

# 通用托管注入 - SDK 编写指南

<http://chengchen.cnblogs.com/>  
[chengchencici@163.com](mailto:chengchencici@163.com)

一、如何编写插件：（以 C# 为例子）

1、用 VS 建立一个 C# 类库。

2、将任意一个类设置为 public 类，而且其中至少要写两个静态方法：

其一就是实现插件的功能的方法：

代码：

```
public static void InjectMain()
{
}
}
```

还有一个是插件的信息描述部分，这个信息会自动显示到选择插件时候的对话框中。

代码：

```
public static string InjectAbout()
{
    return "插件描述";
}
```

**两个方法必须都存在，而且必须为静态方法！！！！如果你使用混淆软件加密，一定要记得这两个方法名称不能被混淆（需要在混淆软件中单独设置这两个方法，将其排除混淆列表），否则系统会无法自动识别。**

可以参考 Plugin\_Source 目录下的插件源代码程序。

二、注意事项

- 1、如果您的插件需要调用其他文件，一定要注意其他文件的位置。因为程序注入进程后，当前的目录已经改变为被注入进程的目录位置，因此其他被调用文件需要放到被注入程序目录中。否则可能提示找不到文件的错误。
- 2、如果您的插件需要调用到第三方 DLL 文件，且你已经考虑到被注入目录问题（获取真实插件目录位置）。你可能会发现这个第三方 DLL 文件也被识别成插件并呈现在下拉列表中，这是因为本程序为了提高性能，并没有在开启注入时候验证所有插件的有效性。只要放在 plugin 目录下的 DLL 文件都会自动识别为插件。为了解决这个问题，您有两种途径，其一：修改 DLL 扩展名为其他扩展名，自己在插件中调用时注意文件名即可。其二：新建一个空文件，其文件名为你需要屏蔽的文件名加上.no。例如：你的插件名字叫做 plug.dll 第三方文件名叫做 myhelp.dll 那么你只要新建一个空文件 myhelp.dll.no 并放置到 plugin 目录下，那么系统会自动屏蔽 myhelp.dll，不会将其识别为插件文件。
- 3、如果您的插件需要调用到第三方 DLL 文件，但是注入后，当前目录名会改变成为被注入目录，会提示找不到第三方 DLL 文件。推荐您在代码中手动指定当前目录调用，参考以下代码：

//在初始化方法中加入Assembly错误加载触发事件：如果Assembly加载失败，就去重新寻找插件目录下的位置。

```
AppDomain.CurrentDomain.ReflectionOnlyAssemblyResolve += new
ResolveEventHandler(CurrentDomain_ReflectionOnlyAssemblyResolve);
```

```
//以下为触发事件。
System.Reflection.Assembly CurrentDomain_ReflectionOnlyAssemblyResolve(object sender,
ResolveEventArgs args)
{
    //获取当前插件的DLL目录位置，并加载第三方DLL文件。
    string path = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    return Assembly.LoadFrom(path + "\\xxxx.dll");
}
```

- 4、请将插件的目标框架设置为 DOTNET 2.0，目标平台设置为 AnyCPU，这样可以使用到最大的兼容性。如果您的插件是 C++/CLI 或者是 VCL.NET 编写，那么需要特别注意 DOTNET2.0 和 DONET4.0 平台的 DLL 并不通用。可以新建一个 DOTNET2.0 的 C#语言插件，然后判断当前平台，并调用对应平台的 C++/CLI, VCL.NET 程序集。代码如下：

//在初始化方法中加入Assembly错误加载触发事件：如果Assembly加载失败，就去重新寻找插件目录下的位置。

```
AppDomain.CurrentDomain.ReflectionOnlyAssemblyResolve += new
ResolveEventHandler(CurrentDomain_ReflectionOnlyAssemblyResolve);
```

```
//以下为触发事件。
System.Reflection.Assembly CurrentDomain_ReflectionOnlyAssemblyResolve(object sender,
ResolveEventArgs args)
{
    //获取当前插件的DLL目录位置，并加载第三方DLL文件。
    string path = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    //获取当前DOTNET Framework 版本，并调用不同的对应的版本的 C++/CLI, VCL.NET程序集。如果
    程序集是C#编写则编译为DOTNET2.0框架可以通用。
    if(Environment.Version.Major == 4)
    {
        return Assembly.LoadFrom(path + "\\4444.dll");
    }
    else
    {
        return Assembly.LoadFrom(path + "\\2222.dll");
    }
}
```