第 2 章　Java 程序设计基础

知识目标

◎ 了解标识符和关键字
◎ 了解 Java 程序中的变量和常量
◎ 掌握 Java 程序中的基本数据类型
◎ 掌握 Java 程序中的运算符和表达式

技能目标

◎ 会定义和使用变量和常量，掌握数据类型转换规则，并熟练进行转换
◎ 熟悉运算符优先级，并熟练运用各种运算符建立表达式

任何程序设计语言都需要操纵和处理数据，程序的基本功能就是处理数据。数据是有类型的，不同的数据类型有不同的运算形式。本章主要介绍 Java 程序设计语言的数据类型、运算符和表达式。

2.1　Java 基本数据类型

Java 语言把数据类型分为基本数据类型和引用（对象）数据类型，图 2-1 显示了 Java 语言数据类型的详细分类。

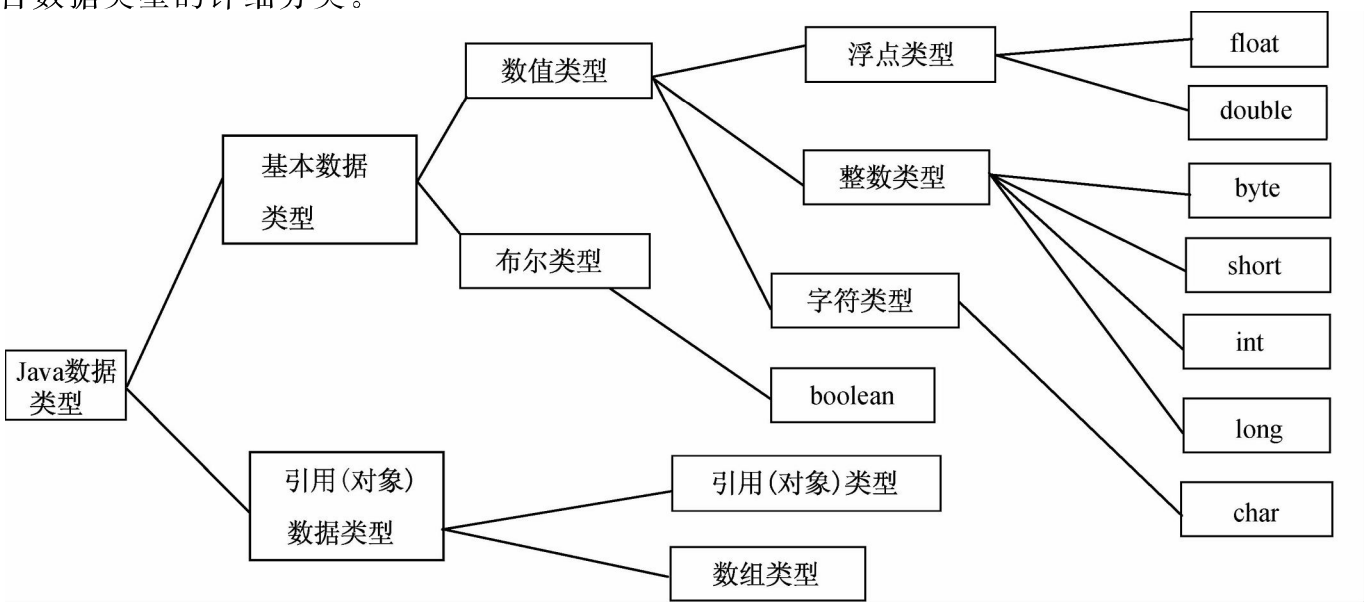

图 2-1　Java 数据类型

本节主要就 Java 基本数据类型进行介绍，引用（对象）数据类型将在第 5 章作进一步介绍。

2.1.1 标识符和关键字

1.标识符

Java 程序中的类、方法、对象、变量等元素都应该有自己的名称，对 Java 程序中的各个元素命名时使用的命名符号称为标识符。

Java 语言中，标识符是以字母、下划线（_）、美元符号（$）开始的一个字符序列，后面可以跟字母、下划线、美元符号和数字，不可以用 Java 关键字。

例如，从以下字符串中挑出合法的 Java 标识符：

2how　　class　　username　　user name

sys_val　　$change　　is_room!　　do

其中，合法的标识符有 username、$change、sys_val，非法的标识符有 2how（第一个字符不能为数字）、class、do（不能为 Java 关键字）、user name（不能有空格）、is_room!（! 不能作为标识符的一部分）。

合法的标识符是符合标识符命名规则的字符串；不符合标识符命名规则的是非法标识符。另外，标识符是区分大小写的，如 myname 和 MyName 是两个不同的标识符。

2.关键字

Java 语言中的关键字是指程序代码中的特殊字符，具有专门的意义和用途，不能当做一般的标识符使用，下面列出了 Java 语言中的所有关键字。

- 类和接口的声明：class、implements、interface、extends。
- 包和引入包的声明：package、import。
- 基本数据类型：int、byte、long、short、boolean、char、double、float。
- 数据类型特定值：true、false、null。
- 流程控制：break、case、continue、default、do、else、for、if、return、switch、while。
- 异常处理：try、catch、finally、throw、throws。
- 修饰符：abstract、final、private、protected、public、synchronized、static、void。
- 创建对象：new。
- 引用：this、super。

Java 语言中的关键字均用小写字母表示，每个关键字都有特殊的作用。例如，package 用于包的声明；class 用于类的声明；void 表示方法没有返回值，在本书的其他章节会陆续介绍其他关键字的作用。

2.1.2 数据类型和长度

在 Java 程序中，每个变量都有指定的数据类型，例如：

```
float a;
char ch;
```

a 是单精度实型的数据变量，编译系统会分配 4 字节的内存空间，不同类型的变量在内存中分配的字节数不同，所以给变量赋值前需要先确定变量的类型。

Java 基本数据类型包括：

- 整数类型(integer)：byte、short、int、long。
- 浮点类型(floating)：float、double。
- 字符类型(textual)：char。
- 布尔类型(logical)：boolean。

1. 整数类型

整数类型包括字节型、短整型、整型和长整型，见表 2-1。

表 2-1　整数类型说明

整数类型	字 节 数	取值范围
字节型(byte)	1	[−128,127]，即 $-2^7 \sim 2^7-1$
短整型(short)	2	[−32 768,32 767]，即 $-2^{15} \sim 2^{15}-1$
整型(int)	4	[−2 147 483 648,2 147 483 647]，即 $-2^{31} \sim 2^{31}-1$
长整型(long)	8	$-2^{63} \sim 2^{63}-1$

2. 浮点类型

浮点类型包括单精度实型、双精度实型，见表 2-2。

表 2-2　浮点类型说明

浮点类型	字 节 数	取值范围
单精度实型(float)	4	−3.402 823 47E+38～3.402 823 47E+38
双精度实型(double)	8	−1.797 693 134 862 315 70E+308～1.797 693 134 862 315 70E+308

3. 字符类型

字符类型也是较为常见的一种基本数据类型，Java 语言对字符采用 Unicode 字符编码，由于计算机只能存储二进制数据，因此必须为各个字符进行编码。所谓字符编码，是指用一串二进制数据来表示特定的字符。Unicode 字符编码既支持英文字符也支持汉字字符，一个 Unicode 编码的字符在内存中占用两个字节。

4. 布尔类型

布尔类型又名逻辑类型，是 Java 语言所特有的，是最简单的数据类型。它是一种用来表达逻辑真假的数据类型。在内部存储上，用一位来表示，具体在使用上，用 true 表示逻辑真，用 false 表示逻辑假，例如：

```
boolean a=true; //定义变量 a 是 boolean 类型的变量，并赋值为 true
```

2.1.3　常量和变量

在 Java 程序中存在大量的数据来代表程序的状态。在实际编写的 Java 程序中，需要根据数据在程序运行中取值是否改变来决定是采用变量还是常量。

1. 常量

常量是指在程序运行过程中值固定不变的量。Java 语言中的常量有整型常量、浮点型

常量、字符型常量和布尔型常量 4 种。

1)整型常量

Java 整型常量有十进制、八进制和十六进制 3 种形式。其中,十进制整型常量是由不以 0 开始的、0～9 的数字组成的数据组合。八进制整型常量是由以 0 开始的、0～7 的数字组成的数据组合。十六进制整型常量是以 0x 或 0X 开始的、0～9 的数字和 A～F 的字母组成的数据组合。

- 十进制整型常量:如 246、－456、4 567。
- 八进制整型常量:以 0 开始,如 0123 表示十进制数 83,－021 表示十进制数－17。
- 十六进制整型常量:以 0x 或者 0X 开始,如 0x123 表示十进制数 291,－0X21 表示十进制数－33。

2)浮点型常量

Java 中的浮点型常量是带有小数的十进制数,又分为以 4 字节存储的单精度常量和以 8 字节存储的双精度常量。不加后缀的、带小数点的十进制数都默认为双精度常量;如果要特别说明为单精度常量,可以在数的末尾加上 f 或 F 作为后缀,如 3.141 5(默认为双精度常量)、3.16f(末尾加 f,为单精度常量)。

在 Java 程序中使用浮点型常量时,可以用小数点和科学计数法两种形式表示。例如,3.14159265f、6.09e8F、9.80404e20D、3.3244e30d 等。

3)字符型常量

字符型常量是指由单引号括起来的单个 Unicode 字符集中的字符,如 A、a、3 等。对于不可显示或有特殊意义的字符,Java 语言允许用带“\”开始的字符序列来表示。表 2-3 中列出了 Java 语言中以“\”开始的特殊字符,这些字符对标准输出起控制作用。

表 2-3 Java 中的转义字符

转义字符	描　　述
\n	换行符
\t	横向跳格制表符
\r	回车符
\\	输出反斜杠字符
\′	输出单引号字符
\″	输出双引号字符

4)布尔型常量

布尔型常量只有两个值,即 true 和 false,分别表示“真”和“假”两种状态。Java 语言中不支持像 C 或 C++语言中用非 0 和 0 表示的“真”和“假”两种状态。

布尔型常量的组成字母都是小写的。

2.变量

变量是指程序运行过程中其值可以发生改变的量。在 Java 中程序是通过变量来操作内存中的数据的，因此程序在使用任何变量之前首先应该在该变量和内存单元之间建立联系，这个过程称为定义变量或声明变量。

定义变量包括定义变量的数据类型和变量名称两个部分。在使用一个变量之前，必须先定义变量。定义变量是通过定义语句来实现的，例如：

＜数据类型＞ ＜变量＞；

＜数据类型＞ ＜变量 1＞，＜变量 2＞，…，＜变量 n＞；

＜数据类型＞ ＜变量＞ ＝ ＜数据值＞；

其中，数据类型可以是一个基本数据类型，也可以为引用(对象)数据类型，而数据值则是相应的数值或者是引用(对象)数据类型的一个实例(对象)，如果同时定义了多个相同类型的变量，则它们之间用逗号分隔。例如：

```
boolean a=true;              //定义变量 a 为布尔型，且赋值为 true
char b='F';                  //定义变量 b 为字符型，且赋值为'F'
int c=8;                     //定义变量 c 为整型，且赋值为 8
double d=3.14;               //定义变量 d 为双精度实型，且赋值为 3.14
float e=45.78f;              //定义变量 e 为单精度实型，且赋值为 45.78
int i,j,m,n;                 //定义多个变量 i、j、m、n 为整型
```

【例 2-1】 定义并输出常量和变量。

参考程序如下：

```
public class MyOne{
    public static void main(String[] args){
        int a=12;
        char b='d';
        double c=23.65;
        float d=34.65f;
        System.out.println("a="+a);
        System.out.println("b="+b);
        System.out.println("c="+c);
        System.out.println("d="+d);
    }
}
```

程序运行结果如下：

```
a=12
b=d
c=23.65
d=34.65
```

【例 2-2】 已知长方形的长和宽，求长方形的周长和面积。

参考程序如下：

```
public class Rectangle{
```

```
    public static void main(String[] args){
        double a=1.9;                          //长方形的长
        double b=0.3;                          //长方形的宽
        double perimeter=(a+b)*2;              //计算周长
        double area=a *b;                      //计算面积
        System.out.println("长方形的周长是:"+perimeter+"米");
        System.out.println("长方形的面积是:"+area+"平方米");
    }
}
```

程序运行结果如下：

```
长方形的周长是:4.3999999999999995 米
长方形的面积是:0.57 平方米
```

2.1.4 类型转换

除了布尔型的数据外，其他不同类型的数据可以进行混合运算，不同类型的数据运算的结果是不同的。

1.自动类型转换

整型、字符型、浮点型都可以看做数值型数据，可以进行混合运算，当不同的数值型数据进行混合数据运算时，有一个潜在的规则：不同类型的数据先转化为同一类型，然后再进行数据运算。

自动类型转换总是按照数据类型的运算精度由低到高进行，byte、char、short 的运算精度最低，double 的运算精度最高，具体见表 2-4。

表 2-4 Java 中的自动类型转换规则

操作数 1	操作数 2	运算后的类型
byte、char、short、int、long 或 float	double	double
byte、char、short、int、long	float	float
byte、char、short、int	long	long
byte、char、short	int	int
byte、char、short	byte、char、short	int

由于字符型数据在内存中是以字符对应的 Unicode 码来存储的，所以从本质上看，整型数据和字符型数据在内存的表示一致，因此两者的转换相当简单，完全可以自由进行，通常不必使用任何强制转换。

2.强制类型转换

不同类型的数据之间进行运算时，低精度类型的数据可以自动转换成高精度类型的数据。若让高精度数据转换成低精度数据，需要强制类型转换，其形式是：

(类型)数据

例如：

```
int a=3.68;          //编译出错,高精度的数据不能给低精度变量赋值,会损失精度
int a=(int)3.68; //将 3.68 强制转换为 int 型数据,然后再给同类型的变量赋值
```

2.2　运算符和表达式

程序中数据的运算通过运算符来实现,Java 语言提供了丰富的运算符,Java 程序中的运算符如果按操作数的数目来分,可以有一元运算符、二元运算符和三元运算符。由运算符和变量、常量组成的式子称为表达式,如 2+3,a * b 等。本节将介绍算术运算符、关系运算符、逻辑运算符、自增运算符、自减运算符、赋值运算符、赋值表达式,以及各种运算符的优先级。

2.2.1　算术运算符

Java 语言提供了丰富的算术运算符。算术运算符用在数学表达式中,可以实现基本的数值运算,其用法和功能与数学中类似,Java 提供的算术运算符见表 2-5。

表 2-5　算术运算符

算术运算符	实际操作	例　　子
+	加法运算	a+b
−	减法运算	a−b
*	乘法运算	a * b
/	除法运算	a/b
%	求余(取模)运算	a%b

在算术运算符中,Java 语言对“+”进行了扩展,它不仅可以对数值型数据进行加法运算,还可以进行字符串之间、数字与字符串之间的连接。例如：

```
System.out.println("A="+24+6); //输出 A=30
```

对于除法运算符,当除法两边都是整型数时,运算的结果也为整型数。例如：

```
int a=14/3;          //a 的变量取值为 4
```

2.2.2　关系运算符和逻辑运算符

1.关系运算符

关系运算符用来比较两个值的关系,其运算结果是布尔型,当运算符对应的关系成立时,运算结果为 true,否则为 false。例如,10<9 的结果为 false,5>1 的结果为 true。Java 语言提供的关系运算符见表 2-6。

表 2-6　关系运算符

关系运算符	实际操作	例　　子
<	小于	a<b
>	大于	a>b

续表

关系运算符	实际操作	例　　子
<=	小于等于	a<=b
>=	大于等于	a>=b
!=	不等于	a!=b
==	等于	a==b

(1)“==”不能写成“=”,“=”为赋值运算符。

(2)任何数据类型都可以通过“==”、“!=”比较两个数是否相等。

(3)关系运算的结果是 true 和 false。

(4)关系运算符的优先级低于算术运算符,高于赋值运算符。

2.逻辑运算符

逻辑运算符实现逻辑与、逻辑或、逻辑非。逻辑运算符的操作数必须是布尔型数据,逻辑运算符可以用来连接关系表达式,Java 语言提供的逻辑运算符见表 2-7。

表 2-7　逻辑运算符

逻辑运算符	实际操作	例　　子
&&	逻辑与,两者同时为 true 时,结果才为 true	a&&b
||	逻辑或,两者中任意一个为 true 时,结果就为 true	a||b
!	逻辑非,对操作数的否定	!a

(1)参与逻辑运算的都是关系表达式或者布尔型的数据。

(2)对于“&&”逻辑运算符,当左边的布尔表达式的值为 false 时,整个表达式的值肯定为 false,此时会忽略右边的布尔表达式。

2.2.3　自增运算符和自减运算符

自增运算符和自减运算符是单目运算符,可以放在操作数之前,也可以放在操作数之后。操作数必须是一个整型或浮点型变量,作用是使变量的值增 1 或减 1。例如:

++a(--a)表示在使用 a 之前,先使 a 的值加 1(减 1);a++(a--)表示在使用 a 之后,使 a 的值加 1(减 1)。

【例 2-3】 Java 语言中自增与自减运算符的用法。

参考程序如下:

```
public class TestMod{
    public static void main(String[] args){
        int a=9;
        a++;                          //先使用,然后变量再增加(a 会增加 1 变为 10)
        System.out.println(a);        //输出 10
        System.out.println(a++);//先使用(输出 10)再增加变为 11
```

```
        System.out.println(++a);  //变量先增 1 变为 12,然后再使用(输出 12)
        System.out.println(a--);  //先使用(输出 12)再减 1 变为 11
        System.out.println(--a);  //变量先减 1 变为 10,然后再使用(输出 10)
    }
}
```

程序运行结果如下：

```
10
10
12
12
10
```

2.2.4　赋值运算符和赋值表达式

赋值运算符是 Java 语言中最基本的运算符，它就是通常数学上的等于符号“=”。

Java 语言中的赋值分为基本赋值和复合赋值两种，见表 2-8。

表 2-8　赋值运算符

赋值运算符	实际操作	例　　子	结　　果
=	赋值	a=4; b=2;	a 的值为 4,b 的值为 2
+=	加等于	a=4; b=2; a+=b;	a 的值为 6,b 的值为 2
-=	减等于	a=4; b=2; a-=b;	a 的值为 2,b 的值为 2
*=	乘等于	a=4; b=2; a*=b;	a 的值为 8,b 的值为 2
/=	除等于	a=4; b=2; a/=b;	a 的值为 2,b 的值为 2
%=	(求余)模等于	a=4; b=2; a%=b;	a 的值为 0,b 的值为 2

1.基本赋值

基本赋值的形式：

变量=表达式；

作用是将“=”右边的值赋给左边的变量，表达式可以是常量表达式或者变量表达式，后面加上分号即构成一个完整的赋值语句。

Java 语言是强类型的语言，所以赋值时要求类型必须匹配，如果左边的变量的类型精度高于右边的表达式的类型精度，右边的表达式类型会自动转换为左边的变量的类型；如果左边的变量的类型精度低于右边的表达式的类型精度，出现精度丢失，编译时提示错误。例如：

```
int a=10;           //类型匹配,直接赋值
double d=100;       //类型不匹配,系统首先自动将 100 转换成 100.0,然后赋值
float c=3.16;       //类型不匹配,无法自动转换,编译时提示错误
```

2.复合赋值

基本赋值运算符与算术运算符组成复合赋值运算符。

【例 2-4】 理解复合赋值运算符。

参考程序如下：

```
public class TestAssign{
    public static void main(String[] args){
        int a=9,b=7;                    //a 赋值为 9,b 赋值为 7
        a+=b;                           //相当于 a=a+b,a 被赋值为 16
        System.out.println("a="+a);
        b*=(a-9);                       //相当于 b=b*(a-9),b 被赋值为 49
        System.out.println("b="+b);
        b%=(a-5);                       //相当于 b=b%(a-5),b 被赋值为 5
        System.out.println("b="+b);
    }
}
```

程序运行结果如下：

```
a=16
b=49
b=5
```

2.2.5 运算符优先级

当运算趋于复杂时，可能在一个运算中出现多个运算符，那么运算时，就按照优先级的高低进行，级别高的运算符先运算，级别低的运算符后运算，具体运算符的优先级见表 2-9。

表 2-9 运算符优先级

优 先 级	运 算 符	结 合 性
1	()、[]、.	从左到右
2	!、+(正)、-(负)、++、--	从右向左
3	*、/、%	从左向右
4	+(加)、-(减)	从左向右
5	<<、>>、>>>	从左向右
6	<<=、>>=	从左向右
7	==、!=	从左向右
8	&&	从左向右
9	||	从左向右
10	?:	从右向左
11	=、+=、-=、*=、/=、%=	从右向左

【例 2-5】 给出一个 4 位数，输出各位上的数字。

参考程序如下：

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

```
        int num=3426;
        int gewei=num%10;           // 分解获得个位数
        int shiwei=num/10%10;       // 分解获得十位数
        int baiwei=num/100%10;      // 分解获得百位数
        int qianwei=num/1000;       // 分解获得千位数
        System.out.println("数字:" + num);
        System.out.println("千位数是:" + qianwei);
        System.out.println("百位数是:" + baiwei);
        System.out.println("十位数是:" + shiwei);
        System.out.println("个位数是:" + gewei);
    }
}
```

程序运行结果如下：

```
数字:3426
千位数是:3
百位数是:4
十位数是:2
个位数是:6
```

习　题　2

1. 在 Java 中字符型数据是采用哪种编码形式？占用的字节数是多少？
2. 在 Java 中八进制和十六进制的数如何表示？
3. Java 语言中基本数据类型有哪些？
4. 什么是变量？如何定义(声明)一个变量？
5. 编程实现给出一个 3 位的八进制数，求出对应的十进制数。
6. 编程实现给出 3 个整数，求平均数。
7. 编程实现给出一个 3 位整数，按相反的顺序输出该数，即输入 247，输出 742。

第 7 章　图形用户界面

知识目标

◎ 了解 Java 图形用户界面的特点
◎ 掌握 Java GUI 布局管理的使用
◎ 掌握 Java GUI 常用组件的使用方法
◎ 理解并掌握 Java 的事件与事件处理机制
◎ 掌握面向对象思想在 Java GUI 中的具体应用

技能目标

◎ 掌握 FlowLayout、BorderLayout、GridLayout 和 CardLayout 布局管理器的构造方法及布局效果
◎ 掌握 Java 事件及其相应的监听器接口，并能熟练运用
◎ 能利用本章所学知识开发图形用户界面程序

在前面的章节中，介绍了 Java 语言的大量基础知识，有了这些内容，就完全可以进行一般的 Java 程序设计开发了。考虑到现实的开发要求，显然，还缺乏一种图形界面程序的开发能力。Java 语言提供了大量图形用户界面程序的设计开发方法。

图形用户界面(graphics user interface，GUI)是用户与程序交互的窗口，它比前面的基于命令行的界面更直观和友好，GUI 是采用图形的方式，借助菜单、按钮等标准的界面元素和鼠标操作，帮助用户方便地向计算机发出指令，启动操作，并将系统运行的结果以图形的方式显示给用户。

7.1　Java 图形用户界面概述

Java 语言提供了大量图形用户界面(GUI)程序的设计开发方法，其中，最重要的就是 AWT 和 Swing 技术。本节主要介绍 AWT 和 Swing 技术以及图形用户界面的两个核心类——组件和容器。

7.1.1　AWT 和 Swing

AWT 和 Swing 是 Java 语言设计 GUI 的基础。Java 语言早期提供的 GUI 开发并不是 Swing，而是 AWT，同样这个看似古怪的词语仍然来自于包，字面意思为 abstract window

toolkit(抽象窗体开发包)。AWT 提供了完整的窗体开发方法,然而,由于技术的局限性,AWT 在很多方面并不好用。从 Java 2 版本推出 Swing 以来,Java 语言作出了极大的改变,如在体系结构上使用了较为稳健的设计框架,在风格上提供了更为灵活的定义方法,等等。在目前的 Java 标准版开发中,Swing 是可以取代 AWT 的主要 Java GUI 开发形式。但是,AWT 包并没有完全过时,很多类仍然可以在 Swing 编程中发挥作用,事实上,Swing 包类也是建立在 AWT 包类基础上的新类。

Swing 是由 100%纯 Java 程序实现的,Swing 组件是用 Java 实现的轻量级(light-weight)组件,没有本地代码,不依赖操作系统的支持,这是它与 AWT 组件的最大区别。由于 AWT 组件通过与具体平台相关的对等(peer)类实现,因此 Swing 比 AWT 组件具有更强的实用性。Swing 在不同的平台上表现一致,并且有能力提供本地窗口系统不支持的其他特性。

下面通过两个例子分别使用 AWT 和 Swing 技术实现 Java 图形用户界面。

【例 7-1】 采用 AWT 窗口框架实现第一个图形用户界面程序。

参考程序如下:

```
import java.awt.*;
public class MyFirstFrame{
    public static void main(String args[]){
    Frame f = new Frame("MyFirstFrame");
        f.setLocation(300, 300);          //指定窗口出现的位置
        f.setSize(300,150);               //设定窗口的大小
        f.setBackground(Color.blue);      //设置背景颜色
        f.setVisible(true);               //窗口可见
    }
}
```

程序运行结果如图 7-1 所示。

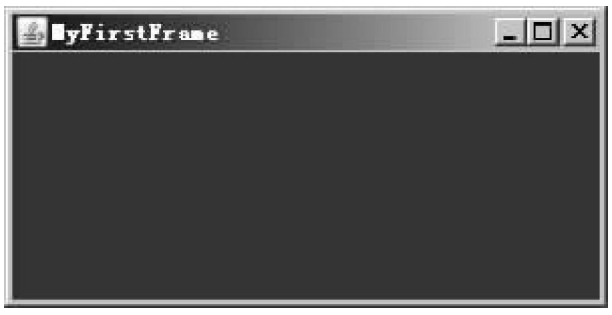

图 7-1 采用 AWT 技术实现图形用户界面

【例 7-2】 采用 Swing 窗口框架实现第一个图形用户界面程序。

参考程序如下:

```
import java.awt.*;
import javax.swing.*;
public class MyFirstJFrame{
    public static void main(String args[]){
        JFrame jf=new JFrame("MyFirstJFrame");
        jf.setSize(300,150);              //设定窗口的大小
```

```
        jf.setBackground(Color.blue);     //设置背景颜色
        jf.setVisible(true);              //窗口可见
    }
}
```

程序运行结果如图 7-2 所示。

通过上面的例子可以看出，尽管第二个例子采用 Swing 包中的组件，然而 AWT 包通常还需要用 import 载入，因为 AWT 包除了包含作为界面元素的组件类外，还包含 Swing 中没有的 Color、Font、各种布局管理类以及负责事件处理的类和接口。

Swing 组件具有美观、易用、组件量大等优点，但也有缺点，即使用 Swing 组件的程序通常会比使用 AWT 组件的程序运行得更慢。但是大家都还是更喜欢用 Swing 组件，因为随着计算机硬件的升级，速度已经不再是问题，用户需要的是更美观的用户界面，开发人员则需要更易用的开发组件，本章重点介绍 Swing 技术实现的 Java 图形用户界面。

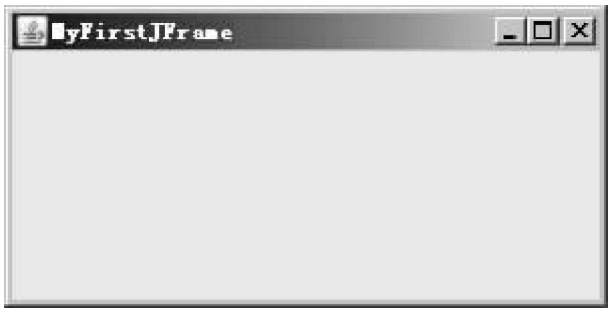

图 7-2　采用 Swing 技术实现图形用户界面

7.1.2　组件和容器

1. 组件和容器概述

Java 语言的图形用户界面的最基本组成部分是组件(component)。组件是一个可以以图形化的方式显示在屏幕上并能与用户进行交互的对象，如一个按钮、一个标签等。组件不能独立地显示出来，必须将组件放在一定的容器中才可以显示出来。

类 java.awt.Component 是许多组件类的父类，Component 类中封装了组件通用的方法和属性，如图形的组件对象、大小、显示位置、前景色和背景色、边界、可见性等，因此许多组件类也就继承了 Component 类的成员方法和成员变量，相应的成员方法包括 getComponentAt(int x, int y)、getFont()、getForeground()、getName()、getSize()、paint(Graphics g)、repaint()、update()、setVisible(boolean b)、setSize(Dimension d)、setName(String name)等。

容器(container)也是一个类，实际上是 Component 的子类，因此容器本身也是一个组件，具有组件的所有性质，但是它的主要功能是容纳其他组件和容器。

为了使生成的图形用户界面具有良好的平台无关性，Java 语言提供了布局管理器来管理组件在容器中的布局，而不使用直接设置组件位置和大小的方式。7.2 节将介绍布局管理器。

2. Swing 组件类

Swing 组件类 JComponent 是一个抽象类，用于定义所有子类组件的一般方法，其类层

次结构如下所示：

```
java.lang.Object
        |
        +--java.awt.Component
                    |
                    +--java.awt.Container
                                |
                                +--javax.swing.JComponent
```

并不是所有的 Swing 组件都继承于 JComponent 类，JComponent 类继承于 Container 类，所以凡是此类的组件都可作为容器使用。

组件从功能上可分为：

(1)顶层容器：JFrame、JApplet、JDialog、JWindow。

(2)中间容器：JPanel、JScrollPane、JSplitPane、JToolBar。

(3)特殊容器：在图形用户界面上起特殊作用的中间层，如 JInternalFrame、JLayeredPane、JRootPane 等。

(4)基本控件：实现人机交互的 GUI 组件，如 JButton、JComboBox、JList、JMenu、JSlider、JTextField 等。

(5)不可编辑信息的显示：向用户显示不可编辑信息的 GUI 组件，如 JLabel、JProgressBar、JToolTip 等。

(6)可编辑信息的显示：向用户显示能被编辑的格式化信息的组件，如 JColorChooser、JFileChoose、JFileChooser、JTable、JTextArea 等。

Swing 组件不能直接添加到顶层容器中，它必须添加到一个与 Swing 顶层容器相关联的内容面板(content panel)上。内容面板是顶层容器包含的一个普通容器，它是一个轻量级组件。基本规则如下：

(1)把 Swing 组件放入一个顶层 Swing 容器的内容面板上。

(2)避免使用非 Swing 的重量级组件。

对 JFrame 添加组件有两种方式：

(1)用 getContentPane()方法获得 JFrame 的内容面板，再对其加入组件 frame.getContentPane().add(childComponent)。

(2)建立一个 JPanel 或 JDesktopPane 之类的中间容器，把组件添加到容器中，用 setContentPane()方法把该容器置为 JFrame 的内容面板：

```
JPanel contentPane=new JPanel();
…//把其他组件添加到 JPanel 中
frame.setContentPane(contentPane);
…//把 contentPane 对象设置成为 frame 的内容面板
```

3.Swing 技术实现 Java 图形用户界面步骤

Swing 技术实现 Java 图形用户界面一般可按照下列步骤进行：

(1)引入 Swing 包。

(2)选择外观。

(3)设置顶层容器。

(4)设置按钮和标签。

(5)向容器中添加组件。

(6)在组件周围添加边界。

(7)进行事件处理。

下面的例子说明了 Swing 技术实现 Java 图形用户界面的结构以及最基本的组件 JButton的用法。在程序中,建立一个 Swing 风格的 GUI 窗口,并在其中添加一个按钮,在每一次单击之后在命令行显示按钮的标签。

【例 7-3】 采用 Swing 技术实现 Java 图形用户界面的结构与基本步骤实例。

参考程序如下:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingApp{
    public static void main( String args[]){
        JFrame jf=new JFrame("SwingApp");
        JButton b=new JButton("确定");
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                System.out.println(((JButton)e.getSource()).getText());
            }
        });
        jf.add(b);
        jf.setSize(300,150);              //设定窗口的大小
        jf.setBackground(Color.blue); //设置背景颜色
        jf.setVisible(true);              //窗口可见
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

程序运行结果如图 7-3 所示。

图 7-3　Swing 技术实现 Java 图形用户界面的结构

在这个程序中可以看到,Swing 组件的使用方法与 AWT 组件的使用方法基本一致,使

用的事件处理机制也完全相同。

7.2 布局管理器

每个容器都有一个布局管理器(layout manager),当容器需要对某个组件进行定位或判断其大小尺寸时,就会调用其对应的布局管理器。

在程序中安排组件的位置和大小时,应该注意以下两点:

(1)容器中的布局管理器负责各个组件的大小和位置,因此用户无法在这种情况下设置组件的这些属性。如果试图使用 Java 语言提供的 setLocation()、setSize()、setBounds() 等方法,都会被布局管理器覆盖。

(2)如果用户确实需要亲自设置组件大小或位置,则应取消该容器的布局管理器,方法为 setLayout(null)。

7.2.1 布局管理器概述

Java 语言中,提供了布局管理器类的对象对容器类的对象进行管理:第一,管理Component在 Container 中的布局,不必直接设置 Component 位置和大小;第二,每个 Container 都有一个布局管理器对象,当 Container 需要对某个组件进行定位或计算其大小尺寸时,就会调用其对应的布局管理器,调用 Container 的 setLayout()方法来改变布局管理器的对象。

AWT 提供了 5 种布局管理器类:

- FlowLayout。
- BorderLayout。
- GridLayout。
- CardLayout。
- GridBagLayout。

下面只介绍前 4 种。

7.2.2 FlowLayout 流布局管理器

FlowLayout 是最简单的布局管理器,是 Panel、JPanel 类的默认布局管理器。它对组件逐行定位,行内从左到右,一行排满后换行。它不会改变组件的大小,而是按照原组件尺寸显示组件,可以设置不同组件之间的距离、行距以及对齐方式等。

FlowLayout 流布局管理器默认的对齐方式是居中。

FlowLayout 的构造方法有以下 3 个:

(1)new FlowLayout(FlowLayout.RIGHT,20,40);

右对齐,组件之间水平间距为 20 个像素,垂直间距为 40 个像素。

(2)new FlowLayout(FlowLayout.LEFT);

左对齐,水平和垂直间距默认为 5 个像素。

(3)new FlowLayout();

使用默认的居中对齐方式，上下左右均间距 5 个像素。

【例 7-4】 FlowLayout 流布局管理器布局效果。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
public class TestFlowLayout extends JFrame{
    public TestFlowLayout(){
        //指定布局管理器，其中参数 FlowLayout.CENTER 可省略
        setLayout(new FlowLayout(FlowLayout.CENTER));
        for(int i=1;i<9;i++)
            add(new JButton("JButton"+i));
    }
    public static void main(String[] args){
        TestFlowLayout frame=new TestFlowLayout();
        frame.setTitle("测试 FlowLayout");
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

程序运行结果如图 7-4 所示。

图 7-4　FlowLayout 流布局管理器布局效果

在默认的情况下，FlowLayout 采用默认居中对齐的方式，如果改变 JFrame 的大小，则每个 JButton 的尺寸保持不变，不过 JButton 和 JFrame 的相对位置发生了变化，如图 7-5 和图 7-6 所示为拖动窗口改变窗口大小后的效果。

图 7-5　FlowLayout 流布局拖动窗口长度后的效果

图 7-6　FlowLayout 流布局拖动窗口宽度后的效果

7.2.3　BorderLayout 边界布局管理器

BorderLayout 是 Frame、JFrame 类的默认布局管理器，它将整个容器的布局划分成了东、西、南、北、中 5 个区域，组件只能被添加到指定的区域，若不指定组件的加入部位，则默认添加到中间区域，每个区域只能加入一个组件，如果加入多个，则先前加入的会被覆盖掉。

下面是 BorderLayout 边界布局管理器的尺寸缩放原则：

(1)北南两个区在水平方向上缩放。

(2)东西两个区在垂直方向上缩放。

(3)中部可以在以上两个方向上缩放。

BorderLayout 的构造方法有以下两个：

(1)public BorderLayout()

创建 BorderLayout 对象，组件之间没有水平和垂直间距。

(2)public BorderLayout(int hgap,int vgap)

以参数指定的水平和垂直间距创建 BorderLayout 对象。

【例 7-5】 BorderLayout 边界布局管理器布局效果。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
public class TestBorderLayout extends JFrame{
    public TestBorderLayout(){
        setLayout(new BorderLayout());  //指定布局管理器,本例可省略
        add(new JButton("North"),BorderLayout.NORTH);
        add(new JButton("South"), BorderLayout.SOUTH);
        add(new JButton("East"), BorderLayout.EAST);
        add(new JButton("West"), BorderLayout.WEST);
        add(new JButton("Center"), BorderLayout.CENTER);//方法调用中的
                                                        //第 2 个参数可省略
    }
    public static void main(String[] args){
        TestBorderLayout frame=new TestBorderLayout();
        frame.setTitle("测试 BorderLayout");
        frame.setSize(300,200);
```

```
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

程序运行结果如图 7-7 所示。

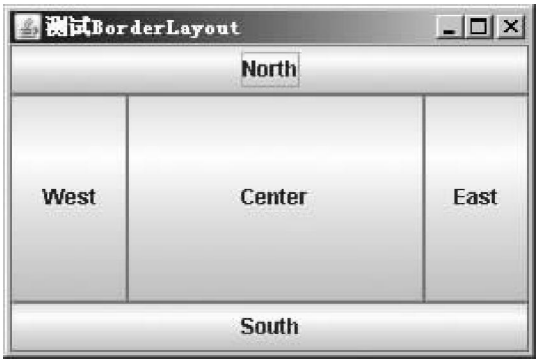

图 7-7　BorderLayout 边界布局管理器布局效果

7.2.4　GridLayout 网格布局管理器

GridLayout 网格布局管理器将容器划分成规则的网格，各个单元格区域大小相等，添加到容器中的组件首先放置在左上角的网格中，然后从左到右放置其他组件，直到占满该行的所有网格，接着继续在下一行中从左到右放置组件。

GridLayout 的构造方法有以下 3 个：

• public GridLayout()

创建 GridLayout 对象，每行中只有一列。

• public GridLayout(int rows,int cols)

以参数指定的行数和列数创建 GridLayout 对象，组件之间的水平和垂直间距为 0。

• public GridLayout(int rows,int cols,int hgap,int vgap)

以参数指定的行数、列数以及组件之间的水平和垂直间距创建 GridLayout 对象。

【例 7-6】　GridLayout 网格布局管理器布局效果。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
public class TestGridLayout extends JFrame{
    public TestGridLayout(){
    setLayout(new GridLayout(3,4));  //创建布局管理器
    for(int i=1;i<9;i++)
        add(new JButton("JButton"+i));
    }
    public static void main(String[] args){
        TestGridLayout frame=new TestGridLayout();
        frame.setTitle("测试 GridLayout");
```

```
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

程序运行结果如图 7-8 所示。

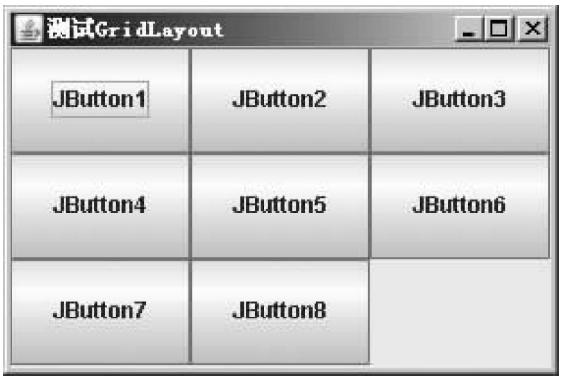

图 7-8 GridLayout 网格布局管理器布局效果

7.2.5 CardLayout 卡片布局管理器

CardLayout 将容器当做一个卡片盒，把添加到容器中的每一个组件当做一张卡片，每次只有一张卡片是可见的。为使某个组件可见，可以调用 CardLayout 对象的如下方法：

- public void first(Container parent)
- public void next(Container parent)
- public void previous(Container parent)
- public void last(Container parent)
- public void show(Container parent, String name)

上述方法中的参数 parent 必须是真正容纳组件的容器类对象，即它可以是 JFrame、JApplet 等的内容窗格或 JPanel 等的对象，但不能直接接收 JFrame、JApplet 等组件的对象。

将组件添加到 CardLayout 容器中时需调用方法：

```
add(Component comp,String name)
```

CardLayout 的构造方法有：

- public CardLayout()
- public CardLayout(int hgap,int vgap)

【例 7-7】 CardLayout 卡片布局管理器布局效果。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestCardLayout extends JFrame{
```

```
    private CardLayout cl=new CardLayout();
    private Container c=getContentPane();
    public TestCardLayout(){
        JButton b1=new JButton("First");
        JButton b2=new JButton("Second");
        JButton b3=new JButton("Third");
        setLayout(cl);
        add(b1,"First Card");
        add(b2,"Second Card");
        add(b3,"Third Card");
        b1.addActionListener(new B());
        b2.addActionListener(new B());
        b3.addActionListener(new B());
    }
    class B implements ActionListener{
        public void actionPerformed(ActionEvent e){
            cl.next(c);  //c 必须是内容窗格
        }
    }
    public static void main(String[] args){
        TestCardLayout frame=new TestCardLayout();
        frame.setTitle("测试 CardLayout");
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

程序运行结果如图 7-9 所示。

图 7-9　CardLayout 卡片布局管理器布局效果

【例 7-7】中采用的卡片布局，首次运行只有一个卡片是可见的，即标签为 First 的按钮所在的卡片，当单击 First 按钮时，就会响应请求，显示下一个卡片，即标签为 Second 的按钮所

在的卡片，依次类推，直到显示完程序设置为止。

7.3 Java GUI 事件处理

当用户与图形用户界面交互，如移动鼠标、单击鼠标、单击按钮、在文本框内输入文本、选择菜单项和关闭窗口时，图形用户界面就会接收到相应的事件，这就需要进行事件处理，本节主要介绍 Java 的事件处理机制。

7.3.1 Java 事件处理概述

要能够让图形用户界面接收用户的操作，就必须给各个组件加上事件处理机制。在事件处理的过程中，主要涉及 3 类对象：

(1)Event：事件，用户对界面操作在 Java 语言上的描述，以类的形式出现，如键盘操作对应的事件类是 KeyEvent。

(2)Event Source：事件源，事件发生的场所，通常就是各个组件，如按钮 JButton。

(3)Event Handler：事件处理者(监听器)，接收事件对象并对其进行处理。

例如，如果用户单击了按钮对象 b，则该按钮 b 就是事件源，而 Java 运行时系统会生成 ActionEvent 类的对象 e，该对象中描述了该单击事件发生时的一些信息，然后，事件处理者对象将接收由 Java 运行时系统传递过来的事件对象 e 并进行相应的处理。

由于同一个事件源上可能发生多种事件，因此 Java 采取了授权处理机制(delegation model)，事件源可以把在其自身所有可能发生的事件分别授权给不同的事件处理者来处理。例如，在 JFrame 对象上既可能发生鼠标事件，也可能发生键盘事件，该 JFrame 对象就可以授权给事件处理者来处理鼠标事件和键盘事件。

有时也将事件处理者称为监听器，主要原因在于监听器时刻监听着事件源上所有发生的事件类型，一旦该事件类型与自己所负责处理的事件类型一致，就马上进行处理。授权模型把事件的处理委托给外部的处理实体进行处理，实现了将事件源和监听器分开的机制。

事件处理者(监听器)通常是一个类，该类如果要能够处理某种类型的事件，就必须实现与该事件类型相对应的接口。

使用授权处理模型进行事件处理的一般方法归纳如下：

(1)对于某种类型的事件×××Event，要想接收并处理这类事件，必须定义相应的事件监听器类，该类需要实现与该事件相对应的接口×××Listener。

(2)事件源实例化以后，必须进行授权，注册该类事件的监听器，使用 add×××Listener(×××Listener)方法来注册监听器。

下面通过一个例子演示一下 Java 事件处理的过程。

【例 7-8】 通过按钮演示 Java 事件处理的过程。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestButtonEvent{
```

```
    public static void main(String args[]){
        JFrame f=new JFrame("TestButtonEvent");
        JButton b=new JButton("单击我!");
        b.addActionListener(new ButtonHandler());
        //注册监听器进行授权,该方法的参数是事件处理者对象,要处理的事件类型
        //可以从方法名中看出,如本方法要授权处理的是 ActionEvent,因为方法名
        //是 addActionListener
        f.setLayout(new FlowLayout()); //设置布局管理器
        f.add(b);
        f.setSize(200,100);
        f.setVisible(true);
    }
}
class ButtonHandler implements ActionListener{
    //实现接口 ActionListener 才能做事件 ActionEvent 的处理者
    public void actionPerformed(ActionEvent e){ //系统产生的 ActionEvent 事件
                                                //对象被当做参数传递给该方法
        System.out.println("事件发生!");
        //本接口只有一个方法,因此事件发生时,系统会自动调用本方法
    }
}
```

程序运行结果如图 7-10 所示。

图 7-10　演示 Java 事件处理的过程

7.3.2　Java 常用事件与事件监听器

1. Java 常用事件

与图形用户界面有关的所有事件类都由 java.awt.AWTEvent 类派生，它也是 EventObject 类的子类。

java.util.EventObject 类是所有事件对象的基础父类，所有事件都是由它派生出来的。AWT 的相关事件继承于 java.awt.AWTEvent 类。Java 事件共有 10 种，可以归纳为两大

类:低级事件和高级事件。低级事件是指基于组件和容器的事件,如鼠标的进入、单击、拖放等,或组件的窗口开关等触发的组件事件;高级事件是基于语义的事件,它可以不和特定的动作相关联,而依赖于触发此事件的类,如在 TextField 中按 Enter 键会触发 ActionEvent 事件,移动滚动条上的滑块会触发 AdjustmentEvent 事件,或是选中项目列表的某一条就会触发 ItemEvent 事件,等等。

1)低级事件

- ComponentEvent(组件事件:组件尺寸的变化、移动)。
- ContainerEvent(容器事件:组件增加、移动)。
- WindowEvent(窗口事件:关闭窗口、窗口闭合、图标化)。
- FocusEvent(焦点事件:焦点的获得和丢失)。
- KeyEvent(键盘事件:键按下、释放)。
- MouseEvent(鼠标事件:鼠标单击、移动)。

2)高级事件

- ActionEvent(动作事件:按钮按下,光标在 TextField 中按 Enter 键)。
- AdjustmentEvent(调节事件:在滚动条上移动滑块以调节数值)。
- ItemEvent(项目事件:选择项目)。
- TextEvent(文本事件:文本对象改变)。

2.事件监听器

每类事件都有对应的事件监听器,监听器是接口,根据动作来定义方法。例如,与键盘事件 KeyEvent 相对应的接口是:

```
public interface KeyListener extends EventListener{
    public void keyPressed(KeyEvent ev);
    public void keyReleased(KeyEvent ev);
    public void keyTyped(KeyEvent ev);
}
```

在本接口中有 3 个方法,Java 运行时,根据这 3 个方法的方法名系统就能够知道应该是什么时候调用哪个方法。当按键刚按下去时,将调用 keyPressed()方法执行;当按键弹起来时,将调用 keyReleased()方法执行;当按键敲击一次时,将调用 keyTyped()方法执行。

又例如,与窗口事件相对应的接口:

```
public interface WindowListener extends EventListener{
    public void windowClosing(WindowEvent e);
    //把退出窗口的语句写在本方法中
    public void windowOpened(WindowEvent e);
    //窗口打开时调用
    public void windowIconified(WindowEvent e);
    //窗口图标化时调用
    public void windowDeiconified(WindowEvent e);
    //窗口非图标化时调用
    public void windowClosed(WindowEvent e);
    //窗口关闭时调用
```

```
    public void windowActivated(WindowEvent e);
    //窗口激活时调用
    public void windowDeactivated(WindowEvent e);
    //窗口非激活时调用
  }
```

组件类中提供注册和注销监听器的方法。

注册监听器：

```
public void add<ListenerType> (<ListenerType>listener);
```

注销监听器：

```
public void remove<ListenerType> (<ListenerType>listener);
```

例如，JButton 类：

```
public class JButton extends JComponent{
    public synchronized void addActionListener(ActionListener l);
    public synchronized void removeActionListener(ActionListener l);
}
```

表 7-1 列出了所有 Java 事件及其相应的监听器接口，一共 10 类事件，11 个接口。

表 7-1　Java 事件及其相应的监听器接口

事件类别	描述信息	接　口　名	方　　法
ActionEvent	激活组件	ActionListener	actionPerformed(ActionEvent)
ItemEvent	选择了某些项目	ItemListener	itemStateChanged(ItemEvent)
MouseEvent	鼠标移动	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
	鼠标单击等	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
KeyEvent	键盘输入	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
FocusEvent	组件收到或失去焦点	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
AdjustmentEvent	移动了滚动条等组件	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
ComponentEvent	对象移动、缩放、显示、隐藏等	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)

续表

事件类别	描述信息	接 口 名	方 法
WindowEvent	窗口收到窗口级事件	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
ContainerEvent	容器中增加或删除了组件	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
TextEvent	文本字段或文本区发生改变	TextListener	textValueChanged(TextEvent)

下面的例子详细展示了 Java 事件模型的应用。

【例 7-9】 Java 事件模型的应用。

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ThreeListener implements MouseMotionListener,MouseListener,
WindowListener{
    //实现了 3 个接口
    private JFrame f;
    private JTextField tf;
    public static void main(String args[]){
        ThreeListener two=new ThreeListener();
        two.go();
    }
    public void go(){
        f=new JFrame("ThreeListener");
        f.add(new JLabel("单击和移动鼠标"),"North");
        tf=new JTextField(30);
        f.add(tf,"South");                    //使用默认的布局管理器
        f.addMouseMotionListener(this);       //注册监听器 MouseMotionListener
        f.addMouseListener(this);             //注册监听器 MouseListener
        f.addWindowListener(this);            //注册监听器 WindowListener
        f.setSize(300,200);
        f.setVisible(true);
    }
    public void mouseDragged (MouseEvent e){
```

```
        //实现 mouseDragged 方法
        String s="鼠标移动 : X="+e.getX()+",Y="+e.getY();
        tf.setText(s);
    }
    public void mouseMoved(MouseEvent e){}
    //对其不感兴趣的方法可以设置方法体为空
    public void mouseClicked(MouseEvent e){}
    public void mouseEntered(MouseEvent e){
        String s="鼠标进入";
        tf.setText(s);
    }
    public void mouseExited(MouseEvent e){
        String s="鼠标离开窗口";
        tf.setText(s);
    }
    public void mousePressed(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}
    public void windowClosing(WindowEvent e){
        //为了使窗口能正常关闭,程序正常退出,需要实现 windowClosing()方法
        System.exit(1);
    }
    public void windowOpened(WindowEvent e){}
    //对其不感兴趣的方法可以设置方法体为空
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowClosed(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
}
```

【例 7-9】程序运行时,鼠标进入窗口内部时,会显示“鼠标进入”;移出窗口内部时,会显示“鼠标离开窗口”;在窗口内部拖动鼠标时,会显示鼠标移动的坐标位置。

【例 7-9】有如下几个特点:

(1)可以声明多个接口,接口之间用逗号隔开。

```
implements MouseMotionListener,MouseListener, WindowListener;
```

(2)可以由同一个对象监听一个事件源上发生的多种事件:

```
f.addMouseMotionListener(this);
f.addMouseListener(this);
f.addWindowListener(this);
```

则对象 f 上发生的多个事件都将被同一个监听器接收和处理。

(3)事件处理者和事件源处在同一个类中。本例中事件源是 JFrame f,事件处理者是类

ThreeListener,其中,事件源 JFrame f 是类 ThreeListener 的成员变量。

(4)可以通过事件对象获得详细资料,如本例中就是通过事件对象获得了鼠标移动时的坐标值。

7.4 常用组件

Swing 组件是 AWT 组件的扩展,它提供了更强大和更灵活的组件集合。除了那些常用组件,如按钮、复选框和标签外,Swing 还包括许多新的组件,如选项板、滚动窗口、树和表格等。同时,即使是诸如按钮等常用组件,在 Swing 中也都增加了新功能。

这部分内容相当多,在此只将一些常用组件的常见用法简单说明一下,以便读者能够掌握 Swing 组件的主要功能和特点。

7.4.1 按钮和标签

1.按钮

按钮(JButton)是一个常用组件,按钮可以带标签或图像。

常用的构造方法有:

```
JButton( )                          //按钮上既无字也无图标
JButton(Icon icon)                  //按钮上显示图标
JButton(String text)                //按钮上显示字符
JButton(String text, Icon icon)     //按钮上既显示图标又显示字符
```

【例 7-10】 采用 Swing 技术实现不同风格的按钮。

参考程序如下:

```
import java.awt.*;
import javax.swing.*;
public class TestJButton extends JFrame{
    public TestJButton(){
        setLayout(new FlowLayout());
        JButton b1=new JButton("测试按钮");
        JButton b2=new JButton(new ImageIcon("bluej.jpg"));
        JButton b3=new JButton("logo",new ImageIcon("bluej.jpg"));
        add(b1);add(b2);add(b3);
    }
    public static void main(String[] args){
        TestJButton frame=new TestJButton();
        frame.setTitle("测试不同的按钮");
        frame.setSize(300,240);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    }
}
```

程序运行结果如图 7-11 所示。

图 7-11　Swing 技术实现不同风格的按钮

2.标签

标签(JLabel)是一种可以显示文字内容的简单组件,它通常在窗体中起到信息提示的作用,相应的 Swing 类为 JLabel。

常用的构造方法有:

```
JLabel()              //创建一个空标签
JLabel(Icon icon)  //创建一个带指定文本的标签
JLabel(String text)//创建一个具有指定图像的标签
```

【例 7-11】 利用标签显示文本和图片。

参考程序如下:

```
import java.awt.*;
import javax.swing.*;
public class TestJLabel extends JFrame{
    public TestJLabel(){
        setLayout(new FlowLayout());
        JLabel l1=new JLabel("BlueJ 图标");
        JLabel l2=new JLabel(new ImageIcon("java.gif"));
        add(l1);add(l2);
    }
    public static void main(String[] args){
        TestJLabel frame=new TestJLabel();
        frame.setTitle("测试不同的标签");
        frame.setSize(400,300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

程序运行结果如图 7-12 所示。

图 7-12　利用标签显示文本和图片

7.4.2　单选按钮和复选框

1.单选按钮

JRadioButton 表示单选按钮，在使用的时候把一组单选按钮加入一个按钮组(ButtonGroup)中，在任何时候，用户只能选择按钮组中的一个按钮，当用户选中一个单选按钮时，将触发一个 ActionEvent 事件，可以用 ActionListener 来响应这个事件。

【例 7-12】　测试单选按钮的使用。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestJRadioButton extends JFrame{
    private Icon[] icons={
            new ImageIcon("bird.gif"),
            new ImageIcon("cat.gif"),
            new ImageIcon("dog.gif"),
            new ImageIcon("pig.gif"),
            new ImageIcon("rabbit.gif")};
    private JRadioButton[] rb={
          new JRadioButton("鸟",true),
          new JRadioButton("猫",false),
          new JRadioButton("狗",false),
          new JRadioButton("猪",false),
          new JRadioButton("兔",false)};
    private ButtonGroup bg=new ButtonGroup();
    private JLabel jl=new JLabel(icons[0]);
    public TestJRadioButton(){
        JPanel jpRadioButton=new JPanel();
```

```
        jpRadioButton.setLayout(new GridLayout(5,1));
        ItemListener il=new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                int n=0;
                JRadioButton rbe=(JRadioButton)e.getSource();
                String srbe=rbe.getText();
                for(int i=0;i<rb.length;i++){
                    if(srbe.equals(rb[i].getText())){
                        n=i;
                        break;
                    }
                }
                jl.setIcon(icons[n]);
            }
        };
        for(int i=0;i<rb.length;i++){
            rb[i].addItemListener(il);
            bg.add(rb[i]);
            jpRadioButton.add(rb[i]);
        }
        setLayout(new BorderLayout());
        add(jl,BorderLayout.CENTER);
        add(jpRadioButton,BorderLayout.WEST);
    }
    public static void main(String[] args){
        TestJRadioButton frame=new TestJRadioButton();
        frame.setTitle("测试 JRadioButton");
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
    }
}
```

程序运行结果如图 7-13 所示，选中 5 个单选按钮中的任意一个，就会出现相应的动物图案。

图 7-13　测试单选按钮的使用

2. 复选框

JCheckBox 表示复选框。用户可以同时选择多个复选框，当用户选中或者取消选中一个复选框时，将触发一个 ActionEvent 事件，可以用 ActionListener 来响应这个事件。

下面通过一个例子来测试一下复选框的使用。

【例 7-13】 测试一下复选框的使用。

参考程序如下：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class TestJCheckBox extends JFrame{
    private JTextArea area=new JTextArea(6, 15);
    private JScrollPane paneWithTextArea=new JScrollPane(area);
    private JPanel paneWithButtons=new JPanel();
    private JLabel label1=new JLabel("兴趣:");
    private JCheckBox
    cb1=new JCheckBox("游泳");
    cb2=new JCheckBox("唱歌");
    cb3=new JCheckBox("旅游");
    private JLabel label2=new JLabel("性别:");
    private JRadioButton
    rb1=new JRadioButton("男",true);                    //默认为选中状态
    rb2=new JRadioButton("女");
    private ButtonGroup group=new ButtonGroup();
    private String sex="男";                            //性别
    private Set<String> hobbies=new HashSet<String>();//兴趣爱好
    private ActionListener listener1=new ActionListener(){
        //监听 JCheckBox 触发的 ActionEvent
        public void actionPerformed(ActionEvent event){
            JCheckBox cb=(JCheckBox)event.getSource();
            if(cb.isSelected())
                hobbies.add(cb.getText());
            else
                hobbies.remove(cb.getText());
            printStatus();
        }
    };
    private ActionListener listener2 =new ActionListener(){
        //监听 JRadioButton 触发的 ActionEvent
        public void actionPerformed(ActionEvent event){
```

```
            JRadioButton rb=(JRadioButton)event.getSource();
            sex=rb.getText();
            printStatus();
        }
    };
    public TestJCheckBox(String title){
        super(title);
        area.setEditable(false);
        cb1.addActionListener(listener1);
        cb2.addActionListener(listener1);
        cb3.addActionListener(listener1);
        rb1.addActionListener(listener2);
        rb2.addActionListener(listener2);
        //把 JRadioButton 加入一个 ButtonGroup 中
        group.add(rb1);
        group.add(rb2);
        paneWithButtons.setLayout(new FlowLayout());
        paneWithButtons.add(label1);
        paneWithButtons.add(cb1);
        paneWithButtons.add(cb2);
        paneWithButtons.add(cb3);
        paneWithButtons.add(label2);
        paneWithButtons.add(rb1);
        paneWithButtons.add(rb2);
        Container contentPane=getContentPane();
        //默认的布局管理器为 BorderLayout
        contentPane.add(paneWithButtons,BorderLayout.NORTH);
        contentPane.add(paneWithTextArea,BorderLayout.CENTER);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
    private void printStatus(){
        area.append("您的兴趣爱好包括:");
        Iterator<String> it=hobbies.iterator();
        while(it.hasNext())
            area.append(it.next()+" ");
        area.append(" 您的性别为:"+sex+"\n" );
    }
    public static void main(String[] args){
```

```
            new TestJCheckBox("Hello");
        }
    }
```

程序运行结果如图 7-14 所示，当用户选中或取消选中一个复选框，以及选中一个单选按钮时，在文本区域内就会显示用户当前的兴趣爱好和性别。

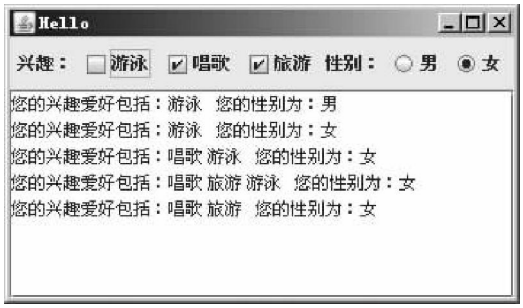

图 7-14 测试一下复选框的使用

7.4.3 文本框

文本框也被称为文本域，它是一种可以接收用户字符输入的常见组件，如用户名和密码等，相应的 Swing 类为 JTextField 和 JPasswordField。

【例 7-14】 测试两类文本框的使用。

参考程序如下：

```
import java.awt.*;
import javax.swing.*;
public class TestTextField{
    public static void main(String[] args){
        JFrame f=new JFrame("测试文本框");
        JLabel jl1=new JLabel("用户姓名:");
        JTextField jtf=new JTextField("请输入姓名");
        jtf.setColumns(10);
        JLabel jl2=new JLabel("用户密码:");
        JPasswordField jpf=new JPasswordField();
        jpf.setColumns(10);
        jpf.setEchoChar('*');
        f.setLayout(new GridLayout(2,2));
        f.add(jl1);
        f.add(jtf);
        f.add(jl2);
        f.add(jpf);
        f.setSize(160,80);
        f.setVisible(true);
    }
}
```

程序运行结果如图 7-15 所示。

图 7-15　测试两类文本框的使用

7.4.4　文本区域和滚动面板

JTextField 表示文本框，只能输入一行文本，而 JTextArea 表示文本区域，可以输入多行文本。当用户在文本框中按 Enter 键时，将触发一个 ActionEvent 事件；当用户在文本区域中按 Enter 键时，仅仅意味着换行输入文本，并不会触发 ActionEvent 事件。

JScrollPane 表示带滚动条的面板，在默认情况下，只有当面板中的内容超过了面板的面积时，才会显示滚动条。

【例 7-15】　测试文本区域和滚动面板的使用。

参考程序如下：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
public class TestTextArea extends JFrame{
    JTextArea area1=new JTextArea(5,10); //创建 5 行 10 列的文本区域
    JTextArea area2=new JTextArea(5,10); //创建 5 行 10 列的文本区域
    //在垂直方向总是显示滚动条,在水平方向只有当需要的时候才显示滚动条
    JScrollPane panel =new JScrollPane(area1,
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    //pane2 按默认方式创建,在垂直和水平方向都只有当需要的时候才显示滚动条
    JScrollPane pane2=new JScrollPane(area2);
    JButton copyButton=new JButton("Copy");
    public TestTextArea(String title){
        super(title);
        copyButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent event){
                //把第一个文本区域中被选中的文本复制到第二个文本区域中
                area2.setText(area1.getSelectedText());
            }
        });
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(panel);
```

```
        contentPane.add(copyButton);
        contentPane.add(pane2);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
    public static void main(String[] args){
        new TestTextArea("Hello");
    }
}
```

程序运行结果如图 7-16 所示，从图中可以看出，第一个文本区域在垂直方向总是显示滚动条。

图 7-16　测试文本区域和滚动面板的使用

7.4.5　下拉列表框

JComboBox 表示下拉列表框。下拉列表框和单选按钮一样，也可以提供多个选项，并且只允许用户选择一项。下拉列表框的优点在于能节省空间，使界面更加紧凑。只有当用户单击下拉列表框右侧的下三角按钮时，才会显示列表中的所有项。

【例 7-16】 测试下拉列表框的使用。

参考程序如下：

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class TestJComboBox extends JFrame{
    private String[] cities={
        "北京","上海","南京","深圳",
        "济南","沈阳","苏州","杭州","常州"
    };
    private JTextField textField=new JTextField(15);
    private JComboBox comboBox=new JComboBox();
    private JButton button=new JButton("添加更多城市选项");
    private int count=0;
    public TestJComboBox(String title){
        super(title);
        for(int i=0;i<4;i++)
```

```
            comboBox.addItem(cities[count++]);
        button.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(count < cities.length)
                    comboBox.addItem(cities[count++]);
            }
        });
        comboBox.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                textField.setText("index: "+ comboBox.getSelectedIndex() +
                " " +comboBox.getSelectedItem());
            }
        });
        comboBox.setEditable(true); //使下拉列表框可以被编辑
        textField.setEditable(false);
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(comboBox);
        contentPane.add(button);
        contentPane.add(textField);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
    public static void main(String[] args){
        new TestJComboBox("TestJComboBox");
    }
}
```

程序运行结果如图 7-17 所示。

图 7-17　测试下拉列表框的使用

7.4.6　菜单

菜单是一种常用的组件，它包含以下几项：

(1)JMenuItem：菜单项，直接指向一个具体的操作，对应一项功能，选定菜单项组件会

引发 ActiveEvent 和 ItemEvent 等事件。

(2)JMenu:菜单,菜单项的集合,一个菜单中可以加入多个菜单项、其他菜单和分隔符。

(3)JMenuBar:菜单栏,菜单的集合,一个菜单栏中可加入多个菜单。

实现菜单的过程如下:

(1)创建菜单栏,并将它设置到某个容器中:

```
JMenuBar mb=new JMenuBar();
JFrame frame=new JFrame();
frame.setJMenuBar(mb);
```

(2)创建菜单条,并将它们添加到菜单栏中:

```
JMenu fileMenu=new JMenu("文件");
JMenu editMenu=new JMenu("编辑");
mb.add(fileMenu);
mb.add(editMenu);
```

(3)创建菜单项,并将它们添加到菜单条中:

```
JMenuItem[] editm={
    new JMenuItem("剪切"),
    new JMenuItem("复制"),
    new JMenuItem("粘贴"),
    new JMenuItem("全选")
};
for(int i=0;i<editm.length;i++){
    editMenu.add(editm[i]);
    if(i==2) editMenu.addSeparator();
}
```

接下来通过一个例子分析一下菜单的使用。

【例 7-17】 测试菜单的使用。

参考程序如下:

```
import java.awt.event.*;
import javax.swing.*;
public class TestJMenu extends JFrame{
    private JTextArea t=new JTextArea();
    private JMenuBar mb=new JMenuBar();
    private JMenu fileMenu=new JMenu("文件");
    private JMenu viewMenu=new JMenu("视图");
    private JMenu toolMenu=new JMenu("工具栏");
    private JMenuItem[] filem={
        new JMenuItem("新建"),new JMenuItem("打开"),
        new JMenuItem("保存"),new JMenuItem("退出")
    };
    private JMenuItem[] viewm={
        new JMenuItem("普通"),new JMenuItem("页面"),
```

```
        new JMenuItem("大纲")
    };
    private JCheckBoxMenuItem[] toolm={
        new JCheckBoxMenuItem("常用"),
        new JCheckBoxMenuItem("绘图"),
        new JCheckBoxMenuItem("符号栏")
    };
    public TestJMenu(String s){
        super(s);
        ActionListener al=new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(((JMenuItem)e.getSource()).getText()=="退出")
                    System.exit(0);
                else
                    t.setText(((JMenuItem)e.getSource()).getText());
            }
        };
        ItemListener il=new ItemListener(){
            public void itemStateChanged(ItemEvent e)
            {
                String s=new String("");
                for(int i=0;i<toolm.length ;i++)
                    if (toolm[i].isSelected())
                        s=s+toolm[i].getText()+"\n";
                if(s.length()==0)
                    t.setText("没有复选");
                else
                    t.setText(s);
            }
        };
        for(int i=0;i<filem.length;i++)
        {
            filem[i].addActionListener(al);
            fileMenu.add(filem[i]);
            if(i==2) fileMenu.addSeparator();
        }
        for(int i=0;i<viewm.length;i++)
        {
            viewm[i].addActionListener(al);
            viewMenu.add(viewm[i]);
```

```
        }
        viewMenu.addSeparator();
        viewMenu.add(toolMenu);
        for(int i=0;i<toolm.length;i++)
        {
            toolm[i].addItemListener(il);
            toolMenu.add(toolm[i]);
        }
        mb.add(fileMenu);
        mb.add(viewMenu);
        setJMenuBar(mb);
        add(new JScrollPane(t));
    }
    public static void main(String[] args){
        TestJMenu f=new TestJMenu("TestJMenu");
        f.setSize(300,200);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
    }
}
```

程序运行结果如图 7-18 所示。

图 7-18　测试菜单的使用

习　题　7

1. Swing 组件与 AWT 组件有什么区别？
2. 简述在 JFrame 窗口中放置组件的步骤。
3. 什么是事件源、事件和事件监听器？
4. 利用 Swing 组件实现一个计算机界面，并能进行加减乘除运算。
5. 使用 Swing 组件的菜单实现一个简单的记事本功能。