 IMPLEMENT\_DYNAMIC是实现“运行时类型识别”宏，与之相对应的是DECLARE\_DYNAMIC(声明“运行时类型识别”宏)。也就是说你在.CPP文件中如果看见有IMPLEMENT\_DYNAMIC，则在.H文件中必定有DECLARE\_DYNAMIC的声明。   
DECLARE\_DYNAMIC/DEClARE\_DYNAMIC是为了确定运行时对象属于哪一个类而定义的宏。   
DEClARE\_DYNCREATE/IMPLEMENT\_DYNCREATE是为了“动态创建"类的实例而定义的宏。new可以用来创建对象，但不是动态的。比如说，你要在程序中实现根据拥护输入的类名来创建类的实例，下面的做法是通不过的：   
char szClassName[60];   
cin >> szClassName;   
CObject\* pOb=new szClassName; //通不过   
这里就要用到DEClARE\_DYNCREATE/IMPLEMENT\_DYNCREATE定义的功能了。

定义：

//////////////////////////////////////////////////////////////////////////////  
// Helper macros for declaring CRuntimeClass compatible classes  
  
#ifdef \_AFXDLL  
#define DECLARE\_DYNAMIC(class\_name) /  
protected: /  
    static CRuntimeClass\* PASCAL \_GetBaseClass(); /  
public: /  
    static const CRuntimeClass class##class\_name; /  
    static CRuntimeClass\* PASCAL GetThisClass(); /  
    virtual CRuntimeClass\* GetRuntimeClass() const; /  
  
#define \_DECLARE\_DYNAMIC(class\_name) /  
protected: /  
    static CRuntimeClass\* PASCAL \_GetBaseClass(); /  
public: /  
    static CRuntimeClass class##class\_name; /  
    static CRuntimeClass\* PASCAL GetThisClass(); /  
    virtual CRuntimeClass\* GetRuntimeClass() const; /  
  
#else  
#define DECLARE\_DYNAMIC(class\_name) /  
public: /  
    static const CRuntimeClass class##class\_name; /  
    virtual CRuntimeClass\* GetRuntimeClass() const; /  
  
#define \_DECLARE\_DYNAMIC(class\_name) /  
public: /  
    static CRuntimeClass class##class\_name; /  
    virtual CRuntimeClass\* GetRuntimeClass() const; /  
  
#endif

引用：

DECLARE\_DYNAMIC/IMPLEMENT\_DYNAMIC  
  
#define DECLARE\_DYNAMIC(class\_name)/      
public:/  
    static CRuntimeClass class##class\_name;/  
    //声明一个类型为CRuntimeClass的静态public成员变量，变量名是由字符串"class"  
    //与所指定的类的类名组成。举例而言，如果你写DECLARE\_DYNAMIC(CMyView)，则等于声明了一个  
    // static CRuntimeClass classCMyView静态变量  
  
    virtual CRuntimeClass\* GetRuntimeClass() const;/  
    //声明一个虚函数，函数名为GetRuntimeClass，返回值为CRuntimeClass类型的指针  
    //无参数，并且是个const函数  
  
#define IMPLEMENT\_DYNAMIC(class\_name,bass\_class\_name)/  
       \_IMPLEMENT\_RUNTIMECLASS(class\_name,base\_class\_name,0xFFFF,NULL)  
  
#define \_IMPLEMENT\_RUNTIMECLASS(class\_name,base\_class\_name,wSchema,pfnNew)/  
    static char \_lpsz##class\_name[]= #class\_name;/  
    //定义一个C类型字符串静态变量，变量名由"\_lpsz"和指定类的类名组成，变量值为该指定类型的名字  
    //比如是CMyView，那么定义的就是static char \_lpszCMyView="CMyView";  
  
    CRuntimeClass class\_name::class##class\_name = {/  
        \_lpsz##class\_name,sizeof(class\_name),wSchema,pfnNew,/  
            RUNTIME\_CLASS(base\_class\_name),NULL};/  
    //给之前在DECLARE\_DYNAMIC里定义的CRuntimeClass类型的静态成员变量赋值  
    //当然，除最后一个m\_pNextClass没有赋值（赋值为NULL，它由下面的结构处理）  
  
    static AFX\_CLASSINIT \_init\_##class\_name(&class\_name::class##class\_name);/  
    //初始化一个名为"\_init\_##class\_name"的AFX\_CLASSINIT静态结构，主要作用是给指定的class\_name的  
    //class##class\_name静态变量的最后一个成员m\_pNextClass赋值，具体见下面解释AFX\_CLASSINIT中  
  
    CRuntimeClass\* class\_name::GetRuntimeClass() const/  
        { return &class\_name::class##class\_name;}/  
    //之前在DECLARE\_DYNAMIC里定义的GetRuntimeClass的实现，很简单，就一个return语句。  
  
#define RUNTIME\_CLASS(class\_name)/  
        (&class\_name::class##class\_name)  
//这部分之所以单独define出一个宏，主要是为了方便从某个指定的class直接得到它的CRuntimeclass静态成员  
  
//以下是解释AFX\_CLASSINIT结构，注意，这不是一个宏  
//为了看得更加清楚，我按照struct定义的惯常格式来写这个struct的定义  
struct AFX\_CLASSINIT {  
    AFX\_CLASSINIT(CRuntimeClass \*pNewClass);  
};  
  
AFX\_CLASSINIT::AFX\_CLASSINIT(CRuntimeClass \*pNewClass)  
{  
    pNewClass->m\_pNextClass = CRuntimeClass::pFirstClass;  
    //让m\_pNextClass指向pFirstClass所指的CRuntimeClass变量  
  
    CRuntimeClass::pFirstClass = pNewClass;  
    //让pFirstClass指向pNewClass所指的变量，也就是本class的CRuntimeClass静态变量  
}