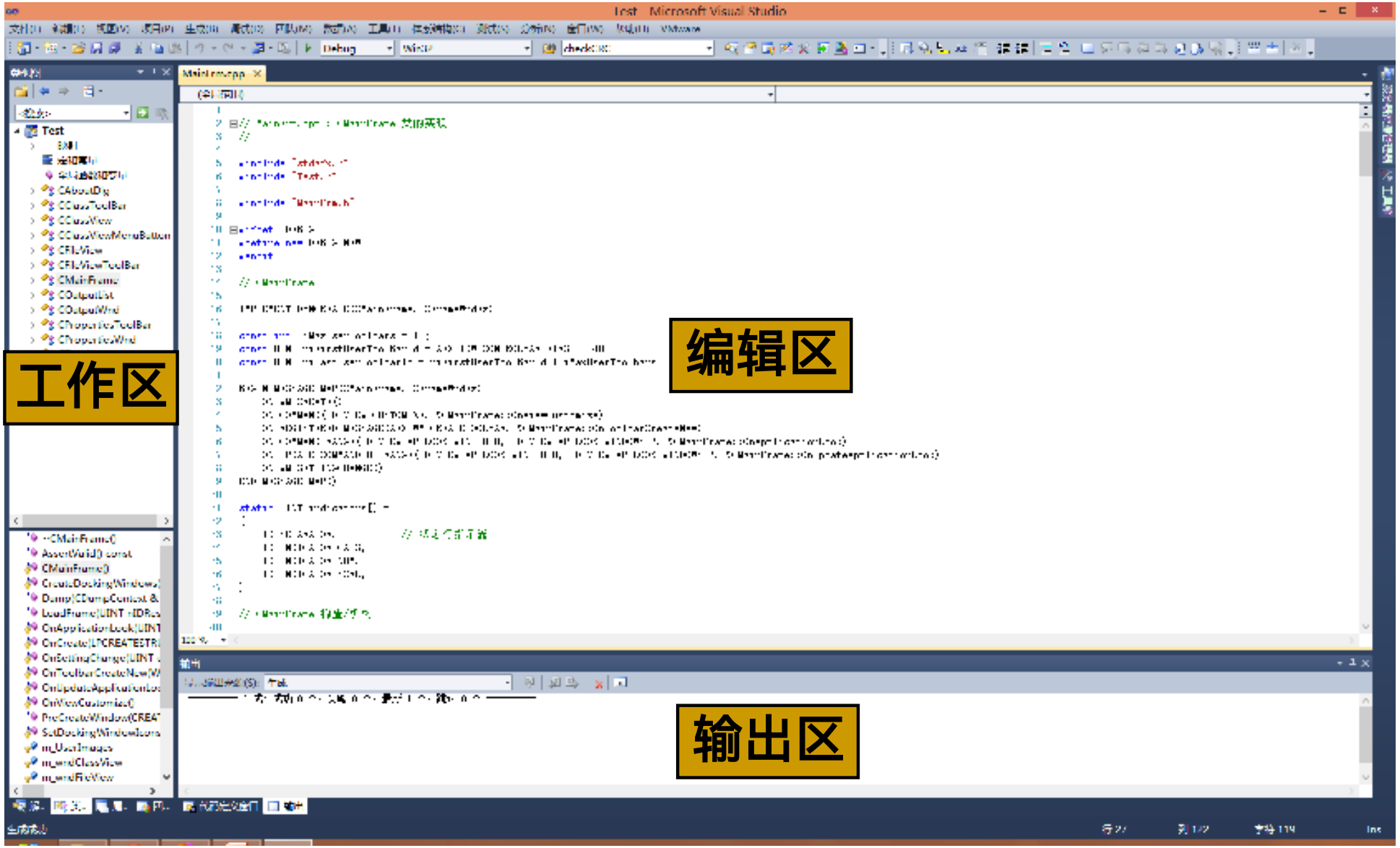
# 第2章 MFC基础与编程方法

- Visual Studio 2010介绍
- MFC类的层次结构
- MFC向导的主要功能
- MFC程序框架分析
- Windows消息机制分析

# Visual Studio 2010平台(1)



# Visual Studio 2010平台(2)

- 解决方案
  - 显示项目中的所有文件
- 类视图
  - 显示项目中的所有类
- 资源视图
  - 显示项目中的所有资源，例如Bitmap、Cursor、Dialog、Icon、Menu、Toolbar等

# Visual Studio 2010平台(3)

- 项目类型

| 大类 | 子类 |
| --- | --- |
| Win32 | Win32控制台与应用程序 |
| MFC | MFC应用程序、DLL与ActiveX控件 |
| CLR | CLR控制台与类库、Windows窗体与控件 |
| ATL | ATL项目 |
| 常规 | 空项目、自定义向导 |

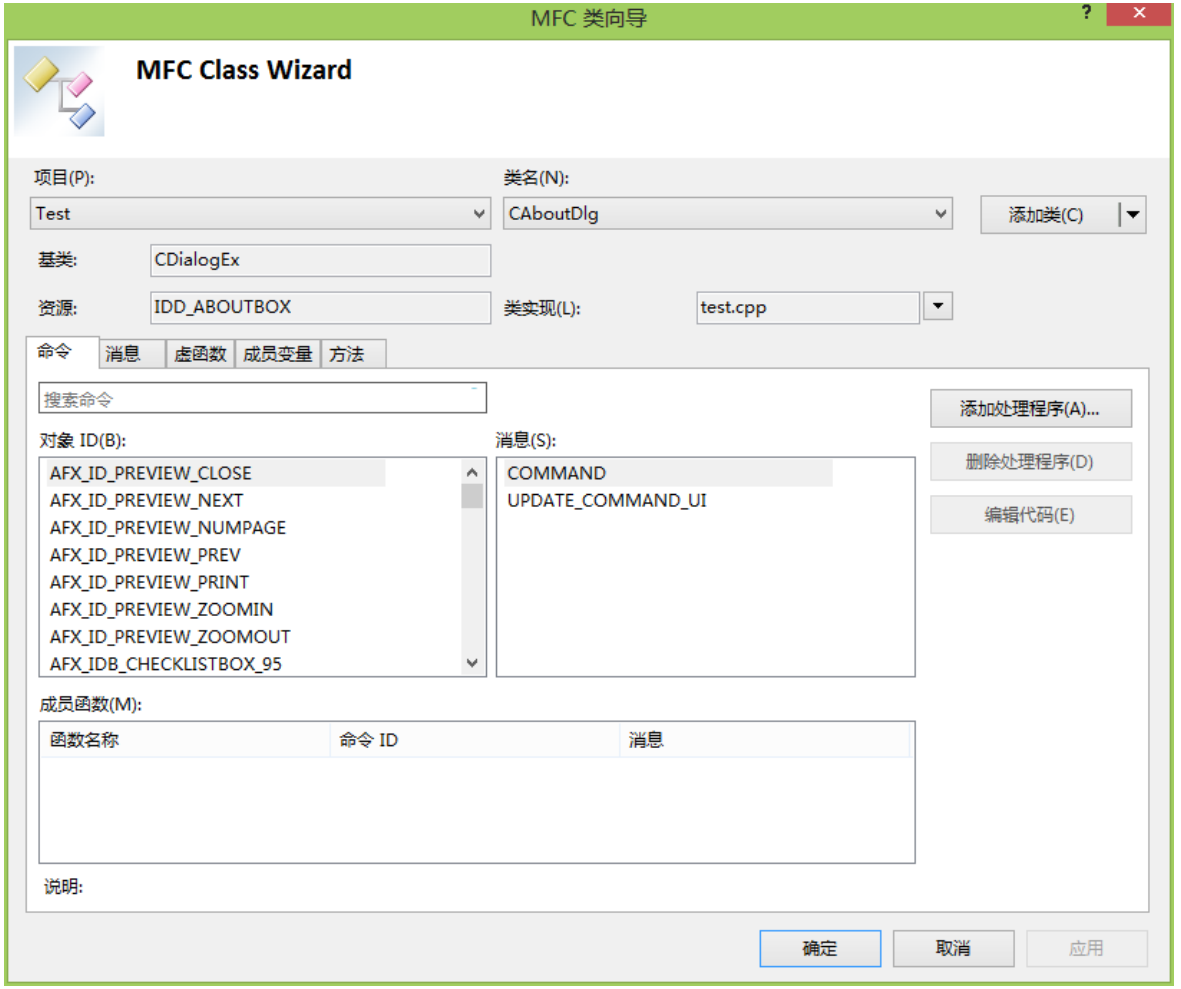
# Visual Studio 2010平台(4)

- 文件类型

| 大类 | 子类 |
| --- | --- |
| 代码 | C++文件、头文件、模块定义 |
| UI | Windows窗体、功能区、控件 |
| 资源 | 资源文件或模板、注册脚本、位图、光标、图标文件 |
| 数据 | XML架构、SQL脚本、报表 |
| Web | HTML页、框架、XML文件、样式表 |

# Visual Studio 2010平台(5)

- MFC类向导



# 程序调试与运行(1)

- 编译(Compile)
- 链接(Link)
- 执行(Execute)
- 调试(Debug)

# 程序调试与运行(2)

- 调试器完成的工作
  - 设置断点
  - 单步执行代码
  - 监视变量、寄存器和内存
  - 修改代码和变量值

# 项目的概念(1)

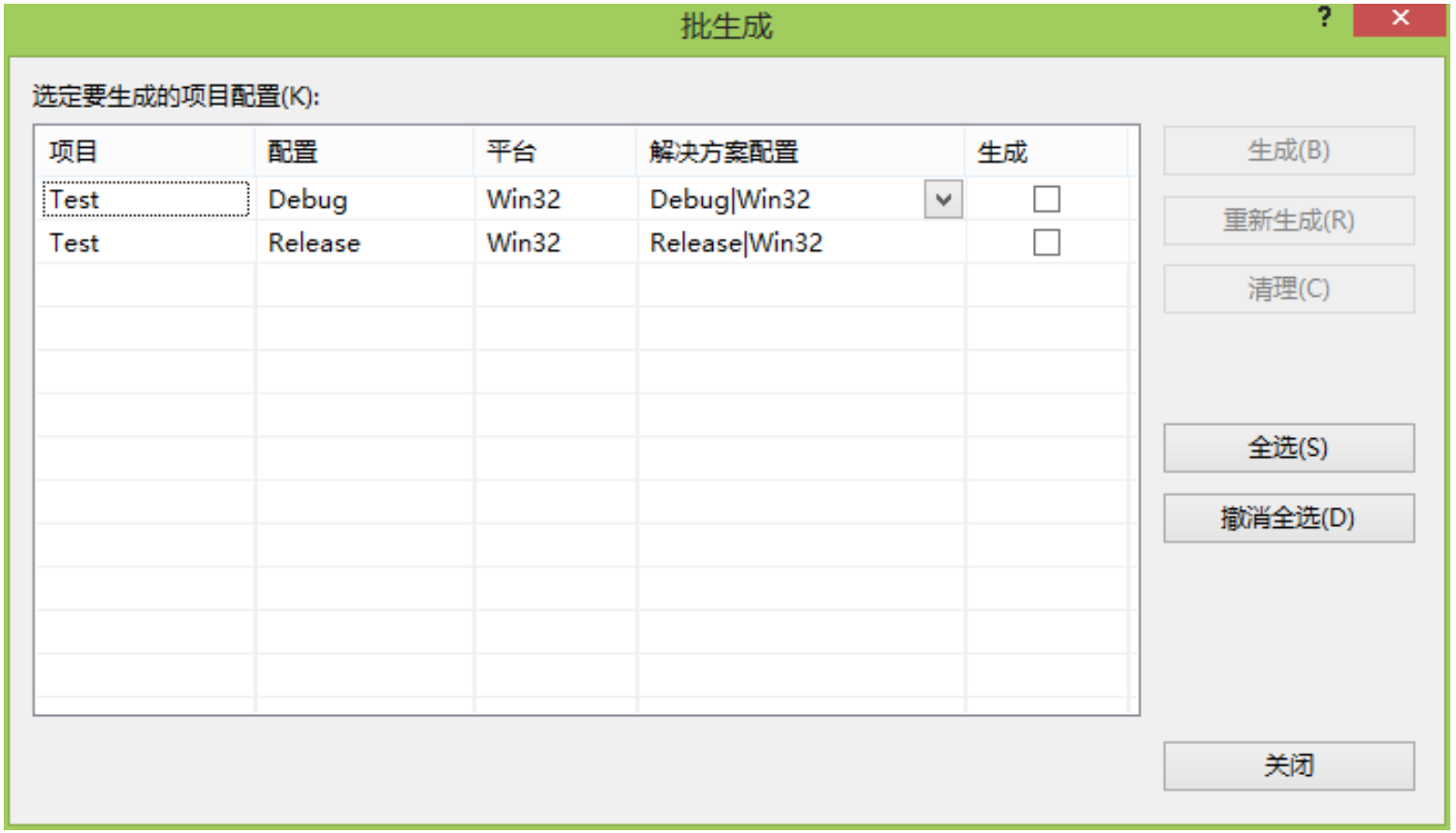
- 项目(Project)由多个源、头文件组成，以及系统提供的函数支持，编译时有些特殊选择，例如版本、优化、链接库等
- 项目文件统一管理整个程序。不同版本的项目文件不同，Visual Studio 2010中为解决方案(*.sln)

# 项目的概念(2)

- 版本类型
  - Debug版：带调试信息、占用空间小、依赖编程环境
  - Release版：不带调试信息、占用空间大、不依赖编程环境

# 项目的概念(3)

- 菜单项(生成→批生成)



# MFC类的结构(1)

- MFC是C++语言的安全子集，也是一个应用程序框架，简化Windows编程难度
- MFC类以层次结构来组织，封装大部分Windows API和控件
- 当前MFC包含100多个类，实现程序大部分功能

# MFC类的结构(2)

- 根类
- 程序框架类
- 可视对象类
- 文档类
- 通用类
- OLE类
- 数据库类
- 网络通信类

# 根类(CObject)

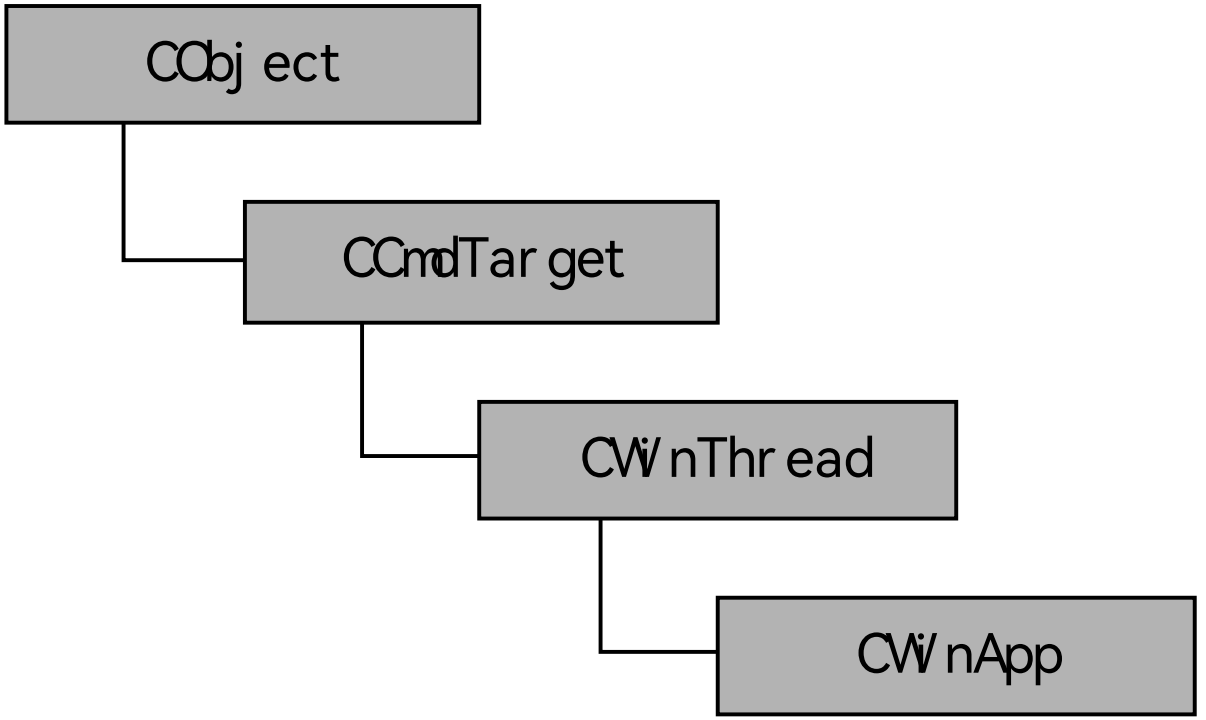
- CObject类是MFC抽象基类
- 多数MFC已有类与自定义类的根类
- 提供编程所需的公共操作
  - 对象建立与删除
  - 串行化支持
  - 运行时信息支持

# 程序框架类(1)

- CCmdTarget类
  - 命令相关类
  - MFC消息映射基类
- CWinThread类
  - 线程相关类
  - MFC线程处理基类

# 程序框架类(2)

- CWinApp类
  - 应用程序基类，每个程序仅一个对象，提供相关操作，例如初始化、运行与终止



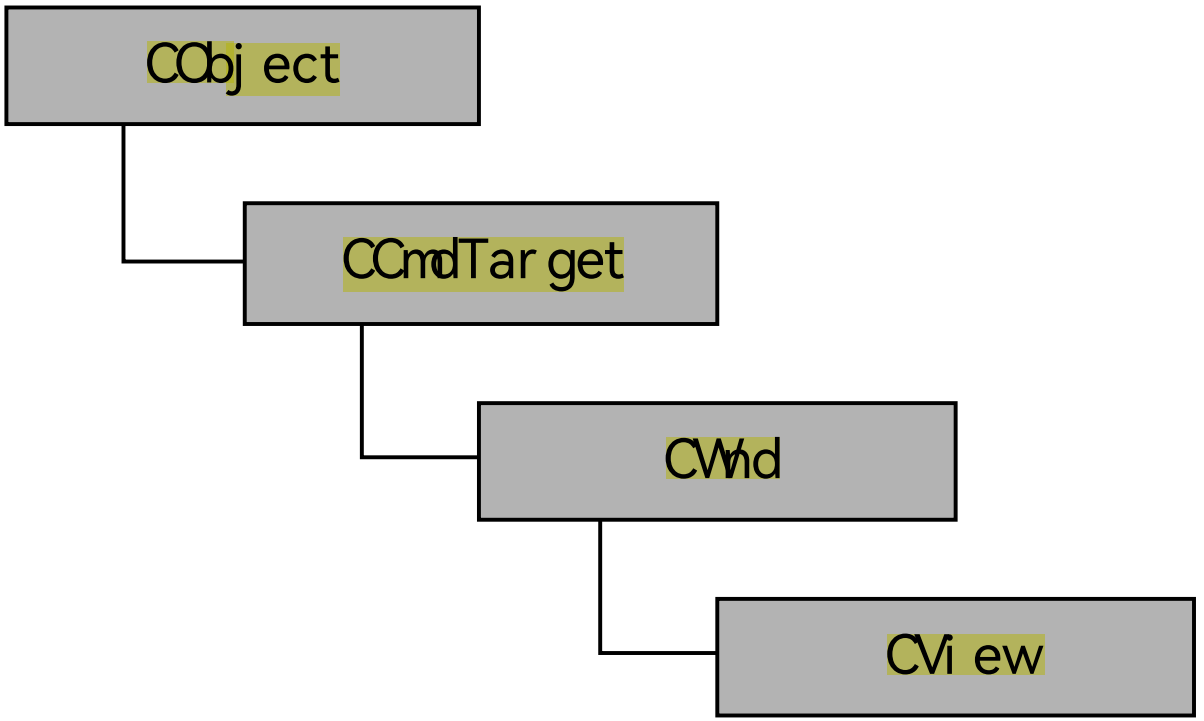
# 程序框架类(3)

- CWinApp类的公有成员函数

| 函数名 | 功能 |
| --- | --- |
| InitInstance | 初始化应用程序 |
| Run | 启动默认的消息循环 |
| ExitInstance | 终止应用程序 |
| LoadCursor | 应用程序加载光标 |
| LoadIcon | 应用程序加载图标 |

# 视图/文档类(1)

- 视图类CView
  - MFC视图基类，实现文档窗口的视图区



# 视图/文档类(2)

- CView派生类

| 派生类名 | 功能 |
| --- | --- |
| CScrollView | 支持滚动功能的视图 |
| CEditView | 支持文本编辑的视图 |
| CListView | 支持列表控件的视图 |
| CTreeView | 支持树状控件的视图 |
| CFormView | 基于表单模板的视图 |
| CRecordView | 支持数据库显示的视图 |
| CPreviewView | 支持打印预览的视图 |

# 视图/文档类(3)

- 文档类CDocument
  - 文档对象由文档模板创建，管理应用程的数据，包括文档创建、打开与保存
- 文档模板类
  - CDocTemplate：文档模板基类
  - CSingleDocTemplate：SDI文档模板
  - CMultiDocTemplate：MDI文档模板

# 可视对象类(1)

- 窗口类CWnd
  - MFC窗口基类，实现不同类型窗口
- CWnd派生类
  - CFrameWnd：单文档框架窗口类
  - CMIDFrameWnd：多文档主框架窗口类
  - CMIDChildWnd：多文档子框架窗口类

# 可视对象类(2)

- 菜单类CMenu
  - MFC菜单类，实现菜单界面
- 对话框类CDialog
  - CFileDialog：文件存取对话框
  - CColorDialog：颜色选择对话框
  - CFontDialog：字体选择对话框
  - CPrintDialog：文件打印对话框
  - CFindReplaceDialog：文本查找对话框

# 可视对象类(3)

- 控件类

| 控件类名 功能 | 控件类名 功能 |
| --- | --- |
| CStatic 文本 | CScrollBar 滚动条 |
| CEdit 编辑框 | CRichEditCtrl 格式编辑 |
| CButton 按钮 | CProgressCtrl 进度条 |
| CSlideCtrl 游标 | CSpinButtonCtrl 旋转钮 |
| CComboBox 组合框 | CTreeCtrl 树状控件 |
| CListBox 列表框 | CAnimateCtrl 动画显示 |

# 可视对象类(4)

- 控件条类CControlBar
  - CControlBar是控件栏基类，实现工具条、状态条与浮动对话框
- CControlBar派生类
  - CStatusBar：状态条
  - CToolBar：带位图按钮的工具条
  - CDialogBar：控件条形式的浮动对话框

# 可视对象类(5)

- 绘图对象类CGdiObject
  - MFC绘图对象基类，实现各种绘图对象
- CGdiObject派生类
  - CBitmap(位图)、CBrush(画刷)、CFont(字体)、CPalette(调色板)、CPen(画笔)、CRgn(区域)

# 可视对象类(6)

- 设备环境类CDC
  - MFC设备环境基类，用于绘图
- CDC派生类
  - CClientDC：客户区设备环境
  - CWindowDC：窗口设备环境
  - CMetaFileDC：图元文件设备环境

# 通用类(1)

- 文件类CFile
  - 文件访问基类，实现文件访问
- CFile派生类
  - CMemFile：支持内存文件访问
  - CStdioFile：支持流式文件访问
- CArchive类
  - 与CFile以串行化实现文件访问

# 通用类(2)

- 异常类CException
  - CNotSupportException：不支持异常
  - CMemoryException：内存异常
  - CFileException：文件异常
  - CResourceException：资源异常
  - COleException：OLE异常
  - CDBException：数据库异常
  - CUserException：用户操作异常

# 通用类(3)

- 模板收集类
  - CArray与CTypedPtrArray：将对象/指针存储到数组
  - CList与CTypedPtrList：将对象/指针存储到链表
  - CMap与CTypedPtrMap：将键/指针映射到值

# OLE类

- OLE是对象链接与嵌入，处理复合文档的方法
  - 普通类：COleDocument、COleItem
  - 客户类：COleClientDoc、COleClientItem
  - 服务类：COleServer、COleTemplate
  - 可视编辑容器类：COleLinkingDoc
  - 传输类：COleDropSource、COleDropTarget
  - 对话类：COleInsertDialog

# 数据库类

- ODBC类是MFC数据库访问类，访问支持ODBC的数据库，完成查询、更新等
  - CDatabase：连接数据源
  - CRecordset：数据源的一组记录
  - CRecordView：表单模式视图
  - CFieldExchange：上下文信息交换
  - CLongBinary：二进制对象句柄

# 网络通信类

- Internet类
  - CInternetSession类、CInternetFile类、CInternetConnection类、CFileFind类、CGopherLocator类
- Socket类
  - CSocket类、CAsyncSocket类

# 单选题 1分

设置

Windows程序入口来自哪个类?

- CMainFrame
- CWinApp
- CView
- CDialog

提交

# MFC全局函数

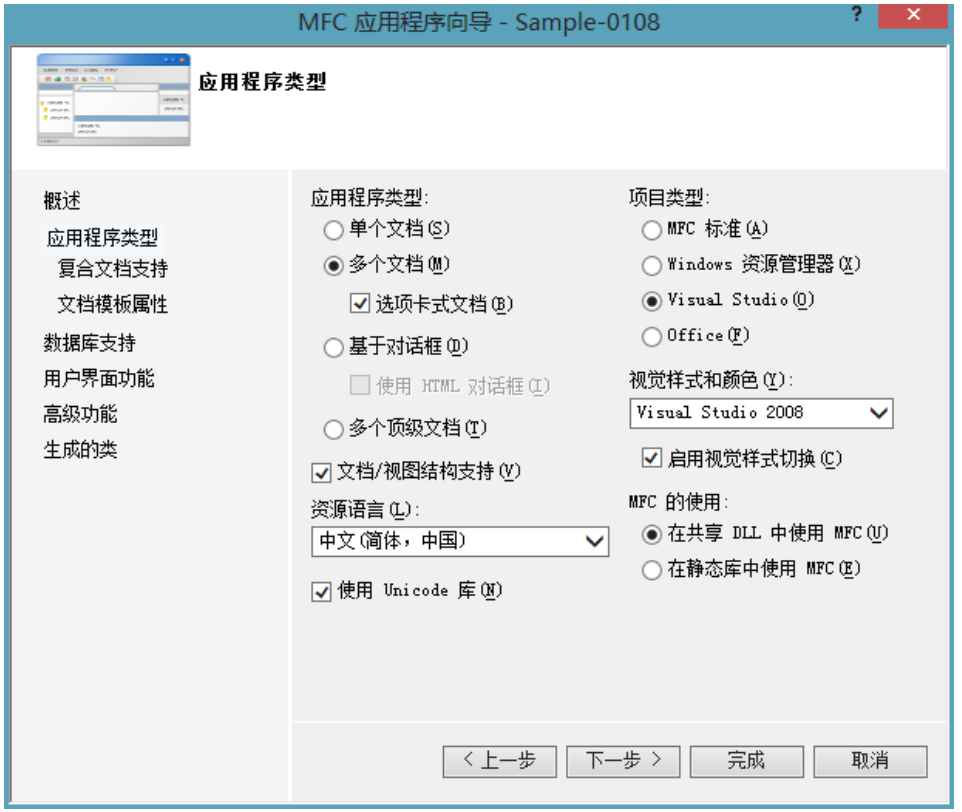
- 前缀为Afx的函数

| 函数名 | 功能 |
| --- | --- |
| AfxAbort | 终止一个应用程序 |
| AfxBeginThread | 创建并执行一个线程 |
| AfxEndThread | 终止正在执行的线程 |
| AfxMessageBox | 弹出一个消息框 |
| AfxGetApp | 返回当前程序对象的指针 |
| AfxRegisterWndClass | 注册一个窗口类 |

# MFC向导功能(1)

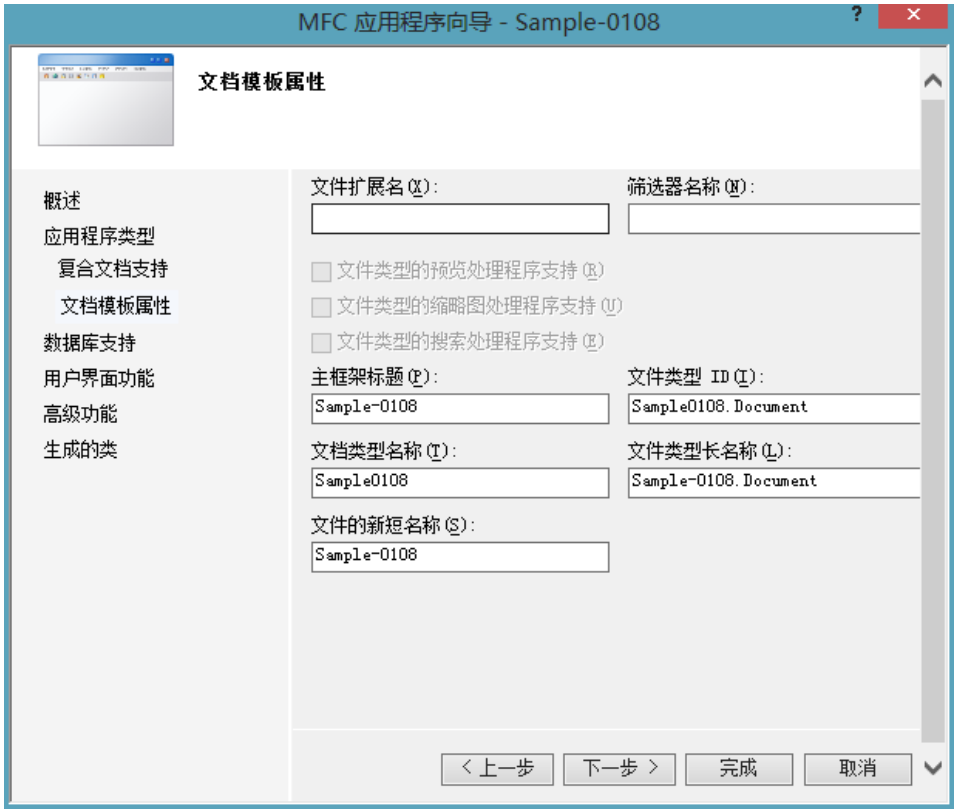
第1步：概述


第2步：应用程序类型



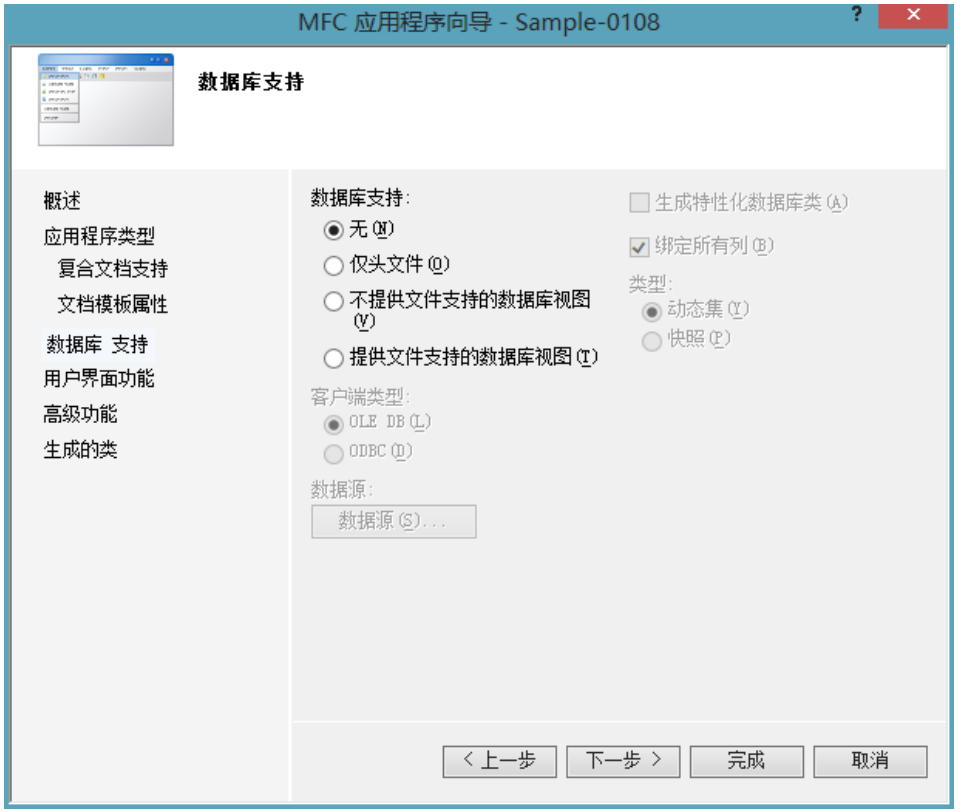
# MFC向导功能(2)

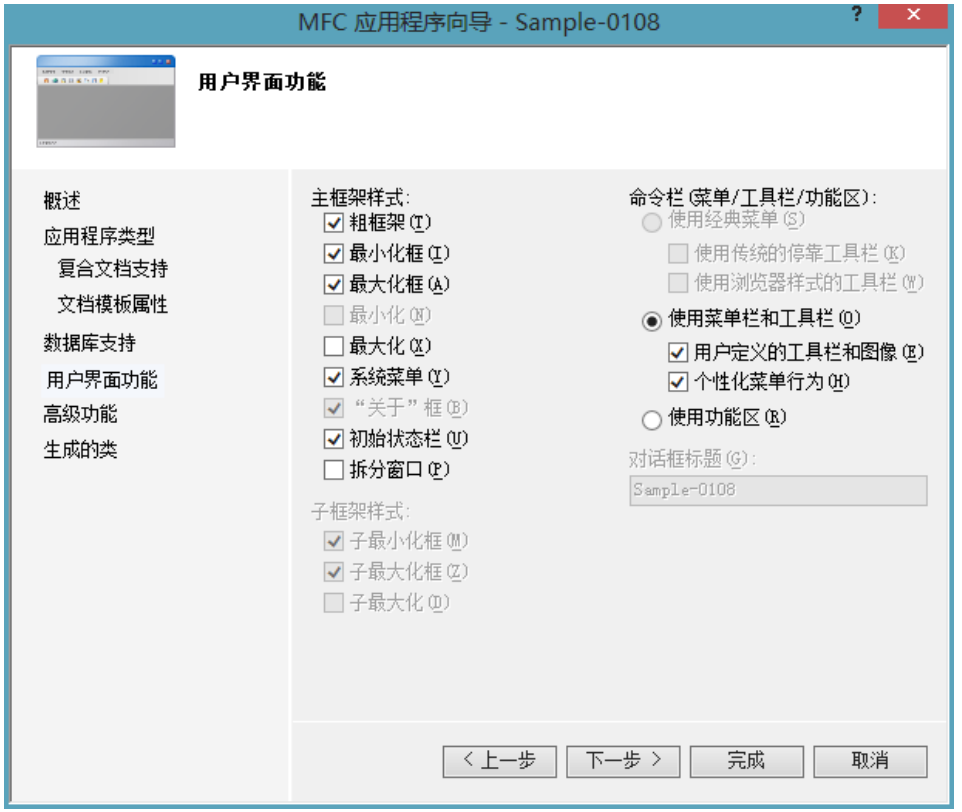
## 第3步：复合文档支持　第4步：文档模板属性

# MFC向导功能(3)

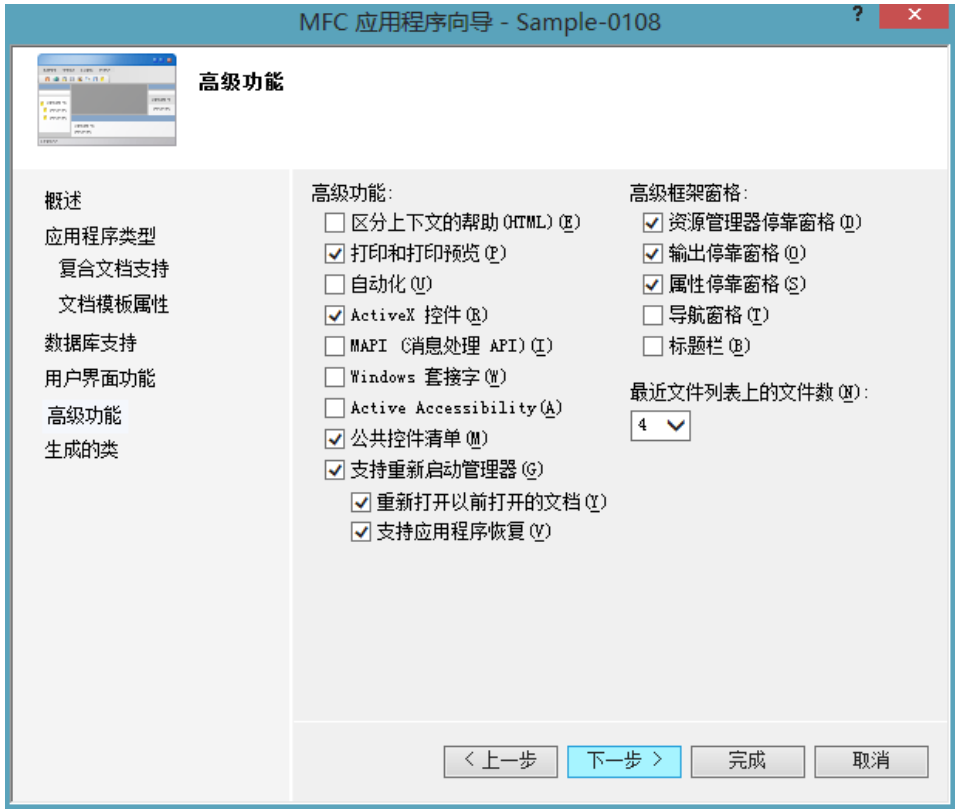
## 第5步：数据库支持　　第6步：用户界面功能

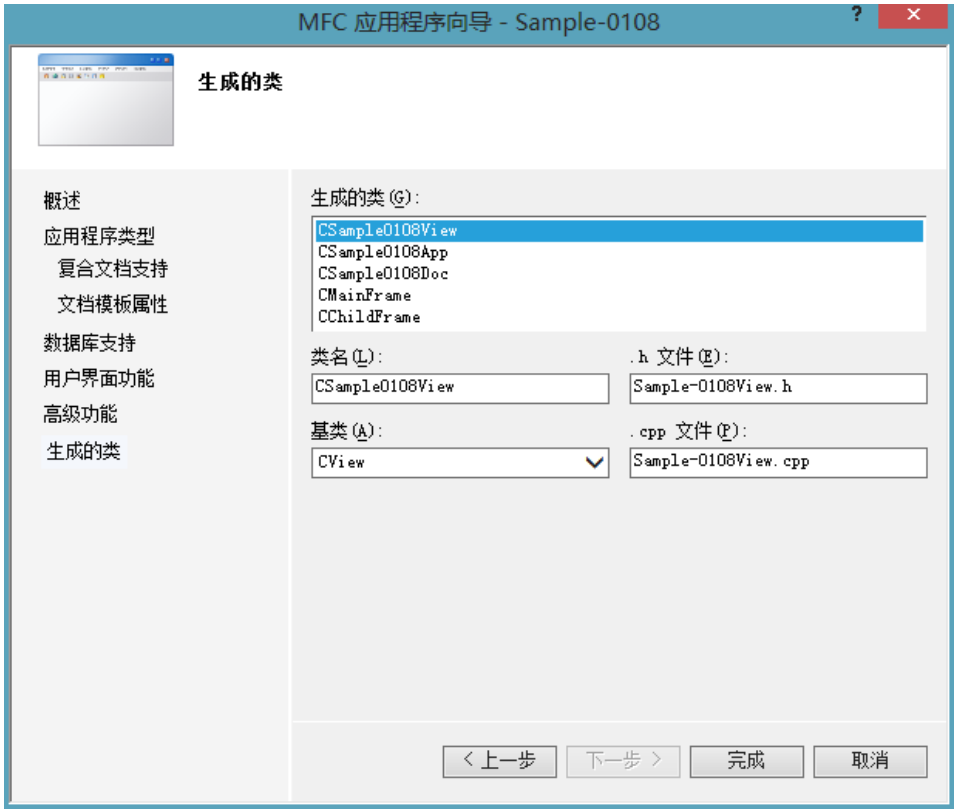


# MFC向导功能(4)

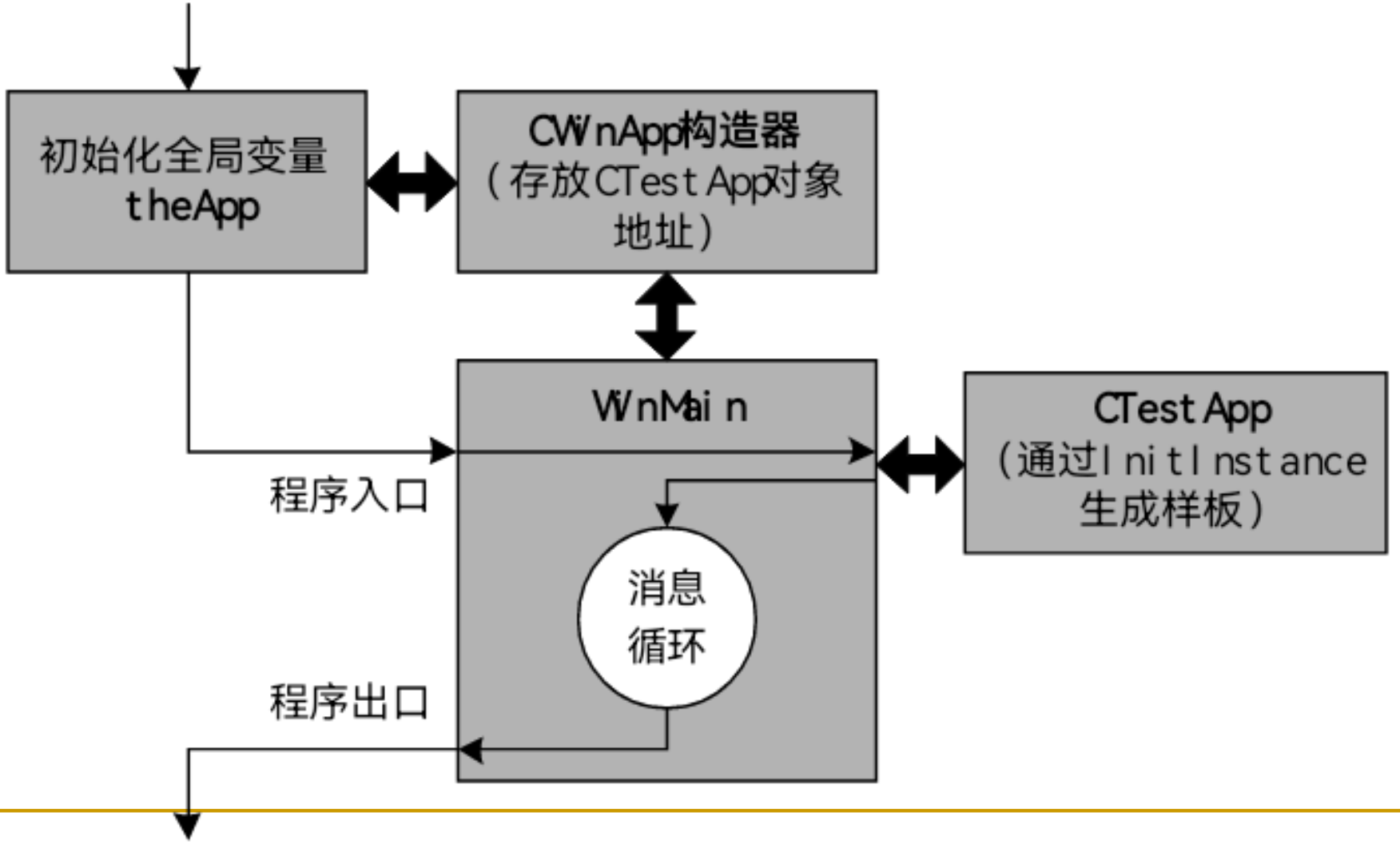
## 第7步：高级功能　　第8步：生成的类



# MFC程序框架分析(1)

- Windows程序核心是CWinApp



# MFC程序框架分析(2)

- 每次启动新的应用程序，WinMain函数都调用InitInstance()
- 创建并注册文档模板

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate=new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CTestDoc),
                RUNTIME_CLASS(CMainFrame),
                RUNTIME_CLASS(CTestView));
AddDocTemplate(pDocTemplate);
```

# MFC程序框架分析(3)

- 装载标准文件选项

```
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
ProcessShellCommand(cmdInfo);
```

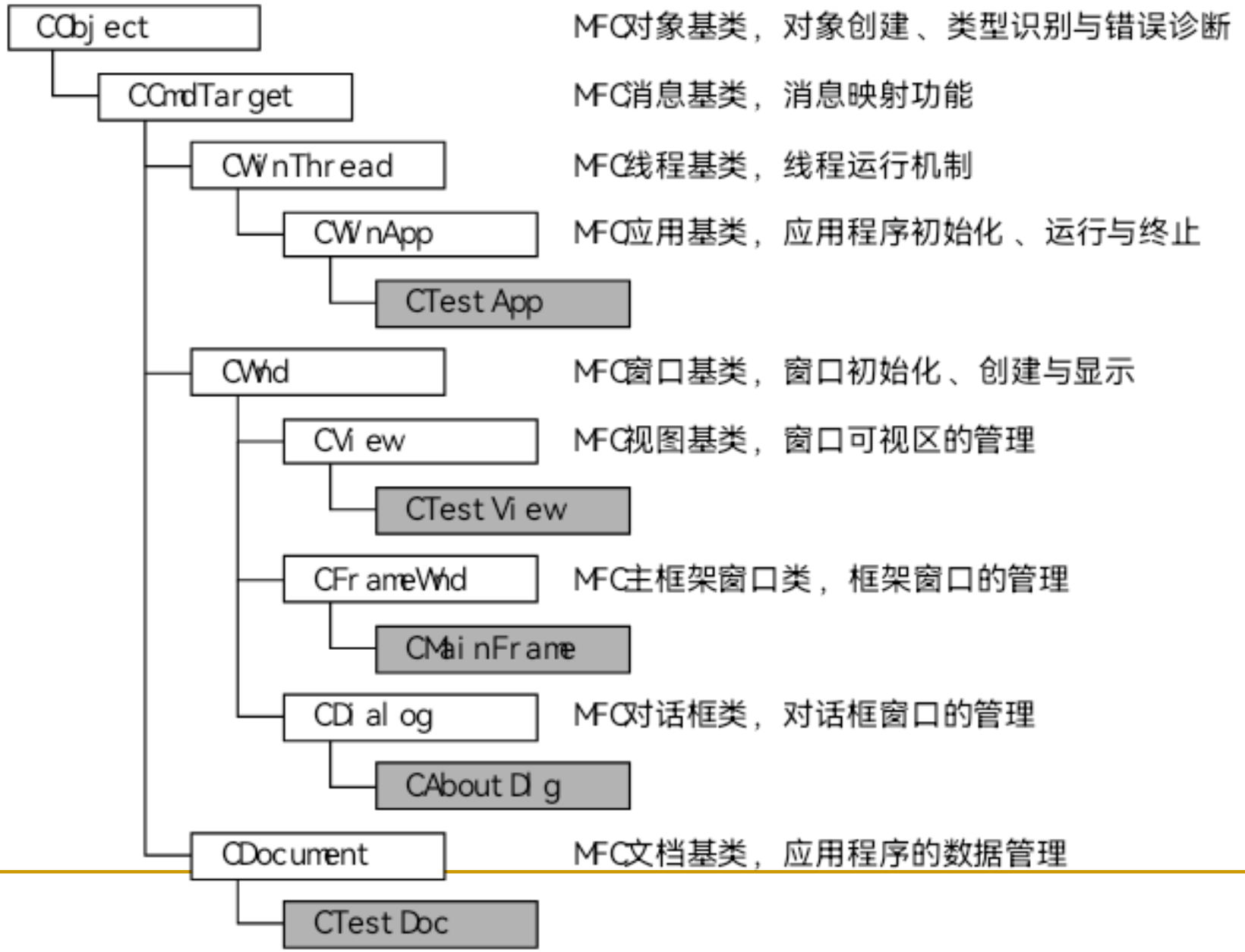
- 创建主边框窗口

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
```

# MFC程序框架分析(4)

- 文档模板存放文档、视图和边框窗口信息
  - CSingleDocTemplate(单文档模板)
  - CMultiDocTemplate(多文档模板)
- 传给文档模板的资源符号串包括多个参数，每个参数之间用“\n”隔开，例如WindowTitle、DocName、FilterName等

# MFC程序框架分析(5)


CObject
MFC对象基类，对象创建、类型识别与错误诊断
CCmdTarget
MFC消息基类，消息映射功能
CWinThread
MFC线程基类，线程运行机制
CWinApp
MFC应用基类，应用程序初始化、运行与终止
CTestApp
CWnd
MFC窗口基类，窗口初始化、创建与显示
CView
MFC视图基类，窗口可视区的管理
CTestView
CFrameWnd
MFC主框架窗口类，框架窗口的管理
CMainFrame
CDialog
MFC对话框类，对话框窗口的管理
CAboutDlg
CDocument
MFC文档基类，应用程序的数据管理
CTestDoc

# 消息的概念(1)

- 消息处理机制是Windows核心，它是应用程序运行的动力
- 消息是一个32位整数值，唯一定义一个事件，向Windows系统发出通知，告诉应用程序发生的事件

# 消息的概念(2)

- PeekMessage：查看队列，仅检测消息
- GetMessage：查看队列，取走消息
- PreTranslateMessage：预处理消息
- TranslateMessage：虚拟键转化为字符，例如Shift+8→*
- DispatchMessage：派发消息，找到处理函数

# 消息的概念(3)

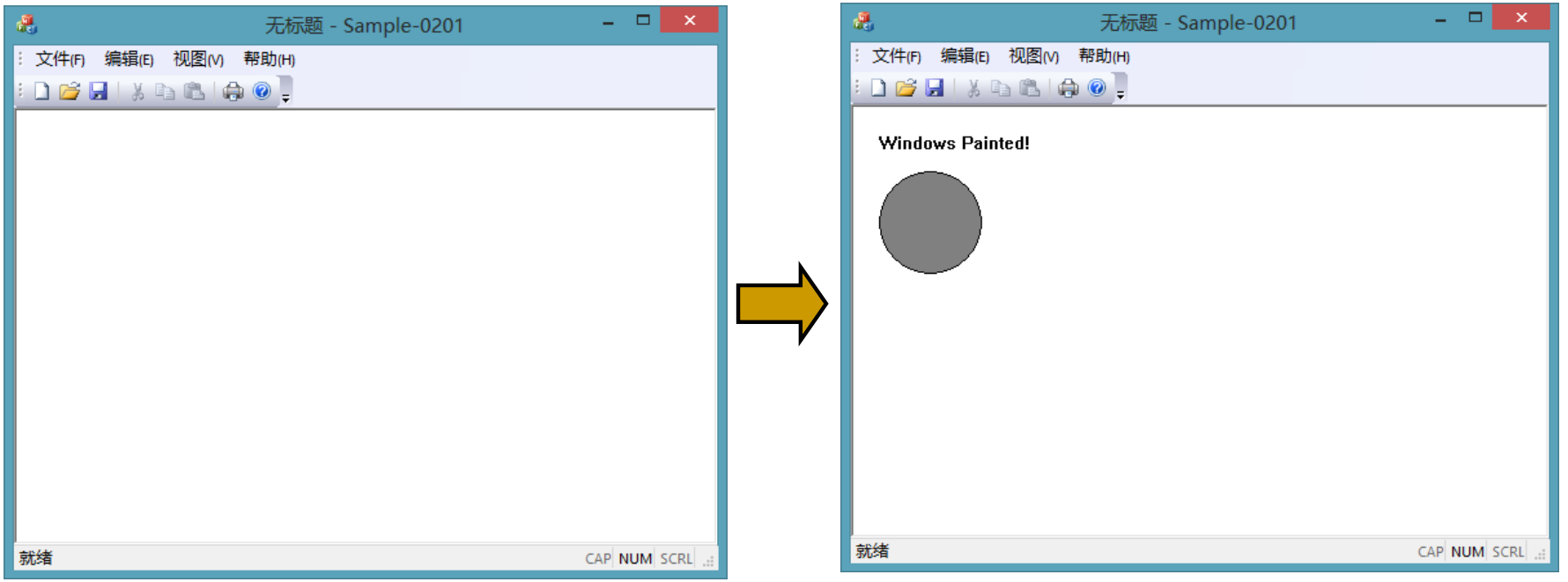
```
//TestView.h
class CTestView : public CView
{ DECLARE_MESSAGE_MAP()
public:
  afx_msg void OnLButtonDown(UINT nFlags,CPoint point);
  afx_msg void OnEditPaste(); };
//TestView.cpp
BEGIN_MESSAGE_MAP(CTestView,CView)
  ON_WM_LBUTTONDOWN()
  ON_COMMAND(ID_EDIT_PASTE,OnEditPaste)
END_MESSAGE_MAP()
```

# 消息控制机制(1)

例2-1

- 在CTestView::OnDraw()中

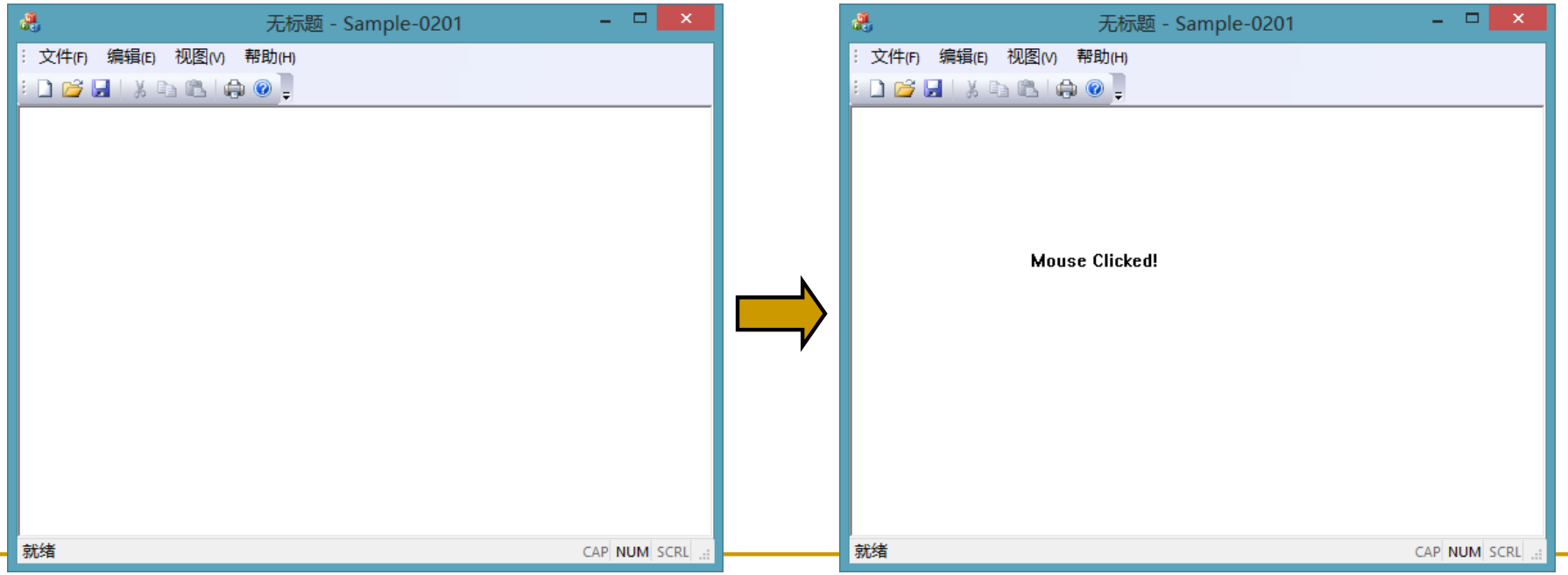
```
pDC->TextOutW(20,20,L"Windows Painted!");
pDC->SelectStockObject(GRAY_BRUSH);
pDC->Ellipse(20,50,100,130);
```



# 消息控制机制(2)

- 鼠标控制消息

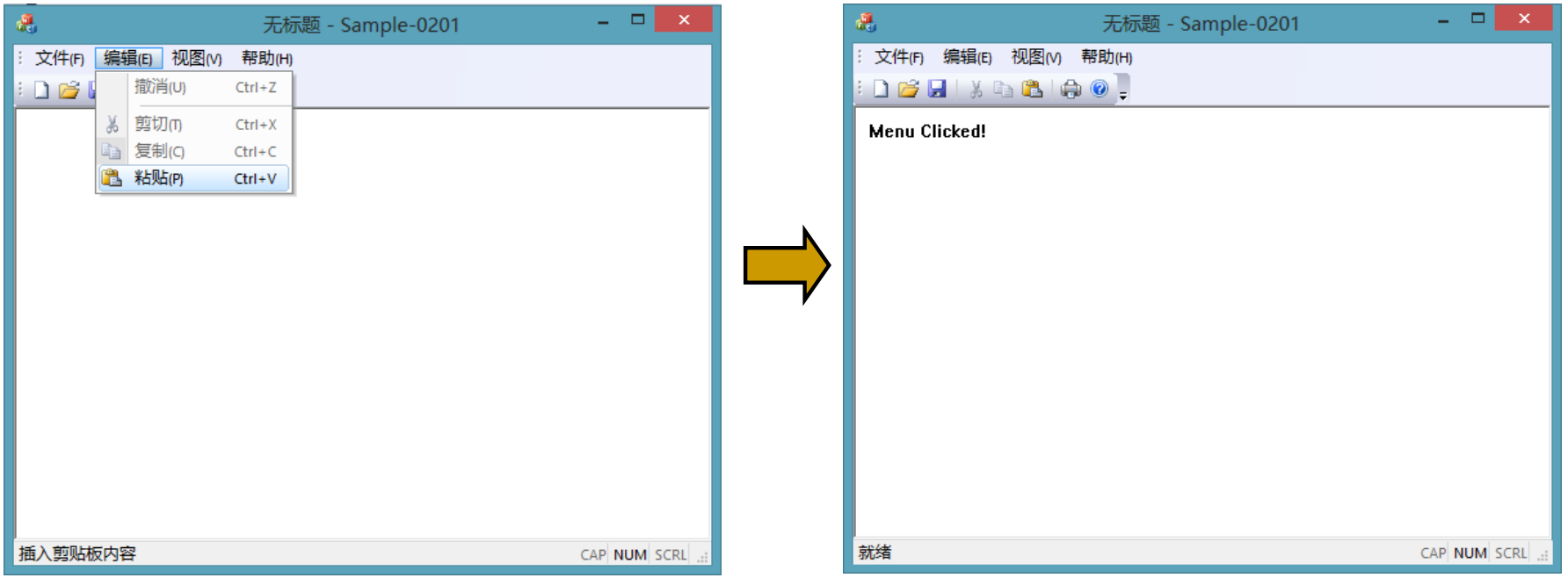
```
CDC *pDC=GetDC();
pDC->TextOutW(point.x,point.y,L"Mouse Clicked!");
```



# 消息控制机制(3)

- 菜单控制消息

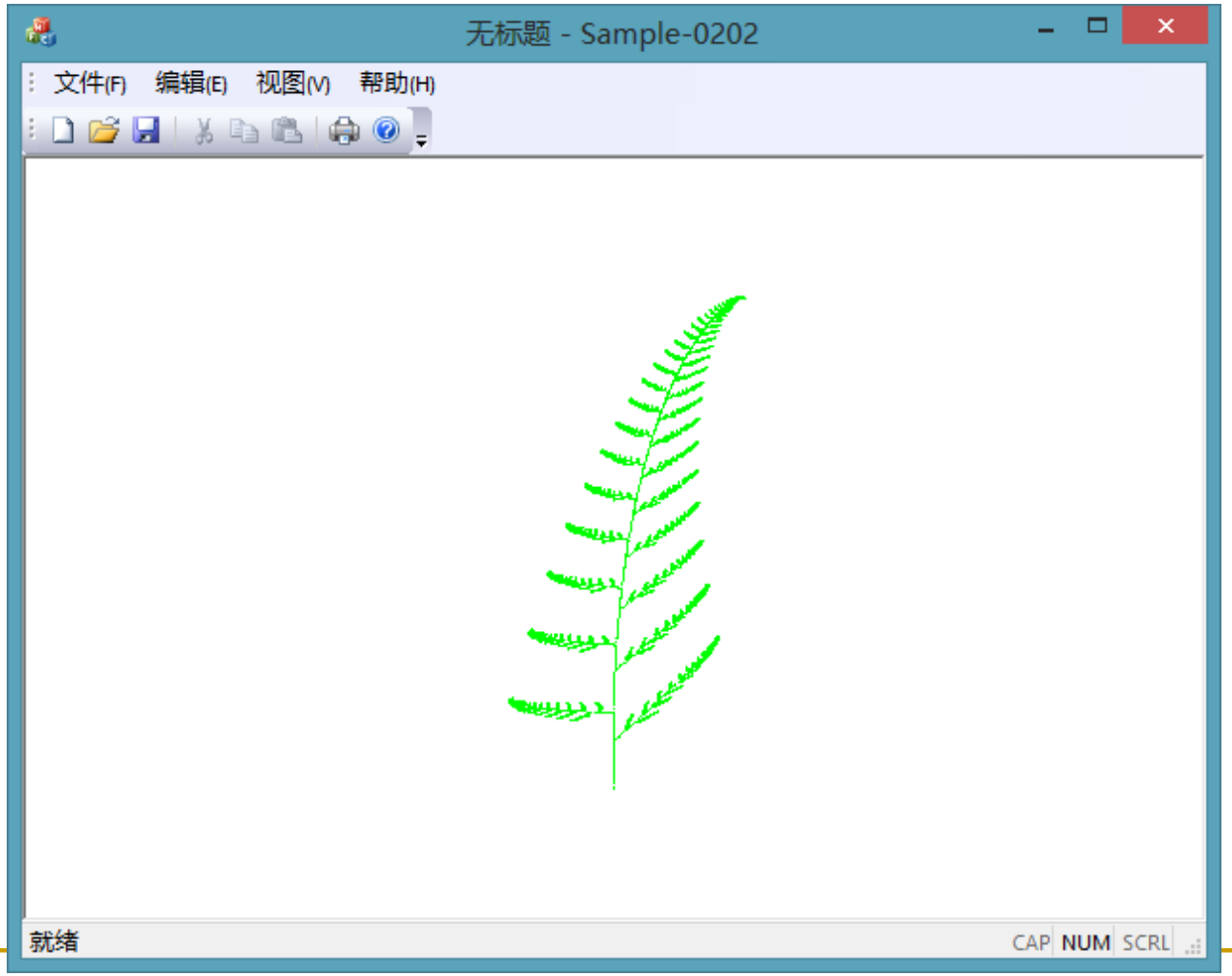
```
CDC *pDC=GetDC();
pDC->TextOutW(10,10,L"Menu Clicked!");
```



# 趣味性例子(1)

例2-2

- 绘制一片绿色的叶子



# 趣味性例子(2)

```
int nTotalPoints=32000;
CRect rect;
GetClientRect(&rect);
int nX=rect.right/2;
int nY=rect.bottom*5/6;
int nScale=(rect.right>rect.bottom?rect.bottom:
rect.right)/15;
COLORREF crColor=0x00FF00;
double dX=0, dY=0;
double dP;
for(int i=0;i<nTotalPoints;i++)
{ dP=1.0*rand()/RAND_MAX;
```

# 趣味性例子(3)

```
  if(dP<=0.01)
  { dX=0;  dY=0.16*dY; }
  if(dP>0.01 && dP<=0.86)
  { dX=0.85*dX+0.04*dY; dY=-0.04*dX+0.85*dY+1.60; }
  if(dP>0.86 && dP<=0.93)
  { dX=0.20*dX-0.26*dY; dY=0.44*dX+0.12*dY+1.60; }
  if(dP>0.93)
  { dX=-0.20*dX+0.26*dY; dY=0.44*dX+0.12*dY+1.00; }
  pDC->SetPixel(nX+int(dX*nScale),nY-int(dY*nScale),
crColor);
}
```

# 消息的分类(1)

- Windows系统将事件以消息发送给目标，目标按消息内容进行处理
  - 目标窗口
  - 消息类型
  - 参数wParam
  - 参数lParam

# 消息的分类(2)

- 标准消息
  - 窗口消息(WM_CREATE、WM_DESTROY)、鼠标消息(WM_LBUTTONDOWN、WM_MOUSEMOVE)、键盘消息(WM_KEYDOWN、WM_CHAR)、滚动消息(WM_HSCROLL)、计时器消息(WM_TIMER)
- 控件消息
  - 控件传递给父窗口的消息
- 命令消息
  - 界面对象(包括菜单、工具栏、加速键等)的WM_COMMAND消息

# 消息处理过程(1)

- 标准消息由触发窗口处理，处理函数在对应窗口类中定义
- 控件消息由容器窗口处理，处理函数在对应窗口类中定义

```
  DECLARE_MESSAGE_MAP()
public:
  afx_msg int OnCreate();
  afx_msg void OnLButtonDown();
```

# 消息处理过程(2)

- WM_COMMAND能被更多对象处理，包括应用程序、边框窗口、视图、文档等
- 命令消息通过命令目标链发送，每个目标检查自己的消息映射，决定能否处理
- 命令目标链处理顺序：当前活动子目标、自己、其它目标

# 消息处理过程(3)

- 命令处理顺序

| 接收命令的类 | 命令处理顺序 |
| --- | --- |
| MDI主边框窗口 | 当前MDI子边框窗口→MDI主边框窗口→应用程序 |
| SDI主边框窗口<br>MDI子边框窗口 | 当前视图→SDI主边框窗口(或MDI子边框窗口→主边框窗口)→应用程序 |
| 视图 | 视图→文档 |
| 文档 | 文档→文档模板 |
| 对话框 | 对话框→父窗口→应用程序 |

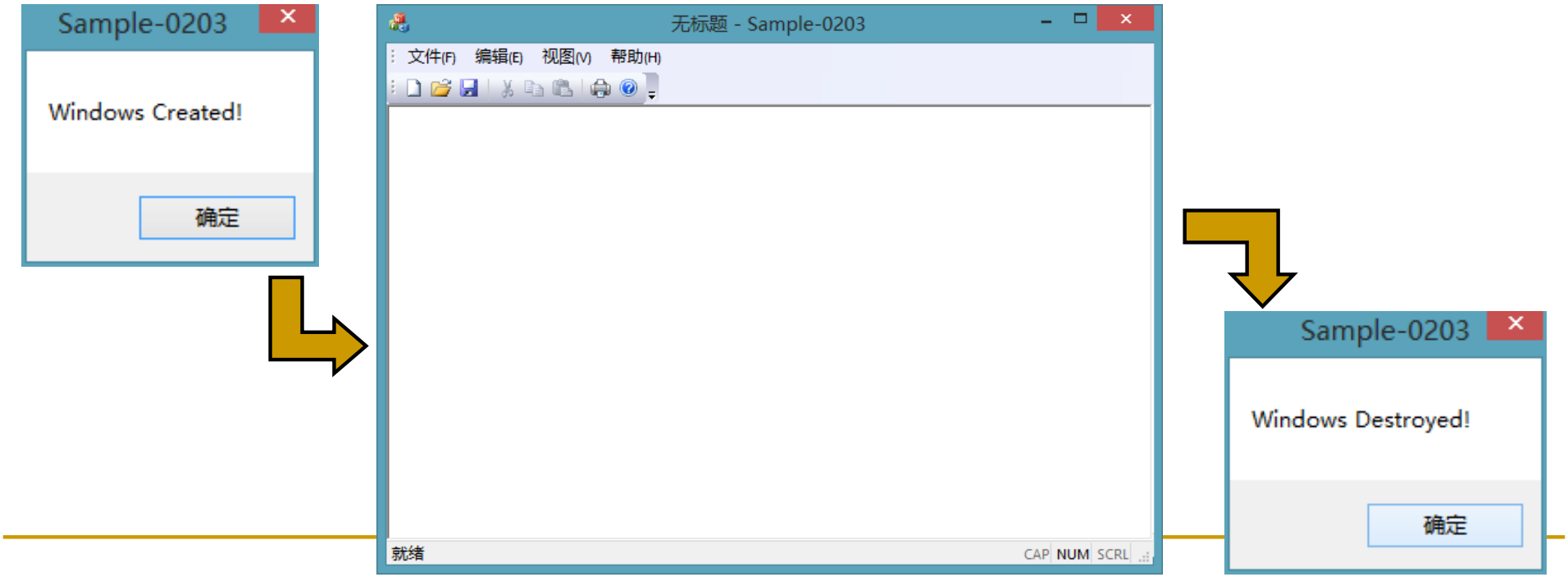
# 窗口消息(1)

- WM_CREATE消息
  - 程序打开，发送WM_CREATE，初始化窗口
- WM_DESTROY消息
  - 程序退出，发送WM_DESTROY，销毁窗口
- WM_PAINT消息
  - 窗口变化，发送WM_PAINT，重绘窗口

# 窗口消息(2)

例2-3

- 添加WM_CREATE消息
  - MessageBox(L"Windows Create!");
- 添加WM_DESTROY消息
  - MessageBox(L"Windows Destroy!");



# 窗口消息(3)

- 在CTestView类定义中

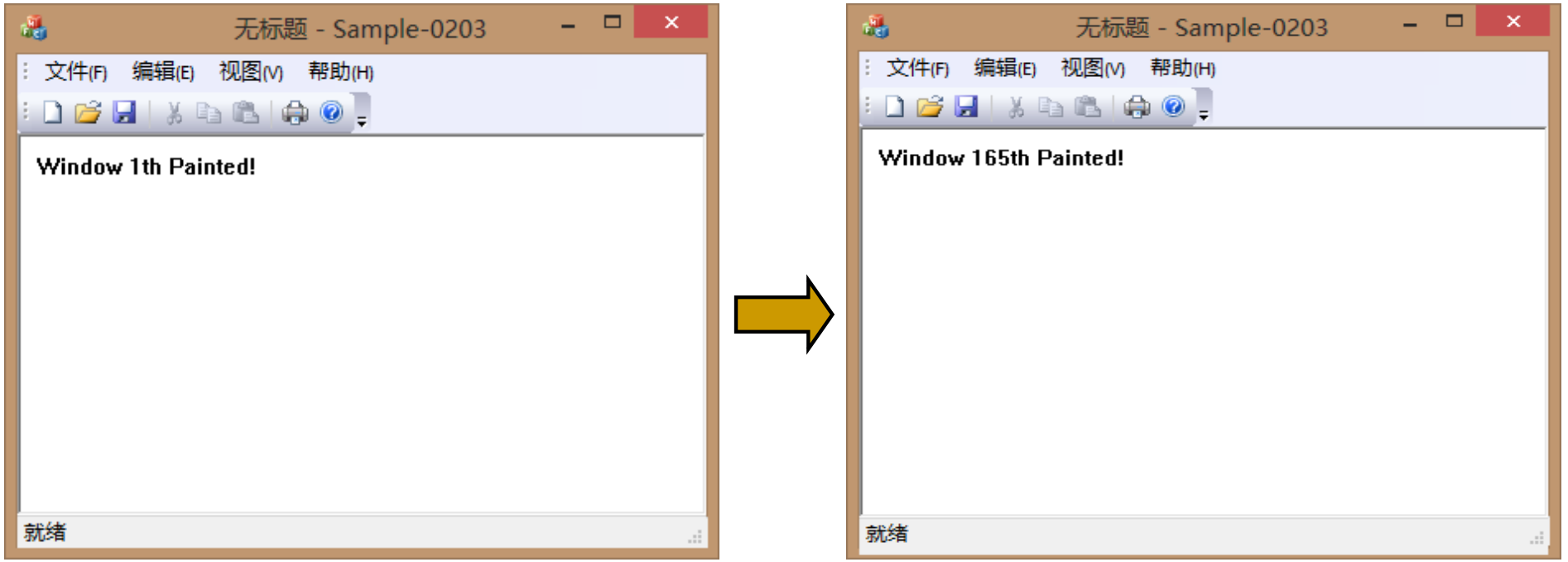
```
private:
  int m_num;
```

- 在CTestView::OnDraw()中

```
CString str;
str.Format(L"Window %dth Painted!",
m_num++);
pDC->TextOutW(10,10,str);
```

# 窗口消息(4)

- WM_PAINT消息→CTestView::OnPaint()
  →CTestView::OnDraw()



# 鼠标消息(1)

- 用户操作鼠标时，产生对应消息，系统将消息发送给窗口
- 鼠标消息主要包括:
  - WM_LBUTTONDOWN(左键按下，右键R/中键M)
  - WM_LBUTTONUP(左键释放，右键R/中键M)
  - WM_LBUTTONDBLCLK(左键双击，右键R/中键M)
  - WM_MOUSEMOVE(鼠标移动)
  - WM_MOUSEWHEEL(滑轮滚动)

# 鼠标消息(2)

- 鼠标消息处理函数参数：nFlag和point
- nFlag：事件发生时，键盘或鼠标键状态，由对应的位表示
  - MK_CONTROL、MK_SHIFT、MK_LBUTTON、MK_MBUTTON、MK_RBUTTON
- point：事件发生时，光标所处位置(客户区)

# 鼠标消息(3)


- 在CTestView::OnLButtonDown()中

```
CDC* pDC=GetDC();
pDC->TextOutW(point.x,point.y,L"Mouse
Clicked!");
ReleaseDC(pDC);
```

- 当窗口尺寸变化，哪些信息保留，哪些消失？如何保留最新信息？

# 鼠标消息(4)

- 在CTestView类定义中

```
private:
  CPoint m_pos;
  CString m_str;
```

- 在CTestView::OnDraw()中

```
pDC->TextOutW(m_pos.x,m_pos.y,m_str);
```

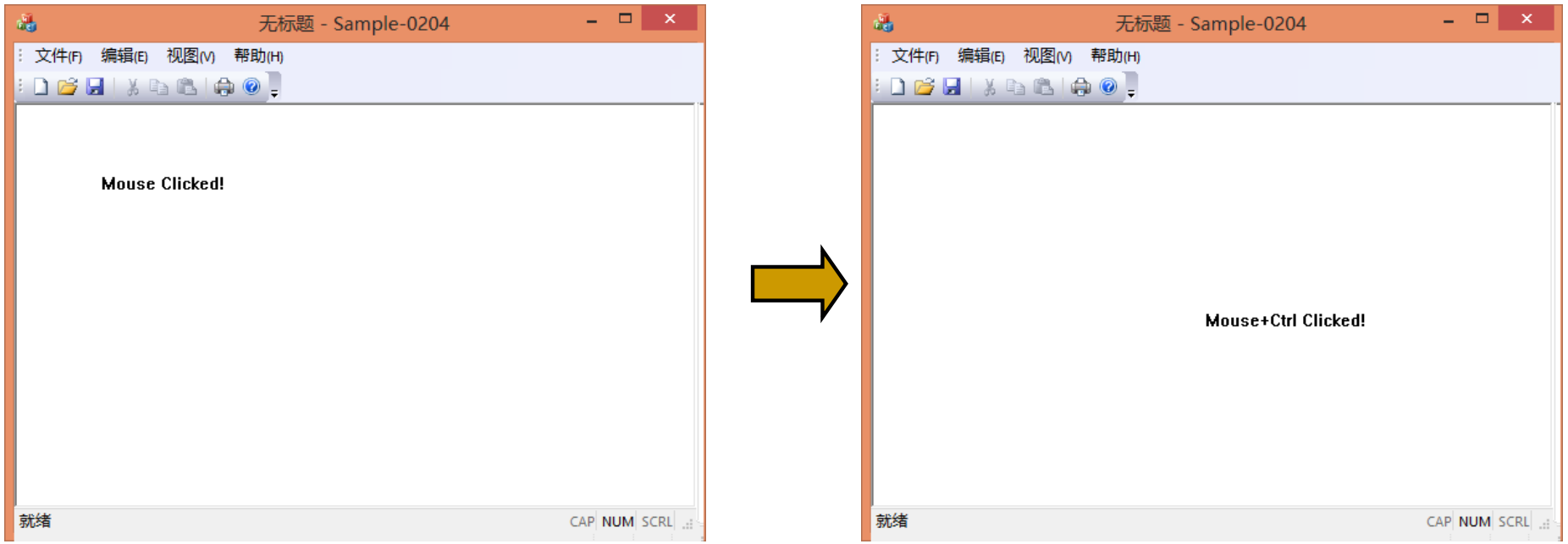
# 鼠标消息(5)

- 在CTestView::OnLButtonDown()中

```
m_pos=point;
if(nFlags & MK_CONTROL)
  m_str=L"Mouse+Ctrl Clicked!";
else
  m_str=L"Mouse Clicked!";
Invalidate(true);
```

# 鼠标消息(6)

- 鼠标单击与标志位处理



# 键盘消息(1)

- 用户操作键盘时，产生对应消息，系统将消息发送给窗口
- 键盘消息主要包括：
  - WM_KEYDOWN：键盘按下
  - WM_KEYUP：键盘释放
  - WM_CHAR：输入一个字符

# 键盘消息(2)


- 在CTestView::OnChar()中

```
void CTestView::OnChar(UINT nChar,UINT
nRepCnt,UINT nFlags)
{
  CString str;
  str.Format(L"%c Key Entered!",nChar);
  MessageBox(str);
}
```

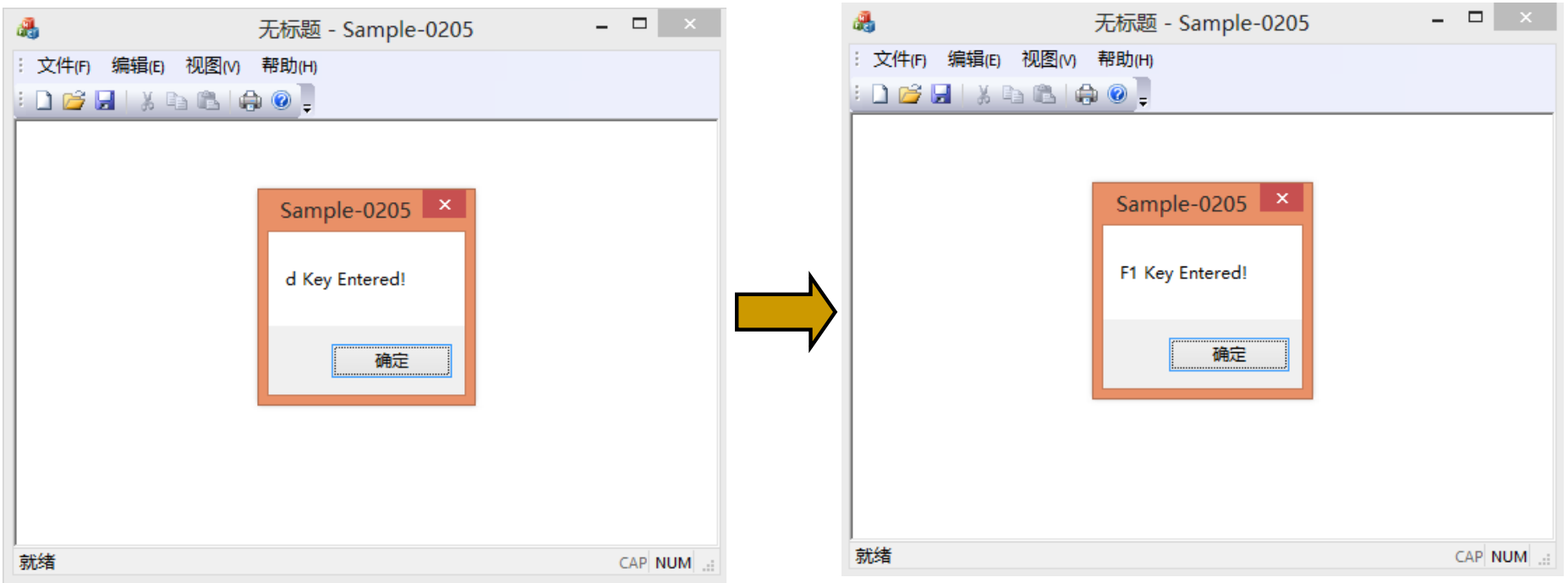
# 键盘消息(3)

- 特殊键处理，例如F1、Ctrl、↑

```
BOOL CTestView::PreTranslateMessage(MSG*
pMsg)
{ if(pMsg->message==WM_KEYDOWN)
  { if(pMsg->wParam==VK_F1)
      MessageBox(L"F1 Key Entered!");
    if(pMsg->wParam==VK_UP)
      MessageBox(L"↑Key Entered!");  }
}
```

# 键盘消息(4)

- 普通键与特殊键处理

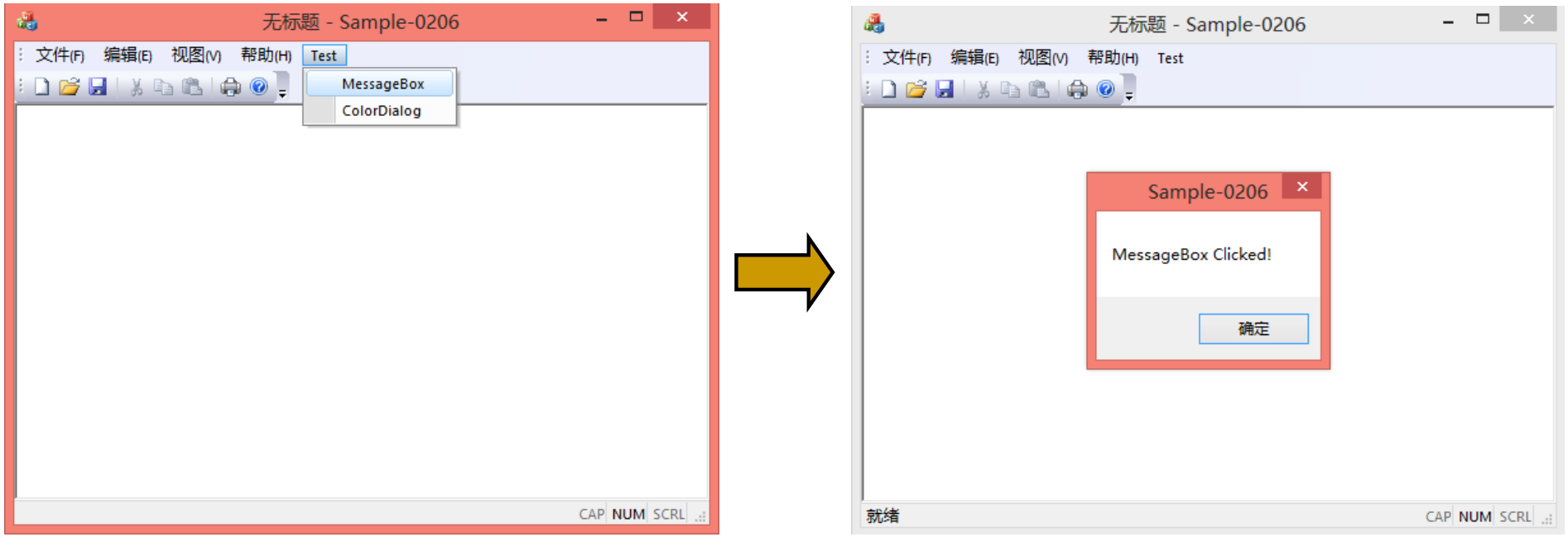


# 菜单与工具栏消息(1)


- 添加菜单消息
  - ResourceView→Menu→IDR_MAINFRAME
  - 一级菜单项“Test”→二级菜单项“MessageBox”(ID_TEST_MESSAGEBOX)
  - 一级菜单项“Test”→二级菜单项“ColorDialog”(ID_TEST_COLORDIALOG)

# 菜单与工具栏消息(2)

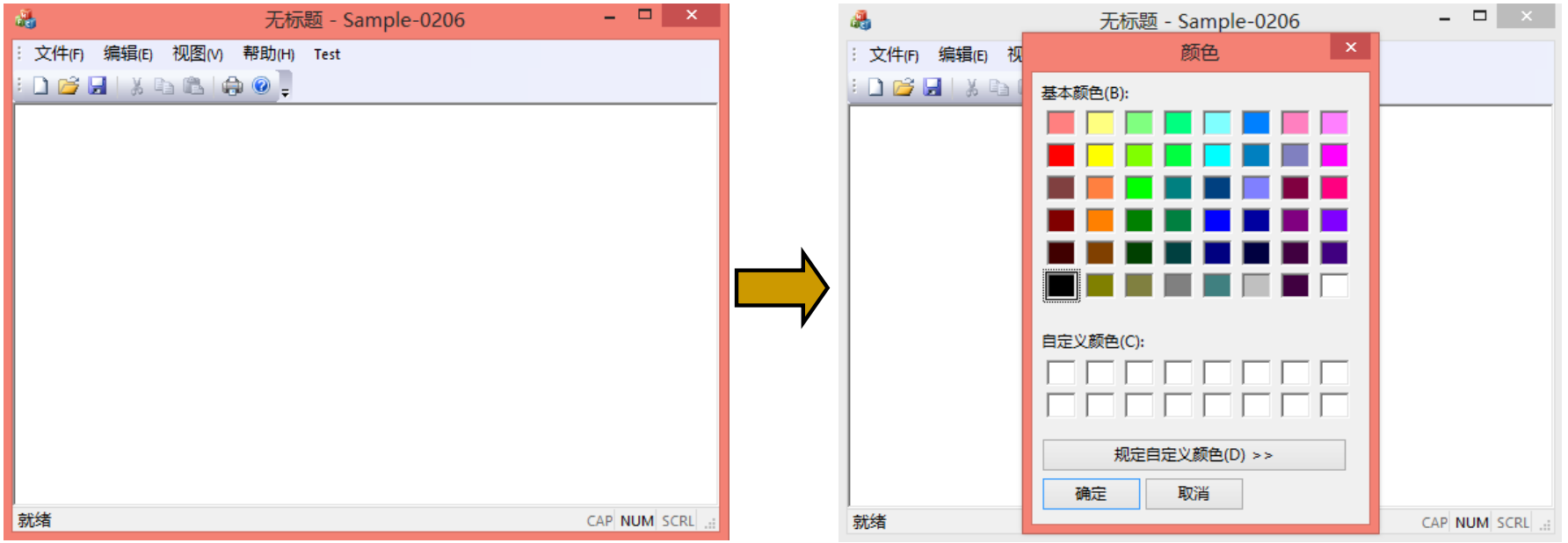
- 在CTestView::OnTestMessage()中
  - MessageBox(L"MessageBox Clicked!");



# 菜单与工具栏消息(3)

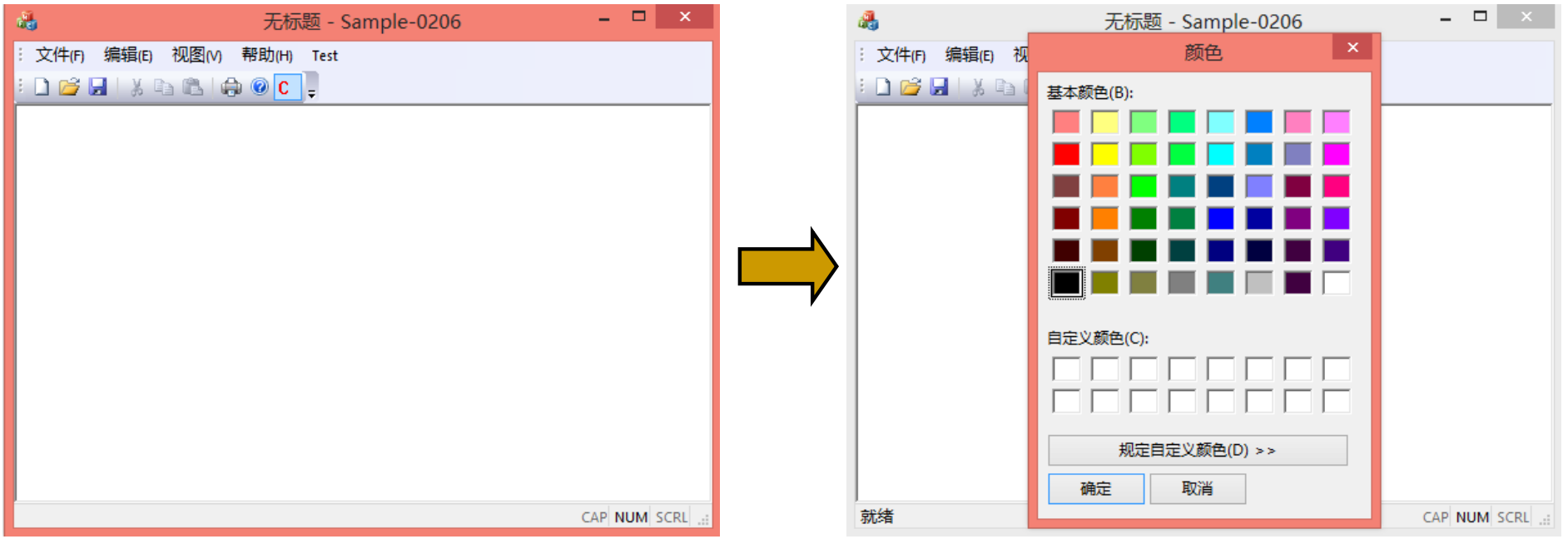
- 在CTestView::OnTestColor()中

```
CColorDialog dlg;
dlg.DoModal();
```



# 菜单与工具栏消息(4)

- 添加工具栏按钮
  - ResourceView→ToolBar→IDR_MAINFRAME，添加按钮(ID_TEST_COLORDIALOG)



# 计时器消息(1)

- WM_TIMER是计时器消息，执行周期性操作
- 通过SetTimer()设置计时器，当到达预定时间间隔，系统产生WM_TIMER消息，并通过参数(nIDEvent)指出对应的计时器
- 通过KillTimer()销毁计时器

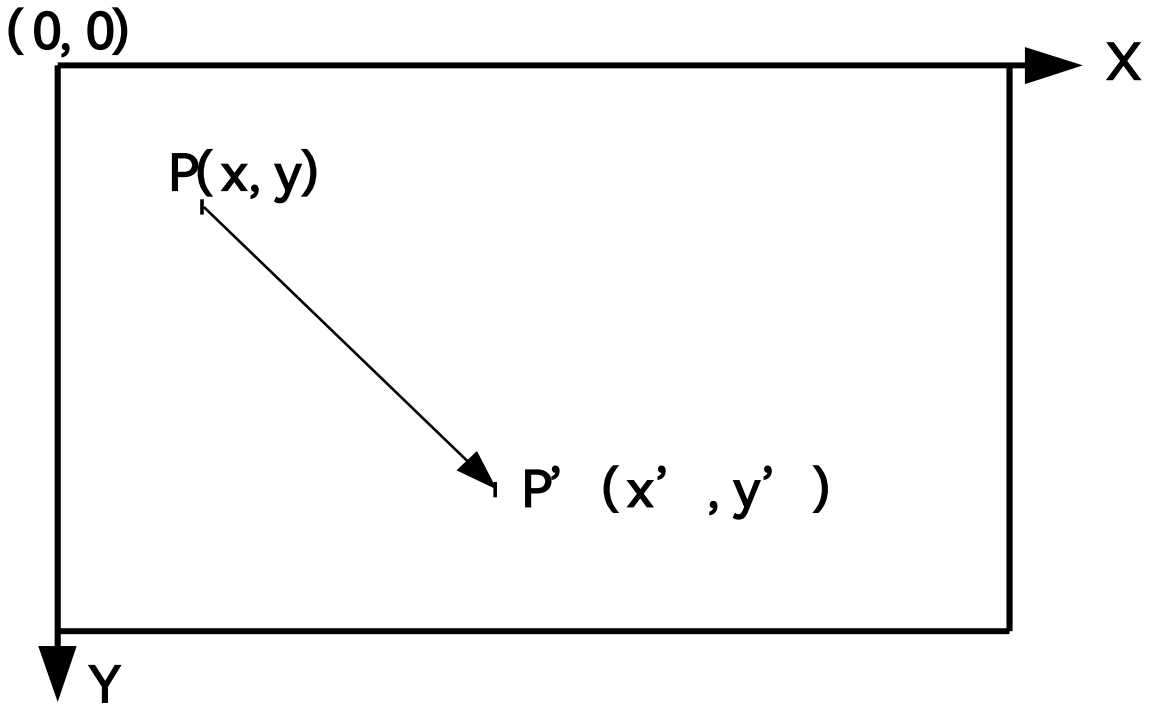
# 计时器消息(2)

- 对WM_CREATE消息，在OnCreate()中
  - SetTimer(1,2000,NULL);
- 对WM_TIMER消息，在OnTimer()中
  - if(nIDEvent==1) { ..... }
- 对WM_DESTROY消息，在OnDestroy()中
  - KillTimer(1);

# 计时器消息(3)

例2-7

- 使用WM_TIMER消息
- 异或方式制作动画：SetROP2(R2_XORPEN)




平移：

X' = X + MoveX

Y' = Y + MoveY

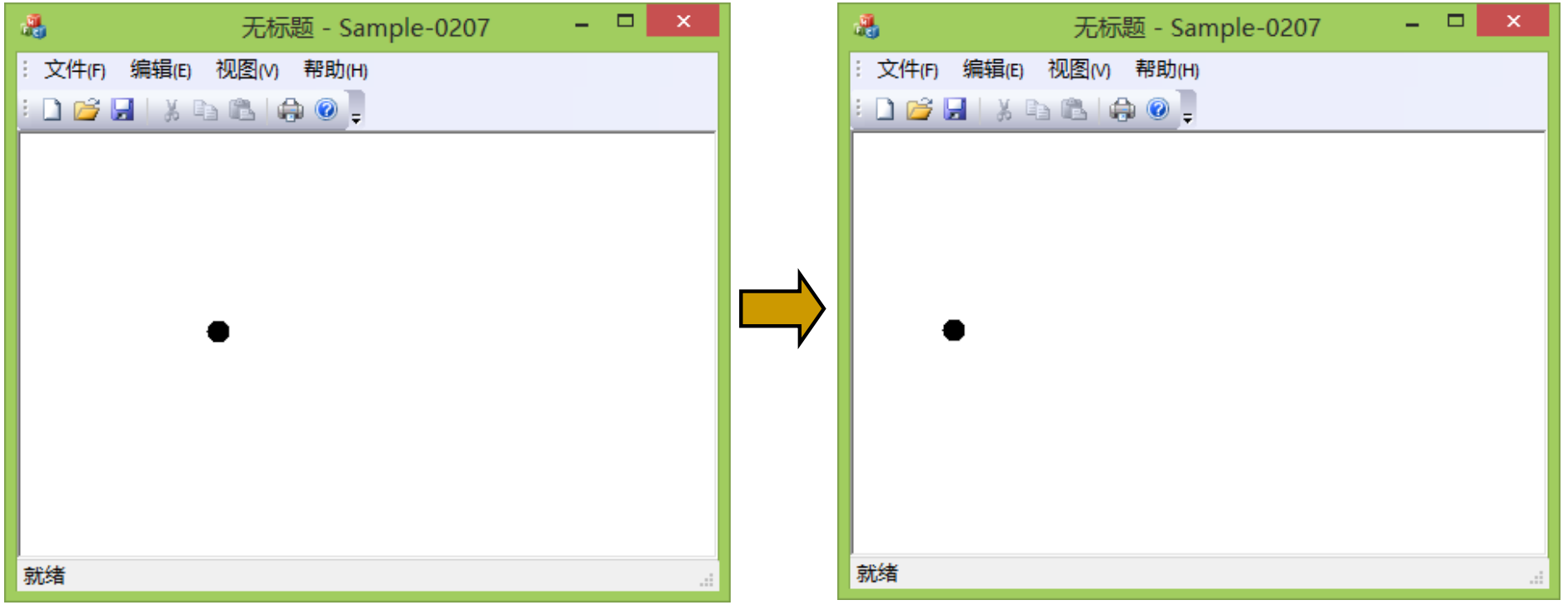
异或方式制作动画原理：

0∧a = a （画小球）

a∧a = 0 （擦除小球）

# 计时器消息(4)

- SmallBall程序效果



# 热键消息(1)


- 热键消息WM_HOTKEY
  - 不论程序处于前台或后台，当用户按某个热键，触发热键消息
- 在CTestView::OnHotKey()中

```
if(nHotKeyId==1001||nHotKeyId==1002)
  MessageBox(L"Hot Key Clicked!");
```

# 热键消息(2)

- 在CTestView::OnCreate()中

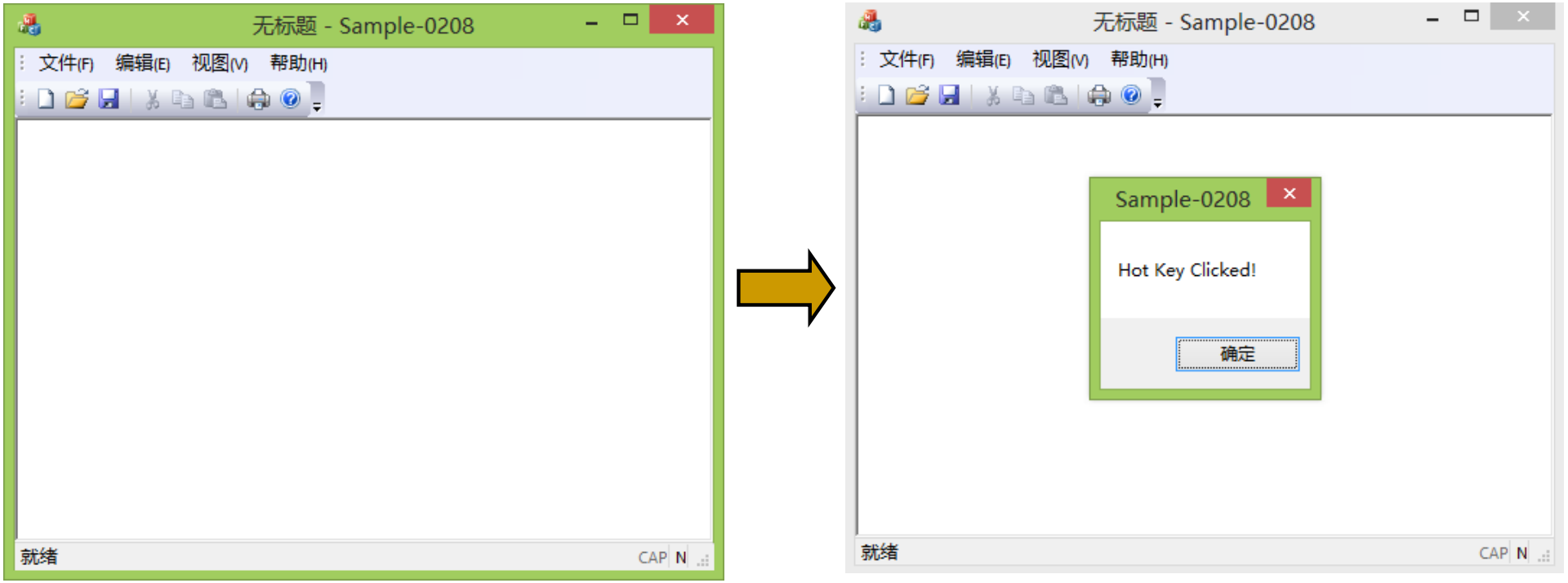
```
RegisterHotKey(m_hWnd,1001,
MOD_CONTROL| MOD_ALT,'z');
RegisterHotKey(m_hWnd,1002,
MOD_CONTROL| MOD_ALT,'Z');
```

- 在CTestView::OnDestroy()中

```
UnregisterHotKey(m_hWnd,1001);
UnregisterHotKey(m_hWnd,1002);
```

# 热键消息(3)

- 热键(Ctrl+Alt+Z)



# 自定义消息(1)

- 用户可自定义内部消息，区别系统定义消息
- 系统不知道消息存在，通过PostMessage()或SendMessage()发送
- 消息是一个整数，小于WM_USER的整数已用，大于的供用户使用
- 用户自定义消息的方式

```
#define WM_MYMESSAGE WM_USER+N
```

# 自定义消息(2)

- 自定义消息的操作步骤
  - 在适当位置声明消息处理函数
  - 将处理函数与消息对应
  - 实现消息处理函数
  - 向发送消息者提供窗口句柄

# 自定义消息(3)


- 声明用户自定义消息

```
#define WM_MYMESSAGE WM_USER+1
```

- 在CTestView类定义中

```
LRESULT OnMyMessage(WPARAM wParam,
LPARAM lParam);
```

- 在CTestView类中

```
ON_MESSAGE(WM_MYMESSAGE,
 OnMyMessage)
```

# 自定义消息(4)

- 在CTestView::OnMyMessage()中

```
CString str;
str.Format(L"Message Param is %d and %d",
wParam,lParam);
MessageBox(str);
```

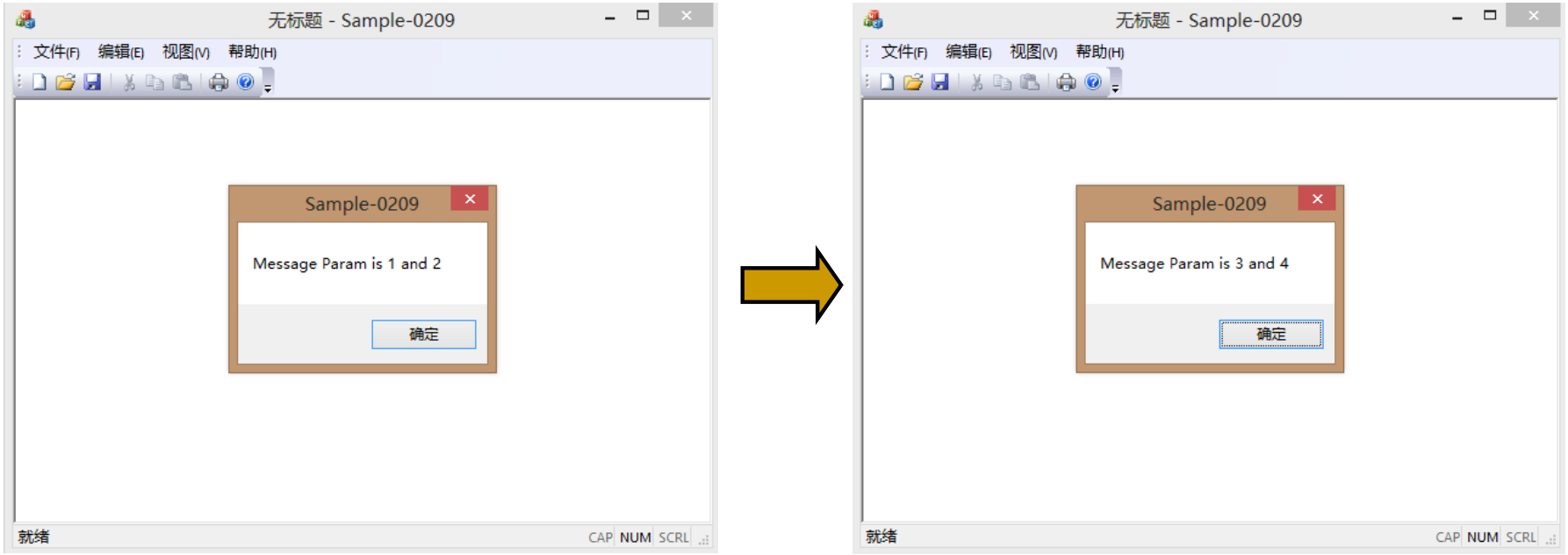
- 在CTestView::OnLButtonDown()中

```
PostMessage(WM_MYMESSAGE,1,2);
```

# 自定义消息(5)

- 在CTestView::OnKeyDown()中

```
SendMessage(WM_MYMESSAGE,3,4);
```



# 单选题 1分

设置

WM_PAINT消息在哪种情况下不被触发?

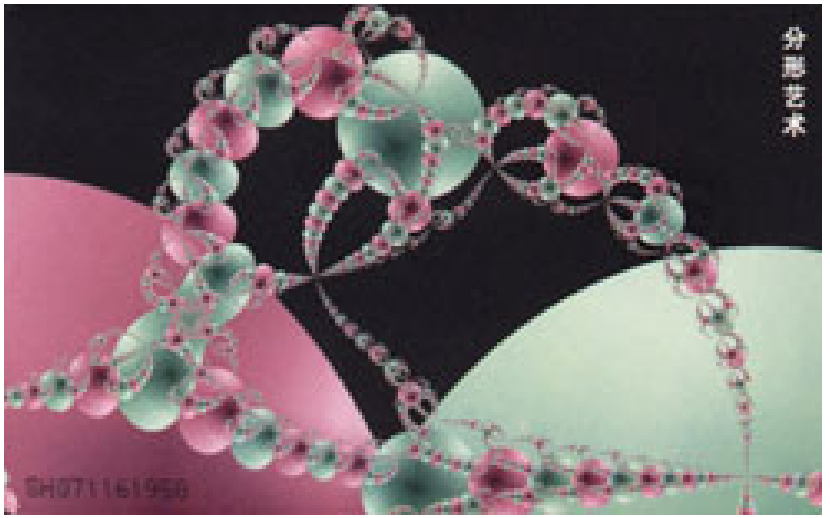
- 窗口位置变化
- 窗口大小变化
- 横向滚动条操作
- Invalidate函数执行

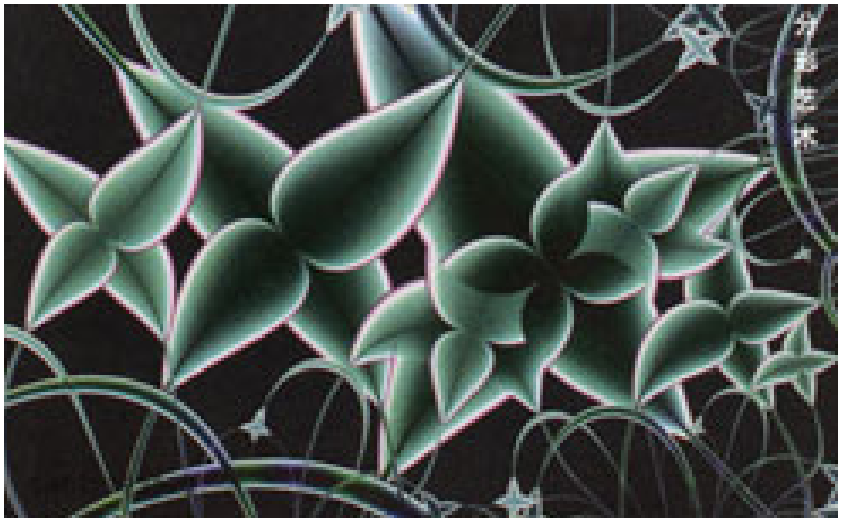
提交

# 有趣的分形理论(1)

例2-10

- 分形理论建立于20世纪70年代，在欧几里得几何学无能为力的领域，分形理论脱颖而出。分形是对没有特征长度、具有一定意义的自相似图形或结构的总称





# 有趣的分形理论(2)

```
CDC* m_pDC;
void CTestView::OnDraw(CDC* pDC)
{ m_pDC=pDC;  CRect rect;  GetClientRect(&rect);
  int iOx=rect.right/2;  int iOy=rect.bottom/2;
  DrawRect(iOx,iOy,(iOx>iOy?iOy:iOx)/3); }
void CTestView::DrawRect(int iX, int iY, int iR)
{ if(iR>0)
  { DrawRect(iX-iR,iY+iR,iR/2);
    DrawRect(iX+iR,iY+iR,iR/2);
    DrawRect(iX-iR,iY-iR,iR/2);
    DrawRect(iX+iR,iY-iR,iR/2);
    m_pDC->Rectangle(iX-iR,iY-iR,iX+iR,iY+iR); } }
```

# 第2次作业

- 编程实现测试程序，满足以下要求:
  - 按下键盘任意键，屏幕显示按键信息
  - 单击鼠标左键，屏幕显示鼠标信息
  - 假设鼠标右键失灵，用Ctrl+鼠标左键代替
  - 自定义WM_MY_MESSAGE消息，带50和100两个参数，由“?”键激活，屏幕显示相应信息
- 编程实现SmallBall程序

# 谢谢大家