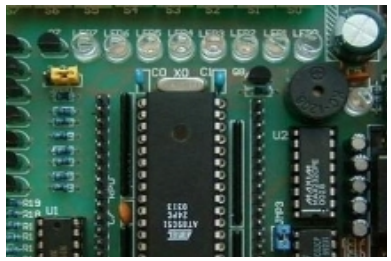

使用Qt编写模块化插件式应用程序

标签： qt dll 跨平台 编译器 windows 平台

2010-04-30 15:45 9850人阅读 评论(4) 收藏 举报

分类： 开发：C/C++（19）

动态链接库技术使软件工程师们兽血沸腾，它使得应用系统（程序）可以以二进制模块的形式灵活地组建起来。比起源码级别的模块化，二进制级别的模块划分使得各模块更加独立，各模块可以分别编译和链接，模块的升级不会引起其它模块和主程序的重新编译，这点对于大系统的构建来说更加实用。另一方面，对于商业目的明显的企业，各模块可以独立设置访问权限，开发成员只能访问自己负责的模块，其它模块是不能也不给看到的，这样减少了整个系统泄漏技术的风险。

一、动态链接库技术概况

动态链接库技术用得很多。事实上，整个Windows就是由一个个动态链接库（DLL）构建起来的，不管是系统内核，或是系统调用的API封装，还是通用工具（如控制面板、ActiveX插件等），都是一个个
件。动态链接库并不是微软独有的技术，它是软件工程发展到一定阶段的必然产物。在类Unix
进制可执行模块技术不叫动态链接库，而被称为共享对象或共享库，后缀名一般为.so（即Sha
写）。为简便，下文将统称这种动态链接的技术为DLL或共享库。

其实，DLL文件跟普通的可执行文件差别不大，都是可执行文件嘛，装载到进程空间后，都是一
（函数代码）、内存分配（变量）等。在Windows中，这些可执行文件被称作PE/COFF格式文件，在Linux则称为ELF文件。从CPU的角度看来，程序中的各个要素，不管是函数还是变量，它们都是一个个入口地址，变量是访问地址；而C++的所谓类或对象，最后也被编译器肢解成了一个个变量和函数代码（这里是形象的说法，严谨技术解说请搜索C++对象模型）。DLL的装载（指导入进程空间，然后执行）方式比可执行文件的装载稍微复杂，因为它把模块链接过程推迟到了运行时。在动态链接库的装载过程中，首要任务就是解决地址重定向问题。我们知道，DLL装载到进程空间的位置（基址）是不确定的（动态装载嘛），即使DLL内部使用的函数调用和全局变量引用，在装载时都要重新计算其地址。Windows采用基址重定向（Rebasing）技术解决这一问题，而linux采用地址无关代码（PIC，通过GOT和PLT表实现）技术。这两种技术各有优缺点。

设计 (1)

文章存档

2017年05月 (1)
2017年01月 (1)
2016年12月 (1)
2016年11月 (1)
2012年05月 (1)

展开

阅读排行

远程唤醒、WOL、Magic (16834)
使用Doxygen生成全中文 (13575)
使用Qt编写模块化插件式 (9843)
Centos Bind9 DNS 服务 (9837)
界面库选型——Qt (9362)
解决 WinSCP 内部编辑器 (7518)
pam_ldap详细配置 (6196)
LDAP 与各系统的集成 (5871)
更改MFC对话框默认的窗 (5800)
关于DNS反解的一些资料 (4663)

二、Qt中的动态链接库编程

使用C++面向对象的类编写DLL是要注意很多细节的，主要是二进制（ABI）兼容问题。COM是一个很成功的例子，只要符合COM的规范，我们就能编写出很好的DLL来，然而COM是微软私生的，要想跨平台，我们还得另找它路。

Qt的跨平台特性同样令人（至少是我）兽血沸腾。如果你认为QT仅仅是一个跨平台界面库，那
我要说的是，它不但是一个通用的跨平台的面向对象的应用程序接口库（包括GUI、数据库、网
XML、数据容器和算法等，常用的编辑资源都有封装，就是说，这些都可以跨平台，而不仅仅是界面、
一种C++语言的扩展，一种编程平台和应用程序框架。信号和槽的机制简化了对象之间的通信，比MFC的消息
映射直观多了；界面的布局管理机制使开发人员可以很轻松地编出优雅的窗体；界面语言翻译机
用；QObject容器管理可以看到Qt在内存管理方面的努力；扩展的foreach循环结构也向现代语言发展

Qt的跨平台特性很好，对于本文的主题——动态链接库的支持也很好。QT对各种平台的动态链接库编程技术都有包装，QT把这种技术统一命名为共享库（Shared Libraries）。通过使用Qt包装过的类和宏，可以编写跨平台的共享库和插件——当然，这只是源代码级别的跨平台，你不要指望用MSVC编译出来的DLL，能集成到ARM平台的Linux程序上面——这是一个很美很美的理想哦。

QT使用以下两个宏来实现符号（函数或全局变量/对象）的导出和导入（跨平台不能用def文件了）：

```cpp
Q_DECL_EXPORT    // 必须添加到符号声明中（共享库项目）
Q_DECL_IMPORT    // 必须添加到符号声明中（使用共享库的客户项目）
```

QT使用 QLibrary 类实现共享库的动态加载，即在运行时决定加载那个DLL程序，插件机制使用。

三、QT共享库和插件范例

本节通过例子，实现一个共享库和一个插件。在Windows平台上开发，使用VS2005编译，QT

本例了将编写以下三类项目：

1. Bil 项目：共享库项目，输出Bil.dll和Bil.lib，基础接口类库，定义一个公共的接口IAnimal（抽象类），供客户项目和插件项目使用；
2. Plugin 类项目：插件类项目，现编写BilDog和BilPanda两插件项目，实现IAnimal的功能，供客户项目加载和测试。两项目输出BilDog.dll和BilPanda.dll；
3. Test 项目：客户应用程序项目，输出Test.exe，界面中可以选择要加载的Animal插件，然后调用Animal的功能函数，完成测试；

1. 编写共享库——**Bil** 项目的实现

该项目定义一个抽象的 IAnimal 类作为导出接口，供客户项目和插件项目使用。项目类型为共享库，将生成Bil.lib和Bil.dll两个文件，Bil.lib供Plugin项目和Test 项目引用，而Bil.dll将给Test.exe运行时动态加载。

新建一个头文件Bil.h，输入如下代码：

```cpp
#ifndef BIL_H
#define BIL_H
#include <Qt/qglobal.h>
// 定义BIL_SHARE，使用者可以不用再处理符号的导入和导出细节
#ifdef BIL_LIB
# define BIL_SHARE Q_DECL_EXPORT
#else
# define BIL_SHARE Q_DECL_IMPORT
#endif
#endif // BIL_H
```

你现在可能不知道BIL_SHARE宏有何用处。没关系，请继续看下面的IAnimal接口定义代码：

```cpp
#ifndef IANIMAL_H
#define IANIMAL_H
#include "Bil.h"
class BIL_SHARE IAnimal
{
public:
```

```
07.     IAnimal();
08.     virtual ~IAnimal();
09. public:
10.     virtual void Eat() = 0;
11.     virtual void Run() = 0;
12.     virtual void Sleep() = 0;
13. };
14.
15. #endif // IANIMAL_H
```

现在知道BIL_SHARE宏的妙用了吧。BIL_SHARE宏会根据项目编译选项BIL_LIB有没有定义，IAnimal是导出类，还是导入类。所以，使用BIL_SHARE宏，我们只需要向IAnimal插件的开发IAnimal定义文件（IAnimal.h）即可。

当然，我们得先在Bil项目的编译选项中定义BIL_LIB宏，使得在Bil项目内，BIL_SHARE就是导出明。插件项目就不要定义BIL_LIB了，因为在Animal插件项目中，IAnimal是导入符号。

编译选项如何定义宏？如果使用Visual Studio工程文件，依次展开：项目属性->配置属性->C/C++->预处理器，在预处理器定义中添加宏BIL_LIB即可；如果是QT工程文件，请在QT工程文件Bil.pro中加入如下定义：

```
[cpp]
01. DEFINES += BIL_LIB
```

在IAnimal接口中，我们定义了三个纯虚函数Eat()、Run()和Sleep()，表示吃、跑和睡眠的动作，这是抽象的，因为不同的动物有不同的吃相和睡眠姿态，而世间的动物何止千千万——无所谓，让这些具体动物的不同表现交给IAnimal插件的编写者发挥吧——这就是接口的魅力，加上插件的思想，整个应用程序就变成开放的，可扩展的了！

继续编写IAnimal类的实现文件IAnimal.cpp：

```cpp
#include "IAnimal.h"
IAnimal::IAnimal()
{
}
IAnimal::~IAnimal()
{
}
```

虽然只实现了构造和析构函数，并且什么工作也不做，但这是必要的，我们暂时不要使用内联的构造和析构函数，否则在插件项目实现IAnimal时可能会出现链接错误。

好了，我们开始编译吧，生成整个Bil项目。最终我们得到两个输出文件：Bil.lib 和 Bil.dll。

我们向Animal插件开发者提供：

两个头文件：Bil.h 和 IAnimal.h
两个库文件：Bil.lib 和 Bil.dll

下面的插件类项目和客户项目就是依赖这些文件实现的，也许你更愿意把Bil看作是一个通用的DLL类库，就像QT或MFC一样——事实上也是如此，Bil就是这样一个动态的共享类库。

2. 编写**Animal**插件**——BilDog**和**BilPanda**项目的实现

现在，让我们来实现两个小插件。BilDog插件很简单，只是汇报下“我是Dog，我正在啃骨头”；BilPanda也是如此——这里仅仅是测试而已，实现的项目中，你可以尽情的发挥——没错，是在遵循IAnimal接下。

创建BilDog项目，把Bil项目输出的Bil.h、IAnimal.h和Bil.lib加入到工程。

创建Dog类的头文件Dog.h：

```cpp
#ifndef CLASS_DOG_H
```

```
02. #define CLASS_DOG_H
03. #include "IAnimal.h"
04. class Dog : public IAnimal
05. {
06. public:
07.     Dog(void);
08.     virtual ~Dog(void);
09. public:
10.     virtual void Eat();
11.     virtual void Run();
12.     virtual void Sleep();
13. };
14. #endif // CLASS_DOG_H
```

创建Dog类的实现文件Dog.cpp：

```
[cpp]

01. #include <QtGui/QMessageBox>
02. #include "Dog.h"
03. Dog::Dog(void)
04. {
05. }
06. Dog::~Dog(void)
07. {
08. }
09. void Dog::Eat()
10. {
11.     QMessageBox::information(NULL, "Hello", "Dog eating ...");
12. }
13. void Dog::Run()
14. {
15.     QMessageBox::information(NULL, "Hello", "Dog running ...");
```

```
16. }
17. void Dog::Sleep()
18. {
19.     QMessageBox::information(NULL, "Hello", "Dog sleeping ...");
20. }
```

调用QT的QMessageBox::information()函数弹出一个信息提示框。

还有一个非常重要的工作，我们得提供一个能够创建（释放）Animal具体对象（这里是Dog）的
这些函数导出，让主程序（Test.exe）能够解析这个接口函数，动态创建Animal对象，并访问其功能。

新建BilDog.h文件，输入下面的代码：

```
[cpp]

01. #ifndef BILDOG_H
02. #define BILDOG_H
03. #include "Dog.h"
04.
05. // extern "C" 生成的导出符号没有任何修饰，方便主程序找到它
06. extern "C"
07. {
08.     Q_DECL_EXPORT IAnimal * CreateAnimal();
09.     Q_DECL_EXPORT void ReleaseAnimal(IAnimal * animal);
10. }
11. #endif // BILDOG_H
```

这两个函数的工作很简单，直接创建和释放对象即可。
下面是BilDog.cpp的代码：

```cpp
#include "bildog.h"

IAnimal * CreateAnimal()
{
    return new Dog();
}

void ReleaseAnimal(IAnimal * animal)
{
    delete animal;
}
```

至此，一个Animal插件总算完成了。编译，生成BilDog项目，输出BilDog.dll插件文件，以供主程
态调用。

BilPanda项目和BilDog项目类似，在这里就不把代码贴出来了。以后开发Animal插件（即使是第三方）的过程都是如此。

我们不打算输出该项目的.lib文件和那些头文件，因为我们打算让主程序在运行时刻根据需要装载dll插件和调

用插件的功能，而不是让主程序项目在编译时就指定具体的插件。

3. 编写客户程序**——Test**项目的实现

Test项目是一个测试程序项目，但它的角色是主程序，是能使用Animal插件的客户程序。

同样，这个项目用到了Bil共享库，所以得先把Bil项目的几个输出文件导入到Test项目。

我们假设Test主程序是一个对话框，上面有一个编辑框和一个“加载并调用”按钮，终端用户在编辑
Animal插件的文件名（比如BilDog，后缀名可省略，Qt会根据平台判断该查找.dll还是.so），点击“加载并调用”进行共享库的加载，并调用动态创建的IAnimal对象的Eat()函数（当然你可以调用Run()函数或Sleep()，这里仅仅是一个示例）。

下面的函数将被“加载并调用”按钮的触发事件调用：

```cpp
// ...
```

```
02. #include <QString>
03. #include <QLibrary>
04. #include <IAnimal.h>
05.
06. // ...
07.
08. // strPluginName为插件的名称，可省略后缀
09. void MainDlg::LoadAndAction(QString strPluginName)
10. {
11.     // 加载插件dll
12.     QLibrary lib(strPluginName);
13.     if (lib.load())
14.     {
15.         // 定义插件中的两个导出函数的原型
16.         typedef IAnimal* (*CreateAnimalFunction)();
17.         typedef void (*ReleaseAnimalFunction)(IAnimal* animal);
18.
19.         // 解析导出函数
20.         CreateAnimalFunction createAnimal =
21.                 (CreateAnimalFunction) lib.resolve("CreateAnimal");
22.         ReleaseAnimalFunction releaseAnimal =
23.                 (ReleaseAnimalFunction) lib.resolve("ReleaseAnimal");
24.
25.         if (createAnimal && releaseAnimal)
26.         {
27.             // 创建Animal对象
28.             IAnimal * animal = createAnimal();
29.             if (animal)
30.             {
31.                 // 使用插件功能
32.                 animal->Eat();
33.                 animal->Sleep();
34.                 // 插件使用完毕，删除对象
35.                 releaseAnimal(animal);
```

```
            }
        }
        // 卸载插件
        lib.unload();
    }
}
```

生成Test项目，输出Test.exe。我们把Test.exe、Bil.dll、BilDog.dll、BilPanda.dll放在同一目录
Test.exe，赶快试下效果吧！注意BilDog.dll或BilPanda.dll依赖于基础接口库Bil.dll，如果系统找
不能加载BilDog.dll或BilPanda.dll，所以请把它们放在同一目录。

四、一些遗憾

DLL的愿望是美好的，只要接口一致，用户可以任意更换模块。但如果不注意细节，很容易陷入它的泥潭中，这就是传说中的DLL Hell（DLL地狱）！

引起DLL地狱问题的主要原因有以下几点：

1. 版本控制不好（主要是接口的版本）

DLL是共享的，如果某程序更新了一个共享的DLL，其它同样依赖于该DLL的程序就可能不能正常工作了！

2. 二制兼容问题（ABI）

即使同一平台，不同编译器（甚至同一编译器的不同版本）编出来的共享库和程序也可能不

二制兼容问题对于C++来说尤其严重。C++的标准是源代码级别的，标准中并没有对如何实
的规定，所以不同的编译器，对标准C++采用不同的实现方式。这些差异主要有：对象在内存
（C++）、构造和析构函数的实现（C++）、重载和模板的实现（C++）、虚函数表结构（C++
和虚基类的实现（C++）、函数调用约定（C）、符号修饰（C/C++）等。此外，不同的运行时
STL等标准库）也会引起ABI兼容问题。可以说，如果你在编写基于类的共享库，如果接口（指
改变，新的DLL与原程序就可能不协同工作了。

关于二进制兼容问题，大家可以参考KDE官网上的一篇文章《Policies/Binary Compatibility Issues With C++》

不过这些都不是大问题，毕竟我们不是编写像Qt一样的通用库。我们引入DLL划分应用程序的模块，目的是减小系统开发和后期升级维护的难度，同时方便项目的管理。如果用户想自己编写插件模块，就得使用我们指定的编译平台和类接口。所以我们仍能从DLL技术中得到很大的实惠。

上一篇 DLL 导出类的问题
下一篇 Qt4.6.2已编译二进制版本在VS2005中的问题

查看评论

4楼 蓝橙_2017 2017-04-18 14:02发表

测试通过

3楼 失之毫厘谬以千里zzp499542 2016-06-25 22:46发表

测试通过！

2楼 调味料来了 2015-01-27 09:28发表

Plugin 类项目：插件类项 这种项目如何创建 我用的qtcreator

1楼 YinYin121 2012-10-17 16:46发表

你好，请教一个问题，我按您的例子做出来后，编译可执行程序总报构造函数和析构函数链接错误呢，
1>Dog.obj : error LNK2019: 无法解析的外部符号 "__declspec(dllimport) public: __thiscall IAnimal::IAnima
(__imp_??0IAnimal@@QAE@XZ)，该符号在函数 "public: __thiscall Dog::Dog(void)" (??0Dog@@QAE@
1>Dog.obj : error LNK2019: 无法解析的外部符号 "__declspec(dllimport) public: virtual __thiscall IAnimal::~
(__imp_??1IAnimal@@UAE@XZ)，该符号在函数 "public: virtual __thiscall Dog::~Dog(void)" (??1Dog@@
用

请问什么原因啊？