常用链接
我的随笔
我的评论
我参与的随笔

留言簿(2)
给我留言
查看公开留言
查看私人留言

我参与的团队

随笔分类
boost(1) XML
WinCE XML
胡思乱想 XML
无线电技术 XML
需求分析(4) XML

随笔档案
2010年12月 (1)
2010年10月 (1)
2010年8月 (3)

文章分类
android(7) XML
assembly/disassembly(4) XML
C/C++(39) XML
data structure(3) XML
DirectX(1) XML
DLL(6) XML
GDI、GDI+(2) XML
GPS(2) XML
multithread(3) XML
STL(1) XML
Ubuntu (7) XML
VC/MFC(16) XML
wince(23) XML
编码(8) XML
调试(8) XML
全志的公用函数(1) XML
通讯(3) XML
图片处理(20) XML

文章档案
2013年7月 (1)
2011年5月 (6)
2011年4月 (8)
2011年3月 (11)
2011年2月 (4)
2011年1月 (7)
2010年12月 (5)
2010年11月 (4)
2010年10月 (7)
2010年8月 (10)
2010年7月 (3)
2010年6月 (1)
2010年5月 (3)
2010年4月 (6)

搜索
搜索

最新评论 XML
1. re: sscanf()的一些使用说明
谢谢
--wewea
2. re: BCD码、十六进制与十进制互转
评论内容较长,点击标题查看
--hustcalm
3. re: C/C++/VC随机数总结
评论内容较长,点击标题查看
--游客

C、C++、MFC宏

C宏定义的简单总结

今天在网上突然发现了下面几个关于c代码中的宏定义的说明，回想下，好像在系统的代码中也见过这些零散的定义，但没有注意，看到别人总结了下，发现果然很有用，虽然不知有的道可用与否，但也不失为一种手段，所以就先把它摘抄下来，增加一点见识：

1,防止一个头文件被重复包含
```
#ifndef BODYDEF_H
#define BODYDEF_H
 //头文件内容
#endif
```
2,得到指定地址上的一个字节或字
```
#define  MEM_B( x )  ( *( (byte *) (x) ) )
#define  MEM_W( x )  ( *( (word *) (x) ) )
```
3,得到一个field在结构体(struct)中的偏移量
```
#define FPOS( type, field ) ( (dword) &(( type *) 0)-> field )
```
4,得到一个结构体中field所占用的字节数
```
#define FSIZ( type, field ) sizeof( ((type *) 0)->field )
```
5,得到一个变量的地址(word宽度)
```
#define  B_PTR( var )  ( (byte *) (void *) &(var) )
#define  W_PTR( var )  ( (word *) (void *) &(var) )
```
6,将一个字母转换为大写
```
#define  UPCASE( c ) ( ((c) >= "a" && (c) <= "z") ? ((c) - 0x20) : (c) )
```
7,判断字符是不是10进值的数字
```
#define  DECCHK( c ) ((c) >= "0" && (c) <= "9")
```
8,判断字符是不是16进值的数字
```
#define  HEXCHK( c ) ( ((c) >= "0" && (c) <= "9") ||((c) >= "A" && (c) <= "F") ||((c) >= "a" && (c) <= "f")
)
```
9,防止溢出的一个方法
```
#define  INC_SAT( val )  (val = ((val)+1 > (val)) ? (val)+1 : (val))
```
10,返回数组元素的个数
```
#define  ARR_SIZE( a )  ( sizeof( (a) ) / sizeof( (a[0]) ) )
```
11,使用一些宏跟踪调试
ANSI标准说明了五个预定义的宏名。它们是:
LINE (两个下划线), 对应%d
FILE 对应%s
DATE 对应%s
TIME 对应%s
STDC

宏中"#"和"##"的用法
我们使用#把宏参数变为一个字符串,用##把两个宏参数贴合在一起.
```
#define STR(s)     #s
#define CONS(a,b)  int(a##e##b)
Printf(STR(vck));          // 输出字符串"vck"
printf("%d\n", CONS(2,3));  // 2e3 输出:2000
```
当宏参数是另一个宏的时候
需要注意的是凡宏定义里有用"#"或"##"的地方宏参数是不会再展开.
```
#define A          (2)
#define STR(s)     #s
#define CONS(a,b)  int(a##e##b)
printf("%s\n", CONS(A, A));              // compile error
```
这一行则是:
```
printf("%s\n", int(AeA));
```
INT_MAX和A都不会再被展开，然而解决这个问题的方法很简单. 加多一层中间转换宏.
加这层宏的用意是把所有宏的参数在这层里全部展开，那么在转换宏里的那一个宏(_STR)就能得到正确的宏参数
```
#define STR(s)     _STR(s)         // 转换宏
#define CONS(a,b)  _CONS(a,b)      // 转换宏
printf("int max: %s\n", STR(INT_MAX));          // INT_MAX,int型的最大值,为一个变量 #include<climits>
```
输出为: int max: 0x7fffffff
```
STR(INT_MAX) -->  _STR(0x7fffffff) 然后再转换成字符串;

printf("%d\n", CONS(A, A));
```
输出为:200

CONS(A, A) --> _CONS((2), (2)) --> int((2)e(2))

"#"和"##"的一些应用特例
1、合并匿名变量名
```
#define ___ANONYMOUS1(type, var, line) type var##line
#define __ANONYMOUS0(type, line) ___ANONYMOUS1(type, _anonymous, line)
#define ANONYMOUS(type) __ANONYMOUS0(type, __LINE__)
```
例:ANONYMOUS(static int); 即: static int _anonymous70; 70表示该行行号;
第一层:ANONYMOUS(static int); --> __ANONYMOUS0(static int, __LINE__);
第二层: --> ___ANONYMOUS1(static int, _anonymous, 70);
第三层: --> static int _anonymous70;
即每次只能解开当前层的宏,所以__LINE__在第二层才能被解开;

2、填充结构
```
#define FILL(a) {a, #a}

enum IDD{OPEN, CLOSE};
typedef struct MSG{
  IDD id;
  const char * msg;
}MSG;

MSG _msg[] = {FILL(OPEN), FILL(CLOSE)};
```
相当于:
```
MSG _msg[] = {{OPEN, "OPEN"},
              {CLOSE, "CLOSE"}};
```

3、记录文件名
```
#define _GET_FILE_NAME(f) #f
#define GET_FILE_NAME(f) _GET_FILE_NAME(f)
static char FILE_NAME[] = GET_FILE_NAME(__FILE__);
```

4、得到一个数值类型所对应的字符串缓冲大小
```
#define _TYPE_BUF_SIZE(type) sizeof #type
#define TYPE_BUF_SIZE(type) _TYPE_BUF_SIZE(type)
char buf[TYPE_BUF_SIZE(INT_MAX)];
    --> char buf[_TYPE_BUF_SIZE(0x7fffffff)];
    --> char buf[sizeof "0x7fffffff"];
```
这里相当于:
```
char buf[11];
```

C++提供的编译预处理功能主要有以下三种:
（一）宏定义
（二）文件包含
（三）条件编译

在C++中，我们一般用const定义符号常量。很显然，用const定义常量比用define定义常量更好。
在使用宏定义时应注意的是:
（a）在书写#define 命令时，注意<宏名>和<字符串>之间用空格分开，而不是用等号连接。
（b）使用#define定义的标识符不是变量，它只用作宏替换，因此不占有内存。
（c）习惯上用大写字母表示<宏名>，这只是一种习惯的约定，其目的是为了与变量名区分，因为变量名通常用小写字母。
如果某一个标识符被定义为宏名后，在取消该宏定义之前，不允许重新对它进行宏定义。取消宏定义使用如下命令:
#undef<标识符>
其中，undef是关键字。该命令的功能是取消对<标识符>已有的宏定义。被取消了宏定义的标识符，可以对它重新进行定义。
宏定义可以嵌套，已被定义的标识符可以用来定义新的标识符。例如:
```
#define PI 3.14159265
#define R 10
#define AREA (PI*R*R)
```
单的宏定义将一个标识符定义为一个字符串，源程序中的该标识符均以指定的字符串来代替。前面已经说过，预处理命令不同于一般C++语句。因此预处理命令后通常不加分号。这并不是说所有的预处理命令后都不能有分号出现。由于宏定义只是用宏名对一个字符串进行简单的替换，因此如果在宏定义命令后加了分号，将会连同分号一起进行置换。

带参数的宏定义
带参数的宏定义的一般形式如下:
#define <宏名>(<参数表>) <宏体>
其中，<宏名>是一个标识符，<参数表>中的参数可以是一个，也可以是多个，视具体情况而定，当有多个参数的时候，每个参数之间用逗号分隔。<宏体>是被替换用的字符串，宏体中的字符串是由参数表中的各个参数组成的表达式。例如:
```
#define SUB(a,b) a-b
```
如果在程序中出现如下语句:
```
result=SUB(2, 3)
```
则被替换为:
```
result=2-3;
```

如果程序中出现如下语句：

result= SUB(x+1, y+2);

则被替换为：

result=x+1-y+2;

在这样的宏替换过程中，其实只是将参数表中的参数代入到宏体的表达式中去，上述例子中，即是将表达式中的a和b分别用2和3代入。

我们可以发现：带参的宏定义与函数类似。如果我们把宏定义时出现的参数视为形参，而在程序中引用宏定义时出现的参数视为实参。那么上例中的a和b就是形参，而2和3以及x+1和y+2都为实参。在宏替换时，就是用实参来替换<宏体>中的形参。

在使用带参数的宏定义时需要注意的是：

(1)带参数的宏定义的<宏体>应写在一行上，如果需要写在多行上时，在每行结束时，使用续行符 "\"结束，并在该符号后按下回车键，最后一行除外。

(2)在书写带参数的宏定义时，<宏名>与左括号之间不能出现空格，否则空格右边的部分都作为宏体。

例如：

```
#define ADD (x,y) x+y
```

将会把"(x,y)x+y"的一个整体作为被定义的字符串。

(3)定义带参数的宏时，宏体中与参数名相同的字符串适当地加上圆括号是十分重要的，这样能够避免可能产生的错误。例如,对于宏定义：

```
#define SQ(x) x*x
```

当程序中出现下列语句：

```
m=SQ(a+b);
```

替换结果为：

```
m=a+b*a+b;
```

这可能不是我们期望的结果，如果需要下面的替换结果：

```
m=(a+b)*(a+b);
```

应将宏定义修改为：

```
#define SQ(x) (x)*(x)
```

对于带参的宏定义展开置换的方法是：在程序中如果有带实参的宏（如"SUB(2,3)"），则按"#define"命令行中指定的字符串从左到右进行置换。如果串中包含宏中的形参（如a、b），则将程序语句中相应的实参（可以是常量、变量或者表达式）代替形参，如果宏定义中的字符串中的字符不是参数字符（如a-b中的-号），则保留。这样就形成了置换的字符串。 C++提供的编译预处理功能主要有以下三种：

（一）宏定义

（二）文件包含

（三）条件编译

在C++中，我们一般用const定义符号常量。很显然，用const定义常量比用define定义常量更好。

在使用宏定义时应注意的是：

（a）在书写#define 命令时，注意<宏名>和<字符串>之间用空格分开，而不是用等号连接。

（b）使用#define定义的标识符不是变量，它只用作宏替换，因此不占有内存。

（c）习惯上用大写字母表示<宏名>，这只是一种习惯的约定，其目的是为了与变量名区分，因为变量名通常用小写字母。

如果某一个标识符被定义为宏名后，在取消该宏定义之前，不允许重新对它进行宏定义。取消宏定义使用如下命令：

```
#undef<标识符>
```

其中，undef是关键字。该命令的功能是取消对<标识符>已有的宏定义。被取消了宏定义的标识符，可以对它重新进行定义。

宏定义可以嵌套，已被定义的标识符可以用来定义新的标识符。例如：

```
#define PI 3.14159265
#define R 10
#define AREA (PI*R*R)
```

单的宏定义将一个标识符定义为一个字符串，源程序中的该标识符均以指定的字符串来代替。前面已经说过，预处理命令不同于一般C++语句。因此预处理命令后通常不加分号。这并不是说所有的预处理命令后都不能有分号出现。由于宏定义只是用宏名对一个字符串进行简单的替换，因此如果在宏定义命令后加了分号，将会连同分号一起进行置换。

带参数的宏定义

带参数的宏定义的一般形式如下：

```
#define <宏名>(<参数表>) <宏体>
```

其中， <宏名>是一个标识符，<参数表>中的参数可以是一个，也可以是多个，视具体情况而定，当有多个参数的时候，每个参数之间用逗号分隔。<宏体>是被替换用的字符串，宏体中的字符串是由参数表中的各个参数组成的表达式。例如：

```
#define SUB(a,b) a-b
```

如果在程序中出现如下语句：

```
result=SUB(2, 3)
```

则被替换为：

```
result=2-3;
```

如果程序中出现如下语句：

```
result= SUB(x+1, y+2);
```

则被替换为：

```
result=x+1-y+2;
```

在这样的宏替换过程中，其实只是将参数表中的参数代入到宏体的表达式中去，上述例子中，即是将表达式中的a和b分别用2和3代入。

我们可以发现：带参的宏定义与函数类似。如果我们把宏定义时出现的参数视为形参，而在程序中引用宏定义时出现的参数视为实参。那么上例中的a和b就是形参，而2和3以及x+1和y+2都为实参。在宏替换时，就是用实参来替换<宏体>中的形参。

在使用带参数的宏定义时需要注意的是：

(1)带参数的宏定义的<宏体>应写在一行上，如果需要写在多行上时，在每行结束时，使用续行符 "\"结

束，并在该符号后按下回车键，最后一行除外。
(2)在书写带参数的宏定义时，<宏名>与左括号之间不能出现空格，否则空格右边的部分都作为宏体。
例如：
#define ADD (x,y) x+y
将会把"(x,y)x+y"的一个整体作为被定义的字符串。
(3)定义带参数的宏时，宏体中与参数名相同的字符串适当地加上圆括号是十分重要的，这样能够避免可能产生的错误。例如,对于宏定义：
#define SQ(x) x*x
当程序中出现下列语句：
m=SQ(a+b);
替换结果为：
m=a+b*a+b;
这可能不是我们期望的结果，如果需要下面的替换结果：
m=(a+b)*(a+b);
应将宏定义修改为：
#define SQ(x) (x)*(x)
对于带参的宏定义展开置换的方法是：在程序中如果有带实参的宏（如"SUB(2,3)"），则按"#define"命令行中指定的字符串从左到右进行置换。如果串中包含宏中的形参（如a、b），则将程序语句中相应的实参（可以是常量、变量或者表达式）代替形参，如果宏定义中的字符串中的字符不是参数字符（如a-b中的-号），则保留。这样就形成了置换的字符串。

解剖MFC自动生成的宏定义

一、关于DECLARE_MESSAGE_MAP宏定义
使用MFC向导，在ApplicationType页面选择DialogBased，生成一个对话框项目，Dialog类命名为CCapturePacketDlg，在CCapturePacketDlg.cpp中自动产生下列代码：

```
BEGIN_MESSAGE_MAP(CCapturePacketDlg, CDialog)
    ON_WM_PAINT()
END_MESSAGE_MAP()
```

1. 先来分析ON_WM_PAINT()，在头文件"afxmsg.h"有它的宏定义，如下：

```
#define  ON_WM_PAINT() \
    { WM_PAINT, 0, 0, 0, AfxSig_vv, \
     (AFX_PMSG)(AFX_PMSGW) \
     (static_cast < void (AFX_MSG_CALL CWnd:: * )( void ) > ( & ThisClass :: OnPaint)) },
```

说明：层次序号x.y.z表示x为根节点也就是上面代码中的行号，y、z为上一级的定义展开。
2.1 #define WM_PAINT 0x000F
2.2 AfxSig_vv = AfxSig_v_v_v
2.2.1 enum AfxSig::AfxSig_v_v_v = 19

3.1 AFX_PMSG:typedef void (AFX_MSG_CALL CCmdTarget::*AFX_PMSG)(void); //为一个函数指针
3.2 AFX_PMSGW:typedef void (AFX_MSG_CALL CWnd::*AFX_PMSGW)(void); //为一个函数指针

将ON_WM_PAINT()完全展开：

```
{
    0x000F,
    0,
    0,
    0,
    19,
    //Converts OnPaint to the type of CCmdTarget finally. Derive Class 's pointer -> Base Class's pointer
    (AFX_MSG_CALL CCmdTarget::*)((AFX_MSG_CALL CWnd::*)(static_cast< void (AFX_MSG_CALL CWnd::*)(void) >(&ThisClass :: OnPaint))
  }
```

2. 再来分析BEGIN_MESSAGE_MAP(CCapturePacketDlg, CDialog)，在"afxwin.h"中有定义：

```
#define BEGIN_MESSAGE_MAP(theClass, baseClass) \
   PTM_WARNING_DISABLE \
   const AFX_MSGMAP* theClass::GetMessageMap() const \
      { return GetThisMessageMap(); } \
   const AFX_MSGMAP* PASCAL theClass::GetThisMessageMap() \
   { \
      typedef theClass ThisClass;                       \
      typedef baseClass TheBaseClass;                   \
      static const AFX_MSGMAP_ENTRY _messageEntries[] =  \
      {
```

2.1 PTM_WARNING_DISABLE:
#define PTM_WARNING_DISABLE \
 __pragma(warning(push)) \ //#pragma warning(push [,n]),Where n represents a warning level (1 through 4).
                                //The pragma warning(push) stores the current warning state for all warnings.
 __pragma(warning(disable : 4867))//Do not issue the specified warning message(s).
//http://msdn2.microsoft.com/en-us/2c8f766e.aspx
// Allows selective modification of the behavior of compiler warning messages.
3.1 struct AFX_MSGMAP
 {
  3.1.1 const AFX_MSGMAP* (PASCAL* pfnGetBaseMap)();
  3.1.2 const AFX_MSGMAP_ENTRY* lpEntries;
 };
3.1.2 struct AFX_MSGMAP_ENTRY
 {
  UINT nMessage;   // windows message
  UINT nCode;   // control code or WM_NOTIFY code
  UINT nID;   // control ID (or 0 for windows messages)
  UINT nLastID;   // used for entries specifying a range of control id's
  UINT_PTR nSig;   // signature type (action) or pointer to message #
  3.1.2.1 AFX_PMSG pfn;   // routine to call (or special value)
 };
 3.1.2.1 typedef void (AFX_MSG_CALL CCmdTarget::*AFX_PMSG)(void);

5.1 #define PASCAL   __stdcall
将BEGIN_MESSAGE_MAP(CCapturePacketDlg, CDialog)**完全展开**:

```
1  __pragma(warning( push )) __pragma(warning( disable : 4867 ))
2     const struct AFX_MSGMAP* CCapturePacketDlg::GetMessageMap() const
3     {
4        return GetThisMessageMap();
5     }
6     const struct AFX_MSGMAP* __stdcall CCapturePacketDlg::GetThisMessageMap()
7     {
8        typedef CCapturePacketDlg ThisClass;
9        typedef CDialog TheBaseClass;
10       static const struct AFX_MSGMAP_ENTRY _messageEntries[] =
11       {
```

3 **最后分析**END_MESSAGE_MAP(), **在**"afxwin.h"**中有定义**:

```
1  #define END_MESSAGE_MAP() \
2        {0, 0, 0, 0, AfxSig_end, (AFX_PMSG)0 } \
3     }; \
4        static const AFX_MSGMAP messageMap = \
5        { &TheBaseClass::GetThisMessageMap, &_messageEntries[0] }; \
6        return &messageMap; \
7     }                              \
8     PTM_WARNING_RESTORE
```

2.1 AfxSig_end:enum AfxSig.AfxSig_end = 0
2.2 AFX_PMSG:typedef void (AFX_MSG_CALL CCmdTarget::*AFX_PMSG)(void);//**函数指**针

4.1 struct AFX_MSGMAP
 {
  const AFX_MSGMAP* (PASCAL* pfnGetBaseMap)();
  const AFX_MSGMAP_ENTRY* lpEntries;
 };

8.1 #define PTM_WARNING_RESTORE \
 __pragma(warning(pop))
//**pop** restores the state of all warnings (including 4705, 4706, and 4707) to what it was at the beginning of the code.

·最后将

```
1  BEGIN_MESSAGE_MAP(CCapturePacketDlg, CDialog)
2     ON_WM_PAINT()
3  END_MESSAGE_MAP()
```

完全展开为:

```
__pragma(warning( push )) __pragma(warning( disable : 4867 ))
    const struct AFX_MSGMAP* CCapturePacketDlg::GetMessageMap() const
    {
        return GetThisMessageMap();
    }
    const struct AFX_MSGMAP* __stdcall CCapturePacketDlg::GetThisMessageMap()
    {
        typedef CCapturePacketDlg ThisClass;
        typedef CDialog TheBaseClass;
        static const struct AFX_MSGMAP_ENTRY _messageEntries[] =
        {
            {
                0x000F,
                0,
                0,
                0,
                19,
                //Converts OnPaint to the type of CCmdTarget finally. Derive Class 's pointer -> Base Class's pointer
                (AFX_MSG_CALL CCmdTarget::*)((AFX_MSG_CALL CWnd::*)(static_cast< void (AFX_MSG_CALL CWnd::*)(void) >(&ThisClass :: OnPaint))
            },
            //add others
            {
                0,
                0,
                0,
                0,
                0,
                (AFX_PMSG)0
            }
        };
        static const struct AFX_MSGMAP messageMap =
        {
            &TheBaseClass::GetThisMessageMap,
            &_messageEntries[0]
        };
        return &messageMap;
    }
__pragma(warning( pop ))

```

其中GetMessageMap()是在哪里声明的呢？在CCapturePacketDlg的定义中有一个这样的宏：
DECLARE_MESSAGE_MAP()
老办法查看它的定义：

```
#define DECLARE_MESSAGE_MAP() \
protected: \
    static const AFX_MSGMAP* PASCAL GetThisMessageMap(); \
    virtual const AFX_MSGMAP* GetMessageMap() const; \
```

注意函数为static，即它们是类的函数。函数中的static变量实际也在类对象未生成之前已经存在。(这种说法不知道是否正确？)
小结：
每次用MFC类向导生成一个类时，系统会在类的声明部分添加两个方法的声明：
GetThisMessageMap()，GetMessageMap()。在类的实现部分.cpp文件中加上这两个方法的定义。
当然这所有的代码都是由系统生成的，如果我们要定义自己的消息处理函数呢，例如，我们要添加一个按钮(ID为：IDC_BUTTON1)的单击处理函数我们可以添加宏ON_NOTIFY(NM_CLICK, IDC_BUTTON1, OnMyClick)，OnMyClick为自定义函数，但是他必须与ON_NOTIFY中的函数原型一致。

二、关于DECLARE_DYNCREATE宏
使用MFC向导，在ApplicationType页面选择SingleDocument，生成一个单文档项目，Document类命名为CDynamicDoc，在CDynamicDoc.h中自动产生DECLARE_DYNCREATE(CDynamicDoc)，CDynamicDoc.cpp中产生IMPLEMENT_DYNCREATE(CDynamicDoc, CDocument)。
1、展开CDynamicDoc.h中的DECLARE_DYNCREATE(CDynamicDoc)：

```
// not serializable, but dynamically constructable
    #define DECLARE_DYNCREATE(class_name) \
        DECLARE_DYNAMIC(class_name) \
        static CObject* PASCAL CreateObject();
```

3.1如下定义：

```
#ifdef _AFXDLL
    #define DECLARE_DYNAMIC(class_name) \
```

```
3     protected: \
4        static CRuntimeClass* PASCAL _GetBaseClass(); \
5     public: \
6        static const CRuntimeClass class##class_name; \
7        static CRuntimeClass* PASCAL GetThisClass(); \
8        virtual CRuntimeClass* GetRuntimeClass() const; \
```

so the result(DECLARE_DYNCREATE(CDynamicDoc)) of combining the above two is following:

```
1  protected:
2        static CRuntimeClass* PASCAL _GetBaseClass();
3     public:
4        static const CRuntimeClass classCDynamicDoc;
5        static CRuntimeClass* PASCAL GetThisClass();
6        virtual CRuntimeClass* GetRuntimeClass() const;
7        static CObject* PASCAL CreateObject();
```

2、展开CDynamicDoc.cpp中的IMPLEMENT_DYNCREATE(CDynamicDoc, CDocument):

```
1  #define IMPLEMENT_DYNCREATE(class_name, base_class_name) \
2     CObject* PASCAL class_name::CreateObject() \
3        { return new class_name; } \
4     IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, 0xFFFF, \
5        class_name::CreateObject, NULL)
```

4.1如下定义:

```
1  #define IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, wSchema, pfnNew, class_init)
\
2     CRuntimeClass* PASCAL class_name::_GetBaseClass() \
3        { return RUNTIME_CLASS(base_class_name); } \
4     AFX_COMDAT const CRuntimeClass class_name::class##class_name = { \
5        #class_name, sizeof(class class_name), wSchema, pfnNew, \
6           &class_name::_GetBaseClass, NULL, class_init }; \
7     CRuntimeClass* PASCAL class_name::GetThisClass() \
8        { return _RUNTIME_CLASS(class_name); } \
9     CRuntimeClass* class_name::GetRuntimeClass() const \
10       { return _RUNTIME_CLASS(class_name); } \
```

4.1.2 CRuntimeClass如下定义:

```
1  struct CRuntimeClass
2     {
3     // Attributes
4        LPCSTR m_lpszClassName;
5        int m_nObjectSize;
6        UINT m_wSchema; // schema number of the loaded class
7        CObject* (PASCAL* m_pfnCreateObject)(); // NULL => abstract class
8     #ifdef _AFXDLL
9        CRuntimeClass* (PASCAL* m_pfnGetBaseClass)();
10    #else
11       CRuntimeClass* m_pBaseClass;
12    #endif
13
14    // Operations
15       CObject* CreateObject();
16       BOOL IsDerivedFrom(const CRuntimeClass* pBaseClass) const;
17
18       // dynamic name lookup and creation
19       static CRuntimeClass* PASCAL FromName(LPCSTR lpszClassName);
20       static CRuntimeClass* PASCAL FromName(LPCWSTR lpszClassName);
21       static CObject* PASCAL CreateObject(LPCSTR lpszClassName);
22       static CObject* PASCAL CreateObject(LPCWSTR lpszClassName);
23
24    // Implementation
25       void Store(CArchive& ar) const;
26       static CRuntimeClass* PASCAL Load(CArchive& ar, UINT* pwSchemaNum);
27
28       // CRuntimeClass objects linked together in simple list
29       CRuntimeClass* m_pNextClass;       // linked list of registered classes
30       const AFX_CLASSINIT* m_pClassInit;
31    };
```

4.1.2.30 AFX_CLASSINIT如下定义:(这个变量非常重要,它完成了将新的类加在List头部的功能,List中的节点类型就是CRuntimeClass)

```
1//////////////////////////////////////////////////////////////////////////////
2     // Basic object model
```

```
3
4    // generate static object constructor for class registration
5    void AFXAPI AfxClassInit(CRuntimeClass* pNewClass);
6    struct AFX_CLASSINIT
7        { AFX_CLASSINIT(CRuntimeClass* pNewClass) { AfxClassInit(pNewClass); } };
8    //C:\Program Files\Microsoft Visual Studio 8\VC\atlmfc\src\mfc\objcore.cpp Line157
9    void AFXAPI AfxClassInit(CRuntimeClass* pNewClass)
10   {
11       AFX_MODULE_STATE* pModuleState = AfxGetModuleState();
12       AfxLockGlobals(CRIT_RUNTIMECLASSLIST);
13       pModuleState->m_classList.AddHead(pNewClass);
14       AfxUnlockGlobals(CRIT_RUNTIMECLASSLIST);
15   }
16   //可以将AfxClassInit()函数的功能简单的如下表示:
17   AFX_CLASSINIT::AFX_CLASSINIT(CRuntimeClass* pNewClass)
18   {
19       pNewClass->m_pNextClass = CRuntimeClass::pFirstClass;
20       CRuntimeClass::pFirstClass = pNewClass;
21   }
```

4.1.3 RUNTIME_CLASS如下定义:

```
1  #define RUNTIME_CLASS(class_name) (class_name::GetThisClass())
```

4.1.4 AFX_COMDAT如下定义:

```
1  #define AFX_COMDAT __declspec(selectany)
```

说明:"#"——operator (#) converts macro parameters to string literals without expanding the parameter definition.

"##"——operator (**##**), which is sometimes called the "merging" operator, is used in both object-like and function-like macros.

4.1.8 _RUNTIME_CLASS如下定义:

```
1  #define _RUNTIME_CLASS(class_name) ((CRuntimeClass*)(&class_name::class##class_name))
```

so the result(IMPLEMENT_DYNCREATE(CDynamicDoc, CDocument)) of combining the aboves is following:

```
1//CDynamicDoc, CDocument->class_name, base_class_name
2  static  CObject* PASCAL CDynamicDoc::CreateObject()
3  {
4      return new CDynamicDoc;
5  }
6
7  static CRuntimeClass* PASCAL CDynamicDoc::_GetBaseClass()
8  {
9      return CDocument::GetThisClass()
10 }
11
12 __declspec(selectany) static const CRuntimeClass CDynamicDoc::classCDynamicDoc =
13 {
14     "CDynamicDoc"
15     , sizeof(class CDynamicDoc)
16     , 0xFFFF
17     , CDynamicDoc::CreateObject
18     , &CDynamicDoc::_GetBaseClass
19     , NULL
20     , NULL
21 };
22
23 static CRuntimeClass* PASCAL CDynamicDoc::GetThisClass()
24 {
25     return ((CRuntimeClass*)(&CDynamicDoc::classCDynamicDoc));
26 }
27
28 CRuntimeClass* CDynamicDoc::GetRuntimeClass() const
29 {
30     return ((CRuntimeClass*)(&CDynamicDoc::classCDynamicDoc));
31 }
```

小结:**注意了,上面的成员变量、很多函数都是**static

如果你想看这些宏的简化版,可以参考侯老的《深入浅出MFC**》,如下:**

```
1  //in header file
2  class CView : public CWnd
3  {
```

```
public:
    static CRuntimeClass classCView;
    virtual CRuntimeClass* GetRuntimeClass() const;
    //.....
};
//in implementation file
 static char _lpszCView = "CView";
 CRuntimeClass CView::classCView =
{
    _lpszCView
    , sizeof(CView)
    , 0xFFF
    , NULL
    , &CWnd::classCWnd
    , NULL
};
 static AFX_CLASSINIT _init_CView(&CView::classCView)
{
    (&CView::classCView)->m_pNextClass = CRuntimeClass::pFirstClass;
    CRuntimeClass::pFirstClass = &CView::classCView;
}
 CRuntimeClass* CView::GetRuntimeClass() const
{
    return &CView::classCView;
}
```

其中他将CRuntimeClass**简化定**义为:

```
struct CRuntimeClass
{
// Attributes
      LPCSTR m_lpszClassName;
      int m_nObjectSize;
      UINT m_wSchema; // schema number of the loaded class
      CObject* (PASCAL* m_pfnCreateObject)(); // NULL => abstract class
      CRuntimeClass* m_pBaseClass;

      // CRuntimeClass objects linked together in simple list
      static CRuntimeClass* pFirstClass; // start of class list
      CRuntimeClass* m_pNextClass;       // linked list of registered classes
};
```

三、宏DECLARE_SERIAL(CStroke)、IMPLEMENT_SERIAL(CStroke, CObject, 1), 给**出它们的宏定**义**及结果**:

```
//declaration file
#define DECLARE_SERIAL(class_name) \
    _DECLARE_DYNCREATE(class_name) \
    AFX_API friend CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb);

    #define _DECLARE_DYNCREATE(class_name) \
    _DECLARE_DYNAMIC(class_name) \
    static CObject* PASCAL CreateObject();

     #define _DECLARE_DYNAMIC(class_name) \
     protected: \
        static CRuntimeClass* PASCAL _GetBaseClass(); \
     public: \
        static CRuntimeClass class##class_name; \
        static CRuntimeClass* PASCAL GetThisClass(); \
        virtual CRuntimeClass* GetRuntimeClass() const; \
//implement file
#define IMPLEMENT_SERIAL(class_name, base_class_name, wSchema) \
    CObject* PASCAL class_name::CreateObject() \
        { return new class_name; } \
    extern AFX_CLASSINIT _init_##class_name; \
    _IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, wSchema, \
        class_name::CreateObject, &_init_##class_name) \
    AFX_CLASSINIT _init_##class_name(RUNTIME_CLASS(class_name)); \
    CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb) \
        { pOb = (class_name*) ar.ReadObject(RUNTIME_CLASS(class_name)); \
          return ar; } \

    // generate static object constructor for class registration
    void AFXAPI AfxClassInit(CRuntimeClass* pNewClass);
```

```
31    struct AFX_CLASSINIT
32        { AFX_CLASSINIT(CRuntimeClass* pNewClass) { AfxClassInit(pNewClass); } };
33
34    void AFXAPI AfxClassInit(CRuntimeClass* pNewClass)
35    {
36        AFX_MODULE_STATE* pModuleState = AfxGetModuleState();
37        AfxLockGlobals(CRIT_RUNTIMECLASSLIST);
38        pModuleState->m_classList.AddHead(pNewClass);
39        AfxUnlockGlobals(CRIT_RUNTIMECLASSLIST);
40    }
41
42
43    #define _IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, wSchema, pfnNew, class_
init) \
44    CRuntimeClass* PASCAL class_name::_GetBaseClass() \
45        { return RUNTIME_CLASS(base_class_name); } \
46    AFX_COMDAT CRuntimeClass class_name::class##class_name = { \
47        #class_name, sizeof(class class_name), wSchema, pfnNew, \
48            &class_name::_GetBaseClass, NULL, class_init }; \
49    CRuntimeClass* PASCAL class_name::GetThisClass() \
50        { return _RUNTIME_CLASS(class_name); } \
51    CRuntimeClass* class_name::GetRuntimeClass() const \
52        { return _RUNTIME_CLASS(class_name); } \
53
54    #define _RUNTIME_CLASS(class_name) ((CRuntimeClass*)(&class_name::class##class_name))
55
56    #define RUNTIME_CLASS(class_name) (class_name::GetThisClass())
```

```
1  //header file
2     protected:
3        static CRuntimeClass* PASCAL _GetBaseClass();
4     public:
5        static CRuntimeClass classCStroke;
6        static CRuntimeClass* PASCAL GetThisClass();
7        virtual CRuntimeClass* GetRuntimeClass() const;
8        static CObject* PASCAL CreateObject();
9        AFX_API friend CArchive& AFXAPI operator>>(CArchive& ar, CStroke* &pOb);
10 //implement file
11    //static
12    static CObject* PASCAL CStroke::CreateObject()
13    {
14        return new CStroke;
15    }
16    //static
17    static CRuntimeClass* PASCAL CStroke::GetThisClass();
18    {
19        return ((CRuntimeClass*)(&CStroke::classCStroke))
20    }
21    //static
22    static CRuntimeClass* PASCAL CStroke::_GetBaseClass()
23    {
24        return (CObject::GetThisClass());
25    }
26    //static
27    static AFX_COMDAT CRuntimeClass CStroke::classCStroke =
28    {
29        "CStroke"
30        , sizeof(class CStroke)
31        , 1
32        , CStroke::CreateObject
33        , &class_name::_GetBaseClass
34        , NULL
35        , &_init_CStroke
36    };
37    CRuntimeClass* CStroke::GetRuntimeClass() const
38    {
39        return ((CRuntimeClass*)(&CStroke::classCStroke));
40    }
41    extern struct AFX_CLASSINIT _init_CStroke;
42    struct AFX_CLASSINIT _init_CStroke
43    {
```

```
    void AFXAPI AfxClassInit(CRuntimeClass* CStroke)
    {
        AFX_MODULE_STATE* pModuleState = AfxGetModuleState();
        AfxLockGlobals(CRIT_RUNTIMECLASSLIST);
        pModuleState->m_classList.AddHead(CStroke);
        AfxUnlockGlobals(CRIT_RUNTIMECLASSLIST);
    }
};
CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb)
{
    pOb = (CStroke*) ar.ReadObject(RUNTIME_CLASS(CStroke));
    return ar;
}
```

总结，一旦RUNTIME_CLASS(CStroke)由#define RUNTIME_CLASS(class_name) (class_name::GetThisClass())
也就是CStroke::GetThisClass() 即
CStroke::classCStroke =
{
"CStroke"
, sizeof(class CStroke)
, 1
, CStroke::CreateObject
, &class_name::_GetBaseClass
, NULL
, &_init_CStroke
}
其中，由extern AFX_CLASSINIT _initCStroke可知_init_CStroke是一个结构体AFX_CLASSINIT的对象，此结构体有构造函数：

```
void AFXAPI AfxClassInit(CRuntimeClass* pNewClass);
struct AFX_CLASSINIT
    { AFX_CLASSINIT(CRuntimeClass* pNewClass) { AfxClassInit(pNewClass); } };

void AFXAPI AfxClassInit(CRuntimeClass* pNewClass)
{
    AFX_MODULE_STATE* pModuleState = AfxGetModuleState();
    AfxLockGlobals(CRIT_RUNTIMECLASSLIST);
    pModuleState->m_classList.AddHead(pNewClass);
    AfxUnlockGlobals(CRIT_RUNTIMECLASSLIST);
}
```

所以一旦返回classCStroke，也就调用了_init_CStroke的构造函数即将类CStroke添加到了全局变量m_classList类的List中了，同时在变量classCStroke中，也可以得到类CStroke的名称、大小、一个CStroke的对象、类CStroke的基类以及AFX_CLASSINIT结构的一个对象。

posted on 2010-10-26 12:00 心羽 阅读(957) 评论(0) 编辑 收藏 引用 所属分类: 需求分析

【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

相关文章：

通用权限管理系统设计篇（三）——概要设计说明书（转）

通用权限管理设计篇（二）——数据库设计（转）

通用权限管理设计篇（一）（转）