# 这一节我们来学习MFC绘图功能，开发工具vs2019 vc++,虽然我们使用新版本的开发工具，但是帮助文档确实vc++6的msdn好用，所以我也安装了vc++6的msdn，注意新增加的MFC类需要使用新版本的帮助文档。

## 1、新建一个MFC项目，取名drawdemo，单文档应用程序，注意用vs2019开发mfc程序，有一个高级选项，里面有高级窗口选项，我们刚刚学习用不到，需要取消勾选，等以后有需要再勾选

|  |
| --- |
|  |

### 注意：vs2019生成的文件的文件名和vc++6不太一样，它第二个字母是小写的。

## 2.先编译运行一下，没有问题，这是我们最经典的mfc应用程序

|  |
| --- |
|  |

## 下面开始学习绘图API

## 3.我们先来尝试给CMainFrame类添加鼠标消息响应，选中CMainFrame类，点击项目-》类向导-》消息，界面如下

|  |
| --- |
|  |

## 4.选中WM\_LBUTTONDOWN消息，点击添加处理程序，vs就会帮你创建一个OnLButtonDown函数，注意一定要选中CMainFrame类

|  |
| --- |
|  |

## 5.点击应用然后点击确定，然后进入消息处理函数

|  |
| --- |
|  |

## 6.我们在这个MainFrame的消息处理函数中添加如下代码

|  |
| --- |
|  |

## 7.我们期望当我们的程序运行的时候，点击鼠标会弹出一个消息框，显示当前鼠标的坐标，但是当我们编译运行程序，发现根本没有任何反应，这是为什么？你是因为mfc这种文档视图架构中，视图类始终是覆盖在框架类上面，所以鼠标其实无法点击到框架类的客户区，所以自然就没有响应了。所以鼠标显示还有交由view类来处理。

## 8.然后我们用同样的方法给view类添加一个鼠标左键按下的信息处理函数

|  |
| --- |
|  |

## 9.然后，我们添加相同的代码，

|  |
| --- |
|  |

## 10.编译运行程序，发现能够响应鼠标左键点击

|  |
| --- |
|  |

## 11.这个坑一定要注意，不要让MainFrame来响应鼠标消息，我们把MainFrame的鼠标左键响应函数删除

|  |
| --- |
|  |

### 关于MFC消息映射的原理，可以参考：[图解MFC消息映射](图解MFC基本框架（深入消息映射机制）.docx)

## 下面我们来完成简单绘图功能

## 12.当鼠标左键按下时，我们需要把这个起点保存下来，先给view类添加一个成员变量m\_ptOrg,类型是CPoint，设置为私有

|  |
| --- |
|  |

## 然后点击确定，点击应用，再点击确定

## 13.然后在view类的构造函数中把这个成员变量初始化为0

|  |
| --- |
|  |

## 14.在view类的OnLButtonDown函数里面用m\_ptOrg来保存应用程序传递进来的点的坐标

|  |
| --- |
|  |

## 15，然后我们需要给view类添加鼠标左键弹起的消息处理函数在这里获取第二个点的坐标

|  |
| --- |
|  |
|  |

## 16.我们可以在这个消息响应函数里面实现绘图功能，我们先使用win32 sdk的方法，添加下面的代码

|  |
| --- |
|  |

### 然后编译运行程序，发现地区是可以绘图了

|  |
| --- |
|  |

## 17.然而，MFC为我们提供了绘图用的DC类：CDC，它封装了所有与作图相关的win32平台sdk的绘图函数。我们可以在帮助文档里面查看一下这个类，发现它有很多成员方法

|  |
| --- |
|  |

### 他有一个m\_hDC成员变量，可以用来保存hdc句柄

|  |
| --- |
|  |

## 18.我们把上面的代码注释了，现在使用CDC来实现上面的功能，注意在MFC中，GetDC是CWnd类的成员函数不是CDC的成员函数，这一点需要注意，绘制完成后也是调用CWnd类的成员函数ReleaseDC来释放设备上下文对象。

|  |
| --- |
|  |

### 编译运行程序，发现还是可以绘图的

|  |
| --- |
|  |

## 19.此外，我们还有一个叫做CClientDC的类，他是CDC的子类，我们也可以使用它来绘图？

### 那么使用这个子类有什么好处呢？CClientDC是CDC的衍生类，产生对应于Windows客户区的对象，在构造时调用CWnd::GetDC，在析构时调用CWnd::ReleaseDC。他在构造对象的时候需要一个CWnd类的指针，我们把代码改为这样子：

|  |
| --- |
|  |

### 我们发现，程序也是可以正常绘图的

|  |
| --- |
|  |

### 如果你想给CClientDC绑定一个MainFrame，就可以在view类中使用CWnd：：GetParent()方法来获取他的指针

|  |
| --- |
|  |

### 不过，对于作图来说没有这个必要

## 20.还有一个CWindowDC类，也是CDC的子类，它在构建对象的时候会自动调用GetWindowDC函数，在析构的时候会自动调用ReleaseDC函数，可访问CWnd的整个屏幕，包括客户区和非客户区。而上面的两个DC只能够访问客户区。我们修改一下代码

|  |
| --- |
|  |

### 编译运行程序，此时你会发现，我们可以把直线绘制到框架窗口的非客户区上

|  |
| --- |
|  |

## 21，此外MFC还提供了一个GetDesktopWindow函数可以获取桌面窗口的指针然后我们可以把它传递给CWindowDC来构造一个能够在桌面上绘制的dc。我们修改一下代码，

|  |
| --- |
|  |

### 运行程序，你会发现，现在我们可以把直线绘制在我们的应用程序的外面，因为我们获取了整个桌面窗口

|  |
| --- |
|  |

## 22.上面我们作图使用的画笔颜色都是黑色的这是系统默认颜色，我们其实可以修改颜色，方法是创建一个其他颜色的画笔，然后把他选择进设备描述表，我们先把dc恢复成CClientDC的对象，并且和view窗口关联，因为默认作图都是在view类窗口。然后我们创建一个CPen对象，实线的，线宽为1，颜色是紫色。创建好后，需要用dc把笔选进设备描述表，同时需要用一个指针保存它的返回值，就是原来的画笔，当绘制结束后需要恢复原来的画笔

|  |
| --- |
|  |

### 运行程序，效果如下

|  |
| --- |
|  |

## 23.现在我们来用一下画刷，把上面的代码注释了，然后输入下面的代码

|  |
| --- |
|  |

### 运行程序，可以看到这样子使用效果跟笔没有两样

|  |
| --- |
|  |

## 24，其实使用画刷后我们可以使用fillRect，我们修改一下代码，如下

|  |
| --- |
|  |

### 还可以简化代码

|  |
| --- |
|  |

### 这个有点意思

|  |
| --- |
|  |

## 25.有时候我们需要图像画刷，我们可以这样子来创建，先创建一个CBitmap对象，然后有这个对象的指针来构建CBrush对象：CBrush( CBitmap\* pBitmap );我们先把一幅图片改为bmp格式，然后拷贝到源文件所在目录，这里是qg.bmp

|  |
| --- |
|  |

## 26.然后把他添加到资源中，方法是在资源过滤器文件夹上面点击右键-》添加-》资源

|  |
| --- |
|  |

### 会打开一个对话框，如图

|  |
| --- |
|  |

### 选择Bitmap，然后点击导入按钮，选择我们刚刚的qg.bmp图片，点击打开，如图

|  |
| --- |
|  |

### 这样子就可以把位图导入，导入后如果是第一个导入的位图，他的id是IDB\_BITMAP1，我们可以创建resource.rc文件来查看。

|  |
| --- |
|  |

## 27.创建CBitmap对象，然后调用它的LoadImage方法来加载我们的位图资源，然后用它来创建位图画刷对象，然后再使用CClientDC对象的FillRect方法来绘制矩形

|  |
| --- |
|  |

### 运行程序，效果如下，还蛮酷的

|  |
| --- |
|  |

# 注意：关于vs2019的新发现，vs2019的mfc项目默认是没有类视图的，但是我们可以点击视图-》类视图打开它，然后你依然可以在这个类上面点击右键-》添加-》选择添加成员函数或者成员变量

|  |
| --- |
|  |

### 也有类向导选项

## 28.先把位图画刷的代码注释了，我们来学习透明画刷，我们先调用dc的Rectangle方法绘制一些矩形，

|  |
| --- |
|  |

### 效果是这样子的

|  |
| --- |
|  |

## 29.有些时候我们需要能够看到下面的图形，这时候就得使用透明画刷，但是CBrush类并没有给我们提供创建透明画刷的构造函数，我们需要使用系统预定义画刷，这个需要使用GetStockObject函数来获取HBRUSH hbrush句柄，然后调用CBrush::FromHandle(hbrush)来得到一个CBrush的指针。注意：这是一个静态方法，需要用类名::方法的形式来调用。代码如下：

|  |
| --- |
|  |

### 注意：类实例变量可以调用类的静态方法，但是类本身不能够调用类对象方法，也就是非静态方法，类名本身只能够调用静态方法。

### 尝试直接用类名来定义非静态方法会报下面的错误

|  |
| --- |
|  |

### 类的静态方法不能够访问类的非静态成员，如果你你尝试在静态方法里面访问非静态成员就会报错，因为这些成员需要创建类实例对象的时候才会分配空间。

|  |
| --- |
|  |

### 记住：类的实例对象可以访问类的静态方法和非静态方法，可以使用类的静态变量和非静态变量，类的非静态方里面可以调用类的静态方法，但是类的静态方法只能够使用类的非静态变量也不能够调用类的非静态方法

### 但是静态成员变量需要在定义的时候初始化，如果没有初始化，会发生链接错误，这样的代码是有问题的

|  |
| --- |
|  |

### 还有一个注意事项，类的静态成员变量需要放在类的外面初始化，初始化时需要使用类的作用域操作符

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 30.虽然我们的程序能够绘制图形，但是我们的线条不是连续的，而且没有平滑是效果，如果需要这种效果，需要添加对鼠标移动消息的响应函数，我们为view类添加一个对WM\_MOUSEMOVE消息的处理函数，然后点击应用

|  |
| --- |
|  |

## 31.在鼠标移动处理函数里面我们需要先判断，只有用户按下鼠标左键并且移动我们才作图，所以我们需要给view类创建一个保存鼠标左键是否按下的布尔型变量

|  |
| --- |
|  |
|  |
|  |

## 32.然后我们先在view类的构造函数里面把它初始化为FALSE，

|  |
| --- |
|  |

### 然后在鼠标左键按下的处理函数中把它设置为true,

|  |
| --- |
|  |

### 在鼠标左键弹起消息处理函数中把它设置为false，注意把绘制代码注释了，因为现在是在鼠标移动函数里面作图

|  |
| --- |
|  |

## 33.在view类的鼠标移动处理函数里面先判断m\_bDraw是否为真，如果是，才绘图，这里和鼠标左键弹起的处理函数有一点点不一样，因为是绘制连续的线段，你需要把第一段的终点作为第二段的起点。代码如下

|  |
| --- |
|  |

### 发现现在可以绘制连续的线段甚至是曲线了，因为曲线其实也可以看作是有许多很短的直线段构成的

|  |
| --- |
|  |

## 34.当然我们也可以修改画笔的颜色，代码如下：

|  |
| --- |
|  |

## 35.假如我们需要绘制扇形，我们需要添加一个成员变量来保存终点位置

|  |
| --- |
|  |

## 36.点击确定，然后在view类的构造函数之中把它初始化为0.

|  |
| --- |
|  |

## 37.修改鼠标左键按下的处理函数的代码，如下

|  |
| --- |
|  |

## 38.修改鼠标移动的处理函数的代码如下

|  |
| --- |
|  |

### 效果如下：

|  |
| --- |
|  |

## 39，如果你还需要它带弧线，需要修改代码如下

|  |
| --- |
|  |

### 效果如下：

|  |
| --- |
|  |

## 40.我们可以学习一个API，CDC::SetROP2，他的作用是设置绘图模式，它只有一个参数nDrawMode可以有以下取值

|  |
| --- |
| R2\_BLACK 像素始终为黑色。  R2\_WHITE 像素始终为白色。  R2\_NOP 像素保持不变。  R2\_NOT 像素为屏幕颜色的反色（可以自动擦除上次所绘图形）。  R2\_COPYPEN 像素为笔的颜色。  R2\_NOTCOPYPEN 像素为笔颜色的反色  R2\_MERGEPENNOT 像素为笔颜色和屏幕颜色反色进行或运算的组合色. (final pixel = (NOT screen pixel) OR pen).  R2\_MASKPENNOT 像素为笔颜色颜色和屏幕颜色的反色进行或运算的组合色(final pixel = (NOT screen pixel) AND pen).  R2\_MERGENOTPEN   像素为笔颜色反色或者和屏幕颜色进行或运算的组合色。(final pixel = (NOT pen) OR screen pixel).  R2\_MASKNOTPEN  像素为笔颜色反色与屏幕颜色的组合色。(final pixel = (NOT pen) AND screen pixel).·   R2\_MERGEPEN  像素为笔颜色或者屏幕颜色的组合色。(final pixel = pen OR screen pixel).  · R2\_NOTMERGEPEN  R2\_MERGEPEN的反色。  · R2\_MASKPEN  像素为笔颜色与屏幕颜色的组合色。(final pixel = pen AND screen pixel).  · R2\_NOTMASKPEN  R2\_MASKPEN的反色。  · R2\_XORPEN   像素为笔颜色与和屏幕颜色异或的颜色。(final pixel = pen XOR screen pixel).  · R2\_NOTXORPEN  R2\_XORPEN的反色 |

## 41.这里我们尝试一下R2\_MERGENOTPEN

|  |
| --- |
|  |

## 42.运行程序，你会发现，当我们移动鼠标，什么都没有，这是为什么？这个模式就是这样子的，在这里可能不适合。我们选择另外一个模式：R2\_MASKNOTPEN

|  |
| --- |
|  |

### 运行程序，效果如下

|  |
| --- |
|  |

# 这个程序有一个问题，就是当窗口大小改变后图形就会消失，这是因为我们没有在OnDraw函数里面添加重绘代码。我们以后会解决它