《C++ 程序设计》插件机制（示例）

vcg

2021 年 6 月 25 日

1 插件的机制

插件用于扩展某个应用程序的功能。它从原应用程序事先定义好的接口出发，所编出来的模块。应用程序并不知道插件的功能以及实现细节（如使用的类名）。chrome 等应用程序支持插件机制。chrome 网上应用店上有很多 chrome 插件。

本示例模拟插件机制。分为两个部分：应用程序，即主控部分；具体的插件。

主控部分: CPluginManager 用于插件管理，包括插件注册、插件卸载、插件查询；CPluginBase 为插件基类，提供了功能扩展的接口。

具体插件：CAdd 模拟具体插件功能

主控

```cpp
//主程序
#include <iostream>
#include <string>
#include "PluginManager.h"
//注意，主控程序中并不知道具体插件的存在，所以没有
//#include "PluginAdd.h", #include "PluginMult.h"

using namespace std;
extern void LoadPlugins();

CPluginManager *g_manager = 0;
int main()
{
    g_manager = new CPluginManager();

    //加载插件
    LoadPlugins();

    //测试插件
    double a = 1.0, b = 2.0;

    CPluginBase *activePlugin;

    //假设选择一个插件
    activePlugin = g_manager->GetPlugin("Add"); //模拟选择方式
    if(activePlugin)
    {
        cout << activePlugin->Calculate(a, b) << endl;
    }

    //假设选择了另一个插件
    activePlugin = g_manager->GetPlugin("Mult");//模拟选择方式
    if(activePlugin)
    {
        cout << activePlugin->Calculate(a, b) << endl;
    }

    delete g_manager;
    return 0;
}
```

插件加载模拟

```cpp
//主程序，模拟插件的加载，选择
#include "PluginManager.h"
#include "PluginAdd.h"
#include "PluginMult.h"
```

```

void LoadPlugins()
{
    //加载插件时总会有一个自动调用的函数。
    RegisterPlugAdd();
    RegisterPlugMult();
}
```

插件基类 CPluginBase

```
//主程序
#pragma once
#include <string>

//插件基类
class CPluginBase
{
private:
    std::string id;
public:
    CPluginBase(const std::string & id ="") { this->id = id; }
    virtual ~CPluginBase() {};

    std::string GetID()const { return id; }
    void SetID(const std::string & id) { this->id = id; }

    virtual double Calculate(double first, double second) = 0;
};

class CPluginManager;
extern CPluginManager *g_manager;
```

插件管理类 CPluginManager

```
//主程序
#pragma once
#include "PluginBase.h"
#include <vector>

//插件管理类
class CPluginManager
{
private:
    struct PluginInfo
    {
        CPluginBase* plugin;
        std::string      id;
    };
    std::vector<PluginInfo> m_pluginInfos;
```

```
16 public:
17     CPluginManager() {}
18     ~CPluginManager() {//清空内存
19         for (unsigned int i = 0; i < m_pluginInfos.size(); i++)
20             delete m_pluginInfos[i].plugin;
21         m_pluginInfos.clear();
22     }
23 
24     //注册插件
25     bool RegisterPlugin(const std::string &id, CPluginBase* plugin)
26     {
27         for (unsigned int i = 0; i < m_pluginInfos.size(); i++)
28         {
29             if (m_pluginInfos[i].id == id)return false;//已经注册过
30         }
31         PluginInfo info;
32         info.id = id;
33         info.plugin = plugin;    //创建一个插件对象
34 
35         m_pluginInfos.push_back(info);
36         return true;
37     }
38 
39     CPluginBase* GetPlugin(const std::string &id)
40     {
41         for (unsigned int i = 0; i < m_pluginInfos.size(); i++)
42         {
43             if (m_pluginInfos[i].id == id)return m_pluginInfos[i].plugin;//已经注册
                过
44         }
45         return NULL;
46     }
47 };
```

插件：CAdd

```
1  //加法插件，外部程序
2  #pragma once
3  #include "PluginBase.h"
4 
5  class CAdd : public CPluginBase
6  {
7  public:
8      double Calculate(double first, double second) { return first + second; };
9  };
10 
11 inline void RegisterPlugAdd()
12 {
13     CPluginBase *plugin = new CAdd();
```

```
14      bool bRet = g_manager->RegisterPlugin("Add", plugin);
15      if (!bRet)delete plugin;
16  }
```

插件：CMult

```
1   //插件乘法，外部程序
2   #pragma once
3   #include "PluginBase.h"
4
5   class CMult : public CPluginBase
6   {
7   public:
8       double Calculate(double first, double second) { return first*second; };
9   };
10
11  inline void RegisterPlugMult()
12  {
13      CPluginBase *plugin = new CMult();
14      bool bRet = g_manager->RegisterPlugin("Add", plugin);
15      if (!bRet)delete plugin;
16  }
```