所有单词的首字母大写。帕斯卡命名法（大驼峰）
  - AaaBbbCcc　AirTest　TvUnit　ComputerJavaHtml
- 变量名、方法名：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写。驼峰命名法（小驼峰）
  - aaaBbbCcc　addButoon　subButton
- 常量名：所有字母都大写。多单词时每个单词用下划线连接。
  - AAA_BBB_CCC

关键字（keyword）

- Java中有特定含义，专门用途的字符串称为关键字。全部为小写

用于定义数据类型的关键字				
class	interface	byte	short	int
long	float	double	char	boolean
void				
用于定义数据类型值的关键字				
true	false	null		
用于定义流程控制的关键字				
if	else	switch	case	default
while	do	for	break	continue
return				
用于定义访问权限修饰符的关键字				
private	protected	public		

关键字（keyword）

用于定义类，方法，变量修饰符的关键字				
abstract	final	static	synchronized	
用于定义类与类之间关系的关键字				
extends	implements			
用于定义建立实例及引用实例，判断实例的关键字				
new	this	super	instanceof	
用于异常处理的关键字				
try	catch	finally	throw	throws
用于包的关键字				
package	import			
其他修饰符关键字				
native	strictfp	transient	volatile	assert
保留字：没有定义用途，但保留备用。				
goto	const			

常量

- 常量：程序中持续不变的值，它是值不能改变的数据。(也叫字面值)
  - 整型常量。所有整数：123　、10、　30、　789
  - 浮点数常量。所有实数：3.1415926　1.32
  - 字符常量。由单引号('')标识的单个字符：'a'、'\t'、'\u0027'　'o'
  - 布尔(逻辑)常量：true、false
  - 字符串常量。由双引号("")标识的一个或多个字符："a"、"hello world"　"button"
  - null常量：表示对象的引用为空

转义字符

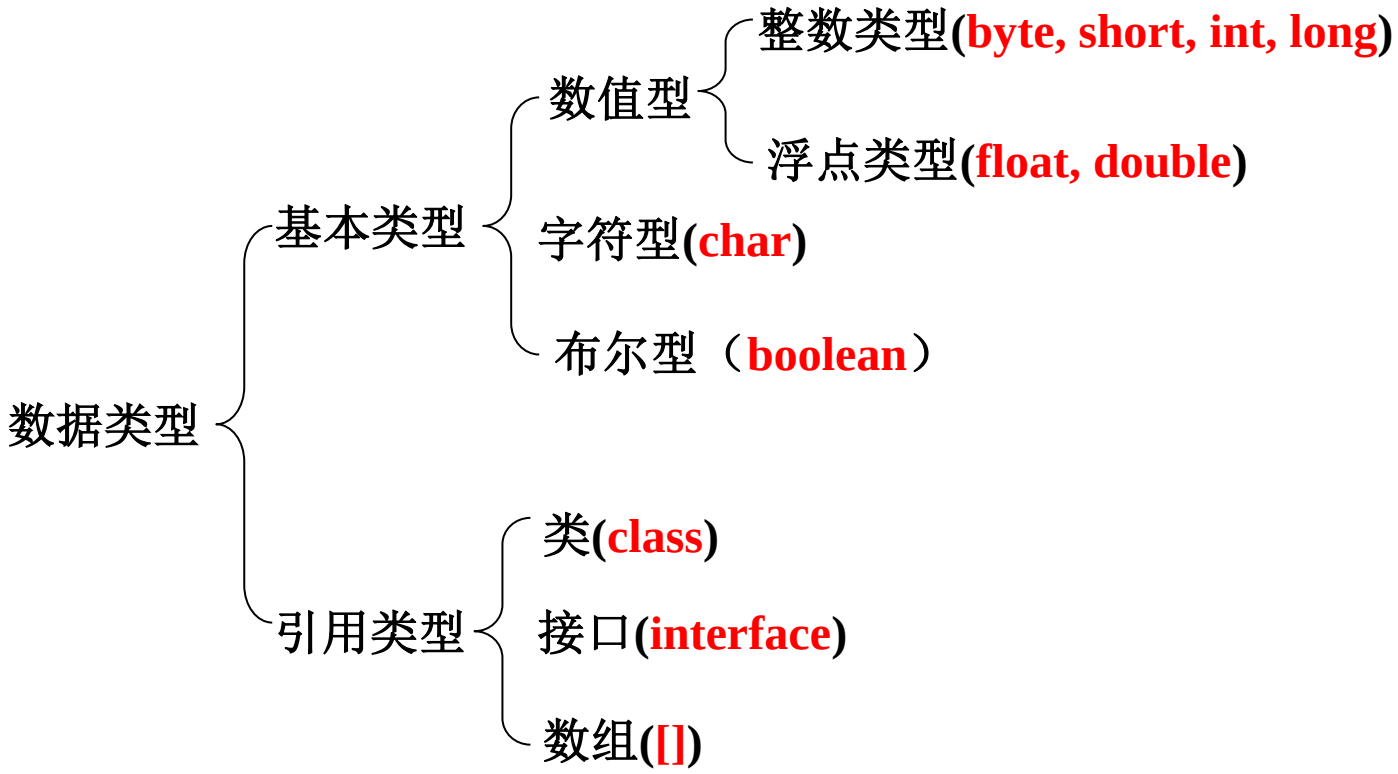
转义字符	名称	描述
\n	换行	将光标移到下一行的第一格。
\r	回车	将光标移到当前行的第一格。
\t	水平制表	将光标移到下一个水平制表位置。
\'	单引号	产生一个单引号。
\"	双引号	产生一个双引号。
\\	斜杠	产生一个斜杠。

2.2 变量

- 变量： (数学的未知数)
  - 内存中的一个存储区域B
  - 该区域有自己的名称（变量名）addButton subutton和类型（数据类型） int double
  - 该区域的数据可以在同一类型范围内不断变化 int addbutton
  - 可以重复使用
- 使用变量注意：
  - 变量的作用范围（一对{}之间有效）
  - 先申明，再初始化值后才能使用。
- 定义变量的格式：
  - 数据类型 变量名 = 初始化值

2.3 数据类型

- Java语言是强类型语言。对于每一种数据都定义了明确的具体数据类型，在内存中分配了不同大小的内存空间



基本数据类型

- Java中定义了四类/八种基本数据类型
  - 布尔型---- boolean
  - 字符型---- char
  - 整数型---- byte, short, int, long
  - 浮点数型---- float, double
- Java中所有的基本数据类型都有固定的存储范围和所占内存空间的大小，而不受具体操作系统的影响，以保证Java程序的可移植性

整数类型

类　型	占用存储空间	存储范围
byte	1字节	-128 ~ 127
short	2字节	-2^{15} ~ $2^{15}-1$
int	4字节	-2^{31} ~ $2^{31}-1$
long	8字节	-2^{63} ~ $2^{63}-1$

- Java语言的整型常量默认为int型，如：int i = 3;
- 声明long型的数据时可以加‘l’或‘L’，如：long l = 3L;

浮点数类型

类　型	占用存储空间	存储范围
float	4字节	-3.403E38~3.403E38
double	8字节	-1.798E308~1.798E308

- Java浮点类型常量有两种表示形式
  - 十进制数形式，必须含有小数点。如：3.14
  - 科学记数法形式。如：3.14e2　　3.14E2
- Java浮点型常量默认为double型。声明float型的数据后面需要加上f或F。如：
  - double d = 3.14;
  - float f = 3.14f;

字符类型

- char型数据用来表示通常意义上“字符”
  - char c = 'A';
- Java字符采用Unicode编码，每个字符占两个字节，因而可用Unicode的十六进制编码形式表示
  - char c1 = '\u0061';

布尔类型

- boolean类型适于逻辑运算，一般用于程序流程控制 。
- boolean类型数据只允许取值true或false。
  - 不可以用0或非0的整数替代true和false。
- 示例：
  - boolean b = false;

基本数据类型之间的转换

- boolean 类型不能转换成任何其它数据类型。
- 自动类型转换：容量小的类型自动转换成容量大的数据类型
  - byte,short,int→long→float→double
  - byte,short,int不会互相转换，它们三者在计算时会转换成int类型
- 强制类型转换：容量大的类型转换成容量小的数据类型时，要加上强制转换符
  - long l = 100L;
  - int i = (int)l;
  - 有可能造成精度降低或数据溢出，使用时要小心。

2.4 运算符

- 算术运算符：+、-、*、/、%、++、--
- 赋值运算符：= 、+= 、-= 、*= 、/= 、%=
- 关系运算符：>、<、>=、<=、==、!=
- 逻辑运算符：! 、&、|、^、&&、||
- 位运算符：&、|、^、~、>>、<<、>>>
- 字符串连接运算符：+

算术运算符

运算符	描述	示例	结果
+	加	5+5	10
-	减	5-4	1
*	乘	5* 3	15
/	除	10/3	3
%	取模(求余)	10%3	1
++	自增(前,后)		
--	自减(前,后)		

- 问题
  - int x=1234; x= x / 1000 * 1000; x=?
  - 10 % 4 = ?; 4 % 4 = ?; 2 % 4 = ?; -1 % 4 = ?
  - int i = 3; int j = 0; j = i++; i = ?, j=?
  - int i = 3; int j = 0; j = ++i; i = ?, j=?

赋值运算符

- 赋值运算符作用是将一个值赋给一个变量
  - 由两个运算符组成而成的运算符，也叫复合运算符。

运算符	描述	示例	结果
=	赋值	a=3; b=2;	a=3 b=2
+=	加等于	a=3;b=2;a+=b;	a=5 b=2
-=	减等于		
*=	乘等于		
/=	除等于		
%=	模等于		

关系运算符

- 关系运算符作用是比较两边的操作数，结果总是boolean型的。也叫比较运算符。

运算符	描述	示例	结果
==	相等于	4==3	false
!=	不等于	4!=3	True
<	小于		
>	大于		
<=	小于等于		
>=	大于等于		

逻辑运算符

- 逻辑运算符用于对boolean型结果的表达式进行运算，运算结果总是boolean型。

运算符	描述	示例	结果
&	与	true & true false & true	true false
|	或	false | true false | false	true false
^	异或	true ^ false true ^ true false ^ false	true false false
!	非(取反)	!true !false	false true
&&	短路与	false && true	false
||	短路或	false || true	true

&和&&的区别

- 单&时，左边无论真假，右边都进行运算；
- 双&时，如果左边为假，那么右边不参与运算。→ 短路

位运算符

- 对两个操作数中的每一个二进制位都进行运算

运算符	描述	示例	结果
~	按位取反(反码)	~1	-2
&	按位与	1 & 1 1 & 0 0 & 0	1 0 0
|	按位或	1 & 1 1 & 0 0 & 0	1 1 0
^	按位异或	1 ^ 1 1 ^ 0 0 ^ 0	0 1 0

- 问题： 6 ^ 3 ^ 3 = ?

移位运算符

运算符	描述	示例	结果
<<	左移	3 << 1	6
>>	带符号右移	3 >> 1 -3 >> 1	1 -2
>>>	无符号右移	3 >>> 1 -3 >>> 1	1 2147483646

- 左移：”a<<b; "将二进制形式的a逐位左移b位，最低位空出的b位补0
- 带符号右移：”a>>b; "将二进制形式的a逐位右移b位，最高位空出的b位补原来的符号位
- 无符号右移：”a>>>b;"将二进制形式的a逐位右移b位，最高位空出的b位补0

思考

- 1.用最有效率的方法算出2乘以8等於几？
- 2.不能使用第三方变量，如何完成对两个整数变量的值进行互换？

字符串连接运算符

- String s="He" + "llo"；结果"Hello"
- "+"除了可用于字符串相连接，也能将字符串与其它的数据类型相连成一个新的字符串。
  - 如：String s="x" + 123; 结果"x123"

三目运算符

- 格式 ：(条件表达式) ? 表达式1 : 表达式2；
  - 如果条件为true，运算后的结果是表达式1；
  - 如果条件为false，运算后的结果是表达式2；
- 示例：
  - 获取两个整数中的大数。

表达式

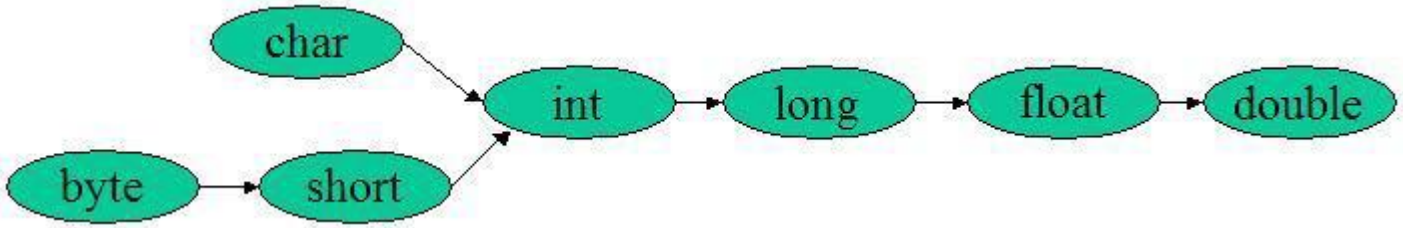
- 表达式是符合一定语法规则的运算符和操作数的序列
  - a
  - 5.0 + a
  - (a - b) * c - 4
  - i < 30 && i % 10 != 0
- 表达式的类型和值
  - 对表达式中操作数进行运算得到的结果称为表达式的值
  - 表达式的值的数据类型即为表达式的类型
- 表达式的运算顺序
  - 首先应按照运算符的优先级从高到低的顺序进行
  - 优先级相同的运算符按照事先约定的结合方向进行

运算符优先级

Separator	. ( ) { } ; ,
R to L	++ -- ~ ! (data type)
L to R	* / %
L to R	+ -
L to R	<< >> >>>
L to R	< > <= >= instanceof
L to R	== !=
L to R	&
L to R	^
L to R	|
L to R	&&
L to R	||
R to L	? :
R to L	= *= /= %= += -= <<= >>= >>>= &= ^= |=

表达式中的自动类型提升

- java算术表达式中包含多个基本类型的值时，整个算术表达式的数据类型将发生自动提升，规则：
  - 1. 所有byte型、short型和char型将被提升到int型。
  - 2. 整个算术表达式的数据类型自动提升到与表达式中最高等级操作数同样的类型。等级排列如图



- 复合赋值运算符会自动将所执行计算的结果转型为其左侧变量的类型。

思考

- short s = 1; s = s+1; 有错吗？
- short s = 1; s += 1; 有错吗？

- short s = 1; s += 123.456; s =?

程序流程控制

- 结构化程序有三种结构：
  - 顺序结构
    - 程序从上到下一行一行的执行代码，没有判断和中转。
  - 选择结构
  - 循环结构

条件语句

- if语句
  - if (条件语句){…}
  - if (条件语句){…} else {…}
  - if (条件语句){…} else if (条件语句){…}
  - if (条件语句){…} else if (条件语句){…}else{…}

- 如果if后面的语句只有一条，可以省略{}

条件语句

- switch语句：
  - switch(表达式){
  - case 取值1: 语句块1；break;
  - case 取值n: 语句块n；break;
  - default: 语句块n+1；break;
  - }
- 相关规则：
  - 表达式的值只能是int, byte, char, short 类型之一。
  - case子句中的取值必须是常量，且所有case子句中的取值应是不同的；
  - default子句是可选的；
  - case子句之间与default子句没有顺序。
  - break语句用来在执行完一个case分支后使程序跳出switch语句块
  - JDK7以后支持字符串的判断

循环语句

- 功能
  - 在循环条件满足的情况下，反复执行特定代码
- 分类
  - while 循环
  - do/while 循环
  - for 循环

while循环语句

- 语法格式
  - while(条件表达式) {
  - 　循环体语句;
  - }
- 特点：先判断后循环

- 示例：用while循环计算1000以内的奇数的和

do/while循环语句

- 语法格式
  - do{
  - 　循环体语句;
  - }while(条件表达式);
- 特点：先循环后判断
  - 不管条件是否满足，循环体至少执行一次

- 示例：用do/while循环计算1000以内的奇数的和

for循环语句

- 语法格式
  - for(初始化表达式;循环的条件表达式;循环后的操作表达式){
  - 　循环体语句;
  - }
- 说明：
  - 初始化表达式只执行一次
  - 判断循环条件，为真就执行循环体。之后再执行循环后的操作表达式，接着又判断循环条件，重复整个过程，直到条件不满足为止。

循环嵌套和无限循环

- 循环嵌套：循环语句中嵌套着循环语句
  - 示例：用星号输出一个实心正方形

- 无限循环格式：
  - while(true){ ...}
  - for(;;){ ...}
- 无限循环存在的原因是并不知道要循环多少次，而是根据运行时的条件变化来控制循环。

练习

- 打印输出0~200之间能被7整除但不能被4整除的所有整数；要求每行显示6个数据；
- 计算10!(阶乘)的结果
  - 10！= 1*2*3*4*...*10
- 有一张无限大,厚度为1毫米的纸，问经过几次折叠后能够达到珠峰的高度（8848米）。
- 打印输出如下图形
  - *
  - **
  - ***
  - ****

特殊流程控制语句

- break 语句用于终止某个语句块的执行
- continue语句用于跳过某个循环语句块的一次执行

```
public class BreakTest{
    public static void main(String [] args){
        for(int i = 0; i<10; i++){
            if(i==3){      break;         }
            System.out.println(" i =" + i);
        }
        System.out.println("Game Over!");
    }
}
```

```
public class ContinueTest {
    public static void main(String [] args){
        for (int i = 0; i < 100; i++) {
            if (i%10==0){ continue; }
            System.out.println(i);
        }
    }
}
```

2.6 方法

- Java的方法类似于其它语言的函数，是一段用来完成特定功能的代码片段，声明格式：

```
[修饰符1 修饰符2 …] 返回值类型 方法名(形式参数列表){
    程序代码;
    return 返回值;
}
```

- 形式参数(形参)：方法被调用时用于接收外界输入的数据
- 实际参数(实参)：调用方法时实际传给方法的数据。
- 返回值类型：方法要返回的结果的数据类型。
  - 若一个方法没有返回值，必须给出返回值类型void
- 返回值：方法在执行完毕后返还给调用者的数据。
- return语句终止方法的运行，并指定要返回的数据。

方法的特点

- 定义方法可以将功能代码进行封装，便于对该功能进行复用
- 对于方法没有具体返回值的情况，返回值类型用关键字void表示，那么该方法中的return语句如果在最后一行可以省略不写。
- 注意：
  - 方法只有被调用才会被执行
  - 方法中只能调用方法，不能在方法内部定义方法。
  - 使用方法时，方法的结果应该返回给调用者，交由调用者处理。

方法的应用

- 两个明确
  - 明确要定义的功能最后的结果是什么？
  - 明确在定义该功能的过程中，是否需要未知数参与运算

- 示例：定义一个功能，实现两个整数的加法运算并将结果返回给调用者
  - 分析：
    - 该功能的运算结果是什么？两个数的和，也是一个整数(int)
    - 在实现该功能的过程中是否有未知内容参与运算？加数和被加数是不确定的。(两个参数int，int)

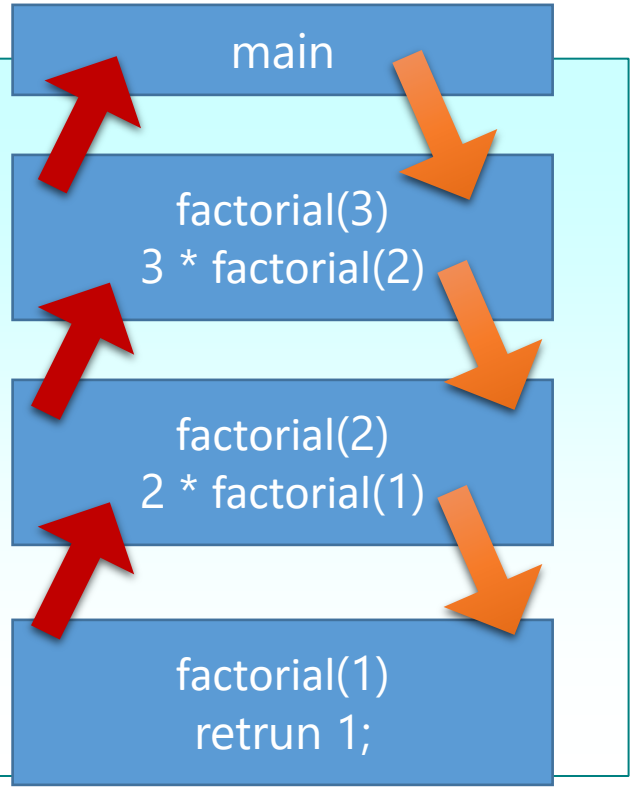
练习

- 定义一个功能，返回两个整数中的大值
- 定义一个功能，返回某个数的阶乘值
- 定义一个功能，根据输入的行数打印直角三角形
  - *
  - **
  - ***
  - ****

递归算法

- 递归：方法执行过程中出现该方法本身的调用。

```
public class RecursionDemo {
    public static int  factorial(int n){
        if( n == 1 || n == 0) {
            return 1;
        } else {
            return n * factorial(n-1);
        }
    }
    public static void main(String[] args){
        System.out.println(factorial(3));
    }
}
```



- 递归算法关键要抓住的是：递归出口
- 递推逐步向出口逼近

递归调用练习

- 求出斐波那契数列第n个数的值
  - 1、1、2、3、5、8、13、21、……

2.7 数组

- 一维数组的声明和初始化
- 增强的for循环(ForEach) - JDK5.0
- 可变参数(Varargs) - JDK5.0
- 命令行参数
- 二维数组的声明和使用
- 数组中的常用算法

数组概述

- 数组是用来存储一组相同类型数据的数据结构
- 数组变量属于引用数据类型
- 数组中的元素可以是任何数据类型(基本类型和引用类型)

一维数组的声明

- 一维数组的声明方式：
  - 数据类型 [] 数组变量名;
  - 数据类型 数组变量名[];
- 例如：
  - int a[];
  - int[] a1;
  - double b [];
- Java语言中声明数组时不能指定其长度(数组中元素的个数)，例如：
  - int a[5];　//非法

数组对象的创建

- Java中使用关键字new来创建数组对象
  - 格式：数组变量名 = new 数组元素的类型 [数组元素的个数];
- 数组一旦被创建，就不能再改变它的大小

数组元素的引用

- 数组元素的引用方式：arrayName[index]
  - index为数组元素下标(索引)，可以是整型常量或整型表达式。如a[3]，b[i]，c[6*i];
  - 数组元素下标从0开始
    - 长度为n的数组，下标取值范围：0 ~ n-1；
- 每个数组都有一个属性length指明它的长度

```
public class ArrayTest {
    public static void main(String[] args) {
        int[] a = new int[6];
        for(int i = 0; i < a.length;  i++) {
            System.out.print(a[i] + "");
        }
    }
}
```

数组的初始化

- 动态初始化
  - 数组的定义和为数组元素分配空间并赋值的操作分开进行

```
public class ArrayTest2{
    public static void main(String args[]){
        int [] a = null;     //定义一个数组
        a = new int[3];    //创建一个数组
        a[0] = 4;    a[1] = 5;    a[2] = 6;    //给数组内的元素赋值
    }
}
```

数组的初始化

- 静态初始化：
  - 在定义数组的同时就为数组元素分配空间并赋值

```
public class ArrayTest3{
    public static void main(String args[]){
        int [] a = {4,6,5};
        int [] b = new int[]{4,5,6};
    }
}
```

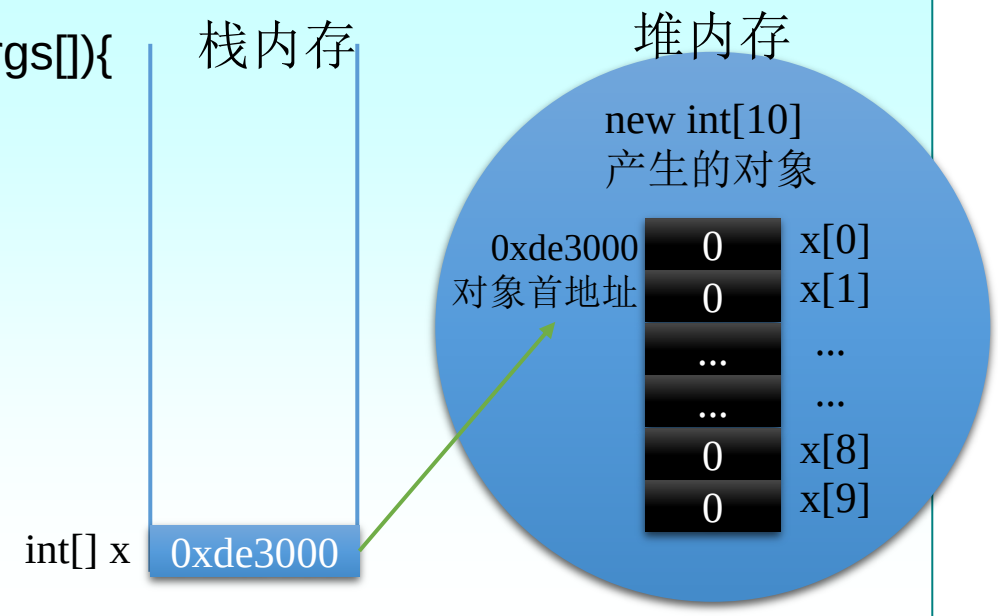
数组元素的默认初始化

- 数组一经分配空间，其中的每个元素也会被隐式初始化。

```
public class ArrayTest4{
    public static void main(String args[]){
        int [] a = new int[3];
        System.out.println(a[2]);
    }
}
```

数组内存结构

```
public class ArrayTest5{
    public static void main(String args[]){
        int [] x = null;
        x = new int[10];
        for ( int i = 0; i < 10; i++ ) {
            x[i] =2 * i + 1;
            System.out.println(x[i]);
        }
    }
}
```



增强的for循环

- 在JDK5.0中新增了一个增强的for循环语法：

```
for(type element : array){
        System.out.println(element);
}
```

```
int[] arr = {2, 4, 6, 7, 3, 5, 1, 9, 8};
for(int element : arr) {
    System.out.print(element + " ");
}
```

- 特点：
  - 使用简单、方便。
  - 遍历数组或集合时无法访问索引(下标)值
  - 只能遍历显示数组或集合中元素的内容。不能修改内容。

可变参数

- JDK5.0中，新增了可变参数：
  - 当定义方法时，传入到方法的参数的个数不固定时，可以使用可变参数来解决。

```
public class VarargsTest {
    public static void main(String[] args) {
        System.out.println(add(9.0, 10.0, 100.0, 123.0));
    }

    public static double add(double... nums){
        double result = 0.0;
        System.out.println(nums.getClass().isArray());
        for (double d : nums) {
            result += d;
        }
        return result;
    }
}
```

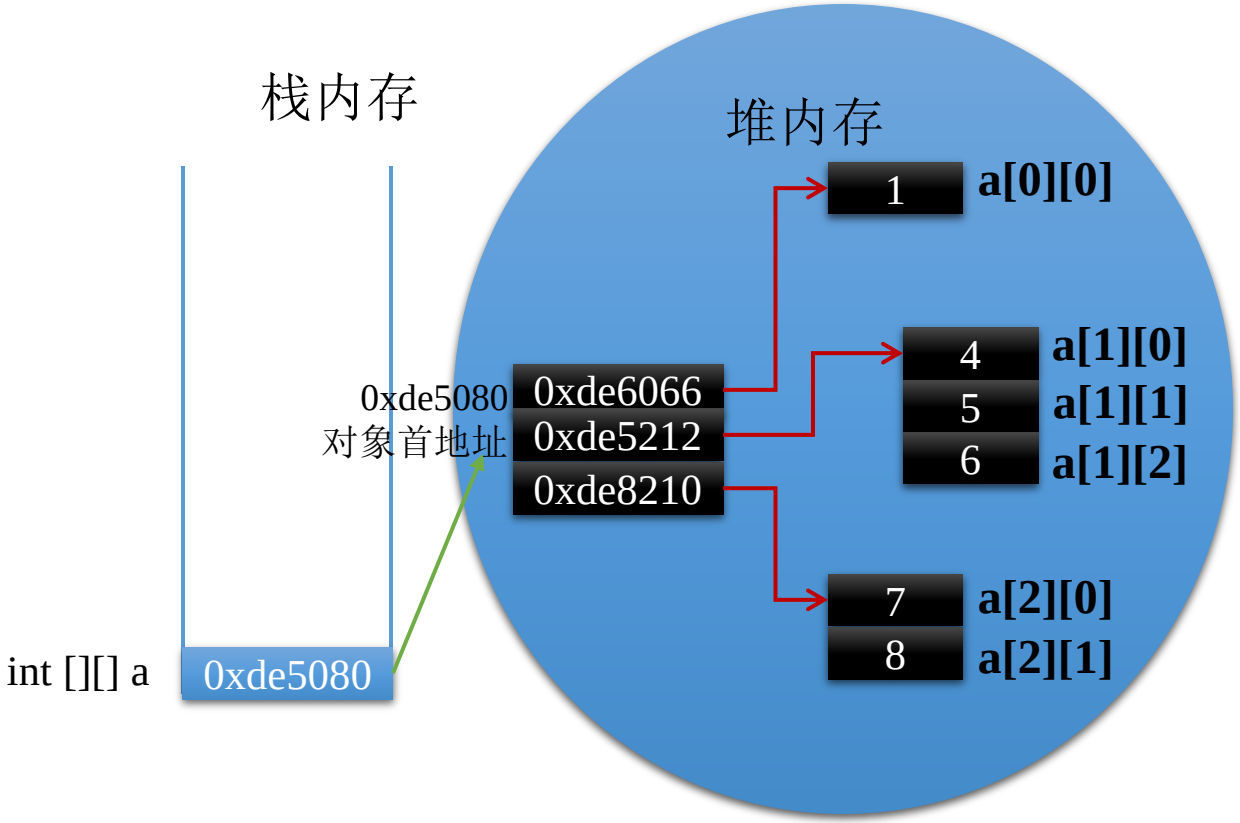
命令行参数

- Java程序的入口方法main 方法，带有String[] args参数。这个参数表示main方法接收了一个字符串数组，也就是命令行参数。

```
public class CommandLineArgsTest {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.print(args[i] + " ");
        }
    }
}
```

二维数组

- Java并没有真正的多维数组，二维数组可以看成以数组为元素的数组。如：
  - int [][] a = { {1}, {4,5,6}, {7,8}};



二维数组的初始化

- 动态初始化
  - int [][] a = new int[4][5];
  - int [][] b = new int[3][];
  - b[0] = new int[2];
  - b[1] = new int[3];
  - b[2] = new int[5];
- 静态初始化：
  - int [][] array = {{1,2},{2,3},{3,4,5}};
  - int [3][2] array1 = {{1,2},{2,3},{4,5}}; //非法的

二维数组示例

```
public class Test{
    public static void main(String args[]){
        int [][] a = { {1,2}, {2,3,4,5}, {5,6,7}};
    for(int i = 0; i < a.length; i++) {
            for(int j = 0; j < a[i].length; j++) {
                System.out.println("a["+ j + "][" + j + "]=" + a[i][j] + ", ");
            }
            System.out.println();
        }
    }
}
```

2.8 常用算法

- 常用算法
  - 最值
  - 排序
  - 搜索

常用算法

- 最大值
- 最小值

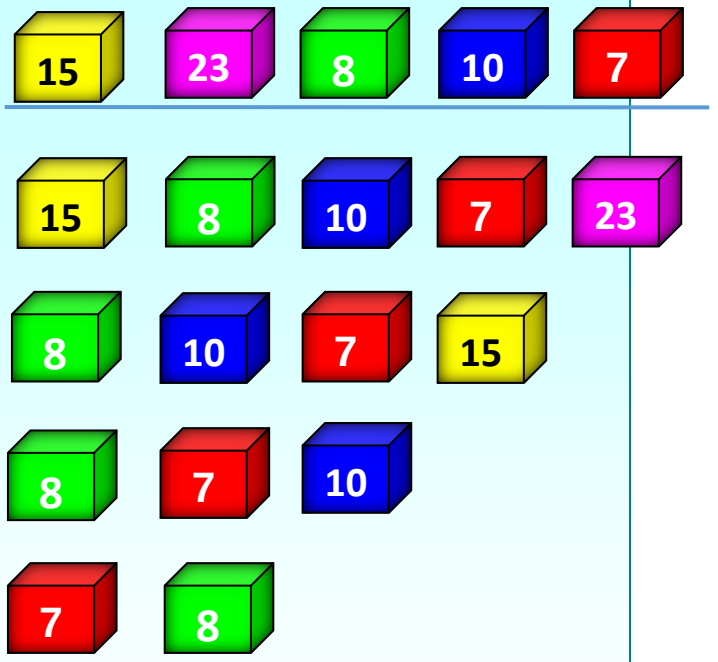
常用算法—排序

- 冒泡排序
- 选择排序

冒泡排序(Bubble Sort)

- 原理：比较两个相邻的元素，将值大的元素交换至右端

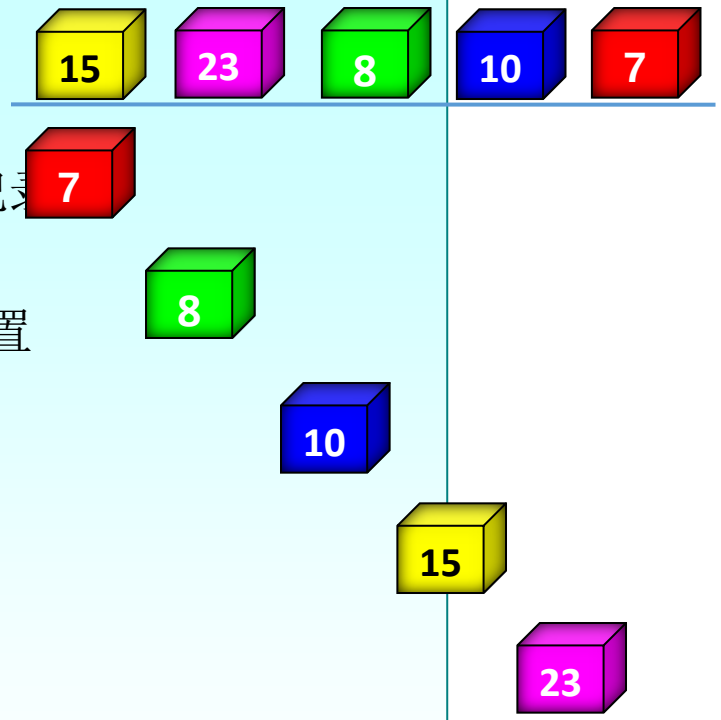
```
public static void bubbleSort(int[] a) {
    //总共进行n-1轮的比较
    for (int i = 0; i < a.length - 1; i++) {
        for (int j = 0; j < a.length -1 – i; j++) {
            if (a[j] > a[j + 1]) {//交换
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
```



选择排序(Selection Sort)

- 原理：每一趟从待排序的记录中选出最小的元素，顺序放在已排好序的序列最后，直到全部记录排序完毕

```
public static void selectionSort(int[] a) {
    for(int i = 0; i < a.length - 1; i++) {// 做第i趟排序
        int m = i;
        for(int j = i + 1; j < a.length; j++){// 选最小的记录
            if(a[j] < a[m]){
                m = j; //记下目前找到的最小值所在的位置
            }
        }
        if(i != m){  //交换a[i]和a[m]
            int temp = a[i];
            a[i] = a[m];
            a[m] = temp;
        }
    }
}
```



常用算法—查找

- 顺序查找
- 二分查找

顺序查找

- 从第一个元素开始顺序比较查找。

```
public static int search(int[] a, int num) {
    for(int i = 0; i < a.length; i++) {
        if(a[i] == num){
            return i;
        }
    }
    return -1;
}
```

二分查找(折半查找)

- 前提条件：已排序的数组中查找
- 二分查找的基本思想是:
  - 首先确定该查找区间的中间点位置： int mid = (low+upper) / 2;
  - 然后将待查找的值与中间点位置的值比较：
    - 若相等，则查找成功并返回此位置。
    - 若中间点位置值大于待查值，则新的查找区间是中间点位置的左边区域。
    - 若中间点位置值小于待查值，则新的查找区间是中间点位置的右边区域。下一次查找是针对新的查找区间进行的。

二分查找示例

```
public static int binarySearch(int[] a, int num) {
    int low = 0;                              // 起点
    int upper = a.length - 1;                 // 终点
    while (low <= upper) {
        int mid = (low + upper) / 2;          // 中间点
        if (a[mid] < num) {                   // 中间点的值小于要查找的值
            low = mid + 1;                    // 更改查找的起点为中间点位置后一位
        } else if (a[mid] > num) {            // 中间点的值大于要查找的值
            upper = mid - 1;                  // 更改查找的终点为中间点位置前一位
        } else {                              // 中间点的值等于要查找的值
            return mid;                       // 返回该位置
        }
    }
    return -1;
}
```

总结

- 标识符
- 关键字
- 局部变量 & 成员变量
  - 变量的作用域
  - 变量在内存中的存放格局
- 基本数据类型
  - 4类8种，类型之间的互相转换
- 条件、循环语句
  - if switch for while do/while
- 方法
  - 形参、实参、返回值
- 递归算法：
  - 找递归出口，设置逻辑递推逐步向出口逼近

总结2

- 一维数组的声明和初始化
- 增强的for循环(ForEach) - JDK5.0
- 可变参数(Varargs) - JDK5.0
- 命令行参数
- 二维数组的声明和使用
- 数组中的常用算法