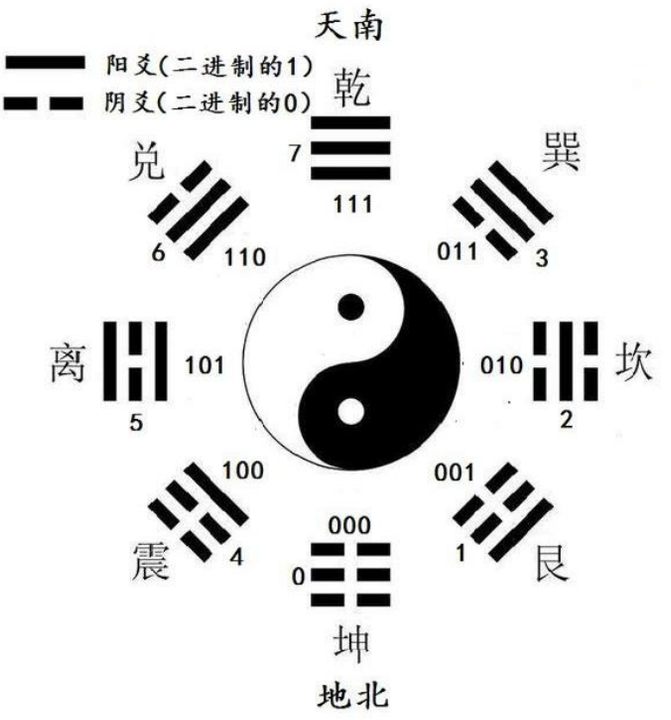
02 变量和数据类型和运算符

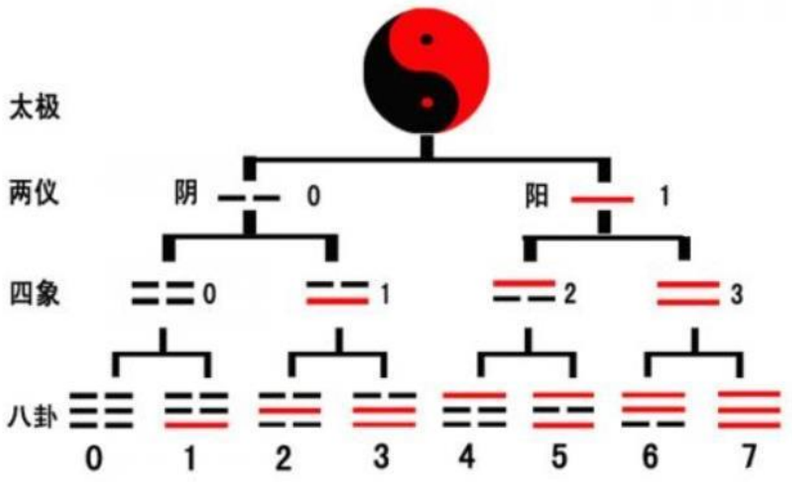
本章介绍一些编程中的基本概念，比如：标识符、变量、常量、数据类型、运算符、基本数据类型的类型转换等。这些是编程中的“砖块”，是编程的基础。 要想开始正式编程，还需要再学“控制语句”，控制语句就像“水泥”，可以把“砖块”粘到一起，最终形成“一座大厦”。控制语句将在第三章学习。

老鸟建议

- 学习本章，一定不要纠结于概念，不要停留，大致了解就快速开始下一章。永远记住“快速入门、快速实战；实战中提高，发展中解决问题”。

二进制

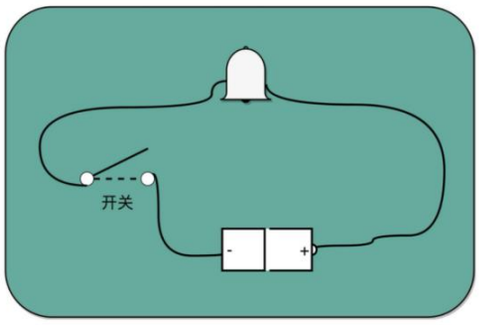


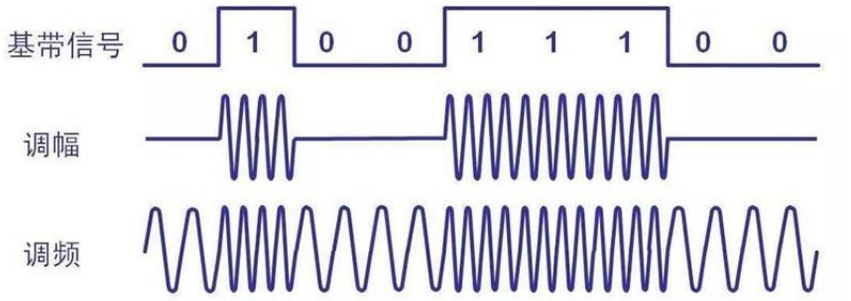


二进制，是计算技术中广泛采用的一种数制，由德国数理哲学大师莱布尼茨于 1679 年发明。二进制数据是用 0 和 1 两个数码来表示的数。它的基数为 2，进位规则是“逢二进一”。

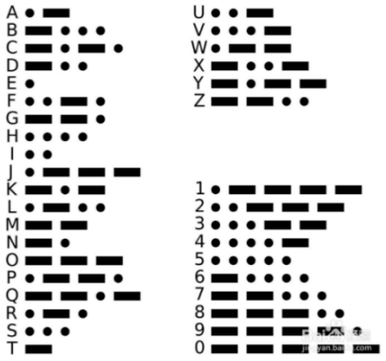
数字计算机只能识别和处理由‘0’、‘1’符号串组成的代码。其运算模式正是二进制。

二进制对应两种状态，广泛应用于电子科学。比如：可以对应电子器件的开关状态、对应信号电压状态（+5V 等价于逻辑"1"，0V 等价于逻辑"0"）、对应卡带是否打孔状态、电磁存储（磁体状态：南为 0，北为 1）等等。





二进制广泛应用于我们生活的方方面面。比如，广泛使用的摩尔斯电码，它由两种基本信号组成：短促的点信号“·”，读“滴”；保持一定时间的长信号“—”，读“嗒”。然后，组成了 26 个字母，从而拼写出相应的单词。

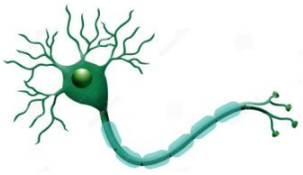


当然，我们没有必要记住这些。如果一定想学，记住 SOS 就好了^_^

万物总有规律；有规律，就可量化；可量化，就能数字化；数字化，就能使用计算机化。

高淇

【脑机接口】未来最难、最有想象力的突破：



二进制和十进制的转化

在线进制转化的工具： https://tool.lu/hexconvert/

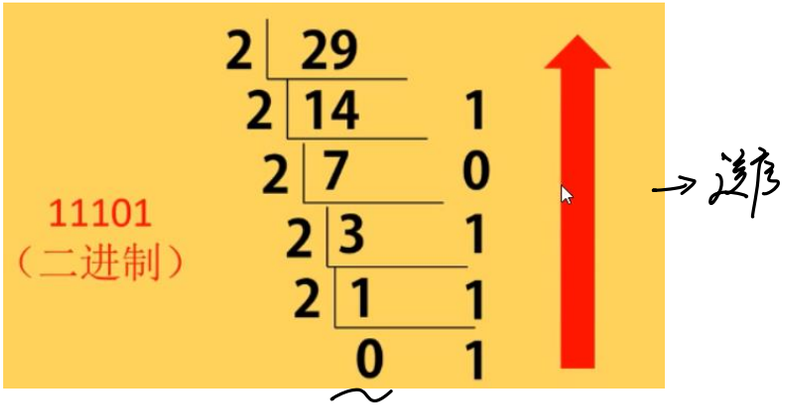
二进制和十进制数的对应

十进制数	二进制	十六进制
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c

13	1101	d
14	1110	e
15	1111	f

十进制转二进制

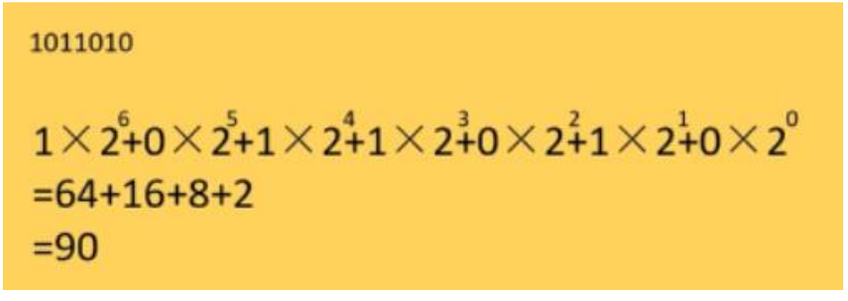
十进制整数转换为二进制整数采用"除 2 取余，逆序排列"法。



十进制数 29 转成二进制就是：11101

二进制转十进制

二进制转十进制采用“权相加法”。



注释

写注释的好处：

做一个人见人爱的程序员！

不写注释的好处：

做一个离职后，前公司还得求你的程序员！

注释不会出现在字节码文件中，即 Java 编译器编译时会跳过注释语句。

在 Java 中根据注释的功能不同，主要分为单行注释、多行注释和文档注释。

- **单行注释**

  单行注释使用“//”开头。

- **多行注释**

  多行注释以“/*”开头以“*/”结尾。注意，多行注释不能嵌套使用。

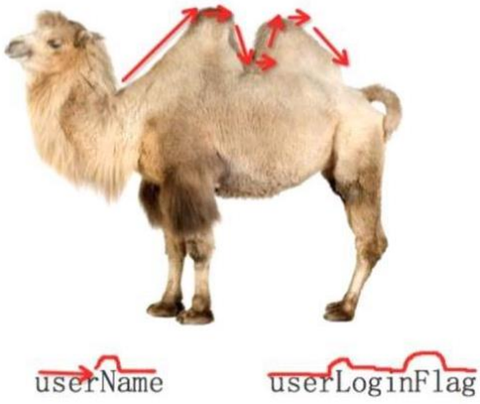
- **文档注释**

  文档注释以“/**”开头以“*/”结尾，注释中包含一些说明性的文字及一些 JavaDoc 标签（后期写项目时，可以生成项目的 API）

【示例 2-1】认识 Java 的三中注释类型

```
/**
 *Welcome 类（我是文档注释）
 * @author 高淇
 * @version 1.0
 */
public class Welcome {
    //我是单行注释
    public static void main(String[ ] args/*我是行内注释 */) {
        System.out.println("Hello World!");
    }
    /*
        我是多行注释!
        我是多行注释!
     */
}
```

标识符和关键字



标识符是用来给变量、类、方法以及包进行命名的。4 大规则：

1. 必须以字母、下划线_、美元符号$开头。
2. 其它部分可以是字母、下划线“_”、美元符“$”和数字的任意组合。
3. 大小写敏感，且长度无限制。
4. 不可以是 Java 的关键字。

标识符的使用规范

- 表示类名的标识符：每个单词的首字母大写，如 Man, GoodMan
- 表示方法和变量的标识符：第一个单词小写，从第二个单词开始首字母大写，我们称之为“驼峰原则”，如 eat(), eatFood()

Java 不采用 ASCII 字符集，而是采用 Unicode 字符集。因此，这里字母的含义不仅仅是英文，还包括汉字等等。但是不建议大家使用汉字来定义标识符！

【示例 2-2】合法的标识符

```
int  a = 3;
int  _123 = 3;
int  $12aa = 3;
int  变量1 = 55;  //不建议使用中文命名的标识符
```

【示例 2-3】不合法的标识符

```
int  1a = 3;    //不能用数字开头
int  a# = 3;    //不能包含#这样的特殊字符
int  int = 3;   //不能使用关键字
```

关键字/保留字

Java 关键字是 Java 语言保留供内部使用的，如 class 用于定义类。 我们不能使用关键字作为变量名或方法名。

表 Java 中的关键字/保留字

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

菜鸟雷区

出于应试教育的惯性思维，很多新手很可能去背上面的单词，从实战思维出发，我们不需要刻意去记！随着学习的深入，自然就非常熟悉了。

变量(variable)

变量的本质

1. 变量本质上就是代表一个”可操作的存储空间”，空间位置是确定的，但是里面放置什么值不确定。

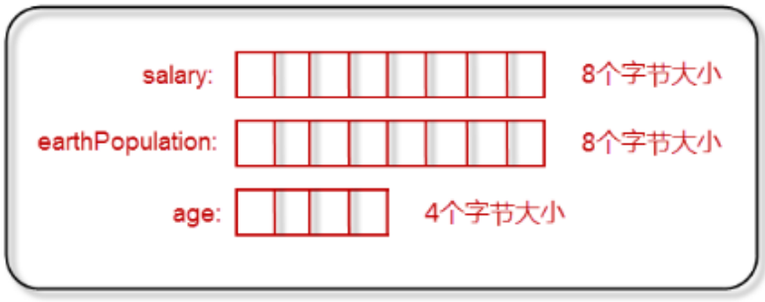
2. 可通过变量名来访问“对应的存储空间”，从而操纵这个“存储空间”存储的值。

3. Java 是一种强类型语言，每个变量都必须声明其数据类型。变量的数据类型决定了变量占据存储空间的大小。 比如，int a=3; 表示 a 变量的空间大小为 4 个字节。

•变量的声明

```
double  salary;
long  earthPopulation;
int  age;
```

不同数据类型的常量会在内存中分配不同的空间，如图 2-1 所示。



【示例】声明变量和初始化

```
int  age = 18;
double  e = 2.718281828;
int  i ,j; //  两个变量的数据类型都是 int
```

变量的分类和作用域

变量有三种类型：局部变量、成员变量(也称为实例变量)和静态变量。

局部变量、成员变量、静态变量的核心区别

类型	声明位置	从属于	生命周期（作用域）
局部变量	方法或语句块内部	方法/语句块	从声明位置开始，直到方法或语句块执行完毕，局部变量消失
成员变量 (实例变量)	类内部，方法外部	对象	对象创建，成员变量也跟着创建。对象消失，成员变量也跟着消失；
静态变量 (类变量)	类内部，static 修饰	类	类被加载，静态变量就有效；类被卸载，静态变量消失。

老鸟建议

❑ 成员变量和静态变量不是目前重点，不要过多纠结理解与否。我们学习面向对象时，再重点讲解成员变量和静态变量。

- **局部变量(local variable)**

方法或语句块内部定义的变量。生命周期是从声明位置开始到到方法或语句块执行完毕为止。局部变量在使用前必须先声明、初始化(赋初值)再使用。

【示例】局部变量的声明

```
public void test() {
    int i;
    int j = i+5 ; // 编译出错，变量 i 还未被初始化
}
```

```
public void test() {
    int i;
    i=10;
    int j = i+5 ; // 编译正确
}
```

- **成员变量（也叫实例变量 member variable）【暂不用掌握，讲面向对象再说】**

方法外部、类的内部定义的变量。从属于对象，生命周期伴随对象始终。如果不自行初始化，它会自动初始化成该类型的默认初始值。

表 2-3 实例变量的默认初始值

数据类型	实始值
int	0
double	0.0
char	'\u0000'
boolean	false

【示例 2-8】实例变量的声明

```
public class Test {
    int i;
}
```

- **静态变量（类变量 static variable）【暂不用掌握，讲面向对象再说】**

使用 static 定义。 从属于类，生命周期伴随类始终，从类加载到卸载。 **（注：讲完内存分析后我们再深入！先放一放这个概念！）**如果不自行初始化，与成员变量相同会自动初始化成该类型的默认初始值，如表 2-3 所示。

注意：

目前大家只需要了解基本的分类概念即可。不需要掌握，后面讲了方法、对象、类以后再深入学习。

常量(Constant)

在 Java 语言中，用关键字 final 来定义一个常量。常量一旦被初始化后不能再更改。

声明格式：

```
final   type   varName = value;
```

【示例 2-9】常量的声明及使用

```
public class TestConstants {
    public static void main(String[ ] args) {
        final double PI = 3.14;
        // PI = 3.15; //编译错误，不能再被赋值!
        double r = 4;
        double area = PI * r * r;
        double circle = 2 * PI * r;
        System.out.println("area = " + area);
        System.out.println("circle = " + circle);
    }
}
```

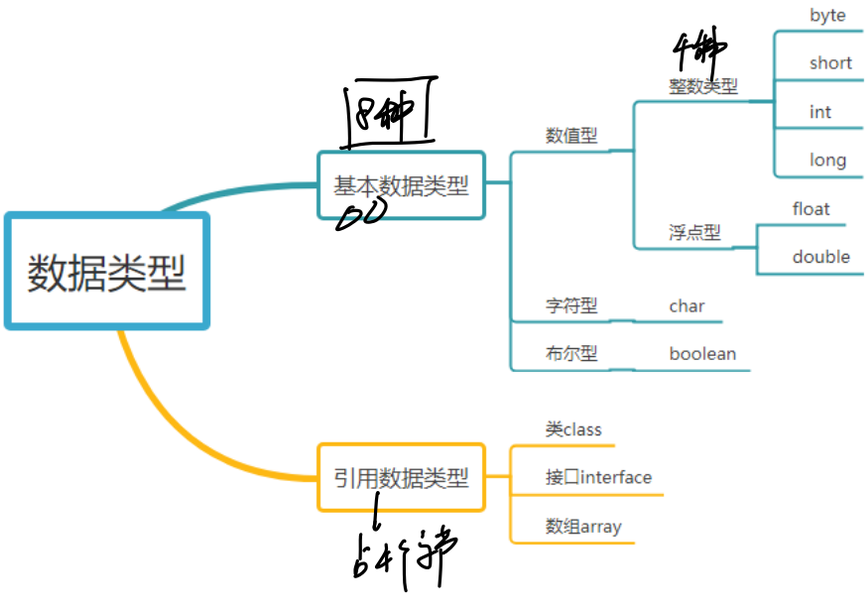
为了更好的区分和表述，一般将 1、2、3、’ a’ 、’ b’ 、true、false、” helloWorld” 等称为**字符常量**，而使用 final 修饰的 PI 等称为**符号常量**。

老鸟建议

变量和常量命名规范

- ❑ 所有变量、方法、类名：见名知义
- ❑ 类成员变量：首字母小写和驼峰原则： monthSalary
- ❑ 局部变量：首字母小写和驼峰原则
- ❑ 常量：大写字母和下划线：MAX_VALUE
- ❑ 类名：首字母大写和驼峰原则： Man, GoodMan
- ❑ 方法名：首字母小写和驼峰原则: run(), runRun()

基本数据类型(primitive data type)

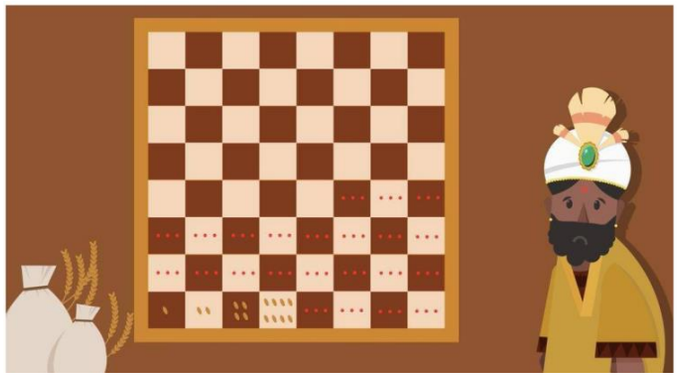


Java 数据类型分为两大类:基本数据类型(primitive data type)和引用数据类型(reference data type）。

注意事项

- ❑ 引用数据类型的大小统一为 4 个字节，记录的是其引用对象的地址!
- ❑ 本章只讲解基本数据类型。引用数据类型在后续数组和面向对象章节讲解。

整型



类型	占用存储空间	表数范围
byte	1 字节	$-2^7 \sim 2^7-1$（-128~127）
short	2 字节	$-2^{15} \sim 2^{15}-1$ （-32768~32767）
int	4 字节	$-2^{31} \sim 2^{31}-1$ (-2147483648~2147483647) 约 21 亿
long	8 字节	$-2^{63} \sim 2^{63}-1$

Java 语言整型常量的四种表示形式

- 十进制整数，如：99, -500, 0
- 八进制整数，要求以 0 开头，如：015
- 十六进制数，要求 0x 或 0X 开头，如：0x15
- 二进制数，要求 0b 或 0B 开头，如：0b01110011

Java 语言的整型常数默认为 int 型，声明 long 型常量可以后加 'l' 或 'L'。

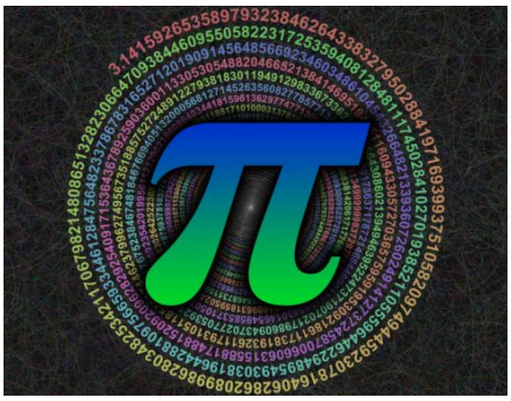
【示例】long 类型常数的写法及变量的声明

```
long a = 55555555;  //编译成功，在 int 表示的范围内(21 亿内)。
long b = 55555555555;//不加 L 编译错误，已经超过 int 表示的范围。
```

报错：The literal 55555555555 of type int is out of range，所以我们需要修改代码为：

```
long b = 55555555555L;
```

浮点型(Floating Point Number)



类型	占用存储空间	表数范围
float	4 字节	-3.403E38~3.403E38
double	8 字节	-1.798E308~1.798E308

1. float 类型又被称作单精度类型，尾数可以精确到 7 位有效数字。

2. double 表示这种类型的数值精度约是 float 类型的两倍，又被称作双精度类型，**绝大部分应用程序都采用 double 类型。**

3. Java 浮点类型常量有两种表示形式

(1) 十进制数形式： 例: 3.14 314.0 0.314

(2) 科学记数法形式 例：3.14e0 3.14E2 3.14E-1

4. 浮点型不精确，不要用于比较

> 浮点数存在舍入误差，数字不能精确表示。**浮点数适合普通的科学和工程计算，精度足够；但不适合精度要求非常高的商业计算，这时候要使用 BigDecimal 进行运算和比较。**

5. 浮点常量默认类型是 double，要改成 float 可以后面加 F 或 f

【示例】使用科学记数法给浮点型变量赋值

```
double f = 314e2;   //314*10^2-->31400.0
double f2 = 314e-2; //314*10^(-2)-->3.14
```

float类型的数值有一个后缀F或者f ，没有后缀F/f的浮点数值默认为double类型。也可以在浮点数值后添加后缀D或者d， 以明确其为double类型。

【示例】float 类型常量的写法及变量的声明

```
float   f = 3.14F;//float 类型赋值时需要添加后缀 F/f
double d1= 3.14;
double d2 = 3.14D;
```

【示例 2-13】浮点型数据的比较一

```
float f = 0.1f;
double d = 1.0/10;
System.out.println(f==d);//结果为 false
```

【示例 2-14】浮点型数据的比较二

```
float d1 = 423432423f;
float d2 = d1+1;
if(d1==d2){
    System.out.println("d1==d2");//输出结果为 d1==d2
}else{
    System.out.println("d1!=d2");
}
```

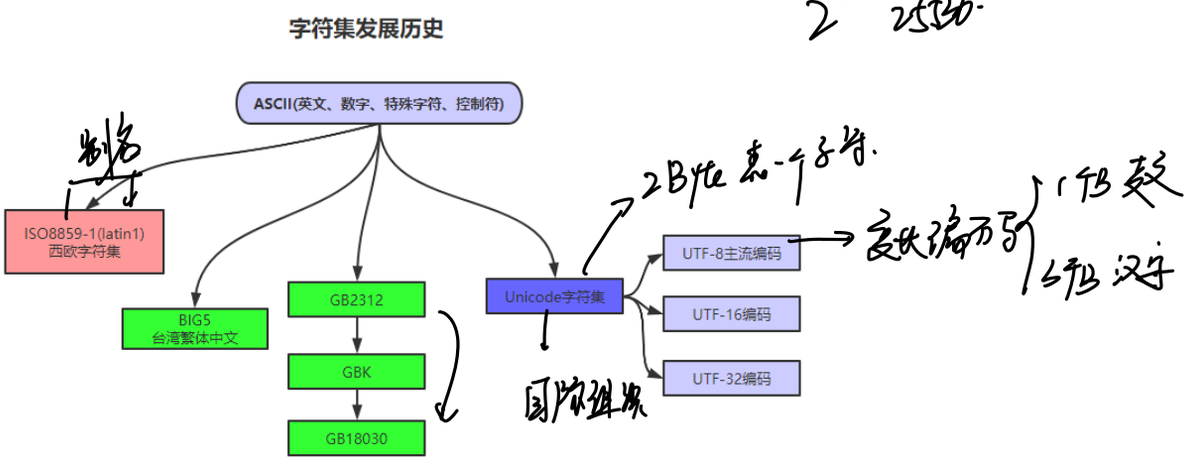
运行以上两个示例，发现示例2-13的结果是“false”，而示例2-14的输出结果是“d1==d2”。这是因为由于字长有限，浮点数能够精确表示的数是有限的，因而也是离散的。 浮点数一般都存在舍入误差，很多数字无法精确表示(例如0.1)，其结果只能是接近， 但不等于。二进制浮点数不能精确的表示0.1、0.01、0.001这样10的负次幂。并不是所有的小数都能可以精确的用二进制浮点数表示。

java.math 包下面的两个有用的类：BigInteger 和 BigDecimal，这两个类可以处理任意长度的数值。BigInteger 实现了任意精度的整数运算。BigDecimal 实现了任意精度的浮点运算。

菜鸟雷区

不要使用浮点数进行比较！很多新人甚至很多理论不扎实的有工作经验的程序员也会犯这个错误！需要比较请使用 BigDecimal 类

字符型



ASCII control characters			ASCII printable characters					
00	NULL	(Null character)	32	space	64	@	96	`
01	SOH	(Start of Header)	33	!	65	A	97	a
02	STX	(Start of Text)	34	"	66	B	98	b
03	ETX	(End of Text)	35	#	67	C	99	c
04	EOT	(End of Trans.)	36	$	68	D	100	d
05	ENQ	(Enquiry)	37	%	69	E	101	e
06	ACK	(Acknowledgement)	38	&	70	F	102	f
07	BEL	(Bell)	39	'	71	G	103	g
08	BS	(Backspace)	40	(	72	H	104	h
09	HT	(Horizontal Tab)	41	)	73	I	105	i
10	LF	(Line feed)	42	*	74	J	106	j
11	VT	(Vertical Tab)	43	+	75	K	107	k
12	FF	(Form feed)	44	,	76	L	108	l
13	CR	(Carriage return)	45	-	77	M	109	m
14	SO	(Shift Out)	46	.	78	N	110	n
15	SI	(Shift In)	47	/	79	O	111	o
16	DLE	(Data link escape)	48	0	80	P	112	p
17	DC1	(Device control 1)	49	1	81	Q	113	q
18	DC2	(Device control 2)	50	2	82	R	114	r
19	DC3	(Device control 3)	51	3	83	S	115	s
20	DC4	(Device control 4)	52	4	84	T	116	t
21	NAK	(Negative acknowl.)	53	5	85	U	117	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v
23	ETB	(End of trans. block)	55	7	87	W	119	w
24	CAN	(Cancel)	56	8	88	X	120	x
25	EM	(End of medium)	57	9	89	Y	121	y
26	SUB	(Substitute)	58	:	90	Z	122	z
27	ESC	(Escape)	59	;	91	[	123	{
28	FS	(File separator)	60	<	92	\	124	|
29	GS	(Group separator)	61	=	93	]	125	}
30	RS	(Record separator)	62	>	94	^	126	~
31	US	(Unit separator)	63	?	95	_		
127	DEL	(Delete)						

ASCII 字符集表示了英文字母、数字、特殊字符、控制符，所有字符集的老祖宗，大家都会兼容它。但是一个字节能够表示 256 个状态，而 ASCII 字符只用到 128 个，后面 128 个一直是空的。

于是有了 ISO8859-1,别名叫 latin-1，包含了 256 个字符。前 128 个字符与 ASCII 中完全相同。后 128 个包括了西欧语言、希腊语、泰语、阿拉伯语、希伯来语对应的文字符号。

随着我国的计算机普及，汉字的处理也有了我们自己的方案。那就是 GB2312，两个字节表示 1 个汉字。两个字节可以表示 65536 个状态，汉字再多也能全部包含。后来，又有了 GBK、GB18030。

我国的台湾地区自己搞了一套显示繁体中文的大五码 BIG5。

全世界各个地方，都有自己的文字编码。由于不互通，经常造成乱码的问题。

如果有一种统一的字符集，将世界上所有语言字符都纳入其中，每一个字符都给予一个全球独一无二的编码，那么乱码问题就会消失。于是，全球所有国家和民族使用的所有语言字符的统一字符集诞生了，这就是 Unicode 字符集。

Unicode 字符集是为了给全世界所有字符一个唯一的编码，“唯一”对应的英文为 Unique，而编码的英文为 code。

Unicode 采用了字符集和编码分开的策略。Unicode 之前，Unicode 诞生之前可以将字符集和字符编码混为一谈，而在 Unicode 中必须严格区分开。

Unicode 字符集统一采用两个字节表示一个字符，包括英文字母。但是，由于英文占据互联网信息的绝大部分。真实存储和传输时，会造成极大的浪费；因此，目前主要采用 UTF-8 编码来实现具体的存储和传输。UTF-8 是变长编码，用 1-6 个字节编码 Unicode 字符。西欧字符仍然是 1 个字节，汉字 3 个字节。

字符型在内存中占 2 个字节，在 Java 中使用单引号来表示字符常量。例如’ A’ 是一个字符，它与” A” 是不同的，” A” 表示含有一个字符的字符串。

char 类型用来表示在 **Unicode** 编码表中的字符。Unicode 编码被设计用来处理各种语言的文字，它占 2 个字节，可允许有 65536 个字符。

【示例 2-16】字符型演示

```
char eChar = 'a';
char cChar ='中';
```

Unicode 具有从 0 到 65535 之间的编码，他们通常用从’ \u0000’ 到’ \uFFFF’ 之间的十六进制值来表示（前缀为 u 表示 Unicode）

【示例 2-17】字符型的十六进制值表示方法

```
char c = '\u0061';
```

Java 语言中还允许使用转义字符 ‘\’ 来将其后的字符转变为其它的含义。常用的转义字符及其含义和 Unicode 值如表 2-6 所示。

【示例 2-18】转义字符

```
char c2 = '\n';   //代表换行符
```

表 2-6 转义字符

转义符	含义	Unicode 值

\b	退格（backspace）	\u0008
\n	换行	\u000a
\r	回车	\u000d
\t	制表符（tab）	\u0009
\ “	双引号	\u0022
\ ’	单引号	\u0027
\\	反斜杠	\u005c

注意事项

以后我们学的 String 类，其实是字符序列(char sequence)，本质是 char 字符组成的数组。

布尔型(boolean)

1. boolean 类型有两个常量值，true 和 false。
2. 在内存中占一个字节或 4 个字节，不可以使用 0 或非 0 的整数替代 true 和 false，这点和 C 语言不同。

【注意点】

JVM 规范指出 boolean 当做 int 处理，也就是 4 字节，boolean 数组当做 byte 数组处理，这样我们可以得出 boolean 类型占了单独使用是 4 个字节，在数组中是确定的 1 个字节。

【示例 2-19】boolean 类型演示

```
boolean flag ;
flag = true;    //或者 flag=false;
if(flag) {
      // true 分支
} else {
      //   false 分支
}
```

老鸟建议

Less is More!！请不要这样写：if（flag = = true），只有新手才那么写。关键也很容易写错成if(flag=true)，这样就变成赋值 flag 为 true 而不是判断！老鸟的写法是 if (flag)或者 if (!flag)

运算符(operator)

计算机的基本用途就是执行数学运算，Java 提供了一套丰富的运算符来操作变量。

算术运算符	二元运算符	+，-，*，/，%
	一元运算符	++，--
赋值运算符		=
扩展运算符		+=，-=，*=，/=
关系运算符		>，<，>=，<=，==，!= instanceof
逻辑运算符		&&，||，!，^
位运算符		&，|，^，~ ，>>，<<
条件运算符		?：
字符串连接符		+

算术运算符

1. +，-，*，/，%属于二元运算符。%是取模运算符，就是我们常说的求余数操作。
2. 算术运算符中++(自增)，--(自减)属于一元运算符。

二元运算符的运算规则：

整数运算：

- 如果两个操作数有一个为 long，则结果也为 long。
- 没有 long 时，结果为 int。即使操作数全为 short，byte，结果也是 int。

浮点运算：

- 如果两个操作数有一个为 double，则结果为 double。
- 只有两个操作数都是 float，则结果才为 float。

取模运算：

- 其操作数可以为浮点数,一般使用整数，结果是“余数”，“余数”符号和左边操

作数相同，如：7%3=1，-7%3=-1，7%-3=1。

【示例】一元运算符++与--

```
int a = 3;
int b = a++;    //执行完后,b=3。先给b赋值，再自增。
System.out.println("a="+a+"\nb="+b);
a = 3;
b = ++a;    //执行完后,b=4。a先自增，再给b赋值
System.out.println("a="+a+"\nb="+b);
```

运行该程序，执行结果如图2-3所示。

```
a=4
b=3
a=4
b=4
```

运行效果图

赋值及其扩展赋值运算符

运算符	用法举例	等效的表达式
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

【示例 2-21】扩展运算符

```
int a=3;
int b=4;
a+=b;//相当于a=a+b;
System.out.println("a="+a+"\nb="+b);
a=3;
a*=b+3;//相当于a=a*(b+3)
System.out.println("a="+a+"\nb="+b);
```

运行该程序，执行结果如图2-4所示。

```
a=7
b=4
a=21
b=4
```

图 2-4 示例 2-21 运行效果图

关系运算符

关系运算符用来进行比较运算。关系运算的结果是布尔值：true/false；

运算符	含义	示例
==	等于	a==b
!=	不等于	a!=b
>	大于	a>b
<	小于	a<b
>=	大于或等于	a>=b
<=	小于或等于	a<=b

注意事项

- =是赋值运算符，而真正的判断两个操作数是否相等的运算符是==。
- ==、!= 是所有（基本和引用）数据类型都可以使用。
- > 、>=、 <、 <= 仅针对数值类型（byte/short/int/long，float/double 以及 char）。

逻辑运算符

逻辑运算的操作数和运算结果都是 boolean 值。

运算符	说明

与	&	只要有一个为 false，则 false
短路与	&&	
或	|	只要有一个为 true， 则 true
短路或	||	
非	!	取反
异或	^	相同为 false，不同为 true

短路与和短路或采用短路的方式。从左到右计算，如果只通过运算符左边的操作数就能够确定该逻辑表达式的值，则不会继续计算运算符右边的操作数，提高效率。

【示例 2-22】短路与和逻辑与

```
//1>2 的结果为 false，那么整个表达式的结果即为 false，将不再计算 2>(3/0)
boolean c = 1>2 && 2>(3/0);
System.out.println(c);
//1>2 的结果为 false，那么整个表达式的结果即为 false，还要计算 2>(3/0)，0 不能做除数，//会输出异常信息
boolean d = 1>2 & 2>(3/0);
System.out.println(d);
```

位运算符

位运算指的是进行二进制位的运算。

位运算符	说明
~	取反
&	按位与
|	按位或
^	按位异或
<<	左移运算符，左移 1 位相当于乘 2
>>	右移运算符，右移 1 位相当于除 2 取商

【示例 2-23】左移运算和右移运算

```
int a = 3*2*2;
int b = 3<<2;  //相当于：3*2*2;
int c = 12/2/2;
int d = 12>>2; //相当于 12/2/2;
```

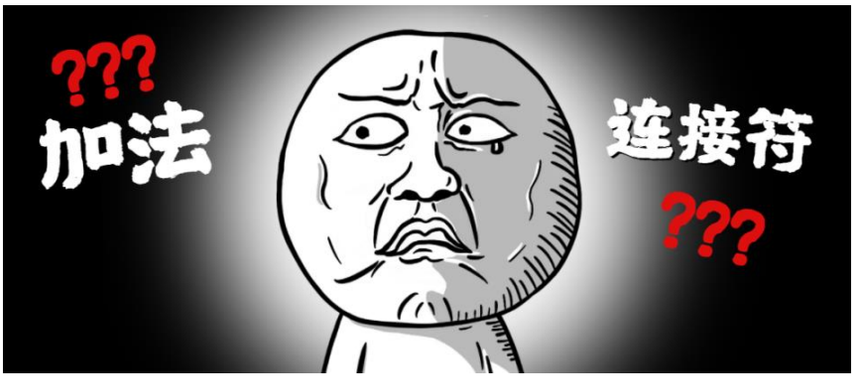
新手雷区

- &和|既是逻辑运算符，也是位运算符。如果两侧操作数都是 boolean 类型，就作为逻辑

运算符。如果两侧的操作数是整数类型，就是位运算符。

- 不要把“^”当做数学运算“乘方”，是“位的异或”操作。

字符串连接符

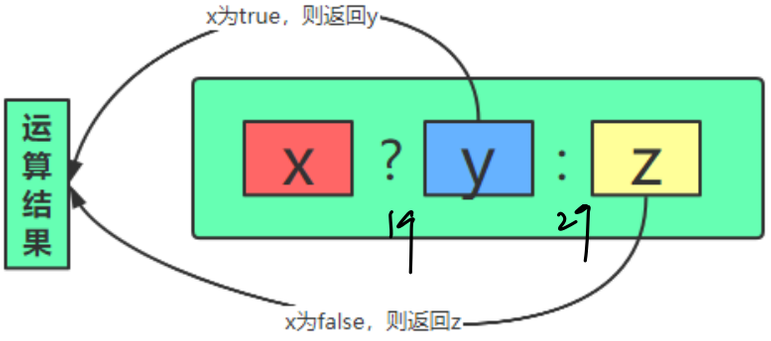


“+”运算符两侧的操作数中只要有一个是字符串(String)类型，系统会自动将另一个操作数转换为字符串然后再进行连接。

【示例 2-24】连接符“+”

```
int a=12;
System.out.println("a="+a);//输出结果: a=12
```

条件运算符




```
String type = score<60?"不及格":"及格";
```

x 为 boolean 类型表达式，先计算 x 的值，若为 true，则整个运算的结果为表达式 y 的值，否则整个运算结果为表达式 z 的值。

【示例 2-25】条件运算符

```
int score = 80;
int x = -100;
String type =score<60?"不及格":"及格";
int flag = x > 0 ? 1 : (x == 0 ? 0 : -1);
System.out.println("type= " + type);
System.out.println("flag= "+ flag);
```

运行结果如图 2-5 所示。

```
type= 及格
flag= -1
```

运行效果图

运算符优先级的问题

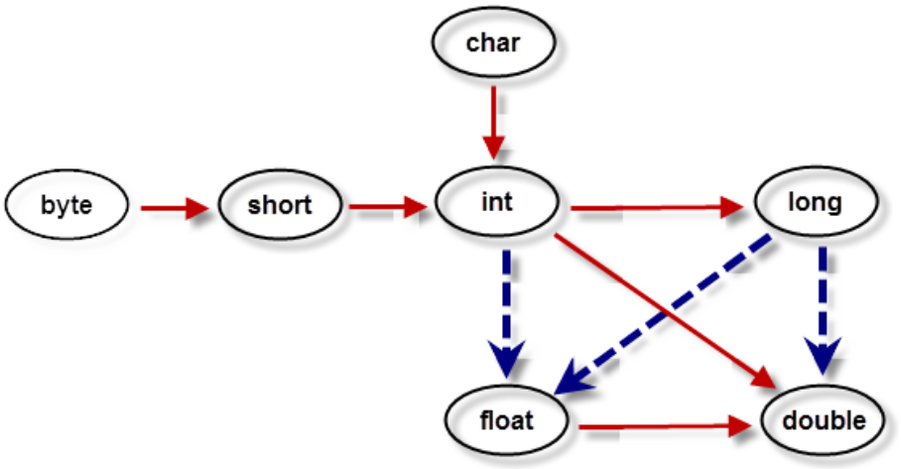
运算符的优先级

优先级	运算符	类
1	()	括号运算符
2	!、+（正号）、-（负号）	一元运算符
2	~	位逻辑运算符
2	++、--	递增与递减运算符
3	*、/、%	算术运算符
4	+、-	算术运算符
5	<<、>>	位左移、右移运算符
6	>、>=、<、<=	关系运算符
7	==、!=	关系运算符
8	&	位运算符、逻辑运算符
9	^	位运算符、逻辑运算符
10	|	位运算符、逻辑运算符
11	&&	逻辑运算符
12	||	逻辑运算符
13	?:	条件运算符
14	=、+=、-=、*=、/=、%=	赋值运算符、扩展运算符

老鸟建议

- 大家不需要去刻意的记这些优先级，表达式里面优先使用小括号来组织！！
- 逻辑与、逻辑或、逻辑非的优先级一定要熟悉！（逻辑非>逻辑与>逻辑或）。如：
- a||b&&c 的运算结果是：a||(b&&c)，而不是(a||b)&&c

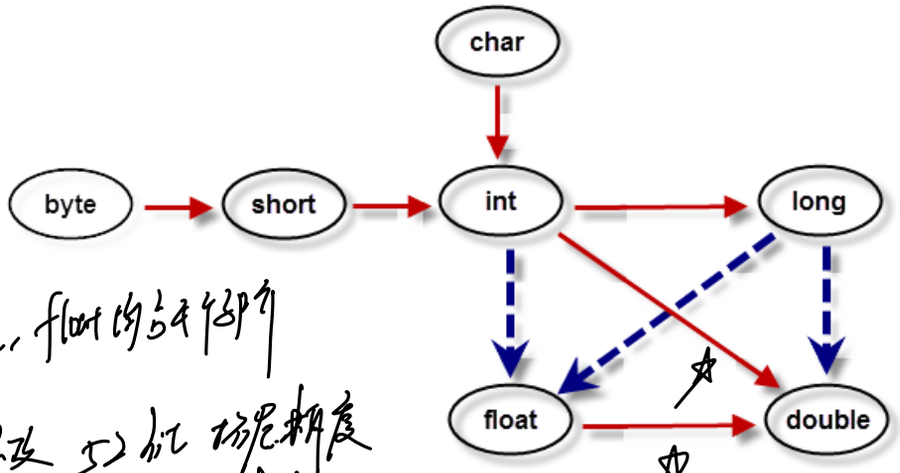
数据类型的转换



我们讲解了八种基本数据类型，除了 boolean 类型之外的七种类型是可以自动转化的。

自动类型转换

自动类型转换指的是容量小的数据类型可以自动转换为容量大的数据类型。如图 2-6 所示，黑色的实线表示无数据丢失的自动类型转换，而虚线表示在转换时可能会有精度的损失。



自动类型转换图

可以将整型常量直接赋值给 byte、 short、 char 等类型变量，而不需要进行强制类型转换，只要不超出其表数范围即可。

【示例 2-26】自动类型转换特例

```
short  b = 12;          //合法
short  b = 1234567;     //非法，1234567 超出了 short 的表数范围
```

强制类型转换

(*type*) var

强制类型转换，又称为造型（cast），用于强制转换数值的类型，可能损失精度。

【示例 2-27】强制类型转换

```
double x  = 3.94;
int nx = (int)x;    //值为 3
char c = 'a';
int d = c+1;
System.out.println(nx);
System.out.println(d);
System.out.println((char)d);
```

运行结果如图 2-7 所示。

```
3
98
b
```

当将一种类型强制转换成另一种类型，而又超出了目标类型的表数范围，就会被截断成为一个完全不同的值。

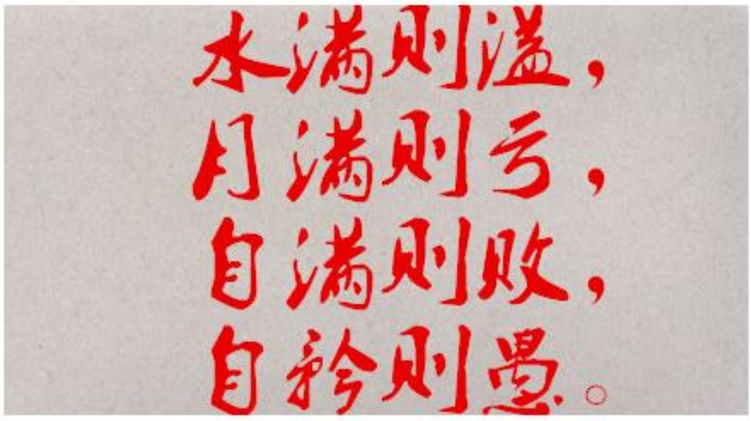
【示例 2-28】强制类型转换特例

```
int x = 300;
byte bx = (byte)x;      //值为 44
```

新手雷区

- 不能在布尔类型和任何数值类型之间做强制类型转换

基本类型转化时常见错误和问题



1. **操作比较大的数时，要留意是否溢出，尤其是整数操作时。**
2. L 和 l 的问题：
   (1) 不要命名名字为 l 的变量，字母 l 容易和数字 1 混淆。
   (2) long 类型使用大写 L，不要用小写 l。

【示例】类型转换常见问题一

```
int money = 1000000000; //10亿
int years = 20;
//返回的total是负数，超过了int的范围
int total = money*years;
System.out.println("total="+total);
//返回的total仍然是负数。默认是int，因此结果会转成int值，再转成long。但是已经发生//了数据丢失
long total1 = money*years;
System.out.println("total1="+total1);
//返回的total2正确:先将一个因子变成long，整个表达式发生提升。全部用long来计算。
long total2 = money*((long)years);
System.out.println("total2="+total2);
```

运行结果如图所示。

```
total=-1474836480
total1=-1474836480
total2=20000000000
```

【示例】类型转换常见问题二

```
int l = 2; //分不清是L还是1,
long a = 23451l;//建议使用大写L
```

```
System.out.println(l+1);
```

Scanner 处理键盘输入

Scanner 让程序和用户通过键盘交互

【示例】使用 Scanner 获取键盘输入

```
import java.util.Scanner;
public class Welcome2 {
    public static void main(String[ ] args) {
        Scanner scanner = new Scanner(System.in);
        // 将输入的一行付给 string1
        String string1 = scanner.nextLine();
        // 将输入单词到第一个空白符为止的字符串付给 string2
        String string2 = scanner.next();
        // 将输入的数字赋值给变量
        int a = scanner.nextInt();
        System.out.println("-----录入的信息如下-------");
        System.out.println(string1);
        System.out.println(string2);
        System.out.println(a * 10);
    }
}
```

运行结果如图 2-9 所示。

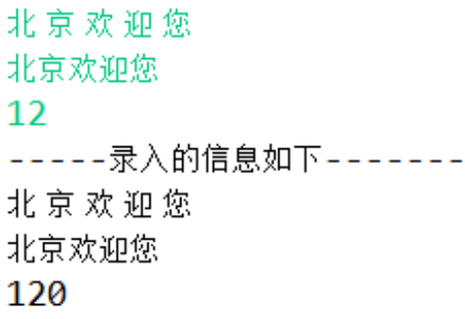


图 2-9 示例 2-31 运行效果图