PB 的扩展 DLL 开发（超级篇）（四）

(PB 史上第一次开放的开发技术)

前面已介绍过 OB_THIS 和 OB_DATA 这两个结构。本章介绍如何获取到 PB 里传给 system library DLL 的各种参数和返回值。

一、参数

第一个函数：

function any GetGlobalVar(string strVarname) system library "PbJson.dll" alias for "GetGlobalVar"

这个函数它有个字符串参数，这个参数在 DLL 里是一个指针，所以取这个参数：

```
BOOL isnull = FALSE;
TCHAR* lpVarName = (TCHAR*)ot_get_valptr_arg(obThis, &isnull);
```

内置函数 ot_get_valptr_arg 可以取所有指针类的参数，然后进行强制转换。它有 2 个参数，obThis 指向 PB 传入的结构指针，isnull 以指针方式获取该参数是否为 NULL。PB 调用时：

```
String str
Any ret
Setnull(str)
Ret = GetGlobalVar(str)
```

这样调用时，因为 Setnull(str)的缘故，所以 DLL 里的 ot_get_valptr_arg(obThis, &isnull) 得到的 isnull 是 TRUE；

类似的函数或者宏，形式都是 **ot_get_xxxx_arg** 这样，例如：

```
ot_get_str_arg
ot_get_bool_arg
ot_get_time_arg
ot_get_date_arg
ot_get_datetime_arg
ot_get_enum_arg
ot_get_binary_arg
ot_get_int_arg
……
```

更多的可以自己查找头文件。

在取参数的时候，要注意取的次数不能大于参数的个数。例如

```
DLLEXPORT DWORD WINAPI SetGlobalVar(POB_THIS obThis, int nArgs)
```

这个函数里，nArgs 为 1，但你代码里进行了 2 次或 2 次以上 ot_get_xxxx_arg ，会导致指针越界，程序会崩溃，或者埋雷。如何避免这个问题呢？在 OB_THIS 里，有个成员 curr_arg_pos，这个在函数开始时值为 0，每调用 ot_get_xxxx_arg 个次，curr_arg_pos 就会增加 1。所以可以这样判断：

```
If(obThis->curr_arg_pos < nArgs)
    ot_get_xxxx_arg(….)
```

这样可以确保取参数时指针不越界，不发生异常情况。

上面是我们知道数据类型的情况下，可以这样取。但有时候，我们不知道数据类型，例如是变参时，PB 里参数声明为 “…”，或者是同一个函数，使用多态，提供不同类型的参数，这时候按参数类型取，就行不通了。这样其实也可以取，直接取 POB_DATA，然后我们自己判断类型：

```
POB_DATA obArg = ot_get_next_evaled_arg(obThis);
```

或

POB_DATA obArg = ot_get_next_evaled_arg_no_convert (obThis);

这样我们就得到一个 POB_DATA 指针，然后根据 ob_get_data_type(obArg)，得到参数的类型。再根据不同参数类型调用 ot_get_data_xxxx 函数，即可以取得实际变量值。

对于引用类型，即 PB 里声明哦 ref 的参数，可以用 ot_get_next_lvalue_arg 取得其引用参数。

我们知道，PB 是面向对象的程序，也许你不使用面向对象的编程方法，但它本身是面向对象的程序逻辑。所以，这里有个隐藏的参数，那就是当前对象是谁？比如你在 cb_1 的 clicked 里：

dw_1.create(ls_syntab)

这样调用 DLL 里的函数，那么当前对象就是 dw_1，它调用了 DW 内部的 create(内部名称不是这个，被 alias for 了)这个函数。我们没有把 dw_1 作为参数传给 Create 函数，那么 create 怎么知道当前对象是 dw_1 的呢？很简单，有一个函数可以实现这个：

OB_INST_ID obInstID= NULL;

ot_get_curr_obinst_expr(obThis, &obInstID, &isnull);

OB_INST_ID 是实例类型， ot_get_curr_obinst_expr 是取当前调用我这个函数的那个对象实例。也就是说，在 create 函数内部，有这个方法就知道调用者是 dw_1，obInstID 就是 dw_1 的实例句柄。通过 obInstID 实例句柄，我们可以取 dw_1 的所有属性，可以调用它所有的函数、事件。

二、返回值

DLL 函数被调用后，通常会有返回值。System libray 函数与普通函数不一样的是：它们的返回方式不一样。

DLL 里的 C/C++和其他普通函数一样，函数本身有返回值。通常是 return 1;如果是 return 0，则说明函数发生了错误，可以在 PB 里用 try 捕捉错误。所以，只要执行正确，就 return 1;

要把数据通过函数返回给 PB 代码，我们需要使用一个函数 ot_set_return_val 来实现。当然，如果是 subroutine 类函数，可以直接 ot_no_return_val，告诉 PB 我没有返回值。

ot_set_return_val 函数具体做什么呢？它其实只是把 ob_this->return_value 这个值设置了一下，PB 代码的返回值直接从 ob_this->return_value 这里取就行了，这个不用我们关心，它会自己去取。

BOOL success = FALSE;

…..

OB_DATA obReturn = { 0 };

ob_set_data_bool(&obReturn, success, BOOL_TYPE, OB_INSTVAR_FIELD);

ot_set_return_val(obThis, &obReturn);

比如这段代码，就是返回了一个 Boolean 类型的值给调用者，具体值是 success 代表的内容。

给 OB_DATA 设置值，有一系列函数或者宏，对照 ot_get_xxxx_arg ，也有一个系列 ob_set_data_xxxxx。

OB_DATA obReturn = { 0 };

ob_set_data_long(&obReturn, 1234, LONG_TYPE, OB_INSTVAR_FIELD);

ot_set_return_val(obThis, &obReturn);

比如这段代码，就是返回了一个 long 类型的值给调用者，具体值是 1234。

这些简单类型可以这样返回。指针类型，就需要分配空间后返回。例如：

TCHAR *lpText = …;

OB_DATA obReturn = { 0 };

ob_set_data_string(&obReturn, lpText, STRING_TYPE, OB_INSTVAR_FIELD);

ot_set_return_val(obThis, &obReturn);

这段代码，就是返回了一个 string 类型的值给调用者。

只是特别强调要注意，我们返回指针类型的数据时，必然在 PBVM 的堆上分配内存。

```
TCHAR *lpText =new TCHAR[32];
_tcscpy(lpText,_T("hello"));
OB_DATA obReturn = { 0 };
ob_set_data_string(&obReturn, lpText, STRING_TYPE, OB_INSTVAR_FIELD);
ot_set_return_val(obThis, &obReturn);
```

如果这样返回，PB 程序是会崩溃的。

所以，lpText 的分配就必须是以下几种方法之一：

```
1. TCHAR *lpText = ob_dup_string(obThis, _T("hello"));
2. TCHAR *lpText = ob_alloc_string(obThis, 32);
_tcscpy(lpText,_T("hello"));
3. TCHAR *lpText = pbstg_alc(obThis-> stgthis,32,obThis->subpool);
_tcscpy(lpText,_T("hello"));
```

这样分配的内存，才可以直接被设置为返回值，并且由 PB 内存管理机制管理，自动释放。注意一下，pbstg_alc 是所有内存分配的基础，其他各种内存分配方式，内部都是调用此函数进行。

还有种数据返回，是通过引用类型。上面说过取引用类型的参数，在得到引用类型参数后，直接 ob_set_data_xxxx 就可以设置引用类型的返回值。

凡是指针类型的返回值，都必须分配空间后，再设置指针值。这些包括 string,blob,date,time,datetime,double,longlong 等等。

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。