(31条消息) 8.DLL导出C++类_renwu-CSDN博客_c++ 导出类

DLL中不仅可以导出函数和变量，也可以导出C++类。只需要在导出**类名前关键字class后**加上_declspec(dllexport)，就可以实现导出类

1.DLL简单导出类代码

```
1. class _declspec(dllexport) Stu
2. {
3. public:
4. Stu(int a);
5. void print();
6.
7. private:
8. int _a;
9. };
```

实例：

.cpp文件：

```
1.
2.
3.
4. #include "stdafx.h"
5.
6. #define _DLL_EXPORTS
7. #include "func.h"
8.
9. #include <iostream>
10. using namespace std;
11.
12. Stu::Stu(int a)
13. {
14. this->_a = a;
15. }
```

```
16.
17. void Stu::print()
18. {
19.     cout << _a << endl;
20. }
```

func.h文件：

```
1. #pragma once
2.
3. #ifdef _DLL_EXPORTS
4. #define DLL_API _declspec(dllexport)
5. #else
6. #define DLL_API _declspec(dllimport)
7. #endif
8.
9. class DLL_API Stu
10. {
11. public:
12. Stu(int a);
13. void print();
14.
15. private:
16. int _a;
17. };
```

在dependency中查看导出的DLL：

可以发现，导出了四个函数，一个为构造函数、一个为拷贝构造函数、一个为析构函数，还有一个就是我们自己定义的print函数

测试：

```
1. #include <iostream>
2. using namespace std;
3. #include "func.h"
4.
5. #pragma comment(lib,"DynamicLib.lib")
```

```
6.
7. int main()
8. {
9. Stu stu1(666);
10.     stu1.print();
11.
12. system("pause");
13. return 0;
14. }
```

结果：

2.简单导出类的缺点

这种简单导出类的方式，除了导出的东西太多、使用者对类的实现依赖太多以外，还有其他问题：必须保证使用同一种编译器。导出类的本质是导出类里的函数，因为语法上直接导出了类，没有对函数的调用方式、重命名进行设置，导致了产生的dll并不通用。

3.导出类的较好方式

定义一个抽象类（都是纯虚函数），调用者跟dll共用一个抽象类的头文件，dll中实现此抽象类的派生类，dll最少只需要提供一个用于获取抽象类对象指针的接口。

4.面向抽象设计优点

这种方式利用了C++类的虚函数，类似**COM思想**，采用接口跟实现分离，可以使得工程的结构更清晰，使用者只需要知道接口，而无需知道具体实现，产生的DLL通用没有特定环境限制。

5.注意事项

调用者跟DLL共用一个抽象类的头文件，调用者依赖于DLL的东西很少，只需要知道抽象类的接口，以及获取对象指针的导出函数，对象内存空间的申请和释放都在DLL模块中完成

6.代码演示

DLL生成代码：

.cpp文件：

```
1.
2.
```

```
 3.
 4. #include "stdafx.h"
 5.
 6. #define _DLL_EXPORTS
 7. #include "func.h"
 8.
 9. #include <iostream>
10. using namespace std;
11.
12. class Cat :public IAnimal
13. {
14. public:
15. Cat()
16.         {
17.                 cout << "Cat is created" << endl;
18.         }
19.         ~Cat()
20.         {
21.                 cout << "Cat is deleted" << endl;
22.         }
23. virtual void eat()
24.         {
25.                 cout << "Cats eat fish" << endl;
26.         }
27. virtual void sleep()
28.         {
29.                 cout << "Cats sleep" << endl;
30.         }
31. virtual void delObj()
32.         {
33. delete this;
34.         }
35. };
36.
37. extern "C" DLL_API IAnimal *GetCat()
38. {
39. return new Cat;
40. }
```

func.h文件：

```
#pragma once

#ifdef _DLL_EXPORTS
#define DLL_API _declspec(dllexport)
#else
#define DLL_API _declspec(dllimport)
#endif

class IAnimal
{
public:
virtual void eat() = 0;
virtual void sleep() = 0;
virtual void delObj() = 0;
};

extern "C" DLL_API IAnimal *GetCat();
```

在dependency中查看导出的DLL：

E	Ordinal ^	Hint	Function	Entry Point
C	1 (0x0001)	0 (0x0000)	GetCat	0x00011064

发现导出的只有一个类名

调用者代码：

```
#include <iostream>
using namespace std;
#include "func.h"

#pragma comment(lib,"DynamicLib.lib")

int main()
{
        IAnimal *p = GetCat();
        p->eat();
        p->sleep();
        p->delObj();

```

```
14. system("pause");
15. return 0;
16. }
```

执行结果：