Java关键字

下面是Java语言保留专用的50个关键字：

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

关键字**goto**和**const**是C++保留的关键字，目前在Java中不能使用。如果它们出现在Java程序中，Java编译器能够识别它们，并产生错误信息。

字面常量true、false和null如同字面值100一样，不是关键字。但是它们也不能用作标识符，就像100不能用作标识符一样。

assert是JDK 1.4增加的关键字，**enum**是JDK 1.5新加的关键字。

⊖ **strictfp**关键字是方法或类的修饰符，用于限制浮点数计算。浮点数运算可以在两种模式（精确或非精确模式）下执行。精确模式保证计算结果在所有Java虚拟机（JVM）实现上保持一致。非精确模式允许中间计算结果以不同于IEEE浮点数格式的扩展格式保存。扩展格式是机器相关的，并且能够使代码执行更快。但是，当使用非精确模式在不同JVM上执行时，可能不会得到相同的结果。默认情况下，非精确模式用于浮点数计算。如果要对方法和类使用精确模式，需要在类或者方法的声明前面加上**strictfp**关键字。精确浮点数比非精确浮点数的精度更好，但这种差异只影响部分应用程序。精确性不能继承；也就是说，出现在类或接口上的**strictfp**不会使扩展的类或接口同样精确。

ASCII码字符集

表B-1和表B-2列出了ASCII字符与它们相应的十进制和十六进制编码。字符的十进制或十六进制编码是字符行下标和列下标的组合。例如，在表B-1中，字母A在第6行第5列，所以它的十进制代码为65；在表B-2中，字母A在第4行第1列，所以它的十六进制代码为41。

表B-1 十进制编码的ASCII字符集

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	-	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

表B-2 十六进制编码的ASCII字符集

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so	si
1	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

运算符优先级表

运算符按照优先级递减的顺序从上到下列出。同一栏中的运算符优先级相同，它们的结合方向如表C-1中所示。

表C-1 运算符优先级

运算符	名 称	结合方向
()	圆括号	从左向右
()	函数调用	从左向右
[]	数组下标	从左向右
.	对象成员访问	从左向右
++	后置增量	从右向左
--	后置减量	从右向左
++	前置增量	从右向左
--	前置减量	从右向左
+	一元加	从右向左
-	一元减	从右向左
!	一元逻辑非	从右向左
(type)	一元类型转换	从右向左
new	创建对象	从右向左
*	乘法	从左向右
/	除法	从左向右
%	求余	从左向右
+	加法	从左向右
-	减法	从左向右
<<	左移	从左向右
>>	用符号位扩展的右移	从左向右
>>>	用零扩展的右移	从左向右
<	小于	从左向右
<=	小于等于	从左向右
>	大于	从左向右
>=	大于等于	从左向右
instanceof	检测对象类型	从左向右
==	相等	从左向右
!=	不等	从左向右
&	(无条件与)	从左向右
^	(异或)	从左向右
|	(无条件或)	从左向右
&&	条件与	从左向右
||	条件或	从左向右
?:	三元条件	从右向左
=	赋值	从右向左
+=	加法赋值	从右向左
-=	减法赋值	从右向左
*=	乘法赋值	从右向左
/=	除法赋值	从右向左
%=	求余赋值	从右向左

Java修饰符

修饰符用于类和类的成员（构造方法、方法、数据和类级块），但final修饰符也可以用在方法中的局部变量上。可以用在类上的修饰符称为*类修饰符*（class modifier）。可以用在方法上的修饰符称为*方法修饰符*（method modifier）。可以用在数据域上的修饰符称为*数据修饰符*（data modifier）。可以用在类级块上的修饰符称为*块修饰符*（block modifier）。表D-1给出Java修饰符的一个总结。

表D-1 Java修饰符

修饰符	类	构造方法	方法	数据	块	解 释
(default) ⊖	✓	✓	✓	✓	✓	类、构造方法、方法或数据域在所在的包中可见
public	✓	✓	✓	✓		类、构造方法、方法或数据域在任何包任何程序中都可见
private		✓	✓	✓		构造方法、方法或数据域只在所在的类中可见
protected		✓	✓	✓		构造方法、方法或数据域在所属包中可见，或者在任何包中该类的子类中可见
static			✓	✓	✓	定义类方法、类数据域或静态初始化模块
final	✓		✓	✓		终极类不能扩展。终极方法不能在子类中修改。终极数据域是常量
abstract	✓		✓			抽象类必须被扩展。抽象方法必须在具体的子类中实现
native			✓			用native修饰的方法表明它是用Java以外的语言实现的
synchronized			✓		✓	同一时间只有一个线程可以执行这个方法
strictfp	✓	✓				使用精确浮点数计算模式，保证在所有的Java虚拟机中计算结果都相同
transient				✓		标记实例数据域，使其不进行序列化

⊖ 默认访问权限没有与之相关的修饰符。例如`class Test {}`。

特殊浮点值

整数除以零是非法的，会抛出异常ArithmeticException，但是浮点值除以零不会引起异常。在浮点运算中，如果运算结果对double型或float型来说数值太大，则向上溢出到无穷大；如果运算结果对double型或float型来说数值太小，则向下溢出到零。Java用特殊的浮点值POSITIVE_INFINITY、NEGATIVE_INFINITY和NaN（Not a Number，非数）来表示这些结果。这些值被定义为Float类和Double类中的特殊常量。

如果正浮点数除以零，结果为POSITIVE_INFINITY。如果负浮点数除以零，结果为NEGATIVE_INFINITY。如果浮点数零除以零，结果为NaN，表示这个结果在数学上是无定义的。这三个值的字符串表示为Infinity、-Infinity和NaN。例如，

```
System.out.print( 1.0 / 0);      // Print Infinity
System.out.print( -1.0 / 0);     // Print -Infinity
System.out.print(0.0 / 0);       // Print NaN
```

这些特殊值也可以在运算中用作操作数。例如，一个数除以POSITIVE_INFINITY算出零。表E-1总结了运算符/、*、%、+和-的各种组合。

表E-1 特殊的浮点值

x	y	x/y	x*y	x%y	x+y	x−y
Finite	±0.0	± ∞	±0.0	NaN	Finite	Finite
Finite	± ∞	±0.0	±0.0	x	± ∞	∞
±0.0	±0.0	NaN	±0.0	NaN	±0.0	±0.0
± ∞	Finite	± ∞	±0.0	NaN	± ∞	± ∞
± ∞	± ∞	NaN	±0.0	NaN	± ∞	∞
±0.0	± ∞	±0.0	NaN	±0.0	± ∞	±0.0
NaN	Any	NaN	NaN	NaN	NaN	NaN
Any	NaN	NaN	NaN	NaN	NaN	NaN

注 如果一个操作数是NaN，则结果一定是NaN。

附录F

Introduction to Java Programming, 8E

数　系

1. 引言

因为计算机本身只能存储和处理0和1，所以其内部使用的是二进制数。二进制数系只有两个数：0和1。在计算机中，数字或字符是以由0和1组成的序列来存储的。每个0或1都称为一个比特（二进制数字）。

我们在日常生活中使用十进制数。当我们在程序中编写一个数字，如20，它被假定为一个十进制数。在计算机内部，通常会用软件将十进制数转换成二进制数，反之亦然。

我们使用十进制数编写程序。然而，如果要与操作系统打交道，需要使用二进制数以达到“机器级”。二进制数冗长烦琐，所以经常使用十六进制数简化二进制数，每个十六进制数可以确切表示四个二进制数。十六进制数系有十六个数：0～9、A～F，其中字母A、B、C、D、E和F对应十进制数10、11、12、13、14和15。

十进制数系中的数是0、1、2、3、4、5、6、7、8和9。一个十进制数是用一个或多个这些数所构成的一个序列来表示的。这个序列中每个数所表示的值和它的位置有关，序列中数的位置决定了10的幂次。例如，十进制数7423中的数7、4、2和3分别表示7000、400、20和3，如下所示：

$$\boxed{7}\boxed{4}\boxed{2}\boxed{3} = 7\times10^3 + 4\times10^2 + 2\times10^1 + 3\times10^0$$

$$10^3\ 10^2\ 10^1\ 10^0 = 7000 + 400 + 20 + 3 = 7423$$

十进制数系有十个数，它们的位置值都是10的整数次幂。10是十进制数系的基数。类似地，由于二进制数系有两个数，所以它的基数为2；而十六进制数系有16个数，所以它的基数为16。

如果1101是一个二进制数，那么数1、1、0和1分别表示：

$$\boxed{1}\boxed{1}\boxed{0}\boxed{1} = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$$

$$2^3\ \ 2^2\ \ 2^1\ \ 2^0 = 8 + 4 + 0 + 1 = 13$$

如果7423是一个十六进制数，那么数字7、4、2和3分别表示：

$$\boxed{7}\boxed{4}\boxed{2}\boxed{3} = 7\times16^3 + 4\times16^2 + 2\times16^1 + 3\times16^0$$

$$16^3\ 16^2\ 16^1\ 16^0 = 28\ 672 + 1024 + 32 + 3 = 29\ 731$$

2. 二进制数与十进制数之间的转换

给定二进制数$b_n b_{n-1} b_{n-2} \cdots b_2 b_1 b_0$，等价的十进制数为

$$b_n\times2^n + b_{n-1}\times2^{n-1} + b_{n-2}\times2^{n-2} + \cdots + b_2\times2^2 + b_1\times2^1 + b_0\times2^0$$

下面是二进制数转换为十进制数的例子：

二 进 制	转 换 公 式	十 进 制
10	$1\times2^1+0\times2^0$	2
1000	$1\times2^3+0\times2^2+0\times2^1+0\times2^0$	8
10101011	$1\times2^7+0\times2^6+1\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0$	171

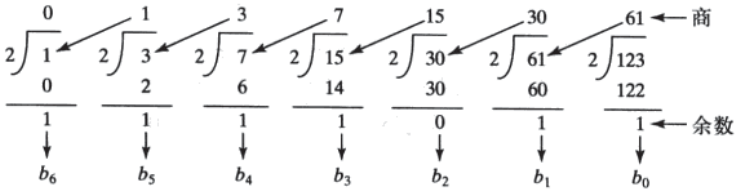
把一个十进制数d转换为二进制数，就是求满足

$$d = b_n\times2^n + b_{n-1}\times2^{n-1} + b_{n-2}\times2^{n-2} + \cdots + b_2\times2^2 + b_1\times2^1 + b_0\times2^0$$

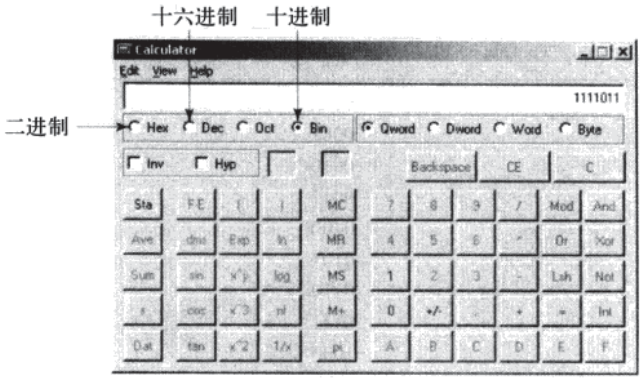
的位b_n，b_{n-1}，b_{n-2}，…，b_2，b_1和b_0。用2不断地除d，直到商为0为止，余数即为所求的位$b_0, b_1, \cdots,$

b_{n-2}, b_{n-1}, b_n。

例如，十进制数123用二进制数1111011表示，所做的转换如下：



提示 Windows操作系统所带的计算器是进行数制转换的一个有效工具，如图F-1所示。要运行它，单击“开始”按钮，依次选择“程序”，“附件”，“计算器”命令。



图F-1 使用Windows的计算器进行数制转换

3. 十六进制数与十进制数的转换

给定十六进制数$h_n h_{n-1} h_{n-2} \cdots h_2 h_1 h_0$，其等价的十进制数为

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

下面是十六进制数转换为十进制数的例子：

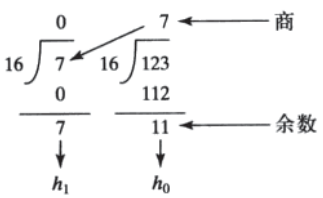
十六进制	转换公式	十进制
7F	$7 \times 16^1 + 15 \times 16^0$	127
FFFF	$15 \times 16^3 + 15 \times 16^2 + 15 \times 16^1 + 15 \times 16^0$	65535
431	$4 \times 16^2 + 3 \times 16^1 + 1 \times 16^0$	1073

将一个十进制数d转换为十六进制数，就是求满足

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \cdots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

的位h_n，h_{n-1}，h_{n-2}，…，h_2、h_1和h_0。用16不断地除d，直到商为0为止。余数即为所求的位h_0，h_1，…，h_{n-2}，h_{n-1}和h_n。

例如，十进制数123用十六进制表示为7B，所做的转换如下：



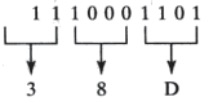
4. 二进制数与十六进制数的转换

将一个十六进制数转换为二进制数，只需利用表F-1，就可以把十六进制数的每一位转换为四位二进制数。

例如，十六进制数7B转换为二进制是1111011，其中7的二进制表示为111，B的二进制表示为1011。

要将一个二进制数转换为十六进制数，从右向左将每四位二进制数转换为一位十六进制数。

例如，二进制数1110001101的十六进制表示是38D，因为1101是D，1000是8，11是3，如下所示：



表F-1 十六进制数转换为二进制数

十六进制	二进制	十进制
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

注意 有时也要用到八进制数，八进制数系有0到7共八个数。十进制数8在八进制数系中的作用就和十进制数系中的10一样。

复习题

1. 将下列十进制数转换为十六进制数和二进制数。
   100；4340；2000
2. 将下列二进制数转换为十六进制数和十进制数。
   1000011001；100000000；100111
3. 将下列十六进制数转换为二进制数和十进制数。
   FEFA9；93；2000