PB 的扩展 DLL 开发（超级篇）

(PB 史上第一次开放的开发技术)

PowerBuilder (pb)作为一个基于数据库的 CS 开发工具，在功能方面不够全面，需要使用 DLL 做功能扩展。

通常对 PB 写 DLL，有 3 种方法。

方法 1：通用 DLL。这种方式的 DLL，所有能写标准 DLL 的语言都可以写。但缺点也比较明显，无法直接访问 PB 对象和属性、事件这些个性内容，不合适直接返回字符串，通过参数返回数据时，需要预分配内存，如果计算错误，会导致程序崩溃。

方法 2：PBNI 法（PowerBuilder Native Interface），即官方开放式接口。此方法突破方法 1 的限制，可以访问 PB 内部对象、事件、属性。但也存在不同版本之间有一定的兼容性问题。目前广为使用。

这里，我介绍的是第 3 种方法，即 system library 方法。

一、system library 方法的来由

一次偶然的机会，逛到了 http://www.vodacoder.com/Sources/fastfuncs 这个网站，看到了全新的 DLL for PB 开发方法。不过，这个里面的 DEMO 都不够成熟，网上查不到任何与此相关的内容，官方也无相关内容。

于是本人开始潜心研究 （研究过程此处省略百万字.....）。初步掌握了 system library 的开发方法，分享出来与大家共同研究。

二、system library 方法与方法 1 的区别

1、普通标准库 DLL 的函数声明：

function long test() library "test.dll"

如果是 PB9 以下版本的 DLL，升级到高版本时，会自动加上;ansi，成了：

function long test() library "test.dll" alias for "test;ansi"

这里说一下，“;ansi”这个东西很恶心，误导害了一大堆 PB 程序员，以为高版本声明必须要加这么个玩艺。事实上，在使用 winapi 时，高版本有对应的函数，根本不需要;ansi 这么个东西。PB 升级时自动加上这个，纯属脱裤子放屁。

例如：

PB9: FUNCTION ulong SetWindowText(ulong hwnd,ref string lpString) LIBRARY "user32.dll" ALIAS FOR "SetWindowTextA"

它升级到 PB10 以上，会自动变成 FUNCTION ulong SetWindowText(ulong hwnd,ref string lpString) LIBRARY "user32.dll" ALIAS FOR "SetWindowTextA;ansi"。这个功能很垃圾。实际上应该是：

PB10 及以上: FUNCTION ulong SetWindowText(ulong hwnd,ref string lpString) LIBRARY "user32.dll" ALIAS FOR "SetWindowTextW"

注意，不加;ansi，而是最后的那个 A 改为 W，表示使用 UNICODE 编码。

2、system library 的声明方式

看上面标准库的声明，它有个 library 关键词。那么 system library 方法，自然就应该是 system library 作为关键词了。system librar 就是 PB 自己内部函数的实现方式。

function long systest() system library "test.dll" alias for "systest"

注意："system library"这个关键词，它告诉 PB，我是自己人，不是请来打工的那些个

家伙。

实际上，我们在 PB 里用的函数，全部都是以 system library 方式存在于 PBVMxxx.DLL 中。

例如 ：如果你不喜欢 MessageBox 这个名称，想使用 MsgBox 这个名称 ，而所有功能要一样。可以这样声明一个函数：

（假设是 PB9） function int MsgBox(string title,string text) system library "pbvm90.dll" alias for "fnMessageBox"

（假设是 PB10） function int MsgBox(string title,string text) system library "pbvm100.dll" alias for "fnMessageBox"

然后 MsgBox("","hello") 与 MessageBox("","hello")功能完全一样，只不过函数改了个名而已。用 system librar 方式声明的函数，即使是从 PB9 升级到高版本，PB 也不会自动加上“;ansi”这么个垃圾东西，因为是自己人嘛，不是外来打工的家伙，待遇不一样的。

三、如何开始写自己的 system library DLL

system library DLL 有自己的固有特征，它通常是：

#include "pbSysLib.h" //这是我自己实现的一个头文件，具体作用就是加载 PBVMxxxx.dll，导出相关 SDK 函数

```
__declspec(dllexport) DWORD __stdcall FuncName(POB_THIS obThis,int nArgCount)
{
    BOOL isnull;
    return 1;
}
```

FuncName 是函数名，POB_THIS obThis 这个由 PB 调用时自动传入，它指向一个结构，可以唯一标识一个 PB 的虚拟空间，或者也可以理解为一个 session。int nArgCount 指的是本次调用有几个参数，而真正的参数放在 obThis 里的一个指针数组里，通过参数个数，可以去取这些个参数。

因此，system library DLL 函数还是比较简单的，这也符合 PB 这种古董级文物的身份，简单而有效。

接下来，就是使用 PBVM 的各种内部函数，来驾驶我们的各种功能，可以在 C、C++ 里，访问 PB 的所有功用，并为 PB 扩展所有需要的功能。

这里就一个具体函数，做一些说明

PB 里的声明： function string TestRef(readonly string src,ref string dst) system library "PBJson.dll" alias for "TestRef"

使用时可以是：

```
string src,dst
src = "hello world"
TestRef(src,dst)
```

这时候 dst 的结果是：this is return by ref: [hello word]

DLL 里的函数源码是：

```
DLLEXPORT DWORD WINAPI TestRef(POB_THIS obThis, int nArgs)
{
    BOOL success = FALSE;
    BOOL isnull = FALSE;
    POT_LVALUE_INFO info = NULL;
```

```
    TCHAR *lpSrc = (TCHAR *)ot_get_valptr_arg(obThis, &isnull);//取第一个参数，指针型，指向 src
    POB_DATA lv = ot_get_next_lvalue_arg(obThis, &info); //取第二个参数，注意第二个参数是 ref 类型
    POB_DATA v = NULL;
    if (lv)
    {
        POT_REF_PAK v = (POT_REF_PAK)lv->val.ptr; //获取得引用定义
        POB_DATA lpArg = ot_access_ref_data(obThis, v); //获取得引用定义里实际指向 PB 的那个变量，即上面的 dst
        if (lpSrc && lpArg && ob_get_data_type(lpArg) == STRING_TYPE) //判断，确实都是有效指针，并且第二个参数是 string 类型
        {
            TCHAR buffer[1024] = { 0 };
            _stprintf(buffer, _T("this is return by ref: [%s]"), lpSrc);
            ob_set_data_ptr(lpArg, ob_dup_string(obThis, buffer), STRING_TYPE, 1); //注意这里面有个 ob_dup_string，它返回了一个从 buffer 复制出来的字符串的字符串，把这个变量绑定给引用的 dst 变量，而不是直接把 buffer 绑定给 dst 变量。
/**
特别强调，这里的 ob_dup_string 是必不可少的。PB 有自己的内存管理，这个复制的字符串，在 PB 函数生命周期结束后，它会被自动释放，内存回收。
这里如果写成：
            TCHAR *buffer = new buffer[1024];
            _stprintf(buffer, _T("this is return by ref: [%s]"), lpSrc);
            ob_set_data_ptr(lpArg, buffer, STRING_TYPE, 1);
也就是自己分配了内存，并且返回，这个函数调用后，PB 是会崩溃的。原因是 buffer 没有使用 PB 的内存堆。
            TCHAR *buffer = ob_alloc_string(obThis,1024);
            _stprintf(buffer, _T("this is return by ref: [%s]"), lpSrc);
            ob_set_data_ptr(lpArg, buffer, STRING_TYPE, 1);
如果是这样，那就毫无问题了，因为 TCHAR *buffer = ob_alloc_string(obThis,1024);是在 PB 自己的内存堆上分配 的内存，可以被 PB 正确释放回收。
*/
            success = TRUE;
        }
    }
    OB_DATA obReturn = { 0 };
    ob_set_data_string(&obReturn, success, BOOL_TYPE, OB_INSTVAR_FIELD);
    ot_set_return_val(obThis, &obReturn);
    return 1;
}
```

如果你对 system library 相关开发方式感兴趣，可到 QQ 群 624409252 共享里大自在的专用目录下下载 DEMO。