第 15 章

流程控制的设计模式

15.1 本章目的

在 LCDUI 的架构下，画面上同一时间只能有一个 Displayable 的子类实体。导致 MIDP 程序设计中，最令人头痛的地方莫过于程序的流程控制，也就是程序中画面之间的切换。本章将针对程序的流程控制，提出一个设计模式，让程序的流程控制能够更方便、更简单且更具可维护性。

本章将从分析设计讲起，慢慢地完成整个系统。

15.2 参考资源与书目

■ 书籍参考资源

Design Patterns 于 Java 语言上的实习应用


15.3　系统分析与设计

首先是系统分析与设计阶段。在此阶段中，我们根据使用案例和相关信息，定义了系统流程，系统流程决定了程序中画面的切换。本章范例所设计的系统流程如图 15-1 所示：

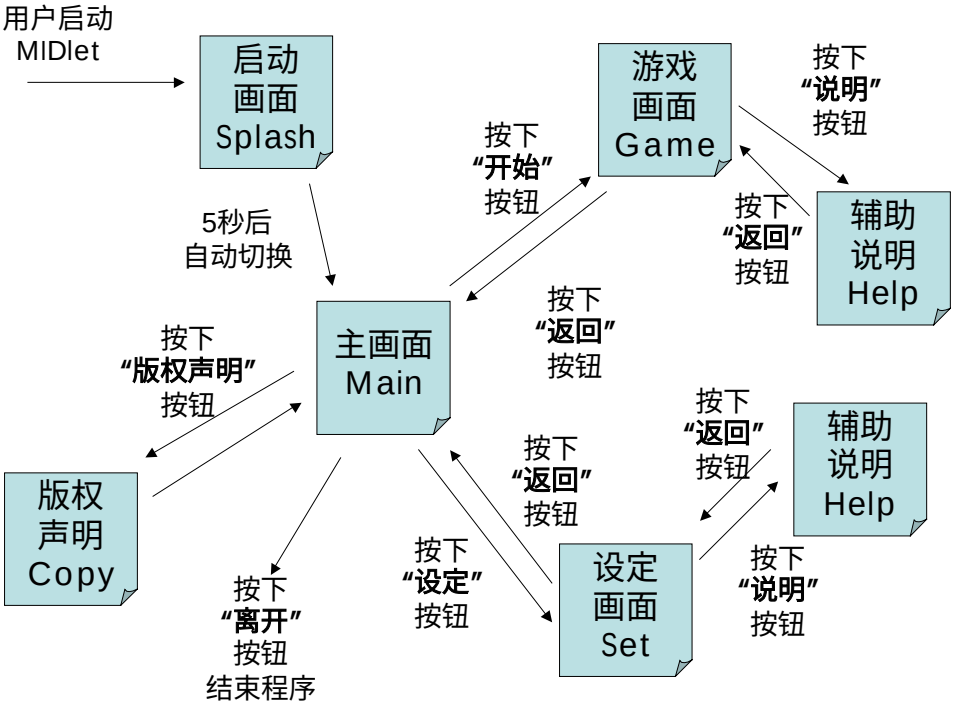

图 15-1　本章范例所设计的系统流程

完成系统流程的分析之后。接下来我们针对每个画面的特性进行分析。

1．辅助说明

每个辅助说明外观上大致上相同，只有说明文字不同而已。因此用 Form 来实现。

2．版权声明

这类型的画面纯粹只是要给用户一些相关信息。因此用 Alert 来实现最恰当。画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用。

3．主画面

为了方便用户操作，用 IMPLICIT 类型的 List 会比较恰当。此画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用。

4．设定画面

为了方便用户操作，用 Form 会比较恰当。此画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用。

5．游戏画面

用 Canvas 来实现最为恰当。此画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用。

6．启动画面

此画面只有在系统一开始用到而已，用 Alert 实现最合适。

15.4 流程控制器

我们将整个系统的流程控制都交给一个统一的流程控制器来完成，流程控制器以 Navigator 类实现。

由于在系统的每个地方都需要用到 MIDlet 实体和 Display 实体，为了避免重复撰写程序，因而在流程控制器中以 midlet 变量用来存放系统中惟一的一个 MIDlet 主体（也就是主程序），以 display 变量存放惟一一个 Display 实体（也就是代表屏幕的对象）。这两个变量必须在 MIDlet 的构造函数之中给予初始化。

另外，我们会为每一个画面定义一个常数整数，并使用 current 变量来记录目前画面。这个变量必须在 MIDlet 的 startApp()之中给予初始化。

根据之前的系统分析，Navigator 类的程序架构如下：

Navigator.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.* ;
public class Navigator
{
    final public static int MAIN_SCREEN = 1 ;
    final public static int GAME_SCREEN = 2 ;
    final public static int SET_SCREEN = 3 ;
    final public static int GAME_HELP_SCREEN = 4 ;
    final public static int SET_HELP_SCREEN = 5 ;


    public static MIDlet midlet ;
    public static Display display ;
    public static int current ;
```

```
    public static void show(Object obj)
    {
        switch(current)
        {
            case MAIN_SCREEN :
                break ;
            case GAME_SCREEN :
                break ;
            case SET_SCREEN :
                break ;
            case GAME_HELP_SCREEN :
                break ;
            case SET_HELP_SCREEN :
                break ;
        }
    }
    public static void flow(String cmd)
    {
        switch(current)
        {
            case MAIN_SCREEN :
                break ;
            case GAME_SCREEN :
                break ;
            case SET_SCREEN :
                break ;
            case GAME_HELP_SCREEN :
                break ;
            case SET_HELP_SCREEN :
                break ;
        }
    }
}
```

在流程控制器中，并没有包含版权声明画面和启动画面的流程控制。这是因为两者皆使用 Alert 实现，Alert 具有“ 时间一到就自动跳回前一个画面 ”的特性，因此我们不另外对其作流程控制比较好。

在流程控制器中其中，show()方法用来统一显示出画面，其 obj 参数是为了特殊用途（例如有些画面在显示前需要作一些初始化）。而 flow()方法用来统一判断程序的流程，利用 current 变量与 cmd 变量，就能轻易地判断目前程序的运作方向。

15.5 画面的设计

我们把流程的控制交给 Navigator.flow()方法，Navigator.flow()方法内部会使用 Navigator.show()方法来转换画面。每个画面的运作方式都以图 15-2 为基础来设计。

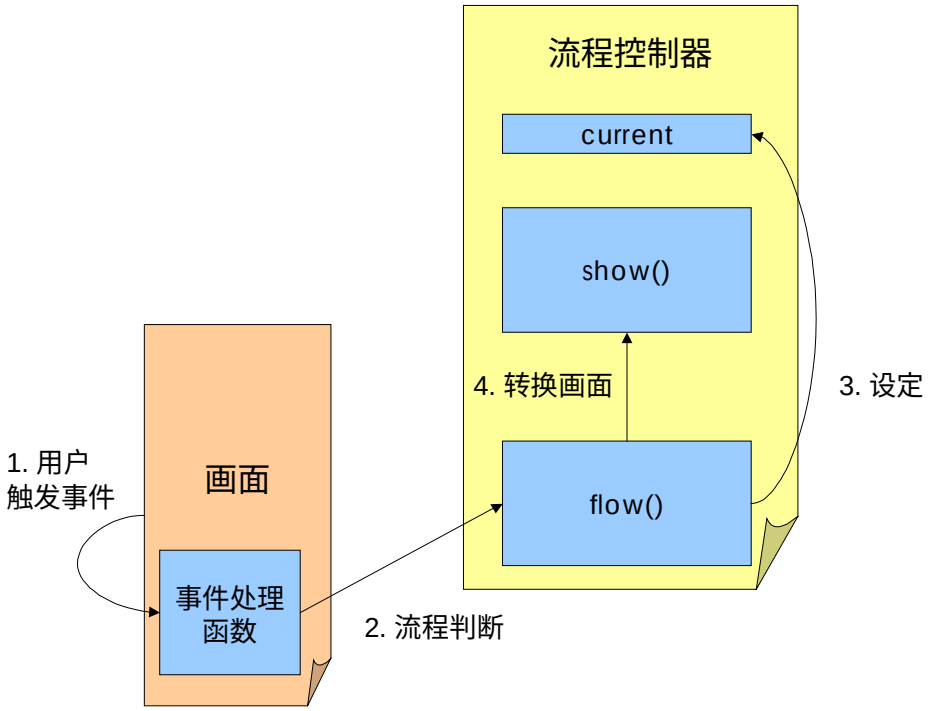

图 15-2 各个画面的运作方式

但是并非所有的画面都会用这个方法来转换，比方说整个系统的第一个画面，或是当我们用到线程的时候，有可能会直接调用 show()帮我们显示画面。

在调用 show()之前，别忘了先设定 current 变量，才能让程序正确地转换画面。

接下来我们实现每个画面如下。

1．辅助说明

由于每个辅助说明外观上大致上相同，只有说明文字不同而已。所以用 Form 来实现，HelpScreen 会根据构造函数传入值的不同而有不同的内容。

HelpScreen.java

```
import javax.microedition.lcdui.* ;
public class HelpScreen extends Form implements CommandListener
{
    public HelpScreen(String c)
    {
        super("辅助说明") ;
        append(c) ;
```

```
        addCommand(new Command("返回",Command.BACK,1)) ;
        setCommandListener(this) ;
    }
    public void commandAction(Command c,Displayable s)
    {
        Navigator.flow(c.getLabel()) ;
    }
}
```

2．版权声明

这类型的画面纯粹只是要给用户一些相关信息。因此用 Alert 来做最恰当。由于我们希望画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用，因此必须采用 *Singleton* 设计模式来实现。

CopyScreen.java

```
import javax.microedition.lcdui.* ;
public class CopyScreen extends Alert
{
    private static Displayable instance ;
    synchronized public static Displayable getInstance()
    {
        if(instance == null)
            instance = new CopyScreen() ;
        return instance ;
    }
    private CopyScreen()
    {
        super("版权声明") ;
        setString("此应用程序之版权属于 XYZ 公司所有") ;
        setType(AlertType.INFO) ;
        setTimeout(Alert.FOREVER) ;
    }
}
```

3．主画面

为了方便用户操作，用 IMPLICIT 类型的 List 会比较恰当。由于我们希望画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用，因此必须采用 *Singleton* 设计模式来实现。

MainScreen.java

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

```
public class MainScreen extends List implements CommandListener
{
   private static Displayable instance ;
   synchronized public static Displayable getInstance()
   {
      if(instance == null)
         instance = new MainScreen() ;
      return instance ;
   }
   private MainScreen()
   {
      super("版权声明",IMPLICIT) ;
      append("开始",null) ;
      append("设定",null) ;
      append("版权声明",null) ;
      append("离开",null) ;
      setCommandListener(this) ;
   }
   public void commandAction(Command c,Displayable s)
   {
      String cmd = getString(getSelectedIndex()) ;
      Navigator.flow(cmd) ;
   }
}
```

4．设定画面

为了方便用户操作，用 Form 会比较恰当。由于我们希望画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用，因此必须采用 *Singleton* 设计模式来实现。

SetScreen.java

```
import javax.microedition.lcdui.* ;
public class SetScreen extends Form
implements CommandListener
{
   private static Displayable instance ;
   synchronized public static Displayable getInstance()
   {
      if(instance == null)
         instance = new SetScreen() ;
      return instance ;
   }

   TextField url ;
   Gauge volume ;
```

```
    private SetScreen()
    {
        super("设定") ;
        url = new TextField(
            "请输入服务器位置",
            "socket://192.168.0.3:99",40,TextField.URL) ;
        append(url) ;
        volume = new Gauge("音量",true,10,3) ;
        append(volume) ;
        addCommand(new Command("辅助说明",Command.HELP,1)) ;
        addCommand(new Command("返回",Command.BACK,1)) ;
        setCommandListener(this) ;
    }
    public void commandAction(Command c,Displayable s)
    {
        Navigator.flow(c.getLabel()) ;
    }
}
```

5．游戏画面

用 Canvas 来实现最为恰当。由于我们希望画面在整个系统之中最好只有一个，除了可以保持本身状态，还可以降低内存的使用，因此必须采用 *Singleton* 设计模式来实现。

GameScreen.java

```
import javax.microedition.lcdui.* ;
public class GameScreen extends Canvas
implements CommandListener
{
    private static Displayable instance ;
    synchronized public static Displayable getInstance()
    {
        if(instance == null)
            instance = new GameScreen() ;
        return instance ;
    }

    private GameScreen()
    {
        addCommand(new Command("辅助说明",Command.HELP,1)) ;
        addCommand(new Command("返回",Command.BACK,1)) ;
        setCommandListener(this) ;
    }
    public void commandAction(Command c,Displayable s)
    {
        Navigator.flow(c.getLabel()) ;
    }
    public void paint(Graphics g)
```

```
    {
        g.setColor(125,125,125) ;
        g.fillRect(0,0,getWidth(),getHeight()) ;
        g.setColor(0,0,0) ;
        g.drawRect(10,10,60,70) ;
    }
}
```

15.6 完成流程控制器

完成所有画面之后，最后根据我们所作的系统流程分析，完成流程控制器如下：

Navigator.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.* ;
public class Navigator
{
    final public static int MAIN_SCREEN = 1 ;
    final public static int GAME_SCREEN = 2 ;
    final public static int SET_SCREEN = 3 ;
    final public static int GAME_HELP_SCREEN = 4 ;
    final public static int SET_HELP_SCREEN = 5 ;

    public static MIDlet midlet ;
    public static Display display ;
    public static int current ;

    public static void show(Object obj)
    {
        switch(current)
        {
            case MAIN_SCREEN :
                display.setCurrent(MainScreen.getInstance()) ;
                break ;
            case GAME_SCREEN :
                display.setCurrent(GameScreen.getInstance()) ;
                break ;
            case SET_SCREEN :
                display.setCurrent(SetScreen.getInstance()) ;
                break ;
            case GAME_HELP_SCREEN :
                display.setCurrent(new HelpScreen((String)obj)) ;
                break ;
            case SET_HELP_SCREEN :
                display.setCurrent(new HelpScreen((String)obj)) ;
                break ;
```

```
        }
    }
    public static void flow(String cmd)
    {
        switch(current)
        {
            case MAIN_SCREEN :
                if(cmd.equals("开始"))
                {
                    current = GAME_SCREEN ;
                    show(null) ;
                }else if(cmd.equals("设定"))
                {
                    current = SET_SCREEN ;
                    show(null) ;
                }else if(cmd.equals("版权声明"))
                {
                    display.setCurrent(CopyScreen.getInstance()) ;
                }else if(cmd.equals("离开"))
                {
                    midlet.notifyDestroyed() ;
                }
                break ;
            case GAME_SCREEN :
                if(cmd.equals("辅助说明"))
                {
                    current = GAME_HELP_SCREEN ;
                    show("游戏的操作方式") ;
                }else if(cmd.equals("返回"))
                {
                    current = MAIN_SCREEN ;
                    show(null) ;
                }
                break ;
            case SET_SCREEN :
                if(cmd.equals("辅助说明"))
                {
                    current = SET_HELP_SCREEN ;
                    show("设定方式") ;
                }else if(cmd.equals("返回"))
                {
                    current = MAIN_SCREEN ;
                    show(null) ;
                }
                break ;
            case GAME_HELP_SCREEN :
                if(cmd.equals("返回"))
                {
                    current = GAME_SCREEN ;
                    show(null) ;
                }
```

```
                break ;
            case SET_HELP_SCREEN :
                if(cmd.equals("返回"))
                {
                    current = SET_SCREEN ;
                    show(null) ;
                }
                break ;
        }
    }
}
```

15.7 MIDlet 主程序的设计

由于系统分析时，有启动画面的设计，我们使用 Alert 来实现。在流程的转换上，由于 Alert 本身的特性，我们采用“先显示主画面，再显示启动画面”的设计方式。程序代码如下：

FlowControl.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class FlowControl extends MIDlet
{
    boolean init = true ;
    public FlowControl()
    {
        Navigator.display = Display.getDisplay(this) ;
        Navigator.midlet = this ;
    }
    public void startApp()
    {
        Navigator.current = Navigator.MAIN_SCREEN ;
        Navigator.show(null) ;

        if(init)
        {
            Alert splash = new Alert("片头画面") ;
            splash.setType(AlertType.CONFIRMATION) ;
            splash.setTimeout(5000) ;
            Navigator.display.setCurrent(splash) ;
            init = false ;
        }
    }
    public void pauseApp()
```

```
    {
    }
    public void destroyApp(boolean con)
    {
    }
}
```

由于在系统的每个地方都需要用到 MIDlet 实体和 Display 实体，为了避免重复撰写相同的程序代码，这两个变量必须在 MIDlet 的构造函数之中给予初始化：

```
Navigator.display = Display.getDisplay(this) ;
Navigator.midlet = this ;
```

另外， current 变量必须在 startApp()之中给予初始化：

```
Navigator.current = Navigator.MAIN_SCREEN ;
Navigator.show(null) ;
```

最后，由于 startApp()不只会执行一次，它会在 MIDlet 从停止状态转换到运作状态时被调用，因此我们利用 init 变量来判断 startApp()是否是第一次执行，只有第一次执行时才需要显示出启动画面。为了简单起见，我们将程序设计成“不管停止状态前程序流程处于何处，回复到运作状态之后，一律回到主画面”的设计方式。

15.8 结束语

本章介绍了 MIDP 中最令程序员困扰的流程控制，并提出一个设计模式，让程序员可以用更方便、更简单且更具可维护性的方法来进行程序流程的控制。

本章介绍的设计模式只是许多设计模式中的一种而已，还有更多更优秀的设计模式等待我们发觉。