aardio 范例: 调用 C 语言生成 DLL

```
//aardio 调用 C 语言生成 DLL
import tcc;
var c = tcc();

//必须使用最新版 tcc 扩展库才能支持 UTF8,UTF16 字符串
c.code = /**
    #include <windows.h>
    #include <stdio.h>

    /*
    入口函数,该函数可以有也可以没有。

    入口函数会自动加锁以保证线性调用,要避免在DllMain内调用下列函数:
    1、调用LoadLibrary或其他可能加载DLL的API函数( CreateProcess等 )
    2、可能再次触发DllMain的函数,例如 CreateThread,ExitThread
    3、GetModuleFileName, GetModuleHandle 等其他可能触发系统锁的API函数
    总之在DllMain最好不要调用API函数.
    */
    int __stdcall DllMain(void * hinstDLL, unsigned long fdwReason, void * lpvReserved) {

        if (fdwReason == 1/*DLL_PROCESS_ATTACH*/ ){

        }
        return 1;
    }

    //__declspec(dllexport) 声明导出函数
    __declspec(dllexport) int Msgbox( HWND hwnd )
    {
        //定义一个结构体
        struct { const char * utf8message;int id; } argument = {
            .utf8message = "测试消息来自C语言",
            .id = GetCurrentThreadId()
        };

        /*
        _WM_THREAD_CALLBACK 使所有回调安全的转发到UI线程。
        _WM_THREAD_CALLBACK 可以跨线程跨语言并且不需要创建回调线程,适用任何普通winform对象。

        与其他回调方案的比较:
        raw.tocdecl raw.tostdcall 不能跨线程使用,
        thread.tocdecl,thread.tostdcall 需要创建回调线程。
        thread.command 则只能在aardio代码中使用,需要将窗体转换为thread.command对象。
        */
        SendMessage(
            hwnd,0xACCE/*_WM_THREAD_CALLBACK*/,
            (WPARAM )"onMessageChange( { string utf8message;int id } )", //要调用的窗体函数名( 结构体原型声明 ); 结构体原型声明应使用aardio语法
            (LPARAM )&argument //将前面定义的结构体作为调用参数
        );
     ;
        return argument.id;
    }

    typedef struct {
        int x;
        int y;
    }  Point;

    __declspec(dllexport) int Test (char * buf,char a,int b, unsigned long long c,Point * ppt,Point pt,double * pd)
    {
        sprintf(buf,"C语言接收到参数 a: %d b:%d c: %llu ppt->x: %d ppt->y:%d pt.x: %d pt.y:%d pd:%g\n",a,b,c,ppt->x,ppt->y,pt.x,pt.y,*pd);
    }
**/


/*
加载需要用到的动态库,或静态库
在"~\lib\tcc\.res\lib" 目录下查找 "动态库名.def" "静态库名.a"
也可以使用 vm.addLibPath() 函数添加搜索库的目录,
其实下面的DLL已经默认加载,这里仅用于演示。
*/
c.addLib(
    "user32",
    "kernel32",
    "gdi32"
)
c.output( "/bin.dll" ) //编译C源码,生成DLL
c.close(); //收工

//创建一个窗体以处理_WM_THREAD_CALLBACK线程回调命令。
import win.ui;
var winform = win.form({})
winform.messageOnly(); //窗体仅用于处理消息
winform.onMessageChange = function(param){
    winform.msgbox("调用:" + param.utf8message )
    win.quitMessage()

    //如果修改并返回参数传入的结构体，则修改C语言中的结构体
    param.id = 123456;
    return param;
}

//加载生成的DLL，下面的"cdecl"指定默认调用约定，免声明调用API时使用此约定，如不指定则默认为 "stdcall"
```

```
var dll = raw.loadDll( "/bin.dll",,"cdecl" );

/*
 aardio 中的字符串是只读的不应修改其内存,
 而 raw.buffer 可分配内存用于创建可修改的字节数组,
 而用 raw.buffer 创建的 buffer 可通用于几乎所有字符串函数。
*/
var buf = raw.buffer(100);

/*
也可以不声明直接调用 API: https://www.aardio.com/zh-cn/doc/library-guide/builtin/raw/directCall.html
使用调用 raw.loadDll 指定的 "cdecl" 调用约定
相对于声明API，免声明调用 API 是更优的选择，频繁调用的免声明 API 会被缓存而非重复创建，
长时间不使用的免声明 API 将会被自动释放。
*/
var ret,ppt,pd = dll.Test(
    buf, //相当于C语言中的 char buf[100] 或者 char * buf
    1,//小于32位的整型数值可以直接传递，自动兼容
    2,//32位整型数值可以直接传递,自动兼容
    math.size64(3), //无符号64位整数，可以传 math.size64 对象
    {int x = 4;int y = 5}, //aardio 在API参数中传结构体，总是传结构体指针，
    6,7, //直接在参数中用结构体传值极其罕见，类似这种字段为32位长的结构体字段可以直接展开为多个参数
    {double v = 8.1} //对等C中的 double * 这种指针，在 aardio 中转换为同类型的结构体指针即可
);
/*
如果C函数的参数使用了 double,float 等浮点数值参数（传值，而不是使用指针传址），
则必须先声明再调用，不声明直接调用无法支持这类参数。
*/

win.msgbox(buf) //在 aardio 中可以将 buffer 字节数组替代几乎所有字符串参数。

/*
声明API: https://www.aardio.com/zh-cn/doc/library-guide/builtin/raw/api.html
第二个参数指定cdecl调用约定(如果使用 raw.loadDll指定的默认调用约定，那么下面可以不指定 )
不建议在函数内部声明API，这会重复创建不必要的 API 对象（虽然也会释放，但 aardio 并非立即释放不使用的对象。
*/
Msgbox = dll.api( "Msgbox","int(addr str)", "cdecl" );
//可使用: aardio 『工具 > 转换工具 > API 转换』自动转为 aardio 声明。

var ret = Msgbox( winform.hwnd );

win.loopMessage()

/*
附：编写DLL避免导出函数名乱码（出现修饰名）的几种方法：
---------------------------------------------------------------------
1、C语言的导出函数使用默认的cdecl调用约定，不要用stdcall调用约定，就不会有修饰名，示例：

    __declspec(dllexport) int Add( int a,int b )
    {
        return a + b;
    }

2、C++编写DLL在导出函数在前面加上extern "C" 使用cdecl导出就不会有修饰名，例如：

    extern "C" __declspec(dllexport) int  Add(int a,int b)
    {
        return a + b;
    }

3、如果上面的方法都不用，就只能添加def文件来避免这个问题了。


如果你调用的是别人编写的DLL出现修饰名了怎么办呢？！
---------------------------------------------------------------------
1、这样的DLL不用可能并不是坏事。
2、用请运行「aadio工具 / 探测器 / DLL查看工具 」把修饰名复制出来使用就可以了。
*/
```

Markdown 格式