Lewolf
保留版权
签名未验证

第七章　应用程序中的面向状态

通过上一章对组件设计的分析，相信读者对 C++ Builder 中如何使用这些新的特性已经有了大致的了解，这一章将介绍如何在应用程序中使用面向状态的技术。

原则上讲，在应用程序中使用面向状态和在组件中使用是一样的，而且比在组件中使用更加灵活，限制更少。作为一个组件，其源代码一般是相对完整的，具有一定的独立性和通用性，因此在设计的时候必须全面考虑，而在应用程序中，则没有这种限制，因为你的代码是私有的，并不必考虑太多的共性，也就是说对完整性的考虑要相对灵活的多，这在后面的例子 Sells 中我们可以看到，很多的完整性实际上是在代码中省略掉的，就如同在一个类中，缺省值为“0”的成员我们并不需要显式的进行初始化，而这并没有违背完整性的要求，恰恰是利用了完整性来简化了代码。

除此之外，在应用程序中使用面向状态的方法时，比在组件中使用也要简单的多，因为属性很多组件中的特性在应用程序中是完全用不到的，甚至是不允许使用的，比如属性的缺省值，在组件中使用可以带来很多优点，然而在应用程序中则完全使用不上。

尽管在阅读了上一章的内容之后，可以认为已经掌握了在应用程序中使用面向状态的技术，但这两者还是有细微的差别，本章将简单的介绍一下这两者的区别，并在后面给出了两个使用面向状态的技术设计的小程序。

1 面向状态在应用程序和组件中的区别

可以这样认为，在应用程序中的使用面向状态，是组件中使用的面向状态的一个子集，因为面向状态的技术本身就是从组件技术发展而来的。

简单的剖析一下，面向状态在 C++ Builder 中使用的主要语言特征有一下几点：

- 属性的使用
- 事件的使用
- 方法的使用

实际上，方法并不能算是在面向状态中产生的概念，读者也可以将方法看作是一般的成员函数。只有属性和事件是面向状态中新产生的概念，而且事件可以看作是一种特殊的属性，那么在面向状态的概念中，实际上只有属性是全新的。前面我们也讲过，属性反映的是程序的状态，是程序状态的载体，因此掌握面向状态的技术就可以归结为对属性应用的掌握。

接下来我们就属性和事件在应用程序和组件中不同之处作以比较。

1.1 应用程序中的属性

在第三章讲解面向状态的思想时，我们就已经举过一个例子：Player，一个播放器，甚至是最初的 HelloWorld，都是在应用程序中使用面向状态的技术。在上一章的控件示例中，大量的使用属性来描述一个对象的特征，相信读者已经对__property 关键字以及

如何使用这个语法有了相当透彻的了解。但在应用程序中使用属性，可能会存在以下的特点：

- 在应用程序中的属性都不是 publish 类型
- 应用程序中的属性没有缺省值
- 属性必须是类的成员

除此之外，在应用程序中，属性的完整性要求和组件中也有很大的区别，在组件设计的时候，一个对象的使用，其环境是无法确定的，对属性的引用条件也是多种多样的，而作为组件的一个重要的对外接口，属性必须是完整的，这就是所谓完整性的要求，在上一章的例子 TDBJpegImage 中，使用了 TDataSource 和 TDataLink 等组件，通过对这两个组件的分析，可以看出组件中对完整性的要求是多么的严格。然而在应用程序中，并不是这样，当你为应用程序中的一个对象定义属性的时候，其应用的环境已经是确定的，如果你没有打算将这部分代码过于频繁的使用，正好可以利用完整性的要求进行反向分析，来简化代码，却可以保证在代码简化的同时没有额外的 Bug 出现。关于这一点，本书并不给出非常系统的分析过程，因为作者自己也是仅仅依靠经验来判断，而没有非常有效分析方法，请读者通过后面例子的学习自己掌握，或许没有作者的干扰，读者更容易另辟蹊径，找到更为合理的分析方法。

1.1.1 没有 publish 的属性

__publish 关键字是为了组件能够在 IDE 中使用而引入的，其含义就是“发布”，组件中只有被发布的属性和事件才可以在 IDE 的 Object Inspector 中可见。而目前 C++ Builder 并不支持在应用程序设计时创建可视属性，在其它一些伪编译的开发环境这样作则很可能是允许的。

一般而言，设计应用程序时的属性，都会是 public 类型，很少使用 private 和 protected 类型。也正是由于这个原因，在应用程序中使用属性，将不具备可视化的特征，不可能通过 ObjectInspector 对某个属性进行设置。其实这并不重要，需要知道，在应用程序中属性所描述的程序运行状态，是随时都在改变的，在 IDE 中对属性设置一个固定的初始值并没有多大的意义，运行时仍然会有大量的代码对其操作。

1.1.2 属性没有缺省值

这里指的缺省值不是说一个属性没有初始状态，而是说在定义一个属性的时候，不能使用类似组件设计时为属性指定缺省值。细心的读者都会发现，在 __property 关键字中的 default 部分实际上并没有真真正正的为属性进行缺省值的初始化工作，这个工作是在对象的构造函数中进行的,这只是为 IDE 提供了判断是否将属性保存在资源中的一个依据。在 C++ Builder 中，一个组件的属性，只有处于 publish 域中才可能被存储在资源中，而应用程序的属性没有 publish 的，也因此不能为其指定 default 值。

与此类似的，诸如 stored 也是为 publish 属性提供的，在应用程序中使用属性，同样也不能为其指定 stored 部分。而 index、read、write 等这些部分则在组件和应

用程序中都是可以使用的，用法也是一样的。

1.1.3 属性必须是类成员

属性必须是类的成员，这一点和组件中使用属性是一样的，这是由于 C++ Builder 语言所限定的。作为属性描述一个对象的状态，也理所当然的应该是对象的属性，从这一方面理解，属性自然而然的应当是类的成员。

在 C++ Builder 中，并没有严格限制属性必须是 TObject 派生类的成员，只要是符合 C++语法规则的类，都可以为其定义属性，而 VCL 组件必须是从 TCommponent 派生而来的，这也是两者的不同之处。C++语言支持多继承，因此如果应用程序中使用了自定义的类，并且是纯 C++语言的类，那么在为其定义的属性也将具备 C++语言的多继承特性，这和组件是大不一样的。由于属性实际上是由类成员和相应的读写函数组合而成的，因此属性的多继承性最终是取决于类成员和读写函数的多继承性。然而这种情况是非常少的，因为既然使用了 C++ Builder，VCL 类库自然是使用最为频繁的类库，而且 VCL 也相当齐全，基本上不需要用户重新从头定义一套自己的类库。

1.2 应用程序中的事件

从语言的角度来看，事件并不是一个新的事物，事件的类型通常是函数指针，这在 C 语言里就已经存在，但是到了 C++ Builder 之中，对这种函数指针进行了一些限制后，应用在了组件技术中，使事件已经不仅仅是一个函数指针。

在组件里面，事件的类型通常都是__closure，而且事件通常都会以属性的形式出现在声明的头文件中，但在应用程序中对这些要求都要宽松的多，因此对事件的使用也要灵活的多。简单的说，事件在应用程序中和在组件中相比有以下的特点：

- 程序中的事件不一定非得是 property 和 closure
- 程序中定义的事件通常都是需要根据状态更换处理句柄的

1.2.1 不是 property 的事件

在组件的设计中，所有的事件都是 proeprty，这是因为只有属性才可以实现设计时在 Object Inspector 中设置，但是所有的事件基本上都是直接读写的属性，没有设置函数和读取函数，也没有更新界面或者引发其它动作，因此在应用程序中定义事件并不一定要定义成 property，这是和组件所不同的。

另外就是组件中使用__closure 类型，是因为在使用一个组件的事件时，为事件指定的处理函数不是组件自己的，而在应用程序中为一个对象的事件指定的处理句柄，很可能就是该对象自身的成员函数，因此不一定非要指定事件为__closure 类型。

总的来说，定义事件在应用程序中比组件中要宽松的多，也灵活的多，但是事件体现着状态的变化这个规律应该是遵循的，在后面的例子 Sells 中我们就使用了事件来载入系统设置模块和报表模块，其类型就是一个自定义的非__closure 类型。

在组件设计中，定义一个事件，通常必须使用一个 TObject 类型的参数 Sender，

这是因为在组件的事件中，处理函数往往不是自身的成员函数，因此需要使用 Sender 来区分触发这个事件的对象，尽管在实际使用，用到 Sender 的机会并不是很多，但有时候是必须使用的，比如在函数中存在有对事件的触发者的操作(参见后面的例子 Drawing)，那么 Sender 则是必须的，尤其是在多个组件中使用同一个处理函数时。在应用程序的设计中，这个参数并不是必须的，因为一个事件的处理函数很可能就是自身的成员函数。

1.2.2 事件多为经常更换处理函数句柄

事件在组件中的应用，是为了提供给外界一个处理的接口，而在应用程序中使用事件，则并不是为了提供一个接口，因为这种接口的必要性在应用程序中并不存在。设计组件的时候，必须考虑允许用户在某种情况下对组件的状态改变而作出反应，这就需要用到事件，而在应用程序中，并不需要提供额外的接口，因为调用方和响应方都是你自己正在编辑的代码，直接调用就可以了。

尽管如此，我们还是要在应用程序中使用到事件，这种情况多出现在调用的函数具有不确定性，或者需要根据程序的不同状态进行更换。这就是我们之前所讲到面向状态中另一个非常重要的思想：“ 根据不同的螺丝来更换不同的扳手 ”。

实际在组件的事件中，根据不同的状态更换事件的处理函数句柄，也是允许的，而且也是非常有效的一种方法，只是大多数事件是不需要这样作的，这是一个习惯问题也是一个处理思想的问题，大多数程序员在使用 C++ Builder 的时候，仍然习惯于运行时不经常改变在 IDE 中设置的属性值，这实际上已经束缚住了设计者的手脚，希望有这个习惯的读者可以通过后面这个例子的示范，解放自己的手脚。

2 示范程序 Sells

这是作者第一次将组件设计技术用到应用程序中的一段代码，也正是这一个小小的程序才使作者最终产生了撰写这一本书的想法。因此作者将这个程序作为例程示范的第一个，尽管距离敲入这段代码的第一个字符已经过去了好几年，C++ Builder 的版本也从 4 到 5 再发展到了最新的版本 6，关于面向状态的思想也和最初编写这段代码时所想的大相径庭，但是作者还是决定将几年前的代码一刀未剪的提供给读者，因为这段代码实际上体现了作者如何从被动使用到主动掌握的一个过程。

2.1 程序功能的介绍

首先说明，这并不是一段优秀的代码，也非常短，编写这段代码的初衷是一位朋友想获取一个适合于单位或者企业餐厅使用的餐饮销售系统，要求非常简单，系统使用的是 PC 或者第三代 POS 机，仅允许使用磁卡消费，而不支持现金消费，在这个例子中磁卡刷卡器采用了最常用的键盘接口，本书的光盘中给出了所有的源代码，包括对对这个程序编制的帮助文件，这里不再对程序的需求进行过多的描述，读者参考光盘内容。下面对这个程序所实现的功能和结构作一个介绍。

这个程序中包括了三个主要的功能模块：

- 销售管理系统
- 系统设置
- 报表系统

系统设置部分和报表我们采用了外挂的动态链接库来完成，在主程序中并不包含，本书中我们仅对销售管理部分作详细的介绍，系统设置部分光盘中给出了源代码，有兴趣的读者自己阅读。接下来我们对这个程序的销售管理部分的功能作个简单介绍，有助于读者阅读后面的代码。

2.1.1 数据库表集

完成这一个销售的过程，至少有四个表集是必须的，他们分别是：

- 记录用户（消费者）信息的表集 Customer
- 记录商品信息的表集 Food
- 记录订单的表集 Sell
- 记录每个订单商品列表的表集 SellDetail

除了这四个表集以外，程序还使用了用于操作员信息存储的表集 System_User，这几个表集的关系如图 7.1 所示：

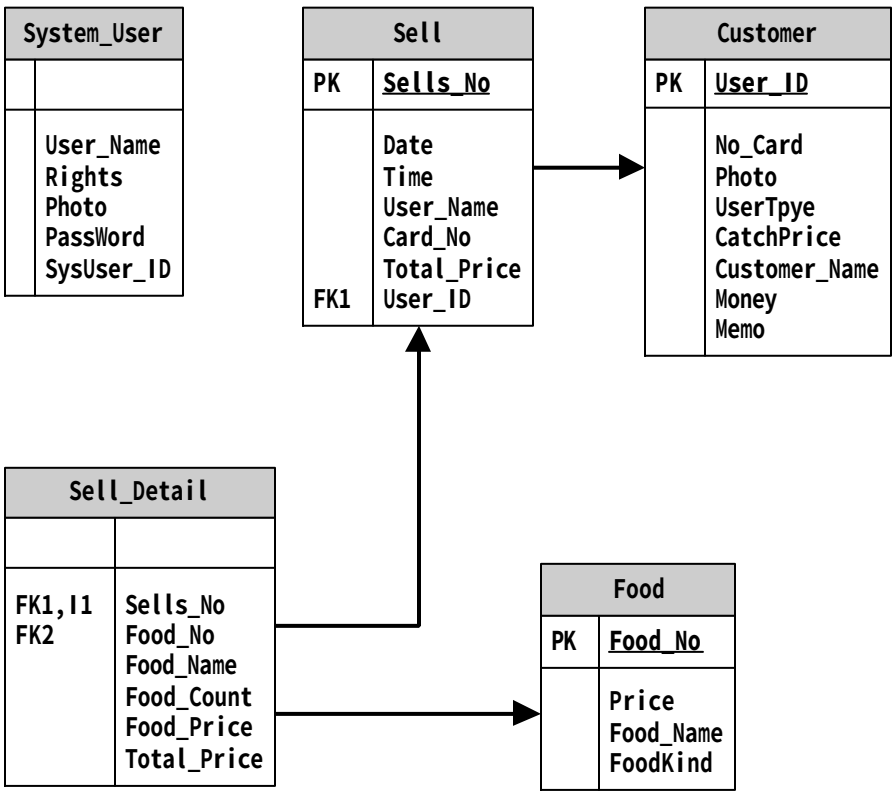


图 7.1 Sells 中数据库的结构

在 Sells 中并不是全部的字段都被使用了，其数据的类型基本可以根据名称进行判

断，除了 PassWord 使用了 Blob 类型之外，其它的字段基本都可以从名称看出来，或者读者可以从源码中获取详细信息，这里不再过多讲解。

2.1.2 程序第一次运行

在这个程序中，数据库使用的 Paradox，而且有一些设置都是保存在注册表里面的，数据库的配置信息也是保存在注册表中的，当程序运行的时候，会从注册表中读取相应的配置，并创建数据库连接组件 TDataBase 对象。如果系统中没有正确的配置数据库，或者还没有配置数据库信息，程序在运行是会自动打开数据库配置对话框，如图 7.2 所示：

用户选择该程序使用的 Paradox 数据库的存储路径，确定后程序对路径的合法性进行验证，若配置正确则保存信息进入主界面，设置的路径不合法则提示用户；若用户点击取消，程序将关闭；这个例子中没有提供该对话框的帮助内容。



图 7.2 Sells 第一次运行的界面

程序第一次运行的时候，其操作员的数据集是空的，用户登录可以采用缺省的用户：“ADM”和缺省的用户密码：“ADM”登录，登录后可对系统进行配置，系统配置的界面如图 7.3 所示：



图 7.3 Sells 系统设置界面

可以在系统维护窗体来配置用户信息（如上界面）、操作员信息、商品信息等，当操作员的数据库不为空的时候，缺省用户和缺省密码将自动失

效，不能再次用来登录。

系统维护模块是通过动态链接库来实现的，本书中不再讲述这部分内容，光盘中有全部的代码。

2.1.3 程序主界面

程序启动后的界面如图 7.4 所示：



图 7.4 Sells 的程序主界面

中央的矩形窗体为启动画面 Splash，启动画面中包含一个图片和一个动画效果，图片是设计在代码中的，动画效果是从硬盘载入的文件。主界面中 6 个功能按钮和用户操作菜单中的 6 个菜单项一一对应，其中只“用户登录”、“帮助”和“退出系统”3 个按钮有效，其它均为无效状态，当用户正确登录以后，其它的按钮才变为有效状态。

用户登录窗体非常简单，和一般程序的登录窗体没有什么两样如图 7.5 所示：



图 7.5 Sells 的登录窗口

2.1.4 销售管理系统

进入销售管理系统后界面如下图 7.6：



图 7.6 Sells 进入销售状态的界面

左边显示的是可供选择的食物列表，右边上面部分是每一次交易的详细商品列表，中间部分为客户的相关信息如照片、预存款金额、本次消费总额等，下面是信息提示栏，提示当前可进行的操作。

这个程序运行的硬件环境虽然安装了鼠标，但是所有销售的操作都是通过键盘操作来完成的，而且是通过代有磁卡刷卡器的小型键盘。当进入销售管理状态后，程序处于等候刷卡状态，刷卡的信息是通过键盘接口输入到计算机的，当客户的磁卡刷卡成功时，程序会将用户的相关信息显示在界面上，如上图；这时提示框中提示输入餐号，操作员根据左边商品信息选择购买的商品序号，当用户键入“+”或者“*”时，程序认为商品序号输入完毕，商品的缺省数量为 1；当操作员键入“+”时，程序认为用户将要继续购买其它商品，继续同上操作输入商品序号；当操作员键入“*”，程序认为用户更改当前输入商品的数量，程序转入接收商品数量状态，数量的精确度为 0.1 个单位，在修改商品数量后，操作员可以键入“+”继续添加商品；购买完毕时，键入回车则完成订单，任意时刻按 ESC 键可以取消订单，任意时刻均可以通过退格健或者是“-”来删除当前的商品。整个操作如同计算器一样使用，如用户购买半份米饭和一份炒菜，在刷卡后键入：

```
1*0.5+2
```

然后回车即可。详细的使用说明参见本例程的帮助文件。

2.2 状态的分析

在这个例子的设计过程中，确实是使用了本书所要介绍的“面向状态”的技术，就是将组件设计中的技术应用到了应用程序中，但是在设计的时候，作者并没有从头开始就使用了状态的分析方法，似乎没有经过分析过程，一方面是因为程序非常简单，几乎不用特别的对状态进行分析，另一方面是因为采用这种技术对逻辑关系处理要简单的多。但在介绍例子之前，我们还是对程序中的状态进行一个大概分析，确切的讲应该是设计后的一个总结。

- IsLogin 属性，记录是否处于登录状态。在这个程序中层次最高的状态应该是登录和非登录状态，在程序运行后只有登录后才可以进行其它的操作，否则任何工作都不能进行，而且登录状态对其它的状态具有最高的约束力。与登录状态相关的其它组件有 Button 的 Enabled，MenuItem 的 Enabled，Button 和 MemuItem 的 Caption 等属性，甚至包括了登录用户的操作权限（例程中并没有作关于用户权限设置的代码，但是作了这方面的考虑）等。因此 bool 类型的 IsLogin 属性代表了整个程序最高层次的状态。
- IsSelling 属性，标记是否处于销售状态。从程序状态的层次来讲，紧接者登录状态的应该是处于那个功能模块，在 Sells 中，报表模块和系统设置模块是通过动态链接库载入的，而且是调用了模态对话框实现的，因此并没有为每一个模块定义相应的状态，而只对销售状态定义了一个属性 IsSelling，bool 类型的属性 IsSelling 为真的必要条件是程序必须已经登录（并且登录用户具有相应的权限），受到 IsSelling 状态约束的其它属性有，相关数据集的 Active、销售界面元素的 Visible 等。
- Status 属性，用于记录在销售状态的键盘处理的状态，这个属性的命名并没有非常明确的含义，这时由于作者在设计的时候考虑不够充分而造成的。实际上使用 SellingStatus 可能会更确切一些。根据前面的介绍，这个属性总共有 3 种状态：等候刷卡、等候商品序号、等候商品数量，例子中为 Status 属性预定义了四个状态，其中_NONE_SELLING 和 IsSelling 实质上是重复的，程序中并没有使用。受到 Status 约束的有操作提示信息等界面元素，主要是在不同的 Status 状态中，使用不同的子键盘处理函数来进行键盘消息的处理。
- SubKeyProc 键盘处理事件，这是一个类似事件的函数处理句柄，受到 IsSelling 和 Status 的约束。在销售状态的时候，程序只支持键盘方式操作，鼠标不能进行销售，因此在程序中使用了 Application 的 OnMessage 事件来接收程序的所有 Windows 消息并进行的预处理，然后根据不同 Status 状态，会将预处理过的键盘消息送给子键盘处理函数进行处理。SubKeyProc 总共有三个不同的处理函数，分别对应等候刷卡、接收商品、改变商品数量这三个状态，这三个函数是本程序的核心代码，所有销售相关的数据处理都是在这三个函数中完成的。

除了上面几个关键的状态属性之外，程序中大部分组件的属性也与程序的状态建立了相互约束和关联的关系，这和一般程序中使用属性没有区别，这里不再一一介绍。

2.3 实现代码及介绍

实现整个程序总共使用了 4 个 Form 单元、一个数据库模块和一个辅助单元 Extend，Extend 单元是将 Main 单元中不能可视化编辑的函数提取出来放在一起的。其它单元各有用途，我们分别介绍。

2.3.1 Main 主单元

主单元是整个程序的主界面，设计时 Form 如下图：

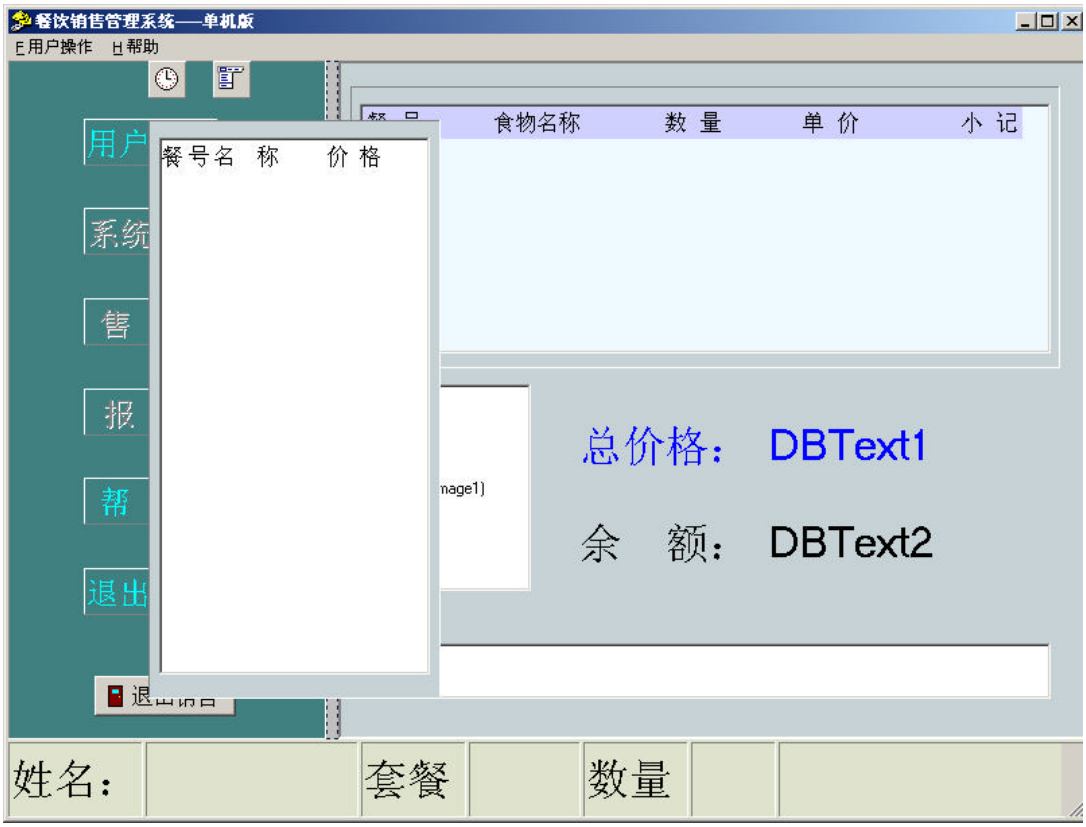


图 7.7 Sells 设计时的主窗体

界面上的元素并不所有的 Visible 都是 true;而是根据运行时的状态；来决定那些元素是可视的，那些不是可视的。

这个 Form 的 dfm 文件特别庞大，这里不再给出，读者可以参考光盘中的源码。在主 Form 中包含了一些非可视化的组件，供数据库等组件使用，读者可以参考下图理解：

这是在 C++ Builder 6 中生成的类——属性关系图，其中符号的含义参考 C++ Builder 6 的帮助内容。

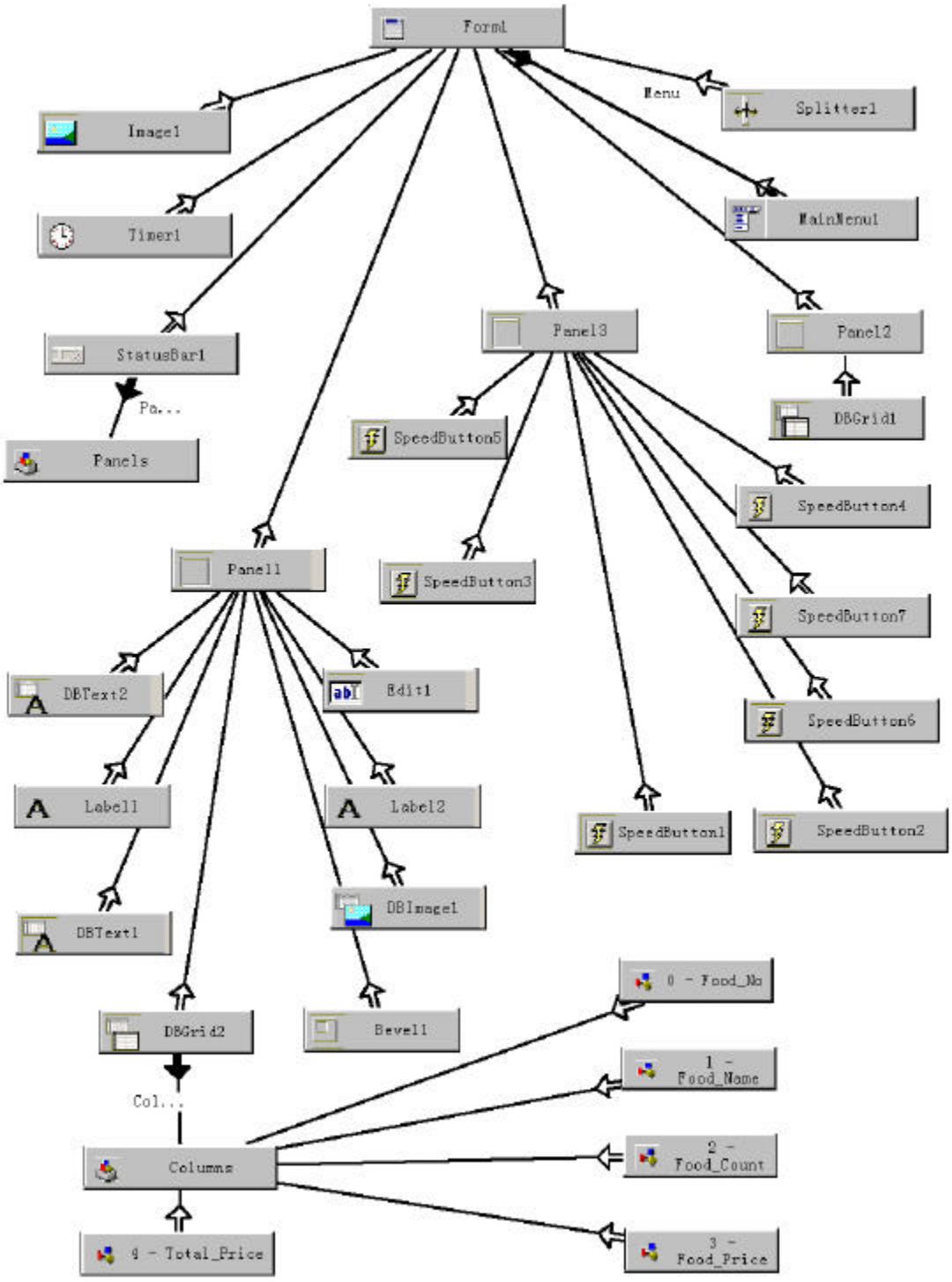


图 7.8 Sells 的主窗体中组件的相对关系

Main **单元的头文件如下：**

```
#ifndef MainH
#define MainH
#define _WAIT_CARD          0x01   //定义状态_WAIT_CARD 的标识符
#define _WAIT_FOOD_NO       0x02   //状态_WAIT_FOOD_NO 的标示符标识符
#define _WAIT_FOOD_COUNT    0x04   //状态_WAIT_FOOD_COUNT 的标识符
#define _NONE_SELLING       0x08   //状态_NONE_SELLING 的标识符
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include <Buttons.hpp>
#include <Db.hpp>
#include <DBTables.hpp>
#include <Menus.hpp>
#include <ComCtrls.hpp>
#include <DBGrids.hpp>
#include <Grids.hpp>
#include <Mask.hpp>
#include <DBCtrls.hpp>
#include "DataBase.h"     //数据库单元

typedef void __fastcall (__closure *TKeyProc)(Word &Key);
   //用于键盘消息处理的Closure 类型；
typedef void __stdcall (*TExtProc)(HWND,AnsiString);
   //用于挂入动态连接库的Proc 类型，供对话框使用时，传递一个窗口句柄，和数据库的路径
//---------------------------------------------------------------------
class TForm1 : public TForm
{
__published:
       TImage *Image1;          //显示背景图片的Image
       TMainMenu *MainMenu1;    //主窗体菜单
       TMenuItem *N1;           //主菜单项
       TMenuItem *N2;
       TMenuItem *N3;
       TMenuItem *N4;
```

```
    TMenuItem *N5;
    TMenuItem *N6;
    TMenuItem *N7;
    TMenuItem *N8;
    TMenuItem *N9;
    TMenuItem *N10;
    TMenuItem *N12;
    TPanel *Panel1;
    TPanel *Panel2;
    TPanel *Panel3;
    TStatusBar *StatusBar1;
    TTimer *Timer1;
    TEdit *Edit1;
    TDBText *DBText1;
    TDBText *DBText2;
    TLabel *Label1;
    TLabel *Label2;
    TDBGrid *DBGrid1;
    TDBGrid *DBGrid2;
    TDBImage *DBImage1;
    TSplitter *Splitter1;
    TSpeedButton *SpeedButton1;
    TSpeedButton *SpeedButton2;
    TSpeedButton *SpeedButton3;
    TSpeedButton *SpeedButton4;
    TSpeedButton *SpeedButton5;
    TSpeedButton *SpeedButton6;
    TSpeedButton *SpeedButton7;
    TBevel *Bevel1;
    void __fastcall Splitter1CanResize(TObject *Sender, int &NewSize,
                                       bool &Accept);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall Timer1Timer(TObject *Sender);
    void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
    void __fastcall SpeedButton1Click(TObject *Sender);
    void __fastcall SpeedButton2Click(TObject *Sender);
    void __fastcall SpeedButton5Click(TObject *Sender);
    void __fastcall SpeedButton6Click(TObject *Sender);
    void __fastcall SpeedButton4Click(TObject *Sender);
```

```
        void __fastcall SpeedButton3Click(TObject *Sender);
        void __fastcall SpeedButton7Click(TObject *Sender);
        void __fastcall FormCanResize(TObject *Sender, int &NewWidth,
                                      int &NewHeight, bool &Resize);
        void __fastcall N10Click(TObject *Sender);
private:
        int  FStatus;              //程序运行状态的内部数据
        bool FIsLogIn;             //记录登录状态的内部数据
        bool FIsSelling;           //销售状态的内部数据
        AnsiString FFoodNo;        //提供内部使用的键盘字符串缓存,记录商品序号
        AnsiString FFoodCount;     //提供内部使用的键盘缓存，记录商品数量
        TKeyProc FSubKeyProc;      //子键盘处理事件的内部存储数据
        TExtProc FSysManProc;      //外挂系统设置的入口指针
        TExtProc FReportProc;      //外挂报表系统的入口指针
public:
        Currency     TotalPriceBuf;  //用于记录当前合计商品价格
        Currency     LeftMoney;      //记录磁卡剩余金额，
        int          SerialNo;       //用于记录交易的系统流水号
        AnsiString   CardNoBuf;      //刷卡信息的内部缓存
             //Status 属性用于记录程序的运行状态，_WAIT_CARD、_WAIT_FOOD_NO
             //以及 _WAIT_FOOD_COUNT 三个状态。
        __property int Status = {read = FStatus,write = SetStatus};
             //记录是否为登录状态的属性
        __property bool IsLogIn = {read= FIsLogIn,write = SetLogIn};
             //IsSelling 记录是否处于销售状态
        __property bool IsSelling = {read = FIsSelling,write = SetSelling};
             //FoodNO 用于记录用户键入的商品序号
        __property AnsiString FoodNo = {read = FFoodNo,write = SetFoodNo};
             //FoodCount 用于记录用户键入的商品数量
        __property AnsiString FoodCount = {read = FFoodCount,write =
                                           SetFoodCount};
             //SubKeyProc 在程序的键盘消息处理触发的子键盘处理事件
        __property TKeyProc SubKeyProc = {read = FSubKeyProc,write =
                                          FSubKeyProc};
             //调用系统设置的函数指针
        __property TExtProc SysManProc = {read = FSysManProc,write =
                                          FSysManProc};
             //调用报表系统的函数指针
        __property TExtProc ReportProc = {read = FReportProc,write =
```

```
                                        FReportProc};
        __fastcall TForm1(TComponent* Owner);          //构造函数
        void __fastcall SetStatus( int Value );        //Status 的设置函数
        void __fastcall SetLogIn( bool Value );        //Login 的设置函数
        void __fastcall SetSelling( bool Value );      //Selling 的设置函数
        void __fastcall SetFoodNo( AnsiString Value ); //FoodNo 的设置函数
        void __fastcall SetFoodCount( AnsiString Value );
                                        //FoodCount 的设置函数
        //主键盘处理函数，当程序出在 Selling 状态时，下面函数为 OnMessage 的处理句柄
        void __fastcall MyKeyProc( tagMSG &Msg, bool &Handled );
             //用于接收磁卡输入的键盘子处理函数
        void __fastcall WaitCardKeyProc(WORD &Key);
             //用于接收输入商品序号的键盘子处理函数
        void __fastcall WaitFoodNoProc(WORD &Key);
             //用于接收输入商品数量的键盘子处理函数
        void __fastcall WaitFoodCountProc(WORD &Key);
};
extern PACKAGE TForm1 *Form1;
#endif
```

上面这段代码中，定义了两个事件类型，一个用于载入的报表模块和系统设置模块，另一个是用于核心代码的子键盘处理函数。在程序的 Status 属性的设置函数中，不同的状态值在更新界面显示的同时，也更换相应的子键盘处理函数，因此在 Application 的 OnMessage 事件的处理函数中并不需要根据当前状态来选择将要调用的键盘子处理函数，而只是简单的触发 SubKeyProc 事件就可以了。

下面是 Main 单元的实现代码：

```
#include <vcl.h>
#pragma hdrstop
#include "Main.h"       //主单元的头文件
#include "Pass.h"       //密码验证单元的头文件
#include "Path.h"       //路径设置单元的头文件
#include "jpeg.hpp"     //C++ Builder 中 Jpeg 文件的处理单元
#include "Splash.h"     //启动画面的头文件

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
    Form7 = new TForm7(NULL);     //在主窗体创建之前，显示 Splash 界面
    Form7->Refresh();             //必须更新方能显示。
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{   //在 Form1 创建时，根据注册表来加载响应的系统设置模块和报表模块
    TRegistry *Reg = new TRegistry;
    AnsiString ReportFile;        //报表模块的文件名
    AnsiString SysManFile;        //系统设置模块的文件名
    Panel2->Top = 10;             //Panel2 在设计时位置偏离运行时需要的位置
    Panel2->Left = 8;
    SubKeyProc = NULL;            //设置子键盘处理句柄为空
    try                           //打开 PlugIn 子键
    {
      Reg->RootKey = HKEY_LOCAL_MACHINE;  //根键
      if ( Reg->OpenKey("\\Software\\Lewolf\\PlugIn",false) )
      {
        ReportFile = Reg->ReadString("Reports");  //读取报表模块文件名
        SysManFile = Reg->ReadString("SysMan");   //读取系统设置模块文件名
        Reg->CloseKey();
      }
      if( ReportFile == "" )                //设置为空时，按照确省文件加载
         ReportFile = "Reports.dll";        //缺省的报表模块文件名
      if( SysManFile == "" )                //设置为空时，按照确省文件加载
         SysManFile = "SysMan.dll";         //缺省的系统设置模块文件名
      //加载系统设置模块并导入入口函数，入口函数为"Manager"
      SysManProc = (TExtProc)GetProcAddress(
                    LoadLibrary(SysManFile.c_str()),"Manager");
     //加载报表模块并导入入口函数 ，入口函数为"Reports"
      ReportProc = (TExtProc)GetProcAddress(
                   LoadLibrary(ReportFile.c_str()),"Reports");
      //Reg->RootKey = HKEY_LOCAL_MACHINE;
      //根据注册表设置加载主界面的背景图片
      if ( Reg->OpenKey("\\Software\\Lewolf\\BackGround",false) )
      {
        AnsiString FileName = Reg->ReadString("MainPicture");
```

```
        Image1->Picture->LoadFromFile( FileName );
        Reg->CloseKey();
      }
      else   //未设置背景键值时，按照确省的图片加载
        Image1->Picture->LoadFromFile( "..\\Images\\Main.jpg" );
   }
   catch(...)
   {    /*有异常时忽略，但需要释放资源*/
      delete Reg;
   }    //还可以使用 finally 关键字来处理 Reg 的资源释放
   delete Reg;
}
//---------------------------------------------------------------------
void __fastcall TForm1::Splitter1CanResize(TObject *Sender, int &NewSize,
     bool &Accept)
{   //Splitter1 是一个多余的控件，最早设计时考虑要作界面分割而保留该控件。
   Accept = false;           //这个界面中，不允许改变 Splitter 的分割区域
}
//---------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{  //定时器每隔一秒钟刷新一次当前时间的显示
   StatusBar1->Panels->Items[6]->Text =
       TDateTime::CurrentDateTime().TimeString();
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
   Status = _NONE_SELLING;    //关闭前必须保证没有处于销售状态
}
//---------------------------------------------------------------------
void __fastcall TForm1::SpeedButton1Click(TObject *Sender)
{  //登录按钮和登录菜单共用的事件句柄
   if( !IsLogIn )               //"登录"或"退出登录"按钮按下时，
      Form2->Visible = true;    //等同于相应菜单的操作，登录成功的话，IsLogin
   else                         //属性的状态在 Form2 中更改
      IsLogIn = false;
}
//---------------------------------------------------------------------
void __fastcall TForm1::SpeedButton2Click(TObject *Sender)
```

```
{   //SpeedButton2 为调入系统设置的按钮
    if(SysManProc)           //系统设置按钮或菜单响应的函数
      SysManProc(Application->Handle,DM->Sell->Directory );
    else
      ShowMessage("系统管理单元没有载入！");
}
//---------------------------------------------------------------------

void __fastcall TForm1::SpeedButton7Click(TObject *Sender)
{   //SpeedButton7 为调入报表的按钮
    if(ReportProc)
      ReportProc(Application->Handle,DM->Sell->Directory );
    else
      ShowMessage("报表单元没有载入！");
}
//---------------------------------------------------------------------

void __fastcall TForm1::SpeedButton5Click(TObject *Sender)
{   //SpeedButton5 为退出销售系统的按钮，
    IsSelling = false;     //将程序状态设置为非销售状态
}
//---------------------------------------------------------------------

void __fastcall TForm1::SpeedButton6Click(TObject *Sender)
{   //SpeddButton6 为打开销售系统
    IsSelling = true;      //将程序的状态设置为销售状态
}
//---------------------------------------------------------------------
void __fastcall TForm1::SetFoodNo( AnsiString Value )
{   //FoodNo 的设置函数，这里没有额外的处理
    FFoodNo = Value;
}
//---------------------------------------------------------------------
void __fastcall TForm1::SetFoodCount( AnsiString Value )
{   //FoodCount 的设置函数，这里没有额外的处理
    FFoodCount = Value;
}
//---------------------------------------------------------------------
void __fastcall TForm1::SpeedButton4Click(TObject *Sender)
```

```
{   //SpeedButton 为退出程序的按钮
    Close();
}
//---------------------------------------------------------------------
void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{   //SpeedButton3 为帮助按钮
    Application->HelpCommand(HELP_CONTENTS, 0);
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCanResize(TObject *Sender, int &NewWidth,
      int &NewHeight, bool &Resize)
{   //阻止 Form1 的大小改变
    Resize = false;
}
//---------------------------------------------------------------------

void __fastcall TForm1::N10Click(TObject *Sender)
{   //N10 菜单为"关于"菜单 ,显示 Splash 窗口
    Form7->Show();
}
```

以上代码都是作为运行时的流程代码，即决定了什么样的条件下要干什么，是和用户的交互过程，基本上没有涉及到程序内部的象的状态之间的约束关系，这些代码都是比较简单的，决定内部逻辑关系的代码我们放在了扩充单元 Extend.cpp 中，在这个单元中，包括了核心的键盘处理代码，以及主要属性的设置函数，这些代码一般设计成功之后变化不会太大，除非改变程序的功能。

2.3.2 Extend 单元

在这个单元中包含了一个消息处理函数，用于 Application 的 OnMessage 事件；三个子键盘处理函数；分别为销售状态的三个子状态键盘处理函数；IsLogin 的设置函数；IsSelling 的设置函数；Status 的设置函数。

其中消息处理函数和子键盘处理函数实际上是整个程序功能实现的核心代码，但这三个函数的内容和处理方法和我们要讲述的面向状态并没有特别大的关系，只是在每个处理函数中都包含了对状态的设置。读者只要仔细分析不难理解，这里也不给出详细的注解说明，代码如下：

```
#include <vcl.h>
#pragma hdrstop
```

```
#include "Main.h"            //主单元头文件
#include "DataBase.h"        //数据库单元的头文件
#pragma package(smart_init)
//---------------------------------------------------------------------
void __fastcall TForm1::MyKeyProc( tagMSG &Msg, bool &Handled )
{   //这是程序在 Selling 状态时 Application 的 OnMessage 事件的处理函数
    if( Msg.message == WM_KEYDOWN )
    {    //只获取键盘按下的消息
      WORD Key;
      Key = Msg.wParam;
      switch (Msg.wParam)     //下面代码将键盘扫描码转换为 Ascii 码，
      {                       //由于 Windows98 和 Winnt 以及 Windows 2000 的
        case VK_NUMPAD0:      //键盘消息有所不通，这个键盘处理函数将主要完成
           Key = '0';         //不同操作系统的兼容性问题，
           break;             //同时，由于这个程序运行的计算机只安装着带有磁卡刷
        case VK_NUMPAD1:      //卡器的小型键盘，这个函数还要解决键盘键的映射问题
           Key = '1';         //关于刷卡器键盘的映射参考 Sells 的帮助文件
           break;
        case VK_NUMPAD2:
           Key = '2';
           break;
        case VK_NUMPAD3:
           Key = '3';
           break;
        case VK_NUMPAD4:
           Key = '4';
           break;
        case VK_NUMPAD5:
           Key = '5';
           break;
        case VK_NUMPAD6:
           Key = '6';
           break;
        case VK_NUMPAD7:
           Key = '7';
           break;
        case VK_NUMPAD8:
           Key = '8';
           break;
```

```
        case VK_NUMPAD9:
           Key = '9';
           break;
        case VK_F2:            //以下几个Case中是将小键盘中的功能键映射为
           Key = '*';          //所需要的Ascii字符编码
           break;              //关于详细的映射关系，在本书的原代码中找到该例
        case VK_F3:            //子的帮助文件，帮助文件中有详细的解释
           Key = '-';
           break;
        case VK_F4:
           Key = '+';
           break;
        case VK_MULTIPLY:
           Key = '*';
           break;
        case VK_ADD:
           Key = '+';
           break;
        case VK_SUBTRACT:
           Key = '-';
           break;
        case VK_DECIMAL:
           Key = '.';
           break;
      }
      if(SubKeyProc)           //将处理后的键盘信息，以Ascii码
         SubKeyProc(Key);      //的形式交给子键盘处理函数
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::WaitCardKeyProc(WORD &Key)
{   //等候刷卡状态的键盘子处理函数，
    switch (Key)
    {       //当接收到回车键（刷卡器可以设置为以回车键结尾）的时候，
       case VK_RETURN:       //认为刷卡结束
          if( DM->Customer_Table->Locate("No_Card",Variant( CardNoBuf ),
                        TLocateOptions()<< loCaseInsensitive ))
          {  //在用户信息表中匹配该卡号的用户，以下是该用户合法时的代码
```

```
            Status = _WAIT_FOOD_NO;    //将程序的状态设置为等候输入商品序号
            StatusBar1->Panels->Items[1]->Text =
                    DM->Customer_Table->FieldValues["Customer_Name"];
                    //更新相应的StatusBar信息
            DM->Sell_Table->Append();              //增加新的销售记录
            DM->Sell_TableSells_No->Value = SerialNo + 1;//设置销售流水号
            DM->Sell_TableDate->Value = TDateTime::CurrentDate();
            //记录交易日期
            DM->Sell_TableTime->Value = TDateTime::CurrentTime();
           //记录交易时间
            DM->Sell_TableCard_No->Value = CardNoBuf;  //记录交易卡号
            DM->Sell_TableUser_Name->Value =           //记录交易人姓名
                   DM->Customer_Table->FieldValues["Customer_Name"];
            DM->Sell_TableUser_ID->Value =
                   DM->Customer_Table->FieldValues["User_ID"];
            DM->SellDetail_Table->Append();        //增加商品列表记录
            DM->Sells_No->Value = DM->Sell_TableSells_No->Value;
                                                   //记录商品列表的订单编号
            CardNoBuf = "";                        //清空刷卡信息缓存
            TotalPriceBuf = 0;                     //设置商品总交易额为0
            DM->Customer_Table->Edit();            //将客户信息表置为可编辑，因为
          }                                        //销售后需要更新客户的预存款金额
          else
          {
            CardNoBuf = "";                        //卡号无效时，清空缓存
          }
          break;                                   //取消时，清空缓存
      case VK_ESCAPE:
          CardNoBuf = "";
          break;
      default :                        //确省处理，即Key为有效的Ascii字符码时
          CardNoBuf = CardNoBuf + (char)Key ; //将该字符续在卡号缓存的最后
    }
}
//---------------------------------------------------------------------
void __fastcall TForm1::WaitFoodNoProc(WORD &Key)
{   //该函数为等候输入商品序号的状态时，键盘的子处理函数
   if( Key >= '0' && Key <= '9' )    //有效输入Ascii字符为0~9，
   {                                  //因为序号为整型数据
```

```
    AnsiString  Temp;                //临时变量，存放新键入的商品序号
    Temp = FoodNo + (char)Key;
          //根据接收到的商品序号，查找相应的商品信息，
          //根据商品信息填写商品列表的相应的字段
          //并且计算总价，以及客户剩余金额
    if( DM->Food_Table->Locate("Food_No",
                Variant(StrToInt( Temp )),TLocateOptions()<<
                loCaseInsensitive) )
    {
      DM->Food_Price->Value = DM->Food_Table->FieldValues["Price"];
      DM->Food_Name->Value = DM->Food_Table->FieldValues["Food_Name"];
      DM->Food_Count->Value = 1;   //确省情况下，商品数量为1
      DM->Food_No->Value = DM->Food_Table->FieldValues["Food_No"];
      DM->Customer_Table->FieldValues["Money"] =
                      DM->Customer_Table->FieldValues["Money"] +
                      (double)TotalPriceBuf;
      TotalPriceBuf = TotalPriceBuf –
                      DM->SellDetail_TableTotal_Price->Value;
      DM->SellDetail_TableTotal_Price->Value = DM->Food_Count->Value *
                      DM->Food_Price->Value;
      TotalPriceBuf = TotalPriceBuf +
                      DM->SellDetail_TableTotal_Price->Value;
      DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
      DM->Customer_Table->FieldValues["Money"] =
                      DM->Customer_Table->FieldValues["Money"] –
                      (double)TotalPriceBuf;
      FoodNo = Temp;     //若商品有效时，则更新商品编号缓存，
    }                    //否则保持不变，忽略用户键入的字符
}
else if( Key == VK_ESCAPE )
{  //用户按ESC键，则取消本次交易，并删除本次交易的商品列表
   //恢复程序状态到等候刷卡状态
   DM->SellDetail_Table->Cancel();
   DM->SellDetail_Table->Last();
   for( ; !DM->SellDetail_Table->Bof ;DM->SellDetail_Table->Delete());
   DM->Sell_Table->Cancel();
   DM->Customer_Table->Cancel();
   FoodNo = "";
   Status = _WAIT_CARD;     //设置状态到等候刷卡状态
```

```
}
else if( Key == VK_RETURN )
{  //用户按回车键，结束本次交易，并将程序状态设置到
   //等候刷卡状态，继续下一个交易
   if( FoodNo == "" && DM->SellDetail_Table->RecordCount == 0)
      return;          //商品列表为空时，不能结束交易，只能取消交易
   TUpdateKind UpdtKind;
   DM->SellDetail_Table->Post();
   DM->Sell_Table->Post();
   DM->Customer_Table->Post();
   SerialNo = SerialNo + 1;
   Status = _WAIT_CARD;    //设置到等候刷卡状态
   FoodNo = "";            //将 FoodNo 商品序号清空
}
else if( Key == VK_BACK || Key == '-' )
{   //当用户安下退格键或者减号（小键盘的支持），时删除商品列表中
    //当前的商品，并重新计算商品总价和用户余额
   if( DM->SellDetail_Table->Modified && FoodNo != "")
   {  //当前商品正在输入的时候，删除当前商品所进行的操作
      DM->Customer_Table->FieldValues["Money"] =
                     DM->Customer_Table->FieldValues["Money"] +
                     (double)TotalPriceBuf;
      TotalPriceBuf = TotalPriceBuf –
                     DM->SellDetail_TableTotal_Price->Value;
      DM->SellDetail_Table->Cancel();
      DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
      DM->Customer_Table->FieldValues["Money"] =
                     DM->Customer_Table->FieldValues["Money"] –
                     (double)TotalPriceBuf;
   }
   else if( DM->SellDetail_Table->RecordCount )
   {  //商品列表不为空时的操作（可能会是前面的商品记录）
      DM->SellDetail_Table->Cancel();
      TotalPriceBuf = TotalPriceBuf –
                     DM->SellDetail_TableTotal_Price->Value;
      DM->Customer_Table->FieldValues["Money"] =
                     DM->Customer_Table->FieldValues["Money"] +
                     (double)TotalPriceBuf;
      DM->SellDetail_Table->Delete();
```

```
            DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
            DM->Customer_Table->FieldValues["Money"] =
                          DM->Customer_Table->FieldValues["Money"] –
                          (double)TotalPriceBuf;
         }
         //删除了当前的商品后，必须重新处于接收商品信息的最初状态等候商品序号
         DM->SellDetail_Table->Append();
         DM->Sells_No->Value = DM->Sell_TableSells_No->Value;
         FoodNo = "";                    //清空商品序号缓存
         Status = _WAIT_FOOD_NO;         //设置到接收商品序号的状态
      }
      else if( Key == '+'  && DM->SellDetail_TableTotal_Price->Value != 0 )
      {  //当用户按下加号键，并且当前商品的合计价格不为 0 时，
         //进入下一个商品的输入，将程序的状态置于等候商品序号的状态
         DM->SellDetail_Table->Post();
         DM->SellDetail_Table->Append();
         DM->Sells_No->Value = DM->Sell_TableSells_No->Value;
         FoodNo = "";
         Status = _WAIT_FOOD_NO;
      }
      else if( Key == '*')
      {  //当用户按下星号键时，表示改变当前商品的数量，
         //将程序的状态设置为等候商品的数量状态
         if( FoodNo != "" )
         {
           Status = _WAIT_FOOD_COUNT;
           FoodCount = "";
         }
      }
   }
   //---------------------------------------------------------------------
   void __fastcall TForm1::WaitFoodCountProc(WORD &Key)
   {  //该函数为等候商品数量状态的键盘子处理函数，有效的输入字符为 0~9 和小数点号，
      //小数点只能有一个，并精确到小数点后一位
      if( Key >= '0' && Key <= '9' )
      {
         if( FoodCount.Pos('.') < FoodCount.Length() )
           return;     //该判断忽略小数点后第二位以上的输入。
         FoodCount = FoodCount + (char)Key;
```

```
    DM->Food_Count->Value = StrToFloat( FoodCount );
    //以下为重新计算商品价格、总价及客户剩余金额
    DM->Customer_Table->FieldValues["Money"] =
                    DM->Customer_Table->FieldValues["Money"] +
                    (double)TotalPriceBuf;
    TotalPriceBuf = TotalPriceBuf –
                    DM->SellDetail_TableTotal_Price->Value;
    DM->SellDetail_TableTotal_Price->Value = DM->Food_Count->Value *
                    DM->Food_Price->Value;
    TotalPriceBuf = TotalPriceBuf +
                    DM->SellDetail_TableTotal_Price->Value;
    DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
    DM->Customer_Table->FieldValues["Money"] =
                    DM->Customer_Table->FieldValues["Money"] –
                    (double)TotalPriceBuf;
    DM->Customer_Table->FieldValues["Money"] =
                    DM->Customer_Table->FieldValues["Money"] +
                    (double)TotalPriceBuf;
    TotalPriceBuf = TotalPriceBuf –
                    DM->SellDetail_TableTotal_Price->Value;
    DM->SellDetail_TableTotal_Price->Value = DM->Food_Count->Value *
                    DM->Food_Price->Value;
    TotalPriceBuf = TotalPriceBuf +
                    DM->SellDetail_TableTotal_Price->Value;
    DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
    DM->Customer_Table->FieldValues["Money"] =
                    DM->Customer_Table->FieldValues["Money"] –
                    (double)TotalPriceBuf;
    //价格计算结束
}
else if( Key == '.' && !FoodCount.Pos('.') )
{  //当用户输入小数点时，判断是否已经包含小数点，已包含则忽略输入
    FoodCount = FoodCount + '.';
}
else if( Key == VK_ESCAPE )
{  //用户按下 ESC 键，同等候商品序号的操作
    DM->SellDetail_Table->Cancel();
    DM->SellDetail_Table->Last();
    for( ; !DM->SellDetail_Table->Bof ;DM->SellDetail_Table->Delete());
```

```
    DM->Sell_Table->Cancel();
    DM->Customer_Table->Cancel();
    FoodNo = "";
    Status = _WAIT_CARD;
}
else if( Key == '+' && DM->SellDetail_TableTotal_Price->Value != 0 )
{  //用户按下加号键，同等候商品序号的操作
    DM->SellDetail_Table->Post();
    DM->SellDetail_Table->Append();
    DM->SellDetail_Table->FieldValues["Sells_No"] =
                    DM->Sell_TableSells_No->Value;
    FoodNo = "";
    Status = _WAIT_FOOD_NO;
}
else if( Key == VK_RETURN )
{  //用户按下回车键，结束交易，同等候商品序号的操作
    DM->SellDetail_Table->Post();
    DM->Sell_Table->Post();
    DM->Customer_Table->Post();
    SerialNo = SerialNo + 1;
    Status = _WAIT_CARD;
    FoodNo = "";
}
else if( Key == VK_BACK || Key == '-')
{  //用户按下退格键或者减号，同等候商品序号的操作
    if(DM-> SellDetail_Table->Modified )
    {
        DM->Customer_Table->FieldValues["Money"] =
                    DM->Customer_Table->FieldValues["Money"] +
                    (double)TotalPriceBuf;
        TotalPriceBuf = TotalPriceBuf –
                    DM->SellDetail_TableTotal_Price->Value;
        DM->SellDetail_Table->Cancel();
        DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
        DM->Customer_Table->FieldValues["Money"] =
                    DM->Customer_Table->FieldValues["Money"] –
                    (double)TotalPriceBuf;
    }
    else if( DM->SellDetail_Table->RecordCount )
```

```
        {
          TotalPriceBuf = TotalPriceBuf –
                        DM->SellDetail_TableTotal_Price->Value;
          DM->Customer_Table->FieldValues["Money"] =
                        DM->Customer_Table->FieldValues["Money"] +
                        (double)TotalPriceBuf;
          DM->SellDetail_Table->Delete();
          DM->Sell_TableTotal_Price->Value = TotalPriceBuf;
          DM->Customer_Table->FieldValues["Money"] =
                        DM->Customer_Table->FieldValues["Money"] –
                        (double)TotalPriceBuf;
        }
        DM->SellDetail_Table->Append();
        DM->Sells_No->Value = DM->Sell_TableSells_No->Value;
        FoodNo = "";
        Status = _WAIT_FOOD_NO;
    }
}
//---------------------------------------------------------------------
void __fastcall TForm1::SetLogIn( bool Value )
{   //IsLogin 的设置函数
    if( FIsLogIn == Value )
        return;
    if( Value )
    {   //更新界面
        SpeedButton2->Enabled = true;
        SpeedButton6->Enabled = true;
        SpeedButton7->Enabled = true;
        N3->Enabled = true;
        N4->Enabled = true;
        N12->Enabled = true;
        SpeedButton1->Caption = "退出登录";
        N2->Caption = "退出登录";
    }
    else
    {   //若处于销售状态，则退出销售状态，并更新界面，IsSelling 受到 IsLogin 的约束
        if( IsSelling )
           IsSelling = false;
        SpeedButton2->Enabled = false;
```

```
        SpeedButton6->Enabled = false;
        SpeedButton7->Enabled = false;
        SpeedButton1->Caption = "用户登录";
        N3->Enabled = false;
        N4->Enabled = false;
        N12->Enabled = false;
        N2->Caption = "用户登录";
    }
    FIsLogIn = Value;
}
//---------------------------------------------------------------------
void __fastcall TForm1::SetSelling( bool Value )
{   //IsSelling 的设置函数
    FIsSelling = Value;
    if( Value )
    {    //设置为销售状态的操作
      if( !IsLogIn )
      {  //未登录时，不允许处于销售状态，在这个程序中，这种情况不会发生，但是
        FIsSelling = false;  //IsSelling 属性是公开，这种约束的完整性是必须考虑的
      }
      else   //若已经登录，销售状态则显示 Panel1、Panel2，打开相应的数据集
      {     //并设置 Application 的 OnMessage 事件，同时将子状态设置为
        Panel1->Visible = true;              //等候刷卡状态
        Panel2->Visible = true;
        DM->Sell_Table->Open();
        DM->Customer_Table->Open();
        DM->Food_Table->Open();
        DM->SellDetail_Table->Open();
        SpeedButton5->Visible = true;
        Status = _WAIT_CARD;  //设置为等候刷卡状态
        Application->OnMessage = MyKeyProc;
        for(DM->Sell_Table->First();!DM->Sell_Table->Eof;
                          DM->Sell_Table->Next())
        {                 //取数据库中的流水号
          if( DM->Sell_Table->FieldValues["Sells_No"] > SerialNo )
            SerialNo = DM->Sell_Table->FieldValues["Sells_No"];
        }
      }
    }
```

```
    else   //设置为非销售状态，关闭相应的数据集，
    {      //恢复 OnMessage 事件，更新界面
        Panel1->Visible = false;      //在 IsSelling 属性中并不对数据集进行判断
        Panel2->Visible = false;      //比如是否更改，是否可以保存等条件是由数
        DM->Sell_Table->Close();      //据集自行判断和设置的
        DM->Customer_Table->Close();
        DM->Food_Table->Close();
        DM->SellDetail_Table->Close();
        SpeedButton5->Visible = false;
        Application->OnMessage = NULL;
    }
}
//---------------------------------------------------------------------
void __fastcall TForm1::SetStatus( int Value )
{   //设置销售状态下的子状态 Status 的函数，完整的设置函数应该包含
    //对上一级约束状态 IsSelling 的检测和设置，在这个例子中，实际
    //上这种约束关系已经包含在前面的代码中，因为对子状态的应用，
    //只存在于键盘处理函数和子处理函数中，只有 Selling 为销售状态时
    //OnMessage 事件才会调用键盘处理函数。
    FStatus = Value;
    switch ( Value )
    {
      case _WAIT_FOOD_NO: ;                    //设置为等候商品序号状态
        Edit1->Text = "请输入餐号...更改相应餐号数量请按*键";
        DBText1->Visible = true;
        DBText2->Visible = true;
        DBImage1->DataField ="Photo";
        DBGrid2->DataSource = DM->Sell_Detail;
        SubKeyProc = WaitFoodNoProc;           //设置键盘子处理函数
        break;
      case _WAIT_FOOD_COUNT: ;                 //设置为等候商品数量状态
        Edit1->Text = "请输入数量...";
        SubKeyProc = WaitFoodCountProc;        //设置键盘子处理函数
        break;
      case _WAIT_CARD:                         //设置为等候刷卡状态
        Edit1->Text = "等候刷卡...";
        StatusBar1->Panels->Items[1]->Text = "";
        StatusBar1->Panels->Items[3]->Text = "";
        StatusBar1->Panels->Items[5]->Text = "";
```

```
        DBText1->Visible = false;
        DBText2->Visible = false;
        DBImage1->DataField = "";
        DBGrid2->DataSource = NULL;
        SubKeyProc = WaitCardKeyProc;        //设置键盘子处理函数
        break;
      default :
        Edit1->Text = "";
    }
}
```

在这个 Sells 的例子中，关键的几个函数和属性设置都包含在这个单元中，程序的逻辑结构也正是在这部分代码中体现出来的，读者需要结合例程的运行状况进行了解。

2.3.3 Pass 密码验证单元

Pass 单元是密码验证单元，比较简单，其 Form 的形状如同前面图中所示，头文件也非常简单，这里只给出实现文件，如下：

```
#include <vcl.h>
#pragma hdrstop
#include "Pass.h"
#include "Main.h"
#include "DataBase.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
//---------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm2::BitBtn2Click(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------
void __fastcall TForm2::BitBtn1Click(TObject *Sender)
{
```

```
unsigned char Temp[256];                    //密码的缓存
TLocateOptions Opts;
Variant locvalues;                          //Local 方法的匹配条件
Opts.Clear();
Opts << loCaseInsensitive;
locvalues = Variant(Edit1->Text);           //Local 方法的匹配参数
DM->SysTemUser->Open();
if( DM->SysTemUser->RecordCount == 0)
{  //如果操作员的数据库为空，通常是第一次运行，按照确省的用户和密码验证
   if( Edit1->Text == "ADM" && Edit2->Text == "ADM" )
   {   //确省的用户和密码均为"AMD"
       Form1->IsLogIn = true;
       Close();
       Edit1->Text = "";
       Edit2->Text = "";
   }
   return;     //第一次运行则忽略后面的代码
}
if( DM->SysTemUser->Locate( "SysUser_ID",locvalues,Opts ))
{  //先按照用户名进行匹配，如果招到该用户，再进行密码的匹配验证
   int nBytes;
   TBlobStream *Stream = new TBlobStream(DM->SysTemUserPassWord,
                                         bmRead);
   Stream->Read(Temp,1);      //读取密码长度的标志，Blob 字段的第一个字节
   nBytes = Temp[0];          //以二进制保存着整个密码或加密存放的密码长度
   Stream->Read(Temp,nBytes + 1 );   //读取密码或加密数据
   //此处可对 Temp 插入相应的解密算法，完成密码的加密保存
   //密码的解密需要和用户设置密码时的加密算法对应。
   if( Edit2->Text == AnsiString( (char*)Temp ) )
   {
       Form1->IsLogIn = true;     //设置程序为登录状态
       Close();                   //关闭登录窗口时清空用户和密码
       Edit1->Text = "";
       Edit2->Text = "";
   }
   else
   {
       Edit2->Text = "";          //密码不匹配时，清空密码输入；
   }
```

```
        delete Stream;                    //销毁临时的BlobStream
    }
    else                                  //当用户名不匹配时，清空所有输入
    {
        Edit1->Text = "";
        Edit2->Text = "";
    }
    DM->SysTemUser->Close();              //登录成功后关闭相应数据集
}
//---------------------------------------------------------------------
void __fastcall TForm2::EditKeyPress(TObject *Sender, char &Key)
{   //用户名和密码的Edit共用一个KeyPress事件句柄。
    if( Key == 13 )                       //回车按照登录操作，0D13=VK_RETRUN
        BitBtn1Click( Sender );
    if( Key == VK_ESCAPE )                //ESC按照取消操作，VK_ESCAPE=0D28
        BitBtn2Click( Sender );
}
//---------------------------------------------------------------------
void __fastcall TForm2::FormShow(TObject *Sender)
{
    Edit1->SetFocus();                    //登录窗口显示后Edit1获得焦点
}
```

2.3.4 Path 路径设置单元

Path 单元是当程序运行时数据库设置不正确,供用户设置 Paradox 数据库路径的窗体，其窗体也是比较简单，头文件也非常简单，同样我们只给出实现部分：

```
#include <vcl.h>
#pragma hdrstop
#include "Path.h"
#include "Main.h"
#include "DataBase.h"
#pragma package(smart_init)
#pragma link "cdiroutl"
#pragma resource "*.dfm"
AnsiString Tables[5] =
{
"Customer",                 //这是数据库中所所使用的5个表集的名称；
```

```
"Food",                  //当用户设置了数据库时，需要逐一验证数据库的有效性
"Sell",                  //更可靠的验证方法是对每一个数据集的字段以及每个字
"Sell_Detail",           //段的数据类型逐一进行验证
"System_User"
};

TForm5 *Form5;
//---------------------------------------------------------------------
__fastcall TForm5::TForm5(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm5::BitBtn2Click(TObject *Sender)
{   //若程序提示设置数据库，而用户取消时，程序将关闭
    if( Reg )
      delete Reg ;
    Form1->Visible = false;      //以上三行代码可以省去，当程序关闭时，所有
    Close();                     //分配的资源均被释放，但是如果该方法不是将
    Form1->Close();              //程序关闭，则不一定可以省略
}
//---------------------------------------------------------------------
void __fastcall TForm5::BitBtn1Click(TObject *Sender)
{
    TTable *TempTable = new TTable(NULL);  //用于测试数据库的临时表。
    TempTable->DatabaseName = DM->Sell->DatabaseName;
    DM->Sell->Directory = DirectoryListBox1->Directory;  //指定的数据库路径;
    for(int i = 0;i < 5;i ++ )
    {
      TempTable->TableName = Tables[i];
      try
      {
        TempTable->Open();       //进行数据集存在性的验证,可以在 Open 和 Close 方
        TempTable->Close();      //法之间加入对数据集字段和类型的验证
      }
      catch(...)
      {                          //当数据集不存在时，报告用户
       Application->MessageBox("指定的路径非法！","错误！",0);
       return;
```

```
      }
    }
    if( !Reg )
      Reg = new TRegistry;          //检验是否已经打开注册表
    try                             //设置的数据库有效时将信息写入注册表
    {
      Reg->RootKey = HKEY_LOCAL_MACHINE;
      if ( Reg->OpenKey("\\Software\\Lewolf\\DataBase",true) )
      {
        Reg->WriteString("DataPath",
                         DirectoryListBox1->Directory);
        Reg->CloseKey();            //关闭注册表
      }
      else                          //无法写入注册表时，提醒用户并关闭程序
      {
        Application->MessageBox( "注册表出现错误，程序即将关闭","错误",0);
        Close();
        Form1->Close();
      }
    }
    catch(...)                      //出现异常时，关闭程序
    {
      Application->MessageBox("程序出现错误\n\n即将退出","错误",0);
      Close();
      Form1->Close();
    }
    if( Reg )                       //释放分配的资源
      delete Reg;
    delete TempTable;
    Close();
}
```

2.3.5 Splash 启动画面单元

Splash 是在启动程序的时候显示的启动画面，这个单元也是十分简单，Form 不是在程序主文件中自动创建运行的，而是在 Form1 的构造函数中使用 new 关键字创建的，这样可以是启动画面在主窗体 Form1 显示之前显示出来。这个 Form 中包含一个 Image 用于存放图片，另外有一个动画，是从文件载入的，这个单元读者参考源代码文件。

2.3.6 DataBase 数据库模块单元

DataBase 是该程序的数据库模块，这个模块中定义了程序所用到的数据库组件和数据集组件，并为部分数据集组件定义了 TField 字段，模块的结构如图 7.9：

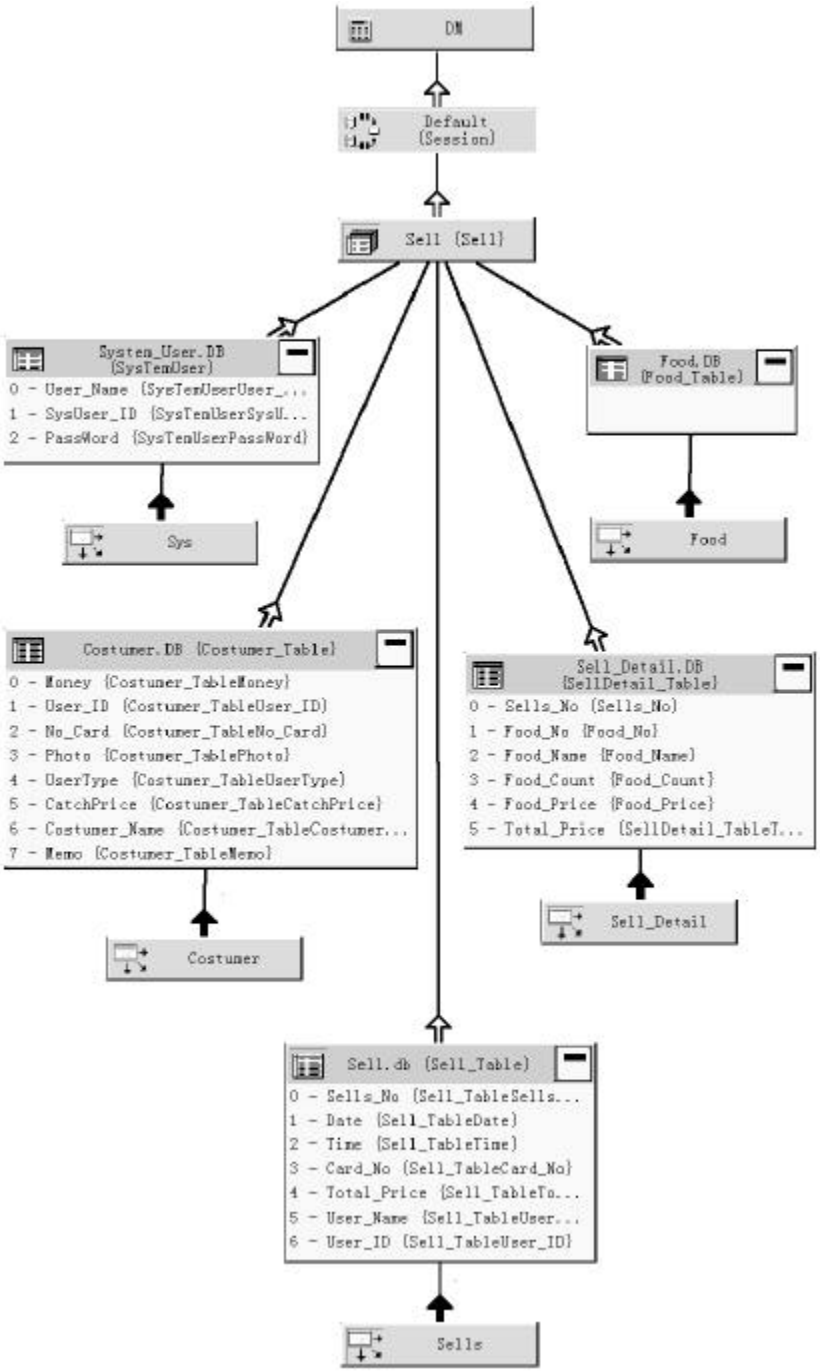


图 7.9 DM 单元的结构

头文件如下：

```
#ifndef DataBaseH
#define DataBaseH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Db.hpp>
#include <DBTables.hpp>
#include "registry.hpp"
#include <DB.hpp>
//---------------------------------------------------------------------
class TDM : public TDataModule
{
__published:
        TTable *SellDetail_Table;
        TIntegerField *Sells_No;
        TSmallintField *Food_No;
        TStringField *Food_Name;
        TFloatField *Food_Count;
        TCurrencyField *Food_Price;
        TCurrencyField *SellDetail_TableTotal_Price;
        TDataSource *Sell_Detail;
        TTable *Food_Table;
        TDataSource *Food;
        TTable *Customer_Table;
        TDataSource *Customer;
        TTable *Sell_Table;
        TIntegerField *Sell_TableSells_No;
        TDateField *Sell_TableDate;
        TTimeField *Sell_TableTime;
        TStringField *Sell_TableCard_No;
        TCurrencyField *Sell_TableTotal_Price;
        TStringField *Sell_TableUser_Name;
        TSmallintField *Sell_TableUser_ID;
        TDataSource *Sells;
        TDatabase *Sell;
```

```
        TTable *SysTemUser;
        TStringField *SysTemUserUser_Name;
        TStringField *SysTemUserSysUser_ID;
        TBlobField *SysTemUserPassWord;
        TDataSource *Sys;
        TCurrencyField *Customer_TableMoney;
        TSmallintField *Customer_TableUser_ID;
        TStringField *Customer_TableNo_Card;
        TGraphicField *Customer_TablePhoto;
        TStringField *Customer_TableUserType;
        TBooleanField *Customer_TableCatchPrice;
        TStringField *Customer_TableCustomer_Name;
        TMemoField *Customer_TableMemo;
        void __fastcall MD(TObject *Sender);
        void __fastcall Sell_TableSells_NoChange(TField *Sender);
        void __fastcall Food_CountChange(TField *Sender);
        void __fastcall Food_NoChange(TField *Sender);
        void __fastcall Sell_TableUser_NameChange(TField *Sender);
        void __fastcall SellDetail_TableAfterEdit(TDataSet *DataSet);
        void __fastcall SellDetail_TableAfterInsert(TDataSet *DataSet);
public:
        bool    SellDetailCanPost;
        inline __fastcall TDM(TComponent* Owner): TDataModule(Owner){};
};
extern PACKAGE TDM *DM;
//---------------------------------------------------------------------------
#endif
```

实现部分代码如下：

```
#include  <vcl.h>
#pragma   hdrstop
#include "DataBase.h"
#include "Main.h"
#include "Path.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TDM *DM;
```

```
//---------------------------------------------------------------------
void __fastcall TDM::MD(TObject *Sender)
{  //TDataModule 创建时，根据注册表信息设置数据库 Sell 的参数
   TRegistry *Reg = new TRegistry;
   try
   {
     Reg->RootKey = HKEY_LOCAL_MACHINE;
     if ( Reg->OpenKey("\\Software\\Lewolf\\DataBase",false) )
     {  //若找到相应的注册表键，则按照注册表设置 Sell 的参数
        //使用 Sell->Params 和 Sell->Directory 可以获得相同的效果，
        //但 Params 要求在数据库打开之前设置,并且 Params 中必须有"PATH"字段
        //Sell->Params->Values["PATH"] = Reg->ReadString("DataPath");
        Sell->Connected = true;
        Sell->Directory = Reg->ReadString("DataPath");
        Reg->CloseKey();
     }
     else   //如果注册表中没有相应的键值，则打开数据库设置对话框 Form5
        Form5->ShowModal();
   }
   catch(...)
   {
      Application->MessageBox("数据库出错\n\n 即将退出\n\n 请与软件提供商联系",
                              "错误",0);
      delete Reg;
      Form1->Close();
   }
   delete Reg;
}
//---------------------------------------------------------------------
void __fastcall TDM::Sell_TableSells_NoChange(TField *Sender)
{  //当销售订单改变是，更新销售列表的过滤器，也可以使用 Master-Detail 关系实现
   SellDetail_Table->Filter = "Sells_No=" +
                       IntToStr( Sell_TableSells_No->Value );
}
//---------------------------------------------------------------------
void __fastcall TDM::Food_CountChange(TField *Sender)
{  //将销售列表中的当前商品数量更新显示到 StatusBar
   Form1->StatusBar1->Panels->Items[5]->Text =
                     FloatToStr( Food_Count->Value );
```

```
}
//---------------------------------------------------------------------
void __fastcall TDM::Food_NoChange(TField *Sender)
{   //将销售列表中的当前商品序号更新显示到StatusBar
    Form1->StatusBar1->Panels->Items[3]->Text =
                    IntToStr( DM->Food_No->Value );
}
//---------------------------------------------------------------------
void __fastcall TDM::Sell_TableUser_NameChange(TField *Sender)
{   //将刷卡后用户信息更新显示到StatusBar
    Form1->StatusBar1->Panels->Items[1]->Text =
                    Customer_Table->FieldValues["Customer_Name"];
}
//---------------------------------------------------------------------
void __fastcall TDM::SellDetail_TableAfterEdit(TDataSet *DataSet)
{  //禁止用户使用方向键移动销售列表数据集的游标，避免移动游标时将记录Post
    SellDetailCanPost = false;
}
//---------------------------------------------------------------------
void __fastcall TDM::SellDetail_TableAfterInsert(TDataSet *DataSet)
{   //禁止用户使用方向键移动销售列表数据集的游标，避免移动游标时将记录Post
    SellDetailCanPost = false;
}
//---------------------------------------------------------------------
```

2.4 总结

在这个例子 Sells 中，我们使用了属性和事件来完成不同状态下，界面显示和代码执行条件的转换，在这个例子中也可以看出应用程序中的属性和组件中的属性之间的差别，应用程序中的事件和组件中的事件也有所不同，但这种程序的设计思想却是从组件的设计思想发展而来。

在这个例子中，每个对象之间的关系相对较为简单一些，主要的应用都在 Main 单元中,程序也是比较简单的,但是程序的状态之间的约束关系却是比较明确的,首先是 Login 状态，这是整个程序运行的首要条件，也是对程序的其它状态具有约束力的状态，然后是在登录状态下的几个运行状态，在销售状态下又具有三种不同时刻的销售状态，这些状态之间具有一些不同的约束关系，以及在不同的状态下，程序有着不同的执行动作，我们正是利用了面向状态技术中的属性以及事件来将这些相对复杂的关系简约化，使整个程序在修改维护上要便利的多。不难想像，对于复杂的程序，这种对象相互之间的约束关系也将会复杂的多，如果仅仅采用常规的设计方法，自然也是可以实现的，但是在实现的过程中将会遇到更多的麻烦，对维护和修改也是极为不利。

接下来我们要看的是一个关于图形编辑方面的例子，在这个例子中，所实现的功能相对要多一些，各个对象之间的关系也要比 Sells 中的复杂一些，但是使用了 VCL 的组件之后，加上在应用程序中使用了面向状态的设计方法，整个程序实现起来要简单的多。

3 示范程序 Drawing

这个例子是在 C++ Builder 6 环境下设计的，也是作者专门为本书而编写的例程，由于程序中使用了 C++ Builder 6 中新增加的组件，因此这部分代码不能兼容以前版本的，程序运行如图 7.10 所示：

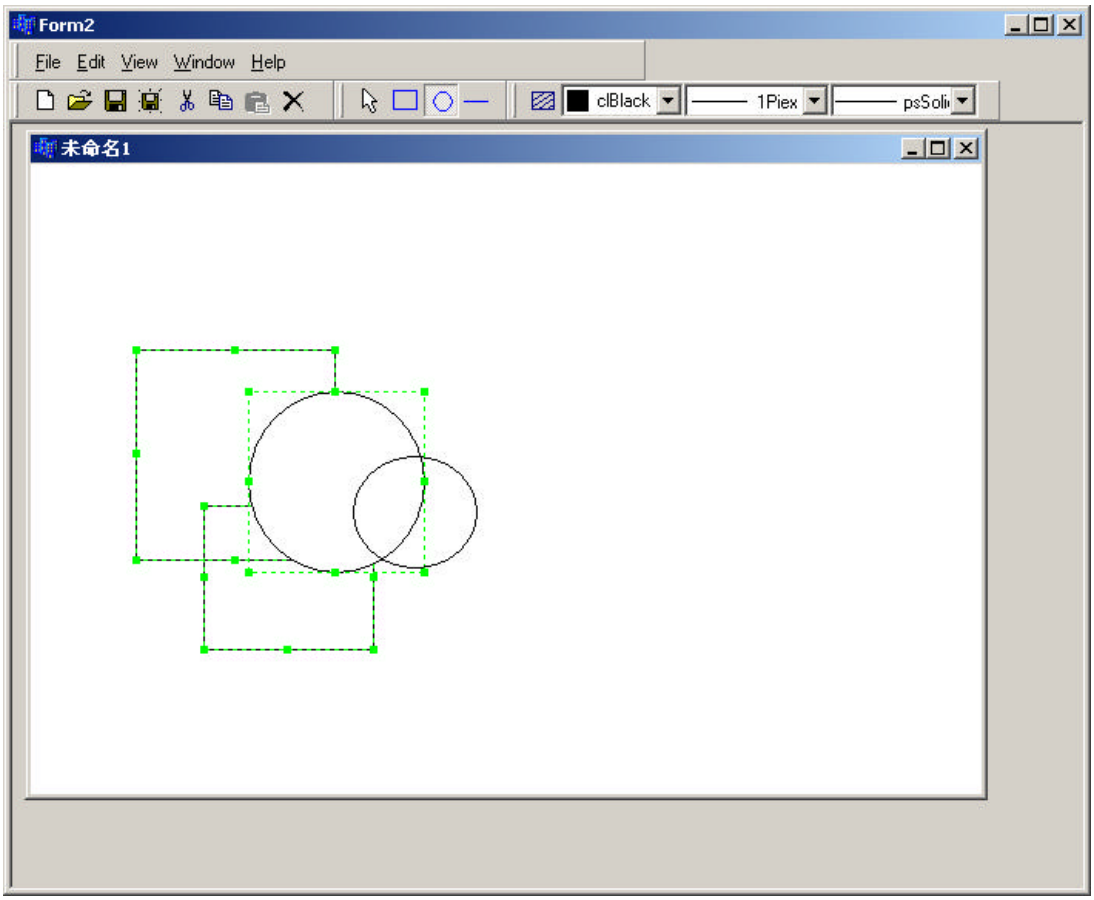


图 7.10 Drawing 运行后的界面

这是一个简单的多文档（MDI）应用程序，其中 MDIChild 窗口为一个简单的矢量绘图窗口，例程提供了三种绘图工具，矩形、椭圆和线段，其中线段涉及到更复杂的派生对象，为了节省篇幅，例程中没有给出代码，但其实现的方法和矩形以及椭圆是基本类同的。

程序总共有三大功能：

- 编辑功能，可以分作整体图形编辑和个体编辑；
- 存储功能，对文件的保存和读取的功能；

- 数据交换，使用 Windows 剪贴板进行数据交换，可以实现拷贝、粘贴等功能。

在这个程序中，涉及到的对象有四种，在本程序中实际上就是四个对象类：

- 一是图形对象，为每一个绘制到窗口上的图形，以矢量数据的方式记录；
- MDIChild 对象，是为每一个绘制的图形集合定义的显示窗体，是和存储文件对应的；
- 针对我们的图形格式而定义的剪贴板对象，来完成特定格式的数据交换；
- MDI 主窗体，是整个应用程序的框架。

在这个程序的设计过程中，首先是建立整个应用程序的框架结构，然后在建立每一个所需要的数据类型和结构，最后完成每一个功能的实现。我们也按照这个顺序来逐步介绍这个例子。

3.1 MDI 主窗体对象 TForm2 类

MDI 主窗体是整个应用程序的框架，工具条和菜单等都在这个单元内，这个类中基本都是由 IDE 生成的可视化组件对象，只有一个用户定义的成员 ClipBoardFormat，用来注册剪贴板中特定的数据格式，之外就是一个 ActiveDraw 属性，纯粹是为了使用方便，不需要频繁的进行数据类型的转换。

这个单元中的方法都属于事件的处理句柄，而且都比较简单，因此这个单元的代码我们将不作详细的介绍，读者参考源码和注解，我们仅对功能和对象之间的关系作个介绍。

在 ToolMain 单元中，除了几个运行时不可见的组件之外，只有一个 ControlBar 是顶层控件，这个 ControlBar 上一共有四个 ToolBar，分别是主菜单、常用按钮、绘图工具和图形属性的工具条，声明文件如下：

```
#ifndef ToolMainH
#define ToolMainH
//----------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <ToolWin.hpp>
#include <ImgList.hpp>
#include <Menus.hpp>
#include "Main.h"
#include "ShapeClass.h"
#include <ActnCtrls.hpp>
#include <ActnMan.hpp>
#include <ActnMenus.hpp>
```

```
#include <Buttons.hpp>
#include <Dialogs.hpp>

class TForm2 : public TForm
{
__published:
        TControlBar *ControlBar1;          //ControlBar 对象
        TImageList *ImageList1;            //为绘图工具提供位图
        TImageList *ImageList2;            //为线型和线宽列表框提供位图
        TImageList *ImageList3;            //为常用工具条提供按钮位图
        TToolBar *ToolBar1;                //主菜单的工具条，包含一下菜单
        TMainMenu *MainMenu1;              //主菜单对象
        TMenuItem *File;                   //主菜单中的 File 菜单项
        TMenuItem *New;                    //File 菜单中的 New 新建菜单项
        TMenuItem *Open;                   //File 菜单中的 Open 打开文件菜单项
        TMenuItem *N1;                     //分割符菜单项
        TMenuItem *Save1;                  //File 中的 Save 保存菜单项
        TMenuItem *SaveAs1;                //File 中的 SaveAs 另存为菜单项
        TMenuItem *N2;                     //分割符菜单项
        TMenuItem *Close1;                 //File 中的 Close 关闭菜单项
        TMenuItem *Exit1;                  //File 中的 Exit 退出菜单项
        TMenuItem *Edit1;                  //主菜单的 Edit 菜单项
        TMenuItem *Cut1;                   //Edit 中的 Cut 剪切菜单项
        TMenuItem *Copy1;                  //Edit 中的 Copy 复制菜单项
        TMenuItem *Paste1;                 //Edit 中的 Paste 粘贴菜单项
        TMenuItem *Delete1;                //Edit 中的 Delete 删除菜单项
        TMenuItem *View1;                  //主菜单中的 View 菜单项
        TMenuItem *MemuBar2;               //View 中的 MenuBar 菜单项
        TMenuItem *EditTools2;             //View 中的 EditTool 菜单项
        TMenuItem *DrawTools2;             //View 中的 DrawTools 菜单项
        TMenuItem *PropertyTools2;         //View 中的 PropertyTool 菜单项
        TMenuItem *Window1;                //主菜单中的 Window 菜单
        TMenuItem *Tite1;                  //Window 中的 Tite 菜单项
        TMenuItem *Cascade1;               //Window 中的 Cascade 菜单项
        TMenuItem *Help1;                  //主菜单的 Help 菜单项
        TMenuItem *About1;                 //Help 的 About 菜单项
        //绘图工具的 ToolBar，该工具条包含以下按钮
        TToolBar *ToolBar2;                //绘图工具的 ToolBar
        TToolButton *SelectButton;         //Select 工具
```

```
    TToolButton *RectButton;          //绘制 Rectangle 的工具
    TToolButton *CircleButton;        //绘制椭圆的工具
    TToolButton *LineButton;          //绘制线段的工具
    //常用工具的 ToolBar，该工具条包含一下按钮
    TToolBar *ToolBar3;               //常用工具的 ToolBar
    TToolButton *ToolButton3;         //新建文件按钮
    TToolButton *ToolButton4;         //打开文件按钮
    TToolButton *SaveButton;          //保存按钮
    TToolButton *SaveAsButton;        //另存为按钮
    TToolButton *CutButton;           //剪切按钮
    TToolButton *CopyButton;          //复制按钮
    TToolButton *PasteButton;         //粘贴按钮
    TToolButton *DeletButton;         //删除按钮
    //图形属性的 ToolBar，该工具条包含一下按钮
    TToolBar *ToolBar4;               //图形属性工具条
    TToolButton *FillButton;          //填充格式按钮
    TColorBox *ColorBox1;             //图形颜色下拉框
    TComboBoxEx *ComboBoxEx1;         //线宽下拉框
    TComboBoxEx *ComboBoxEx2;         //线型下拉框
    //ControlBar 的弹出菜单，基本同 View 菜单，功能也同 View 菜单项
    TPopupMenu *ToolMenu;             //
    TMenuItem *MemuBar1;              //ToolMenu 菜单项，检查是否显示主菜单
    TMenuItem *EditTools1;            //决定是否显示常用工具条
    TMenuItem *DrawTools1;            //决定是否显示绘图工具条
    TMenuItem *PropertyTools1;        //决定是否显示图形属性工具条
  //DMIChild 窗体的弹出菜单，在 Edit 菜单项的内容中增加了改变图形叠放层次的菜单项
    TPopupMenu *Popup;                //
    TMenuItem *Cut2;                  //用于剪切
    TMenuItem *Copy2;                 //用于复制
    TMenuItem *Paste2;                //用于拷贝
    TMenuItem *Delete2;               //用于删除
    TMenuItem *N5;                    //分割符菜单项
    TMenuItem *MenuItem1;             //图形上移一层，未使用
    TMenuItem *MenuItem2;             //图形下移一层，未使用
    TMenuItem *N3;                    //将所选的图形移到顶层
    TMenuItem *N4;                    //将所选的图形移到底层
    TOpenDialog *OpenDialog1;         //打开文件对话框
    TSaveDialog *SaveDialog1;         //保存文件对话框
    void __fastcall RectButtonClick(TObject *Sender);
```

```
        void __fastcall CircleButtonClick(TObject *Sender);
        void __fastcall SelectButtonClick(TObject *Sender);
        void __fastcall ComboBoxEx1Change(TObject *Sender);
        void __fastcall ColorBox1Change(TObject *Sender);
        void __fastcall ComboBoxEx2Change(TObject *Sender);
        void __fastcall Cascade1Click(TObject *Sender);
        void __fastcall Title1Click(TObject *Sender);
        void __fastcall NewClick(TObject *Sender);    //新建文件按钮的句柄
        void __fastcall ToolMenuPopup(TObject *Sender);
        void __fastcall MemuBar1Click(TObject *Sender);
        void __fastcall EditTools1Click(TObject *Sender);
        void __fastcall DrawTools1Click(TObject *Sender);
        void __fastcall PropertyTools1Click(TObject *Sender);
        void __fastcall DeletButtonClick(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall CopyButtonClick(TObject *Sender);
        void __fastcall PasteButtonClick(TObject *Sender);
        void __fastcall CutButtonClick(TObject *Sender);
        void __fastcall PopupPopup(TObject *Sender);
        void __fastcall FillButtonClick(TObject *Sender);
        void __fastcall N3Click(TObject *Sender);
        void __fastcall N4Click(TObject *Sender);
        void __fastcall SaveClick(TObject *Sender);
        void __fastcall SaveAsClick(TObject *Sender);
        void __fastcall OpenClick(TObject *Sender);    //打开按钮的句柄
        void __fastcall CloseClick(TObject *Sender);
        void __fastcall ExitClick(TObject *Sender);
protected:
        void __fastcall SetActiveDraw(TForm1* Value); //ActiveDraw的写方法
        TForm1 * __fastcall GetActiveDraw(void);      //ActiveDraw的读方法
public:
        WORD    ClipBoardFormat;                      //向剪贴板注册数据格式
        __property TForm1* ActiveDraw={read=GetActiveDraw,
                                       write=SetActiveDraw};
        __fastcall TForm2(TComponent* Owner);
};
extern PACKAGE TForm2 *Form2;
#endif
```

MenuBar：

主菜单中包含 5 个菜单项，分别是 File、Edit、View、Window 和 Help，其结构如下图：

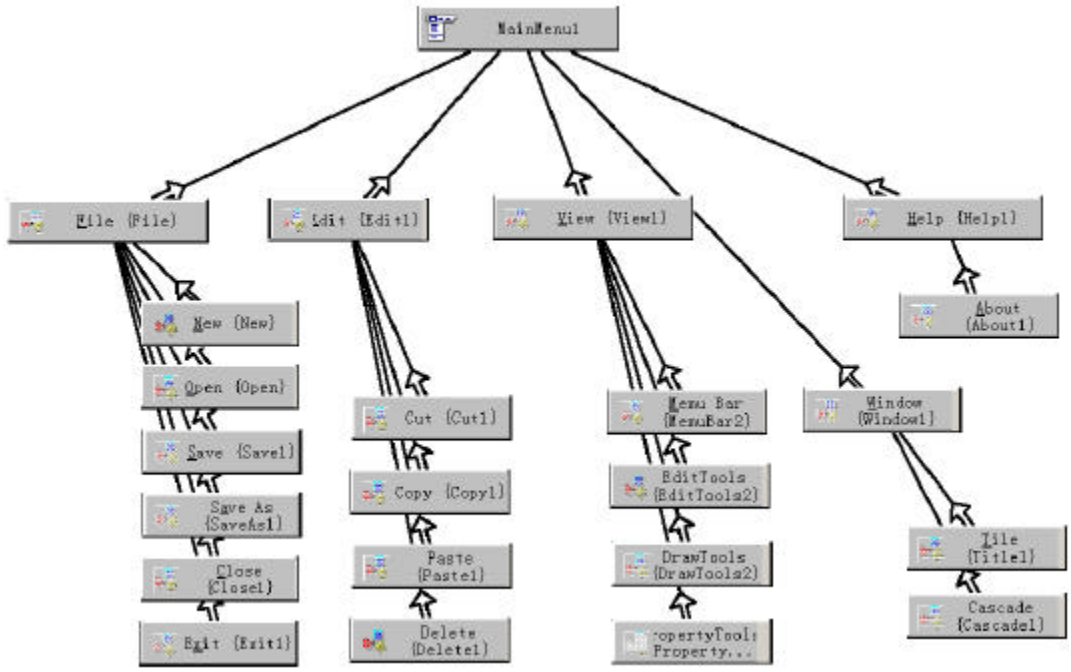


图 7.11 MainMenu 的结构

File 菜单中包含了 New、Open、Save 等常用的菜单项，其功能不需要进一步介绍，其中的 Save 菜单项是和当前活动的 MDIChild 窗体的编辑状态有关，每一个 MDIChild 对象中都有一个记录编辑状态的属性 Modified，文件保存后其 Modified 属性变为 false，当任何的编辑动作执行后，Modified 属性变为 true，Save 菜单项的 Enabled 属性和活动 MDIChild 的 Modified 属性一致。

Edit 菜单中包含一般的编辑命令：Cut、Copy、Paste、Delete 等，其中 Cut、Copy、Delete 菜单项的 Enabled 和当前活动 MDIChild 中的被选择图形有关，只有被选择图形的数量不为 0 的时候，这几个菜单的 Enabled 属性为 true，否则为 false，这几个菜单的操作对象也是当前活动 MDIChild 所选中的图形，Paste 菜单项的 Enabled 和当前剪贴板中的内容有关，只有剪贴板中包含程序能够识别的数据格式时，Paste 的 Enabled 为 true，否则为 false。

View 菜单中的几个菜单项都是 Checked 类型的，他们的 Checked 属性分别和主窗体的四个 ToolBar 的 Visible 属性一致，这就类似 Word 等程序中的工具条，当在工具条上点击右键的时候，弹出的快捷菜单 ToolMenu 可以选择显示的工具条，和 View 菜单的内容相同，因此对 ToolMenu 我们不再介绍。

Window 菜单是多文档界面中基本的两个窗体排列方法 Tite、Cascade；Help 菜单中只有一个 About 菜单项。

绘图工具条：

绘图工具条中包含四个按钮，分别是 Select 工具、矩形绘图工具、椭圆绘图工具和线段绘图工具，这四个按钮为相互排斥的，不允许自行弹起，同时只有一个按钮的 Down 属性为 true，这四个按钮组成一个组和当前活动的 MDIChild 窗体中的 Shape 属性相对应，当活动 MDIChild 改变时，工具栏中的按钮状态随之改变，用户点击不同工具时，可以改变活动 MDIChild 的 Shape 属性的状态。Shape 属性记录着用户操作的绘图工具类型，在后面的 Main 单元中我们再详细介绍。

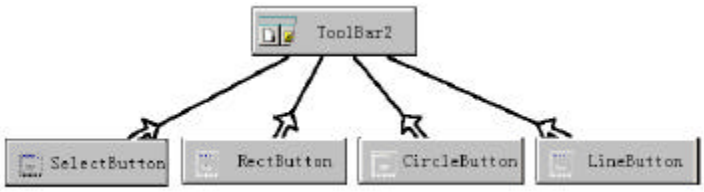


图 7.12 绘图工具 ToolsBar 的结构

常用工具栏：

这个工具条实际上就是一些常用功能的快捷按钮，一共有 8 个按钮：新建文档、打开文档、保存、另存为、删除、剪切、拷贝、粘贴等，其功能和主菜单中相应的菜单项的功能相同，这几个按钮的 Enabled 属性和主菜单中相应的菜单项的 Enabled 属性是一致的，都和当前活动的 MDIChild 的状态以及剪贴板中的内容相关。

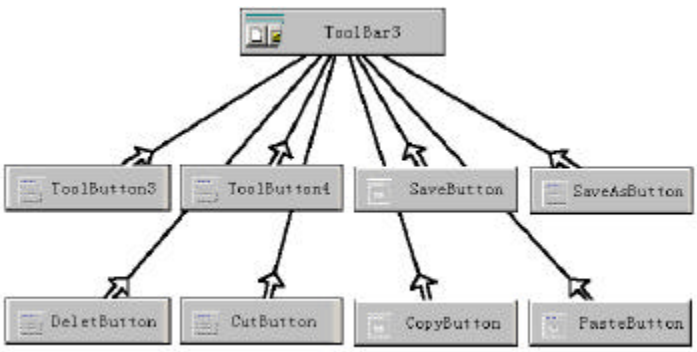


图 7.13 常用工具条的结构

图形属性工具条：

这个工具是一个显示所选择图形的属性的工具条，同时也是一个编辑所选择图形属性的工具条，在我们这个例子中，一个图形对象的属性，指的是图形固有的特征，同时也是程序代码中对象的属性，比如一个矩形的位置、大小、颜色、边框的线宽、线型等。在图形属性的工具条中，包含一个按钮，用于标识和编辑图形填充方式，一个 Color 下拉列表框、一个线宽的下拉列表框和一个线型的下拉列表框，分别用于显示当前活动文档中所选择图形的填充方式、颜色、线宽和线型，当活动文档没有任何图形被选中的时候，这四个控件分别显示活动文档的缺省填充方式、颜色、线宽和线型，每次创建新的图形对象时，均按照缺省的这几个属性来绘制新的图形，通过这四个控件可以改变一个 MDIChild 窗体

中相应的缺省属性；当活动文档中有图形被选中的时候，这四个控件将显示被选中图形的相应属性，通过改变控件的属性可以改变被选择图形的相应属性，当被选择图形为多个，并且它们的颜色、线宽和线型不一致的时候，对应下拉列表框的索引为-1，显示列表中的内容为空。

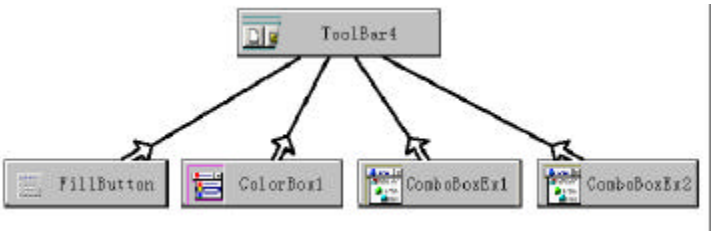


图 7.13 属性工具条的结构

ActiveDraw 属性：

这个属性和 ActiveMDIChild 是一致的，这里只是将类型转换后的 ActiveMDIChild 作为读取方法的返回值，设置方法本身是可以省略的，我们在例子中提供了一个设置方法，这样程序中可以通过属性的设置来改变活动的 MDIChild 窗体。

3.2 MDIChild 对象 Main 单元

这个单元定义了 MDIChild 子窗体的对象类 TForm1，应用程序中可以包含多个 TForm1 对象，每一个 TForm1 对象和一个数据文件对应，同时 TForm1 对象还承担着对图形对象进行整体编辑任务。

Main 单元中除了 TForm1 类之外，还有一个枚举数据类型，用于标识 TForm1 当前所处的绘图工具的状态，和一个用于剪贴板使用的 TDrawClipborad 类，整个声明文件如下：

```
#ifndef MainH
#define MainH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <ToolWin.hpp>
#include <ImgList.hpp>
#include <Buttons.hpp>
#include <Menus.hpp>
#include "Clipbrd.hpp"
#include "ShapeClass.h"
```

```
enum TShapeStyle
{
    _Select,_Rect,_Circle,_Line
};
//定义用于标识当前画布图形类型的枚举类型，画布提供四种
//状态：选择、矩形、椭圆、线

struct TFileHeader       //定义一个顺序存储的文件头标识结构
{                        //在这个结构中至少包含图像数量的信息
    char Sign[24];       //用于存放文件数据的标识
    int  DrawCount;      //Sign 是一个标签，用于区分文件类型的，
};                       //本例中没有使用 Sign，
                         //DrawCount 记录着整个文件中图形对象的数量
//---------------------------------------------------------------------
//本例中，Form1 为画布，是一个 MDI 窗体，所有的图形都绘制在画布上
class TForm1 : public TForm
{
__published:
    //OnMouseDown 事件的处理函数
    void __fastcall DrawMouseDown(TObject *Sender, TMouseButton Button,
                                  TShiftState Shift, int X, int Y);
    //OnMouseMove 事件的处理函数
    void __fastcall DrawMouseMove(TObject *Sender, TShiftState Shift,
                                  int X, int Y);
    //OnMouseUp 事件的处理函数
    void __fastcall DrawMouseUp(TObject *Sender, TMouseButton Button,
                                TShiftState Shift, int X, int Y);
    //OnActive 事件的处理函数
    void __fastcall FormActivate(TObject *Sender);
    //OnClose 事件的处理函数，改变 Action 使 MDI 窗口关闭时一并销毁。
    void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
    //OnCloseQuery 事件的处理函数，对于为保存数据的提示
    void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
private:
    TShapeStyle  FShape;     //Shape 属性的内部存储数据
    bool         FModified;  //Modified 属性的内部存储数据
    int          FScale;     //显示比例的内部存储数据，本例子中没有使用显示比例
protected:
```

```
    void __fastcall SetShape(TShapeStyle Value);      //Shape 的设置函数
    void __fastcall SetModified(bool Value);          //Midified 的设置函数
    void __fastcall SetScale(int Value);              //Scale 的设置函数
public:
    bool     MouseDown;              //记录鼠标键是否按下
    bool     AutoUpDate;             //标识是否需要将图形显示自动更新
    TPoint   OrMousePos;             //原始鼠标位置
    TPoint   MousePos;               //记录当前鼠标位置
    TList    *Draw;                  //图形数据的存储列表
    TList    *Selection;             //被选中绘图列表
    AnsiString    File;              //存储数据的文件名称
    TFileStream * FileStream;        //存储数据的文件流
    TColor      DefaultColor;        //画布的缺省绘图颜色
    int          DefaultLineWidth;   //画布的缺省绘图线宽
    TPenStyle    DefaultPenStyle;    //画布的缺省画笔类型
    TBrushStyle DefaulyBrushStyle;   //画布的缺省 Brush 填充类型
    __property   Active;             //继承的 Active 属性
    //用于记录画布内容是否被更改的属性，
    __property bool Modified = {read=FModified,write=SetModified};
    //记录画布绘图工具的属性
    __property TShapeStyle Shape={read=FShape,write=SetShape};
    //控制显示比例的属性
    __property int Scale = {read=FScale,write=SetScale};
    //用于更新 MDI 主窗体工具按钮的函数
    void __fastcall UpDateButton(void);
    //删除被选择图形的函数
    void __fastcall Delete();
    //清除所有被选择内容（但并不删除图形）
    void __fastcall UnSelected();
    //整体移动被选择的图形
    void __fastcall MoveSelection(int DeltaX,int DeltaY);
    //向被选择图形列表中添加一项，Sole 标识添加的图形是否是唯一的被选图形
    void __fastcall AddSelect(TDraw* Draw,bool Sole);
    //从被选择图形列表中移去一个指定的图形对象，但并不删除
    void __fastcall RemoveSelect(TDraw* Draw,bool Sole);
    //存储文件，若未指定文件名或未打开指定文件，调用 SaveAs 进行存储
    void __fastcall Save(void);
    //按照指定的文件名存储图形数据
    void __fastcall SaveAs(AnsiString FileName);
```

```
    //打开指定文件中的图形
    void __fastcall OpenFile(AnsiString FileToOpen);
    //构造函数和析构函数
    __fastcall TForm1(TComponent* Owner);
    __fastcall ~TForm1( );
};

class TDrawClipborad : public TClipboard
{   //针对 TForm1 定义的剪贴板对象，
public:
    HANDLE    Handle;   //交换数据的内存区域句柄，并非必要的，可以使用局部变量
    virtual __fastcall TDrawClipborad();   // 空的构造函数
    virtual __fastcall ~TDrawClipborad();  //空的析构函数
    //重载 Assign 方法，用于拷贝 TForm1 对象中的数据
    virtual void __fastcall Assign(Classes::TPersistent* Source);
    //重载 AssignTo 方法，供 TForm1 对象拷贝数据时使用
    virtual void __fastcall AssignTo(Classes::TPersistent* Dest);
};
extern PACKAGE TForm1 *Form1;
#endif
```

在这个单元中，TForm1 是和 ToolMian 单元中的 Form2 关系最为密切的，Form2 在一个每个进程中只有一个对象，而每个 Form2 对象可以包含多个 TForm1 对象，这就是多文档程序。Form2 和 Form1 对象使用 MDIChild 和 Parent 相互建立这种逻辑包含关系的，作为控件设计，如果 Form1 要获取 Form2 只能使用 Parent 并进行类型转换而得到，而在应用程序中，我们的 Form1 是直接对 Form2 进行操作的，这是因为它们的这种包含关系在这个程序中是固定的，在编写第一行代码的时候，就已经确立了，而编写控件的时候，这种关系是无法确定的，这也是控件设计和应用程序设计的一个区别之处。

TShapeStyle 是一个简单的枚举类型，不需要过多介绍，TFrom1 中有一个该类型的属性，体现着当前的绘图工具的状态，同时该属性和 Form2 中的绘图工具的按钮状态是一致的，Shape 属性的设置函数完成了这一工作。

这个单元中 MouseDown、AutoUpDate、OrMousePos、MousePos、File、FileStream 、 DefaultColor 、 DefaultLineWidth 、 DefaultPenStyle 、DefaulyBrushStyle、Draw 和 Selection 等不同类型的成员都是供运行时内部使用的变量，部分成员的功能前面已经提到了，后面的内容中有进一步的说明。

Active 是一个继承而来的属性，用于判断该文档（MDIChild）是否为当前活动的内容，Modified 属性记录当前文档是否在存储后被更改过，和 MDI 主窗体的相应菜单项及按钮的状态相关。

Scale 是控制显示比例属性，这个例子中并没有在 MDI 主窗体中使用，当改变 Scale

属性的时候，会改变整个 Form1 中图形在屏幕上的显示比例，这个功能是通过 Form 提供的 ChangeScale 方法来完成的，因此功能也比较简单，更符合实际的是在改变显示位置、大小等比例的同时也要改变线宽等属性的显示比例，有兴趣的读者可以自行完善。

TForm1 类提供的更多的接口是使用方法来实现的，除了 UpDateButton 方法是共内部更新按钮使用的以外，其它方法基本都是提供编辑功能的方法，我们在后面作详细介绍。UpDateButton 是在文档别激活、编辑状态改变等时候，来更新 MDI 主窗体各个按钮状态的方法。

TDrawClipborad 对象是为 TForm1 使用剪贴板对象而定义的专用类，为了能够正确的使用剪贴板，我们在程序运行的最初，Form2 创建的时候，需要向系统注册一个专用的剪贴板数据类型。这个类型的注册结果保存在 Form2 的 ClipBoardFormat 成员中。关于 TDrawClipborad 的详细使用我们在后面介绍。

3.3 TDraw 对象

有了框架（TForm2）和画布（TForm1）对象之后，剩下来所需的就是图形对象 TDraw。所谓矢量绘图实际上就是按照一定的数据格式将图形存储在文件中，并能够正确的显示在计算机屏幕上。

数据的存储没有什么问题，完全可以是自己定义的格式和数据结构。而图形显示无非实在画布上将预先定义的数据格式绘制出来。在早先的程序设计中，这些工作都是有画布来完成的，诸如编辑、移动等使用鼠标进行操作也都是画布根据不同的坐标，来判断对那一个图形进行操作，并改变数据重新绘制图形。但自从面向对象的技术产生以后，解决和分析问题就有了新的思路，将每一个图形理解为一个对象，是符合实际，也使程序的设计更加简单化，现在的问题是我们如何来创建图形对象？

从我们要实现的功能来看，图形对象需要具备一下几个基本的特点：

- 能够记录自身的数据，并根据数据在画布上正确绘图；
- 能够独立响应鼠标信息，并完成基本的个体编辑功能；
- 需要和画布建立一种特定的逻辑关系。

这几点是必须的，否则程序无法摆脱复杂的逻辑结构和设计代码。

要记录自身的数据是没有问题，任何一个对象最根本的就是数据成员，要正确的绘制图形，就必须和画布发生关系，同时也会和其它的图形对象发生关系，这一要求我们可以通过图形和画布之间的逻辑关系来实现；

图形对象没有必要独立的响应 Windows 消息，因为仅仅是对部分鼠标的消息能够响应，这个可以通过对画布的消息分解，然后分发给各个图形对象来实现。个体编辑功能实际上就是通过鼠标的操作来完成的，只要图形对象能够响应鼠标消息，就能够完成个体的编辑功能。因此第二个要求实际上也是画布和图形对象之间关系的问题。

图形是属于画布的图形，这种逻辑关系是显而易见，画布还必须对每一个图形对象具有完全的控制能力，诸如刷新显示、移动图形、删除图形等。在建立这种逻辑关系的时候，必须为前两个要求创建条件。

由此看来 TDraw 对象是比较复杂的一个类，其实不用担心，我们完全没有必要从头开始定义一套图形对象的类，还要满足以上的几个条件，VCL 中为我们提供的类库就有满

足其基本要求的对象。这就是 TGraphicControl，不要觉得是个控件就会很庞大，其实 TGraphicControl 是相当简洁而高效的，除非你是一个图形方面程序设计多年的专家，否则不要企图重新设计一套用于画布和图形对象的类库，因为即使设计的类库在使用性能上比 TGraphicControl 更高，但在开发周期和类库的完整性上是很难保证的。

在本例程中，我们使用了 TShape 作为 TDraw 的基类，因为 TShape 不仅对鼠标的消息进行了相应的处理，还提供了绘图的 Canvas、Pen、Brush 属性以及 Paint 方法等，可以使代码更加简化一些，我们仅需要对其中一些方法重载就可以实现大部分要求，如果希望获得更高的程序性能，可以使用 TControl 作为 TDraw 的基类。

TDraw 所在的单元 ShapeClass 的定义文件如下：

```
#ifndef ShapeClassH
#define ShapeClassH
#include <Controls.hpp>
#include <ExtCtrls.hpp>

enum TMouseProperty
{
   _MOVE,_LEFT,_RIGHT,_TOP,_BOTTOM,_TOPLEFT,_TOPRIGHT,_BOTTOMLEFT,
   _BOTTOMRIGHT
};
//定义图形内部鼠标位置属性的枚举类型，每一个图形对象除了本身显示的图形之外，
//还有 8 个表示位置大小等信息的属性控制点。鼠标拖动属性控制点可以对图形的大小
//等进行编辑
//---------------------------------------------------------------------
struct TDataToSave
{   //定义用于存储文件和剪贴板交换数据的图形结构，定义这一结构的目的是将
    int          Top;        //需要保存的类成员转换成为顺序存储结构类型
    int          Left;       //在这个结构中没有指针的存在，都是原始的数
    int          Width;      //据类型
    int          Height;     //以上四个成员同控件中的属性，
    TColor       PenColor;   //图形中 Pen 的 Color 属性
    int          PenWidth;   //Pen 画笔的 Width
    TPenStyle    PenStyle;   //画笔的类型
    TColor       BrushColor; //图形的 Brush 的 Color 属性
    TBrushStyle  BrushStyle; //Brush 的类型，即填充格式
    TShapeType   Shape;      //图形类型标记
};                           //这个结构中的成员是图形对象需要存储或者必须记录
//的信息，属于固有信息，或者叫图形的固有属性，而 Selected 则只是在运行时使用的
```

```
//定义图形对象的类，在本例中，图形对象是从 TShape 派生而来
class TDraw : public TShape
{
private:
    int         Ox;          //用于记录鼠标的历史位置，比如鼠标按下时，
    int         Oy;          //用 Ox 和 Oy 记录鼠标键按下的位置
    bool        IsMouseDown;         //用于记录鼠标是否按下
    TRect       PropertyRect[8];     //用于记录 8 个属性控制点的矩形区域
protected:
    void __fastcall SetSelected(bool Value); //Selected 属性的设置函数
    void __fastcall UpdatePropertyRect();    //用于更新属性控制点的区域
public:             //当图形的位置和大小改变时，需要更新属性控制点
    virtual TMouseProperty __fastcall MousePosClass(int X,int Y);
    //返回这指定点相对图形的位置属性，该函数返回位置属性的同时，也更新
    //图形上鼠标显示，对于不同的图形对象从 TDraw 派生的可以重载该函数
    bool        FSelected;     //Selected 属性的内部存储数据，与其它的
    //属性不一样，Selected 是和画布背景相关的属性，FSelected 使用了公开的数据
    virtual void __fastcall SetBounds(int ALeft, int ATop, int AWidth,
                        int AHeight);  //虚拟函数，用于设置控件的
    //位置和大小，在这个函数中需要对图形的属性控制点进行更新
    virtual void __fastcall Paint(void);   //重载的绘图函数
    DYNAMIC void __fastcall MouseDown(TMouseButton Button,
                    TShiftState Shift, int X, int Y);  //重载的鼠标函数
    DYNAMIC void __fastcall MouseMove(TShiftState Shift,
                    int X, int Y);                     //重载的鼠标函数
    DYNAMIC void __fastcall MouseUp(TMouseButton Button,
                    TShiftState Shift, int X, int Y); //重载的鼠标函数
    __property bool Selected = {read=FSelected,write=SetSelected};
    //Selected 属性，用于记录图形是否每选中，同时画布对象中的
    __property Color;            //发布 Color 属性
    __fastcall TDraw();          //免参数的构造函数
    __fastcall TDraw(int ALeft,int ATop,int ARight,int AButtom);
    //指定了位置和大小的构造函数
    __fastcall TDraw(int ALeft,int ATop );
    //指定了起始位置的构造函数
};
#endif
```

在这个单元中，TMouseProperty **枚举类型定义了鼠标位于图形中的几种不同状态。**

对于一个图形对象，当被选择的时候，不仅要将整个图形高亮显示，还需要提供几个用于编辑的“属性控制点”，使用鼠标拖动属性控制点的时候可以对一个图形对象的大小、长、宽等特征进行编辑，TDraw 对象中将有一个判断当前鼠标位置特征的方法或者属性。

TDataToSave 结构是用于保存数据或者使用剪贴板时使用的，TDraw 的成员很多，但只有很少一部分是用户真正关心的数据，这些数据就是矢量图形的参数，保存文件或者使用剪贴板只关心这些参数数据就足够，另一个方面是存储和使用剪贴板的时候，需要一个顺序存储的数据结构，而 TDraw 类并不是顺序存储的，很多图形参数，比如线型、线宽等都是对象成员的属性，我们知道对象成员在 VCL 类库中只能以指针方式出现，因此定义这个结构是完成存储和数据交换所必要的。

尽管将 TShape 的部分方法重载就可以满足 TDraw 的大部分功能，但这远远不够的，作为 TDraw 对象，还应该具备选择、高亮显示、移动等编辑功能，这些是 TShape 原本不具备的，而且在重载的方法中，还需要一些内部使用的成员变量，我们逐一介绍。

Ox、Oy、IsMouseDown 是处理鼠标消息的函数内部使用的，PropertyRect 是 8 个用于高亮显示的属性控制点的位置区域，当图形被选中的时候，这几个矩形区域都处于高亮显示，鼠标位于属性控制点区域内时，将可以通过鼠标拖动来改变图形的大小。

Selected 属性标识该图形对象是否被选择，SetSelected 为其设置函数，Selected 属性不仅和自身的显示有关，而且和画布中的 Selection 属性建立对应关系，这种关系是一对多的，画布中的 Selection 列表中可以为空，也可以包含多个被选择的对象，同时，当 Selection 属性发生改变的时候，也会影响到 MDI 主窗体的菜单和按钮的状态。在这里，Selected 属性的内部存储数据和以前的有所不同，FSelected 是一个公开的成员，这也是我们前面讲过的，Delphi 中和 C++ Builder 的一个差异。SetSelected 的源代码如下：

```
void __fastcall TDraw::SetSelected(bool Value)
{   //Selected 的设置函数
    if(FSelected != Value)
    {  //在设置图形对象的 FSelected 的同时，需要将图像添加到画布的
       FSelected = Value;          //被选图像列表总
       if( Value )                 //添加唯一的一个被选图形
          ((TForm1*)Parent)->AddSelect(this,true);
       else                        //从画布的被选图形列表中移去该图形
          ((TForm1*)Parent)->RemoveSelect(this,true);
       Refresh();                  //更新绘图
    }//除了 SetSelected 方法可以更改 FSelected 之外，TForm1 的方法也是允许改变这个
}   //成员的，因为 Selected 和 Selection 是一对多的关系，为了不产生环路调用，并且允许
    //是唯一的被选择图形或者多个被选择图形之一，必须允许 Form1 修改 FSelected
```

UpdatePropertyRect 方法用于更新 PropertyRect 数据的，当控件的位置和大小等属性发生改变的时候，图形的属性控制点也会跟随变化，在 TDraw 中我们通过重载

SetBounds 方法，并在 SetBounds 方法中调用 UpdatePropertyRect 方法来完成属性控制点的更新。UpdatePropertyRect 方法的源代码如下：

```
void __fastcall TDraw::UpdatePropertyRect()
{                                                    //属性控制点的计算函数
    PropertyRect[0] = TRect(0,0,5,5);                    //左上角
    PropertyRect[1] = TRect(Width/2-2,0,Width/2 + 2,5); //上边中点
    PropertyRect[2] = TRect(Width-4,0,Width,5);          //右上角
    PropertyRect[3] = TRect(Width - 4,Height             //右边中点
                    /2 - 2,Width,Height/2 + 2);
    PropertyRect[4] = TRect(Width - 4,
                    Height - 4,Width,Height);            //右下角
    PropertyRect[5] = TRect(Width/2-2,Height - 4,
                    Width/2 +2,Height);                  //下边中点
    PropertyRect[6] = TRect(0,Height - 4,5,Height);      //左下角
    PropertyRect[7] = TRect(0,Height/2 - 2,5,
                    Height /2 + 2);                      //左边中点
}
```

SetBounds 的源代码如下：

```
void__fastcallTDraw::SetBounds(int ALeft,int ATop, int AWidth, int AHeight)
{
    TShape::SetBounds(ALeft,ATop,AWidth,AHeight); //继承父类的方法
    if(Parent)              //画布是有效的时候，需要将 Modified 标记为 true
      ((TForm1*)Parent)->Modified = true;
    UpdatePropertyRect();    //更新属性控制点区域
}
```

MousePosClass 用于判断指定的坐标处于图形的位置特征是属于图形内部的还是属性控制点，并且根据位置特征更新图形的鼠标 Cursor 类型。这是一个完全供内部使用的方法，也可以定义为保护类型的。MouseClass 的源代码如下：

```
TMouseProperty __fastcall TDraw::MousePosClass(int X,int Y)
{                                  //根据指定点所在的区域，返回
    TPoint Pt = TPoint(X,Y);    //位置属性，并设置鼠标指针
    if(PtInRect(PropertyRect[0],Pt))
    {
       Cursor = crSizeNWSE;
```

```
        return _TOPLEFT;
    }
    else if(PtInRect(PropertyRect[1],Pt))
    {
        Cursor = crSizeNS;
        return _TOP;
    }
    //其它点位代码类同，请读者参考源码
    else if(PtInRect(PropertyRect[7],Pt))
    {
        Cursor = crHSplit;
        return _LEFT;
    }
    else                        //缺省的状态，是 Move
    {
        Cursor = crSizeAll;
        return _MOVE;
    }
}
```

Paint 方法是重载 TShape 的方法，并且完全抛弃了原有的 Paint，没有继承，在 Piant 方法中，图形需要根据线型、线宽、颜色、填充方式以及图形的形状等参数将图形正确的绘制在 Canvas 上，当 Selected 属性为 true 的时候，图形还有进行高亮显示并绘制属性控制点。Paint 方法源代码如下：

```
void __fastcall TDraw::Paint(void)
{  //图形对象的绘图函数，TForm1 对象为画布，所有图形均绘制在 Panret 对象上
   //本例中 TDraw 只能在 TForm1 对象上绘制
   if(!((TForm1*)Parent)->AutoUpDate )
     return;       //AutoUpDate 用于记录是否自动更新，这个目的是防止多个
     //图形对象同时操作的时候，逐一触发更新消息，而引起图形闪烁或者效率下降
   Canvas->Pen->Assign(Pen);       //使用图形对象中的画笔 Pen
   Canvas->Brush->Assign(Brush);   //使用图形对象中的 Brush
   switch (Shape)                 //根据不同的 Shape 选择的绘图方法
   {
     case stRectangle:            //绘图区域比控件的区域小 2 个像素，这是由于
       Canvas->Rectangle(2,2,Width-2,Height-2);   //属性控制点的区域可能会在
       break;                     //图形区域之外，但是属于对象中的点
     case stEllipse:              //椭圆同样缩小两个像素
```

```
        Canvas->Ellipse(2,2,Width-2,Height-2);
        break;
    }
    if(Selected)                    //若图形被选择，则使用 clLime 颜色绘制出
    {                               //属性控制点，并且使用 psDot 线型以单线
      Canvas->Brush->Style = bsSolid;      //覆盖绘制图像区域
      Canvas->Brush->Color = clLime;
      Canvas->Pen->Width = 2;               //线宽为 2 可以将矩形绘制成实心
      Canvas->Pen->Color = clLime;
      Canvas->Rectangle(PropertyRect[0]); //左上角
      Canvas->Rectangle(PropertyRect[1]);
      Canvas->Rectangle(PropertyRect[2]);
      Canvas->Rectangle(PropertyRect[3]);
      Canvas->Rectangle(PropertyRect[4]);
      Canvas->Rectangle(PropertyRect[5]);
      Canvas->Rectangle(PropertyRect[6]);
      Canvas->Rectangle(PropertyRect[7]);
      Canvas->Brush->Style = bsClear;       //bsClear 填充格式不进行内部填充
      Canvas->Pen->Width = 1;               //只绘制出外框
      Canvas->Pen->Style = psDot;
      Canvas->Rectangle(2,2,Width-2,Height-2);
    }
}
```

TDraw 的 MouseDown、MouseUp 和 MouseMove 都属于编辑使用的方法，我们在后面介绍，TDraw 有三个不同版本的构造函数，属于创建新图形的方法，我们也放在后面介绍。

3.4 个体编辑

我们将图形的编辑划分为个体编辑和整体编辑是为了明确编辑动作的实施者，所谓个体编辑只和图形自身有关，不涉及到其它的图形对象，而整体编辑则是对一个图形或者几个图形同时进行编辑，并且编辑是和其它图形有关联的，因此整体编辑适合由 TForm1 对象来实现，而个体编辑适合由 TDraw 对象来完成。在个体编辑的功能中有创建新图形和拖动编辑两种，实际上创建新图形并不是 TDraw 单独完成的，也是需要 TForm1 来发起的。

3.4.1 创建新的图形

创建一个新图形的过程是这样的：首先，TForm1 的绘图工具不处于 Select 状态时，

在画布上按下鼠标左键，拖动到适当位置释放，释放鼠标左键时，根据所选择的绘图工具在鼠标拖动的区域创建相应的图形对象，并按照缺省的线宽、线型、颜色、填充方式等设置新建图形的属性，与此同时还要正确的建立图形对象和 TForm1 对象之间的逻辑关系，VCL 对象所必须的逻辑关系是 Parent 和 Child 的关系，作为我们图形编辑还存在的逻辑关系是 Form1 的 Draw 属性必须添加新建的图形。

TForm1 的 OnMouseDown 事件处理函数如下：

```
void __fastcall TForm1::DrawMouseDown(TObject *Sender, TMouseButton Button,
                                      TShiftState Shift, int X, int Y)
{   //鼠标事件的处理函数，当左键按下时进行操作
    if(Button == mbLeft)
    {
       MouseDown = true;       //记录鼠标被按下
       OrMousePos.x = X;       //记录鼠标始按下的原位置
       OrMousePos.y = Y;
       MousePos.x = X;         //记录鼠标消息的当前位置
       MousePos.y = Y;         //按下时原始位置和当前位置相同
       UnSelected();           //清除画布上被选择的图形列表
       UpDateButton();         //更新 MDI 主窗体的按钮显示
       Invalidate();           //强迫重新绘图
    }
}
```

TForm1 的 OnMouseMove 事件处理函数如下：

```
void __fastcall TForm1::DrawMouseMove(TObject *Sender, TShiftState Shift,
     int X, int Y)
{   //鼠标移动事件的处理函数
    if(MouseDown)
    {  //当左键被按下时移动鼠标按下面操作
       Canvas->Rectangle(TRect(OrMousePos,MousePos));
       MousePos.x = X;         //绘制鼠标轨迹的区域，蓝色虚线矩形
       MousePos.y = Y;         //按照异或算法绘图，二次绘制复原
       Canvas->Rectangle(TRect(OrMousePos,MousePos));
    }                          //画布上始终显示原始鼠标位置和鼠标当前移动到的位置
}                              //所定义的矩形
```

上面这两个事件的处理方法并不复杂，最复杂的是释放鼠标按键时需要根据不同的绘图工具类型进行不同的操作，TForm1 的 OnMouseUp 事件处理函数如下：

```
void __fastcall TForm1::DrawMouseUp(TObject *Sender, TMouseButton Button,
                                    TShiftState Shift, int X, int Y)
{  //释放鼠标按键的处理函数，在不同的状态下，所作的操作不同
   if(MouseDown && Button == mbLeft)     //仅在鼠标左键按下并释放时动作
   {                                     //忽略其它按键以及键盘状态
      TDraw * NewDraw;                   //临时使用的图形对象指针
      int Temp;
      Canvas->Rectangle(TRect(OrMousePos,MousePos)); //释放鼠标时，清除
      if( OrMousePos.x > MousePos.x )                //鼠标移动轨迹的矩形
      {                                  //调整按下鼠标按键的位置和释放鼠标
        Temp = OrMousePos.x;             //按键的位置，
        OrMousePos.x = MousePos.x;       //使 OrMousePos 为鼠标移动区域矩形的
        MousePos.x = Temp;               //左上角
      }                                  //MousePos 为右下角
      if( OrMousePos.y > MousePos.y )
      {
        Temp = OrMousePos.y;
        OrMousePos.y = MousePos.y;
        MousePos.y = Temp;
      }   //若鼠标的 X 方向和 Y 方向的移动距离不为 0 时有效。
      if(OrMousePos.x != MousePos.x && OrMousePos.y != MousePos.y)
      { //根据不同的 Shape 类型作不同的操作
        if(Shape != _Select)             //绘图工具不为"选择"的时候
        {                                //则创建新的图形对象
          NewDraw = new TDraw(OrMousePos.x,OrMousePos.y,
                              MousePos.x + 1,MousePos.y + 1);
          NewDraw->Pen->Width = DefaultLineWidth;        //按照缺省的线宽
          NewDraw->Pen->Color = DefaultColor;            //缺省的颜色
          NewDraw->Pen->Style = DefaultPenStyle;         //缺省的线型
          NewDraw->Brush->Style = DefaulyBrushStyle;     //以及缺省的填充
          NewDraw->Parent = this;            //在画布上绘制新建的图形
          NewDraw->Color = clWhite;          //填充的颜色始终为白色
          Draw->Add(NewDraw);                //添加到图形列表中
          Modified = true;                   //设置画布的状态为被修改
        }
        switch (Shape)       //根据 Shape 工具类型继续进行以下操作
        {
          case _Select:      //Shape 为选择工具时，选择区域内的图形，
```

```
{                    //按照鼠标移动的方向,分作两种选择方式,
  TRect Rect;        //从左向右的时候,选择鼠标移动区域内包含的图形
  Rect = TRect(OrMousePos,MousePos);   //即图形对象的四个
  TDraw* DrawTemp;       //临时变量      //顶点都在矩形区域之内
  if(MousePos.x > X)     //从左到右的时候,MousePos.x 和 X 相等
    for(int i = 0;i < Draw->Count; i++ )
    {                    //逐一遍历图形列表的每一个元素
      DrawTemp = (TDraw*)Draw->Items[i];
      if(PtInRect(Rect,TPoint(DrawTemp->Left,
                  DrawTemp->Top))||
         PtInRect(Rect,TPoint(DrawTemp->Left +
                  DrawTemp->Width,DrawTemp->Top))||
         PtInRect(Rect,TPoint(DrawTemp->Left,DrawTemp->Top +
                  Height))||
         PtInRect(Rect,TPoint(DrawTemp->Left +
                  DrawTemp->Width,DrawTemp->Top +
                  DrawTemp->Height)))
      {      //只要上下左右四个顶点之一在鼠标区域之内的图形都添加到
        Selection->Add(DrawTemp);    //选择列表中
        DrawTemp->FSelected = true;  //并设置 FSelected 标记
      }
    }
  else          //反之则是从左向右,逐一遍历图形列表中的元素
    for(int i = 0;i < Draw->Count; i++ )
    {         //上下左右四个顶点所有的都在矩形区域之内的时候
      DrawTemp = (TDraw*)Draw->Items[i];
      if(PtInRect(Rect,TPoint(DrawTemp->Left,
                      DrawTemp->Top))&&
         PtInRect(Rect,TPoint(DrawTemp->Left +
                      DrawTemp->Width,DrawTemp->Top))&&
         PtInRect(Rect,TPoint(DrawTemp->Left,DrawTemp->Top +
                      DrawTemp->Height))&&
         PtInRect(Rect,TPoint(DrawTemp->Left +
                      DrawTemp->Width,DrawTemp->Top +
                      DrawTemp->Height)))
      {      //则将该图形添加到选择列表清单中
        Selection->Add(DrawTemp);    //并设置 FSelected
        DrawTemp->FSelected = true;       //标记
      }
```

```
            }
            Refresh();                          //刷新图形显示
            UpDateButton();                     //更新 MDI 主窗体按钮
            break;
          }                                     //当 Shape 工具为矩形时
          case _Rect:
            NewDraw->Shape = stRectangle;       //将新建的图形对象设置为
            break;                              //矩形
          case _Circle:                         //工具为椭圆时
            NewDraw->Shape = stEllipse;         //将新建的图形对象设置为
            break;                              //椭圆
          case _Line:                           //画线工具空置,未作相应
                                                //的处理代码
            break;
        }
      }
      MouseDown = false;                        //将鼠标按键标记为释放
    }
}
```

TForm1 的 MouseUp 事件中，除了创建图形之外，还有选择图形的工具，本程序中画布的工具处于 Select 状态的时候，提供了两中不同的选择方式，一种是从左向右拖动鼠标，这时要求图形完全包含在鼠标区域之内才能够被选择，另一种方式是从右向左拖动鼠标，此时只要图形的上、下、左、右四个顶点之一包含在鼠标区域之内的图形都会被选中。当图形工具为矩形、椭圆或者线段的时候，Form1 根据鼠标位置创建新的图形，TDraw 的构造函数如下：

```
__fastcall TDraw::TDraw(int ALeft,int ATop,int ARight,int AButtom)
              :TShape(NULL)
{
    Left = ALeft - 2;                       //构造函数给出的是图形的区域
    Top = ATop - 2;                         //控件的实际大小比图形区域大
    Width = ARight - ALeft + 2;             //两个像素
    Height = AButtom - ATop + 2;
    UpdatePropertyRect();                   //计算属性控制点的区域
}
__fastcall TDraw::TDraw(int ALeft,int ATop ) : TShape(NULL)
{
    Left = ALeft - 2;                       //只指定了位置左上角时按照
```

```
    Top = ATop - 2;                          //缺省的大小创建对象
    Width = 100;
    Height = 30;
    UpdatePropertyRect();
}
```

TDraw 总共有三个构造函数，其中一个是空的，我们就不介绍了，那是为文件读取和剪贴板而使用的，程序中用户创建新图形的构造函数是带有四个位置坐标的构造函数，TDraw 的实际大小要比图形大两个像素，因为图形对象被选中时显示的属性控制点要超出图形的范围。带有起始位置的构造函数在本例程中并没有使用，和带有四个参数的构造函数大致一样，只是使用了缺省的大小来创建新图形。

3.4.2 拖动改变位置及大小

拖动编辑是一个纯粹的由 TDraw 自身完成的编辑动作，主要功能是鼠标位于属性控制点内的时候，可以使用左键拖动改变属性控制点的位置，实际上就是改变了图形的上、下、左、右边界或者四个顶点的位置，TDraw 的鼠标处理我们使用了重载函数来实现，而不是 TForm1 中那样使用事件句柄来实现的，两者实现起来没有本质的区别，但也稍有差异，在 Form1 中使用事件句柄是为了在不同的状态可以通过更改事件的处理句柄来改变鼠标的响应动作，而在 TDraw 中使用重载函数的方法来响应鼠标消息，则是为了更好的体现多态性优点，当 TDraw 的派生类具有和 TDraw 不太一样的鼠标动作时，只需要重载该函数就可以实现，这比使用事件句柄更方便合理一些。

鼠标按键按下的处理函数如下：

```
void __fastcall TDraw::MouseDown(TMouseButton Button,
     TShiftState Shift, int X, int Y)
{  //重载鼠标的点击函数，用于处理鼠标按下的动作
   if(Button == mbLeft)                  //只对左键反应
   {
      if(Shift.Contains(ssShift))        //若 Shift 键同时按下，则允许多个图形
      {                                  //对象被同时选中
         IsMouseDown = true;             //记录鼠标已经按下
         Ox = X;                         //同时记录按下的位置坐标
         Oy = Y;
         if(FSelected)                   //如果对象原本就处于
         {                               //被选择状态
            FSelected = false;           //则从被选图形列表中移去该图像
            ((TForm1*)Parent)->RemoveSelect(this,false);
            Refresh();                   //更新图形显示
```

```
        }
        else                                    //若还没有处于选择状态，则加入
        {                                       //画布的被选择图像列表中
           FSelected = true;
           ((TForm1*)Parent)->AddSelect(this,false);
           Refresh();                           //刷新图像显示
        }
      }
      else                                      //若没有按下 Shift 键
      {                                         //则记录鼠标已经按下，并记录
         IsMouseDown = true;                    //相应坐标，同时设置 Selected 属性
         Selected = true;                       //Selected 属性会自动清除其它
         Ox = X;                                //被选择的图形对象
         Oy = Y;
      }
   }
}
```

鼠标指针移动的处理函数如下：

```
void __fastcall TDraw::MouseMove(TShiftState Shift,
     int X, int Y)
{
   TMouseProperty MouseProperty;                //记录鼠标位置属性的临时变量
   if(IsMouseDown)                              //若鼠标左键被按下
   {                                            //获取按下鼠标键时鼠标的位置属性
      MouseProperty = MousePosClass(Ox,Oy);
      switch (MouseProperty)                    //根据不同的位置属性来决定不同
      {                                         //的操作
        case _LEFT:                             //根据鼠标的移动改变图形的左边界
          SetBounds(Left - Ox + X,Top,Width + Ox - X,Height);
          break;
        case _RIGHT:                            //根据鼠标的移动改变图形的右边界
          SetBounds(Left,Top,Width - Ox + X,Height);
          Ox = X;
          break;
        case _TOP:                              //根据鼠标的移动改变图形的上边界
          SetBounds(Left,Top - Oy + Y,Width,Height + Oy - Y);
          break;
```

```
        case _BOTTOM:                          //根据鼠标的移动改变图形的下边界
          SetBounds(Left,Top,Width,Height - Oy + Y);
          Oy = Y;
          break;
        case _TOPLEFT:                         //根据鼠标的移动改变图形的左、上边界
          SetBounds(Left - Ox + X,Top - Oy + Y,Width + Ox - X,
                    Height + Oy - Y);
          break;
        case _TOPRIGHT:                        //根据鼠标的移动改变图形的右、上边界
          SetBounds(Left,Top - Oy + Y,Width - Ox + X,Height + Oy - Y);
          Ox = X;
          break;
        case _BOTTOMLEFT:                      //根据鼠标的移动改变图形的左、下边界
          SetBounds(Left - Ox + X,Top,Width + Ox - X,Height - Oy + Y);
          Oy = Y;
          break;
        case _BOTTOMRIGHT:                     //根据鼠标的移动改变图形的右、下边界
          SetBounds(Left,Top,Width - Ox + X,Height - Oy + Y);
          Ox = X;
          Oy = Y;
          break;
        case _MOVE:                            //根据鼠标的移动改变图形的位置
          ((TForm1*)Parent)->MoveSelection(X - Ox,Y - Oy);
      }                                        //使用画布的方法进行整体移动
    }
    else                                       //利用该函数根据当前鼠标位置
      MousePosClass(X,Y);                      //进行鼠标指针的设置
}
```

释放鼠标按键的数量函数如下：

```
void __fastcall TDraw::MouseUp(TMouseButton Button,
     TShiftState Shift, int X, int Y)
{
   if(IsMouseDown && Button == mbLeft)
      IsMouseDown = false;                     //鼠标键弹起，只对左键反应
}
```

3.5 整体编辑

整体编辑通常是通过 TForm1 对象来完成的，整体编辑指的是对一个或多个图形对象同时进行某种编辑，而且是相对于整个画布的编辑动作，包含移动图形、删除图形和改变图形的属性，和个体编辑一样，整体编辑首先需要选中被编辑的对象，整体编辑是对所有被选中的图形进行的。

3.5.1 移动所选图形

移动图形是由被选择的图形所发起的，而且只有鼠标位于被选择的图形的内部，并不在属性控制点内的时候，才能进行的，但是操作是由 TForm1 对象来完成的，读者可以参考 TDraw 的 MouseMove 方法。

在 TForm1 中我们提供了移动被选择图形的方法 MoveSelection，源代码如下：

```
void __fastcall TForm1::MoveSelection(int DeltaX,int DeltaY)
{   //移动被选择图形的方法
    AutoUpDate = false;          //禁止图形对象自动更新而闪烁
    for(int i = 0; i < Selection->Count; i++)
      ((TDraw*)Selection->Items[i])->SetBounds(((
                      TDraw*)Selection->Items[i])->Left + DeltaX,
                      ((TDraw*)Selection->Items[i])->Top + DeltaY,
                      ((TDraw*)Selection->Items[i])->Width,
                      ((TDraw*)Selection->Items[i])->Height);
    AutoUpDate = true;           //使用图形的 SetBounds 方法进行移动
    Modified = true;             //设置 Modified 标记并更新画布
    Refresh();
}
```

AutoUpDate 用于 TDraw 图形的 Paint 方法是否绘图，整体移动的时候，需要使之无效，这样 TDraw 不会在每一个图形移动的时候都刷新显示界面而产生闪烁，整体移动结束后，统一使用 Refresh 进行画布的刷新。

3.5.2 删除所选图形

删除动作是由 MDI 主窗体发起的，但是操作也是有画布完成的，当用户选择了图形对象的时候，MDI 主窗体的 Delete 按钮和菜单项都会变为有效，删除动作就是由删除按钮和菜单发起的，相应的代码如下：

```
void __fastcall TForm2::DeletButtonClick(TObject *Sender)
{
```

```
    if(ActiveDraw)
      ActiveDraw->Delete();
}

void __fastcall TForm1::Delete()
{   //将画布中所选择的图形删除
    if( Selection->Count > 0)    //必须使用逆序循环，否则代码复杂的多
      for(int i = Selection->Count - 1; i >= 0; i--)
      { //首先从图形列表中清除
        Draw->Delete(Draw->IndexOf(Selection->Items[i]));
        delete (TDraw*)Selection->Items[i];  //销毁对象
        Selection->Delete(i);                //从被选择列表中清除
      }
}
```

3.5.3 改变图形属性

改变图形的属性包括了四个部分的内容，分别和图形属性工具条上的四个控件对应，这部分功能可以和 Delete、Move 一样，由 TForm1 来完成，甚至是由 TForm1 的属性来完成，同样 TForm2 的控件状态只和这几个属性发生关系，通过改变属性就可以完成图形属性的编辑。本例中没有这样作，因为多文档界面中一个 MDI 主窗体和多个 MDIChild 窗体对应，使用 TForm1 的属性并不能完全实现对 MDI 主窗体界面的刷新，因此使用了方法实现界面的刷新，编辑图形的属性也是由 TForm2 发起并完成的，具体代码如下：

改变填充方式的方法如下：

```
void __fastcall TForm2::FillButtonClick(TObject *Sender)
{
    if(!ActiveDraw)                    //确保当前有活动的图形文件
      return;
    if(FillButton->Down)               //改变后的状态为实心填充时
      if(ActiveDraw->Selection->Count > 0)   //并且活动图形文件中有选择图形
        for(int i = 0; i < ActiveDraw->Selection->Count; i++)
          ((TDraw*)ActiveDraw->Selection->Items[i])->Brush->Style
                        = bsSolid;          //逐一改变填充方式
      else                             //若没有被选择图形时，改变缺省的填充方式
        ActiveDraw->DefaulyBrushStyle = bsSolid;
    else                               //若改变为 bsClear 不填充时
      if(ActiveDraw->Selection->Count > 0)   //并且活动图形文件中有选择图形
        for(int i = 0; i < ActiveDraw->Selection->Count; i++)
```

```
            ((TDraw*)ActiveDraw->Selection->Items[i])->Brush->Style
                        = bsClear;    //逐一改变填充方式
      else                            //若没有被选择图形时，改变缺省的填充方式
        ActiveDraw->DefaulyBrushStyle = bsClear;
}
```

改变线宽的的方法如下：

```
void __fastcall TForm2::ComboBoxEx1Change(TObject *Sender)
{
   if(!ActiveDraw)              //确保当前有活动的图形文件
     return;                    //当选择的图形不为0时，逐一改变图形的线宽
   if(ActiveDraw->Selection->Count > 0)
      for(int i = 0; i < ActiveDraw->Selection->Count; i++)
        ((TDraw*)ActiveDraw->Selection->Items[i])->Pen->Width =
                          ComboBoxEx1->ItemIndex + 1;
   else                           //当选择的图形为0时，改变画布缺省的线宽
      ActiveDraw->DefaultLineWidth = ComboBoxEx1->ItemIndex + 1;
}
```

改变颜色的方法如下：

```
void __fastcall TForm2::ColorBox1Change(TObject *Sender)
{
   if(!ActiveDraw)              //确保当前有活动的图形文件
     return;                    //当选择的图形不为0时，逐一改变图形的颜色
   if(ActiveDraw->Selection->Count > 0)
      for(int i = 0; i < ActiveDraw->Selection->Count; i++)
        ((TDraw*)ActiveDraw->Selection->Items[i])->Pen->Color =
                          ColorBox1->Selected;
   else                           //当选择的图形为0时，改变画布缺省的颜色
      ActiveDraw->DefaultColor = ColorBox1->Selected;
}
```

改变线型的方法如下：

```
void __fastcall TForm2::ComboBoxEx2Change(TObject *Sender)
{
   if(!ActiveDraw)              //确保当前有活动的图形文件
```

```
        return;                          //当选择的图形不为0时，逐一改变图形的线型
    if(ActiveDraw->Selection->Count > 0)
        for(int i = 0; i < ActiveDraw->Selection->Count; i++)
            ((TDraw*)ActiveDraw->Selection->Items[i])->Pen->Style =
                                (TPenStyle)ComboBoxEx2->ItemIndex ;
    else                                 //当选择的图形为0时，改变画布缺省的线型
        ActiveDraw->DefaultPenStyle = (TPenStyle)ComboBoxEx2->ItemIndex;
}
```

3.6 文件存取功能

文件存取功能是一个编辑软件的基本功能，如果不能保存数据，所作的一切编辑工作都等于0，幸好在C++ Builder中，Borland为我们提供了更加便捷的文件操作方式，这就是流式文件对象TFileStream。

要实现文件操作，除了TFileStream对象之外，还需要确定要保存的数据格式，对于一个图形对象而言，尽管TShape对象中有很多成员，但是只有很少一部分是真正用户创建的图形的要素，TDraw的大多数成员都是在程序运行时所用的，包括我们定义的8个属性控制点的区域以及Selected属性，这些都没有必要保存的，有一些属性比如Parent，每次运行都是不一样的，因此我们必须筛选图形对象中属于图形参数的数据进行保存，同时在保存的时候，需要将这些数据转换为顺序存储结构的数据，而不能是指针等非顺序存储的结构。

顺序存储数据结构我们前面的声明文件中已经包括了，这里不在介绍。文件操作功能至少包括三个功能：一是保存文件；二是另存为新的文件；三是读取文件。保存文件是按照一定的个数和顺序将数据和相应的文件信息保存到文件中，另存为是按照新指定的文件将数据存储，读取则是存储的逆向过程，文件的保存、读取和另存为新文件都是由TForm1完成的，整个程序中这几个操作都是由TForm2发起的。

3.6.1 保存文件

我们在TForm1中提供了用于保存文件的方法Save，其源代码如下：

```
void __fastcall TForm1::Save(void)
{   //画布对象的存储方法
    TFileHeader Header;                  //文件的标记头，同剪贴板中的标记头
    TDataToSave Data;
    TDraw*      DrawItem;
    Header.DrawCount = Draw->Count;      //记录图形的数量
    if(!FileStream)                      //若没有打开文件，即新建的图形文件
    {                                    //则调用MDI主窗体的SaveAs处理句柄
        Form2->SaveDialog1->FileName = Caption;
```

```
        Form2->SaveAsClick(this);          //并以画布自身为 Sender 触发
    }
    else if(Modified)                      //图形已经属于打开的文件
    {                                      //则存储图形
      delete FileStream;                   //重新创建 FileStream 可以完整的覆盖
      FileStream = new TFileStream(File,fmCreate);       //原有文件的内容
      FileStream->Write(&Header,sizeof(TFileHeader));    //写入标记头
      for(int i = 0; i < Draw->Count; i++)               //逐一写入图形对象
      {
        DrawItem = (TDraw*)Draw->Items[i];
        Data.Top = DrawItem->Top;
        Data.Left = DrawItem->Left;
        Data.Width = DrawItem->Width;
        Data.Height = DrawItem->Height;
        Data.PenColor = DrawItem->Pen->Color;
        Data.PenStyle = DrawItem->Pen->Style;
        Data.PenWidth = DrawItem->Pen->Width;
        Data.BrushColor = DrawItem->Brush->Color;
        Data.BrushStyle = DrawItem->Brush->Style;
        Data.Shape = DrawItem->Shape;
        FileStream->Write(&Data,sizeof(TDataToSave));    //写入文件
      }
      Modified = false;                                  //存储后标记为未被更改
    }
  }
```

在 Save 方法中，要对文件进行保存，必须确保文件对象已经存在，如果没有文件流式对象，说明是新建的图形文档，因此调用 Form2 的 SaveAs 事件的处理句柄，并且以 this 作为 Sender 参数，在 Form2 的 SaveAs 事件处理函数中根据 Sender 参数来判断是菜单和按钮触发的 SaveAs 事件还是由 Form1 对象触发的 SaveAs 事件，这样作的目的是为了使 Form1 的 Save 方法消除相关性，不管 Form1 是否处于激活状态，其 Save 方法总是可以保证在调用 Form2 的 SaveAs 句柄时，能够的到正确的处理。Form2 的 SaveAs 句柄在打开存储对话框后，会调用活动文档的 SaveAs 或者 Sender 的 SaveAs 方法进行存储，这取决于 SaveAs 事件句柄是由谁触发的。

3.6.2 另存为其它文件

另存为其它文件的发起者有两种情况，一种是用户点击 MDI 主窗体的菜单项和另存为按钮，另一种情况是 Form1 对象在 Save 方法中发起的。第一种情况，由事件的发起者

直接调用活动文档的 SaveAs 方法进行存储，第二种情况则是由 Form1 的 Save 方法调用 Form2 的 SaveAs 处理函数，然后间接的调用 Form1 对象的 SaveAs 方法进行存储。

Form1 对象的 SaveAs 方法需要重新按照指定的文件名创建新的文件,然后调用自身的 Save 方法将数据保存。以下是 TForm2 的 SaveAs 事件的处理函数和 TForm1 的 SaveAs 方法。

```
void __fastcall TForm2::SaveAsClick(TObject *Sender)
{
   if(!ActiveDraw)                  //确保有活动的文档
     return;
   if(SaveDialog1->Execute())       //打开存储对话框
     if(Sender == SaveAsButton || Sender == SaveAs1)  //判断事件的触发者
       ActiveDraw->SaveAs(SaveDialog1->FileName);
     else       //当该事件不是按钮和菜单项触发的时候，调用 Sender 的 SaveAs 方法
       ((TForm1*)Sender)->SaveAs(SaveDialog1->FileName);
}

void __fastcall TForm1::SaveAs(AnsiString FileName)
{  //画布的 SaveAs 方法 ，需要指定文件名称
   File = FileName;              //保存文件名称到成员 File 中
   Caption = File;               //更换画布窗体的标题
   FileStream = new TFileStream(File,fmCreate);
   Modified = true;              //创建指定名称的文件，文件已经存在则覆盖
   Save();                       //设置 Modified 并调用 Save 方法进行存储
}
```

3.6.3 读取文件

读取文件是存储过程的逆向过程，这个功能是由 MDI 主窗体 TForm2 对象发起的，同时 TForm2 对象需要创建一个新的 TForm1 对象,并指定文件,调用 Form1 对象的 Open 方法打开文件读取数据。

TForm2 的打开文件的菜单和按钮的处理函数如下：

```
void __fastcall TForm2::OpenClick(TObject *Sender)
{
   if(OpenDialog1->Execute())       //打开文件对话框
   {
     if(ActiveDraw)                 //如果有空文件，则将其关闭
       if(ActiveDraw->File == "" && !ActiveDraw->Modified)
```

```
            ActiveDraw->Close();
        TForm1* NewDraw = new TForm1(NULL);        //创建新的画布对象 TForm1
        NewDraw->OpenFile(OpenDialog1->FileName);  //打开指定文件
    }
}
```

TForm1 的 Open 方法如下：

```
void __fastcall TForm1::OpenFile(AnsiString FileToOpen)
{   //打开文件并读入图形数据
    TFileHeader  Header;          //文件标记头的临时变量
    TDataToSave  Data;            //临时变量
    TDraw * DrawItem;             //临时变量
    File = FileToOpen;            //记录打开文件的名称
    Caption = File;               //设置窗体标题
    FileStream = new TFileStream(File,fmOpenReadWrite);       //打开文件
    FileStream->Read(&Header,sizeof(TFileHeader));            //读取文件头
    for(int i = 0; i < Header.DrawCount; i++)
    {                             //根据文件头的信息逐一读入数据
       DrawItem = new TDraw();
       FileStream->Read(&Data,sizeof(TDataToSave));
       DrawItem->Top = Data.Top;
       DrawItem->Left = Data.Left;
       DrawItem->Width = Data.Width;
       DrawItem->Height = Data.Height;
       DrawItem->Pen->Color = Data.PenColor;
       DrawItem->Pen->Style = Data.PenStyle ;
       DrawItem->Pen->Width = Data.PenWidth;
       DrawItem->Brush->Color = Data.BrushColor;
       DrawItem->Brush->Style = Data.BrushStyle;
       DrawItem->Shape = Data.Shape;
       DrawItem->Parent = this;
       Draw->Add(DrawItem);
    }
    Modified = false;             //新打开的文件是未被更改的
}
```

3.7 使用剪贴板

对剪贴板的使用和文件操作一样，也是编辑的一个基本功能，我们可以采用类似文件操作的方式，但在这个例子中，我们采用了 VCL 提供的类 TClipboard 作为基类，派生了一个专门用于该程序的剪贴板类 TDrawClipborad。

对 Windows 剪贴板的使用需要分配全局的堆内存，C++ Builder 中提供了对 TComponent 对象的拷贝、粘贴的方法，但每次只能对一个对象操作，而且是将所有的数据拷贝，并不适合我们的程序，我们将在 TClipboard 的基础上建立自己的专用剪贴板对象。

使用通用拷贝函数 Assign 是一个不错的选择，前面我们也讲过，Assign 方法具有非常好的向后兼容性和扩展性,在 TDrawClipborad 中重载 Assign 和 AssignTo 方法就可以到达这个目的。为了能够正确的交换数据，我们必须向 Windows 注册特定的剪贴板格式来区分其它程序的数据，关于这一部分内容请读者参考 Windows SDK。

另外，剪贴板中交换数据实际上和文件类似，最后使用顺序存储结构的数据，前面定义的 TDataToSave 在这里也同样能够使用。

3.7.1 复制数据

使用 TDrawClipborad 的 Assing 方法将 TForm1 对象的图形数据拷贝到剪贴板中，这一个过程是由 TForm2 对象发起的，下面是它们的源代码：

复制按钮和菜单的 OnClick 事件的处理函数：

```
void __fastcall TForm2::CopyButtonClick(TObject *Sender)
{
   if(ActiveDraw)      //确保当前活动画布有效
   {
     ((TDrawClipborad*)Clipboard())->Assign(ActiveDraw);
     ActiveDraw->UpDateButton();
   }
}
```

TDrawClipboard 的 Assign 方法

```
void __fastcall TDrawClipborad::Assign(Classes::TPersistent* Source)
{                                  //剪贴板的拷贝函数
   TForm1* Src;                    //临时变量
   TDataToSave * DataItem;         //临时变量
   TFileHeader * Header;           //临时变量
   char *DataPoint;                //临时变量
   TDraw*  DrawCopyData;           //临时变量
```

```
    int i;                          //临时变量
    Src = (TForm1*)Source;          //目的使代码中不出现太多的类型转换
    if(Src->Selection->Count == 0)   //指定的画布 Selection 列表为空
      return;                       //则放弃操作
    Open();                         //打开剪贴板，禁止其它程序访问
    Handle = (void*)GetAsHandle(Form2->ClipBoardFormat);
    if(Handle)                      //获取相应数据格式的句柄，将原有的
      GlobalFree(Handle);           //数据清除
    Handle = GlobalAlloc(GMEM_SHARE|GMEM_MOVEABLE,  //重新分配全局内存
           sizeof(TDataToSave) * Src->Selection->Count +
           sizeof(TFileHeader));                  //按照图形数量和标记头
    DataPoint = (char*)GlobalLock(Handle);        //所需要的大小分配内存
    Header = (TFileHeader*)DataPoint;             //锁定内存并转化为标记头指针
    Header->DrawCount = Src->Selection->Count;    //用于记录拷贝图形的数量
    DataItem = (TDataToSave*)(DataPoint + sizeof(TFileHeader));
    for( i = 0 ; i < Src->Selection->Count; i++ )     //从标记头的结尾处开始
    {                                             //写入被拷贝图形的相应数据
      DrawCopyData = (TDraw*)Src->Selection->Items[i];
      DataItem[i].Top = DrawCopyData->Top;        //以下是需要存储所必须的数据
      DataItem[i].Left = DrawCopyData->Left;      //也是图形对象的用户数据
      DataItem[i].Width = DrawCopyData->Width;
      DataItem[i].Height = DrawCopyData->Height;
      DataItem[i].PenColor = DrawCopyData->Pen->Color;
      DataItem[i].PenWidth = DrawCopyData->Pen->Width;
      DataItem[i].PenStyle = DrawCopyData->Pen->Style;
      DataItem[i].BrushColor = DrawCopyData->Brush->Color;
      DataItem[i].BrushStyle = DrawCopyData->Brush->Style;
      DataItem[i].Shape = DrawCopyData->Shape;
    }                                             //释放内存指针，将内存的
    GlobalUnlock(DataPoint);                      //句柄交给剪贴板
    SetAsHandle(Form2->ClipBoardFormat,(int)Handle);
    Close();                                      //关闭剪贴板
}
```

3.7.2 粘贴数据

粘贴数据是复制数据的逆向过程，当需要从剪贴板中粘贴数据的时候，只需要调用 TForm1 的 Assign 方法即可，在 TForm1 中我们并没有重新定义 Assign 方法，当使用

TDrawClipboard 对象作为 TForm1 的复制源的时候，最终实际调用的是 TDrawClipboard 的 AssignTo 方法来完成数据复制的，这个机制我们在前面章节中已经讲过了，这里不在赘述，TDrawClipboard 的 AssignTo 方法的代码如下：

```
void __fastcall TDrawClipborad::AssignTo(Classes::TPersistent* Dest)
{                                                    //剪贴板的赋值方法
    TForm1* DestDraw;                                //是和拷贝方法相反的
    TDataToSave * DataItem;                          //过程
    TFileHeader * Header;
    char *DataPoint;
    TDraw*  DrawCopyData;
    int i;
    DestDraw = (TForm1*)Dest;
    Open();                                          //打开剪贴板，获取所需
    Handle = (void*)GetAsHandle(Form2->ClipBoardFormat);
    if( !Handle )                                    //数据格式的句柄
    {                                                //为空表示没有相应格式的
       Close();                                      //数据，关闭剪贴板并返回
       return;
    }
    DataPoint = (char*)GlobalLock(Handle);           //锁定内存为指针
    if(!DataPoint)                                   //若锁定后指针无效
    {                                                //说明拷贝的数据块
       GlobalUnlock(DataPoint);                      //无效，释放指针、
       Close();                                      //关闭剪贴板并返回
       return;
    }
    Header = (TFileHeader*)DataPoint;                //读取标记头
    if(Header->DrawCount == 0)                       //若复制的图形数量
    {                                                //为 0，表示数据无效
       GlobalUnlock(DataPoint);                      //释放指针，关闭剪贴板
       Close();                                      //并返回，本应加入对标记头
       return;                                       //验证的代码，这里省略
    }                                                //从标记头结尾处开始
    DataItem = (TDataToSave*)(DataPoint + sizeof(TFileHeader));
    DestDraw->UnSelected();                          //读取拷贝的数据，并清除
    for( i = 0 ; i < Header->DrawCount; i++ )        //原有被选择图形
    {
       DrawCopyData = new TDraw();
```

```
        DrawCopyData->Top = DataItem[i].Top;
        DrawCopyData->Left = DataItem[i].Left;
        DrawCopyData->Width = DataItem[i].Width;
        DrawCopyData->Height = DataItem[i].Height;
        DrawCopyData->Pen->Color  = DataItem[i].PenColor;
        DrawCopyData->Pen->Width  = DataItem[i].PenWidth;
        DrawCopyData->Pen->Style  = DataItem[i].PenStyle;
        DrawCopyData->Brush->Color  = DataItem[i].BrushColor;
        DrawCopyData->Brush->Style =  DataItem[i].BrushStyle;
        DrawCopyData->Shape = DataItem[i].Shape;
        DrawCopyData->Parent = DestDraw;          //设置画布为指定的Dest
        DrawCopyData->FSelected = true;           //将拷贝获取的图形设置为
        DestDraw->Draw->Add(DrawCopyData);        //新的被选择对象
        DestDraw->AddSelect(DrawCopyData,false);
    }
    GlobalUnlock(DataPoint);                      //释放指针，关闭剪贴板
    Close();
}
```

TForm2 的粘贴事件的处理函数如下：

```
void __fastcall TForm2::PasteButtonClick(TObject *Sender)
{
    if(ActiveDraw)                                //确保活动文档有效
    {
      ActiveDraw->Assign((TDrawClipborad*)Clipboard());
      ActiveDraw->UpDateButton();
    }
}
```

3.8 总结

在这个例程中，我们使用了多个对象类来完成最终的功能，每一个对象之间都存在着一定的逻辑关系，从这段代码中也可以看出，并不是所有的关系或者功能都一定要使用属性来完成，属性只是一种方式，最主要的是我们在设计程序的过程中，将对象的状态作为了程序的核心，围绕着状态之间的变化规律以及约束规律来进行代码的设计。

在应用程序中并没有一种特定的方式来确定如何设计才是合理，只要达到的最终的目的，而且可扩充性强，维护方便，那么就是合理的代码，在这个例子中 TForm1 的鼠标移动的处理方法中，我们完全可以采用类似 Sells 中的处理方法，引入一个鼠标消息的子

处理函数的事件，当不同的工具被选择的时候，更换不同的鼠标消息子处理函数句柄，这对于程序拥有很多绘图工具的时候是非常有利的，这个例子中并没有这样作，因为绘图工具只有三个，处理起来并不复杂，有兴趣的读者可以将 Fotm1 的 OnMouseUp 方法中的部分代码提取出来，声明一个用于鼠标消息子处理事件的类型，再为每一个种绘图工具编写鼠标处理子函数，用类似 Sells 例子中键盘消息处理的方式来实现。

在这个例子中我们缺少的是编辑中另一支柱，Redo（重复）和 Undo（撤销），这需要使用到另一种数据结构“栈”，C++ Builder 中为 TStack 类，重复和撤销是编辑中稍微复杂的功能，不仅需要预先规划好重复和撤销的路径，还有对每一种可能的操作定义相应的数据格式，代码比以上介绍的编辑功能要复杂的多，代码量也会大一些，我们在这个例子中将不再示范。

除此之外，本例程中还有一些辅助功能的方法，上面没有介绍，比如程序关闭时的 CloseQuery 方法，用于保证更改的数据在退出程序之前需要提醒用户保存等，我们不再一一介绍，本书所附光盘中有本程序的所有代码，读者自行阅读。

4 小节

本章从程序设计方面应用面向状态的技术作了两个简单是示例，实际中的程序比这两个例子要复杂的多，而且对象之间的关系也复杂的多，本章的例子仅仅是在几个方面将这种设计程序的思想和方法作个示范，这两个例子的代码并不是优秀的代码，只能给读者在将 VCL 组件的设计技术应用到应用程序的设计中起到一个抛砖引玉的作用。

本章将是本书的最后的内容，希望整本书给读者会在某些方面有所帮助。C++ Builder 是一个非常优秀的程序设计工具，C++语言也是一种非常强大的程序设计语言，其语言的博大精深是本书所难以叙述，涉及到的方方面面内容。Windows 系列操作系统是目前最为流行的桌面操作系统，任何一个程序员都不可能全面掌握，但是作为应用程序的设计师，至少应当对 Windows 的底层原理有所了解。

程序设计是一个系统的工程，并非一种两种技巧和技术所能决定的，任何一种应用的设计者都应该是该应用领域的专家，否则不可能出现 PhotoShop、AutoCAD 等这些非常优秀的软件。任何一个程序员都应该将精力放在需要解决的问题上，其次才是如何使用工具来解决。本书虽然是就 C++ Builder 的使用来编写的，但最终并不是为了向读者介绍如何使用 C++ Builder，这方面的书籍比较多，而且也相当全面，本书只是希望通过对 C++ Builder 的介绍，能够让读者站在一个更高的层次来分析、来组织、来面对程序设计，这样思路会更加开阔，对问题的分析也会更加透彻。

世界上的万事万物本来就是相通的，其运动的基本规律是一致的，我国的古代的哲学思想是这样认为的，而西方的近代哲学思想，包括唯物主义哲学也是这样认为的。对于一个程序设计者而言，主导思想往往才是关键。

本书的所有内容到这里就全部结束了，作者希望本书能够对读者有所帮助，但以作者

的水平，目前力所能及的只有这些，如果一个 C++ Builder 的初学者通过阅读本书而掌握了如何使用__property 关键字，那么作者就已经心满意足了。对于本书中出现的错误、遗漏、疏忽之处，希望读者能够谅解并给予批评指正。

参考资料及书籍：

《C++ Builder 连接帮助》
Borland 公司随机产品
《Borland C++ Builder 5 高级开发技术》
中国水利水电出版社 李冬 王宏等著
《面向对象的理论与 C++实践》
清华大学出版社 王燕 编著
《C/C++程序设计大全》
机械工业出版社 H.M.Deitel D.J.Deitel 著
薛万鹏 等译
《C++ Builder 5 开发人员指南》
电子版由 China-pub.com 提供
梁志刚、汪浩、康向东、刘存根等译