OllyDbg 插件开发入门

0.11 版

肖梓航（Claud）

2010 年 2 月

E-mail：iClaudXiao@gmail.com
Website/Blog：http://www.iclaud.net

文档编号：2010-004
最后更新：2010 年 2 月 19 日

目　　录

前言

我是一个菜鸟，这是毫无疑问的。

开发 OllyDbg 插件，大抵不是一件难事。因为网络上没有多少资料——只有 API 手册和两三篇文章而已，但依然有人开发了大量的插件。

而我却花了不少时间才初窥门径。

也正因为水平太低，我不得不努力地去理解插件是怎么工作的、去阅读每一个函数、去分析每一行示例代码、去花费 N 天写一个 hello,world。

为了防止又把它们给忘了，还得记录下来，以便需要的时候查阅。

再后来，我想，干脆写得更傻瓜一点吧，让其他人不费脑子不费时间地也能学会。于是就有了这个文档。

但我的水平实在太低，老实说，除了 hello,world 还没写出第二个插件。

所以这个文档将会随着进一步学习，不定期添加新的内容。

如果你希望看到后续版本，或者有好的建议，或者发现了错误，或者找不到文章中提到的资源，或者想骂我太菜，都可以访问下面这个页面：

http://www.iclaud.net/2010/02/ollydbg_plugin/

我会尽量保证它的长期可用性。如果它失效了，也可以给我写邮件。

Just for fun!

Claud

（bughouse、EricCRC 也是俺）

2010.02

好的开始，是成功的一半。

一、 前期准备

1、OllyDbg 插件工作原理

OllyDbg 是一款优秀的用户态调试工具。它不仅拥有强大的反汇编能力和动态分析能力，还具有良好的扩展结构，允许用户自行开发插件完成特定的工作。

在开发插件之前，需要大致了解插件在 OllyDbg 中工作的方式。

插件以单独的动态链接库（DLL）文件的形式提供给 OllyDbg 使用。在 OllyDbg 主菜单中，依次选择：选项→界面→目录，就可以看到插件路径。一般情况下，我们在 OllyDbg.exe 所在目录下建立名为 plugin 的子目录，并在上述插件路径中填入该子目录的绝对路径。（请注意，如果这里的值是形如./plugin/的相对路径，插件往往不能正常地工作。）

OllyDbg 与插件是如何交互完成工作的？可以从两个方面来看这个问题。

（1）从 OllyDbg 的角度

在 OllyDbg 的启动过程中，有一步是检查插件路径下是否存在 DLL 文件。如果存在，逐一进行如下扫描：

- 加载该 DLL 文件，找到其入口点
- 通过回调函数，获取插件名称、版本等信息
- 通过回调函数，对插件进行初始化，包括申请资源、恢复全局参数等

如果某个 DLL 文件无法顺利执行这三步，OllyDbg 的启动将失败、报错并退出。

OllyDbg 启动以后，会一直维护插件的队列，并在以下情况（但不仅限于这些情况）出现时向该队列发送消息，或者直接调用插件中定义的函数：

- 用户通过插件菜单或快捷键主动执行插件某功能
- 正在调试的程序状态发生改变，例如载入、运行、暂停、结束、重启等
- 系统自身的启动、关闭
- 系统收到无法识别的消息（比如组合键）
- 系统在配置文件中发现无法识别的数据

最后，当 OllyDbg 被关闭时，还会调用插件中的回调函数，释放插件申请到的资源，并将需要保存的参数、配置和附加信息分别予以保存。

（2）从插件的角度

插件的工作可以分为以下几类：

- 搜集和整理调试过程中的信息供用户参考
- 增加一些辅助信息让调试更加方便
- 直接参与调试

- 通过加载脚本程序，将一部分行为自动化

因此，插件既需要从 OllyDbg 中获取各种信息，又需要对 OllyDbg 进行各种操作。插件通过调用 OllyDbg Plugin API 来做到这些。

另外，插件也可以有自己的窗口逻辑和功能函数。事实上，我们可以将它看成这样一个 Windows 程序，它拥有自己的消息循环和窗口过程，但它的启动是由 OllyDbg 发起的，具体功能的实现也通过调用 OllyDbg 提供的函数来实现。通过本文后面的部分，这一认识将会愈加清晰。

2、学习建议

在学习开发 OllyDbg 插件的过程中，你也许会用到：

- 一些 Win32 窗口程序开发的知识和经验
- 对 OllyDbg 功能的基本了解
- 简单的汇编语言知识

现在假定你已经具备了上述能力，下文的叙述将以此为基础。

如果你已经有了其他软件的插件开发经验，并有一定的英文阅读能力，请扔掉这份文档，直接阅读 OllyDbg Plugin API 手册和实例源代码。

反之，建议先阅读本文档，并实际动手操作；然后下载更多的插件源代码，边阅读边查阅 API 手册；最后，动手写自己的插件。

3、开发资源与环境

为了开发 OllyDbg 插件，首先需要获取插件开发包。下载地址是：

http://www.ollydbg.de/plug110.zip

这一开发包是针对 OllyDbg 1.10 版的。虽然目前 OllyDbg 最新版是 2.0，但作者已经表明，1.10 版是支持自主开发插件的最后一个版本。

在这个开发包中，最重要的文件有三个：

- Plugins.hlp　开发文档，详细定义了所有提供的 API
- Plugin.h　API 定义的头文件
- Ollydbg.lib　导入库文件

此外，作者还提供了两个插件的代码作为示范，一个是命令行插件，一个是书签插件。我们将在第三部分仔细分析它们。

接下来考虑开发环境。

OllyDbg 作者 Oleh Yuschuk 使用的是 Borland C++ 5.5。也有人使用 Visual C++、Delpi、MASM 甚至 Visual Basic 进行开发。

这里我们推荐使用 Visual C++ 6.0 进行开发。理由如下：

- Win32 API 原生地支持 C/C++，OllyDbg Plugin API 也是以 C++形式提供
- 网络上几乎所有已有的插件，源代码都是基于 C/C++的，便于进一步学习
- 相比于 Borland C++，Visual C++更容易获取，在使用过程中遇到问题，也更容易通过搜索和询问获得解决

需要注意的是，在 plug110.zip 包中提供的 Ollydbg.lib 是无法直接用于 Visual C++ 6.0 的，原因是每个引出函数前面都多了一个下划线[2]。可以在这里下载到专为 Visual C++ 6.0 准备的开发包：

http://bbs.pediy.com/showthread.php?t=31344

或者你不想注册的话，也可以到本文的更新页面去看看：

http://www.iclaud.net/2010/02/ollydbg_plugin/

这个开发包并没有对 Ollydbg.lib 进行修改，但在 Plugin.h 中以

```
#define ODBG_Plugindata     ODBG_Plugindata
……     ……
```

的方式使之能与 Ollydbg.lib 配合使用。

4、必要的设置

（1）Visual C++中的设置

- Project→Add to project→Files，添加 Plugin.h
- 左侧窗口 File View→Resource Files，右键 Add Files into Folder，添加 Ollydbg.lib
- Project→Settings→C/C++，在 Project Options 最后添加“/J”，使 char 默认为 unsigned 类型，这是 OllyDbg 中的约定

（2）C++程序中的设置

- 不要忘了在 cpp 文件中`#include "Plugin.h"`
- 在 cpp 文件中添加 DLL 入口点函数如下：

```
HINSTANCE g_hModule;
BOOL APIENTRY DllMain(
    HINSTANCE hModule,
    DWORD fdwReason,
    LPVOID lpReserved)
{
        if(DLL_PROCESS_ATTACH == fdwReason)
        {
            g_hModule = hModule;
        }
    return TRUE;
}
```

也有的代码中使用了这个入口点函数：

```
BOOL WINAPI DllEntryPoint(
    HINSTANCE hinstDLL,
    DWORD fdwReason,
    LPVOID lpvReserved);
```

它们的功能是一样的。请确保它的存在，并将 DLL 模块句柄保存到一个全局变量中，在后面将会用到。缺少此函数虽然能编译通过，但会导致 OllyDbg 加载插件失败。

二、常用函数

1、回调函数

回调函数（Callback Function）之所以得名，是因为通常情况下是插件调用 OllyDbg 中实现并提供的函数，而回调函数则反了过来，是由插件来实现，OllyDbg 调用它并执行。显然，回调函数是 OllyDbg 管理插件的方式和途径。

回调函数名全部以“ODBG_Plugin”开头。此外，由于 OllyDbg 使用的参数传递和堆栈平衡方式是 cdecl，为了正常工作，所有回调函数在实现的时候，都需要申明这一方式。

OllyDbg 中有 14 个回调函数，其中有 2 个是必须在插件中实现的，另外 12 个可以根据插件的具体需要有选择地实现。

（1）必须的回调函数

ODBG_Plugindate()用于指定插件的名称，函数原型如下：

```
int ODBG_Plugindata(
    char *shortname);
```

其中唯一的参数指向一个长度不超过 31 字节的已赋值字符串。

在实际使用时，往往也通过它返回 PLUGIN_VERSION，这是一个定义在 Plugin.h 中的值，当使用 v1.10 开发包时，它等于 110。在后面，我们会将这个值与 OllyDbg 的版本号比较，以判断它们是否兼容。

因此，具体实现一般如下：

```
char g_szPluginName[] = "Our OllyDbg Plugin's Name";
extc int _export cdecl ODBG_Plugindata(
    char shortname[32])
{
    strcpy(shortname, g_szPluginName);
    return PLUGIN_VERSION;
}
```

另一个必须实现的回调函数是 ODBG_Plugininit()，它用于初始化插件。一般会在其中包含初始化工作并分配资源。函数原型如下：

```
int ODBG_Plugininit(
    int ollydbgversion,
    HWND hw,
    ulong *features);
```

其中：

ollydbgversion 就是 OllyDbg 的版本号，用于与前面返回的 PLUGIN_VERSION 进行比较；

hw 是 OllyDbg 主窗口的句柄，一般将其保存到一个全局变量中；

features 为将来的扩展而保留，不使用它。

另外，这个函数在成功时必须返回 0，而在失败时，须释放资源并返回-1。

因此，一般将其实现如下：

```
01 HWND g_hWndMain = NULL;
02 extc int _export cdecl ODBG_Plugininit(
03     int ollydbgversion,
04     HWND hw,
05     ulong * features)
06 {
07     if(ollydbgversion < PLUGIN_VERSION)
08         return -1;
09
10     g_hWndMain = hw;
11
12     /* Do some initialize works here.*/
13     /* When it fail, release resources and return -1 */
14
15     Addtolist(0,0,"Our plugin's name. v1.00");
16     Addtolist(0,-1,"  Copyright (C) 2010 Claud")
17
18     if(
19         Plugingetvalue(VAL_RESTOREWINDOWPOS)!=0
20         &&
21         Pluginreadintfromini(hinst,  // new line
               "Restore Our Plugin Window",0)!=0
22     )
23         CreateOurPluginWindow();
24
25     return 0;
26 }
```

在 12—13 行之间，我们可以加入需要的初始化代码。比如说，注册窗口类：

```
Registerpluginclass(
    ourwinclass,
    NULL,
    g_hModule,
    Ourwinproc)
```

Registerpluginclass()是由 OllyDbg 封装的窗口类注册函数，其中 ourwinclass 返回类名，Ourwinproc 是我们自己实现的类处理函数。这个函数失败时返回一个负数值，此时应该遵循 ODBG_Plugininit()的要求，以-1 返回。

初始化工作完成后，在 15、16 行我们使用 Addtolist()函数将插件加载信息输出到记录窗口（log window），在 OllyDbg 中，按下 Alt+L 可以查看该窗口。这个函数有三个参数，其中第二个的值取 0、1、-1 分别表示正常显示、高亮、低亮。OllyDbg 作者推荐使用代码中所示两行记录格式，第一行说明插件名称和版本，第二行说明版权信息。

下面说明 18—23 行，它们不是必须存在的。

在 23 行，if 语句中，调用了一个函数 CreateOurPluginWindow()，这是一个我们自己定义的函数，作用是创建插件的主窗口。

if 语句中，条件由两个函数的返回值构成。这两个函数分别查阅系统中恢复窗口位置的设置是否打开、以及 ollydbg.ini 配置文件中是否有上一次插件关闭时加入的恢复插件窗口的标记，它们的具体用法将在下一节“插件函数”中介绍。

因此，18—23 行的作用是重启 OllyDbg 时根据设定恢复插件窗口。

（2）可选的回调函数

在 12 个可选的回调函数中，ODBG_Pluginmenu()和 ODBG_Pluginaction()往往也是需要定义的。

ODBG_Pluginmenu()用于创建插件的子菜单。其函数原型为：

```
int ODBG_Pluginmenu(
    int origin,
    char data[4096],
    void *item);
```

OllyDbg 支持在主菜单的“插件”菜单下或任何一个数据窗口右键中建立插件的子菜单。origin 参数指明了创建菜单的命令来源于何处——插件菜单、反汇编窗口、断点窗口等等。它的取值在 Plugin.h 中预定义好了，以 PM_开头，例如插件菜单是 PM_MAIN，其他值可以在 API 手册中查询。

data 指向一个最长为 4096 字节的字符串，它定义了菜单的项目和显示格式。例如：

```
strcpy(data,”0 &Aaa,1 &Bbb|2 &Ccc”);
```

将创建三个菜单项 Aaa、Bbb、Ccc，其中第二个和第三个之间有一条分栏线。而

```
strcpy(data,”#A{0 &Aaa, 1& Bbb}”);
```

则创建一个弹出式菜单 A，有两个子项目。

请注意这里的 0、1、2 等数字，它们并不会在菜单上显示，而是作为项目的索引，将在 ODBG_Pluginaction()中用到。它们的范围应在 0 — 63 之间。

参数 item 或者指向显示于窗口的有序数据的选定元素，或者指向数据窗口的描述符，通过它，我们能更加细致地决定如何显示菜单。

最后，ODBG_Pluginmenu()成功时返回 1，失败时返回 0。

（OllyDbg Plugin API 在返回状态上的定义并不一致，因此需多加注意。）

我们来看看两种典型的实现。如果只在主菜单中显示，可以如下：

```
extc int _export cdecl ODBG_Pluginmenu(
    int origin,
    char data[4096],
    void *item)
{
    if(origin != PM_MAIN)
        return 0;
    strcpy(data,"0 &main function|1 &Help,2 &About");
    return 1;
}
```

而如果要在多个窗口中显示，则可以如下：

```
extc int _export cdecl ODBG_Pluginmenu(
    int origin,
    char data[4096],
    void *item)
{
    t_dump *pd;
    switch (origin)
    {
    case PM_MAIN:
        strcpy(data, "/* blah blah blah */");
        return 1;
    case PM_DISASM:
        pd = (t_dump *)item;
        if (NULL == pd)
            return 0;
        if (/* pd->blah blah pd->blah */)
       {
          // sprintf blah blah blah
        }
        return 1;
    case blah blah:
        blah……
        return 1;
    }
    return 0;
}
```

ODBG_Pluginaction()用于实现各菜单项的功能。其函数原型是：

```
void ODBG_Pluginaction(
    int origin,
    int action,
    void *item);
```

其中 origin、item 与 ODBG_Pluginmenu()中相同。action 就是上面定义项目时的那个索引数值。

针对上述前一个例子，我们实现它：

```
extc void _export cdecl ODBG_Pluginaction(
    int origin,
    int action,
    void *item)
{
    if (origin!=PM_MAIN)
        return;
    switch (action)
```

```
    {
    case 0:  // main function
        CreateOurPluginWindow();
        break;
    case 1:  // help
        WinHelp(hwmain,"ourplugin.hlp",HELP_CONTENTS,0);
        break;
    case 2:  //about
        MessageBox(
            hwmain,
            "Some Plugin v0.10\n Written by Claud",
            "About",
            MB_OK|MB_ICONINFORMATION
        );
        break;
    default:
        break;
  };
}
```

接下来我们简单地看一看其他 10 个回调函数。

```
void ODBG_Pluginmainloop(
    DEBUG_EVENT *debugevent);
```

在 OllyDbg 主窗口的每次窗口循环时，都会调用一下这个函数。因此可以把一些周期性的工作放在这里。但不推荐这样做，因为这个调用并不是公平的，而且也会影响运行速度。

其中参数 debugevent 当调试事件发生时，会指向事件的类型（参考 MSDN），否则为 NULL。

```
void ODBG_Pluginsaveudd(
    t_module *pmod,
    int ismainmodule);
```

这个函数用于将模块或应用程序相关的信息保存到相应的.udd 文件中去。请将这两种情况区分开，分别存储。

其中 pmod 指向模块描述符，ismainmodule 表明调用发自于主模块还是被调试的程序。

保存信息需要通过这个函数实现：

```
int Pluginsaverecord(
    ulong tag,
    ulong size,
    void *data);
```

其中 tag 是一个唯一的插件标识符，为了不混淆，应当向 Oleh Yuschuk 本人申请这一标志符。size 是要保存数据的长度，但不得超过 USERLEN（=4096）；data 指向要保存的数据。

既然我们可以向.udd 文件写入数据，当然也能读取。下面这个函数就提供了这一功能。

```
int ODBG_Pluginuddrecord(
    t_module *pmod,
    int ismainmodule,
    ulong tag,
    ulong size,
    void *data);
```

事实上，OllyDbg 在读取.udd 文件时，会调用这个函数，并将无法识别的记录传递给插件，如果该记录属于本插件，须通过这个函数处理并返回 1，否则返回 0，然后系统会将其传递给下一个插件。

```
int ODBG_Pluginshortcut(
    int origin,
    int ctrl, int alt, int shift, int key,
    void *item);
```

这个函数用于识别组合键。当 OllyDbg 收到组合键并且不能识别时，就将其传递给插件。事实上一次组合键将会被传递两次，一次是来自于全局的主窗口，一次来自于当前激活的子窗口，origin 就标识了这一来源。如果插件接受这个组合键，返回 1，否则返回 0。

该函数常被用于实现快捷键或加速键。我们来看一个简单的应用：

```
extc int _export cdecl ODBG_Pluginshortcut(
    int origin,
    int ctrl, int alt, int shift, int key,
    void *item)
{
    if (ctrl==0 && alt==1 && shift==0 && key==VK_F1)
    {
        CreateOurPluginWindow();
        return 1;
    }
    return 0;
}
```

当按下 Alt+F1 时，打开插件窗口。

```
void ODBG_Pluginreset(void);
```

当用户打开一个新的程序调试，或者重新调试当前的程序，OllyDbg 会调用这个函数。可以在此重置插件的一些临时数据。

```
void ODBG_Pluginclose(void);
```

当用户终止 OllyDbg 时，OllyDbg 将调用此函数，注意此时此时 WM_CLOSE 消息还未发送。当正确执行后，应当返回 0。返回任何非 0 值将停止 OllyDbg 的关闭过程，此时应详细地向用户报告发生了什么问题。

通常我们在这个函数中调用下面这个函数

```
int Pluginwriteinttoini(
    HINSTANCE dllinst,
```

```
    char *key,
    int value);
```

来把关于插件的全局配置信息保存到 ollydbg.ini 文件中去，后者的具体用法将在下一部分说明。

```
void ODBG_Plugindestroy(void);
```

OllyDbg 在最后退出前调用这个函数，此时 WM_DESTROY 消息已经收到。这个函数应当回收一切分配出去的资源，包括窗口类、文件、内存等。因此经常在其中用到

```
void Unregisterpluginclass(char *classname);
```

```
int ODBG_Paused(
    int reason,
    t_reg *reg);

int ODBG_Pausedex(
    int reason,
    int extdata,
    t_reg *reg,
    DEBUG_EVENT *debugevent);
```

OllyDbg 在调试暂停下来时调用它们。如果都定义了，后者的优先级高于前者。

```
int ODBG_Plugincmd(
    int reason,
    t_reg *reg,
    char *cmd);
```

OllyDbg 遇到条件记录断点时，将调用这个函数，将命令传递给每个插件。插件必须回应是否是传给自己的，是则返回 1，否则返回 0。

其中，reason 是引起中断的原因，reg 指向引起中断的寄存器或线程，cmd 是要传递的命令。

2、插件函数

另一类 OllyDbg Plugin API 函数是插件函数（Plugin Function），用于主动与 OllyDbg 系统进行交互。按功能将其分为四类。

（1）注册窗口类

使用 Registerpluginclass()函数注册窗口类，一般在 ODBG_Plugininit()中调用。其原型为：

```
int Registerpluginclass(
    char *classname,
    char *iconname,
    HINSTANCE dllinst,
    WNDPROC classproc);
```

其中：

classname 指向至少 32 字节的字符串空间，在函数执行后将返回类名；

iconname 是图标资源名，为 NULL 时使用默认的插件图标；

dllinst 是插件实例句柄，在入口点 DllMain()中获取；

classproc 是这个窗口类的窗口过程函数，需要我们自己实现。

注册成功时返回 0，失败时返回-1。

使用 ODBG_Plugindestroy()注销窗口类，一般在 ODBG_Plugindestroy()中调用。原型为：

```
void Unregisterpluginclass(char *classname);
```

（2）.ini 文件交互

OllyDbg 使用 ollydbg.ini 文件来保存系统相关的设置，它提供了读和写该配置文件的接口。读、写都分为针对整数（int）和针对字符串（string）两种类型，因此有四个函数：

```
int Pluginwriteinttoini(
    HINSTANCE dllinst,
    char *key,
    int value);
int Pluginwritestringtoini(
    HINSTANCE dllinst,
    char *key,
    char *s);
int Pluginreadintfromini(
    HINSTANCE dllinst,
    char *key,
    int def);
int Pluginreadstringfromini(
    HINSTANCE dllinst,
    char *key,
    char *s,
    char *def);
```

其中，dllinst 是插件实例句柄，key 是要写入或查询的键名，value、s 是要写入的键值，def 是默认的值。

对写函数，成功时返回 1，失败时返回 0；对读函数，返回要读取的值。

（3）查询系统信息

OllyDbg 提供了两个函数用于获取 OllyDbg 自身的信息。其中

```
int Plugingetvalue(int type);
```

是较为常用的一个，用于查阅 OllyDbg 的设定和变量信息。唯一的参数 type 是要查阅的信息，有众多的取值，具体含义在手册[1]中有说明。

另一个是

```
t_status Getstatus(void)
```

它返回正在被调试的进程的状态。返回值的含义如下：

```
STAT_NONE          当前没有进程被调试
STAT_STOPPED 进程被挂起
STAT_EVENT         正在处理调试时间，进程暂停
STAT_RUNNING 进程在运行
STAT_FINISHED      进程被终止
STAT_CLOSING TerminateProcess()被调用，等待确认
```

（4）.udd 文件交互

OllyDbg 使用.udd 文件保存被调试文件及模块相关的信息，因此也提供了一个函数将记录保存到.udd 文件中，其原型为（前面已经提到过）：

```
int Pluginsaverecord(
    ulong tag,
    ulong size,
    void *data);
```

这个函数仅当位于回调函数 ODBG_Pluginsaveudd()中时有效。

3、其他函数

除了回调函数和插件函数，OllyDbg 还提供了一百多个其他函数供插件开发使用，它们功能各异，是实现插件具体工作的重要组成。

但出于以下原因，这里我们不再对它们一一进行介绍：

- 它们数量繁多，而且其中一部分并不常用到
- 它们并不需要你理解或者记下来，而只需要要用的时候去查阅就可以了
- 学习它们最好的方式不是讲解，而是去读他人的代码或者自己动手去用
- 我对它们的理解还不深，免得误导大家

但也有可能我会在本文档的手续更新中加入对常用的函数的介绍。

三、实例分析

和任何一种语言一样，熟练地开发 OllyDbg 插件建立在大量的练习和阅读他人源代码的基础之上。在本章，我们先自己写一个 Hello,world，然后分析两份实例代码。

1、Hello,world

通过前面的学习，要写一个简单的 hello,world 已经不困难了。下面这个三页半的程序就是。

它的功能非常简单，选择第一个菜单时，出来一个 Windows 窗口，其中显示 Hello,world!；选择第二个菜单时，弹出来一个对话框，说明版权信息。

强烈建议大家也自己动手来实现一个。（当然，你也可以对它不屑一顾:-P）

```
001 #include <windows.h>
002 #include "Plugin.h"
003
004 static char g_szPluginName[] = "Hello,world!";
005 static HWND g_hWndMain = NULL;
006 static HINSTANCE g_hModule = NULL;
007 static char g_szHelloClass[32];
008
009 static HWND CreateHelloWindow(void);
010 LRESULT CALLBACK HelloWndProc(
011     HWND hWnd,
012     UINT msg,
013     WPARAM wParam,
014     LPARAM lParam);
015
016 BOOL APIENTRY DllMain(
017     HINSTANCE hModule,
018     DWORD reason,
019     LPVOID lpReserved)
020 {
021     if (DLL_PROCESS_ATTACH == reason)
022     {
023         g_hModule = hModule;
024     }
025     return TRUE;
026 }
```

```
027
028 extc int _export cdecl ODBG_Plugindata(
029     char shortname[32])
030 {
031     strcpy(shortname, g_szPluginName);
032     return PLUGIN_VERSION;
033 }
034
035 extc int _export cdecl ODBG_Plugininit(
036     int ollydbgversion,
037     HWND hw,
038     ulong * features)
039 {
040     int nRetCode;
041
042     if(ollydbgversion < PLUGIN_VERSION)
043         return -1;
044
045     g_hWndMain = hw;
046
047     nRetCode = Registerpluginclass(
048         g_szHelloClass,
049         NULL,
050         g_hModule,
051         HelloWndProc);
052     if(nRetCode < 0)
053         return -1;
054
055     Addtolist(0,0,"Hello,World! v1.0");
056     Addtolist(0,-1,"  Copyright (C) 2010 Claud");
057     return 0;
058 }
059
060 extc int _export cdecl ODBG_Pluginmenu(
061     int origin,
062     char data[4096],
063     void *item)
064 {
065     if(PM_MAIN == origin)
066     {
067         strcpy(data,"0 Hello | 1 About");
068         return 1;
069     }
070     return 0;
```

```
071 }
072
073 extc void _export cdecl ODBG_Pluginaction(
074     int origin,
075     int action,
076     void *item)
077 {
078     if(PM_MAIN == origin)
079         switch(action)
080         {
081         case 0:
082             CreateHelloWindow();
083             break;
084         case 1:
085             MessageBox(
086                 g_hWndMain,
087                 "Writen by Claud",
088                 g_szPluginName,
089                 MB_OK);
090             break;
091         }
092 }
093
094 extc void _export cdecl ODBG_Plugindestroy(void)
095 {
096     Unregisterpluginclass(g_szHelloClass);
097 }
098
099 LRESULT CALLBACK HelloWndProc(
100    HWND hWnd,
101    UINT msg,
102    WPARAM wParam,
103    LPARAM lParam)
104 {
105     RECT rc;
106     PAINTSTRUCT ps;
107     HBRUSH hbr;
108     HDC dc;
109     switch(msg)
110     {
111    case WM_PAINT:
112         dc=BeginPaint(hWnd,&ps);
113         GetClientRect(hWnd,&rc);
114         hbr=CreateSolidBrush(GetSysColor(COLOR_BTNFACE));
```

```
115         FillRect(dc,&rc,hbr);
116         TextOut(dc,100,60, // new line
                "Hello,world!",strlen("Hello,world!"));
117         DeleteObject(hbr);
118         EndPaint(hWnd,&ps);
119         break;
120     default:
121         return DefWindowProc(hWnd,msg,wParam,lParam);
122     }
123     return 0;
124 }
125
126 static HWND CreateHelloWindow(void)
127 {
128     HWND hw;
129     hw = CreateWindow(
130         g_szHelloClass,
131         "Message",
132         WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU,
133         400,400,300,200,
134         NULL,
135         NULL,
136         (HINSTANCE)Plugingetvalue(VAL_HINST),
137         NULL);
138     ShowWindow(hw,SW_SHOWNORMAL);
139     UpdateWindow(hw);
140     return hw;
141 }
```

唯一需要解释的是 112 — 118 行，除了用于显示文本的 TextOut 外，其余六行是用默认按钮颜色填充背景。这一操作是必要的，因为 Registerpluginclass()将 hbrBackground 设为 NULL 了。

2、Command

在 OllyDbg Plugin 开发包里，作者附带了两个插件的示例代码，分别为 Command Line 和 Bookmarks（汉化版翻译为“命令行”和“书签”）。这一节和下一节我们就来分析它们。请大家自己打开 OllyDbg 先熟悉一下它们的功能，这将会大大减少阅读时理解的难度。

命令行插件源代码由 Command.c 和 Cmdexec.c 两个文件构成，其中前者负责与 OllyDbg 的交互，后者负责命令的解释和执行。这里我们只分析前者。

不要被它的篇幅（九页）所吓倒。你会发现其中有六成以上的代码非常熟悉，事实上，前文提到的很多示例都来源于此，正好可以复习一下；其他部分则使用了一些没有介绍过的 API 函数，拿起你手头的 API 手册吧。

作为 OllyDbg 作者提供的两个示范插件之一，这份源代码还经常作为模板使用。在阅读时，请保持大局观。

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include "plugin.h"

#define VERSIONHI   1       // 最高插件版本 这两行的定义非常奇怪
#define VERSIONLO   10      // 最低插件版本 而且似乎并没有用到
#define DX  370     // 命令行窗口的宽
#define DY  130     // 命令行窗口的高

// 在.udd 文件中记录数据所需的插件唯一标识符
#define TAG_CMDLINE     0x6C6D430AL
#define ID_HWBOX        1001    // hwbox 的标识符
#define ID_HWERR        1002    // hwerr 的标识符
#define NHIST       32      // 历史中最多记录 32 条命令

static HINSTANCE hinst;     // DLL 句柄
static HWND     hwmain;     // OllyDbg 主窗口句柄
static HWND     hwcmd;      // 命令行窗口
static HWND     hwbox;      // 编辑和显示历史的下拉框
static HWND     hwedit;     // 下拉框内部的编辑控制
static WNDPROC  oldhweditproc;  // hwedit 原来的窗口过程
static HWND     hwerr;      // 错误信息窗口

static char     cmdlinewinclass[32];    // 命令行窗口类的名字
static int      posx;       // 窗口的 X 坐标
static int      posy;       // 窗口的 Y 坐标
static char     hist[NHIST][TEXTLEN];   //被保存的命令记录
static int      nhist;      // 历史中现有命令记录数量
static int      poponstop;  // 暂停时将窗口前置

// Execute()在文件 cmdexec.c 中实现，功能为解释并执行命令
int Execute(char *text,char *answer);

// 将一行记录插入到历史的第一位，并刷新下拉框
static void Addline(char *text)
{
    int i;
    if (text == NULL)   // 清除历史
    {
        nhist = 0;
        SetWindowText(hwerr, "");
    }
    else if (text[0]! = '\0')
```

```
    {
        for(i = nhist-1; i >= 0; i--)       // 移除历史中重复的字符串
        {
            if ( strcmp(hist[i], text) == 0 )
            {
                nhist--;
                if (i < nhist)
                    memmove(hist[i], hist[i+1], (nhist-i)*TEXTLEN);
            }
        } //for
        if (nhist >= NHIST)     // 插入新的字符串
            nhist = NHIST-1;
        memmove(hist[1], hist[0], nhist*TEXTLEN);
        strcpy(hist[0], text);
        nhist++;
    } // end if (text[0]! = '\0')
    if (hwcmd != NULL)  // 将历史记录拷贝到下拉框
    {
        SendMessage(hwbox, CB_RESETCONTENT, 0, 0);
        for (i = 0; i < nhist; i++)
            SendMessage(hwbox, CB_ADDSTRING, 0, (LPARAM)hist[i]);
        if (text != NULL && nhist > 0)
        {
            SetWindowText(hwbox, text);
            SendMessage(hwbox, CB_SETEDITSEL, 0, TEXTLEN);
        }
    }
}

// 下拉框中编辑子窗口的窗口过程
LRESULT CALLBACK Editsubclass(
    HWND hw,
    UINT msg,
    WPARAM wp,
    LPARAM lp)
{
    char s[TEXTLEN], answer[TEXTLEN];
    if (msg == WM_KEYDOWN)
    {
        switch (wp)
        {
            case VK_RETURN:
                GetWindowText(hwbox, s, TEXTLEN);
                if (Execute(s, answer) == 0)    //命令被执行
```

```
                    Addline(s);          // 当命令有效时，添加一行记录
                SetWindowText(hwerr, answer);
                SetForegroundWindow(hwcmd); // 激活命令行窗口
                SetFocus(hwbox);
                return 0;
            case VK_ESCAPE:
                SetWindowText(hwbox, "");   // 清除编辑框文本和信息
                SetWindowText(hwerr, "");
                return 0;
            default: break;
        } // switch
    }// end if (msg == WM_KEYDOWN)
    return CallWindowProc(oldhweditproc, hw, msg, wp, lp);
    //将窗口过程转给 oldhweditproc 处理
}

//命令行窗口的窗口过程
LRESULT CALLBACK Cmdlinewinproc(
    HWND hw,
    UINT msg,
    WPARAM wp,
    LPARAM lp)
{
    RECT rc;
    PAINTSTRUCT ps;
    HBRUSH hbr;
    HDC dc;
    switch (msg)
    {
        case WM_DESTROY:
            hwcmd = NULL;
            break;
        case WM_SETFOCUS:
            SetFocus(hwbox);
            break;
        case WM_CLOSE:      // 获取命令行窗口的当前位置并保存到.ini 文件中
            GetWindowRect(hw,&rc);
            posx = rc.left;
            posy = rc.top;
            Pluginwriteinttoini(hinst,"Command line window X",rc.left);
            Pluginwriteinttoini(hinst,"Command line window Y",rc.top);
            return DefWindowProc(hw,msg,wp,lp);
        case WM_PAINT:      // 将命令行窗口的背景用默认按钮颜色填充
            dc = BeginPaint(hw, &ps);
```

```
            GetClientRect(hw, &rc);
            hbr = CreateSolidBrush( GetSysColor(COLOR_BTNFACE) );
            FillRect(dc, &rc, hbr);
            DeleteObject(hbr);
            EndPaint(hw, &ps);
            break;
        default: return DefWindowProc(hw, msg, wp, lp);
    }
    return 0;
}

// 创建命令行窗口。如果窗口已经存在，将它提到前台。
static HWND Createcmdlinewindow(void)
{
    HFONT hf;
    RECT rc;
    POINT pt;
    if (hwcmd == NULL)  // 窗口还未出现，创建之
    {
        hwcmd = CreateWindow(
            cmdlinewinclass,
            "Command line",
            WS_POPUPWINDOW | WS_CAPTION | WS_VISIBLE | DS_3DLOOK,
            posx,posy,DX,DY,
            hwmain,
            NULL,
            (HINSTANCE)Plugingetvalue(VAL_HINST),
            NULL);
        if (hwcmd == NULL)
            return NULL;
        GetClientRect(hwcmd, &rc);
        // 创建包含历史记录的编辑子窗口
        hwbox = CreateWindow(
                    "COMBOBOX",
                    "",
                    WS_CHILD | WS_TABSTOP | WS_BORDER | WS_VISIBLE |
                        WS_VSCROLL | CBS_SIMPLE | CBS_HASSTRINGS |
                        CBS_NOINTEGRALHEIGHT | CBS_AUTOHSCROLL |
                        CBS_DISABLENOSCROLL,
                    5, 5, rc.right-10, rc.bottom-32,
                    hwcmd,
                    (HMENU)ID_HWBOX,
                    (HINSTANCE)Plugingetvalue(VAL_HINST),
                    NULL);
```

```
        // 将字体设为 OllyDbg 默认字体，
        hf = (
             (HFONT *)Plugingetvalue(VAL_FONTS)
            )[Plugingetvalue(VAL_DEFFONT)];
        SendMessage(hwbox, WM_SETFONT, (WPARAM)hf, 1);
        // 限制输入命令的长度为 OllyDbg 默认宽度
        SendMessage(hwbox, CB_LIMITTEXT, TEXTLEN-1, 1);
        // 为了拦截返回的键，将编辑子窗口作为下拉框里面的子类
        pt.x = pt.y = 1;
        hwedit = ChildWindowFromPoint(hwbox, pt);
        oldhweditproc =
            (WNDPROC)SetWindowLong(
                hwedit,
                GWL_WNDPROC,
                (long)Editsubclass
            );
        // 创建文本子窗口显示错误信息
        hwerr = CreateWindow(
                    "STATIC",
                    "",
                    WS_CHILD | WS_VISIBLE | SS_LEFT | SS_SUNKEN,
                    5,rc.bottom-22,rc.right-10,17,
                    hwcmd,
                    (HMENU)ID_HWERR,
                    (HINSTANCE)Plugingetvalue(VAL_HINST),
                    NULL);
        hf = (
             (HFONT *)Plugingetvalue(VAL_FONTS)
            )[SYSFONT];
        SendMessage(hwerr, WM_SETFONT, (WPARAM)hf, 1);
        Addline("");        // 刷新下拉框
    } //函数起始处那个 if
    // 把窗口拉到前台来
    SetForegroundWindow(hwcmd);
    SetFocus(hwbox);
    return hwcmd;
}

BOOL WINAPI DllEntryPoint(
    HINSTANCE hi,
    DWORD reason,
    LPVOID reserved)
{
    if (reason == DLL_PROCESS_ATTACH)
```

```
        hinst=hi;
        return 1;
}

extc int _export cdecl ODBG_Plugindata(
    char shortname[32])
{
        strcpy(shortname, "Command line");
        return PLUGIN_VERSION;
}

extc int _export cdecl ODBG_Plugininit(
    int ollydbgversion,
    HWND hw,
    ulong *features)
{
    int maxx,maxy;
    if (ollydbgversion < PLUGIN_VERSION)
        return -1;
    hwmain = hw;
    if(Registerpluginclass(
            cmdlinewinclass,
            NULL,
            hinst,
            Cmdlinewinproc) < 0 )
        return -1;
    Addtolist(0, 0, "Command line plugin v1.10");
    Addtolist(0, -1, "  Written by Oleh Yuschuk");
    // 从.ini 文件获取命令行窗口最后的位置，并确认窗口是完全可见的
    posx = Pluginreadintfromini(hinst, "Command line window X", CW_USEDEFAULT);
    posy = Pluginreadintfromini(hinst, "Command line window Y", CW_USEDEFAULT);
    maxx = GetSystemMetrics(SM_CXSCREEN) - DX;
    if (posx > maxx)
        posx = maxx;
    if (posx < 0)
        posx = 0;
    maxy = GetSystemMetrics(SM_CYSCREEN) - DY;
    if (posy > maxy)
    posy = maxy;
    if (posy < 0)
    posy = 0;
    if (Plugingetvalue(VAL_RESTOREWINDOWPOS) != 0 &&
        Pluginreadintfromini(hinst, "Restore command line window", 0) != 0
    )
```

```
        Createcmdlinewindow();
    return 0;
}

// 当被调试的程序停止下来时，将命令行窗口置于前台
extc void _export cdecl ODBG_Pluginmainloop(
    DEBUG_EVENT *debugevent)
{
    t_status status;
    if (hwcmd != NULL)
    {
        status = Getstatus();
        if (status == STAT_NONE || status == STAT_RUNNING)
            poponstop=1;
        else if (
            poponstop != 0 &&
            (status == STAT_STOPPED || status == STAT_FINISHED)
        ){
            SetForegroundWindow(hwcmd);
            SetFocus(hwbox);
            poponstop = 0;
        }
    }
}

// 将命令历史保存到.udd 文件中
extc void _export cdecl ODBG_Pluginsaveudd(
    t_module *pmod,
    int ismainmodule)
{
    int i;
    if (ismainmodule==0)
        return;         // 仅将历史保存在主文件中
    //从.udd 文件中恢复数据时，最后保存的历史条目将成为第一条
    for (i = nhist-1; i >= 0; i--)
    {
        Pluginsaverecord(TAG_CMDLINE, strlen(hist[i])+1, hist[i]);
    }
}

//从.udd 文件中恢复数据
extc int _export cdecl ODBG_Pluginuddrecord(
    t_module *pmod,
    int ismainmodule,
```

```
    ulong tag,
    ulong size,
    void *data)
{
    if (tag != TAG_CMDLINE)
        return 0;
    if (ismainmodule == 0)
        return 0;
    Addline((char *)data);
    return 1;
}

extc int _export cdecl ODBG_Pluginmenu(
    int origin,
    char data[4096],
    void *item)
{
    if (origin != PM_MAIN)
        return 0;
    strcpy(data,"0 &Command line\tAlt+F1 | 1 &Help , 2 &About");
    return 1;
}

extc void _export cdecl ODBG_Pluginaction(
    int origin,
    int action,
    void *item)
{
    if (origin != PM_MAIN)
        return;
    switch (action)
    {
        case 0:         // "Command line"
            Createcmdlinewindow();
            break;
        case 1:         // "Help"
            WinHelp(hwmain,"cmdline.hlp",HELP_CONTENTS,0);
            break;
        case 2:         // "About"
            MessageBox(hwmain,
                "Command line plugin v1.10\nWritten by Oleh Yuschuk",
                "Command line",
                MB_OK | MB_ICONINFORMATION);
            break;
```

```
        default: break;
  }
}

// 命令行窗口识别全局快捷键 Alt+F1
extc int _export cdecl ODBG_Pluginshortcut(
    int origin,
    int ctrl, int alt, int shift, int key,
    void *item)
{
    if (ctrl == 0 && alt == 1 && shift == 0 && key == VK_F1)
    {
        Createcmdlinewindow();
        return 1;
    }
    return 0;
}

// 用户打开新的程序或者重新调试当前程序时，清除命令行历史。
extc void _export cdecl ODBG_Pluginreset(void)
{
        poponstop=1;
        Addline(NULL);
}

extc int _export cdecl ODBG_Pluginclose(void)
{
    RECT rc;
    // 为了自动恢复，在.ini 文件中标识命令行窗口是否被打开并保存坐标
    Pluginwriteinttoini(hinst, "Restore command line window", hwcmd!=NULL);
    if (hwcmd != NULL)
    {
        GetWindowRect(hwcmd,&rc);
        Pluginwriteinttoini(hinst, "Command line window X", rc.left);
        Pluginwriteinttoini(hinst, "Command line window Y", rc.top);
    }
    return 0;
}

extc void _export cdecl ODBG_Plugindestroy(void)
{
    Unregisterpluginclass(cmdlinewinclass);
}
```

```
extc int _export cdecl ODBG_Plugincmd(
    int reason,
    t_reg *reg,
    char *cmd)
{
    char answer[TEXTLEN];
    // 命令行插件只接受以句点（.）开头的命令
    if (cmd == NULL || cmd[0] != '.' || cmd[1] == '\0')
        return 0;
    if (Execute(cmd+1, answer) == 0)    //命令被执行
        Addline(cmd+1);     // 当命令有效时，添加一行记录
    if (hwmain != NULL && hwerr != NULL)
        SetWindowText(hwerr, answer);
    return 1;
}
```

3、Bookmark

和 Command 相比，Bookmark 更长了——因为所有功能都在这一个文件中实现。

阅读 bookmark.c 是很划得来的，我们至少可以从中学到以下内容：

- 有序表的定义、构造、操作、使用
- 如何创建多种样式的菜单以及与数据相关的菜单项
- 如何与最常用的 CPU 窗口交互
- 更复杂但更接近于实际的消息处理过程
- OllyDbg 中对窗口绘制的支持和简化
- 如何在内存中读取代码并将其反汇编

需要理解的地方已经在代码中以注释的方式标明。再次提示，请先到 OllyDbg 中试用一下这个插件，了解它的功能再来阅读代码，会轻松很多。

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include "plugin.h"

HINSTANCE       hinst;
HWND            hwmain;
char            bookmarkwinclass[32];

// 自己定义有序表
// 表中须有双字型的 address、size、type
typedef struct t_bookmark {
    ulong   index;      // 书签的索引，0 — 9
    ulong   size;       // 索引大小，在这里始终为 1
    ulong   type;       // 记录类型，始终为 0
    ulong   addr;       // 书签的地址
```

```
} t_bookmark;

t_table     bookmark;   // 有序表描述符

int Bookmarksortfunc(
    t_bookmark *b1,
    t_bookmark *b2,
    int sort);
LRESULT CALLBACK Bookmarkwinproc(
    HWND hw,
    UINT msg,
    WPARAM wp,
    LPARAM lp);
int Bookmarkgettext(
    char *s,
    char *mask,
    int *select,
    t_sortheader *ph,
    int column);
void Createbookmarkwindow(void);

BOOL WINAPI DllEntryPoint(
    HINSTANCE hi,
    DWORD reason,
    LPVOID reserved)
{
    if (reason==DLL_PROCESS_ATTACH)
        hinst = hi;
    return 1;
}

extc int _export cdecl ODBG_Plugindata(
    char shortname[32])
{
    strcpy(shortname,"Bookmarks");
    return PLUGIN_VERSION;
}

extc int _export cdecl ODBG_Plugininit(
    int ollydbgversion,
    HWND hw,
    ulong *features)
{
    if (ollydbgversion<PLUGIN_VERSION)
```

```
        return -1;
    hwmain = hw;
    // 初始化有序表描述符 bookmark
    if(
        Createsorteddata(
            &(bookmark.data),               // 有序数据的描述符
            "Bookmarks",                    // 有序数据的名称
            sizeof(t_bookmark),             // 表项大小
            10,                             // 初始化表项数
            (SORTFUNC *)Bookmarksortfunc,   // 自定义的排序函数
            NULL                            // 自定义的析构函数
            )
        != 0
        )               // 初始化失败，未分配空间
        return -1;      // 因此直接返回，无需回收
    if(
        Registerpluginclass(
            bookmarkwinclass,
            NULL,
            hinst,
            Bookmarkwinproc) < 0 )          // 注册窗口类失败
    {
        Destroysorteddata(          // 注销有序表描述符
            &(bookmark.data));      // 并释放分配的内存
        return -1;
    }
    Addtolist(0,0,"Bookmarks sample plugin v1.10 (plugin demo)");
    Addtolist(0,-1,"  Copyright (C) 2001-2004 Oleh Yuschuk");
    if(
        Plugingetvalue(VAL_RESTOREWINDOWPOS) != 0 &&
        Pluginreadintfromini(
            hinst,"Restore bookmarks window",0) != 0
    )
        Createbookmarkwindow();
  return 0;
}

// 自定义的排序函数
// b1 > b2，返回 1；b1 == b2，返回 0；b1 < b2，返回-1
int Bookmarksortfunc(
    t_bookmark *b1,
    t_bookmark *b2,
    int sort)
{
```

```
    int i = 0;
    if (sort == 1)  // 以 addr 排序
    {
        if (b1->addr < b2->addr)
            i = -1;
        else if (b1->addr > b2->addr)
            i = 1;
    }
    if (i == 0)     //若 addr 相同，则以 index 排序
    {
        if (b1->index < b2->index)
            i = -1;
        else if (b1->index > b2->index)
            i = 1;
    }
    return i;
}

// 如果一个窗口用来显示有序表，应当把它收到的
// 绝大部分消息传递给 Tablefunction()先处理
LRESULT CALLBACK Bookmarkwinproc(
    HWND hw,
    UINT msg,
    WPARAM wp,
    LPARAM lp)
{
    int i, shiftkey, controlkey;
    HMENU menu;
    t_bookmark *pb;
    switch (msg)
    {
        // 正常的 Windows 消息
        case WM_DESTROY:
        case WM_MOUSEMOVE:
        case WM_LBUTTONDOWN:
        case WM_LBUTTONDBLCLK:
        case WM_LBUTTONUP:
        case WM_RBUTTONDOWN:
        case WM_RBUTTONDBLCLK:
        case WM_HSCROLL:
        case WM_VSCROLL:
        case WM_TIMER:
        case WM_SYSKEYDOWN:
            Tablefunction(&bookmark,hw,msg,wp,lp);
```

```
        // 只退出不返回，将消息再传给最后的 DefMDIChildProc()
        break;
    // 滚动或选择的消息
    case WM_USER_SCR:
    case WM_USER_VABS:
    case WM_USER_VREL:
    case WM_USER_VBYTE:
    case WM_USER_STS:
    case WM_USER_CNTS:
    case WM_USER_CHGS:
        // 这些消息给 Tablefunction()处理完就行了
        return Tablefunction(&bookmark,hw,msg,wp,lp);
    // MDI 窗口的 WM_WINDOWPOSCHANGED 消息
    case WM_WINDOWPOSCHANGED:
        return Tablefunction(&bookmark,hw,msg,wp,lp);
    case WM_USER_MENU:
        menu=CreatePopupMenu();
        // 定位被选的书签，需使用 Getsortedbyselection()
        // 因为有序数据采用了特殊的排序索引表
        pb = (t_bookmark *)Getsortedbyselection(
                &(bookmark.data),
                bookmark.data.selected);
        if (menu != NULL && pb != NULL)     // 构建弹出菜单
        {
            AppendMenu(menu, MF_STRING, 1, "&Follow\tEnter");
            AppendMenu(menu, MF_STRING, 2, "&Delete\tDel");
        }
        // 这里也要让 Tablefunction()先处理一下
        // 它的返回值需要是依据消息类型而定的
        i = Tablefunction(&bookmark,
                hw,WM_USER_MENU,0,(LPARAM)menu);
        if (menu != NULL)
            DestroyMenu(menu);
        if (i == 1)     // 选择了跟随
            // Setcpu()让 CPU 窗口转到指定的指令
            Setcpu(
                0, pb->addr, 0, 0,
                CPU_ASMHIST | CPU_ASMCENTER | CPU_ASMFOCUS
                );
        else if (i==2)  // 选择了删除
        {
            Deletesorteddata(&(bookmark.data), pb->index);
            // 需要自己更新窗口
            InvalidateRect(hw,NULL,FALSE); };
```

```
            return 0;
        case WM_KEYDOWN:
            shiftkey = GetKeyState(VK_SHIFT) & 0x8000;
            controlkey = GetKeyState(VK_CONTROL) & 0x8000;
            if (wp == VK_RETURN && shiftkey == 0 && controlkey == 0)
            {       // 回车键，跟随
                pb = (t_bookmark *)Getsortedbyselection(
                    &(bookmark.data),
                    bookmark.data.selected
                    );
                if (pb != NULL)
                    Setcpu(
                        0,pb->addr,0,0,
                        CPU_ASMHIST | CPU_ASMCENTER | CPU_ASMFOCUS
                        );
            }
            else if (wp == VK_DELETE && shiftkey == 0 && controlkey == 0)
            {       // DEL 键，删除
                pb = (t_bookmark *)Getsortedbyselection(
                    &(bookmark.data),
                    bookmark.data.selected
                    );
                if (pb != NULL)
                {
                    Deletesorteddata(&(bookmark.data), pb->index);
                    InvalidateRect(hw,NULL,FALSE);
                }
            }
            else
                Tablefunction(&bookmark,hw,msg,wp,lp);
            break;
        case WM_USER_DBLCLK:    // 双击，跟随
            pb=(t_bookmark *)Getsortedbyselection(
                &(bookmark.data),
                bookmark.data.selected
                );
            if (pb!=NULL)
                Setcpu(
                    0,pb->addr,0,0,
                    CPU_ASMHIST | CPU_ASMCENTER | CPU_ASMFOCUS
                    );
            return 1;
        case WM_USER_CHALL:
        case WM_USER_CHMEM:
```

```
            // 重绘窗口
            InvalidateRect(hw,NULL,FALSE);
            return 0;
        case WM_PAINT:
            // 所有 OllyDbg 窗口都应该使用 Painttable()来绘制
            Painttable(hw,&bookmark, Bookmarkgettext);
            return 0;
        default: break;
    }
    return DefMDIChildProc(hw,msg,wp,lp);
}

extc void _export cdecl ODBG_Pluginmainloop(
    DEBUG_EVENT *debugevent)
{
}

#define TAG_BOOKMARK   0x236D420AL

extc void _export cdecl ODBG_Pluginsaveudd(
    t_module *pmod,
    int ismainmodule)
{
    int i;
    ulong data[2];
    t_bookmark *pb;
    if (ismainmodule == 0)
        return;
    pb=(t_bookmark *)bookmark.data.data;
    for (i = 0; i < bookmark.data.n; i++, pb++)
    {
        data[0] = pb->index;
        data[1] = pb->addr;
        Pluginsaverecord(
            TAG_BOOKMARK,
            2*sizeof(ulong),
            data);
    }
}

extc int _export cdecl ODBG_Pluginuddrecord(
    t_module *pmod,
    int ismainmodule,
    ulong tag,
```

```
    ulong size,
    void *data)
{
    t_bookmark mark;
    if (ismainmodule == 0)
        return 0;
    if (tag!=TAG_BOOKMARK)
        return 0;
    mark.index=((ulong *)data)[0];
    mark.size=1;
    mark.type=0;
    mark.addr=((ulong *)data)[1];
    // 将从.udd 文件中读取的记录添加到列表中
    Addsorteddata(&(bookmark.data),&mark);
    return 1;
}

extc int _export cdecl ODBG_Pluginmenu(
    int origin,
    char data[4096],
    void *item)
{
    int i, n;
    t_bookmark *pb;
    t_dump *pd;
    switch (origin)
    {
        case PM_MAIN:   // 主菜单
            strcpy(data,"0 &Bookmarks|1 &About");
            return 1;
        case PM_DISASM: // 反汇编窗口
            pd=(t_dump *)item;
            if (pd == NULL || pd->size == 0)
                return 0;   // 窗口是空的，就不显示菜单了
            // 开始构造二级弹出菜单的描述字符串
            // sprintf()的返回值是它所构造的字符串长度
            n=sprintf(data,"Bookmark{");
            // 如果还有空的书签位，并且有反汇编记录被选择
            // 就显示一个“插入书签 n”
            pb=(t_bookmark *)bookmark.data.data;
            for (i = 0; i < bookmark.data.n; i++)
                if (pb[i].index != (ulong)i)
                    break;
            if (i < 10 && pd->sel1 > pd->sel0)
```

```
                n += sprintf(
                        data+n,
                        "%i &Insert bookmark %i\tAlt+Shift+%i,",
                        i, i, i);
            // 显示“删除书签 i”
            for (i = 0; i < bookmark.data.n; i++)
            {
                n += sprintf(
                        data+n,
                        "%i Delete bookmark %i,",
                        pb[i].index+10 ,pb[i].index);
            }
            // 添加一个分栏符
            data[n++]='|';
            // 显示“前往书签 i Alt+i”
            for (i = 0; i < bookmark.data.n; i++)
            {
                if (pb[i].addr == pd->sel0)
                    continue;
                n += sprintf(
                        data+n,
                        "%i Go to bookmark %i\tAlt+%i,",
                        pb[i].index+20,
                        pb[i].index,
                        pb[i].index);
            }
            // 结束构造二级弹出菜单的描述字符串
            sprintf(data+n,"}");
            return 1;
        default: break;
    }
    return 0;
}

extc void _export cdecl ODBG_Pluginaction(
    int origin,
    int action,
    void *item)
{
    t_bookmark mark,*pb;
    t_dump *pd;
    if (origin == PM_MAIN)
    {
        switch (action)
```

```
        {
            case 0:
                Createbookmarkwindow();
                break;
            case 1:
                MessageBox(hwmain,
                    "Bookmark plugin v1.10\n"
                    "(demonstration of plugin capabilities)\n"
                    "Copyright (C) 2001-2004 Oleh Yuschuk",
                    "Bookmark plugin",
                    MB_OK|MB_ICONINFORMATION);
                break;
            default: break;
        }
    }
    else if (origin == PM_DISASM)
    {
        pd = (t_dump *)item;
        if (action >= 0 && action < 10)//插入书签
        {
            mark.index = action;
            mark.size = 1;
            mark.type = 0;
            mark.addr = pd->sel0;
            //添加有序数据
            Addsorteddata(&(bookmark.data), &mark);
            if (bookmark.hw != NULL)
                InvalidateRect(bookmark.hw, NULL, FALSE);
        }
        else if (action>=10 && action<20)  //删除书签
        {
            //查找有序数据
            pb = (t_bookmark *)Findsorteddata(
                &(bookmark.data),
                action-10);
            if (pb!=NULL)
            {
                //删除有序数据
                Deletesorteddata(
                    &(bookmark.data),
                    action-10);
                if (bookmark.hw != NULL)
                    InvalidateRect(bookmark.hw, NULL, FALSE);
            }
```

```
        }
        else if (action >= 20 && action < 30)  //跳转到书签
        {
            pb = (t_bookmark *)Findsorteddata(
                    &(bookmark.data),
                    action-20);
            if (pb!=NULL)
            {
                Setcpu(
                    0, pb->addr, 0, 0,
                    CPU_ASMHIST | CPU_ASMCENTER | CPU_ASMFOCUS);
            }
        }
    }
}

// 函数 Painttable()提供了 OllyDbg 窗口重绘的功能
// 只需自己写另一个函数提供字符串即可
int Bookmarkgettext(
    char *s,
    char *mask,
    int *select,
    t_sortheader *ph,
    int column)
{
    int n;
    ulong cmdsize, decodesize;
    char cmd[MAXCMDSIZE], *pdecode;
    t_memory *pmem;
    t_disasm da;
    t_bookmark *pb = (t_bookmark *)ph;
    if (column == 0)    //第一栏，书签名称：Alt+i
    {
        n = sprintf(s, "Alt+%i", pb->index);
        *select = DRAW_MASK;
        memset(mask, DRAW_GRAY, 4);
        mask[4] = DRAW_NORMAL;
    }
    else if (column == 1)   // 第二栏，书签地址
        n = sprintf(s, "%08X", pb->addr);
    else if (column == 2)   //第三栏，反汇编指令
    {
        // 查找包含代码的内存块
        pmem = Findmemory(pb->addr);
```

```
        if (pmem == NULL)
        {
            *select=DRAW_GRAY;
            return sprintf(s,"???");
        }
        // 计算指令长度
        cmdsize = pmem->base + pmem->size - pb->addr;
        if (cmdsize>MAXCMDSIZE)
            cmdsize=MAXCMDSIZE;
        if (
            Readmemory(     // 在内存中读指令
                cmd,
                pb->addr,
                cmdsize,
                MM_RESTORE | MM_SILENT
                )
            != cmdsize
            )
        {
            *select = DRAW_GRAY;
            return sprintf(s,"???");
        }
        // 查找解码信息
        pdecode = Finddecode(pb->addr, &decodesize);
        if (decodesize < cmdsize)
            pdecode=NULL;
        // 将解码信息反汇编成指令
        // Disasm()是API中最复杂也最常用的函数之一
        Disasm(
            cmd,            // 要解码的二进制指令
            cmdsize,        // 指令的长度
            pb->addr,       // 指令的地址
            pdecode,        // 分析器产生的解码数据
            &da,            // 接收反汇编结果的t_disasm结构
            DISASM_CODE,    // 反汇编模式
            0               // 包含了寄存器的线程标识
            );
        strcpy(s, da.result);
        n=strlen(s);
    }
    else if (column == 3)   // 第四栏，注释
        // 只显示用户的注释
        n=Findname(pb->addr, NM_COMMENT, s);
    else n=0;
```

```
    return n;
}

// 创建 MDI 窗口时，OllyDbg 将完成绝大部分工作
// 插件只需描述要创建几栏以及他们的属性
void Createbookmarkwindow(void) {
    if (bookmark.bar.nbar==0)
    {
        bookmark.bar.name[0]    =   "Bookmark";
        bookmark.bar.defdx[0]   =   9;
        bookmark.bar.mode[0]    =   0;
        bookmark.bar.name[1]    =   "Address";
        bookmark.bar.defdx[1]   =   9;
        bookmark.bar.mode[1]    =   0;
        bookmark.bar.name[2]    =   "Disassembly";
        bookmark.bar.defdx[2]   =   32;
        bookmark.bar.mode[2]    =   BAR_NOSORT;
        bookmark.bar.name[3]    =   "Comment";
        bookmark.bar.defdx[3]   =   256;
        bookmark.bar.mode[3]    =   BAR_NOSORT;
        bookmark.bar.nbar       =   4;
        bookmark.mode=
            TABLE_COPYMENU | TABLE_SORTMENU |
            TABLE_APPMENU | TABLE_SAVEPOS | TABLE_ONTOP;
        bookmark.drawfunc = Bookmarkgettext;
    }
    // 如果窗口已经存在，Quicktablewindow()并不会重复创建它
    // 而只是激活它，并拉到前台来
    Quicktablewindow(&bookmark,15,4,bookmarkwinclass,"Bookmarks");
}

extc int _export cdecl ODBG_Pluginshortcut(
    int origin,
    int ctrl, int alt, int shift, int key,
    void *item)
{
    t_dump *pd;
    t_bookmark mark,*pm;
    // 如果在反汇编窗口传来 Alt+Shift+Fi，设置第 i 个书签
    if (ctrl == 0 && alt != 0 && key >= '0' && key <= '9')
    {
        if (shift != 0 && origin == PM_DISASM && item != NULL)
        {
            pd = (t_dump *)item;
```

```
            mark.index = key-'0';
            mark.size = 1;
            mark.type = 0;
            mark.addr = pd->sel0;
            Addsorteddata(&(bookmark.data),&mark);
            if (bookmark.hw!=NULL)
                InvalidateRect(bookmark.hw,NULL,FALSE);
            return 1;
        }
        else if (shift == 0) // Alt+Fi，跳到第 i 个书签
        {
            pm=Findsorteddata(      //查找有序表数据
                &(bookmark.data),
                key-'0');
            if (pm==NULL)
                // 在 OllyDbg 底部闪一下错误消息
                Flash("Undefined bookmark");
            else
                // CPU 窗口转到某个位置
                Setcpu(0, pm->addr, 0, 0,
                    CPU_ASMHIST | CPU_ASMCENTER | CPU_ASMFOCUS);
            return 1;
        }
    }
    return 0;
}

extc void _export cdecl ODBG_Pluginreset(void)
{
    // 删除有序数据
    Deletesorteddatarange(
        &(bookmark.data),
        0,
        0xFFFFFFFF);
}

extc int _export cdecl ODBG_Pluginclose(void)
{
    Pluginwriteinttoini(
        hinst,
        "Restore bookmarks window",
        bookmark.hw != NULL);
    return 0;
}
```

```
extc void _export cdecl ODBG_Plugindestroy(void)
{
    Unregisterpluginclass(bookmarkwinclass);
    // 注销有序表描述符，释放内存
    Destroysorteddata(&(bookmark.data));
}
```

参考文献

[1] Oleh Yuschuk. *OllyDbg Plugin API* v1.10 . http://www.ollydbg.de/plug110.zip
[2] 罗聪. OllyDbg 插件编写入门. 程序员. 2005.9
[3] prince. 简单编写 OD 插件. http://bbs.pediy.com/showthread.php?t=11621