2020/06/23_SDK_第3课_绘图消息和键盘消息

笔记本: SDK编程
创建时间: 2020/6/23 星期二 15:29
作者: ileemi
标签: 绘图消息和键盘消息

绘图消息 WM_PAINT

WM_PAINT

说明:
当系统或其他应用程序请求绘制一个应用程序窗口的一部分时，WM_PAINT消息被发送。当UpdateWindow或RedrawWindow函数被调用时，或者当应用程序通过GetMessage或PeekMessage函数获得WM_PAINT消息时，消息被DispatchMessage函数发送。

简单来说就是当窗口需要进行绘制的时候，发送一个WM_PAINT消息。

定义：

```
C: > Users > lh > Desktop > Test.cpp > ...
1   LRESULT CALLBACK WindowProc(
2     HWND hwnd,        // handle to window
3     UINT uMsg,        // WM_PAINT
4     WPARAM wParam,    // not used -- 未使用
5     LPARAM lParam     // not used
6   );
7
```

参数：无

返回值：如果应用程序处理此消息，则返回零。

进行绘制时需要调用对应的API函数

通过WM_PAINT 显示文本信息到客户区需要使用的API函数为 -- DrawText

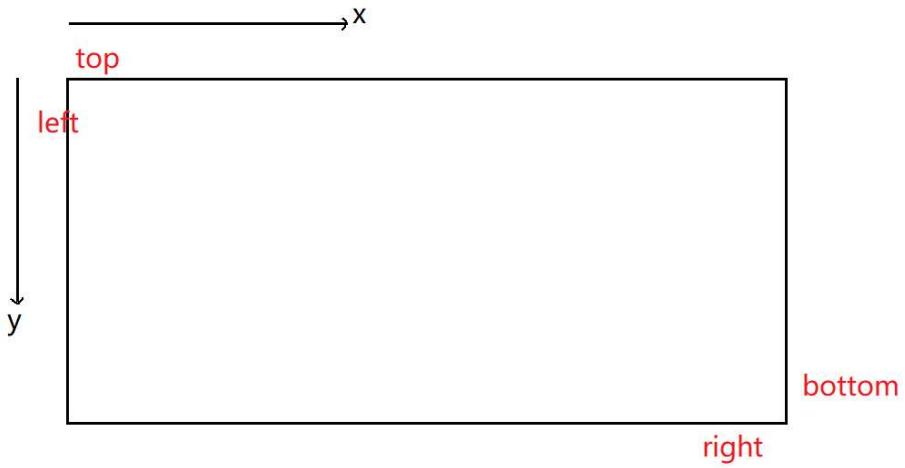
DrawText

说明：
DrawText 函数在指定的矩形中绘制格式化文本。它根据指定的方法(展开制表符、调整字符、换行等等)格式化文本。

若要指定其他格式化选项，请使用 **DrawTextEx** 函数。

DrawText 定义：

```
8
9   int DrawText(
10    HDC hDC,          // handle to DC -- DC的句柄
11    LPCTSTR lpString, // text to draw -- 存储文本的缓冲区
12    int nCount,       // text length  -- 文本的长度
13    LPRECT lpRect,    // formatting dimensions -- 指向一个矩形结构的指针，该结构包含要在其中格式化文本的矩形(逻辑坐标)
14    UINT uFormat      // text-drawing options -- 指定格式化文本的方法，其参数可以是表中的一个或多个值
15  );
16
17  // 参数 lpRect 对应的结构体 其有四个参数，对应两个坐标点
18  typedef struct _RECT {
19    LONG left;    // 左上角
20    LONG top;
21
22    LONG right;   // 右上角
23    LONG bottom;
24  } RECT, *PRECT;
25
```



概述：
DC 对屏幕做了一个抽象（可认为是一个对象），DOS系统中 将屏幕抽象成一块内存（将要显示的内容写入到这个内存中，之后系统再将内存中的内容显示到显示器上），Windows的设计类似于DOS，Windows将屏幕抽象成DC，只要将要显示的内容传递给DC，传递的内容系统自动将其显示的屏幕上。**注意：** 每个窗口都有自己的DC，要想往对应的窗口绘制内容，只需要拿到该窗口对应的DC就可以。

获取DC句柄的 API 函数为 -- BeginPaint

BeginPaint

函数说明：
BeginPaint 函数为绘画准备指定的窗口，并用关于绘画的信息填充一个绘画结构。用来获取DC。**使用BeginPaint** 进行绘制的时候，会向系统申请资源，申请的资源如果不及时释放，会造成资源泄漏。使用 **EndPaint** 来释放资源。

函数定义：

```
HDC BeginPaint(
  HWND hwnd,            // handle to window -- 窗口的句柄
  LPPAINTSTRUCT lpPaint // paint information -- 传出参数，指向将接收绘制信息的绘制结构的指针
);
// 返回值 HDC

// PAINTSTRUCT 了解即可
typedef struct tagPAINTSTRUCT {
  HDC  hdc;
  BOOL fErase;
  RECT rcPaint;
  BOOL fRestore;
  BOOL fIncUpdate;
  BYTE rgbReserved[32];
} PAINTSTRUCT, *PPAINTSTRUCT;
```

在弹出窗口内的客户区绘制文本

- 获取DC句柄
- 绘制文本

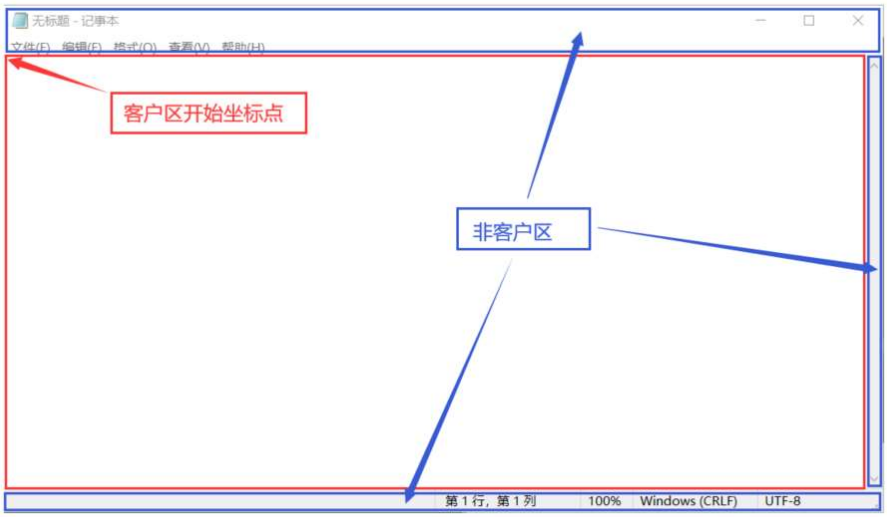
代码示例：

```
case WM_PAINT:
{
    // 通过BeginPaint 获取 DC 句柄
    PAINTSTRUCT ps;       // 传出参数
    HDC hDC = BeginPaint(hwnd, &ps);
    RECT rc = {
        0, 0, 800, 600
    };
    // 绘制文本
    char szBuff[] = { "Hello World!" };
    DrawText(
        hDC,              // DC句柄
        szBuff,           // 显示的文本
        strlen(szBuff),   // 显示文本的长度
        &rc,              // 格式化文本矩形的坐标参数
        DT_CENTER         // 对齐
    );
    return 0;
}
```

客户区和非客户区

客户区：标题栏，菜单栏，滑动条，底部状态栏除外的地方
非客户区：标题栏，菜单栏，滑动条，底部状态栏

示例：



说明： 应用程序不应该调用 **BeginPaint**，除非是为了响应 **WM_PAINT** 消息。对 **BeginPaint** 的每个调用都必须有对 **EndPaint** 函数的相应调用。

BeginPaint 和 EndPaint 应该配对使用

EndPaint 函数标记指定窗口中绘制的结束。这个函数在每次调用 BeginPaint 函数时都是必需的，但只有在绘制完成之后才需要。

名称	PID	状态	用户名	会话 ID	CPU	CPU 时间	GDI 对象	UAC 虚拟化	GPU
RuntimeBroker.exe	21760	正在运行	ileemi	1	00	0:00:01	36	已禁用	00
ZoomIt64.exe	23828	正在运行	ileemi	1	00	0:00:03	33	已禁用	00
DingTalk.exe	2948	正在运行	ileemi	1	00	0:00:17	32	已禁用	00
notepad.exe	2220	正在运行	ileemi	1	00	0:00:02	25	已禁用	00
StartMenuExperienceHost.exe	8664	正在运行	ileemi	1	00	0:00:38	23	已禁用	00
EvernoteSubprocess.exe	16104	正在运行	ileemi	1	00	0:00:26	23	已禁用	00
dwm.exe	16340	正在运行	DWM-1	1	00	0:40:06	20	已禁用	00
vmware-tray.exe	12132	正在运行	ileemi	1	00	0:00:00	20	已禁用	00
ShellExperienceHost.exe	12876	已挂起	ileemi	1	00	0:00:47	20	已禁用	00
MicrosoftEdge.exe	15172	已挂起	ileemi	1	00	0:00:00	18	已禁用	00
SystemSettings.exe	15608	已挂起	ileemi	1	00	0:00:00	18	已禁用	00
QQLiveBrowser.exe	20984	正在运行	ileemi	1	00	0:00:00	14	已禁用	00
taskhostw.exe	7492	正在运行	ileemi	1	00	0:00:00	13	已禁用	00
igfxEM.exe	5140	正在运行	ileemi	1	00	0:02:22	13	已禁用	00
EvernoteClipper.exe	12124	正在运行	ileemi	1	00	0:00:00	13	已禁用	00
QQLiveService.exe	9696	正在运行	ileemi	1	00	0:00:06	12	已禁用	00
conhost.exe	28608	正在运行	ileemi	1	00	0:00:00	10	已禁用	00
SpeechRuntime.exe	7440	正在运行	ileemi	1	00	0:02:02	10	已禁用	00
conhost.exe	11584	正在运行	ileemi	1	00	0:00:00	10	已禁用	00
Work.exe	7360	正在运行	ileemi	1	00	0:00:00	10	已禁用	00
chrome.exe	26692	正在运行	ileemi	1	00	0:02:06	9	已禁用	00
ServiceHub.ThreadedWaitDialo...	25640	正在运行	ileemi	1	00	0:00:01	8	已禁用	00
EvernoteTray.exe	22332	正在运行	ileemi	1	00	0:00:00	7	已禁用	00
LockApp.exe	9904	已挂起	ileemi	1	00	0:00:03	6	已禁用	00

WN_PAINT 消息来的时机

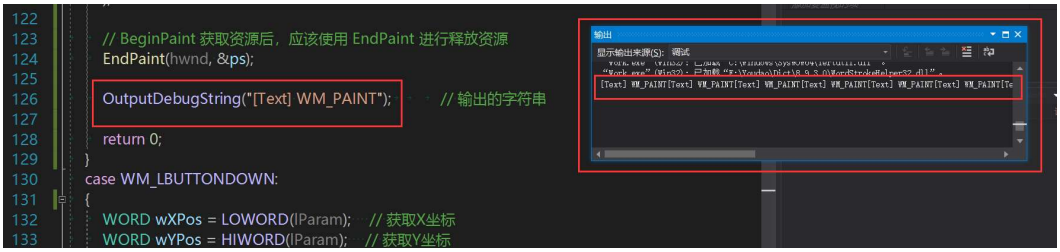
- 窗口的创建
- 窗口尺寸进行修改
- 最大化最下化
- 窗口遮盖部分重现的时候

OutputDebugString

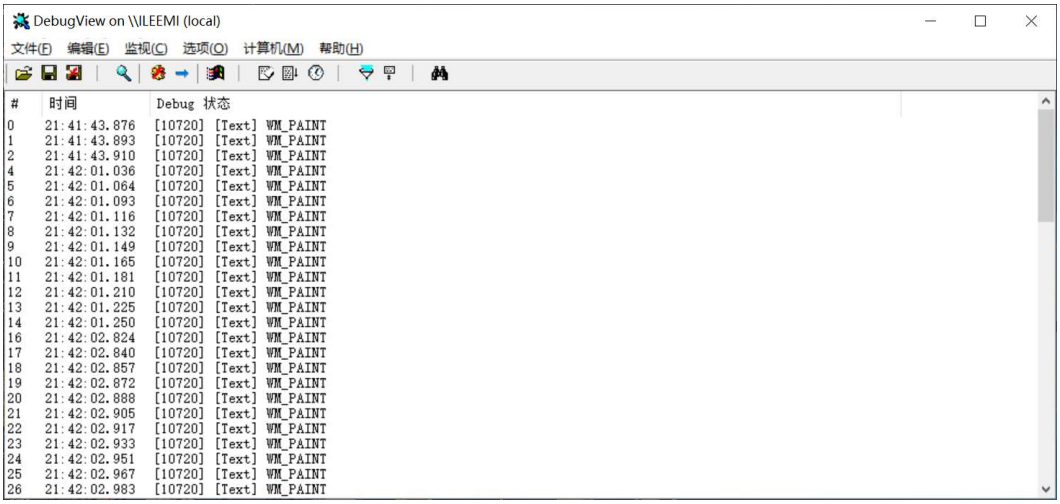
函数说明：OutputDebugString 函数向调试器发送一个字符串以进行显示。

注意：

- 使用 OutputDebugString 输出错误字符串的时候，当在VS 跑程序的时候，按F5调试程序，此时的 **错误字符串** 会输出到VS的输出面板中，如下图所示：



- 按下Ctrl + F5 不调试，此时的 **错误字符串** 会输出到 DebugView on 软件的面板中，如下图所示：



移动窗口 WM_PAINT 错误消息不回来，系统对其进行了优化。

系统不自动发送 WM_PAINT 时候，可通过下面两种办法进行绘图

- 手动制造需要重新绘制的区域 -- 对应的API函数 InvalidateRect
- GetDC -- 不在Paint消息里进行绘画，可以在任何地方进行绘画（但是不能在 BeginPaint 中使用）

InvalidateRect

InvalidateRect 会手动产生无效区

函数说明：

InvalidateRect 函数将一个矩形添加到指定窗口的更新区域。更新区域表示窗口的客户区域中必须重绘的部分。

定义：

```
50
51  BOOL InvalidateRect(
52    HWND hWnd,            // handle to window -- 窗口的句柄
53    CONST RECT* lpRect,   // rectangle coordinates -- 更新区域
54    BOOL bErase           // erase state -- 是否擦除背景
55  );
56
57  /*
58  参数2
59  指向一个矩形结构的指针，该结构包含要添加到更新区域的矩形的客户端坐标。
60  如果该参数为NULL，则将整个客户区添加到更新区域。
61  */
62
63  /*
64  参数3
65  指定在处理更新区域时是否擦除更新区域内的背景。
66  如果该参数为真，则当BeginPaint函数被调用时，背景将被擦除。如果该参数为假，则背景保持不变
67  */
68
```

使用示例：

```
case WM_LBUTTONDOWN:
{
#if 0 非活动预处理器块
#endif // 获取鼠标点击的坐标

    // 鼠标点击时，WM_PAINT 消息不会来，需要手动通过InvalidateRect函数手动调用
    // 鼠标点击时，通过InvalidateRect函数使其自动绘制一块区域
    InvalidateRect(hwnd, NULL, TRUE);
    break;
}
```

GetDC

对应的释放资源的函数 ReleaseDC

函数定义：

```
HDC GetDC(
  HWND hWnd    // handle to window -- 窗口的句柄
);

int ReleaseDC(
  HWND hWnd,   // handle to window -- 窗口的句柄
  HDC hDC      // handle to DC -- 要释放的DC的句柄。
);
```

使用示例：

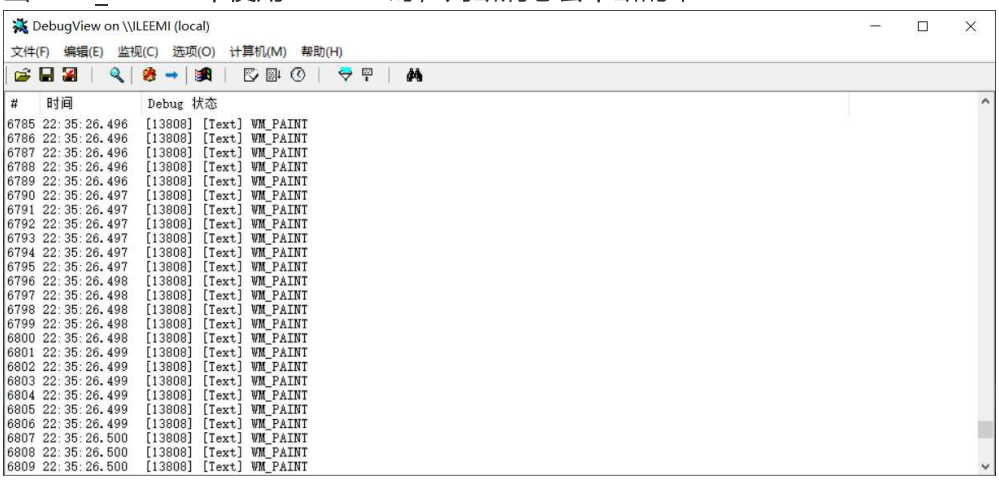
```
PAINTSTRUCT ps;      // 传出参数
HDC hDC = GetDC(hwnd);  // 这里获取 DC 的时候，会申请资源，对应的资源用完后，应该及时进行释放
RECT rc =
{
    100, 100, 800, 600
};
// 绘制文本
char szBuff[] = { "First Window" };
DrawText
(
    hDC,            // DC句柄
    szBuff,         // 显示的文本
    strlen(szBuff), // 显示文本的长度
    &rc,            // 格式化文本矩形的坐标参数
    DT_LEFT         // 对齐
);

ReleaseDC(hwnd, hDC);

break;
```

BeginPaint 和 GetDC 的区别

- 当 WM_PAINT 中使用 GetDC 时，判断消息会不断的来



- 当调用 BeginPaint 的时候，其会将窗口对象中保存的无效区清除掉，而 GetDC 不会清除。
- BeginPaint 绘制区域不能超过无效区
- GetDC 会无视无效区
- BeginPaint 只能在 PAINT 消息中进行使用（BeginPaint 画图跟无效区有关系，没有无效区，使用 BeginPaint 是画不出数据的）

简单来说：BeginPaint 只在 PAINT 中使用，GetDC 在其它消息中使用。

产生无效区 才会触发 WM_PAINT

无效区

当窗口需要重新绘制的时候，所重新绘制的区域就是无效区（一块矩形区域）。
什么时候产生无效区？当窗口需要重新绘制的时候会产生无效区。

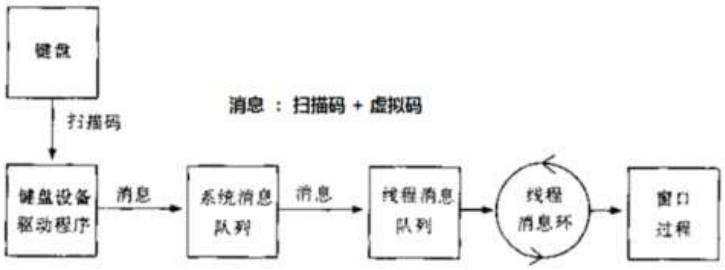
无效区的矩形存储在窗口对象中，系统会检测该窗口对象，如果该窗口对象中的矩形区域不为空，其会一直发送判断消息。

键盘消息

从键盘输入一个字符到显示器显示的流程：

从键盘按下字符（字符对应的扫描码（虚拟码））--> 键盘设备的驱动程序（转换为消息）--> 进入到系统的消息队列中（系统将消息给应用程序的消息队列）--> 应用程序循环从应用程序的消息队列中取出消息 --> 取出的消息交给过程函数进行处理 --

> 之后过程函数在将按下的字符显示出来。



类似鼠标，有按下和弹起操作。

虚拟键：键盘的驱动和程序会把扫描码转换成虚拟码（与硬件无关）。
扫描码：键盘上的每个键都对应一个扫描码（硬件上），各厂商可能不一样。

对应的API函数 -- WM_KEYDOWN / WM_KEYUP

WM_KEYDOWN

函数说明：当一个非系统键被按下时，WM_KEYDOWN消息被张贴到与键盘焦点的窗口。非系统键是在ALT键未按下时按下的键。

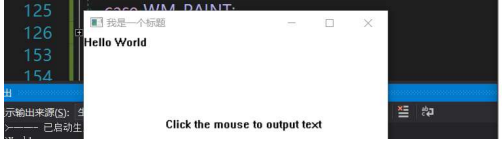
函数定义：

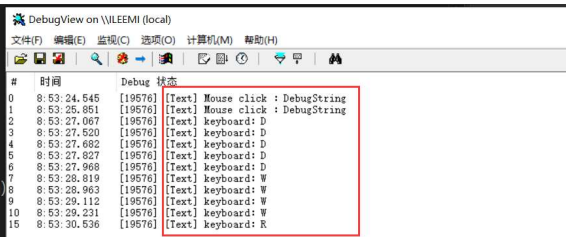
```
79
80  LRESULT CALLBACK WindowProc(
81    HWND hwnd,        // handle to window -- 窗口的句柄
82    UINT uMsg,        // WM_KEYDOWN -- 消息ID
83    WPARAM wParam,    // virtual-key code -- 虚拟码
84    LPARAM lParam     // key data
85  );
86
```

返回值：如果处理此消息，应用程序应该返回零。

应用实例：

```
113 {
114   // uMsg 消息ID(消息类型)
115   switch (uMsg)
116   {
117
118   case WM_KEYDOWN:
119   {
120     char szBuff[MAXBYTE] = { 0 };
121     wsprintf(szBuff, "[Text] keyboard: %c", wParam);
122     OutputDebugString(szBuff);
123     return 0;
124   }
125   case WM_PAINT:
126
153
154
```





注意：WM_KEYDOWN WM_KEYUP 没有大小写之分，默认按下的字符都是大写。

怎样按下字符时显示对应的字符（按下小写显示小写，按下大写显示大写）？

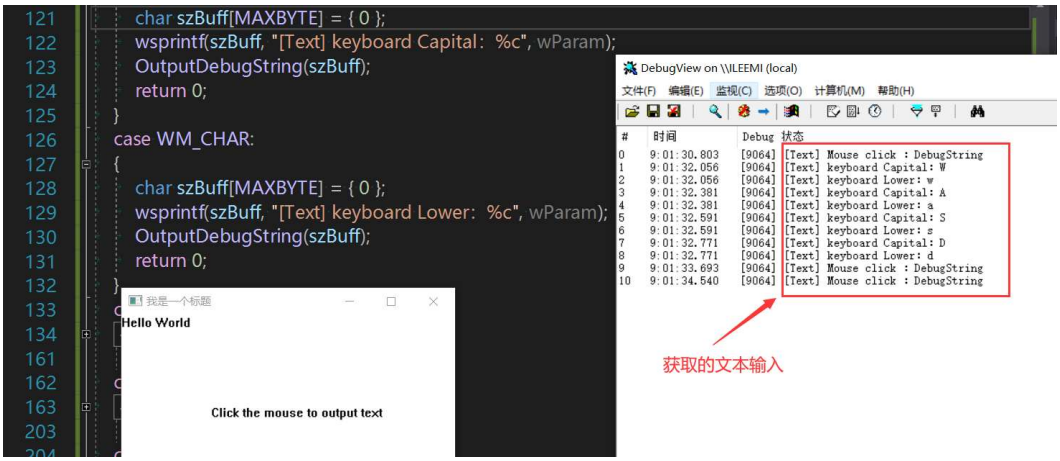
- 检查键盘的大写键是否开启或者shift键是否和字母同时按下（比较复杂）
- 微软提供了一个API函数 -- WM_CHAR（简单好用）

WM_CHAR

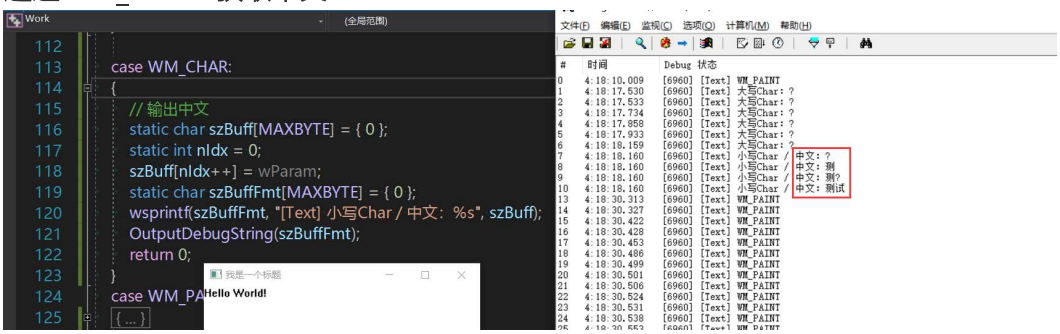
使用 WM_CHAR API 函数还需要配合一个 **TranslateMessage** 函数使用，这个函数可以将虚拟键消息转换为字符消息。区分大小写字母。

API -- TranslateMessage (参数)

应用实例：



通过WM_CHAR 获取中文：



TranslateMessage

函数说明：
TranslateMessage 函数将虚拟键消息转换为字符消息。字符消息被发布到调用线程的消息队列中，以便下一次线程调用GetMessage或PeekMessage函数时读取。

函数定义：

```
87
88  BOOL TranslateMessage(
89    CONST MSG *lpMsg   // message information
90  );
91
92  /*
93  参数解析：
94  指向MSG结构的指针，该结构包含通过使用GetMessage或PeekMessage函数从调用线程的消息队列中检索到的消息信息。
95  */
```

参数：
指向MSG结构的指针，该结构包含通过使用GetMessage或PeekMessage函数从调用线程的消息队列中检索到的消息信息。

用法示例：
用在 DispatchMessage 函数之前

定时器消息

API函数 SetTimer，每隔一段时间向指定的窗口发送消息。通过 WM_TIME 进行相应。

WM_TIMER
返回值：如果处理此消息，应用程序应该返回零。

开定时器： SetTimer 关闭定时器：KillTimer

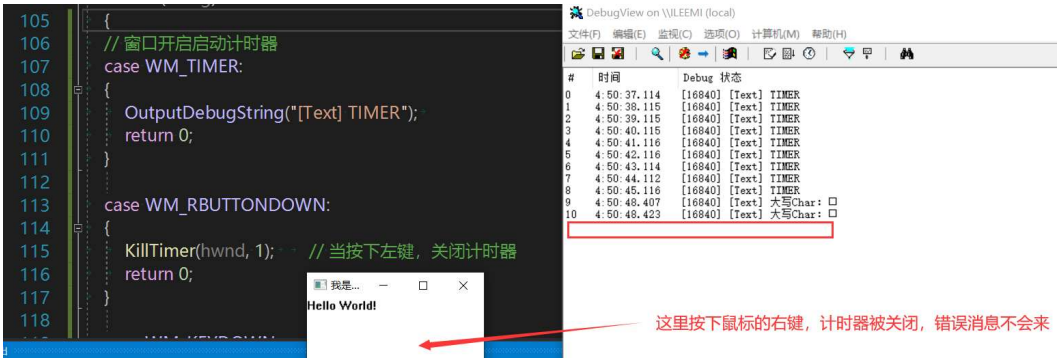
函数说明：
SetTimer函数使用指定的超时值创建计时器。
KillTimer函数销毁指定的计时器。

函数定义：

```
97  // 开定时器
98  UINT_PTR SetTimer(
99    HWND hWnd,              // handle to window -- 设置窗口句柄后，每过一段时间发送一个消息
100   UINT_PTR nIDEvent,      // timer identifier -- 定时器ID
101   UINT uElapse,           // time-out value   -- 时间间隔
102   TIMERPROC lpTimerFunc   // timer procedure  -- 回调函数，一般给NULL
103 );
104
105 // 关定时器
106 BOOL KillTimer(
107   HWND hWnd,              // handle to window -- 设置窗口句柄后，每过一段时间发送一个消息
108   UINT_PTR nIDEvent,      // timer identifier -- 定时器ID
109 );
110
```

用法示例：

```
84    // 开启定时器
85    SetTimer(hwnd, 1, 1000, NULL);
```



作业

1、创建一个程序，测试WM_KEYDOWN、WM_KEYUP、WM_CHAR的响应顺序；

通过DebugView 观察可以发现WM_KEYDOWN、WM_KEYUP、WM_CHAR的响应顺序依次为：

WM_KEYDOWN --> WM_CHAR --> WM_KEYUP