LBS脚本引擎开发册

2012-6-8

目录

[1. 简介 4](#_Toc432080094)

[2. 基本语法 4](#_Toc432080095)

[2.1. 书写格式 4](#_Toc432080096)

[2.2. 数据类型 6](#_Toc432080097)

[2.3. 关键字 6](#_Toc432080098)

[2.4. 变量定义 9](#_Toc432080099)

[2.5. 函数定义 9](#_Toc432080100)

[2.6. 类定义 10](#_Toc432080101)

[2.7. 运算符优先级 11](#_Toc432080102)

[2.8. 下标使用规则 13](#_Toc432080103)

[2.9. 表达式 13](#_Toc432080104)

[3. 流程控制 14](#_Toc432080105)

[3.1. if-else语句 14](#_Toc432080106)

[3.2. while语句 15](#_Toc432080107)

[3.3. for语句 15](#_Toc432080108)

[3.4. switch语句 15](#_Toc432080109)

[3.5. break语句 16](#_Toc432080110)

[3.6. continue语句 16](#_Toc432080111)

[3.7. return语句 17](#_Toc432080112)

[4. 内置函数 17](#_Toc432080113)

[4.1. 变量对象内置函数 18](#_Toc432080114)

[4.1.1. length 18](#_Toc432080115)

[4.1.2. bytes 19](#_Toc432080116)

[4.1.3. reSize 19](#_Toc432080117)

[4.1.4. getByte 19](#_Toc432080118)

[4.1.5. toString 20](#_Toc432080119)

[4.1.6. toHexString 20](#_Toc432080120)

[4.1.7. toInteger 20](#_Toc432080121)

[4.1.8. toFloat 21](#_Toc432080122)

[4.1.9. toBinary 21](#_Toc432080123)

[4.1.10. toLowerCase 21](#_Toc432080124)

[4.1.11. toUpperCase 21](#_Toc432080125)

[4.1.12. trimSpace 22](#_Toc432080126)

[4.1.13. substring 22](#_Toc432080127)

[4.1.14. indexOf 22](#_Toc432080128)

[4.1.15. lastIndexOf 22](#_Toc432080129)

[4.1.16. reverseBytes 23](#_Toc432080130)

[4.1.17. reverse 23](#_Toc432080131)

[4.1.18. removeAt 23](#_Toc432080132)

[4.1.19. getName 23](#_Toc432080133)

[4.1.20. getTypeName 24](#_Toc432080134)

[4.1.21. getToken 24](#_Toc432080135)

[4.2. 全局内置函数 25](#_Toc432080136)

[4.2.1. 数学函数 25](#_Toc432080137)

[4.2.2. 文件操作函数 27](#_Toc432080138)

[4.2.3. 资源管理函数 30](#_Toc432080139)

[4.2.4. 脚本系统函数 32](#_Toc432080140)

[4.2.5. 操作系统函数 33](#_Toc432080141)

[4.2.6. 配置文件函数 38](#_Toc432080142)

[4.2.7. 内存操作函数 40](#_Toc432080143)

[4.2.8. 数据结构函数 41](#_Toc432080144)

[4.2.9. 线程操作函数 43](#_Toc432080145)

[4.2.10. 其它 46](#_Toc432080146)

[5. 使用平台 46](#_Toc432080147)

[6. 相关示例 51](#_Toc432080148)

[6.1. 自定义文件操作类 51](#_Toc432080149)

[6.2. 使用自定义类 54](#_Toc432080150)

[6.3. 应用程序与脚本之间交换类结构体 54](#_Toc432080151)

[6.3.1. 应用程序向脚本传递结构体 54](#_Toc432080152)

[6.3.2. 脚本向应用程序传递类结构体 56](#_Toc432080153)

[6.4. 函数域内函数 58](#_Toc432080154)

[6.5. 字符串变量与下标的使用 58](#_Toc432080155)

# 简介

LBS脚本引擎是一款轻量级基于解释执行的脚本引擎平台。它小巧、简洁、不需编译、不产生中间代码、实时解释执行，并可以与应用程序进行交互、扩展。

它有以下几大特点：

**书写简洁优美**：LBS脚本语言看起来象是C++与JS的结合体，它有C++一样简洁优美的书写格式，又具有JS一样简单明了的定义与调用方式。

**与应用程序交互**：在应用程序中可以调用脚本中的函数，并能通过函数参数或者返回值来与脚本交换数据。

**自行扩展脚本功能**：在应用程序中可以自行将一个函数定义为一个导出的脚本函数，这样脚本中就可以使用该函数的功能。

**支持类与对象编程：**LBS脚本支持类的定义和对象的操作，能很好的实现复杂脚本的功能封装，并能提高脚本应用系统的可继承性。

**使用简单**：引擎平台采用DLL导出类库的形式，非常适合C++程序员在工程中使用它。

下面将从介绍LBS脚本语言开始，一步步了解LBS脚本引擎平台。

# 基本语法

## 书写格式

LBS脚本语言书写格式和C++近似，以空格符、制表符为关键字分割符，以分号(;)为语句结束符。

变量名，函数名、类名等命名支持中英文混合使用。

所有命名都区分大小写。

示例：

#include "./lib/File.h"

#define DOUBLE\_PI 3.141592654

/////////////////////

//文件类测试函数

function testFile()

{

//var files=listDirectory(getScriptRootDirectory());

//for(var i=0;i<files.length();i++)

//{

// MessageBox(files[i]);

//}

var file =new CFile;

if(!file.Open(getScriptRootDirectory() + "\data.txt"))

{

MessageBox("打开文件失败!");

return;

}

var str;

var len=0;

while( (len=file.ReadString(str)) > 0 )

{

switch(str)

{

case "123":

{

MessageBox("长度:"+len.toString()+LF+"内容:"+str+LF+"十进制");

}

case "456":

{

MessageBox("长度:"+len.toString()+LF+"内容:"+str+LF+"十进制");

}

case "789":

{

MessageBox("长度:"+len.toString()+LF+"内容:"+str+LF+"十进制");

}

case "abc":

{

MessageBox("长度:"+len.toString()+LF+"内容:"+str+LF+"十六进制");

}

case "def":

{

MessageBox("长度:"+len.toString()+LF+"内容:"+str+LF+"十六进制");

}

default:

{

execute('MessageBox("unknown string" + LF + str);');

}

}

}

var pos =file.GetPosition();

MessageBox("当前文件指针位置："+pos.toString());

file.Close();

}

//执行函数

testFile();

function 求圆周长(radius)

{

return (2.0\*DOUBLE\_PI\*radius);

}

function 求圆面积(radius,area)

{

area =DOUBLE\_PI\*radius\*radius;

MessageBox("圆面积="+area.toString());

return area;

}

## 数据类型

在LBS脚本中基本的数据类型有 **整数、浮点数、字符串、二进制数据块**。

**整数**：十进制 或者 十六进制书写。如：10000， 0x8000。

**浮点数**：1000.123 带小数点的十进制数。

**字符串**：用引号括起来的字符序列。引号可以选择单引号和双引号。如：”123” 或者‘123’。

**二进制数据块**：用于处理文件读写等功能时会用到。

## 关键字

LBS脚本语言关键字见下表：

|  |  |  |
| --- | --- | --- |
| 关键字 | 说明 | 示例 |
| var | 用于定义变量的关键字 | var a,b=1,c=”abc”; |
| #include | 文件包含关键字 | #include “./lib/file.h” |
| #define | 宏定义关键字 | #define PI 3.141592654 |
| // | 行注释关键字 | //var a=0; |
| /\*…\*/ | 段注释关键字 | /\*var a=0;\*/ |
| function | 函数定义关键字 | function strcpy(src,dst) |
| + | 加法运算操作符 | A=a+b;  B=”abc”+”12345”; |
| - | 减法运算操作符 | B=a-b; |
| \* | 乘法运算操作符 | C=a\*b; |
| / | 除法运算操作符 | D=a/b; |
| = | 赋值运算操作符 | A=b;  B=”123”; |
| > | 比较运算符 大于 | If(a>b){} |
| >= | 比较运算符 大于等于 |  |
| < | 比较运算符 小于 |  |
| <= | 比较运算符 小于等于 |  |
| == | 比较运算符 等于 |  |
| != | 比较运算符 不等于 |  |
| && | 逻辑运算符 条件与 |  |
| || | 逻辑运算符 条件或 |  |
| ! | 逻辑运算符 条件非 | a=!a; |
| & | 位运算符 与 |  |
| | | 位运算符 或 |  |
| ~ | 位运算符 非 |  |
| << | 位运算符 左移 |  |
| >> | 位运算符 右移 |  |
| ^ | 位运算符 异或 |  |
| % | 模除运算符 |  |
| new | 类新实例运算符 | A=new Array; |
| [] | 取下标运算符 | a[0]=b[0]; |
| “ | 双引号 | “aaaa” |
| ‘ | 单引号 | ‘aaaa’  **单/双引号可以嵌套使用。**  **如: var a=’ abc=”123” ’** |
| , | 分割符 | var a,b,c;  function tst(a,b,c) |
| null | 常量 空 | if(a==null) |
| false | 常量 假 | if(a==false) |
| true | 常量 真 | if(a!=true) |
| if | 条件语句 如果 | If(a>0){…} |
| else | 条件语句 或者 | If(a>0){…}  else{…} |
| for | for循环语句 | for(a=0;a<10;a++){…} |
| switch | 条件选择语句 | switch(a){…} |
| case | 条件选择子句 | switch(a)  {  case 1:{}  case 2:{}  } |
| default | 条件选择默认子句 |  |
| break | 循环中断 |  |
| return | 函数返回 | return (a+b+c\*d);  return 0; |
| continue | 循环继续 |  |
| CR | 回车符 | A=”123”+CR; |
| LF | 换行符 | A=”123”+LF; |
| TAB | 制表符 | A=”123”+TAB+”abc”+LF; |
| ( | 参数列表开始、  子句开始、优先级 | A=(a+b)\*c;  MessageBox(“aaa”); |
| ) | 参数列表结束、  子句结束 |  |
| . | 子域 | Var a=”234”;  a.toInteger(); |
| { | 函数体、代码块开始 |  |
| } | 函数体、代码块结束 |  |
| ; | 语句结束 |  |
| class | 类定义关键字 | class CFile{ var m\_hFile;} |

## 变量定义

变量定义关键字: **var** 。

示例：

var a;表示定义一个变量名为a，类型未初始化。

var a=10; 则表示 定义一个变量a并初始化值为10，类型为整型。

var a=10,b=10.5,c=”abc”;定义三个变量分别为a,b,c，并进行了初始化。其中a为整型，b为浮点数，c为字符串。

变量名命名规则：

**支持中英文混合。**

**区分大小写。**

**名字当前域唯一。**

## 函数定义

函数定义关键字：**function**。

示例1：

function test(radius)

{

var area =3.141592654\*radius\*radius;

}

定义了一个函数，名为：test 只有一个形参，没有返回值.

示例2：

function 求圆面积(radius)

{

var area =3.141592654\*radius\*radius;

return area;

}

定义了一个函数，名为“求圆面积”，输入参数为圆半径，返回圆的面积。

示例3：

function 求圆面积和周长(radius,area,length)

{

area =3.141592654\*radius\*radius;

length=2.0\*3.141592654\*radius;

return true;

}

定义了一个函数，名为“求圆面积和周长“。有三个参数：第一个参数为输入参数，圆半径；第二个参数为输出参数，输出圆面积；第三个参数也是输出参数，输出圆周长。

示例4：

下面是一个递归函数调用示例。

function 递归测试(i)

{

if(i<0)

return;

trace( i.toString() );

**i--;**

递归测试(i);

}

//调用函数

递归测试(5);

//输出结果为：

[LBS]: 5[LBS]: 4[LBS]: 3[LBS]: 2[LBS]: 1[LBS]: 0

## 类定义

类定义关键字 **class**

示例：

定义了一个名为CFile的类。

class **CFile**

{

var m\_hFile=null; //类的成员变量，可以在这里初始化，也可在构造函数里初始化。

function CFile()//类的构造函数，可以不定义

{

m\_hFile =null;

}

function Open(filename)//类的成员函数

{

return true;

}

function Close()//类成员函数

{

return true;

}

}

**类实例化：**

var file =new **CFile**;

**访问类的成员变量：**

var hfile=file.m\_hFile;

**访问类的成员函数：**

var ret =file.Open(“c:\1.txt”);

## 运算符优先级

|  |  |
| --- | --- |
| 运算符 | 优先级 |
| () | 1 |
| [] | 2 |
| . | 2 |
| & | 3 |
| | | 3 |
| ~ | 3 |
| << | 3 |
| >> | 3 |
| ^ | 3 |
| \* | 4 |
| / | 4 |
| % | 4 |
| + | 5 |
| - | 5 |
| > | 6 |
| >= | 6 |
| < | 6 |
| <= | 6 |
| == | 6 |
| != | 6 |
| && | 7 |
| || | 7 |
| ! | 7 |
| , | 8 |
| = | 9 |

**运算时按优先级从小到大运算，同级运算从左到右。**

## 下标使用规则

在LBS脚本语言中，所有的变量都可以使用下标来引用指定索引的值，还可以用下标来扩展变量的数据。下标取值与C++相同，即0~N。

示例：

var a=1;

var b=2;

var c=a+b;

var d=a[0]+b[0];

结果： c=3,d=3;

也可以在变量初始化后扩展变量数据。

示例：

var a=1;

a[0] =2;

a[1]=3;

a[10]=10;

此时 变量a是一个有10个成员的整型数组.未赋值的初始化为0.

还可以这样书写：

a[a[1]]=4; 相当于 a[3]=4;

**特殊下标：字符串的下标指向字串中的字符。**

**var a=”12345”;**

**其中：a[2] == “3”;**

## 表达式

**表达式根据运算符优先级由低到高运算，同级的从左到右运算。**

示例：a= 2+2\*3-2;

结果：a=6;

**可以通过()来改变运算优先级。**

示例：a=(2+2)\*(3-2);

结果：a=4;

**字串也可以进行加法运算**

示例：a=”123”+”abc”;

结果：a=”123abc”;

**常量也可以引用变量对象内置函数**

trace( (2).toHexString() );

**表达式也可以引用变量对象内置函数**

trace( (20\*10.0+0.5).toString().toInteger().toString() );

**函数返回值也可以引用变对象内置量函数**

trace( cos(20\*10.0+0.5).toString() );

来看看下面这个复杂的表达式

**trace( "cos="+cos(20\*10.0+0.5).toString()+**

**":len="+cos(20\*10.0+0.5).toString().length().toString() );**

# 流程控制

## if-else语句

**if-else语句判断条件为真时，执行if域中的语句，为假时执行else域中的语句。**

**示例1：**

If(**a>0**)

B=true;//单句可以不要{}

else

B=false;

**示例2：**

If(**a>0**)

{

B=1;

C=2;

}

else

{

B=0;

C=1;

}

## while语句

**while循环语句，当条件为真是将一直重复执行while域中的语句。**

**示例1：**

var a=0;

while(a<10)

a=a+1;

**示例2：**

Var a=0;

While(a<10)

{

a=a+1;

MessageBox(a.toString());

}

## for语句

**for循环语句，循环直到条件不满足。**

**for(s1;s2;s3)**

**其中 s1执行1次，用于初始化。**

**s2为判断依据，**

**s3为附加语句，每次循环结果后执行一次。**

**示例：**

for(i=0;i<10;i++)

{

MessageBox(i.toString());

}

## switch语句

**条件选择语句。不必像C一样在CASE语句后加上break;**

**示例：**

var a=2;

switch(a)

{

case 1:

{

MessageBox(“1”);

}

case 2:

{

MessageBox(“2”);

}

case 3:

{

MessageBox(“3”);

}

default:

{

MessageBox(“default”);

}

}

## break语句

**用于跳出循环的语句。**

**示例：**

for(i=0;i<10;i++)

{

If(i>2)

break;

}

While(i<10)

{

If(i>2)

break;

i=i+1;

}

## continue语句

示例：

var a=0;

for(i=0;i<10;i++)

{

If(i=2)

continue;

a=a+1;

}

a=0;

While(i<10)

{

i=i+1;

If(i=2)

continue;

a=a+1;

}

## return语句

从函数中返回，可带返回值。

function test()

{

return 1;

}

function test2()

{

for(var i=0;i<10;i++)

{

if(i>2)

return;

}

}

function test3()

{

var a=1;

return a+10;

}

# 内置函数

LBS引擎平台仅提供常用的内置函数。

在应用中如果内置函数不能满足全部需求，用户还可以在应用程序中扩展脚本的内置函数。

**扩展脚本的内置函数示例：**

函数定义格式：

CLBS\_ObjBase\* **script\_MessageBox**(CLBS\_ObjListBase& IN theParaList,CLBS\_ObjBase\* IN pThis,DWORD& OUT errorCode)

{

errorCode =LBS\_AEC\_SUCESS;

if(theParaList.GetCount() != 1)

{//参数数量不正确

errorCode =LBS\_AEC\_PARA\_COUNT\_ERROR;

return NULL;

}

CLBS\_Variable\* pPara =(CLBS\_Variable\*)theParaList.GetObj(0);

if(pPara != NULL )

{

if(pPara->GetVarType() != LVT\_STRING)

{//参数类型不正确

errorCode =LBS\_AEC\_PARA\_TYPE\_ERROR;

return NULL;

}

MessageBox(AfxGetMainWnd()->m\_hWnd,(LPCSTR)\*pPara,"来自：LB\_Script",MB\_ICONINFORMATION);

}

return NULL;

}

添加到脚本系统:

CLBS\_ScriptDriver::GetInstance().AddSystemFunction("MessageBox",**script\_MessageBox**);

在脚本中使用：

var a=100;

MessageBox(a.toString());

## 变量对象内置函数

### length

* 功能说明：获取变量成员数量，并返回一个整型的数。
* 函数原型：LONG length();
* 适用范围：所有类型变量
* 示例：

var a=1;

var len =a.length();

此时len ==1;

a[1]=2;

len =a.length();

此时 len ==2;

### bytes

* 功能说明：获取变量在存储空间中占用字节数，并返回一个整型的数。
* 函数原型：LONG bytes();
* 适用范围：所有类型变量
* 示例：

var a=1,b=”123”,c=1.0;

a.bytes() == 4;

b.bytes() == 3;

c.bytes() == 8;

### reSize

* 功能说明：调整字串或者二进制类型变量所用的存储空间。
* 函数原型：reSize(int size);
* 适用范围：字串或者二进制类型变量
* 示例：

var a=”1234”;

a.reSize(3);

则：a=”123”;

操作同：a[3]=0;

### getByte

* 功能说明：返回指向变量所在的内存数据。
* 函数原型：

BYTE\* getByte();

BYTE\* getByte(index);

BYTE\* getByte(index,size);

* 适用范围：所有类型变量
* 示例：

var a=0x1234;

trace(a.toString() + CR +LF);

var b=a.getByte();

trace( b[0].toString() + ”,” + b[1].toString() + ”:” + b[2].toString() +CR+LF);

### toString

* 功能说明：根据变量类型，返回该变量对应的字符串类型的新变量。
* 函数原型：char\* toString(int precision);
* 适用范围：整型，浮点，字串
* 示例：

var a=123.321;

var str=a.toString();

则：str==”123.321”;

str=a.toString(1);

则：str==”123.3”;

var b=10;

str = b.toString(3);

则：str==”010”;

### toHexString

* 功能说明：根据变量类型，返回该变量对应的十六进制字符串类型的新变量。
* 函数原型：char\* toHexString();
* 适用范围：整型，浮点，字串
* 示例：

var a=100;

var str=a.toHexString();

str == “0x64”;

### toInteger

* 功能说明：根据变量类型，返回该变量十进制整数类型的新变量。
* 函数原型：LONG toInteger();
* 适用范围：整型，浮点，字串，二进制数据
* 示例：

var str=”100”;

var a=str.toInteger();

a==100;

### toFloat

* 功能说明：根据变量类型，返回该变量十进制浮点数类型的新变量。
* 函数原型：double toFloat();
* 适用范围：整型，浮点，字串，二进制数据
* 示例：

var str =”3.1415926”;

var a=str.toFloat();

a==3.1415926;

### toBinary

* 功能说明：将变量转换为二进制数据，并返回转换后的变量。
* 函数原型：void\* toBinary();
* 适用范围：所有类型。
* 示例：

var str =”123”;

var a=str.toBinary()[0].toInteger();

此时：

a==49==0x31;

### toLowerCase

* 功能说明：得到字串变量转换为小写字母的新变量。
* 函数原型：char\* toLowerCase();
* 适用范围：字串
* 示例：

var str1 =”ABC”;

var str2=str1.toLowerCase();

str2==”abc”;

### toUpperCase

* 功能说明：得到字串变量转换为大写字母的新变量。
* 函数原型：char\* toUpperCase();
* 适用范围：字串
* 示例：

var str=”abc”;

var str2=str.toUpperCase();

str2==”ABC”;

### trimSpace

* 功能说明：得到字串变量去掉首尾空格的新变量。
* 函数原型：char\* trimSpace();
* 适用范围：字串
* 示例：

var str=” a bc “;

var s=str.trimSpace();

s==”a bc”;

### substring

* 功能说明：得到字串变量指定起始位置和长度的子串组成的新的变量。
* 函数原型：char\* subString(int iStart,int iLen);
* 适用范围：字串
* 示例：

var s=”0123456789”;

var s1=s.subString(1,4);

s1==”1234”;

### indexOf

* 功能说明：得到字串中指定子串第一次出现的位置，返回整数下标。可选参数：iStart指定搜索的起始位置，默认为从字串头部开始。
* 函数原型：char\* indexOf(char\* pSubString,int iStart=0);
* 适用范围：字串
* 示例：

var s=”01234567890123456789”;

var index =s.indexOf(“5”);

则：index == 5;

var index2=s.indexOf(“5”,6);

则index2 == 15;

### lastIndexOf

* 功能说明：得到字串中指定子串最后一次出现的位置，返回整数下标。
* 函数原型：char\* lastIndexOf(char\* pSubString);
* 适用范围：字串
* 示例：

var s=”123456123456123456”;

var index =s.lastIndexOf(“1”);

则：index == 12;

### reverseBytes

* 功能说明：将变量按字节顺序翻转，并返回翻转后的新变量。
* 函数原型：var reverseBytes();
* 适用范围：所有类型变量
* 示例：

var s=”123”;

var s2=s.reverseBytes();

则s==”321”;s2==”321”;

### reverse

* 功能说明：将变量按下标顺序翻转，并返回翻转后的新变量。
* 函数原型：var reverse();
* 适用范围：所有类型变量
* 示例：

var s=”123”;

var s2=s.reverse();

则s==”321”;s2==”321”;

var a=0;

a[1]=1;

a.reverse();

则a[0]==1,a[1]==0;

### removeAt

* 功能说明：移除指定变量下标顺的成员，返回成功失败的标志。
* 函数原型：bool removeAt(int iIndex);
* 适用范围：所有类型变量
* 示例：

var a=”1234”;

a.removeAt(2);

a==”124”;

### getName

* 功能说明：返回变量的名字字串。
* 函数原型：char\* getName();
* 适用范围：所有类型变量
* 示例：

var str;

var name=str.getName();

name==”str”;

### getTypeName

* 功能说明：返回变量的类型名。
* 函数原型：char\* getTypeName();
* 适用范围：所有类型变量

返回值取值：

|  |  |
| --- | --- |
| 类型名称 | 说明 |
| integer | 整型变量类型 |
| string | 字符串型变量类型 |
| float | 浮点数变量类型 |
| data | 二进制数据变量类型 |
| pointer | 指针类型变量 |
| class | 类的对象类型 |
| combineArray | 复合型数组对象类型 |
| unknown | 未初始化的变量类型 |

### getToken

* 功能说明：根据指定的分割符，返回字串变量被分割后的字串数组。
* 函数原型：char\*\* getToken(char\* pToken);
* 适用范围：字串
* 示例：

var s=”a1,b1,c1,d1”;

var sa=s.getToken(“,”);

则：

sa={“a1”,”b1”,”c1”,”d1”};

sa[1]==”b1”;

## 全局内置函数

### 数学函数

#### sin

* 功能说明：得到指定弧度的正弦值。
* 函数原型：double sin(double angle);
* 示例：

var a=sin(3.1415926\*0.5);

则a=1.0;

#### cos

* 功能说明：得到指定弧度的余弦值。
* 函数原型：double cos(double angle);

#### tan

* 功能说明：得到指定弧度的正切值。
* 函数原型：double tan(double angle);

#### asin

* 功能说明：得到指定弧度的反正弦值。
* 函数原型：double asin(double angle);
* 示例：

var a=asin(1.0);

则a=90.0;

#### acos

* 功能说明：得到指定弧度的反余弦值。
* 函数原型：double acos(double angle);

#### atan

* 功能说明：得到指定弧度的反正切值。
* 函数原型：double atan(double angle);

#### sqrt

* 功能说明：得到指定数的平方根。
* 函数原型：double sqrt(var v);

#### exp

double exp(double v);

#### log

double log(double v);

#### log10

double log10(double v);

#### floor

double floor(double v);  
得到地板数。

如：

floor(2.8) == 2.0;

floor(-2.8)==-3.0;

#### ceil

double ceil(double v);

得到最大数。

如：

ceil(2.8) == 3.0;

ceil(-2.8)== -2.0;

#### fmod

浮点求模。

double fmod(double v,double m);

#### pow

求一个数的n次方

double pow(double,int);

如：

pow(2.0,3)==8.0;

#### rand

获取小于指定数的一个随机数。

long rand(long max);

如：

var r=rand(100);

则得到一个小于100的随机数。

#### abs

求一个数的绝对值。

long abs( long value);

double abs(long value);

### 文件操作函数

#### fopen

* 功能说明：打开一个指定文件名的文件，并返回文件句柄。
* 函数原型：HANDLE fopen(char\* pFilename);
* 示例：

var hFile =fopen(“c:\1.txt”);

if(!hFile)

{

tace(“打开文件失败!”);

}

#### fclose

* 功能说明：关闭一个打开的文件
* 函数原型： fclose(HANDLE hFile);

#### fread

* 功能说明：根据打开的文件句柄读取指定大小的数据存入缓冲区，并返回实际读出的数据长度。
* 函数原型：long fread(HANDLE hFile , void\* buf , long size);
* 参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hFile | 文件句柄变量 | IN | 利用fopen打开的文件的句柄 |
| buf | 变量 | OUT | 用于存储数据的缓冲区变量 |
| size | 整型常量/变量 | IN | 要读取的字节数 |

* 返回值：实际读取字节数，整型变量。
* 示例：

var hFile =fopen("c:\1.txt");

var buf,len;

if(hFile)

{

len = fread(hFile,buf,100);

}

#### fwrite

* 功能说明：向打开的文件中写入指定长度的数据。
* 函数原型：void fwrite(HANDLE hFile , void\* buf);
* 参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hFile | 文件句柄变量 | IN | 利用fopen打开的文件的句柄 |
| buf | 变量 | IN | 待写入数据的缓冲区变量 |

* 返回值：无返回值。

#### freadString

* 功能说明：根据指定的文件句柄，读取一行字符。
* 函数原型：void fwrite(HANDLE hFile , char\* str);
* 参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hFile | 文件句柄变量 | IN | 利用fopen打开的文件的句柄 |
| str | 变量 | OUT | 用于存储读出数据的缓冲区变量 |

* 返回值：实际读取的数据长度，整型变量。

#### fwriteString

* 功能说明：向打开的文件中写入一个字串。
* 函数原型：void fwrite(HANDLE hFile , char\* str);
* 参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hFile | 文件句柄变量 | IN | 利用fopen打开的文件的句柄 |
| str | 字串型常量/变量 | IN | 待写入的字串 |

* 返回值：无返回值。

#### fseek

* 功能说明：定位文件指针到指定位置。
* 函数原型：void fseek(HANDLE hFile ,long pos);
* 参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hFile | 文件句柄变量 | IN | 利用fopen打开的文件的句柄 |
| pos | 整型常量/变量 | IN | 要定位的位置，相对于文件起始位置的偏移量 |

* 返回值：无返回值。

#### fseekToBegin

* 功能说明：定位文件指针到文件起始位置。
* 函数原型：void fseekToBegin(HANDLE hFile );
* 返回值：无返回值。

#### fseekToEnd

* 功能说明：定位文件指针到文件结束位置。
* 函数原型：void fseekToEnd(HANDLE hFile );
* 返回值：无返回值。

#### fgetLength

* 功能说明：获取文件长度。
* 函数原型：long fgetLength(HANDLE hFile );
* 返回值：返回文件当前长度值，整型变量。

#### fsetLength

* 功能说明：设置文件长度。
* 函数原型：void fsetLength(HANDLE hFile ,long len);
* 返回值：无返回值。

#### fgetPosition

* 功能说明：获取文件指针位置。
* 函数原型：long fgetPosition(HANDLE hFile );
* 返回值：返回文件当前指针位置，整型变量。

### 资源管理函数

#### createDirectory

* 功能说明：创建目录。
* 函数原型：void createDirectory(char\* pDirectory);
* 返回值：无返回值。
* 示例：

createDirectory(“c:\1\2\3”);

将在c盘创建目录树1\2\3。

#### setCurrentDirectory

* 功能说明：设置当工作前目录。
* 函数原型：long setCurrentDirectory(char\* pDirectory);
* 返回值：返回真表示成功，返回假表示失败。

#### getCurrentDirectory

* 功能说明：获取当前工作目录。
* 函数原型：char\* getCurrentDirectory();
* 返回值：返回指向当前工作目录的字符串型变量。

#### deleteDirectory

* 功能说明：删除指定的目录，文件目录必须为空目录。
* 函数原型：long deleteDirectory(char\* pDirectory);
* 返回值：返回真表示成功，返回假表示失败。

#### deleteFile

* 功能说明：删除指定的文件。
* 函数原型：long deleteFile(char\* pFileName);
* 返回值：返回真表示成功，返回假表示失败。

#### listDirectory

* 功能说明：列表指定目录下的所有文件和目录。
* 函数原型：char\*\* listDirectory(char\* pDirectory);
* 返回值：返回一个字符串数组，包含该目录下所有文件和目录的名字。
* 示例：

var files=listDirectory(getScriptRootDirectory());

for(var i=0;i<files.length();i++)

{

trace(files[i]);

}

#### isFile

* 功能说明：判断给定的路径名字是否是文件。
* 函数原型：long isFile(char\* path);
* 返回值：返回真表示是文件。

#### isDirectory

* 功能说明：判断给定的路径名字是否是目录。
* 函数原型：long isDirectory(char\* path);
* 返回值：返回真表示是目录。

#### fileIsExist

* 功能说明：判断给定的路径名字是否存在。
* 函数原型：long fileIsExist (char\* path);
* 返回值：返回真表示存在。

### 脚本系统函数

#### getScriptRootDirectory

* 功能说明：得到当前打开的脚本根目录。
* 函数原型：char\* getScriptRootDirectory();
* 返回值：返回当前脚本根目录的字串变量。

#### getScriptLoaderAppDirectory

* 功能说明：得到当前应用程序所在目录。
* 函数原型：char\* getScriptLoaderAppDirectory();
* 返回值：返回当前应用程序目录的字串变量。

#### execute

* 功能说明：动态解析执行代码。
* 函数原型：bool execute(char\* pCodeString);
* 返回值：为真表示成功，否则失败。
* 示例：

var code;

if(a>0)

code =’trace(“a>0”);’;

else

code =’trace(“a<=0”);’;

execute(code);

execute(“trace(‘动态执行的代码！’);”);

### 操作系统函数

#### createTimer

* 功能说明：创建一个定时器。
* 函数原型：var createTimer(char\* pTimerFuncName,int interval);
* 返回值：指向这个定时器的一个变量。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pTimerFuncName | 字串型常量/变量 | IN | 用于响应定时器的函数名 |
| interval | 整型常量/变量 | IN | 定时器间隔时间，单位毫秒 |

* 示例：

//定时器响应函数

function TimerFunc()

{

trace("on timer...");

}

//创建一个定时器，每秒响应一次

var timer=createTimer("TimerFunc",1000);

#### killTimer

* 功能说明：结束一个由createTimer创建的定时器。
* 函数原型：void killTimer(var timer);

#### getDate

* 功能说明：获取当前系统日期。
* 函数原型：void getDate(long year,long month,long day);
* 返回值：无返回值。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| year | 整型常量/变量 | OUT | 当前系统时间的年份 |
| month | 整型常量/变量 | OUT | 当前系统时间的月份 |
| day | 整型常量/变量 | OUT | 当前系统时间的日 |

* 示例：

var year,month,day;

getDate(year,month,day);

trace(year.toString()+month.toString()+day.toString());

#### getTime

* 功能说明：获取当前系统时间。
* 函数原型：void getDate(long hour,long minute,long second);
* 返回值：无返回值。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mour | 整型常量/变量 | OUT | 当前系统时间的小时 |
| minute | 整型常量/变量 | OUT | 当前系统时间的分钟 |
| second | 整型常量/变量 | OUT | 当前系统时间的秒 |

#### getTickCount

* 功能说明：获取当前系统启动以来经过的毫秒数。
* 函数原型：long getTickCount();
* 返回值：返回毫秒数，整型变量。

#### getCpuTime

* 功能说明：获取当前系统CPU时钟级毫秒数。
* 函数原型：double getCpuTime();
* 返回值：返回毫秒数，浮点变量。
* 示例：

var **t0** =getCpuTime();

**delayFunction();**//用于测试执行时间的函数

var **t1** =getCpuTime();

**t1 =t1-t0;**//得到时间间隔

trace("用时:"+t1.toString()+"ms");

#### trace

* 功能说明：在当前应用程序调试器中输出调试信息。
* 函数原型：void trace(char\* str);

#### shellExecute

* 功能说明：用系统shell按选项(oper)执行输入的文件名。
* 函数原型：bool shellExecute(char\* oper,char\* filename,char\* cmdLine,char\* directory);
* 返回值：返回真表示成功。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| oper | 字串型常量/变量 | IN | 系统shell的执行选项  取值范围：  (edit , explore, find ,open , print) |
| filename | 字串型常量/变量 | IN | 要执行的文件名 |
| cmdLine | 字串型常量/变量 | IN | 命令行参数 （可选） |
| directory | 字串型常量/变量 | IN | 工作目录 (可选) |

#### sleep

* 功能说明：休眠指定毫秒数。
* 函数原型：void sleep(int ms);
* 返回值：无。
* 示例：

sleep(1000);//休眠1000毫秒(1秒)

#### executeDosCommand

* 功能说明：执行DOS命令行。
* 函数原型：bool executeDosCommand(char\* pCmdLine);
* 返回值：返回真表示成功。
* 示例：

executeDosCommand("dir c:\ >c:\1.txt");

#### openUDP

* 功能说明：打开一个UDP通讯。
* 函数原型：dword openUDP(char\* pCallbackFunc,int port,char\* pAddr=null);
* 返回值：返回UDP对象的ID。
* 示例：

function **OnRecvFunction**(**dword** id,**char**\* pFromIp,**int** fromPort, **var** data)

{

}

var id =**openUDP**(“OnRecvFunction”,6060);

或者

var id =**openUDP**(“OnRecvFunction”,8080,”127.0.0.1”);

#### sendToUDP

* 功能说明：通过UDP对象向指定的IP和端口发送数据。
* 函数原型：dword sendToUDP(char\* ip,int port,var data);
* 返回值：返回成功发送数据的长度。
* 示例：

var len =sendToUDP(“127.0.0.1”,8080,”aaa”);

#### getUDPPort

* 功能说明：获取UDP对象的端口。
* 函数原型：dword getUDPPort(dword id);
* 返回值：返回UDP对象的端口。
* 示例：

var port = getUDPPort(id);

#### closeUDP

* 功能说明：关闭一个UDP通讯。
* 函数原型：closeUDP(dword id);
* 返回值：无。
* 示例：

closeUDP(id);

#### openTCP

* 功能说明：打开一个流式套接字服务端用于监听客户端的连接。
* 函数原型：handle openTCP(char\* ip,int port,char\* onAccept,char\* onReceive,char\* onClose);
* 返回值：返回套接字句柄。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| ip | 字串型常量/变量 | IN | 监听的IP地址 |
| Port | 整型常量/变量 | IN | 监听的端口 |
| onAccept | 字串型常量/变量 | IN | 响应当有新的客户端连接成功时的回调函数名 |
| onReceive | 字串型常量/变量 | IN | 响应当有套接字收到数据时的回调函数名 |
| onClose | 字串型常量/变量 | IN | 响应当有套接字断开连接时的回调函数名 |

#### connectTCP

* 功能说明：打开一个流式套接字并连到到指定的IP和端口。
* 函数原型：handle connectTCP(char\* ip,int port,char\* onAccept,char\* onReceive,char\* onClose);
* 返回值：返回套接字句柄。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| ip | 字串型常量/变量 | IN | 服务端IP地址 |
| Port | 整型常量/变量 | IN | 服务端监听端口 |
| onAccept | 字串型常量/变量 | IN | 响应当有新的客户端连接成功时的回调函数名 |
| onReceive | 字串型常量/变量 | IN | 响应当有套接字收到数据时的回调函数名 |
| onClose | 字串型常量/变量 | IN | 响应当有套接字断开连接时的回调函数名 |

#### sendToTCP

* 功能说明：打开一个流式套接字并连到到指定的IP和端口。
* 函数原型： sendToTCP(handle hSocket,var data);
* 返回值：无返回值。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hSocket | 整型变量 | IN | 目标套接字句柄 |
| data | 常量/变量 | IN | 待发送数据 |

#### closeTCP

* 功能说明：关闭一个流式套接字。
* 函数原型： closeTCP(handle hSocket);
* 返回值：无返回值。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hSocket | 整型变量 | IN | 目标套接字句柄 |

#### getTCPIP

* 功能说明：获取一个流式套接字的IP地址。
* 函数原型： char\* getTCPIP(handle hSocket);
* 返回值：返回指定套接字的IP地址。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hSocket | 整型变量 | IN | 目标套接字句柄 |

#### getTCPPort

* 功能说明：获取一个流式套接字的端口号。
* 函数原型： int getTCPIP(handle hSocket);
* 返回值：返回指定套接字的端口号。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hSocket | 整型变量 | IN | 目标套接字句柄 |

#### getLocalHostDefaultIp

* 功能说明：获取本地主机的默认IP地址。
* 函数原型： char\* getLocalHostDefaultIp();
* 返回值：返回本地主机的默认IP地址。

#### getLocalHostIpArray

* 功能说明：获取本地主机IP地址列表数组。
* 函数原型： string\* getLocalHostIpArray();
* 返回值：返回本地主机IP地址列表字串型数组。

#### raisePrivileges

* 功能说明：本地提权函数。
* 函数原型： raisePrivileges();
* 返回值：无返回值。

### 配置文件函数

#### getIniString

* 功能说明：从ini配置文件中读取一个字串。
* 函数原型：char\* getIniString(char\*mainKey,char\*subKey,char\*default,char\*filename);
* 返回值：返回读取的字串。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mainKey | 字串型常量/变量 | IN | 主键名 |
| subKey | 字串型常量/变量 | IN | 子键名 |
| default | 字串型常量/变量 | IN | 黙认值 |
| filename | 字串型常量/变量 | IN | ini配置文件名 |

#### getIniInteger

* 功能说明：从ini配置文件中读取一个整数。
* 函数原型：long getIniInteger(char\*mainKey,char\*subKey,long default,char\*filename);
* 返回值：返回读取的整数型变量。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mainKey | 字串型常量/变量 | IN | 主键名 |
| subKey | 字串型常量/变量 | IN | 子键名 |
| default | 整型常量/变量 | IN | 黙认值 |
| filename | 字串型常量/变量 | IN | ini配置文件名 |

#### getIniFloat

* 功能说明：从ini配置文件中读取一个浮点数。
* 函数原型：float getIniFloat(char\*mainKey,char\*subKey,float default,char\*filename);
* 返回值：返回读取的浮点型变量。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mainKey | 字串型常量/变量 | IN | 主键名 |
| subKey | 字串型常量/变量 | IN | 子键名 |
| default | 浮点型常量/变量 | IN | 黙认值 |
| filename | 字串型常量/变量 | IN | ini配置文件名 |

#### setIniString

* 功能说明：向ini配置文件中写入一个字串。
* 函数原型：char\* setIniString(char\*mainKey,char\*subKey,char\*value,char\*filename);

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mainKey | 字串型常量/变量 | IN | 主键名 |
| subKey | 字串型常量/变量 | IN | 子键名 |
| value | 字串型常量/变量 | IN | 要写入的字串 |
| filename | 字串型常量/变量 | IN | ini配置文件名 |

#### setIniInteger

* 功能说明：向ini配置文件中写入一个整数。
* 函数原型：char\* setIniInteger(char\*mainKey,char\*subKey,long value,char\*filename);

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mainKey | 字串型常量/变量 | IN | 主键名 |
| subKey | 字串型常量/变量 | IN | 子键名 |
| value | 整型常量/变量 | IN | 要写入的数 |
| filename | 字串型常量/变量 | IN | ini配置文件名 |

#### setIniFloat

* 功能说明：向ini配置文件中写入一个字串。
* 函数原型：char\* setIniFloat(char\*mainKey,char\*subKey,float value,char\*filename);

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| mainKey | 字串型常量/变量 | IN | 主键名 |
| subKey | 字串型常量/变量 | IN | 子键名 |
| value | 浮点型常量/变量 | IN | 要写入的数 |
| filename | 字串型常量/变量 | IN | ini配置文件名 |

### 内存操作函数

#### memCopy

* 功能说明：将一个变量中的内存中指定字节数的数据复制到另一个变量的内存中。
* 函数原型：void memCopy(dst,src,size);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| dst | 常量/变量(整数、浮点、字串、二进制) | IN | 目变量 |
| src | 常量/变量(整数、浮点、字串、二进制) | IN | 源变量。 |
| size | 整型常量/变量 | IN | 复制的字节数 |

示例：

var dst =”12345678”;

var src =”abcdefg”;

**memCopy(dst,src,3);**

此时：src = “abc45678”;

**注：**

该操作对变量类型有要求。

**src,dst**必须为 **整型、字串、浮点、二进制**类型中的一种。

#### memCopyEx

* 功能说明：将一个变量中的内存中指定字节数的数据复制到另一个变量的内存中。
* 函数原型：void memCopy(dst,dstPos,src,srcPos,size);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| dst | 常量/变量(整数、浮点、字串、二进制) | IN | 目变量 |
| dstPos | 整型常量/变量 | IN | 目变量起始写入位置 |
| src | 常量/变量(整数、浮点、字串、二进制) | IN | 源变量。 |
| srcPos | 整型常量/变量 | IN | 源变量起始读入位置 |
| size | 整型常量/变量 | IN | 复制的字节数 |

示例：

var dst =”12345678”;

var src =”abcdefg”;

**memCopyEx(dst,1,src,2,3);**

此时：src = “1cde5678”;

**注：**

该操作对变量类型有要求。

**src,dst**必须为 **整型、字串、浮点、二进制**类型中的一种。

### 数据结构函数

#### arrayCreate

* 功能说明：创建一个复合型变量数组，用于存储变量。
* 函数原型：HANDLE arrayCreate();
* 返回值：返回创建的复合型变量数组对象。

#### arrayAdd

* 功能说明：向一个复合型变量数组尾部新增一个元素。
* 函数原型：void arrayAdd(HANDLE hArray,var value,char\* name);

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hArray | 变量 | IN | 由arrayCreate创建的数组对象 |
| value | 各类型常量/变量 | IN | 欲加入的变量。 |
| name | 字串常量/变量 | IN | 加入内容的名字，用于标识和查找。可以重名。 |

#### arrayInsert

* 功能说明：向一个复合型变量数组指定位置插入一个元素。
* 函数原型：long arrayInsert(HANDLE hArray,long index,var value,char\* name);
* 返回值：为真时成功。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hArray | 变量 | IN | 由arrayCreate创建的数组对象 |
| index | 整型常量/变量 | IN | 待插入位置的索引，必须存在。 |
| value | 各类型常量/变量 | IN | 欲插入的变量。 |
| name | 字串常量/变量 | IN | 加入内容的名字，用于标识和查找。可以重名。 |

#### arrayGetAt

* 功能说明：获取复合型变量数组指定位置的元素。
* 函数原型：var arrayGet(HANDLE hArray,long index);
* 返回值：返回指定位置的变量。

#### arraySetAt

* 功能说明：更新复合型变量数组指定位置的元素。
* 函数原型：long arraySetAt(HANDLE hArray,long index,var value,char\* name);
* 返回值：为真时成功。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hArray | 变量 | IN | 由arrayCreate创建的数组对象 |
| index | 整型常量/变量 | IN | 更新位置的索引，必须存在。 |
| value | 各类型常量/变量 | IN | 新的变量。 |
| name | 字串常量/变量 | IN | 新元素的名字，用于标识和查找。可以重名。 |

#### arrayRemoveAt

* 功能说明：移除复合型变量数组指定位置的元素。
* 函数原型：long arrayRemoveAt(HANDLE hArray,long index);
* 返回值：为真时成功。

#### arrayGetLength

* 功能说明：获取复合型变量数组中元素数量。
* 函数原型：long arrayGetLength(HANDLE hArray);
* 返回值：返回数组中元素的数量。

#### arrayClear

* 功能说明：清空复合型变量数组中元素。
* 函数原型：void arrayClear(HANDLE hArray);

#### arrayDestroy

* 功能说明：销毁复合型变量数组。销毁后将不再可用。
* 函数原型：void arrayDestroy(HANDLE hArray);

#### arrayFind

* 功能说明：根据元素名在复合型变量数组中查找指定的元素。
* 函数原型：long arrayFind(HANDLE hArray,char\* name,long index);
* 返回值：返回元素所在位置，小于0时表示未找到。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| hArray | 变量 | IN | 由arrayCreate创建的数组对象 |
| name | 字串常量/变量 | IN | 要查找的元素的名字。 |
| index | 整型常量/变量 | IN | 查找的起始位置  （可选，默认为0） |

### 线程操作函数

#### createThread

* 功能说明：创建一个线程。
* 函数原型：pointer createThread( char\* pThreadFunctionName);
* 返回值：返回创建的线程对象。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| FunctionName | 字串常量/变量 | IN | 线程函数名称 |

#### terminateThread

* 功能说明：终止一个线程。
* 函数原型：void terminateThread( pThreadObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pThreadObject | 变量 | IN | 由createThread创建的线程对象 |

#### suspendThread

* 功能说明：挂起一个线程。
* 函数原型：void suspendThread(pThreadObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pThreadObject | 变量 | IN | 由createThread创建的线程对象 |

#### resumeThread

* 功能说明：线程从挂起状态结束继续执行。
* 函数原型：void resumeThread(pThreadObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pThreadObject | 变量 | IN | 由createThread创建的线程对象 |

#### createLock

* 功能说明：创建一个同步锁对象，用于线程间进行同步。
* 函数原型：pointer createThread();
* 返回值：返回创建的同步锁对象。

#### destroyLock

* 功能说明：销毁一个同步锁对象。
* 函数原型：void destroyThread( pLockObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pLockObject | 变量 | IN | 由createLock创建的同步锁对象 |

#### lock

* 功能说明：同步锁进入锁定状态。
* 函数原型：void lock(pLockObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pLockObject | 变量 | IN | 由createLock创建的同步锁对象 |

#### unlock

* 功能说明：同步锁离开锁定状态。
* 函数原型：void unlock(pLockObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pLockObject | 变量 | IN | 由createLock创建的同步锁对象 |

#### createEvent

* 功能说明：创建一个线程间同步的事件对象。
* 函数原型：pointer createEvent();
* 返回值：返回创建的同步事件对象。

#### destroyEvent

* 功能说明：销毁一个同步事件对象。
* 函数原型：void destroyThread( pEventObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pEventObject | 变量 | IN | createEvent创建的同步锁对象 |

#### setEvent

* 功能说明：给指定的同步事件置有有信号。
* 函数原型：void setEvent( pEventObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pEventObject | 变量 | IN | createEvent创建的同步锁对象 |

#### waitEvent

* 功能说明：无限等待一个同步事件对象的信号。
* 函数原型：void waitEvent( pEventObject);
* 返回值：无。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| pEventObject | 变量 | IN | createEvent创建的同步锁对象 |

### 其它

#### strGetToken

* 功能说明：分割字串。
* 函数原型：char\*\* strGetToken(char\* str,char\*token);
* 返回值：返回分割后的字串数组。

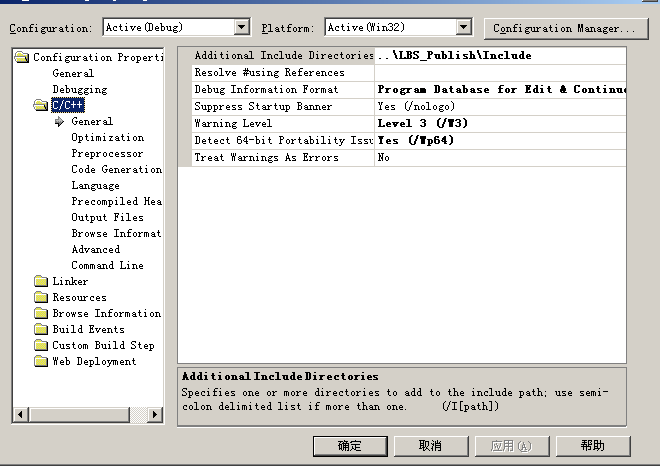
|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 类型 | 行为 | 含义 |
| str | 字串常量/变量 | IN | 要分割的字串 |
| token | 字串常量/变量 | IN | 分割符或者子串 |

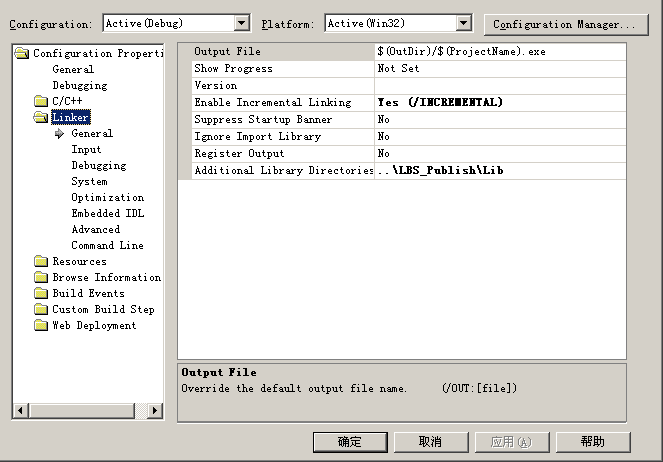
# 使用平台

**Step1:将LBS脚本引擎开发包(LBS\_Publish)复制到工程文件夹中。如下图：**

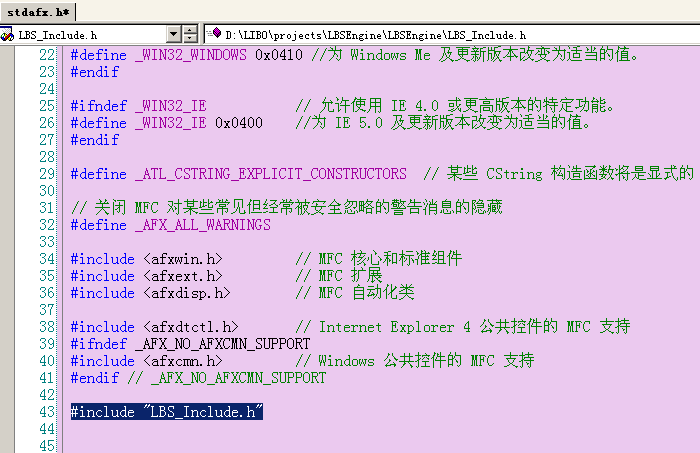


**Step2:在工程选项中设置头文件和库文件路径。如下图：**





**Step3:在工程中包含LBS引擎头文件。如下图：**

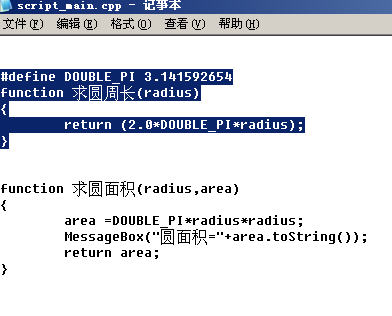
****

**Step4:初始化脚本驱动并载入脚本。如下图：**

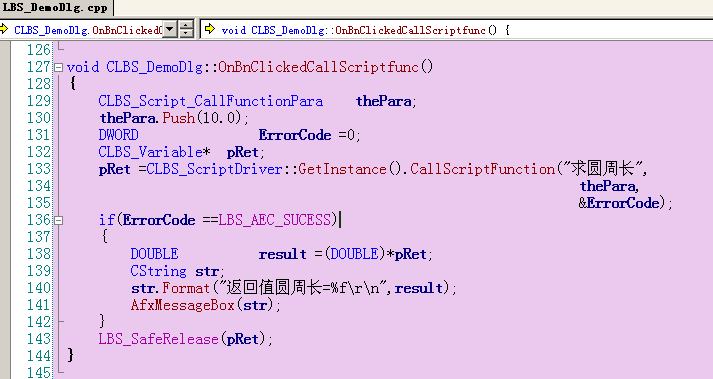


**Step5:在应用程序中调用脚本函数。**

**定义脚本函数。如下图：**



**应用程序调用脚本函数。如下图：**

****

**Step6:为脚本添加应用程序函数。**

定义函数如下：

**CLBS\_ObjBase\* script\_MessageBox(CLBS\_ObjListBase& IN theParaList,CLBS\_ObjBase\* IN pThis,DWORD& OUT errorCode)**

{

errorCode =LBS\_AEC\_SUCESS;

if(theParaList.GetCount() != 1)

{//参数数量不正确

errorCode =LBS\_AEC\_PARA\_COUNT\_ERROR;

return NULL;

}

CLBS\_Variable\* pPara =(CLBS\_Variable\*)theParaList.GetObj(0);

if(pPara != NULL )

{

if(pPara->GetVarType() != LVT\_STRING)

{//参数类型不正确

errorCode =LBS\_AEC\_PARA\_TYPE\_ERROR;

return NULL;

}

MessageBox(AfxGetMainWnd()->m\_hWnd,(LPCSTR)\*pPara,"来自：LB\_Script",MB\_ICONINFORMATION);

}

return NULL;

}

**将函数导入脚本系统：**

**CLBS\_ScriptDriver::GetInstance().AddSystemFunction("MessageBox",script\_MessageBox);**

**Step7:在脚本中使用应用程序扩展的函数。如下图：**

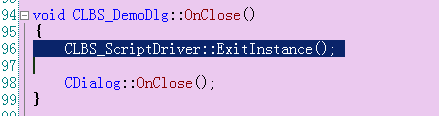
****

**Step8:执行脚本文件。**

if(!CLBS\_ScriptDriver::GetInstance().Execute())

AfxMessageBox(CLBS\_ScriptDriver::GetInstance().GetLastError());

**Step9:销毁脚本对象。如下图：**

****

**step10:编译工程并复制DLL**

工程编译后将LBS\_Publis\DLL下的所有文件复制到可执行文件目录下。

# 相关示例

## 自定义文件操作类

//自定义 文件读写操作类

class CFile

{

///////////////////////////////////////////

var m\_hFile =null; //成员变量 文件句柄

///////////////////////////////////////////

function Open(fileName)

{

m\_hFile =fopen(fileName);

if(!m\_hFile)

return false;

SeekToBegin();

return true;

}

function Close()

{

if(m\_hFile)

{

fclose(m\_hFile);

m\_hFile=null;

}

}

function Read(dataBuffer,readSize)

{

if(!m\_hFile)

{

trace("读文件失败！文件句柄无效!"+CR+LF);

return 0;

}

return fread(m\_hFile,dataBuffer,readSize);

}

function Write(dataBuffer)

{

if(!m\_hFile)

{

trace("写文件失败！文件句柄无效!"+CR+LF);

return false;

}

fwrite(m\_hFile,dataBuffer);

return true;

}

function ReadString(Str)

{

if(!m\_hFile)

{

trace("读文件失败！文件句柄无效!"+CR+LF);

return 0;

}

return freadString(m\_hFile,Str);

}

function WriteString(Str)

{

if(!m\_hFile)

{

trace("写文件失败！文件句柄无效!"+CR+LF);

return false;

}

fwriteString(m\_hFile,Str);

return true;

}

function Seek(Pos)

{

if(!m\_hFile)

{

trace("文件定位失败！文件句柄无效!"+CR+LF);

return false;

}

fseek(m\_hFile,Pos);

return true;

}

function SeekToBegin()

{

if(!m\_hFile)

{

trace("文件定位失败！文件句柄无效!"+CR+LF);

return false;

}

fseekToBegin(m\_hFile);

return true;

}

function SeekToEnd()

{

if(!m\_hFile)

{

trace("文件定位失败！文件句柄无效!"+CR+LF);

return false;

}

fseekToEnd(m\_hFile);

return true;

}

function GetLength()

{

if(!m\_hFile)

{

trace("获取文件长度失败！文件句柄无效!"+CR+LF);

return 0;

}

return fgetLength(m\_hFile);

}

function SetLength(len)

{

if(!m\_hFile)

{

trace("设置文件长度失败！文件句柄无效!"+CR+LF);

return false;

}

fsetLength(m\_hFile,len);

return true;

}

function GetPosition()

{

if(!m\_hFile)

{

trace("获取文件读写指针位置失败！文件句柄无效!"+CR+LF);

return 0;

}

return fgetPosition(m\_hFile);

}

};

## 使用自定义类

#include "./lib/File.h" //包含类所在文件

//文件类测试函数

function testFile()

{

var file =new CFile;

if(!file.Open(getScriptRootDirectory() + "\data.txt"))

{

MessageBox("打开文件失败!");

return;

}

var str;

var len=0;

while( (len=file.ReadString(str)) > 0 )

{

MessageBox("长度:"+len.toString()+LF+"内容:"+str+LF);

}

MessageBox(file.GetPosition().toString());

file.Close();

}

**testFile();** //执行函数

## 应用程序与脚本之间交换类结构体

**注意：**

1. **交互的结构体必须是1字节对齐的。**
2. **脚本中的浮点数为DOUBLE型 (8字节)**
3. **脚本中整数均为32位整数**
4. **交换的结构体内不支持指针和对象数据类型**

### 应用程序向脚本传递结构体

**应用程序中定义如下结构体：**

**#pragma pack(push,1)**

struct **trans**

{

LONG a;

DOUBLE b;

CHAR c[32];

LONG d;

BYTE e;

};

**#pragma pack(pop)**

**用结构体定义变量，并将其赋值后传递给脚本函数。**

CLBS\_Script\_CallFunctionPara thePara;

trans data;

data.a=1;

data.b=2;

strcpy(data.c,"cccccCCCCC");

data.d=3;

data.e=4;

thePara.Push((VOID\*)&data,sizeof(data));

DWORD ErrorCode =0;

CLBS\_Variable\* pRet =NULL;

pRet =CLBS\_ScriptDriver::GetInstance().CallScriptFunction("结构传递",

thePara,

&ErrorCode);

LBS\_SafeRelease(pRet);

**在脚本中定义与应用程序相同的类。**

**class CTrans**

{

var a,b,c,d,e;

function CTrans()

{

a=0;

b=0.0;

c=””;

d=0;

e=””;

e.reSize(1);

c[31]=0; //在构造函数中将变量c变成长度为32字节的字串数组。此处也可以使用reSize函数。c.reSize(32);

}

};

**定义交换结构体的函数**

function 结构传递(data)

{

**var trans=new CTrans;**

**trans=data; //结构体赋值，后面可以直接访问类CTrans对应的成员了。**

MessageBox(trans.a.toString()+' |'+trans.b.toString()+ ' |' +trans.c +'| ' +trans.d.toString() +’ |’ + trans.e.getByte()[0].toInteger().toString() );

}

### 脚本向应用程序传递类结构体

**在脚本中定义如下类：**

class CStructTest

{

var a=1;

var b=2.0;

var c="1234567890";

var d;

};

**在脚本中定义一个函数，用于返回类的对象：**

function GetStruct()

{

var r=new CStructTest;

r.d =new CStructTest;

r.d.a=2;

r.d.b=3.0;

r.d.c="abcdefg";

return r;

}

**在应用程序中定义相应的结构体：**

struct testStruct

{

LONG a;

DOUBLE b;

CString c;

testStruct \*d;

};

**在应用程序中调用脚本函数：**

void CLBS\_DemoDlg::OnBnClickedCallScriptfunc()

{

CLBS\_Script\_CallFunctionPara thePara;

DWORD ErrorCode =0;

CLBS\_Variable\* pRet ;

pRet=CLBS\_ScriptDriver::GetInstance().CallScriptFunction("GetStruct",

thePara,

&ErrorCode);

if(ErrorCode ==LBS\_AEC\_SUCESS)

{

if(pRet->GetVarType() == LVT\_CLASS)

{

CLBS\_Script\_ClassBase\* pClass =(CLBS\_Script\_ClassBase\*) (LPVOID)\*pRet;

if(pClass != NULL)

{

CLBS\_Variable\* pV =(CLBS\_Variable\*)pClass->GetMemberVariable("d");

if(pV->GetVarType()== LVT\_CLASS)

{

testStruct ttt;

ttt.d=new testStruct;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("a");

ttt.a =(LONG)\*pV;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("b");

ttt.b=(DOUBLE)\*pV;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("c");

ttt.c=(LPCSTR)\*pV;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("d");

pClass =(CLBS\_Script\_ClassBase\*) (LPVOID)\*pV;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("a");

ttt.d->a =(LONG)\*pV;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("b");

ttt.d->b=(DOUBLE)\*pV;

pV =(CLBS\_Variable\*)pClass->GetMemberVariable("c");

ttt.d->c=(LPCSTR)\*pV;

ttt.d->d=NULL;

delete ttt.d;

}

}

}

}

LBS\_SafeRelease(pRet);

}

## 函数域内函数

**脚本支持函数内部定义 域内函数。**

**如下：**

**function main()**

{

var a=0;

**function test1()**

{

a++;//父域内的变量

trace("test1 function ..." + LF);

var b=0;

}

**function test2()**

{

a++;//父域内的变量

trace("test2 function ..." + LF);

}

trace("main function ..." + LF);

test1(); //调用域内函数

test2(); //调用域内函数

trace("a==" + a.toString() + LF);

}

//主函数入口

main();

## 字符串变量与下标的使用

**编写如下脚本：**

function test()

{

var str ="0123456789";

trace(str+CR+LF);

trace(str[1]+CR+LF);

str[1]="a";

trace(str+CR+LF);

str[1]="abc";

trace(str+CR+LF);

str[str.length()]="abcdefg";

trace("str="+str+" , len="+str.length().toString()+CR+LF);

}

//执行test函数

test();

**执行脚本，输出如下结果：**

//输出结果

[LBS]: 0123456789

[LBS]: 1

[LBS]: 0a23456789

[LBS]: 0abc456789

[LBS]: str=0abc456789abcdefg , len=17