PB 的扩展 DLL 开发（超级篇）（三）

(PB 史上第一次开放的开发技术)

最核心的一个结构，OB_THIS ，这个在上一章已做过介绍。本章介绍 OB_DATA 这个结构。这个结构的重要程度仅次于 OB_THIS。

OB_DATA 是 PB 里各种类型的数据参数在 DLL 里的存在形式。你在 PB 里写的代码例如 Int I; long n;string ls_text; window w;..... 这些，对照到 C 内部的映射，它们全部都是一个 OB_DATA 指针。在这个函数里__declspec(dllexport) DWORD __stdcall FuncName(POB_THIS obThis,int nArgCount) ，我们已经知道 obThis 指向的是一个 VM 或者 session，另一个参数是 nArgCount，它指明了参数个数。那参数在哪呢？obThis->evaled_arglist 这个结构成员保存的就是参数列表。我们可以这样：

```
POB_DATA lpArgs = (POB_DATA) obThis->evaled_arglist;
For(int i=0;i< nArgCount;i++)
{
    POB_DATA lpArg = lpArgs[i];
    ……
}
```

（实际写代码过程中，一般不会象上面这样使用，这个方法太不安全了。更好的方法会在下面章节里介绍。）

来取得一个在 PB 代码里传入 进来的参数。这样不管多少个参数，都可以能取得到。

这里说一下 PB 里的不固定参数，比如有这样一个函数：

```
function int Printf(string strFormat,...) system library "PbJson.dll" alias for "Printf"
```

就是以 ... 为声明方式来传递不固定参数。对 DLL 内部来说，可以把所有参数给取出来，与实际参数名称没有任何关系。

先看看 OB_DATA 的声明：

```
typedef struct ob_data
{
    OB_VALUE            val;
    OB_INFO_FLAGS       info;           // Data node info flags
    OB_CLASS_ID         type;           // Data Type
} OB_DATA, FAR* POB_DATA;
```

它的项目相关成员和结构的声明：

```
typedef USHORT      OB_BASE_ID;
typedef OB_BASE_ID      OB_CLASS_ID, FAR* POB_CLASS_ID;
typedef USHORT          OB_INFO_FLAGS, FAR* POB_INFO_FLAGS;
typedef union ob_value
{
    SHORT               int_val;        // Integer value
    FLOAT               fl_val;             // Float value
    PVOID               ptr;            // Ptr to value
    OB_CONST_REF        const_ref;      // Constant value
```

```
        PVOID           ob_inst;        // There for compatibility, ptr should be
used instead
        USHORT          id;             // Id value
        USHORT          uint_val;       // Unsigned integer
        LONG            long_val;       // Long integer
        ULONG           ulong_val;      // Unsigned Long integer
        BYTE            byte_val;       // Byte value
    } OB_VALUE, FAR* POB_VALUE;
```

OB_DATA 这个结构，有 3 个成员：

1、OB_VALUE val; 这是一个 union 体，它的大小是 8 个字节。它实际保存的是 PB 变量或参数的值或者指针，这些值保存 PB 里的值，它的指针是由专用内存管理函数从自己的堆上分配，可以由 PB 自己管理，自动释放。这个值的具体的类型由 OB_CLASS_ID type 决定。

3、OB_CLASS_ID 就一个 ushort 的类型，也就是一个整值，它是可以是下列这些值：

```
#define NO_TYPE 0
#define INT_TYPE 1
#define LONG_TYPE 2
#define FLOAT_TYPE 3
#define DOUBLE_TYPE 4
#define DEC_TYPE 5
#define STRING_TYPE 6
#define BOOL_TYPE 7
#define ANY_TYPE 8
#define UINT_TYPE 9
#define ULONG_TYPE 10
#define BINARY_TYPE 11
#define DATE_TYPE 12
#define TIME_TYPE 13
#define DATETIME_TYPE 14
#define CURSOR_TYPE 15
#define PROC_TYPE 16
#define BASIC_TYPE 17
#define CHAR_TYPE 18
#define HANDLE_TYPE 19
#define LONGLONG_TYPE 20
#define BYTE_TYPE 21
#define NUMBER_OF_ottypes 22
```

看到这些定义，自然就联想到 PB 的那些变量类型了 int long ulong string 之类，没错，就是与 PB 的类型一一对应。只不过，有些类型的值它是直接放在 ob_value 里的，比如 INT_TYPE 就可以直接取 lpArg->int_val ，LONG_TYPE 可以直接取 lpArg->long_val 。但有些值它存的是指针，比如 STRING_TYPE、BINARY_TYPE、DECI_TYPE、LONGLONG_TYPE，这些是以指针方式保存，得到指针后需要自己转换使用。

看最后一项：NUMBER_OF_ottypes 22，最多只有 22 个？显然不是的。

***这里要注意一下：它还有个 PVOID ob_inst 成员。当 OB_CLASS_ID type 的值大于 22 时，它是一个对象的实例句柄。在 PB 里，你的各种对象实例，window lw; commandbutton btn; datawindow dw ; datastore ds 这些，传进来 ，它们只有一个类型：OB_CLASS_ID，实例句柄保存在 ob_inst 这个成员里面这里成员使用的是 PVOID 类型，实际上，它有个专门的定义类型 OB_INST_ID，它的定义是这样的 typedef PVOID OB_INST_ID。

3、OB_INFO_FLAGS info。这个成员是干嘛的？开始的时候，我也迷糊好久，后来也没全弄清楚。

比如有个字符串传进来了:

PB 里: string str; str = “hello”;

DLL 里这样取：TCHAR *text = (TCHAR *)lpArg->ptr_val;这样没问题。

如果 PB 里: string str; setnull(str);

DLL 里这样取：TCHAR *text = (TCHAR *)lpArg->ptr_val;这样就可能有问题了，得到的 text 可能是个无效的指针。那我们怎么知道它是有效还是无效指针呢？那应该这样：

```
If(!(lpArg->info & DATA_NULLVAL_MASK))
{
    TCHAR *text = (TCHAR *)lpArg->ptr_val;
}
```

加上一个是不是空的判断。IS_NULL 是这样定义的：#define DATA_NULLVAL_MASK 0x0001。

所以， OB_INFO_FLAGS info 应该是各种标志，空标志，引用标志，作用域标志等等。

OB_DATA 实在是设计得很巧妙的一个结构！

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。