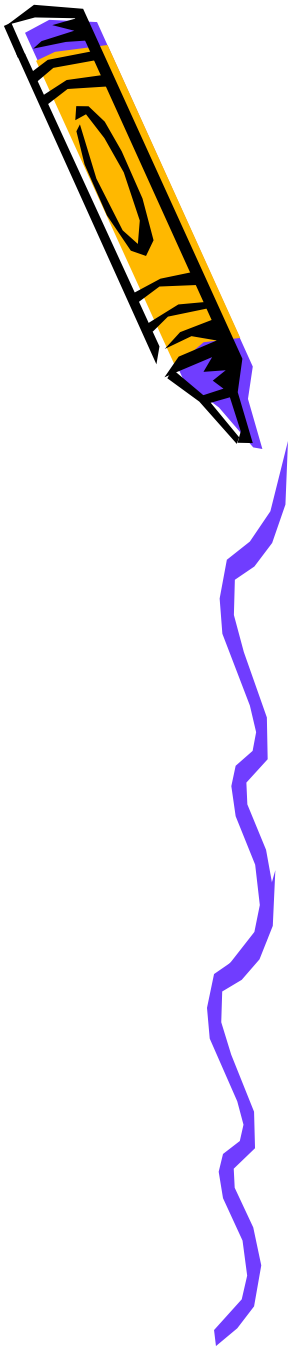
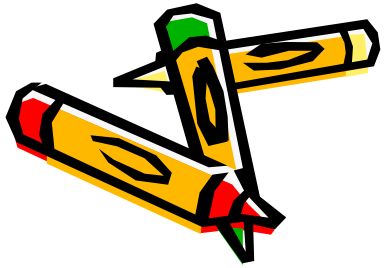
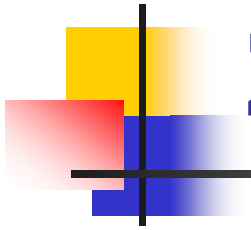
第2章 Java语言基础



- 2.1 语言成分
- 2.2 流程控制语句
- 2.3 方法



電子工業出版社
Publishing House of Electronics Industry



2.1 语言成分

1. 2.1.1 标识符与关键字
2. 2.1.2 基本数据类型
3. 2.1.3 变量与常量
4. 2.1.4 运算符与表达式

2.1.1 标识符与关键字

1. 关键字
2. 标识符
3. 分隔符
4. 注释
5. 程序书写风格

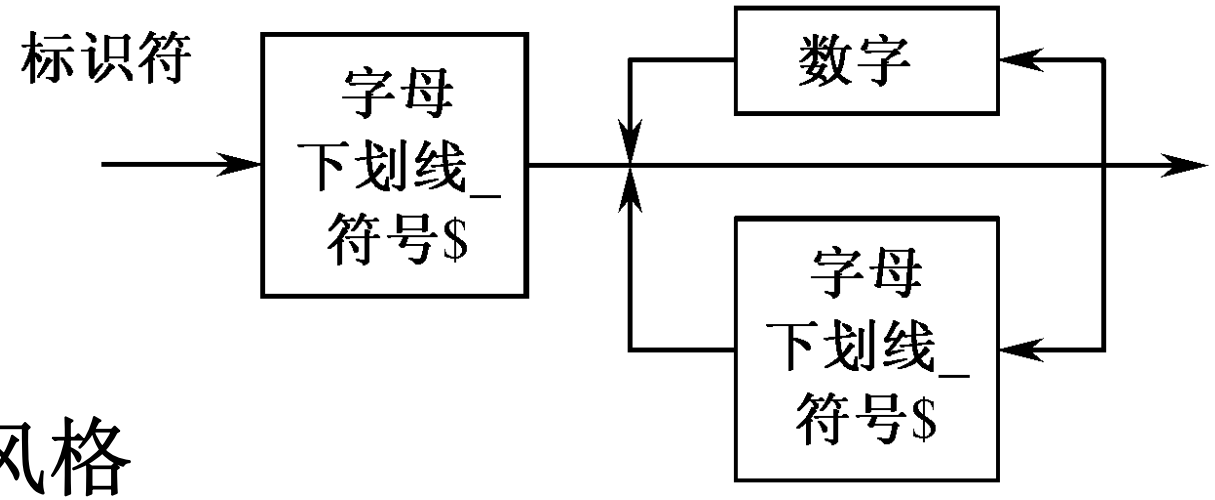


图2.1 Java标识符的语法图

2.1.1 标识符与关键字

Java关键字是**Java**语言保留供内部使用的，如**class**用于定义类。关键字也可以为保留字,它们的意思是一样的。不能使用关键字作为变量名或方法名。

abstract	assert	boolean	break	byte	case
catch	char(character)	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				

2.1.1 标识符与关键字

标识符用作给变量、类和方法命名。注意：

表示类名的标识符用大写字母开始。如：**Man, GoodMan**

表示方法和变量的标识符用小写字母开始，后面的描述性词以大写开始。**eat(),eatFood()**

Java 标识符有如下命名规则：

① 标识符必须以字母、下划线_ 、美元符**$**开头。

② 标识符其它部分可以是字母、下划线“_”、美元符“**$**”和数字的任意组合。

③ **Java** 标识符大小写敏感，且长度无限制。

④ 不可以是**Java**的关键字。

JAVA不采用通常语言使用的**ASCII**字符集，而是采用**16**位**Unicode**这样的标准的国际字符集。因此，这里的字母的含义：英文、汉字等等。**(**不建议大家使用汉字来定义标识符！**)**

2.1.1 标识符与关键字

合法的标识符：

```
int a = 3;
int _123 = 3;
int $12aa = 3;
int 变量1 = 55;
```

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
```

2.1.1 标识符与关键字

字符集简介

ISO8859-1 ——西欧字符集

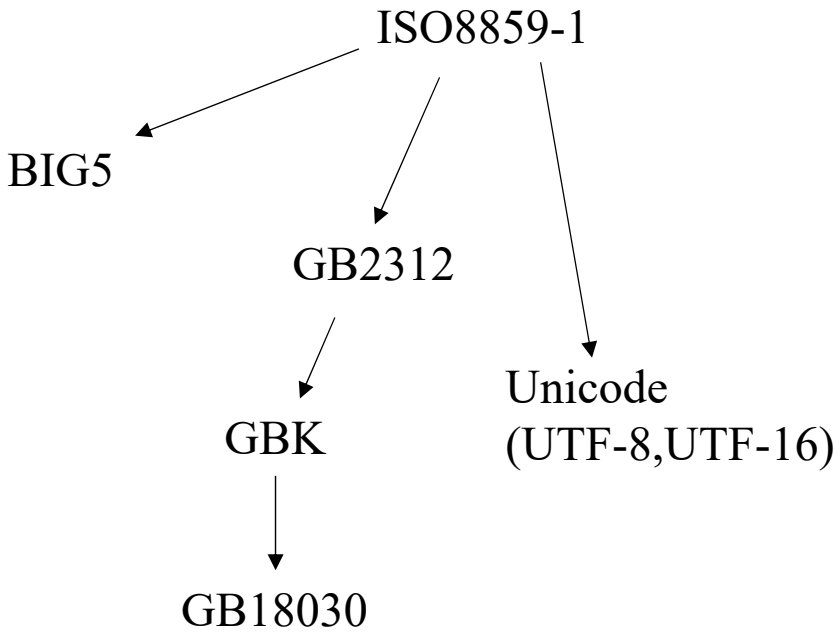
GB2312 ——大陆最早使用的简体中文字符集

GBK ——**GBK2312** 的扩展，可以表示繁体字

GB18030 ——最新**GBK**的扩展，中国所有非手持/嵌入式计算机系统的强制性实施标准。可以表示汉字、维吾尔文、藏文等中华民族字符。

BIG5 ——台湾的五大码，表示繁体字

Unicode ——国际通用字符集

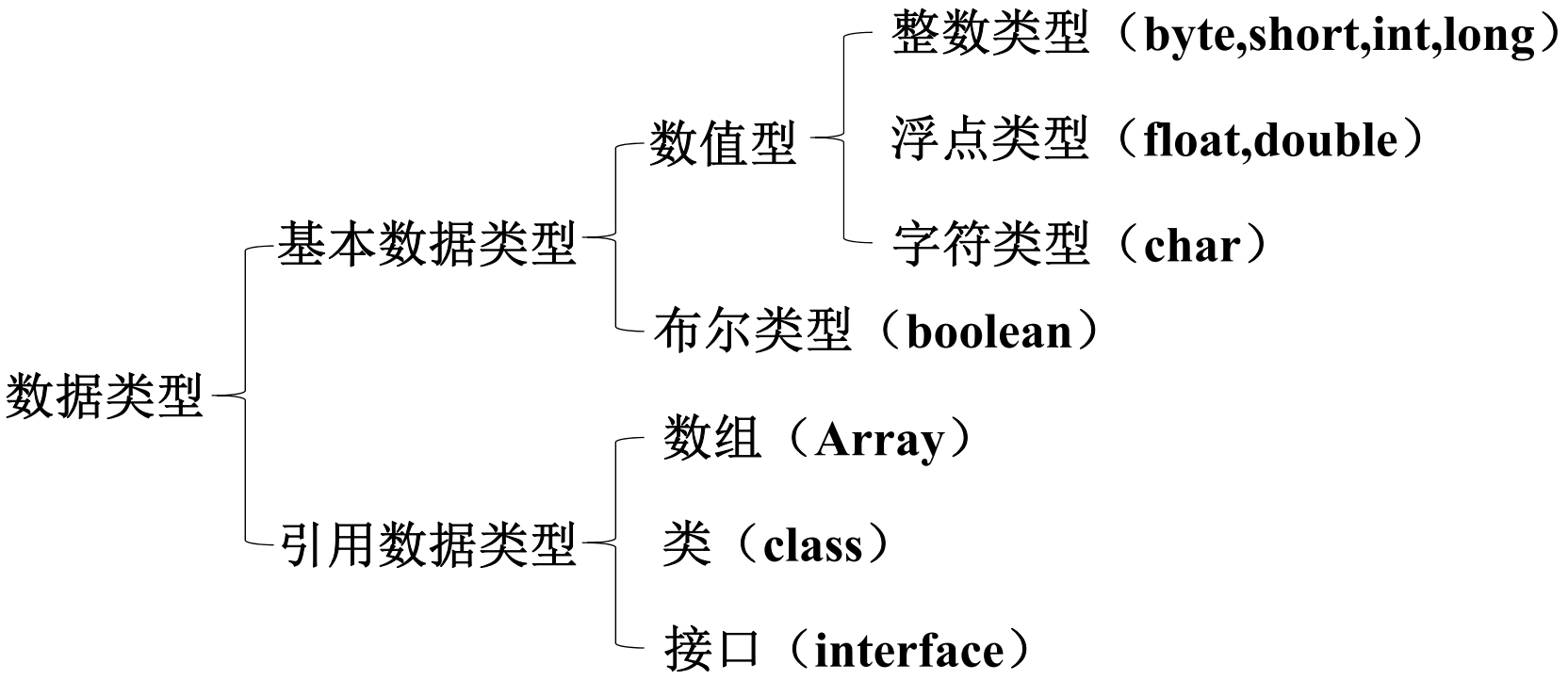


2.1.2 基本数据类型

1. 什么是数据类型（数值集合和之上的操作集合）
2. 数据类型分类
   - ① 基本数据类型：**8**种
   - ② 引用数据类型：**3**种，数组、类（**class**）和接口（**interface**）
3. 整数类型——**byte(1),short(2),int(4),long(8)**
4. 浮点数类型——**float(4)，double(8**字节**)**
5. 布尔类型——**boolean（true，false）**
6. 字符类型——**char（\u0000～\uFFFF ）**

2.1.2 基本数据类型

Java是一种强类型语言，每个变量都必须声明其类型



注：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！

2.1.2 基本数据类型

整型用于表示没有小数部分的数值，它允许是负数。

int整数占**32**位，与运行**Java**代码的机器无关，这正是**Java**程序具有很强移植能力的原因之一。于此相反，**C**和**C++**程序需要针对不同的处理器选择最有效的整形。

类型	占用存储空间	表数范围
byte	1字节	-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	$-2^{63} \sim 2^{63}-1$

2.1.2 基本数据类型

Java 语言整型常数的三种表示形式：

- 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 。

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’，如：

```
long a = 55555555;  //不出错，在Int表示的范围内(21亿内)。
long b = 55555555555;//不加l出错，已经超过int表示的范围。报错：
The literal 55555555555 of type int is out of range
```

2.1.2 基本数据类型

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，在很多情况下，**float**类型的精度很难满足需求。

② **double**表示这种类型的数值精度是**float**类型的两倍，又被称作双精度，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

- 十进制数形式，例如**:**

  3.14　　314.0　　0.314

- 科学记数法形式，如

  314e2　　314E2　　314E-2

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

④ **Float**类型的数值有一个后缀**F/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/d,** 以明确其为**double**类型

2.1.2 基本数据类型

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。

如果需要进行不产生舍入误差的精确数字计算，需要使用**BigDecimal**类。主要理由：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。

最好完全避免使用浮点数比较。

大数值：

Java.math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。

2.1.2 基本数据类型

浮点数使用总结：

① 默认是**double**类型。

② 浮点数存在舍入误差，很多数字不能精确表示。如果需要进行不产生舍入误差的精确数字计算，需要使用**BigDecimal**类。

③ 避免比较中使用浮点数。

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

2.1.2 基本数据类型

- 单引号用来表示字符常量。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。
- **char** 类型是在**0-65535**范围，运算时直接当作整数来运算。可以把**0-65535**之间的整数直接转型为**char**。
- 字符常量有两种表示方法：单引号方式；直接**Unicode**值表示，前缀是**\u**。
- **Unicode**码被设计用来处理各种语言的所有文字，可允许有**65536**个字符；标准**ASCII**码可允许有**128**个字符，是**Unicode**编码表中前**128**个字符。

```
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**）

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

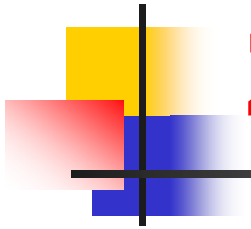
- **Java** 语言中还允许使用转义字符 '****' 来将其后的字符转变为其它的含义，

```
char c2 = '\n';  //代表换行符
```

2.1.2 基本数据类型

以后我们学的**String类**，其实是字符序列**(char sequence)**。

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



2.1.2 基本数据类型

boolean类型有两个值，**true**和**false,**不可以 **0** 或非 **0** 的整数替代 **true** 和 **false** ，这点和**C**语言不同。

boolean类型只有一位，注意不是一个字节！

boolean 类型用来判断逻辑条件，一般用于程序流程控制。

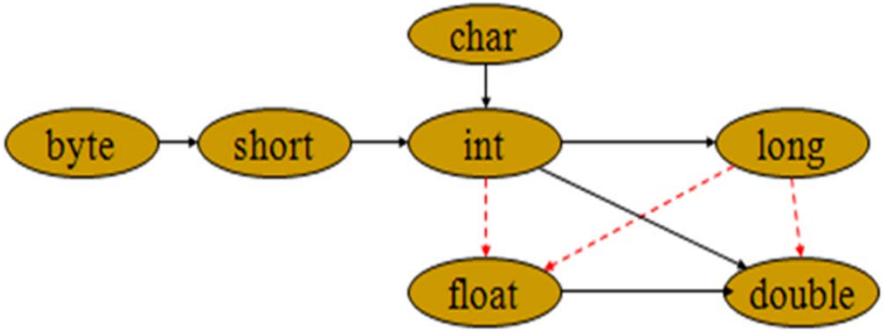
不要这样写：**if (is == true && done == false)** 。

对于任何程序员 **if (whether && !done)** 都不难理解，所以去掉所有的 **==fasle** 和 **==true**。

2.1.2 基本数据类型

- 自动类型转换：容量小的数据类型可以自动转换为容量大的数据类型（容量表示类型的范围，而不是字节数）。在图中，黑色的实线表示无数据丢失的自动类型转换，而红色的虚线表示在转换时可能会精度的损失。
- 特例：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

```
short  b = 12;                //合法
short  b = 1234567;           //非法
```



2.1.2 基本数据类型

- 强制类型转换（**Cast**）： 强制类型转换，又被称为造型，用于显式的转换一个数值的类型. 在有可能丢失信息的情况下进行的转换是通过造型来完成的，但可能造成精度降低或溢出。
- 强制类型转换的语法格式：“**(type)var**”，运算符”()”中的**type**表示将值**var**想要转换成的目标数据类型。

例如：

```
double x  = 3.14;
int nx = (int)x;   //值为3
char c = 'a';
int d = c+1;
System.out.println(d);
System.out.println((char)d);
```

2.1.2 基本数据类型

- 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示范围，就会被截断成为一个完全不同的值。

  例如： int x = 300;

  byte bx = (byte)x; //值为44

- 不能在布尔类型和任何数值类型之间做强制类型转换。
- 运行时表达式中的类型提升问题。所有的二元运算符（**+ - * / %**），都会有类型提升的问题。

2.1.2 基本数据类型

基本类型转化时常见错误和问题：

- 操作比较大的数时，要留意是否溢出，尤其是整数操作时。

```
int money = 1000000000;  //10亿
int years = 20;
int total = money*years;   //返回的是负数
long total1 = money*years;  //返回的仍然是负数。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 total2 = money*((long)years);   //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。
```

- **L和l 的问题：**

不要命名名字为**l**的变量，**long**类型使用大写**L**不要用小写。

```
int l = 2;
long a = 23451l;
System.out.println(l+1);
```

2.1.3 变量与常量

变量(variable)

我们通过变量来操纵存储空间中的数据，变量就是指代这个存储空间！空间位置是确定的，但是里面放置什么值不确定！

Java是一种强类型语言，每个变量都必须声明其类型。

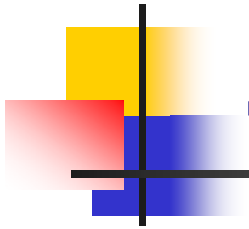
- **Java**变量是程序中最基本的存储单元，其要素包括变量名，变量类型和作用域。
- 变量在使用前必须对其声明，只有在变量声明以后，才能为其分配相应长度的存储单元，声明格式为：

  type varName [=value] [{,varName[=value]}] ；

  注意事项：

  每个变量都有类型，类型可以是基本类型，也可以是引用类型。

  变量名必须是合法的标识符。
- 变量声明是一条完整的语句，因此每一个声明都必须以分号结束



2.1.3 变量与常量

- 变量声明举例：

  double salary ； boolean done；

  long earthPopulation ； int age ；

- 可以在一行中声明多个变量：

  int i ,j; // both are integers

  不提倡这种风格，逐一声明每一个变量可以提高程序可读性。

- 可以将变量的声明和初始化放在同一行中，例如：

  int age = 18; float e = 2.718281828f;

2.1.3 变量与常量

变量可划分为：

- 局部变量（**lacal 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”**}**”为止。在使用前必须先声明和初始化（赋初值）。
- 实例变量（成员变量 **member variable**）：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，他会自动初始化成该类型的默认初始值（数值型变量初始化成**0**或**0.0**，字符型变量的初始化值是**16**位的**0**，布尔型默认是**false**）。
- 静态变量（类变量 **static variable**）：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，他会自动初始化成该类型的默认初始值（数值型变量初始化成**0**或**0.0**，字符型变量的初始化值是**16**位的**0**，布尔型默认是**false**）

2.1.3 变量与常量

常量(Constant)

- **CONSTANT: public static final double PI = 3.14;**
- 其值无法改变的量
- 只能被初始化一次（只要赋值则其值无法改变）
- 常量的命名通常使用全大写，如果单词较多则使用下划线进行分隔 如：**MAX_VALUE**

变量命名规则(规范)

- 所有变量、方法、类名：见名知意
- 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()**

2.1.3 变量与常量

1. 变量

① 变量声明

[修饰符] 类型 变量 [= 表达式] {, 变量[= 表达式]}

例如：**int i, j;**

② 变量参与的运算和操作

2. 最终变量

final int value;

value=100;

3. 常量

① 直接常量和符号常量

② 常量声明

final int MAX=10;

2.1.4 运算符与表达式

Java 语言支持如下运算符(operator)

- 算术运算符：+（正）、-（负）、++（自增）、--（自减） 、+（加）、-（减）、*（乘） 、/（除）、%（取余）
- 赋值运算符 =
- 关系运算符：>，<，>=，<=，==，!= instanceof
- 逻辑运算符：&（与）、|（或）、!（非）、∧（逻辑异或，相同为false,不同为true）、&&（短路与）、||（短路或）
- 位运算符：~（非）、&（与）、|（或）、^（异或）、<<（左移位）、>>（右移位）、>>>（无符号右移、忽略符号位扩展、0补最高位）
- 条件运算符 ？：
- 扩展赋值运算符:+=，-=，*=，/=

2.1.4 运算符与表达式

算术运算符：

二元运算符类型提升（有两个操作数）

- 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**shot,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**。

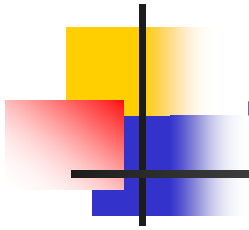
一元运算符（**++**，**--**）

如：**int a=3;**

如：**int b = a++； //**执行完后，**b=3** ，先给**b**赋值，再自增

如：**int c = ++a； //**执行完后，**c=5** ，先自增，再给**b**赋值

取模运算**--**其操作数可以为浮点数，一般使用整数。一般都是正整数运算（如果是负数参与取模运算，符号与被除数相同）。



2.1.4 运算符与表达式

注意：**java**中的乘幂处理：

int a = 3^2;　//java中不能这么处理，　^是异或符号。

double b = Math.pow(3, 2);

Math类提供了很多科学和工程计算需要的方法和常数。

2.1.4 运算符与表达式

逻辑运算符（只能操作布尔变量）

逻辑与：**&**只要有一个为**false**则为**false**，只有全为**true**才会是**true**（短路**&&**问题）

逻辑或：**|**只要有一个为**true**则为**true**，只有全为**false**才会是**false**（短路**||**问题）

逻辑非：！相反

逻辑与和逻辑或采用短路的方式：从左到右计算，如果确定值则不会再计算下去。

2.1.4 运算符与表达式

位运算符（按位操作）对进制进行操作（**0**是**false**，**1**为**true**）（计算机中用补码表示负数，并且有一定的计算方式；另外，用二进制的最高位表示符号，**0**表示正数、**1**表示负数。）

~ -- 取反

& -- 按位与：只要有一个为**0**则为**0**，只有全为**1**才会是**1**。

| -- 按位或：只要有一个为**1**则为**1**，只有全为**0**才会是**0**。

^ -- 按位异或：相异（两个值不一样），相同则为**0**，不同则为**1**。

<<：左移运算符（丢弃最高位，**0**补最低位），**>>**：右移运算符（符号位不变，左边补上符号位），**>>>**：无符号移位运算符（无符号右移、忽略符号位扩展、**0**补最高位。无符号右移中的符号位也跟着变，无符号的意思是将符号位当作数字位看待。）。右移一位相当于除**2**取商。左移一位相当于乘**2**。

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

2.1.4 运算符与表达式

扩展运算符

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

2.1.4 运算符与表达式

字符串连接符 +

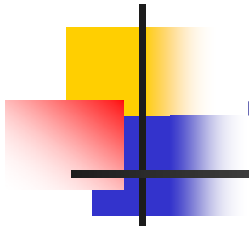
- “ + ”运算符两则的操作数中只要有一个字符串（**string**）类型，系统会自动将另一个操作数转换为字符串然后再进行连接。

三元运算符

- 语法格式： —— **x ? y : z**
- ——其中**x**为**boolean** 类型表达式，先计算**x**的值，若为**true** ，则整个三目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。
- 经常用来代替简单的**if –else** 判断！

运算符优先级

- 表达式里面优先级使用小括号来组织

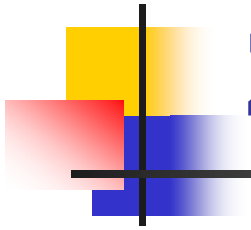


2.1.4 运算符与表达式

运算符

【例2.1】 求明天是星期几。

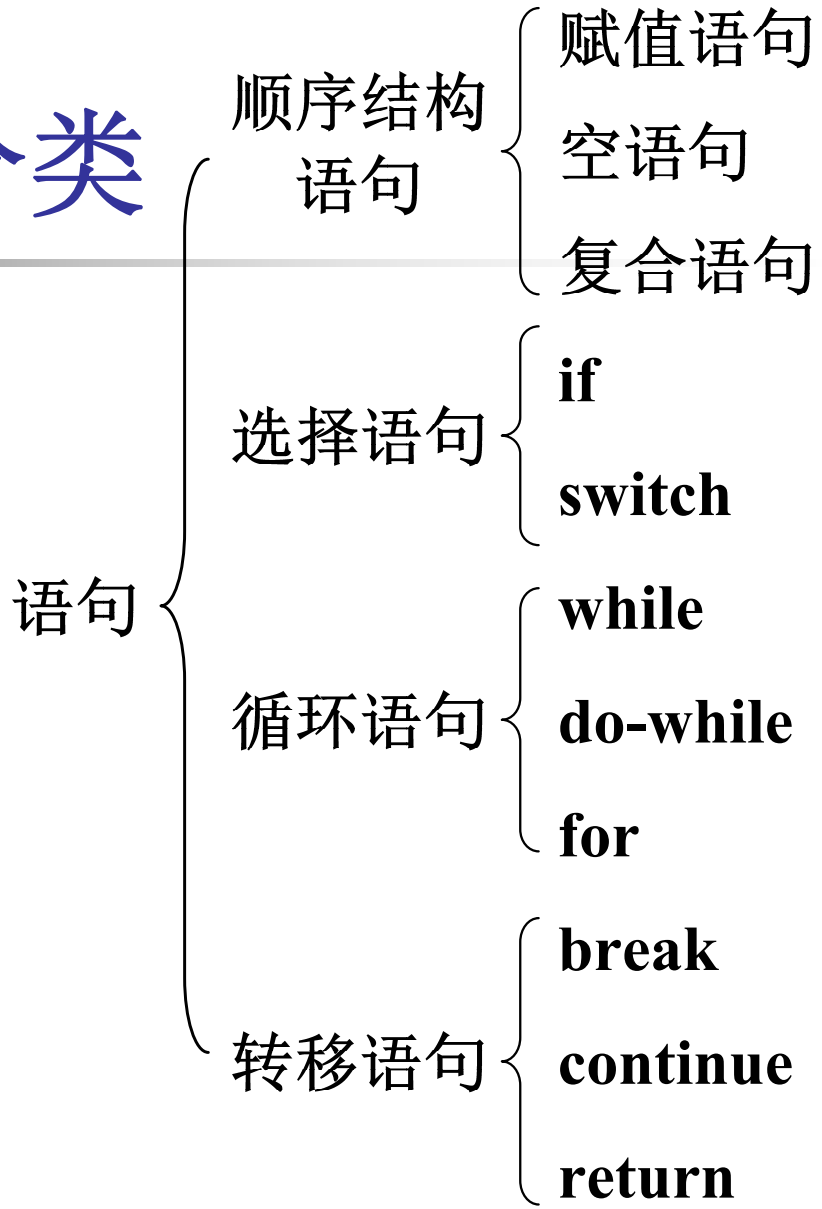
【例2.2】 判断一个年份是否为闰年（能被4整除而不能被100整除或者能被400整除）。



2.2 流程控制语句

1. 2.2.1 顺序结构语句
2. 2.2.2 选择语句
3. 2.2.3 循环语句
4. 2.2.4 转移语句

图2.6 语句分类


语句
顺序结构语句
赋值语句
空语句
复合语句
选择语句
if
switch
循环语句
while
do-while
for
转移语句
break
continue
return

2.2.1 顺序结构语句

1. 赋值语句

   变量=表达式;

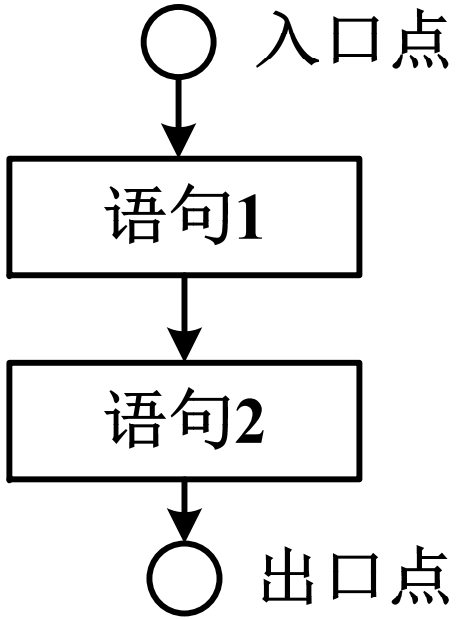
2. 空语句

3. 复合语句

```
{
    [变量声明或常量声明];
    语句序列;
}
```

注意：Java没有以下表达式语句。

表达式;



2.2.2 选择语句

1. **if语句**

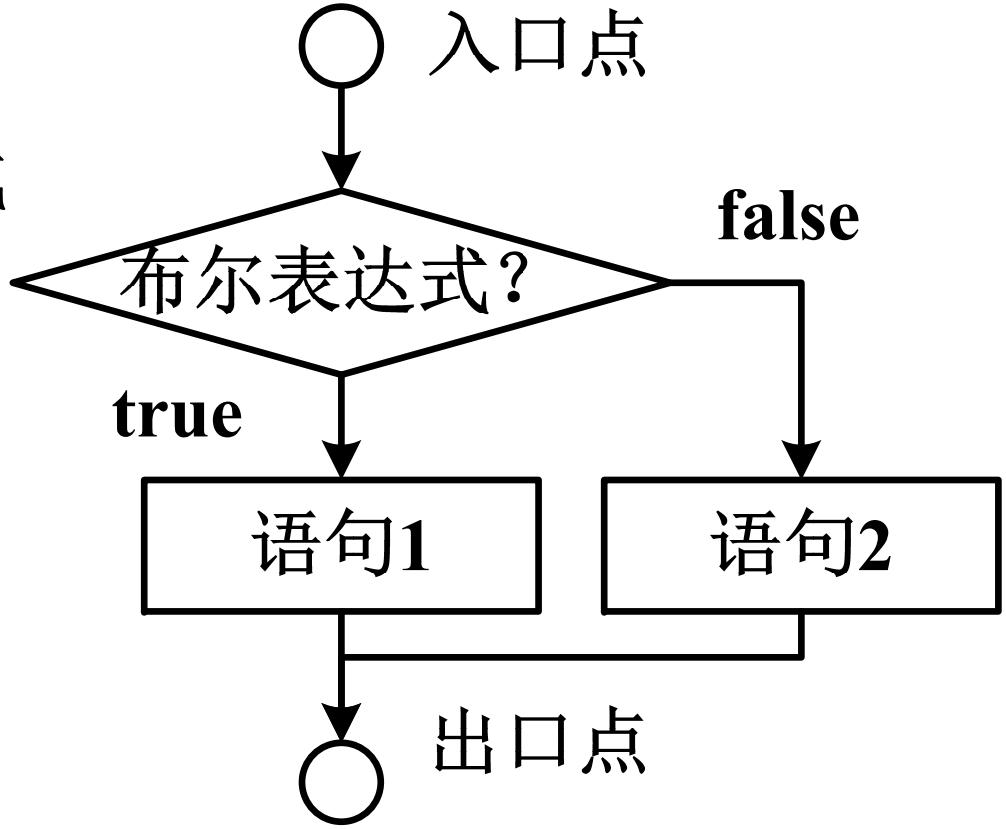
① **if语句的语法格式**

if **(** 布尔表达式 **)**

语句**1;**

[else

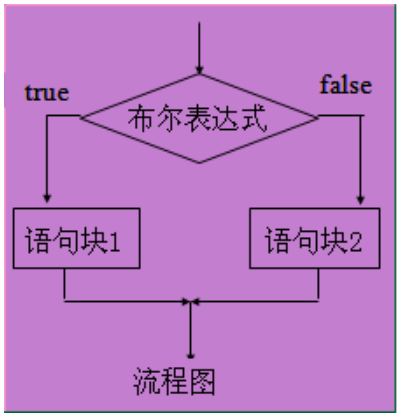
语句**2;]**



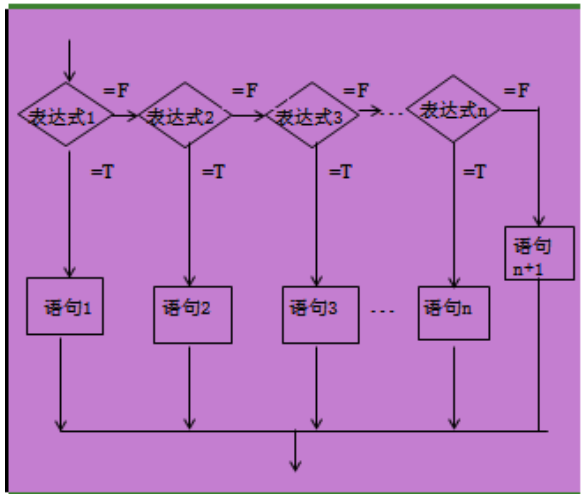
1. if语句

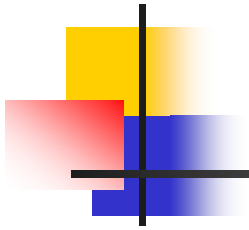
if单选择结构

if-else双选择结构



if-elseif-else多选择结构





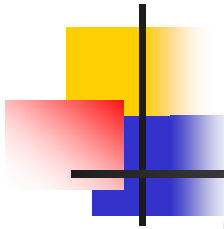
1. if语句

② 复合条件

```
if (n>=100) && (n<=999)
```

③ if语句嵌套

```
if (n>=100)
    if (n<=999)
```

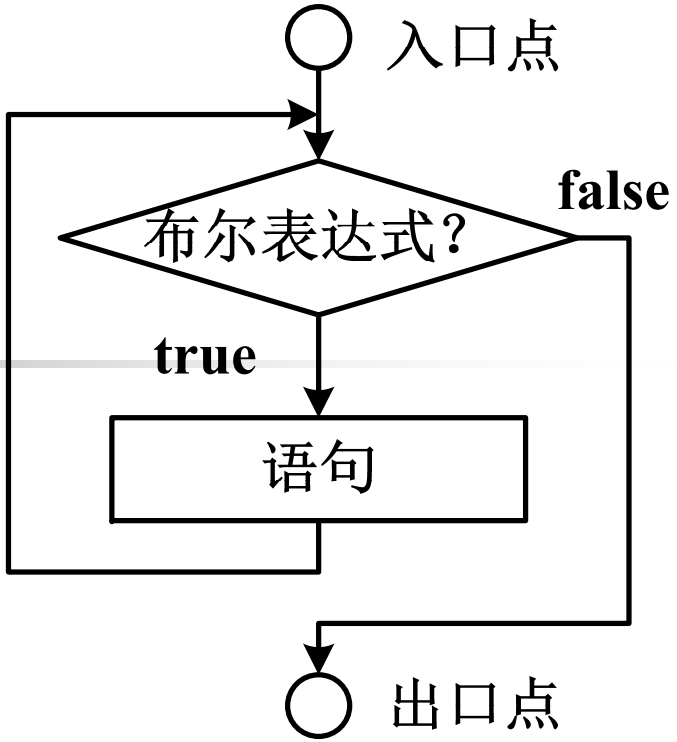


2. switch语句

```
switch (表达式)
{
   case 常量表达式1: 语句序列1;
                [break;]
   case 常量表达式2: 语句序列2;
                [break;]
   [default: 语句序列;]
}
```

注意点：**switch**语句会根据表达式的值从相匹配的**case**标签处开始执行，一直执行到**break**语句处或者是**switch**语句的末尾。与任一**case**值不匹配，则进入**default**语句(如果有的话) 。**case**子句中的值必须是常量，且所有**case**子句中的值应是不同的。**case**值可以是**int**或者自动可以转换为**int**的类型**byte**、**char**、**short**，以及枚举和字符串，理解**case**穿透现象，一般每个**case**后面都要加**break**，防止**case**穿透。

2.2.3 循环语句



1. **while**语句

① **while**语句语法

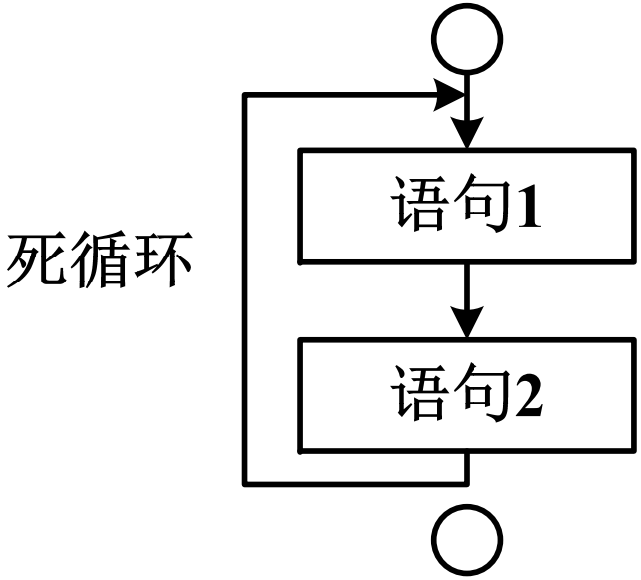
while (布尔表达式)

语句;

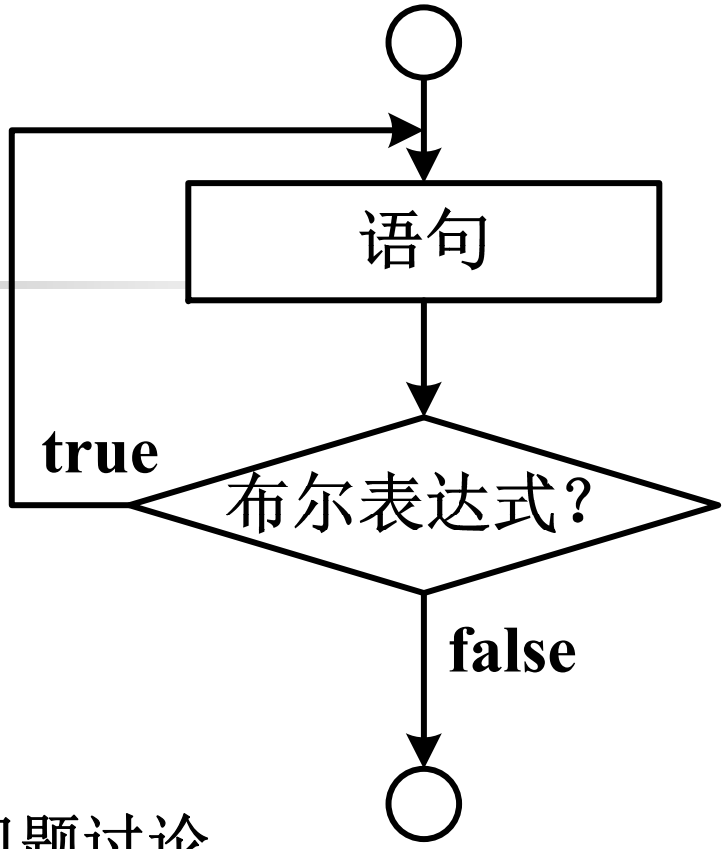
② **while**语句循环执行次数问题讨论

当型：当**P**条件成立时**(T),**反复执行**A,**直到**P**为“假”时才停止循环

语句中应有使循环趋向于结束的语句，否则会出现无限循环（死循环）。



2. do-while语句

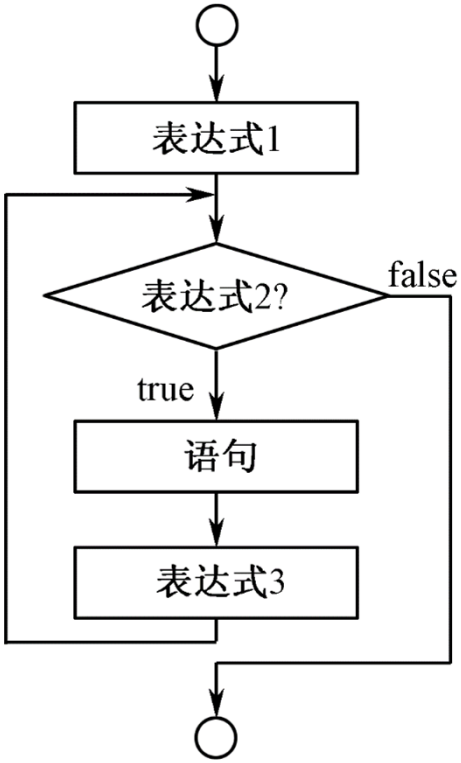


1. **do-while语句语法**

```
do
{
    语句;
} while (布尔表达式);
```

2. **do-while语句循环执行次数问题讨论**
3. 直到型：先执行A, 再判断P,若为T,再执行A,如此反复,直到P为F

3. for语句



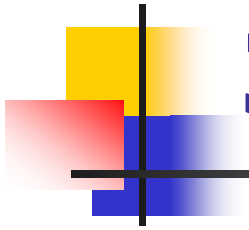
1. **for**语句语法

   for (表达式**1; **表达式**2; **表达式**3)**

   语句**;**

2. **for**循环在第一次反复之前要进行初始化。随后，它会进行条件测试，而且在每一次反复的时候，进行某种形式的“步进”。**for**循环在执行条件测试后，先执行程序部分，再执行步进。在**for**循环语句中初始化部分声明的变量，其作用域为整个**for**循环体。

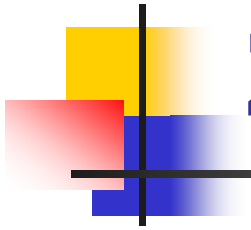
【例**2.4**】 求一个日期（年月日）对应的是星期几。



3. for语句

课堂作业：

1. 用**while** 和**for** 循环分别计算**200**以内奇数和偶数的和，并输出。
2. 用**while** 和**for**循环输出**1—500** 之间能被**5**整除的数，且每行输出三个。
3. 打印输出九九乘法表。



2.2.4 转移语句

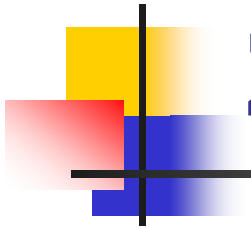
1. **return语句**

   return [返回值];

2. **break语句和continue语句**

   ① **break;** //退出循环

   ② **continue;** //继续循环



2.2.4 转移语句

- 在任何循环语句的主体部分，均可用**break**控制循环的流程。**break**用于强行退出循环，不执行循环中剩余的语句。**(break**语句也在**switch**语句中使用**)**。
- **continue** 语句用于在循环语句体中，用于终止某次循环过程，即跳过循环体中尚未执行的语句，接着进行下一次是否执行循环的判定。

2.2.4 转移语句

带标签的**continue**语句：

- **goto**关键字很早就在程序设计语言中出现。尽管**goto**仍是**Java**的一个保留字，但并未在语言中得到正式使用；**Java**没有**goto**。然而，在**break**和**continue**这两个关键字的身上，我们仍然能看出一些**goto**的影子**---**带标签的**break**和**continue**。
- **“**标签**”**是指后面跟一个冒号的标识符，例如：**label:**
- 对**Java**来说唯一用到标签的地方是在循环语句之前。而在循环之前设置标签的唯一理由是：我们希望在其中嵌套另一个循环，由于**break**和**continue**关键字通常只中断当前循环，但若随同标签使用，它们就会中断到存在标签的地方。

2.3 方法

1. 方法声明

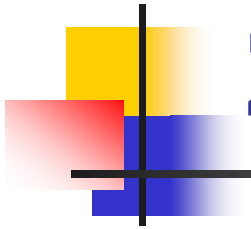
```
[修饰符...] 返回值类型 方法名([参数列表])
{
    语句序列;
    [return [返回值]];
}
```

2. 方法调用

```
方法([参数列表])
```

3. 声明main方法

```
public static void main(String args[])
```



2.3 方法

递归方法：**A**方法调用**B**方法，我们很容易理解！递归就是：**A**方法调用**A**方法！就是自己调用自己，因此我们在设计递归算法时，一定要指明什么时候自己不调用自己。否则，就是个死循环！递归结构包括两个部分：

- 定义递归头。解答：什么时候不调用自身方法。如果没有头，将陷入死循环；
- 递归体。解答：什么时候需要调用自身方法。

2.3 方法

求n!的递归方法。

f(n) = n*f(n-1)

5!=5*4*3*2*1=5*4!

0! = 1 或 1! = 1

n! = n*(n-1)!

$$n!=\begin{cases}1 & n=0,1\\ n\times(n-1)! & n\geq 2\end{cases}$$