# 1 关键字

## 1.1 关键字的概述

Java 的关键字对 java 的编译器有特殊的意义，他们用来表示一种**数据类型**，或者表示**程序的结构**等，关键字不能用作变量名、方法名、类名、包名。

## 1.2 常见的关键字

| 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 | | |
| 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 |

**备注：**不必死记硬背，如果使用关键字作为标识符，编译器能提示错误。
goto 是 java 的保留关键字，意思是 java 并没有使用 goto，以后是否使用未定。

# 2 标识符

## 2.1 什么是标识符

就是程序员在定义 java 程序时，自定义的一些名字，例如 helloworld 程序里关键字 class 后跟的 Demo，就是我们定义的类名。类名就属于标识符的一种。
标识符除了应用在类名上，还可以用在变量、函数名、包名上。(要求同学们先记住，以后会详细见到这些)。

## 2.2 标识符必须遵循以下规则

1. 标识符由 26 个英文字符大小写（a~zA~Z)、数字(0~9)、下划线(_)和美元符号($)组成。
2. 不能以数字开头，不能是关键字
3. 严格区分大小写
4. 标识符的可以为任意长度

## 2.3 标识符案例

### 2.3.1 合法的标识符

ComputeArea，radius，area $itcast _itcast gz_itcast
注意：由于 Java 严格区分大小写，ITCAST 和 itcast 是完全不同的标识符

### 2.3.2 非法标识符

1. class (关键字)
2. 100java(不能以数字开头)
3. Hello java (空格不是组成标识符的元素)

### 2.3.3 Java 中的标识符命名规范

**1．包名**
多个单词组成时所有字母小写（例:package com.itcast）
**2．类名和接口**
多个单词组成时所有单词的首字母大写(例：HelloWorld)
**3．变量名和函数名**
多个单词组成时第一个单词首字母小写,其他单词首字母大写(例:lastAccessTime、getTime)。

**4. 常量名**

多个单词组成时，字母全部大写，多个单词之间使用_分隔（例：INTEGER_CACHE）

**注意**：只是为了增加规范性、可读性而做的一种约定，标识符在定义的时候最好见名知意，提高代码阅读性。

# 3 3.注释

## 3.1 注释的作用

通过注释提高程序的可读性，是 java 程序的条理更加清晰，易于区分代码行与注释行。另外通常在程序开头加入作者，时间，版本，要实现的功能等内容注释，方便后来的维护以及程序员的交流。

## 3.2 注释的种类

1.单行注释（line comment）用//表示，编译器看到//会忽略该行//后的所文本

2.多行注释(block comment)用/**/表示，编译器看到/*时会搜索接下来的*/,忽略掉/**/之间的文本。

3. 文档注释用/** */表示，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。

```
public static void main(String[] args) {
            // 第一步： 获取半径？并将半径保存在程序中
            double radius = 5;
            // 第二步：计算面积，并将面积保存在程序中
            /*
            double area = radius * radius * 3.1415;
            // 第三步：在控制台现实面积
        System.out.println("半径为" + radius + "的圆的面积为：" + area);
            */


        }
```

注意：多行注释中可以嵌套单行注释，多行注释不能嵌套多行注释。错误!!!

```
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
         /*
            注意：多行注释嵌套多行注释是不行的。
        */
```

```
        */
    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 java");
    }
```

文档注释　（编写软件说明书）

1. 需要使用 sum 给我们提供的 javadoc 工具生成一个 html 的说明文档。
2. 只能抽取 public 的属性或者方法内容。

格式：

Javadoc –d 指定存储文档的路径　-version –author（可选）　目标文件

```
/**
 需求：在控制台上打印hellowrod
  @author ztl
  @version 1.0
 */
public class Demo1 {
    /**
        该方法是程序的主入口，保证该类可以独立运行
      @param args  运行该类的时候需要使用的参数
     */
    public static void main(String[] args) {
        int n = 10;  //
        System.out.println("可见:"+n);
    }
}
```

@author 作者

@version 版本

@param 方法的参数

@return 返回值

注释的使用细节：

三种注释可以出现在程序的任何地方，但是不推荐找任意位置。

1. 编程习惯：
    1. 给那条语句进行说明，注释应该写在该语句的旁边。
    2. 单行注释一般写在语句的后面多行注释和文档注释一般写在语句的上面

注意：文档注释只能出现在类、属性、方法的上面。

```
class Demo2
{
    /*
        多行注释应该写在方法类或者属性的上方.
    */
    /**
        注意：文档注释只能出现在方法与类的上方，否则提取不到.
    */

    public static void main(String[] args)
    {
        System.out.println("Hello !");  // 这个语句是输出hello
        System.out.println("World !");  //这个语句是输出helloWord
    }
}
```

2　注释的嵌套

1.单行注释可以在单行注释里面。

2.多行注释不能嵌套在多行注释里面。

```
class Demo2
{
    /*
        多行注释应该写在方法类或者属性的上方.

    /*
        多行注释的嵌套.是不允许的.
    */
    */
    /**
        注意：文档注释只能出现在方法与类的上方，否则提取不到.
    */
    public static void main(String[] args)
    {
        System.out.println("Hello !");  // 这个语句是输出hello //单行注释是允许的，但是不推荐这样子使用.
        System.out.println("World !");  //这个语句是输出helloWord
    }
}
```

3　注释的调试作用:

1. 可以作为初学者的调试方式。
2. 可以帮组初学者确定代码的错误之处。

```
class Demo3
{
    /*
        这是一个出现问题的程序，但是不清楚那行出现的问题.
        可以使用注释对其进行程序错误的调试.
    */
    public static void main(String[] args)
    {
        /*
        System.out.println("Hello World1");
        System.out.prin("Hello World2");
        System.out.println("Hello World3");
        System.out.println("Hello World4");
        System.out.println("Hello World5");
        */
    }
}
```

# 4 常量

## 4.1 常量的概述

常量是指在程序运行过程中其值不能改变的量。

# 4.2 常量类型

Java 中常量的分类：
整数常量 ： 所有整数
小数常量 ： 所有小数
布尔常量 ： 只有 true 和 false
字符常量 ：使用’’引起来的单个字符
字符串常量 ：使用“”引起来的字符序列，“” 、“a” 、” ”
null 常量 ： 只有一个值 null

### 3.char 类型

char 类型表示的是单个字符类型，任何数据使用单引号括起来的都是表示字符。字符只能有一个字符，比如：普通的老百姓穿上军装就是军人。

**注意：**特殊字符的转义序列：转义字符

转义字符的概述：

特殊字符使用”\”把其转化成字符的本身输出，那么使用”\”的字符称作为转移字符。

需求：使用输出语句，打印出带引号的信息例如输出。

System.out.println("teacher said"java is fun"");编译是无法正常通过的。语法有错误，编译器读到第二个引号就认为是字符串的结束，剩余的不知道怎么处理。如何解决这个问题：java 中使用转义字符来表示特殊的字符。一个转义字符以反斜杠（\）开始。

**问题：**想要打印带引号的字符串怎么办，就可以使用反斜杠（\）后跟字符，这个反斜杠就是转义字符。

| 转义字符 | 名称 | Unicode |
|---|---|---|
| \b | Backspace （退格键） | \u0008 |
| \t | Tab （Tab键盘） | \u0009 |
| \n | Linefeed （换行） | \u000A |
| \r | Carriage Return（回车） | \u000D |
| \\ | Backslash （反斜杠） | \u005C |
| \' | Single Quote （单引号） | \u0027 |
| \" | Double Quote （双引号） | \u0022 |

\r 表示接受键盘输入，相当于按下回车。
\n 表示换行。
\t 制表符，相当于 Table 键
\b 退格键，相当于 Back Space
\’ 单引号
\’’ 双引号
\\ 表示一个斜跨

上述问问题解决：System.*out*.println("teacher said\"java is fun\"");
注意：换行符就是另起一行，回车符就是回到一行的开头，所以我们平时编写文件的回车符应该确切来说叫做回车换行符

### 4．boolean 类型

boolean 由数学家 Geogore Boole 发明
boolean 类型用来存储布尔值，在 java 中布尔值只有 2 个，true 和 false。

```
boolean flag=true;
flag=false;
```

Java 中这 8 中基本数据类型都是小写的。

# 5 进制的转换

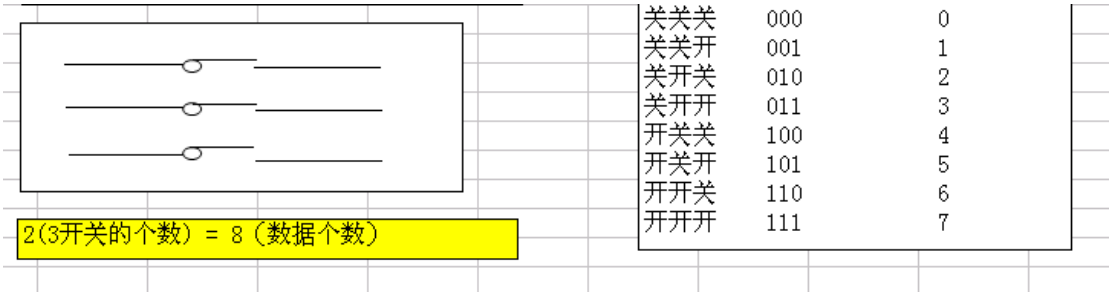
进制：进制是一种记数方式 ，可以用有限的数字符号代表所有的数值。由特定的数值组成。

## 5.1 整型的表现形式

1. 十进制： 都是以0-9这九个数字组成，不能以0开头。
2. 二进制： 由0和1两个数字组成。
3. 八进制： 由0-7数字组成，为了区分与其他进制的数字区别，开头都是以0开始。
4. 十六进制：由0-9和 A-F 组成。为了区分于其他数字的区别，开头都是以 ox 开始。

## 5.2 进制的由来

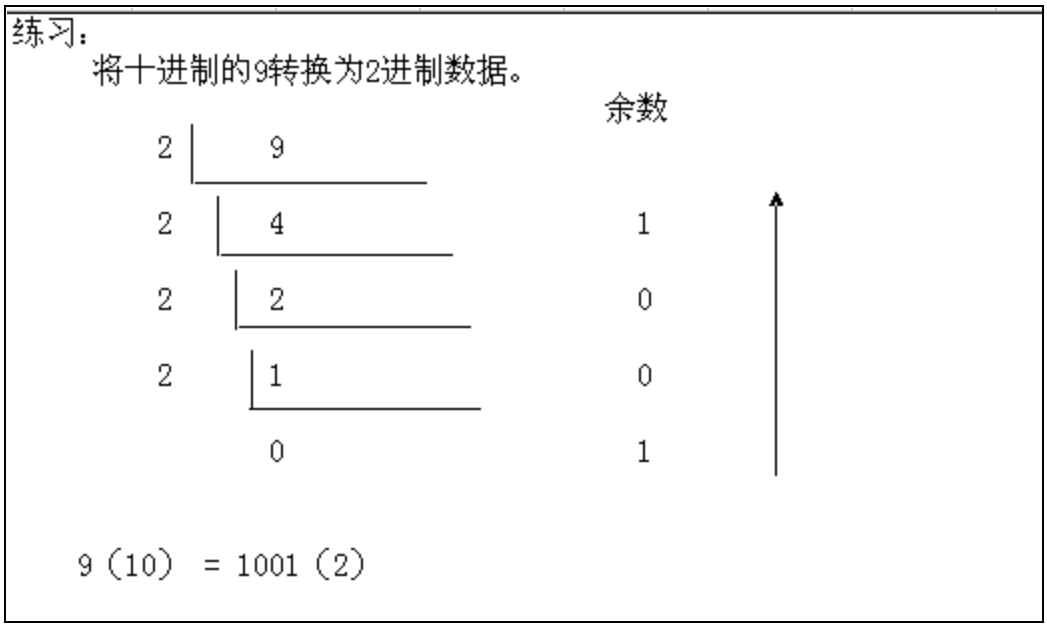
几乎每个民族最早都使用都十进制计数法，这是因为人类计数时自然而然地首先使用的是十个手指。 但是这不等于说只有十进制计数法一种计数方法。例如，世界各国在计算年月日时不约而同地使用“十二进制”12 个月为一年又如：我国过去 16 两才算为一斤，这就是“十六进计数法”,一个星期七天，这个就是”七进制计算法”。 计算机是由逻辑电路组成，逻辑电路通常只有两个状态，开关的接通与断开，这两种状态正好可以用“1”和“0”表示。



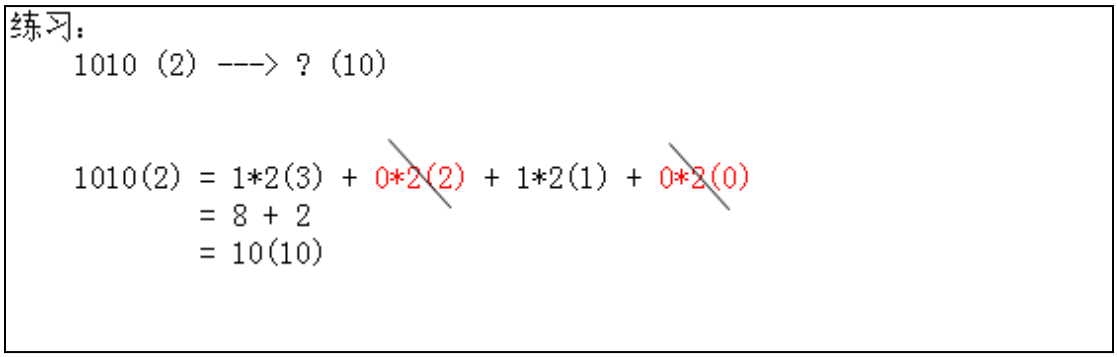

如果要在计算机里面保存十进制的 7.

### 5.2.1 十进制与二进制之间的转换

十进制转二进制的转换原理：除以 2，反向取余数，直到商为 0 终止。




二进制转十进制的转换原理:就是用二进制的每一个乘以 2 的 n 次方，n 从 0 开始，每次递增 1。然后得出来的每个数相加




存在问题：书写特别长，不方便记忆。

```
class Demo14
{
    public static void main(String[] args)
    {
        System.out.println(Integer.toBinaryString(12345321));  //101111000101111111101001
    }
}
```

## 5.2.2 十进制与八进制之间转换

**1. 八进制的由来**

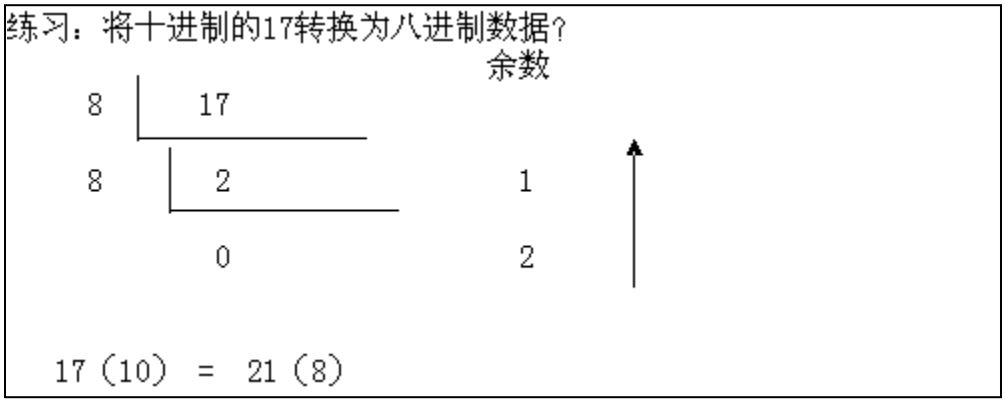
二进位制在计算机内部使用是再自然不过的。但在人机交流上，二进位制有致命的弱点——数字的书写特别冗长。例如，十进位制的 100000 写成二进位制成为 11000011010100000。为了解决这个问题，在计算机的理论和应用中还使用两种辅助的进位制——八进位制和十六进位制。二进位制的三个数位正好记为八进位制的一个数位，这样，数字长度就只有二进位制的三分之一，与十进位制记的数长度相差不多。例如，十进位制的

100000 写成八进位制就是 303240。十六进位制的一个数位可以代表二进位制的四个数位，这样，一个字节正好是十六进位制的两个数位。十六进位制要求使用十六个不同的符号，除了 0—9 十个符号外，常用 A、B、C、D、E、F 六个符号分别代表（十进位制的）10、11、12、13、14、15。这样，十进位制的 100000 写成十六进位制就是 186A0。

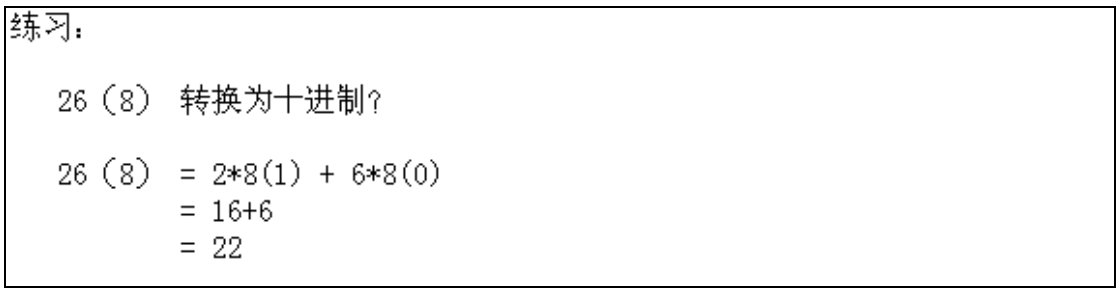
**2. 八进制的特点**

由数字 0-7 组成。即使用三个开关表示一个八进制数。

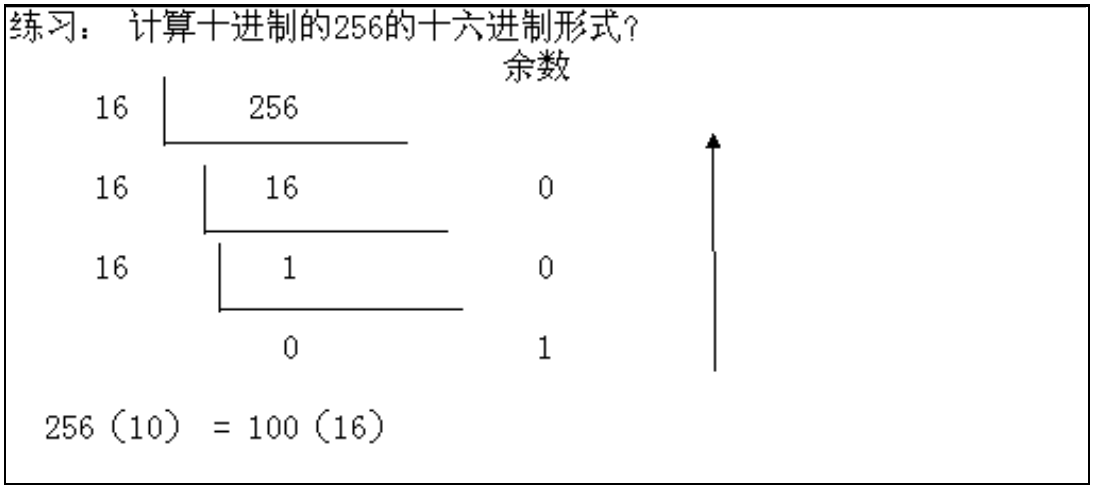
10 进制转换 8 进制原理：就是用十进制的数字不断除于 8，取余数。




八进制转十进制原理： 用把进制的数不断乘以 8 的 n 次方，n 从 0 开始，每次递增 1。




除了这种方法之外，我们还有另一种方法，因为三个开关表示一个八进制数。



练习：

1100(2) = ？(8)

思路一：2进制---》10进制---》8进制

1100(2) = 12(10) = 14(8)= 1*8(1)+4*8(0) = 8+4 = 12(10)

思路二：

1100(2) 有4个二进制数据　4个开关

001,100(2) = 14 (8)

练习：将八进制的45转换为二进制？

45(8) = 4,5 (8)
= 100, 101
= 100101(2)

十进制与十六进制之间的转换

十六进制特点：由 0~9 a(10) b(11) c(12) d(13) e(14) f(15)组成。

十进制转十六进制原理：就是不断除以 16，取余数。

练习：将十六进制的100转换为十进制？

100(16) = 1*16(2)
= 256(10)

练习：110000转换为十六进制？

```
110000(2) = 0011,0000
          =    3 ,0
          = 30(16)

30(16) = 3*16(1)+0
       = 48

111001111(2) = 0001,1100,1111
             = 1,c,f
             = 1cf(16)
```

代码体现：

人使用的十进制 、 计算机底层处理的数据是二进制、八进制、十六进制，

那么如果给计算机输入不同的进制数据呢？

```
System.out.println( 42 );      // 默认是十进制
System.out.println( 042 );     // 将一个数值前面加0代表这个数据是八进制  4*8(1)+2*8(0)=32+2 = 34
System.out.println( 0x42 );    // 将一个数值前面加0x代表一个十六进制数据

进制使用细节:

System.out.println( 0429 );    // 当0在前面指定的是一个八进制数据的时候，后面的数值如果超过7编译报错：过大的整数
System.out.println( o42 );     // 八进制的标志是零(0)不是o
```

# 6 变量

## 6.1 变量的概述

1. **变量的概述**
   用于存储可变数据的容器。
2. **变量存在的意义**
   计算机主要用于处理生活中的数据，由于生活中存在大量的可变数据，那么计算机就必须具备存储可变数据的能力。
   比如：
   1.时间每一秒都在发生变化，根据不同的时间要有不同的处理方式。
   2.气象站会根据温度发布天气预报信号。
3. **变量的特点**

正常情况下牛奶盒装的都是牛奶，而且会根据牛奶的多少去决定要多大的容量的牛奶盒，A 和 B 两位同学同时把牛奶盒放进篮子里面，但是需要区分两个牛奶盒是谁的，都需要在牛奶盒上做一个标志。

特点：

1. 必须要有大小
2. 存储一定格式的可变数据
3. 必须要有名字

## 6.2 变量的声明

根据上述变量的特点，所以我们声明一个变量的时候需要确定变量的大小，类型、名字三个特点：

错误： 1024byte temp = 1000000；

错误原因，java 有自己的变量类型。

### 6.2.1 变量的数据类型

**1. 整型**

| | | | | | |
|---|---|---|---|---|---|
| byte | 代表一个字节的大小 | 8bit | 2(8) | -128~127 | 256 |
| short | 代表两个字节的大小 | 16bit | 2(16) | -2(15)~2(15)-1 | |
| int | 代表四个字节的大小 | 32bit | 2(32) | -2(31)~2(31)-1 | |
| long | 代表八个字节的大小 | 64bit | 2(64) | -2(63)~2(63)-1 | |

**如果一个数值没有采取特殊的处理，那么该整数默认的类型是 int。**

**可以使用数值后面添加 L 或小写 L 改变默认的整数类型。**

**2. 浮点型**

float 代表四个字节的大小 32bit

double 代表八个字节的大小 64bit

java 程序中所有的小数默认的类型是 double 类型，所以需要使用特殊的符号改变默认的小数类型。

**3. 字符型**

char 代表两个字节的大小 16bit 2(16)

原理：将字符映射为码表中对应的十进制数据加以存储。

**4. 布尔型**

boolean 占一个字节。只有 true 与 false 两个值。

### 6.2.2 变量的声明

| 格式: 数据类型 变量名字 1，变量名字 2,……变量名字 n,; |
|---|

案例：

int i 声明了一个整形的变量。

double d 声明了一个 double 数据类型的变量

float f 声明了一个 float 数据类型的变量。

**备注：**变量名的首字母都一般都是以小写字母开始。

## 6.2.3 变量的初始化

## 6.2.4 变量的初始化方式

初始化方式 1： 数据类型 变量名字 = 数值。
初始化方式 2： 数据类型 变量名字 , 变量名字 = 数值。
案例：
方式 1： double d = 3.14;
方式 2： double d ; d = 3.14 ;

# 7 java 数据类型的转换

Java 中可以进行不同数据类型的加减乘除运算吗？是可以的。在算术运算符中已经体验过如果两个整数（int）相除会去掉小数部分。如果需要保留小数部分，可以让除数或者被除数变为 double 类型的（5 变为 5.0）。其实 Java 是自动的将 int 的那个数变为了 double 类型了也就是 Java 自动的将整数变为了浮点数。例如 5/2.0 其实是 5.0/2.0

**1、 自动类型转换（也叫隐式类型转换）**

可以将一个数赋值给更大数值范围的变量，例如可以经 byte 变量赋值给 short 变量可以将 short 变量赋值给 int 变量可以将 int 变量赋值给 long 变量。

Java 内部其实做了工作就是自动将数值进行了类型提升，就叫做自动类型转换（也叫隐式类型转换）

```
byte b = 1; //00000001
short s = b;  //00000000 00000001
int i = s;
long lon = i;
double d = lon; //1.0
```

自动类型转换（也叫隐式类型转换）

要实现自动类型的转换，需要满足两个条件，第一两种类型彼此兼容，第二目标类型取值范围必须大于源类型。所有的数字类型，包括整形和浮点型彼此都可以进行转换。

例如：

```
byte b=100;
int x=b;
System.out.println(x);//程序把 b 结果自动转换为 int 类型。
```

**2、 强制类型转换（也叫显式类型转换）**

不可以将一个数值赋给范围更小数值范围的变量，除非进行类型转换。

```
byte b = 100;
b = b + 2;
System.out.println(b);
```

上述例子发生了什么，发生了类型转换。

b+2 遇到了加法运算，2 默认是 int 类型，byte 类型 b 变量存储的值自动类型提升为了 int 类型。执行完加法运算后的结果就是 int 类型，想要将 int 的类型值放入到 byte 类型变量 b 中，无法放入，编译报错。

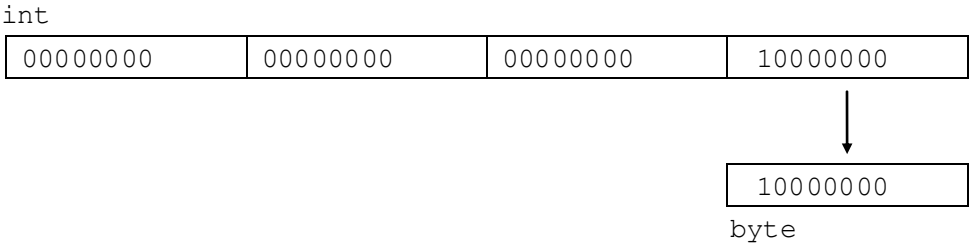
```
byte b=1;
b=(byte)(b+2);
```

当两种类型彼此不兼容，或者目标类型取值范围小于源类型（目标是 byte 源是 int）无法自动转换，此时就需要进行强制类型转换。

**强制类型转换需要注意：**

损失精度!!!

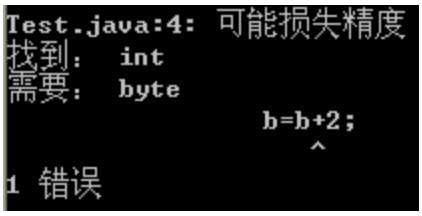
```
int a=128;
byte b=(byte)a;
System.out.println(b);//-128
/*
 * 此时的强转已经造成了数值的不准确
*/
```




再次分析此行代码

```
byte b = 100;
b = b + 2;
System.out.println(b);
```

编译：提示如下错误。




**3、类型转换的原理**

可以把 byte 理解为 1 两的碗，short 2 两的碗，int 4 两的碗，long 8 两的碗。1 两碗的满碗酒可以倒入 2 两 4 两 8 两的碗中。但是 4 两碗的酒倒入 1 两碗的酒就有一

些问题。

**1、什么时候要用强制类型转换**

比如小数部分只想保留整数部分.

一定要清楚要转换的数据在转换后数据的范围内否则会损失精度.

```
public static void main(String[] args) {
        byte b = 100;
        b = (byte) (b + 2);
        System.out.println(b); // 102
        //舍弃小数部分
        double d=5.5;
        int num=(int)d;
}
```

**2、表达式的数据类型自动提升**

算术表达式，逻辑表达式

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'+1)结果?

自动类型提升

```
        byte b = 3;
        int x = 4;
        x = x + b;// b会自动提升为int 类型参与运算。
        System.out.println(x);// 7
```

强制类型转换

```
        byte b = 2;
        /*
         * 强制类型转换，强制将b+2强制转换为byte类型，再赋值给b
         */
        b = (byte) (b + 2);
        System.out.println(b);// 4
```

思考 1

byte b=126;

问：既然数据默认的有数据类型，那么 126 默认是 int 类型的，为什么存储到 byte 类型时不会报错呢。

126 是常量 java 在编译时期会检查该常量（每个常量）是否超出 byte 类型的范围。如果没有可以赋值。

思考 2：byte b=128;能否正常的编译和运行。

该语句会出现编译错误，128 超出了 byte 变量的存储范围，所以出现编译错误。

思考 2

byte b1=3，b2=4，b;

b=b1+b2;

b=3+4;

哪一句编译失败？为什么？

b =3+4， 3 和 4 都是常量，所以 java 在编译时期会检查该常量（每个常量）是否超出 byte 类型的范围。如果没有可以赋值。例如 b=128+1 就无法编译通过。b=127+1; 也是无法通过。

b =b1+b2 不可以，因为 b1 和 b2 是变量，表达式求值时，变量值会自动提升为 int 型，表达式结果也就成了 int 型，这是要赋值给 byte 型的 b，必须进行强制类型转换了。

**6、System.out.println('a'+1)结果**

美国人为了让计算机识别他们生活中的文字,让二进制表示生活中的文字.所以一个字母代表了一个二进制.,二进制也有十进制的表现形式.,把生活中的字母都用数字来标识,例如 97 代表 a ，98 代表 b。打印'a'就把 a 作为输出显示,没有疑问。但是 'a'+1 有加号涉及到了运算。根据 java 自动类型提升规则，同样道理 char 提升为 int 。就把'a'代表的数字体现了出来。a 表示的是 97 97+1 就是 98; 那么 想要查看 98 表示的 char 是什么 怎么实现呢 ?就要用到刚才介绍的强制类型转换了 System.out.println(char('a'+1));就取到了 98 在 ASCII 码表中表示的字符。大写 A 和小写 a 在 ASCII 有不同的表现。还有一个概念字符'1' 在 ASCII 中 不是数字 1,可以运行代码查看,到此就可以明白了 char 类型,char 类型也是可以参与运算的,为什么可以参与运算呢。因为字符在 ASCII 表中都有对应的数字体现。所有的计算机兼容 ASCII。

```
System.out.println('a'+1); //98
System.out.println((char)('a'+1));  //b
```

补充问题：

```
int i='a'+'b';
System.out.println(i);//结果?
System.out.println("hello"+'j'); //结果?
```

总结：

所有数值运算符都可以用在 char 型数据上，如果另一个操作数是一个数字或者字符，那么 char 会自动提升为 int 型，如果另一个操作数是字符串，那么字符就会和字符串相连。

# 8 java 运算符

## 8.1.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" |

**正负号(+,-)**

```
byte age = +30;                 // + 充当整数的符号 代表正数  -代表负数
System.out.println( age );

int a = 23;
int b = 34;
int c = a + b;                  // 分别取出两个变量名的值    求和    将和赋值给c变量
System.out.println( c );
System.out.println( 23-34+(-12) );
```

**除法**

```
int a = 1234;
int b = 10;
int c = a / b;                  // 分别取出两个变量名的值    求商  取整
System.out.println( c );
System.out.println( 1234/10 );  // 两个变量做元算，结果的类型取决于两个参与运算中最大那个数据类型
System.out.println( 1234.0/10 );


System.out.println( 1234/10*10 ); // 1230

int a = 12,b=0;
System.out.println( a/b );      // 除数不能为0

报异常:
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Demo6.main(Demo6.java:42)
```

**%取模**

求余数

```
public static void main(String[] args) {
    int i  = 10 ;
    int j = 3;
    System.out.println(i%j);// 1
}
```

取模的正负取决与被除数：

```
public static void main(String[] args) {
    System.out.println(+10%-3);  //1
    System.out.println(+10%+3);  //1
    System.out.println(-10%3);   //-1
    System.out.println(-10%-3);  //-1
}
```

**1. 自增**

（++）前自增：先自增完毕，再运算整个表达式，语句分号前面的都是运算表达式；

```
public static void main(String[] args) {
    int i = 0 ;
    int sun = 0;
    sun = ++i; //前自增
    System.out.println("sum:" + sun); //sun=1
}
```

后自增，先运算完整个表达式（分号前面的都是表达式），再进行自增；

```
public static void main(String[] args) {
    int i = 0 ;
    int sun = 0;
    sun = i++; //后自增
    System.out.println("sum:" + sun+" i:"+ i); //sun=1  i = 0
}
```

备注：参与自增运算的操作数据每次会加 1.

结论：

如果运算符在变量的前面，则该变量自增 1 或者自减 1，然后返回的是变量的新值，如果运算符在变量的后面，则变量也会自增或者自减 1，但是返回的是变量原来的值。++在前就是先运算，再取值，++在后就是先取值，再运算。

自增自减运算符案例：

完成如下运算：

一；

```
int i = 10;
int newNum = 10 * i++;
System.out.println(newNum);//?
```

二：

```
int i = 10;
int newNum = 10 * ++i;//?
System.out.println(newNum); //?
```

一可以理解为

```
int i = 10;
int newNum = 10 * i;
i = i + 1;
```

二可以理解为

```
int i = 10;
i = i + 1;
```

```
int newNum = 10 * i;
```

**练习：**

1. 使用程序判断一个整数是偶数还是奇数
2. 使用程序判断假设今天是星期 4，那么问 10 天后的今天是星期几？
3. 将数值表达式使用 java 程序翻译，并通过程序求出运算结果

其中 int x=1;int y=2,int a=3,int b=4,int c=5;

$$\frac{3+4x}{5}-\frac{10(y-5)(a+b+c)}{x}+9\left(\frac{4}{x}+\frac{9+x}{y}\right)$$

案例一：

```
public static void main(String[] args) {
        // 判断一个整数一奇数还是偶数
        int x = -100;
        // 奇数是，1,3,5...偶数是2,4,6...显然整数除2能整除，也就%（取模）结果为0就是偶数。      int result = x % 2;
        System.out.println(result);
        // 使用判断语句进行判断。
        if (result == 0) {
            System.out.println(x + "是偶数");
        } else {
            System.out.println(x + "是奇数");
        }
    }
```

方案二 使用判断该数结果是否是奇数。

（但是该算法有问题，如果被判断的整数为负数是否有效？）

```
public static void main(String[] args) {
        // 判断一个整数一奇数还是偶数
        int x = 1;
        // 奇数是，1,3,5...偶数是2,4,6...显然奇数%的结果为1.
        int result = x % 2;
        System.out.println(result);
        // 使用判断语句进行判断。
        if (result == 1) {
            System.out.println(x + "是奇数");
        } else {
            System.out.println(x + "是偶数");
        }
    }
```

改进

```
public static void main(String[] args) {
        // 判断一个整数一奇数还是偶数
        int x = -1;
```

```
        // 奇数是，1,3,5...偶数是2,4,6...显然奇数%的结果为1.
        int result = x % 2;
        System.out.println(result);
        // 使用判断语句进行判断。
        if (result != 0) {
            System.out.println(x + "是奇数");
        } else {
            System.out.println(x + "是偶数");
        }
    }
```

案例三：判断星期

```
public static void main(String[] args) {
        // 设定今天是星期1，用int 1表示星期一，0表示星期天
        int today = 1;
        // 十天后是星期几？，一个星期是7天，7天之后又是星期1，可以用?
        int future = (today+10) % 7;
        if (future == 0) {
            System.out.println("10天后是星期天");
        } else {
            System.out.println("10天后是星期: " + future);
        }

    }
```

案例 4：

```
int x = 1;
        int y = 2;
        int a = 3;
        int b = 4;
        int c = 5;
        int result = (3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9* (4 / x + (9 + x) / y);
        System.out.println(result); // 442
```

# 8.1 赋值运算符

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

| 运算符 | 运算 | 范例 | 结果 |
|---|---|---|---|
| = | 赋值 | a=3,b=2 | a=3,b=2 |
| += | 加等于 | a=3,b=3;a+=b; | a=5,b=2; |
| -= | 减等于 | a=3,b=2,a-=b; | a=1,b=2; |
| *= | 乘等于 | a=3,b=2,a*=b; | a=6,b=2 |
| /= | 除等于 | a=3,b=2,a/=b; | a=1,b=2; |

| %= | 模等于 | a=3,b=2,a%=b; | a=1,b=2 |
|---|---|---|---|

a+=b 可以想象成 a=a+b;

变量声明完了之后，可以使用赋值语句（assignment statement）给变量赋一个值，Java 中使用等号（=）作为基本的赋值运算符（assignment operator），

格式如下：

```
variable = expression;
变量     =  表达式;
```

变量我们已经知道如何声明，表达式具体如何定义？

表达式的定义：

表达式涉及到值（常量），变量和通过运算符计算出的值，以及他们组合在一起计算出的新值。

x =y+1;

例如：

```
public static void main(String[] args) {
        int x = 1; // 声明int变量x，  赋值1给变量x
        int y = 0; // 声明int变量y，  赋值0给变量y
        double area; // 声明double变量area
        double radius = 1.0; // 声明double变量radius，并赋值1.0给变量radius
        x = 5 * (3 / 2) + 3 * 2; // 将=右半部分表达式的计算结果赋值给变量x
        x = y + 1; // 将变量y和1的求和的值赋值给变量x
        area = radius * radius * 3.14159; // 将计算面积的值赋值给变量area
    }
```

赋值运算符小问题

问题 1：

```
int x;
Syst2intln(x = 1);
```

如何理解？

答：等价于

```
x=1;
System.out.println(x);
```

注意：不能 1=x,变量名必须在赋值运算符的左边。

问题二：

```
        int x;
        int y;
        int z;
        x = y = z = 100;
```

如何理解？

答：等价于

```
int x;
int y;
int z;
z = 100;
y = z;
x = y;
```

问题三：short s1 = 1;

s1= s1+1;

s1+=1;

问：s1= s1+1; s1+=1; 与有什么不同？

对于 short s1 = 1; s1 = s1 + 1; 由于 s1+1 运算时会自动提升表达式的类型，所以结果是 int 型，再赋值给 short 类型 s1 时，编译器将报告需要强制转换类型的错误。

对于 short s1 = 1; s1 += 1;由于 += 是 java 语言规定的运算符，java 编译器会对它进行特殊处理，因此可以正确编译。

## 8.2 比较运算符

如何比较两个值？使用比较运算符 3 和 5 谁大，在 java 中如何比较？

比较运算符比较的两边操作数，结果都是 boolean 的，只有 true 和 false 两种结果。

| 运算符 | 运算 | 例子 | 结果 |
|---|---|---|---|
| == | 相等于 | 4= =3 | false |
| != | 不等于 | 4!= 3 | true |
| < | 小于 | 4 < 3 | flase |
| > | 大于 | 4>3 | true |
| <= | 小于等于 | 4<=3 | false |
| >= | 大于等于 | 4>=3 | true |
| Instanceof | 检查是否是类的对象 | "hello"instanceof String | true |

注意的细节：

1.使用比较运算符的时候，要求两种数据类型必须一致。

byte、short、char 会自动提升至 int。

```
int age1 = 23;
int age2 = 34;
byte age3 = 34;
long age4 = 34L;
boolean rel =  age1 < age2 ;
boolean tag = true;
System.out.println( rel );
//System.out.println( age2 > tag );
System.out.println( age2 == age3 );
System.out.println( age3 == age4 );
```

# 8.3 逻辑运算符

什么是逻辑运算符？连接比较运算符的符号称之为逻辑运算符。那么为什么要连接比较运算符？ 举例：当你去公司应聘，招聘要求，男性（判断为真），并且开发经验 1 年（判断为假）那么，我们还适合去面试吗，不能，因为只满足了一项，总体是不满足的（总体结果为假）。

逻辑运算符用于对 boolean 型结果的表达式进行运算，运算的结果都是 boolean 型。我们的比较运算符只能进行一次判断，对于对此判断无能为力，那么逻辑运算符就可以经将较运算符连接起来。

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 |

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：单与时，左边无论真假，右边都进行运算；双与时，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。

& 与 | 或 ^ 异或 ！ 非

**1、& 与**

```
true & true = true ;
false & true= false;
true & false = false;
false & false= false;
```

总结 & 符号特点

& ：只要两边的 boolean 表达式结果,有一个 false.那么结果就是 false

只有两边都为 true ,将结果为 true.

**2、| 或**

```
true  | true  =true;
ture  | false =true;
false | true  =true;
false | false =flase;
```

总结 | ：两边只要有一个为真结果就为真,当两边同为假时结果才为假.

**3、^ 异或**

```
true  ^  true =false;
  ture  ^  false =true;
  false  ^  true= true;
  false  ^  false=flase;
```

^ ：两边相同结果是 false
两边不同结果是 true;

**4、！非**

```
! true = false
! false= true
```

**5、&& 短路**

研究发现，&运算只有两边全为真的时候，结果才为真，那么当左边为假的时候就没有必要在进行判断，&&就产生了。

```
int a =4;
a >3 && a< 6;
a >3 & a< 6 ;
```

在这种情况下世没有区别的
如果：

```
a =2
a >3 & a< 6
```

2 大于 3 为假，接着运算 2 小于 6 为真,总的结果为假

```
a >3 && a< 6;
```

此时 a 不大于 3 结果为 false 右边不运算了.即短路.所以&& 比& 效率稍微高了一点.

```
public static void main(String[] args) {
        int x = 0;
        int y = 1;
        if (x == 0 && y == 1) {
            System.out.println(x + y);
        }
    }
```

# 8.4 位运算符

按位操作符用来操作整数基本数据类型中的单个比特（bit），就是二进制，按位操作符会对两个参数中对应的位（bit）执行布尔运算，最终生成一个结果。按位操作符来源于 C 语言面向底层的操作，Java 设计的初衷是嵌入式电视机机顶盒，所以面向底层的操作也保留了下来。

任何信息在计算机中都是以二进制的形式保存的，”&”、“|”、“^”除了可以作为逻辑运算符也可以作为位运算符。位运算是直接对二进制进行运算。他们对两个操作数中的每一个二进制位都进行运算。例如 int 是由 32 个二进制数组成，因此使用位运算符可以对整数

值的二进制数进行运算。

位（bit）运算符：

| 位运算符 | 运算符含义 |
|---|---|
| **&** | 与（AND） |
| **\|** | 或（OR） |
| **^** | 异或 |
| **~** | 取反 |

规则：

可以把 1 当做 true 0 当做 false

只有参与运算的两位都为 1，&运算的结果才为 1，否则就为 0。

只有参加运算的两位都是 0，| 运算的结果才是 0，否则都是 1。

只有参加运算的两位不同，^ 运算的结果才为 1，否则就为 0。

**1、& 与运算**

& 参见运算的两位数都为 1，&运算符结果才为 1，否则就为 0。

6&3

| | | | | |
|---|---|---|---|---|
| 00000000 | 00000000 | 00000000 | 00000110 | 6 |
| 00000000 | 00000000 | 00000000 | 00000011 | 3 |
| 00000000 | 00000000 | 00000000 | 00000010 | & =2 |

**2、| 或运算**

| 参与运算的两位都为 0，|运算的结果才为 0，否则就为 1。

| | | | | |
|---|---|---|---|---|
| 00000000 | 00000000 | 00000000 | 00000110 | 6 |
| 00000000 | 00000000 | 00000000 | 00000011 | 3 |
| 00000000 | 00000000 | 00000000 | 00000111 | \| =7 |
| | | | | |

**3、^ 异或运算**

^只有参加运算的两位不同，^运算的结果才为 1，否则就为 0。

| | | | | |
|---|---|---|---|---|
| 00000000 | 00000000 | 00000000 | 00000110 | 6 |
| 00000000 | 00000000 | 00000000 | 00000011 | 3 |
| 00000000 | 00000000 | 00000000 | 00000101 | ^ =5 |

**1、~ 反码**

就是取反，二进制只有 1 和 0，取反就是如果为 1，取反就是 0，如果是 0，取反就是 1。

| | | | | |
|---|---|---|---|---|
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0110 | 6 |
| 1111-1111 | 1111-1111 | 1111-1111 | 1111-1001 | 取反 -7 |

```
System.out.println(~6);//-7
```

结论：当参与取反的数值是正数时，把对应的值加上负号，再-1；

当参与取反的数值是负数时，把对应的值加上负号，再-1；

负数的表现形式就是对应的正数取反，再加 1。负数的最高位肯定是 1。

**4、负数表示**

负数对应的正数的二进制-1,然后取反。

-6

| | | | | |
|---|---|---|---|---|
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0110 | 6 |
| 1111-1111 | 1111-1111 | 1111-1111 | 1111-1001 | 取反 |
| 1111-1111 | 1111-1111 | 1111-1111 | 1111-1010 | 加 1 |

**5、异或特点**

一个数异或同一个数两次,结果还是那个数． 用处一个简单的加密思想．

6^3^3

| | | | | |
|---|---|---|---|---|
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0110 | 6 |
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0011 | ^3 |
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0101 | |
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0011 | ^3 |
| 0000-0000 | 0000-0000 | 0000-0000 | 0000-0110 | 结果是 6 |

除了这些位运算操作，还可以对数据按二进制位进行移位操作，Java 的移位运算符有三种。

练习：取出一个二进制的某一段。

```
 00000000-00000000-00000000-00001110   14
& 00000000-00000000-00000000-00001111   15
-----------------------------------------
 00000000-00000000-00000000-00001110   14
```

使用异或（^）数据对数据加密

```
6 ^ 3 =
 110
^011
-----
 101
^011
-----
 110    6^3^3 = 6;

异或符号的特点：一个数异或另一个
数两次，结果还是这个数。
```

对两个变量的值进行互换。

方式 1：

```
int x = 3,y = 6;

int z;
z = x;
x = y;
y = z;
```

对两个变量进行值交换（不能使用第三个变量）

方式 2：

```
x = x + y;// x = 3 + 6;  x = 9;

y = x - y;// y = 9 - 6;  y = 3;

x = x - y;// x = 9 - 3;  x = 6;
```

两个数相加的时候，值有可能超出 int 表示范围,不推荐。

方式 3：

```
x = x ^ y;// x = 3 ^ 6;
y = x ^ y;// y = (3 ^ 6) ^ 6; y = 3;
x = x ^ y;// x = (3 ^ 6) ^ 3; x = 6;
```

该方式虽然效率高，而且避免了超出 int 值，但是可读性较差。

三种方式都可以对两个变量的值进行交换，但是推荐使用第一种。(面试除外)

## 8.5 移位操作符

<< 左移

>> 右移

>>> 无符号右移

| 位运算符 | | |
|---|---|---|
| 运算符 | 运算 | 范例 |
| << | 左移 | 3 << 2 = 12 --> 3*2*2=12 |
| >> | 右移 | 3 >> 1 = 1 --> 3/2=1 |
| >>> | 无符号右移 | 3 >>> 1 = 1 --> 3/2=1 |
| & | 与运算 | 6 & 3 = 2 |
| \| | 或运算 | 6 \| 3 = 7 |
| ^ | 异或运算 | 6 ^ 3 = 5 |
| ~ | 反码 | ~6 = -7 |

| 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；<br>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<br><br>不相同二进制位 ^ 运算结果是 1。1^0=1 ， 0^1=1 |

技巧：可以理解为二进制 1 就是 true，0 就是 false。

案例：

**1、左移 （算术移位）**

3<< 2 是如何在计算机里是实现的?

首先将 3 转换为 2 进制，

| | | | | |
|---|---|---|---|---|
| 00000000 | 00000000 | 00000000 | 00000011 | 3 的二进制 |
| 00000000 | 00000000 | 00000000 | 000011 | 左移 2 位,砍掉高位 |
| 0000 0000 | 0000 0000 | 0000 0000 | 0000 1100 | 低位补 0 |

结果是 12，所以 3<<2 =12;

结论：左移就相当于乘以 2 的位移个数次幂.

**2、右移**

6>>2

| | | | | |
|---|---|---|---|---|
| 00000000 | 00000000 | 00000000 | 00000110 | 6 的二进制 |
| 000000 | 00000000 | 00000000 | 00000001 | 右移 10 被砍掉 |
| 00000000 | 00000000 | 00000000 | 00000001 | 高位补 0 |

结果是 1，所以 6>>2 =1;

结论一个数往左移越移越大,往右边移越来越小.

**推论**

3<<2=12; 3<<1=6 ; 3<<3=24;

3*4=12 ; 3*2=6; 3*8=24;

$3*2^2=12$; $3*2^1=6$ $3*2^3=24$;

结论往左移几位就是乘以 2 的几次幂。

右移规律

6>>2=1 ;6>>1=3 ;

6/4=1 ; 6/2=3 ;

右移两位就是除以 2 的 2 次方,右移一位就是除以 2 的一次方。

总结 ： >> 是除以 2 的移动位数次幂

<< 是乘以 2 的移动位数次幂

**用处:最快的运算是位运算。**

**练习：最有效率的方式算出 2 乘以 8 等于几?**

**3、无符号右移 （逻辑移位）**

通过演示发现右移时高位就空了出来， >> 右移时高位补什么要按照原有 数据的最高位来决定。

1111-1111 1111-1111 1111-1111 1111-1010 -6>>2

1111-1111 1111-1111 1111-1111 1111-0010

最高位补什么要看原有最高位是什么

那么使用>> 后原来是最高位 1 的那么空出来的最高位还是 1 的，是 0 的还是 0。

如果使用>>> 无论最高位是 0 还是 1 空余最高位都拿 0 补,这就是无符号右移。

1111-1111 1111-1111 1111-1111 1111-1010 -6>>>2

001111-1111 1111-1111 1111-1111 1111-10

结果是；1073741822

## 8.6 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：

1 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。

```
int x = 1;
int y = 2;
int z;
z = x > y ? x : y;
System.out.println(z); //2
```

2 判断一个数是奇数还是偶数。

```
int x=5;
System.out.println((x%2==0?"偶数":"奇数"));
```

# 9 运算符的优先级与结合性

运算符优先级表

| 优先级 | 运算符 | 结合性 |
| --- | --- | --- |
| 1 | () [] . | 从左到右 |
| 2 | ! +(正) -(负) ~ ++ -- | 从右向左 |
| 3 | * / % | 从左向右 |
| 4 | +(加) -(减) | 从左向右 |
| 5 | << >> >>> | 从左向右 |
| 6 | < <= > >= instanceof | 从左向右 |
| 7 | == != | 从左向右 |
| 8 | &(按位与) | 从左向右 |
| 9 | ^ | 从左向右 |
| 10 | \| | 从左向右 |
| 11 | && | 从左向右 |
| 12 | \|\| | 从左向右 |
| 13 | ?: | 从右向左 |
| 14 | = += -= *= /= %= &= \|= ^= ~= <<= >>= >>>= | 从右向左 |

```
int a = 2;
int b = 3;
/*
             1 + a + 4 + a*b/(4-2)
 1. 优先级
             1 + a + 4 + a*b/2
             1 + a + 4 + 6/2
             1 + a + 4 + 3
 2. 结合性

             (1 + a) + 4 + 3
             (3+4) + 3
             7 + 3
             10
    结合性在同级优先级的操作符之间使用



System.out.println( 1 + a + 4 + a*b/(4-2) );
```

```
a += b+= c += 5+5;

c += 5+5;  c = c+(5+5) = 4+5+5 = 14
b += c;    b = b+c = 3 + 14 = 17
a += b;    a = a+b = 2 + 17 = 19



int c = 4;
a += b+= c += 5+5;
System.out.println( a );
System.out.println( b );
System.out.println( c );
```

# 10 作业

1. 按照标准步骤完成 hello world 打印。
2. 如果定义一个变量，如何使用变量。
3. 基本数据类型有哪些？
4. 'a'+1，结果是什么？为什么？都做了什么事情呢？
5. ++在前，在后的区别？
6. short s = 3; s = s+2; s+=2,有什么区别，为什么？
7. &和&&的区别?