《面向对象程序设计实践》课程设计

总结报告

院　（系）： 计算机科学学院

专业班级： 计科 12105 班

学　　号： 202106341

姓　　名： 许健跃

指导教师： 詹泽梅

课设时间： 2022.12.5 - 2022.12.22

课设地点： 4 教 2 号机房和腾讯会议

2022 – 2023 学年 第 一 学期

目录

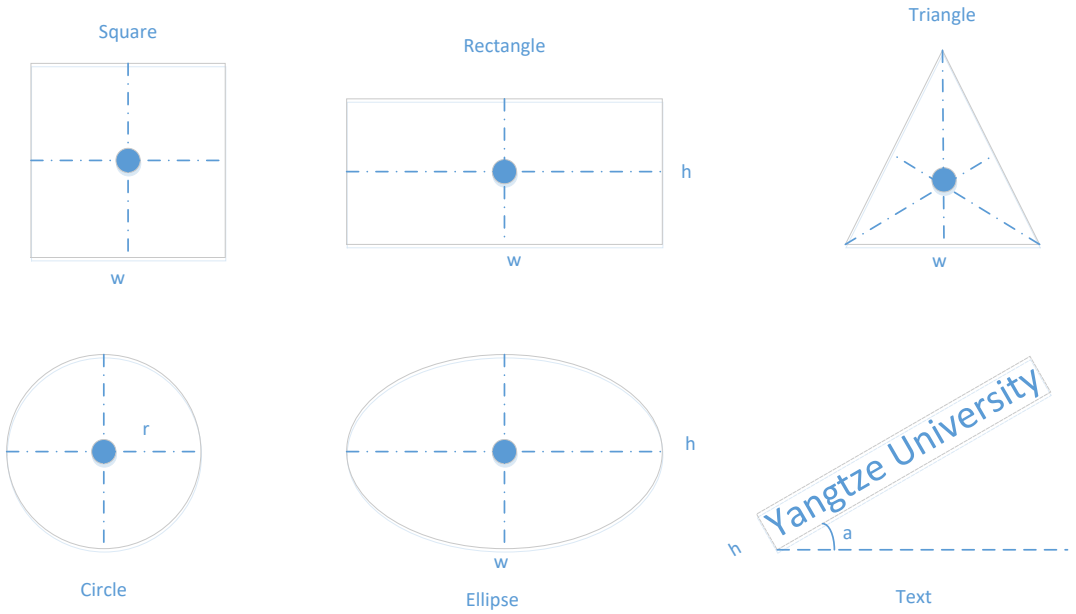
一、课程设计目的

（1） 感悟具体开发流程的开发细节。
（2） 通过使用 MFC（Microsoft Foundation Classes）微软基础类进行开发，加强对于 OOP（面向对象程序设计）的理解。
（3） 培养宏观架构能力和具体完善每一个模块的编码能力。
（4） 巩固 C/C++语言的基础知识。
（5） 培养书写开发报告的能力。

二、课程设计的内容与设计思路

1.课程设计的内容

本课程设计要求开发一个简单的图形编辑系统，可以添加、修改与删除图形元素，以形成图形画面。系统所支持操作的六种图元类型如下：



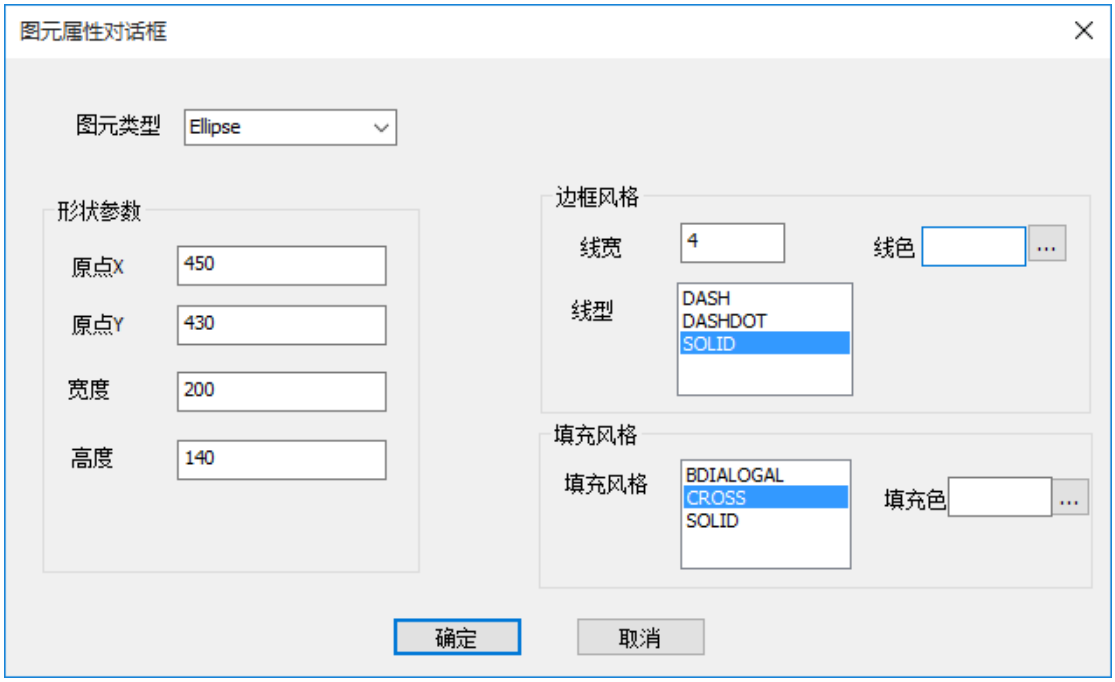
上图中，每个具体图元中间的小黑点代表该图元的原点，w 表示宽度 width，h 表示高度 height，r 表示半径 radius，a 表示字符串的逆时针旋转角度 angle。其中添加，删除与修改的具体实现截图见（四、运行效果图）。

另外这些上诉这些图元是否填充，用什么模式填充，具体要求如下：

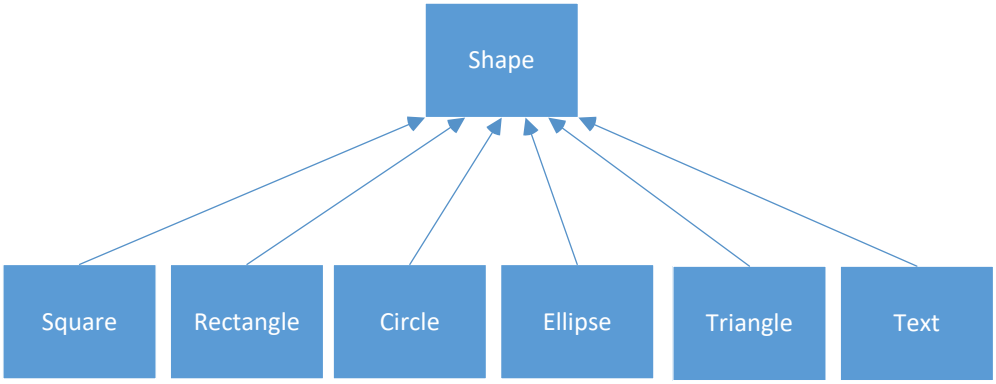
1、采用单文档方式，文档中存储图形画面的各个图元数据，视图负责图形的绘制。
2、文档支持图形的序列化（连载），提供新建、打开、保存等操作。
3、视图除了绘制图形，还提供图形交互，能够按住 Ctrl 键再鼠标左键单击来创建图元，鼠

标左键双击编辑修改图元属性，鼠标右键双击删除图元。

4、图元创建与修改时的参数由参数对话框来编辑。创建时以鼠标左击时光标的所在位置作为基点来创建图元。



5、使用图元基础类 shape 作为所有六个图元类的基类，设计派生各个具体的图形类，要求支持上述功能。



2.设计思路

从程序外部看，需要实现的功能有：

（1） 编辑图形，包括图形对象的新建、删除和修改等 3 项功能。

（2） 文件操作，要求实现程序菜单中文件的新建、打开、保存、另存为、关闭等功能。

另外，需要对 6 种图形进行类的设计，包括：正方形、矩形、圆、椭圆、正三角形、文本等。设计内容参见表 1。

表 1.设计内容

功能	操作	类/方法、属性	功能说明	隐含/相关功能
1. 编辑图形	**包括图形对象的新建、删除和修改等操作**			

1.1 新建图形	Ctrl+鼠标左键单击	CDrawingView::OnLButtonDown()	在鼠标点击处创建图形（6 种）	存储图形 显示图形 设置图形属性
1.2 删除图形	鼠标左键双击	CDrawingView::OnLButtonDblClk()	删除鼠标点击处所选中的图形	选中图形
1.3 修改图形	鼠标右键双击	CDrawingView::OnRButtonDblClk()	修改鼠标点击所选中图形的属性	选中图形 设置图形属性
1.4 存储图形	（隐含）	CDrawingDoc 类	存储 6 种图形对象，应分配到文档类中存储	
1.5 显示图形	（隐含）	CDrawingView::OnDraw()	绘制文档内存储的所有图形	6 个图形类需分别实现各自的绘图方法 Draw()
1.6 选中图形	（隐含）	6 个图形类的 IsMatched()	判断鼠标点击处是否落在图形内	6 个图形类需分别实现 IsMatched()
1.7 设置图形属性对话框	（隐含）	CShapeDlg 类	创建或修改图形时，设置图形属性	
2. 文件操作	**实现程序菜单中文件的新建、打开、保存、另存为、关闭等功能。 CDocument 已实现并封装了这些功能，只需实现 2 个函数：Serialize()、DeleteContents()。 另外，在图形的新建、删除和修改时，应调用 CDocument::SetModifiedFlag()方法，说明文档已被修改；这样，在文件的新建、打开时会提示：文件已修改，是否保存？**			
2.1 序列化	（隐含）	CDrawingDoc::Serialize()	文档内图形对象的文件读、写操作	6 个图形类需分别实现 Serialize()
2.2 删除文档内容	（隐含）	CDrawingDoc::DeleteContents()	在文件的新建、打开时会自动调用，该函数负责清空文档内容（即所有图形对象）	
2.3 释放动态内存对象	（隐含）	CDrawingDoc::~CDrawingDoc()	关闭程序时，delete 删除由 new 分配的所有对象	
3 图形类	**设计并实现正方形、矩形、圆、椭圆、正三角形、文本等 6 个图形类**			
3.1 正方形	（隐含）	CSquare 类	包括各自的属性以及构造函数、Draw()、Serialize()、IsMatched()等方法	
3.2 矩形	（隐含）	CRectangle 类		
3.3 圆	（隐含）	CCircle 类		
3.4 椭圆	（隐含）	CEllipse 类		
3.5 正三角形	（隐含）	CTriangle 类		
3.6 文本	（隐含）	CText 类		

设计一个抽象类 CShape 作为 6 个图形类的基类，设计类图参见图 2。

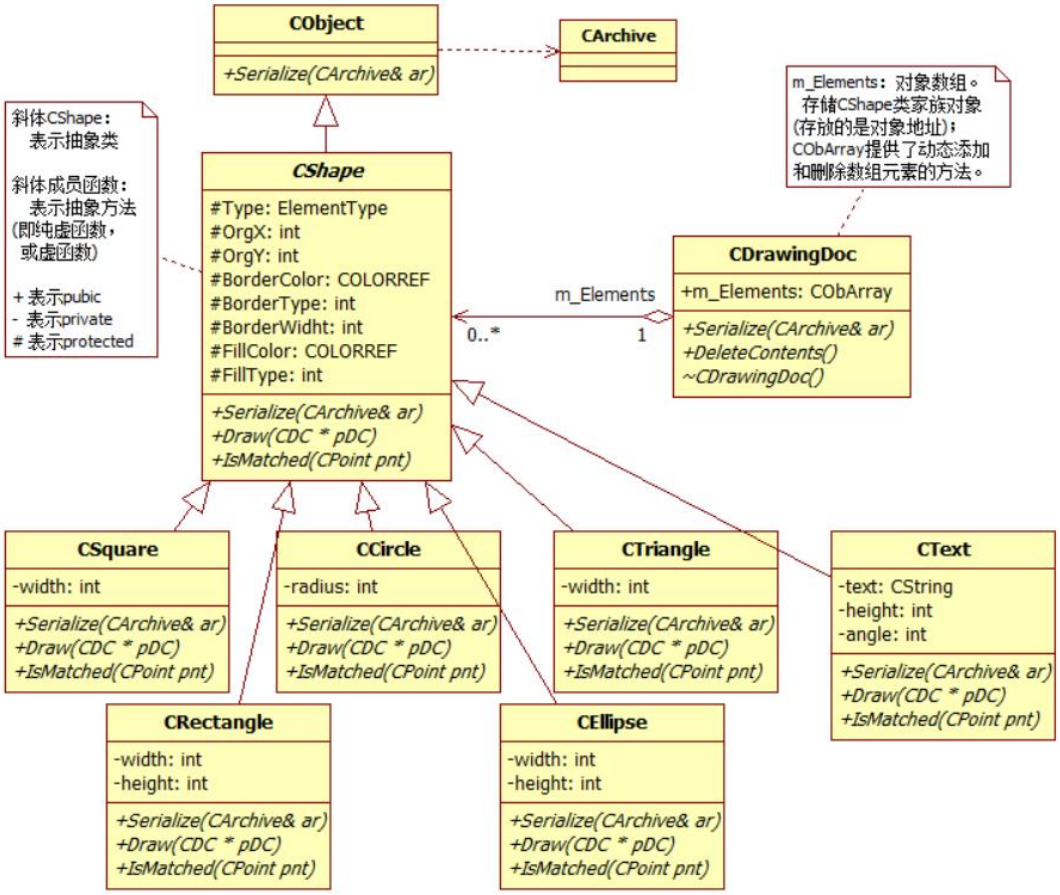


图 2. 图形类及其在文档中的存储设计

图形类设计要点：

（1） 采用 new 动态创建图形对象，因为图形对象由用户的操作创建，编程时不可能预知用户想要创建哪个图形和多少个图形；

（2） 图形对象存储在 CDrawingDoc 类中，且可以动态添加和删除，故应该采用集合类（指 MFC 的 CArray、CList、CMap 等类）来存储（选择 CObArray 类较好），且应将 6 个图形类设计成一个类族；

（3） 支持序列化，即 Serialize()方法，故应从 MFC 的 CObject 类派生；

（4） Serialize()、Draw()、IsMatched()等方法应该采用动态联编，即虚函数。

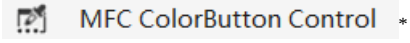
接着我们需要创建对话框说明与类，具体设计如下：

创建图元属性对话框并添加 CShapeDlg 类，操作如下：

（1） 在“解决方案资源管理器”的“资源文件”中双击“Drawing.rc”，可打开“资源视图”；

（2） 在“资源视图”的“Dialog”（对话框）上，点击右键菜单“插入 Dialog”，可插入并打开新对话框，更改对话框属性 ID 为“IDD_SHAPE_DLG”；

（3） 在新对话框中，点击右键菜单“添加类”，然后在添加类向导中输入类名：CShapeDlg；

添加控件时，其中的 2 个**颜色控件**推荐使用： 

*。*注：在本文的设计和实现中并未使用。

设计类图，参见图 3。

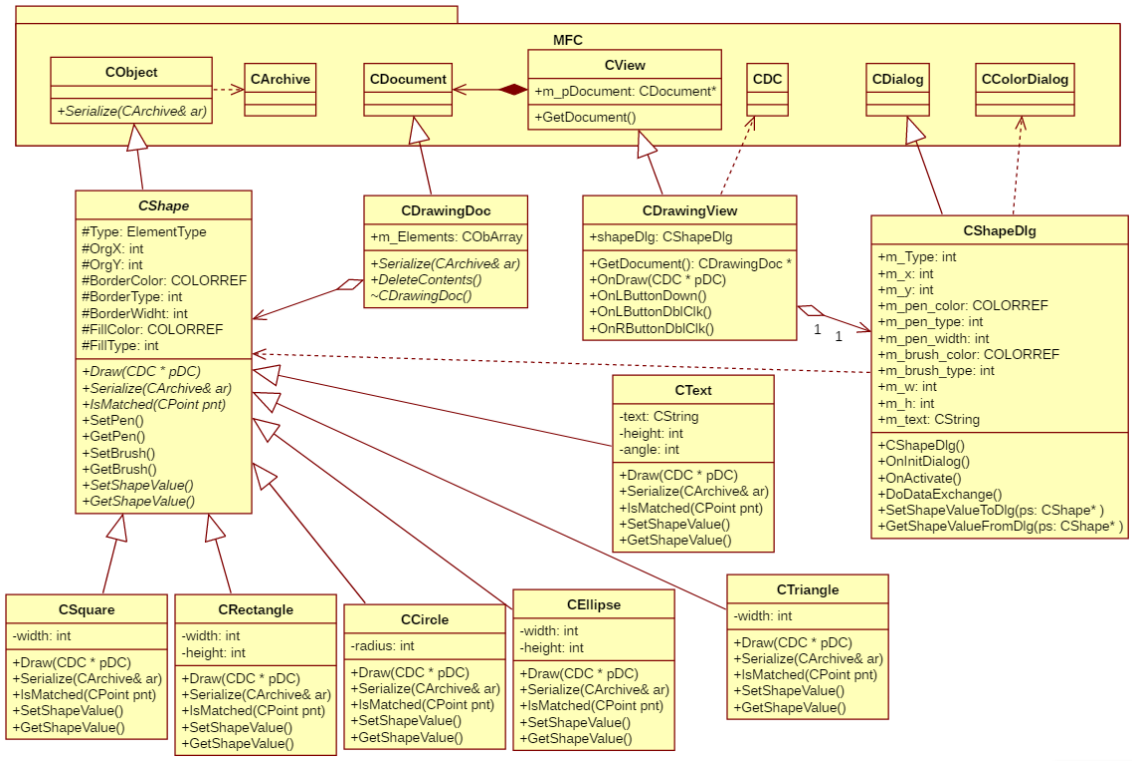


图 3. 设计类图（仅供参考）

设计说明：

（1） 在 CdrawingView 类中增加一个 CShapeDlg 类对象成员 shapeDlg，这样在程序启动并创建 CdrawingView 类对象的同时，就能创建对象 shapeDlg（即图元属性对话框）；这样设计的好处是，避免每次使用图元属性对话框时都要重新创建该对话框，从而减少时间上的开销；

（2） 在新建图元对象或修改图元对象时，可使用 shapeDlg.DoModal()来打开对话框；

（3） 对话框的“确定”和 “取消”按钮，采用默认代码即可，即在 CDialogEx::OnOK()；和 CDialogEx::OnCancel()；之前不用添加代码；

（4） 在 CShape 及其子类中添加函数以便与图元属性对话框进行数据交换，如：画笔、画刷和图元形状等参数的交换。

图元对话框及与图元对象的数据交换：创建新图元对象的过程，参见图 4。

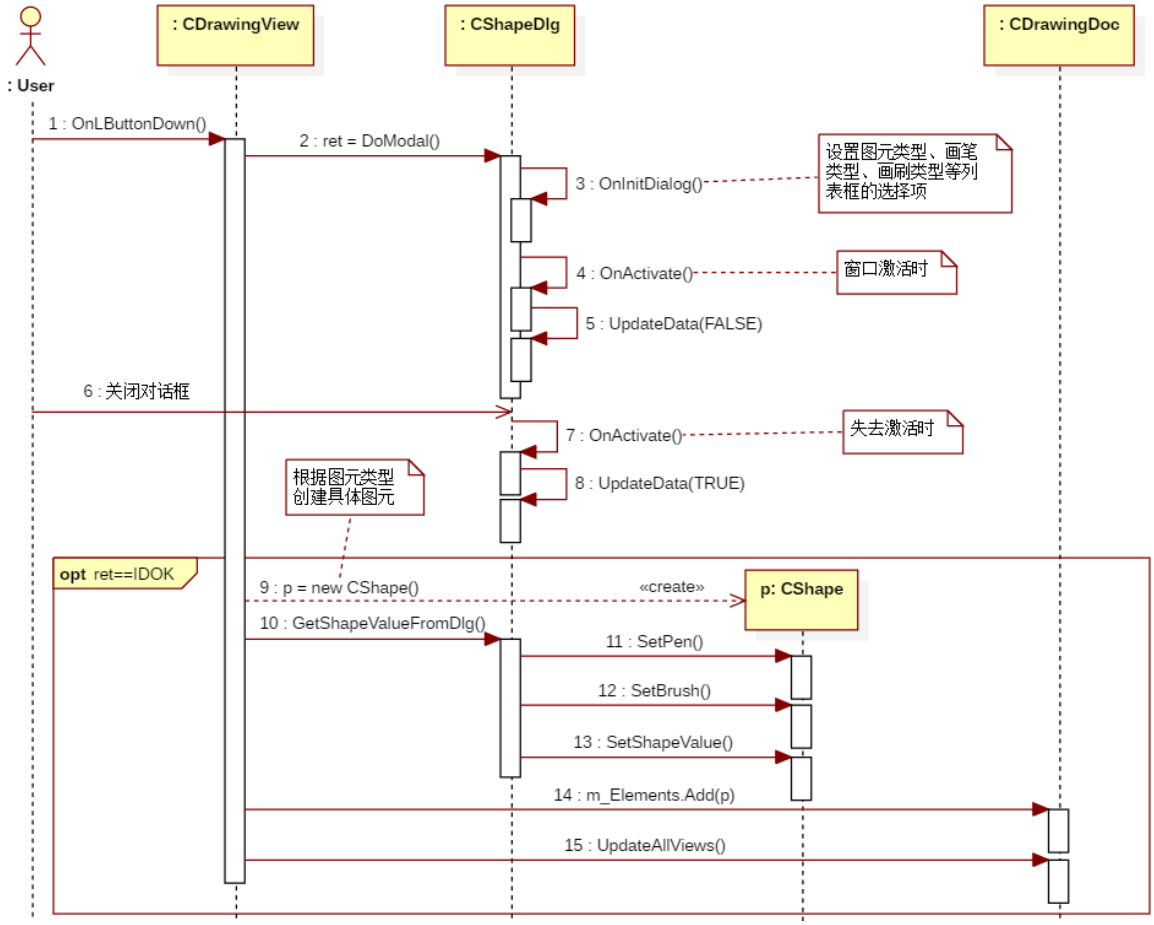


图 4 添加新图元时，图元对象与对话框交换数据的 UML 时序图

说明：

（1） 在 CDrawingView::OnLButtonDown()函数中创建新图元，整个时序图就是说明该函数的实现过程；图中从上到下表示时间顺序（故称为时序图或顺序图），函数（或消息）前的序号表示执行顺序。

（2） CShapeDlg::OnInitDialog()函数在对话框的类向导“虚函数”中添加，CShapeDlg::OnActivate()函数在对话框的类向导“消息”WM_ACTIVATE 处添加。

（3） 图中函数调用箭头指向哪个类的对象，就表示该函数属于（或者说分配到）哪个类。

三、 程序实现过程与细节

1.声明图元类型

对于本程序而言，我们先从如何定义一个图元类型来介绍它的具体实现过程。由于在本程序中，我们需要实现三角形，圆形，矩形等六种图元，为了使得他们具有一部分相同特性的同时又具备特异性，因此我们先定义一个基类型（CShape），它是继承于 CObject 的一个派生类型，便于我们后面对于图元进行 CObject 所支持的各种操作。CShape 的具体声明详见（六、主要代码清单）。有了 CShape 图形基类的声明之后，我们就可以通过 CShape 的派生类来声明其他图元类型。正方形，三角形，矩形，圆形，椭圆，字符串分别为 CSquare，

CTriangle，CRectangle，CCircle，CEllipse，CText。

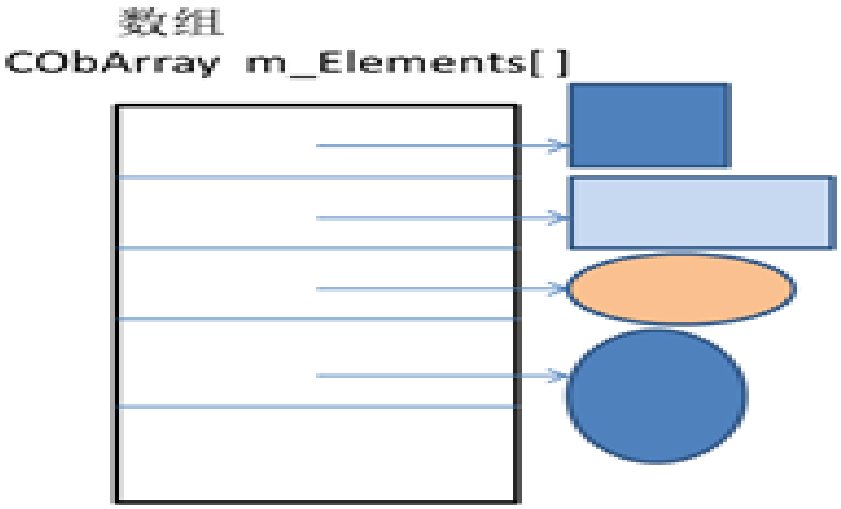
每一个具体图元都有以下几个声明，分别为：

1. `void Draw(CDC* pDC); //将此图像显示在视图窗口中`
2. `bool  IsMatched(CPoint pnt); //判断该图元是否被鼠标选中`
3. `virtual void Serialize(CArchive& ar); //支持该图元进行序列化`
4. `DECLARE_SERIAL(具体类型) //支持序列化的类型`
5. `virtual void SetValue(ElementType t, int x, int y, int w, int h, 具体类型); //由对话框设置该图元的各种数值`
6. `virtual void GetValue(ElementType& t, int& x, int& y, int& w, int& h, 具体类型的引用); //将该图元的各种数值传到对话框`

2.存储图元的数据结构以及图元在视图显示的绘制逻辑

有了各个图元类型的具体声明，接下来我们考虑如何存储这些图元类型的变量。在文档类 CDrawingDoc.h 中恰好有一种 CObArray 的变量，它其实是一个 CObject 数组，里面的每一个变量都是 CObject 类型，而我们的图元类型又是继承与 CObject 所以 CObArray 可以用来存储我们的每个图元。

存储样图如下所示：



CObArray 数组的存储数据结构

解决了存储问题，接下来我们解决如何有序的将 CObArray 数组中的元素显示在用户的视图区域。

首先，我们先在 CDrawingView 类中编写相应的 OnDraw 函数，该函数用于将 CObArray 数组的每一个元素都显示在视图窗口当中。

```
for (int i = 0; i < pDoc->m_Elements.GetCount(); i++)
    {
        CShape* p = (CShape*)pDoc->m_Elements[i]; //p 指向数组中的元素
        p->Draw(pDC); //调用 p 所指向元素的 Draw 函数
    }
```

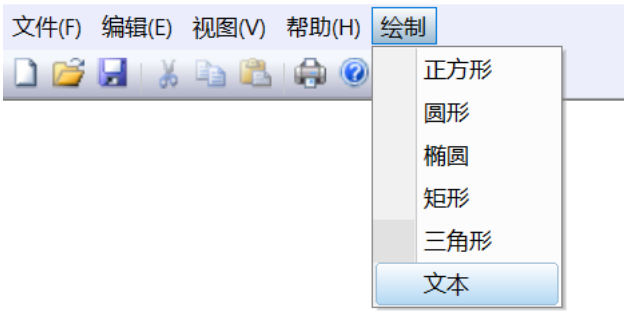
这样，我们通过一个循环用指向基类的指针 p 指向不同的图元类型，再通过调用每一具

体图元的 Draw 函数，从而实现了在视图中显示图像的问题。

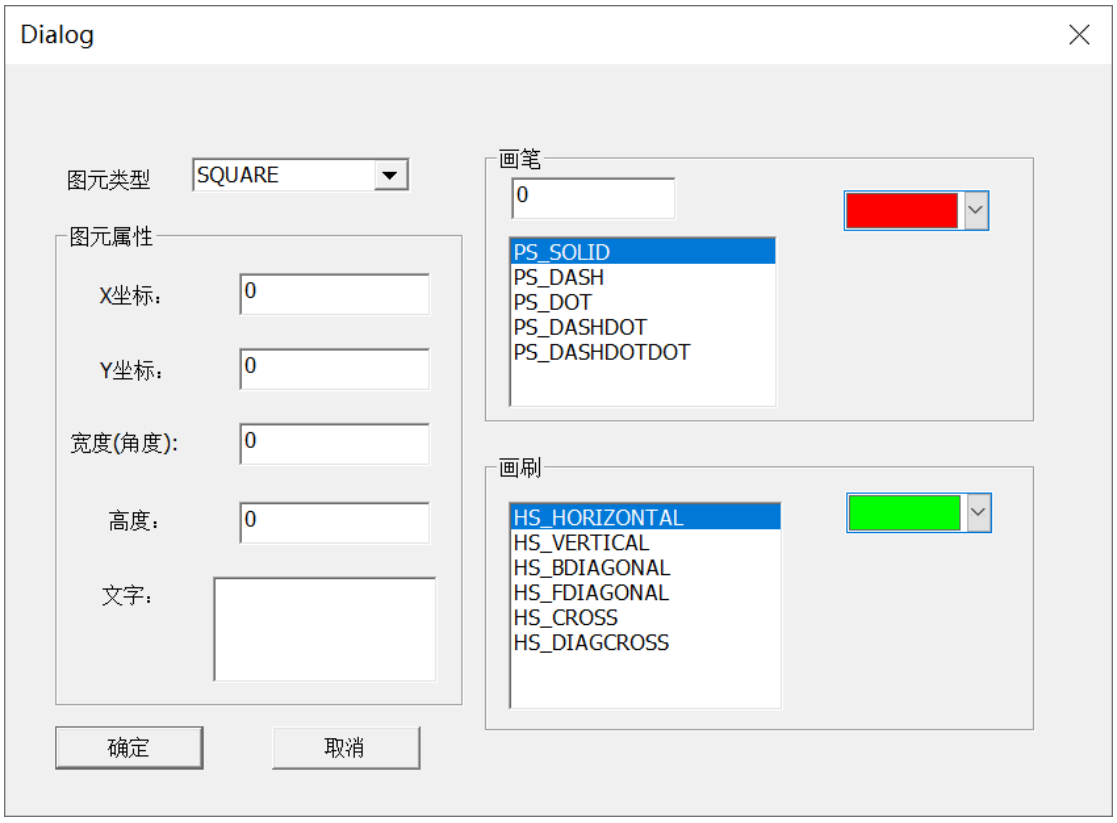
3.视图窗口菜单绘图功能的具体实现

有了一整套在用户视图窗口显示图像的具体逻辑，接下来我们就要运用这套逻辑去实现本课程设计所要求的两种功能即：

1. 通过单击用户视图界面上方的菜单来实现图元在视图窗口的显示。



2. 通过在空白区域输入 Ctrl+鼠标左键，调出一个新的对话框来进行图元各项属性的设置，从而在用户视图窗口绘制出具体的图像。

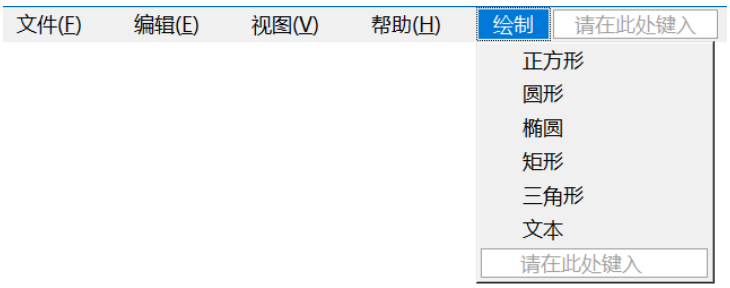


3. 在已经绘制的图形上输入 Ctrl+鼠标左键，调出一个新的对话框来进行图元各项属性的设置，从而在用户视图窗口对已经绘制的图形进行修改。

我们先来讲解如何实现第一种功能，既通过单击用户视图界面上方的菜单来实现图元在视图窗口的显示。首先我们在资源视图中找到 Menu 菜单下的 IDR_MAINFRAM 视图，在上方添

加绘制一栏，点开后再分别添加正方形，圆形，椭圆，矩形，三角形，文本六项菜单。

如下图所示：



通过 IDR_MAINFRAM 视图添加绘制菜单

然后对于每项具体的菜单我们逐一添加其消息处理函数，下面以正方形举例。

```
void CDrawingView::OnSquare()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticSquareX = 100, StaticSquareY = 100;//正方形的中心点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CSquare(StaticSquareX, StaticSquareY, 100));
    StaticSquareX += 25;
    StaticSquareY += 50; //下一个正方形的中心点坐标值增加
    pDoc->UpdateAllViews(NULL);
}
```

正方形绘制菜单的消息处理函数

该函数是将一个新 new 出的正方形对象添加到我们上面所定义的 m_Elements 数组当中，接着刷新窗口，然后系统将自动调用我们上面所定义的 CDrawingView 类中编写相应的 OnDraw 函数，从而将 m_Elements 数组的每个具体图元元素显示在用户的视图窗口当中。然后继续照葫芦画瓢，用同样的逻辑，便可完善除正方形图元以外的其他五种图元的显示，第一种功能即（通过单击用户视图界面上方的菜单来实现图元在视图窗口的显示），得到实现。

4.双击鼠标左键删除图像的实现逻辑

在实现第二种功能之前，我们不妨先完善一下用户视图窗口的其他功能。以下将讲述如何实现在视图窗口的空白处双击鼠标左键删除图像。

想要实现这个功能，我们得有一套判定逻辑，用于判断鼠标是否位于该图形内，基于此逻辑我们给出每一种图元 IsMatched 函数的具体定义。以下以正方形举例：

```
bool CSquare::IsMatched(CPoint pnt)
{
  if ((pnt.x >= OrgX - width / 2) && (pnt.x <= OrgX + width / 2) && (pnt.y >=
OrgY - width / 2) && (pnt.y <= OrgY + width / 2))
      return true;
  else
      return false;
}
```

正方形的 IsMatched 函数

该函数通过正方形的中心点坐标，以及正方形的宽度，和鼠标当前的位置（OrgX，OrgY），用一个数学表达式判断鼠标是否位于正方形内，从而判定正方形是否被选中。

有了判定鼠标的逻辑，接下来我们实现双击删除用户视图界面图像逻辑。我们可以通过对 CDrawingView 类添加鼠标双击的事件处理函数实现用户视图界面图像的删除。

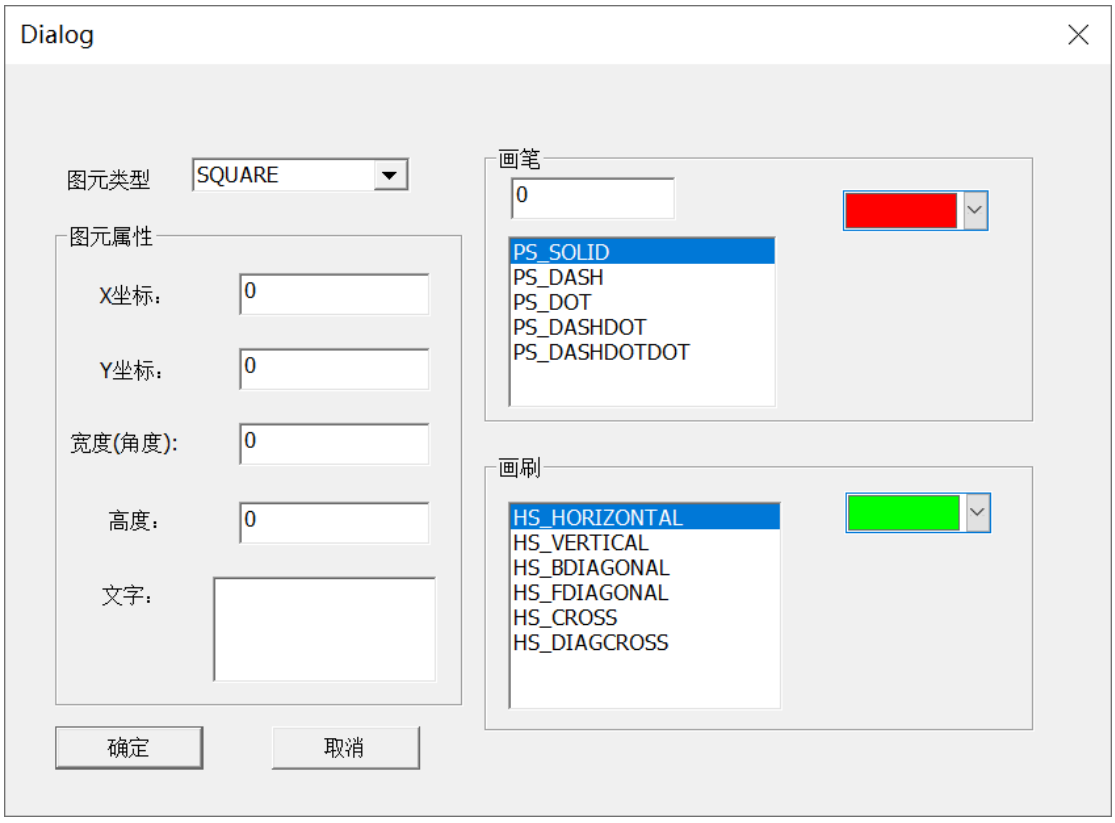
```
void CDrawingView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    CDrawingDoc* pDoc = GetDocument();
    CShape* p;
    for (int i = pDoc->m_Elements.GetSize() - 1; i >= 0; i--)
    {
        p = (CShape*)pDoc->m_Elements[i];
        if (p->IsMatched(point))
        {
            pDoc->m_Elements.RemoveAt(i);
            delete p;
                pDoc->UpdateAllViews(NULL);
            break;
        }
    }
    CScrollView::OnLButtonDblClk(nFlags, point);
}
```

鼠标左键双击删除图像的事件处理函数

该函数通过获取当前图像的指针，判定当前图像是否被选中，如果被选中，则将该图像的变量从 m_Elements 数组中删除，最后刷新界面，系统重新自动执行 OnDraw 函数，将 m_Elements 数组中的元素重新绘制在用户视图界面中，从而完成对于鼠标选中图像的双击删除操作。

5.设计与完善对话框界面

再进行对话框相关功能实现之前，我们不妨先完善我们的对话框界面。在资源视图的 Dialog 文件下新建 IDD_ShapeDlg 视图，完成如下布局：



对话框界面的按钮布局

对于对话框当中的每一个按钮，我们都先添加其对应的变量和消息处理程序，变量对照表如下图所示：

成员变量(V):

控件 ID	类型	成员
IDC_COMBO1	CComboBox	m_CmbShapeType
IDC_EDIT1	int	m_x
IDC_EDIT2	int	m_y
IDC_EDIT3	int	m_width
IDC_EDIT4	int	m_height
IDC_EDIT5	CString	m_text
IDC_EDIT6	int	m_PenWidth
IDC_LIST1	CListBox	m_LstPenType
IDC_LIST2	CListBox	m_LstBrushType
IDC_MFCCOLORBUTTON1	CMFCColorButton	m_ColorPen
IDC_MFCCOLORBUTTON2	CMFCColorButton	m_ColorBrush

变量对照表

各按钮的消息处理函数详见（六、主要代码清单）

6.通过对话框界面双击修改图元数据

上面我们完善了对话框界面，接下来我们实现用户视图窗口的第二个功能即（在选中的图像上双击鼠标右键修改图形）

首先，我们先在 CDrawingView 类中添加双击鼠标右键的消息处理函数：

```
void CDrawingView::OnRButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    CDrawingDoc* pDoc = GetDocument();
```

```
    CShape* p;
    for (int i = pDoc->m_Elements.GetSize() - 1; i >= 0; i--)
    {
        p = (CShape*)pDoc->m_Elements[i];
        if (p->IsMatched(point))
        {
            CShapeDlg d1;
            d1.pshape = p;
            if (d1.DoModal() == IDOK)
            {
                d1.DlgToShape();
            }
            pDoc->UpdateAllViews(NULL);
            break;
        }
    }

    CScrollView::OnRButtonDblClk(nFlags, point);
}
```

双击鼠标右键的消息处理函数

该函数的整体逻辑和双击鼠标左键类似，两者只在被选中时所执行的操作不同，在该函数中，当我们选中了某一图形，便会弹出对应的对话框 Dlg，调用 DlgToShape 函数，从而对于图元的各个属性进行重新设置，最后刷新窗口，系统自动调用 OnDraw 函数，用户界面视图得到更新。

7.通过对话框添加图形

此功能的逻辑与上述功能逻辑类似，我们唯一没有解决的问题是如何输入 Ctrl+鼠标左键调出对话框，事实上该功能的实现也非常简单，我们只需要为 CDrawingView 类添加消息处理函数 OnLButtonDown：

```
void CDrawingView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    if ((nFlags & MK_CONTROL) == MK_CONTROL)//Ctrl键按下
    {
        CShapeDlg mydlg;
        mydlg.pshape = NULL;
        if (mydlg.DoModal() == IDOK)
        {
            switch (mydlg.Type)
            {
            case SQUARE:mydlg.pshape = new CSquare(); break;
            case RECTANGLE:mydlg.pshape = new CRectangle(); break;
            case CIRCLE:mydlg.pshape = new CCircle(); break;
            case ELLIPSE:mydlg.pshape = new CEllipse(); break;
```

```
                case TRIANGLE:mydlg.pshape = new CTriangle(); break;
                case TEXT:mydlg.pshape = new CText(); break;
                }
                mydlg.DlgToShape();
                CDrawingDoc* pDoc = GetDocument();
                pDoc->m_Elements.Add(mydlg.pshape);
                pDoc->UpdateAllViews(NULL);
            }
        }

        CScrollView::OnLButtonDown(nFlags, point);
    }
```

Ctrl+鼠标左键打开对话框的消息处理函数

该函数通过打开对话框，读取对话框中用户所选择的图元类型，以及图元的各种数据，new 出一个全新的图元对象添加到 m_Elements 数组当中，最后刷新界面，系统自动调用 OnDraw 函数用户自己添加的图元在用户视图界面显示。

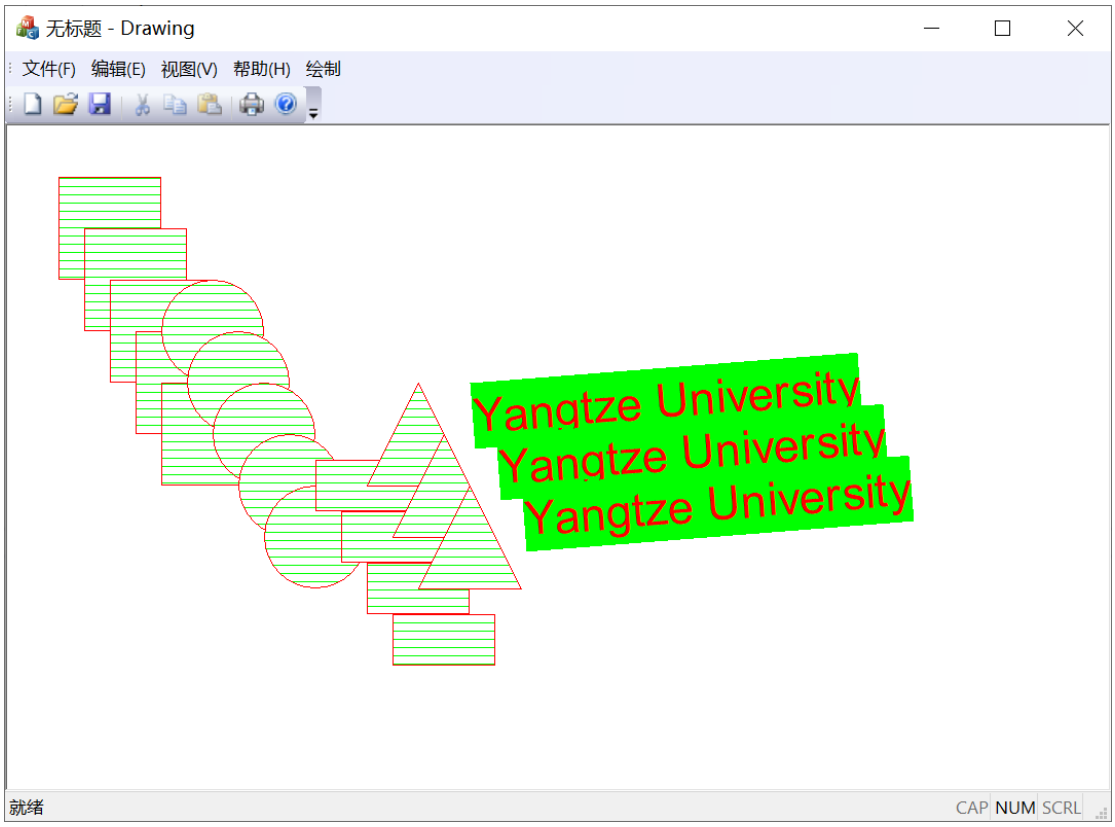
8.保存与读取

完善了以上各种功能之后，最后我们讲述如何实现保存与读取功能。在 1.声明图元类型中，我们已经声明了具体图元的支持序列化操作，所以对于每一具体的图元类型，我们可以编写其对应的序列化函数具体代码见（六、主要代码清单）。

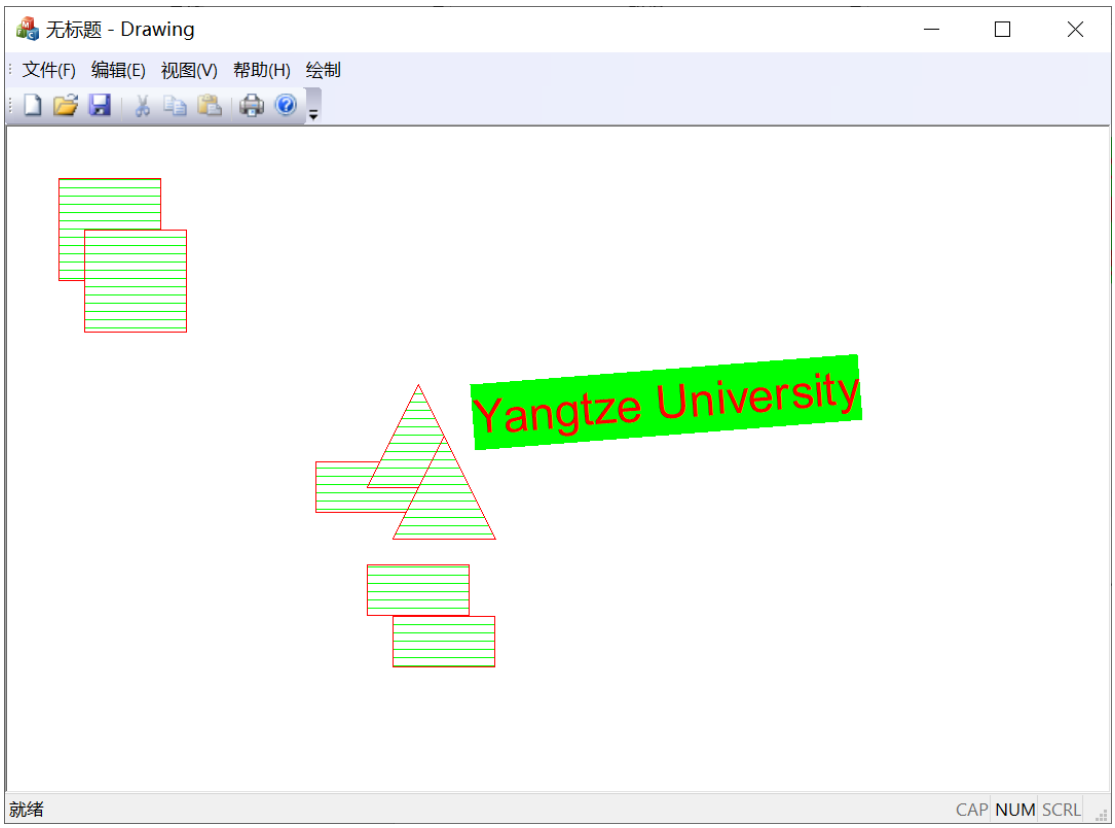
而文档的存储主要通过文档类重载成员函数 Serialize 来实现，实现用文档来保存 m_Elements 数组成员，最后便可实现画布的保存与读取。

四、 运行效果

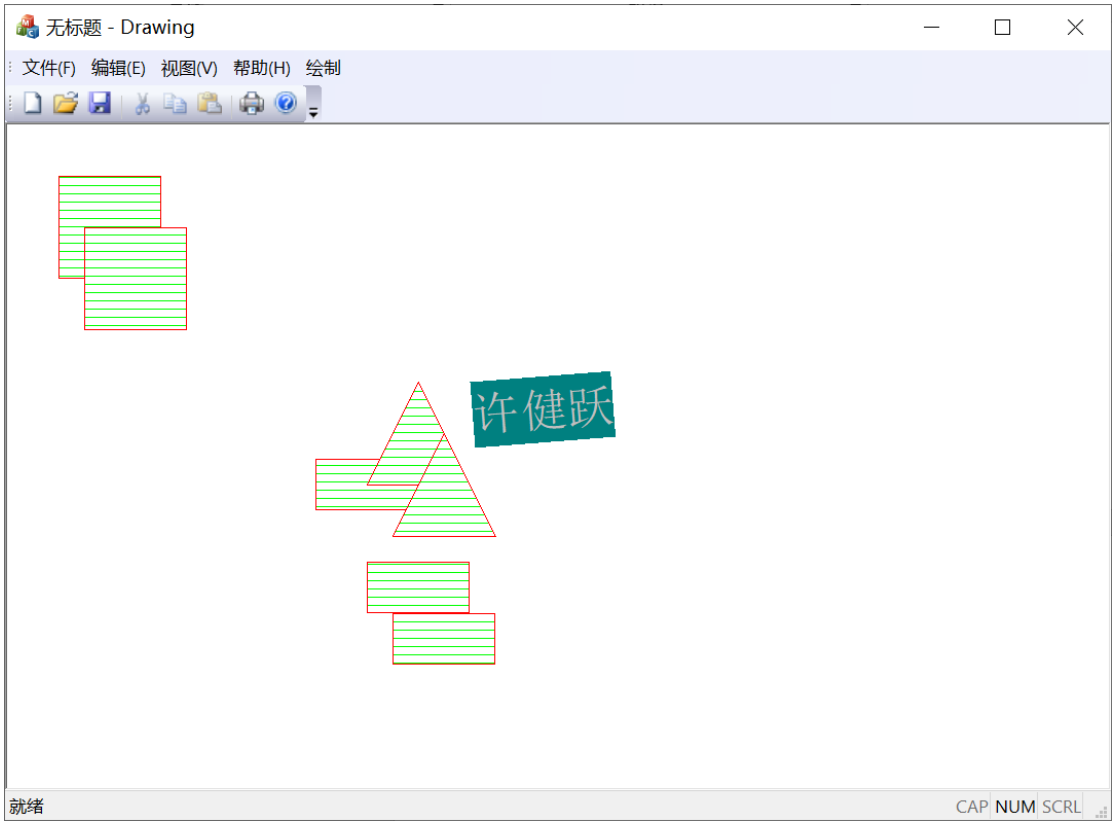
1.通过视图窗口菜单绘制图元



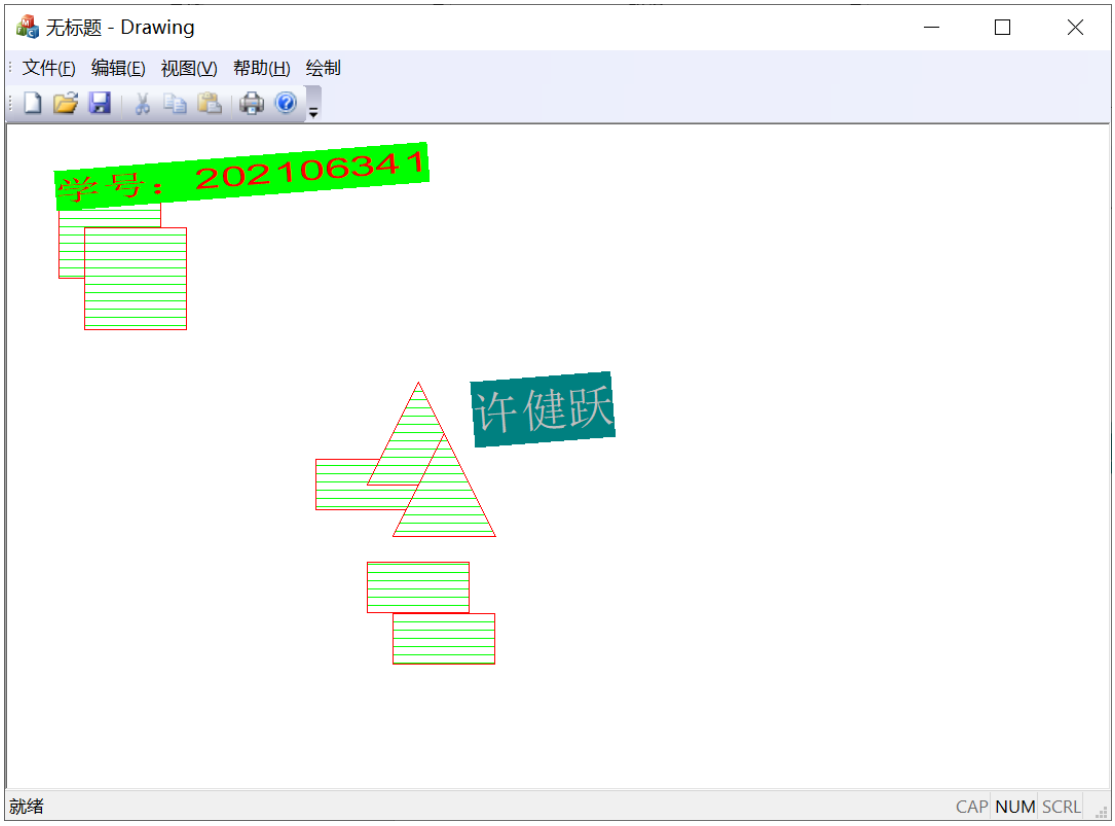
2.在用户视图窗口双击删除图元



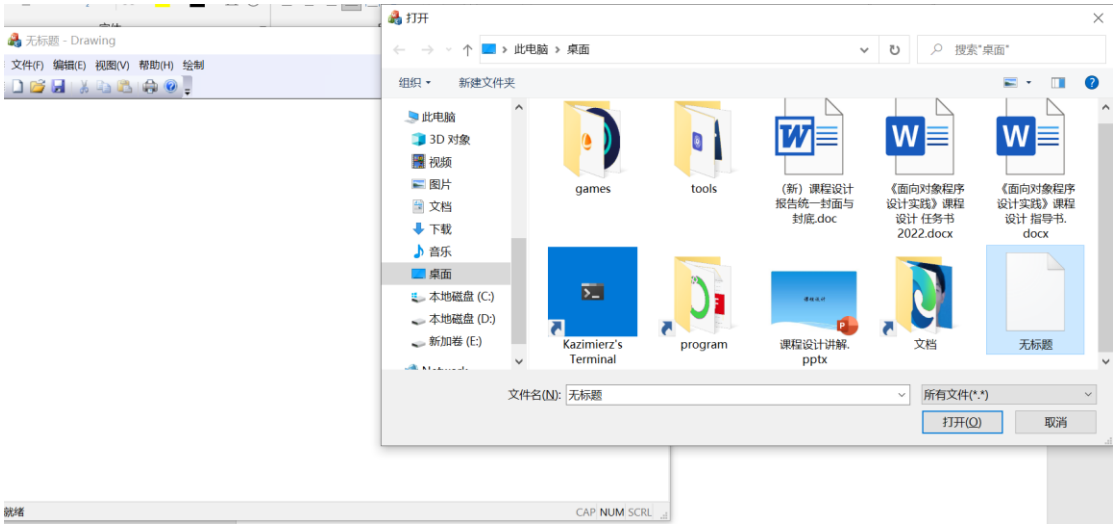
3.双击鼠标右键修改图元信息



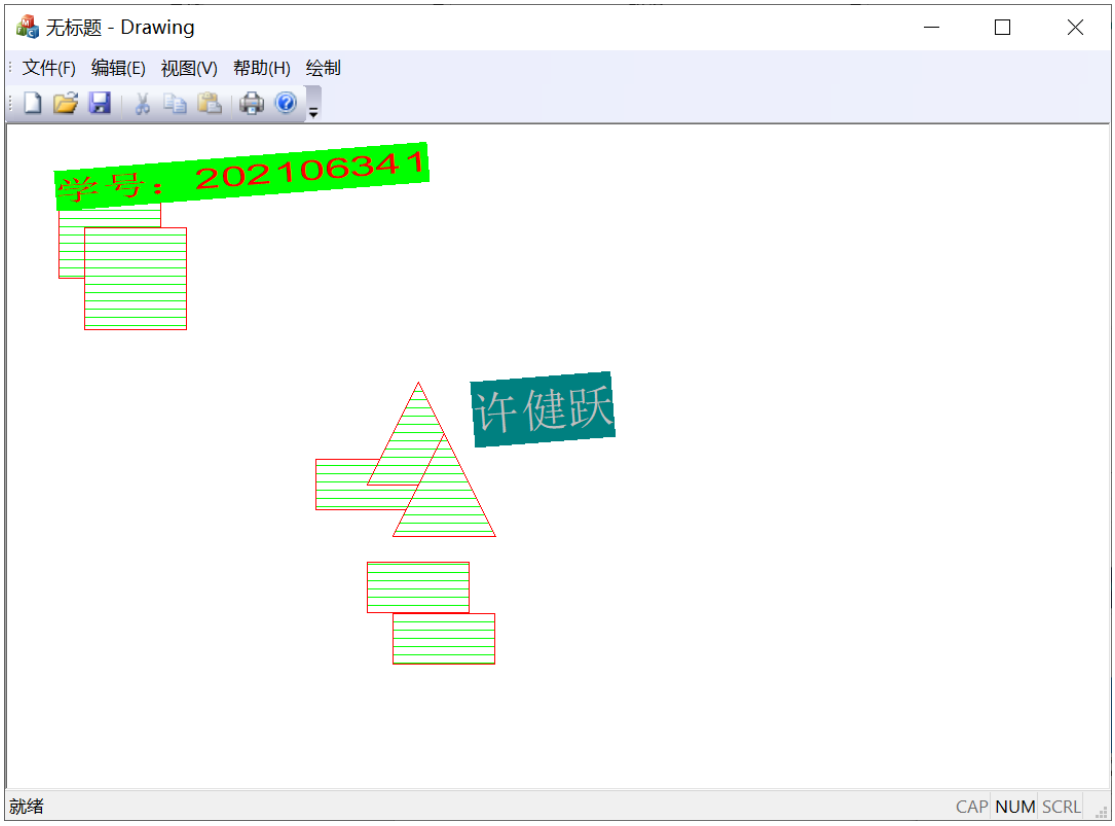
4.Ctrl+鼠标左键添加新图元



5.画布的保存与读取



打开刚刚保存的文件



打开结果

五、课程设计小结

首先我由衷的感谢老师提供给我们这样一个锻炼自己的机会，经过这几周的学习，本次课程设计即将结束，总的来说，经过这门课的学习收获还是相当大的。回顾这段时间的课程设计，至今我仍感慨万分。的确，从学习到开始制作，从理论到实践，在这几周的实训日子里，可以说得是苦多于甜，但是可以学到很多很多的的东西，同时不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。

就拿本次的 MFC 课程设计举例，它要求我们设计一个绘图程序，我们就要思考这个图元它应该怎么保存？支持哪些操作？运用之前所学的 OOP（面向对象程序设计）思路，首先整体抽象出图元类的声明，然后具体实现直接 To Do 放到最后再去完成，我认为这样的实践是可以使我们加深理解什么是面向对象，什么又是程序架构。

从设计到架构到实现再到完成，我们亲身体验了程序开发的整个流程，在此过程中不仅巩固了 C/C++语言的基础理论知识，又提升了我们的编码能力，还使得我们了解一种全新的框架，MFC 程序设计，本次课程设计的经历对于我们以后的学习生活的帮助无疑是很大的。

六、主要代码清单

1.Shape.h

```
#pragma once

enum  ElementType { NOTSET, SQUARE, RECTANGLE, CIRCLE, ELLIPSE, TRIANGLE, TEXT };
//声明CShape类，
class CShape : public CObject
{
public:
    CShape();
    virtual ~CShape();
    virtual void Draw(CDC* pDC) = 0;//绘制图元
    virtual bool IsMatched(CPoint pnt) = 0;// Cshape类
    virtual void Serialize(CArchive& ar) = 0;
    virtual void SetValue(ElementType t, int x, int y, int width, int height,CString text)
= 0;
    virtual void GetValue(ElementType& t, int& x, int& y, int& width, int& height,CString&
text) = 0;
    void SetPen(COLORREF bcolor, int btype, int bwidth);
    void GetPen(COLORREF& bcolor, int& btype, int& bwidth);
    void SetBrush(COLORREF fcolor, int ftype);
    void GetBrush(COLORREF& fcolor, int& ftype);

protected:
    ElementType Type;//图元类型
    int OrgX;//原点坐标
    int OrgY;
    COLORREF   BorderColor;//边界颜色
    int BorderType;//边界线型--实线、虚线、虚点线等
    int BorderWidth;//边界宽度
    COLORREF  FillColor;//填充颜色
    int FillType;//填充类型--实心、双对角、十字交叉等
};


//声明正方形类CSquare
class CSquare : public CShape
{
public:
    CSquare();
    CSquare(int x, int y, int w);
    void Draw(CDC* pDC);//绘制正方形
    bool  IsMatched(CPoint pnt);// CSquare类;
    virtual void Serialize(CArchive& ar);//序列化正方形图元
    DECLARE_SERIAL(CSquare)//声明类CSquare支持序列化
    virtual void SetValue(ElementType t, int x, int y, int w, int h,CString text);
```

```
    virtual void GetValue(ElementType & t, int& x, int& y, int& w, int& h, CString & text);

private:
    int width;

};

//圆形类CCircle
class CCircle : public CShape
{
public:
    CCircle();
    CCircle(int x, int y, int w);
    void Draw(CDC* pDC);//绘制圆形
    bool  IsMatched(CPoint pnt);// CCircle类;
    virtual void Serialize(CArchive& ar);//序列化圆形图元
    DECLARE_SERIAL(CCircle)//声明类CCircle支持序列化
    virtual void SetValue(ElementType t, int x, int y, int w, int h, CString text);
    virtual void GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text);

private:
    int width;
};

//椭圆类CEllipse
class CEllipse:public CShape
{
public:
    CEllipse();
    CEllipse(int x, int y, int w,int h);
    void Draw(CDC* pDC);//绘制椭圆形
    bool  IsMatched(CPoint pnt);// CEllipse类;
    virtual void Serialize(CArchive& ar);//序列化椭圆图元
    DECLARE_SERIAL(CEllipse)//声明类CEllipse支持序列化
    virtual void SetValue(ElementType t, int x, int y, int w, int h, CString text);
    virtual void GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text);

private:
    int width;
    int height;
};

//三角形类CRectangle
class CRectangle :public CShape
```

```
{
public:
    CRectangle();
    CRectangle(int x, int y, int w, int h);
    void Draw(CDC* pDC);//绘制三角形
    bool  IsMatched(CPoint pnt);// CRectangle类；
    virtual void Serialize(CArchive& ar);//序列化图元
    DECLARE_SERIAL(CRectangle)//声明类支持序列化
    virtual void SetValue(ElementType t, int x, int y, int w, int h, CString text);
    virtual void GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text);

private:
    int width;
    int height;
};

//声明三角形类CTritangle
class CTriangle : public CShape
{
public:
    CTriangle();
    CTriangle(int x, int y, int w);
    void Draw(CDC* pDC);//绘制
    bool  IsMatched(CPoint pnt);
    virtual void Serialize(CArchive& ar);//序列化图元
    DECLARE_SERIAL(CTriangle)//声明类支持序列化
    virtual void SetValue(ElementType t, int x, int y, int w,int h, CString text);
    virtual void GetValue(ElementType& t, int& x, int& y, int& w,int& h, CString& text);

private:
    int width;
};

//声明文本类CText
class CText : public CShape
{
public:
    CText();
    CText(int x, int y, CString text,int height,int w);
    void Draw(CDC* pDC);//绘制文本
    bool  IsMatched(CPoint pnt);
    virtual void Serialize(CArchive& ar);//序列化
    DECLARE_SERIAL(CText)//声明类支持序列化
    virtual void SetValue(ElementType t, int x, int y, int w, int h, CString text);
```

```
    virtual void GetValue(ElementType& t, int& x, int& y, int& w, int& h,CString& text);

private:
    CString text; //字体内容
    int height; //字体高度
    int width; //旋转角度
};
```

2.Shape.cpp

```
#include "pch.h"
#include "Shape.h"

CSquare::CSquare()
{
    Type = SQUARE;//图元类型
    OrgX = 100;//原点坐标
    OrgY = 100;
    BorderColor = RGB(255, 0, 0);//边界颜色
    BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
    BorderWidth = 1;//边界宽度
    FillColor = RGB(0, 255, 0);//填充颜色
    FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
    width = 100;
}

CSquare::CSquare(int x, int y, int w)
{
    Type = SQUARE;//图元类型
    OrgX = x;//原点坐标
    OrgY = y;
    BorderColor = RGB(255, 0, 0);//边界颜色
    BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
    BorderWidth = 1;//边界宽度
    FillColor = RGB(0, 255, 0);//填充颜色
    FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
    width = w;
}

void CSquare::Draw(CDC* pDC)
{
    CPen* pOldPen, * pNewPen;
    pNewPen = new CPen(BorderType, BorderWidth, BorderColor);
    pOldPen = pDC->SelectObject(pNewPen);
```

```
	CBrush* pOldBrush, * pNewBrush;
	pNewBrush = new CBrush(FillType, FillColor);
	pOldBrush = pDC->SelectObject(pNewBrush);
	pDC->Rectangle(OrgX - width / 2, OrgY - width / 2, OrgX + width / 2, OrgY + width / 2);
	pDC->SelectObject(pOldPen);
	pDC->SelectObject(pOldBrush);
	delete pNewPen;
	delete pNewBrush;

}

bool CSquare::IsMatched(CPoint pnt)
{
	if ((pnt.x >= OrgX - width / 2) && (pnt.x <= OrgX + width / 2) && (pnt.y >= OrgY - width /
2) && (pnt.y <= OrgY + width / 2))
		return true;
	else
		return false;
}

IMPLEMENT_SERIAL(CSquare, CObject, 1)//实现类 CSquare 的序列化，指定版本为 1
void CSquare::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		ar << (WORD)Type;
		ar << OrgX << OrgY;//原点坐标
		ar << BorderColor;//边界颜色
		ar << BorderType;
		ar << BorderWidth;//边界宽度
		ar << FillColor;//
		ar << FillType;
		ar << width;
	}
	else
	{
		WORD w;
		ar >> w;
		Type = (ElementType)w;
		ar >> OrgX >> OrgY;//原点坐标
		ar >> BorderColor;//边界颜色
		ar >> BorderType;
		ar >> BorderWidth;//边界宽度
		ar >> FillColor;
```

```
            ar >> FillType;
            ar >> width;
        }
}

void CSquare::SetValue(ElementType t, int x, int y, int w, int h, CString text)
{
    Type = t; OrgX = x; OrgY = y; width = w;
}
void CSquare::GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text)
{
    t = Type; x = OrgX; y = OrgY; w = width; h = 0,text = "0";
}

CShape::CShape()
{
}

CShape::~CShape()
{
}

void CShape::SetPen(COLORREF bcolor, int btype, int bwidth)
{
    BorderColor = bcolor; BorderType = btype; BorderWidth = bwidth;
}
void CShape::GetPen(COLORREF& bcolor, int& btype, int& bwidth)
{
    bcolor = BorderColor; btype = BorderType; bwidth = BorderWidth;
}
void CShape::SetBrush(COLORREF fcolor, int ftype)
{
    FillColor = fcolor; FillType = ftype;
}
void CShape::GetBrush(COLORREF& fcolor, int& ftype)
{
    fcolor = FillColor; ftype = FillType;
}

CCircle::CCircle()
{
    Type = CIRCLE;//图元类型
    OrgX = 200;//原点坐标
    OrgY = 200;
```

```
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = 100;
}

CCircle::CCircle(int x, int y, int w)
{
	Type = CIRCLE;//图元类型
	OrgX = x;//原点坐标
	OrgY = y;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = w;
}

void CCircle::Draw(CDC* pDC)
{
	CPen* pOldPen, * pNewPen;
	pNewPen = new CPen(BorderType, BorderWidth, BorderColor);
	pOldPen = pDC->SelectObject(pNewPen);
	CBrush* pOldBrush, * pNewBrush;
	pNewBrush = new CBrush(FillType, FillColor);
	pOldBrush = pDC->SelectObject(pNewBrush);
	pDC->Ellipse(OrgX - width / 2, OrgY - width / 2, OrgX + width / 2, OrgY + width / 2);
	pDC->SelectObject(pOldPen);
	pDC->SelectObject(pOldBrush);
	delete pNewPen;
	delete pNewBrush;
}

bool CCircle::IsMatched(CPoint pnt)
{
	if ((pnt.x - OrgX) * (pnt.x - OrgX) + (pnt.y - OrgY) * (pnt.y - OrgY) < width * width)
		return true;
	else
		return false;
}
```

```
IMPLEMENT_SERIAL(CCircle, CObject, 1)//实现类 CCircle 的序列化，指定版本为 1
void CCircle::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		ar << (WORD)Type;
		ar << OrgX << OrgY;//原点坐标
		ar << BorderColor;//边界颜色
		ar << BorderType;
		ar << BorderWidth;//边界宽度
		ar << FillColor;//
		ar << FillType;
		ar << width;
	}
	else
	{
		WORD w;
		ar >> w;
		Type = (ElementType)w;
		ar >> OrgX >> OrgY;//原点坐标
		ar >> BorderColor;//边界颜色
		ar >> BorderType;
		ar >> BorderWidth;//边界宽度
		ar >> FillColor;
		ar >> FillType;
		ar >> width;
	}
}

void CCircle::SetValue(ElementType t, int x, int y, int w, int h, CString text)
{
	Type = t; OrgX = x; OrgY = y; width = w;
}

void CCircle::GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text)
{
	t = Type; x = OrgX; y = OrgY; w = width; h = 0,text = "0";
}

CEllipse::CEllipse()
{
	Type = ELLIPSE;//图元类型
	OrgX = 300;//原点坐标
	OrgY = 300;
```

```
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = 50;
	height = 100;
}

CEllipse::CEllipse(int x, int y, int w,int h)
{
	Type = ELLIPSE;//图元类型
	OrgX = x;//原点坐标
	OrgY = y;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = w;
	height = h;
}

void CEllipse::Draw(CDC* pDC)
{
	CPen* pOldPen, * pNewPen;
	pNewPen = new CPen(BorderType, BorderWidth, BorderColor);
	pOldPen = pDC->SelectObject(pNewPen);
	CBrush* pOldBrush, * pNewBrush;
	pNewBrush = new CBrush(FillType, FillColor);
	pOldBrush = pDC->SelectObject(pNewBrush);
	pDC->Ellipse(OrgX - height/2, OrgY - width/2, OrgX + height/2, OrgY + width/2);
	pDC->SelectObject(pOldPen);
	pDC->SelectObject(pOldBrush);
	delete pNewPen;
	delete pNewBrush;
}

bool CEllipse::IsMatched(CPoint pnt)
{
	if ((pnt.x - OrgX) * (pnt.x - OrgX) + (pnt.y - OrgY) * (pnt.y - OrgY) < width * width)
		return true;
	else
		return false;
```

```
}

IMPLEMENT_SERIAL(CEllipse, CObject, 1)//实现类 CEllipse 的序列化，指定版本为 1
void CEllipse::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		ar << (WORD)Type;
		ar << OrgX << OrgY;//原点坐标
		ar << BorderColor;//边界颜色
		ar << BorderType;
		ar << BorderWidth;//边界宽度
		ar << FillColor;//
		ar << FillType;
		ar << width;
		ar << height;
	}
	else
	{
		WORD w;
		ar >> w;
		Type = (ElementType)w;
		ar >> OrgX >> OrgY;//原点坐标
		ar >> BorderColor;//边界颜色
		ar >> BorderType;
		ar >> BorderWidth;//边界宽度
		ar >> FillColor;
		ar >> FillType;
		ar >> width;
		ar >> height;
	}
}

void CEllipse::SetValue(ElementType t, int x, int y, int w, int h, CString text)
{
	Type = t; OrgX = x; OrgY = y; width = w,height = h;
}

void CEllipse::GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text)
{
	t = Type; x = OrgX; y = OrgY; w = width; h = height,text = "0";
}

CRectangle::CRectangle()
```

```
{
	Type = RECTANGLE;//图元类型
	OrgX = 350;//原点坐标
	OrgY = 350;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = 100;
	height = 50;
}

CRectangle::CRectangle(int x, int y, int w, int h)
{
	Type = RECTANGLE;//图元类型
	OrgX = x;//原点坐标
	OrgY = y;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = w;
	height = h;
}

void CRectangle::Draw(CDC* pDC)
{
	CPen* pOldPen, * pNewPen;
	pNewPen = new CPen(BorderType, BorderWidth, BorderColor);
	pOldPen = pDC->SelectObject(pNewPen);
	CBrush* pOldBrush, * pNewBrush;
	pNewBrush = new CBrush(FillType, FillColor);
	pOldBrush = pDC->SelectObject(pNewBrush);
	pDC->Rectangle(OrgX - width / 2, OrgY - height / 2, OrgX + width / 2, OrgY + height / 2);
	pDC->SelectObject(pOldPen);
	pDC->SelectObject(pOldBrush);
	delete pNewPen;
	delete pNewBrush;
}

bool CRectangle::IsMatched(CPoint pnt)
{
```

```
    if ((pnt.x >= OrgX - width / 2) && (pnt.x <= OrgX + width / 2) && (pnt.y >= OrgY - height / 2) && (pnt.y <= OrgY + height / 2))
        return true;
    else
        return false;
}

IMPLEMENT_SERIAL(CRectangle, CObject, 1)//实现类 CEllipse 的序列化，指定版本为 1
void CRectangle::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << (WORD)Type;
        ar << OrgX << OrgY;//原点坐标
        ar << BorderColor;//边界颜色
        ar << BorderType;
        ar << BorderWidth;//边界宽度
        ar << FillColor;//
        ar << FillType;
        ar << width;
        ar << height;
    }
    else
    {
        WORD w;
        ar >> w;
        Type = (ElementType)w;
        ar >> OrgX >> OrgY;//原点坐标
        ar >> BorderColor;//边界颜色
        ar >> BorderType;
        ar >> BorderWidth;//边界宽度
        ar >> FillColor;
        ar >> FillType;
        ar >> width;
        ar >> height;
    }
}

void CRectangle::SetValue(ElementType t, int x, int y, int w, int h, CString text)
{
    Type = t; OrgX = x; OrgY = y; width = w, height = h;
}

void CRectangle::GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text)
```

```
{
	t = Type; x = OrgX; y = OrgY; w = width; h = height,text = "0";
}

CTriangle::CTriangle()
{
	Type = TRIANGLE;//图元类型
	OrgX = 400;//原点坐标
	OrgY = 300;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = 80;
}

CTriangle::CTriangle(int x, int y, int w)
{
	Type = TRIANGLE;//图元类型
	OrgX = x;//原点坐标
	OrgY = y;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	width = w;
}

void CTriangle::Draw(CDC* pDC)
{
	CPen* pOldPen, * pNewPen;
	pNewPen = new CPen(BorderType, BorderWidth, BorderColor);
	pOldPen = pDC->SelectObject(pNewPen);
	CBrush* pOldBrush, * pNewBrush;
	pNewBrush = new CBrush(FillType, FillColor);
	pOldBrush = pDC->SelectObject(pNewBrush);
	CPoint pt[3] = { CPoint(OrgX-width/2,OrgY+width/2),CPoint(OrgX,OrgY-width/2),
CPoint(OrgX+width/2,OrgY+width/2) };
	pDC->Polygon(pt, 3);
	pDC->SelectObject(pOldPen);
	pDC->SelectObject(pOldBrush);
	delete pNewPen;
```

```
    delete pNewBrush;
}

bool CTriangle::IsMatched(CPoint pnt)
{
    if ((pnt.x >= OrgX - width / 2) && (pnt.x <= OrgX + width / 2) && (pnt.y >= OrgY - width /
2) && (pnt.y <= OrgY + width / 2))
        return true;
    else
        return false;
}

IMPLEMENT_SERIAL(CTriangle, CObject, 1)//实现类 CEllipse 的序列化，指定版本为 1
void CTriangle::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << (WORD)Type;
        ar << OrgX << OrgY;//原点坐标
        ar << BorderColor;//边界颜色
        ar << BorderType;
        ar << BorderWidth;//边界宽度
        ar << FillColor;//
        ar << FillType;
        ar << width;
    }
    else
    {
        WORD w;
        ar >> w;
        Type = (ElementType)w;
        ar >> OrgX >> OrgY;//原点坐标
        ar >> BorderColor;//边界颜色
        ar >> BorderType;
        ar >> BorderWidth;//边界宽度
        ar >> FillColor;
        ar >> FillType;
        ar >> width;
    }
}

void CTriangle::SetValue(ElementType t, int x, int y, int w,int h, CString text)
{
    Type = t; OrgX = x; OrgY = y; width = w;
```

```
}

void CTriangle::GetValue(ElementType& t, int& x, int& y, int& w,int &h, CString& text)
{
	t = Type; x = OrgX; y = OrgY; w = width, h = 0,text = "0";
}

CText::CText()
{
	Type = TEXT;//图元类型
	OrgX = 450;//原点坐标
	OrgY = 250;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	text = "Yangtze University";
	this->height = 50;
	this->width = 45;
}

CText::CText(int x, int y, CString text, int height, int w)
{
	Type = TEXT;//图元类型
	OrgX = x;//原点坐标
	OrgY = y;
	BorderColor = RGB(255, 0, 0);//边界颜色
	BorderType = PS_SOLID;//边界线型--实线、虚线、虚点线等
	BorderWidth = 1;//边界宽度
	FillColor = RGB(0, 255, 0);//填充颜色
	FillType = HS_HORIZONTAL;//填充类型--实心、双对角、十字交叉等
	this->text = text;
	this->height = height;
	this->width = w;
}

void CText::Draw(CDC* pDC)
{
	CFont font;
	font.CreateFont(
		height,                    // nHeight  文字高度
		20,                        // nWidth  文字宽度默认 20
		width,                     // nEscapement  旋转角度
```

```
            0,                                  // nOrientation
            FW_NORMAL,                          // nWeight
            FALSE,                              // bItalic
            FALSE,                              // bUnderline
            0,                                  // cStrikeOut
            ANSI_CHARSET,                       // nCharSet
            OUT_DEFAULT_PRECIS,                 // nOutPrecision
            CLIP_DEFAULT_PRECIS,                // nClipPrecision
            DEFAULT_QUALITY,                    // nQuality
            DEFAULT_PITCH | FF_SWISS,           // nPitchAndFamily
            text);
        pDC->SelectObject(font);
        pDC->SetTextColor(BorderColor); //文字颜色
        pDC->SetBkColor(FillColor); //填充颜色
        pDC->TextOut(OrgX, OrgY, text); //显示文字
        pDC->SetBkColor(RGB(255,255,255)); //还原填充色为白色
}

bool CText::IsMatched(CPoint pnt)
{
        if ((pnt.x >= OrgX) && (pnt.x <= OrgX + text.GetLength()*20) && (pnt.y >= OrgY) &&
(pnt.y <= OrgY + text.GetLength() * height))
                return true;
        else
                return false;
}

IMPLEMENT_SERIAL(CText, CObject, 1)//实现类 CEllipse 的序列化，指定版本为 1
void CText::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                ar << (WORD)Type;
                ar << OrgX << OrgY;//原点坐标
                ar << BorderColor;//边界颜色
                ar << BorderType;
                ar << BorderWidth;//边界宽度
                ar << FillColor;
                ar << FillType;
                ar << text;
                ar << height;
                ar << width;
        }
        else
```

```
    {
        WORD w;
        ar >> w;
        Type = (ElementType)w;
        ar >> OrgX >> OrgY;//原点坐标
        ar >> BorderColor;//边界颜色
        ar >> BorderType;
        ar >> BorderWidth;//边界宽度
        ar >> FillColor;
        ar >> FillType;
        ar >> text;
        ar >> height;
        ar >> width;
    }
}

void CText::SetValue(ElementType t, int x, int y, int w, int h, CString text)
{
    Type = t; OrgX = x; OrgY = y; this->text = text,this->height = h,this->width = w;
}

void CText::GetValue(ElementType& t, int& x, int& y, int& w, int& h, CString& text)
{
    t = Type; x = OrgX; y = OrgY; w = 0, h = this->height, text = this->text,w = this->width;
}
```

3.DrawingView.cpp 中的各消息处理函数

```
void CDrawingView::OnSquare()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticSquareX = 100, StaticSquareY = 100;//正方形的中心点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CSquare(StaticSquareX, StaticSquareY, 100));
    StaticSquareX += 25;
    StaticSquareY += 50; //下一个正方形的中心点坐标值增加
    pDoc->UpdateAllViews(NULL);
}


void CDrawingView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    CDrawingDoc* pDoc = GetDocument();
```

```
    CShape* p;
    for (int i = pDoc->m_Elements.GetSize() - 1; i >= 0; i--)
    {
        p = (CShape*)pDoc->m_Elements[i];
        if (p->IsMatched(point))
        {
            pDoc->m_Elements.RemoveAt(i);
            delete p;
                pDoc->UpdateAllViews(NULL);
            break;
        }
    }
    CScrollView::OnLButtonDblClk(nFlags, point);
}


void CDrawingView::OnRButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    CDrawingDoc* pDoc = GetDocument();
    CShape* p;
    for (int i = pDoc->m_Elements.GetSize() - 1; i >= 0; i--)
    {
        p = (CShape*)pDoc->m_Elements[i];
        if (p->IsMatched(point))
        {
            CShapeDlg d1;
            d1.pshape = p;
            if (d1.DoModal() == IDOK)
            {
                d1.DlgToShape();
            }
            pDoc->UpdateAllViews(NULL);
            break;
        }
    }

    CScrollView::OnRButtonDblClk(nFlags, point);
}


void CDrawingView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
```

```
    if ((nFlags & MK_CONTROL) == MK_CONTROL)//Ctrl键按下
    {
        CShapeDlg mydlg;
        mydlg.pshape = NULL;
        if (mydlg.DoModal() == IDOK)
        {
            switch (mydlg.Type)
            {
            case SQUARE:mydlg.pshape = new CSquare(); break;
            case RECTANGLE:mydlg.pshape = new CRectangle(); break;
            case CIRCLE:mydlg.pshape = new CCircle(); break;
            case ELLIPSE:mydlg.pshape = new CEllipse(); break;
            case TRIANGLE:mydlg.pshape = new CTriangle(); break;
            case TEXT:mydlg.pshape = new CText(); break;
            }
            mydlg.DlgToShape();
            CDrawingDoc* pDoc = GetDocument();
            pDoc->m_Elements.Add(mydlg.pshape);
            pDoc->UpdateAllViews(NULL);
        }
    }

    CScrollView::OnLButtonDown(nFlags, point);
}


void CDrawingView::OnCircle()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticCircleX = 200, StaticCircleY = 200;//圆形的中心点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CCircle(StaticCircleX, StaticCircleY, 100)); /*to do*/
    StaticCircleX += 25;
    StaticCircleY += 50; //下一个圆形的中心点坐标值增加
    pDoc->UpdateAllViews(NULL);
}

void CDrawingView::OnEllipse()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticEllipseX = 300, StaticEllipseY = 300;//中心点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CEllipse(StaticEllipseX, StaticEllipseY, 50,100));
    StaticEllipseX += 25;
```

```
    StaticEllipseY += 50; //下一个中心点坐标值增加
    pDoc->UpdateAllViews(NULL);
}

void CDrawingView::OnRectangle()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticRectangleX = 350, StaticRectangleY = 350;//中心点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CRectangle(StaticRectangleX, StaticRectangleY, 100, 50));
    StaticRectangleX += 25;
    StaticRectangleY += 50; //下一个中心点坐标值增加
    pDoc->UpdateAllViews(NULL);
}


void CDrawingView::OnTriangle()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticTriangleX = 400, StaticTriangleY = 300;//中心点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CTriangle(StaticTriangleX, StaticTriangleY, 100));
    StaticTriangleX += 25;
    StaticTriangleY += 50; //中心点坐标值增加
    pDoc->UpdateAllViews(NULL);
}


void CDrawingView::OnText()
{
    // TODO: 在此添加命令处理程序代码
    static int StaticTextX = 450, StaticTextY = 250;//左上角点坐标
    CDrawingDoc* pDoc = GetDocument();
    pDoc->m_Elements.Add(new CText(StaticTextX, StaticTextY, "Yangtze
University",50,45));
    StaticTextX += 25;
    StaticTextY += 50; //下一个点坐标值增加
    pDoc->UpdateAllViews(NULL);
}
```

4.CShapeDlg.cpp 中的各消息处理函数

```
BOOL CShapeDlg::OnInitDialog()
{
```

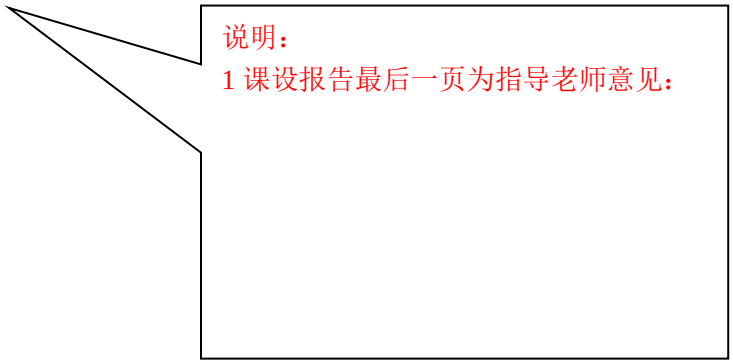
```
    CDialogEx::OnInitDialog();

    // TODO:  在此添加额外的初始化
    m_LstPenType.AddString("PS_SOLID");
    m_LstPenType.AddString("PS_DASH");
    m_LstPenType.AddString("PS_DOT");
    m_LstPenType.AddString("PS_DASHDOT");
    m_LstPenType.AddString("PS_DASHDOTDOT");

    m_LstBrushType.AddString("HS_HORIZONTAL");
    m_LstBrushType.AddString("HS_VERTICAL");
    m_LstBrushType.AddString("HS_BDIAGONAL");
    m_LstBrushType.AddString("HS_FDIAGONAL");
    m_LstBrushType.AddString("HS_CROSS");
    m_LstBrushType.AddString("HS_DIAGCROSS");

    if (pshape)  ShapeToDlg();
    else {
        UpdateData(false);
        Type = SQUARE; m_CmbShapeType.SetCurSel(0);
        BorderColor = RGB(255, 0, 0); m_ColorPen.SetColor(BorderColor);
        FillColor = RGB(0, 255, 0); m_ColorBrush.SetColor(FillColor);
        BorderType = PS_SOLID; m_LstPenType.SetCurSel(0);
        FillType = HS_HORIZONTAL; m_LstBrushType.SetCurSel(0);
    }

    return TRUE;  // return TRUE unless you set the focus to a control
                  // 异常: OCX 属性页应返回 FALSE
}


void CShapeDlg::OnCbnSelchangeCombo1()
{
    // TODO: 在此添加控件通知处理程序代码
    // 图元类型组合框
    int i = m_CmbShapeType.GetCurSel();
    switch (i)
    {
    case 0: Type = SQUARE;  break;
    case 1: Type = RECTANGLE;  break;
    case 2: Type = CIRCLE;  break;
    case 3: Type = ELLIPSE;  break;
    case 4: Type = TRIANGLE;  break;
    case 5: Type = TEXT;  break;
```

```
    }
}


void CShapeDlg::OnLbnSelchangeList1()
{
    // TODO: 在此添加控件通知处理程序代码
    // 边线样式列表框
    int i = m_LstPenType.GetCurSel();
    switch (i)
    {
    case 0:BorderType = PS_SOLID; break;
    case 1:  BorderType = PS_DASH; break;
    case 2:  BorderType = PS_DOT; break;
    case 3:  BorderType = PS_DASHDOT; break;
    case 4:  BorderType = PS_DASHDOTDOT; break;
    }
}


void CShapeDlg::OnLbnSelchangeList2()
{
    // TODO: 在此添加控件通知处理程序代码
    // 画刷阴影风格
    int i = m_LstBrushType.GetCurSel();
    switch (i)
    {
    case 0:FillType = HS_HORIZONTAL; break;
    case 1:FillType = HS_VERTICAL; break;
    case 2:FillType = HS_FDIAGONAL; break;
    case 3:FillType = HS_BDIAGONAL; break;
    case 4:FillType = HS_CROSS; break;
    case 5:FillType = HS_DIAGCROSS; break;
    }
}


void CShapeDlg::OnBnClickedMfccolorbutton1()
{
    // TODO: 在此添加控件通知处理程序代码
    // 颜色按钮
    BorderColor = m_ColorPen.GetColor();
}
```

```
void CShapeDlg::OnBnClickedMfccolorbutton2()
{
    // TODO: 在此添加控件通知处理程序代码
    FillColor = m_ColorBrush.GetColor();
}


void CShapeDlg::OnBnClickedOk()
{
    // TODO: 在此添加控件通知处理程序代码
    UpdateData(true);
    CDialogEx::OnOK();
}

void CShapeDlg::DlgToShape()
{
    if (pshape)
    {
        pshape->SetBrush(FillColor, FillType);
        pshape->SetPen(BorderColor, BorderType, m_PenWidth);
        pshape->SetValue(Type, m_x, m_y, m_width, m_height,m_text);
    }
}

void CShapeDlg::ShapeToDlg()
{
    if (pshape)
    {
        pshape->GetBrush(FillColor, FillType);
        pshape->GetPen(BorderColor, BorderType, m_PenWidth);
        pshape->GetValue(Type, m_x, m_y, m_width, m_height,m_text);
        UpdateData(false);
        switch (Type)
        {
        case SQUARE:m_CmbShapeType.SetCurSel(0);
            break;
        case RECTANGLE:m_CmbShapeType.SetCurSel(1);
            break;
        case CIRCLE:m_CmbShapeType.SetCurSel(2);
            break;
        case ELLIPSE:m_CmbShapeType.SetCurSel(3);
            break;
        case TRIANGLE:m_CmbShapeType.SetCurSel(4);
```

```
            break;
        case TEXT:m_CmbShapeType.SetCurSel(0);
            break;
        default:
            break;
        }
        switch (BorderType)
        {
        case PS_SOLID:m_LstPenType.SetCurSel(0); break;
        case PS_DASH:m_LstPenType.SetCurSel(1); break;
        case PS_DOT:m_LstPenType.SetCurSel(2); break;
        case PS_DASHDOT:m_LstPenType.SetCurSel(3); break;
        case PS_DASHDOTDOT:m_LstPenType.SetCurSel(4); break;
        }
        switch (FillType)
        {
        case HS_HORIZONTAL:m_LstBrushType.SetCurSel(0); break;
        case HS_VERTICAL:m_LstBrushType.SetCurSel(1); break;
        case HS_BDIAGONAL:m_LstBrushType.SetCurSel(2); break;
        case HS_FDIAGONAL:m_LstBrushType.SetCurSel(3); break;
        case HS_CROSS:m_LstBrushType.SetCurSel(4); break;
        case HS_DIAGCROSS:m_LstBrushType.SetCurSel(5); break;
        }
        m_ColorPen.SetColor(BorderColor);
        m_ColorBrush.SetColor(FillColor);
    }
}
```

指导老师意见：



成绩：_____　　教师签名：_____

年　　月　　日