第3章 GUI交互功能设计——事件处理

Java程序通过对用户操作的响应实现与用户的交互，主要工作就是对由于用户操作触发的GUI事件进行处理。本章介绍Java GUI程序的事件处理概念和机制，详细介绍事件监听器的设计方法，通过实例介绍常用事件及其监听器接口的实现方法；介绍使用SwingWorker改进程序GUI反应速度和性能的原理及方法。

3.1 事件处理的概念及委托事件处理模型

Java GUI程序通过事件循环反复检测用户在界面上的操作来处理程序与用户的交互。图1.13的程序(程序清单1.1)中，每当用户单击“求和”“清除”和“退出”按钮时，都会触发事件处理的执行。

3.1.1 事件的概念

所谓事件就是发送给GUI系统的消息，该消息通知GUI系统某种事情已经发生，要求做出响应。用户在界面组件上执行了操作，将导致事件发生。

Java中的事件用对象描述——描述事件的发生源、事件的类别、事件发生前和发生后组件状态的变化等。如程序清单1.1中，当用户在按钮上单击鼠标时，发生一个ActionEvent类型的事件，此时Java运行时环境自动产生一个ActionEvent类的对象。

引发产生事件的组件对象称为事件源，如程序清单1.1中的jButton1、jButton2和jButton3对象即事件源。根据来源事件可分为以下几种。

(1) 计算机输入输出设备产生的中断事件，如鼠标和键盘与GUI系统的交互操作。这种事件是原生的底层事件，一般都需要组件做进一步处理，由此触发更高抽象层次的逻辑事件。

(2) GUI系统触发的逻辑事件。这种事件是上面所说的原始事件经过组件的处理后派发的高级事件，如程序清单1.1中单击jButton1产生的ActionEvent e、通知界面重绘的事件等。

(3) 应用程序触发的事件。有以下两种方式。

① 通过将事件添加到系统事件队列进行派发。Swing程序中通过postEvent()、repaint()及invokeLater()等方法，向系统事件队列添加事件。这种触发机制实质上是调度，触发事件的线程和事件派发线程可以不是同一个线程。事件被添加到系统事件队列后触发过程结束，之后要在事件派发线程上等待执行事件的处理代码。

② 通过调用组件的派发方法(Swing中是fireEventXxx())触发。使用这种方法，事件

对象不会发送到系统事件队列，而是直接传递给事件处理方法进行处理。它的触发机制实质上是方法调用。这种事件触发方式要求事件处理线程必须同时是事件派发线程。

3.1.2 事件处理模型

Java GUI 系统对用户在组件上的某些操作（发生的事件）执行特定方法或运行特定程序，从而使用户与 Java GUI 应用程序进行数据交换，或对程序的运行过程进行控制，Java 中把这个过程称为事件处理。

Java 2（JDK 1.1）及之后的版本使用委托事件模型对组件上发生的事件进行监听和处理（见图 3.1）。事件监听器是一个实现了监听器接口的类的实例，在该类中编写发生某种事件的相关动作需要执行的代码。在事件源上通过 addXxxListener()方法给该组件注册发生特定类型事件时对其进行处理的事件监听器。

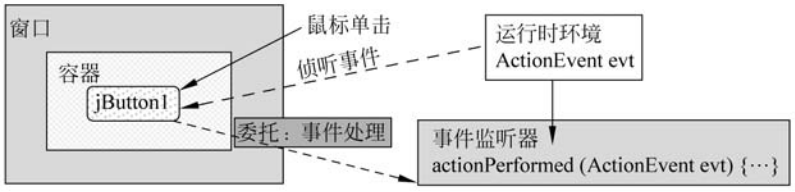

图 3.1 Java 委托事件模型

Java GUI 程序的运行过程中，用户在事件源上做了动作，GUI 平台操作系统会生成 GUI 事件（如鼠标操作）并添加到该程序的事件队列，由工具包线程将底层事件转换并包装成 Swing 的逻辑事件（如 ActionEvent evt），挂入该程序的事件队列中。事件派发线程在事件队列中检测到该事件时，调度事件监听器的相应方法执行事件处理代码，对用户的操作做出响应。

如程序清单 1.1 中的语句：

```
jButton1.addActionListener(new java.awt.event.ActionListener() {//③注册事件监听器
    public void actionPerformed(java.awt.event.ActionEvent evt) {//处理鼠标单击动作
        jButton1ActionPerformed(evt);
    }
});
```

调用了 jButton1 对象的 addActionListener()方法，把实现了 ActionListener 接口的匿名监听器对象注册为它的单击按钮事件 ActionEvent 监听器。这个匿名监听器类中的方法 actionPerformed(ActionEvent evt) 调用方法 jButton1ActionPerformed(evt)（程序清单 1.1 ⑥），实现对两个文本字段中输入数据的格式转换和求和操作，并将和数设置为“运算结果”右侧文本字段的内容，从而实现了该按钮所标注的“求和”功能。也就是说，“求和”按钮注册这个事件监听器，就是委托该事件监听器完成用户单击它时所应该完成的任务。

3.1.3 Swing GUI 事件处理程序的设计步骤

在 Swing GUI 程序中事件处理的设计主要包括以下四个步骤。

（1）定义一个 XxxEvent 事件类，描述 GUI 的 Xxx 事件。

（2）定义一个事件处理器接口 XxxListener，声明所有与该事件相关的处理方法。

（3）在触发事件的组件中定义处理 Xxx 事件的注册方法 addXxxListener()和注销方法 removeXxxListener()。

（4）编写实现事件监听器接口的类，实现具体的事件处理方法。

其中，事件类、事件监听器接口和组件的注册及注销方法已经在 AWT 和 Swing 类库中有明确的定义，可以直接使用。编写实现事件监听器接口的类则是应用程序设计的主要任务。下面介绍事件监听器接口实现类的设计。

3.2 事件处理的设计

在 Java GUI 程序设计中，所谓事件处理的设计就是事件监听器接口实现类的设计。以下介绍具体的设计方法。

3.2.1 实现监听器接口

Java 语言的语法规定，不能直接生成一个接口的对象，但能够以一个或多个接口为基础设计一个类，在该类中实现接口的方法。如果实现了接口的所有方法，则可以生成这个类的对象，即生成一个事件监听器。

例 3.1 在学生成绩管理系统的用户登录界面中，为“登录”按钮设计一个事件监听器，检查输入的用户名和密码是否合法，如果合法则关闭登录窗口，显示一个新窗口欢迎该用户使用该系统。否则清空“用户名”和“密码”文本框，让用户重新输入。

分析：

（1）在一个系统中，同一身份的用户名应该是唯一的。本章简单地使用一个文本文件存放该系统中合法的用户，一行一个用户账户，格式是“用户名:密码:身份”。文件名为 users.txt，存放在项目的根目录下。

（2）单击一个按钮时会产生 ActionEvent 事件，该事件的监听器是 ActionListener，只有一个方法 actionPerformed。因此，可以写一个监听器类实现该方法。

（3）在 actionPerformed()方法中，查找输入的用户名、密码和身份是否在用户账户文件中有匹配的记录。如果有，就关闭登录窗口并显示欢迎窗口，否则清空“用户名”和“密码”文本框。

（4）用面向对象的设计方法，将用户信息设计为一个类 User，包含用户名、密码和身份及相关方法。将全部用户账户设计为一个类 UsersSet，将各个用户信息存放在一个 Set 中，该类提供查找特定用户的方法，若找到，则该方法返回 boolean：true。

解：设计步骤如下。

（1）准备账户文件 users.txt。

右击项目名 StdScoreManager，选择菜单 New→Other 命令，Categories 选择 Other，File Types 选择 Empty File，文件名输入“users.txt”，单击 Finish 按钮。在该文件中输入下列 4 行文字，并保存。

```
zhangsan:123:0
lisi:456:1
```

```
lisi2:123:0
wangwu:456:2
```

(2) 设计用户信息类 User。

右击项目 StdScoreManager 的包 book.chap02.stdscoreui,选择菜单 New→Java Class 菜单项,类名输入“User”,单击 Finish 按钮。

在类中定义两个 String 类型私有实例变量 name 和 password,一个 int 类型私有变量 job,分别存放用户名、密码和身份。并定义三个 int 类型静态常量 STUDENT、TEACHER 和 ADMIN,取值分别为 0、1 和 2。

生成这三个变量的取值/设值方法。选择菜单 Source→Insert Code→Getter and Setter 命令,在新对话框中选择变量 job、name 和 password,单击 Generate 按钮。

生成该类的 equals()方法。选择菜单 Source→Insert Code→equals() and hashCode() 命令,在新对话框中左右两栏均单击 Select All,单击 Generate 按钮。

生成该类的 toString()方法。选择菜单 Source→Insert Code→toString()命令,在新对话框中选择这三个变量,单击 Generate 按钮。

生成该类的构造函数。选择菜单 Source→Insert Code→Constructor 命令,在 Generate Constructor 对话框中单击 Select All,单击 Generate 按钮。

(3) 设计类 UsersSet。

在 book.chap02.stdscoreui 包中创建 UsersSet 类,在该类中定义一个 HashSet<User>类型私有实例变量 usersSet。在 UsersSet 类体中生成无参构造函数,在构造函数中输入以下代码。

```
usersSet = new HashSet<User>() ;
String str = null ;
String[] userStr = null ;
try {
    FileReader fir = new FileReader("..\\users.txt") ;
    BufferedReader bir = new BufferedReader(fir) ;
    while((str = bir.readLine())!= null) {
        userStr = str.split(":") ;
        usersSet.add(new User(userStr[0].trim(),
                              userStr[1].trim(), Integer.parseInt(userStr[2]))) ;
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch(IOException e) {
    e.printStackTrace();
}
```

在输入过程中,可以使用 IDE 的代码辅助功能,如 Source→Fix Imports、自动生成 try/catch 块(Surround Block with try-catch)等。

在该类中编写判断是否合法用户的方法,代码如下。

```
public boolean isValid(User user) {
    boolean userValid = false ;
    if(usersSet.contains(user)) {
```

```
            userValid = true ;
        }
        return userValid ;
    }
```


(4) 设计欢迎窗口。

在包 book. chap02. stdscoreui 中新建一个 JFrame 窗体，类名为 ScoreMana。切换到 ScoreMana 窗体的 Source 视图，在类体的第一行右击，选择快捷菜单中的 Insert Code→Add Property 菜单项，在 Add Property 对话框（见图 3.2）中 Name 输入“user”，值（=后）为“null”，类型输入“User”，其他默认，单击 OK 按钮。

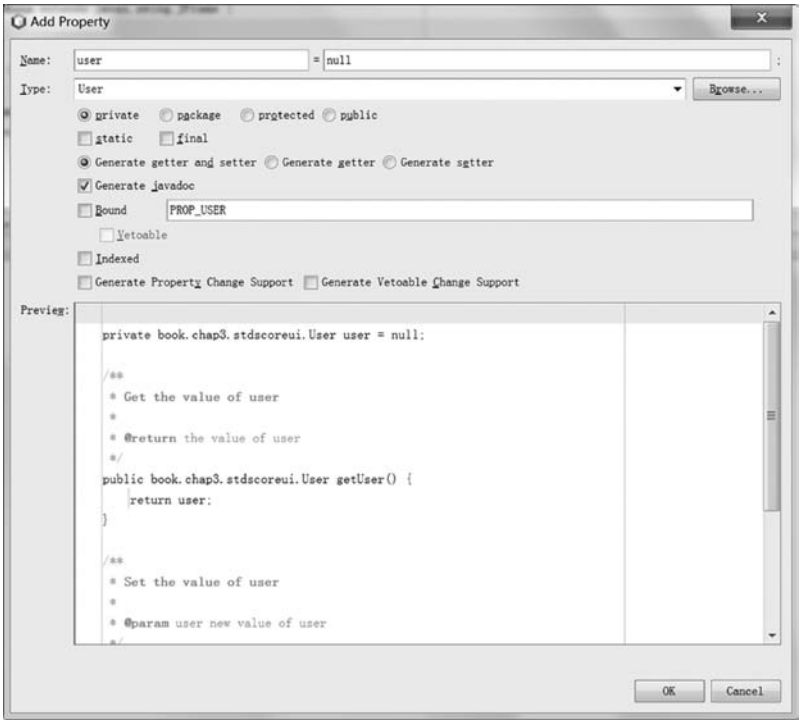

图 3.2　为新建的 ScoreMana 窗体添加 user 属性

在构造方法名处右击，选择快捷菜单中 Refactor→Change Method Parameters 菜单项。在新出现的对话框（见图 3.3）中单击 Add 按钮，参数类型输入“User”，参数名称输入“user”，其他默认，单击 Refactor 按钮完成。在该构造方法的“initComponents();”语句前面添加语句“this. user=user;”。

在该窗体中创建一个 JLabel 组件，设置 text 属性为定制代码：jLabel1. setText("欢迎"+user. getName()+"同学使用本系统。")。

(5) 为登录界面的“登录”按钮添加事件监听器。

在 UserLogin 窗体的 Design 视图中右击“登录”按钮，选择 Events→Action→actionPerformed 菜单项。切换到 Source 视图，单击 [+] Genarated Code 行前面的 [+] 号，展开 initComponents()方法的代码，可以看到以 ActionListener 为父接口生成了一个匿名监听器。

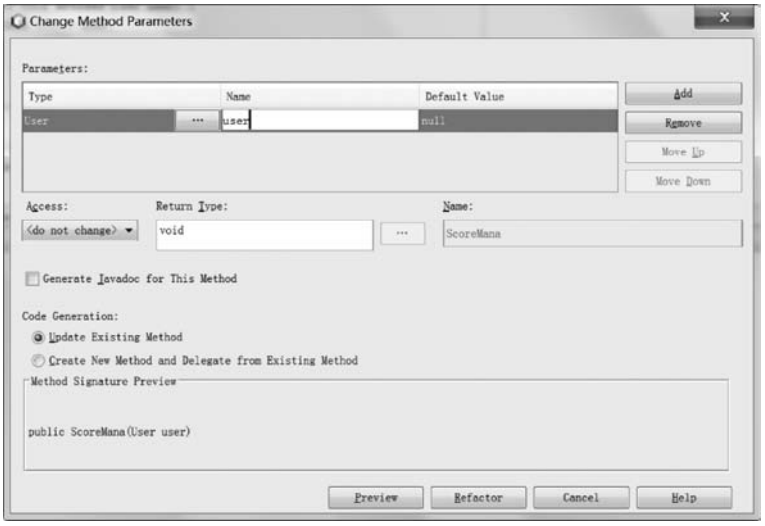

图 3.3　更改构造方法参数对话框

```
jButtonOK.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
          jButtonOKActionPerformed(evt);
     }
});
```

右击 jButtonOKActionPerformed 方法名，选择快捷菜单中的 Navigate→Go to Source 命令，光标转到了 jButtonOKActionPerformed 方法代码处。在方法体中输入以下代码。

```
String name = jTextFieldUserName.getText().trim();
String password = new String(jPasswordFieldLogin.getPassword()).trim();
int job = jRadioButtonStd.isSelected() ? 0 : (jRadioButtonTch.isSelected() ? 1 : 2);
User user = new User(name, password, job);
if(new UsersSet().isValid(user)) {
    new ScoreMana(user).setVisible(true);
    this.dispose() ;
} else {
    jTextFieldUserName.setText("") ;
    jPasswordFieldLogin.setText("") ;
}
```

3.2.2　从事件适配器派生

一些事件监听器接口声明了多个方法，如按键事件监听器接口 KeyListener 就有键按下 keyPressed、键释放 keyReleased 和按键 keyTyped 三个方法声明(见图 3.4)，但有时应用程序只关心一种或少数几种操作，如按键 keyTyped 动作的处理，而不需要处理其他操作。若设计实现事件监听器接口的类，就必须实现所有方法，尽管有些方法并不关心，否则就无法创建该类的对象，也就无法生成事件监听器。为解决这个问题，Swing 类库对具有两个或两个以上方法的事件监听器接口都设计了一个对应的事件适配器类，对各个方法做了空实现。这样，Swing 应用程序的事件监听器就可以从相应的事件适配器类派生，在其子类中只

实现需要的方法，从而减轻了设计工作量。

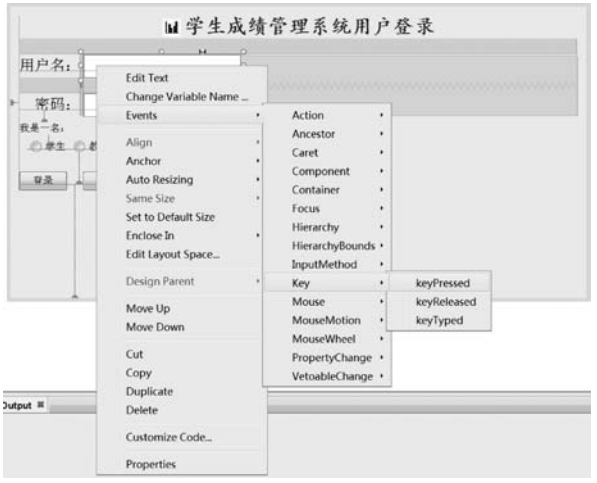

图 3.4　KeyListener 监听器有多个方法

可以这样判断，鼠标移到 Events 子菜单的某个事件上（如 Key），若其级联菜单有多个菜单项，则该事件监听器接口有对应的适配器类。适配器类名一般是将 Listener 换成 Adapter 即可，如按键事件适配器类名为 KeyAdapter。

例 3.2　在学生成绩管理系统的用户登录界面中，规定用户名必须由字母和数字组成，否则为非法用户名。给“用户名”文本框 jTextFieldUserName 设计并注册一个校验器，防止输入非法字符。

分析：文本字段组件可以监听 KeyEvent 事件，在文本字段输入时发生。通过 KeyEvent 对象的 getKeyChar()方法可以获取用户按键所对应的字符。因此，可以设计 KeyEvent 事件的监听器，在 typedText()方法中监测用户输入内容，防止输入非法用户名。

解：设计步骤如下。

（1）打开 StdScoreManager 项目，打开其中的文件 UserLogin.java。

（2）在设计区域右击 jTextFieldUserName 文本框（见图 3.4），选择 Events→Key→keyTyped 菜单项。

（3）在 initComponents()中自动生成代码：

```
jTextFieldUserName.addKeyListener(new java.awt.event.KeyAdapter(){
    public void keyTyped(java.awt.event.KeyEvent evt) {
        jTextFieldUserNameKeyTyped(evt);
    }
});
```

可以看到，从 KeyAdapter 类派生了一个事件监听器类（匿名类），并创建了一个该类的对象作为按键事件监听器。类体中重写了 keyTyped()方法，具体事件处理逻辑在该方法所调用的方法 jTextFieldUserNameKeyTyped()中。方法代码如下。

```
private void jTextFieldUserNameKeyTyped(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
}
```

在 jTextFieldUserNameKeyTyped()方法体内输入以下事件处理代码。

```
char c = evt.getKeyChar() ;
if(!(c >= 'a'&&c <= 'z'||c >= 'A'&&c <= 'Z'||c >= '0'&&c <= '9')) {
    JOptionPane.showMessageDialog(rootPane,
                "输入有误。输入必须是字母和数字,其他字符无效!");
    evt.setKeyChar('\0');
}
```

此段代码获取并检测用户输入的字符。一旦检测到字母和数字之外的字符出现,则执行 if 块。该 if 块中首先显示一个对话框,对输入内容进行提示,然后清除输入的非法字符。

3.2.3 匿名内部事件监听器类

事件监听器类是实现了事件监听器接口或从事件适配器类派生的类。从 GUI 构建器自动生成的代码来看,这个类是一个匿名内部类。例 3.1 的步骤(5)给“登录”按钮生成的监听器代码,通过 new 操作符创建了一个对象,该对象所属的类实现了 java.awt.event.ActionListener 接口,该类没有命名。该对象作为实参直接传递给了 addActionListener()方法,作为“登录”按钮 jButtonOK 的监听器。

同样地,例 3.2 的步骤(3)为输入用户名的文本字段生成的事件监听器代码,也是用 new 操作符创建了一个对象并传递给方法 addKeyListener()作为文本字段 jTextFieldUserName 的监听器。该对象所属的类是从父类 java.awt.event.KeyAdapter 派生而来,重写了父类的方法 keyTyped(),但该类没有名字。

观察这两个匿名类的位置发现,它们都是在类 UserLogin 的内部定义的,且在该外部类的方法 initComponents()中定义,因此它们是匿名局部内部类。Java 的语法规定,内部类的对象可以无限制地访问其所在外部类的任何成员变量和方法。

分析例 3.2 完成后的 UserLogin.java 的程序代码,有如下结构。

```
…
public class UserLogin extends javax.swing.JFrame {
    //(1) 开始: 外部类的构造方法
    public UserLogin() {
        initComponents();
    }
    //结束: 外部类的构造方法
    @SuppressWarnings("unchecked")
    private void initComponents() {//(2) 创建并初始化界面
        …
        jTextFieldUserName = new javax.swing.JTextField();
        …
        jButtonOK = new javax.swing.JButton();
        …
        jTextFieldUserName.setColumns(20);
        jTextFieldUserName.setFont(new java.awt.Font("宋体", 0, 18));
    //(3) 开始: 匿名内部类事件监听器,监听和处理用户名输入
        jTextFieldUserName.addKeyListener(new java.awt.event.KeyAdapter(){
            public void keyTyped(java.awt.event.KeyEvent evt) {
```

```
                jTextFieldUserNameKeyTyped(evt);
            }
        });
    //结束: 匿名内部类事件监听器,监听和处理用户名输入
        …
        jButtonOK.setText("登录");
        jButtonOK.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
    //(4) 开始: 匿名内部类事件监听器,监听和处理单击"登录"按钮操作
        jButtonOK.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonOKActionPerformed(evt);
            }
        });
    //结束: 匿名内部类事件监听器,监听和处理单击"登录"按钮操作
        …
        pack();
    }//initComponents()方法结束
    //(5) 开始: 事件处理方法——处理单击"登录"按钮操作
    private void jButtonOKActionPerformed(java.awt.event.ActionEvent evt) {
        String name = jTextFieldUserName.getText().trim();
        String password = new String(jPasswordFieldLogin.getPassword()).trim();
        int job = jRadioButtonStd.isSelected()?0:(jRadioButtonTch.isSelected()?1:2);
        User user = new User(name, password, job);
        if(new UsersSet().isValid(user)) {
            new ScoreMana(user).setVisible(true);
            this.dispose() ;
        } else {
            jTextFieldUserName.setText("") ;
            jPasswordFieldLogin.setText("") ;
        }
    }
    //结束: 事件处理方法——处理单击"登录"按钮操作
//(6) 开始: 事件处理方法——处理输入用户名按键操作
    private void jTextFieldUserNameKeyTyped(java.awt.event.KeyEvent evt) {
        char c = evt.getKeyChar() ;
        if(!(c>='a'&&c<='z'||c>='A'&&c<='Z'||c>='0'&&c<='9')) {
            JOptionPane.showMessageDialog(rootPane,
                "输入有误。输入必须是字母和数字,其他字符无效!");
            evt.setKeyChar('\0');
        }
    }
    //结束: 事件处理方法——处理输入用户名按键操作
    //(7) 开始: 程序入口 main()方法
    public static void main(String args[]) {
        …
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new UserLogin().setVisible(true);
            }
        });
    }
```

```
    //结束: 程序入口 main()方法

    //(8) 开始: 类中成员变量定义
    private javax.swing.ButtonGroup buttonGroupActor;
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonOK;
    private javax.swing.JLabel jLabelActor;
    private javax.swing.JLabel jLabelPassword;
    private javax.swing.JLabel jLabelTitle;
    private javax.swing.JLabel jLabelUserName;
    private javax.swing.JPasswordField jPasswordFieldLogin;
    private javax.swing.JRadioButton jRadioButtonAdmin;
    private javax.swing.JRadioButton jRadioButtonStd;
    private javax.swing.JRadioButton jRadioButtonTch;
    private javax.swing.JTextField jTextFieldUserName;
    //结束: 类中成员变量定义
}
```

程序运行从块(7)即 main()方法开始,先创建了一个 UserLogin 的对象 this(语句 new UserLogin().setVisible(true);),外部类 UserLogin 构造方法(1)的执行调用块(2)方法(语句"initComponents();"),在输入用户名时调用块(3)→(6),单击"登录"按钮时调用块(4)→(5)。执行块(3)或块(4)时创建了匿名局部内部类的对象并执行该匿名对象的方法。即执行顺序是:创建外部类对象→执行外部类的方法→创建内部类对象→执行内部类方法。

当创建了一个内部类对象时,该内部类对象获得了其所在方法所属的外部类对象的引用,内部类对象通过这个引用可以无限制地访问外部类对象的成员。块(3)和块(4)中直接访问块(8)(即外部类)成员变量正是通过这个引用进行的。

注意:局部内部类对象不可以访问其所在方法的非 final 局部变量。如块(3)和块(4)不可以访问 initComponents()方法中的 layout。确需访问,必须给该局部变量加上 final 修饰符。

如果监听器接口只有一个抽象方法,则该接口就是函数式接口,可以使用 Lambda 表达式更简洁地实现。例如,对登录窗口的"取消"按钮可以使用 Lambda 表达式实现监听器接口,设计方法是:右击"取消"按钮,单击快捷菜单中的 Customize Code 菜单项,在 Code Customizer 对话框中//Code adding the component to the parent container -not shown here 一行下输入代码 "jButtonCancel.addActionListener(**e-> this.dispose()**);",单击 OK 按钮(见图 3.5)。可以看到,NetBeans 只是在 initComponents()方法中添加了此行代码,用粗体标记的一个简单 Lambda 表达式代替了内部类,程序可以按题意运行。

3.2.4 代码保护及事件处理代码的复用

匿名内部类只能创建一个对象,某些情况下不能满足需要。例如登录程序的例子,如果也限制密码输入只允许使用字母和数字,那么为口令字段 jPasswordFieldLogin 编写的监听器与用户名输入文本字段基本相同,但由于没有办法再次引用块(3)创建的对象,需要重复编写基本相同的代码块,造成了代码冗余。按照一般面向对象编程的思想,自然想到将这部

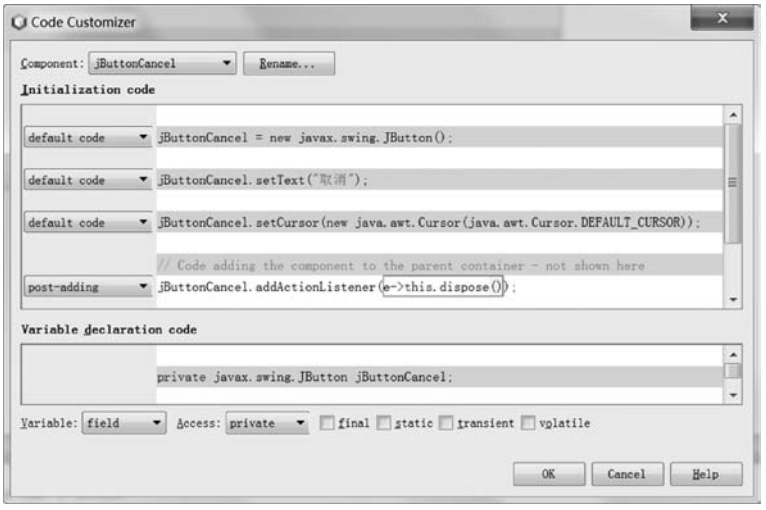

图 3.5　使用 Lambda 表达式为“取消”按钮设计 ActionListener

分代码抽取出来组织成一个命名的内部类，这样就可以创建多个对象来多次使用这个内部类。但是实际试一试却发现行不通。

原因在于，当使用 NetBeans IDE 的 GUI 构建器创建 GUI 窗体及其包含的组件时，IDE 会自动产生保护代码块。保护内容包括：组件变量定义（如 3.2.3 节代码段中的块(8)）、initComponents()方法、所有事件处理程序的头及尾括号“}”。再切换到 Source 视图发现，凡是以灰色背景显示的代码都是不可修改的，这其中就包括各个组件的事件处理器注册代码及作为其实参出现的事件监听器匿名内部类（对象）。因此，不能把事件监听器匿名内部类改写为命名的内部事件监听器类。而以白色背景显示的代码可以修改，其中包括事件监听器匿名内部类体中实现的方法所调用的方法。如 3.2.3 节代码段中块(3)所调用的块(6)的方法体是可以修改的，而这正是实际的事件处理代码。

既然事件处理的实质代码是单独的方法，方法是可以多次调用的，因此通过方法的复用而实现事件处理代码的复用。对本节开头提出的问题，可以在 jPasswordFieldLogin 按键处理方法 private void jPasswordFieldLoginKeyTyped (java.awt.event.KeyEvent evt) {} 的方法体中加入语句“jTextFieldUserNameKeyTyped (evt);”，也就是复用用户名 jTextFieldUserName 的事件处理方法即可。

3.2.5　管理事件监听器

NetBeans IDE 为简化对组件注册和设计事件监听器提供了一系列的功能，可以编写更少的代码完成工作。除了右击事件源注册和设计事件监听器外，还可以使用以下步骤及方法设计和管理事件监听器。

(1) 首先在 Design 视图或 Navigator 中选择事件源组件。如选择用户名 jTextFieldUserName 文本字段。

(2) 在 Properties 窗口中单击 Events 标签切换到“事件”选项卡。

(3) 找到需要处理的事件操作，如 keyTyped，单击该行右侧的“…”按钮。

(4) 在该操作的处理程序对话框中，Add 按钮用于添加事件处理方法，Rename 按钮用

于改名选定的事件处理方法，Remove 按钮用于删除选定的事件处理方法（见图 3.6(a)）。

(5) 若单击 Properties 窗口该行右侧的下三角按钮，在下拉列表中单击已存在的方法（见图 3.6(b)），则可导航到该方法的源代码段，编写和修改事件处理代码。

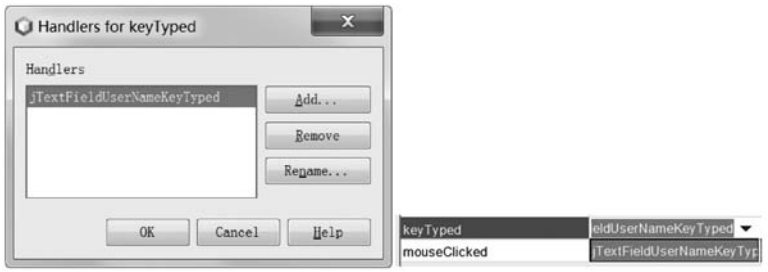

图 3.6　管理事件处理方法

3.2.6　用 NetBeans IDE 连接向导设置事件

可以使用连接向导在一个窗体的两个组件之间设置事件而不用手工编写代码。例如，新建名为 EventTest 的 Java 项目，在其中创建 book.chap3.eventsTest 包及名为 ComponentLink 的窗体，窗体中创建一个标签组件 jLabel1 和一个文本字段组件 jTextField1，可以按照以下步骤设置在 jTextField1 中输入时，把文本字段的内容显示在标签 jLabel1 上。

(1) 单击 GUI 编辑器窗口中工具栏上的 Connection Mode 按钮。

(2) 在 Design 视图的窗体上或在 Navigator 中选择事件源组件，如文本字段 jTextField1。此时，被选组件以红色加亮显示。

(3) 选择事件影响其状态的目标组件，如 jLabel1。该组件也会以红色加亮显示。

(4) 出现 Connection Wizard。在 Select Source Event 页面找到需要处理的事件，扩展该节点，选择需要处理的事件操作。在 Method Name 文本框中可以修改所产生的方法名。例如，找到 key→keyTyped，方法名默认（见图 3.7）。单击 Next 按钮。

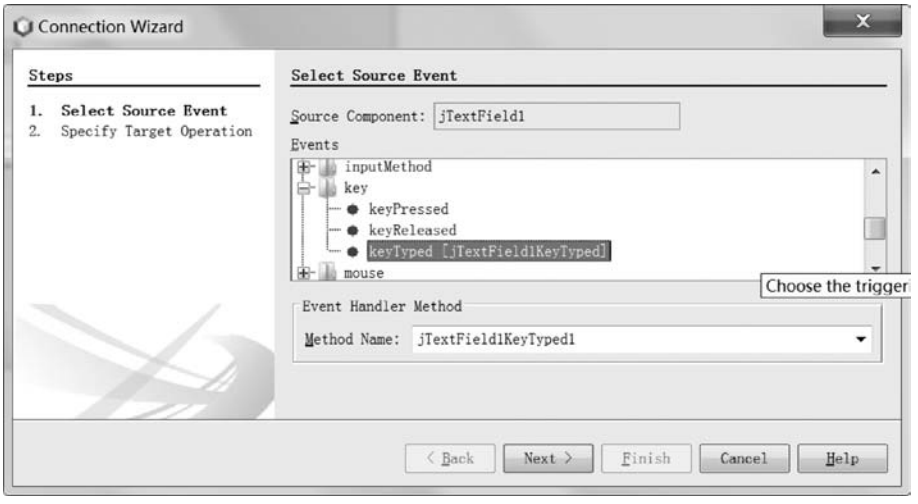

图 3.7　连接向导——选择源事件

（5）在 Specify Target Operation 页面，可以选择设置目标组件的属性、调用目标组件的方法，或编写用户代码。例如，选择对 jLabel1 组件 Set Property，在列表中选择 text 属性。单击 Next 按钮（见图 3.8）。

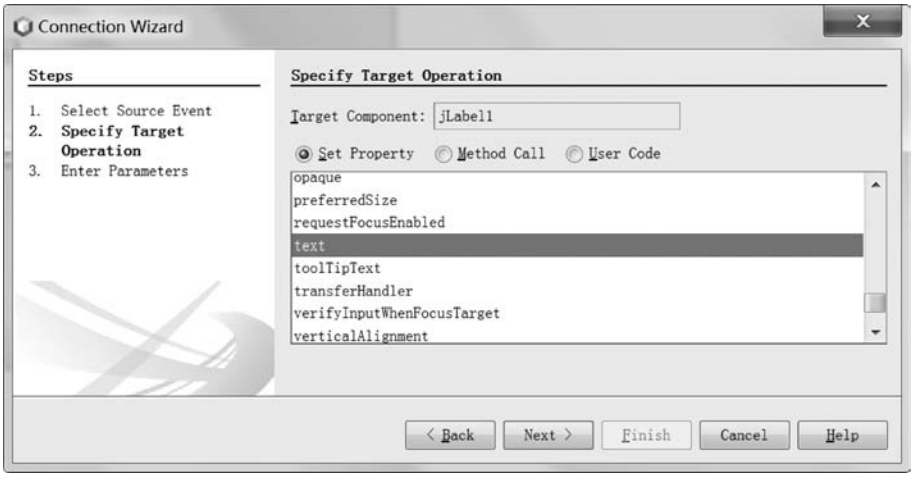

图 3.8　连接向导——指定目标操作

（6）在 Enter Parameters 页面设置参数来源。例如，设置参数来源为 Property，单击属性右侧的“…”按钮，在 Select Property 对话框中选择 Component 为 jTextField1，在 Properties 选项框中选择 text，单击 OK 按钮返回（见图 3.9）。

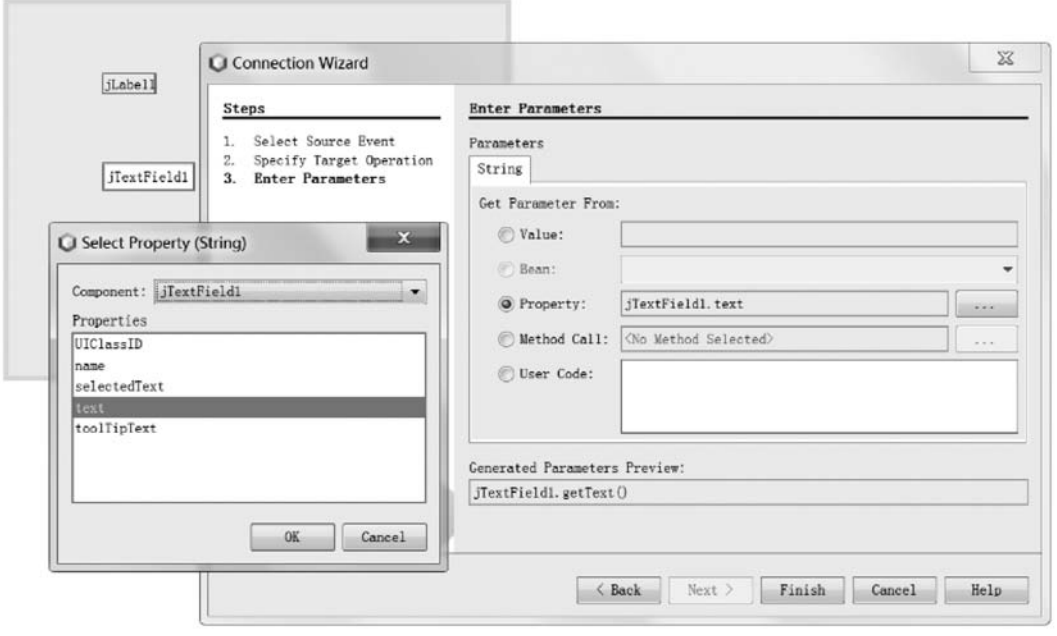

图 3.9　连接向导——输入参数

（7）单击 Finish 按钮完成向导。之后，界面切换到 Source 视图，光标插入点在事件源组件的事件处理方法内。可以进一步编辑该方法。

3.3 常用事件监听器

一般而言，Java GUI 程序的设计，大量创造性工作是编写事件监听器类。GUI 程序中用户与程序交互的主要事件是用鼠标和键盘对窗口及窗口内的组件操作发生的。表 3.1 列出了各类组件事件监听器。其中，Component 是所有 Swing 组件的祖先类。因此，焦点事件、按键事件、鼠标事件等是各类组件共有的。不同的组件也有其特有的事件，如窗口有 Window 事件而其他组件没有，按钮、文本字段和单选按钮有 Action 事件但窗体和标签没有。

表 3.1 各类组件事件监听器

事　件	事件方法	监　听　器	监听器方法	组　件
ActionEvent	getAction， Command， getModifiers	ActionListener	actionPerformed	AbstractButton， JComboBox， JTextField，Timer
Adjustment Event	getAdjustable， getAdjustmentType， getValue	Adjustment Listener	adjustmentValue Changed	JScrollbar
ItemEvent	getItem， getItemSelectable， getStateChange	ItemListener	itemStateChange	AbstractButton， JComboBox
FocusEvent	isTemporary	FocusListener	focusGained， focusLost	Component
KeyEvent	getKeyChar，getKeyCode， getKeyModifiesText， getKeyText，isActionKey	KeyListener	keyPressed， keyReleased， keyTyped	Component
MouseEvent	getClickCount，getX，getY， getPoint，translatePoint	MouseListener	mouseClicked， mousePressed， mouseReleased， mouseEntered， mouseExited	Component
		MouseMotion Listener	mouseDragged， mouseMoved	Component
MouseWheel Event	getWheelRotation， getScrollAmount	MouseWheel Listener	mouseWheelMoved	Component
WindowEvent	getWindow	WindowListener	windowOpened， windowClosing， windowClosed， windowIconified， windowDeiconified， windowActivated， windowDeactivated	Window
	getOpposite Window	WindowFocus Listener	windowGainedFocus， windowLostFocus	Window
	getOldState，getNewState	WindowState Listener	windowStateChanged	Window

以下简单介绍主要事件及其事件处理。

3.3.1 鼠标事件

在窗口系统中，鼠标几乎是必备设备。一般来说，窗口中鼠标操作有鼠标单击、鼠标双击、鼠标光标进入窗口、鼠标光标退出窗口、鼠标移动及鼠标滚轮等，在 Swing 中用 MouseEvent 类表示鼠标单击和鼠标移动事件，用 MouseWheelEvent 类表示鼠标滚轮事件。有三个相应的接口用于监听鼠标事件。

在以 MouseEvent evt 为参数的方法中输入“evt.”之后，弹出 MouseEvent 成员变量和方法列表（见图 3.10）。使用这些方法可以获取鼠标事件发生时的信息，如鼠标当时的坐标、哪个鼠标键被按动、单击鼠标时是否按下 Alt 键及事件源组件是哪个等。如果程序中采用鼠标单击与键盘修饰键组合的方式操作，那么可以使用位掩码测试按下了哪个修饰键。MouseEvent 类中定义了一些常量表示这些掩码，在代码区输入“MouseEvent.”之后，弹出 MouseEvent 常量和静态方法列表（见图 3.11）。

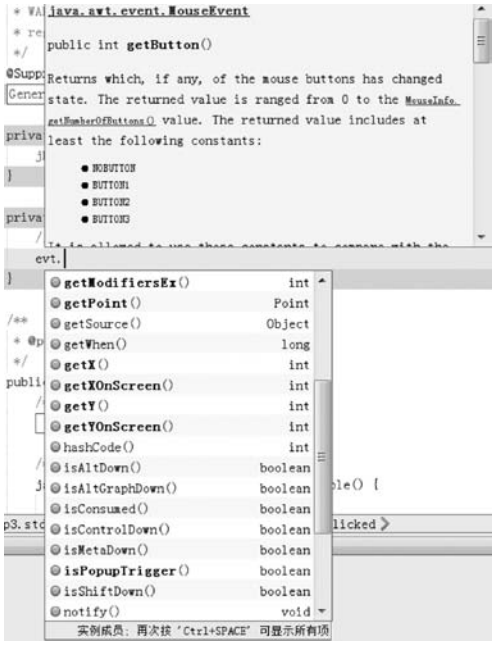

图 3.10　MouseEvent 对象的成员

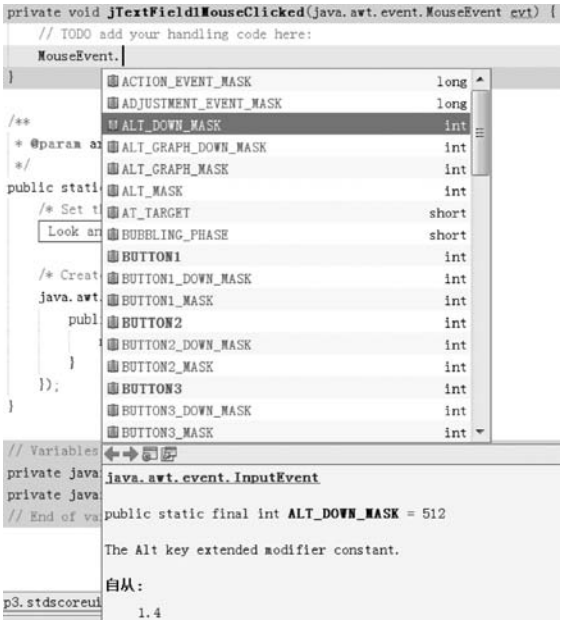

图 3.11　MouseEvent 类的常量及静态方法

使用 MouseEvent 对象的 getModifiersEx()方法可以精准地检测鼠标事件的鼠标按键和键盘修饰符。

1. MouseListener 接口

对 MouseEvent 事件通过实现 MouseListener 接口的实例来响应，鼠标单击（Clicked）、按下（Pressed）、松开（Released）、鼠标光标移入（Entered）及鼠标光标移出（Exited）操作会发生 MouseEvent 事件，该监听器有五个对应的方法（见表 3.1）。

一次单击鼠标按键动作，首先执行 mousePressed()方法，然后执行 mouseReleased()方法，最后会执行 mouseClicked()方法。使用鼠标掩码与鼠标事件的 getModifiersEx()方法

可以检测单击的是哪个鼠标键。

在 3.2.6 节的简单例子中,对文本字段监听鼠标按下操作,事件处理方法如下。

```
private void jTextField1MousePressed(java.awt.event.MouseEvent evt) {
   if((MouseEvent.BUTTON3_DOWN_MASK & evt.getModifiersEx())!= 0) {
       jLabel1.setText("鼠标右键被按下。");
   }
}
```

在 Windows 系统定义的鼠标右键掩码是 MouseEvent.BUTTON3_DOWN_MASK。运行该文件,当在文本字段中按下鼠标右键时,标签显示“鼠标右键被按下。”的信息。

2. MouseMotionListener 接口

当鼠标在窗口上移动时,窗口会收到一连串的鼠标移动事件。设计鼠标单击监听器 MouseListener 和鼠标移动监听器 MouseMotionListener,前者只处理鼠标单击及进出组件的事件,后者处理鼠标移动事件,有利于提高效率。通过实现 MouseMotionListener 接口实现类的实例响应鼠标移动时发生的 MouseEvent 事件。该接口提供以下两个方法。

(1) mouseDragged()方法,鼠标键在组件内按下,同时鼠标移动时执行。

(2) mouseMove()方法,鼠标键没有按下,同时鼠标在组件内移动时执行。

3. MouseWheelListener 接口

目前有一些 GUI 程序使用鼠标滚轮操作。拨动鼠标滚轮发生 MouseWheelEvent 事件,通过实现了 MouseWheelListener 接口的类的实例响应该事件。该接口有一个方法:mouseWheelMoved()。

4. 鼠标事件实例

例 3.3 为了更深入地理解鼠标事件,下面通过具体的实例演示如何响应鼠标事件。

解:操作步骤如下。

(1) 在项目 EventTest 的 book.chap3.eventsTest 包中新建名为 ExMouseEvent 的 JFrame 窗体。

(2) 向窗体中添加组件。一个按钮名为 jButton1,text 为“初始按钮”;一个文本字段名为 jTextField1。选择 jTextField1,拖动下边框的中间调整控柄为 100、右边框的中间调整控柄为 250。

(3) 处理鼠标在 jButton1 上的 MouseEvent 事件。

为按钮添加鼠标事件监听器。右击按钮 jButton1,选择 Events → mouse → mouseClicked 菜单项。为生成的匿名事件监听器的事件处理方法编写事件处理代码,代码体中加入“jTextField1.setText("鼠标单击了 "+evt.getSource().getClass().toString());”语句。

用同样方法分别为其他 4 个事件操作编写事件处理方法代码,方法代码如下。

```
private void jButton1MouseEntered(java.awt.event.MouseEvent evt) {
    jTextField1.setText("鼠标进入了 " + evt.getSource().getClass().toString());
}
private void jButton1MouseExited(java.awt.event.MouseEvent evt) {
    jTextField1.setText("鼠标退出了 " + evt.getSource().getClass().toString());
}
```

```
private void jButton1MousePressed(java.awt.event.MouseEvent evt) {
    jTextField1.setText("按下鼠标键: " +
                        MouseEvent.getModifiersExText(evt.getModifiersEx()));
}
private void jButton1MouseReleased(java.awt.event.MouseEvent evt) {
    jTextField1.setText("释放鼠标键: " +
                    MouseEvent.getMouseModifiersText(evt.getModifiersEx()));
}
```

运行程序看到，当鼠标移动到按钮上时，文本框中显示按钮的类名。当按下鼠标键时，文本框显示所按鼠标键的名字。当松开鼠标键时，显示鼠标键名（运行快看不清时，先注释单击方法中的语句），紧接着，文本框中显示“鼠标单击”＋按钮的类名。当鼠标移出按钮时，文本框中显示按钮的类名。

（4）处理鼠标在窗口中的移动事件。在窗体中创建一个标签 jLabel1。在导航器窗口右击窗体 JFrame，选择 Events→MouseMotion→mouseDragged 菜单项。编写如下事件处理代码。

```
private void formMouseDragged(java.awt.event.MouseEvent evt) {
    jLabel1.setText("鼠标拖动到: (" + evt.getXOnScreen() + "," + evt.getYOnScreen() + ")");
}
```

运行程序看到，当按下鼠标键（左键、中键或右键）在窗口中移动时，标签 jLabel1 显示鼠标在整个屏幕上的当前坐标。

同样地，为鼠标光标移动方法 mouseMoved()编写事件处理代码如下。

```
private void formMouseMoved(java.awt.event.MouseEvent evt) {
    jLabel1.setText("鼠标光标移动到: (" + evt.getX() + "," + evt.getY() + ")");
}
```

运行程序看到，当鼠标光标在窗口中移动时（不按下鼠标键），标签 jLabel1 显示鼠标在窗口中的当前坐标。

（5）当在窗口中滚动鼠标滚轮时，加大或减小按钮 jButton1 的宽度。在导航器窗口右击窗体 JFrame，选择 Events→MouseWheel→mouseWheelMoved 菜单项。编写如下事件处理代码。

```
private void formMouseWheelMoved(java.awt.event.MouseWheelEvent evt) {
    if(evt.getWheelRotation() == 1) {
       jButton1.setSize(jButton1.getWidth() + 20,jButton1.getHeight());
    } else if(evt.getWheelRotation() == -1) {
       jButton1.setSize(jButton1.getWidth() - 20,jButton1.getHeight());
    }
}
```

运行程序看到，向下滚动鼠标滚轮时按钮宽度变大，向上滚动鼠标滚轮时按钮宽度变小。例 3.3 的部分运行界面如图 3.12 所示。

3.3.2 键盘事件

键盘事件是最简单也是最常用的事件。键按下、键松开和按键操作会触发键盘事件

图 3.12　例 3.3 鼠标事件监听器示例运行效果

KeyEvent，监听器 KeyListener 定义了与三种操作对应的三个处理方法。

在以 KeyEvent evt 作为参数的 KeyListener 监听器的方法中，IDE 弹出的提示框列出 evt 的方法和 KeyEvent 类的常量如图 3.13 所示。

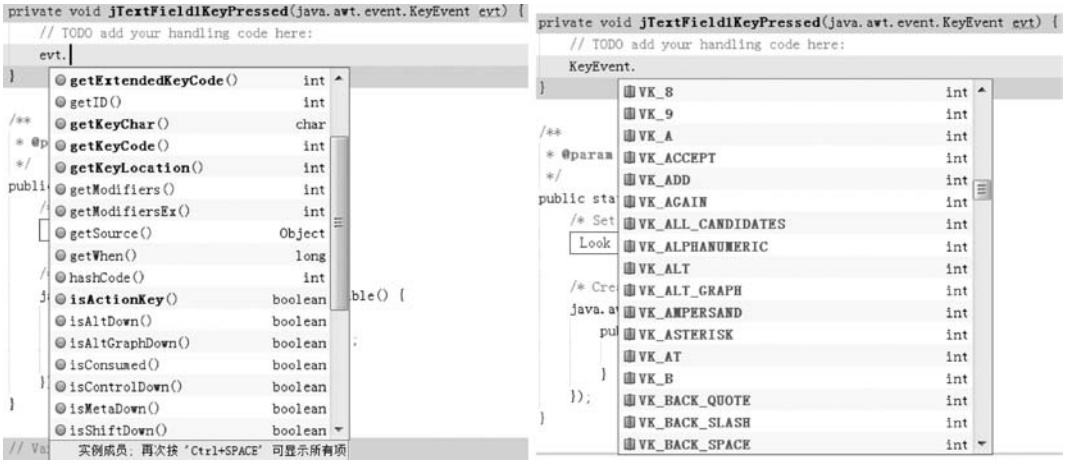

图 3.13　evt 的方法和 KeyEvent 类的常量列表

KeyEvent 事件对象的 getKeyChar()返回按键对应的字符，getKeyCode()返回按键对应的键码。某些键，如箭头键、数字键以及翻页键等存在于键盘的两个部位，getKeyLocation()返回按键所在的区域(如数字键盘区)。例 3.2 中使用了键盘事件检测输入时所按的键是否为字母或数字。

3.3.3　焦点事件

在窗口系统中，当组件获得焦点或失去焦点时触发 FocusEvent 事件。Swing 通过

FocusListener 监听焦点事件，focusGained（FocusEvent evt）方法响应组件获得焦点、focusLost(FocusEvent evt)方法响应组件失去焦点。

例 3.4 新建 JFrame 窗体 FocusEvent，添加 jButton1、jButton2 和 jButton3 按钮及一个标签 jLabel1，为每个按钮注册焦点事件，当某个按钮获得焦点时在 jLabel1 显示获得焦点的信息和失去焦点的按钮信息。

右击 jButton1 按钮，选择 Events→Focus→FocusGained 命令。在生成的事件监听器的 jButton1FocusGained()方法体中输入以下代码。

```
String str = jLabel1.getText() ;
str = str.endsWith("。")?str.substring(str.length() - 17):"";
jLabel1.setText(str + "按钮 " + ((JButton)evt.getSource()).getText() + " 获得焦点。");
```

右击 jButton1 按钮，选择 Events→Focus→FocusLost 命令，在方法 jButton1FocusLost()中输入以下代码。

```
String str = jLabel1.getText() ;
str = str.endsWith("。")?str.substring(str.length() - 17):"";
jLabel1.setText(str + "按钮 " + ((JButton)evt.getSource()).getText() + " 失去焦点。");
```

用同样方法为 jButton2 和 jButton3 按钮注册并编写事件处理方法，代码与上面的一样。

3.3.4 组件专用事件

许多组件有其专有的事件和事件监听器。例如，窗口 JFrame 有 Window、WindowFocus 和 WindowState 操作的 WindowEvent 事件及其监听器 WindowListener、WindowFocusListener 和 WindowStateListener；文本字段有 CaretEvent 事件及其监听器 CaretListener；按钮 JButton 和单选按钮 JRadioButton 有 ItemEvent 事件及其监听器 ItemListener。这些根据具体组件而不同的事件及其监听器，右击组件之后就可以在 Events 菜单中看到，通过选择这些菜单项就可以直接对组件专有事件进行处理。

1. 窗口事件 WindowEvent

在窗口打开、关闭、图标化（最小化）、恢复窗口、转入活动状态（前台）、转入非活动状态（后台）、获得焦点、失去焦点等状态改变时，都会触发窗口事件 WindowEvent。对应地有三个监听器分别处理有关操作。

1）WindowEvent 对象常用方法

Window getWindow()：返回发生此事件的 Window 对象。

Window getOppositeWindow()：若发生了焦点转移，返回另一个参与此事件的 Window 对象，或者 null。

int getOldState()：返回窗口变化前的状态，可取值为 NORMAL、ICONIFIED、MAXIMIZED_BOTH。

int getNewState()：返回窗口变化后的状态。

2）WindowListener 接口

该接口处理 windowOpened、windowClosing、windowClosed、windowIconified、

windowDeiconified、windowActivated、windowDeactivated 操作触发的 WindowEvent 事件。

3）WindowFocusListener 接口

该接口处理 windowGainedFocus 和 windowLostFocus 操作触发的 WindowEvent 事件。

4）WindowStateListener 接口

该接口处理 WindowStateChanged 操作触发的 WindowEvent 事件。

2. ItemEvent 事件

在单击按钮 JButton、单选按钮 JRadioButton、复选框 JCheckBox 和菜单项 JMenuItem 等组件，或者在列表中选择条目时，它们的状态发生改变，触发 ItemEvent 事件。该事件对象的主要方法如下。

(1) Object getItem()：取得被选取的元素。注意，返回值是 Object，一般需要进行强制类型转换。

(2) ItemSelectable getItemSelectable()：ItemSelectable 是一个接口，代表那些包含 n 个可供选择的子元素的对象，它的方法 Object[] getSelectedObjects() 返回已选择的那些对象。此方法的作用主要在于，如果一个列表框是允许多选的，应该用此方法得到列表对象，再取得被选中的多个元素。

(3) int getStateChange()：取得选择的状态，是 SELECTED 或 DESELECTED。

该事件的监听器接口是 ItemListener，事件处理方法是 itemStateChanged()。

3. CaretEvent 事件

当文本组件中插入点的位置改变时触发 CaretEvent 事件。该事件有以下两个主要方法。

(1) public abstract int getDot()：返回插入符的位置。

(2) public abstract int getMark()：取得一个逻辑选择另一端的位置。如果没有选择，则与 getDot()方法相同。

CaretEvent 事件监听器接口是 CaretListener，事件处理方法是 caretUpdate (CaretEvent e)，当插入符的位置改变时调用。

以上介绍了大部分常用事件及其监听器，其余的都可以在 NetBeans IDE 设计窗口快捷菜单的 Events 级联菜单中得到帮助。

3.4 使用 SwingWorker

Java Swing GUI 程序启动时，JVM 会启动多个线程。但是，Swing 并不是线程安全的，如果处理不当，Swing GUI 程序可能会反应迟钝，造成用户反感。从 Java SE 6 开始引进的 SwingWorker 能帮助用户编写多线程 Swing 程序，改善 Swing 程序的结构，提高界面响应的灵活性。

3.4.1 正确使用事件派发线程

在主线程 main()方法中执行

```
java.awt.EventQueue.invokeLater(new Runnable() {
```

```
        public void run() {
          new NumberAdditionUI().setVisible(true);
        }
    });
```


语句，invokeLater()方法将 Runnable 任务提交给事件派发线程(EDT)。此点是创建 UI 的点，也是程序开始将控制权交给 UI 的点。之后，主线程结束，程序在 EDT 中运行。由于 EDT 线程负责 GUI 组件的绘制和更新，所有事件处理都是在 EDT 上进行，程序同 UI 组件和其基本数据模型的交互只允许在 EDT 上进行。可见，EDT 线程的事件队列很繁忙，几乎每一次 GUI 交互和事件处理都是通过 EDT 完成的。事件队列上的任务必须非常快地完成，否则就会阻塞其他任务的执行，使队列里阻塞了很多等待执行的事件，造成界面响应不灵活，让用户感觉到界面响应速度慢而失去兴趣。理想情况下，任何需时超过 30～100ms 的任务不应放在 EDT 上执行，否则用户就会觉察到输入和界面响应之间的延迟。因此，Swing 编程时应该注意以下几点。

(1) 从非 EDT 线程访问 UI 组件及其事件监听器会导致界面更新和绘制错误。

(2) 在 EDT 上执行耗时任务会使程序失去响应，使 GUI 事件阻塞在队列中得不到处理。

(3) 应该使用独立的任务线程来执行耗时计算或输入输出密集型任务，比如同数据库通信、访问网站资源、读写大数据量的文件等。

总之，任何干扰或延迟 UI 事件的处理只应该出现在独立任务线程中；在主线程或任务线程同 Swing 组件或其默认数据模型进行的交互都是非线程安全性操作。

3.4.2 SwingWorker 类

SwingWorker 类帮助管理任务线程与 Swing EDT 之间的交互。尽管 SwingWorker 不能解决并发线程中遇到的所有问题，但的确有助于分离 Swing EDT 和任务线程，使它们各负其责：对于 EDT 来说，就是绘制和更新界面，并响应用户输入；对于任务线程来说，就是执行和界面无直接关系的耗时任务和 I/O 密集型操作。

1. SwingWorker 类的定义

SwingWorker 类的定义如下。

```
public abstract class SwingWorker<T, V> extends Object implements RunnableFuture
```

SwingWorker 是抽象类，因此必须继承它才能执行所需的特定任务。该类对象封装类型为 T 的结果以及类型为 V 的进度数据。

接口 RunnableFuture 是 Runnable 和 Future 两个接口的简单封装。由于 SwingWorker 实现了 Runnable 接口，因此 SwingWorker 有一个 run()方法。Runnable 对象一般作为线程的一部分执行，当 Thread 对象启动时，它激活 Runnable 对象的 run()方法。又由于 SwingWorker 实现了 Future 接口，因此 SwingWorker 使用 get()方法获取类型为 T 的结果值，并提供同线程交互的方法。SwingWorker 实现了这两个父接口的大部分方法。

(1) boolean cancel(boolean mayInterruptIfRunning)：取消正在进行的工作。

(2) T get()：获取类型为 T 的结果值。该方法将一直处于阻塞状态，直到结果可用。

(3) T get(long timeout,TimeUnit unit)：获取类型为 T 的结果值。将会一直阻塞直

到结果可用或超时。

(4) boolean isCancelled()：判断任务线程是否被取消。

(5) boolean isDone()：判断任务线程是否完成。

(6) 实际编程仅需要实现 SwingWorker 的抽象方法：abstract T doInBackground()。

该方法作为任务线程的一部分执行，负责完成线程的基本任务，并以返回值(类型为 T)作为线程的执行结果。继承 SwingWorker 的类必须覆盖该方法并确保包含或代理任务线程的基本任务。不要直接调用该方法，应使用任务对象的 execute()方法来调度执行。

(7) 在 doInBackground()方法完成后，SwingWorker 在 EDT 上激活 done()方法。

```
protected void done()
```

如果需要在任务完成后使用线程结果更新 GUI 组件或者做些清理工作，可覆盖 done()方法来完成。

(8) void publish(V... data)：传递中间进度数据到 EDT。从 doInBackground()方法中调用该方法。

(9) void process(List < V > data)：覆盖该方法处理任务线程的中间结果数据。

(10) void execute()：为 SwingWorker 线程的执行预定该 SwingWorker 对象。

任务线程有几种状态，使用 SwingWorker. StateValue 枚举值表示：PENDING、STARTED 和 DONE。任务线程一创建就处于 PENDING 状态，当 doInBackground()方法开始时，任务线程就进入 STARTED 状态，当 doInBackground()方法完成后，任务线程就处于 DONE 状态。随着线程进入各个阶段，SwingWorker 超类自动设置这些状态值。可以注册监听器，当这些属性发生变化时接收通知。

任务对象有一个进度属性，可以随着任务的进展，将这个属性从 0 更新到 100 标识任务的进度，当该属性发生变化时，任务通知处理器进行处理。

2. SwingWorker 的工作模型

当 Swing GUI 程序要执行耗时任务时，在 EDT 中创建一个 SwingWorker 对象。在该对象的 doInBackground()方法中执行耗时操作，该方法在 EDT 中被 execute()方法调用，在 SwingWorker 线程中执行。在 doInBackground()方法中不时地调用 publish()来发布中间进度数据。publish()方法使得 process()方法在 EDT 中执行来处理进度数据。当工作完成时，在 EDT 中调用 done()方法以便完成 UI 的更新。在 done()方法中可以使用 get()方法获取 doInBackground()的执行结果(见图 3.14)。

3.4.3 SwingWorker 类的使用

编写 Swing 应用程序常见的错误是误用 Swing 事件派发线程(EDT)。要么从非 UI 线程访问 UI 组件，要么不考虑事件执行顺序，要么不使用独立任务线程而在 EDT 线程上执行耗时任务，结果使编写的应用程序变得响应迟钝、速度很慢。耗时计算和输入/输出(I/O)密集型任务不应放在 Swing EDT 上运行，而应该使用 SwingWorker 启动一个任务线程来异步执行，并马上返回 EDT 线程，允许 EDT 立即继续处理后续的 UI 事件。

在前面开发的例 3.1 学生成绩管理系统的用户登录程序中，需要从磁盘读取用户注册信息文件 users. txt。在这个程序中，用户单击登录窗口的“登录”按钮会执行 new UsersSet().

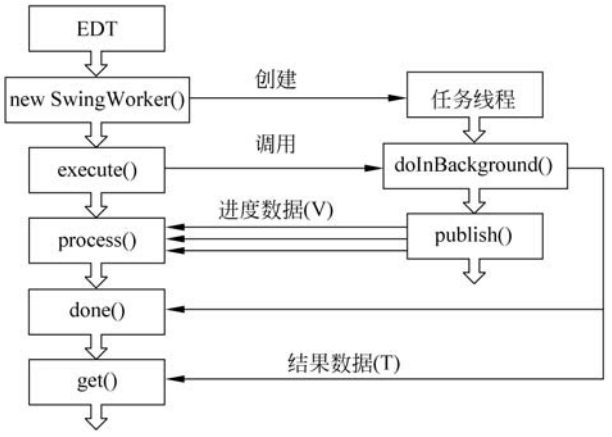

图 3.14　SwingWorker 工作模型

isValid(user)，在执行 UsersSet()构造方法时会读取这个文件。如果 users.txt 文件比较大，例如记录了几万个用户注册信息，那么读取文件就要花费较长时间。而读写文件的操作都直接在按钮的事件处理程序中执行，会明显影响 GUI 的反应速度，会使程序慢而滞涩。因此，需要对该例子程序进行改进，方法就是使用 SwingWorker 类对象单独创建一个操作文件 users.txt 的任务线程，将耗时的 I/O 操作从 EDT 中分离出去。

例 3.5　修改例 3.1 学生成绩管理系统的用户登录程序，当用户未单击“登录”按钮时“取消”按钮是失效的。当用户单击了“登录”按钮时该按钮变为无效状态，而“取消”按钮变为有效状态，直到登录操作完成，两个按钮恢复初始状态。此外，与窗体的底边对齐创建一个标签，设置为可水平调整大小。使用 SwingWorker 类单独创建一个操作文件 users.txt 的任务线程，将耗时的 I/O 操作从 EDT 中分离出去，从文件中每读一条用户信息，就在进度标签显示 10 个“I”字符。为了演示耗时操作，每从文件中读一条用户信息，任务线程就休眠 2 秒。

解：按照题意，设计步骤如下。

（1）右击 Projects 窗口的 StdScoreManager 节点，在快捷菜单中选择 Copy 命令，打开 Copy Project 对话框（见图 3.15），在 Project Name 文本框中，将新项目名称 StdScoreManager_1 修改为 StdScoreManager0.1，其他选项为默认，单击 Copy 按钮。以下操作在该项目中进行。

（2）打开项目 StdScoreManager0.1 的 UserLogin 窗体，设置 jButtonCancel 按钮的 enabled 属性为未选取状态。在窗体中创建一个标签 JLabel，放置在左边框到容器左边框的首选位置，标签底边与窗体底边对齐，变量名称改为 jLabelPrb，text 属性设置为“”，设置 Change horizantal resizability 属性为选取状态。

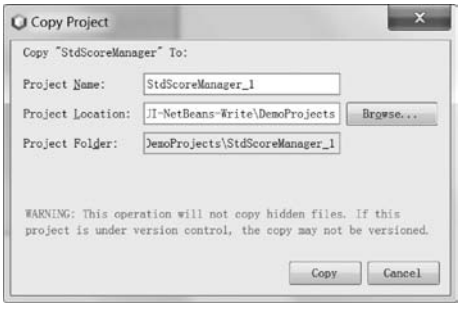

图 3.15　复制 NetBeans IDE 项目

（3）使用 SwingWorker 处理任务线程。首先为登录窗体 UserLogin 设计一个内部类封装进度数据，数据项包括当前处理的行号和使用该行构

造的 User 对象。该类见程序清单 3.1。

程序清单 3.1　进度数据封装类

```
public class ProgressData {
    private User user ;
    private int number ;
    public ProgressData(User user, int number) {
        this.user = user;
        this.number = number;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    public int getNumber() {
        return number;
    }
    public void setNumber(int number) {
        this.number = number;
    }
}
```

(4) 在登录窗体 UserLogin 中设计 SwingWorker 类的子类作为内部类,完成耗时的读取用户注册文件 users.txt 的操作。该类最后结果是一个封装了所有用户信息的 HashSet 对象(集合),中间数据是 ProgressData 类的对象。在该类的 doInBackground()方法中读取 users.txt 文件,每次读一行封装为 User 对象,并调用 publish()方法发布该行行号和 User 对象。每读取一行之后休眠 2000 毫秒以模拟对长文件的操作。

在 progress()方法中更新进度标签 jLabelPrb,并检测本次 publish()方法发送来的进度数据中的 User 对象,若与登录用户 user 相同则完成登录过程,界面转到欢迎界面。若该 SwingWorker 线程被中止,则改变按钮的 enabled 状态。

在 done()方法中清除进度标签 jLabelPrb 文字及“用户名”和“密码”输入文本框,改变按钮的 enabled 状态。

初学者对于这种涉及多线程的程序最好先规划好各线程的任务,再开始编码。程序清单 3.2 是 UserLogin 类中的内部类 UserReader,它是一个 SwingWorker 子类。

程序清单 3.2　UserLogin 窗体的 SwingWorker 子类 UserReader

```
public class UserReader extends SwingWorker < HashSet < User >, ProgressData > {
    User user = null;
    JFrame jfr = null;
    HashSet < User > userSet = null;

    public UserReader(JFrame jfr, User user) {
        this.jfr = jfr;
        this.user = user;
    }
```

```
        public HashSet<User> getUserSet() {
            return userSet;
        }
        @Override
        protected void done() {
            jButtonCancel.setEnabled(false);
            jButtonOK.setEnabled(true);
            jTextFieldUserName.setText("");
            jPasswordFieldLogin.setText("");
            jLabelPrb.setText("");
        }
        @Override
        protected void process(java.util.List<ProgressData> chunks) {
            if (isCancelled()) {
                jButtonCancel.setEnabled(false);
                jButtonOK.setEnabled(true);
                return;
            }
            int numLine = chunks.get(chunks.size() - 1).getNumber();
            String prb = jLabelPrb.getText() + numLine + "IIIIIIIIII";
            jLabelPrb.setText(prb);
            for (ProgressData data : chunks) {
                if (data.getUser().equals(user)) {
                    this.cancel(true);
                    new ScoreMana(user).setVisible(true);
                    jfr.dispose();
                    break;
                }
            }
        }
        @Override
        protected HashSet<User> doInBackground() throws Exception {
            int lineNumber = 0;
            userSet = new HashSet<User>();
            Scanner sc = new Scanner(new File("..\\users.txt"));
            String str = null;
            String[] s = null;
            User user = null;
            ProgressData pd = null;
            while (sc.hasNext()) {
                str = sc.nextLine();
                lineNumber++;
                s = str.split(":");
                user = new User(s[0],s[1],Integer.parseInt(s[2]));
                userSet.add(user);
                pd = new ProgressData(user, lineNumber);
                publish(pd);
                Thread.sleep(2000);
            }
            return userSet;
        }
    }
```

(5) 修改用户登录窗体“登录”按钮 jButtonOK 的事件处理方法，将读取用户信息文件，判断是否合法用户，以及更新进度和界面等工作交给工作器 UserReader 的对象调度任务线程和 EDT 线程分工配合执行。下面给出该方法代码。

程序清单 3.3　UserLogin 窗体的事件处理方法

```
private void jButtonOKActionPerformed(java.awt.event.ActionEvent evt) {
    jButtonOK.setEnabled(false);
    jButtonCancel.setEnabled(true);
    String name = jTextFieldUserName.getText().trim();
    String password = new String(jPasswordFieldLogin.getPassword()).trim();
    int job = jRadioButtonStd.isSelected()?0:(jRadioButtonTch.isSelected()?1:2) ;
    User user = new User (name, password, job);
    UserReader ur = new UserReader(this, user);
    ur.execute();
}
```

(6) 在 Source 视图下右击程序清单 3.3 的语句“UserReader ur = new UserReader (this, user);”中的变量名 ur，选择 Refactor→Introduce→Field 菜单项，在出现的蓝色背景语句行单击，在 Introduce Field 对话框中单击 OK 按钮，会将局部变量 ur 抽取为域变量。修改“取消”按钮的定制代码 post-eding 为“jButtonCancel.addActionListener(e→{if(ur!=null)ur.cancel(true);});”，使用户单击“取消”按钮时中止当前输入的用户名和密码的检测登录，以便开始输入新的用户名和密码进行登录。

完成以上修改后运行程序，运行中间单击窗口中的“取消”按钮，发现程序反应灵活。为了对照，删除 doInBackground()方法中的语句“Thread.sleep(2000);”，使用一个具有 5000 行的 users.txt 文件测试，体会 SwingWorker 对程序的改进。

习　　题

1. 解释下列名词：

事件；事件源；事件处理；事件监听器；Java 的委托事件模型

2. 简述 Java GUI 程序对单击按钮的事件处理机制。

3. 事件适配器与事件监听器有什么关系？它们是否一一对应？

4. 什么是静态内部类？什么是局部内部类？

5. 采用匿名内部类实现事件监听器，对它所在方法中的局部变量访问时有什么要求？

6. 如何检测用户单击的是哪个鼠标键？

7. 什么是事件派发线程？对它的使用需要注意什么？

8. 图示说明 SwingWorker 的工作模型，并举例说明如何使用这个模型。

9. 在窗体上创建一个文本字段和滑块 JSlider 组件，使用 NetBeans 的连接向导设计事件处理，使用户通过调整滑块更改文本字段中的数值。

10. 为前面开发的用户登录窗体 UserLogin 添加“修改密码”按钮，设计修改密码窗体，并应用 SwingWorker 实现其功能。