Java软件设计基础

2. Java语言基础

2.1 符号

- 标识符
  - 程序中要用到许多名字，诸如类、对象、变量、方法等。标识符就是用来标识它们的唯一性和存在性的名字。
  - Java采用Unicode字符集，由16位构成。
  - 标识符分为两类:
    - 保留字：是Java预定义的标识符，都具有特定的含义，保留字又称关键字。

abstract	double	int	strictfp	boolean	else
interface	super	break	extends	long	switch
byte	final	native	synchronized	case	finally
new	this	catch	float	package	throw
char	for	private	throws	class	goto
protected	transient	const	if	public	try
continue	implements	return	void	default	import
short	volatile	do	instanceof	static	while
assert	enum				

- 用户定义标识符：是程序设计者根据自己的需要为定义的类、对象、变量、方法等的命名。
- 用户自定义标识符的定义规则：以字母、下划线或$符开头的字母、下划线、数字、 $符的序列。
  - 标识符区分大小写。
  - 标识符不能与保留字同名。
  - 标识符遵守先定义后使用的原则。
  - 虽然true、false和null并不是关键字，但其代表的是值，也不可以用以上三个作为自定义标识符的名字。

标识符的长度是任意的。虽然如此，但不宜过长，也不宜取难以理解的简写。最好有象征性含义，起到见文生意的作用，提高程序的可读性。

一些由开发环境自动生成的名称中会带有$符或下划线，因此虽然规则允许，但是自定义的变量名称中应尽量避免使用$符或下划线。

2a(不应以数字2开头）、
break（禁止使用关键字）、
TWO WORDS（含有空格）、
.NO（不允许以字符圆点开头）

A、a1、$Systembol、
square、ex_sa

- 分隔符
  - 规定任意两个相邻标识符、数、保留字或两个语句之间必须至少有一个分隔符，以便编译程序能识别。
  - 分隔符不能互相代用。
  - 分隔符的分类
    - 空白分隔符
      - 空格、TAB制表符、换行符与回车符都是典型的空白分隔符。
      - 为了程序的可读性和美观，语句的成分之间可以插入任意多个空白分隔符，在编译时系统会自动忽略多余的空白分隔符。

- 普通分隔符
  - {}用来定义复合语句、类体、方法体以及进行数组的初始化等。
  - ;表示一条语句的结束。
  - ,用来分隔变量的说明和方法的参数等。
  - :说明语句标号等。

- 注释语句
  - 注释用来对程序中的代码做出解释。注释部分对程序的执行不产生任何影响，可增加程序的可读性，有利于程序的修改、调试、交流。
  - 注释语句的格式
    - 第一种用于行注释；第二、三种用于多行注释。

注释格式

```
//注释说明内容
```

注释格式

```
/*
        注释说明内容
*/
```

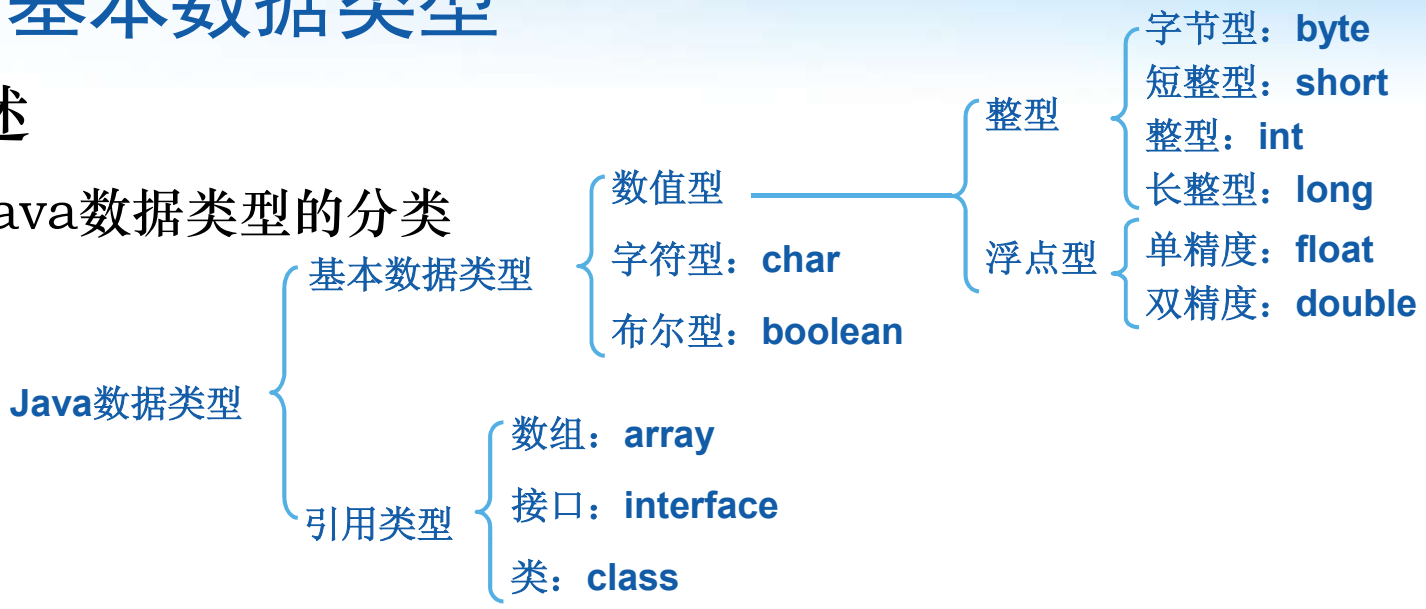
注释格式

```
/**
        注释说明内容
*/
```

- 标识符命名建议
  - 尽量使用完整的英文描述符；
  - 采用大小写混合使名字可读，采用适用于相关领域的术语；
  - 尽量少用缩写；
  - 避免使用长的和类似的名字，或仅仅是大小写不同的名字；
  - 除静态常量外，尽量少用下划线。
- 源文件命名规则
  - 源程序中包含有公共类的定义，源文件名必须与该公共类的名字一致。在一个源程序中至多只能有一个公共类的定义；
  - 源程序中不包含公共类，则该文件名只要和某个类名字相同即可；
  - 源程序中有多个类的定义，编译时将会为每个类生成一个class文件。

2.2 基本数据类型

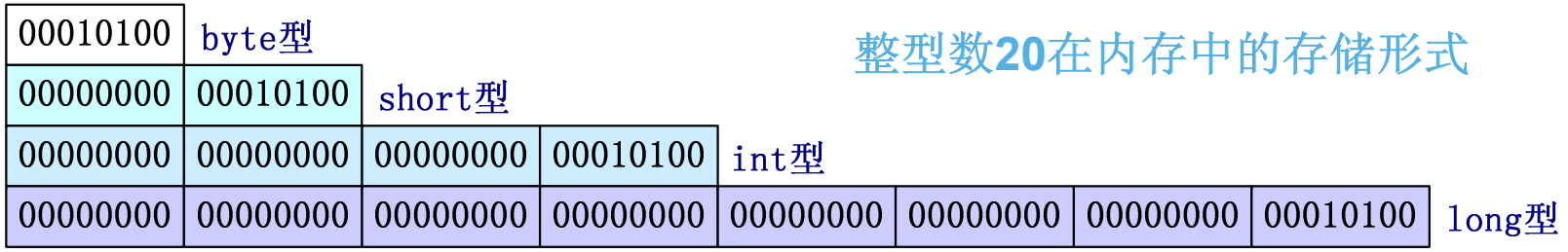
- 概述
  - Java数据类型的分类



  - Java数据存储空间大小

数据类型名	占用内存空间	数值范围	缺省值	说明
byte	1 字节	(－128) ～127	0	整型
short	2 字节	(－32768) ～32767	0	
int	4 字节	(－2147483648) ～2147483647	0	
long	8 字节	(－9223372036854775808) ～9223372036854775807	0	
float	4 字节	(±3.4028347E+38) ～ (±1.40239846E－45)	0.0f	实型（浮点型）
double	8 字节	(±1.79769313486231570E+308) ～ (±4.94065645841246544E－324)	0.0d	
char	2 字节	\u0000～\uFFFF	\u0000	字符型
boolean	1 字节	true 或 false	false	布尔型

- 整数类型
  - 不同整数数据类型的意义在于它们所需的内存空间大小不同，这也决定了它们所能表达的数值范围的不同。



  - 在内存紧张的情况下，可将byte或short用于大型数组以便节省内存。
  - int是整数数值的默认选择类型，以下两种情况除外：
    - 将一个较小的数（byte或short范围内）赋值给一个byte或short变量，系统会自动把该整数值当成byte或者short类型来处理；
    - 当将超过int值范围的值付给long类型的时候，必须在值后加上L或者l。

- Java中整数有四种表示形式
  - 十进制整数，如33，58，-90。
  - 八进制整数，以0开头，如010表示十进制的8。
  - 十六进制整数，以0X(0x)开头，如0x10表示十进制的16。
  - 二进制整数，以0B(0b)开头
- 字符型
  - Java的字符使用16位的Unicode编码表示，它可以支持世界上的所有语言。“\u0000”~“\u00FF”用来表示ASCII码集。
  - 表示形式
    - 包括在单引号之内的单个字符；
    - 用单引号括起来的Unicode字符，形式为’\uxxxx’，x的范围是0~F
    - 通过转义字符表示特殊字符，如’\n’等

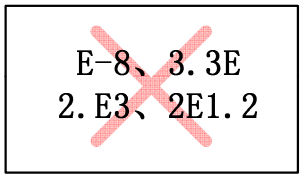
换码序列	含义	Unicode 值
\b	退格	\u0008
\t	Tab（制表）	\u0009
\n	换行	\u000a
\r	硬回车	\u000d
\"	双引号	\u00022
\'	单引号	\u00027
\\	反斜杠	\u005c

- 注意
  - char类型在保存时实际是保存其对应的编号，即0~65535中的某个值，它相当于一个16位的无符号数，可以进行数学运算。
- 浮点型
  - 采用二进制数据的科学计数法表示：
    - float：第1位是符号位，接下来的8位表示指数，接下来的23位表示尾数；
    - double: 第1位是符号位，接下来的11位表示指数，接下来的52位表示尾数；

- 表示形式
  - 十进制形式：512.00，
  - 科学计数法或指数形式
    - 字母E表示以10为底的指数。采用科学表示法时，尾数必须有，但小数部分可无，阶码必须有且是整数。

−1.2345678901E+12f

尾数　　阶码



  - 注意：Java语言的浮点类型默认是double类型，给float变量赋予带小数的初值时必须在数值后加f或F。
- Java7提供了下划线功能，用于分隔很长的数值，方便观看：

```
double pi=3.14_15_92_65_36;
```

- 布尔型
  - 被用作真/假条件的简单标志，值为true或false，它的值不与任何整数值对应。

Java不强制指定boolean类型的变量所占用的内存空间，虽然该变量只需要一位即可保存，但由于大部分计算机在分配内存时允许分配的最小内存单元是字节（1 byte），所以部分时候实际上占用1byte。

基本数据类型

```
public class BasicJava {
        public static void main(String[] args)
        {
                long la=2547483647L;
                long la=2547483648;             //编译错误；
                float fa=2;
                double fb=2;
                float fc=2.6;                   //编译错误；
                float fd=1.23e1;                //编译错误；
                float fe=1.23e1f;
                char ch1='B';
                char ch2=67;
                System.out.println(ch1+3);
        }
}
```

- 字面量/直接量（literal）
  - 字面量是固定值的源代码表现形式，直接出现在代码中。
  - 直接量的类型：
    - 整形、浮点型、布尔型、字符型、String、null类型

字面量

```
boolean result=true;
char capitalC='c';
byte b=100;
short s=10000;
double d=12.45;
```

    - 特殊的字面量null
      - 可以用作任何引用类型的值，可以赋值给除了属于原始数据类型的变量之外的任何变量。除了测试其存在之外，对null值没有什么可做的操作，因此在程序中常使用null作为标识，表示某个对象不可用。

- 默认值
  - 声明字段时不必为其赋值。被声明但是没有初始化的字段会被编译器设置为合理的默认值。

数据类型	默认值
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	‘\u0000’
String(或者任何对象)	null
boolean	false

  - 局部变量稍有不同，编译器永远都不会给未初始化的局部变量分配默认值。如果在声明局部变量时不能初始化它，就要确保使用之前为其赋值，否则将导致编译错误。

默认值例程

```
public class Test1{
        //声明字段 i
        static int i;
        public static void main(String args[]){
                //声明局部变量 j
                int j;
                System.out.println(i);
                System.out.println(j);
        }
}
```

```
E:\javadoc\Test1.java:6: 可能尚未初始化变量 j
        System.out.println(j);
                           ^
1 错误
```

2.3 常量与变量

- 常量
  - 常量是指在程序运行过程中其值不变的量。常量在表达式中用文字串表示，它有整型常量、字符型常量等等不同的类型。
  - 常量通过用关键字final来实现声明，通常写在最前面。

```
final 类型 常量名=常量值;
```

- 变量
  - 变量用来存放指定类型的数据，其值在程序运行过程中是可变的。
  - 变量的声明
    - 使用一个变量之前必须先声明它。一方面给该变量分配内存空间，另一方面防止在以后使用此变量时因错误输入而对不存在的变量进行操作。

```
[修饰符] 类型名 变量名1[,变量名2][,...];
```

```
[修饰符] 类型名 变量名1[=初值1][,变量名2][=初值2][,...];
```

- 以“变量名”为名建立一个某类型的变量；
- ;表示声明语句的结束；
- [ ]表示可选；
- 可在一条语句中定义多个同类型的变量，中间用逗号隔开；
- 在声明的同时可以赋值；
- 修饰符也称为作用域，指明作用域的类型；
- 数据类型决定了变量所包含的值的范围、可对变量进行哪些操作以及如何定义这些操作；

- 变量的使用
  - 变量的初始化是简单的赋值使用，当在语句中使用到该变量的名称时，编译器会自动将当时变量的值取来用。
  - 局部变量在使用前必须初始化。

- 变量的类型
  - 基本数据类型（布尔型、字符型、整型、浮点型）、引用类型（数组、接口、类）
- 变量的作用域
  - Java中的变量有一定的生存期和有效范围，变量的作用域指明可访问该变量的一段代码。
  - 按照作用域划分：
    - 成员变量：可以在整个类中被访问；
    - 局部变量/方法参数：在方法或方法的一个代码块中声明，它的作用域为它所在的代码块；

```
public class BasicJava {
        public static void main(String[] args)
        {
                int i=15;
                {
                        int j=58;
                        System.out.println("i="+i);
                        System.out.println("j="+j);
                }
                j=i;                                    //编译错误；
                System.out.println("i="+i);
                System.out.println("j="+j);     //编译错误；
        }
}
```

- 将语句”int j=58;”放在语句”int i=15;”后面，则变量j的作用域变为整个main()方法，程序编译通过。

2.4 运算符

- 程序中用来处理数据、表示数据运算、赋值和比较的符号称为运算符，参与运算的数据称为操作数。
- 分类
  - 算术运算符
    - 算术运算符用于对整型数和实型数的运算。按照其要求的操作数个数分为一元运算符和二元运算符两类。
      - +（正号）、-（负号）、++（自增）、--（自减）；
      - +（数值加、连接字符串）、-（减）、*（乘）、/（除）、%（取模）。
    - ++x是在变量参与运算之前自增1，然后用新值参与运算；而x++是先用以前的值参与运算，再自增1；--x和x--同理。
    - 除号“/”中，如果运算符两边的操作数都是整数，则计算结果只保留整数部分。
    - 当+作为字符串连接操作符时，可以连接两个字符串、字符串与字符、字符串与数字。

算术运算符

```
public class BasicJava {
        public static void main(String[] args)
        {
                int x1=10,x2=10,x3=10,x4=10;
                        {int y1,y2,y3,y4;
                        y1=++x1;
                        y2=x2++;
                        y3=--x3;
                        y4=x4--;
                        System.out.println("y1="+y1+" y2="+y2+" y3="+y3+" y4="+y4);
                        System.out.println("x1="+x1+" x2="+x2+" x3="+x3+" x4="+x4);}
                        {d1=x1/x3;
                        System.out.println(d1);}
                        {String s1="my";
                        String s2="java";
                        char c1='s';
                        System.out.println(s1+s2);
                        System.out.println(s1+c1);
                        System.out.println(s1+5);}
        }
}
```

- 比较运算符
  - 又称关系运算符，有==（等于）、!=（不等于）、>（大于）、<（小于）、>=（大于等于）、<=（小于等于）、instanceof（是否为某类对象）几种；
  - 在一个比较运算符两边的数据类型应该一致，比较逻辑成立为true，不成立为false；
  - 如果操作数是对象变量，则对象变量引用同一个对象或都为null的时候为true，否则为false。
- 逻辑运算符
  - 又称布尔运算符，用于对布尔型操作数进行计算。
    - ！（非）
    - &（与）、|（或）、^（异或）、&&（短路与）、||（短路或）。
    - &&和&的区别是，前者只要左边的操作数为false时，就不再计算，直接给出结果是false；||和|的区别同样。简化了计算过程。

- 位运算符
  - 对整数数值二进制表示中的每位进行测试、置位、移位等处理。
    - ~（位反）
    - &（位与）、|（位或）、^（位异或）、<<（位左移）、>>（位右移）、>>>（不带符号的位右移）
  - Java使用补码来表示二进制码，最高位为符号位，整数的符号位为0，负数的符号位为1。
    - 正数：$[x]_{原码}$= $[x]_{反码}$=$[x]_{补码}$
      - **5的补码为**00000101。
    - 负数：$[x]_{反码}$为$[x]_{原码}$除了最高位以外按位求反，$[x]_{补码}$为$[x]_{反码}$+1。
      - **-5的原码为**10000101，**反码为**11111010，**补码为**11111011。
  - 在移位时，位左移右边补0；位右移时，高位移入原来最高位的值；无符号右移时，低位被舍弃，高位补0。
  - 在对byte和short类型的值进行位移运算时，Java将自动把这些类型扩大为整形。

位运算举例

```
public class Bit_Log {
        public static void main(String[] args)
        {
                int x1= 124,x2=55;
                int y1= 52,y2= 45;
                int z=15;
                System.out.println(~x1);
                System.out.println(Integer.toBinaryString(~x1));
                System.out.println(x1&y1);
                System.out.println(Integer.toBinaryString(x1&y1));
                System.out.println(x1|y1);
                System.out.println(Integer.toBinaryString(x1|y1));
                System.out.println(z<<2);
                System.out.println(Integer.toBinaryString(z<<2));
                System.out.println(x2>>2);
                System.out.println(Integer.toBinaryString(x2>>2));
                System.out.println(y2>>2);
                System.out.println(Integer.toBinaryString(y2>>2));
                System.out.println(x2>>>2);
                System.out.println(Integer.toBinaryString(x2>>>2));
                System.out.println(y2>>>2);
                System.out.println(Integer.toBinaryString(y2>>>2));
        }
}
```

x1=-124
x2=55
y1=-52
y2=-45
z=15;

$[x1]_{补码}$=11111111111111111111111110000100
$[x2]_{补码}$=110111
$[y1]_{补码}$=11111111111111111111111111001100
$[y2]_{补码}$=11111111111111111111111111010011
$[z]_{补码}$=1111

~x1=123	$[\sim x1]_{补码}$=1111011
x1&y1=-124	$[x1\&y1]_{补码}$=11111111111111111111111110000100
x1|y1=-52	$[x1\|y1]_{补码}$=11111111111111111111111111001100
z<<2=60	$[z<<2]_{补码}$=111100
x2>>2=13	$[x2>>2]_{补码}$=1101
y2>>2=-12	$[y2>>2]_{补码}$=11111111111111111111111111110100
x2>>>2=13	$[x2>>>2]_{补码}$=1101
y2>>>2=1073741812	$[y2>>>2]_{补码}$=11111111111111111111111111110100

- 条件运算符
  - 是一个三元运算符“？：”，它是Java中唯一的三元运算符，形式如下：

    布尔表达式?表达式1: 表达式2
  - 其中表达式的值为一个布尔值，如果结果为true，则整个表达式的值为表达式1的值；否则为表达式2的值，表达式1和表达式2应返回相同的数据类型。
- 赋值运算符
  - 简单赋值运算符“=”：把运算符右边的值赋给左边的常量或变量。在一个赋值表达式内也可以连续赋值。
  - 扩展赋值运算符：在“=”前加上其他运算符。
    - +=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=、>>>=

    s op= i ⟺ s=s op i

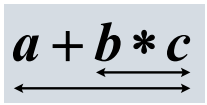
- 其他运算符

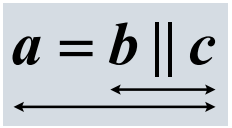
运算符	功能
( )	表达式加括号优先执行
(参数表)	方法参数传递，多个参数时用逗号隔开
(类型)	强制类型转换
.	分量运算符，用于对象属性或方法的引用
[ ]	下标运算符，用于数组
new	对象实例化运算符，实例化一个对象，即为其分配内存

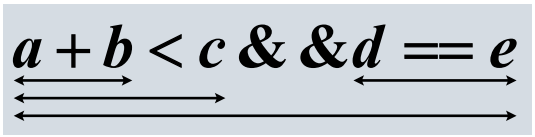
- 运算符的优先级
  - 表达式的运算次序取决于表达式中各种运算符的优先级。优先级高的先运算，优先级低的后运算。
  - 另外，还可以用“()”改变优先级次序。

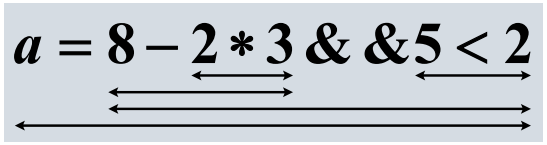
优先级	运算符	描述	结合性
1	. [ ] ( )	域、数组、括号	左→右
2	++ -- + - ! ~ instanceof	一元操作符	右→左
3	new (type)	新建对象、强制类型转换	左→右
4	* / %	乘、除、取余	左→右
5	+ -	加、减	左→右
6	>> << >>>	位运算	左→右
7	> < >= <=	比较运算	左→右
8	== !=	比较运算	左→右
9	&	与	左→右
10	^	异或	左→右
11	|	或	左→右
12	&&	逻辑运算	左→右
13	||	逻辑运算	左→右
14	?:	条件运算符	右→左
15	= += -= *= /= %= ^=	扩展赋值运算符	右→左
16	&= |= <<= >>= >>>=	扩展赋值运算符	右→左

- 例：按照上表的优先级规定，下面表达式有唯一的计算顺序：









- 注意
  - 上述表达式的写法比较难以理解并容易产生错误。因此在写程序时尽量使用括号来使表达式的计算次序一目了然。维持这种编程习惯可以使代码更易于阅读和维护。
  - 数学上的表达式如a<b<c在Java中必须写成(a<b)&&(b<c)这样的形式。

2.4 表达式、语句和块

- 表达式（Expression）
  - 表达式是由操作数和运算符按照一定的语法形式组成的符号序列，计算出单一值，该值的类型取决于表达式中使用的元素。
  - 表达式是语句的核心部分。
  - Java允许使用各种较小的表达式构成复合表达式，但表达式各个部分的数据类型要匹配。

表达式

```
int cadence=0;
anArray[0]=100;
System.out.println("Element 1 at index 0:"+anArray[0]);
int result=1+2;
if(value==value2)
        System.out.println("value==value2");
```

- 语句（Statement）
  - 语句是程序的基本组成单位，组成了一个完整的执行单元，大致相当于自然语言中的句子。下面的表达式类型以;结尾时可以组成一个语句：
    - 赋值表达式；
    - ++或--；
    - 方法调用；
    - 对象创建表达式。
  - 上述语句被称为表达式语句。除了这些，还有另外两种语句：
    - 声明语句
      - 用来声明一个变量等。
    - 控制流语句
      - 控制语句的执行次序。

- 块（block）
  - 是位于成对大括号之间的零个或多个语句的语句组，可以在允许使用单一语句的任何位置使用块。

块

```
class BlockDemo {
        public static void main(String[] args)
        {
                boolean condition = true;
                if (condition)
                {//块1开始
                        System.out.println("Condition is true.");
                }//块1结束
                else
                {//块2开始
                        System.out.println("Condition is false.");
                }//块1结束
        }
}
```

2.5 类型转换

- 自动类型转换
  - 整型、实型、字符型数据可以混合运算。运算过程中，不同类型的数据会自动转换为同一类型，然后进行运算。
  - 自动转换按低级类型数据转换成高级类型数据的规则进行，最后生成的值也是高级类型数据。
  - Java定义了若干适用于表达式的类型提升，所有的byte和short型的值都会被系统提升到int型。
    - 如果一个操作数是long型，那么整个表达式会被提升到long型。
      - (1)(byte或short) op int → int
      - (2)(byte或short或int) op long→long
      - (3)(byte或short或int或long) op float→float
      - (4)(byte或short或int或long或float) op double→double
      - (5)char op int→int

- 强制类型转换
  - 不是所有的数据类型都允许隐含性的自动转换。当把占位较长的数据转化为占位较短的数据时，会出现信息丢失的情况，因为不能自动转换。高级数据类型要转换成低级数据类型，需要用到强制数据类型转换。其一般形式为：

    （数据类型）表达式　　数据类型（表达式）

    - 经过强制类型转换将得到一个在( )中声明的类型的数据。
    - 将占用位数较长的数据转化成占用位数较短的数据时，可能会造成数据超出较短数据类型的取值范围，造成溢出。
  - 复合数据类型也可以进行转化。
  - 一般使用强制类型转换可能会导致数值溢出或精度下降，应尽量避免。

类型转换

```
public class TypeChange {
        public static void main(String[] args)
        {
                byte x=7;short y=12;
                int z=x+y;
                short a=7,b=5;
                short c=a+b;                    //报错
                int i=5;
                double dd=i+2.5;
                int i1=dd+7;                    //报错
                boolean bool=true&i;            //报错
        }
}
```

short c=a+b;
因为Java的自动提升功能，导致结果是int类型，无法赋值给范围更小的short类型

int i1=dd+7;
dd+7的值是double型，无法赋值给int类型，如果改为“int i1=(int)dd+7;”则编译通过

boolean bool=true&i;
布尔型与数字类型、字符型互相不兼容。

2.6 数组

- 数组是相同类型的数据元素按顺序组成的一种引用类型变量。
- 特点
  - 一个数组中所有的元素应该是同一类型；
  - 数组中的元素是有序的；
  - 数组中的一个元素通过数组名和数组下标来确定。
- 分类
  - 基本数据类型的数组与复合数据类型的数组
  - 一维数组和多维数组

- 一维数组
  - 声明格式：

```
类型[] 数组名;
```

```
类型 数组名[];
```

    - 数组名的命名方法同简单变量，可以是任何合法的标识符；
    - 类型标识符可以是基本数据类型或者是类、接口；

一维数组的声明

```
int[ ] myArrayA;
double myArrayB[ ];
```

Tips:对于这两种声明格式而言，通常推荐第一种格式，因为第一种很容易理解这是定义一个变量myArrayA，该变量的类型是int[]

  - 分配数组空间
    - 在声明数组时，不直接指出数组中的元素个数（即数组长度）。数组声明之后不能立即被访问，因为此时定义为数组的这个变量只是一个引用类型的变量，这个引用变量还未指向任何有效的内存，也就没有内存空间来存储数组元素。因此只有使用new操作来构造数组，为其分配内存空间后，数组才能使用。

```
数组名 = new 类型[元素个数];
```

分配内存空间

```
myArrayA = new int[8];
```

- 元素个数即数组长度。数组分配空间是连续的，可以通过属性length获得该数组的元素个数，方法如下：

```
数组名.length;
```

- 可在声明数组的同时分配数组空间

```
类型[] 数组名 = new 类型[元素个数];
```

声明并分配内存空间

```
int[ ] myArrayA = new int[8];
double[ ] myArrayB = new double[10];
```

- 数组一旦创建之后，就不能再改变其长度！

- 数组的初始化
  - 用new分配空间后，系统将为每个数组元素都赋予一个初值，初值取决于数组元素的类型：
    - 数值型：0/0.0
    - 字符型：不可见ISO控制符（\u000）
    - 布尔型：false
    - 引用类型：null
  - 当不希望数组的初值为系统给定的默认值时，可以用赋值语句对数组进行初始化。
  - 初始化可以在数组声明时进行，也可以声明以后，在构造数组的时候赋值。

```
类型[] 数组名 = {初值表};
```

```
数组名[下标] = 初值;
```

    - 其中初值表是用逗号隔开的一组值。

数组初始化

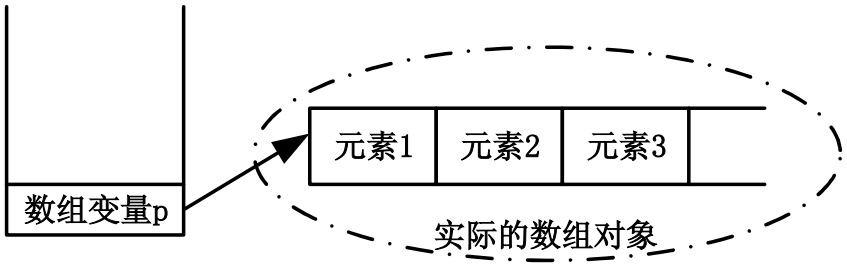
```
byte[ ] myArrayC = {1,2,3,4,5};
myArrayB[0]=0.1;
```

- 数组元素的使用
  - 当声明了一个数组并分配了内存空间后，就可以在程序中任何可以使用变量的地方使用数组元素，其格式为：

数组名[下标]

  - 其中下标为非负的整型常数或表达式，其数据类型为int、short、byte，但不可以是long。
  - Java会对数组的下标进行越界检查以保证安全性。下标的范围是从0到数组的长度减一。如果下标不合法（<0或>length-1），则运行时出现异常：

- 数组的内存机制



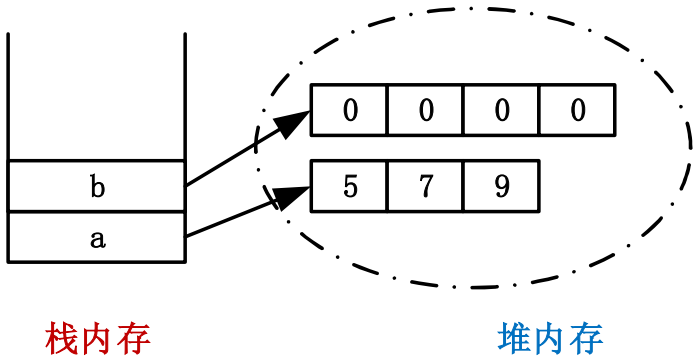
方法执行时定义的变量存放在栈内存中，方法结束后该内存栈自然销毁

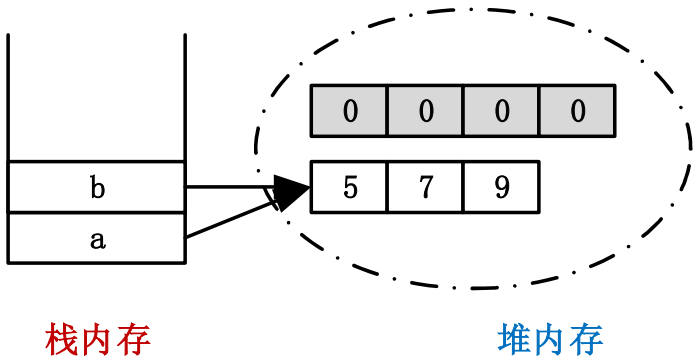
创建对象时该对象被保存到运行时数据区即堆内存中以便反复利用（因为对象创建成本较大）。堆内存中的对象不会随方法结束而销毁，因为方法结束后可能该对象还可能被另一个引用变量所引用。只有当该对象没有任何引用变量引用它时，才会被系统垃圾回收器回收。

数组引用范例

```
public class ArrayLen{
    public static void main(String[] args) {
        int[] a={5,7,9};
        int[] b=new int[4];
        System.out.println("b的长度是："+b.length);
        b=a;
        System.out.println("b的长度是："+b.length);
    }
}
```

```
b的长度是：4
b的长度是：3
```





- 多维数组
  - 严格的说Java并不支持多维数组，多维数组的声明是通过对一维数组的嵌套形式声明来实现的。Java中的数组变量是一个引用，指向真实的数组内存，如果某个数组中其元素也是引用，并指向真实的数组内存，这种情形看上去就像多维数组一样。
  - 二维数组的声明

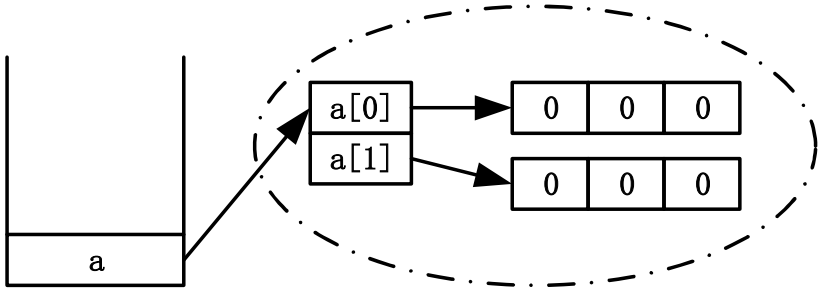
```
类型[][] 数组名;
```

```
数组名 = new 类型[行数][列数];
```

- 分配二维数组空间
  - 直接为每一维分配空间

分配数组空间

```
int[ ][ ] a = new int[2][3];
```

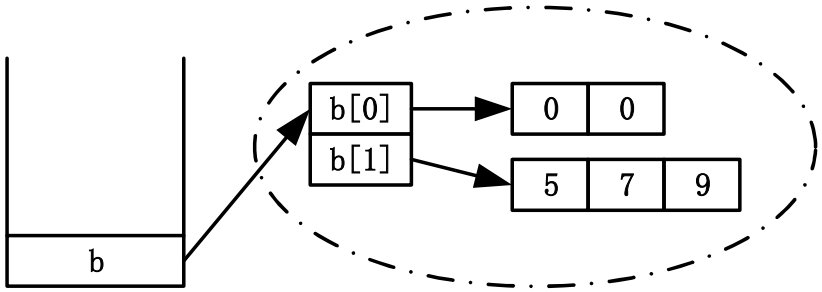


栈内存　　堆内存

  - 从最高维开始，分别为每一维分配空间

分配数组空间

```
int[ ][ ] b=new int[2][ ];
b[0]=new int[2];
b[1]=new int[ ]{5,7,9};
```



栈内存　　堆内存

- 多维数组的创建和使用
  - 声明 `类型[][]...[] 数组名;`
  - 创建 `类型[][]...[] 数组名 = new 类型[长度1][长度2]...[长度n];`
  - 使用 `数组名[下标1][下标2]...[下标n]`
  - 可以通过属性length获得该数组的元素个数，方法如下：
    - 数组名.length求出多维数组中第一维的长度；
    - 数组名[0].length求出多维数组中第二维的长度；
    - 依此类推求出其余维的长度。
  - 在使用new来分配内存时，对于多维数组至少要给出最高维的大小。

- 一个简单的复制数组的方法
  - 通过System类提供的arraycopy方法，格式如下

```
public static void arraycopy(Object src,int srcpos,Object dest,int destpos,int length)
```

  - 其中src和dest分别表示被复制的数组和目的数组。srcpos、destpos和length分别表示源数组中的开始位置、目的数组中的开始位置和要复制的数组元素的数量。

数组复制例程

```
public class ArrayCopyDemo {
        public static void main(String[] args) {
                char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
                char[] copyTo = new char[7];
                System.arraycopy(copyFrom, 2, copyTo, 0, 7);
                System.out.println(new String(copyTo));
        }
}
```

2.7 字符串

String

- 不可变类，一旦一个String对象被创建以后，包含在这个对象中的字符序列是不可改变的，直至这个对象被销毁

StringBuffer

- StringBuffer对象代表一个字符序列可变的字符串

StringBuilder

- 与StringBuffer基本相似，但没有实现StringBuffer具有的线程安全功能，所以性能略高

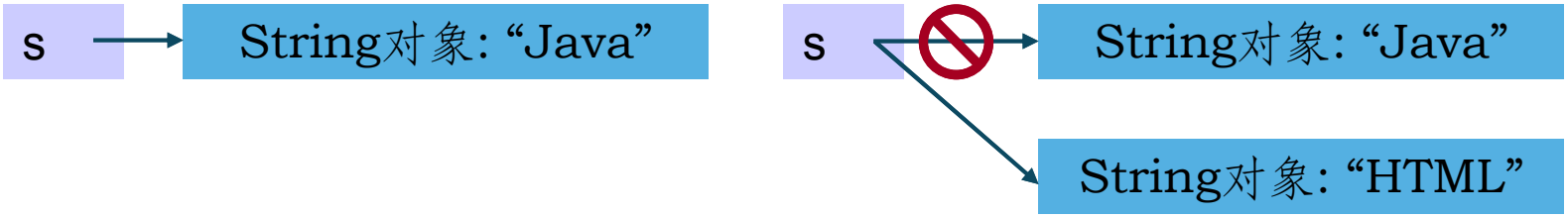
- String字符串的声明和创建

```
String 字符串对象名 = "字符串内容";          //直接赋值
String 字符串对象名 = new String(字符串内容);   //创建对象
```

- String对象是不可变的，它的内容不能改变。

```
String s="Java";
s="HTML";
```

- 上述代码不能改变字符串的内容。
  - 第一条语句创建了一个内容为“Java”的字符串对象，并将其引用赋值给s；
  - 第二条语句创建了一个内容为“HTML”的新String对象，并将其引用赋给s。
  - 赋值后第一个String对象仍然存在，但是不能再访问它，因为变量s指向了新的对象。



- 字符串是不可变的，又被频繁的使用。当字符串字面量具有相同的字符序列时，通过使用唯一的实例，可提高JVM的效率并节约内存。
- 在使用直接赋值的方式时，Java会确保一个字符串内容只有一个拷贝。

```
String s1="java";
String s2="java";
String s3="ja"+"va";
System.out.println(s1==s2);
System.out.println(s1==s3);
```

- 此时打印结果：

true

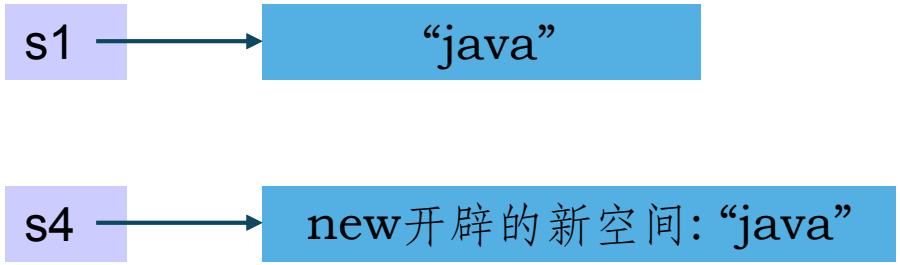
true

- 用new String()创建的字符串不是字面量，是使用new操作符额外新开辟的内存空间；

```
String s1="java";
String s4=new String("java");
System.out.println(s1==s4);
```

- 打印结果为：

  false



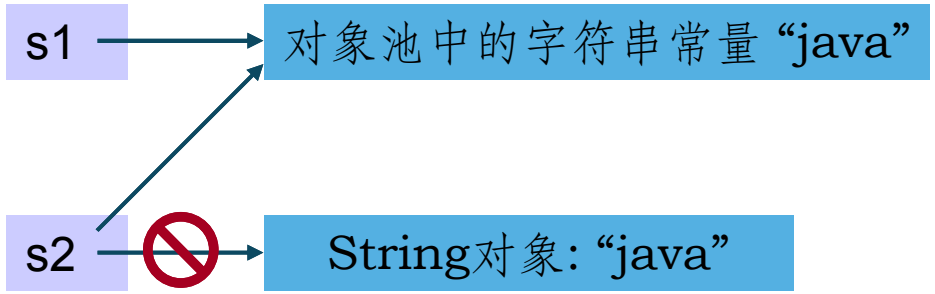
- 在Java中，当一个String对象调用intern方法时，Java查找对象池中是否有相同Unicode的字符串常量，如有，则返回对它的引用；否则在对象池中增加一个该Unicode的字符串常量，并返回对它的引用。

```
String s1="java";
String s2=new String("java“);
System.out.println(s1==s2);
s2=s2.intern();
System.out.println(s1==s2);
```

- 打印结果为

  false

  true



- 在上例中，运算符“==”只能检测字符串是否指向同一个对象，而不能判断是否具有相同的内容即字符序列。
- 字符串的比较可使用equals()方法对对象的内容进行相等比较，例如：在上例中，s1.equals(s4)的值为true。

Java中的String不属于基本数据类型，所有出现的“……”字符串的值，实际上是作为String类的匿名对象形式存在的。

```
String s1=“hello”;
System.out.println(“hello”.equals(str));
```

执行结果为：true

内容比较的方法equals由“hello”这个字符串直接调用，说明它是对象，该匿名对象由系统自动生成

- 字符串也可以使用campareTo方法进行比较，当两个字符串相等时返回值为0，当不相等时，返回两个字符串第一个不相等字符间的差值。

```
String s="abc";
String s1="abg";
int a=s.compareTo(s1);
```

  - 此时a的值应为-4，因为s与s1第一个不相等的字符分别为c和g，c比g小4，因此返回-4。

- String类提供的字符串操作方法：

方法		功能
构造方法	public String()	用于创建一个空的String字符串常量
	public String(String value)	创建一个已经存在的内容为value的String字符串常量
	public String(char value[])	创建一个已经存在的内容为value[ ]数组的String字符串常量
	public String(StringBuffer buffer)	用一个已经存在的StringBuffer对象来创建一个字符串常量
成员方法	int length()	返回字符串的长度
	char charAt(int index)	返回字符串位置index处的字符
	int compareTo(String anotherStr)	当前字符串与anotherStr比较，相等为0，大于为正，小于为负
	boolean equals(Object o)	比较两个字符串对象是否相等
	boolean equalsIgnoreCase(String o)	忽略大小写的比较两个字符串对象是否相等
	String substring(int beginIn,int endIn)	在当前字符串中求从起始位置到结束位置的子串
	int indexOf(int ch)	从头向后查找字符ch首次出现的位置，未找到返回-1

方法		功能
成员方法	int indexOf(String str)	在当前字符串中寻找str首次出现的位置，未找到返回-1
	String toLowerCase()	将字符串换成小写
	String toUpperCase()	将字符串换成大写
	String trim()	去掉原字符串开头和结尾的空格，并返回新的字符串
	String replace(char oldCh,char newCh)	用newCh替换字符串中所有的oldCh
	int parseInt(String str)	将数字格式字符串转换为int类型
	int parselong(String str)	将数字格式字符串转换为long类型
	String valueOf(int i)	将int数据类型转换为字符串
	String concat(String str)	将str连接到原字符串后面
	boolean startsWith(String Prefix)	从当前字符串的起始位置寻找字符串Prefix
	boolean endsWith(String suffix)	看当前字符串是否以字符串suffix为结尾

- StringBuffer字符串的创建

```
StringBuffer 字符串对象名 = new StringBuffer();
```

```
StringBuffer 字符串对象名 = new StringBuffer(字符串内容);
```

  - StringBuffer类比String类更灵活，可以在字符串缓冲区中添加、插入或追加新的内容。
  - 字符串的长度一定小于或等于字符串缓冲区的容量，如果有更多的字符加入字符串缓冲区，超出其容量，则缓冲区的容量会自动增加。
  - 在计算机内部，字符串缓冲区是一个字符数组，如果超过缓冲区的容量，就用新的数组取代现有数组，新数组的长度为2*(原数组长度+1)。
- 如果一个字符串不需要改变，则使用String类而不要使用StringBuffer类。

- 字符串对象的使用
  - 字符串对象声明以后，就可以为其赋值和使用。对于String对象，可以直接赋值或通过构造方法；对于StringBuffer对象，则可以通过该类提供的append、insert等方法修改其值。

字符串变量赋值引用

```
String s=new String();
StringBuffer ss=new StringBuffer();
s=”Computer”;
ss=ss.append(“Computer”);
```

- 字符串连接操作
  - 字符串的简单连接利用“+”来完成。
  - 其他类型的数据与字符串进行“+”运算时，将自动换成成字符串。
    - 以下两段语句在表面效果上是等价的：

字符串连接操作

```
age=18;
String s=”He is”+age+”years old”;
```

字符串连接操作

```
age=”18";
String s=new StringBuffer(“He is”).append(age).append(”years old”).toString();
```

String

```
public class StringTest{
        public static void main(String args[]){
                String s=new String();
                for(int i=0;i<15;i++)
                        s=s+"String";
        }
}
```

StringBuffer

```
public class StringTest{
        public static void main(String args[]){
                StringBuffer s=new StringBuffer();
                for(int i=0;i<15;i++)
                        s=s.append("String");
        }
}
```

- 第一个例子将会产生15个String对象，依次为“String”、“String String”、“String StringString”……
- 第二个例子则始终只在一个StringBuffer对象上修改。

- StringBuffer类提供的字符串操作方法

方法		功能
构造方法	public StringBuffer()	创建可容纳16个字符的StringBuffer对象，大于16个字符自动增长
	public StringBuffer(int size)	创建可容纳size个字符的StringBuffer对象，大于size个字符自动增长
	public StringBuffer(String str)	按str创建一个动态可变的StringBuffer对象
成员方法	StringBuffer append(Object o)	将obj转换为字符串并连接到在当前字符串末尾
	StringBuffer insert(int offset, Object o)	将obj转换为字符串并插入到当前字符串offset位置
	int capacity()	返回当前StringBuffer类对象分配的字符空间的数量
	void setCharAt(int index,char ch)	将当前对象中的index位置的字符替换为ch
	delete(int beginIn,int endIn)	删除从beginIn开始到endIn结束之间的字符
	String toString()	把StringBuffer字符串转换为String