十、Windows API编程-消息驱动

外行小明学编程，开始时学的是为win32编程。当他看到一个程序窗口弹出时，脑海中涌出无数个问号。这里仅列举若干重要的问题，来看看小明是如何学会Windows编程的。

1、用win32 API创建一个窗口应用程序

1.1 安装IDE

首先需要安装集成开发环境Visual Studio 2019。可按照以下网文的指示下载安装：

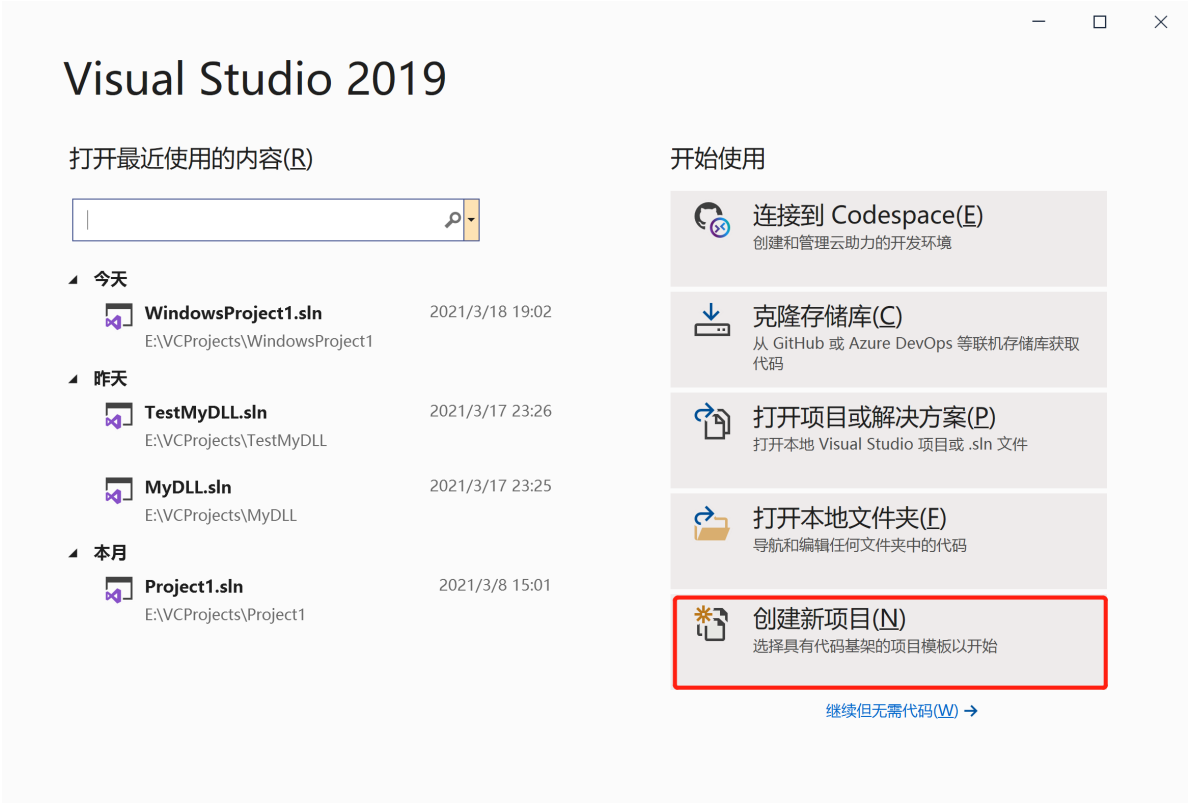
https://mp.weixin.qq.com/s/dHjB6LTZHlV2Ww7KYqnWQw 。

压缩包加压密码：rjaz

注意：安装成功后，VS IDE应用程序位于"Microsoft Visual Studio\2019\Community\Common7\IDE\devenv"。

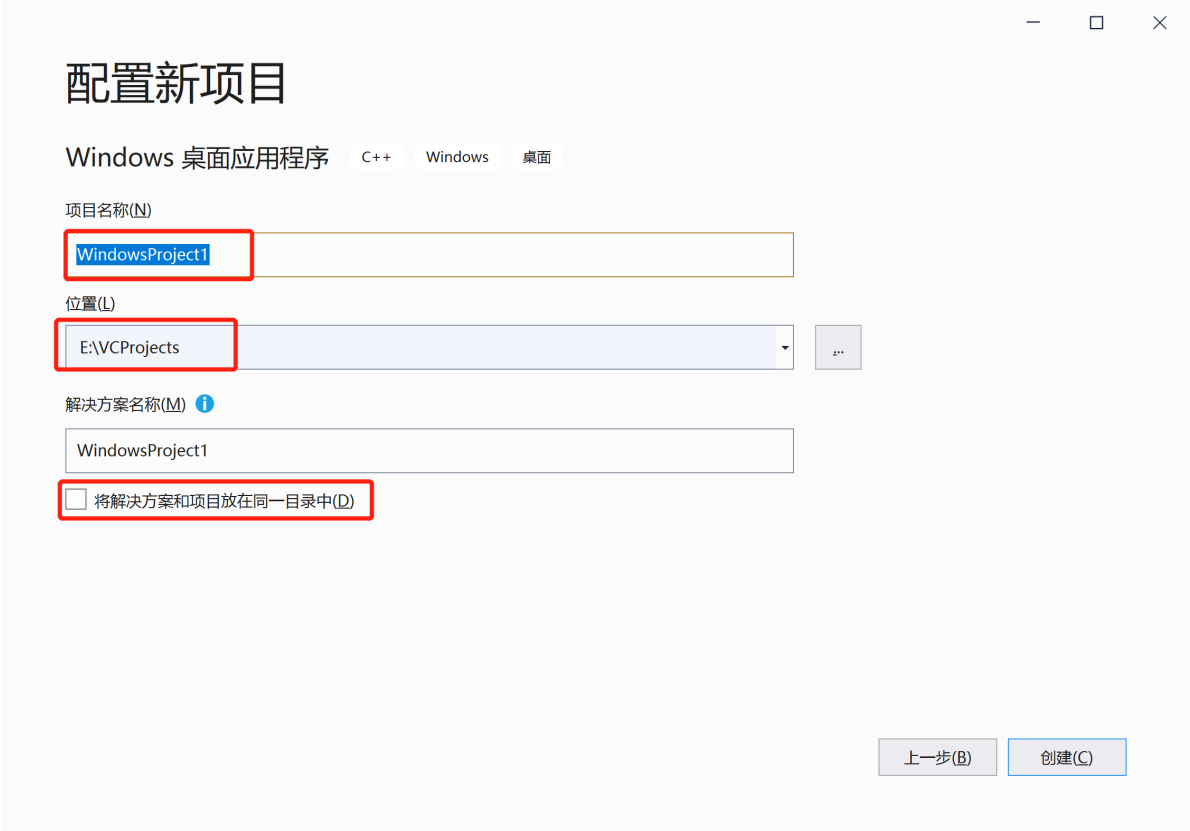
1.2 创建第一个Windows窗口应用程序

启动Visual Studio 2019，开始界面如下图：



点击“创建新项目”按钮，在下面界面中选择项目类型：

先在右上角的下拉菜单中选择“桌面”，然后在筛选出来的所有桌面应用程序列表中，选择带有“C++”标签的“Windows桌面应用程序”，然后点击“下一步”按钮，进入“配置新项目”窗口：



在其中输入“项目名称”，然后选择自己工程的存放目录，“解决方案名称”缺省地与“项目名称”保持一致，注意不要勾选“将解决方案和项目放在同一目录中”，然后点击“创建”按钮，即可完成项目创建。

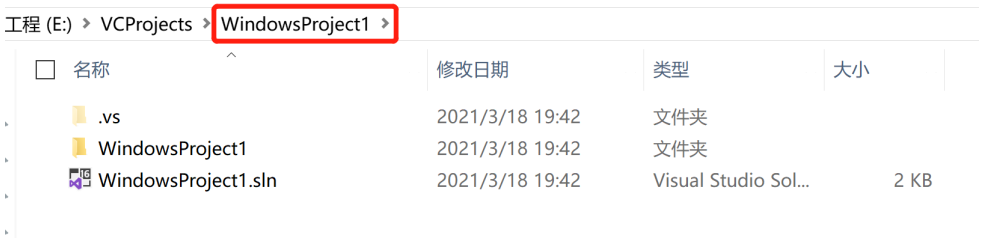
1.3 VS中的解决方案与项目

在VS中，**“解决方案”与“项目”是什么关系？**在 Visual Studio 中，可以将工作组织为**项目和解决方案。**

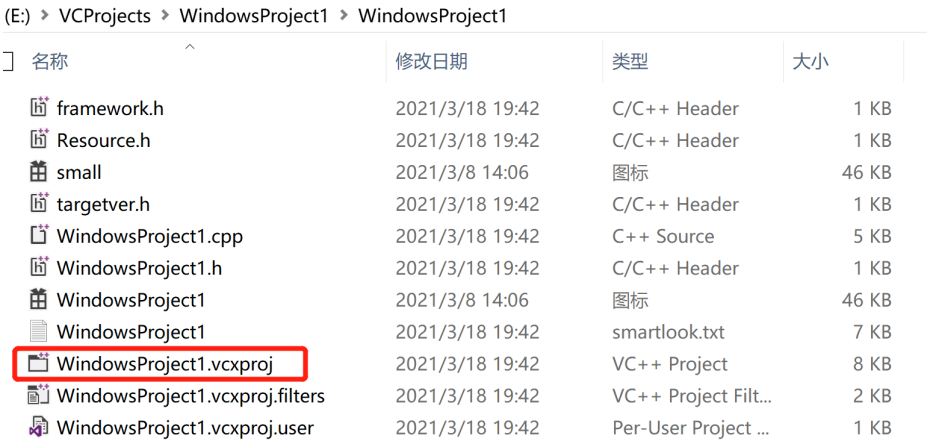
一个解决方案可以包含多个项目，例如，一个 DLL 和一个引用该 DLL 的可执行文件。解决方案只是一个容器，用于包含一个或多个相关项目，以及生成信息、Visual Studio 窗口设置和不与特定项目关联的任何杂项文件。解决方案由格式唯一的文本文件（**扩展名 .sln**）描述；不应对其进行手动编辑 。

项目是一组要编译到单个程序集(在某些情况下，是单个模块)中的所有源代码文件和资源。例如，项目可以是类库，或Windows GUI应用程序。项目文件的扩展名代表了不同的项目类型，比如C++项目是.vcxproj，C# 项目 是.csproj，数据库项目 是.dbproj，等等。

我们上面刚创建了新项目“WindowsProject1”，查看它所在的文件夹：



在"WindowsProject1"解决方案文件夹下，存在着一个同名的项目文件夹，以及一个"WindowsProject1.sln"解决方案文件。点击进入WindowsProject1项目目录：

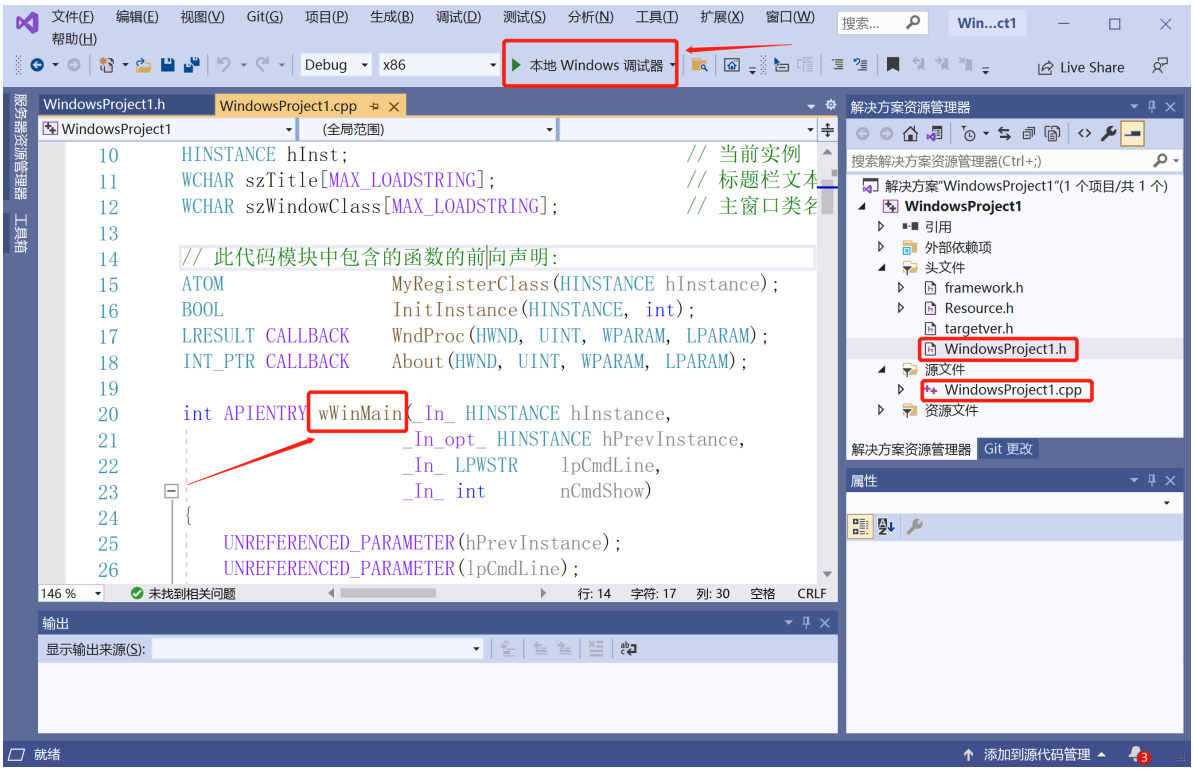


其中的"WindowsProject1.vcxproj"就是项目文件，是一个XML文件。

"WindowsProject1.h"和"WindowsProject1.cpp"是项目主类的头文件和源文件。

1.4 Windows初始工程

VS创建一个C++窗口项目后，进入新项目界面如下：



初始工程自动给了主类WindowsProject1，在其源文件中可以找到其主函数wWinMain。

点击工具栏中的"本地Windows调试器"，可调试运行程序。

如果点击【调试】【开始执行（不调试）】，就是正常执行程序。

2、程序的窗口是从哪里来的?

小明运行初始工程，呆呆地看着弹出来的窗口："这个窗口是从哪里来的？"

回头再看示例程序的源代码，查看主函数WinMain，抛开奇形怪状的符号和代码，小明发现创建一个Windows窗口只需要4步，分别是：

1. 注册窗口；
2. 创建窗口；
3. 显示窗口；

4. 更新窗口。

并且，小明发现完成以上4步的办法是调用了4个奇怪的函数：

1. RegisterClass
2. CreateWindow
3. ShowWindow
4. UpdateWindow

这4个函数奇怪之处在于找不到它们的定义，好像凭空就出现了。小明经过调查研究发现，这些函数就是win32操作系统对应用程序提供的编程接口，一些C语言函数，也称API函数。

小明刨根问底继续追问：win32操作系统是如何向应用程序提供编程接口的？

经过调查发现，是win32操作系统中有一个名叫user的模块，位于C:\Windows\System32\user32.dll。注：以dll为后缀的文件叫做"**动态链接库**"。在这个动态链接库中，封存着一批预先写好的C函数，能够创建并管理窗口、菜单、对话框、控件等GUI元素，并且动态链接库向外部导出了这些函数，供应用程序调用。

调查到这里，小明感到心里非常地踏实，总算弄明白了窗口的来源。进一步调查得知，在win32的前身DOS系统时，有大神就用C语言编写出了窗口，代码非常非常复杂。于是，小明很感谢操作系统提供的帮助，如果没有操作系统提供的API函数，自己要创建出一个窗口来，不知道需要花费多大的代价。

由此小明也得到了一个提示：那些各种应用程序中最常见的、共同的代码，比如创建窗口的代码，可以预先找高手写好，并且可以被调用。其他程序员在开发中若遇到那些常见的功能，就可以调用高手写好的代码。在这里，是微软公司预先把创建窗口的代码直接"沉淀"到了操作系统中，成了操作系统的一部分，供在操作系统之上的应用程序调用。

3、窗口是怎么知道用户点鼠标的?

解开了窗口来源之谜后，小明拿鼠标点一下窗口中的菜单按钮，于是窗口中弹出了一个菜单，小明又陷入了疑惑："窗口是怎么知道用户的鼠标点击并作出响应的？"

于是小明又去研究源代码，发现了线索，一个看起来跟消息有关的API函数GetMessage，被放在一个While循环的条件中，如下：

```
while (GetMessage(&msg, hwnd, NULL, 0)){

    TranslateMessage(&msg);   //翻译消息

    DispatchMessage(&msg);    //派发消息

}
```

首先看这个GetMessage函数的定义：

```
BOOL GetMessage( LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax );
```

查阅win32 API文档以及各种讲解文章发现，此函数的功能就是获取窗口上发生的消息，比如鼠标点击、键盘按下等，这里的窗口由第二个参数传入的、前面已经创建注册好的。得到的消息被填写进第一个参数，这个参数传进去的是地址，所以可以在其中写东西。第三第四个参数先不用管。

在GetMessage()获取到消息是虚拟键消息，需要进行转换处理，调用TranslateMessage(MSG *lpMsg)，这个函数的功能是将虚拟键消息转换成字符消息，通俗说就是翻译消息。

TranslateMessage()之后，需要对消息进行派发，将每个消息发送到相应的需要到达的目标窗口或控件，派发函数：

```
LRESULT DispatchMessage(MSG *lpMsg);
```

在这里，小明又产生了一个疑问：**这里的while循环是如何保持继续的?**

如果这个循环结束，这个WinMain主函数就会执行到最后而导致程序结束。一个窗口应用程序应该一直等着用户来操作，直到用户关闭退出这个程序。

GetMessage的返回值为布尔型变量，API文档指出：

当收到WM_QUIT消息时，GetMessage返回false，循环停止，同时应用程序终止。

当受到其它类型的消息时，GetMessage返回true，执行while循环内的逻辑（翻译、派发）。

当没收到任何消息时，会发生什么事情呢?

小明经过调查发现，`GetMessage`是个**阻塞式函数**，意思是：假如没有收到消息，windows会让该函数在内部等待（具体等待方式为如果没有消息，windows就没收该线程的时间片，直至有消息）。

4、消息是从哪里来的?

`GetMessage`获得用户消息，那么这些消息又是从哪里来的？小明又陷入了困惑。

经过调查，小明发现这是由Windows操作系统提供的功能，消息机制如下：

首先要知道的是，Windows操作系统在内部创建了一个"**系统消息队列**"，并且为每个窗口应用程序都维护着一个**应用程序消息队列。**

当用户运行一个应用程序，通过对鼠标的点击或键盘按键，产生一些特定事件。由于Windows一直监控着I/O设备，该事件首先会被翻译成消息，由系统捕获，存放于**系统消息队列。**

经分析，Windows知道该消息应由那个应用程序处理，则拷贝到相应的**应用程序消息队列。**

由于应用程序的消息循环不断检索自身的消息队列，当发现应用程序消息队列里有消息，就用GetMessage()取出消息，封装成Msg()结构。如果该消息是由键盘按键产生的，用TranslateMessage()翻译为WM_CHAR消息，否则，用DisPatchMessage()将取出的消息分发到相应的应用程序窗口，交由窗口处理程序处理。

这么一看，小明感叹：**Windows操作系统为应用程序运行默默地做了不少事情啊!**

5、如何处理用户消息?

当DispatchMessage将消息派发出去了，发给谁呢？小明发现，DispatchMessage派发出去的消息又交给了操作系统，然后由操作系统调用了名为`WndProc`的回调函数（CALLBACK）。这个函数也叫窗口过程，专门处理窗口的消息。

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

为什么操作系统就知道把消息给这个函数呢？因为在注册窗口之前，我们已经把这个函数的名字嵌入到描述窗口属性的结构体中了，也就是通过向操作系统注册这个步骤，告诉操作系统：一旦这个窗口上产生了用户消息，请把它传到这个回调函数处理。

```
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_WINDOWSPROJECT1));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_WINDOWSPROJECT1);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
```

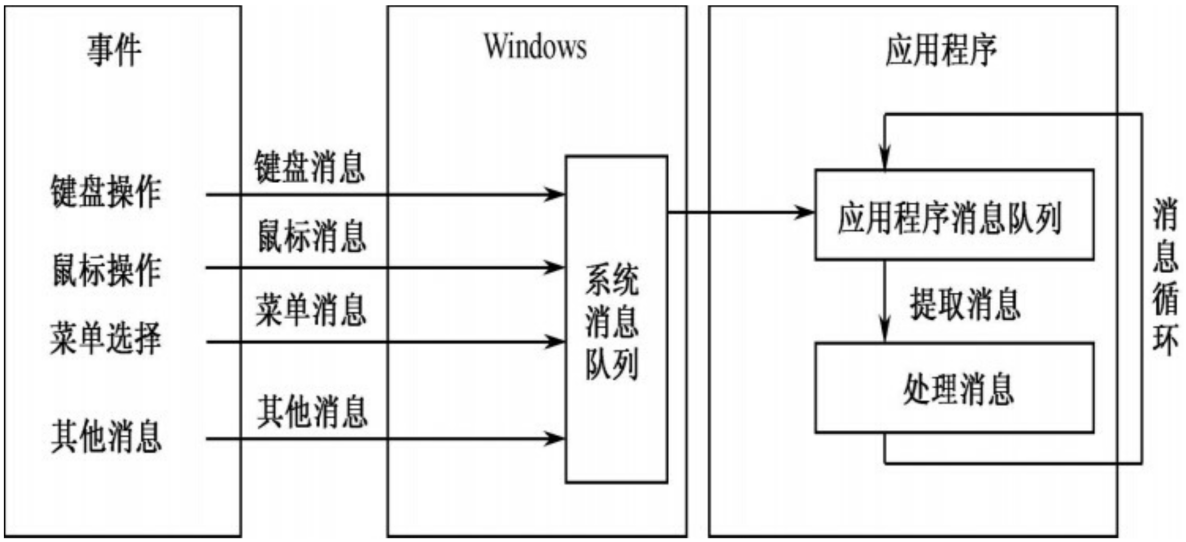
小明发现，这里又出了一个新名词“**回调函数**”！很新奇。原来应用程序可以调用操作系统的API函数，现在操作系统也能调用应用程序中的函数了，所以叫回调。

在这`WndProc`函数中，如下图所示是一个switch/case结构，我们程序员可以通过重载该函数，处理我们“感兴趣”的消息。对于不感兴趣的消息，则由系统默认的窗口过程处理程序做出处理。

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_COMMAND:
        { ... }
        break;
    case WM_PAINT:
        { ... }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

6、Window消息机制

将上面的关于消息接收、转发、处理的知识点总结一下，形成下图：



这张图很好地解释了Windows消息机制的运行原理：

运行程序->事件操作引发消息->消息先存在系统消息队列->再存入到应用程序消息队列->用消息循环提取消息->处理消息->再返回消息队列...

7、实验：Win32 鼠标消息处理

7.1 鼠标消息

鼠标消息是系统队列消息，鼠标响应事件同样在窗体程序的消息回调函数`LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)`作为事件之一处理。鼠标消息如下：

鼠标消息	含义
WM_LBUTTONDOWN	鼠标左键按下
WM_LBUTTONUP	鼠标左键抬起
WM_RBUTTONDOWN	鼠标右键按下
WM_RBUTTONUP	鼠标右键抬起
WM_MOUSEMOVE	鼠标移动消息
WM_LBUTTONDBCLK	鼠标左键双击
WM_RBUTTONDBCLK	鼠标右键双击
WM_MOUSEWHEEL	鼠标滚轮消息

7.2 消息参数

以上消息附加参数（WPARAM和LPARAM）说明：

1. 以上消息中的**LPARAM**参数均表示当前鼠标的位置，其中，**LOWORD**表示X坐标，**HIWORD**表示Y坐标
2. 以上消息除了WM_MOUSEWHEEL消息之外，其他消息的**WPARAM**均表示其他按键的状态，如ALT、SHIFT等
3. WM_MOUSEWHEEL消息的WPARAM的LOWORD表示其他按键的状态，如ALT、SHIFT等。HIWORD表示滚轮的偏移量，这个值是120的倍数，通过正负值表示滚动方向（正：向前滚动，负

向后滚动）。

7.3 鼠标消息处理

软件需求：写一个窗体程序，如果出现点击，显示出鼠标点击点在窗体的坐标。

分析：我们将用两种方法显示坐标：1、在消息框中显示；2、在窗口客户区图形显示。显示坐标功能，涉及到将int型转换成不同类型的字符串；还涉及到多个字符串的拼接。

代码：在Win32Mouse.cpp窗体项目主文件的窗口消息处理回调函数WndProc中，添加鼠标左右键点击处理的代码，如下所示：

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC         hdc;         //设备环境句柄
    switch (message)
  {
  case WM_LBUTTONDOWN:     //鼠标左键消息。用消息框显示信息
  {
    int x = LOWORD(lParam);//鼠标的横坐标
    int y = HIWORD(lParam);//鼠标的纵坐标
    //把整数x转换成字符串wchar_t
    wchar_t sx[80];
    wsprintf(sx, L"%d", x);
    wprintf(L"[%ls]\n", sx);
    //把整数y转换成字符串wchar_t
    wchar_t sy[80];
    wsprintf(sy, L"%d", x);
    wprintf(L"[%ls]\n", sy);
    //将多个wchar_t拼接起来
    wchar_t info[200];
    wcscpy_s(info, L"坐标：(");
    wcscat_s(info, sx);
    wcscat_s(info, L",");
    wcscat_s(info, sy);
    wcscat_s(info, L")");
    wprintf(L"%ls\n", info);
    //弹出消息框，在其中显示鼠标右键点击处的坐标
    MessageBox(NULL, info, L"鼠标左键点击", MB_OK);
    }
    break;
  case WM_RBUTTONDOWN:     //鼠标右键消息。用图形显示信息
  {
    hdc = GetWindowDC(hWnd);    //返回hWnd参数所指定的窗口的设备环境。
    InvalidateRect(hWnd, NULL, true);  //使得窗口客户区域无效，并擦除客户区的图形
    //UpdateWindow 函数绕过应用程序的消息队列，直接发送 WM_PAINT 消息给指定窗口的窗口过程。
    UpdateWindow(hWnd);
    int x = LOWORD(lParam);//鼠标的横坐标
    int y = HIWORD(lParam);//鼠标的纵坐标
    TCHAR sx[100],sy[100];
    _itow_s(x,sx,10);       //将整型x转换成TCHAR型变量sx
    _itow_s(y, sy, 10);     //将整型y转换成TCHAR型变量sy
```

```
    //将"鼠标右键:("、sx、","、sy、")"几个字符串拼接起来
    TCHAR msg[] = _T("鼠标右键:(");
    TCHAR sep[] = _T(",");
    TCHAR tail[] = _T(")");
    //字符串长度
    int length = lstrlen(sx) + lstrlen(sy) + lstrlen(msg)+ lstrlen(sep)+
lstrlen(tail);
    TCHAR *info = new TCHAR[length];
    info[0] = _T('\0');
    //下面将几个字符串拼接起来
    lstrcat(info, msg);
    lstrcat(info, sx);
    lstrcat(info, sep);
    lstrcat(info, sy);
    lstrcat(info, tail);
    //将字符串在窗口客户区以图形形式显示
    TextOut(hdc, x, y, info, _tcslen(info));
    ReleaseDC(hWnd, hdc);     //函数释放设备环境供其他应用程序使用

}
        break;
    ........
  ........
```

8、实验：win32键盘消息处理

8.1 键盘消息

因为大多数PC只有一个键盘，所以所有运行中的Windows程序都必须公用它。Windows操作系统负责将击键消息

传送给既有输入焦点的那个应用程序的消息队列中，有输入焦点的应用程序的标题条总是高亮显示。

键盘消息如下所示：

键盘消息	含义
WM_KEYDOWN	按键被按下时产生
WM_KEYUP	按键被放开时产生
WM_SYSKEYDOWN	系统键按下时产生 比如ALT、F10
WM_SYSKEYUP	系统键放开时产生
WM_CHAR	字符消息(TranslateMessage函数发送的)

8.2 消息参数

按键消息：

WPARAM: 按键的虚拟键码值（Virtual Key），代表按下或释放的键

LPARAM: 按键的参数，例如按下次数

WM_CHAR消息：

WPARAM: 输入的字符的ASCII字符编码值

LPARAM: 按键的相关参数

8.3 键盘消息处理

软件需求：用键盘的上下左右键控制一个椭圆运动。

分析：处理`WM_KEYDOWN`消息，在`WM_PAINT`消息处理区绘制椭圆。

代码：在Win32KeyMsg.cpp窗体项目主文件的窗口消息处理回调函数WndProc中，添加键盘处理的代码，如下所示：

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  HDC hdc;            //设备环境句柄，图形绘制工具
  PAINTSTRUCT ps;     //窗口客户区绘图用的信息结构
  //圆的左上角和右下角坐标
  static int x1 = 100, y1 = 100, x2 = 200, y2 = 200;

  switch (message)
  {
  case WM_KEYDOWN:
    switch (wParam) {
    case VK_LEFT:
      x1 -= 50;
      x2 -= 50;
      //使得窗口客户区域无效，并擦除客户区的图形
      InvalidateRect(hWnd, NULL, TRUE);
      //UpdateWindow 函数绕过应用程序的消息队列，直接发送 WM_PAINT 消息给指定窗口的窗口过程。
      UpdateWindow(hWnd);
      break;
    case VK_RIGHT:
      x1 += 50;
      x2 += 50;
      InvalidateRect(hWnd, NULL, TRUE);
      UpdateWindow(hWnd);
      break;
    case VK_UP:
      y1 -= 50;
      y2 -= 50;
      InvalidateRect(hWnd, NULL, TRUE);
      UpdateWindow(hWnd);
      break;
    case VK_DOWN:
      y1 += 50;
      y2 += 50;
      InvalidateRect(hWnd, NULL, TRUE);
      UpdateWindow(hWnd);
      break;
    default:
```

```
            break;
    }
  case WM_PAINT:
    //为指定窗口进行绘图工作的准备，并用将和绘图有关的信息填充到一个PAINTSTRUCT结构中。
    hdc = BeginPaint(hWnd, &ps);
    Ellipse(hdc, x1, y1, x2, y2); //用设备环境句柄绘制椭圆
    EndPaint(hWnd, &ps);    //绘图的结束，释放绘图区
    break;
  ......
  ......
```

9、消息驱动编程模式

Windows是用于个人电脑（PC）上最成功的图形用户界面操作系统，其最早的版本是1985年发布的Windows1.0x。比微软Windows更早的是苹果公司的Mac OS System 1.0，这是第一个划时代的图形界面操作系统，因为它其中的很多技术到今天还在使用，比如，基于窗口用图标的UI，窗口可以被鼠标移动，可以使用鼠标拖动文件和目录以完成文件的copy和move。

微软比苹果晚了一年，可微软是图形用户界面操作系统的狂热者，因为比尔盖茨希望世界上人手有一台电脑。而个人电脑必须要易学易用，原先的那些命令行界面的操作系统肯定不行，只有图形化的用户界面才让普通人接受。所以微软用全部的力量去搞图形用户界面操作系统，最终形成了PC操作系统的垄断地位。

普通人使用电脑，不会像专业人士那样需要处理数据，需要繁重的计算，而是为了办公与娱乐。所以个人电脑上的应用程序有一个基本特点：

有一个图形化的操作界面，可以响应用户操作，用户不操作时进入等待状态。

这就是“**交互式软件**”的特点。由于这种交互式软件占所有软件的绝大部分，太重要了，所以Windows平台的设计目标之一，就是为这种应用软件的开发提供足够的支持。于是就有了上面所述的**Windows的消息机制**。

Windows消息机制对编程产生了深刻的影响，形成了**消息驱动的编程模式**，即：

应用程序员的精力主要放在对窗口消息的处理上，而创建窗口、消息的捕获、传递、翻译、分派等工作交给操作系统。

10、平台与软件

我们还是按照“平台与软件”的角度，来总结今天所学的内容：

1. 平台干了什么？win32操作系统负责创建窗口和窗口消息传递。
2. 程序员干什么？在窗口回调函数里，对各种消息进行处理。
3. 平台与软件的交互机制时什么？应用程序可以调用win32的API，win32也可以调用应用程序注册的回调函数。

附录：源程序中奇怪符号解释

1、HINSTANCE：是“句柄型”数据类型。相当于装入到了内存的资源的ID。HINSTANCE对应的资源是instance.句柄实际上是一个 无符号长整数。

2、ATOM：无符号短整型。

3、LRESULT：就是long，也就是长整形。说明这个函数的返回值是某个结果。

4、INT_PTR：和指针长度相等的INT。在32位操作系统里，一个int是4个字节。64位操作系统上，一个int是8个字节。指针的大小也是同样。所以用INT_PTR代替int理论上可以让代码具有更好的移植性。

5、CALLBACK： __stdcall调用类型：该调用只是通过堆栈来push和pop参数。push参数时，顺序是从右到左。

6、APIENTRY：指示编译器按照API的参数入栈方式编译。

7、*In*：这是一个宏，它的实际意义就是告诉你，这个变量或参数是输入值，即你必须给这个变量填写好以后提交给某个函数去执行。

8、UNREFERENCED_PARAMETER：这是个宏，作用是告诉编译器，已经使用了该变量，不必检测警告!

9、WndProc：hwnd是要处理窗口的句柄，message是消息ID，代表了不同的消息类型，wParam的值为按下按键的虚拟键码，lParam则存储按键的相关状态信息，比如当鼠标消息发出时，wParam值为鼠标按键的信息，而lParam则储存鼠标的坐标，高字节代表y坐标，低字节代表x坐标。即g_y =HIWORD(lParam)，g_x = LOWORD(lParam)。

10、WM_COMMAND：当用户点击菜单、按钮、下拉列表框等控件时候，会触发WM_COMMAND消息。