李延春（华北计算技术研究所 100083）

软件插件技术的原理与实现

Theory and Implementation of Plug-in Technology

摘要： 基于插件的应用系统拥有良好的可扩充性、可定制性和可维护性，本文论述了插件技术的基本原理，并给出了三种不同的实现插件系统的方法。

关键词： 插件 plug-in 动态链接 COM 组件类别

1 引言

插件是近年来十分常见的一种技术，插件结构有助于编写有良好的扩充和定制功能的应用程序。许多软件甚至操作系统或其外壳程序都使用了这种技术。一套著名的使用插件机制的软件是Winamp，Winamp早期的成功虽然在于其快速的解码引擎，但在MP3播放器中能够保持长久的霸主地位，也正是由于内置了健全的插件功能：后期的Winamp中增加的MIDI、MOD、WAVE等音乐格式的播放功能完全是靠插件实现的。本文将论述插件技术的基本原理，并给出三种不同的实现插件系统的方法。

2 插件系统的基本原理

插件的本质是在不修改程序主体的情况下对软件功能进行加强，当插件的接口被公开时，任何人都可以自己制作插件来解决一些操作上的不便或增加一些功能。一个插件框架包括两个部分：主程序(host)和插件(plug-in)。主程序即是"包含"插件的程序。插件必须实现若干标准接口，由主程序在与插件通信时调用。

编程实现方面包括两个部分，一部分是主体程序的插件处理机制，用来进行初始化每个插件的过程，并且管理好每个插件接口。另一部分是插件的接口函数定义，将所有的插件接口函数进行封装，以便开发者自由调用。

而最重要的部分则是插件和主程序之间的交互。插件一般是一个遵循了某些特定规则的DLL，而主程序将所有插件接口在内存中的地址传递给插件，插件则根据这些地址来调用插件接口完成所需功能、获取所需资源等。

3 插件系统的开发

本文将通过一个模拟的音频播放器（使用VC++ 6.0）来介绍插件的三种实现方法：

(1) 普通的输出函数的DLL方式；

(2) 使用C++的多态性；

(3) 使用COM类别（category）机制。

首先对此音频播放器作以下说明：① 这不是一个真的播放器，当然也不能真地播放音频文件；② 每个插件"支持"一种格式的音频文件，如"wma"、"mp3"等，通过增加插件可以使系统"支持"更多的音频格式；③ 插件接口简单，其功能实现只是弹出一个对话框，表明哪个插件的哪个功能被调用了。制作这个播放器的真正目的是演示插件技术的原理和实现方法，只要掌握了其原理和方法，就完全可以开发出有用的插件系统。

不管用什么手段实现，插件和主程序之间的交互必须有一个协议，对于方法（1）这个协议是一系列的函数，这些函数由插件DLL引出，由主程序调用；对于方法（2）协议则是一个（或多个）基类，通常是抽象类，插件需要构造一个类来继承此抽象类并实现其接口方法，再为主程序提供一个创建和销毁此实现类的对象的公共方法，这个公共方法理所当然也应成为协议的一部分；对于方法（3）则是一个（或多个）COM接口，插件是一个COM组件，它实现了这些接口，并注册到约定的组件类别（component category）下。

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）。我们的播放器将提供这四个功能，但主程序本身并不会直接实现这些功能，而是调用插件的实现，上文已经说过，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，在主程序打开某个格式的音频文件时，根据文件扩展名来决定调用哪个插件的功能。主程序可以在启动时加载所有插件，也可以在打开文件时动态加载所需插件，甚至可以在启动时加载一部分常用的插件，而在需要时加载其余插件，开发者可以有很高的自由度。

现在我们来详细讨论三种实现方法。

3.1 第一种方法

3.1.1 插件的实现

我们创建一个动态链接库Plug1.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);
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

void Play（）；void Pause（）；void Stop（）；

这些函数可以简单实现为只弹出一个消息框，表明是哪个插件的哪个函数被调用了。

为了使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。下面来看主程序的实现。

3.1.2 主程序的实现

主程序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主程序所在目录的 Plugins 子目录下）并使用 Win32 函数 LoadLibrary 加载所有插件。每加载一个插件 DLL，就调用另一个 Win32 函数 GetProcAddress 获取引出函数 GetSupportedFormat 的地址，并调用此函数返回插件所支持的格式名（即是音频文件的扩展名）。然后把（格式名，DLL 句柄）二元组保存下来。

当用户通过菜单打开文件时，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，如：

```
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 FreeLibrary 函数卸载插件。

到此为止，第一种实现插件系统的方法就介绍完了。可以看出，其实现的关键在于插件输出函数的约定以及把插件所支持的格式名映射到插件 DLL 的句柄。后面将会看到，实际上每一种实现都是基于这种原理，只不过是方式不同而已。

3.2 第二种方法

第一种实现方法完全是结构化程序设计，存在接口不易维护等缺点，从而我们自然而然想到面向对象的解决方案——把 API 封装到类里。

3.2.1 插件的实现

我们定义抽象类如下：

```
class ICppPlugin
{
public:
    ICppPlugin(){};
    virtual ~ICppPlugin() = 0 {}
    virtual void Release() = 0;
virtual void GetSupportedFormat(char* szFormat) = 0;
virtual void LoadFile(const char* szFileName) = 0;
virtual void Play() = 0;
    virtual void Stop() = 0;
virtual void Pause() = 0;
};
```

其中，Release 成员函数将在后面介绍，其他成员函数意义与第一种实现中的同名函数相同。插件程序需要实现此抽象类，每个插件都有不同的实现，而主程序仅通过接口（抽象类）来访问它们。

现在来制作一个插件 CppPlugin1.dll，它包含继承于 ICppPlugin 的类 CppPlugin1：

```
class CppPlugin1 : public ICppPlugin {...}; // 实现代码略
```

为使主程序能创建 CppPlugin1 对象，插件输出一个函数：

```
bool CreateObject(void ** pObj) {*pObj = new CppPlugin1(); return *pObj != 0;}
```

对象是在动态库中创建的，也应该在动态库中被销毁，这就是 ICppPlugin 的成员函数 Release 的作用，当主程序使用完对象后，需要调用此函数来销毁对象，它的实现如下：

```
void CppPlugin1::Release() { delete this;    // 删除自己}
```

我们还可以再制作更多的插件，这些插件只需要给出 ICppPlugin 的不同实现，即改变类 CppPlugin1 的成员函数实现即可。现在来看主程序的处理过程。

3.2.2 主程序的实现

插件的加载过程与第一种方法相似，所不同的是，加载 DLL 后，首先调用的是插件程序的 CreateObject 输出函数来创建对象：

```
typedef bool (* _CreateObject)(void ** pObj); // 定义一个函数指针类型
// 获取 CreateObject 的地址，hInst 为 DLL 的句柄
_CreateObject createObj = (_CreateObject):: GetProcAddress(hInst, "CreateObject");
    ICppPlugin* pObj = 0;   // 定义一个 ICppPlugin 的指针
createObj((void**)&pObj);   // 创建对象
```

接下来查询插件所支持的格式名，本方式中，GetSupportedFormat 已成为 ICppPlugin 的成员函数：

```
CString str;   pObj->GetSupportedFormat(str.GetBuffer(8));   str.ReleaseBuffer();
```

另外，需要保存的除（格式名，DLL 句柄）二元组映射外，还需保存（格式名，创建对象函数指针）二元组映射以备后用：

```
//str 存放的是格式名字符串的小写形式
m_formatMap [str] = hInst;   m_factoryMap [str] = createObj;
```

同样，在打开文件时选择使用哪个插件：

```
// m_pObj 存放当前使用的对象的指针，定义如下：ICppPlugin*   m_pObj; 在程序初始 // 化时要把它置为 0
    if(m_pObj)  { m_pObj->Release(); m_pObj = 0; }
    m_factoryMap [strEx]((void**) &m_pObj);   // 调用 CreateObject
m_pObj->LoadFile((LPCSTR)strFileName); // strFileName 是音频文件全路径名
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

以后就可以使用m_pObj来调用其他操作了，例如：if(m_pObj)　m_pObj->Play();

在主程序退出时需要卸载DLL，不必重复。

现在第二种实现插件系统的方式也介绍完了。这种方式基于C++的多态性，需要注意的是对象的创建和销毁方式。

3.3 第三种方法

第二种实现方法其实已经是组件化程序的雏形了，可以胜任开发小型的插件系统，若要开发大中型的系统，则需要完全组件化的设计。

COM（Component Object Model，组件对象模型）实际上就是一个实现插件的极好技术。基于COM建立的插件系统，主程序和各个插件可以用不同的编程语言写成（C++、VB、Delphi、Java等），COM能使它们无缝地结合在一起。篇幅所限，本文不详细介绍COM的原理与编程。

在这种实现方法中，插件是一个COM组件，确切地说，插件程序作为COM组件程序，包含了一个或多个COM对象，这些COM对象都实现了相同的COM接口，主程序通过这个COM接口来访问COM对象，即COM接口是主程序与插件通信的唯一手段。比如播放器插件所包含的COM对象都实现了如下COM接口（IDL定义）：

```
interface ICoPlugin : IUnknown
{
    HRESULT LoadFile([in] BSTR bstrFileName);
    HRESULT GetSupportedFormat([out, retval] BSTR *pbstrFormat);
    HRESULT Play();
    HRESULT Stop();
    HRESULT Pause();
};
```

于是，插件的开发就是COM组件的开发，这里不再详述。唯一的问题是主程序如何知道哪些是它能使用的插件（就是COM组件）。

前两种实现中，我们需要插件的具体位置和名字，所以约定插件都存放在主程序所在目录的Plugins子目录下。但是因为COM组件对COM客户是位置透明的，所以主程序需要知道的已不是插件的具体位置和名字，而是COM组件的CLSID或ProgID。可以选择把这些信息存放到指定的注册表子键下，也可以放到ini文件中等等，然而更好的方式是使用COM的组件类别（Component Category）机制。

COM允许实现者可以把相关的一组COM类组织到逻辑组（即组件类别）中。通常，一个类别（category）中的所有COM类都实现同一组接口，这些COM类共享同一个类别ID，称为CATID（category ID）。CATID也是GUID，它作为COM类的属性被保存在注册表中COM类的"Implemented Categories"子键下，在组件自注册时加入。每个类别在注册表中都有它自己唯一的子键，由它的CATID命名。

另外，系统提供一个称为组件类别管理器（component category manager）的COM类，它实现了ICatRegister和ICatInfomation接口，分别用来注册和查询类别信息。

于是，基于COM的插件系统就可以这样实现：

(1) 注册一个组件类别：CATID_Plugin；

(2) 插件实现包含实现了ICoPlugin接口的COM类，并注册为CATID_Plugin类别；

(3) 主程序在启动时使用组件类别管理器查询CATID_Plugin类别信息，得到此类别的所有COM类的CLSID，并创建相应的COM对象，获取其ICoPlugin接口，然后调用接口的GetSupportedFormat方法得到该插件所支持的格式名，保存（格式名，ICoPlugin接口指针）映射；

(4) 程序在打开音频文件时，根据扩展名决定使用哪个ICoPlugin接口指针调用LoadFile方法，并设置当前使用的接口指针m_pICoPlugin为该接口指针；

(5) 以后的操作（Play等）都使用m_pICoPlugin来调用，直到打开不同类型的文件。

(6) 程序退出时，释放掉COM对象，并释放COM库所占用的资源。

详细代码这里不再给出。

至此三种创建插件系统的方式都介绍完毕。程序使用Visual C++ 6.0开发，在Windows 2000 Server上运行通过。

3.4 小结

上文所演示的例子中，调用是单向的，即由插件暴露出接口，由主程序来调用，在实际应用中主程序也完全可以暴露出接口，由插件来调用，从而使系统更加灵活。三种方法从结构化程序设计到面向对象的方法再到基于组件的软件开发，难度依次升高，功能逐渐强大，系统也越来越灵活。根据所要创建的插件系统的不同，开发人员可以选择合适的实现方式。掌握技术原理是容易的，其实真正困难的是如何进行详细的应用分析，抽象出合适的接口，这样才能使整个插件系统拥有强大的可扩展性、灵活性、健壮性和良好的可维护性。

4 结语

插件作为特殊的组件，具备组件的所有优秀的特性。这些特性使其在开发、推广、应用方面有重要的现实意义。基于插件技术的软件开发可以使产品专业化，标准化，系列化，通过不同规格和系列的插件的组合，可以快速地完成应用系统原型，而通过对插件的局部修改来满足客户的需求和升级。

参考文献

1 Jeffery Richter著，王书洪等译，Windows高级编程指南（第三版），清华大学出版社，1999.6。

2 Erich Gamma等著，李英军等译，设计模式－可复用面向对象软件的基础，机械工业出版社，2000.9。

3 Don Box著，潘爱民译，COM本质论，中国电力出版社，2001.8。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn