## 2.3 基本数据类型

在Java中有8种基本数据类型来存储数值、字符和布尔值，如图2.1所示。

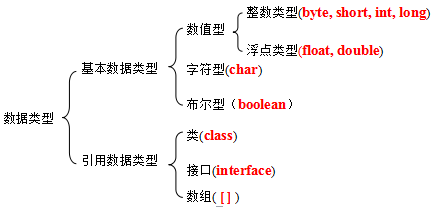


图2.1 Java数据类型

### 2.3.1 整数类型

整数类型用来存储整数数值，即没有小数部分的数值。可以是正数、负数，也可以是0。根据所占内存的大小不同，可以分为byte、short、int和long 4种类型。他们所占的内存和取值范围如表2.2所示。整数默认的类型为int型。

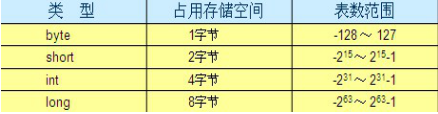


图2.2 整数类型(1个字节是8位)

**1.byte型**

使用byte关键字来定义byte型变量,可以一次定义多个变量并对其进行赋值,也可以不进行赋值。byte型是整型中分配内存空间最少的，只分配1个字节，取值范围也是最小的，只在-128~127之间，在使用时一定要注意，以免数据溢出产生错误。

【例2.1】定义byte型变量,并对其进行简单赋值操作,验证其几点要素;

public class BytePractice{

public static void main(String[] args){

byte a = 45 , b = 127 , c ;

System.out.println(a+b);//整数类型的默认类型为int,所以再给byte类型进行运算时会先将其转换为int类型,然后进行运算

//System.out.println(c); 我们声明了变量却未给其赋值,这里会报未初始化变量

/\* b = 128 ; byte类型的内存空间为1字节，取值范围为-128-127，所以这里会报损失精度的问题

System.out.println(b);\*/

}

}

**2.Short型**

Short型是短整型，使用short关键字来定义short型变量，可以一次定义多个变量并对其赋值，也可以不进行赋值。系统给short型在内存中分配2个字节，取值范围也比byte大得多，在-32768~32767之间，虽然取值范围变大，但仍要注意溢出。

【例2.2】定义short型变量

short a = 23 ; //定义short型变量a、b、c,并赋值给a、b

**3.int型**

int型即整型,使用int关键字来定义int型变量，可以一次定义多个变量并对其赋值，也可以不进行赋值。int型变量取值范围很大，在-2147483648~2147483647之间，足够在一般情况下使用，所以是整数变量中应用最广泛的。

【例2.3】定义int型变量

int a = 23 , b = 78 ; ////定义int型变量a、b、c,并赋值给a、b

**4.long型**

long型即为长整形,使用long关键字来定义long型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。而在对long型变量进行赋值时结尾必须加上“L”或“l”（因为I和i的大写相似，所以我们一般使用L，便于阅读。），否则将不被认为是long型，而会被默认作为int型处理，这时可能会出现溢出内存的错误。所以当数值过大超过int型范围时就用long型，系统分给long型8个字节，取值范围更大，在-9223372036854775808~9223372036854775807之间。

【例2.4】定义long型变量，并对其进行一些分析；

class LongPractice{

public static void main(String[] args){

long a = 3890 , b； //当我们定义一个变量时如果他的数值超过int的范围,却又不对其进行末尾强制,

//b = 922337203685477 ; 那么程序会将其当做int处理,则会发生错误

//System.out.println(b); 提示我们数值过大

System.out.println(a);

b = 922337203685477L;

System.out.println(b);

b = 922337203685477l;

System.out.println(b);

//b = 922337203685477 L; 我们在定义long型变量时要注意,初始化的后面紧跟"L"或"l",不能有空格的存在,否则会提示错误

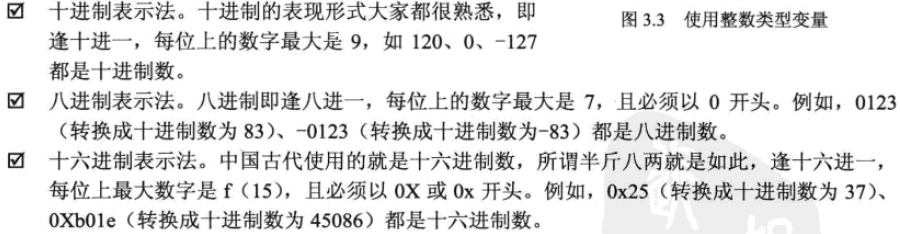
System.out.println(b);

}

}

我们在定义long型数据时末尾最好加L，因为l和1是不好区分的；

上面的四种整数类型在Java程序中有3种表示形式，分别为十进制、八进制、和十六进制。



【例2.5】整数的各型运算

public class IntegerPractice{

public static void main(String[] args){

byte a = 127 ;

short b = 3276 ;

int c = 214748364 ;

long d = 21474836423444L ;

long resultL = a + b + c + d ;

// int resultD = a + b + c + d ;当一个高精度的数据类型向低精度转换时，会报可能损失精度的错误

//int dd = (int)d;

int dd = new Long(d).intValue();

int resultD = a + b + c + dd ;

System.out.println("结果为"+resultL);

System.out.println("结果为"+resultD);

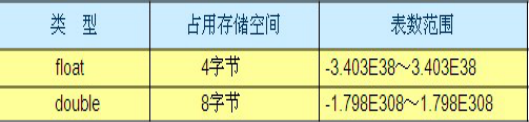
}

}



### 2.3.2 浮点类型

浮点类型表示有小数部分的数字。Java语言中浮点类型分为单精度浮点类型（float）和双精度浮点类型（double），它们具有不同的取值范围，如图所示。浮点型默认的类型为double。



1. **float型**

Float型即为单精度浮点型，使用float关键字来定义float型变量，可以一次定义一个或多个变量并对其进行赋值，也可以不进行赋值。在对float型进行赋值时末尾必须加上“f”或“F”，如果不加，系统会将其默认作为double型来处理，float型的取值范围在1.4E-45~3.4028235E38之间。

【例2.6】对float型进行定义；

class FloatPractice{

public static void main(String[] args){

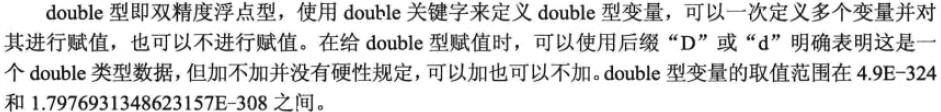
float a = 1.4E-45F ;

System.out.println(a);

}

}

1. **2.double型**



**单精度和双精度的区别在于**

单精度实数在内存中占32bit 有效数字【所谓有效数字：具体地说，是指在分析工作中实际能够测量到的数字。所谓能够测量到的是包括最后一位估计的，不确定的数字。对于一个近似数，从左边第一个不是0的数字起，到精确到的位数止，所有的数字都叫做这个数的有数字。】为6~7位，双精度实数占内存单元为64bit 有效数字为15~16位没有说明时,实型常量是作为双精度处理的,若要使用单精度需在后面加上字母f或F,如123.45f。如果我们在实际操作中操作一个单精度型变量，且其有效数字超出7位，那么我们若在其末尾加“F”或“f”，那么会只保留7位有效数字，7位后面的一位数字若大于等于5，则进1，否则其余的全部舍掉，若不加“F”或“f”，会提示错误可能损失精度，同理double型操作有效数字超出其范围也会如上操作，只保留16位有效数字，7位后面的一位数字若大于等于5，则进1，否则其余的全部舍掉。

【例2.6】定义一个double型变量和float型变量，并对它们进行赋值、运算和输出；

class DoublePractice{

public static void main(String[] args){

double a = 987.987123456789012D , b = 789.789879887918799d ;

double c = 68.8978978977 ;

float aa = 987.987123F ; //当我们定义一个float型变量却又不对其进行末尾"F"或"f",那么会提示错误可能损失精度

float d = 987.987123456789012F ;

float e = 987.987153456789012F , f = 987.987144456789012f ;

System.out.println(a+"和"+b) ;

System.out.println(c) ;

System.out.println(aa+"和"+d+"和"+e+"和"+f) ;

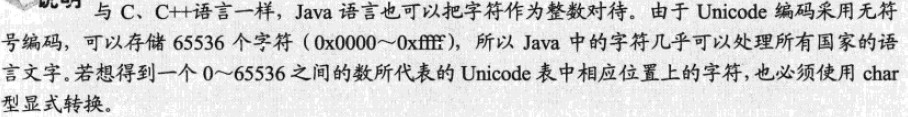
}



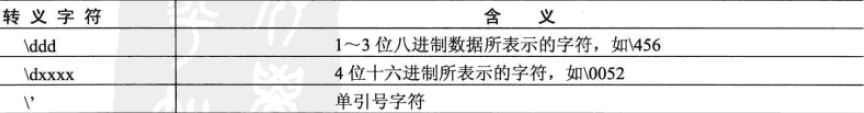
### 2.3.3 字符类型

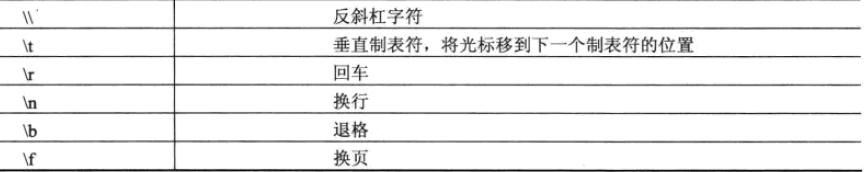
Char型即字符类型。使用char关键字进行声明，用于存储单个字符，系统分配两个字节的内存空间。在定义字符型变量时，**要用单引号‘’括起来**。如‘s’表示一个字符，且**单引号中只能有一个字符**，多了就不是字符类型，而是字符串类型，需要用双引号进行声明。

Java 字符采用 Unicode 编码，每个字符占两个字节，因而可用十六进制编码形式表示。【注：Unicode是全球语言统一编码】



在字符类型中有一种特殊的字符，以反斜线“\”开头，后跟一个或多个字符，具有特定的含义，不同于字符原有的意义，叫做转义字符，Java中的转义字符如下所示。





**注意：转义字符也是字符，所以在使用时同样要加‘’单引号。**

【例2.7】创建一个类CharPractice，实现将Unicode表中某些位置上的一些字符以及一些字符在Unicode表中的位置在控制台上面输出。

class CharPractice{

public static void main(String[] args){

int a = 'd' , aa = 97 ;

char b = 97 , c = 'a' ;

char d = '\\' , e = '\b' , f = '\n' , g = '\u2605' , h = '\u0052' ; //转义字符

char i = '\u4e08';

System.out.println(a+"和"+aa+"和"+b+"和"+c) ;

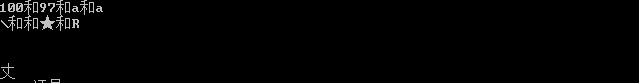
System.out.println(d+"和"+e+"和"+"和"+g+"和"+h) ;

System.out.println(f) ;

System.out.println(i);

}

}



### 2.3.4 布尔类型

布尔类型又称逻辑类型，只有“true”和“false”两个值，分别代表布尔逻辑中的“真”和“假”，使用boolean关键字来声明布尔类型变量，通常被用在流程控制中作为判断条件。

【例2.8】声明一个布尔变量，并对其进行输出；

boolean abc = true , cba = false ;

System.out.println(abc+"还是"+cba) ;

if(abc)System.out.println("真就是真！");

else{

System.out.println("假就是假！");

}

if(cba)System.out.println("真就是真！");

else{

System.out.println("假就是假！");

}



## 枚举类型

有时候变量的取值只在一个有限的集合中。例如衣服的号码。当然，可以将这些尺寸分别编码为1,2,3,4或者S，M，L，X。但这样存在着一定的隐患。那就是在变量中保存的是一个错误的值，甚至于后期修改也是一件很麻烦的时。针对这种情况，Java为我们提供了枚举类型。枚举类型包括有限个命名的值。例如，

Enum Size{SMALL,MEDIUM,LARGE,EXTRA\_LARGE};

现在我们就可以声明这种类型的变量:

Size s = Size.MEDIUM;

枚举类型的变量只能存储这个类型声明中给定的某个枚举值,或者null值,null表示这个变量没有设置任何值。关于枚举我们后面会专门用一个知识点来介绍；

# 3.类型转换

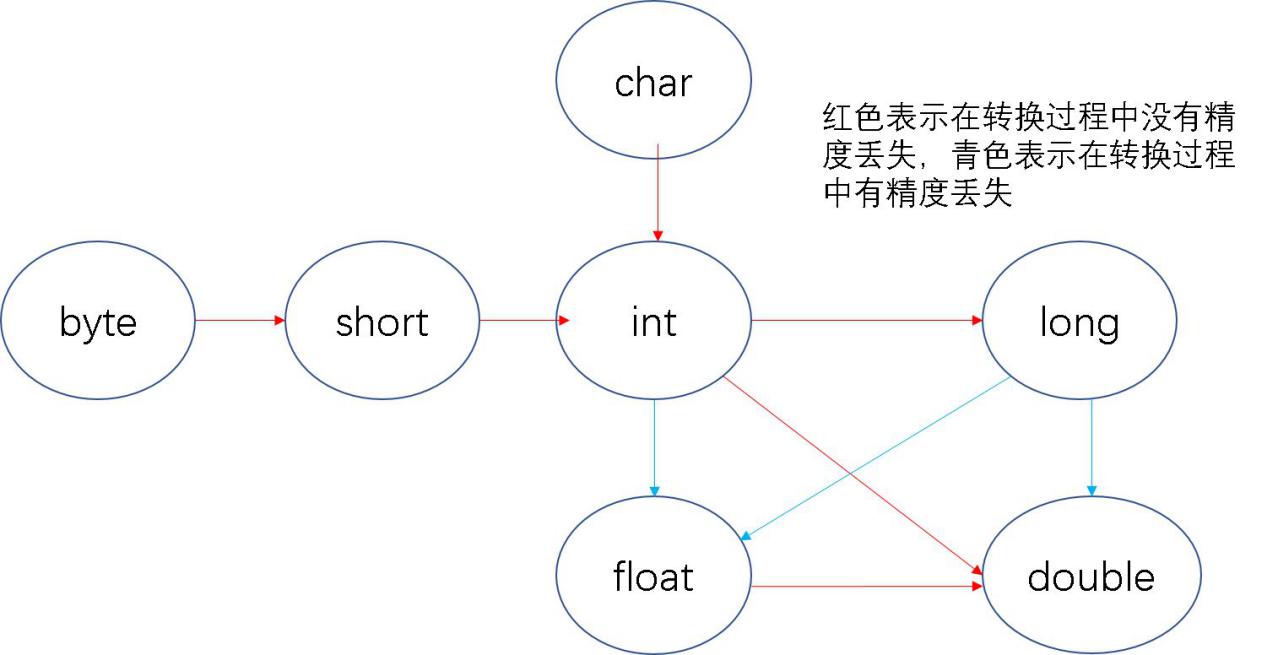
前面曾说过Java是一种强类型语言,经常需要将一种数据类型转换为另一种数据类型。根据数据类型的分类我们也将这种转换分为两类，基本数据类型的转换和运用类型的转换，下面我们来分别说明一下；

## 3.1 基本数据类型转换

类型转换是将一个值从一种类型更改为另一种类型的一个过程。例如，可以将String类型数据“457”转换为一个数值型，而且可以将任意类型的数据转换为String类型。

如果从低精度数据类型向高精度数据类型转换，则永远不会溢出，并且总是成功的；而把高精度数据类型向低精度数据类型转换时则必然会有信息损失，有可能失败。

数据类型转换有两种方式，即隐式转换和显示转换。在下图给出了数值类型之间的合法转换。例如给定一个大整数123456789是一个大整数，它包含的位数比float类型所能表达的位数多。当将这个整数数值类型转换为float类型时，将会得到同样大小的结果，但却失去了一定的精度。



int n = 123456789;///n float f = n;//f is 1.234567892E8

当使用上面两个数值进行二元操作时,先要将操作数转变为同一种类型,然后再进行计算。

* 如果两个操作数中有一个是double类型，另一个操作数就会转换为double类型。
* 否则，如果其中一个操作数是float类型，另一个操作数将会转换为float类型。
* 否则，如果其中一个操作数是long类型，另一个操作数将会转换为long类型。
* 否则，两个操作数都将被装换为int类型。

### 3.1 隐式数据类型转换

从低精度类型向高精度类型的转换，系统将会自动执行，程序员无需进行任何操作。这种类型的转换称为隐式转换，也叫自动类型转换。下列基本数据类型会涉及数据转换，不包括逻辑类型和字符类型。这些类型按精度从低到高排列顺序为**byte < short < int< long < float < double**。

**注意：byte,short,char之间不会互相转换，他们三者在计算时首先会转换为int类型，有多种类型的数据混合运算时，系统首先自动的将所有数据转换成容量最大的那一种数据类型，然后再进行计算。**

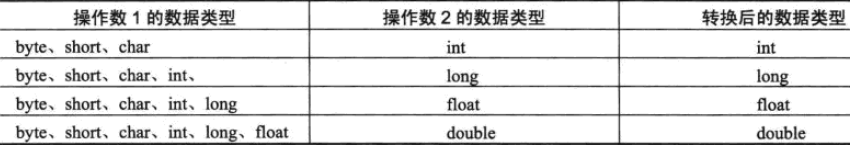
【例3.11】将下面int型数据转换为float型数据；

int x = 50 ;

float y = x ;

System.out.println(y) ; //输出的结果为50.0

隐式数据类型转换也需要遵守一定的规则，那么在运算时各个类型之间要怎么转换呢？各种情况下数据类型之间转换一般遵守如下规则。这些规则实际上是遵循上面的四条规则的。



【例3.12】创建一个类ImplicitConvertPractice，在这个类中声明各种基本数据类型变量，然后进行一系列操作，查看结果。

class ImplicitConvertPractice{

public static void main(String[] args){

byte a = 127 ;

short b = 37 ;

char ch = 'a' ;

System.out.println(ch) ;

int xy = a + b + ch ;

System.out.println("当byte short和char一起运算时,他们会先转换为int型在进行计算,因为整数类型默认为int型: " + xy) ;

int x = 50 ;

long c = 922068547 ;

float y = x ;

double d = 92233747.9798698 ;

System.out.println("当byte型和short型在一起运算时其运算结果为:" + (a+b) ) ;

System.out.println("当byte、short型和int型在一起运算时其运算结果为:" + (a+b+x)) ;

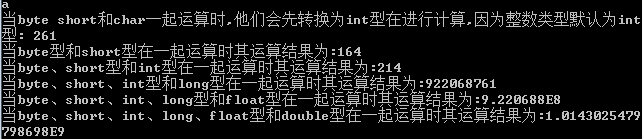
System.out.println("当byte、short、int型和long型在一起运算时其运算结果为:" + (a+b+c+x)) ;

System.out.println("当byte、short、int、long型和float型在一起运算时其运算结果为:" + (a+b+x+c+y)) ;

System.out.println("当byte、short、int、long、float型和double型在一起运算时其运算结果为:" + (a+b+c+x+y+d)) ;

}

}



### 3.2 显式数据类型转换

当把高精度的变量赋值给低精度的变量时，必须使用显示类型转换（又叫强制类型转换）。语法如下：

（低精度类型名）要转换的值

当我们把一个高精度想一个低精度转换时，不可超出这些变量取值范围，否则会发生数据溢出的现象，此时会造成数据丢失，所以在使用强制类型转换时，一定要加倍小心，不要超出变量的取值范围，否则就得不到的想要结果。比如我们进行一次byte和short之间的显示转换

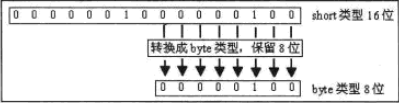
byte a = 127 ;

short b = 516 ;

byte ab = (byte)b ; //由于short的范围超出了byte的127所以发生溢出,数据损失

System.out.println(ab) ; //结果为4

原理如下:



boolean类型不能被转换为其他类型,反之亦然;

【例3.21】我们创建一个类ExplicitConvertPractice，声明各种数据类型，并对其进行转换操作；

class ExplicitConvertPractice{

public static void main(String[] args){

byte a = 127 ;

short b = 516 ;

byte ab = (byte)b ;

byte abc = (byte)(b - a - a - a - a) ;

System.out.println("当数据范围过大时会发生数据损失: "+ ab ) ; //显示转换数据丢失

System.out.println("当数据范围合适时显示转换: "+ abc ) ;

int x = 50 ;

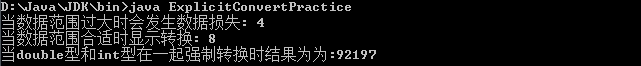
double d = 92247.8698 ;

int bx = (int)(d - x) ;

System.out.println("当double型和int型在一起强制转换时结果为为:" + bx ) ;

}

}



【例3.12】:分析System.out.println(‘a’)与System.out.println(’a’+1) 的区别;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？

class QuestionPractice{

public static void main(String[] args){

char a = 'a' ;

System.out.println('a') ;//因为字符类型没有参与运算，所以仍以字符类型输出

System.out.println(a);

System.out.println('a'+1) ;//字符类型和一个常量进行运算,字符类型和常量会都先转换为int类型,然后运算

byte b1 = 3 , b2 = 4 , b3 , b4 ;

// b3 = b1 + b2 ; b1和b2是变量，因为变量的值会变化，不确定具体的值，所以默认使用int类型进行存储,他会先自动转换为int型,计算结果为int型,而b3是byte型,是强制转换,

b3 = (byte)(b1 + b2) ;

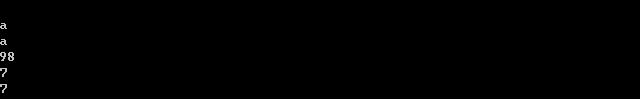
b4 = 3 + 4 ; //3和4都是常量，所以java在编译时期会检查该常量的和是否超出byte类型的范围。如果没有可以赋值。

System.out.println(b3);

System.out.println(b4);

}

}



分析：当我们声明一个字符变量，并对它进行输出时，那么我们这是输出的就只是一个字符，当我们对着个字符比变量实现运算输出时，那么这时字符会先转换成int型然后计算，结果以int型输出；

【例3.13】比较浮点类型的强制转换与Math类的round方法；

|  |
| --- |
| /\*Math类提供了四舍五入的方法  \* static long round(double a) 返回参数中最接近的 long ，其中 long四舍五入为正无穷大。  static int round(float a) 返回参数中最接近的 int ，其中 int四舍五入为正无穷大。  \* \*/  @Test  **public** **void** round() {  //如果参数为正无穷大或者是大于等于Long.MAX\_VALUE的值,那么转换后结果为Long.MAX\_VALUE  **double** d = Double.***MAX\_VALUE***;  **float** f = Float.***MAX\_VALUE***;  **long** ld= Math.*round*(d);  **int** i = Math.*round*(f);  System.***out***.println(Long.***MAX\_VALUE*** + ":" + ld + "," + i);  ld = (**long**) d;  i = (**int**) f;  System.***out***.println(Long.***MAX\_VALUE*** + ":" + ld + "," + i);  //如果参数为NaN,那么转换后结果为0  d = Double.***NaN***;  ld = Math.*round*(d);  System.***out***.println(ld);  ld = (**long**) d;  System.***out***.println(ld);  //如果参数为负无穷小或者是大小于等于Long.MIN\_VALUE的值,那么转换后结果为Long.MIN\_VALUE  d = Double.***NEGATIVE\_INFINITY***;  ld = Math.*round*(d);  System.***out***.println(ld);  ld = (**long**) d;  System.***out***.println(ld);  **double** d1 = -3.45, d2 = -3.65;  **int** i1 = (**int**) d1;  **int** i2 = (**int**) d2;  System.***out***.println(i1 + ":" + i2);  i1 = (**int**) Math.*round*(d1);  i2 = (**int**) Math.*round*(d2);  System.***out***.println(i1 + ":" + i2);  d1 = 3.45;  d2 = 3.65;  i1 = (**int**) d1;  i2 = (**int**) d2;  System.***out***.println(i1 + ":" + i2);  i1 = (**int**) Math.*round*(d1);  i2 = (**int**) Math.*round*(d2);  System.***out***.println(i1 + ":" + i2);  } |
| 9223372036854775807:9223372036854775807,2147483647  9223372036854775807:9223372036854775807,2147483647  0  0  -9223372036854775808  -9223372036854775808  -3:-3  -3:-4  3:3  3:4 |

总结：浮点类型通过强制转换为正数类型时会舍去小数部分，不管正负都是这种处理方式，而Math类的round方法则返回四舍五入的结果。round方法的处理存在着如下的几种情况:

* 如果参数是NaN，结果为0。
* 如果参数为负无穷大或小于或等于Integer.MIN\_VALUE的值，则结果等于Integer.MIN\_VALUE的值。
* 如果参数为正无穷大或大于或等于Integer.MAX\_VALUE的值，则结果等于Integer.MAX\_VALUE的

【例3.14】将boolean类型转变为数值类型；

|  |
| --- |
| @Test  **public** **void** booleanToNum() {  **boolean** boo = **false**;  System.***out***.println(boo?1:0);//0  } |

前面说过boolean不能和数值类型进行转换,但是有时候我们可能真的需要将boolean类型转变为数值,那么我们可以通过程序逻辑来完成这个操作,如上就是一个示例。

## 3.2 引用数据类型转换