# Разработка приложения с модулями расширения

Модули расширения (плагины) стали неотъемлемой частью современных приложений. С их помощью расширяется функциональность приложений без компиляции или быстро изменять бизнес-правила, на основе которых работает приложение. Кроме того, для разработки плагинов не нужно иметь доступа к исходному коду приложения, и они могут разрабатываться третьими лицами. В .NET разработка плагинов является достаточно простой задачей, благодаря технологии рефлексии (reflection).

Рефлексия (отражение) позволяет получать информацию о типах и сборках во время выполнения программы, т.е. предоставляет возможность чтения метаданные. Рефлексия используется для динамической загрузки сборок, получения информации о методах, свойствах, событиях и полях классов, динамического создания новые типы и вызова методов во время выполнения.

Рефлексия в .NET является основой позднего связывания.

Классы и интерфейсы для рефлексии находятся в пространстве имен System.Reflection.

## Класс Assembly

Для работы со сборками используется класс Assembly. Сборка может быть загружена с помощью статических методов класса Load и LoadFrom. Load загружает сборку по ее имени, заданным строкой, или на основе информации хранящейся в объекте AssemblyName (версия, криптографический ключ, ифнормация о культуре). В имя сборки не входит расширения файла. Например, имя сборки (MyAsm.dll будет MyAsm). LoadFrom напрямую загружает сборку из файла, путь к которому передается методу. Можно загружать сборки и вызовом метода Load для объектов домена AppDomain. Например, чтобы загрузить сборку в текущий домен можно воспользоваться таким кодом

AppDomain.CurrentDomain.Load(assemblyName);

## Класс Type

Основной класс для динамического получения информации о классах, интерфейсах, их полях, методах и перечислениях - Type. Для получения объекта Type можно воспользоваться несколькими способами:

статический метод Type.GetType, который по имени типа возвращает объект Type

методы GetInterface, GetInterfaces, FindInterfaces, GetElementType и GetTypeArray класса Type

методы GetType, GetTypes и GetExportedTypes класса Assembly

методы GetType, GetTypes и FindTypes класса Module

оператор typeof

При получении объекта класса Type для типа, появляется возможность получить информацию о нем. Например, метод GetInterfaces возвращает все реализуемые или наследуемые интерфейсы, метод GetFields возвращает все открытые поля, метод GetMethods возвращает все открытые методы, а метод GetProperties возвращает все открытые свойства. Свойства IsAbstract, IsArray, IsClass, IsEnum, IsInterface, IsValueType позволяют узнать является ли тип абстрактным, массивом, классом, перечислением, интерфейсом, структурой.

## Создание экземпляров типов

При помощи объекта класса Type можно не только определять параметры типа, но и создавать его экземпляры и вызывать их методы. Для этого также существует несколько способов:

методы CreateInstance, CreateInstanceAndUnrap, CrateInstanceFrom и CrateInstanceFromAndUnrap класса AppDomain. После вызова методов, названия которых не оканчиваются на AndUnrap, для доступа к реальным данным нужно вызывать дополнительную функцию Unrap, т.к. эти методы возвращают wrapper (объект класса ObjectHandle) для нового экземляра типа

методы CreateInstance и CreateInstanceFrom класса Activator. Это специальный класс для создания экземпляров типов и получения ссылок на удаленные объекты. Методу CreateInstance передаются объект Type или название инстанцируемого типа, массив объектов, соответствующих параметрам конструктора типа и объекты CultureInfo. Методу CreateInstanceFrom дополнительно передается имя сборки, содержащий тип. Методы, не принимающие в качестве параметра объект Type, также возвращают wrapper's ObjectHandle

метод CreateInstance класса Assembly, создающий тип по его имени

метод Invoke класса ContructorInfo

метод InvokeMember класса Type

## Использование интерфейсов

При создании плагинов обычно используются интерфейсы, определяющие методы и свойства, которые должны реализовываться плагином. Для получения интерфейсов, которые есть у типа, используются методы GetInterface,GetInterfaces и FindInterfaces класса Type. Метод GetInterface по имени интерфейса позвращает объект Type для этого интерфейса или null если такого интерфейса у типа нет. Метод GetInterfaces возвращает массив объектов Type с информацией об интерфейсах. Метод FindInterfaces возвращает массив интерфейсов, выбранных с помощью фильтра - делегата, вызываемого для каждого интерфейса.

Если класс реализует несколько интерфейсов, у которых есть методы с одинаковыми названиями, то нужно использовать метод GetInterfaceMap класса Type. Он возвращает объект InterfaceMapping для определения соотношения методов интерфейсов и методов класса, которые их реализуют.

Вызов методов

Обычно методы вызываются с помощью метода InvokeMember класса Type. Процесс вызова метода состоит из двух этапов - привязки, при котором находится нужный метод, и непосредственно вызова. Для вызова метода необходимо указать:

* имя метода (в качестве метода может быть обычный метод, конструктор, свойство или поле);
* битовую маску из значений BindingFlags для поиска метода. В маске можно указать тип доступа метода, тип метода (поле, свойство и т.д.), тип данных и пр.;
* объект Binder для связывания членов и аргументов;
* объект, у которого вызывается метод;
* массив аргументов метода;
* массив объектов ParameterModifier;
* объект CultureInfo.

## Разработка приложения с модулями расширения

Для демонстрации применения рефлексии при создании плагинов разработаем приложение, состоящее из 4 проектов:

* MainApp - основное приложение, к которому будут подключаться плагины. Приложение загружает из графических файлов изображения и выводит их на форме
* Interface - определяет интерфейсы IPlugin для плагинов и IMainApp для приложений, к которым будут подключаться плагины
* RandomPlugin и ReversePlugin - плагины для добавления шума к изображениям и отражения изображения по вертикали.

### Проект «Interface»

Проект Interface содержит только определения интерфейсов. Приложение, которое подключает плагины, должно реализовывать интерфейс IMainApp. Этот интерфейс объявляет единственное свойство Image, с помощью которого плагины получают изображение и возвращают его после преобразования.

namespace Interface

{

public interface IMainApp

{

Bitmap Image { get; set; }

}

}

Интерфейс для плагинов называется IPlugin и содержит объявления трех свойств и одного метода. Свойства используются для получения информации о плагинах - названия, номера версии и автора. Метод Transform вызывается при преобразовании изображения, ему передается интерфейс IMainApp.

namespace Interface

{

public interface IPlugin

{

string Name { get; }

string Version { get; }

string Author { get; }

void Transform(IMainApp app);

}

}

### Основное приложение

Приложение MainApp, к которому мы будем подключать плагины, это windows-forms приложение. Приложение содержит отду экранную форму, реализующую интерфейс IMainApp. На форме находится PictureBox для вывода изображения. Для реализации интерфейса IMainApp определим у формы свойство Image для доступа к изображению.

public Bitmap Image

{

get

{

return (Bitmap)pictureBox.Image;

}

set

{

pictureBox.Image = value;

}

}

В конструкторе формы вызывается метод FindPlugins, который находит плагины в папке с приложением и загружает их сборки. Для поиска и загрузки применяется рефлексия. Существует и другой подход - создать для приложения конфигурационный файл, в котором прописаны пути ко всем плагинам. При этом мы не сможем устанавливать плагины путем простого копирования сборок.

void FindPlugins()

{

// папка с плагинами

string folder = System.AppDomain.CurrentDomain.BaseDirectory;

// dll-файлы в этой папке

string[] files = Directory.GetFiles(folder, "\*.dll");

foreach (string file in files)

try

{

Assembly assembly = Assembly.LoadFile(file);

foreach (Type type in assembly.GetTypes())

{

Type iface = type.GetInterface("Interface.IPlugin");

if (iface != null)

{

Interface.IPlugin plugin = (Interface.IPlugin)Activator.CreateInstance(type);

plugins.Add(plugin.Name, plugin);

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки плагина\n" + ex.Message);

}

}

Вначале определяется папка для поиска плагинов. Т.к. у нас все плагины лежат в одной папке вместе с основным приложением, то мы используем свойство BaseDirectory для домена нашего приложения. Затем получаем все dll файлы из папки - их массив возвращает статическая функция GetFiles. Сборку для проверки на наличие плагина загружаем методом LoadFile и в цикле проходим по всем типам, определенным в сборке. Если тип содержит интерфейс IPlugin (при этом метод GetInterface возвращает не null), то создаем экземпляр этого типа (инстанцируем) методом Activator.CreateInstance. Для последующего использования мы сохраняем инстанцированный тип в словаре plugins, ключем которого является название плагина.

Dictionary<string, IPlugin> plugins = new Dictionary<string, IPlugin>();

Потенциальной проблемой для нашего кода может стать то, что из домена приложения нельзя выгрузить сборку. Если в папке с приложением окажется много сборок, которые будут загружаться в процессе поиска плагинов, то это приведет к ненужному расходу памяти. В таком случае можно создать новый домен, вызвав статическую функцию AppDomain.CreateDomain, загрузить все сборки в созданный домен и получить названия только тех сборок, которые содержат плагины, выгрузить домен функцией AppDomain(Unload) и загрузить сборки с плагинами в домен.

После того, как все плагины найдены, создаем для них в функции CreatePluginsMenu пункты меню. Названия пунктов меню берутся из словаря. Для обработки событий от меню для вызова плагинов создается обработчик OnPluginClick. В обработчике определяется названия пункта меню, который выбрал пользователь, и по нему, как по ключу в хеш-таблице, получаем интерфейс IPlugin соответствующего плагина. У плагина вызывается метод Transform, в качестве параметра this (т.к. класс формы наследуется от интерфейса IMainApp).

void CreatePluginsMenu()

{

foreach (string name in plugins.Keys)

{

ToolStripMenuItem item = new ToolStripMenuItem(name);

item.Click+=OnPluginClick;

фильтрыToolStripMenuItem.DropDownItems.Add(item);

}

}

private void OnPluginClick(object sender, EventArgs args)

{

Interface.IPlugin plugin = (Interface.IPlugin)plugins[((ToolStripMenuItem)sender).Text];

plugin.Transform(this);

}

### Создание плагина

Для создания плагинов создаем новый проект ClassLibrary, добавляем ссылку на сборку Interface и реализуем свойства и методы интерфейса IPlugin. Для примера были разработаны 2 плагина: ReverseTransform для отражения изображения по вертикали и RandomTransform для внесения случайного шума в изображение.

namespace PlugIn

{

public class ReverseTransform : Interface.IPlugin

{

public string Name

{

get

{

return "Переворот изображения";

}

}

public string Version

{

get

{

return "1.0";

}

}

public string Author

{

get

{

return "Me";

}

}

public void Transform(Interface.IMainApp app)

{

Bitmap bitmap = app.Image;

for (int i = 0; i < bitmap.Width; ++i)

for (int j = 0; j < bitmap.Height / 2; ++j)

{

Color color = bitmap.GetPixel(i, j);

bitmap.SetPixel(i, j, bitmap.GetPixel(i, bitmap.Height - j - 1));

bitmap.SetPixel(i, bitmap.Height - j - 1, color);

}

app.Image = bitmap;

}

}

public class RandomTransform : Interface.IPlugin

{

public string Name

{

get

{

return "Случайная трансформация";

}

}

public string Version

{

get

{

return "1.0";

}

}

public string Author

{

get

{

return "Me";

}

}

public void Transform(Interface.IMainApp app)

{

Bitmap bitmap = app.Image;

Random rand = new Random(DateTime.Now.Millisecond);

int pixels = (int)(0.1 \* bitmap.Width \* bitmap.Height);

for (int i = 0; i < pixels; ++i)

bitmap.SetPixel(rand.Next(bitmap.Width - 1), rand.Next(bitmap.Height), Color.FromArgb(rand.Next(255), rand.Next(255), rand.Next(255)));

app.Image = bitmap;

}

}

}

Свойства для получения номера версии плагина и авторства в нашей программе не используются, но с их помощью можно выводить информацию об установленных плагинах, как в Adobe Photoshop. После компиляции нужно поместить плагины в папку с приложением MainApp.

|  |  |
| --- | --- |
|  |  |