

C 语言究竟能干什么

序言

鉴于现在已经大三了，很多同学很迷茫，自己学的东西到底能做什么，将来自己到底能干什么？我不想看着同学迷茫的面孔，

特别是几个好兄弟，有几个想学习编程，但又苦苦找不到门路的兄弟，所以想写点东西，希望对大家又点略微的帮助，以尽兄弟我的微薄之力。

很多同学学了 C 语言之后，可能难免会有所感叹：这就是 C 语言！总是感觉 C 语言竟然能写出 Windows、Linux？为了解除同学们

的疑惑，也愿为同学们指点编程之道吧。我写的这些东西采用 C 语言，计划通过编程实例来讲解 C 编程的一些知识，让大家对 C

能又更深一层的理解和认识。当然，大家不要指望看了这些之后会写出一个操作系统来，但是我想，如果你认真看了的话，写

一个类似与 QQ 的聊天程序应该不难。

回顾一个自己的学习经历，高二时，有个上大学的表哥，学的是计算机，暑假到他家里玩，不经意间，看到了他的 C 语言书，

是谭浩强编的那本，是第几版倒是记不起来了。当时其实都不知道计算机到底是咋回事，那时显示器一般都是 CRT 的，就认为那个方块的显示器就是电脑的全部，根本就不晓得主机是啥玩意儿。那次就看了一点 C 的语法，记了点模模糊糊的东西，好像是知道了有那么个叫循环语句的东西，但也不晓得这东西咋能编出程序来呢。其实，那之前连键盘都没摸过几次。

再后来，高三时，洛阳市第二次大练习之后，感觉自己的压力重，心情颇为郁闷，就和一个同学去网吧解闷，申请了平生的第一个 QQ 号，到现在我都用这着，当时不知道申请了多少次，被腾讯公司给忽悠了多少遍，才终于申请到一个，倍感来之不易，虽号不吉利，仍不忍弃之。

高考后，同学们都说该放松放松了，基本都跑到网吧去放松了。以前去过没去过的，会开机的不会开机的都到网吧了。以至于大小黑正网吧爆满，机器不够用。广大学生疯狂通宵。结果呢，第二天就有某网吧中因人夺机打人致死的消息，于是乎有

关部门就下令网吧不得开放通宵，午夜 12 点一律关门。

那个时候，也知道了电脑是由主机的，但是不知道主机的两个按钮是干啥用的，不知道哪个是开机键，哪个是复位键，反正

到了网吧，要不电脑是开着的，若不开，把两个按钮都按了，电脑自然是会开的，那时纯粹是好奇，感觉电脑挺好玩的。

高考后，没有进入理想的大学，就选择了复读。复读时，自认为功力已到火候，就没有了百尺竿头更进一步的耐力了，日

练几题，维持水平而已。闲的时间不再是埋头做题，而是总结得分技巧，看看一些时尚读物，有空时还到阅览室去看看。由于

暑假对电脑萌生爱意，于是就去找有关电脑的读物读。学校这方面的书可真是少的可怜，看得懂的就更少了。唯有《电

脑爱好者》还算能看懂上面的一些汉字，不过都是明日黄花——一年前的过期的了。当时看了上面的在互联网上筑巢，感觉做网站

的确很酷，知道在互联网申请免费的空间来放网页。然对其是何东东尚且不知，只知道按上面说的一步一步做就是了。书上说

用的 home4u.china.com 申请的，自己就以自己名字的汉语拼音申请了个网址（姑且这么说吧，当时自己根本就没见过域名这个

词）：liangxingqi.home4u.china.com 的 10M 的空间。当时这个系统做得非常的不好，反应速度非常慢。第一次我就在上面放

上了一些个人的信息，不过还是一种成就感自心中油然而生，使我高兴良久。

后来，又在一期过时的《电脑爱好者》上看到了利用 Google AdSense 赚钱的文章，于是就照书上所说的注册了一个 Google 帐

户，待审核成功后，我直接把 Google 提供的广告代码，放到我那个空间上，广告果然显示了，Google 账户里还真有点钱了，这

更增加了我学网页制作的信心。这样了一段时间之后，就不满足空间提供商所提供的功能组件了，因为其反应速度实在是太慢

了，会浪费很多时间，当时又是在网吧，也会花去我好多 RMB 的。本来人就穷，都是从饭钱里省出来上网的（想起来，那真是

一段苦涩的日子呀）。后来注意那个系统旁边有那么一个警示，大概意思是这样的：如果系统反应速度慢，请先用 FrontPage

等网页设计工具设计好后，直接将代码拷过来提交即可。当时什么也不懂，于是就按照上面说的做，到县城的图书馆里买了一本介绍用 FrontPage 设计网页的书，花了我三十多块钱，对于那个年代的我来说，可真是大出血了，当时就

想一定要把这钱给赚回来。买了书回去就猛看了一通，可谓是废寝忘食。到了星期天就没有回家，晚上在网吧通宵了一把。按照书上的步骤：开始→程序→FrontPage 去启动 FrontPage。找了一个晚上，硬是没有找到，甚是郁闷。也就从那次之后，才慢慢的知道了什么是软件，也知道了微软不是生产电路板的了。虽然开机都有 Microsoft 字样，但已经隐隐约约地知道了，这不是电路设计的结果。后来，在一次一个同学说微软是生产电脑的时候，我理直气壮地说：微软是生产电脑的，那中国的长城、联系是干嘛的。

想想当时自己就知道了软件与硬件的区别，的确挺牛的，哈哈。毕竟很多人到了大学还找不到开机按钮的。

知道了 FrontPage 是软件后，就想着到网吧下载就是了。到了网吧去下载，当时上网的人下载并不疯狂。迅雷之类的下载软件也都还处于襁褓之中。下载纯粹就是 IE 的下载功能，记得当时点了下载，看到出现的下载对话框上显示“剩余时间约概 51 小时”，我都差点吐血，随即就点了“取消”。自己来网吧最长时间也不过 8 小时（通宵），连一个软件都下载不了，而且自己也没有那么多的上网费。就这样，学习用 FrontPage 制作网页的梦想也就破灭了。

后来，和一个上大学的同学（现在在支付宝工作）聊起来，他说他们学习用 Dreamweaver 设计网页的。于是就决定再次出血买本 Dreamweaver 的书，这次花了四十多块钱，没把我给吐死。还好，在洛阳市第一次演练中，数学考了全市的 53 名，得了 50 元钱，将此血口给堵上了，才使我没有损失多少元气，能够活到现在。

到网吧下载 Dreamweaver 才发现它的安装文件才 70 多 M，真是善哉，善哉！不到一个小时就能够下载完，时间上还算能说的过去。于是就彻底弃 FrontPage 而投 Dreamweaver。买书时不知道软件还有版本，拿着一本 3.0 的书去操作 8.0 的软件，错误实在是难免了，很多都做不成书上所说的效果。如今想起，不免淡笑。那本 Dreamweaver 的书中有很多 html 标记，由于手头上就一本书，又不敢再出血买书，岁看不懂，但仍每天翻之数遍，数月之后，竟然把 html 的大多数标记给记住了，并且能够手写代码设计写简单的网页，真是天助我也！

懂了 html 的一些标记后，网页设计对我来说也不是那么羞涩难懂了。那时在放 Google 的广告，也不会感到手足无措，不知道放哪合适了。虽然广告是没有赚到几个银子，倒花了不少钱，但学到的一些知识，总让我洋洋自得，认为这是值得的，多少给自己以慰藉。

哎，也许是天意弄人，也许是上天注定……

再次高考，再次落榜，无奈在补报志愿中，选了一个垃圾学校的信息工程专业，因为感觉可能是学计算机的。（不想提及伤心往事，此段就是简之）

高考的再次失利，对我的打击是沉重的，以至于整个大一都经常做噩梦，茶不思，饭不想，身体日趋消瘦。大一，在苦闷中度过了大一。

刚到大学时，第一月学校不让新生进图书馆。发下来的书都是通时课的，没有什么实质性、针对性，就想买基本计算机的书看下，希望可以略释我的愁思。

到了校外的小书摊上，看到一本《黑客手册》。想着学黑客那不酷毙了，而且书也不贵，才四块钱，还带光盘，不过就是过期的了（这也许就是我技术落后别人的原因了吧，老看过期的书）。那本《黑客手册》我是翻得最烂的了。最初只是利用上面的工具，用的最多的就是明小子 Domain3.5 和 啊 D 的注入工具。

当时对那些 asp 的网站，真是神了，填上网址就可以拿到后台。哎，当时的网络安全性可真差呀！

我也是由于这个偶然的机会学习了一些黑客技术。这个时候做网页的水平仍旧是停止不前。有一次想做一个网页上的登录功能，却不知道如何实现这个功能，界面出来了，却不知道该怎么去处理。于是就向我那个同学讨教，他说让我学习下 asp。后来图书馆开放了，就借了一大堆 asp 的图书，猛扑上去，这个时候，知识也像泉水一样涌进我的大脑。

大一第二学期，自己就带个破电脑（此言不虚，那是要多破有多破的）到学校，算是班里早一批有电脑的人士了吧。有了电脑，就告别了网吧中那乌烟瘴气的学习环境，而且也不用担心时间按的问题了。从此自己的学习可谓是一日千里，加之电脑超破，经常跟我闹别扭，于是闲暇之时，常常拆卸，摸索修理，对电脑的各个部件也有了一定的了解，为了解决故障，经常到网上查一些故障的解决办法，慢慢地积累多了，就成为经验，就成为技术了。后来同学陪电脑要我去，电脑有点小毛病也让我去帮忙看下。给了我很大的自信和很多的学习实践机会，还有很大的面子，在此感谢他们。

到了大二，自己的电脑水平已经是上了一层楼了。此时拥有的知识有 asp、vbscript、javascript、sql、css、xml，这些都是算上编程的东西吧，对网络我也算是颇有研究吧，也懂得传输的一些协议细节与原理。还有一些 Windows 系统的一些设置和一些黑客知识，还有一点硬件知识。

大二下学期，算是找了一个所谓的兼职吧，给人维护一个网站和开发新的功能。为了解决一些问题，我当时可是绞尽脑汁、想方设法，并且有了自己的解决方案，还编程实现了，性能也可以。用了一点 ajax，这个时候也了解了浏览器处理 html 的原理。而后，有学习了 php、asp.net 虽然并不深入，然想深入研究，对我来说，未必是件难事。

到了此时，学习遇到了瓶颈，感觉再进展是举步维艰。就像 asp 中的有些问题，知道解决方法，但是并不知道原由来。有些问题也许就是 asp.dll 的造成的。因此这个时候就想着学习下底层的東西，以便能清楚地知道计算机运行过程的一些细节，不求甚解，但求了解个大概。

光阴似箭，回首之间，一是大三了，刚好这学期开了《微机原理——基于 32 位的汇编语言》，加上以前电路知识和其他知识的积累，这门课对我来说并不是太难。

哎哎哎！！天有不测风云，人有旦夕祸福！就当我兴趣很浓地学习一些底层动词的时候，身体一向很好的母亲却脑出血，于是回家一个月，母亲仍没有脱离生命危险。但已经快该考试了，于是就回学校准备考试。这段时间按真是苦闷的日子，估计也是我这一生中最苦闷的了（现在也是），为了掩盖着该死的苦闷，我拼死的学习底层知识，写代码，希望能把这该死的痛苦给压下去。

黄天不负苦心人，自己的能力是得到了很大的提升。然上天也故意捉弄人，眼看母亲就快好了，却又把她给带走了。

本来书是假期里照顾妈妈时写的，原想是 1、不让自己的水平停滞不前，温故知新（两个月的假期是很长的）

2. 帮助一些同学，解决编程上的困惑

3. 希望妈妈快点康复

4. 让母亲和家里人知道自己一直都很努力，我是好样的

但现在母亲已经能够不在了，所以第三点，就改为原母亲在天之灵得到慰藉，早日放下烦恼，不用惦记我们。

C 语言的基本语法我是不打算再提了，很多 C 语言编程的书，就是将一些基本的数据类型、数据结构、语法，然后就是一些数值

计算的实例，大多数都是雷同的，难免有抄袭之嫌，而且页没有多少实用价值。

本书以实用实例作为编程指导，指引大家编写真正实用的程序。了解到大家对黑客程序、病毒、窗口类程序比较感兴趣，因此我就拿这些实例进行讲解。基于大家基本都用 Windows XP SP3，我也就在这个系统上把程序调试成功后再给大家讲解。编程环境，我还是喜欢 Visual C++ 6.0

本书计划从四个大的方面来讲，这四个方面是：窗口类、文件操作类、网络类、数据库类。

都是时下流行的编程必备技术，也是软件开发者，必须掌握的技术。中间以实例讲解，逐步学习，相信大家看完后会有很大的提高的。

第一章 窗口类程序的编写

这一章就先来讲解下窗口类程序的编写。因为现在程序没有界面，就像人没有脸面一样，而且好的界面更能吸引人。从基本的界面开始，相信能给大家指明出一条路的，使大家很容易地掌握窗口序的编写。其实界面设计利用 VC 6.0 的 MFC，很容易地制作出来。这里从底层开始写代码来写界面程序，使大家知道一些底层的東西，为以后学习打下好的基础，相信您学了这些，再用 VC 的 MFC 会得心应手的。

1.1

用 C 写的第一个窗口程序

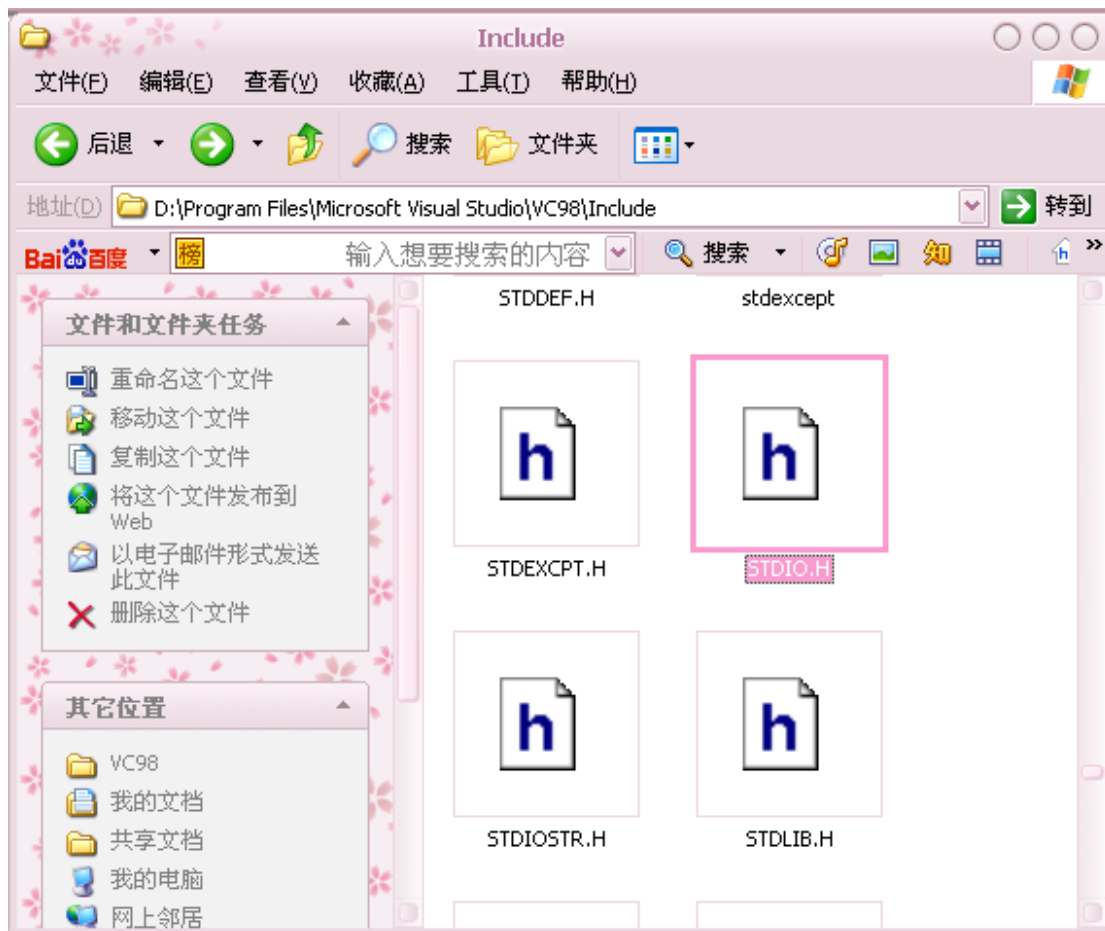
作为编程的开始，我们还是以一个 Hello World 来开始我们的学习之旅。代码如下：

```
#include <stdio.h>

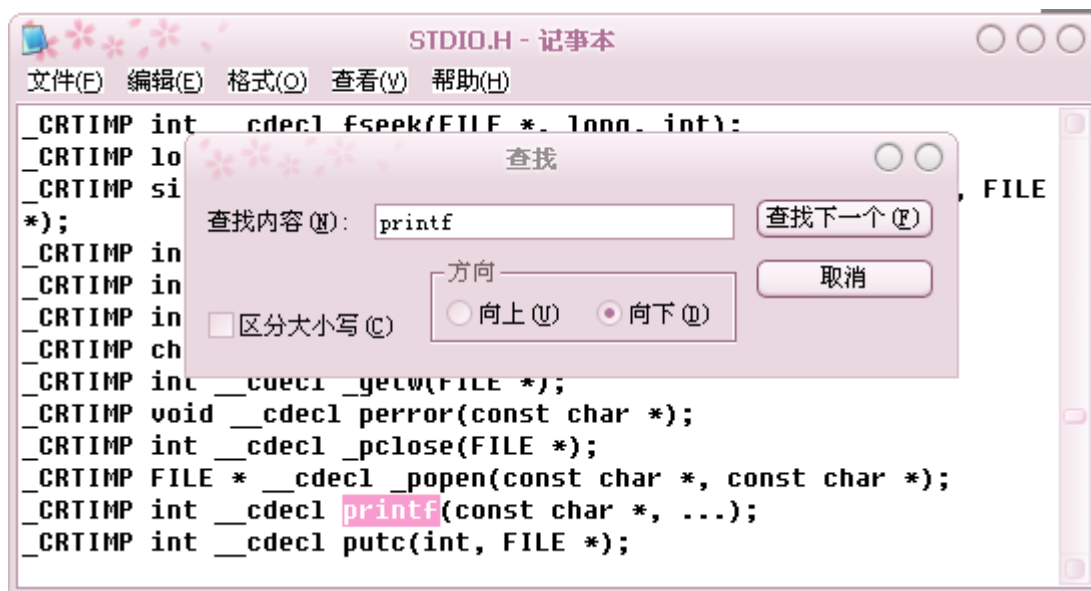
void main()
{
    printf("Hello World!");
}
```

这是一个再简单不过的 C 程序了，只要有点 C 语言的知识就能够懂的，不过这里估计还有些人，到现在还不知道#include

<stdio.h>中的头文件 stdio.h 到底是什么东西，我就来说下了，stdio.h 是一个文本文件，存在于磁盘上的，已 VC 为例它的位置如下图：



也许你听说过 `printf()` 函数是在 `stdio.h` 中预定义的，但是你见过其定义的形式没有，没有且看下图



其定义形式，就如图中所示，也许你并不懂前面那些东西是什么，不用担心，以后我会慢慢解释给大家的。函数是先定义才能使用的，所以 `stdio.h` 中定义 `printf` 函数，我们在引用了 `stdio.h` 头文件后就可以在程序中调用 `printf` 函数了。

上面是在命令行中显示一个“Hello World!”，没什么意思，下面我写一个窗口程序，显示个 Hello World!

```
#include <windows.h>
```

```

void main()
{
    MessageBox(NULL," Hello World!","我的第一个窗口程序",MB_OK);
}

```

编译运行后如下图：



弹出的是一个对话框，上面有 **Hello World**，还有一个标题和一个“确定”按钮。

当然你会说这对话框也算个窗口吗？这里肯定的告诉你：是的，对话框是窗口程序的一个子集。你可能还会这样问，这样一个简单的窗口有啥用呢，其实这样的窗口非常有用，我们在操作计算机的时候，会出现一些警告或提示的对话框，都是基本是这种方法写出来的。就算是这个很简单，学习本来不就是有易向难，有浅显深奥去的过程吗。

整个效果几乎就是靠一个函数 **MessageBox** 的功劳。这里也先不介绍这个函数了，说些其他的。

其实用 C 编写一些恶程序，就是把编程环境中所提供的一些函数熟悉了基本就可以了。用 VC 来写成序，其中的头文件有很多，定义了很多 Windows API 函数、数据结构、宏，可以让我们大家运用，通过它们，我们可以快速开发出使用的程序。这些 Windows API 在微软的 MSDN 上查，上面有很多说明，部分还有代码示例。不会是可以输入函数名，查找相关信息，建议大家用英文版的 Library，因为其内容比中文版的全面，英语不好的同学呢，就先看中文了

中文 MSDN:<http://msdn.microsoft.com/library/zh-cn/>

英文 MSDN:<http://msen.micorsoft.com/library/en-us/>

到这里，我们就完成第一个有界面程序的编写，你感觉写有界面的程序难吗？显然不难。

下面看一个向锋和波波感兴趣的程序：九九乘法

采用命令行形式

```
#include "stdio.h"
```

```
int i=0,j=0;
```

```
for(i=1;i<10;i++)
```

```
    for(j=1;j<i+1;j++)
```

```
        printf("%d*%d=%d\t",j,i,j*i);
```

```
    printf("\n");
```

和那个 javascript 效果都是一样的，所以语言只要学好一样，其他的就很容易旁通的，学习就捡一种学好，不要贪多。

好的，这一节就这样吧，大家先各自了解下微软的 MSDN，对以后的学习会有很大的帮助的。

1.2 第一个真正的窗口程序

上一节中，我们用 **MessageBox** 函数轻松地实现了一个对话框窗口，可能你会说，那仅仅是个没有用的对话框而已，是的，只是对话框而已。我之所以以一个对话框为例呢，是因为我只是想让你知道写一个有界面的程序并不是件难办的事。明白了这一点后，我们继续。今天来编写一个真正的窗口程序。

下面就该罗嗦一段了，由于大家以前并没有写过什么窗口程序，写的都是命令行下的，我们知道在命令行下的程序都有一个主函

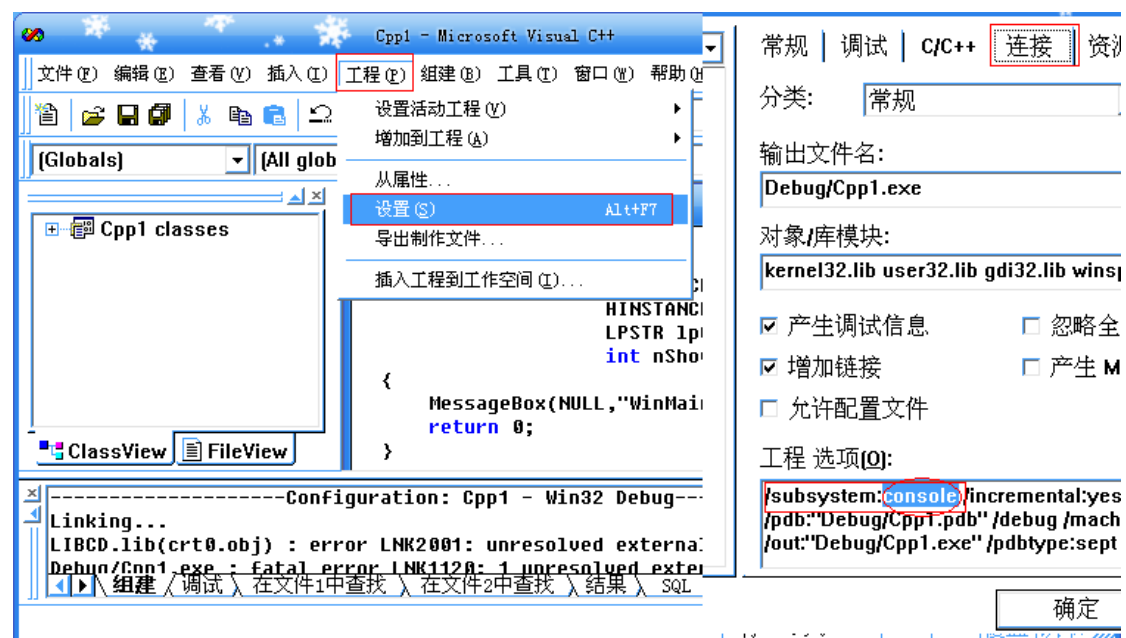
数 `main`，这个函数也就是程序的入口函数。我们现在用 VC 6.0 来写，而且要写窗口类程序，VC 6.0 给我们提供了一个专门用作窗口类程序的入口函数 `WinMain()`

这个函数原型是这样的

```
int WINAPI WinMain(  
    HINSTANCE hInstance,  
    HINSTANCE hPrevInstance,  
    LPSTR lpCmdLine,  
    int nCmdShow  
);
```

大家是不是感觉这个函数挺复杂的，有这么几个参数，而像 `main` 好像就没有参数。其实 `main` 是有参数，这个向锋和小四是知道了的。但是 `main` 函数的参数是可以省略的，而 `WinMain` 是不可以省的。这里也要对 VC 6.0 的编译模式改下

看下图



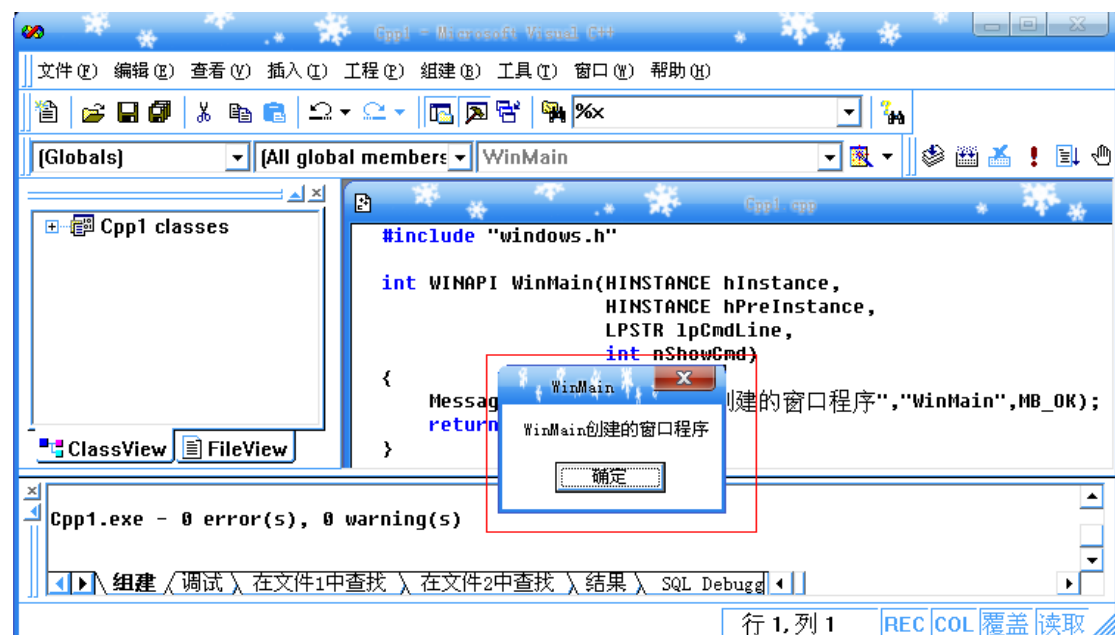
依次是“工程”→“设置”→“连接”，在“工程选项”里把 `console` 改为 `windows` 就可以了。如果认真学了汇编，或是手写命令编译连接过 C 程序，就会知道这是干什么的。`Console` 是控制台的意思，以前我们用 `main` 函数写的程序都是以控制台模式连接的，所以很少会有界面的。现在我们要写有界面的程序，所以要选 `Windows`（窗口）模式了。

我们写入以下代码，并按照上面说的方法去做，看看结果

```
#include "windows.h"
```

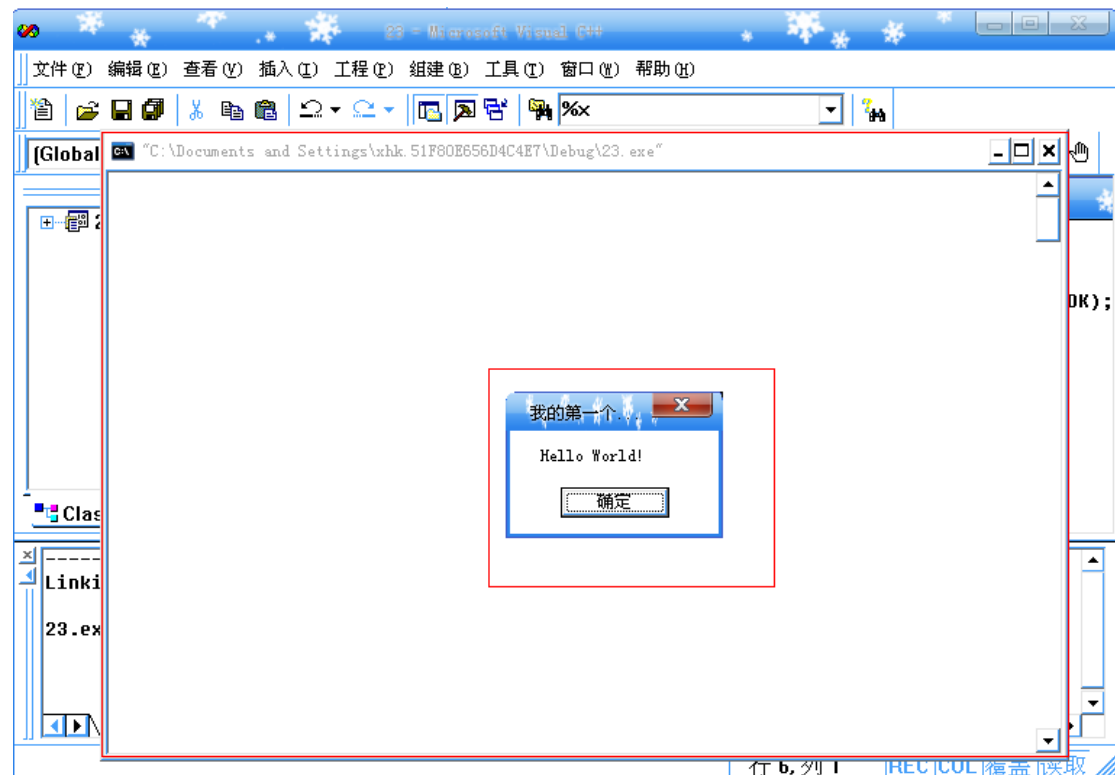
```
int WINAPI WinMain(HINSTANCE hInstance,  
    HINSTANCE hPreInstance,  
    LPSTR lpCmdLine,  
    int nShowCmd)  
{  
    MessageBox(NULL, "WinMain 创建的窗口程序", "WinMain", MB_OK);  
    return 0;  
}
```


结果如下图:



与第一节中的这段代码代码比较下

```
#include "windows.h"
void main()
{
    MessageBox(NULL, "Hello World!", "我的第一个窗口程序", MB_OK);
}
```



两者比较下,后者多了个 cmd 窗口.可见用 main 写的并没有完全脱离命令行呀.所以以后我们写窗口程序就用 winmain 了。

好了，转过来，我们来看看 WinMain()函数，其中有 4 个参数

先看下解释（看不明白得先看完）：

hInstance：应用程序当前事例的句柄。

hPreInstance：应用程序的先事例的句柄。对于同一个程序打开两次，出现两个窗口第一次打开的窗口就是先前实例的窗口。对于一个 32 的位程序，该参数总为 NULL。

lpCmdLine：指向应用程序命令行的空字符串的指针，不包括函数名。获得整个命令行，参看 GetCommandLine。

nCmdShow：指明窗口如何显示（是隐藏还是显示，有没有最大化按钮之类的）。取值可以参考 MSDN

这里我相信有一个词大家好应该比较陌生，句柄(HANDLE)是吧。下面我就来简单的说下

句柄其实就是 Windows 系统中一个东西的唯一标识。就是系统中有很多运行的程序或者资源之类的，为了更好的管理使用，Windows 系统给它们每人一个 ID 一样。懂得网页制作的人应该知道网页中各个元素的 ID 吧，网页的 ID 如果重复话可能出现错误。那么系统的句柄会不会有相同的，那是肯定不会有的了，就和我们的学号一样，系统自动分配每一个模块的句柄，是不会相同的了。

对于句柄大家可以先这样理解着，不用一下子搞懂得。以后学着学着就明白了。

估计大家对那几个参数的类型改犯迷糊了吧。其实那几个类型，并不是什么新类型，都是 Windows 开发人员为了自己与他人编程方便，同过基本的 C 语言语法定义一种新的结构体，或者是共同体，再者就是枚举类型。我知道结构体、共同体和枚举类型，很多老师是没有讲到的，因为在书的后边，很多教 C 的，又是很垃圾的老师，所以不会讲那么快的。其实结构体这些数据类型，就是通过我们常用的字符、整型、浮点等数据类型构造一个比较复杂的类型而已，举个例子，就是我们知道 C 没有一个数据类型可以描述一个人吧，那么我构造一个是不是很方便我们编程呢。我们可以这样构造一个

```
struct People
```

```
{
    int age;//年龄
    char sex[2];//性别
    int height;//身高
    .....
}
```

我们这样定义以后就可以在我们以后的程序中利用这个数据类型了，People zhangsan;把 zhangsan 的身高 172 放到 zhangsan.height 中。这样可以方便完成很多工作。所以结构体是很简单的，还有其他的复杂数据类型也是很简单的，都是有常用的简单的类型来结合到一起构造一个复杂的而已。这和 JAVA 定义类是很相似的，java 定义个人类，不是可以这样的

```
public class People
```

```
{
    public int age;
    public string sex;
    public height;
    .....
}
```

看起来都差不多，而且用法也很相像。唯一的差别其实就是类可以有方法，而结构体是没有的（经过特殊处理也是可以的，这里不用考虑）。

上面是为了让大家了解下复杂数据类型的定义，罗嗦了一大堆。下面来看下 WinMain 中第一个参数的类型 HINSTANCE 这个只是个结构体而已，实际上和 HANDLE 这个类型差不多，但是有一点差别，而 HANDLE 是这样 typedef PVOID HANDLE;定义的，PVOID 是什么呢，我们来看下 typedef void *PVOID;说明 PVOID 是一个指针，初始指向空（void）。因此可以知道句柄也是个指针而已。看着这么复杂原来也只是指针。

这些都可以在微软的 msdn 上查得到的，而且很详细的

那个第二个 LPSTR 根据字面上的意思就知道是字符串类型了。查一查果然是。

大家一定要利用好 msdn，很有用的。

本节就到此结束了，主要是说明了一个 WinMain 函数和结构体的事情，东西也不算太多，大家应该能接受得了吧。下节就来点复杂点深点的东西，希望大家做好心理准备。

1.3 窗口程序的编写

在来啰嗦之前，希望大家能够做好准备，这一节知识有点多，内容有点长。但愿大家能够一口气读完，如果一口气读不完，那就换口气接着读。

上节中我们用 MessageBox()就实现了一个真正的窗口。MessageBox()中的原型如下：

```
Int MessageBox(HWND hWnd,
                LPCTSTR lpText,
                LPCTSTR lpCaption,
                UINT      uType);
```

参数解释

hWnd 所属对话框所属窗口的句柄，如果是 NULL，则此对话框不属于任何一个窗口。

lpText 对话框窗口的显示内容。

lpCaption 对话框窗口的标题。

uType 对话框的样式和动作（像是确定按钮，还是取消按钮就是设置这里的）

关于这个函数的细节可以看这里

[http://msdn.microsoft.com/en-us/library/ms645505\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms645505(VS.85).aspx)

到此为止，你也算是会了窗口程序的编写，但只是一个开始，不过这已经很好，可能会让你感觉到了 C 的魅力，也可能会稍微解点 C 语言能干什么的疑惑。在开始写代码之前，我有必要把细节和原理先说明下。

Windows 下一个窗口创建的过程有以下几个步骤：

1. 程序创建一个窗口，首先要向 Windows 系统注册一个窗口类 wndclassex，其实就是定义一个变量，变量的类型是 WNDCLASSEX(结构体)。该结构体的定义与介绍看这里 ([http://msdn.microsoft.com/en-us/library/ms633577\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms633577(VS.85).aspx)),

```
typedef struct {
    UINT  cbSize;

    UINT  style;

    WNDPROC lpfnWndProc;
```

```

    int    cbClsExtra;

    int    cbWndExtra;

    HINSTANCE  hInstance;

    HICON  hIcon;

    HCURSOR  hCursor;

    HBRUSH  hbrBackground;

    LPCTSTR  lpszMenuName;

    LPCTSTR  lpszClassName;

    HICON  hIconSm;
} WNDCLASSEX, *PWNDCLASSEX;

```

成员介绍

cbSize 值为 `sizeof(WNDCLASSEX)`, 在调用 `GetClassInfoEx` 前必须要先设置它值。

style 窗口类的样式，它的值可以是窗口样式值的任意组合。

可以有以下的值

lpfnWndProc 指向窗口处理函数（回调函数）。处理窗口事件，像单击鼠标会怎样，右击鼠标会怎样，都是由此函数控制的。

cbClsExtra 为窗口类的额外信息做记录，系统初始化为 0。

cbWndExtra 记录窗口实例的额外信息，系统初始为 0. 如果程序使用 `WNDCLASSEX` 注册一个从资源文件里创建的对话框，则此参数必须设置为 `DLGWINDOWEXTRA`

hIcon 窗口类的图标，为资源句柄，如果设置为 `NULL`，系统将为窗口提供一个默认的图标。

hCursor 窗口类的鼠标样式，为鼠标样式资源的句柄，如果设置为 `NULL`，系统提供一个默认的鼠标样式。

hbrBackground 窗口类的背景刷，为背景刷句柄，也可以为系统颜色值，如果颜色值已给出，则必须转化为以下的 `HBRUSH` 的值

- `COLOR_ACTIVEBORDER`
- `COLOR_ACTIVECAPTION`
- `COLOR_APPWORKSPACE`
- `COLOR_BACKGROUND`
- `COLOR_BTNFACE`
- `COLOR_BTNSHADOW`
- `COLOR_BTNTEXT`
- `COLOR_CAPTIONTEXT`

- COLOR_GRAYTEXT
- COLOR_HIGHLIGHT
- COLOR_HIGHLIGHTTEXT
- COLOR_INACTIVEBORDER
- COLOR_INACTIVECAPTION
- COLOR_MENU
- COLOR_MENUTEXT
- COLOR_SCROLLBAR
- COLOR_WINDOW
- COLOR_WINDOWFRAME
- COLOR_WINDOWTEXT

lpszMenuName 指向一个以 NULL 结尾的字符串，同目录资源的名字一样。如果使用整型 **id** 表示菜单，可以用 **MAKEINTRESOURCE** 定义一个宏。如果它的值为 NULL,那么该类创建的窗口将都没有默认的菜单。

lpszClassName 窗口类的名字，字符串类型。

hIconSm 小图标的句柄，在任务栏显示的图标，可以和上面的那个一样。

定义一个 **WNDCLASSEX** 类型变量后，在给变量成员初始化后，我们就可以用 **RegisterWindowEx(&wndclassex)**来注册这个窗口类了。

这个注册过程，就和我们平常创建一个项目一样，都要先注册才能创建。

2. 创建窗口

这一步很简单，就是利用 **CreateWindowEx()**函数来创建就是了。

CreateWindowEx 函数的原型如下：

```
HWND CreateWindowEx(
    DWORD    dwExStyle,
    LPCTSTR  lpClassName,
    LPCTSTR  lpWindowName,
    DWORD    dwStyle,
    int      x,
    int      y,
    int      nWidth,
    int      nHeight,
```

```

    HWND    hWndParent,

    HMENU    hMenu,

    HINSTANCE hInstance,

    LPVOID    lpParam

);

```

参数说明

dwExStyle: 指定窗口的扩展风格。该参数可以是下列值:

WS_EX_ACCEPTFILES: 指定以该风格创建的窗口接受一个拖拽文件。

WS_EX_APPWINDOW: 当窗口可见时, 将一个顶层窗口放置到任务条上。

WS_EX_CLIENTEDGE: 指定窗口有一个带阴影的边界。

WS_EX_CONTEXTHELP: 在窗口的标题条包含一个问号标志。

WS_EX_CONTROLPARENT: 允许用户使用 **Tab** 键在窗口的子窗口间搜索。

WS_EX_DLGMODALFRAME: 创建一个带双边的窗口; 该窗口可以在 **dwStyle** 中指定 **WS_CAPTION** 风格来创建一个标题栏。

WS_EX_LEFT: 窗口具有左对齐属性, 这是缺省设置的。

WS_EX_LEFTSCROLLBAR: 如果外壳语言是如 **Hebrew**, **Arabic**, 或其他支持 **reading order alignment** 的语言, 则标题条 (如果存在) 则在客户区的左部分。若是其他语言, 在该风格被忽略并且不作为错误处理。

WS_EX_LTRREADING: 窗口文本以 **LEFT** 到 **RIGHT** (自左向右) 属性的顺序显示。这是缺省设置的。

WS_EX_MDICHILD: 创建一个 MD 子窗口。

WS_EX_NOPATARENTNOTIFY: 指明以这个风格创建的窗口在被创建和销毁时不向父窗口发送 **WM_PARENTNOTIFY** 消息。

WS_EX_OVERLAPPED: **WS_EX_CLIENTEDGE** 和 **WS_EX_WINDOWEDGE** 的组合。

WS_EX_PALETTEWINDOW: **WS_EX_WINDOWEDGE**, **WS_EX_TOOLWINDOW** 和 **WS_EX_TOPMOST** 风格的组合 **WS_EX_RIGHT**: 窗口具有普通的右对齐属性, 这依赖于窗口类。

WS_EX_RIGHTSCROLLBAR: 垂直滚动条在窗口的右边界。这是缺省设置的。

WS_EX_RTLREADING: 如果外壳语言是如 **Hebrew**, **Arabic**, 或其他支持读顺序对齐 (**reading order alignment**) 的语言, 则窗口文本是一自左向右) **RIGHT** 到 **LEFT** 顺序的读出顺序。

WS_EX_STATICEDGE: 为不接受用户输入的项创建一个 3 一维边界风格

WS_EX_TOOLWIDOW: 创建工具窗口, 即窗口是一个游动的工具条。

WS_EX_TOPMOST: 指明以该风格创建的窗口应放置在所有非最高层窗口的上面并且停留在其上, 即使窗口未被激活。使用函数 **SetWindowPos** 来设置和移去这个风格。

WS_EX_TRANSPARENT: 指定以这个风格创建的窗口在窗口下的同属窗口已重画时, 该窗口才可以重画。

由于其下的同属窗口已被重画, 该窗口是透明的。

lpClassName: 窗口类的名字。

lpWindowName:指向一个指定窗口名的空结束的字符串指针。其实就是窗口的名字。

dwStyle:指定创建窗口的风格。该参数可以是下列窗口风格的组合再加上说明部分的控制风格。

x:窗口的横坐标。

y:窗口的竖坐标。

nWidth:窗口的宽度。

nHeight:窗口的高度。

hMenu: 菜单句柄，或依据窗口风格指明一个子窗口标识。

hInstance: 与窗口相关联的模块事例的句柄。

lpParam:指向一个值的指针，该值传递给窗口 WM_CREATE 消息

返回值：如果函数成功，返回值为新窗口的句柄；如果函数失败，返回值为 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

3. 显示窗口

显示窗口就是更简单的事情了。

连个函数轻松搞定，第一个函数就是 ShowWindow ()，原型如下：

```
BOOL ShowWindow(  
  
    HWND hWnd, //当前的窗口句柄  
  
    int nCmdShow //可见状态  
  
);
```

因为 CreateWindowEx 函数创建的窗口是在内存中的，并没有显示到显示器上，用 ShowWindow() 函数，设定窗口的可见状态，并把数据从内存中移动到显卡上，以便显示。

第二个函数就是 UpdateWindow()；

函数原型：

```
BOOL UpdateWindow(HWND hWnd);
```

描述：

这个 UpdateWindow 函数通过发送重绘消息 WM_PAINT 给目标窗体来更新目标窗体客户区的无效区域。如果那个窗体的无效区域没有，就不发送重绘消息 WM_PAINT 了。注意了，这个 API 函数是直接发送消息 WM_PAINT 给目标窗体的，没有进入过消息队列。

函数参数：

hWnd 一个要更新的窗体的句柄

函数返回值：

如果函数调用成功，返回值为非零值。

如果函数调用不成功，返回值为零。

经过这三步后，一个窗口就实现了，就创建了出来，难不，也真够难的，Windows 想的正周到，把创建过程的每一个细节都给想到了，每毫秒可能发生的事情都想到了，难怪 Windows 那么贵，还不开源。也

算是人间的产品嘛，费的心血可真不少呀。说难其实也不难，创建一个窗口程序也就三步：一注册，二创建，三显示。很容易就 **ok** 了。这 **T** 他妈容易了吧。

原来就是这些的，我想我已经说的挺明白的了，如果你有什么疑惑，可以给我发邮件（cangsanbujin@126.com）

下面我们就按照上面所说的来编程实现一个窗口：

```
#include <windows.h>
```

```
//回调函数
```

```
LRESULT WINAPI WinProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
```

```
{
    switch(Msg)//处理消息过程，什么是消息，下节再讲
    {
        case WM_DESTROY://响应鼠标单击关闭按钮事件
            PostQuitMessage(0); //退出消息队列，至于什么是消息队列，下节说
            return 0; //退出函数
    }

    return DefWindowProc(hWnd, Msg, wParam, lParam);
}
```

```
//主函数
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
```

```
{
    char *cName = "myWindow";

    WNDCLASSEX wc;

    HWND hWnd;

    MSG Msg;

    wc.cbClsExtra = 0;

    wc.cbWndExtra = 0;

    wc.cbSize = sizeof(WNDCLASSEX);

    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); //通过函数来设置一个白色的背景，这里大家设置为 NULL 看看，会很有趣的
```



```

wc.hCursor = NULL;//不设置

wc.hIcon = NULL;//不设置

wc.hIconSm = NULL;//不设置

wc.hInstance = hInstance;//当前程序的句柄，hInstance 是有系统给传递的

wc.lpfnWndProc = WinProc;//窗口处理过程的回调函数。

wc.lpszClassName = (LPSTR) cName;//窗口类的名字。

wc.lpszMenuName = NULL;//目录名，不设置

wc.style = CS_HREDRAW | CS_VREDRAW;

RegisterClassEx(&wc);//在系统中注册

hWnd = CreateWindowEx(WS_EX_CLIENTEDGE, cName, "我的窗口我喜欢", WS_OVERLAPPEDWINDOW,
    200, 100, 600, 400, NULL, NULL, hInstance, NULL);//创建窗口，窗口标题为"我的窗口我喜欢"
if(hWnd == NULL)
{
    //容错处理
    MessageBox(NULL, "There's an Error", "Error Title", MB_ICONEXCLAMATION|MB_OK);
    return 0;
}

ShowWindow(hWnd, nShowCmd);//显示窗口

UpdateWindow(hWnd);

//下面是对消息的循环处理，大家先不必管这些，下节课我会细说的
while(GetMessage(&Msg, NULL, 0, 0))
{
    TranslateMessage(&Msg);//翻译消息
    DispatchMessage(&Msg);//分派消息
}

return Msg.message;
}


```

编译运行后，可以看到一个白色背景的窗口出来了。如下图



哎，这一节，篇幅可是真有点长的，看完估计得换几口气吧，但是只要你看到了这些，你的水平就立马上了一个档次。想你看完后也许会头昏脑胀的，没有再看下去的信心的，但是估计当你把我的代码复制到 VC 中编译运行后，看到一个可爱的窗口时，肯定又会重新点燃你心中学习的热情吧，因为你已经看到了成功，看到了成就，一种成就感犹然自心中生，自信也提起来了，这比什么都好，人嘛就得对自己充满信心的。所以大家要发扬持之以恒的精神，坚持和我一起把这段苦闷的入门过程给走完，那么编程就不再是痛苦，而是一种乐趣。其实写这些程序很多东西都不用去记的想 `WNDCLASSEX` 结构的成员及成员作用，这些都不用去死记，只要知道有这么个东西，到时时再查就可以了，编程用到的函数、结构体那么多，谁想记呀。这一节已经留下了些问题，在下节介绍的，大家如果有余力的话，可以先查下资料的。

1.4 鼠标指针特效

大家在都玩过网络游戏吧，里面的界面都是很吸引人的，好的界面的确能给人以美的感受。而里面的鼠标并不是我们平常见到的箭头了，而是独具匠心的。网游我就只玩过魔域，所以就以魔域为例，魔域中的鼠标是这样的。今天我们就来实现让鼠标到程序窗口上就变为我们想要的图案。

在写代码之前，我们还是先看下先驱知识，这里要说的就是上节说资源了，当时大家看了可能并不知道什么是资源，这里就详细说一下。

大家知道 Windows 程序都有图标，鼠标有光标，窗口上有图片、按钮、文字等等，这些都是程序的部分，这样就是程序的资源。程序没有进入内存运行的时候，我们就叫它可执行文件吧，在磁盘保存的时候，并不只是保存了程序运行的代码部分（即 cpu 指令部分），还有一些图片、字符、按钮、图标并不是在代码段的。可执行文件的大致机构如下图



一个可执行文件是很复杂的，这里就简单的画这么一个难看的图，知道资源所在的大概位置，能理解程序的执行部分和知道程序的图标是从哪来的就可以了。

今天我们只是修改鼠标的指针，所以用到的资源，只有鼠标的光标资源而已。资源的源文件是以 rc 为扩展名的脚本文件（仍然是 C 语言格式的，很简单），有资源编译器 Rc.exe 编译成以 res 为扩展名的二进制资源文件，最后用连接器，把 res 文件和 obj 文件连接到一起就成了我们的程序 exe 文件了，现在知道了程序编译后要连接了吧。光标图片格式有两中 cur 和 ani 的。这个文件我在魔域的图片库里面找到了就复制到，当前项目目录下。下面来定义下资源文件 myOwnCursor.rc

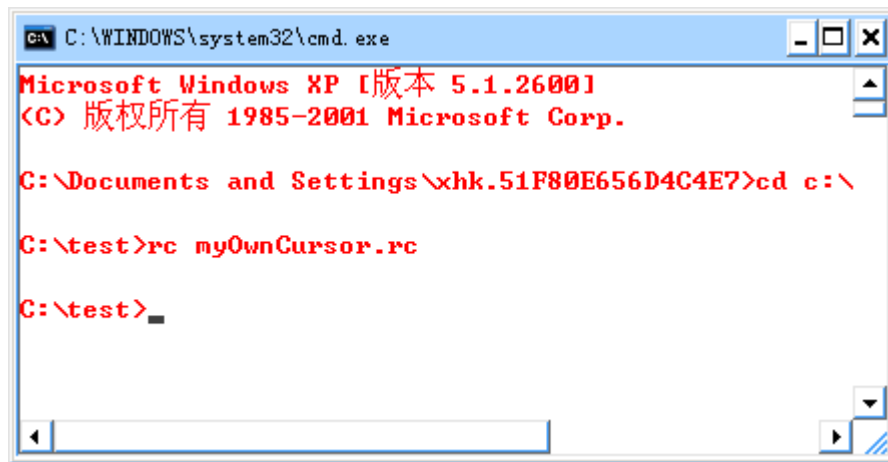
```
//myOwnCursor.rc written by xhk 2009.3.1

#include      <resource.h>  //资源文件要用到的图文件

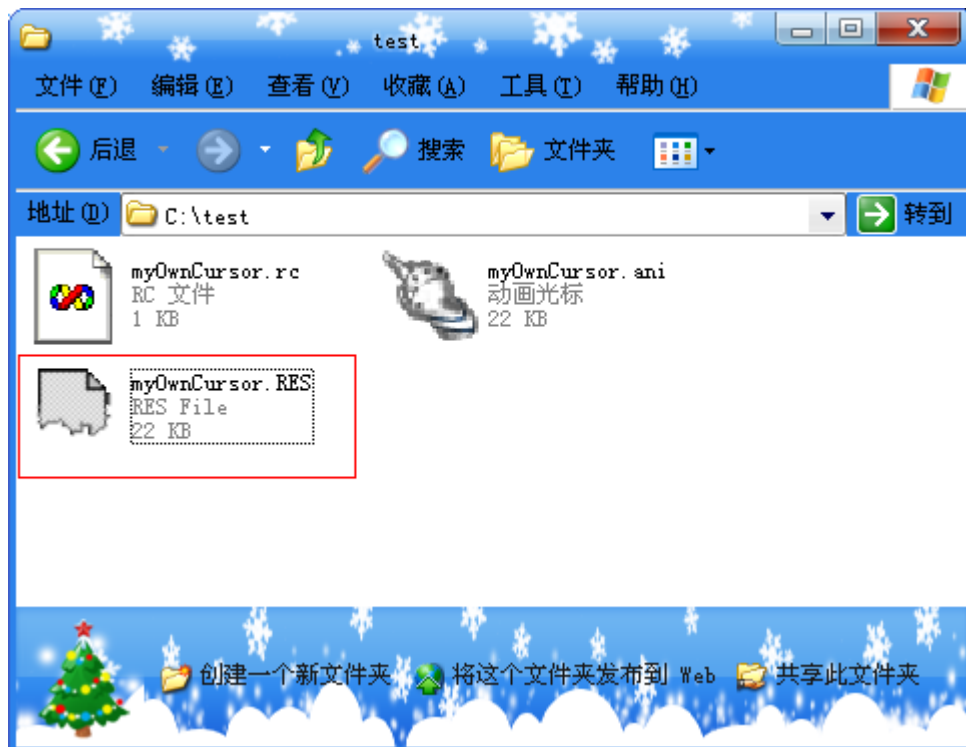
#define      CUR          0x1000          //定义资源的 ID，为整型 id
```

```
CUR CURSOR      "myOwnCursor.ani"    //用到的光标图案
```

写完后，在命令提示符下进入目录，然后用 rc.exe 编译，输入 rc myOwnCursor.rc 命令，回车



我们查看下项目目录下多了个 myOwnCursor.RES 的文件，就是编译生成的二进制资源文件。



接下来就该编写代码了，来应用这个资源文件，建立 myOwnCursor.c 文件，其实代码和上节所写代码很相似的，只是稍微加以修改而已。

```
#include <windows.h>
```

```
#define CUR 0x1000    //预定义光标的 id
```

```
//回调函数
```

```

LRESULT WINAPI WinProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
    switch(Msg)//处理消息过程，什么是消息，下节再讲
    {
        case WM_DESTROY://响应鼠标单击关闭按钮事件
            PostQuitMessage(0); //退出消息队列，至于什么是消息队列，下节说
            return 0;//退出函数
    }

    return DefWindowProc(hWnd, Msg, wParam, lParam);
}

```

//主函数

```

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{

```

```

    char *cName = "myWindow";

```

```

    WNDCLASSEX wc;

```

```

    HWND hWnd;

```

```

    MSG Msg;

```

```

    wc.cbClsExtra = 0;

```

```

    wc.cbWndExtra = 0;

```

```

    wc.cbSize = sizeof(WNDCLASSEX);

```

```

    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); //通过函数来设置一个白色的背景，这里大家

```

设置为 NULL 看看，会很有趣的

```

    wc.hCursor = LoadCursor(hInstance, MAKEINTRESOURCE(CUR)); //这里改了，来载入光标资源

```

```

    wc.hIcon = NULL; //不设置

```

```

    wc.hIconSm = NULL; //不设置

```

```

    wc.hInstance = hInstance; //当前程序的句柄，hInstance 是有系统给传递的

```

```

    wc.lpfnWndProc = WinProc; //窗口处理过程的回调函数。

```

```

    wc.lpszClassName = (LPSTR)cName; //窗口类的名字。

```

```

    wc.lpszMenuName = NULL; //目录名，不设置

```

```

    wc.style = CS_HREDRAW | CS_VREDRAW;

```

```

RegisterClassEx(&wc); //在系统中注册

hWnd = CreateWindowEx(WS_EX_CLIENTEDGE, cName, "我的窗口我喜欢", WS_OVERLAPPEDWINDOW,
    200, 100, 600, 400, NULL, NULL, hInstance, NULL); //创建窗口，窗口标题为"我的窗口我喜欢"
if(hWnd == NULL)
{
    //容错处理
    MessageBox(NULL, "There's an Error", "Error Title", MB_ICONEXCLAMATION|MB_OK);
    return 0;
}

ShowWindow(hWnd, nShowCmd); //显示窗口

UpdateWindow(hWnd);

//下面是对消息的循环处理，大家先不必管这些，下节课我会细说的

while(GetMessage(&Msg, NULL, 0, 0))
{
    TranslateMessage(&Msg); //翻译消息
    DispatchMessage(&Msg); //分派消息
}

return Msg.message;
}

```

用 vc 编译生成 myOwnCursor.obj, 把 myOwnCursor.obj 和 myOwnCursor.res 放到同一个文件夹下，然后在命令行下进入它们所在的目录，输入命令：link kernel32.lib user32.lib gdi32.lib /subsystem:windows myOwnCursor.obj myOwnCursor.res 把两个文件连接成 myOwnCursor.exe. 运行后界面如下；



看到了吧，当鼠标移入窗口的时候，光标就变成了那个手型图案了，这和魔域的是一样的。到现在想想一个特效又咋地，不还是一句一句代码写出来的，而特效和普通程序往往只有数据代码不同而已。网络游戏的界面很好看，也只不过是资源文件用的比较多而已，而且计算量很大，所以网游总是很占内存的，因为图片、声音文件都很大，而且变换比较多、快，就比较占用资源了。

其实再好的程序，只要有了思路，就能写出来，而且写出来也难的，是不是，今天大家应该会有点收获了，都会设计个性的鼠标光标了，比起以前学习 C 的东西，应该有一种层次感了吧。这些东西都比较接近系统了，所以学了之后，你对 Windows 系统也会有很深的了解的。如果各位看官看到本节还有兴趣继续看下去，那么这对小人就是一种支持，小人在此谢过了；如果看官觉得看这些没有半点收获，那么请看官不要再勉强自己看下去了，免得浪费看官大人的宝贵时间，那是小人所承担不起的。

总之了，要想写好程序，就得多练，编译连接过程中很容易发现错误的存在，那么这时你解决一个错误你就提高一次，解决的错误越多越快，你就学的越多越快。终于后来你会发现，你太难找到错误了，那么恭喜你，你已经升级为大虾了，已经完全脱离了菜菜级了。希望大家继续努力！

1.5 在窗口上写上“Hello World”

这一节我们乘胜追击，来继续深入学习下，学习窗口处理时间的东东。

也许你以前听说过，windows 系统是消息驱动的，可是可能根本就不知道什么消息，更不知道什么消息驱动了。其实什么是消息呢，说白了就是我们点鼠标击键盘而程序发生反应，消息是一种数据，就是我们点鼠标击键盘后，系统把我们的操作封装到数据中，然后发送给程序，让程序对我们点鼠标击键盘的动作做出反应，当然程序也可以置之

不理。Windows 可是一个多任务的系统，而且同时可能产生很多的击键动作，那么同时可能能会有很多消息，windows 系统为了更好的管理维护这些消息，就把这些消息加入消息队列中，消息队列其实就是消息的集合。

学过 VB 的人知道，VB 中的程序是事件驱动的，因为一般都是发生时，调用相应的事件处理函数，所以整个处理过程都好像是事件引发的一样。这里的事件就是指我们击键的动作等。

学过 JAVA 的人知道，JAVA 中有事件适配器，来捕获相应的事件，并交给相应的处理方法进行处理。

其实三种语言的处理过程也都是大同小异，只不过 JAVA 和 VB 把这些处理过程给封装了，VB 尤其封装的更厉害，所以编程者不必考虑和知道这中间的细节问题，仍然可以编写出实用的程序，但正是由于细节的原因，用 VB 的开发的程序并不能高效地处理问题。

而 C 语言本身就是面向过程的语言，所以这一过程可以用 C 语言更好地表现出来，这也是我用 C 而不用 C++ 的原因之一。

通过前几节的学习，我们知道了，在窗口程序中都有一个处理窗口的函数，其实所有的消息将会得到怎样的处理，都是此函数安排的。现在大家再回去看看那个程序代码和注释，相信应该能明白些了吧。

系统产生的消息是不断的，但是中间是有间隔的，程序要想知道有没有自己的消息，得不停地去问系统，问系统当前有没有属于自己的消息，这就需要有一个循环来实现了。

下面先看下前两节种用到的消息循环代码：

```
while (GetMessage (&Msg, NULL, 0, 0))
{
    TranslateMessage (&Msg); //翻译消息
    DispatchMessage (&Msg); //分派消息
}
```

Windows 为消息定义一种新的数据类型 MSG，用于保存消息的相关信息。在 windows 中 GetMessage 函数从消息队列种取得消息，填写好 MSG 结构并返回，如果获取的消息是 WM_QUIT 消息，则退出循环。

TranslateMessage 将 MSG 结构传给 Windows 进行一些键盘消息的转换，当有键盘按下或者放开时，Windows 产生 WM_KEYDOWN 和 WM_KEYUP 或 WM_SYSKEYDOWN 和 WM_SYSKEYUP 消息（像 WM_KEYDOWN 这些都是微软定义的一些宏，是什么意思，看字面意思就可以知道了），但是这些消息的参数种包含的是按键的扫描码（暂时不用理会），转换成常用的 ASCII 码要经过查表，很不方便，TranslatMessage 遇到键盘消息则将扫描码转换成 ASCII 码并在消息队列种插入 WM_CHAR 或 WM_SYSCHAR 消息，参数就是转换好的 ASCII 码，如此一来，要处理键盘消息的话只要是处理 WM_CHAR 消息就好了。菊与刀非键盘消息 TranslateMessage 则不做处理。

最后，由 DispatchMessage 将消息发送到窗口对应的窗口过程去处理。窗口过程返回后 DispatchMessage 函数才返回，然后开始新一轮的消息循环。

想想我们这节的目的是为了在洁白的窗口种写下“Hello World”，那么我们怎么来留下我们的笔迹呢？窗口我们是能做出来了，那么怎么在上面写东西呢，等等，在上面写东西的前提是不是窗口做出来之后，当初我是这么想的，

后来看到别人的代码才知道原来可以在窗口绘制过程就绘制“Hello world”了。Windows 有时真是个细心的家伙，把窗口创建到显示的一瞬间又给划分了很多小的过程。在绘制窗口时，Windows 会产生 WM_PAINT 消息，那么我们在得到这个消息的时候，来留下我们的笔迹，岂不就是下手最早的时刻。其实 Windows 在屏幕上输出文字和图像是一样的，都是在屏幕上画，和我们在纸上画图 and 写字是一样的，都是用笔来画的，只不过用的笔不一样而已，如果勉强用一支笔去做所有的工作，效果并不会理想的。Windows 的笔也是这样的，不过这些笔是函数而已，画图和画文字的函数不一样而已。

下面就接着上节修改的代码继续修改，必要的注释和改变的地方我会标明的

```
#include <windows.h>
```

```
#define CUR 0x1000          //预定义光标的 id
```

```
HDC hDC;//HDC 是指设备上下文（暂时不用管，只要能这样用就可以了）的句柄
```

```
//PAINTSTRUCT 要绘制的信息，详情请登陆 http://msdn.microsoft.com/en-us/library/dd162768\(VS.85\).aspx
```

```
//了解下就可以了，没什么重要的东西
```

```
PAINTSTRUCT paint;
```

```
RECT rect;//RECT 用来存储窗口信息的结构，只要是窗口的坐标、宽度和高度。
```

```
//回调函数
```

```
LRESULT WINAPI WinProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
    switch(Msg)//处理消息过程，什么是消息，下节再讲
```

```
    {
```

```
        case WM_PAINT:
```

```
            //BeginPaint 做些绘画的开始工作，填充 PAINTSTRUCT 结构，返回设备上下文（暂时不用理解）句柄
```

```
            hDC=BeginPaint(hWnd, &paint);
```

```
            //GetClientRect 用来获取窗口所在客户区的位置大小信息
```

```
            GetClientRect(hWnd, &rect);
```

```
            //DrawText 就是 Windows 用来“画字”的笔了，DT_*之类是指文字的样式，看字面意思也能看懂的
```

```
            //有多少样式呢，可以查看这里 http://msdn.microsoft.com/en-us/library/ms901121.aspx
```

```
            //本例中是单线、水平居中和竖直居中。
```

```
            DrawText(hDC, "Hello World!", -1, &rect, DT_SINGLELINE|DT_CENTER|DT_VCENTER);
```

```

        //EndPaint 就是做些收尾的工作了。

        EndPaint(hWnd, &paint);

        break;

case WM_DESTROY://响应鼠标单击关闭按钮事件

    PostQuitMessage(0); //退出消息队列，至于什么是消息队列，下节说

    return 0; //退出函数

}

return DefWindowProc(hWnd, Msg, wParam, lParam);

}

//主函数

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{

    char *cName = "myWindow";

    WNDCLASSEX wc;

    HWND hWnd;

    MSG Msg;

    wc.cbClsExtra = 0;

    wc.cbWndExtra = 0;

    wc.cbSize = sizeof(WNDCLASSEX);

    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); //通过函数来设置一个白色的背景，这里大家
    设置为 NULL 看看，会很有趣的

    wc.hCursor = LoadCursor(hInstance, MAKEINTRESOURCE(CUR)); //这里改了，来载入光标资源

    wc.hIcon = NULL; //不设置

    wc.hIconSm = NULL; //不设置

    wc.hInstance = hInstance; //当前程序的句柄，hInstance 是有系统给传递的

    wc.lpfnWndProc = WinProc; //窗口处理过程的回调函数。

    wc.lpszClassName = (LPSTR)cName; //窗口类的名字。

    wc.lpszMenuName = NULL; //目录名，不设置

    wc.style = CS_HREDRAW | CS_VREDRAW;

```

```

RegisterClassEx(&wc); //在系统中注册

hWnd = CreateWindowEx(WS_EX_CLIENTEDGE, cName, "我的窗口我喜欢", WS_OVERLAPPEDWINDOW,
    200, 100, 600, 400, NULL, NULL, hInstance, NULL); //创建窗口，窗口标题为"我的窗口我喜欢"

if(hWnd == NULL)
{
    //容错处理
    MessageBox(NULL, "There's an Error", "Error Title", MB_ICONEXCLAMATION|MB_OK);
    return 0;
}

ShowWindow(hWnd, nShowCmd); //显示窗口

UpdateWindow(hWnd);

//下面是对消息的循环处理，大家先不必管这些，下节课我会细说的

while(GetMessage(&Msg, NULL, 0, 0))
{
    TranslateMessage(&Msg); //翻译消息
    DispatchMessage(&Msg); //分派消息
}

return Msg.message;
}

```

最终运行结果：



其实就是比原来的多了三个变量和几句代码，多的我也标出来了，而且都说明，那些简单的函数，大家可以自己查下，很简单的，我就不再为一些简单的函数来打字了，这样也可以锻炼大家的动手能力。

编译连接后，大家看看预期的结果出现了吧，洁白的窗口上留下了我们的字迹。建议学过 VB 或 JAVA 的读者，可以联系起来想一想，把 C 的处理消息过程给理解下，理解下消息的结构和概念，熟悉 Windows 的消息机制，这样就可以为以后编写优质的软件打下坚实的基础。此言不虚的，像金山词霸的屏幕取词功能就是对 Windows 消息巧妙的运用；键盘记录器（木马）也是利用了截获 Windows 消息，而记录我们的按键行为，从而盗取信息的。大家好好理解下本节内容，自己动手写点东西，查些其他的事件信息，改进下程序，多熟练下，为后面的学习做一点铺垫。

1.6 让窗口响应鼠标的事件

为了让大家能够多熟悉下事件和消息的概念，本节再以一个小的例子看下鼠标事件的应用。鼠标的事件有单击、右击、双击和滚动轮的，我们这里先让鼠标响应两种事件：单击和右击。我们在实现在窗口上单击时弹出一个上面有“你击了左键”的对话框，右击时弹出一个上面有“你击了右键”的对话框。代码仍用上节的，只在窗口处理过程的，消息处理语句（switch）中加入一下代码：

```
case WM_LBUTTONDOWN: // 鼠标左键松开时  
    MessageBox(hWnd, "你击了左键", "提示", MB_OK);
```

```
break;

case WM_RBUTTONDOWN://鼠标右键松开时

    MessageBox(hWnd, "你击了右键", "提示", MB_OK);

    break;
```

编译运行，单击左键如下图



当然，至于是弹出对话框还是干别的什么，你可以自己添加代码的。不管怎样，我想通过这个例子，你应该理解了程序是怎么处理鼠标的单击和右击了吧，应该对消息驱动有了更好一点理解吧。自己多写写代码，多查查资料成就很快的。就在写这个实例的时候，因为 VC 6.0 的 MFC 中定义的消息和 API 中的不太一样，一时忘了 API 中鼠标事件的宏是怎么写的，我查了 MSDN、百度和谷歌，竟然没有查出来（真是岂有此理），最后，我就只有自己解决了，硬是在 winuser.h 的头文件中找到鼠标事件消息的宏定义的。真个过程中有一种山穷水尽疑无路，柳暗花明又一村的感觉，就在我快要放弃的时候，想起来了用基本的方法，直接查看头文件的定义，真可谓天才，然是最笨的方法了。不过这样也好，一下子看了很多消息的宏定义。大家学习一定要自己多查多练习，相信聪明的你一定会轻松解决遇到的问题。想我这么笨的人，都学会了用 C 写 Windows 程序，又何况聪明的你呢。

1.7 单击鼠标来改变窗口的位置

目的还是为了大家进一步熟悉 Windows 的窗口实现消息的机制，也使大家了解多一点的 Windows API 函数，从而利于日后的实际编程。平常我们都是用鼠标拖着窗口来改变窗口的，今天我们来点新鲜的，通过单击鼠标来使窗口改变位置。

从前面的知识中，我们知道，窗口的初始位置是在 CreateWindow 函数中设定的，Windows 既然可以让用户通过鼠标拖来改变窗口位置，那么肯定就有函数是专门用来改变窗口位置的。是的，的确有这样的函数，常用的有两个，它们是 SetWindowPos 和 MoveWindow。两个函数的详细情况如下：

SetWindowPos

函数功能：该函数改变一个子窗口，弹出式窗口或顶层窗口的尺寸，位置和 Z 序。子窗口，弹出式窗口，及顶层窗口根据它们在屏幕上出现的顺序排序、顶层窗口设置的级别最高，并且被设置为 Z 序的第一个窗口。

函数原型：BOOL SetWindowPos (HWND hWnd, HWND hWndInsertAfter,int X, int Y,int cx, int cy,UNIT. Flags)；

参数：

hWnd:窗口句柄。

hWndInsertAfter:在 z 序中的位于被置位的窗口前的窗口句柄。该参数必须为一个窗口句柄，或下列值之一：

HWND_BOTTOM:将窗口置于 Z 序的底部。如果参数 hWnd 标识了一个顶层窗口，则窗口失去顶级位置，并且被置在其他窗口的底部。

HWND_NOTOPMOST:将窗口置于所有非顶层窗口之上（即在所有顶层窗口之后）。如果窗口已经是非顶层窗口则该标志不起作用。

HWND_TOP:将窗口置于 Z 序的顶部。

HWND_TOPMOST:将窗口置于所有非顶层窗口之上。即使窗口未被激活窗口也将保持顶级位置。

查看该参数的使用方法，请看说明部分。

x:以客户坐标指定窗口新位置的左边界。

Y:以客户坐标指定窗口新位置的顶边界。

cx:以像素指定窗口的新的宽度。

cy:以像素指定窗口的新的高度。

uFlags:窗口尺寸和定位的标志。该参数可以是下列值的组合：

SWP_ASYNCWINDOWPOS:如果调用进程不拥有窗口，系统会向拥有窗口的线程发出需求。这就防止调用线程在其他线程处理需求的时候发生死锁。

SWP_DEFERERASE:防止产生 WM_SYNCPAINT 消息。

SWP_DRAWFRAME:在窗口周围画一个边框（定义在窗口类描述中）。

SWP_FRAMECHANGED:给窗口发送 WM_NCCALCSIZE 消息，即使窗口尺寸没有改变也会发送该消息。如果未指定这个标志，只有在改变了窗口尺寸时才发送 WM_NCCALCSIZE。

SWP_HIDEWINDOW:隐藏窗口。

SWP_NOACTIVATE:不激活窗口。如果未设置标志，则窗口被激活，并被设置到其他最高级窗口或非最高级组的顶部（根据参数 hWndInsertAfter 设置）。

SWP_NOCOPYBITS: 清除客户区的所有内容。如果未设置该标志，客户区的有效内容被保存并且在窗口尺寸更新和重定位后拷贝回客户区。

SWP_NOMOVE: 维持当前位置（忽略 X 和 Y 参数）。

SWP_NOOWNERZORDER: 不改变 z 序中的所有者窗口的位置。

SWP_NOREDRA: 不重画改变的内容。如果设置了这个标志，则不发生任何重画动作。适用于客户区和非客户区（包括标题栏和滚动条）和任何由于窗口移动而露出的父窗口的所有部分。如果设置了这个标志，应用程序必须明确地使窗口无效并重新重画窗口的任何部分和父窗口需要重画的部分。

SWP_NOREPOSITION: 与 SWP_NOOWNERZORDER 标志相同。

SWP_NOSENDCHANGING: 防止窗口接收 WM_WINDOWPOSCHANGING 消息。

SWP_NOSIZE: 维持当前尺寸（忽略 cx 和 cy 参数）。

SWP_NOZORDER: 维持当前 Z 序（忽略 hWndInsertAfter 参数）。

SWP_SHOWWINDOW: 显示窗口。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误消息，请调用 GetLastError 函数。

备注：如果设置了 SWP_SHOWWINDOW 和 SWP_HIDEWINDOW 标志，则窗口不能被移动和改变大小。如果使用 SetWindowLong 改变了窗口的某些数据，则必须调用函数 SetWindowPos 来作真正的改变。使用下列的组合标志：SWP_NOMOVE|SWP_NOSIZE|SWP_FRAMECHANGED。

有两种方法将窗口设为最顶层窗口：一种是将参数 hWndInsertAfter 设置为 HWND_TOPMOST 并确保没有设置 SWP_NOZORDER 标志；另一种是设置窗口在 Z 序中的位置以使其在其他存在的窗口之上。当一个窗口被置为最顶层窗口时，属于它的所有窗口均为最顶层窗口，而它的所有者的 z 序并不改变。

如果 HWND_TOPMOST 和 HWND_NOTOPMOST 标志均未指定，即应用程序要求窗口在激活的同时改变其在 Z 序中的位置时，在参数 hWndInsertAfter 中指定的值只有在下列条件中才使用：

在 hWndInsertAfter 参数中没有设定 HWND_NOTOPMOST 和 HWND_TOPMOST 标志。

由 hWnd 参数标识的窗口不是激活窗口。

如果未将一个非激活窗口设定到 z 序的顶端，应用程序不能激活该窗口。应用程序可以无任何限制地改变被激活窗口在 Z 序中的位置，或激活一个窗口并将其移到最高级窗口的顶部或非最高级窗口的顶部。

如果一个顶层窗口被重定位到 z 序的底部（HWND_BOTTOM）或在任何非最高序的窗口之后，该窗口就不再是最顶层窗口。当一个最顶层窗口被置为非最顶级，则它的所有者窗口和所有者窗口均为非最顶层窗口。

一个非最顶端窗口可以拥有一个最顶端窗口，但反之则不可以。任何属于顶层窗口的窗口（例如一个对话框）本身就被置为顶层窗口，以确保所有被属窗口都在它们的所有者之上。

如果应用程序不在前台，但应该位于前台，就应调用 SetForegroundWindow 函数来设置。

Windows CE: 如果这是一个可见的顶层窗口，并且未指定 SWP_NOACTIVATE 标志，则这个函数将激活窗口、如果这是当前的激活窗口，并且指定了 SWP_NOACTIVATE 或 SWP_HIDEWINDOW 标志，则激活另外一个可见的顶层窗口。

当在这个函数中的 nFlags 参数里指定了 SWP_FRAMECHANGED 标志时，WindowsCE 重画窗口的整个非客户区，这可能会改变客户区的大小。这也是重新计算客户区的唯一途径，也是通过调用 SetWindowLong 函数改变窗口风格后通常使用的方法。

SetWindowPos 将使 **WM_WINDOWPOSCHANGED** 消息向窗口发送，在这个消息中传递的标志与传递给函数的相同。这个函数不传递其他消息。

MoveWindow

函数功能：该函数改变指定窗口的位置和尺寸。对于顶层窗口，位置和尺寸是相对于屏幕的左上角的；对于子窗口，位置和尺寸是相对于父窗口客户区的左上角坐标的。

函数原型：`BOOL MoveWindow (HWND hWnd,int x,int y,int nWidth,int nHeight,BOOL bRepaint) ;`

参数：

hWnd：窗口句柄。

x：指定窗口的新的位置的左边界。

Y：指定窗口的新的位置的顶部边界。

nWidth：指定窗口的新的宽度。

nHaight：指定窗口的新的高度。

bRepaint:确定窗口是否被刷新。如果该参数为 **TRUE**，窗口接收一个 **WM_PAINT** 消息；如果参数为 **FALSE**，不发生任何刷新动作。它适用于客户区，非客户区（包括标题栏和滚动条），及由于移动子窗口而露出的父窗口的区域。如果参数为 **FALSE**，应用程序就必须明确地使窗口无效或重画该窗口和需要刷新的父窗口。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 **GetLastError** 函数。

备注：如果 **bRepaint** 为 **TRUE**，系统在窗口移动后立即给窗口过程发送 **WM_PAINT** 消息（即由 **MoveWindow** 函数调用 **UpdateWindow** 函数）。如果 **bRepaint** 为 **FALSE**，系统将 **WM_PAINT** 消息放在该窗口的消息队列中。消息循环只有在派遣完消息队列中的其他消息时才派遣 **WM_PAINT** 消息。

MoveWindow 给窗口发送 **WM_WfNOWPOSCHANGING**, **WM_WINDOWPOSCHANGED**, **WM_MOVE**, **WM_SIZE** 和 **WM_NCCALCSIZE** 消息。

以上的东西，都是从 **msdn** 上翻译过来的，把它们翻译过来，是在有故意添文字之嫌。看了函数说明就好的多了吧，我们只把上节中的代码稍加修改即可，我这里给出我的代码，大家可以借鉴下（我觉得知道这两个函数怎么用，真是没什么要说的了）：

```
#include <windows.h>
```

```
#define CUR 0x1000    //预定义光标的 id
```

```
HDC hDC;//HDC 是指设备上下文（暂时不用管，只要能这样用就可以了）的句柄
```

```
//PAINTSTRUCT 要绘制的信息
```

```
//详情请登陆 http://msdn.microsoft.com/en-us/library/dd162768\(VS.85\).aspx
```

```
//了解下就可以了，没什么重要的东西
```

```
PAINTSTRUCT paint;
```

```
RECT rect;//RECT 用来存储窗口信息的结构，只要是窗口的坐标、宽度和高度。
```



```
//自定义函数 MoveLeft,使窗口向左移动 5 像素,此函数中调用 MoveWindow 函数
int MoveLeft(HWND hWnd)
{
    GetWindowRect(hWnd,&rect);//获取窗口的信息
    MoveWindow(hWnd,rect.left-5,rect.top,rect.right-rect.left,rect.bottom-rect.top,TRUE);
    return 1;
}
```

```
//自定义函数 MoveRight,是窗口向右移动 5 像素,此函数中调用 SetWindowPos 函数 (换个口味)
int MoveRight(HWND hWnd)
{
    GetWindowRect(hWnd,&rect);//获取窗口的信息
    SetWindowPos(hWnd,HWND_NOTOPMOST,
        rect.left+5,rect.top,
        rect.right-rect.left,
        rect.bottom-rect.top,
        SWP_NOZORDER);

    return 1;
}
```

//回调函数

```
LRESULT WINAPI WinProc(HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam)
{
```

```
    switch(Msg)//处理消息过程,什么是消息,下节再讲
    {
```

```
case WM_PAINT:
```

```
    //BginPaint 做些绘画的开始工作,填充 PAINTSTRUCT 结构,返回设备上下文(暂时不用理解)句柄
```

```
    hDC=BeginPaint(hWnd,&paint);
```

```
    //GetClientRect 用来获取窗口所在客户区的位置大小信息
```

```
    GetClientRect(hWnd,&rect);
```

```
    //DrawText 就是 Windows 用来“画字”的笔了,DT_*之类是指文字的样式,看字面意思也能看懂
```

```
    //有多少样式呢,可以查看这里 http://msdn.microsoft.com/en-us/library/ms901121.aspx
```

```
    //本例中是单线、水平居中和垂直居中。
```

```
    DrawText(hDC,"Hello World!",-1,&rect,DT_SINGLELINE|DT_CENTER|DT_VCENTER);
```

```
    //EndPaint 就是做些收尾的工作了。
```

```
    EndPaint(hWnd,&paint);
```

柄

的

```

        break;
    case WM_LBUTTONDOWN://鼠标左键松开时
        MoveLeft(hWnd);
        break;
    case WM_RBUTTONDOWN://鼠标右键松开时
        MoveRight(hWnd);
        break;
    case WM_DESTROY://响应鼠标单击关闭按钮事件
        PostQuitMessage(0);//退出消息队列，至于什么是消息队列，下节说
        return 0;//退出函数
    }
    return DefWindowProc(hWnd,Msg,wParam,lParam);
}

//主函数
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int
nShowCmd)
{
    char *cName = "myWindow";
    WNDCLASSEX wc;
    HWND hWnd;
    MSG Msg;

    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);//通过函数来设置一个白色的
背景，这里大家设置为 NULL 看看，会很有趣的
    wc.hCursor = LoadCursor(hInstance,MAKEINTRESOURCE(CUR));//这里改了，来载入光标资源
    wc.hIcon = NULL;//不设置
    wc.hIconSm = NULL;//不设置
    wc.hInstance = hInstance;//当前程序的句柄，hInstance 是有系统给传递的
    wc.lpfnWndProc = WinProc;//窗口处理过程的回调函数。
    wc.lpszClassName =(LPSTR)cName;//窗口类的名字。
    wc.lpszMenuName = NULL;//目录名，不设置
    wc.style = CS_HREDRAW | CS_VREDRAW;

    RegisterClassEx(&wc);//在系统中注册

```

```

        hWnd = CreateWindowEx(WS_EX_CLIENTEDGE,cName,"我的窗口我喜欢",
        WS_OVERLAPPEDWINDOW,
        200,100,600,400,NULL,NULL,hInstance,NULL);//创建窗口，窗口标题为"我的窗口我喜欢"
        if(hWnd == NULL)
        { //容错处理
            MessageBox(NULL,"There's an Error","Error Title",MB_ICONEXCLAMATION|MB_OK);
            return 0;
        }
        ShowWindow(hWnd,nShowCmd);//显示窗口
        UpdateWindow(hWnd);

        //下面是对消息的循环处理，大家先不必管这些，下节课我会细说的
        while(GetMessage(&Msg,NULL,0,0))
        {
            TranslateMessage(&Msg);//翻译消息
            DispatchMessage(&Msg);//分派消息
        }
        return Msg.message;
    }
}

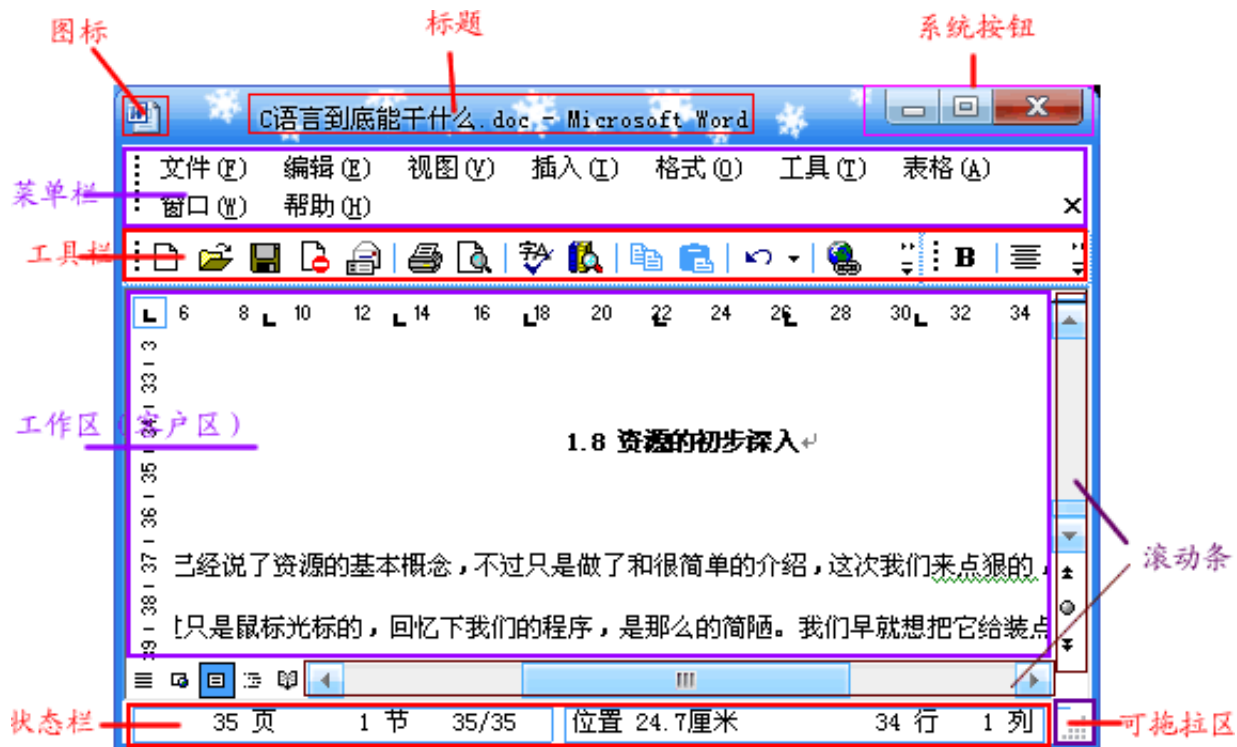
```

这个结果，不能用静态的图片来说明什么，大家自己编译连接后，击鼠标试试，看能出现预期的效果不，我这里就不贴图出来了，贴静态的没啥意思，贴动态的，有点不想整，我也懒，哈。

经过这几次折腾，如果大家真的每一次都手写了，相信其中的那关键的且相同的那部分代码应该是非常熟悉了，到此就我们就该升级了，就行高一层次的修炼，后面两节，我准备给大家说些资源的深入细节，还有再在写几个完全实用的小程序和几个恶作剧程序，不知大家意下如何。

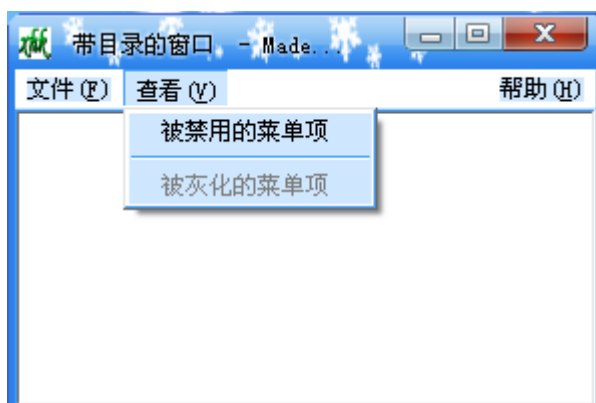
1.8 资源的初步深入

前面已经说了资源的基本概念，不过只是做了和很简单的介绍，这次我们来点狠的，深入的。前面我虽然也用了资源，不过只是鼠标光标的，回忆下我们的程序，是那么的简陋。我们早就想把它给装点下了吧，不用着急，学完了这节后，你就可以成为一个雕刻师了，想让你的窗口咋样基本都可以了（需要练习了，呵呵）。



以当前我这个 Word 编辑窗口为例，可以看到一个窗口有很多项的，而我们之前的串口跟这个相比，真可谓小巫见大巫。前面的程序连最起码的菜单栏都没有，真是惭愧呀。

在 Vb 做界面，简直就跟画图是一样一样的，Java 中可以在编程时，一个一个组件往窗体对象 (JFrame) 上画（也许有 IDE 可以手画的），VC 中呢，也可以画，但是注意的细节明显比 VB 要多。其实手画的过程，只是程序帮了我们，帮我们写了资源文件。这和用网页设计工具是一样的，我们只顾点鼠标，代码则是网页设计工具生成的了。同样，其他编程也是这样，这样的好处是：一可以让初学者很容易进入状态，二是可以加快开发，可以少写 n 拖代码。坏处是：不懂得底层机制，很多人写了 n 久的程序，也只能是比葫芦画瓢，写的程序界面还是自己学习时候的那种样式，界面单调死板，开发不出个性界面的。鉴于工具带来的负面影响，我才给大家从基本说起，虽然我们是用 VC 6.0 的环境，但是我还是手写资源来教大家定义资源文件，并不利用 VC 中 IDE 工具。如果大家资源文件写的很熟练的话，再用 VC 中的 IDE 工具，不用去看多余的书，自然一看就知道是怎么回事，到时用起来就是得心应手。说实话，如果不理解 Windows 的一些处理机制，上去直接去学习 VC，我敢肯定学一段时间后，大部分人会头昏脑胀，事倍功半，虽有收获，然仍是皮毛，有放弃之想。好了废话不多说了，言归正传。



如上图，是我这节要实现的效果，上面有菜单栏，其中点击“查看”可以菜单子菜单项，弹出的有禁用的菜单、分割线和灰化的菜单项。还有一个我自己做的图标（xhk 字样的，左上角）。单击标题栏上的图标可以弹出系统菜单，在有的程序，在窗口中击鼠标右键，就可以弹出“快捷菜单”，这些菜单都属于弹出式菜单。

菜单中的菜单项有好几种，从资源定义的角度来看，分割用的横线也是一个菜单项。除横线外其他菜单项可以供用户选择，也可以设置为“禁止”或“灰化”状态暂时停用，如果上图的。

快捷键，这个不用说了，大家都知道是做什么用的。菜单项显示的字符都是在资源文件中定义，至于如何来响应按键则要在消息处理函数中添写代码了，本节先不讨论怎样获取这些消息和处理这些消息，这写留到下节中完成，本节先常用资源的定义格式说下，先完成界面上的东东。

1. 菜单资源的定义

在资源脚本文件菜单中的定义格式是这样的：

```
菜单          ID MENU [DISCARDABLE]

BEGIN

    菜单项的定义

END
```

也可以这样定义：

```
菜单          ID MENU [DISCARDABLE]

{

    菜单项的定义

}
```

“菜单 ID MENU [DISCARDABLE]”可以用来制定菜单的 ID 值和内存属性，菜单 ID 可以是 16 位（二进制位）的整数，也可以是字符串。但是如果 ID 位字符串的话，在程序中引用的时候就要用字符串指针代替菜单 ID 值，显然这样不太方便，所以在我们经常用整数来做菜单的 ID 值。MENU 关键词后面的 DISCARDABLE 是菜单的内存属性，表示菜单在不再使用的时候可以暂时从内存中释放以节省内存，是个可选属性。菜单项的定义必须在 BEGIN 和 END 关键词之内，这两个关键词也可以用 {和} 来代替。

菜单项目的定义方法有三类：

1. 常用的

```
MENUITEM 菜单文字, 命令 ID [, 选项列表]
```

2. 分割线

```
MENUITEM SEPARATOR
```

3. 下级菜单

和菜单定义的方式一样

```
POPUP          菜单文字  [, 选项列表]
```

BEGIN

Item-definitions

END

下面对这三类加以说明

第一类:

菜单文字——显示在菜单项中的字符串。像上图中的“被禁用的菜单项”和“被灰化的菜单项”。

命令 ID——不同菜单项的标识。当菜单被选中的时候，Windows 会向窗口过程发送 WM_COMMAND 消息，消息的参数就是这个命令 ID。这个可以分辨用户选中了哪个菜单项，如果想让两个菜单项具有相同的功能，可以设置为相同的 ID。

选项列表——用来形容菜单项的各种属性，它可以是下列选项：

CHECKED——表示打上选定标识。

GRAYED——表示菜单项是灰化的。

INACTIVE——表示菜单项是禁用的。

MENUBREAK 或 MENUBARBREAK——表示将这个菜单项和以后的那个列到新的列中。

第二类:

菜单项之间的分割线，没什么好说的了。

第三类:

弹出式菜单，前文有解释，这里说下它的选项：

GRAYED——灰化。

INACTIV——禁用。

HELP——表示本项和以后的菜单项是右对齐的，像上图中的“帮助”菜单。

2. 快捷键的定义

快捷键定义是很简单的，格式如下：

快捷键 ID ACCELERATORS

BEGIN

键名, 命令 ID[, 类型][, 选项]

END

BEGIN 和 END 仍然可以用 {和} 替换。

键名——表示加速键对应的按键，可以有 3 中定义方式：

“`字母”：表示 Ctrl 键加上字母键。

“字母”：表示字母，这时类型必须指明 VIRTKEY。

数值：表示 ASCII 码，这时类型必须为 ASCII

命令 ID——按下快捷键后，Windows 就向程序发送此命令 ID。

类型——用来指定键的定义方式，可以是 VIRTKEY 和 ASCII，分别用来表示“键名”字段定义的是虚拟键还是 ASCII 码。

选项——可以使 Alt, Control 或 Shift 中的单个或多个，如果指定多个，则中间用逗号隔开，表示快捷键是按键加上这些控制键的组合键。

说了这么多，考验我们的时候终于到了，下面我们就来写程序了。

兵马未动，粮草先行，我们先来把界面定义好，定义一个 MyMenu.rc 的资源文件，内容如下：

```
/******MyMenu.rc Written By XHK 2009.3.3******/
```

```
#include <resource.h>
```

```
#define ICO_MAIN 0X1000    //图标
```

```
#define IDM_MAIN 0X2000    //菜单
```

```
#define IDA_MAIN 0X2000    //快捷键
```

```
#define IDM_OPEN 0X4101    //“打开”菜单项
```

```
#define IDM_INACTIVE      0X4201    //“被禁用的菜单项”
```

```
#define IDM_GRAYED 0X4202    //“灰化的菜单项”
```

```
#define IDM_HELP 0X4301    //“帮助”菜单项
```

```
/******The ico file of the window******/
```

```
ICO_MAIN ICON    "xhk.ico"
```

```
/*******/
```

```
/***Next is the definition of the Menus******/
```

```
IDM_MAIN menu discardable
```

```
{
```

```
    popup    "文件(&F)"
```

```
    {
```

```
        menuitem "打开(&O)\tCtrl+Alt+O", IDM_OPEN
```

```
    }
```

```
    popup    "查看(&V)"
```

```

    {

        menuitem"被禁用的菜单项", IDM_INACTIVE, INACTIVE

        menuitemseparator

        menuitem"被灰化的菜单项", IDM_GRAYED, GRAYED

    }

    popup    "帮助(&H)", HELP

    {

        menuitem"帮助主题(&H)\tF1", IDM_HELP

    }

}

```

//下面定义快捷建

```

IDA_MAIN accelerators

{

    VK_F1, IDM_HELP, VIRTKEY          //F1

    "O", IDM_OPEN, VIRTKEY, CONTROL, ALT      //Ctrl+Alt+O

}

```

把我们用到的资源 ico 文件 xhk.ico 也和此文件放到同一目录下，然后用资源编译器 rc.exe 把 MyMenu.rc 编译成 MyMenu.res

下面该出兵了， 程序代码，采用最精简的：

```

/*****MyMenu.c      Written By XHK 2009.3.3*****/

```

```

#include <windows.h>

```

```

#define ICO_MAIN 0x1000    //图标

#define IDM_MAIN 0x2000    //菜单

#define IDA_MIAN 0x2000    //快捷键

```

```

//回调函数

```

```

LRESULT WINAPI WinProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)

```



```

{

    switch (Msg) //处理消息过程，什么是消息，下节再讲
    {

        case WM_DESTROY://响应鼠标单击关闭按钮事件

            PostQuitMessage(0); //退出消息队列，至于什么是消息队列，下节说

            return 0; //退出函数

        }

    return DefWindowProc(hWnd, Msg, wParam, lParam);

}

//主函数

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int

nShowCmd)

{

    char *cName = "myWindow";

    char *cCaption = "带目录的窗口 - Made By XHK";

    WNDCLASSEX wc;

    HWND hWnd;

    MSG Msg;

    wc.cbClsExtra = 0;

    wc.cbWndExtra = 0;

    wc.cbSize = sizeof(WNDCLASSEX);

    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);

    wc.hCursor = NULL;

    wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(ICO_MAIN)); //载入图标

    wc.hIconSm = NULL;

    wc.hInstance = hInstance;

    wc.lpfnWndProc = WinProc;

    wc.lpszClassName = (LPSTR)cName;

    wc.lpszMenuName = NULL;

```

```

wc.style = CS_HREDRAW | CS_VREDRAW;

RegisterClassEx(&wc);

hWnd = CreateWindowEx(WS_EX_CLIENTEDGE, cName, cCaption, WS_OVERLAPPEDWINDOW,
    200, 100, 300, 200, NULL, LoadMenu(hInstance, MAKEINTRESOURCE
(IDM_MAIN)), hInstance, NULL);

if(hWnd == NULL)
{
    //容错处理
    MessageBox(NULL, "There's an Error", "Error
Title", MB_ICONEXCLAMATION|MB_OK);

    return 0;
}

ShowWindow(hWnd, nShowCmd); //显示窗口

UpdateWindow(hWnd);

while(GetMessage(&Msg, NULL, 0, 0))
{
    TranslateMessage(&Msg); //翻译消息
    DispatchMessage(&Msg); //分派消息
}

return Msg.message;
}

```

把此便以为 MyMenu.obj, z 再和 MyMenu.res 进行连接成 MyMenu.exe, 运行看看和我截的图一样不。当然你也可以定义自己想要的界面, 不过如果是初学者, 可能没有那么高的悟性吧, 不急, 慢慢来, 你会成为高手的。

本节又是长篇大论, 可能劳您心烦, 然资源这方面的知识, 在网上也不太好找, 想介绍简单点, 怕大家日后碰到没见过而又不好找, 所以我尽量压缩篇幅, 依然是冗余漫长。如果大家能够看到这里, 说明您的耐力是很强的, 是做大事者, 相信您有如此精神, 一定会光宗耀祖, 出人头地, 成就一番辉煌的事业的。

1.9 让菜单和快捷键起作用

上节中我们把界面完成了，可是只是界面，没有一点作用，这节我们就让上面的菜单和快捷键起作用，上节中已经介绍了，在我们单击菜单或者快捷键时，Windows 会把相应的 ID 传给窗口处理过程。这个时候我们只要在消息处理代码中加入要处理的代码之后，那么这个程序就可以响应菜单和快捷键了。今天大家又要了解两个新函数，主要是处理快捷键，有点小特别。首先和其他资源一样要先载入快捷键资源，这里用到 LoadAccelerators 函数，具体情况如下：

LoadAccelerators

函数功能：调入加速键表。该函数调入指定的加速键表。

函数原型：HACCEL LoadAccelerators (HINSTANCE hInstance, LPCTSTR lpTableName)；

参数：

hInstance:模块的一个事例的句柄，该模块的可执行文件中包含将要调入的加速键表。

lpTableName:指向一个以空结尾的字符串的指针，该字符串包含了即将调入的加速键表的名字。另一种可选的方案是，该参数可以在加速键表资源的低位字中指定资源标识符，而高位字中全零。

MAKEDINTRESOURCE 宏可被用于创建该值。

返回值：若函数调用成功，则返回非零值。若函数调用失败，则返回值为零。若要获得更多的错误信息，可以调用 GetLastError 函数。

备注：若加速键表尚未装入，该函数可从指定的可执行文件中将它装入。从资源中装入的加速键表，在程序结束时可自动释放。Windows CE：资源不被拷贝到 RAM 中，因而不能被修改。

对于加速键的消息处理也有点特别，因为程序发现消息来源是快捷键就直接把消息发送到窗口处理过程（窗口处理函数）进行处理，和其他消息不太一样，所以这次消息处理代码需要这样来了：

```
while (GetMessage (&Msg, NULL, 0, 0))
{
    if (!TranslateAccelerator (hWnd, hAccel, &Msg))
    {
        TranslateMessage (&Msg); //翻译消息
        DispatchMessage (&Msg); //分派消息
    }
}

return Msg.message;
```

需要先行判断是不是快捷键消息，不是才进行消息的翻译和派送。

下面请看 TranslateAccelerator 的详细内容：

TranslateAccelerator

函数功能：翻译加速键表。该函数处理菜单命令中的加速键。该函数将一个 WM_KEYDOWN 或 WM_SYSKEYDOWN 消息翻译成一个 WM_COMMAND 或 WM_SYSCOMMAND 消息（如果在给定的加速键表中有该键的入口），然后将 WM_COMMAND 或 WM_SYSCOMMAND 消息直接送到相应的窗口处理过程。

TranslateAccelerator 直到窗口过程处理完消息后才返回。

函数原型：int TranslateAccelerator (HWND hWnd, HACCEL hAccTable, LPMSG lpMsg);

参数：

hWnd:窗口句柄，该窗口的消息将被翻译。

hAccTable:加速键表句柄。加速键表必须由 LoadAccelerators 函数调用装入或由 CreateAcceleratorTable 函数调用创建。

LpMsg:MSG 结构指针，MSG 结构中包含了从使用 GetMessage 或 PeekMessage 函数调用线程消息队列中得到的消息内容。

返回值：若函数调用成功，则返回非零值；若函数调用失败，则返回值为零。若要获得更多的错误信息，可调用 GetLastError 函数。

备注：为了将该函数发送的消息与菜单或控制发送的消息区别开来，使 WM_COMMAND 或 WM_SYSCOMMAND 消息的 wParam 参数的高位字值为 1。用于从窗口菜单中选择菜单项的加速键组合被翻译成 WM_SYSCOMMAND 消息；所有其他的加速键组合被翻译成 WM_COMMAND。若

TranslateAccelerator 返回非零值且消息已被翻译，应用程序就不能调用 TranslateMessage 函数对消息再做处理。每个加速键不一定都对应于菜单命令。若加速键命令对应于菜单项，则 WM_INITMENU 和 WM_INITMENUPOPUP 消息将被发送到应用程序，就好像用户正试图显示该菜单。然而，如下的任一条件成立时，这些消息将不被发送：窗口被禁止，菜单项被禁止。

加速键组合无相应的窗口菜单项且窗口已被最小化。鼠标抓取有效。有关鼠标抓取消息，参看 SetCapture 函数。若指定的窗口为活动窗口且窗口无键盘焦点（当窗口最小化时一般是这种情况），TranslateMessage 翻译 WM_SYSDEYUP 和 WM_SYSKEYDOWN 消息而不是 WM_KEYUP 和 WM_KEYDOWN 消息。

当按下相应于某菜单项的加速键，而包含该菜单的窗口又已被最小化时，TranslateMessage 不发送 WM_COMMAND 消息。但是，若按下与窗口菜单或某菜单的任一项均不对应的加速键时，TranslateMessage 将发送一 WM_COMMAND 消息，即使窗口已被最小化。

既然已经理论了一番，那么下来就实践了，来，让我把这可恨有该死的编程代码贴出来供大家参考下：

```
/*****MyMenu.c Written by XHK 2009.3.3****/
```

```
#include <windows.h>
```

```
#define ICO_MAIN 0X1000 //图标
```

```
#define IDM_MAIN 0X2000 //菜单
```

```
#define IDA_MAIN 0X2000 //快捷键
```

```

#define IDM_OPEN          0X4101    // “打开” 菜单项

#define IDM_INACTIVE      0X4201    // “被禁用的菜单项”

#define IDM_GRAYED        0X4202    // “灰化的菜单项”

#define IDM_HELP 0X4301    // “帮助” 菜单项


//回调函数

LRESULT WINAPI WinProc(HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam)
{
    switch(Msg)//处理消息过程，什么是消息，下节再讲
    {
        case WM_COMMAND:
            switch(0x0000ffff&wParam)
            {
                case IDM_OPEN:
                    MessageBox(hWnd,“你单击了\”打开\”菜单项”,“提示”,MB_OK);

                    break;

                case IDM_HELP:
                    MessageBox(hWnd,“你单击了\”帮助主题\”菜单项”,“提示”,MB_OK);

                    break;
            }

            break;

        case WM_DESTROY://响应鼠标单击关闭按钮事件

            PostQuitMessage(0);//退出消息队列，至于什么是消息队列，下节说

            return 0;//退出函数

    }

    return DefWindowProc(hWnd,Msg,wParam,lParam);
}


//主函数

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nShowCmd)
{
    char *cName = “myWindow”;

```

```

char *cCaption = "带目录的窗口 - Made By XHK";

WNDCLASSEX wc;

HWND hWnd;

HACCEL hAccel;//快捷键表句柄

MSG Msg;


wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.cbSize = sizeof(WNDCLASSEX);

wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);

wc.hCursor = NULL;

wc.hIcon = LoadIcon(hInstance,MAKEINTRESOURCE(ICO_MAIN)); //载入图标

wc.hIconSm = NULL;

wc.hInstance = hInstance;

wc.lpfnWndProc = WinProc;

wc.lpszClassName =(LPSTR) cName;

wc.lpszMenuName = NULL;

wc.style = CS_HREDRAW | CS_VREDRAW;


RegisterClassEx(&wc);


hWnd = CreateWindowEx(WS_EX_CLIENTEDGE, cName, cCaption, WS_OVERLAPPEDWINDOW,

    400, 300, 300, 200, NULL, LoadMenu(hInstance, MAKEINTRESOURCE(IDM_MAIN)), hInstance, NULL);

if(hWnd == NULL)

{ //容错处理

    MessageBox(NULL, "There's an Error", "Error Title", MB_ICONEXCLAMATION|MB_OK);

    return 0;

}

ShowWindow(hWnd, nShowCmd); //显示窗口

UpdateWindow(hWnd);


//获取快捷键句柄

```

```

hAccel=LoadAccelerators(hInstance,MAKEINTRESOURCE(IDA_MAIN));

while(GetMessage(&Msg,NULL,0,0))
{
    //先判断是不是快捷键消息
    if(!TranslateAccelerator(hWnd,hAccel,&Msg))
    {
        TranslateMessage(&Msg); //翻译消息
        DispatchMessage(&Msg); //分派消息
    }
}

return Msg.message;
}

```

大家想写的写下练练手，不想的就直接拷吧，如果有错误就自己该吧，总之最后，你得使你的程序可以响应击目录和快捷键，废话不说了，就此打住。

1.10 菜单的高级操作和快捷菜单的编程

上节中说道的是菜单的一些基本操作，显然满足不了实际上的需要，因为在我们操作程序的时候，有的菜单项有时是可用的，有时是不可用的，也有可能根据情况动态地添加、删除和修改菜单，本节将解决这些问题。此外，快捷菜单往往能大大地提高工作的效率，所以还要说下快捷菜单的东西。

菜单项的添加、删除、和修改，也就是通过这个 API 而已，所以这节就是学习这个函数的使用了。

1. 添加菜单项

```

BOOL AppendMenu (hMenu hMenu, UINT uFlags, UINT uIDNewItem, LPCTSTR lpNewItem);

```

2. 插入菜单项

```

BOOL InsertMenu(HMENU hMenu,UINT uPosition,UINT uFlags,UINT uIDNewItem,LPCTSTR
lpNewItem);

```

3. 修改菜单项

```

BOOL ModifyMenu(HMENU hMnu,UINT uPosition,UINT uFlags,UINT uIDNewItem,LPCTSTR
lpNewItem);

```

4. 删除菜单项

```
BOOL DeleteMenu(HMENU hMenu,UINT uPosition,UINT uFlags);

BOOL RemoveMenu( HMENU hMenu, UINT uPosition, UINT uFlags
);
```

其实 **AppendMenu** 和 **InsertMenu** 函数都是添加菜单项函数，只不过 **AppendMenu** 是在菜单的最后添加菜单项，**InsertMenu** 则是在菜单项中间插入菜单项。**DeleteMenu** 和 **RemoveMenu** 都可以删除菜单，两者的不同之处在于：当它们用于 **popup** 属性的菜单项时，**DeleteMenu** 不仅删除菜单项，而且将这个 **popup** 菜单项的所有子项目全部删除，而 **RemoveMenu** 函数进菜单项中移去这个 **popup** 菜单项，整个 **popup** 菜单在内存中还是存在的。

这些函数的参数基本上都差不多，**hMenu** 是要操作的菜单的句柄；**uPosition** 是菜单项的位置。位置的表示方法有两种：用命令 ID 定位或用位置索引。函数使用哪一种方法，是有后面的 **uFlags** 决定的。当 **uFlags** 为 **MF_BYCOMMAND** 时，**uPosition** 应为菜单项的命令 ID；而 **uFlags** 为 **MF_BYPOSITION** 时，**uPosition** 表示菜单项的位置索引，索引是从 0 开始的，第一个菜单项的索引为 0。

AppendMenu 和 **InsertMenu** 函数中的 **uIDNewItem** 表示这个新菜单项的命令 ID，**lpNewItem** 是新菜单项的文本字符串指针。**ModifyMenu** 函数则是修改这两个参数了。

为了编程上的方便，接下来就把快捷菜单的给说了，然后在同一个程序中实现菜单的高级操作和快捷菜单的操作。前面也说了，快捷菜单也是弹出时菜单，只不过在鼠标单击右键时弹出，弹出的时候，是在鼠标单击的位置弹出。这里要用到一个函数 **TrackPopupMenu** 函数。

TrackPopupMenu

函数功能：该函数在指定位置显示快捷菜单，并跟踪菜单项的选择。快捷菜单可出现在屏幕上的任何位置。

函数原型：**BOOL TrackPopupMenu (HMENU hMenu, UINT uFlags, int x, int y, int nReserved, HWND hWnd, CONST RECT*prcRect) ;**

参数

hMenu：被显示的快捷菜单的句柄。此句柄可为调用 **CreatePopupMenu** 创建的新快捷菜单的句柄，也可以为调用 **GetSubMenu** 取得的与一个已存在菜单项相联系的子菜单的句柄。

uFlags：一种指定功能选项的位标志。用下列标志位之一来确定函数如何水平放置快捷菜单：

TPM_CENTERALIGN：若设置此标志，函数将按参数 **x** 指定的坐标水平居中放置快捷菜单。

TPM_LEFTALIGN：若设置此标志，函数使快捷菜单的左边界与由参数 **x** 指定的坐标对齐。

TPM_RIGHTALIGN：若设置此标志，函数使快捷菜单的右边界与由参数 **x** 指定的坐标对齐。

用下列标志位之一来确定函数如何垂直放置快捷菜单：

TPM_BOTTOMALIGN：若设置此标志，函数使快捷菜单的下边界与由参数 **y** 指定的坐标对齐。

TPM_TOPALIGN：若设置此标志，函数使快捷菜单的上边界与由参数 **y** 指定的坐标对齐。

TPM_VCENTERALIGN：若设置此标志，函数将按参数 **y** 指定的坐标垂直居中放置快捷菜单

用下列标志位之一来确定在菜单没有父窗口的情况下用户的选择：

TPM_NONOTIFY: 若设置此标志，当用户单击菜单项时函数不发送通知消息。

TPM_RETURNCMD: 若设置此标志；函数将用户所选菜单项的标识符返回到返回值里。

(补充：当 **TrackPopupMenu** 的返回值大于 0，就说明用户从弹出菜单中选择一个菜单。以返回的 ID 号为参数 **wParam** 的值，程序给自己发送了一个 **WM_SYSCOMMAND** 消息)

用下列标志位之一来确定在快捷菜单跟踪哪一个鼠标键：

TPM_LEFTBUTTON: 若设置此标志，用户只能用鼠标左键选择菜单项。

TPM_RIGHTBUTTON: 若设置此标志，用户能用鼠标左、右键选择菜单项。

X: 在屏幕坐标下，快捷菜单的水平位置。

Y: 在屏幕坐标下，快捷菜单的垂直位置。

NReserved: 保留值，必须为零。

hWnd: 拥有快捷菜单的窗口的句柄。此窗口接收来自菜单的所有消息。函数返回前，此窗口不接受来自菜单的 **WM_COMMAND** 消息。

如果在参数 **uFlags** 里指定了 **TPM_NONOTIFY** 值，此函数不向 **hWnd** 标识的窗口发消息。但必须给 **hWnd** 里传一个窗口句柄，可以是应用程序里的任一个窗口句柄。

PrcRect: 未用。

返回值：如果在参数 **uFlags** 里指定了 **TPM_RETURNCMD** 值，则返回值是用户选择的菜单项的标识符。如果用户未作选择就取消了菜单或发生了错误，则返回值是零。如果没在参数 **uFlags** 里指定 **TPM_RETURNCMD** 值，若函数调用成功，返回非零值，若函数调用失败，返回零。若想获得更多的错误信息，请调用 **GetLastError**

函数：

备注：Windows CE 不支持参数 **uFlags** 取下列值：**TPM_NONOTIFY**；**TPM_LEFTBUTTON**；**TPM_RIGHTBUTTON**。

至于获取鼠标的位置，可以请出这个函数：

GetCursorPos

函数功能：该函数检取光标的位置，以屏幕坐标表示。

函数原型：**BOOL GetCursorPos (LPPOINT lpPoint) ;**

参数：

lpPoint: **POINT** 结构指针，该结构接收光标的屏幕坐标。

使用时要先定义一个数据结构：**Public Type POINTAPI**

x As Long

y As Long

End Type

GetCursorPos biao

那么 **biao.x** 用来存放当前光标的 **x** 轴坐标, **biao.y** 用来存放当前 **y** 轴的坐标。

返回值：如果成功，返回值非零；如果失败，返回值为零。若想获得更多错误信息，请调用 **GetLastError** 函数。

使用 TrackPopupMenu 要注意的是,弹出的菜单句柄一般为 popup 类型的,而用 LoadMenu 函数装载的并不是 popup 类型的, popup 只能在第二层或者够多层中定义,这时就要用 GetSubMenu 函数来获取第二层菜单句柄,也就是 popup 型的菜单句柄, GetSubMenu 这个函数比较简单,大家看下就会明白的,不多说。

继上节中的代码, MyMenu.rc 修改为如下代码:

```
/******MyMenu.rc Written By XHK 2009.3.3******/
```

```
/*****MyMenu.c Modified by XHK 2009.3.4*****/
```

```
#include <resource.h>
```

```
#define ICO_MAIN 0X1000 //图标
```

```
#define IDM_MAIN 0X2000 //菜单
```

```
#define IDA_MAIN 0X2000 //快捷键
```

```
#define IDM_OPEN 0X4101 //“打开”菜单项
```

```
#define IDM_INACTIVE 0X4201 //“被禁用的菜单项”
```

```
#define IDM_GRAYED 0X4202 //“灰化的菜单项”
```

```
#define IDM_HELP 0X4301 //“帮助”菜单项
```

```
#define IDM_SHORTCUT 0X4400 //快捷菜单
```

```
#define IDM_APPEND 0X4401 //“添加新菜单项”
```

```
#define IDM_DELETE 0X4402 //“删除菜单项”
```

```
/******The ico file of the window******/
```

```
ICO_MAIN ICON "xhk.ico"
```

```
/*******/
```

```
/*Next is the definition of the Menus******/
```

```
IDM_MAIN menu discardable
```

```
{
```

```
popup "文件(&F)"
```

```
{
```

```
menuitem "打开(&O)\tCtrl+Alt+O", IDM_OPEN
```

```

    }

    popup    "查看(&V)"

    {

        menuitem"被禁用的菜单项", IDM_INACTIVE, INACTIVE

        menuitemseparator

        menuitem"被灰化的菜单项", IDM_GRAYED, GRAYED

    }

    popup    "帮助(&H)", HELP

    {

        menuitem"帮助主题(&H)\tF1", IDM_HELP

    }

}

```

```

IDM_SHORTCUT  menu    discardable

{

    popup    "    "//不想起名字了，用空格了

    {

        menuitem"添加新菜单项", IDM_APPEND

        menuitem"删除菜单项", IDM_DELETE

    }

}

```

//下面定义快捷建

```

IDA_MAIN accelerators

{

    VK_F1, IDM_HELP, VIRTKEY    //F1

    "O", IDM_OPEN, VIRTKEY, CONTROL, ALT    //Ctrl+Alt+O

}

```

程序代码文件修改如下：

```

/****MyMenu.c Written by XHK 2009.3.3****/

/****MyMenu.c Modified by XHK 2009.3.4****/

#include <windows.h>


#define ICO_MAIN          0X1000    //图标
#define IDM_MAIN          0X2000    //菜单
#define IDA_MAIN          0X2000    //快捷键


#define IDM_OPEN          0X4101    // “打开” 菜单项
#define IDM_INACTIVE      0X4201    // “被禁用的菜单项”
#define IDM_GRAYED        0X4202    // “灰化的菜单项”
#define IDM_HELP 0X4301    // “帮助” 菜单项


#define IDM_SHORTCUT      0X4400
#define IDM_APPEND        0X4401
#define IDM_DELETE        0X4402


HMENU hMenu;//定义全局变量， 方面函数调用

HMENU hMenuShort;//同上


//定义弹出快捷菜单函数
int PopupShortcutMenu(hWnd)
{
    POINT point;

    GetCursorPos(&point);//获取鼠标的位置

    //弹出菜单

    TrackPopupMenu(GetSubMenu(hMenuShort, 0), TPM_CENTERALIGN, point.x, point.y, 0, (HWND) hWnd, NULL);

    return 1;
}


//回调函数

```

```

LRESULT WINAPI WinProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)

{
    //HMENU hMenu;

    //HINSTANCE hInstance;

    switch(Msg)
    {
    case WM_COMMAND:

        switch(0x0000ffff&wParam)

        {

            case IDM_OPEN:

                MessageBox(hWnd, "你单击了\“打开\“菜单项", "提示", MB_OK);

                break;

            case IDM_HELP:

                MessageBox(hWnd, "你单击了\“帮助主题\“菜单项", "提示", MB_OK);

                break;

            case IDM_APPEND:

                //添加新菜单项

                AppendMenu(GetSubMenu(hMenuShort, 0), MF_CHECKED, 0x4403, "新添加的菜单项");

                break;

            case IDM_DELETE:

                //删除菜单项

                DeleteMenu(GetSubMenu(hMenuShort, 0), 0x4403, MF_BYCOMMAND);

                break;

        }

        break;

            case WM_RBUTTONDOWN:

                PopupShortcutMenu(hWnd);

                break;

            case WM_DESTROY://响应鼠标单击关闭按钮事件

                PostQuitMessage(0);

                return 0;//退出函数

        }

```

```

        return DefWindowProc(hWnd, Msg, wParam, lParam);
    }

//主函数

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
    char *cName = "myWindow";

    char *cCaption = "带目录的窗口 - Made By XHK";

    WNDCLASSEX wc;

    HWND hWnd;

    HACCEL hAccel;//快捷键表句柄

    MSG Msg;

    wc.cbClsExtra = 0;

    wc.cbWndExtra = 0;

    wc.cbSize = sizeof(WNDCLASSEX);

    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);

    wc.hCursor = NULL;

    wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(ICO_MAIN)); //载入图标

    wc.hIconSm = NULL;

    wc.hInstance = hInstance;

    wc.lpfnWndProc = WinProc;

    wc.lpszClassName = (LPSTR)cName;

    wc.lpszMenuName = NULL;

    wc.style = CS_HREDRAW | CS_VREDRAW;

    RegisterClassEx(&wc);

    hMenuShort = LoadMenu(hInstance, MAKEINTRESOURCE(IDM_SHORTCUT));

    hMenu = LoadMenu(hInstance, MAKEINTRESOURCE(IDM_MAIN));

    hWnd = CreateWindowEx(WS_EX_CLIENTEDGE, cName, cCaption, WS_OVERLAPPEDWINDOW,

        400, 300, 300, 200, NULL, hMenu, hInstance, NULL);

```

```

if (hWnd == NULL)

{ //容错处理

    MessageBox(NULL, "There's an Error", "Error Title", MB_ICONEXCLAMATION | MB_OK);

    return 0;

}

ShowWindow(hWnd, nShowCmd); //显示窗口

UpdateWindow(hWnd);


//获取快捷键句柄

hAccel=LoadAccelerators(hInstance, MAKEINTRESOURCE(IDA_MAIN));


while(GetMessage(&Msg, NULL, 0, 0))

{

    //先判断是不是快捷键消息

    if(!TranslateAccelerator(hWnd, hAccel, &Msg))

    {

        TranslateMessage(&Msg); //翻译消息

        DispatchMessage(&Msg); //分派消息

    }

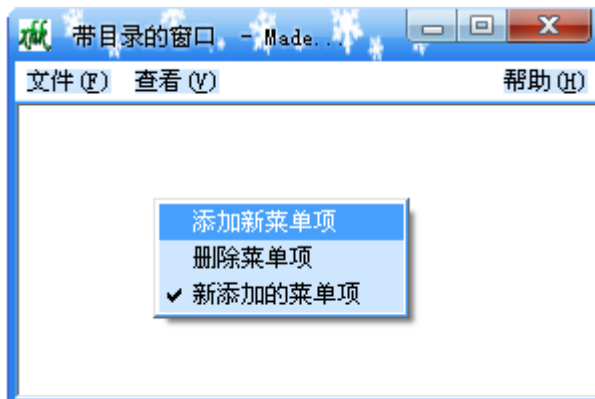
}

return Msg.message;

}

```

给最终效果拍个照，给大家秀秀

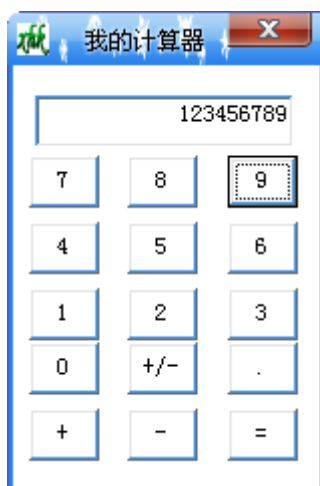


不好意思，照的时候，它头一偏，结果一部没照到，不过要看的部分都有，大家就将就下了，如果你不将就看的话，那就自己写一个出来了，哈。其实我是很希望你们能够自己写，不要直接 Ctrl+C、Ctrl+V 就可以了，哪怕是抄，

也希望你能亲自讲那些羞涩的字符敲进去。因为编程时才能发现错误，解决了错误，你就进步了，如果我上面的代码没有错的话，你直接复制粘贴就了事，即使你说你没一行代码都看过了，都理解了，但我可以肯定的说你没有进步（针对没有写过这类程序的人说的，知道的飞过）。谦虚使人进步，你想成为高手吗，那就虚心地去写代码去吧。

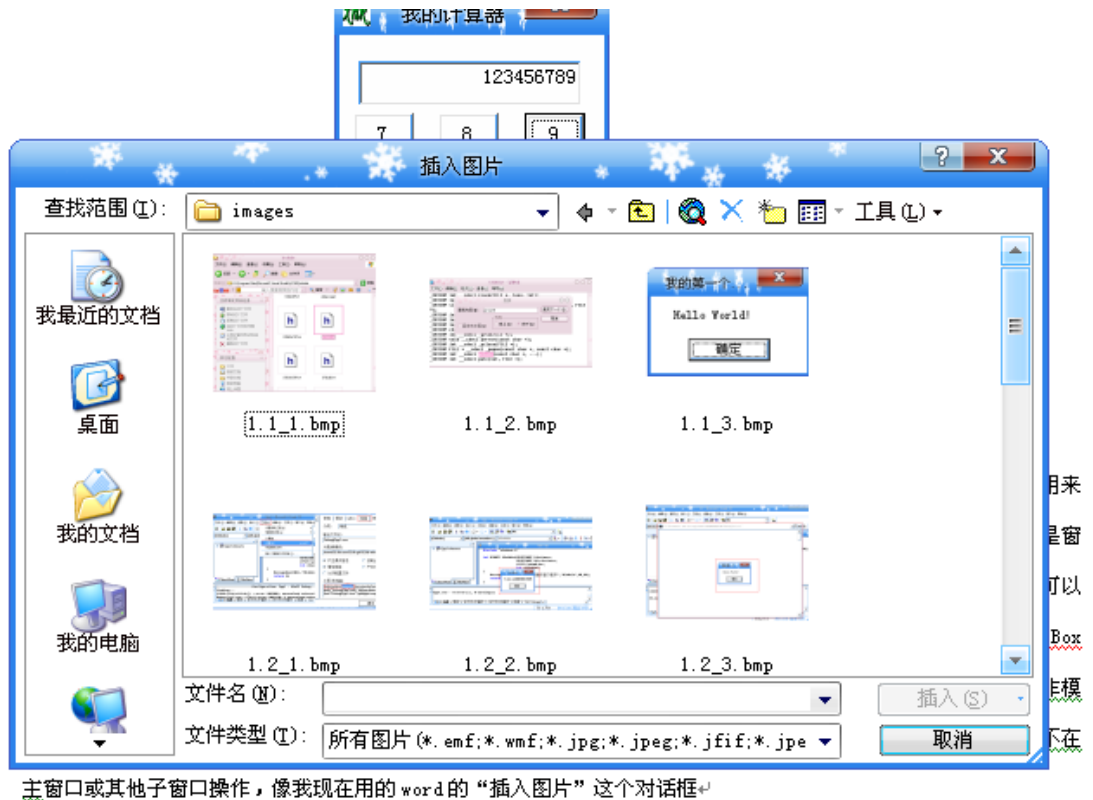
1.11 对话框资源及其他一些常用资源的使用

本来是想给大家详细说说常用的每一个资源，然一位读者说资源那么多，界面也是千变万化的，就算说完估计也不会取得什么好的效果，说完大家只会觉得啰嗦，应该留一点让读者自己去发现，去写。想想也是，资源是很多的，写完简直就是长篇大论了，估计也许的确够烦人的，想想这样，干脆做个了断算了，本节主要说下对话框资源，其他的资源如果能用到的就说下，用不到的就不管了，其实就是资源定义，很简单的，大家可以自己查下就可以解决的。这节我们的目标是做一个计算器。



界面如上图，这个用对话框、编辑框控件和按钮控件资源写的。下面就对这三种资源给详细说一下。

我们已经很熟悉了，用 `MessageBox` 函数可以轻而易举地创建一个对话框，不过那只是其中的一类，对话框是用来给人们“对话”的，而对话的方式不可能只有一种，可以是任意的，所以对话框的样式是千变万化的。对话框只是窗口程序的一个子集，所以对话框和一般的窗口程序没有太大的区别。而且很好的一点，写一个对话框窗口，我们可以很简单的做到，因为 Windows 为我们提供了这样的函数，我只要用一个函数就可以创建一个对话框窗口，像 `MessageBox` 一样，很简单地就可以创建一个窗口出来，还有看上面的计算器就是对话框做的窗口。对话框有两种：模式的和非模式的。什么是模式的，什么是非模式的呢？模式的对话框，就是对话框弹出后，你只能在对话框中进行操作，而不在主窗口或其他子窗口操作，像我现在用的 word 的“插入图片”这个对话框。



在我插入图片或者取消插入之前，点其他地方是没有任何反应的，只有这个对话框返回后，我才能继续写或进行其他操作。知道了什么是模式对话框，根据字面意思也知道什么是非模式对话框了，这里仍以 word 为例截图说明。

像这个“查找”对话框，当它弹出来后，我可以继续进行写入或者其他操作。主窗口或者其他子窗口不用等待非模式对话框的返回。

上面说了，建立对话框只需一个函数即可，但对话框有两种，所以分别用两个不同的函数来创建。DialogBoxParam 函数用来创建模式对话框，CreateDialogParam 函数用来创建非模式对话框。用函数就可以省去普通窗口创建过程中的注册、创建、显示和更新的过程，甚是方便的，建议大家写简单的程序就用基于对话框的窗口程序。

Windows 在这两个函数的内部调用 CreateWindowEx 来建立对话框，使用的风格、大小和位置等参数取自资源中定义的对话框模版，使用的窗口类则是 Windows 内部定义的类。窗口过程也被定义到了内部的“对话框管理器”代码中，Windows 在这里处理对话框的大部分消息，对话框管理器在初始化对话框时会根据对话框模版中定义的子窗口控件建立对话框中所有的子窗口。

用户程序中的对话框过程是由对话框管理器调用的，在处理消息前，对话框管理器会先调用用户制定的对话框过程，再根据对话框过程的返回值决定是否处理它们。

Windows 对模式对话框和非模式对话框的吃力有些不同。创建并显示模态对话框后，Windows 会为它在内部建立一个消息循环，在这个消息循环，在这个消息发送给对话框管理器，对话框管理器在处理在处理消息的过程中会调用用户定义的对话框过程，当对话框关闭的时候，Windows 退出内建的消息循环，并从 DialogBoxParam 函数返回。而对于

非模式对话框 `CreateDialogParam` 函数在创建对话框后直接返回，对话框窗口的消息是通过用户程序中的消息循环派送。

由于模式对话框的特征，使得用它来做小程序的主窗口非常方便，因为用 `DialogBoxParam` 函数就可以搞定了，这样可以省好多代码，何乐而不为呢？

对话框资源定义

在资源脚本中定义对话框的语法是：

```
对话框 ID DIALOG [DISCARDABLE] x 坐标, y 坐标, 宽度, 高度 [可选属性]
{
    子窗口控件
}
```

重用的可选属性

标题文字	CAPTION	“文字”	定义显示在窗口标题栏上的文字
窗口类	CLASS	“类名”	定义对话框窗口使用的窗口类，若不定义则使用 Windows 内建的类
窗口风格	STYLE	风格组合	定义对话框的风格
扩展风格	EXSTYLE	风格组合	定义对话框的扩展风格
字体	FONT	大小 “字体名”	定义对话框包括子窗口的字体
菜单	MENU	菜单 ID	对话框使用的菜单，在同一脚本资源文件中定义

我们这个实例在对话框上用到了编辑框控件和按钮控件，像这些控件都是 Windows 的一些预定义类，每一个控件也算是一个窗口，只不过属于子窗口罢了，所以所有控件的定义和使用和对话框的定义和使用是大同小异的。同样，这些子窗口控件不用我们写代码去创建它们（在对话框中使用的时候），我们只要定义到资源文件中就可以了，对话框管理器会在初始化对话框的时候，根据资源脚本的定义语句自动创建所有的子窗口的。

子窗口的定义方法有：

`CONTROL 文本, ID, 风格, x, y, 宽度, 高度, [, 扩展风格]`

或

`CONTROL [文本,] ID, x, y, 宽度, 高度, [, 风格][, 扩展风格]`

常用的控件有按钮（`PUSHBUTTON`）、默认按钮（`DEFPBUTTON`）、复选框（`CHECKBOX`）、单选按钮（`RADIOBUTTON`）编辑框（`EDITTEXT`）、组合框（`COMBOBOX`）、列表框（`LISTBOX`）、分组框（`GROUPBOX`）、滚动条（`SCROLLBAR`）和图标框（`ICON`）等。大家用的时候，如果不知道哪个控件的风格或者扩展风格怎么定义，可以到 msdn 上搜下就知道了，因为那么多的风格和扩展风格神仙也不一定能记得了，反正我是不记的，给大脑节省点空间记这个英语单词也比记微软程序员定义的宏好的多。

做一个计算器的知识已经准备的差不多了，下载就该练兵了

还是先做界面，先写资源文件。

```
/******MyCalculator.rc******/
```

```
#include <resource.h>
```

```
#define ICO_MAIN 0x1000
```

```
#define DLG_MAIN 1
```

```
#define IDB_0 0x4400 //0
```

```
#define IDB_1 0x4401 //1
```

```
#define IDB_2 0x4402 //2
```

```
#define IDB_3 0x4403 //3
```

```
#define IDB_4 0x4404 //4
```

```
#define IDB_5 0x4405 //5
```

```
#define IDB_6 0x4406 //6
```

```
#define IDB_7 0x4407 //7
```

```
#define IDB_8 0x4408 //8
```

```
#define IDB_9 0x4409 //9
```

```
#define IDB_PLUS 0x4410 //+
```

```
#define IDB_SUB 0x4411 //-
```

```
#define IDB_EQU 0x4412 //=
```

```
#define IDB_DOT 0x4413 //.
```

```
#define IDB_PAS 0x4414 //正负号
```

```
#define IDB_EDIT 0x4415 //编辑框
```

```
ICO_MAIN ICON "xhk.ico"
```

```
DLG_MAIN DIALOG 300,150,102,140 STYLE DS_SETFONT | DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
```

```
CAPTION "我的计算器"
```

```
FONT 9,"宋体"
```

```
{
```

```

//定义编辑框控件

EDITTEXT IDB_EDIT, 7, 10, 86, 17, ES_RIGHT


PUSHBUTTON "7", IDB_7, 5, 30, 23, 17

PUSHBUTTON "8", IDB_8, 38, 30, 23, 17

PUSHBUTTON "9", IDB_9, 71, 30, 23, 17


PUSHBUTTON "4", IDB_4, 5, 52, 23, 17

PUSHBUTTON "5", IDB_5, 38, 52, 23, 17

PUSHBUTTON "6", IDB_6, 71, 52, 23, 17


PUSHBUTTON "1", IDB_1, 5, 74, 23, 17

PUSHBUTTON "2", IDB_2, 38, 74, 23, 17

PUSHBUTTON "3", IDB_3, 71, 74, 23, 17


PUSHBUTTON "0", IDB_0, 5, 92, 23, 17

PUSHBUTTON "+/-", IDB_PAS, 38, 92, 23, 17

PUSHBUTTON ". ", IDB_DOT, 71, 92, 23, 17


PUSHBUTTON "+", IDB_PLUS, 5, 114, 23, 17

PUSHBUTTON "-", IDB_SUB, 38, 114, 23, 17

PUSHBUTTON "=", IDB_EQU, 71, 114, 23, 17

}

```

下面还是主程序代码：

```

/*****MyCalculator.c*****/

#include <windows.h>

#include <stdio.h>


#define ICO_MAIN 0X1000

#define DLG_MAIN 1

```

```

#define IDB_0    0X4400    //0
#define IDB_1    0X4401    //1
#define IDB_2    0X4402    //2
#define IDB_3    0X4403    //3
#define IDB_4    0X4404    //4
#define IDB_5    0X4405    //5
#define IDB_6    0X4406    //6
#define IDB_7    0X4407    //7
#define IDB_8    0X4408    //8
#define IDB_9    0X4409    //9
#define IDB_PLUS 0X4410    //+
#define IDB_SUB   0X4411    //-
#define IDB_EQU   0X4412    //=
#define IDB_DOT   0X4413    //.
#define IDB_PAS   0X4414    //正负号
#define IDB_EDIT 0x4415    //编辑框

```

```
int  num1=0;//定义了第一个数字
```

```
int  num2=0;//定义了第二个数字
```

```
char s[10];//为了方面参数的传递，定义了这个全局变量，完全没有这个必要
```

```
UINT uFlags=1;//标识是否按下了加号或者等号，切换给 num1 和 num2 赋值
```

```
char oPration='+';//操作符标志，判断按下的是什么操作符，默认为加
```

```
//把字符串转化成数字
```

```
int StrToNum(char * str)
```

```

{
    return  atoi(str);
}

```

```
//把数字转化成字符串
```

```
char * NumToStr(int nNum)
```

```

{

    itoa(nNum, s, 10);

    return s;

}

```

//修改编辑框控件的文字

```

int SetEditValue(int nNum, HWND hEdit)
{
    if(uFlags==1)
    {
        num1 = num1*10+nNum;//可以使数字进位（向左移）

        SetWindowText(hEdit, NumToStr(num1));
    }
    else
    {
        num2 = num2*10+nNum;//可以使数字进位（向左移）

        SetWindowText(hEdit, NumToStr(num2));
    }

    return 0;
}

```

LRESULT WINAPI DialogProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)

```

{

    HICON hIcon;

    HWND hEdit = GetDlgItem(hWnd, IDB_EDIT);

    switch(Msg)
    {

    case WM_INITDIALOG:

        //设置图标

        hIcon = LoadIcon(GetModuleHandle("MyCalculator.exe"), MAKEINTRESOURCE(ICO_MAIN));
    }
}

```

```
SendMessage(hWnd, WM_SETICON, ICON_BIG, (long)hIcon);

SetWindowText(hEdit, "0."); // 让编辑框控件的内容为"0."

return TRUE;

case WM_COMMAND:

    switch(LOWORD(wParam)) // LOWORD(wParam) 用来取出命令 ID
    {

        // 一下处理过程可以更简单的，大家自己想想，看怎样处理好
        // 自己完成计算器可以满足支持小数点和正负号

        case IDB_0:

            SetEditValue(0, hEdit);

            break;

        case IDB_1:

            SetEditValue(1, hEdit);

            break;

        case IDB_2:

            SetEditValue(2, hEdit);

            break;

        case IDB_3:

            SetEditValue(3, hEdit);

            break;

        case IDB_4:

            SetEditValue(4, hEdit);

            break;

        case IDB_5:

            SetEditValue(5, hEdit);

            break;

        case IDB_6:

            SetEditValue(6, hEdit);

            break;

        case IDB_7:

            SetEditValue(7, hEdit);

            break;
```

```

case IDB_8:

    SetEditValue(8, hEdit);

    break;

case IDB_9:

    SetEditValue(9, hEdit);

    break;

case IDB_PLUS:

    oPration = '+';

    uFlags=0;

    break;

case IDB_SUB:

    oPration = '-';

    uFlags=0;

    break;

case IDB_DOT:

    MessageBox(hWnd, "自己写代码让计算器支持小数点", "提示", MB_OK);

    break;

case IDB_PAS:

    MessageBox(hWnd, "自己写代码让计算器支持负数", "提示", MB_OK);

    break;

case IDB_EQU:

    //按等号，显示结果，并把 num1 和 num2 清零，准备下一次运算

    if(oPration=='+')

        SetWindowText(hEdit, NumToStr(num1+num2));

    else

        SetWindowText(hEdit, NumToStr(num1-num2));

    uFlags=1;

    num1=0;

    num2=0;

    break;

}

```



```

        break;

case WM_CLOSE://关闭消息

    EndDialog(hWnd, 0); //终止模态对话框

    return TRUE;

}

return FALSE;

}

int WINAPI WinMain(HINSTANCE hInstance,

                    HINSTANCE hPrevInstance,

                    LPSTR lpCmdLine,

                    int nShowCmd)

{

    //很简单的创建了窗口

    DialogBoxParam(hInstance, MAKEINTRESOURCE(DLG_MAIN), NULL, DialogProc, 0);

    return 1;

}

```

这样就写了一个很简单的计算器，由于本人生性愚钝，写不出很好的算法来，这么简单的计算器（只是整数的加减法），写了如此多的代码，大家想想用简单的点，并且也支持小数点和正负数，这里权当抛砖引玉了。前面说过了我自己写资源文件，是为了让大家知道，那界面到底是怎样出来的，如果大家已经熟悉了的话，可以用资源编辑工具直接编辑，那样更直观更快。这样以后有新的控件出现，我也不详细说明了，大家应该可以根据字面意思就能猜到是什么用的。下节中我们做一个 QQ 登陆的界面出来。

1.12 模范 QQ 登录界面

在序言中，我曾给大家承诺过，大家看了我写的这些东西，写一个类似 QQ 的聊天工具应该不难，这里就先给大家写一个模范 QQ 登陆窗口的界面来。下面是 QQ 2009 的界面和我自己模仿的。



当然我模仿的很粗糙，跟腾讯的比起来简直是大巫见小巫，没法比呀，毕竟人家是优秀的高级人才，从美观，才算法都比我这好数百倍的，我的这个，给人的第一感觉：难看，第二感觉：还是难看，第三感觉：超级难看。大家就将就下，毕竟我太仓促了，有些细节没有关注掉，而且有些地方是需要特殊处理的（像控件的背景），但这些特殊处理的知识我会在后面陆续推出的。大体上和 QQ 差不多，这节主要是练习一下上节课中讲的控件的应用，也就是在资源脚本中是怎样定义的，我把注释写在代码里，大家直接看代码吧。

```
/******MyqqWnd.rc Written By XHK******/
```

```
#include <resource.h>
```

```
#define DLG_MAIN 1
```

```
#define ICO_MAIN 0x1000
```

```
#define IDB_QQ 2
```

```

#define IDE_USER 0X4101      //帐号输入框

#define IDE_PASS 0X4102      //密码输入框


#define IDC_RECORD  0X4201    //记录密码复选框

#define IDC_AUTO  0X4202      //自动登录复选框


#define IDB_CHECK  0X4301     //查杀木马按钮

#define IDB_SET     0X4302     //设置按钮

#define IDB_SUBMIT  0X4303     //提交按钮


#define IDC_QQ      0X4304


ICO_MAIN ICON"qq.ico"

IDB_QQ  BITMAP  "qq.bmp"


DLG_MAIN DIALOG  255, 205, 222, 145

STYLE WS_SYSMENU | WS_MINIMIZEBOX

CAPTION "QQ 2009 Made By XHK"

FONT 9, "宋体"

{

    CONTROL "AAAAA", IDC_QQ, "Static", SS_BITMAP | WS_CHILD |

WS_VISIBLE, 0, 0, 20, 40


    GROUPBOX "", -1, -1, 48, 224, 75


    RTEXT "帐号:", IDC_STATIC, 14, 55, 20, 15, SS_CENTERIMAGE

    EDITTEXT IDE_USER, 37, 55, 120, 15, ES_NUMBER

    LTEXT "注册新帐号", -1, 160, 55, 50, 15, SS_CENTERIMAGE


    RTEXT "密码:", -1, 14, 75, 20, 15, SS_CENTERIMAGE

    EDITTEXT IDE_PASS, 37, 75, 120, 15, ES_PASSWORD

```

```

LTEXT "取回密码",-1,160,75,45,15,SS_CENTERIMAGE

CHECKBOX "记住密码",IDC_RECORD,20,105,50,15

CHECKBOX "自动登录",IDC_AUTO,100,105,50,15


PUSHBUTTON "查杀木马",IDB_CHECK,5,125,50,15

PUSHBUTTON "设置",IDB_SET,60,125,50,15

DEFPUSHBUTTON "登录",IDB_SUBMIT,150,125,50,15

}

```

其实窗口代码跟上节的稍微不同：

```

#include <windows.h>


#define DLG_MAIN 1

#define ICO_MAIN 0X1000

#define IDB_QQ      2


#define IDE_USER 0X4101

#define IDE_PASS 0X4102


#define IDC_RECORD  0X4201

#define IDC_AUTO  0X4202


#define IDB_CHECK  0X4301

#define IDB_SET    0X4302

#define IDB_SUBMIT 0X4303


#define IDC_QQ  0X4304

```

```

LRESULT WINAPI DialogProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
    HICON hIcon;

    HWND hImage;

    HBITMAP hBitmap;

    HINSTANCE hInstance;

    PAINTSTRUCT ps;

    HDC hDC;

    RECT rc;

    switch(Msg)
    {
    case WM_INITDIALOG:
        //设置窗口的图标

        hInstance = GetModuleHandle(NULL);

        hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(ICO_MAIN));

        SendMessage(hWnd, WM_SETICON, ICON_BIG, (long)hIcon);

        //加载那个图片 qq.bmp

        hBitmap = LoadBitmap(hInstance, MAKEINTRESOURCE(IDB_QQ));

        hImage = GetDlgItem(hWnd, IDC_QQ);

        SendMessage(hImage, STM_SETIMAGE, IMAGE_BITMAP, (long)hBitmap);


        break;

    case WM_PAINT:
        //下面是给窗口填充为蓝色的背景

        GetClientRect(hWnd, &rc);

        hDC = BeginPaint(hWnd, &ps);

        FillRect(hDC, &rc, (HBRUSH)CreateSolidBrush(RGB(200, 227, 255)));

        //填充为蓝色

        EndPaint(hWnd, &ps);

        return 0;
    }
}

```

```

        case WM_CLOSE:

            EndDialog(hWnd, 0);

            return TRUE;

        }

        return FALSE;
    }

int WINAPI WinMain(HINSTANCE hInstance,

                    HINSTANCE hPreInstance,

                    LPSTR lpCmdLine,

                    int nShowCmd)

{

    DialogBoxParam(hInstance, MAKEINTRESOURCE(DLG_MAIN), NULL, DialogProc, 0);

    return 1;
}

```

相当简单的窗口，没有任何功能，只是界面。像那些图片按钮或者是按钮背景都是需要编程去实现的，而不是在资源文件中定义的，所以这个界面没有那种效果。那种效果处理起来，跟 qq.bmp 的加载，窗口背景色的设置是差不多的，因为任何控件都窗口，Windows 在创建它们的时候，都用到了 `CreatWindows` 函数，这个过程当然不了解也没有大的关系，但只要知道有这么回事就可以了，那样可以有助于举一反三，因为理解了它们都是窗口都是同一样东西，值要会一个控件特效的设置，其它就都是雷同了，很简单的，本节主要为了练习资源脚本的定义，为以后做基础。

1.3 定时器

我们在写程序处理一些问题时，有时需要在一定时间内重复执行某一个步骤，需要重复执行某个程序或者函数，想定时关机程序，其实很多程序都需要这样做的，QQ 也需要这样的处理的，只是大家都不知道它工作细节而已，我这里说下吧，QQ 在一定时间内就到服务上查看有没有信息的消息，如果有的就提示或者显示，没有的话，就不会叫。所以说定时器是很重要的，本章虽然是说窗口界面的讲解，但是由于定时器的重要的地位，为了以后编写程序做基础，这里就要啰嗦一下了。

定时器的定义：

Microsoft Windows 定时器是一种输入设备,它周期性地在每经过一个指定的时间间隔后就通知应用程序一次。Windows 定时器是非常重要的,这里只说下编程时怎么来用,不说它具体的细节和原理的东西(底层和硬件的,相信大家也不想去学的)。

在写 Windows 程序使用定时器时,可以用 SetTimer 函数向 Windows 申请一个定时器,要求系统在指定的时间以后“通知”应用程序,如果申请成功的话,系统会以指定的时间周期调用 SetTimer 函数指定的回调函数,或者向指定的窗口过程发送 WM_TIMER 消息,比如我们设置时间周期为 1000ms(1s),那么 Windows 就会每隔一秒向窗口发送一个 WM_TIMER 消息。周期时间是从 1~4,294,967,295ms(约 50 天)。如果要终止定时器,可以用 KillTimer 函数来取消定时器。

使用定时器时有一点要注意的是 WM_TIMER 是一个优先级低的消息,Windows 在派送消息时,如果队列中有其他消息,就不会发送 WM_TIMER,直到消息队列中没有其他消息,才会发送 WM_TIMER;而且如果窗口过程忙于处理某个消息没有返回的,使消息队列中消息积累起来,那么就会丢弃 WM_TIMER 消息;消息队列中也不会有多条 WM_TIMER 消息,如果消息队列中已经有一条没有来得及处理的 WM_TIMER 消息,那么到了定时的时刻,多条 WM_TIMER 消息就会被合成一条。所以应用程序不能依靠定时器来保证某件事情必须在规定时刻被处理,另外,也不能依赖定时器消息计数来确定过去了多少时间(可以采用时间差)。

今天这个实例采用定时器,不停地交替显示两张图片,我截了不同时刻的图:



我在百度上随便找了两张汽车的图片,加载到程序中显示,代码非常简单,聪明的你应该一下就知道该怎么写了,不过我还是贴上我的代码,算是献丑了:

```
/******Timer.rc******/
```

```
#include <resource.h>
```

```
#define DLG_MAIN 1
```

```
#define ICO_MAIN 0x1000
```

```
#define IDB_BITMAP1 0x4101
```

```
#define IDB_BITMAP2 0x4102
```

```
#define IDC_BMP 0x4103
```

```
ICO_MAIN ICON "XHK.ICO"
```

```
IDB_BMP1 BITMAP "1.BMP"
```

```
IDB_BMP2 BITMAP "2.BMP"
```

```
DLG_MAIN DIALOG 255,205,92,69
```

```
STYLE WS_SYSMENU | WS_MINIMIZEBOX
```

```
CAPTION "Timer"
```

```
FONT 9,"宋体"
```

```
{
```

```
    CONTROL "AAAAA", IDC_BMP, "Static", SS_BITMAP | WS_CHILD | WS_VISIBLE, 0, 0, 20, 40
```

```
}
```

下面是窗口程序的代码，也极为简单（关于各个新见函数，请参加 msdn 或者网络吧，老是我翻译没什么意思）。

```
/******Timer.c*****
```

```
#include <windows.h>
```

```
#define DLG_MAIN 1
```

```
#define IDB_BMP1 0x4101
```

```
#define IDB_BMP2 0x4102
```

```
#define IDC_BMP 0x4103
```

```
#define ICO_MAIN 0x1000
```

```
UINT uFlags=1;
```

```
LRESULT CALLBACK DialogProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
    HINSTANCE hInstance = GetModuleHandle(NULL);
```

```
    HBITMAP hBitmap;
```

```
    HWND hIdc = GetDlgItem(hWnd, IDC_BMP);
```



```

HICON hIcon;

switch (uMsg)
{
case WM_INITDIALOG:

    hBitmap = LoadBitmap(hInstance, MAKEINTRESOURCE(IDB_BMP1));

    hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(ICO_MAIN));

    SendMessage(hWnd, WM_SETICON, ICON_BIG, (long)hIcon);

    SendMessage(hIdc, STM_SETIMAGE, IMAGE_BITMAP, (long)hBitmap);

    SetTimer(hWnd, 0, 500, NULL); // 申请定时器

    return TRUE;

case WM_TIMER:

    // 根据情况加载不同的图片资源

    if (uFlags)

        hBitmap = LoadBitmap(hInstance, MAKEINTRESOURCE(IDB_BMP1));

    else

        hBitmap = LoadBitmap(hInstance, MAKEINTRESOURCE(IDB_BMP2));

    SendMessage(hIdc, STM_SETIMAGE, IMAGE_BITMAP, (long)hBitmap);

    uFlags = uFlags ^ 1; // 进行异或，若是 1 则变为 0，若是 0 则变为 1

    return TRUE;

case WM_CLOSE:

    KillTimer(hWnd, 0); // 取消定时器

    EndDialog(hWnd, 0);

    return TRUE;

}

return FALSE;
}

```

```

int WINAPI WinMain(HINSTANCE hInstance,

                  HINSTANCE hPreInstance,

                  LPSTR lpCmdline,

```

```

        int                nShowCmd)

{
    DialogBoxParam(hInstance, MAKEINTRESOURCE (DLG_MAIN), 0, DialogProc, 0);

    return 1;
}

```

编译连接运行的你程序，看看达到预期效果没，你也可以用定时器制作其他的程序的，展开你的思维，想想什么可以做的，做点送给朋友也可以，说实话，我就是做了几张照片循环播放的程序，是为一个女孩子做的，我把她的照片处理成图标类型，和 bmp 类型的，然后加载，设定换图片的时间，一个很简单的图片播放程序就 OK 了，相信她是会喜欢的，虽然简陋，但毕竟是量身定做的嘛。

1.14 简单的整蛊——窗口抖动程序的实现

一次朋友曾给我传了个有趣的程序，单击后振动的窗口飘来飘去，其实也就是不停地移动窗口，后来分析后，才知道原来只是移动当前活动的窗口而已，想想这个也很好玩，就是获得当前活动的窗口，然后用 MoveWindow 函数去改变它的位置不就可以了，原来还以为这个程序有什么特别的呢，后来想也没什么了，这节我们来写个这样的程序，我也不想复杂的实现，就简单点。

```

/*****WobbleWnd.rc*****/

#include <resource.h>

#define DLG_MAIN 1

#define ICO_MAIN 0x1000

//#define IDB_BMP 0x4000

ICO_MAIN ICON "XHk.ICO"

//IDB_BMP BITMAP "XHk.BMP"

DLG_MAIN DIALOG 255, 205, 100, 100

STYLE WS_SYSMENU | WS_MINIMIZEBOX

```

```
CAPTION "Wobble Window"
```

```
FONT 9, "宋体"
```

```
{
```

```
}
```

下面是程序实现代码：

```
/******WobbleWnd.c******/
```

```
#include <windows.h>
```

```
#define DLG_MAIN 1
```

```
#define ICO_MAIN 0X1000
```

```
LRESULT CALLBACK DialogProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
    HINSTANCE hInstance = GetModuleHandle(NULL);
```

```
    HICON hIcon;
```

```
    HWND hDestWnd;
```

```
    RECT rc;
```

```
    int i=0;
```

```
    switch(uMsg)
```

```
    {
```

```
    case WM_INITDIALOG:
```

```
        hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(ICO_MAIN));
```

```
        SendMessage(hWnd, WM_SETICON, ICON_BIG, (long)hIcon);
```

```
        SetTimer(hWnd, 0, 10, NULL);
```

```
        return TRUE;
```

```
    case WM_TIMER:
```

```

        i=(rand()-rand())/1000;

        if((hDestWnd=GetForegroundWindow())!=NULL) {
            GetWindowRect(hDestWnd, &rc);
            if(rc.top<0||rc.bottom>800)
                rc.top=100;
            if(rc.left<0||rc.right>1000)
                rc.left=100;

            MoveWindow(hDestWnd, rc.left+i, rc.top+i, rc.right-rc.left, rc.bottom-rc.top, TRUE);
        }

        //SendMessage(

        return TRUE;
    case WM_CLOSE:
        KillTimer(hWnd, 0); //取消定时器
        EndDialog(hWnd, 0);

        return TRUE;
    }

    return FALSE;
}

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPreInstance,
                   LPSTR lpCmdline,
                   int nShowCmd)
{
    DialogBoxParam(hInstance, MAKEINTRESOURCE(DLG_MAIN), 0, DialogProc, 0);

    srand((unsigned)time(NULL));

    return 1;
}

```

代码是相当简单的，没有任何要学习的东西，纯粹是为了吸引大家的兴趣，让大家看看自己的学的东西到底能用到什么地方。当我完成这个之后，把代码给我同学看，同学们都不相信这么少的代码就能做到。这个程序如果不要窗口和资源，写成控制台的程序，代码会更少。由于我没有很好的想这个程序那个随机的算法，可能窗口会有向左上角移动的趋势，而且最终窗口会变得的小。这个问题大家是会解决的，不会编程都可以解决的，何况会编程的你呢。

总结：

这样吧，这章就这样结束了，当然窗口类程序的编写，这里说的只是一点皮毛而已，但这些都是基础，学好了对以后就会很有帮助的，而且你现在已经可以做出窗口的程序了，是不？说明你已经可以脱离菜鸟级了，Come on 吧，很快你就会成为高手的。俗话说，师傅领进门，修行靠个人，我以前把你领到这里了，你想去哪里就去哪里吧。