

一、变量与运算符

1、关键字（keyword）

定义：被 Java 语言赋予了特殊含义，用做专门用途的字符串。
特点：全部关键字都是小写字母。
说明：1>关键字一共 50 个，其中 **const** 和 **goto** 是保留字(reserved word)。
2>**true, false, null** 不是关键字，其实是**字面量**，表示特殊的布尔值和空值。

用于定义数据类型的关键字				
class	interface	enum	byte	short
int	long	float	double	char
boolean	void			
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
const	goto			
* 用于定义数据类型值的字面值				
true	false	null		

2、标识符

1) Java 中变量、方法、类等要素命名时使用的字符串序列，称为标识符。
2) 标识符的命名规则（必须遵守的硬性规定）：
1>由 26 个英文字母大小写，0-9，_或\$组成；
2>数字不可以开头；（避免歧义：int 123L=12; long l=123L;l 的值是 123 还是 12？）
3>不可以使用**关键字和保留字**，**但能包含**关键字和保留字；
4>Java 中**严格区分大小写**，**长度无限制**；
5>标识符**不能包含空格**。
3) 标识符的命名规范
1>包名：多单词组成时**所有字母都小写**：xxxxyyzzz；
2>类名、接口名：多单词组成时，**所有单词的首字母大写**：XxxYyyZzz；
3>变量名、方法名：（**驼峰**）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；
4>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。

3、变量

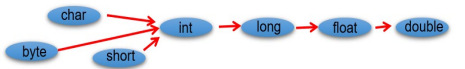
1) 变量是程序中不可或缺的组成单位，最基本的存储单元；
2) 内存中的一个存储区域，该区域的数据可以在同一类型范围内不断变化；
3) 变量的构成三个要素：**数据类型、变量名、存储的值**。

4、整数类型：byte、short、int、long

类 型	占用存储空间	表数范围
byte	1字节=8bit位	-128 ~ 127
short	2字节	-2 ¹⁵ ~ 2 ¹⁵ -1
int	4字节	-2 ³¹ ~ 2 ³¹ -1 (约21亿)
long	8字节	-2 ⁶³ ~ 2 ⁶³ -1

1) 定义 long 类型的变量，赋值时需要以**"l"或"L"**作为**后缀**，在 int 范围内可以不加后缀。
2) Java 程序中变量通常声明为 int 型，除非不足以表示较大的数，才使用 long。
3) Java 的**整型常量默认为 int 型**。
【补充】：计算机存储单位
字节（Byte）：是计算机用于**计量存储容量的基本单位**，一个字节等于 8 bit。
位（bit）：是**数据存储的最小单位**。二进制数系统中，每个 0 或 1 就是一个位，叫做 bit（比特），其中 **8 bit** 就称为一个字节(Byte)。
5、浮点类型：float、double

类 型	占用存储空间	表数范围
单精度float	4字节	-3.403E38 ~ 3.403E38
双精度double	8字节	-1.798E308 ~ 1.798E308

1) 浮点型常量有两种表示形式：
1>**十进制数**形式。如：5.12 512.0f .512 (**必须有小数点**)。
2>**科学计数法**形式如：5.12e2 512E2 100E-2。
2) float: 单精度，尾数可以精确到 7 位有效数字。很多情况下，精度很难满足需求。
3) double: 双精度，精度是 float 的两倍。通常采用此类类型。
4) 定义 float 类型的变量，赋值时需要以**"f"或"F"**作为**后缀**，不然报错。
5) Java 的**浮点型常量默认为 double 型**。
6) 关于浮点型精度的说明：
1>并不是所有的小数都能可以精确的用二进制浮点数表示。二进制浮点数不能精确的表示 0.1、0.01、0.001 这样 10 的负次幂。（即解释了 0.1+0.2≠0.3）
2>浮点类型 float、double 的数据**不适合**在不容许舍入误差的金融计算领域。如果**需要精确数字计算或保留指定位数的精度**，需要使用 **BigDecimal** 类。
6、字符类型：char（占 2 字节）
1) char 型数据用来表示通常意义上“字符”；
2) Java 中的所有字符都使用 **Unicode 编码**，故一个字符可以存储一个字母，一个汉字，或其他书面语的一个字符。
3) 字符型变量的三种表现形式：
形式 1：使用单引号' '括起来的单个字符。
例如：char c1='a'; char c2='中'; char c3='9';
形式 2：直接使用 Unicode 值来表示字符型常量：**"\uXXXX"**。其中，XXXX 代表一个十六进制整数。
例如：\u0023 表示 '#'。
形式 3：Java 中还允许使用转义字符\ '来将其后的字符转变为特殊字符型常量。
例如：char c3='\n'; // '\n'表示换行符。
【注意】char 类型是可以进行运算的。因为它都对应有 Unicode 码，可看做是一个数值。
7、布尔类型：boolean
1) boolean 类型用来判断逻辑条件，一般用于流程控制语句中。
2) boolean 类型数据**只有两个值：true、false**，无其它；
1>不可以使用 **0 或非 0 的整数替代 false 和 true**，这点和 C 语言不同。
2>拓展：Java 虚拟机中没有任何供 boolean 值专用的字节码指令，Java 语言表达所操作的 **boolean 值，在编译之后都使用 java 虚拟机中的 int 数据类型来代替**：true 用 1 表示，false 用 0 表示。
经验之谈：写法是 if(isFlag)或者 if(!isFlag)。
8、基本数据类型变量间运算规则
不同的基本数据类型变量的值经常需要进行相互转换。（只有 7 种，不包含 boolean 类型）
两种转换方式：**自动类型提升、强制类型转换**。
1) 自动类型提升
规则：将取值范围小（或容量小）的类型自动提升为取值范围大（或容量大）的类型。


```
graph LR; byte --> int; short --> int; char --> int; int --> long; long --> float; float --> double;
```


1>当把存储范围小的值（常量值、变量的值、表达式计算的结果值）**赋值**给了存储范围大的变量时，**自动提升**；
2>当存储范围小的数据类型与存储范围大的数据类型变量一起**混合运算**时，会按照其中最大的类型运算；
3>当 **byte,short,char** 数据类型的变量进行**算术运算时，按照 int 类型处理**。
byte b1=1; byte b2=2;

byte b3=b1+b2;//编译报错，【注意】
改成 **int b3=b1+b2;**或 **b3=(byte)(b1+b2);**
char c1='0'; char c2='A';
int i=c1+c2;//至少需要使用 **int 类型**来接收
2) 强制类型转换
规则：将取值范围大（或容量大）的类型强制转换成取值范围小（或容量小）的类型。
1>当把存储范围大的值（常量值、变量的值、表达式计算的结果值）强制转换为存储范围小的变量时，**可能会损失精度或溢出**；
double d=1.2; int num=(int)d;//**损失精度**
int i=200; byte b=(byte)i;//**溢出**
2>当某个值想要提升数据类型时，**也可以使用强制类型转换**。这种情况的强制类型转换是没有风险的，通常省略。
3>声明 **long 类型变量**时，可以出现**省略后缀的情况（看成 int 自动提升，前提在 int 范围内）**。**float 则不同（浮点型默认为 double，不能自动转换为 float）**。
long l1=123L;
long l2=123;//此时可以看做是 int 类型的 123 自动类型提升为 long 类型
//long l3=123123123123; // 报 错 ， 因为 123123123123 超出了 int 的范围。
long l4=123123123123L;
//float f1=12.3; //报错，因为 12.3 看做是 double，不能自动转换为 float 类型
float f2=12.3F; float f3=(float)12.3;
9、基本数据类型与 String 的运算
字符串类型：String；属于引用数据类型；使用一对""来表示一个字符串，内部可以包含 0 个、1 个或多个字符。
运算规则
1) 任意八种基本数据类型的数据与 String 类型**只能进行连接"+"运算，且结果一定也是 String 类型**；
2) String 类型**不能通过强制类型转换**转为其他的类型，**需借助包装类**的方法才能转。
10、运算符（Operator）
运算符的分类：
1>按照功能分为：**算术运算符、赋值运算符、比较(或关系)运算符、逻辑运算符、位运算符、条件运算符、Lambda 运算符**。
2>按照操作数个数分为：**一元运算符（单目运算符）、二元运算符（双目运算符）、三元运算符（三目运算符）**。
1) 算术运算符
1>取模时，**结果与被模数符号相同**；
2>对于+两边都是数值的话，+就是加法的意思；对于+两边至少有一边是**字符串**的话，+就是**拼接**的意思。
System.out.println("5+5="+5+5); // 5+5=55
System.out.println("5+5="+ (5+5)); // 5+5=10
3>++和--**运算不会改变变量的数据类型**
byte bb1=127; bb1++;//01111111→10000000
System.out.println("bb1="+bb1);//-128
4> int i=2; int j=i++; //j=2
int i=2; int j=++i; //j=3
int m=2; m=m++; //m=2
2) 赋值运算符
1>当"="两侧数据类型不一致时，可以使用自动类型转换或强制类型转换原则进行处理。
2>支持连续赋值。
int a2,b2; a2=b2=10;
3>举例说明+=、-=、*=、/=、%=的操作不会改变变量本身的数据类型：
推荐：short s1=10; s1+=2;
//编译通过，因为在得到 int 类型的结果后，JVM 自动完成一步强制类型转换，将 int 类型强转成 short。
注：short s2=10; s2=(short)(s2+2);//正确
//s2=s2+2;//编译报错，因为 int 类型的结果赋值给 short 类型的变量 s 时，可能损失精度。

3）比较(关系)运算符

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

1>比较运算符的结果都是 boolean 型，也就是要么是 true，要么是 false；
2 > " > " " < " " > " " = " " <= " " : 只适用于基本数据类型（除 boolean 类型之外）；
3> " == " " != " : 适用于基本数据类型和引用数据类型；
4 >比较运算符 " == " 不能误写成 " = " 。

4）逻辑运算符

a	b	a&b	a&&b	a b	a b	!a	a^b
true	true	true	true	true	true	false	false
true	false	false	false	true	true	false	true
false	true	false	false	true	true	true	true
false	false	false	false	false	false	true	false

逻辑运算符，操作的都是 **boolean 类型**的变量或常量，且**运算结果也是 boolean 类型的值**。
1> 运算符说明：
逻辑运算符用于连接布尔型表达式，在 Java 中**不能写成 3 < x < 6，应写成 x > 3 & x < 6。**
2> 区分“&”和“&&”：（右边操作执不执行）
相同点：如果符号左边是 true，则二者都执行符号右边的操作；
不同点：&：如果符号左边是 false,则继续执行符号右边的操作；
&&：如果符号左边是 false,则不再继续执行符号右边的操作；
建议：开发中，推荐使用&&。

3> 区分“|”和“||”：（右边操作执不执行）
相同点：如果符号左边是 false，则二者都执行符号右边的操作；
不同点：|：如果符号左边是 true，则继续执行符号右边的操作；
||：如果符号左边是 true，则不再继续执行符号右边的操作；
建议：开发中，推荐使用||。

5）位运算符（难点、非重点）
1> 左移：<<
运算规则：在一定范围内，数据每向左移动一位，**相当于原数据*2**。（正数、负数都适用）

【注意】当左移的位数 **n 超过该数据类型的总位数**时，相当于**左移（n-总位数）**位。
3<<4 类似于 3*2 的 4 次幂 => 3*16 => 48；
-3<<4 类似于 -3*2 的 4 次幂 => -3*16 => -48；
2> 右移：>>

运算规则：在一定范围内，数据每向右移动一位，**相当于原数据/2**。（正数、负数都适用）
【注意】如果不能整除，**向下取整**。
69>>4 类似于 69/2 的 4 次= 69/16 =4；
-69>>4 类似于 -69/2 的 4 次= -69/16 = -5；

3> 无符号右移：>>>
运算规则：往右移动后，左边空出来的位直接补 0。（正数、负数都适用）
69>>>4 类似于 69/2 的 4 次= 69/16 =4；
-69>>>4

```
/*
-69的二进制：
补码：1000 0000 0000 0000 0000 0000 0100 0101
反码：1111 1111 1111 1111 1111 1111 1011 1010
补码：1111 1111 1111 1111 1111 1111 1011 1011
-69>>>4: 0000 1111 1111 1111 1111 1111 1011 1011
无符号右移 右边移出 4位
反码：0000 1111 1111 1111 1111 1111 1011 1011
原码：0000 1111 1111 1111 1111 1111 1011 1011
最高位是0，是正数，那么原码，反码，补码一样
*/
计算用补码，看结果用原码
```

4> 按位与：&
运算规则：对应位都是 1 才为 1，否则为 0。

```
9 & 7 = 1
/*
9的二进制：0000 0000 0000 0000 0000 0000 1001
7的二进制：0000 0000 0000 0000 0000 0000 0111
9&7: 0000 0000 0000 0000 0000 0000 0001
*/
-9 & 7 = 7
/*
-9的二进制：
原码：1000 0000 0000 0000 0000 0000 1001
反码：1111 1111 1111 1111 1111 1111 0110
补码：1111 1111 1111 1111 1111 1111 0111
7的二进制：0000 0000 0000 0000 0000 0000 0111
-9&7: 0000 0000 0000 0000 0000 0000 0111
补码：0000 0000 0000 0000 0000 0000 0111
反码：0000 0000 0000 0000 0000 0000 0111
原码：0000 0000 0000 0000 0000 0000 0111
*/
```

5> 按位或：|
6> 按位异或：^
7> 按位取反：~；~0 就是 1；~1 就是 0。
案例 1：高效的方式计算 2 * 8 的值
答案：2 << 3、8 << 1。

案例 2：交换两个变量 m,n 的值
方式 1：借助一个临时变量
优：适用于不同数据类型；
缺：需要额外定义变量；
int temp = m;
m = n;
n = temp;

方式 2：一加两减
优：没有额外定义变量；
缺：可能超出 int 的范围；只适用于数值类型；
m = m + n; //15 = 10 + 5
n = m - n; //10 = 15 - 5
m = m - n; //5 = 15 - 10

方式 3：借助异或运算（根据异或运算的性质：一个数进行两次异或运算还是其本身）
优：没有额外定义变量；
缺：只能适用于数值类型；
m = m ^ n;
n = m ^ n; //(m ^ n) ^ n -> m
m = m ^ n; //(m ^ n) ^ m -> n

6）条件运算符
格式：（条件表达式）？表达式 1:表达式 2；
说明：条件表达式是 boolean 类型的结果，根据 boolean 的值选择表达式 1 或表达式 2（true 则表达式 1；false 则表达式 2）。
注意：1>如果运算后的结果赋给新的变量，要求表达式 1 和表达式 2 为同种或兼容的类型；
2>凡是可以使用条件运算符的地方，都可以改写为 if-else 结构。反之，不成立。

//获取两个数的较大值
int max1 = (m1 > m2)? m1 : m2;
//获取三个数的较大值
//写法 1：
int tempMax = (n1 > n2)? n1:n2;
int finalMax = (tempMax > n3)? tempMax : n3;
//写法 2：不推荐，可读性差
int finalMax1 = (((n1 > n2)? n1:n2) > n3)? ((n1 > n2)? n1:n2) : n3;

7）运算符优先级
不要过多的依赖运算的优先级，可读性太差，尽量使用()来控制表达式的执行顺序。

二、流程控制语句

1、流程控制语句是用来控制程序中各语句执行顺序的语句，可以把语句组合成能完成一定功能的小逻辑模块。
程序设计中规定的三种**流程结构**，即：

1) 顺序结构：程序从上到下逐行地执行，中间没有任何判断和跳转。
2) 分支结构：根据条件，选择性执行某段代码。（有 if...else 和 switch-case 两种分支语句）
3) 循环结构：根据循环条件重复性执行某段代码。（有 for、while、do-while 三种循环语句）
【补充】JDK5.0 提供了 **foreach 循环**，方便的遍历集合、数组元素。

2、if-else 条件判断结构

结构 1：单分支条件判断：if；
结构 2：双分支条件判断：if...else...；
结构 3：多分支条件判断：if...else if...else...；
注意：1）一旦条件表达式为 true，则进入执行相应的语句块。执行完对应的语句块之后，就跳出当前结构。
2）当条件表达式之间是“**互斥**”关系时（即彼此没有交集），条件判断语句及执行语句间顺序无所谓；当条件表达式之间是“**包含**”关系时，“**小上大下 / 子上父下**”，否则范围小的条件表达式将不可能被执行。

3）if...else 嵌套：在 if 的语句块中，或者是在 else 语句块中，又包含了另外一个条件判断（可以是单分支、双分支、多分支），就构成了嵌套结构。
4）语句块只有一条执行语句时，一对{}可以省略，但建议保留。当 if-else 结构是“多选一”时，最后 else 是可选的，根据需要可以省略。

3、switch-case 选择结构

使用注意点：1）**switch(表达式)**中表达式的值必须是下述类型之一：**byte, short, char, int, 枚举 (jdk 5.0), String (jdk 7.0)；**
2）case 子句中的值**必须是常量，不能是变量名或不确定的表达式值或范围**；
3）同一个 switch 语句，所有 case 子句中的**常量值互不相同**；
4）break 语句用来在执行完一个 case 分支后使程序跳出 switch 语句块；如果没 break，程序会顺序执行到 switch 结尾（**case 的穿透性**）；
5）default 子句是可选的。同时位置也是灵活的。当**没有匹配的 case 时**，执行 default 语句。

4、if-else 语句与 switch-case 语句比较

1) 结论：凡是使用 **switch-case 的结构**都可以**转换为 if-else 结构**。反之，不成立。
2) 开发经验：如果既可以使用 **switch-case**，又可以使用 if-else，建议使用 switch-case。因为**效率稍高**。
3) 细节对比：

if-else 语句优势

1> if 语句的条件是一个布尔类型值，if 条件表达式为 true 则进入分支，可以用于**范围的判断**，也可以用于等值的判断，使用范围更广。
2> switch 语句的**条件是一个常量值**（byte,short,int,char,枚举,String），只能判断某个变量或表达式的结果**是否等于某个常量值**，使用场景较狭窄。

switch 语句优势

1> 当条件是判断某个变量或表达式是否等于某个固定的常量值时，if 和 switch 都可以，习惯上使用 switch 更多，因为效率稍高。当条件是区间范围的判断时，只能使用 if 语句。
2> 使用 switch 可以利用**穿透性**，同时执行多个分支，而 if...else 没有穿透性。

5、循环语句

1）如何结束一个循环结构？
情况 1：循环结构中的**循环条件部分返回 false**；
情况 2：循环结构中**执行了 break**。

2）for 循环

for (①初始化部分; ②循环条件部分; ④迭代部分) { ③循环体部分; }
说明：• for(;;)中的两个；不能多也不能少；
• ①**初始化**部分可以声明多个变量，但必须是**同一个类型**，用**逗号分隔**；
• ②循环条件部分为 boolean 类型表达式，当值为 false 时，退出循环；
• ④可以有多个变量更新，用逗号分隔。

3) while 循环

①初始化部分
while(②循环条件部分) {
③循环体部分; ④迭代部分; }
【注意】：for 循环与 while 循环的区别：**初始化条件部分的 作用域不同**。

4) do-while 循环

①初始化部分;

do{ ③循环体部分

④迭代部分

}while(②循环条件部分);

注意: 1> do {} while();最后有一个分号;

2> do-while 结构的循环体语句是至少会执行一次。(和 for 和 while 的区别)而 for 和 while 循环先判断循环条件语句是否成立, 然后决定是否执行循环体。

5) 如何选择

1> 遍历有明显的循环次数(范围)的需求, 选择 for 循环;

2> 遍历没有明显的循环次数(范围)的需求, 选择 while 循环;

3> 如果循环体语句块至少执行一次, 可以考虑使用 do-while 循环。

6、"无限"循环

适用场景: 开发中, 有时并不确定需要循环多少次, 需要根据循环体内部某些条件, 来控制循环的结束(使用 break)。

7、嵌套循环(或多重循环)

特点: 外层循环执行一次, 内层循环执行一轮。

8、关键字 break 和 continue 的使用

break:一旦执行就结束(跳出)当前循环结构;

continue:一旦执行结束(跳出)当次循环结构;

【注意】: 1) 很多语言都有 goto 语句, goto 语句可以随意将控制转移到程序中的任意一条语句上, 然后执行它, 但使程序容易出错。Java 中的 break 和 continue 是不同于 goto 的。

2) 带标签的使用(了解)

break\continue 出现在多层嵌套的语句块中时, 可以通过标签指明要终止的是哪一层语句块。

```
class BreakContinueTest2 {  
    public static void main(String[] args) {  
        label:for(int i = 1; i <= 4; i++){  
            for(int j = 1; j <= 10; j++){  
                if(j % 4 == 0){  
                    //break label;  
                    continue label;  
                }  
                System.out.print(j);  
            }  
            System.out.println();  
        }  
    }  
}
```

9、Scanner: 键盘输入功能的实现

如何从键盘获取不同类型(基本数据类型、String 类型)的变量: 使用 Scanner 类:

```
//9. 导包  
import java.util.Scanner;  
public class ScannerTest1 {  
    public static void main(String[] args) {  
        //9. 创建 Scanner 的对象  
        //System.in 默认代表键盘输入  
        Scanner scanner = new Scanner(System.in);  
        //9. 根据提示, 调用 Scanner 的方法, 获取不同类型的变量  
        System.out.print("请输入你的网名: ");  
        String name = scanner.next();  
        System.out.print("请输入你的年龄: ");  
        int age = scanner.nextInt();  
        System.out.print("请输入你的体重: ");  
        double weight = scanner.nextDouble();  
        System.out.print("你是否单身 (true/false): ");  
        boolean isSingle = scanner.nextBoolean();  
        System.out.print("请输入你的性别: ");  
        //先按照字符串接收, 然后再取字符串的第一个字符 (下标为 0)  
        char gender = scanner.next().charAt(0);  
        System.out.println("网名: " + name + "\n年龄: " + age + "\n体重: " + weight + "\n单身: " + isSingle + "\n性别: " + gender);  
        //9. 关闭资源  
        scanner.close();  
    }  
}
```

1) 键盘输入代码的四个步骤:

1>导包: import java.util.Scanner;

2>创建 Scanner 类型的对象: Scanner scan = new Scanner(System.in);

3>调用 Scanner 类的相关方法(字符串: next()
/其他基本类型: nextXxx()), 来获取指定类型

的变量;

4>释放资源: scan.close()。

2) 【注意】: 1>需要根据相应的方法, 来输入指定类型的值。如果输入的数据类型与要求的类型不匹配时, 会报异常导致程序终止。

2>Scanner 类中提供了获取 byte\short\int\long\float\double\boolean\String 类型变量的方法, 注意, 没有提供获取 char 类型变量的方法, 需要使用 scanner.next().charAt(0); (0 表示第 0 个字符)。

10、标准输入、输出流

1) System.in 和 System.out 分别代表了系统标准的输入和输出设备; 默认输入设备是: 键盘, 输出设备是: 显示器。

1>System.in 的类型是 InputStream;

2>System.out 的类型是 PrintStream, 其是 OutputStream 的子类 FilterOutputStream 的子类。

3) 重定向: 通过 System 类的 setIn, setOut 方法对默认设备进行改变。

- public static void setIn(InputStream in)

- public static void setOut(PrintStream out)

11、拓展:

System 类中有三个常量对象: System.out、System.in、System.err;

查看 System 类中这三个常量对象的声明:

public final static InputStream in = null;

public final static PrintStream out = null;

public final static PrintStream err = null;

1) 这三个常量对象有 final 声明, 但是却初始化为 null。final 声明的常量一旦赋值就不能修改, 那么 null 不会空指针异常吗?

2) 这三个常量对象为什么要小写? final 声明的常量按照命名规范不是应该大写吗?

3) 这三个常量的对象有 set 方法? final 声明的常量不是不能修改值吗? set 方法是如何修改它们的值的?

【原因】final 声明的常量, 表示在 Java 的语法体系中它们的值是不能修改的, 而这三个常量对象的值是由 C/C++等系统函数进行初始化和修改值的, 所以它们故意没有用大写, 也有 set 方法。

12、打印流

实现将基本数据类型的数据格式转化为字符串输出。打印流: PrintStream 和 PrintWriter。

```
PrintStream ps = null;  
try {  
    FileOutputStream fos = new FileOutputStream(new File("D:\\IO\\text.txt"));  
    // 创建打印输出流, 设置为自动刷新模式(写入换行符或字节 '\n' 时都会刷新输出缓冲区)  
    ps = new PrintStream(fos, true);  
    if (ps != null) { // 把标准输出流(控制台输出)改成文件  
        System.setOut(ps);  
    }  
    for (int i = 0; i <= 255; i++) { // 输出 ASCII 字符  
        System.out.print((char) i);  
        if (i % 50 == 0) { // 每 50 个数据一行  
            System.out.println(); // 换行  
        }  
    }  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
}  
finally {  
    if (ps != null) {  
        ps.close();  
    }  
}
```

三、JDK8-17 新特性