河北师范大学软件学院
Software College of Hebei Normal University

# 第十五章 软件体系结构风格

任课教师：武永亮
wuyongliang@edu2act.org

# 上节回顾

## 课程内容

- 插件技术背景
- 插件机制（Mechanism）
- 插件技术基础——动态链接库（DLL）
- 基于插件技术的系统案例

## 课程内容

- 插件技术背景
- 插件机制（Mechanism）
- 插件技术基础——动态链接库（DLL）
- 基于插件技术的系统案例

## 插件Plug-in定义

- 插件（Plug-in，又称addin、add-in、addon或add-on，又译外挂）也称为扩展，是一种遵循一定规范的应用程序接口编写出来的程序，主要是用来扩展软件功能。

# 典型的插件系统

- Microsoft Office
- Graphic software
- Web browsers
- Software development environment
- ……

# 典型的插件系统


开心网
首页
好友
群
消息
邀请 | 帐户 | 隐私 | 退出
照片
日记
记录
朋友买卖
争车位
姓名缘分
动他一下
礼物
音乐
网盘
朋友印象
婚外情
就这么回事，哥哥不再来玩了。开心网的网站策划很强大，多谢。
性别：男
出生日期：2008年08月08日
现居住地：北京
工作情况：IT - 至今
修改头像
个人资料
帐户管理
隐私设置
功能地图
博客设置
访问我的博客
消息中心
短消息：0条新
留言板：0条新
评论：0条新
系统消息：0条新
留言回复：0条新
评论回复：0条新
好友动态
目前没有好友动态
日记
最近来访
所有访客>>

## 使用插件的意义

- 支持特性扩展
- 支持第三方开发
- 降低应用程序大小

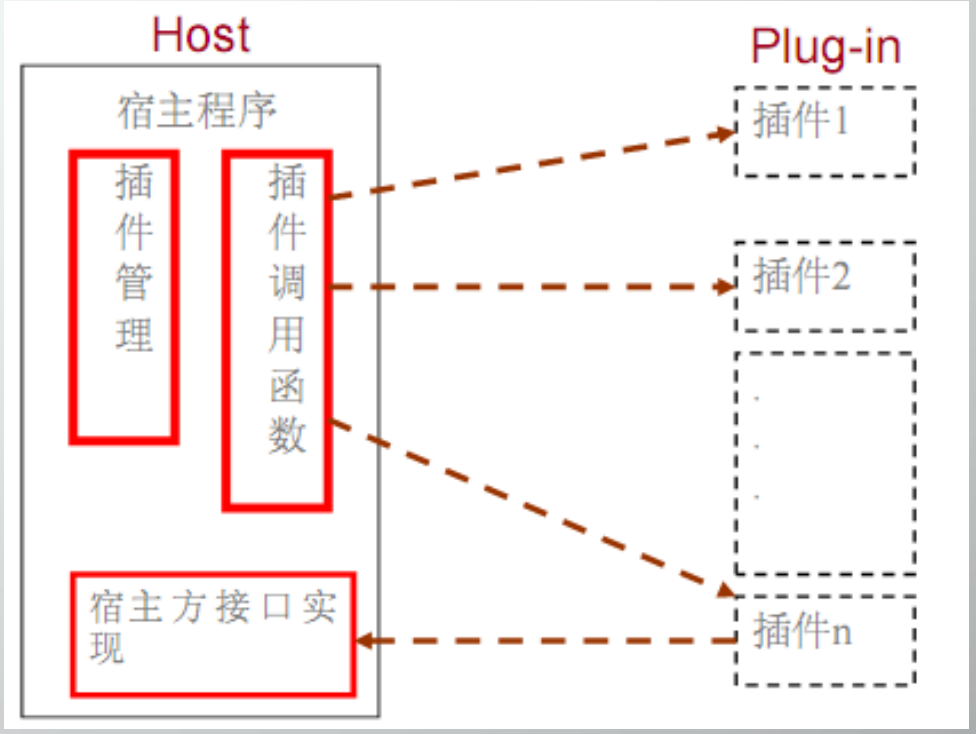
**插件不是简单的功能扩展！**

## 课程内容

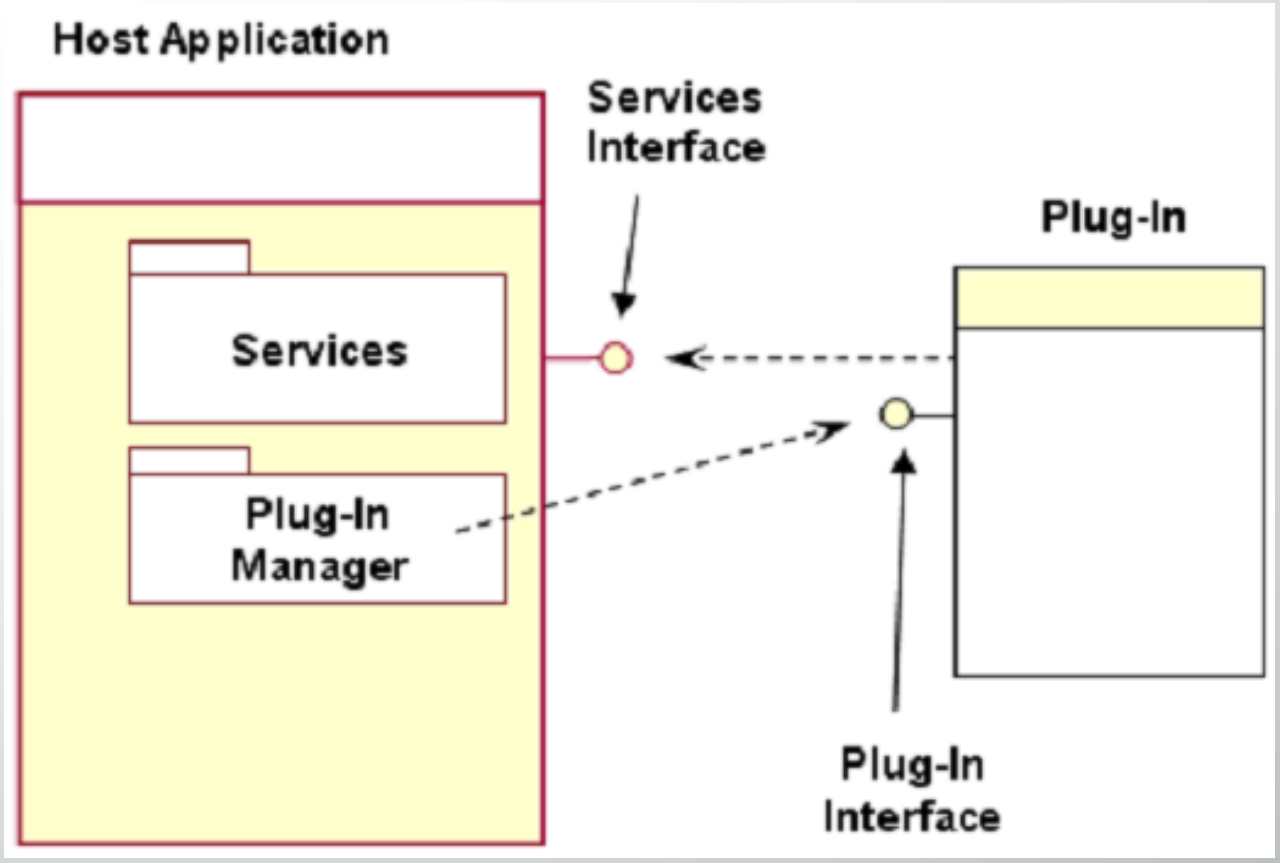
- 插件技术背景
- 插件机制（Mechanism）
- 插件技术基础——动态链接库（DLL）
- 基于插件技术的系统案例

# 什么是插件技术

- 在程序设计过程中，把应用程序分成两部分
  - 宿主程序（host）
  - 插件（plug-in）
  - 契约（contract）



# 插件系统架构


Host Application
Services Interface
Plug-In
Services
Plug-In Manager
Plug-In Interface

## 插件系统实现步骤

- 设计契约（取决于业务背景）
- 设计插件
- 设计宿主程序

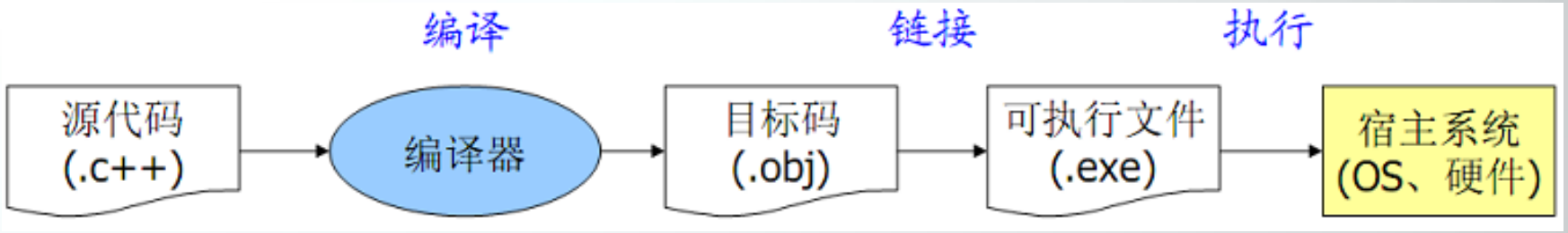
## 课程内容

- 插件技术背景
- 插件机制（Mechanism）
- 插件技术基础——动态链接库（DLL）
- 基于插件技术的系统案例

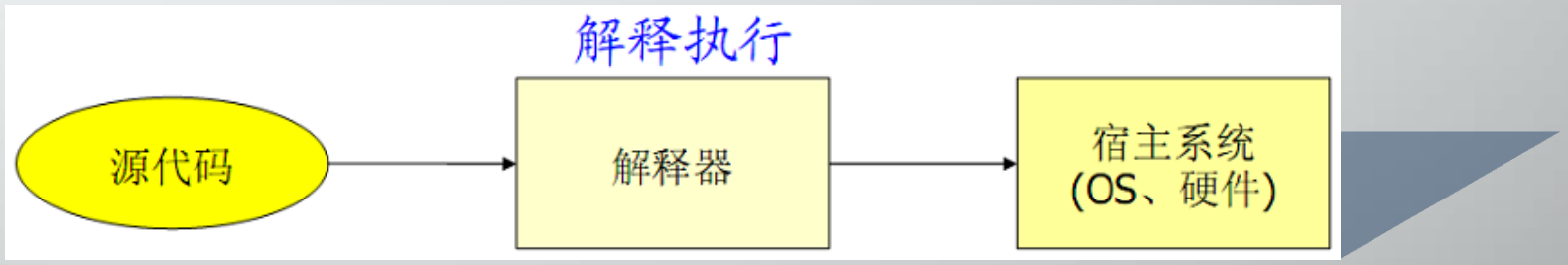
## 编译器和解释器

- 早期的程序语言环境就分为编译（Complication）和解释（Interpretation）两大类
  - 二者在目标、功能与实现上有何差别？
  - 二者的性能有何差别？

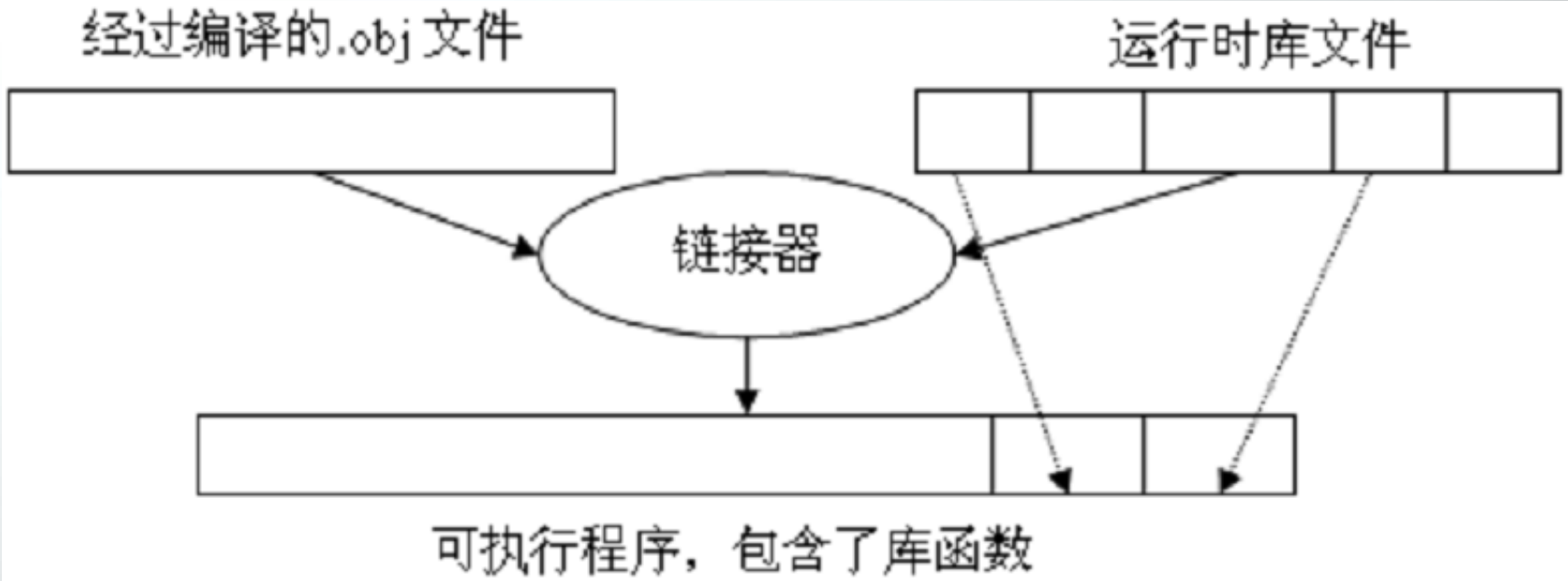
## 编译器（Compiler）




## 解释器（Interpreter）



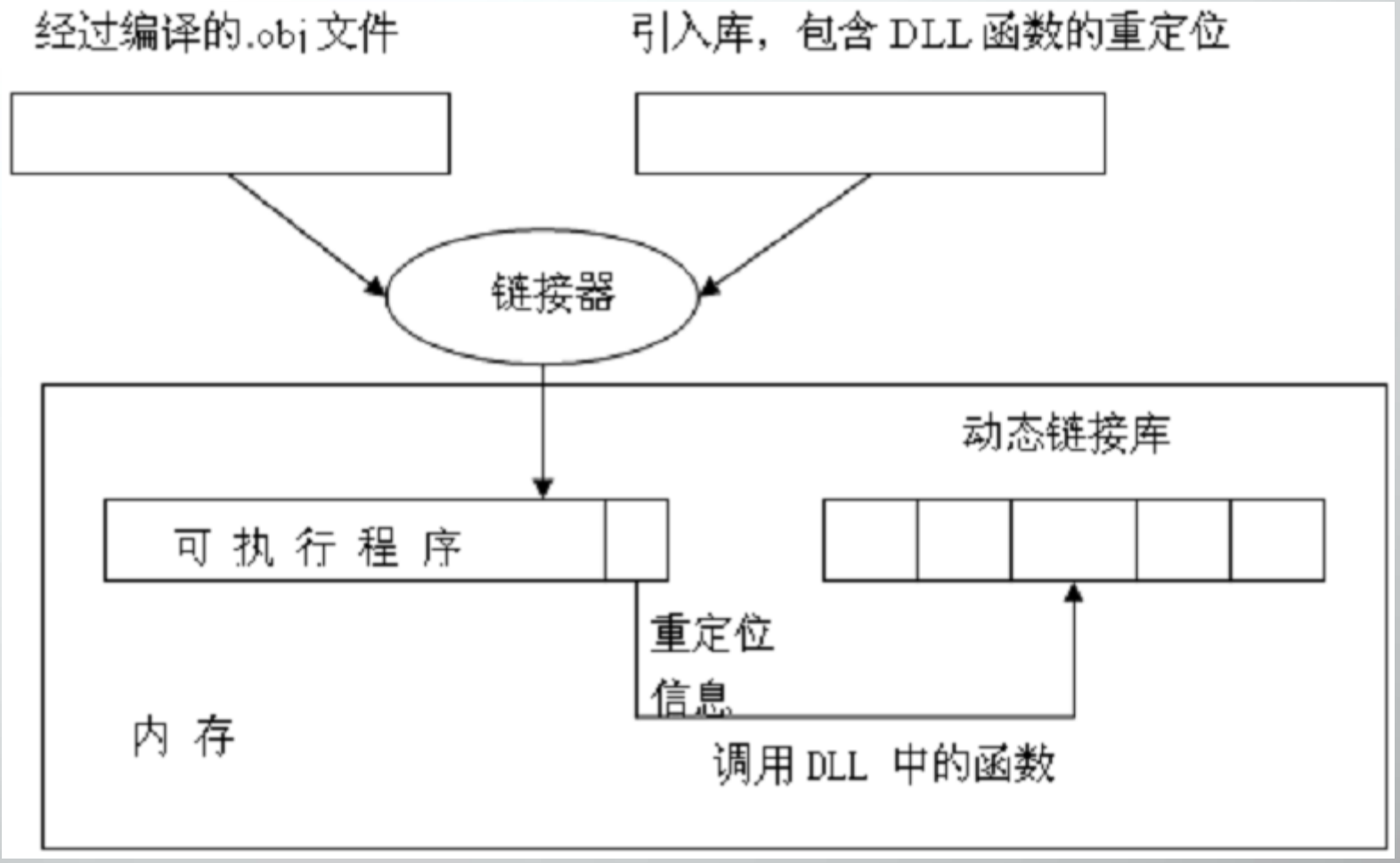
# 静态链接



## 动态链接

- 动态链接指的是在链接时并没有将库函数中的函数复制到应用程序的可执行文件。链接是在程序中运行时动态来执行的。
- 采用动态链接方式的库文件即为DLL（Dynamic Linkable Library）。

# 动态链接


经过编译的.obj文件
引入库，包含DLL函数的重定位
链接器
动态链接库
可执行程序
重定位
信息
内存
调用DLL中的函数

## DLL到进程地址空间的映射

- 要调用DLL中的函数，首先必须把DLL的文件映像映射到调用进程的地址空间中。有两种方法可以实现这一映射：
  - 装入时动态链接（load-time dynamic linking）
  - 运行时动态链接（run-time dynamic linking）

## 装入时动态链接

- 当应用程序运行时，操作系统在装载应用程序是要查看exe文件映像的内容，并将所有被引用的DLL文件映像映射到进程的地址空间中。
- 系统在寻找DLL文件时，按以下目录次序搜索
  - 包含可执行应用程序的目录
  - 当前目录
  - Windows系统目录，使用GetSystemDirectory函数可以返回该目录路径
  - Windows目录，使用GetWindowsDirectory函数可以返回该目录路径。

**如果按以上次序找不到DLL，应用程序即被终止。**

## 运行时动态链接

- 将链接推迟到运行期间，那么正确的DLL就可以判定，然后被动态链接，这便是运行时动态链接的基本思路

1）HMODULE LoadLirary(LPCTSTR lpszLibFile)；

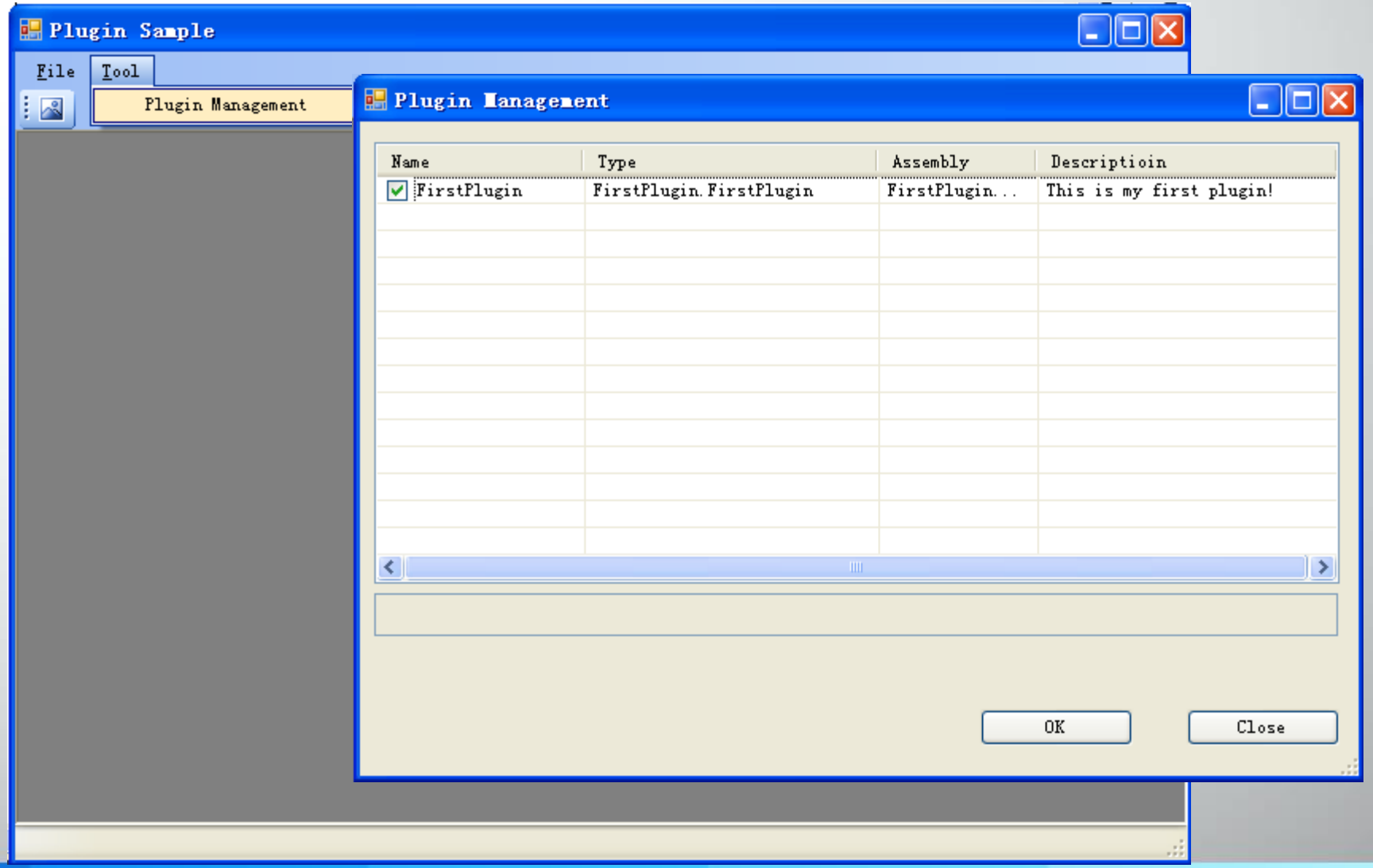
2）BOOL FreeLibrary(HMODULE hLibModule)；

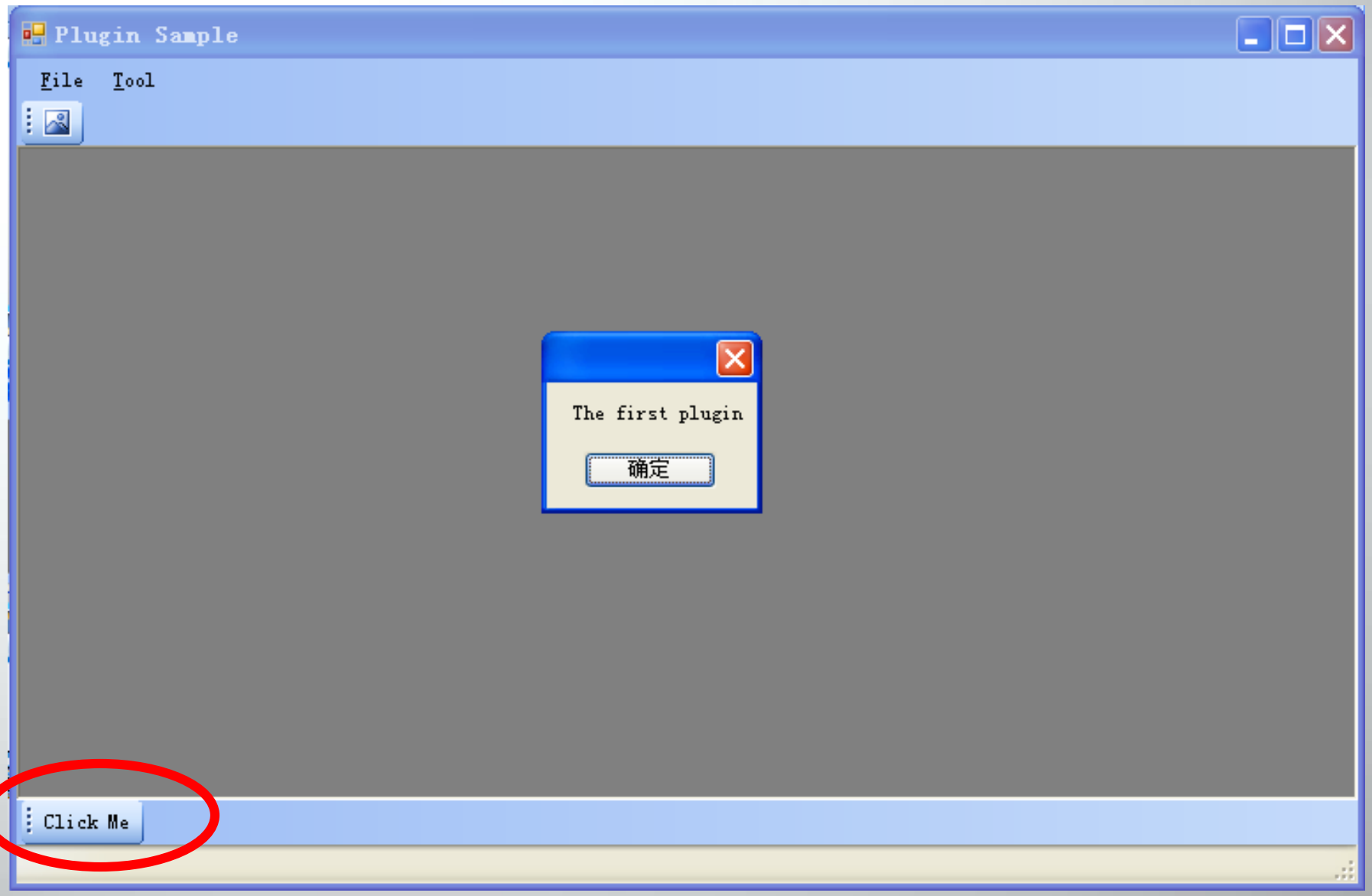
3）FARPROC GetProcAddress（HMODULE hModule, LPCTSTR lpszProc）；

## 课程内容

- 插件技术背景
- 插件机制（Mechanism）
- 插件技术基础——动态链接库（DLL）
- 基于插件技术的系统案例

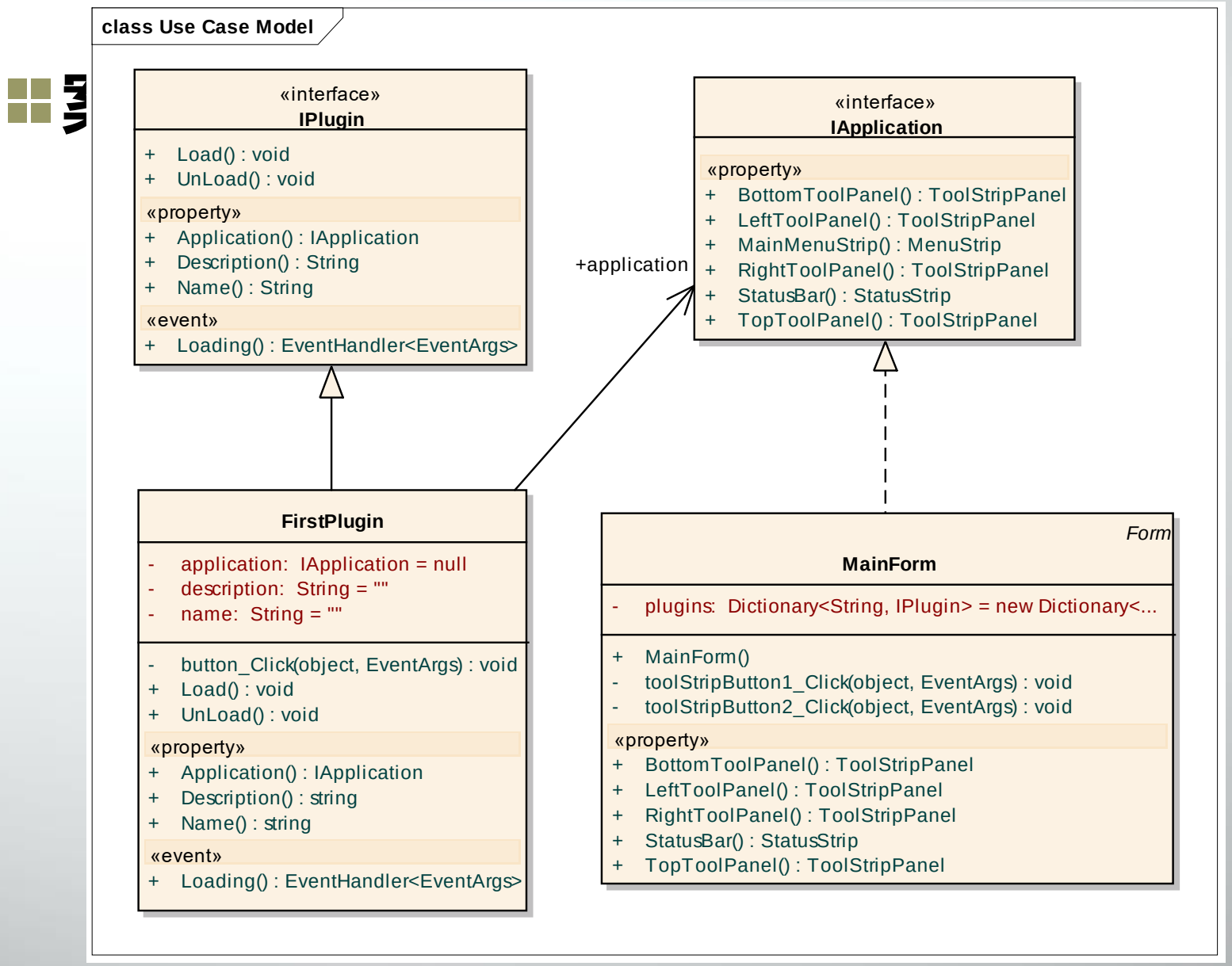
# 案例：基于C#实现插件管理器




Plugin Sample
File
Tool
The first plugin
确定
Click Me

# 案例：基于C#实现插件管理器

- Net技术实现插件机制的步骤
  - 动态加载
    - Assembly类的几个静态的Load（Load，LoadFile，LoadFrom）方法来动态加载
  - 获得所有类型
  - 判定是否为插件接口类型
  - 创建插件实例


class Use Case Model
«interface»
IPlugin
+ Load() : void
+ UnLoad() : void
«property»
+ Application() : IApplication
+ Description() : String
+ Name() : String
«event»
+ Loading() : EventHandler<EventArgs>
«interface»
IApplication
«property»
+ BottomToolPanel() : ToolStripPanel
+ LeftToolPanel() : ToolStripPanel
+ MainMenuStrip() : MenuStrip
+ RightToolPanel() : ToolStripPanel
+ StatusBar() : StatusStrip
+ TopToolPanel() : ToolStripPanel
+application
FirstPlugin
- application: IApplication = null
- description: String = ""
- name: String = ""
- button_Click(object, EventArgs) : void
+ Load() : void
+ UnLoad() : void
«property»
+ Application() : IApplication
+ Description() : string
+ Name() : string
«event»
+ Loading() : EventHandler<EventArgs>
Form
MainForm
- plugins: Dictionary<String, IPlugin> = new Dictionary<...
+ MainForm()
- toolStripButton1_Click(object, EventArgs) : void
- toolStripButton2_Click(object, EventArgs) : void
«property»
+ BottomToolPanel() : ToolStripPanel
+ LeftToolPanel() : ToolStripPanel
+ RightToolPanel() : ToolStripPanel
+ StatusBar() : StatusStrip
+ TopToolPanel() : ToolStripPanel

# 设计契约：应用程序契约

```
namespace PluginFramework
{
    public interface IApplication
    {
        ToolStripPanel LeftToolPanel { get; }
        ToolStripPanel RightToolPanel { get; }
        ToolStripPanel TopToolPanel { get; }
        ToolStripPanel BottomToolPanel { get; }

        MenuStrip MainMenuStrip { get; }
        StatusStrip StatusBar { get; }
    }
}
```

# 设计契约：插件契约

```
namespace PluginFramework
{
    public interface IPlugin
    {
        IApplication Application { get;set;}
        String Name { get;set;}
        String Description { get;set;}
        void Load();
        void UnLoad();

        event EventHandler<EventArgs> Loading;
    }
}
```

# 插件代码

```
public void Load()
{
    if (application != null && application.BottomToolPanel != null)
    {
        //创建一个向主程序添加的ToolStrip
        ToolStrip sampleToolStrip = new ToolStrip();
        sampleToolStrip.Name = "SampleToolStrip";
        ToolStripButton button = new ToolStripButton("Click Me");
        button.Click += new EventHandler(button_Click);
        sampleToolStrip.Items.Add(button);

        //在主程序的底端添加ToolStrip
        application.BottomToolPanel.Controls.Add(sampleToolStrip);

    }
}
```

# 插件代码

```
void button_Click(object sender, EventArgs e)
{
    MessageBox.Show("The first plugin");
}

public void UnLoad()
{
    foreach (ToolStrip ts in application.BottomToolPanel.Controls)
    {
        if (ts.Name == "SampleToolStrip")
        {
            application.BottomToolPanel.Controls.Remove(ts);
            break;
        }
    }
}
```

# 宿主程序代码

```
string[] files = Directory.GetFiles(Application.StartupPath);// + @"/plugins");
try
{
    foreach (string file in files)
    {
        string subName = file.Substring(file.LastIndexOf("."));
        if (subName == ".dll")
        {
            Assembly assembly = Assembly.LoadFile(file);
            Type[] types = assembly.GetTypes();
            foreach (Type type in types)
            {
                if (type.FullName == "FirstPlugin.FirstPlugin")
                {
                    IPlugin instance = (IPlugin)Activator.CreateInstance(type);
                    instance.Application = this;
                    instance.Load();
                    //plugins[assembly.FullName.ToString()] = instance;
                    plugins[type.FullName.ToString()] = instance;
                }
            }
            break;
        }
    }
```

## 扩展案例：音频播放器

- 通过模拟的音频播放器来介绍插件的实现技术
- 一般音频播放器都有这样一些基本功能：
  - 装载音频文件（LoadFile）
  - 播放（Play）
  - 暂停（Pause）
  - 停止（Stop）
- 本例将提供这四个功能。但宿主程序本身不会直接实现这些功能，而是调用插件的实现。每个插件支持一种音频格式，所以每个插件的功能实现都是不同的。

## 扩展案例：音频播放器——插件的实现

- 创建一个动态链接库Plug.dll，为了支持四个基本功能，它输出相应的四个函数：
  - Void LoadFile ( const char *szFileName ) ;
  - Void Play ( ) ;
  - Void Pause ( ) ;
  - Void Stop ( ) ;

## 扩展案例：音频播放器——插件的实现

- 为了使宿主程序在运行时能知道这个插件可以支持什么格式的音频文件，插件程序还应输出一个函数供宿主程序查询用：
  - Void GetSupportedFormat(char *szFormat)
    if(szFormat!=0) {strcpy(szFormat,“mp3”);}
- 至此，这个插件就制作完了。可以依样画葫芦再做一个Plug2.dll,它“支持”.wma文件。

## 扩展案例：音频播放器——宿主的实现

- 宿主是一个基于对话框的标准Windows程序。
- 它启动时会搜索约定目录（可以约定所有插件都存放在宿主程序所在目录的Plugins子目录下）
- 使用Win32函数Loadliabrary加载所有插件。
- 每加载一个插件DLL，就调用另一个Win32函数GetSupportedFormat的地址，并调用此函数返回插件所支持的格式名（即是音频文件的的扩展名）
- 然后把（格式名，DLL句柄）二元组保存下来。

## 扩展案例：音频播放器——宿主的实现

- 当用户通过菜单打开文件时，宿主程序会根据扩展名决定调用哪个插件的LoadFile函数
- 并指明此插件DLL的句柄为当前使用的插件的DLL句柄（比如保存到变量m_hInst中）
- 此后当用户通过按钮调用Play等其他功能时，就调用句柄为m_hInst的插件的相应功能。

## 扩展案例：音频播放器——宿主的实现

```
typedef void（*PLAY）();
if(m_hInst)
{
    PLAY Play =
    (PLAY)::GetProcAddress(m_hInst,”Play”);
    Play();
}
```

**另外，当程序退出时，应该调用FreeLiabrary函数卸载插件**

# 小结

# 作业

- 通过插件机制实现音频播放器，或者自拟题目实现插件系统

Thank You，谢谢！