第 1 部分：介绍 MIDP 的高层 UI

一、高层 MIDP 事件

高层 MIDP 事件分为两类： Command 和 Item 事件。最简单地说，Command 事件是由设备上的键击触发的，而 Item 事件则是显示中的可视化组件改变的结果。在本节中将介绍如何处理这两种事件类型。

Command 对象

在移动设备中发生一个事件时，就会有一个 Command 对象携带有关这个事件的信息。这些信息包含所执行的命令的类型、命令的标签以及其优先级。在 J2ME 中，命令通常是由设备上的软按钮（soft-button）表示的。

如果在屏幕上要显示很多命令，那么设备就会创建一个菜单以包含多重命令。

用 Command 对象进行事件处理

可以管理命令的 MIDP 组件只有 Form、TextBox、List 和 Canvas。

用 Command 对象处理事件的基本步骤如下所示：

1. 创建一个 Command 对象。
2. 将这个 Command 添加到 Form、TextBox、List 或者 Canvas 上。
3. 创建一个监听器。

检测到事件后，监听器会产生对 commandAction() 方法的调用。用这个方法可以确定是哪一条命令生成了这个事件并对它进行相应处理，显示在下一面板中。

下面一段代码显示了用一个 Command 对象处理一个事件的过程。

```
private Form fmMain;              // Form
private Command cmExit;           // Command to exit the MIDlet
...
fmMain = new Form("Core J2ME");   // Create Form and give it a title
// Create Command object, with label, type and priority
cmExit = new Command("Exit", Command.EXIT, 1);
...
fmMain.addCommand(cmExit);        // Add Command to Form
fmMain.setCommandListener(this);  // Listen for Form events
...
public void commandAction(Command c, Displayable s)
{
  if (c == cmExit)
  {
     destroyApp(true);
     notifyDestroyed();
  }
}
```

Item 对象

还可以用 Item 对象处理事件。在 MIDP 中预定义了几个 Item 用于处理特定事件类型。例如，DateField item 让用户可以选择在屏幕上显示的日期和时间，而 TextField item 让用户可以输入一组字符数字和特殊字符。

用 Item 对象处理事件

除了 StringItem、Spacer 和 ImageItem，每一个 Item 都可以识别事件。像 Command 对象一样，在可以识别事件之前必须首先创建一个监听器。当给定 item 上发生了改变时 —— 例如更新了 TextField 组件中的文字 —— 就生成一个事件。

在任何 Item 组件中发生改变后，就调用 itemStateChanged() 方法。在这个方法中您可以确定是哪一个 Item 生成了事件。

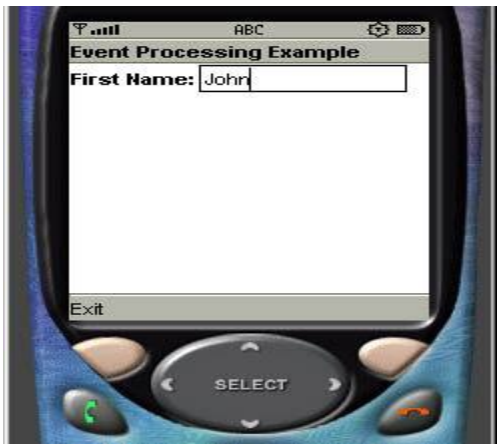
不同的设备实现处理 Item 事件的方式可能会有所不同。MIDP 没有指定必须识别哪一个 Item 事件，所以不是一个 Item 上的每一个事件都会自动调用 itemStateChange() 。例如，如果更新了 DateField 组件中的年份，在检测到这种改变时也许不会调用 itemStateChange() 方法。可能只有当用户转移到屏幕中其他组件时设备才会识别这个事件。

下面的一段代码显示了 DateFieldItem 对象的简单事件处理。

```
private Form fmMain;                  // Form
private DateField dfToday;            // DateField item
...
fmMain = new Form("Core J2ME");                      // Create Form object
dfToday = new DateField("Today:", DateField.DATE);  // Create DateField
...
fmMain.append(dfToday);              // Add DateField to Form
fmMain.setItemStateListener(this);  // Listen for Form events


...

public void itemStateChanged(Item item)
{
  // If the datefield initiated this event
  if (item == dfToday)
    ...
}
```

我们的第一个 MIDlet 将同时展示 Command 和 Item 事件处理。

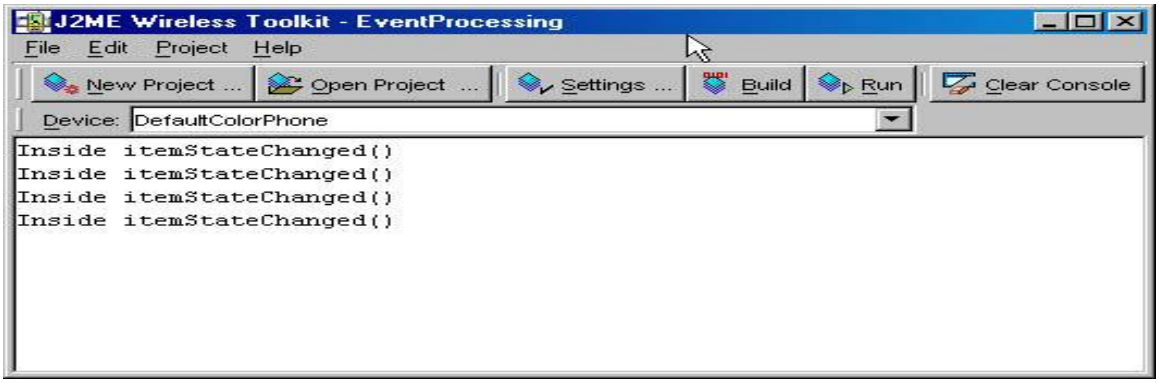
```
/*-------------------------Test-------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Test extends MIDlet implements
ItemStateListener, CommandListener
{
      private Display display;     // Reference to Display object for this MIDlet
      private Form fmMain;         // Main Form
      private Command cmExit;      // Command to exit the MIDlet
      private TextField tfText;    // TextField component (item)
      public Test()
      {
        display = Display.getDisplay(this);
        // Create the date and populate with current date
        tfText = new TextField("First Name:", "", 10, TextField.ANY);
        cmExit = new Command("Exit", Command.EXIT, 1);
        // Create the Form, add Command and DateField
        // listen for events from Command and DateField
        fmMain = new Form("Event Processing Example");
        fmMain.addCommand(cmExit);
        fmMain.append(tfText);
        fmMain.setCommandListener(this);      // Capture Command events (cmExit)
        fmMain.setItemStateListener(this);    // Capture Item events (dfDate)
      }
      // Called by application manager to start the MIDlet.
      public void startApp()
      {
        display.setCurrent(fmMain);
      }
      public void pauseApp()
      { }
      public void destroyApp(boolean unconditional)
      { }
      public void commandAction(Command c, Displayable s)
      {
        System.out.println("Inside commandAction()");
        if (c == cmExit)
        {
          destroyApp(false);
          notifyDestroyed();
        }
      }
      public void itemStateChanged(Item item)
      {
        System.out.println("Inside itemStateChanged()");
      }
}

```



效果如下：

输入每一个字符时，就会生成一个 Item 事件。回过头来看 itemStateChanged() 方法中的代码，对每一个事件我们在 WTK 控制台中打印一个消息，如下所示：

```
public void itemStateChanged(Item item)
{
  System.out.println("Inside itemStateChanged()");
}
```



二、Display、Displayable 和 Screen 对象—— 它们共同构成了 MIDP 的设备显示机制。

Display 对象

MIDlet 有 Display 对象的一个实例。这个对象用于获得有关当前显示的信息

- 如可用的颜色支持
- 并包括请求被显示的对象（即 Form 和 Textbox）的方法。

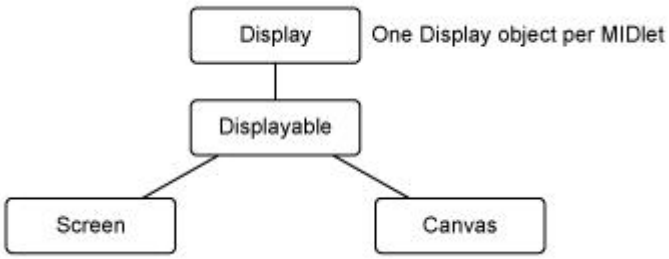
Display 对象实质上是设备显示屏的管理程序，它控制在设备上显示什么内容。

Displayable 对象

尽管每个 MIDlet 只有一个 Display 对象，但是 MIDlet 中的许多对象可能是 *可显示的（displayable）* —— 它们是 Form、TextBox、ChoiceGroup 等。

一个 Displayable 对象是在设备上**可见的组件**。MIDP 包含 Displayable 的两个子类：Screen 和 Canvas。下面是每个子类的类定义：

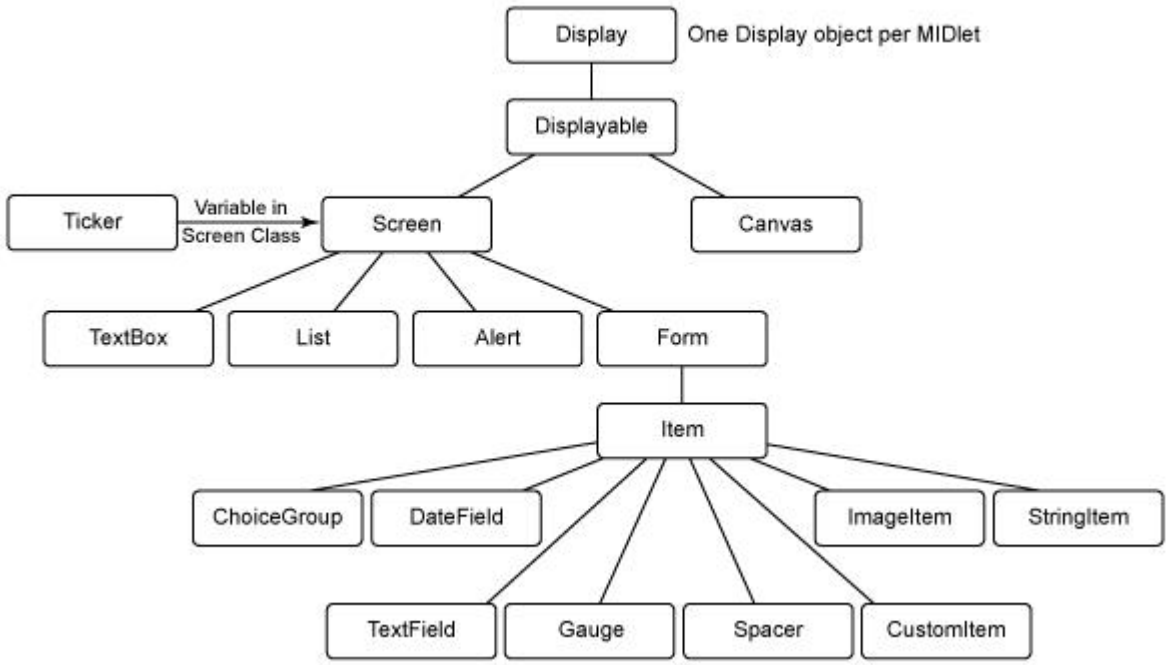
```
abstract public class Displayable
public abstract class Canvas extends Displayable
public abstract class Screen extends Displayable
```



Screen 对象

Screen 对象在设备上是不可见的。相反，由 Screen 派生出高层组件子类，最终用户是与所显示的这些子类进行交互的。

从 Screen 派生的子类



三、Form 和 Item 组件

Form 实质上是包含其他组件的容器，其中每一个组件都是 Item 类的一个子类。
我们将分析组成 MIDP 的高层 API 的每一个设备显示组件，它们是：

- DateField
- Gauge
- StringItem
- TextField
- ChoiceGroup
- Spacer
- CustomItem
- Image and ImageItem

DateField 组件

DateField 组件提供了可视化地操纵在 java.util.Date 中定义的 Date 对象的方法。在创建一个 DateField 对象时，指定用户可以编辑日期、时间，或者二者皆可编辑。

DateField 的构造函数如下：

```
DateField(String label, int mode)
DateField(String label, int mode, TimeZone timeZone)
```

下面是创建 DateField 对象并用当前日期和时间填充这个对象的部分代码清单。

```
private DateField dfAlarm;
// DateField with label, that allows both date and time to be changed
dfAlarm = new DateField("Set Alarm Time", DateField.DATE_TIME);
dfAlarm.setDate(new Date());
```

DateField 模式

通过指定适当的 *模式（mode）* 参数，还可以指定 DateField 组件为只能调整日期 *或者* 时间。下面是显示每一种可用 *模式（mode）* 选项的声明：

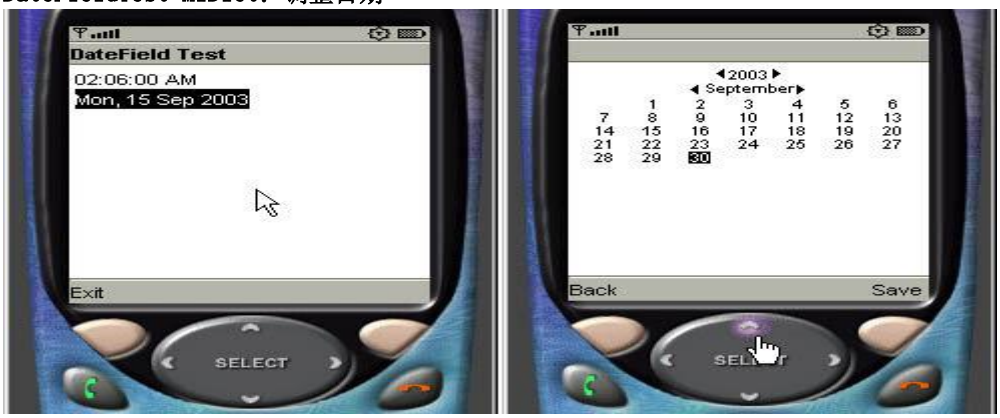
```
DateField("Set Alarm Time", DateField.DATE_TIME);
DateField("Set Alarm Time", DateField.TIME);
DateField("Set Alarm Time", DateField.DATE);
```

例子程序如下：

```
/*-------------------------------------------------------
* DateFieldTest.java
*-------------------------------------------------------*/
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Timer;
import java.util.TimerTask;
public class DateFieldTest extends MIDlet implements ItemStateListener, CommandListener
{
  private Display display;         // Reference to display object
  private Form fmMain;               // Main form
  private Command cmExit;            // Exit MIDlet
  private DateField dfAlarm;         // DateField component
  public DateFieldTest()
  {
    display = Display.getDisplay(this);
    // The main form
    fmMain = new Form("DateField Test");
    // DateField with todays date as a default
    dfAlarm = new DateField("Set Alarm Time", DateField.DATE_TIME);
    dfAlarm.setDate(new Date());
    // All the commands/buttons
    cmExit = new Command("Exit", Command.EXIT, 1);
    // Add to form and listen for events
    fmMain.append(dfAlarm);
    fmMain.addCommand(cmExit);
    fmMain.setCommandListener(this);
    fmMain.setItemStateListener(this);
  }
  public void startApp ()
  {
    display.setCurrent(fmMain);
  }
  public void pauseApp()   { }
  public void destroyApp(boolean unconditional) { }
  public void itemStateChanged(Item item)
  {
    System.out.println("Date field changed.");
  }
  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```

DateFieldTest MIDlet：调整时间

DateFieldTest MIDlet：调整日期



我的学习代码：


```
/*------------------------Test-------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TCI extends MIDlet implements
ItemStateListener, CommandListener
{
private Form fr;
private Command qc,hc,pc;
private Display display;     // Reference to Display object
for this MIDlet
private TextField text;
public TCI() {
     display = Display.getDisplay(this);

     fr = new Form("关于按钮测试");
     qc = new Command("Exit",Command.EXIT,1);
     hc = new Command("Help",Command.HELP,2);
     pc = new Command("Pause",Command.HELP,2);
     text= new TextField("姓名:","",10,TextField.ANY);
      fr.append(text);
     fr.addCommand(qc);
     fr.addCommand(hc);
     fr.addCommand(pc);
     fr.setCommandListener(this);
     fr.setItemStateListener(this);
     }
public void startApp() {
     display.setCurrent(fr);
     }
public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
```

```
public void commandAction(Command c,Displayable s) {
     if (c==qc){
          destroyApp(true);
          notifyDestroyed();

          }
     }
 public void itemStateChanged(Item item) {
        System.out.println("Inside itemStateChanged()");
        }

}
```

Gauge 组件

Gauge 组件显示一个进程度量风格的界面。有两种 Gauge：交互式 和 非交互式。前者使用户可以对 gauge 做出改变。后者要求开发者更新 gauge。

下面是 Gauge 组件的构造函数：

Gauge(String label, boolean interactive, int maxValue, int initialValue)

下面是创建交互式的 Gauge 的一小段代码：

```
private Gauge gaVolume;    // Volume adjustment
gaVolume = new Gauge("Sound Level", true, 100, 4);
```

实际应用：

```
/*-------------------------------------------------------
* InteractiveGauge.java
*-------------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class InteractiveGauge extends MIDlet implements CommandListener
{
  private Display display;    // Reference to display object
  private Form fmMain;        // The main form
  private Command cmExit;     // Exit the form
  private Gauge gaVolume;     // Volume adjustment

  public InteractiveGauge()
  {
    display = Display.getDisplay(this);

    // Create the gauge and exit command
    gaVolume = new Gauge("Sound Level", true, 50, 4);
    cmExit = new Command("Exit", Command.EXIT, 1);

    // Create form, add commands, listen for events
    fmMain = new Form("");
    fmMain.addCommand(cmExit);
    fmMain.append(gaVolume);
    fmMain.setCommandListener(this);
  }

  // Called by application manager to start the MIDlet.
  public void startApp()
  {
    display.setCurrent(fmMain);
  }

  public void pauseApp()
  { }

  public void destroyApp(boolean unconditional)
  { }

  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```

StringItem 组件

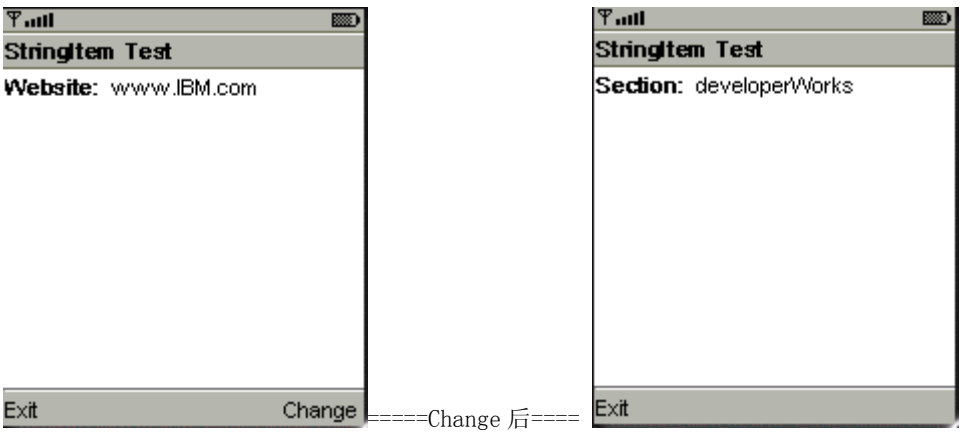
StringItem 组件用于显示标签和/或文本字符串。因为应用程序运行时，用户既不能改变标签也不能改变文字，所以 StringItem 不能识别事件。

StringItem 的构造函数如下所示：

StringItem(String label, String text)

实际应用：

```
/*-------------------------------------------------
* StringItemTest.java
*-------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class StringItemTest extends MIDlet implements CommandListener
{
  private Display display;       // Reference to Display object
  private Form fmMain;           // Main form
  private StringItem siMsg;      // StringItem
  private Command cmChange;      // Change the label and message
  private Command cmExit;        // Exit the MIDlet

  public StringItemTest()
  {
    display = Display.getDisplay(this);
    // Create text message and commands
    siMsg = new StringItem("Website: ", "www.IBM.com");
    cmChange = new Command("Change", Command.SCREEN, 1);
    cmExit = new Command("Exit", Command.EXIT, 1);
    // Create Form, add Command and StringItem, listen for events
    fmMain = new Form("StringItem Test");
    fmMain.addCommand(cmExit);
    fmMain.addCommand(cmChange);
    fmMain.append(siMsg);
    fmMain.setCommandListener(this);
  }
  // Called by application manager to start the MIDlet.
  public void startApp()
  {
    display.setCurrent(fmMain);
  }
  public void pauseApp()  { }
  public void destroyApp(boolean unconditional)  { }
  public void commandAction(Command c, Displayable s)
  {
    if (c == cmChange)
    {
      siMsg.setLabel("Section: "); // Change label
      siMsg.setText("developerWorks"); // Change text
      fmMain.removeCommand(cmChange);  // Remove the command
    }
    else if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```



TextField 组件

Textfield 模拟了所有典型的文本输入字段。可以指定标签、最大字符数以及所接受的数据类型。TextField 组件还实现了在输入时屏蔽字符的密码修改器。

下面是 TextField 的构造函数：

TextField(String label, String text, int maxSize, int constraints)

参数 *constraints* 是为 TextField 组件定义了以下约束：

- ANY 允许所有字符。
- EMAILADDR 只允许有效的电子邮件地址。
- NUMERIC 允许所有数字值。
- PHONENUMBER 只允许电话号码。
- URL 只允许在 URL 中有效的字符。
- PASSWORD 在输入时屏蔽所有字符。

现在让我们看一看在这个示例 TextField 中添加第二个约束 —— 密码修改器 —— 后会有什么现象。

下面是原来的 TextField：

```
tfText = new TextField("Phone:", "", 10, TextField.PHONENUMBER);
```

而下面是新的：

```
tfText = new TextField("Phone:", "", 10, TextField.PHONENUMBER | TextField.PASSWORD);
```

保存新的源代码，生成并运行这个 MIDlet。现在可以看到在输入每一个字符时，它是如何被屏蔽的，如图所示。

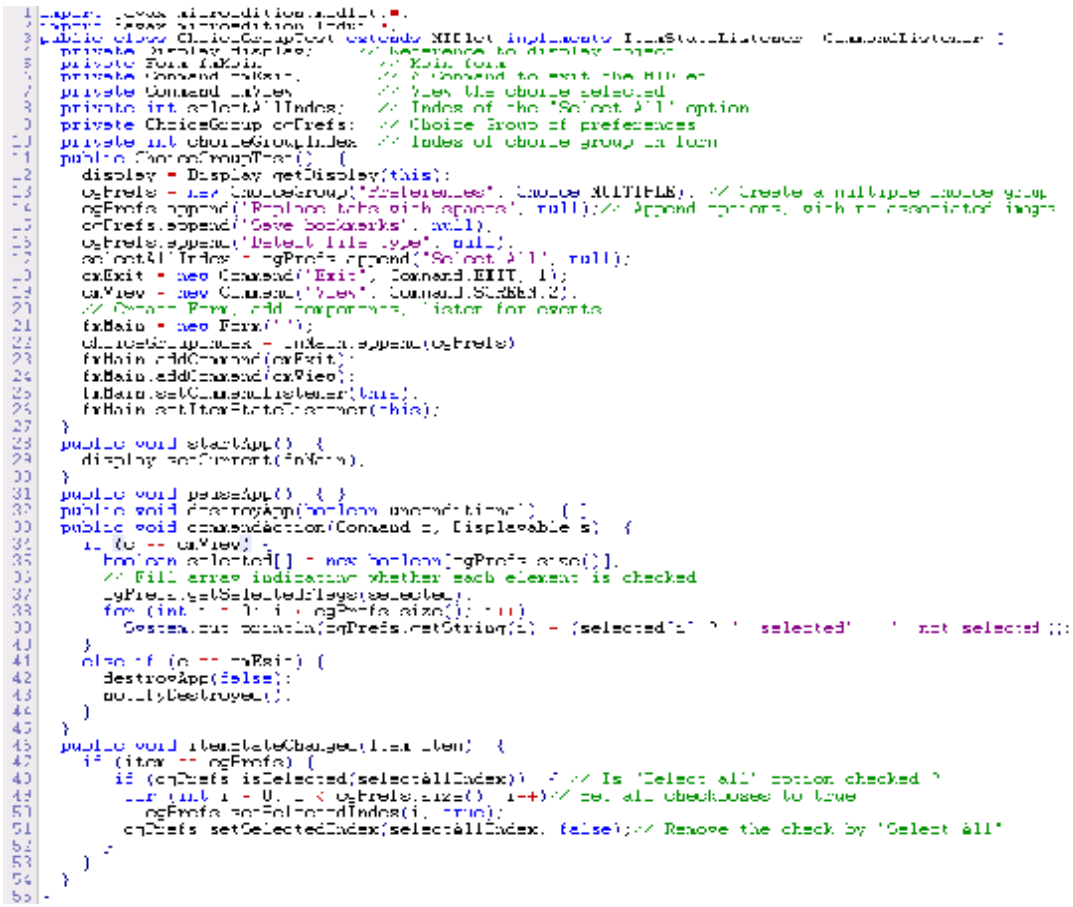
一个具有密码约束的 TextField 组件

ChoiceGroup 组件

ChoiceGroup 组件使用户可以从预先定义的一组条目中选择。有两种 ChoiceGroup 形式：

- *多选（multi-selection）* —— 它常用来指复选框
- *单选（exclusive-selection）* —— 它实质上就是单选按钮组

实际应用：



Spacer 组件

Spacer 是一个不可见的组件，它帮助显示中其他项目的定位。只要指定每一个 Spacer 的宽度和高度，就可以用它提供组件之间垂直和水平的间隔空间。因为 Spacer 是不可见的组件，所以它不会识别事件。

CustomItem 组件

CustomItem 组件使您可以创建自己的 Item 组件。像其他 Item 一样，这种组件可以添加到 Form 上并可以识别并处理事件。

CustomItem 是用 paint() 方法绘制到显示中的。作为这个组件的创建器，需要由您来编写 paint() 中的代码。创建 CustomItem 的过程与处理任何扩展其他类的其他 Java 平台对象没有区别。下面显示了一个简单 CustomItem 组件的外壳：

```
public class NewItem extends CustomItem
{
    public NewItem(String label)
    {
      super(label);
      ...
    }
```

```
    protected void paint(Graphics g, int width, int height)
    {
      ...
    }
    protected int getMinContentHeight()
    {
      ...;
    }
    protected int getMinContentWidth()
    {
      ...
    }
    protected int getPrefContentHeight(int width)
    {
      ...
    }
    protected int getPrefContentWidth(int height)
    {
      ...
    }
...

}
```

虽然 CustomItem 是 MIDP 高层接口中最令人兴奋的一个，但是创建它就要了解更多有关 CustomItem 组件的内容，请参阅 参考资料。

ImageItem 和 Image 组件

有两个类用于处理显示图像：Image 和 ImageItem。Image 用于创建一个图像对象并包含像高度和宽度、以及图像是否可变这样的信息。ImageItem 定义了如何显示一个图像，即图像是否对中、靠左、在屏幕上方等。

MIDP 提供了两种图像：不可变的和可变的。一个 *不可变* 图像在创建后不能再改变。通常，这种图像是从像文件这样的资源中读取的。一个 *可变* 图像实质上是一块内存。通过将图像内容编写进这个内存块而创建它。在下面几页中我们将讨论不可变图像。在第 2 部分讨论低层界面时将介绍可变图像。

Image 和 ImageItem 类的构造函数

下面是 Image 和 ImageItem 类的构造函数：

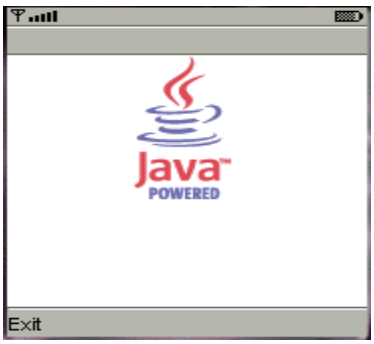
```
Image createImage(String name)
Image createImage(Image source)
Image createImage(byte[] imageDate, int imageOffset, int imageLength)
Image createImage(int width, int height)
Image createImage(Image image, int x, int y, int width,int height, int transform)
Image createImage(InputStream stream)
Image createRGBImage(int[] rgb, int width, int height,boolean processAlpha)
ImageItem(String label, Image img, int layout, String altText)
```

显示一个图像的步骤

下面的代码显示了如何从文件中创建一个图像、将它与一个 ImageItem 对象相关联、并将图像附加到 Form 上。

```
Form fmMain = new Form("Images");
...
// Create an image
Image img = Image.createImage("/house.png");
// Append to a form
fmMain.append(new ImageItem(null, img, ImageItem.LAYOUT_CENTER, null));
```

```java
/*-----------------------------------------------------
* ImageTest.java
*-----------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ImageTest extends MIDlet  implements CommandListener
{
  private Display display;      // Reference to Display object
  private Form fmMain;          // The main form
  private Command cmExit;       // Command to exit the MIDlet

  public ImageTest()
  {
    display = Display.getDisplay(this);
    cmExit = new Command("Exit", Command.EXIT, 1);
    fmMain = new Form("");
    fmMain.addCommand(cmExit);
    fmMain.setCommandListener(this);
    try
    {
      // Read the appropriate image based on color support
      Image im = Image.createImage((display.isColor()) ? "/2.png":"/1.png");
      fmMain.append(new ImageItem(null, im, ImageItem.LAYOUT_CENTER, null));
      display.setCurrent(fmMain);
    } catch (java.io.IOException e)
    {
      System.err.println("Unable to locate or read .png file");
    }
  }

  public void startApp()
  {
    display.setCurrent(fmMain);
  }

  public void pauseApp()  {  }
  public void destroyApp(boolean unconditional)  {  }

  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```



四、List、TextBox、Alert 和 Ticker 组件

MIDP 高层界面的 List、TextBox、Alert 和 Ticker 组件，与提供垂直滚动以容纳所有显示的组件的 Form 组件不同，List、TextBox 或者 Alert 对象本身占满了整个显示屏。Ticker 与其他组件有一些不同，它不是 Screen 的子类。相反，它是在 Screen 类中定义的一个变量。因此，所有从 Screen 派生出的对象都可以显示一个 Ticker。

List 组件

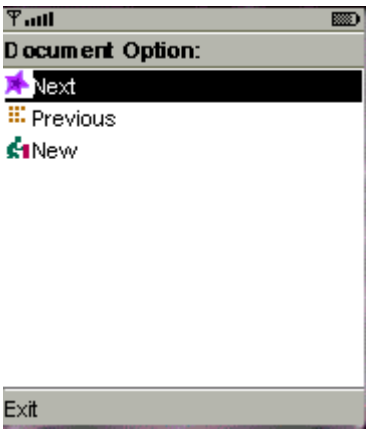
List 包含以三种形式之一展示的一组选择。在前面讨论 ChoiceGroup 时我们已经看到了多选和单选形式，第三种可用的形式是隐式。*隐式列表* 通常用于表示选择菜单。

实际应用：

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ImplicitList extends MIDlet implements CommandListener
{
  private Display display;     // Reference to Display object
  private List lsDocument;      // Main list
  private Command cmExit;       // Command to exit
  public ImplicitList()  {
    display = Display.getDisplay(this);
    // Create the Commands
    cmExit = new Command("Exit", Command.EXIT, 1);
    try {
      // Create array of image objects
      Image images[] = {Image.createImage("/1.png"),
                        Image.createImage("/2.png"),
                        Image.createImage("/3.png")};
      // Create array of corresponding string objects
      String options[] = {"Next", "Previous", "New"};
   // Create list using arrays, add commands, listen for events
    lsDocument = new List("Document Option:", List.IMPLICIT, options, images);
    // If you have no images, use this line to create the list
    //       lsDocument = new List("Document Option:", List.IMPLICIT, options, null);
    lsDocument.addCommand(cmExit);
    lsDocument.setCommandListener(this);
    } catch (java.io.IOException e) {
      System.err.println("Unable to locate or read .png file");
    }
  }
  public void startApp()  {
    display.setCurrent(lsDocument);
  }
  public void pauseApp()  {  }
  public void destroyApp(boolean unconditional)  {  }
  public void commandAction(Command c, Displayable s)
  {
    // If an implicit list generated the event
    if (c == List.SELECT_COMMAND)    {
      switch (lsDocument.getSelectedIndex())
      {
        case 0:
          System.out.println("Next selected");
          break;
        case 1:
          System.out.println("Previous selected");
          break;
        case 2:
          System.out.println("New selected");
          break;
      }
    }
    else if (c == cmExit)  {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```

在这个例子中，我们加入了几个图像，让它们与清单中的每一条目并排显示。为每一个字符串指派图像和标签的代码，如下所示：

```
// Create array of image objects
Image images[] = {Image.createImage("/1.png"),
                  Image.createImage("/2.png"),
                  Image.createImage("/3.png")};
// Create array of corresponding string objects
String options[] = {"Next", "Previous", "New"};
// Create list using arrays, add commands, listen for events
lsDocument = new List("Document Option:", List.IMPLICIT, options, images);
```



TextBox 组件

TextBox 组件用于允许多行输入。TextBox 和 TextField 组件有相同的指定允许的内容类型的约束

- ANY 允许所有字符。
- EMAILADDR 只允许有效的电子邮件地址。
- NUMERIC 允许所有数字值。
- PHONENUMBER 只允许电话号码。
- URL 只允许在 URL 中有效的字符。
- PASSWORD 在输入时屏蔽所有字符。

下面是 TextBox 的构造函数：

```
TextBox(String title, String text, int maxSize, int constraints)
```

如果需要改变用户可以在 TextBox 中输入的内容的类型，那么只需要改变约束参数。例如，如果添加了如下所示的声明，就将只接受数字值。

```
tbClip = new TextBox("Textbox Test", "Contents go here..", 125, TextField.NUMERIC);
```

Alert 和 AlertType 组件

Alert 实质上是一个非常简化的对话框。有两种类型的 Alert：*有模式（modal）* —— 在用户确认之前它一直显示对话框，和 *无模式（non-modal）* —— 它显示指定的秒数。

Alert 的构造函数如下所示：

```
Alert(String title)
Alert(String title, String alertText, Image alertImage, AlertType alertType)
```

AlertType 组件使用声音而不是视觉提示通知用户一个事件。例如，可以编写代码让 AlertType 发出特定的声音以通知用户出现了错误。

AlertType 组件有五种预定义的声音：alarm、confirmation、error、info 和 warning。

回头看 Alert 构造函数，注意 Alert 还可以包含对 AlertType 的引用。最终结果是先有一个表明 AlertType 的声音，然后是一个 Alert 对话框。

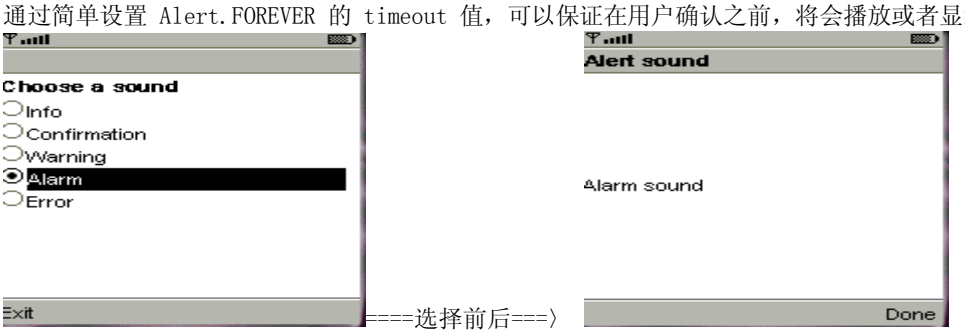
实际应用：

创建一个有模式 Alert

同样，下面的代码展示了如何同时实现 Alert 和 AlertType 组件。这段代码还展示了如何创建一个有模式（相对于无模式）对话框

```
al = new Alert("Alert sound", "Error sound", null, AlertType.INFO);
...
// Wait for user to acknowledge the alert
al.setTimeout(Alert.FOREVER);
```

通过简单设置 Alert.FOREVER 的 timeout 值，可以保证在用户确认之前，将会播放或者显示报警。



```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class AlertTest extends MIDlet implements ItemStateListener, CommandListener
{
  private Display display;     // Reference to display object
  private Form fmMain;         // Main form
  private Command cmExit;      // Command to exit the MIDlet
  private ChoiceGroup cgSound; // Choice group
  public AlertTest()
  {
    display = Display.getDisplay(this);
    // Create an exclusive (radio) choice group
    cgSound = new ChoiceGroup("Choose a sound", Choice.EXCLUSIVE);
    // Append options, with no associated images
    cgSound.append("Info", null);
    cgSound.append("Confirmation", null);
    cgSound.append("Warning", null);
    cgSound.append("Alarm", null);
    cgSound.append("Error", null);
    cmExit = new Command("Exit", Command.EXIT, 1);
    // Create Form, add components, listen for events
    fmMain = new Form("");
    fmMain.append(cgSound);
    fmMain.addCommand(cmExit);
    fmMain.setCommandListener(this);
    fmMain.setItemStateListener(this);
  }
  public void startApp()
  {
    display.setCurrent(fmMain);
  }
  public void pauseApp()  { }
  public void destroyApp(boolean unconditional)  { }
  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
      {
        destroyApp(false);
        notifyDestroyed();
      }
  }
  public void itemStateChanged(Item item)
  {
    Alert al = null;
    switch (cgSound.getSelectedIndex())
    {
      case 0:
        al = new Alert("Alert sound", "Info sound",null, AlertType.INFO);
        break;
      case 1:
        al = new Alert("Alert sound", "Confirmation sound", null, AlertType.INFO);
        break;
      case 2:
        al = new Alert("Alert sound", "Warning sound", null, AlertType.INFO);
        break;
      case 3:
        al = new Alert("Alert sound", "Alarm sound", null, AlertType.INFO);
        break;
      case 4:
        al = new Alert("Alert sound", "Error sound", null, AlertType.INFO);
        break;
    }
    if (al != null)    {
      // Wait for user to acknowledge the alert
      al.setTimeout(Alert.FOREVER);
      // Display alert, show main form when done
      display.setCurrent(al, fmMain);
    }
  }
}
```

Ticker 组件

Ticker 组件展示了水平滚动横幅。可以对 Ticker 指定的惟一参数是要显示的文本消息。滚动的速度和方向是由设备实现决定的。

Ticker 的构造函数如下：

Ticker(String str)

下面是 Ticker 的定义在 Screen 类中所示：

```
public abstract class Screen extends Displayable
{
   ...
   private Ticker ticker = null;
```

```
    ...
}
```

实际应用：

Ticker 在显示屏的上方滚动，同时 List 显示一系列产品

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TickerTest extends MIDlet implements CommandListener
{
  private Display display;    // Reference to Display object
  private List lsProducts;    // Products
  private Ticker tkSale;      // Ticker
  private Command cmExit;     // Command to exit the MIDlet
  public TickerTest()
  {
    display = Display.getDisplay(this);
    cmExit = new Command("Exit", Command.SCREEN, 1);
    tkSale = new Ticker("Sale: Real Imitation Cuban Cigars...10 for $10");
    lsProducts = new List("Products", Choice.IMPLICIT);
    lsProducts.append("Wicker Chair", null);
    lsProducts.append("Coffee Table", null);
    lsProducts.addCommand(cmExit);
    lsProducts.setCommandListener(this);
    lsProducts.setTicker(tkSale);
  }

  public void startApp()
  {
    display.setCurrent(lsProducts);
  }

  public void pauseApp()  {  }
  public void destroyApp(boolean unconditional)  {  }

  public void commandAction(Command c, Displayable s)
  {
    if (c == List.SELECT_COMMAND)    {
      switch (lsProducts.getSelectedIndex())
      {
        case 0:
          System.out.println("Chair selected");
          break;
        case 1:
          System.out.println("Table selected");
          break;
      }
    }
    else if (c == cmExit)    {
      destroyApp(true);
      notifyDestroyed();
    }
  }
}
```

结束语

您已完成了对 J2ME 和 MIDP 的共两部分的介绍的第一部分。在教程的这第一部分中，介绍了构成 MIDP 的高层 API 的用户界面组件，以及对这个 API 的两个事件处理技术。通过本教程，您还获得了在 J2ME 环境中创建和运行 MIDlet 的大量实际经验。

现在您已经掌握了足够的知识，可以开始创建自己具有简单用户界面能力的 MIDlet。在教程的第 2 部分，我们将学习更多的知识，介绍 MIDP 的低层用户界面组件。低层 API 组件比这里讨论的组件更复杂，但是它们也会为您提供对移动设备应用程序外观的更多控制。

风花雪月　　e 梦情缘

网络 ID: wnhoo 或 sos_admin

网名: e 梦缘

Mail: wnhoo@163.com