

# 第二章

## Java语言基础

毕向东

## 2 Java语言基础组成

---

- 2.1 关键字
- 2.2 标识符
- 2.3 注释
- 2.4 常量和变量
- 2.5 运算符
- 2.6 语句
- 2.7 函数
- 2.8 数组

## 2.1 关键字

### 关键字的定义和特点

定义：被Java语言赋予了特殊含义的单词

特点：关键字中所有字母都为小写

### 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				

## 2.1 关键字

用于定义访问权限修饰符的关键字				
private	protected	public		
用于定义类，函数，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

## 2.2 标识符

---

- 在程序中自定义的一些名称。
- 由26个英文字母大小写，数字：0-9 符号：\_ \$ 组成
- 定义合法标识符规则：
  - 1，数字不可以开头。
  - 2，不可以使用关键字。
- Java中严格区分大小写。
- 注意：在起名字的时，为了提高阅读性，要尽量有意义。

## 2.2 标识符

---

Java中的名称规范:

- 包名: 多单词组成时所有字母都小写。
  - xxxyyyzzz
- 类名接口名: 多单词组成时, 所有单词的首字母大写。
  - XxxYyyZzz
- 变量名和函数名: 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。
  - xxxYyyZzz
- 常量名: 所有字母都大写。多单词时每个单词用下划线连接。
  - XXX\_YYY\_ZZZ

## 2.3 注释

---

- 用于注解说明解释程序的文字就是注释。
- 提高了代码的阅读性。
- Java中的注释格式：
  - 单行注释
    - 格式：//注释文字
  - 多行注释
    - 格式：/\* 注释文字 \*/
  - 文档注释
    - 格式：/\*\* 注释文字 \*/

## 2.3 注释

---

- 对于单行和多行注释，被注释的文字，不会被JVM（**java**虚拟机）解释执行。
- 对于文档注释，是**java**特有的注释，其中注释内容可以被JDK提供的工具 **javadoc** 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。
- 注释是一个程序员必须要具有的良好编程习惯。
- 初学者编写程序可以养成习惯：先写注释再写代码。
- 将自己的思想通过注释先整理出来，在用代码去体现。
- 因为代码仅仅是思想的一种体现形式而已。



## 2.3 注释

---

```
/**  
这是我的Hello World程序。  
@author 小强  
*/  
class Demo  
{  
    /*  
    这是主函数，是程序的入口  
    它的出现可以保证程序的独立运行，  
    */  
    public static void main(String[] args)  
    {  
        //这是输出语句用于将括号内的数据打印到控制台。  
        System.out.println("Hello World");  
    }  
}
```

## 2.4 常量与变量

---

- 常量表示不能改变的数值。
- Java中常量的分类:
  - 1, 整数常量。所有整数
  - 2, 小数常量。所有小数
  - 3, 布尔型常量。较为特有, 只有两个数值。**true false**。
  - 4, 字符常量。将一个数字字母或者符号用单引号('')标识。
  - 5, 字符串常量。将一个或者多个字符用双引号标识。
  - 6, null常量。只有一个数值就是:null.
- 对于整数: java有三种表现形式。
  - 十进制: 0-9 , 满10进1.
  - 八进制: 0-7 , 满8进1. 用0开头表示。
  - 十六进制: 0-9, A-F, 满16进1. 用0x开头表示。

## 2.4 常量与变量

---

- 进制的基本转换
  - 十进制 二进制 互转
    - 十进制转成二进制 除以2取余数
    - 二进制转成十进制 乘以2的幂数
  - 十进制 八进制 互转
  - 十进制 十六进制 互转
  - 负数的二进制表现形式
    - 对应的正数二进制取反加1

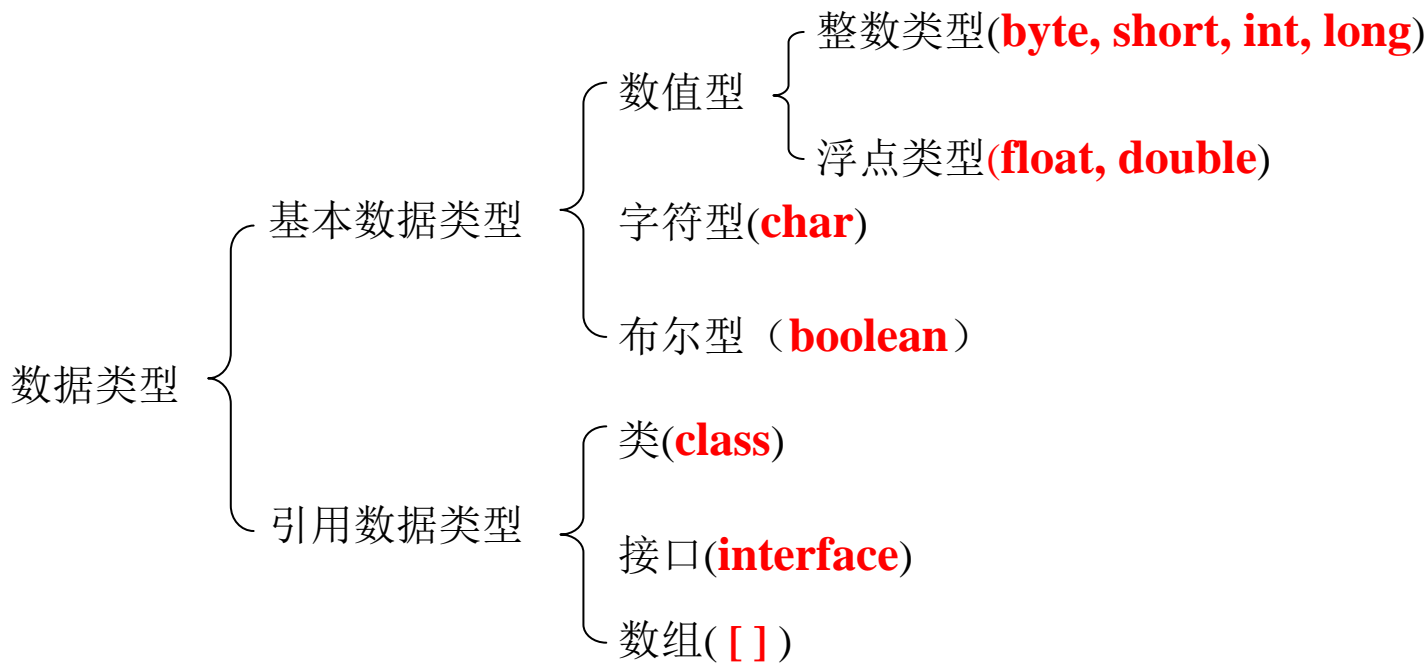
## 2.4 常量与变量

---

- 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）
  - 该区域的数据可以在同一类型范围内不断变化
- 为什么要定义变量：
  - 用来不断的存放同一类型的常量，并可以重复使用
- 使用变量注意：
  - 变量的作用范围（一对{}之间有效）
  - 初始化值
- 定义变量的格式：
  - 数据类型 变量名 = 初始化值；
  - 注：格式是固定的，记住格式，以不变应万变。
- 理解：变量就如同数学中的未知数。

## 2.4 常量与变量

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，在内存总分配了不同大小的内存空间



整数默认: **int**    小数默认: **double**

## 2.4 常量与变量

---

- 自动类型转换（也叫隐式类型转换）
- 强制类型转换（也叫显式类型转换）
- 类型转换的原理
- 什么时候要用强制类型转换？
- 表达式的数据类型自动提升
  - 所有的byte型、short型和char的值将被提升到int型。
  - 如果一个操作数是long型，计算结果就是long型；
  - 如果一个操作数是float型，计算结果就是float型；
  - 如果一个操作数是double型，计算结果就是double型。
- 分析
  - **System.out.println('a')**与**System.out.println('a'+1)** 的区别。

## 2.4 常量与变量

---

- 自动类型提升

```
byte b = 3;
```

```
int x = 4;
```

```
x = x + b; // b会自动提升为int类型进行运算。
```

- 强制类型转换

```
byte b = 3;
```

```
b = b + 4; // 报错
```

```
b = (byte)b + 4; // 强制类型转换，强制将b+4的结果转换为byte类型，再赋值给b。
```

- 思考：

```
byte b1=3,b2=4,b;
```

```
b=b1+b2;
```

```
b=3+4;
```

哪句是编译失败的呢？为什么呢？

## 2.5 运算符

---

- 算术运算符
- 赋值运算符
- 比较运算符
- 逻辑运算符
- 位运算符
- 三元运算符



## 2.5.1 算术运算符

算术运算符

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4;-b;	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模	5%5	0
++	自增 (前)	a=2;b=++a;	a=3;b=3
++	自增 (后)	a=2;b=a++;	a=3;b=2
--	自减 (前)	a=2;b=--a	a=1;b=1
--	自减 (后)	a=2;b=a--	a=1;b=2
+	字符串相加	"He"+"llo"	"Hello"

## 2.5.1 算术运算符

---

- 算术运算符的注意事项

- 如果对负数取模，可以把模数负号忽略不记，如： $5\%-2=1$ 。但被模数是负数就另当别论。
- 对于除号“/”，它的整数除和小数除是有区别的：整数之间做除法时，只保留整数部分而舍弃小数部分。
  - 例如：`int x=3510;x=x/1000*1000;` x的结果是？
- “+”除字符串相加功能外，还能把非字符串转换成字符串，
  - 例如：`System.out.println("5+5="+5+5);`打印结果是？

## 2.5.2 赋值运算符

---

- 符号:

= , +=, -=, \*=, /=, %=

- 示例:

```
int a,b,c; a=b=c =3;
```

```
int a = 3; a+=5;等同运算a=a+5;
```

- 思考:

```
short s = 3;
```

```
s=s+2;
```

```
s+=2;
```

有什么区别?

## 2.5.3 比较运算符

比较运算符

运算符	运算	范例	结果
==	相等	4==3	false
!=	不等	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	false
instanceof	检查是否是类的对象	"Hello" instanceof String	true

- 注1：比较运算符的结果都是boolean型，也就是要么是true，要么是false。
- 注2：比较运算符“==”不能误写成“=”。

## 2.5.4 逻辑运算符

逻辑运算符

运算符	运算	范例	结果
&	AND (与)	false&true	false
	OR (或)	false true	true
^	XOR (异或)	true^false	true
!	Not (非)	!true	false
&&	AND (短路)	false&&true	false
	OR (短路)	false  true	true



## 2.5.4 逻辑运算符

---

- 逻辑运算符用于连接布尔型表达式，在Java中不可以写成 $3 < x < 6$ ，应该写成 $x > 3 \ \& \ x < 6$ 。
- “&”和“&&”的区别：
  - 单&时，左边无论真假，右边都进行运算；
  - 双&时，如果左边为真，右边参与运算，如果左边为假，那么右边不参与运算。

“|”和“||”的区别同理，双或时，左边为真，右边不参与运算。
- 异或(^)与或(|)的不同之处是：当左右都为true时，结果为false。

## 2.5.5 位运算符

位运算符		
运算符	运算	范例
<<	左移 	$3 \ll 2 = 12 \rightarrow 3 * 2 * 2 = 12$
>>	右移 	$3 \gg 1 = 1 \rightarrow 3 / 2 = 1$
>>>	无符号右移	$3 \ggg 1 = 1 \rightarrow 3 / 2 = 1$
&	与运算	$6 \& 3 = 2$
	或运算	$6   3 = 7$
^	异或运算	$6 \wedge 3 = 5$ 
~	反码	$\sim 6 = -7$

位运算是直接对二进制进行运算。

## 2.5.5 位运算符

### 位运算符的细节

<<	空位补0，被移除的高位丢弃，空缺位补0。
>>	被移位的二进制最高位是0，右移后，空缺位补0； 最高位是1，空缺位补1。
>>>	被移位二进制最高位无论是0或者是1，空缺位都用0补。
&	二进制位进行&运算，只有1&1时结果是1，否则是0；
	二进制位进行 运算，只有0 0时结果是0，否则是1；
^	任何相同二进制位进行^运算，结果是0； $1^1=0$ ， $0^0=0$ 不相同二进制位^运算结果是1。 $1^0=1$ ， $0^1=1$





## 2.5.5 位运算符

### ● 练习:

- 1.最有效率的方式算出2乘以8等于几?  $2 \ll 3$ ;
- 2.对两个整数变量的值进行互换(不需要第三方变量)

答案: 位运算

```
class Test{  
    public static void main(String[] args){  
        int a=9;  
        int b=2;  
        System.out.println("a= "+a+" b= "+b);  
        a = a ^ b;  
        b = a ^ b;  
        a = a ^ b;  
        System.out.println("a= "+a+" b= "+b);  
    }  
}
```

## 2.5.6 三元运算符

---

- 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:

- 获取两个数中大数。
- `int x=3,y=4,z;`
- `z = (x>y)?x:y;` z变量存储的就是两个数的大数。

## 2.6 程序流程控制

---

- 判断结构
- 选择结构
- 循环结构

## 2.6.1 判断结构

### if 语句

#### 三种格式:

1. if (条件表达式)

{

    执行语句;

}

2. if (条件表达式)

{

    执行语句;

}

else

{

    执行语句;

}

3. if (条件表达式)

{

    执行语句;

}

else if (条件表达式)

{

    执行语句;

}

.....

else

{

    执行语句;

}

## 2.6.1 判断结构

---

if语句特点:

a,每一种格式都是单条语句。

b,第二种格式与三元运算符的区别: 三元运算符运算完要有值出现。好处是: 可以写在其他表达式中。

c,条件表达式无论写成什么样子, 只看最终的结构是否是true 或者 false;

## 2.6.2 选择结构

---

### switch语句

格式:

```
switch(表达式)
{
    case 取值1:
        执行语句;
        break;
    case 取值2:
        执行语句;
        break;
    ....
    default:
        执行语句;
        break;
}
```

## 2.6.2 选择结构

---

### switch语句特点:

a,switch语句选择的类型只有四种: byte, short, int, char。

b,case之间与default没有顺序。先执行第一个case, 没有匹配的case执行default。

c,结束switch语句的两种情况: 遇到break, 执行到switch语句结束。

d,如果匹配的case或者default没有对应的break, 那么程序会继续向下执行, 运行可以执行的语句, 直到遇到break或者switch结尾结束。

## 2.6.3 循环结构

---

代表语句: while , do while , for

while语句格式:

先循环再执行

```
while(条件表达式)
{
    执行语句;
}
```

do while语句格式:

先执行再循环

```
do
{
    执行语句;
}while(条件表达式);
```

do while特点是条件无论是否满足，循环体至少被执行一次。



## 2.6.3 循环结构

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)
{
    执行语句;
}
```

注:

a,for里面的连个表达式运行的顺序, 初始化表达式只读一次, 判断循环条件, 为真就执行循环体, 然后再执行循环后的操作表达式, 接着继续判断循环条件, 重复找个过程, 直到条件不满足为止。

函数体内部的局部变量

b,while与for可以互换, 区别在于for为了循环而定义的变量在for循环结束就是在内存中释放。而while循环使用的变量在循环结束后还可以继续使用。

函数体内部的全局变量

c,最简单无限循环格式: while(true), for(;;),无限循环存在的原因是并不知道循环多少次, 而是根据某些条件, 来控制循环。

## 2.6.4 其他流程控制语句

---

**break(跳出), continue(继续)**

break语句：应用范围：选择结构和循环结构。

continue语句：应用于循环结构。

注：

a,这两个语句离开应用范围，存在是没有意义的。

b,这个两个语句单独存在下面都不可以有语句，因为执行不到。

c,continue语句是结束本次循环继续下次循环。

d,标号的出现，可以让这两个语句作用于指定的范围。

## 语句练习

---

- 语句的嵌套应用
- 累加 求和，计数器
- 循环嵌套