导航
C++博客
首页
新随笔
联系
聚合XML
管理

< 2009年8月 >

日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

随笔档案
2011年4月 (1)
2009年8月 (27)
2009年7月 (21)

extern "C" __declspec(dllexport) __declspec(dllimport) 和 def

前面的extern "C" __declspec(dllexport) __declspec(dllimport)都是用于函数或者变量，甚至类的声明的（可以把extern "C"放在class的前面，但是编译器会忽略掉，最后产生的还是C++修饰符，而不是C修饰符）这样的用法有个好处就是下面的代码可以在混有类的函数和变量上使用下面的宏，虽然对类不起作用：

```
#ifdef __cplusplus
extern "C"
{
//函数声明
//变量声明，变量一般前面都有extern
//类声明，这个不起作用，编译器直接忽略掉class前面的extern “C”
#ifdef __cplusplus
}
#endif
```

C 和C++ 对应不同的调用约定，产生的修饰符也各不相同，如下：

调用约定	extern "C" 或 .c 文件	.cpp、.cxx 或 /TP
C 命名约定 (**__cdecl**)	_test	?test@@ZAXXZ
Fastcall 命名约定 (**__fastcall**)	@test@0	?test@@YIXXZ
标准调用命名约定 (**__stdcall**)	_test@0	?test@@YGXXZ

__declspec(dllexport) __declspec(dllimport)一般也是使用宏的形式：

```
#ifdef ONEDLL_EXPORTS
#define ONEDLL_API __declspec(dllexport)
#else
#define ONEDLL_API __declspec(dllimport)
#endif
```

这样在DLL代码本身就是__declspec(dllexport)，在使用DLL的程序中就变成了__declspec(dllimport)，这两标志分别用来指明当前的函数将被导出，和当前函数是被导入的。

上面的两个宏结合一下就是下面这样的了：

```
// 下列 ifdef 块是创建使从 DLL 导出更简单的
// 宏的标准方法。此 DLL 中的所有文件都是用命令行上定义的 ONEDLL_EXPORTS
```

统计
随笔 - 49
文章 - 0
评论 - 6
引用 - 0

留言簿(1)
给我留言
查看公开留言
查看私人留言

阅读排行榜
1. extern "C" __declspec(dllexport) __declspec(dllimport) 和 def(15937)
2. def文件一个比较详细的例子(13900)
3. 一个C++DLL的简单例子(1638)
4. exe、dll的进入点，以及main、winmain、dllmain的关系(1301)
5. 一个time的例子 SYSTEMTIME (1120)

评论排行榜
1. VC 2008 sp1 中 deque的erase函数的问题(2)
2. def文件一个比较详细的例子(1)
3. 很奇怪，input1和input2总是相同(1)
4. 随机数函数rand和种子函数srand的使用(1)
5. 刚刚上课两天，记录点小代码（求素数的）(0)

```
// 符号编译的。在使用此 DLL 的
// 任何其他项目上不应定义此符号。这样，源文件中包含此文件的任何其他项目都会将
// ONEDLL_API 函数视为是从 DLL 导入的，而此 DLL 则将用此宏定义的
// 符号视为是被导出的。
#ifdef ONEDLL_EXPORTS
#define ONEDLL_API __declspec(dllexport)
#else
#define ONEDLL_API __declspec(dllimport)
#endif

// 此类是从 OneDll.dll 导出的
#ifdef __cplusplus
extern "C"
{
#endif
class ONEDLL_API COneDll {
public:
    COneDll(void);
    ~COneDll(void);

    // TODO: 在此添加您的方法。
    int m_a;
    int m_b;
    int *m_p;
    int m_n;

    void AddValue();

};

extern ONEDLL_API int nOneDll;

ONEDLL_API int fnOneDll(void);

#ifdef __cplusplus
}
#endif
```

如果调用模块和被调用模块都是C++（而且是同一种编成环境，如VC，甚至需要同一版本的VC），那么就不需要extern “C”了，因为这个标志的作用就是用在函数和变量声明前，无论是调用模块，还是被调用模块，都将生成C修饰符，调用模块将需要C修饰符的函数，而被调用模块将产生C修饰符的函数，所以这个标志在两者都是C++的时候使用并不受影响，不使用这个标志，也不受影响。

但是如果C模块要调用C++ 模块，那么C++模块就需要使用extern “C”，当然C不用，由于是在头文件的声明中使用，所以使用下面的宏能够使得这个头文件也在C中顺利使用：

```
#ifdef __cplusplus
extern "C"
{
//函数声明
//变量声明，变量一般前面都有extern
//类声明，这个不起作用，编译器直接忽略掉class前面的extern “C”
#ifdef __cplusplus
}
#endif
```

如果C++模块要调用C模块，那么C++模块还是需要extern “C”，当然C不用，由于是在头文件的声明中使用，所以使用上面的宏同样能够使得这个头文件也在C中顺利使用。

总结一下就是加上extern “C”在什么情况下都没错，但是要注意函数重载的问题。

def文件是一种比较麻烦的方法，下面是MSDN中的部分内容：

模块定义 (.def) 文件是包含一个或多个描述 DLL 各种属性的 Module 语句的文本文件。如果不使用 **__declspec(dllexport)** 关键字导出 DLL 的函数，则 DLL 需要 .def 文件。

.def 文件必须至少包含下列模块定义语句：
1.文件中的第一个语句必须是 LIBRARY 语句。此语句将 .def 文件标识为属于 DLL。LIBRARY 语句的后面是 DLL 的名称。链接器将此名称放到 DLL 的导入库中。
2.EXPORTS 语句列出名称，可能的话还会列出 DLL 导出函数的序号值。通过在函数名的后面加上 @ 符和一个数字，给函数分配序号值。当指定序号值时，序号值的范围必须是从 1 到 N，其中 N 是 DLL 导出函数的个数。

例如，包含实现二进制搜索树的代码的 DLL 看上去可能像下面这样：

```
LIBRARY   BTREE
EXPORTS
  Insert   @1
  Delete   @2
  Member   @3
  Min   @4
```

提示：

如果希望优化 DLL 文件的大小，请对每个导出函数使用 **NONAME** 属性。使用 **NONAME** 属性时，序号存储在 DLL 的导出表中而非函数名中。如果导出许多函数，这样做可以节省相当多的空间。

其实**__declspec(dllexport)**的作用就是让编译器按照某种预定的方式（前面大致解释了这种方式的规则）来输出导出函数及变量的符号，而def文件则是自己为每一个函数和变量指定导出符号，所以def是一个非自动化，手工很强的方式，不是特殊情况的话，实在没有必要浪费这些时间。
还有一个问题，就是使用def会把调用方式和**__declspec(dllexport)**的作用全部覆盖掉，所以还需要自己处理调用方式不同产生的错误。
一般使用def文件的情况是你需要使用运行时加载，并且需要使用GetProcAddress函数获得函数地址，这个函数需要直接指明函数产生的导出符号，而可以自己指定导出符号的方式就是使用def。
def文件的具体语法可以看看msdn。

posted on 2009-08-03 17:12 把握命运 阅读(15937) 评论(0) 编辑 收藏 引用