[了解编译javac 2](#_Toc69731305)

[Java 语言的组成部分 2](#_Toc69731306)

[2.1关键字 2](#_Toc69731307)

[2.2标识符 2](#_Toc69731308)

[2.3注释 2](#_Toc69731309)

[2.4常量和变量 2](#_Toc69731310)

[2.4.1常量 2](#_Toc69731311)

[2.4.2变量 4](#_Toc69731312)

[2.5运算符 6](#_Toc69731313)

[2.5.1算术运算符 6](#_Toc69731314)

[2.5.2赋值运算符 6](#_Toc69731315)

[2.5.3比较运算符 6](#_Toc69731316)

[2.5.4逻辑运算符 6](#_Toc69731317)

[2.5.5位运算符 6](#_Toc69731318)

[2.5.6三元运算符 7](#_Toc69731319)

[2.6 流程控制 8](#_Toc69731320)

[2.6.1 if 判断结构 8](#_Toc69731321)

[2.6.2 switch选择结构 8](#_Toc69731322)

[2.6.3 循环结构——while, do…while, for 9](#_Toc69731323)

[2.6.4其他流程控制语句——break, continue 10](#_Toc69731324)

[2.7 函数（方法） 10](#_Toc69731325)

[2.7.1 函数的定义和特点 10](#_Toc69731326)

[2.7.2 函数的应用 11](#_Toc69731327)

[2.7.3 函数的重载（overload） 11](#_Toc69731328)

[2.8 数组 12](#_Toc69731329)

[2.8.1 数组的定义和特点 12](#_Toc69731330)

[2.8.2 数组的内存图解（堆&栈） 13](#_Toc69731331)

[2.8.3 数组的第二种定义方式 14](#_Toc69731332)

[2.8.4 数组的应用 15](#_Toc69731333)

[2.8.5 二维数组[][] 17](#_Toc69731334)

# 了解编译javac

计算机能理解的机器语言是二进制的字符，如：0101110，01000001.

而人编写的源代码是java源文件，如：public class HelloWorld{}.

那么要让计算机能读懂我们编写的代码，就需要一个翻译器，将java代码翻译为二进制字符。这就是 javac.exe 工具的作用了。

javac.exe实际上是一个可执行的应用程序，它本质上也是java应用程序，当调用这个应用程序时，它就会启动java编译器。

# Java 语言的组成部分

2.1关键字

在java语言中被赋予了特殊含义的单词，如public, static, void, class……

2.2标识符

在程序中定义的一些名称，比如类名，方法名，变量名等等，由a-z, A-Z, 0-9, \_, $组成，但是注意不能以数字开头，且不可以使用关键字。

2.3注释

注解说明程序的文字，可以对程序进行一些说明。有三种表现形式：

**//** —— 单行注释

**/\* 注释内容 \*/** —— 多行注释，以 /\* 开始， 以 \*/ 结束

**/\*\* 注释内容 \*/** —— 文档注释， doc插件可以根据这种注释生成代码说明书

写注释是开发过程中一个非常重要的部分，良好的注释包括：

需求说明，实现逻辑说明，参数，返回值

2.4常量和变量

2.4.1常量

常量是不能被改变的数据，包括：

* 整数常量（正数、负数、0），
* 小数常量，
* 布尔型常量（True, False），
* 字符常量（’’，单引号里面只能放一个字符，无论是字母，数字还是操作符），
* 字符串常量（””，双引号里面的内容都是字符串），
* null常量（null）

既然说到常量，有必要来了解进制。计算机能识别的语言都是二进制，如：11000100，

只有1和0两个数字，每个数字都是一个 bit，计算机以8个bit为一个单位，叫做byte

（字节）。这就是二进制的由来。

然而后来，有人觉得8位数字表示一个值有点太长了，就希望能以3位，4位来表示，这 就出现了八进制和十六进制。

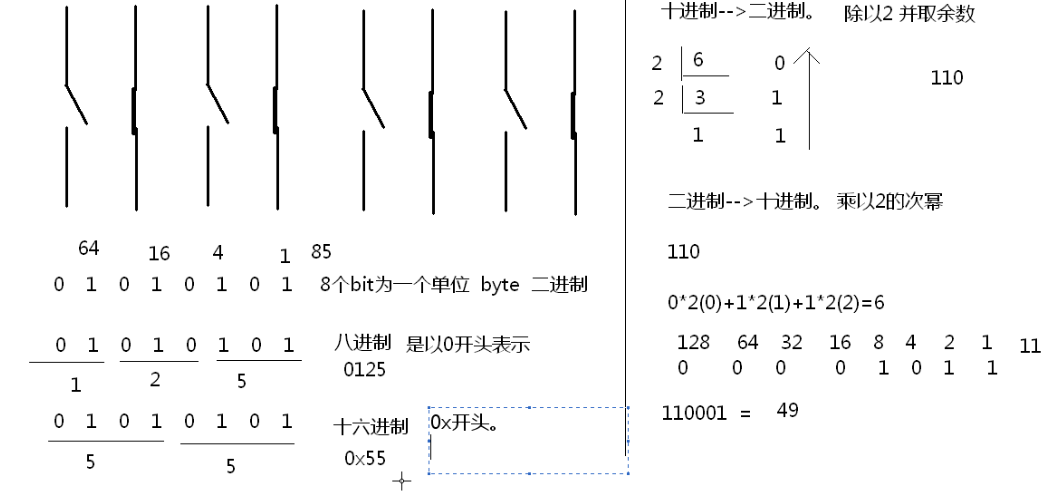
另外，需要了解一下二进制和十进制之间的转换：

十进制 to 二进制：不断的以商除以2并取余数。

二进制 to 十进制：当前位置的数字乘以2的次幂并相加。

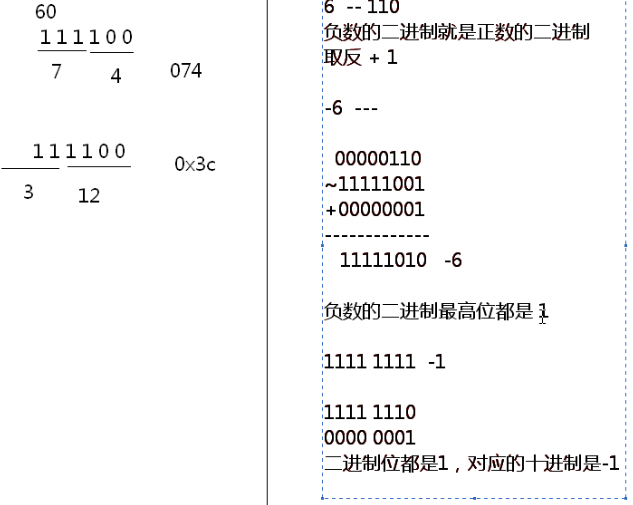
当十进制和八进制、十六进制进行转换时，先转换为二进制，然后再进行转换。

整体的表示如下图所示：



负数的二进制：8位的二进制数字，第一位表示正负，0表示正数，1表示负数。

计算如下图所示：



### 2.4.2变量

* 变量的概念：

内存中的一个存储区域；

该区域有自己的名称（变量名）和类型（数据类型）；

该区域存储的数据可以在同一类型范围内不断变化；

* 为什么需要定义变量：

用来不断的存放同一类型的值，并且可以重复使用。

* 使用变量要注意：

变量的作用范围

变量的初始化值

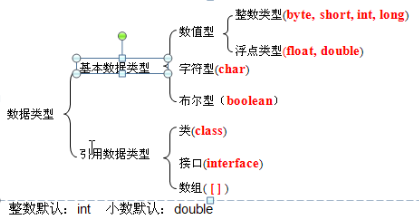
* 定义变量的格式：

数据类型 变量名 = 初始化值;

注意：格式都是固定的

有了以上的认识之后，是不是觉得变量其实就像是数学中学的未知数？

我们知道，一个变量必须要有自己的类型，不同的类型在内存中有不同大小的内存空间。



为什么整数类型居然有4种不同的单位？这是因为整数的范围非常大，所以用不同的单位来表示不同的取值范围。而小数为什么有2种不同的单位？这是因为要用小数点后的位数表示不同的精度。float表示单精度，double是双精度

|  |  |  |
| --- | --- | --- |
| 数据类型 | 在内存中的位数 | 取值范围 |
| byte | 8 bit | -128 - 127 |
| short | 16 bit | -32768 - 32767 |
| int | 32 bit | -2^32 – 2^32 - 1 |
| long | 64 bit | -2^64 – 2^64 - 1 |
| float | 32 bit，符号位1bit，阶码8bit，小数点后尾数23bit | 浮点数最多有7位十进制有效数字 |
| double | 64 bit，符号位1bit，阶码11bit，小数点后尾数52bit | 浮点数可以表示十进制的15或16位有效数字 |
| char | 16 bit | 0 – 65535 |

* 自动类型转换和强制类型转换：

变量在运算的过程中会做自动类型提升，如：byte b = 1;

在上面的代码中，我们都知道1默认会被认为是int类型，那么为什么我们可以把int型的数值赋值给byte类型？这是因为1在计算机中的内存表示是：0000 0000 0000 0001，它的数值范围在byte 的范围内，所以当执行 byte b = 1; 时，计算机内可以直接将内存转换为 0001，作为b的内存区域。这就是所谓的自动类型提升。

但是如果代码是这样：byte b = 1; b = b + 1;

编译时就会报错，因为 b=b+1;中，等号右边是变量，变量意味着值不确定，那么它有可能已经超出了 byte 类型的范围，所以编译器不会再做类型自动提升。

**总结：**当等号右侧是常量时可以做类型自动提升，是变量时就不会做。

但是其实我们知道，b=b+1;的结果是2，实际上是没有超出byte类型的范围的，那么为什么编译器还是会报错呢？这是因为编译器只会做语法检查，它并不进行运算，也就是说它不知道结果是2，所以会报错引起警惕，因为b作为一个变量，它不仅可能是1，也可能是127，那么127 + 1 = 128，就会超出byte的范围。

但是如果我们非要进行转换呢？也是可以的，通过下面的方式：

b = (byte)(b + 1);

这代码是什么意思呢？就是说将 b+1 的运算结果强制的转换为 byte 类型，由于在运行后才知道 b+1 的结果，所以编译时并不会报错。这就叫做强制类型转换。

* 字符类型的转换：

特殊的是，当char类型的数据进行运算时，整数型的数字可以比较方便的转换为计算机内的二进制，那么字母怎么办呢？比如[abc.\_$+-\*/]这种字母和特殊字符，如何转换为计算机也能识别的二进制数字呢？

所以，美国定义了一张编码表，也就是ASCII表，ASCII 就是American Standard Code For Information Interchange，美国信息交换标准代码。

中国后来也有了自己的中文和二进制的编码表，GBK表。

有了这些编码表，所有的特殊字符在计算机中都有了对应的二进制数字，自然就能被编译和执行了。

有个很有意思的代码，可以试试看：

char ch = 97; 这段代码编译是通过的，因为97对应的二进制码在ASCII编码表中有对应的字符a。

char ch2 = ‘a’;

System.out.println(ch + 1);

这行代码编译也是成功的，且运行后得到的结果是98。这是由于char类型是16bit，1默认是int类型是32bit，所以在做ch+1 的运算时，ch会做自动类型提升转换为int值也就是97，那么97 + 1得到的结果自然就是98.

## 2.5运算符

## 2.5.1算术运算符

加减乘除[+, -, \*, /], 取余数又叫模[%], 自增即对自己加一[++]，自减即对自己减一[--]。

当++ 运算符和别的运算符在一起时，它的前后顺序是很重要的：

int a = 3,b;

b = ++a; // 这一步的运算过程是 a先自增，然后将a自增后的结果赋值给b为4.

b = a++; // 这个过程就和上面完全不同了，这里是先临时记录a的值3，然后a再自增变成4，最后给b赋值，但是给b赋值是用的a自增前的值，b的最终结果是3。

### 2.5.2赋值运算符

=，+=， -=， \*=， /=， %=

看一个很有意思的例子：

short s = 4;

s = s+5; // 这里会发生什么？编译会失败，因为这里等号右边是int，左边是short

s += 5; // 这行代码和上面一样吗？不一样，+= 是一个独立的赋值符号，它右边是常量.

### 2.5.3比较运算符

>，<，==，!=，>=，<=

### 2.5.4逻辑运算符

连接布尔型的表达式，就要用到逻辑运算符。

|  |  |
| --- | --- |
| 运算符 | 含义 |
| & | 与，先计算两边的结果，两边都为真，结果为真 |
| | | 或，先计算两边的结果，两边只要有一边为真，结果就为真 |
| ^ | 异或，先计算两边的结果，两边不同为真，相同为假 |
| ! | 非，取反 |
| && | 短路与，先计算左边，如果左边为真，再计算右边，两边都为真才为真，如果左边为假，直接判断为假，右边不计算 |
| || | 短路或，先计算左边，如果左边为真，直接判断为真，右边不计算 |

由上可知，&& 和&，以及|| 和 |，结果是一样的，但是计算过程不同。

### 2.5.5位运算符

对于二进制数字的运算符，如果是其他进制的数，则先将数字准换为二进制然后进行移位。

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 含义 | 运算逻辑 | 位运算的意义 |
| << | 左移：整体向左移动，左边移出的部分舍弃，右边空出的位置补0. | x左移n位的结果就是: x \* 2^n | 对于二进制的运算，移位的速度非常快。但是限制在于只能操作2的次幂 |
| >> | 右移：整体向右移动，右边移出的部分舍弃，左边空出的部分补什么取决于原来的数字是正数还是负数，正数补0，负数补1。因为最高位是符号位 | x 右移n位的结果就是：x / (2^n) |  |
| >>> | 无符号右移：整体右移，但是左边空出的位置只补0. |  | 无符号右移三位可以将二进制转换为八进制，移四位可以转换为十六进制。  所以当需要对二进制数字中的某一段操作时，使用无符号右移最方便。 |
| & | 与运算：转换为二进制后将每一位的数字进行&运算。只有全为1时才为1. | 6 & 3，转换为二进制：  0110 &  0011  ——  0010 = 2 | 用来取二进制位中的1时可以用&运算。  如：取000001101011中的后三位，只需要将它和000000000111进行&运算。 |
| | | 或运算 |  |  |
| ^ | 异或运算 | 6 ^3^3 = 6 | 一个数异或另一个数两次，结果还是这个数。可以做简单加密，比如一串需要保密的数字，你可以随便选一个数字（密码）异或它得到的就是解密后的数字，然后用你选的密码再异或一次就可以解密。 |
| ~ | 取反 | ~6 + 1 = -6 | 一个数取反再加一就是它的负数 |

练习题：OperatorExam.java

1. 最有效的方式计算 2 乘以 8

2. 对两个整数变量的值进行互换（不能使用第三方变量）

### 2.5.6三元运算符

（条件表达式）？表达式1：表达式2

当条件表达式为真时，结果为表达式1，否则为表达式2.但是必须要有一个结果。例如：a = a>3?5:3;

## 2.6 流程控制

### 2.6.1 if 判断结构

三种格式：if 后面的{}里的内容是if 的控制范围

1. if(){

}

1. if(){

}else{

}

这种方式和三元运算符很像，三元运算符某种程度上是 if else的简写，但是它有弊端，并不是所有的if else 都可以写成三元运算符，因为三元运算符运算完必须有结果，而if else 里面的内容不一定有结果。

1. if(){

}else if(){

}else{

}

练习题：IfExam.java

根据用户给定的数字，显示该数值对应的星期，如用户输入2，输出星期二。

### 2.6.2 switch选择结构

格式：

switch(表达式){ // 注意这里的表达式类型非常有限：byte,short,int,char,enum,String

case 取值1:

执行语句;

break;

case 取值2:

执行语句;

break;

case 取值3:

执行语句;

break;

…….

default:

执行语句;

break;

}

选择结构在写的时候没有顺序，比如可以把default 语句写在最上面，但是运行过程中是有顺序的，从上向下执行。

选择语句是从匹配的那一个值开始运行，到下面的break结束，如果当前case内没有break，而下面一个case有break，那么会运行下面一个case的内容，并到break结束。

switch 和 if 有什么区别？

If 可以用于判断数值，也可以用于判断区间，只要运算结果是boolean 类型都可以进行判断，而switch只能对固定的值进行判断，而且判断的值的类型有限：byte,short,int,char,String,Enum

对固定的几个值做判断时，最好是使用switch

### 2.6.3 循环结构——while, do…while, for

当需要循环执行某种操作时就可以使用循环结构。

1. while：先判断条件，当条件满足时，就进入花括号执行循环的内容。

格式：

while(条件){

循环执行的内容；

}

1. do…while：首先进入花括号执行一次循环的内容，然后再判断条件，条件满足接着执行，否者退出循环。意味着无论条件是什么，循环体至少执行一次。

格式：

do{

循环的内容；

}while(条件);

1. for：执行的顺序是：初始化表达式—》循环条件表达式判断—》判断为true则执行循环体，为false则退出循环—》执行循环后的操作表达式。

注意：初始化表达式只执行一次！

格式：

for(初始化表达式; 循环条件表达式; 循环后的操作表达式){

循环执行的内容;

}

思考：for 循环和 while 循环有什么区别？

for循环中的初始化表达式定义的变量，它的作用域是for的花括号，出了这个花括号内存中就不再存在了。

1. 循环嵌套：外循环和内循环，内循环是外循环的循环体，循环体执行完后才会执行外循环的执行后表达式。

练习题：ForLoopExam.java

在屏幕上打印出以下图形：

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

### 2.6.4其他流程控制语句——break, continue

break应用范围：只能在switch结构，或者循环结构中使用。在别的地方使用会编译报错。

break是指跳出，可以跳出正常的程序流程。

continue应用范围：只能在循环结构中使用，在别的地方使用会编译报错。

continue是指继续循环，在当前continue的位置直接进行下一次循环，意味着会跳过continue下面的代码。

到这里流程控制语句就讲完了，做一个总练习吧：FlowControlExam.java

1. 根据用户输入的月份，显示月份对应的季节。3/4/5春季，6/7/8夏季，9/10/11秋季，12/1/2 冬季，别的数字显示不属于月份。分别用if和switch实现。
2. 对给定的整数，获取其十六进制的表现形式。
3. 三个数，显示最大的数。
4. 在屏幕上显示如下内容：

54321

5432

543

54

5

1. 在屏幕上打印九九乘法表：

1\*1=1

1\*2=2 2\*2=4

1\*3=3 2\*3=6 3\*3=9

……

1. 在屏幕上打印出 “Hello World!”。注意，屏幕上也要有双引号

## 2.7 函数（方法）

### 2.7.1 函数的定义和特点

- 什么是函数？函数就是定义在类中的具有特定功能的一段小程序。函数也叫方法。

- 为什么需要函数？举例说明：

我需要打印一个矩形图案多次，正常情况下打印一次需要一个双层循环，那么打印多次就

是写多个双重循环，那么就有很多重复的代码，代码复用性很差。引发思考，我们是否可

以将打印矩形定义成一个单独的功能？然后调用这个功能来实现多次打印矩形？当然可以，这就是为什么需要函数。

总结：当我们需要实现某一个功能，且要循环地使用这个功能时，就可以把这个功能定义成一个单独的函数，这样可以增加代码的复用性。

- 函数的定义格式

这里我们先不直接提出函数的格式，而是先思考一个函数里应该有什么东西？函数其实就

是实现一个功能，然后被调用。为了区别不同的函数，首先它要有个名称（函数名），函数有自己的范围，必然需要用大括号括起来，里面就是它实现功能的代码，一般而言实现功能需要一些参与运算的数据（参数列表），计算完成后需要返回一个结果（返回值），返回的结果不确定，但是它的类型（返回值类型）肯定是一定的，否者调用它的人不知道怎么定义它，另外你在定义的函数的时候就要指定它可以被调用的范围（修饰符）以供调用者调用。

综上，函数的固定格式就出来了：

修饰符 返回值类型 函数名(参数列表){

执行语句;

return 返回值; // return关键字是为了结束该功能，并将结果返回给调用者

}

### 2.7.2 函数的应用

如何定义一个函数呢？不要直接套格式，从我们的需求开始思考，再代入格式。

首先，函数就是为了实现一个功能，功能需要什么？1. 结果 2.实现功能的内容

那么你每次定义函数前，就先思考两个问题：

1. 结果是什么？就可以明确返回值类型
2. 要实现什么功能？需要一些变量吗？需要的话，参数列表就明确了。

注意：一个行数最好不要超过20行，方便阅读和调试，如果功能很复杂，可以把功能分离成多个小功能再定义多个函数。

### 2.7.3 函数的重载（overload）

- 重载的概念：在同一个类中，允许存在多个同名的函数，只要它们的参数列表（参数个数或类型）不同即可。简单理解就是，允许重复载入函数。

- 重载的特点：与返回类型无关，只关注参数列表。

为什么允许多个重名的函数？因为方法名其实就是为了描述方法的功能，比如求和功能，方法名就是sum，但是可以给两个数求和，也可以给三个数求和，所以可以定义多个名为sum的函数，只要它们的参数列表不同。

问题：

sum(int a, double b) 和 sum(double b, int a) 重载了吗？

重载了，因为参数类型不同，注意，函数的参数是有顺序的。按照顺序判断参数类型。

- 函数在内存中的调用过程：

如下面示例：

public static void main(String[] args){

int sum = add(2,5);

System.out.println(sum);

}

public static int add(int a, int b){

return a+b;

}

在上面的示例中，有两个方法，那么它们在内存中的执行顺序是怎样的呢？首先需要知道，main方法是程序执行的入口，所以当程序开始执行，首先执行main方法。

main

sum = 7

1.首先JVM执行main方法，main方法在内存中有自己的一块运行空间.

里面的代码一行一行的按顺序执行，执行到 sum 时，需要调用add方法，此时内存中会创建add方法的运行空间

add

a = 3

b = 4

2.内存中创建add的运行空间，它有自己的变量a和b，然后进行 a+b 的运算，运算结束后，return 用来结束add函数并将这个结果传递给main方法。

3. 当add返回值之后，add运算结束，内存中会释放add的空间。

栈

上述运行过程中存储运算的内存专业名称叫做栈，执行过程中每执行一个方法，就在栈中为该方法创建一片空间，专业术语叫做压栈又叫进栈，运行完之后释放这片空间叫做弹栈又叫出栈。特点是先进后出。是不是和手枪放入子弹和打出子弹很相似？

## 2.8 数组

当我们求和时，试过两个数求和，三个数求和，但是如果我要对100个数进行求和怎么办？由此引申出数组的概念。数组字面意思就是一组数据。

### 2.8.1 数组的定义和特点

- 数组的概念：同类型的一组数据的集合。数组就是一个容器。

- 如何定义数组：老样子先不要直接套格式，先思考。

首先需要知道，当需要一个变量存储很多的数据时，就不能像普通的数据那样直接定义了，需要用到一个关键字，就是 new，意思是新建一个容器。

其次数组有自己的形式，用[]表示，那么简单的思考就是 new []，然后继续思考，你这个数组要装几个元素啊？例如装5个，那么就是 new [5]，再思考，数组是同一类型的数据集合，那么你的数组存的数据是什么类型呢？例如是int类型，那么就是 new int[5]。这样就定义好了一个数组。

然后你得把它赋值给一个变量啊，假如变量名字是arr，这个变量是数组类型，数组用[] 表示，所以是 [] arr = new int[5]; 然后继续思考，你这个数组存的是什么类型的数据啊，哦是int类型的，所以就是 int[] arr = new int[5]; 这个语句就定义好了一个长度为5，存储的数据是int类型的数组。那么第一种格式就出来了

格式1：元素类型[] 数组名 = new 元素类型[元素个数又叫数组长度];

- 数组的特点和好处：数组可以存储多个元素，并且可以对里面的元素进行编号，编号的官方术语叫做索引，从0开始，依次增加1。如果你定义了数组，但是没有往里添加元素，数组依然有默认值，它每个元素的默认值就是该元素对应的那个类型的默认值，如int型默认为0。

### 2.8.2 数组的内存图解（堆&栈）

想要更清楚的了解数组，还是需要了解它在内存中的空间。了解数组内存之前，必须先介绍一下内存以及java在内存当中的分布情况。

首先，什么是内存？

内存（Memory）其实是电脑的一个硬件组成部分，它是与CPU直接相连，电脑当中所有程序的运行都是在内存中进行的。内存也被称为内部存储器，作用是暂时存放CPU中的运算数据，以及与外部存储器交换的数据。只要电脑在运行中，CPU就会把需要运算的数据调到内存中进行运算，运算完成后CPU再将结果传送出来。

然后需要知道，任何一个应用程序，它在内存中运行之前，必须要先进行空间的划分。

Java也是一样，java应用程序启动时，也会划分Java应用程序的内存区，而且java内存区中还有细分。为什么细分？因为每一片细分的内存区域处理数据的方式不同。共有5片细分：

栈、堆、方法区、本地方法区、寄存器

1. 栈内存：

存储局部变量，只要是方法中定义的变量都是局部变量。栈中会分配各个方法的内存区，然后给方法中的局部变量分配空间。

问题：for 循环定义的变量在栈内存吗？当然是。

特点：一旦变量的生命周期结束，该变量就被释放。

2. 堆内存：

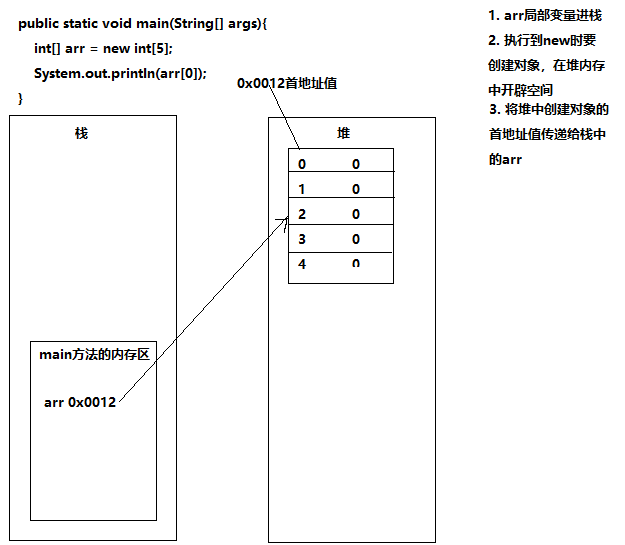
存储实体，也就是对象。凡是用new创建的都是对象。

特点：- 每一个对象都有一个首地址值。

- 堆内存中的变量都有默认初始化值，不同类型不一样，int-0, double-0.0, float-0f, boolean-false, char-‘\u0000’

- 当对象不再被使用时，不会自己释放空间，而是会被垃圾回收机制处理。

下图就解释了main方法中创建数组的内存机制：



ArrayIndexOutOfBoundsException: 数组索引越界，意思是访问的索引在数组内不存在。

NullPointerException: 空指针异常。当你把数组赋值为null后再进行索引操作，就会发生空指针异常。

### 2.8.3 数组的第二种定义方式

元素类型[] 数组名 = new 元素类型[]{元素1， 元素2， 元素3……}

如：int[] arr = new int[]{1,2,3,4,5,6};

这种方式和第一种有什么区别？

1. new后面的[]中不写数组长度了，为什么？
2. [] 后面跟了一对{}，而且里面列举了所有的数组元素。既然已经列举了所有的元素，那么数组的长度已经确定，自然就不需要在[]中写数组长度了。

观察以上的定义方式，可以发现{}里面已经列举了所有的元素，可以确定元素的个数和元素的类型，那么new int[] 就显得有些多余了。所以上面这种方式可以简化为：

int[] arr ={元素1， 元素2， 元素3……}}; // 如：int[] arr = {1, 2,3,4,5,6};

如何获取数组的长度？通过arr.length 来获取，length是数组类型的一个属性。

### 2.8.4 数组的应用

通过练习题来逐步了解数组的应用。

练习题：

1. 对多个整数求和。
2. 在多个整数中获取最大值。
3. 将数组中的元素转换成字符串，如将{34,35,36} 转换为 “[34， 35， 36]”

数组是一种容器，我们知道容器就是用来存储多个数据，虽然我们现在只学习了数组容器，但是以后还会学习到别的容器，那么我们什么时候用数组容器？什么时候用别的容器呢？

首先回顾一下数组的特点：1.元素个数确定；2.元素都有自己的编号，且是有规律的编号。

那么当我们分析需求时，在满足以下情况时考虑使用数组：

* 需求中的数据个数确定。
* 需求中的数据有对应的有序编号。
* 需求中的数据需要有序编号。

例如：我们之前学习if和switch时做过一个练习，根据用户输入的数值打印对应的星期。这个需求中需要的数据个数确定，有7个，且数据和结果间有很明确的对应关系，那么就可以考虑用数组实现，把用户输入的数据作为索引，把星期作为数组的元素。

以上的例子中，数组是不是就相当于一个表？我们通过索引去查表找到对应的值，这就叫查表法。查表法的特点是：数据之间存在对应关系（只要有关系就可以，不一定非要有序），就可以通过一方查询另外一方。

练习题：

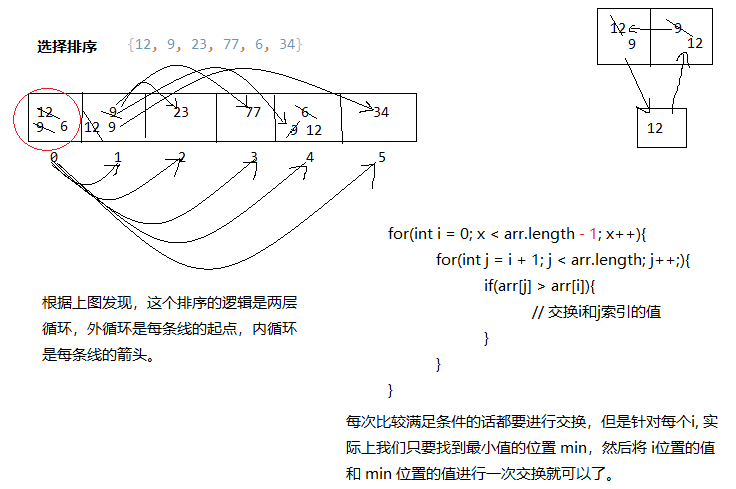
1. 十进制转十六进制使我们之前做过的练习，现在试着把十六进制的结果的每个位按顺序存在数组里: ArrayExam1.java

打印这个十六进制数，并且去除多余的0.如：得到的值是0x0000003c，但是打印0x3c。思考如何去除0？每次运算后都要右移，如果右移后剩余的二进制位全为0，那就意味着没有有效位了，就不必再进行下一次运算。

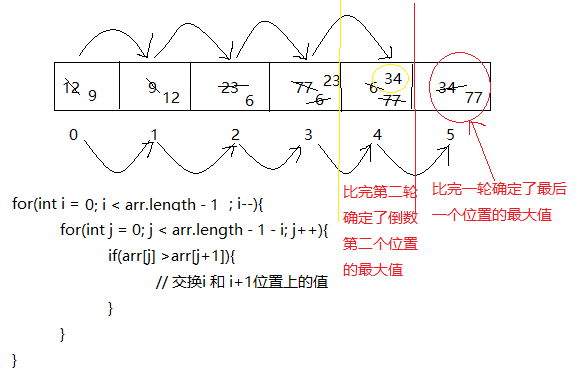
既然已经实现了十进制转十六进制，扩展一下吧，提供一个通用的进制转换方法，实现可以从十进制转二进制、八进制和十六进制。

1. 对一组数据进行从小到大的排序：ArrayExam2.java.

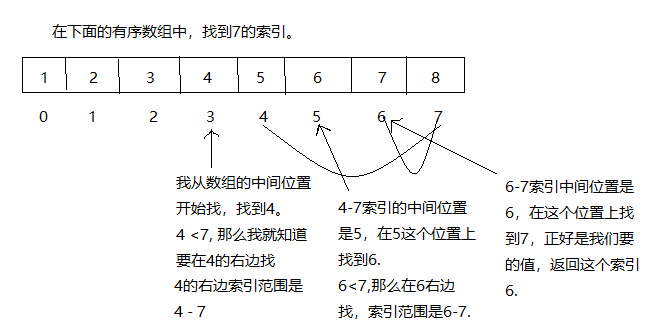
图解选择排序：



图解冒泡排序：



1. 在数组中查找某个元素的位置：ArrayExam3.java.
2. 将数组从头到尾遍历一遍，如果找到这个元素，返回位置。
3. 二分查找法图解：



1. 数组反转：ArrayExam4.java。

对一个给定的数组进行反转，如：{23, 14, 88, 5} 转换为 {5, 88, 14, 23}:

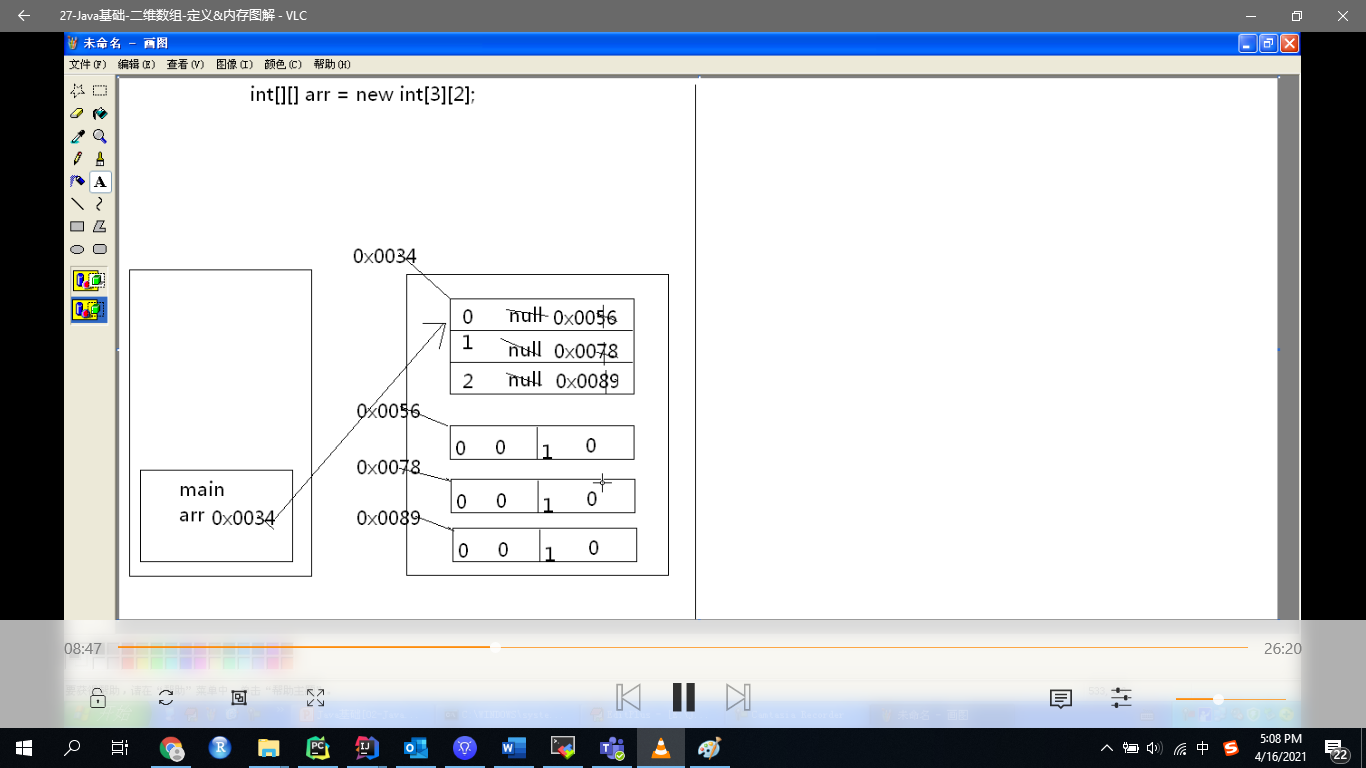
1. 获取插入点：ArrayExam5.java。

有序数组中插入一个元素并继续保持有序，如何获取该元素应该 插入的位置。

### 2.8.5 二维数组[][]

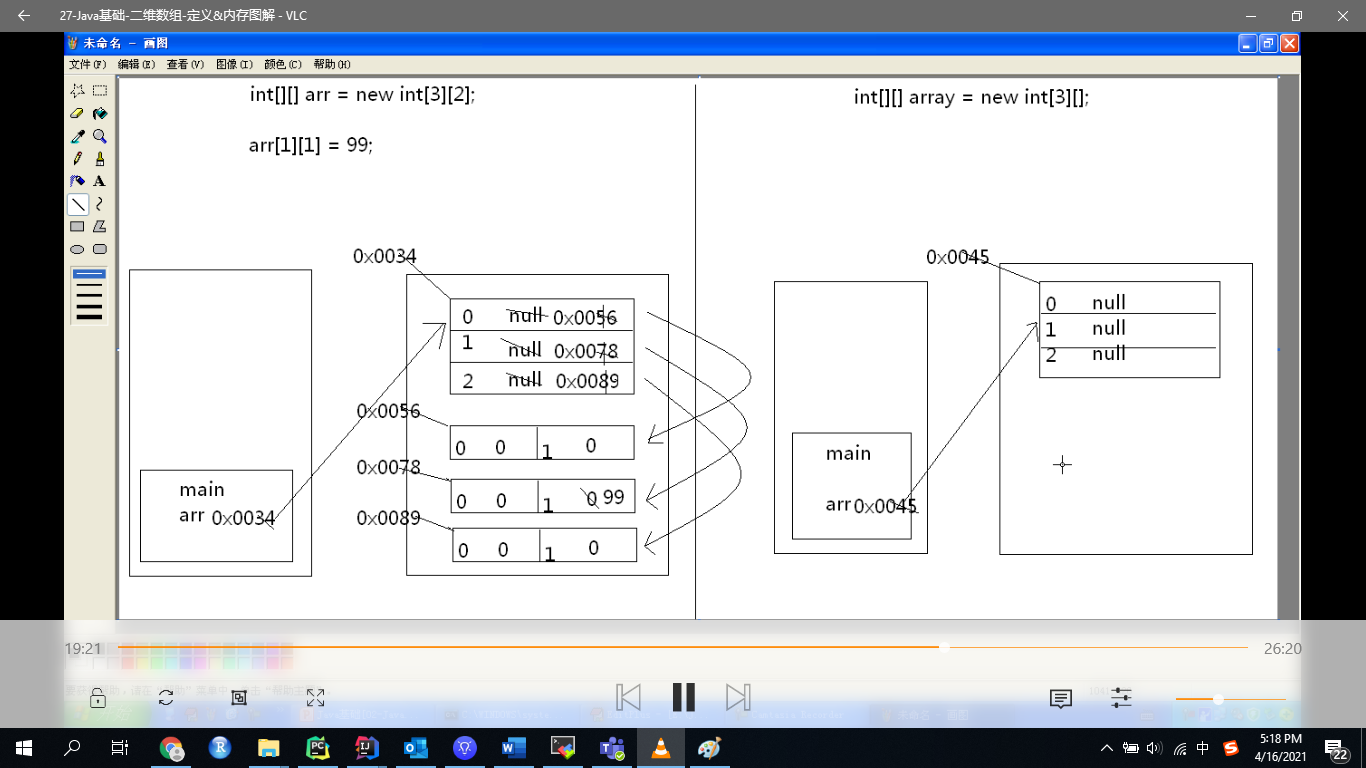
定义：int[][] arr = new int[3][2];

二维数组的内存图解：



定义的第二种方式：int[][] arr = new int[3][]; // 明确了二维数组的长度，没有明确具体的一维数组。

内存图解：当没有定义一维数组的长度时，堆内不会为它创建空间。数组是实体，实体的默认值是null。



二维数组里面的一维数组是可以有不同长度的：

Arr[0] = new int[3];

Arr[1] = new int[1];

Arr[2] = new int[2];

定义的第三种方式：int[][] arr = {{1,2,3}, {4,5,6}, {7,8,9}};