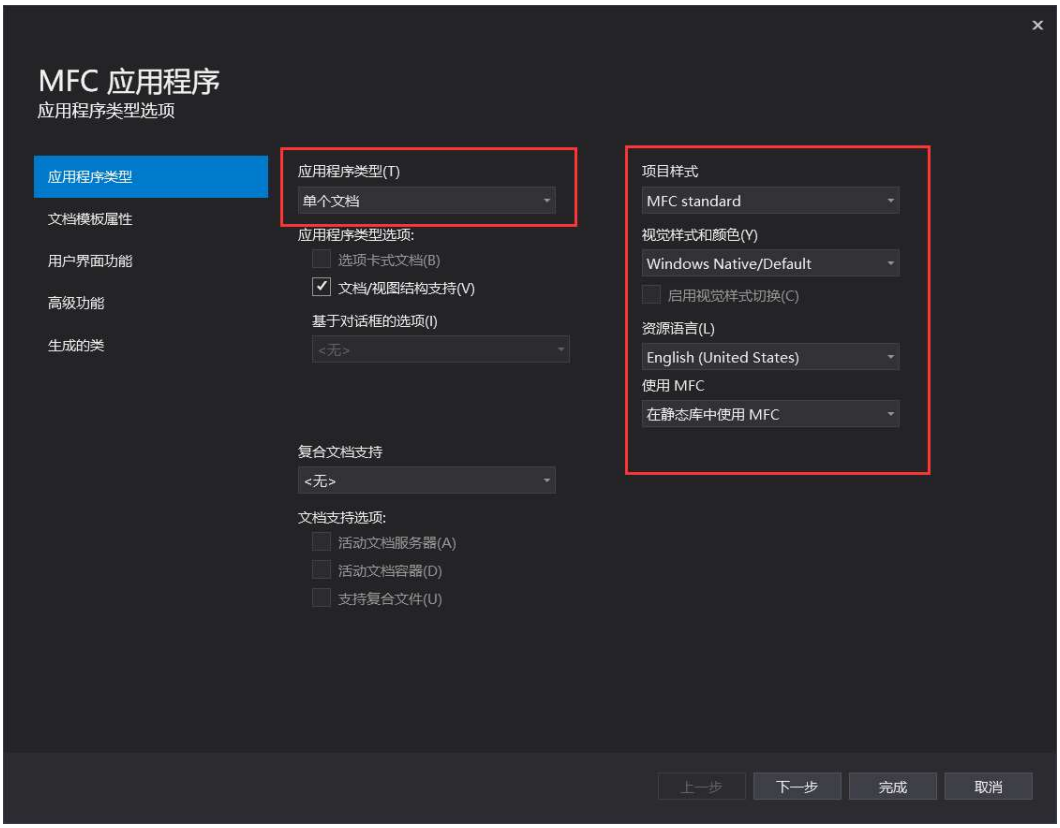
2020/07/03_MFC_第5课_CAD01_直线的绘制以及文件遍历

笔记本: MFC
创建时间: 2020/7/3 星期五 18:55
作者: ileemi
标签: MFC单文档, 双缓冲, 文件遍历

CAD（绘制图形）

MFC 单文档的创建及使用

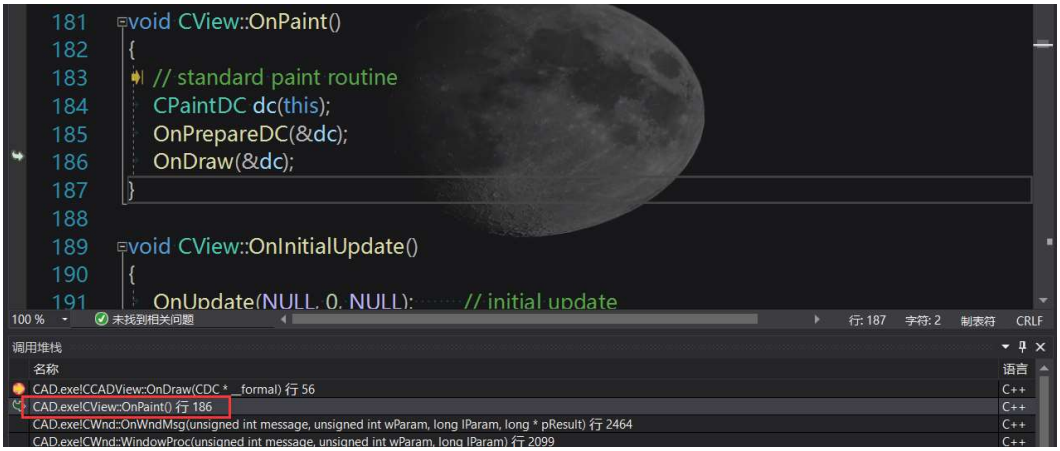


什么时候绘制

自己写的代码需要写在消息响应中，同样的画图也需要在消息响应中，在界面上画图，在窗口上画图需要响应 **WM_PAINT** 消息。绘制的图形绘制在客户区。

MFC单文档结构：
...APP -- 放置应用程序相关的（实例句柄等）
...Doc -- 用于存储数据
...View -- 负责客户区
...MainFrame -- 用于设置窗口的菜单栏，工具栏，状态栏等

在 ...View 中使用虚函数 OnDraw 响应 WM_PAINT 消息，不应该直接去相应 WM_PAIN 消息，OnDraw 来自与父类 OnPaint 的调用，OnPaint 实际上就是 WM_PAINT 的消息响应函数，说白了 OnDraw 就是在 WM_PAINT 里进行的调用。如下图所示：



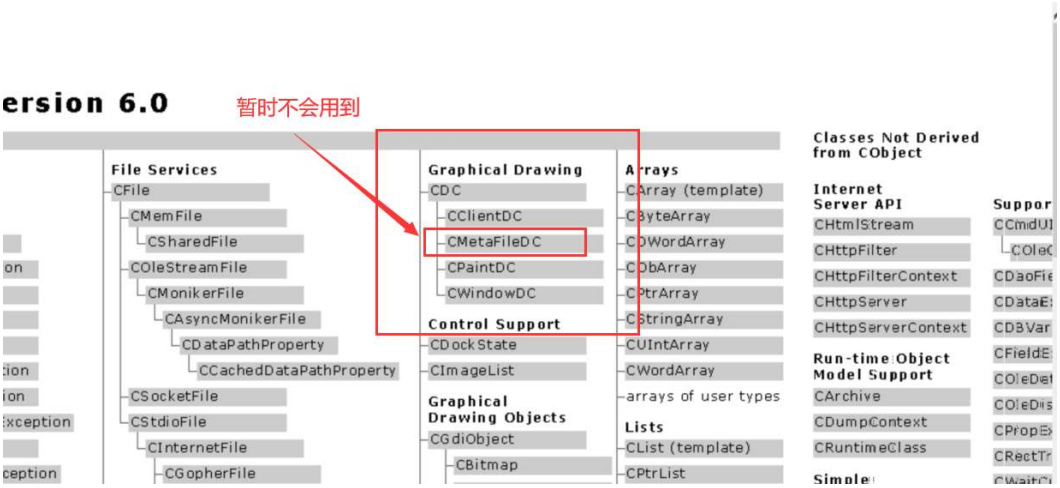
```
18  BEGIN_MESSAGE_MAP(CView, CWnd)
19      ON_WM_PAINT()
20      ON_WM_MOUSEACTIVATE()
21      ON_WM_CREATE()
22      ON_WM_DESTROY()
23
24      // Standard commands for split pane
25      ON_COMMAND_EX(ID_WINDOW_SPLIT, &CView::OnSplitCmd)
423   #define ON_WM_PAINT() \
424     { WM_PAINT, 0, 0, 0, AfxSig_vv, \
425       (AFX_PMSG)(AFX_PMSGW) \
426       (static_cast< void (AFX_MSG_CALL CWnd::*)(void) > ( &ThisClass :: OnPaint)) },
427
```

结论：在OnDraw里进行绘制

在哪里绘制

在客户区进行绘画，将绘画的内容显示在客户区对应的那块屏幕上，由于没有办法直接对屏幕进行操作，会将绘制的图形画到 DC 上，之后系统将 DC 中的内容显示到对应的屏幕上。SDK 中获取 DC 的两种方式，**BeginPaint**，**GetDC**。

MFC 对 DC 进行了封装 -- **CDC**



DC 对应关系：

```
50
51  /*
52                          对应关系
53              SDK          MFC          绘制区域
54        Beginpaint         CPaintDC       无效区
55        GetDC              CClientDC      客户区
56        GetWindowDC        CWindowDC      整个窗口
57      CreateCompatibleDC   CDC            内存
58
59  */
60
```

将屏幕的内容显示到客户区

API -- **BitBlt**

说明：BitBlt函数执行将与矩形像素对应的颜色数据从指定的源设备上下文传输到目标设备上下文的**位块传输。**

简单来说就是，就是将一个DC中的内容拷贝到另一个DC中。
API定义：

Platform SDK: Windows GDI

BitBlt

The **BitBlt** function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

```
BOOL BitBlt(
  HDC hdcDest, // handle to destination DC
  int nXDest,  // x-coord of destination upper-left corner
  int nYDest,  // y-coord of destination upper-left corner
  int nWidth,  // width of destination rectangle
  int nHeight, // height of destination rectangle
  HDC hdcSrc,  // handle to source DC
  int nXSrc,   // x-coordinate of source upper-left corner
  int nYSrc,   // y-coordinate of source upper-left corner
  DWORD dwRop  // raster operation code
);
```

光栅操作　　常用值：SRCCOPY -- 将源矩形直接复制到目标矩形

操作步骤：
OnDraw 回调函数 -- OnPaint，OnPaint 内部通过 CPaintDC 获取DC，CPaintDC 内部又调用了 BeginPaint。所以目标 DC 就是 OnDraw 方法的参数（可以直接使用即可）。

GetClientRect -- 获取客户区的窗口大小，参数为LPRECT（RECT）的一个结构体指针。
GetWindowRect -- 获取屏幕的窗口大小

一般使用MFC封装好的矩形类获取窗口的大小 -- **CRect**

```
62  //TODO:在此处为本机数据添加绘制代码
63
64  //MFC中对矩形进行了封装，推荐使用CRect
65  //获取客户区的大小
66  //RECT rcClient;//客户区的长度
67  CRect rcClient;
68  GetClientRect(rcClient);
```

获取原DC：
GetWindowDC -- 参数为空，获取整个屏幕的DC
GetDC -- 如果该值为NULL, GetDC 将检索整个屏幕的设备上下文
CreateDC -- CreateDC函数使用指定的名称为设备创建设备上下文(DC)，四个参数，第一个参数填写 **字符串的 DISPLAY**，后面三个参数填NULL。

CreateDC 属于 DC 的API，在MFC中会被封装到CDC中。

使用BitBlt拷贝图片，从屏幕到客户区：

```
62  // TODO: 在此处为本机数据添加绘制代码
63
64  // MFC 中对矩形进行了封装，推荐使用CRect
65  // 获取客户区的大小
66  // RECT rcClient; //客户区的长度
67  CRect rcClient;
68  GetClientRect(rcClient);
69
70  // 获取屏幕DC
71  CDC dcScreen;
72  dcScreen.CreateDC(_T("DISPLAY"), NULL, NULL, NULL);
73
74  // 拷贝图片，从屏幕到客户区
75  // 注意使用客户区的DC进行调用
76  pDC->BitBlt(
77      0, 0,              // 拷贝到客户区的左上角位置上
78      rcClient.Width(), rcClient.Height(), // 拷贝大小为客户区的大小
79      &dcScreen,         // 原DC
80      0, 0,              // 从屏幕的左上角开始拷贝
81      SRCCOPY            // 光栅代码，将源矩形直接复制到目标矩形
82  );
83  }
```

之后需要一个定时器，去不断的捕捉屏幕，在 CView 中的初始化函数需要添加虚函数 **OnInitialUpdate** 来进行初始化操作 -- 框架在视图首次附加到文档之后，但在视图最初显示之前调用。

使用 **WM_TIMER** 响应定时器消息，在 **CView** 类向导中进行添加，在对应的方法中使用 **InvalidateRect** 刷新无效区。

```
126 // CCADView 消息处理程序
127 // 初始化操作
128 void CCADView::OnInitialUpdate()
129 {
130     CView::OnInitialUpdate();
131     // TODO: 在此添加专用代码和/或调用基类
132
133     SetTimer(1, 1, NULL);  // 添加计时器，时间间隔为毫秒
134 }
135
136 // 使用 WM_TIMER 消息 响应定时器消息
137 void CCADView::OnTimer(UINT_PTR nIDEvent)
138 {
139     // 不刷新无效区，刷新无效区会闪
140     InvalidateRect(NULL, FALSE);
141
142     CView::OnTimer(nIDEvent);
143 }
```

如何绘制

在画图工具中绘制直线，响应的消息：
鼠标消息：左键按下（**WM_LBUTTONDOWN**），左键弹起（**WM_LBUTTONUP**），鼠标移动（**WM_MOUSEMOVE**）。
绘制直线的API：**MoveTo**（定位直线的起点），**LineTo**（定位直线的终点）

画直线：鼠标按下开始画直线，鼠标左键弹起停止画直线。

获取鼠标消息的API：

SetCapture：当鼠标移除客户区（窗口）外后，仍然接受鼠标消 息。
ReleaseCapture：不在接受客户区（窗口）外的鼠标消息。

在MFC中，上面的两个 API 不需要带参数。
当**鼠标左键按下**的时候，开始接收窗口外的鼠标消息。
当**鼠标右键弹起**的时候，停止接收窗口外的鼠标消息。
代码示例：

```
175  void CPaintLineView::OnLButtonDown(UINT nFlags, CPoint point)
176  {
177  	// 鼠标左键按下，保存要绘制直线的起点坐标
178  	m_ptBegin = point;
179
180  	SetCapture();		// 接收窗口外的鼠标消息
181
182  	// 测试MFC链表中的迭代器的用法
183  #if 0 非活动预处理器块
202  #endif // 0
203
204  #if 0 非活动预处理器块
211  #endif // 0 // 测试pair的使用
212
213  	CView::OnLButtonDown(nFlags, point);
214  }
215
216  // 左键弹起
217  void CPaintLineView::OnLButtonUp(UINT nFlags, CPoint point)
218  {
219  	// 鼠标左键弹起，保存直线的终点位置坐标
220  	m_ptEnd = point;
221  	// 设置无效区，窗口内进行绘制
222  	InvalidateRect(NULL, FALSE);
223
224  	// 鼠标弹起，将绘制的的直线数据保存到链表中
225  	m_listLines.AddHead(pair<CPoint, CPoint>(m_ptBegin, m_ptEnd));
226
227  	// 鼠标左键弹起，不在接收窗口外的鼠标消息
228  	ReleaseCapture();
229
230  	CView::OnLButtonUp(nFlags, point);
231  }
```

直线的保存

只保存一条直线的起点和终点，绘制直线的时候只会显示一条直线。

绘制多条直线：将直线绘制的直线保存起来，之后再遍历全部显示。

鼠标弹起时候，将直线数据存入到链表中。
画图的时候，遍历链表，将链表中的直线全部显示。

定义一个MFC的链表：CList

```
14  private:
15  	CPoint m_ptBegin;		// 直线的起点
16  	CPoint m_ptEnd;			// 直线的终点
17  	CList<pair<CPoint, CPoint>> m_listLnes;		// 保存直线的链表
18  // 特性
19  public:
20  	CPaintLineDoc* GetDocument() const;
```

MFC链表CList

MFC中的链表：
CList -- 内置迭代器，支持遍历
使用示例：

```
CList<int> lstTest;
lstTest.AddHead(1);
lstTest.AddHead(2);
lstTest.AddHead(3);
lstTest.AddHead(5);
lstTest.AddHead(6);

// 获取链表头结点
POSITION pos = lstTest.GetHeadPosition();

// 遍历取出每个结点中存储的数据
while (pos)
{
    int nVal = lstTest.GetNext(pos);      // 获取链表中下一个数据
    CString strFmt;
    strFmt.Format(TEXT("%d"), nVal);
    AfxMessageBox(strFmt);
}
```

pair的使用

CList 一个参数如何存储两个数值?
使用 **Pair（STL）** -- 用于存储一对数据，也只能存储一对数据，两个数据可以是不同类型的数据。其是一个模板。

使用时需要包含头文件：
#include <utility>
using namespace std;

使用示例：存入并取出
xxx.first -- 取出第一个
xxx.second -- 取出第二个

```
// 存储一对不同类型的数据
pair<int, CString> par(999, "Hello World");

// 去除存储的一对数据
int nResult = par.first;
CString strResult = par.second;
```

使用双缓冲绘图，解决画直线闪烁问题

保存直线

直线的保存，需要在鼠标左键弹起的时候进行保存。

代码示例：

```
//左键弹起
void CPaintLineView::OnLButtonUp(UINT nFlags, CPoint point)
{
//鼠标左键弹起，保存直线的终点位置坐标
m_ptEnd = point;
//设置无效区，窗口内进行绘制
InvalidateRect(NULL, FALSE);

//鼠标弹起，将绘制的的直线数据保存到链表中
m_listLines.AddHead(pair<CPoint, CPoint>(m_ptBegin, m_ptEnd));

//鼠标左键弹起，不在接收窗口外的鼠标消息
ReleaseCapture();

CView::OnLButtonUp(nFlags, point);
}
```

绘制的直线都显示

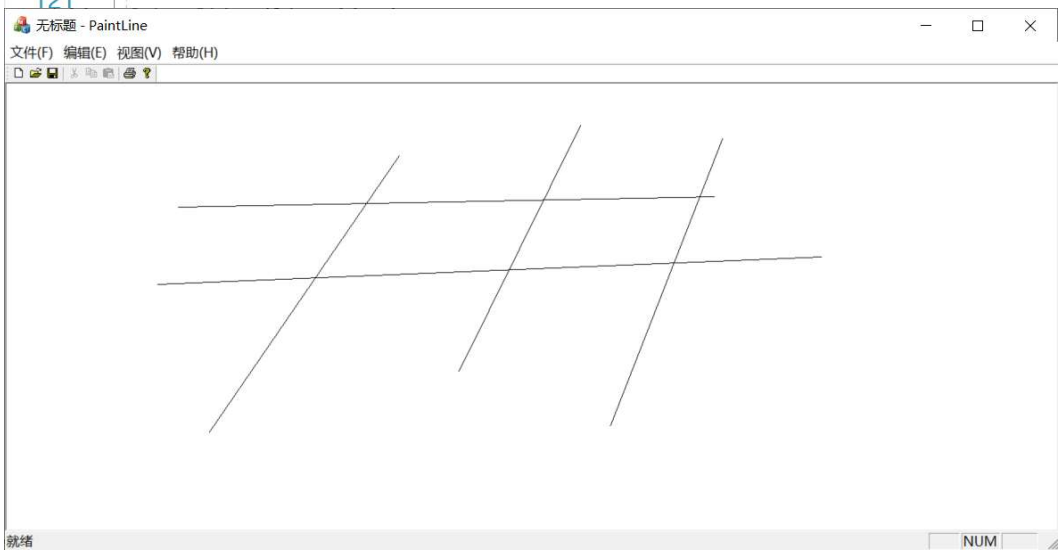
在 OnDraw 内将链表中保存的直线数据都进行绘制，达到绘制的直线都显示的效果。遍历链表即可。

代码示例：

```
//3. 在内存中绘制
POSITION pos = m_listLines.GetHeadPosition();//获取链表头结点
//遍历取出每个结点中存储的数据
while (pos)
{
auto line = m_listLines.GetNext(pos);
//pair<CPoint, CPoint> line = m_listLines.GetNext(pos);

//将获取的数据显示
dcMem.MoveTo(line.first);
dcMem.LineTo(line.second);
}

//根据直线的起点和终点绘制直线
dcMem.MoveTo(m_ptBegin);
dcMem.LineTo(m_ptEnd);
```



双缓冲绘图

绘制直线的时候，不将直线直接绘制在客户区中，应将直线先绘制到内存中，之后从内存中将这条直线贴到客户区中。

双缓冲步骤

1. 创建内存DC（兼容DC）
2. 创建兼容位图 并选入DC -- 位图（计算机绘画是在位图中绘制的，相当于一块画布，虚拟的画板）
3. 在内存中绘制，获取链表中的头结点
4. 将内存中绘制的图片贴到客户区中 pDC->BitBlt();

- 创建兼容DC
  CDC dcMem
  xxx.**CreateCompatibleDC** -- 创建兼容DC
- 获取控件客户区域的大小
  CRect reCLient
  GetClientRect(&reCLient)
- CBitmap bmpMen -- 位图类
  bmpMen.**CreateCompatibleBitmap** -- 创建兼容位图，参数：指定设备的DC，设备位图的宽度，设备位图的高度
- dcMem.**SelectObiect**(bmpMem) -- 将兼容DC和位图DC关联
  dcMem.FillSolidRect(&reClient, RGB(255,255,255)) -- 用纯色填充一块矩形

代码示例：

```
86  //使用双缓冲绘制直线
87  #if 1
88  //1.创建兼容DC（内存DC）
89  CDC dcMem;
90  dcMem.CreateCompatibleDC(pDC); //将客户区的DC当作参数
91
92  //2.创建兼容DC并选入DC
93  CRect rcClient;          //获取客户区的大小
94  GetClientRect(&rcClient);
95
96  CBitmap bmpMem;
97  bmpMem.CreateCompatibleBitmap(pDC, rcClient.Width(), rcClient.Height());
98
99  //将兼容DC和位图DC关联
100 dcMem.SelectObject(&bmpMem);
101
102 //设置客户区的背景为白色
103 dcMem.FillSolidRect(&rcClient, RGB(255, 255, 255));
104
```

```cpp
	//3.在内存中绘制
	POSITION pos = m_listLines.GetHeadPosition();//获取链表头结点
	//遍历取出每个结点中存储的数据
	while (pos)
	{
		auto line = m_listLines.GetNext(pos);
		//pair<CPoint, CPoint> line = m_listLines.GetNext(pos);

		//将获取的数据显示
		dcMem.MoveTo(line.first);
		dcMem.LineTo(line.second);
	}

	//根据直线的起点和终点绘制直线
	dcMem.MoveTo(m_ptBegin);
	dcMem.LineTo(m_ptEnd);

	//4.将内存中绘制的直线贴到客户区中
	pDC->BitBlt(
		0, 0,
		rcClient.Width(), rcClient.Height(),
		&dcMem,
		0, 0,
		SRCCOPY
	);
#endif //0

}
```

设置客户区背景

API -- FillSolidRect

文件遍历

CFileFind -- 文件类

GetLogicalDriveStrings -- 用指定系统中有效驱动器的字符串填充缓冲区，可以获取系统有效的盘符。

代码示例：