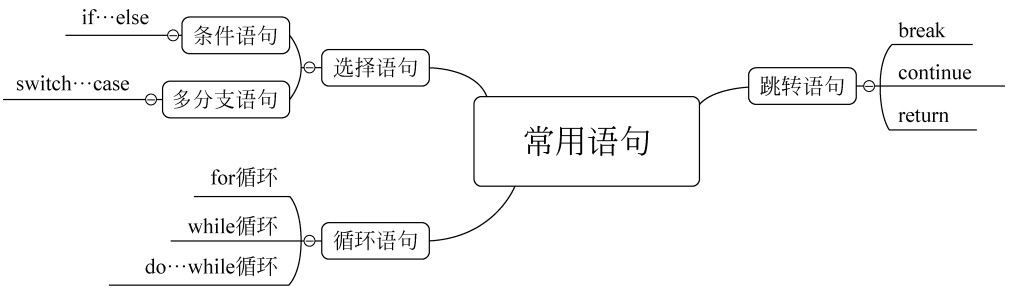
第 3 章　运算符、表达式和语句

本章导图：



主要内容：

- 运算符与表达式。
- 选择控制。
- 循环控制。
- 跳转控制。

难点：

- 循环语句与跳转语句的掌握。

3.1　运　算　符

运算符就是用来进行运算的符号。运算符通常必须与操作数一起使用组成 Java 的表达式才有意义。Java 的运算符有多种，分为赋值运算符、算术运算符、关系运算符和逻辑运算符等。运算符按操作数的个数又可为单目运算符（如++、－）、双目运算符（如＋、>）和三目运算符。

3.1.1　赋值运算符与赋值表达式

赋值运算符为“＝”，用于将运算符右边表达式的值赋给左边的变量。赋值运算符的使用格式如下。

```
变量 = 表达式;
```

下面是一些简单的赋值运算示例。

```
j = 1;
```

```
k = j;
l = i + j * 4;
```

3.1.2 算术运算符与算术表达式

算术运算符用于完成算术运算，有双目运算符、单目运算符两种。

1. 双目算术运算符

双目算术运算符如表 3.1 所示。

表 3.1 双目运算符

运 算 符	运 算	例 子	解 释
+	加	a+b	求 a 与 b 相加的和
−	减	a−b	求 a 与 b 相减的差
*	乘	a * b	求 a 与 b 相乘的积
/	除	a/b	求 a 与 b 相除的商
%	求余	a%b	求 a 除以 b 所得的余数

Java 对加运算符进行了扩展，使它能够进行字符串的连接，如"abc"+"de"，得到字符串"abcde"。与 C、C++不同，对取模运算符%来说，其操作数可以为浮点数，如 37.2%10=7.2。

2. 单目算术运算符

单目算术运算符如表 3.2 所示。

表 3.2 单目算术运算符

运 算 符	运 算	例 子	解 释
++	自增 1	a++或++a	a=a+1
--	自减 1	a--或--a	a=a−1
−	求相反数	−a	a=−a

需要注意的是，“++”和“--”在变量左右使用的区别。例如，i++与++i 是不一样的，i++发生在使用 i 之后，使 i 的值加 1；++i 发生在使用 i 之前，使 i 的值加 1。i--与--i 类似。

例如：

```
int x = 2;
int y = (++x) * 3;
```

运行结果是 x=3；y=9。

又例如：

```
int x = 2;
int y = (x++) * 3;
```

运行结果是 x=3，y=6。

这是为什么呢？因为第一个例子中是 x 已经等于 3 后再算 y，而后一个例子中则是先用 x=2 算出 y 后，再算 x，因为++符号在后面，这就是++x 和 x++的区别。下面的例子说明了算术运算符的使用。

例 3.1

Example3_1.java

```
public class Example3_1{
public static void main(String[] args) {
        // TODO code application logic here
        int a = 5 + 4;              //a = 9
        int b = a * 2;              //b = 18
        int c = b / 4;              //c = 4
        int d = b - c;              //d = 14
        int e = -d;                 //e = -14
        int f = e % 4;              //f = -2
        double g = 18.4;
        double h = g % 4;           //h = 2.4
        int i = 3;
        int j = i++;                //i = 4,j = 3
        int k = ++i;                //i = 5,k = 5
        System.out.print("a = " + a);
        System.out.println(" b = " + b);
        System.out.print("c = " + c);
        System.out.println(" d = " + d);
        System.out.print("e = " + e);
        System.out.println(" f = " + f);
        System.out.print("g = " + g);
        System.out.println(" h = " + h);
        System.out.print("i = " + i);
        System.out.println(" j = " + j);
        System.out.println("k = " + k);
    }
}
```

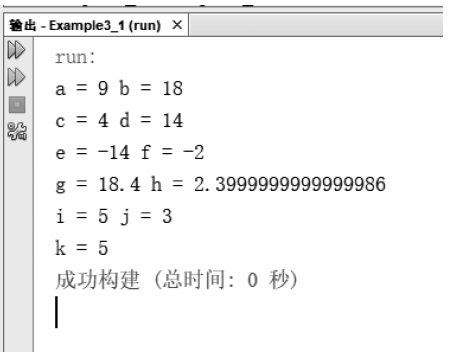


图 3.1　例 3.1 的程序运行结果

例 3.1 的程序运行结果如图 3.1 所示。

3.1.3　关系运算符与关系表达式

关系运算符用来比较两个值，比较结果是布尔类型的值 true 或 false。关系运算符都是二元运算符，如表 3.3 所示。

表 3.3　关系运算符

运　算　符	运　　算	例　　子	解　　释
==	等于	a==b	a 等于 b 为真，否则为假
!=	不等于	a!=b	a 不等于 b 为真，否则为假
>	大于	a>b	a 大于 b 为真，否则为假
<	小于	a<b	a 小于 b 为真，否则为假
>=	大于或等于	a>=b	a 大于或等于 b 为真，否则为假
<=	小于或等于	a<=b	a 小于或等于 b 为真，否则为假

关系运算的结果是布尔值，只有“真”和“假”两种取值，例如：

```
int x = 5,y = 7;
boolean b = (x == y);
```

则 b 的值是 false。

Java 中，任何数据类型的数据(包括基本类型和组合类型)都可以通过==或!=来比较是否相等(这与 C/C++不同)。关系运算的结果返回 true 或 false，而不是 C/C++中的 1 或 0。关系运算符常与逻辑运算符一起使用，作为流控制语句的判断条件，如 if(a > b&&b==c)。

3.1.4 逻辑运算符与逻辑表达式

布尔逻辑运算符用来进行布尔逻辑运算，逻辑运算是针对布尔型数据进行的运算，运算的结果仍然是布尔型量。常用的布尔逻辑运算符如表 3.4 所示。

表 3.4 常用的逻辑运算符

运 算 符	运 算	例 子	解 释
&	与	x&y	x,y 都为真时结果才为真
|	或	x|y	x,y 都为假时结果才为假
!	非	! x	x 为真时结果为假，x 为假时结果为真
^	异或	x^y	x,y 都为真或都为假时结果为假
&&	简洁与(条件与)	x&&y	x,y 都为真时结果才为真
||	简洁或	x||y	x,y 都为假时结果才为假

对于布尔逻辑运算，先求出运算符左边的表达式的值，对“或”运算，如果为 true，则整个表达式的结果为 true，不必对运算符右边的表达式再进行运算。同样，对“与”运算，如果左边表达式的值为 false，则不必对右边的表达式求值，整个表达式的结果为 false。

例 3.2 关系运算符和布尔逻辑运算符的使用。

Example3_2.java

```
public class Example3_2{
        public static void main(String args[]){
            int a = 25,b = 3;
         boolean d = a < b;              //d = false
         System.out.println("a < b = " + d);
            int e = 3;
            if(e!= 0 && a/e > 5)
            System.out.println("a/e = " + a/e);
         int f = 0;
         if(f!= 0 && a/f > 5)
             System.out.println("a/f = " + a/f);
         else
             System.out.println("f = " + f);
    }
 }
```

例 3.2 的程序运行结果如图 3.2 所示。

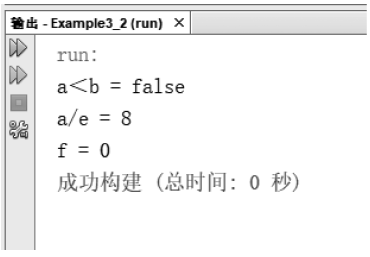


图 3.2 例 3.2 的程序运行结果

注意：例 3.2 中，第二个 if 语句在运行时不会发生除 0 溢出的错，因为 e!=0 为 false，所以就不需要对 a/e 进行运算了。

3.1.5 位运算符

位运算是对操作数以二进制位为单位进行的操作和运算，位运算的操作数和结果都是整形变量。Java 中提供了如表 3.5 所示的位运算符。

表 3.5 常见的位运算符

运 算 符	运 算	例 子	解 释
～	位反	～x	将 x 逐位取反
&	位与	x&y	x、y 逐位进行与操作
|	位或	x|y	x、y 逐位进行或操作
^	位异或	x^y	x、y 逐位进行，相同取 0，相异取 1
<<	左移	x<<y	x 向左移动，位数是 y
>>	右移	x>>y	x 向右移动，位数是 y
>>>	不带符号右移	x>>>y	x 向右移动，位数是 y，空位补 0

位运算符中，除“～”以外，其余均为二元运算符。操作数只能为整型和字符型数据。

3.1.6 三目运算符和复杂运算符

Java 除了一般的运算符外，还有三目运算符和复杂运算符。

Java 中的三元运算符与 C 语言中的三元运算符完全相同，使用格式为：

```
x?y:z;
```

运算时先计算 x 的值，若 x 为真，整个表达式的结果为表达式 y 的值；若 x 为假，则整个表达式的值为表达式 z 的值。

例如：

```
int x = 5, y = 8, z = 2;
int k = x < 3?y:z;                    //因为 x < 3,所以 k = 2
int j = x > 0?x: - x;                 //y 的值始终为 x 的绝对值
```

复杂运算符是在先进行某种运算后，再把运算结果赋给变量。表 3.6 列举出了复杂运算符及相关运算实例。

表 3.6 复杂运算符

运 算 符	实 例	解 释
＋＝	x＋＝a	x＝x＋a
－＝	x－＝a	x＝x－a
＊＝	x＊＝a	x＝x＊a
/＝	x/＝a	x＝x/a
%＝	x%＝a	x＝x%a
&＝	x&＝a	x＝x&a
|＝	x|＝a	x＝x|a
^＝	x^＝a	x＝x^a

续表

运 算 符	实 例	解 释
<<=	x<<=a	x=x<<a
>>=	x>>=a	x=x>>a
<<<=	x<<<=a	x=x<<<a

3.1.7 instanceof 运算符

在强制转换前，可使用 instanceof 运算符判断是否可以成功转换，从而避免异常产生。instanceof 运算符前面使用对象的引用，后面是一个类或接口。返回一个逻辑类型值，指出对象是否是特定类的一个实例。instanceof 运算符用法：

```
<引用变量> instanceof <类名称/接口名称>
```

如果引用变量是类名称/接口名称的一个实例，则 instanceof 运算符返回 true。如果引用变量不是类名称/接口名称的一个实例，或者引用变量是 null，则返回 false。

3.1.8 运算符优先级

运算符的优先级决定了表达式中不同运算执行的先后顺序，而运算符的结合性决定了并列的相同运算的先后执行顺序。表 3.7 列出了 Java 语言中主要运算符的优先级和结合性。

表 3.7 运算符优先级

优 先 级	运 算 符	描 述	结 合 性
1	. [ ] ( )	域、数组、括号	从左至右
2	++ -- - ! ~	单目运算符	从右至左
3	* / %	乘、除、取余	从左至右
4	+ -	加、减	从左至右
5	<< >> >>>	位运算	从左至右
6	< <= > >=	逻辑运算	从左至右
7	== !=	逻辑运算	从左至右
8	&	按位与	从左至右
9	^	按位异或	从左至右
10	|	按位或	从左至右
11	&&	逻辑与	从左至右
12	||	逻辑或	从左至右
13	?:	条件运算符	从右至左
14	= *= /= %= += -= <<= >>= >>>= &= ^= |=	赋值运算符	从右至左

3.2 选择(条件)控制

条件分支语句提供了一种根据条件判断的结果进行程序流程控制的方法,使得程序的执行可以跳过一些语句不执行,而转去执行特定的语句。

3.2.1 条件语句(if…else)

if…else 语句根据判定条件的真假来执行两种操作中的一种,它的格式为:

```
if (条件表达式)
语句块 1;
[else
语句块 2;]
```

语法说明:

(1) 若条件表达式的值为 true,则程序执行语句块 1,否则执行语句块 2。

(2) 每个单一的语句后都必须有分号。

(3) 语句块 1、2 可以为复合语句,这时要用大括号{}括起。建议对单一的语句也用大括号括起来,这样程序的可读性强,可以在其中添加新的语句,有利于程序的扩充。{}外面不加分号。

(4) else 子句是任选的。

(5) else 子句不能单独作为语句使用,它必须和 if 配对使用。else 总是与离它最近的 if 配对。可以通过使用大括号{}来改变配对关系。if…else 语句可以嵌套使用。

下面的例子中,嵌套使用了 if…else 语句。

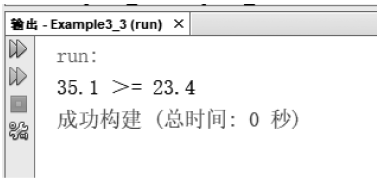
```
if(testscore>=90){
grade='A';
}else if(testscore>= 80){
grade='B';
}else if(testscore>=70){
grade='C';
}else if(testscore>=60){
grade='D';
}else{
grade='F';
}
```

例 3.3 比较两个数的大小,并按从小到大的次序输出。

Example3_3.java

```
public class Example3_3{
  public static void main(String args[]){
            double d1=23.4;
      double d2=35.1;
            if(d2>=d1)
```

```
            System.out.println(d2 + " >= " + d1);
        else
            System.out.println(d1 + " >= " + d2);
    }
}
```



例 3.3 的程序运行结果如图 3.3 所示。

图 3.3　例 3.3 的程序运行结果

3.2.2　多分支语句（switch…case）

switch 语句又称为多分支语句，它根据表达式的值来选择执行多个操作中的一个，它的一般格式如下。

```
switch (表达式){
case 判断值 1: 语句块 1; break;
case 判断值 2: 语句块 2;break;
…
case 判断值 n:语句块 n;break;
[default:语句块 n + 1; ]
}
```

语法说明：

（1）switch 语句首先计算表达式，根据表达式的返回值与每个 case 子句中的判断值进行比较，如果相等，则执行该 case 子句后的语句块。

（2）表达式的类型必须是 byte、short、int 型和字符型，case 子句中的判断值必须为常量，类型与表达式的类型相同，所有 case 子句中的判断值的值是不同的。

（3）default 子句是可选的。当表达式的值与任一 case 子句中的判断值都不相等时，程序执行 default 后面的语句块。如果表达式的值与任一 case 子句中的判断值都不相等且没有 default 子句时，则程序不做任何操作，而是直接跳出 switch 语句。

（4）break 语句用来在执行完一个 case 分支后，跳出 switch 语句，即终止 switch 语句的执行。因为 case 子句只是起到一个标号的作用，用来查找匹配的入口值，从此处开始执行，对后面的 case 子句不再进行匹配，而是直接执行其后的语句序列，因此应该在每个 case 分支后，用 break 来终止后面的 case 分支语句的执行。

在一些特殊情况下，多个不同的 case 值要执行一组相同的操作，这时可以不用 break。

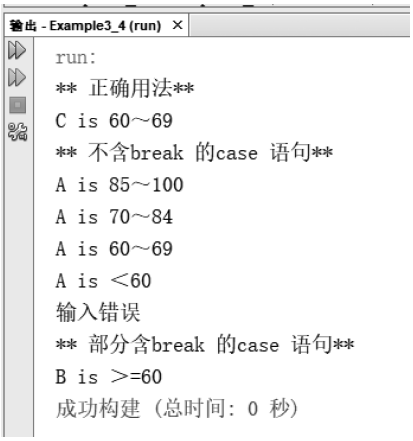
（5）switch 语句的功能可以用 if…else 来实现，但在某些情况下，使用 switch 语句更简练，可读性更强，且程序的执行效率更高。

例 3.4　根据考试成绩的等级打印出百分制分数段。从本例中可以看到 break 语句的作用。

Example3_4.java

```
public class Example3_4{
 public static void main( String args[] ){
      System.out.println(" ** 正确用法 ** ");
          char grade = 'C';              //正确用法
      switch( grade ){
                case 'A':
                    System.out.println(grade + " is 85～100");
                break;
```

```
                case 'B':
                System.out.println(grade + " is 70～84");
                break;
            case 'C':
            System.out.println(grade + " is 60～69");
            break;
            case 'D':
            System.out.println(grade + " is < 60");
            break;
            default:
                System.out.println("输入错误");
            }
            System.out.println(" ** 不含 break 的 case 语句 ** ");
            grade = 'A';          //一个不含 break 的 case 语句
        switch( grade ){
            case 'A':
                System.out.println(grade + " is 85～100");
            case 'B':
            System.out.println(grade + " is 70～84");
            case 'C':
                System.out.println(grade + " is 60～69");
            case 'D':
                System.out.println(grade + " is < 60");
        default:
            System.out.println("输入错误");
            }
        System.out.println(" ** 部分含 break 的 case 语句 ** ");
        grade = 'B';              //部分含 break 的 case 语句
        switch( grade ){
          case 'A':
          case 'B':
          case 'C':
      System.out.println(grade + " is >= 60");
            break;
            case 'D':
      System.out.println(grade + " is < 60");
            break;
            default:
            System.out.println("输入错误");
        }
    }
}
```



例 3.4 的程序运行结果如图 3.4 所示。

图 3.4　例 3.4 的程序运行结果

3.3　循 环 控 制

循环语句的作用是反复执行一段代码，直到满足终止循环的条件为止。

一个循环语句一般应包括以下四部分内容。

循环初始化部分，用来设置循环的一些初始条件，使计数器清零等。

循环体部分，这是反复循环的一段代码，可以是单一的一条语句，也可以是复合语句。

迭代部分，这是在当前循环结束后，下一次循环开始前时执行的语句，常常用来使循环计数器加 1 或减 1。

终止部分，通常是一个条件表达式，每一次循环要对该表达式求值，以验证是否满足循环终止条件。

Java 提供的循环语句有 for 语句、while 语句、do…while 语句等。

3.3.1 for 语句

for 语句是最常使用的一种循环语句，它的一般格式为：

```
for (初始化表达式; 终止表达式; 迭代表达式){
循环体}
```

语法说明：

(1) for 语句执行时，首先执行初始化操作，然后判断终止条件是否满足，如果满足，则执行循环体中的语句，最后执行迭代部分。完成一次循环后，重新判断终止条件。

(2) 可以在 for 语句的初始化部分声明一个变量，它的作用域为整个 for 语句。

(3) for 语句通常用来执行循环次数确定的情况(如对数组元素进行操作)，也可以根据循环结束条件执行循环次数不确定的情况。

(4) 在初始化部分和迭代部分可以使用逗号语句，来进行多个操作。逗号语句是用逗号分隔的语句序列。例如：

```
for(i = 0,j = 10;i < j;i++,j -- ){
…
}
```

(5) 初始化、终止以及迭代部分都可以为空语句(但分号不能省)，三者均为空的时候，相当于一个无限循环。

3.3.2 while 语句

while 语句实现当型循环，即先判断再决定是否执行循环体。while 语句的一般格式为：

```
while (条件表达式){
循环体
}
```

语法说明：

(1) 当条件表达式的值为 true 时，循环执行大括号中的语句。

(2) while 语句首先计算终止条件，当条件满足时，才去执行循环体中的语句，这是当型循环的特点。

例 3.5 一个双重循环的例子。程序的功能是在屏幕上输出一个三角形。

Example3_5.java

```
public class Example3_5 {
    public static void main(String[] args) {
```

```
        int i = 1, j, n = 5;
        while (i <= n) {
            for (j = 1; j <= i * 2 - 1; j++) {
                System.out.print("* ");
            }
            i++;
            System.out.println();
        }
    }
}
```

例 3.5 的程序运行结果如图 3.5 所示。

程序的第一重循环即 while 循环，用于控制构成三角形的行数。第二重 for 循环用于控制每一行 * 的个数，它的循环次数要随着第一重的循环变量 i 变化。当 i=1 时，它要循环 i×2−1 遍，也就是 1 遍，当 i=5 时，它要循环 5×2−1=9 遍。

另外，程序打印出来的三角形高度不是固定的，可以通过修改 n 的值而使得三角形具有不同的高度。

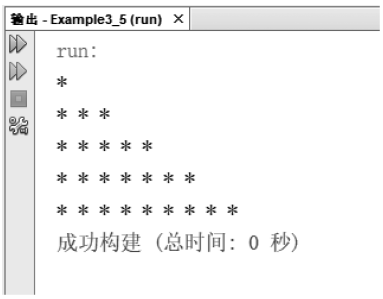


图 3.5　例 3.5 的程序运行结果

3.3.3　do…while 语句

do…while 语句的判断过程正好与 while 语句相反，它是先执行循环体再判断决定是否执行下一次循环。do…while 语句一般格式为：

```
do{
循环体
}while (条件表达式);
```

语法说明：

(1) do…while 语句首先执行循环体，然后计算终止条件，若结果为 true，则循环执行大括号中的语句，直到布尔表达式的结果为 false。

(2) 与 while 语句不同的是，do…while 语句的循环体至少执行一次，这是它的特点。

例 3.6　用 while、do…while 和 for 语句实现累加求和，可以比较这三种循环语句的使用方式。

Example3_6.java

```
public class Example3_6 {

    public static void main(String[] args) {
        System.out.println("** while 语句 **");
        int n = 10, sum = 0;
        while (n > 0) {
            sum += n;
            n--;
        }
        System.out.println("sum is " + sum);
```

```
        System.out.println("** do_while 语句 **");
        n = 0;
        sum = 0;
        do {
            sum += n;
            n++;
        } while (n <= 10);
        System.out.println("sum is " + sum);
        System.out.println("** for 语句 **");
        sum = 0;
        for (int i = 1; i <= 10; i++) {
            sum += i;
        }
        System.out.println("sum is " + sum);
    }
}
```

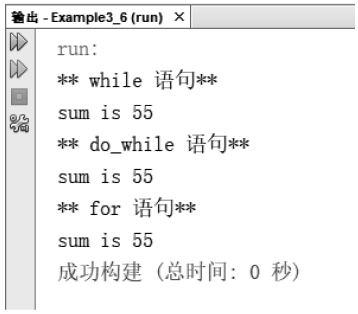


图 3.6　例 3.6 的程序运行结果

例 3.6 的程序运行结果如图 3.6 所示。

3.4　跳转控制

在程序执行时，跳转控制语句可使程序的执行流程转向。Java 的跳转控制语句有 break、continue 和 return 三种。Java 中没有 goto 语句来实现任意的跳转，因为 goto 语句会破坏程序的可读性，而且影响编译的优化。

3.4.1　break 语句

在 switch 语中，break 用来终止 switch 语句的执行，程序流程转向 switch 语句后的第一条语句。同样，在循环语句 for、while、do…while 中，break 立即终止正在执行的循环，程序流程转向循环语句后的第一条语句。

break 语句分为不带标号的 break 语句和带标号的 break 语句。

不带标号的 break 的语法格式为：

```
break;
```

带标号的 break 的语法格式为：

```
标号：语句块
break 标号;
```

语法说明：

(1) 不带标号的 break 语句终止当前语句的执行，程序流程转向到从当前语句的下一条语句开始执行。

(2) 带标号的 break 语句从当前正在执行的程序中跳出，程序流程转向到标号后语句开始执行。

例如在以下代码中，执行了 break b 后，程序流程转向到用标号 b 标识的、用大括号{}

括起来的一段代码。

```
{…
b:    {…                          //标号 b
{…
break b;                          //转移到标号 b 后的语句块执行
…
}
…
}
…
}
```

可以看出,Java 用 break 语句可实现 goto 语句所特有的一些优点。如果 break 语句后指定的标号不是一个代码块的标号,而是一个语句,则这时 break 语句完全实现 goto 语句的功能,不过应该避免这种方式的使用。

3.4.2 continue 语句

continue 语句只能用在循环语句中,它中断本次循环,立即开始下次循环。

continue 语句也分为不带标号的 continue 语句和带标号的 continue 语句。

不带标号的格式为:

```
continue;
```

带标号的格式为:

```
continue 标号;
```

语法说明:

(1) 不带标号的 continue 语句使控制无条件地转移到循环语句的条件判定部分,即首先结束本次循环,跳过循环体中下面尚未执行的语句,接着进行终止条件的判断,以决定是否继续循环。对于 for 语句,在进行终止条件的判断前,还要先执行迭代语句。

(2) 带标号的 continue 语句通常用在多重循环中,标号应放在外循环的开始处。例如,在以下代码中,当满足 j > i 的条件时,程序执行 continue outer 后,立即从内循环跳转到标号 outer 标志的外循环。

```
outer: for( int i = 0; i < 10; i++){    //外循环
for( int j = 0; j < 20; j++){           //内循环
if( j > i ){
…
continue outer;                         //跳出内循环,开始下一次外循环
}
…
}
…
}
```

例 3.7 求 100～200 的所有素数。本例通过一个嵌套的 for 语句来实现。

Example3_7.java

```
public class Example3_7{
      public static void main(String args[]){
      System.out.println(" ** 界于 100 — — 200 的素数 ** ");
      int n = 0;
            outer: for(int i = 101;i < 200;i += 2){
      int k = 15;
            for(int j = 2;j < = k;j++){
            if(i % j == 0)
      continue outer;
            }
      System.out.print(" " + i);
      n++;
            if(n < 10)
      continue;
          System.out.println();
      n = 0;
      }
      System.out.println();
          }
  }
```

例 3.7 的程序运行结果如图 3.7 所示。

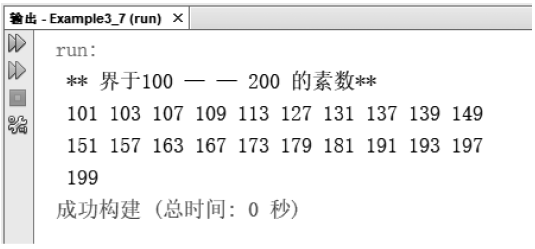


图 3.7 例 3.7 的程序运行结果

3.4.3 return 语句

return 语句从当前方法中返回到调用该方法的语句处，并继续执行后面的语句（有关方法的内容，将在第 6 章介绍）。返回语句有以下两种格式。

第一种格式：

```
return 表达式;
```

第二种格式：

```
return;
```

语法说明：

(1) 第一种 return 语句返回一个值给调用该方法的语句，返回值的数据类型必须与在

方法声明中的返回值类型一致，如果不一致可以使用强制类型转换来使类型一致。

(2) 第二种 return 语句不返回任何值。

(3) return 语句通常用在一个方法体的最后，退出该方法并返回一个值。Java 中单独的 return 语句用在一个方法体的中间时，会产生编译错误，因为这时会有一些语句执行不到。但可以通过把 return 语句嵌入某些语句中(如 if…else)使程序在未执行完方法中的所有语句时退出。

例如在下例中，方法 method 根据 num 是否大于 0，决定是否返回整数 num。

```
int method (int num){
if (num > 0)
return num;                              //如果 num > 0 ,返回整数 num
…
}
```

3.5 其他语句

Java 除了常规的流程控制语句外，还有特殊的语句，如异常处理语句。另外，Java 同样也有注释语句，用来专门在程序中给出注释。

异常处理语句包括 try、catch、finally 以及 throw、throws 语句。异常处理语句是 Java 所特有的，将在后面章节中做专门的介绍。

Java 中可以采用下面三种注释方式。

//——用于单行注释。注释从//开始，终止于行尾。

/ * … * /——用于多行注释。注释从/ * 开始，到 * /结束，且这种注释不能互相嵌套。

/ ** … * /——是 Java 所特有的 java doc 注释。它以/ ** 开始，到 * /结束。这种注释主要是为支持 JDK 工具 javadoc 而采用的。javadoc 能识别注释中用标记@标识的一些特殊变量，并把 java doc 注释加入它所生成的 HTML 文件。对 javadoc 的详细讲述可参见 JDK 的相关工具参考书。

应用实例
视频讲解

3.6 应用实例：图形界面的简单计算器

利用 NetBeans 可视化的图形界面开发工具，结合本章所学内容可实现一个图形界面的简单计算器。

打开 NetBeans 开发工具，选择“文件”→“新建项目”命令，项目名称设置为“JFrameCal”，主类设置为“Main”，如图 3.8 所示。

项目建立完成后，鼠标右击主类 Main.java 所在的包 jframecal，在弹出的快捷菜单中选择“新建”→“JFrame 窗体”命令，如图 3.9 所示。

在新建窗体的对话框中将类命名为 CalJFrame，如图 3.10 所示。

选中新建的 CalJFrame.java，可以通过 NetBeans 的可视化功能对计算器的界面进行编辑，在开发工具右上侧的组件面板中选中“文本字段”，可用鼠标拖放到窗体上，并可适当调整其大小和位置，如图 3.11 所示。文本字段可用于让用户输入文本信息。

图 3.8　新建 Java 应用程序

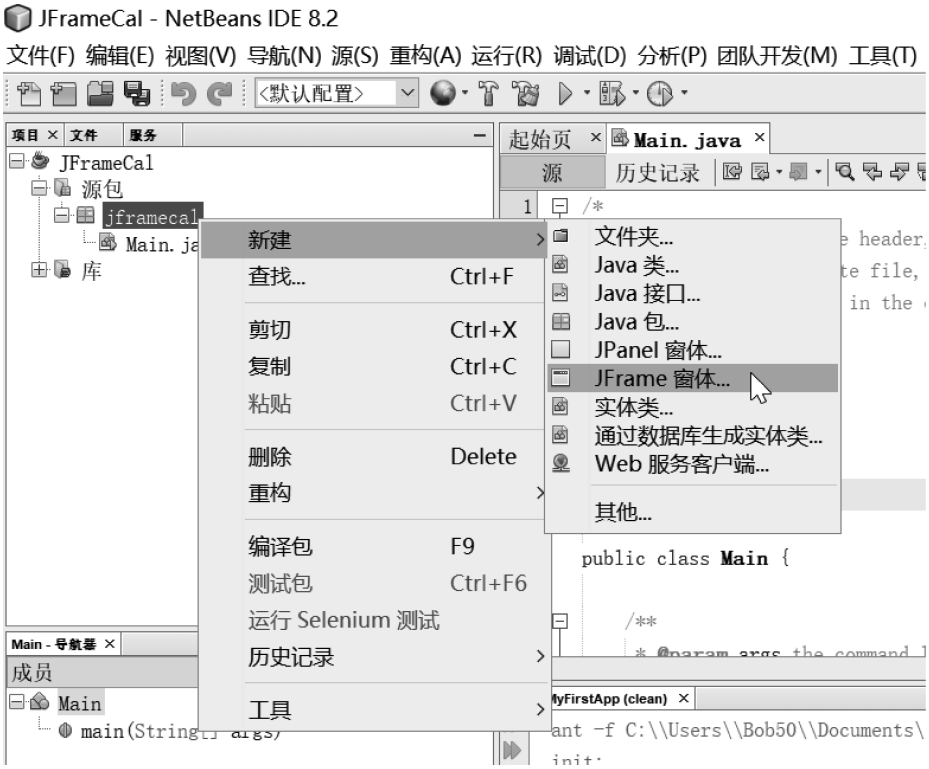


图 3.9　选择“新建”→“JFrame 窗体”命令

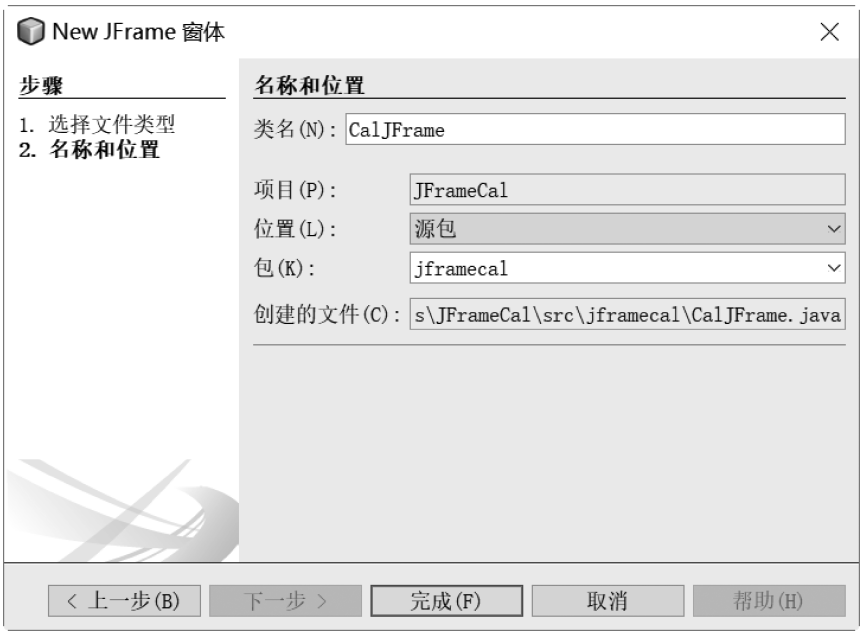


图 3.10　在“New JFrame 窗体”对话框设置名称和位置

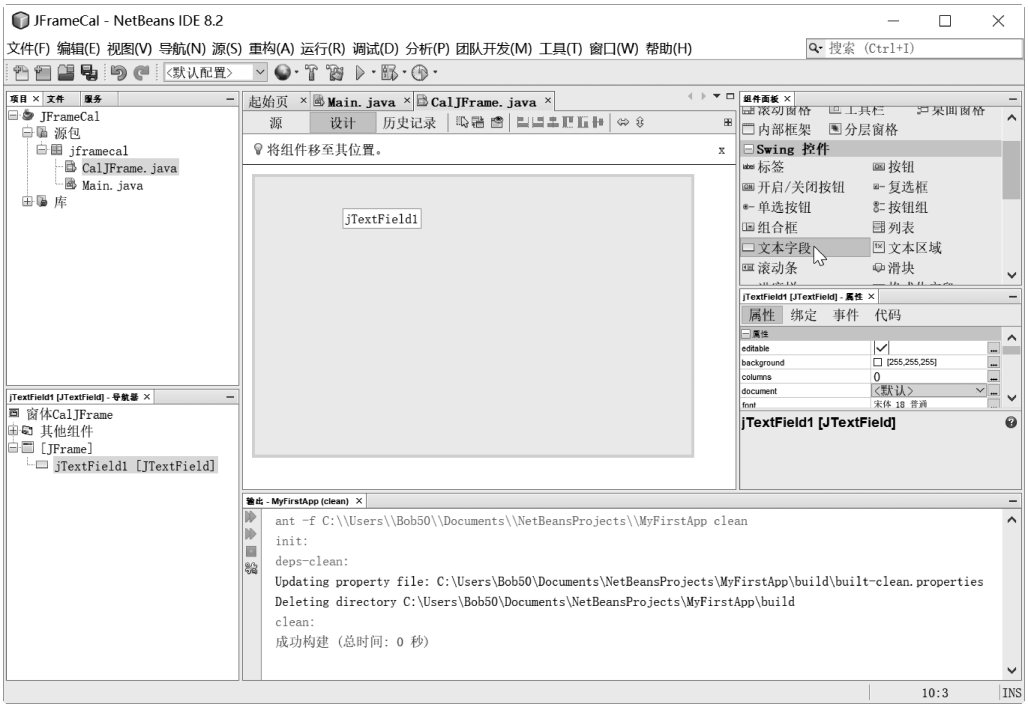


图 3.11　窗体的可视化编辑界面

继续在窗体中拖入另一个文本字段。这两个文本字段用于用户录入两个操作数。选中任一文本字段，在开发工具的右下侧属性窗体中，可将文本字段的 Text 属性设置为空，如图 3.12 所示。

图 3.12　修改文本字段的 Text 属性

在窗体中继续拖放入一个组合框，单击组合框的 model 属性按钮，如图 3.13 所示。

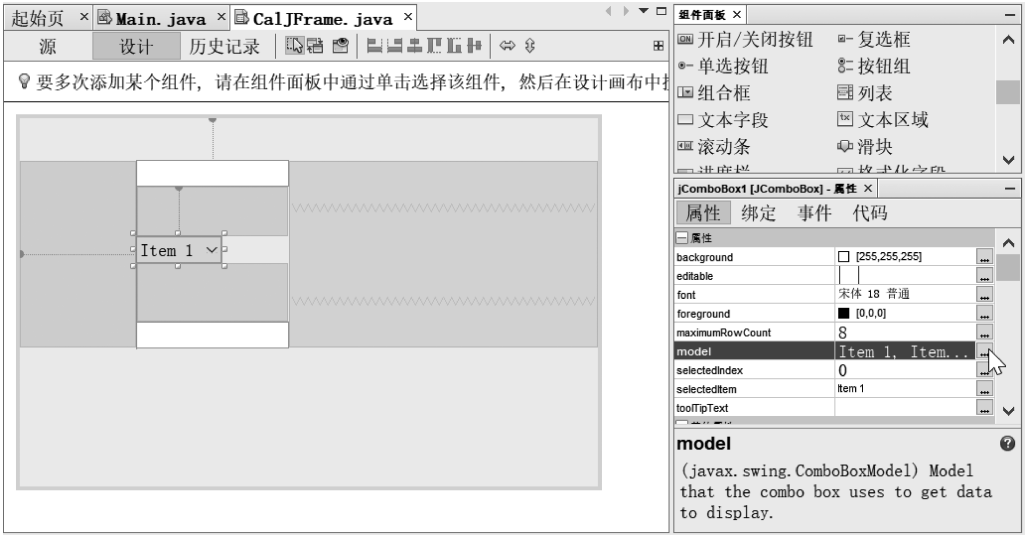


图 3.13　单击组合框 model 属性按钮

在 model 属性的设置对话框中，将组合框项设置为“＋、－、＊、/”四项，如图 3.14 所示。通过该组合框可让用户选中适当的运算符进行运算。

最后在窗体上拖入一个标签和一个按钮组件。将标签的 Text 属性设置为“结果”，用于显示最终的运行结果。将按钮的 Text 属性设置为“运算”，单击该按钮进行运算，并将运算结果显示在结果标签上。最终的界面如图 3.15 所示。

至此，整个计算器的界面由两个文本字段、一个组合框、一个标签和一个按钮构成。在开发工具左下角的“导航器”窗口中可以看到，这些组件所对应的对象名依次为

图 3.14　组合框 model 属性的设置

图 3.15　修改按钮的 Text 属性

jTextField1、jTextField2、jComboBox1、jLabel1 和 jButton1，如图 3.16 所示。

单击 CalFrame.java 的“源”视图可切换到该类的源代码编辑界面，单击“设计”视图可重新切换到可视化的编辑界面，如图 3.17 所示。

在源代码编辑界面，CalJFrame 类中定义三个私有成员变量，如图 3.18 所示。

同第 2 章控制台简单计算器类似，double 类型的 num1 和 num2 这两个变量用来保存运算用的操作数 1 和操作数 2；而 char 类型的 sign 变量用来保存所使用的运算符。

重新切换至“设计”视图，鼠标右击“计算”按钮，在弹出的右键菜单中依次选择“事件”→Action→actionPerformed 命令，为“计算”按钮增加相应的单击事件的处理程序（具体原理会

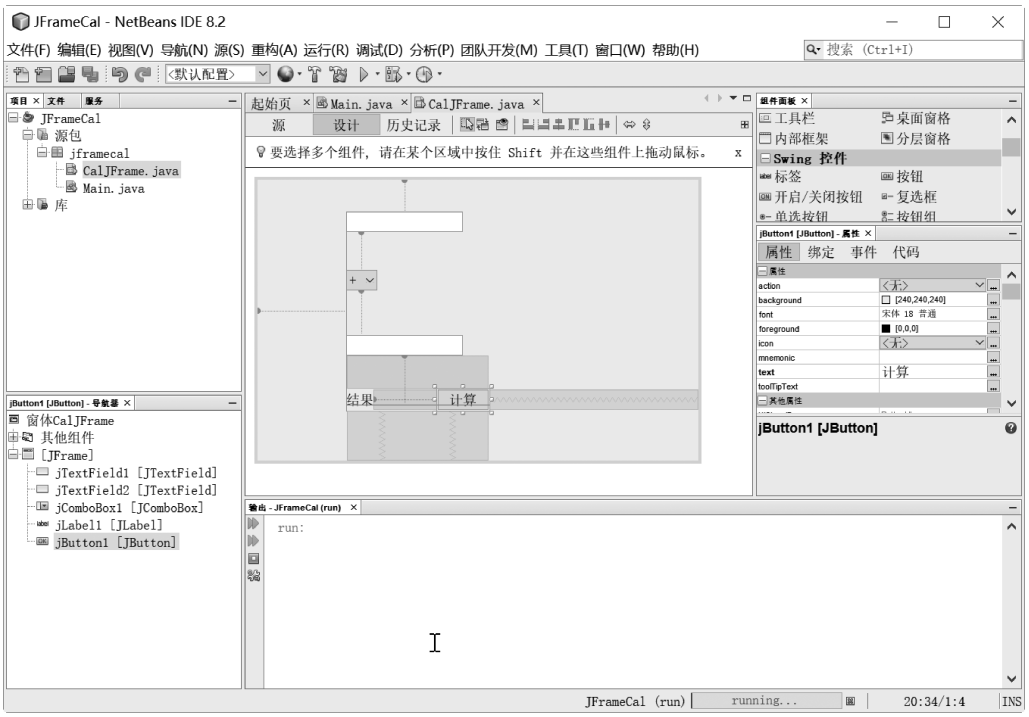


图 3.16　开发工具左下角的“导航器”窗体

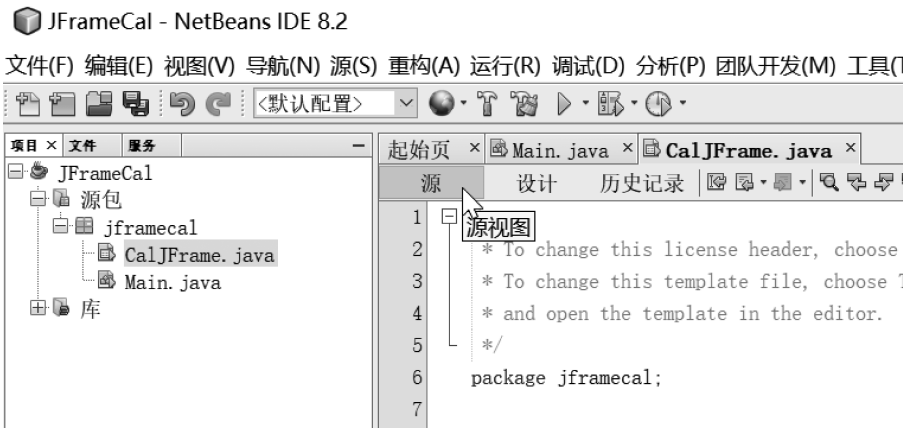


图 3.17　“源”视图与“设计”视图

在第 9 章 Java Swing 图形用户界面中进行详细阐述)。在自动生成的 jButton1ActionPerformed (java.awt.event.ActionEvent evt)方法中对计算过程进行具体实现,其详细代码如下。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        //通过文本字段获得第一个操作数
        String snum1 = this.jTextField1.getText().trim();
        //通过文本字段获得第二个操作数
        String snum2 = this.jTextField2.getText().trim();
```

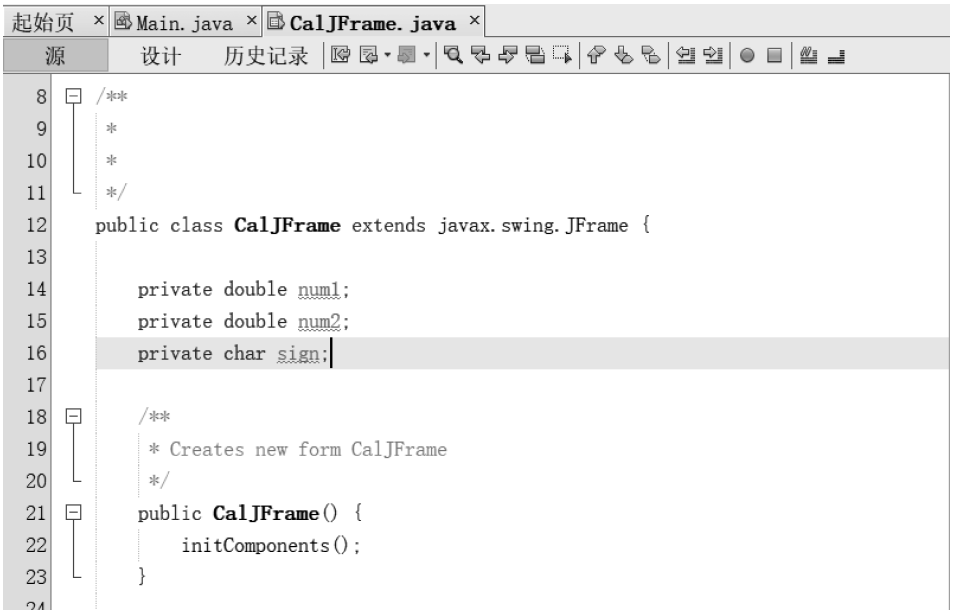


图 3.18　在 CalJFrame 类中定义三个私有成员变量

```
        //将第一个操作数由 String 类型转换为 double 类型
        num1 = Double.parseDouble(snum1);
        //将第二个操作数由 String 类型转换为 double 类型
        num2 = Double.parseDouble(snum2);
        //通过组合框获取运算符
        sign = this.jComboBox1.getSelectedItem().toString().charAt(0);
        //设置运算结果,初始值为 0
        double result = 0;
        //根据运算符进行相应运算
        switch(sign)
        {
            case '+':
                result = num1 + num2;
                break;
            case '-':
                result = num1 - num2;
                break;
            case '*':
                result = num1 * num2;
                break;
            case '/':
                result = num1 / num2;
                break;
        }
        //将运算结果转换为 String 类型设置到标签的 Text 属性
        this.jLabel1.setText(String.valueOf(result));
    }
```

图形界面的简单计算器的程序运行结果如图 3.19 所示。

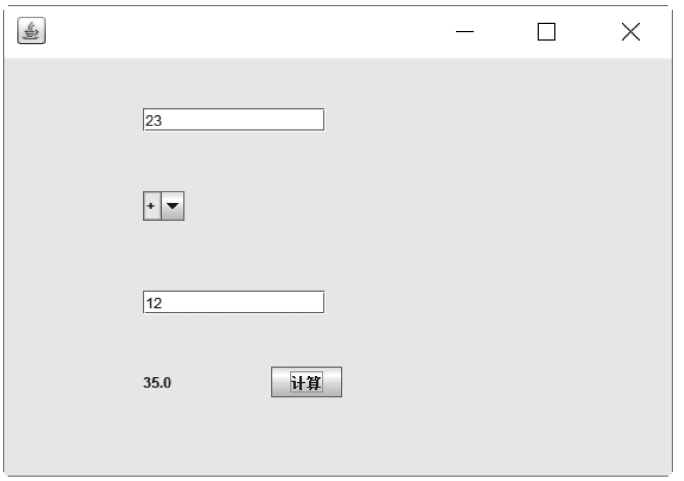


图 3.19　图形界面的简单计算器的程序运行结果

小　　结

本章重点介绍了 Java 表达式和各种运算符的使用方法、Java 的控制语句(用于判断的 if 和 if…else,选择结构 switch,循环结构 do…while、while、for),并介绍了跳转语句的使用。

在编写程序的时候,构造良好的控制结构,可提高整个程序的质量和可读性。

习　　题

1. 用 for 循环结构求出 1～100 中所有偶数的和。

2. 用 while 循环语句和计数变量 x 打印 1～50 中的奇数。要求每行只打印 6 个。

3. 编制程序计算 1+1/2+1/4+1/8+…+1/100 的和。

4. 编制程序输出一个杨辉三角形。

5. 求阶乘 $n!$,分别采用 int 和 long 数据类型,看 n 在什么取值范围内,结果不会溢出。

6. 输出 100 000 以内的所有完全数,如果一个数恰好等于它的所有真因子(即除了自身以外的约数)的和,则称该数为“完全数”。例如,第一个完全数是 6,它有约数 1、2、3、6,除去它本身 6 外,其余 3 个数相加 1+2+3=6。