# 模块3

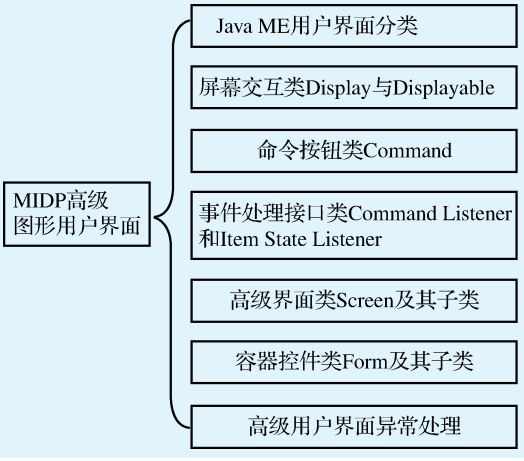
# MIDP 高级图形用户界面

用户界面在程序的开发中非常重要，因为它是程序与用户之间交互的桥梁。MIDP 分别提供了高级用户界面接口和低级用户界面接口。

高级用户界面接口针对移动设备的特点设计，封装性强，具备较好的可移植性，使用简单，外观趋于标准化，但是，留给开发者控制的余地非常有限。例如，开发者不能随意定制组件的外观（如形状、颜色、字体）。高级用户界面主要由 Screen 类及其子类 Alert、List、TextBox 和 Form 类（包括其子类）实现。

**【岗位技能分析图】**

本模块中要掌握的主要岗位技能分析如下图所示。



# 3.1 用户界面组成

用户界面是应用程序和用户之间交互的接口。在 Java ME 中，由于考虑到移动设备的性能比较有限，没有采用 Java SE 中的 AWT/Swing 作为 MIDP 的图形用户界面库，而是重新设计了一个适合移动设备的小巧的 javax. microedtion. lcdui 图形用户界面库，也称为 lcdui 包。lcdui 包中提供了高级用户界面接口和低级用户界面接口，实现了与用户的交互。本模块介绍的各种类都来源于 lcdui 包。

用户界面主要包括以下组成元素。

(1)提供用户与应用程序交互的屏幕交互类，如 Command。

(2)对交互进行响应的事件监听处理接口，如 CommandListener。

(3)负责对用户界面中的组成元素进行管理的界面类，如 Screen 类及其子类。

(4)负责容纳用户界面组成元素的容器控件类，如 Form 及其子类。

## 3.1.1 用户界面类

用户界面类库 lcdui 包，将用户界面类分为两种类型：高级用户界面类 Screen 和低级用户界面类 Canvas。Screen 类及其子类提供了高级用户界面的实现，Canvas 类及其子类提供了低级用户界面的实现。

### 1. 高级用户界面类

高级用户界面类的特点如下。

(1)封装了一些基本的界面控件，如 Alert、List、TextBox 和 Form 等。

(2)主要用于业务处理的应用程序开发。

(3)完成了较高层次的抽象，移植性好。

(4)无须关心各种界面控件的颜色、字体和外观，不同的硬件设备上可能具有不同的设置。

(5)滚动、翻页等交互都由界面控件完成，设计者无须关心。

### 2. 低级用户界面类

低级用户界面类的特点如下。

(1)允许设计者在屏幕上较精确地绘制图形。

(2)主要用于游戏界面的设计。

(3)抽象较少，不能保证程序在不同的硬件设备上运行，也不能保证不同硬件设备的运行结果相同，可移植性差。

(4)可以进行细致的布局、绘制每像素点、接收较低层次的事件、直接获得用户的按键消息。

(5)可以设计复杂的用户交互。

注意：高级用户界面类和低级用户界面类可以在同一个 MIDlet 中使用，但在同一个屏幕对象中，只能使用其中之一。

### 3.1.2 lcdui 包

lcdui 包定义了所有的用户界面类，高级用户界面由 Screen 类及其子类实现，低级用户界面由 Canvas 类及其子类实现。下面介绍 lcdui 包中类的结构及其中的高级用户界面类的结构。

#### 1. lcdui 包的结构

Lcdui 包的结构如图 3-1 所示。

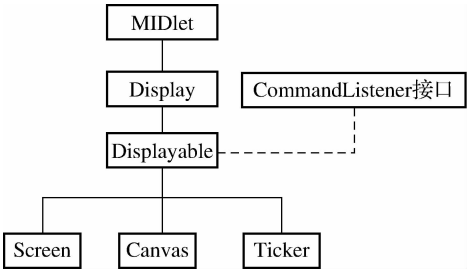



图 3-1 lcdui 包的结构

在 MIDP 程序中，每一个应用都有一个屏幕显示类 Display，用于进行用户界面的管理。当需要将某个对象显示在屏幕上时，使用该对象的 setCurrent 方法将此对象写入当前屏幕的 Displayable 类对象。

#### 2. 高级用户界面类的结构

高级用户界面类的结构如图 3-2 所示。

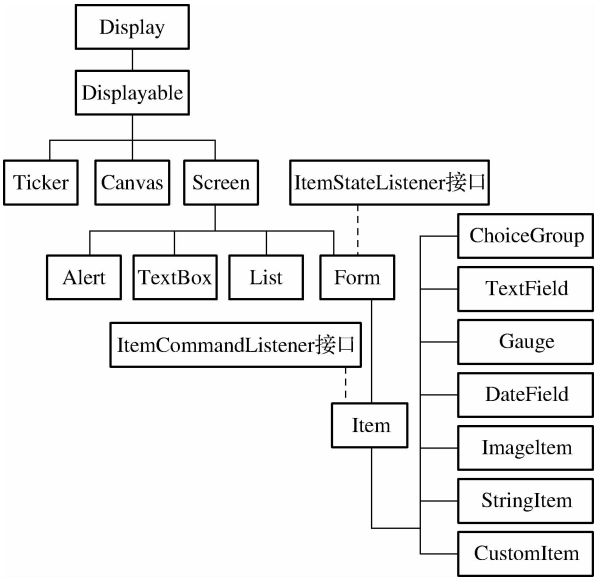



图 3-2 高级用户界面类的结构

Screen 类派生了 4 种类：Alert、TextBox、List 和 Form。前 3 种类不能继承派生，即不

能再向类中添加其他组件。

下面是对各种类的说明。

(1)Alert:表示一个提示信息的文本框,用户可以预定义,但是不能编辑。

(2)TextBox:表示编辑文本的控件,允许用户输入字符串并及时回显。

(3)List:表示列表控件,提供多项选择。

(4)Form:这是一个控件容器,它可以容纳其他控件。这里的控件主要指 Item 类的控件,常用的 Item 控件如下。

- ChoiceGroup:显示选择功能的控件。
- TextField:用户编辑文本的控件。
- Gauge:显示过程进度的控件。
- DateField:用户编辑时间的控件。
- ImageItem:显示图像的控件。
- StringItem:显示字符串的控件。
- CustomItem:用户自定义的 Item 控件。

Form 类和 Item 类可以分别实现 ItemStateListener 和 ItemCommandListener 接口,用于监听状态变化和命令。

## 3.2 屏幕交互类

MIDP 应用程序使用 Display 和 Displayable 类进行用户界面的管理,负责与应用程序进行交互。

### 3.2.1 Display 类

Java SE 应用程序允许用户在显示设备的屏幕上放置多个窗口,但在 Java ME 中,只允许在显示设备的屏幕显示一个小窗口,如果显示设备上有多个 MIDlet 同时运行,那么只有其中的一个可以访问屏幕。Display(显示)类就表示一个逻辑屏幕,代表了一种屏幕管理器。

Display 类的主要方法如表 3-1 所示。

表 3-1　Display 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 描　述 |
| --- | --- | --- | --- |
| callSerially | void | Runnable | 使事件串行化,当 Runnable 对象运行完毕后立即运行该方法 |
| flashBacklight | boolean | int | 设置背景灯闪烁时间,参数表示闪烁时长,如果不支持闪烁,则返回值为 false |
| GetBestImageHeight | int | int | 获得给定图像的最佳高度,参数为整型值 LIST_ELEMENT、CHOICE _ GROUP _ ELEMENT 和 ALERT三者之一 |

续表

| 方法名称 | 返回值类型 | 参数类型 | 描　　述 |
|---|---|---|---|
| GetBestImageWidth | int | int | 获得给定图像的最佳宽度，参数为整型值 LIST_ELEMENT、CHOICE _ GROUP _ ELEMENT 和 ALERT三者之一 |
| GetBorderStyle | int | boolean | 获得笔画的类型，参数为 true 时，返回描绘高亮度边线时的画笔类型；参数为 false 时，返回边线的画笔风格（包括实线形和虚线形） |
| getColor | int | int | 获得系统颜色的 RGB 值，参数为前景色 COLOR_FOREGROUND、背景高亮色 COLOR_HIGHLIGHT _FOREGROUND、前景高亮色 COLOR _ HIGHLIGHT_FOREGROUND、边框色 COLOR_BORDER 和边框高亮色 COLOR_HIGHLIGHT_BORDER |
| getCurrent | Displayable | void | 获得当前 MIDlet 的 Displayable 对象 |
| getDisplay | Static Display | MIDlet | 获得指定 MIDlet 的 Display 对象 |
| isColor | boolean | void | 判断当前设备是否支持彩色显示，true 表示支持，false 表示不支持 |
| numAlphaLevels | int | void | 获取硬件设备支持透明显示的层数 |
| numColors | int | void | 获取当前设备支持的颜色数或者灰度级数 |
| setCurrent | void | Alert，Displayable | 将 Alert 对象设置为当前显示对象，当 Alert 超时后，显示第二个参数指定的 Displayable 对象 |
| setCurrent | int | Displayable | 直接将参数中的 Displayable 对象设置为当前屏幕显示内容 |
| setCurrentItem | int | Item | 直接将参数中的 Item 对象设置为当前屏幕显示内容 |
| vibrate | boolean | int | 设置硬件设备为振动状态，参数指定了振动的时长，如果设备不支持振动，返回 false，否则返回 true |

在 MIDlet 的生命周期中，一般在类的构造方法中调用静态方法 getDisplay 来获得当前 MIDlet 的屏幕类对象。Display 类的语法格式如下。

```
public static Display(MIDlet midlet);
```

参数表示获得哪个 MIDlet 的屏幕类对象，返回值是一个 Display 类的对象。在任何时候，都可以调用该方法，因为每个 MIDlet 都有一个自己的 Display 实例，在多数情况下，将获得的 Display 实例放在成员变量中，从而避免频繁调用 getDisplay 方法。

但 Display 实例中的内容并不会马上显示在屏幕上，而是当调用了 Display 实例的 setCurrent 方法后，才将实例中的内容显示在屏幕上成为当前显示对象，这里的内容是指 Displayable 对象。setCurrent 方法的语法格式如下。

```
public void setCurrent(Displayable displayable);
```

其中，displayable 表示要显示的内容，因为小型设备中一次只能显示一个屏幕，因此执行一次 setCurrent 方法，意味着擦除前一屏 Display 中的内容。

【例 3-1】　使用 Display 对象判断当前设备的屏幕是否支持彩色显示，如果支持就输出它的颜色数量。

在构造方法中，通过 getDisplay 方法获得当前屏幕类对象，存放在成员变量 display 中，使用 Display 类的 isColor 方法判断当前设备是否支持彩色显示，如果支持，就构造字符串“支持彩色显示！ 共支持的颜色数是”；如果不支持，则构造字符串“不支持彩色显示！”，然后将字符串作为构造 TextBox 控件的一部分。在 startApp 方法中调用 setCurrent 方法，将 TextBox 控件显示在屏幕上。

具体代码如下。

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
public class ColorSupported extends MIDlet implements CommandListener{
     private Display display;
     private TextBox t;
     private Command exitCommand;
     public ColorSupported(){
          super();
          display=Display.getDisplay(this);
          exitCommand=new Command ("退出",Command.SCREEN,1);
          String message=null;
          if (display.isColor()){
                message="支持彩色显示！ 共支持的颜色数是"+display.numColors();
          }
          else{
                message="不支持彩色显示！";
          }
          t=new TextBox("支持的颜色",message,17,0);
          t.addCommand(exitCommand);
          t.setCommandListener(this);
     }
  protected void startApp()throws MIDletStateChangeException{
                display.setCurrent(t);
```

```
        }
        protected void pauseApp(){
        }
        protected void destroyApp(boolean arg0){
        }
        public void commandAction(Command c,Displayable d){
            if(c==exitCommand){
                destroyApp(true);
                notifyDestroyed();
            }
        }
}
```

运行结果如图 3-3 所示，显示的是不支持彩色显示的模拟器的运行结果。


图 3-3　不支持彩色显示模拟器的运行结果

### 3.2.2　Displayable 类

Displayable 是一个虚类，不能直接产生它的实例。在显示设备的屏幕上显示的用户界面为 Displayable 子类的对象。在同一时刻应用程序最多只能有一个可显示的 Displayable 对象。Displayable 类包含两个子类：Screen 和 Canvas。

Displayable 类的主要方法如表 3-2 所示。

表 3-2　Displayable 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 描　　述 |
|---|---|---|---|
| addCommand | void | Command | 向一个 Displayable 对象中添加一个 Command 对象 |
| removeCommand | void | Command | 从 Displayable 对象中移除指定的 Command 对象 |

续表

| 方法名称 | 返回值类型 | 参数类型 | 描　　述 |
|---|---|---|---|
| getHeight | int | void | 获得 Displayable 对象的高度 |
| getWidth | int | void | 获得 Displayable 对象的宽度 |
| getTicker | Ticker | void | 获得 Displayable 对象的滚动条 |
| getTicker | void | Ticker | 设置 Displayable 对象的滚动条为参数指定的 Ticker 对象 |
| getTitle | String | void | 获得 Displayable 对象的标题，返回值为标题的字符串 |
| setTitle | void | String | 设置 Displayable 对象的标题为参数指定的字符串 |
| isShown | boolean | void | 判断当前的 Displayable 对象是否真正可见 |
| setCommandListener | void | CommandListener | 判断当前的 Displayable 对象的命令监听器是否为参数指定的 CommandListener 对象 |
| sizeChanged | Protected void | int，int | 当 Displayable 对象的大小发生变化时自动调用的方法，其中第一个参数表示变化后的宽度，第二个参数表示变化后的高度 |

由表 3-2 可以看出，Displayable 类有以下特性。

- 允许添加若干 Command 对象，用于响应用户在界面上的按键。
- 可以绑定一个命令触发监听器，一旦用户根据界面提示，按下数字键盘上的某一个键后，命令触发监听器捕捉到该事件并作出相应处理，处理程序为 CommandAction()。
- 允许设置标题，显示在屏幕上的标准标题位置。
- 允许设置滚动条，用于在屏幕上滚动显示文本。

【例 3-2】　Displayable 对象的使用。要求设置 Displayable 对象的标题、滚动条以及高度和宽度的信息。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayableMIDlet extends MIDlet{
    private Display display;
    private TextBox textBox;
    private Ticker ticker;
    public DisplayableMIDlet(){
        display=Display.getDisplay(this);
        textBox=new TextBox("Displayable 对象","测试 Displayable 对象",50,0);
        ticker=new Ticker("测试滚动条");
```

```
            textBox.setTicker(ticker);
            textBox.setTitle("标题内容");
            String height="Displayable 对象的高度:"+textBox.getHeight();
            String width="Displayable 对象的宽度:"+textBox.getWidth();
            textBox.setString(height+width);
        }
        public void startApp(){
            display.setCurrent(textBox);
        }
        public void pauseApp(){
        }
        public void destroyApp(boolean uncondition){
        }
    }
```

程序运行结果如图 3-4 所示。


图 3-4　Displayable 对象的使用

在【例 3-2】中，Ticker 是运行在 TextBox 类上面的一个滚动条，它直接继承自 Displayable 类，与其他控件（如 TextBox）是并列关系。在某控件上设置滚动条时，需要使用控件的 setTicker 方法，如本例中的 textBox. setTicker(ticker)方法就是用于实现在文本框 textBox 中显示滚动条 ticker。

### 3.2.3　Command 类

由 Command 类产生的实例表示一个命令控件，该控件用于提供给用户执行某个特别命令的接口，完成用户与程序的交互。Command 类的构造方法需要 3 个参数：标签、类型和优先级。构造方法的返回值表示构造完毕的 Command 类的实例。如下语句格式定义了一个 Command 类的实例。

```
Exit=new Command("退出",Command.EXIT,1);
```

其中，第一个参数“退出”表示该命令在屏幕中所显示的标题；第二个参数 Command. EXIT 表示命令的类型，有关命令类型及其说明如表 3-3 所示；第三个参数“1”定义了优先级，数值越小优先级越高，JAM 根据优先级的高低处理命令之间的冲突。

表 3-3　命令类型及其描述

| 优先级值 | 命令类型 | 描　　述 |
|---|---|---|
| 1 | Command. SCREEN | 应用程序自定义的命令按钮，在设备上没有直接对应的按键，而仅仅是对应到屏幕上 |
| 2 | Command. BACK | 回退前一个屏幕，用于界面的导航 |
| 3 | Command. CANCEL | 取消当前操作或对当前的提示给予否定的回答 |
| 4 | Command. OK | 确定当前操作或对当前的提示给予肯定的回答 |
| 5 | Command. HELP | 获得帮助信息 |
| 6 | Command. STOP | 停止当前操作或进程 |
| 7 | Command. EXIT | 退出应用程序 |
| 8 | Command. ITEM | 在屏幕上显示多个选项 |

一旦对 Command 类的实例构造完成，就不能再对标题、类型和优先级进行任何修改，但允许分别使用方法 getLabel、getCommandType 和 getPriority 获得一个已经存在的 Command 类的实例的标题、类型和优先级。

使用 Displayable 类的 addCommand 方法可以将已经构造好的 Command 类的实例添加到 Displayable 类的实例中，一旦获得前台显示，Command 类的实例就显示在屏幕上。当然，如果一个 Displayable 对象添加的 Command 类的实例超过了设备的实际按键数，那么这些 Command 类的实例将以菜单的形式组织起来，显示在屏幕上。

【例 3-3】　Command 类的显示。

首先使用 Command 类的构造方法定义 5 个命令按钮实例：openCommand、editCommand、saveCommand、undoCommand 和 exitCommand，使用 TextBox 类的构造方法定义一个文本框对象 textBox，接着把 5 个命令按钮使用 addCommand 方法添加到文本框对象 textBox 中。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class CommandMIDlet extends MIDlet{
      private Display display;
      private TextBox textBox;
      private Command openCommand;
      private Command editCommand;
      private Command saveCommand;
      private Command undoCommand;
      private Command exitCommand;
      public CommandMIDlet(){
```

```
            display=Display.getDisplay(this);
            textBox=new TextBox("Command 对象","测试 Command 对象",50,0);
            openCommand=new Command("打开",Command.SCREEN,1);
            editCommand=new Command("编辑",Command.SCREEN,1);
            saveCommand=new Command("保存",Command.SCREEN,1);
            undoCommand=new Command("撤销",Command.SCREEN,1);
            exitCommand=new Command("退出",Command.SCREEN,1);
            textBox.addCommand(openCommand);
            textBox.addCommand(editCommand);
            textBox.addCommand(saveCommand);
            textBox.addCommand(undoCommand);
            textBox.addCommand(exitCommand);
        }
        public void startApp(){
            display.setCurrent(textBox);
        }
        public void pauseApp(){
        }
        public void destroyApp(boolean uncondition){
        }
    }
```

程序的运行结果如图 3-5 所示。当用户按下手机键盘上的右软键选择“编辑”菜单时，显示添加到屏幕上的命令。按上下方向键可以选择不同的命令，选择选中键可以激活命令。在本例中，并没有添加命令对应的动作，因此选择某个命令后屏幕没有任何动作。


图 3-5　Command 类实例演示

本例类似的以下两行代码：

```
openCommand＝new Command("打开",Command.SCREEN,1);
textBox.addCommand(openCommand);
```

都可以合并成如下形式。

```
textBox.addCommand(new Command("打开",Command.SCREEN,1));
```

## 3.3 事件处理接口

当用户和应用程序之间发生了交互事件(如用户按下某个按钮,或者由于用户的操作引起某个控件的状态发生变化)时,应用程序如何监听和处理这种交互事件呢? 这就需要相应的事件处理接口。Java ME 中提供的事件处理接口有两个:CommandListener(命令监听器)和 ItemStateListener(状态监听器)。CommandListener 接口主要用于监听命令类控件的交互,ItemStateListener 接口主要用于监听 Item 类控件的交互。当应用程序要对用户的交互事件进行监听和处理时,必须先在产生交互事件的对象上使用 Displayable 类的 setCommandListener 或 setItemListener 方法注册监听者,然后在程序中实现相应接口的方法 commandAction 或 itemStateChange。

### 3.3.1 CommandListener 接口

在高级用户界面中,程序与用户的交互是通过命令触发监听器来实现的,当在界面中添加用户触发的命令按钮并做出处理时,需要 Screen 子类的实例实现 CommandListener(命令监听器)接口,该接口声明的 commandAction 方法作为响应用户命令的处理过程。

命令监听器接口的使用步骤如下。

(1)程序定义一系列的命令按钮(Command 类的实例)。

(2)将命令按钮添加到 Displayable 类的对象中,如 Form、Alert、List、TextBox 等。

(3)调用 Displayable 类的 setCommandListener 方法,将 CommandListener 接口注册到需要监听用户交互事件的对象上。

(4)在需要响应用户命令的类中,声明实现 CommandListener 接口,并在该接口的 commandAction 方法中添加处理这些用户命令的代码。

【例 3-4】　CommandListener 接口的使用。

本例是在【例 3-3】的基础上实现了 CommandListener 接口,并在该接口的 commandAction方法中添加了相关处理代码。

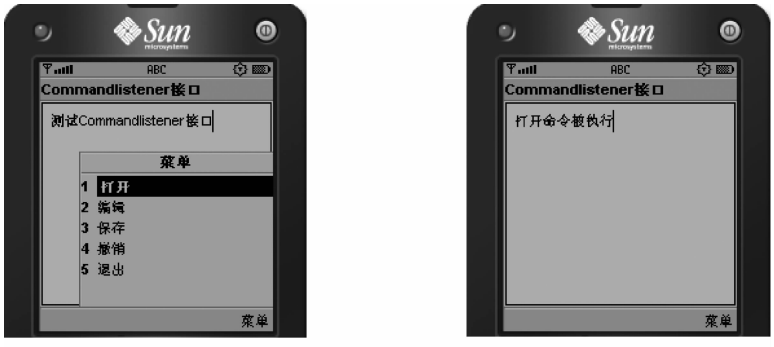
具体代码如下。

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class CommandListenerMIDlet extends MIDlet implements CommandListener{
    private Display display;
```

```
    private TextBox textBox;
    private Command openCommand;
    private Command editCommand;
    private Command saveCommand;
    private Command undoCommand;
    private Command exitCommand;
    public CommandListenerMIDlet(){
        display=Display.getDisplay(this);
textBox=new TextBox("CommandListener 接口","测试 CommandListener 接口",50,0);
        openCommand=new Command("打开",Command.SCREEN,1);
        editCommand=new Command("编辑",Command.SCREEN,1);
        saveCommand=new Command("保存",Command.SCREEN,1);
        undoCommand=new Command("撤销",Command.SCREEN,1);
        exitCommand=new Command("退出",Command.SCREEN,1);
        textBox.addCommand(openCommand);
        textBox.addCommand(editCommand);
        textBox.addCommand(saveCommand);
        textBox.addCommand(undoCommand);
        textBox.addCommand(exitCommand);
    }
public void startApp(){
        textBox.setCommandListener(this);
        display.setCurrent(textBox);
}
public void pauseApp(){
}
public void destroyApp(boolean uncondition){
}
public void commandAction(Command c,Displayable s){
        if(c==openCommand){
            textBox.setString("打开命令被执行");
        }
        if(c==editCommand){
            textBox.setString("编辑命令被执行");
        }
        if(c==saveCommand){
            textBox.setString("保存命令被执行");
        }
```

```
            if(c==undoCommand){
                  textBox.setString("撤销命令被执行");
            }
            if(c==exitCommand){
                  textBox.setString("退出命令被执行");
                  destroyApp(false);
                  notifyDestroyed();
            }
      }
}
```

程序的运行结果如图 3-6 所示。




(a)选择“打开”命令　　(b)“打开”命令执行效果

图 3-6　CommandListener 接口演示

在本例中，为每一个命令定义一个对应操作。建立命令监听器监听用户的命令，当用户选择图 3-6(a)所示的“打开”命令时，监听器自动运行 commandAction 方法，方法的参数是 Command 类型和 Displayable 类型的变量，分别表示用户激活的 Command 类及 Displayable 类的实例，然后根据不同的用户命令执行不同的代码，本例中是打印相应的字符串，当执行“打开”命令时，屏幕输出“打开命令被执行”的字符串，如图 3-6(b)所示。

### 3.3.2　ItemStateListener 接口

当用户对屏幕上某个 Item 控件(如文本编辑框、数字编辑框、日期编辑框、单选框、复选框、图像区域等)的内容进行修改时，可以认为这也是一种交互，这一点类似于 Command 类，都需要创建一个实例，然后设置监听器用于与用户的交互。不同的是 Command 类一般用于监听命令类控件，而 Item 类用于响应用户对控件内容的修改或者控件状态的改变，如当文本框的文本由于用户的编辑而发生改变时，或者当单选框由于用户的选择而发生选项变化时。

Item 类的监听器是 ItemStateListener 接口，由 itemStateChange 方法实现当 Item 控件发生变化时，程序所要执行的操作。Command 类和 Item 类的监听器的比较如表 3-4 所示。

表 3-4　Command 类与 Item 类的监听器的比较

| 比较类别 | Command 类的监听器 | Item 类的监听器 |
|---|---|---|
| 接口类名称 | CommandListener | ItemStateListener |
| 承载的实例 | Displayable 子类的实例 | Item 类的实例 |
| 与实例关联的方法 | setCommandListener | setItemListener |
| 响应的方法 | commandAction | itemStateChange |
| 参数 | 触发的命令 Command 类实例和当前屏幕 Displayable 子类实例 | 被设置为监听器的且发生状态变化的 Item类实例 |

注意：只有当用户的操作改变 Item 的内容或状态时，程序才会调用 itemStateChange 方法，如果是应用程序自身引起 Item 发生变化，那么系统不进行特别处理。

## 3.4 高级界面显示类

高级界面显示类 Screen 本身并没有界面显示和用户交互的功能，也没有定义方法，它只是起到组织高级界面类的作用，所以在实际使用过程中都是使用 Screen 类的子类。高级界面显示类的组成结构在图 3-2 中已经介绍，在此不再叙述。需要注意的是，在 MIDP 2.0 规范中，将原来派生自 Screen 类的 Ticker 类移到了与 Screen 类并列的位置，相应的方法也移到了 Displayable 类中，因此严格来说，Ticker 类并不是 Screen 类的子类。

### 3.4.1 Alert 类

Alert(警告窗口)类的实例可以在屏幕上显示一个指定内容的对话框，通常情况下用于显示表示警告含义的信息。当显示 Alert 类的实例时，它会自动获得焦点，原有屏幕则失去焦点，当用户按下任意键或者等待一段时间后，警告框被解除，恢复为原有的屏幕内容。

Alert 类的主要方法如表 3-5 所示。

表 3-5　Alert 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| Alert | 无 | String | 构造一个 Alert 类的实例 |
| Alert | 无 | String,String,Image,AlertType | 构造一个 Alert 类的实例 |
| addCommand | void | Command | 添加一个命令 |
| removeCommand | void | Command | 删除指定的命令 |
| getDefaultTimeout | int | void | 获得默认的超时时间，单位是毫秒 |
| getImage | Image | void | 获得警告框使用的图像 |
| setImage | void | Image | 设置警告框使用的图像 |
| getIndicator | Gauge | void | 获得警告框中的进度，即当前已经持续的时间，单位是毫秒 |

续表

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| setIndicator | void | Image | 设置警告框的进度 |
| getString | String | void | 获得警告框中的信息 |
| setString | Int | String | 设置警告框中的信息 |
| getTimeout | void | int | 获得警告框超时时间，单位是毫秒 |
| setTimeout | void | void | 设置警告框超时时间，单位是毫秒 |
| getType | AlertType | void | 获得警告框类型 |
| setType | void | AlertType | 设置警告框类型 |
| setCommandListener | void | CommandListener | 设置命令监听器 |

Alert 类有以下两种构造方法。

(1)Alert(String s)，其中 s 表示警告框的标题。

(2)Alert(String t,String s,Image img,AlertType typename)，其中 t 表示警告框的标题，s 表示警告框的内容，img 表示警告框使用的图标，typename 表示警告框的类型。

警告框的类型如表 3-6 所示。

表 3-6　警告框的类型

| 类　型 | 说　明 |
|---|---|
| AlertType. ALARM | 提示警告信息的警告框 |
| AlertType. CONFIRMATION | 提示用户对某个操作进行确认 |
| AlertType. ERROR | 向用户警告一个错误信息 |
| AlertType. INFO | 提供一般性的信息 |
| AlertType. WARNING | 警告用户一个危险的动作可能被执行 |

各种类型警告框的界面都是类似的，只不过不同的警告框对应不同的声音效果。

在构造警告框时，也可以不指定构造方法的所有参数，例如：

```
Alert(null,null,null,null);
```

表示一个空白的警告框。另外，警告框不能接收用户的任何输入，但可以添加 Command 类的对象。

注意：在 MIDP 1.0 规范中，Alert 类不允许添加 Command 类的实例，在 MIDP 2.0 中允许添加 Command 类的对象，也可以设置监听器。

【例 3-5】　Alert 类的使用。

本例中首先调用构造方法建立 Alert 类的实例，在初始屏幕中显示 TextBox 实例，其中添加了两个命令："警告"和"退出"，当用户打开菜单并触发"警告"命令时，警告框被设置为当前屏幕显示。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class AlertMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Alert alert;
    private TextBox textBox;
    private Command alertCommand;
    private Command exitCommand;
    public AlertMIDlet(){
        display=Display.getDisplay(this);
        textBox=new TextBox("Alert","测试警告框",50,0);
        alertCommand=new Command("警告",Command.SCREEN,1);
        exitCommand=new Command("退出",Command.SCREEN,1);
        textBox.addCommand(alertCommand);
        textBox.addCommand(exitCommand);
        alert=new Alert("警告框标题","警告框内容",null,AlertType.INFO);
        alert.setTimeout(Alert.FOREVER);
    }
    public void startApp(){
        textBox.setCommandListener(this);
        display.setCurrent(textBox);
    }
    public void pauseApp(){
    }
    public void destroyApp(boolean uncondition){
    }
    public void commandAction(Command c,Displayable s){
        if(c==alertCommand){
            display.setCurrent(alert,display.getCurrent());
        }
        if(c==exitCommand){
            textBox.setString("退出命令被执行");
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

运行结果如图 3-7 所示。


(a)选择“警告”菜单　　(b)显示警告内容

图 3-7　Alert 警告框

警告框标题和内容显示在屏幕上，因为设置 Alert 的类型是 AlertType. FOREVER，所以直到用户响应后警告框才被释放。另外本例中“display. setCurrent(alert, display. getCurrent());”表示在警告框解除后，屏幕中的内容自动设置为 display. getCurrent()。

### 3.4.2　List 类

List(列表)类可以实现 Choice 接口，在屏幕上显示一系列选项，用户可以选择其中的一项或多项。List 类的实例对象提供了 Choice. EXCLUSIVE(单选)、Choice. MULTIPLE(多选)、Choice. IMPLICIT(隐含)3 种类型。表 3-7 列出了 List 类的主要方法。

表 3-7　List 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| List | 无 | String, int | 生成一个空的列表，其中，标题由第一个参数指定，类型由第二个参数指定 |
| List | 无 | String, int, String[], Image[] | 生成一个具有选项的列表，其中，标题由第一个参数指定，类型由第二个参数指定，第三个参数表示各选项的文本内容，第四个参数表示各选项的图像内容 |
| setSelected-Command | void | Command | 用于 IMPLICIT 类型的列表，设置选择的命令 |
| Remove-Command | void | Command | 删除列表的指定命令 |

List 类实例的 3 种选项类型中，EXCLUSIVE 类型和 MULTIPLE 类型必须注册一个 Command 类的实例，并与一个监听器关联。IMPLICIT 类型的 List 类自身包含一个 Command 实例，只需建立与其中一个监听器的关联即可。

在监听器接口中的 actionCommand 方法中捕获命令按钮事件，并作出相应处理，事件

处理代码格式如下。

```
public void actionCommand(Command c,Displayable s){
    if(c==SELECTED_COMMAND){
        //添加处理代码
    }
}
```

一般对于单选框，使用 getSelectedIndex 方法获得选中选项的索引号；对于复选框使用 getSelectedFlags 方法获得各选项的选择状态，而 getSelectedIndex 方法将返回－1。

【例 3-6】 在屏幕上显示 List 类实例的各种选项类型。

首先调用构造方法建立 3 种类型的 List 类实例，分别为单选列表、显示列表和多项列表，当用户从 3 种类型的 List 中选择一种时，命令监听器执行 commandAction 方法，根据用户命令的不同选择，在屏幕上显示相应的 List 类实例的选项类型。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ListMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command backCommand;
    private Command exitCommand;
    private List menuList;
    private List exclusiveList;
    private List multipleList;
    public ListMIDlet(){
        display=Display.getDisplay(this);
        backCommand=new Command("后退",Command.SCREEN,1);
        exitCommand=new Command("退出",Command.SCREEN,1);
        String[]listType={
            "单选列表",
            "显示列表",
            "多项列表"
        };
        menuList=new List("选择列表类型",Choice.IMPLICIT,listType,null);
        menuList.addCommand(exitCommand);
        menuList.setCommandListener(this);
        String[] exclusiveListContent={
            "单选项 1",
            "单选项 2",
            "单选项 3",
```

```
                "单选项 4",
                "单选项 5",
                "单选项 6"
        };
        exclusiveList=new List("单选列表",Choice.EXCLUSIVE,
        exclusiveListContent,null);
        exclusiveList.addCommand (backCommand);
        exclusiveList.addCommand (exitCommand);
        exclusiveList.setCommandListener(this);
        String[] multipleListContent={
                "复选项 1",
                "复选项 2",
                "复选项 3",
                "复选项 4",
                "复选项 5",
                "复选项 6"
        };
        multipleList=new List("复选列表",Choice.MULTIPLE,multipleList-
                                                Content,null);
        multipleList.addCommand(backCommand);
        multipleList.addCommand(exitCommand);
        multipleList.setCommandListener(this);
}
public void startApp(){
        display.setCurrent(menuList);
}
public void pauseApp(){
}
public void destroyApp(boolean uncondition){
}
public void commandAction(Command c,Displayable d){
        if(d.equals(menuList)){
                if(c==List.SELECT_COMMAND){
                        switch(((List)d).getSelectedIndex()){
                                case 0:
                                        display.setCurrent(exclusiveList);
                                        break;
                                case 1:
                                        display.setCurrent(multipleList);
```

```
                        break;
                    }
                }
            }
            else if(c==backCommand){
                display.setCurrent(menuList);
            }
            else if(c==exitCommand){
                destroyApp(false);
                notifyDestroyed();
            }
        }
    }
```

程序运行结果如图 3-8 所示。


(a)选择“单选列表”　　(b)单选列表选项

图 3-8　List 列表框演示

在图 3-8(a)中选择了单选列表后，屏幕出现了图 3-8(b)所示的含有 6 个选项的单选列表，在列表中添加了“后退”和“退出”命令，可以选择单选列表的某一项，也可以选择“后退”命令回到前一个界面，选择“退出”命令退出程序。

【例 3-7】　布局策略的使用，具体功能为：在默认情况下，对于超出屏幕的文本使用换行显示的布局策略。

在 List 实例中添加 3 个 Command 类实例用于改变当前布局策略。

具体代码如下。

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class FitPolicyMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command offCommand;
```

```
    private Command onCommand;
    private Command defaultCommand;
    private List fitPolicyList;
    public FitPolicyMIDlet(){
        display=Display.getDisplay(this);
        fitPolicyList=new List("FitPolicy 演示",Choice.IMPLICIT);
        fitPolicyList.append("有可能超过屏幕显示宽度的字符串有可能超过屏
                              幕显示宽度的字符串",null);
        offCommand=new Command("WRAP_OFF",Command.OK,1);
        fitPolicyList.addCommand(offCommand);
        onCommand=new Command("WRAP_ON",Command.OK,1);
        fitPolicyList.addCommand(onCommand);
        defaultCommand=new Command("WRAP_DEFAULT",Command.OK,1);
        fitPolicyList.addCommand(defaultCommand);
    }
    public void startApp(){
        fitPolicyList.setCommandListener(this);
        display.setCurrent(fitPolicyList);
    }
    public void commandAction(Command c,Displayable d){
        if(c==offCommand){
            ((List)d).setFitPolicy(Choice.TEXT_WRAP_OFF);
        }
        if(c==onCommand){
            ((List)d).setFitPolicy(Choice.TEXT_WRAP_ON);
        }
        if(c==defaultCommand){
            ((List)d).setFitPolicy(Choice.TEXT_WRAP_DEFAULT);
        }
    }
    public void pauseApp(){
    }
    public void destroyApp(boolean unconditional){
    }
}
```

程序运行结果如图 3-9 所示。


图 3-9　布局策略演示结果

当用户在菜单中选择 WRAP_OFF 选项时，commandAction 方法使用 setFitPolicy 将当前的布局策略设置为 WRAP_OFF，此时超过屏幕宽度的字符串不能显示。

### 3.4.3　TextBox 类

TextBox(文本框)类的实例是一个可以包含多行文本的编辑框，用于接收用户的输入。TextBox 类直接继承自 Screen 类。

#### 1. TextBox 类的方法

TextBox 类的构造方法的语法格式如下。

```
TextBox(String title,String initText,int maxLength,int constrains);
```

其中，title 表示文本框标题，initText 表示文本初始内容，maxLength 表示文本框允许输入的最大长度，constrains 表示输入限制。这些参数可以在构造时指定，也可以在构造完成后指定。TextBox 类的主要方法如表 3-8 所示。

表 3-8　TextBox 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| TextBox | 无 | String,String,int,int | 构造方法，第一个参数指定文本框标题，第二个参数指定文本框最初的内容，第三个参数指定文本长度的最大值，第四个参数指定输入文本的限制 |
| delete | void | int,int | 删除指定位置和长度的文本内容，第一个参数指定删除的起始位置，第二个参数指定删除的长度 |
| getCarePosition | int | void | 获得当前文本框中光标的位置 |
| getChars | int | char[] | 获得当前文本框的内容，并放置到参数指定的字符数组中 |
| setChars | void | char[],int,int | 用第一个参数指定的字符串替换当前文本框中的指定文本，第二个参数指定文本的起始位置，第三个参数指定文本的长度 |

续表

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| getConstrains | int | void | 获得当前文本框的输入限制 |
| setConstrains | void | int | 设置当前文本框的输入限制 |
| getMaxSize | int | void | 获得当前文本框允许输入的最大字符数 |
| setMaxSize | int | int | 设置当前文本框允许输入的最大字符数 |
| getString | String | void | 获得当前文本框的标题 |
| setString | void | String | 设置当前文本框的标题 |
| insert | void | char[],int,int, int | 将第一个参数指定的字符串部分内容插入当前文本框中指定的位置 |
| insert | void | String,int | 将第一个参数指定的字符数组中的所有内容插入当前文本框中的指定位置,第二个参数指定了这个位置 |
| setInitialInputMode | void | String | 设置初始输入模式 |

下面分别从几个方面介绍 TextBox 类的使用。

### 2. TextBox 类的使用

1)文本最大长度限制

使用 setMaxSize 方法设置文本编辑框中文本的最大长度,用 getMaxSize 方法获得文本最大长度。

在设置文本最大长度时要注意以下几点。

(1)不能设置成 0 或－1。

(2)不能等于或超过 300 000。

(3)如果在构造方法中设置最大长度,则其值不能小于初始字符串的长度。

当文本不能在一行中显示时,程序会自动提供滚动条,而不需要程序进行任何设置或修改。

2)文本输入类型限制

文本输入类型限制是指对用户向 TextBox 控件输入的内容进行限制,如只允许用户输入数字,只允许输入字母等。在 MIDP 规范中,定义了两种类型的限制:输入限制和系统限制。

(1)输入限制。输入限制是 TextBox 控件所接受的字符集合和格式,包括以下 6 种类型。

- TextField. ANY:允许用户输入任何字符和数字。
- TextField. NUMERIC:只允许用户输入数字。
- TextField. EMAILADDR:只允许用户输入邮件地址。
- TextField. PHONENUMBER:只允许用户输入电话号码。
- TextField. URL:只允许用户输入 URL 地址。
- TextField. DECIMAL:只允许用户输入数值。

(2)系统限制。系统限制是对 TextBox 控件的状态、显示的限制,而不关心用户输入的内容,它包括以下 6 种类型。

- TextField. PASSWORD:以星号显示用户输入的每一个密码字符。
- TextField. UNEDITABLE:设置当前的 TextBox 为不可编辑的状态。

- TextField.SENSITIVE:不保存用户当前在 TextBox 中的任何输入。
- TextField.NON_PREDICTIVE:禁用输入法对词组的预测功能。
- TextField.INITIAL_CAPS_WORD:设置输入时在每个空格之后切换到大写输入,随后切换到小写输入。
- TextField.INITIAL_CAPS_SENTENCE:设置输入时在每个句子的开头切换到大写输入,随后切换到小写输入。

这些限制可以同时使用,但前提是同时使用的限制不能相互矛盾。同时使用的限制之间用“|”分隔。

3)文本内容的设置和获取

getString 方法和 getChars 方法都可以获得当前 TextBox 中的文本内容,getString 方法返回的是一个 String 类型的对象,getChars 方法将文本内容组织成字符数组。字符数组的长度要超过文本内容,以避免 ArrayOutofBoundsException 异常。

setString 方法和 setChars 方法可以设置当前 TextBox 中的内容。insert 方法提供了向当前文本中插入指定字符串的功能。

【例 3-8】 文本输入类型限制的实现。

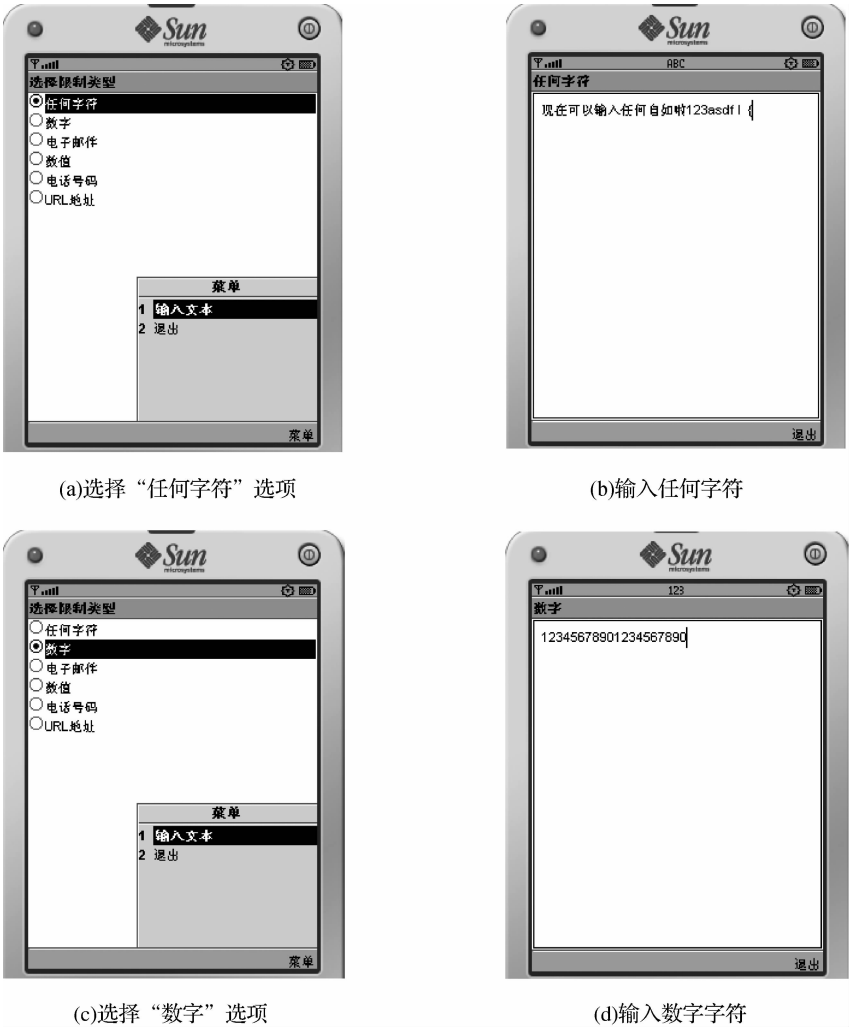
建立一个单选类型的 List 控件,用于显示 6 种输入限制,将这 6 种限制的名称存储在字符串数组 constraintsName 中,将对应的限制类型存储在 constraintsType 数组中。在初始化过程中,建立两个按钮 inputCommand 和 exitCommand,再建立一个 TextBox 用于接收用户的输入,最后建立 List 控件,将 6 种限制以单选按钮的形式显示在屏幕上。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TextBoxConstraintsMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command exitCommand;
    private Command inputCommand;
    private List constraintsList;
    private TextBox textBox;
    static final String[] constraintsName={
        "任何字符",
        "数字",
        "电子邮件",
        "数值",
        "电话号码",
        "URL 地址"
    };
    static final int[] constraintsType={
        TextField.ANY,
        TextField.NUMERIC,
```

```
            TextField.EMAILADDR,
            TextField.DECIMAL,
            TextField.PHONENUMBER,
            TextField.URL
        };
        public TextBoxConstraintsMIDlet(){
            display=Display.getDisplay(this);
            exitCommand=new Command("退出",Command.SCREEN,1);
            inputCommand=new Command("输入文本",Command.SCREEN,1);
            textBox=new TextBox("","",200,0);
            constraintsList=new List("选择限制类型",Choice.EXCLUSIVE,
            constraintsName,null);
            constraintsList.addCommand(inputCommand);
            constraintsList.addCommand(exitCommand);
            constraintsList.setCommandListener(this);
        }
        public void startApp(){
            display.setCurrent(constraintsList);
        }
        public void pauseApp(){
        }
        public void destroyApp(boolean uncondition){
        }
        public void commandAction(Command c,Displayable d){
            if(d.equals(constraintsList)){
                if(c==inputCommand){//当用户选择“输入文本”命令时
                    int t=((List)d).getSelectedIndex();
                    textBox.setTitle(constraintsName[t]);
                    textBox.setConstraints(constraintsType[t]);
                    textBox.addCommand(exitCommand);
                    textBox.setCommandListener(this);
                    display.setCurrent(textBox);
                }
            }
            else if(c==exitCommand){
                destroyApp(false);
                notifyDestroyed();
            }
        }
    }
```

程序运行结果如图 3-10 所示。




(a)选择“任何字符”选项　(b)输入任何字符

(c)选择“数字”选项　(d)输入数字字符

图 3-10　文本输入限制演示

从图 3-10 中可以看出，选择一种类型的限制，按下右软键打开菜单，选择“输入文本”命令，即 inputCommand，此时 CommandListener 监听到用户的命令，在 commandAction 方法中找到 inputCommand 对应的操作，设置 TextBox 控件的标题为限制类型的名称，设置限制类型为用户选择的限制类型，然后使用 setCurrent 方法使屏幕显示 TextBox 控件，此时在屏幕上的输入受到先前选择的限制，如图 3-10(c)所示，选中“数字”单选按钮后，在 TextBox 中只能输入数字，如图 3-10(d)所示。

本例的关键在于，通过 commandAction 方法的参数 d，将用户对限制的选择传递到 TextBox 控件中，从而执行设置的操作。

【例 3-9】　获得和设置 TextBox 中的文本内容。

在 TextBox 控件中添加 3 个命令，分别用于输入文本、清除用户的输入和退出程序。程序启动时，TextBox 控件显示在整个屏幕上，用户可以自由输入文本，如图 3-11(a)所示，选

择菜单中的“输入文本”命令时，命令监听器找到对应的 commandAction 方法，使用 getString 方法获得当前 TextBox 中的内容，再调用 insert 方法，在 size()中插入指定文本，因为 size 方法返回的是整个“TextBox 的文本长度，所以 textBox. insert("string",textbox. size());”等价于将 string 插入当前文本的最后，如图 3-11(b)所示，将“您好，欢迎进入 TextBox”的内容添加到用户输入的后面。最后选择菜单中的“清除输入”命令，命令监听器执行 setString 方法，重新对 TextBox 中的内容进行设置，由于参数是空格符，即将内容设置为空格，相当于清除 TextBox 中的内容。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TextBoxEditMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command exitCommand;
    private Command inputCommand;
    private Command clearCommand;
    private TextBox textBox;
    public TextBoxEditMIDlet(){
        display=Display.getDisplay(this);
        exitCommand=new Command("退出",Command.SCREEN,1);
        inputCommand=new Command("输入文本",Command.SCREEN,1);
        clearCommand=new Command("清除输入",Command.SCREEN,1);
        textBox=new TextBox("请输入您的姓名","",200,TextField.ANY);
        textBox.addCommand(inputCommand);
        textBox.addCommand(clearCommand);
        textBox.addCommand(exitCommand);
        textBox.setCommandListener(this);
    }
    public void startApp(){
        display.setCurrent(textBox);
    }
    public void pauseApp(){
    }
    public void destroyApp(boolean uncondition){
    }
    public void commandAction(Command c,Displayable d){
        if(c==inputCommand){
            textBox.insert("您好,欢迎进入 TextBox",textBox.size());
```

```
            }
            else if(c==clearCommand){
                textBox.setString("");
            }
            else if(c==exitCommand){
                destroyApp(false);
                notifyDestroyed();
            }
        }
    }
```

程序的运行结果如图 3-11 所示。


(a)选择“输入文本”命令


(b)输入字符


(c)选择“清除输入”命令


(d)原有字符全部清除

图 3-11　编辑 TextBox 中的文本

## 3.5 Form 类

前面介绍了 Screen 类及其 Alert、List、TextBox 子类的使用方法。下面介绍 Form（容

器控件)类及其子类 Item。

### 3.5.1　Form 类概述

Form 类是一个典型的容器控件类,用于包含其他高级用户界面控件,一般不单独出现在屏幕上。Item 类是可以内含于 Form 容器中的控件类。在很多情况下,屏幕界面由一个 Form 类的实例和其中的各种 Item 类的实例共同组成。MIDlet 可以实现整个 Form 的外观,包括布局、元素控件焦点的切换和滚动。

每一个 Item 类的实例只能存在于唯一的 Form 容器中,如果同一实例放在不同的 Form 容器中,就会产生 IllegalStateException 异常。

在布局方面,Form 类是按列进行组织的,每一类都具有相同的宽度,没有水平方向上的滚动条。在垂直方向上,Form 的高度与其中 Item 的个数和高度有关。

#### 1. Form 类的方法

Form 类的主要方法如表 3-9 所示。

表 3-9　Form 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| Form | 无 | String | 构造方法,根据参数指定的标题构造一个新的 Form 实例 |
| Form | 无 | String,Item[] | 构造方法,根据第一个参数指定标题,第二个参数指定 Form 中的 Item 控件,建立新的 Form 实例 |
| append | int | Image | 为当前 Form 追加一个图像控件 |
| append | int | Item | 为当前 Form 追加一个 Item 控件 |
| append | int | String | 为当前 Form 追加一个字符串控件 |
| delete | void | int | 删除 Form 中指定索引号的 Item 控件,参数指定了索引号 |
| deleteAll | void | void | 删除当前 Form 中所有的 Item 控件 |
| get | Item | int | 获得 Form 中指定索引号的 Item 控件,参数为指定索引号 |
| getHeight | int | void | 获得当前 Form 的高度 |
| getWidth | int | void | 获得当前 Form 的宽度 |
| insert | void | int,Item | 在第一个参数指定的索引号 Item 控件前插入一个由第二个参数指定的 Item 控件 |
| set | void | int,Item | 更改第一个参数指定的索引号 Item 控件为第二个参数指定的 Item 控件 |
| set | void | ItemStateListener | 为当前 Form 设置监听器 ItemStateListener |
| size | int | void | 获得当前 Form 中的 Item 控件数 |

### 2. Form 类的使用

下面分别从几个方面介绍 Form 类的使用。

1)Form 对象的构造

可以使用 Form 类的两个构造方法(具体见表 3-9 中的前两个方法)来创建 Form 类的对象,创建对象时,可以仅指定 Form 对象的标题,使用的构造方法是:

```
Form(String caption);
```

也可同时指定初始 Item 控件,使用的构造方法是:

```
Form(String caption,Item[] itemList);
```

上述构造方法中,caption 是 Form 对象的标题,itemList 是存放多个初始控件的数组。

2)添加和删除 Item

在构造 Form 对象时,可以初始化 Item 控件,一旦 Form 对象创建完毕,就可以使用 append、delete 和 insert 方法来追加、删除和插入 Item 控件。

与 List 控件类似,Form 中的每一个 Item 都有一个索引号唯一标识它,索引号从 0 开始。在进行插入和删除操作时,需要指定索引号。如果索引号错误,就会产生 IndexOutofBoundException 异常。

3)添加 Command 类的实例

Form 对象可以使用 addCommand 方法添加 Command 类的实例,使用 setCommandListener 方法设置命令监听器,用于与用户的交互,例如:

```
myForm.addCommand(cmdQuit);  //在 Form 对象 myForm 中添加命令按钮 cmdQuit
myForm.setCommandListener();  //为 Form 对象 myForm 设置命令监听器
```

4)设置 Item 状态监听器

当 Form 中某个 Item 的状态发生变化时,可能会引起某些操作。可以在 Form 中设置 Item 状态监听器,即 itemStateListener。在 Item 状态发生变化或者执行了 notifyStateChanged 方法后,会发出一个状态改变的消息,Form 中的 itemStateListener 会捕捉到这个消息,而自动激活 itemStateChanged 方法。可以在 itemStateChanged 方法中添加需要执行的代码,其中参数是发出状态改变消息的 Item 类的实例。

## 3.5.2 Item 类

Item 类是 Form 类的派生类,通过改变 Item 类的派生类的实例状态,用户可以和应用程序进行交互。Item 类包含 8 个直接的子类:ChoiceGroup(选择组)、CustomItem(自定义控件)、DateField(日期域)、Gauge(进度条)、ImageItem(图像控件)、Spacer(定位控件)、StringItem(字符串控件)和 TextField(文本域)。后面将详细介绍这 8 个类。

### 1. Item 类的属性

Item 类具有如下属性。

1)标题

所有的 Item 对象都有一个标题,显示在屏幕上,起到标识的作用。MIDP 会自动调整

Item对象和其标题的显示位置，保证它们具有明显的相关性。在用户拖动滚动条或者进行其他交互操作时，系统总能保证 Item 对象和标题在合理的位置。

2）布局

Form 中可以包含多个 Item，它们的类型不一定相同，为此，MIDP 通过布局指示安排这些 Item 的位置。布局指示标识符如表 3-10 所示。

表 3-10　布局指示标识符

| 布局指示标识符 | 含　义 |
|---|---|
| LAYOUT_DEFAULT | 默认布局 |
| LAYOUT_LEFT | 水平方向左对齐 |
| LAYOUT_RIGHT | 水平方向右对齐 |
| LAYOUT_CENTER | 水平方向居中 |
| LAYOUT_TOP | 垂直方向上对齐 |
| LAYOUT_BOTTOM | 垂直方向下对齐 |
| LAYOUT_VCENTER | 垂直方向居中 |
| LAYOUT_NEWLINE_BEFORE | 置于新行或列前 |
| LAYOUT_NEWLINE_AFTER | 置于新行或列后 |
| LAYOUT_SHRINK | 在水平方向上缩小以适应屏幕 |
| LAYOUT_VSHRINK | 在垂直方向上缩小以适应屏幕 |
| LAYOUT_EXPAND | 在水平方向上扩展以适应屏幕 |
| LAYOUT_VEXPAND | 在垂直方向上扩展以适应屏幕 |
| LAYOUT_2 | 使 MIDP 1.0 的布局适应 MIDP 2.0 布局 |

水平方向的布局可以和垂直方向上的布局结合使用，中间用“|”分隔，水平方向或垂直方向的布局内部不能结合使用。

3）尺寸

Item 的尺寸有两个概念：最小尺寸（minimum size）和偏好尺寸（preferred size）。其中最小尺寸是指在运行时至少需要的空间，偏好尺寸是指运行时的最佳显示尺寸。

可以分别使用 getMinimumHeight 方法和 getMinimumWidth 方法获得最小高度和最小宽度，也可以使用 getPreferredHeight 方法和 getPreferredWidth 方法获得偏好高度和宽度，setPreferredSize 方法用于设置偏好尺寸。

4）状态改变

设置 Item 监听器的 Form 会获得 Item 状态发生变化时的消息。在以下几种情况下，会发出状态变化的消息。

- ChoiceGroup 中的选择状态发生变化。
- Gauge 中的值发生变化。
- TextFiled 中的内容发生变化。
- DateFiled 中的日期发生变化。
- 调用了 notifyStateChanged 方法。

注意：由于程序自身原因，改变 Item 的某个属性或者在 Form 之间切换时，不会发出状

态改变的消息。

5)外观

Item 中字符串的外观一般有 3 种:PLAIN(普通)、HYPERLINK(超链接)和 BUTTON(按钮)。

PLAIN 用于显示无交互的文本或图像,HYPERLINK 用于显示表示链接地址的文本,BUTTON 表示按钮上的文本。

### 2. Item 类的方法

在 Item 类的实例中可以有 Command 类的对象,也可以通过命令监听器与用户交互。在 Item 类中,将 commandListener 称为 ItemCommandListener,其用法与 CommandListener 类似。ItemCommandListener 的使用步骤如下。

(1)使用 addCommand 添加命令对象。

(2)使用 setItemCommandListener 方法设定监听器。

(3)使用 CommandAction 方法设置执行触发命令时的操作。

不同的是,Item 类可以通过 setDefaultCommand 方法设置默认的命令。Item 类的主要方法如表 3-11 所示。

表 3-11　Item 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| addCommand | void | Command | 向当前 Item 实例添加命令 |
| removeCommand | void | Command | 从当前 Item 实例中删除指定的命令 |
| setDefaultCommand | void | Command | 设置当前 Item 的默认命令 |
| setItemCommandListener | void | ItemCommandListener | 设置当前 Item 的命令监听器 |
| getLabel | String | void | 获得当前 Item 的标题 |
| setLabel | void | String | 设置当前 Item 的标题为参数指定的字符串 |
| getLayout | int | void | 获得当前 Item 的布局 |
| setLayout | void | int | 设置当前 Item 的布局为参数指定的布局 |
| getMinimumHeight | int | void | 获得当前 Item 最小尺寸的高度 |
| getMinimumWidth | int | void | 获得当前 Item 最小尺寸的宽度 |
| getPreferredHeight | int | void | 获得当前 Item 偏好尺寸的高度 |
| getPreferredWidth | int | void | 获得当前 Item 偏好尺寸的宽度 |
| setPreferredSize | void | int,int | 设置当前 Item 的偏好尺寸,第一个参数为宽度,第二个参数为高度 |
| notifyStateChanged | void | void | 当希望通知 Form 当前 Item 状态发生变化时调用此方法 |

除了 getLabel 和 setLabel 方法是在 MIDP 1.0 中就开始使用的外，其他方法都是 MIDP 2.0 中新增加的方法。

### 3.5.3　StringItem 类

#### 1. StringItem 类的方法

StringItem(操作字符串)类用于显示一个字符串，它包含的文本是静态的，不能被用户编辑，但它可以被程序自身修改。它的主要方法如表 3-12 所示。

表 3-12　StringItem 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| StringItem | 无 | String,String | 构造方法，第一个参数指定初始内容 |
| StringItem | 无 | String,String,int | 构造方法，第一个参数指定标题，第二个参数指定初始内容，第三个参数指定外观类型 |
| getAppearanceMode | int | void | 获得当前 StringItem 的外观类型 |
| getFont | Font | void | 获得当前 StringItem 的字体类型 |
| setFont | void | Font | 设置当前 StringItem 的字体类型 |
| getText | String | void | 获得当前 StringItem 的文本内容 |
| setText | void | String | 设置当前 StringItem 的文本内容 |
| setPreferredSize | void | int,int | 设置当前 StringItem 的偏好尺寸，第一个参数是宽度，第二个参数是高度 |

#### 2. StringItem 类的使用

下面从 4 个方面介绍 StringItem 类的用法。

1)构造方法

StringItem 有两个构造方法，具体使用分别如下。

```
StringItem item1=new StringItem("标题","初始化内容");
StringItem item2=new StringItem("标题","初始化内容",Item.PLAIN);
```

将构造完毕的 StringItem 实例放入 Form 容器中，代码如下。

```
Form form1=new Form("Form 标题");
form1.append(item1);
```

2)外观设置

StringItem 的外观与整个 Item 类的外观概念相同，也有 PLAIN、HYPERLINK 和 BUTTON 之分。

3)字体设置

通过 getFont 和 setFont 方法可以获得和设置 StringItem 中文本的字体。字体设置在模块 4 的 4.4 节中详细讲述。

4)文本显示方式

StringItem 中的文本是水平显示的，当文本长度超过屏幕宽度时，自动换行，并尽量保

证一个单词不被分割成两部分。

【例 3-10】 不同字体外观的显示实现。

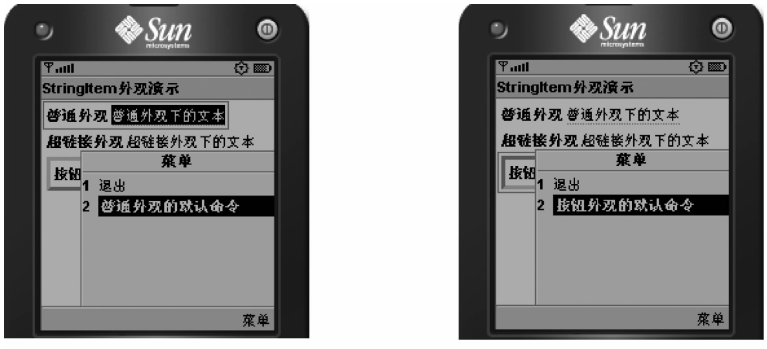
先建立一个 Form 容器，然后创建 3 个 StringItem 控件，分别使用 PLAIN、HYPERLINK和 BUTTON 外观，为每个 StringItem 控件设置一个默认的 Command 类的实例(如果不关联到默认的 Command 类，则不能看出不同外观之间的差别)，随后调用 Form 类的 append 方法将 StringItem 控件添加到 Form 窗体对象 myform 中，最后将 myform 设置为当前屏幕。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class StringItemApperenceMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command exitCommand;
    private Form myform;
    public StringItemApperenceMIDlet(){
        display=Display.getDisplay(this);
        exitCommand=new Command("退出",Command.SCREEN,1);
        myform=new Form("StringItem 外观演示");
        StringItem myPlainItem=new StringItem("普通外观","普通外观下的文
            本",Item.PLAIN);
        StringItem myHyperlinkItem=new StringItem("超链接外观","超链接外
          观下的文本",Item.HYPERLINK);
        StringItem myButtonItem=new StringItem("按钮外观","按钮外观下的文
          本",Item.BUTTON);
        myPlainItem.setDefaultCommand(new Command("普通外观的默认命令",
          Command.ITEM,1));
        myHyperlinkItem.setDefaultCommand(new Command("超链接外观的默认命
          令",Command.ITEM,1));
        myButtonItem.setDefaultCommand(new Command("按钮外观的默认命令",
          Command.ITEM,1));
        myform.append(myPlainItem);
        myform.append(myButtonItem);
        myform.insert(1,myHyperlinkItem);
        myform.addCommand(exitCommand);
        myform.setCommandListener(this);
    }
    public void startApp(){
        display.setCurrent(myform);
    }
    public void pauseApp(){
```

```
        }
        public void destroyApp(boolean unconditional){
        }
        public void commandAction(Command c,Displayable d){
            if(c==exitCommand){
                destroyApp(false);
                notifyDestroyed();
            }
        }
}
```

程序的运行结果如图 3-12 所示。




(a)普通外观显示　　(b)按钮外观显示

图 3-12　StringItem 的外观

### 3.5.4　ImageItem 类

ImageItem(操作图像)类是专门用于显示图形图像的控件,可以实现图像和文字的相互嵌入。

#### 1. ImageItem 类的方法

ImageItem 类的主要方法如表 3-13 所示。

表 3-13　ImageItem 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| ImageItem | 无 | String,Image,int,String | 构造方法,第一个参数指定控件标题,第二个参数指定图像内容,第三个参数指定布局类型,第四个参数指定可选文本 |
| ImageItem | 无 | String,Image,int,String, int | 构造方法,第一个参数指定控件标题,第二个参数指定图像内容,第三个参数指定布局类型,第四个参数指定可选文本,第五个参数指定外观类型 |

续表

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| getAltText | String | void | 获得当前 ImageItem 控件的可选文本 |
| setAltText | void | String | 设置当前 ImageItem 控件的可选文本为参数指定的字符 |
| getApperenceMode | int | void | 获得当前 ImageItem 控件的外观类型 |
| getImage | Image | void | 获得当前 ImageItem 控件的内容 |
| setImage | void | Image | 设置当前 ImageItem 控件的内容 |
| getLayout | int | void | 获得当前 ImageItem 控件的布局类型 |
| setLayout | void | void | 设置当前 ImageItem 控件的布局类型 |

### 2. ImageItem 类的使用

下面分别从几个方面介绍 ImageItem 类的使用。

1)实例构造

ImageItem 的构造方法有两种,语法格式分别如下。

```
public static ImageItem(String title,Image image,int layout,String altText);
public static ImageItem(String title,Image image,int layout,String altText,int
  apperence);
```

- title 是整个控件的标题。
- image 是 Image 类的一个对象,是图像的内容。
- layout 是布局的类型,与 Form 布局的概念一致。
- altText 是可选文本。
- apperence 是外观类型,可以是 PLAIN、HYPERLINK 或 BUTTON。

这些参数都是可选的,如果在创建时忽略,则用 null 代替。

使用构造方法创建 ImageItem 实例之前,要先建立 Form 实例,然后创建 Image 实例,并以此建立 ImageItem 实例。使用 Image 类的 createImage 方法建立 Image 实例需要注意以下两点。

(1)一定要书写 try…catch 的异常捕捉语句,防止因为图像文件无法打开而造成程序崩溃。

(2)参数是指定图像的路径,在 WTK 开发环境下,图像文件被认为是资源文件,默认的根目录是 res,因此程序中指定的"/pic.png"表示在 res 的根目录下。

使用构造方法建立 ImageItem 实例,注意布局的设置和替代文本,最后调用 Form 的 append 方法将 ImageItem 实例加入 Form 中。

2)可选文本

可选文本是控件中的图像不能在屏幕上显示时,代替显示的文本。一般都是说明图像内容的文字。

3)布局设置

ImageItem 类的布局与 Item 类的布局一致,具体内容参考表 3-11。

4)外观设置

ImageItem 类的外观与 StringItem 类一致,专指文本的显示方式。

【例 3-11】　ImageItem 控件的创建和布局。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ImageItemMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command exitCommand;
    private Form myform;
    private ImageItem myImageItem1;
    private ImageItem myImageItem2;
    private Image img=null;
    public ImageItemMIDlet(){
        display=Display.getDisplay(this);
        exitCommand=new Command("退出",Command.SCREEN,1);
        myform=new Form("ImageItem 演示");
        try{
            img=Image.createImage("/pic.png");
        }catch(Exception e){
            myform.append("加载图像文件失败");
        }
        myImageItem1=new ImageItem("图片 1",img,Item.LAYOUT_LEFT,"图片 1
          替代文本");
        myImageItem2=new ImageItem("图片 2",img,Item.LAYOUT_RIGHT|
        Item.LAYOUT_BOTTOM,"图片 2 替代文本");
        myform.append(myImageItem1);
        myform.append(myImageItem2);
        myform.addCommand(exitCommand);
        myform.setCommandListener(this);
    }
    public void startApp(){
        display.setCurrent(myform);
    }
    public void pauseApp(){
    }
    public void destroyApp(boolean uncondition){
    }
    public void commandAction(Command c,Displayable d){
```

```
            if(c==exitCommand){
                destroyApp(false);
                notifyDestroyed();
            }
        }
    }
```

程序运行结果如图 3-13 所示。根据布局，图片 1 处于左上角，图片 2 是水平右对齐和垂直底对齐。


图 3-13　ImageItem 演示结果

### 3.5.5　Spacer 类

Spacer(空间填充)类是一个比较特殊的控件，它并不可见。通常情况下，它起辅助定位的作用。一旦指定了 Spacer 的宽度和高度，就可以用该 Spacer 作为 Form 内部控件间的空隙(或者称为填充物)。Spacer 类的方法有以下两个。

(1)Spacer(int width,int height)：用于构造一个 Spacer 实例，其中 width 表示 Spacer 的宽度，height 表示 Spacer 的高度。

(2)setMinimumSize(int width,int height)：用于设置 Spacer 实例的宽度和高度。

注意：Spacer 不能使用 addCommand、setDefaultCommand 和 setLabel 方法，因此 Spacer 不具有标题，也不能与用户交互。

【例 3-12】　Spacer 类的使用。

首先构造 Form 实例，并添加 3 个 StringItem 控件，顺序放入 Form 容器中，前两个 StringItem 水平并列，选择菜单中的“插入 Spacer”命令，监听器调用对应的 commandAction 方法，执行下面的操作。

```
myform.insert(1,new Spacer(12,12));
myform.insert(2,new Spacer(60,40));
```

这两条语句表示向 Form 中的第二个 Item 和第三个 Item 前分别插入宽为 12、高为 12 和宽为 60、高为 40 的 Spacer 实例，可见 Spacer 的填充效果。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class SpacerMIDlet extends MIDlet implements CommandListener{
     private Display display;
     private Command exitCommand;
     private Command insertSpacerCommand;
     private Form myform;
     public SpacerMIDlet(){
          display=Display.getDisplay(this);
          exitCommand=new Command("退出",Command.SCREEN,1);
          insertSpacerCommand=new Command("插入 Spacer",Command.SCREEN,1);
          myform=new Form("Spacer 填充演示");
          StringItem myStringItem1 = new StringItem("控件 1","控件内容 1",
            Item.PLAIN);
          StringItem myStringItem2=new StringItem("","控件内容 2",
            Item.PLAIN);
          StringItem myStringItem3 = new StringItem("控件 3","控件内容 3",
            Item.PLAIN);
          myform.append(myStringItem1);
          myform.append(myStringItem2);
          myform.append(myStringItem3);
          myform.addCommand(insertSpacerCommand);
          myform.addCommand(exitCommand);
          myform.setCommandListener(this);
     }
     public void startApp(){
          display.setCurrent(myform);
     }
     public void pauseApp(){
     }
     public void destroyApp(boolean uncondition){
     }
     public void commandAction(Command c,Displayable d){
          if(c==insertSpacerCommand){
               myform.insert(1,new Spacer(12,12));
               myform.insert(2,new Spacer(60,40));
          }
          else if(c==exitCommand){
```

```
                destroyApp(false);
                notifyDestroyed();
            }
        }
    }
```

程序运行结果如图 3-14 所示。


(a)选择“插入Spacer”命令　　(b)插入Spacer后的效果

图 3-14　Spacer 填充效果

### 3.5.6　ChoiceGroup 类

ChoiceGroup(选择组)类和 List 类一样，都实现了 Choice 接口，它们具有很大的相似性，但主要有以下 3 点不同之处。

- List 是一个容器，而 ChoiceGroup 是一个控件，必须放置在 Form 容器中。
- ChoiceGroup 不能使用 Choice.IMPLICT 类型的 Choice。
- ChoiceGroup 具有一种特殊类型的 Choice 接口，即 Choice.POPUP，它是在 MIDP 2.0 中引入的。

ChoiceGroup 类的方法大部分都与 List 类类似，只是在构造方法上有所不同，构造 ChoiceGroup 的语法格式如下。

```
public static ChoiceGroup(String title,int choiceType);
public static ChoiceGroup(String title,int choiceType,String[] strings,Image
  [] imgs);
```

其中，title 指定了 ChoiceGroup 的标题；choiceType 指定了 ChoiceGroup 的选项类型，只能是单选类型 EXCLUSIVE、多选类型 MULTIPLE 和弹出式类型 POPUP 三者之一；strings 指定了各选项的文本内容；imgs 指定了各选项的图像内容。

ChoiceGroup 构造完毕后，还可以随时添加、编辑和删除其中的选项。较长文本的显示仍然有截断显示和换行显示两种方式。可以使用 getFont 和 setFont 方法获得和设置字体。

在前面介绍 Item 类时，讲到了当 Item 状态发生改变时，会触发 ItemStateChanged 事件，具体到 ChoiceGroup 控件，当其中选项的选中状态发生变化时，触发该事件，如果设置了

ItemStateListener，则捕捉到该事件并执行对应的操作。

【例 3-13】　改变 ChoiceGroup 的类型和状态。

本例中先建立一个 Form 实例，然后向这个 Form 实例中添加 3 个 ChoiceGroup 实例，类型分别为 EXCLUSIVE、MULTIPLE 和 POPUP，用于接收用户对性别、学校和婚姻状况的选择，分别调用 append 方法为这 3 个 ChoiceGroup 实例的每个选项添加文字和图片。再向 Form 实例中添加一个 StringItem 控件，用于现实用户的输入。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ChoiceGroupMIDlet extends MIDlet implements CommandListener,
ItemStateListener{
    private Display display;
    private Command exitCommand;
    private Form myform;
    private ChoiceGroup myExclusiveChoiceGroup;
    private ChoiceGroup myMultipleChoiceGroup;
    private ChoiceGroup myPopupChoiceGroup;
    private StringItem myChoice;
    private Image img=null;
    public ChoiceGroupMIDlet(){
        display=Display.getDisplay(this);
        exitCommand=new Command("退出",Command.SCREEN,1);
        myform=new Form("ChoiceGroup 演示");
        try{
            img=Image.createImage("/pic.png");
        }catch(Exception e){myform.append("加载图像文件失败");}
        myExclusiveChoiceGroup=new ChoiceGroup("选择性别",
          ChoiceGroup.EXCLUSIVE);
        myMultipleChoiceGroup=new ChoiceGroup("选择学校",ChoiceGroup.MULTIPLE);
        myPopupChoiceGroup=new ChoiceGroup("选择婚姻状况",ChoiceGroup.POPUP);
        myExclusiveChoiceGroup.append("男",img);
        myExclusiveChoiceGroup.append("女",img);
        myMultipleChoiceGroup.append("南开大学",img);
        myMultipleChoiceGroup.append("天津大学",img);
        myMultipleChoiceGroup.append ("清华大学",img);
        myMultipleChoiceGroup.append ("北京大学",img);
        myPopupChoiceGroup.append("未婚",img);
        myPopupChoiceGroup.append("已婚",img);
        myChoice=new StringItem("您选择的内容是:","");
        myform.append(myExclusiveChoiceGroup);
```

```
        myform.append(myMultipleChoiceGroup);
        myform.append(myPopupChoiceGroup);
        myform.append(myChoice);
        myform.addCommand(exitCommand);
        myform.setItemStateListener(this);
        myform.setCommandListener(this);
        myform.setItemStateListener(this);
    }
    public void startApp(){
        display.setCurrent(myform);
    }
    public void pauseApp(){
    }
    public void destroyApp(boolean uncondition){
    }
    public void commandAction(Command c,Displayable d){
        if(c==exitCommand){
            destroyApp(false);
            notifyDestroyed();
        }
    }
    public void itemStateChanged(Item item){
        if(item==myExclusiveChoiceGroup){
            int selected=myExclusiveChoiceGroup.getSelectedIndex();
            String temp=myExclusiveChoiceGroup.getString(selected);
            myChoice.setText(temp);
        }
        if(item==myMultipleChoiceGroup){
            boolean selected;
            String temp="";
            for(int i=0;i<myMultipleChoiceGroup.size();i++){
                selected=myMultipleChoiceGroup.isSelected(i);
                if(selected){
                    temp+=myMultipleChoiceGroup.getString(i);
                }
            }
            myChoice.setText(temp);
        }
        if(item==myPopupChoiceGroup){
```

```
                int selected=myPopupChoiceGroup.getSelectedIndex();
                String temp=myPopupChoiceGroup.getString(selected);
                myChoice.setText(temp);
            }
        }
    }
```

程序运行结果如图 3-15 所示。


(a)初始运行效果


(b)选中“女”单选按钮


(c)选择多选类型


(d)选择弹出式类型

图 3-15　ChoiceGroup 演示

在本例中，为了响应用户的选择，需要建立与 ItemStateListener 的关联，声明 ChoiceGroupMIDlet 类的代码如下。

```
public class ChoiceGroupMIDlet extends MIDlet implements CommandListener,
  ItemStateListener
```

它实现了两个监听器 CommandListener 和 ItemStateListener，即建立了关联。在类的定义中，定义了两个方法：commandAction 和 itemStateChanged，分别对应于两个监听器的操作，一旦触发了事件，监听器就自动执行相应的方法。

当用户在 ChoiceGroup 中选择选项造成选项状态的变化时，ItemStateListener 捕获到该变化并执行相应的操作，当用户选择某项时，需要将 StringItem 控件中的内容更新为用户的选择内容。于是在 itemStateChanged 方法中添加了更新 StringItem 的代码。具体代码如下。

```
if(item==myExclusiveChoiceGroup){
      int selected=myExclusiveChoiceGroup.getSelectedIndex();
      String temp=myExclusiveChoiceGroup.getString(selected);
      myChoice.setText(temp);
}
```

如果发生改变的 Item 是 myExclusiveChoiceGroup，则调用 getSelectedIndex 方法得到用户选中选项的索引号，并将其存储到 selected 中。再通过 getSelectedIndex 方法获得这个索引号下选项的文本内容，最后使用 StringItem 类的 setText 方法设置文本为用户选择的选项。具体代码如下。

```
if(item==myMultipleChoiceGroup){
      boolean selected;
      String temp="";
      for(int i=0;i<myMultipleChoiceGroup.size();i++){
           selected=myMultipleChoiceGroup.isSelected(i);
           if(selected){
                temp+=myMultipleChoiceGroup.getString(i);
           }
      }
      myChoice.setText(temp);
}
```

如果发生改变的 Item 是 myMultipleChoiceGroup，因为复选框中选中的选项有可能是多个，所以就不能使用 getSelectedIndex 方法，而使用 for 循环依次检查每一个选项是否被选中，这个检查是使用 isSelected 方法指定索引号，返回是否选中的布尔值，并存储到变量 selected 中，然后判断变量 selected 是否为 true，若是，则获得选中选项的文本，并追加到字符串 temp 中。最后将 temp 作为 StringItem 控件的内容显示在屏幕上。

如果发生改变的 Item 是 myPopupChoiceGroup，因为用户只能选择一个选项，所以处理方法同单选按钮一致。

### 3.5.7　TextField 类

TextField(文本区域)是一个可以放置在 Form 中的文本编辑控件,其构造方法的语法格式如下。

```
Public TextField(String title,String initText,int maxSize,int constraints);
```

其中,title 表示控件的标题;initText 表示控件的初始内容;maxSize 表示控件所能容纳的最大字符数;constraints 表示允许用户输入内容的限制。

TextField 类的功能类似于 TextBox,其很多方法也类似,如都具有标题、初始内容、最大的容量限制和对用户输入文本的限制等,但它们也有不同之处,主要表现在以下两方面。

- TextField 只支持单行文本的编辑,而 TextBox 则支持多行文本的编辑。
- TextField 与 Form 共同使用,而 TextBox 单独使用。

TextField 控件获得和设置输入类型限制的方法、最大文本容量的方法、编辑文本的方法与 TextBox 相同,但是需要注意以下几点。

- NUMERIC 用于限制用户输入整数值。
- PHONENUMBER 用于限制用户输入电话号码,该类型的 TextField 可以自动将输入的数字字符转化为电话号码的格式,一般来说都是“国家码＋地区码＋电话号码”,也有可能在电话号码前加“＋”前缀。
- DECIMAL 用于限制用户输入数值必须符合 Java 语言的语法定义,允许输入减号、小数分隔符和数字符号等。

【例 3-14】　TextField 控件及其限制的用法。

先创建容纳 TextField 控件的 Form 类的实例 myform,然后使用 append 方法添加各种输入限制类型的 TextField 控件,在各控件中输入文本,体会其对文本内容的限制。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TextFieldMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command exitCommand;
    private Form myform;
    public TextFieldMIDlet(){
        display=Display.getDisplay(this);
        exitCommand=new Command("退出",Command.SCREEN,1);
        myform=new Form("TextField 演示");
        myform.append(new TextField("各种文本","",20,TextField.ANY));
        myform.append(new TextField("电子邮箱","",20,TextField.EMAILADDR));
        myform.append(new TextField("数字字符","",20,TextField.NUMERIC));
        myform.append(new TextField("数　　值","",20,TextField.DECIMAL));
```

```
                myform.append(new TextField("电话号码","",20,
                  TextField.PHONENUMBER));
                myform.append(new TextField("密      码","",20,TextField.PASSWORD));
                myform.addCommand(exitCommand);
                myform.setCommandListener(this);
        }
        public void startApp(){
              display.setCurrent(myform);
        }
        public void pauseApp(){
        }
        public void destroyApp(boolean uncondition){
        }
        public void commandAction(Command c,Displayable d){
              if(c==exitCommand){
                    destroyApp(false);
                    notifyDestroyed();
              }
        }
}
```

程序运行结果如图 3-16 所示。


图 3-16　TextField 控件演示

### 3.5.8　DateField 类

DateField(处理日期和时间)类是专门用于处理日期和时间的类,目的是方便用户对日

期和时间的输入。它可以嵌入 Form 控件容器中，共有 3 种模式。

- DATE：日期，只允许输入年、月、日的信息。
- TIME：时间，只允许输入小时、分钟、秒的信息。
- DATE_TIME：日期时间，允许同时输入日期和时间信息。

DateField 类的主要方法如表 3-14 所示。

表 3-14　DateField 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| DateField | 无 | String，int | 构造方法，第一个参数指定控件标题，第二个参数指定模式 |
| DateField | 无 | String，int，TimeZone | 构造方法，第一个参数指定了控件标题，第二个参数指定模式，第三个参数指定当前时区 |
| getDate | Date | void | 获得当前控件中的日期时间 |
| setDate | void | Date | 设置当前控件中的日期时间 |
| getInoutMode | int | void | 获得当前控件的模式 |
| setInputMode | void | int | 设置当前控件的模式 |

【例 3-15】　3 种模式下 DateField 类的使用方法。

先建立一个 Form 容器对象 myform，然后使用 DateField 类的构造方法，分别建立 3 个不同模式的 DateField 控件，然后使用 myform 的 append 方法将 3 个 DateField 控件添加到 myform 中。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DateFieldMIDlet extends MIDlet implements CommandListener{
    private Display display;
    private Command exitCommand;
    private Form myform;
    private DateField datefield;
    private DateField timefield;
    private DateField datetimefield;
    public DateFieldMIDlet(){
        display=Display.getDisplay(this);
        exitCommand=new Command("退出", Command.SCREEN,1);
        myform=new Form("DateField 演示");
        datefield=new DateField("处于日期模式下的 DateField", DateField.DATE);
        timefield=new DateField("处于时间模式下的 DateField", DateField.TIME);
        datetimefield=new DateField("处于日期时间模式下的 DateField",
```

```
            DateField.DATE_TIME);
            myform.append(datefield);
            myform.append(timefield);
            myform.append(datetimefield);
            myform.addCommand(exitCommand);
            myform.setCommandListener(this);
        }
        public void startApp(){
            display.setCurrent(myform);
        }
        public void pauseApp(){
        }
        public void destroyApp(boolean uncondition){
        }
        public void commandAction(Command c,Displayable d){
            if(c==exitCommand){
                destroyApp(false);
                notifyDestroyed();
            }
        }
    }
```

程序的运行结果如图 3-17 所示。按模拟器上的方向键，可以选择编辑哪个模式的 DateField 控件。选择日期模式的控件，得到图 3-17(b)所示的界面，按方向键选择年、月和日。按下右软键进行保存；选择时间模式的控件，得到图 3-17(c)所示的界面，按方向键选择小时、分钟和秒；选择日期时间模式的控件，分别设置时间和日期，保存得到图 3-17(d)所示的界面，此时日期时间编辑完毕。


(a)初始运行效果


(b)日期模式下的显示效果


(c)时间模式下的显示效果


(d)日期时间模式下的显示效果

图 3-17　DateField 实例演示

### 3.5.9　Gauge 类

Gauge(进度条)类用图形的方式指示当前过程的进度,其实例可以放置在 Form 容器中。每个 Gauge 实例都具有一个取值范围(最小值为 0,最大值由程序在控件初始化时指定)和一个当前值(在取值范围内,且为整数),这个当前值指示了在整个取值范围的位置。

#### 1. Gauge 类的模式

Gauge 类有两种模式:交互式和非交互式。交互式的 Gauge 是一条从左到右上升的弧线,用户可以更改当前值;非交互式的 Gauge 从左到右是水平的,用户不能更改当前值。但无论是否允许更改,当前值必须在规定的取值范围内。

交互式模式一般用于指示当前的状态,它允许用户进行一定程度上的修改,如当前声音的音量可以用 Gauge 控件表示,最小音量就是 Gauge 的最小值,最大音量是 Gauge 的最大值,当前值是当前的音量值,如果用户想调整音量,还可以通过设置 Gauge 的当前值通知系统调整音量。

非交互式模式用于指示某一过程的进度,只起到提示的作用而不允许用户进行修改。例如,需要执行一个运行时间较长的程序,Gauge 控件可以给用户一个反馈信息,指示当前程序完成的百分比,防止用户以为程序死锁。

使用非交互式 Gauge 的注意事项如下。

- 构造方法的第二个参数应为 false。
- 不能使用 addCommand 方法添加命令。
- 不能连接 ItemCommandListener。
- 不能设置标题。
- 不能设置布局类型。
- 不能设置偏好尺寸。

### 2. Gauge 类的方法

Gauge 类的主要方法如表 3-15 所示。

表 3-15　Gauge 类的主要方法

| 方法名称 | 返回值类型 | 参数类型 | 含　义 |
|---|---|---|---|
| Gauge | 无 | String,boolean,int,int | 构造方法,第一个参数指定控件标题;第二个参数指定 Gauge 类型,true 表示交互式,false 表示非交互式;第三个参数表示最大值;第四个参数表示初始值 |
| addCommand | void | Command | 向当前 Gauge 控件添加命令 |
| setDefaultCommand | void | Command | 设置当前 Gauge 控件的默认命令 |
| setItemCommandListener | void | ItemCommandListener | 为当前 Gauge 控件添加 Item 命令监听器 |
| getValue | int | void | 获得当前 Gauge 控件的当前值 |
| setValue | void | int | 设置当前 Gauge 控件的当前值为参数指定的整数值 |
| getMaxValue | int | void | 获得当前 Gauge 控件的当前最大值 |
| setMaxValue | void | int | 设置当前 Gauge 控件的当前值为参数指定的整数值 |
| setLabel | void | String | 设置当前 Gauge 控件的标题为参数指定的字符串 |
| setLayout | void | int | 设置当前 Gauge 控件的标题为参数指定的布局 |
| setPreferredSize | void | int,int | 设置当前 Gauge 控件的偏好尺寸,第一个参数指定宽度,第二个参数指定高度 |
| isInteractive | boolean | void | 获得当前 Gauge 控件是否为交互式,返回 true 表示是交互式,返回 false 表示是非交互式 |

【例 3-16】　交互式 Gauge 的使用。

先建立 Form 容器,使用如下构造方法。

```
Gauge gauge=new Gauge("显示进度条",true,6,3);
```

创建交互式 Gauge 控件,其中 true 指示 Gauge 的类型是交互式,6 表示最大值,3 表示当前值。共有 6 个条形图,此时按左右方向键可以调整当前值。

具体代码如下。

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ShowGauge extends MIDlet implements CommandListener{
    private Display display;
    private Form props;
    private Command exitCommand=new Command("退出",Command.EXIT,1);
    public ShowGauge(){
        display=Display.getDisplay(this);
    }
    public void startApp(){
        props=new Form("进度条 Gauge 演示");
        Gauge gauge=new Gauge("显示进度条",true,6,3);
        props.append(gauge);
        props.addCommand(exitCommand);
        props.setCommandListener(this);
        display.setCurrent(props);
    }
    public void commandAction(Command c,Displayable s){
        if(c==exitCommand){
            destroyApp(false);
            notifyDestroyed();
        }
    }
    public void destroyApp(boolean unconditional){
    }
    public void pauseApp(){
        display.setCurrent(null);
        props=null;
    }
}
```

程序运行结果如图 3-18 所示。


图 3-18　交互式 Gauge 进度条演示

注意：用户的一次按键对 Gauge 控件当前值的作用是增加 1，当最大值是 6 时，每按一次键，Gauge 上的表示是移动一格，而当最大值是 12 时，由于图形上只有 6 个格，因此每按两次键，Gauge 才移动一格。

## 3.6 高级界面异常处理

前面介绍了高级界面类的定义和使用，这里将重点围绕高级界面类在使用中可能产生的异常进行说明。

### 3.6.1 Alert 类异常

在如下两种情况下，使用 Alert 类会产生 IllegalArgumentException 异常，使用 try…catch 可以捕获异常。

(1)在构造 Alert 类的实例时，使用构造方法 publicAlert(String title,String text,Image img,AlertType type)并指定图标文件 img 不是静态图像。

(2)调用 setTimeOut 方法时，使用了非正整数或 FOREVER 参数。

### 3.6.2 TextBox 类异常

在如下几种情况下，使用 TextBox 类会产生 IllegalArgumentException 异常，使用 try…catch 可以捕获异常。

(1)在构造 TextBox 类的实例时，指定的类型不是 EXCLUSIVE、MULTIPLE 和 IMPLICIT 之一。

(2)在构造 TextBox 类的实例时，指定的初始文本违反限制约束或超出了最大长度值。

(3)在调用 setString 方法时，指定的初始文本违反限制的约束或超出了最大长度值。

在任何情况下，如果控件中的文本超出了最大长度值，或者达到了环境所能允许的最大文本字符数，都会产生 IllegalArgumentException 异常。

### 3.6.3　List 类异常

在如下几种情况下，使用 List 类会产生 IllegalArgumentException 异常，使用 try…catch 可以捕获异常。

(1)在构造 List 类的实例时，指定的类型不是 EXCLUSIVE、MULTIPLE 和 IMPLICIT 之一。

(2)在构造 TextBox 类的实例时，指定的图像项的个数与构造方法规定图像项的个数不一致或者图像项中包含非静态图像。

除了 IllegalArgumentException 异常外，当指定的文本项数组为空或文本项之一为空时，产生 NullPointerException 异常；当调用 isSelected 方法，并指定了超过选项数的索引值时，将会产生 IndexOutofBoundsException 异常。

### 3.6.4　Item 类异常

在如下几种情况下，使用 Item 类会产生 IllegalArgumentException 异常，使用 try…catch 可以捕获异常。

(1)指定 Gauge 控件的最大值为非正整数时。

(2)在构造 TextField 类的实例时，指定了非正整数的最大长度值、非法限制类型值。

(3)在构造 TextField 类的实例时，指定的初始文本违反限制约束或超过了最大长度值。

(4)调用 TextField 类的 setString 方法时，指定的初始文本违反限制约束或超过了最大长度值。

(5)在构造 DateField 类时，指定了 DATE、TIME 和 DATE_TIME 三种模式以外的模式。

(6)在构造或设置 StringItem 类的外观时，指定了 PLAIN、HYPERLINK 和 BUTTON 三种类型以外的外观。

(7)在构造或设置 ImageItem 类的布局和外观时，指定了规定类型以外的布局和外观。

(8)在构造 ChoiceGroup 类的实例时，指定的类型不是 EXCLUSIVE、MULTIPLE 和 POPUP 之一。

(9)在构造 ChoiceGroup 类的实例时，指定的图像项的个数与文本项的个数不一致或者图像项中包含非静态图像。

除了 IllegalArgumentException 异常外，当指定 ChoiceGroup 类的实例的文本项数组为空或文本项之一为空时，也会产生 NullPointerException 异常；当调用 ChoiceGroup 类的isSelected方法，并指定了超过项数的索引值时，会产生 IndexOutofBoundsException 异常。

# 3.7 习　　题

## 一、选择题

1. Java ME 用户界面类分为高级用户界面类和低级用户界面类，以下描述正确的是(　　)。
A. Screen 类属于低级用户界面类
B. Canvas 类属于高级用户界面类
C. Screen 类和 Canvas 都属于高级用户界面类
D. Displayable 类是 Screen 类和 Canvas 类的父类
2. 以下 Command 类的命令中，优先级最高的是(　　)。
A. Command. BACK　　B. Command. ITEM
C. Command. HELP　　D. Command. STOP
3. Java ME 中所有高级用户界面类的父类是(　　)。
A. Canvas 类　　B. Form 类　　C. Screen 类　　D. Graphics 类
4. Command 类的事件监听器接口是(　　)。
A. CommandListener　　B. ItemStateCommandListener
C. commandAction　　D. itemStateChange
5. Alert 类中，表示警告框的类型是提示用户对某个操作进行确认的属性是(　　)。
A. AlertType. ALARM　　B. AlertType. INFO
C. AlertType. ERROR　　D. AlertType. CONFIRMATION
6. 在 List 类中，获取选中的某个单选项的索引号使用的方法是(　　)。(参数略)
A. getSelectedIndex()　　B. getSelectedFlags()
C. setSelectedCommand()　　D. removeCommand()
7. 在 TextBox 类的方法中，设置文本框允许输入的最大字符数的方法是(　　)。(参数略)
A. getMaxSize()　　B. setMaxSize()
C. getString()　　D. setString()
8. 设置文本框中仅允许输入电话号码的属性是(　　)。
A. TextField. PASSWORD　　B. TextField. DECIMAL
C. TextField. NUMERIC　　D. TextField. PHONENUMBER
9. 设置 Form 对象 myForm 的标题为“窗体测试”的语句是(　　)。
A. myForm＝new Form("窗体测试")
B. myForm＝new Command("窗体测试")
C. myForm＝new TextBox("窗体测试")
D. myForm＝new List("窗体测试")
10. Item 类或其子类对象的状态发生改变时，会触发(　　)事件。
A. commandAction　　B. itemStateChanged
C. ItemStateCommandListener　　D. CommandListener

11. 获取进度条 myGauge 的当前进度值的方法是(　　)。

A. setValue()　　B. getValue()

C. setMaxValue()　　D. getMaxValue()

## 二、实训题

1. 利用 Alert 类和 Ticker 类制作图片闪烁，同时在屏幕上显示一行滚动的文字，图片在屏幕上显示 3 秒后消失的效果。运行效果如图 3-19 所示。

2. 利用 List 类制作一个文字菜单，菜单项的内容为开始游戏、游戏帮助、关于游戏、音乐开关和退出游戏，并且每个菜单项的前面有一个小图片。菜单显示效果如图 3-20 所示。


图 3-19　图片闪烁和滚动文字效果


图 3-20　文字菜单显示效果

3. 结合本模块所学知识，设计一个 MIDlet，其运行效果如图 3-21 所示。(提示：新建工程中的 MIDlet 时，注意选择 Displayable 类的类型为 javax. microedition. lcdui. Form)


图 3-21　MIDlet 效果