# Loadrunner函数总结

**V1.0**

[Loadrunner函数总结 1](#_Toc433006917)

[版本变更 2](#_Toc433006918)

[1 基本函数 3](#_Toc433006919)

[1.1 lr\_paramarr\_random 3](#_Toc433006920)

[1.2 lr\_save\_string 3](#_Toc433006921)

[1.3 lr\_eval\_string 3](#_Toc433006922)

[1.4 lr\_advance\_param 3](#_Toc433006923)

[1.5 sprintf 3](#_Toc433006924)

[1.6 lr\_decrypt()：解密函数 4](#_Toc433006925)

[1.7 Rand()a+b 4](#_Toc433006926)

[1.1 time() 5](#_Toc433006927)

[1.2 日志相关函数 5](#_Toc433006928)

[1.2.1 lr\_error\_message 5](#_Toc433006929)

[1.2.2 其它日志函数 6](#_Toc433006930)

[2 字符串函数 6](#_Toc433006931)

[2.1 memset 6](#_Toc433006932)

[2.2 strstr 7](#_Toc433006933)

[2.3 strcat 7](#_Toc433006934)

[2.1 strnct 8](#_Toc433006935)

[2.1 strchr &strrchr 8](#_Toc433006936)

[2.2 memcpy 9](#_Toc433006937)

[2.3 strlen 9](#_Toc433006938)

[2.1 strcpy &strncpy 9](#_Toc433006939)

[2.2 strcmp 11](#_Toc433006940)

[2.1](#_Toc433006941) **[Strupr](#_Toc433006941)** [11](#_Toc433006941)

[2.1 strdup &strlwr 12](#_Toc433006942)

[3 转换函数 12](#_Toc433006943)

[3.1 int](#_Toc433006944) **[atoi](#_Toc433006944)** [12](#_Toc433006944)

**[3.2](#_Toc433006945)** [int](#_Toc433006945) **[itoa](#_Toc433006945)** [13](#_Toc433006945)

[3.3 lrs\_decimal\_to\_hex\_string 13](#_Toc433006946)

[4 动态内存分配函数 13](#_Toc433006947)

[4.1 calloc 13](#_Toc433006948)

[4.2 malloc 14](#_Toc433006949)

[4.3 free 15](#_Toc433006950)

[5 Socket常用函数 17](#_Toc433006951)

[5.1 lrs\_create\_socket 17](#_Toc433006952)

[5.2 lrs\_send 17](#_Toc433006953)

[5.3 lrs\_receive 17](#_Toc433006954)

[5.4 lrs\_set\_send\_buffer 17](#_Toc433006955)

[5.5 lrs\_get\_received\_buffer 17](#_Toc433006956)

[5.6 Socket关联函数 17](#_Toc433006957)

[5.6.1](#_Toc433006958) **[lrs\_save\_param](#_Toc433006958)** [18](#_Toc433006958)

[5.6.2](#_Toc433006959) **[lrs\_save\_param\_ex](#_Toc433006959)** [18](#_Toc433006959)

[5.6.3](#_Toc433006960) **[lrs\_save\_searched\_string](#_Toc433006960)** [19](#_Toc433006960)

[5.7 Socket超时函数 19](#_Toc433006961)

[5.7.1 lrs\_set\_recv\_timeout 19](#_Toc433006962)

[5.7.2 lrs\_set\_recv\_timeout2 20](#_Toc433006963)

[5.8 项目实例： 20](#_Toc433006964)

[6 Tuxedo常用函数 20](#_Toc433006965)

[6.1 lrt\_set\_env\_list 20](#_Toc433006966)

[6.2 lrt\_tuxputenv 20](#_Toc433006967)

[6.3 lrt\_tpinitialize 20](#_Toc433006968)

[6.4 lrt\_abort\_on\_error 20](#_Toc433006969)

[6.5 lrt\_tpalloc 21](#_Toc433006970)

[6.6 lrt\_tpcall 21](#_Toc433006971)

[6.7 lrt\_save32\_fld\_val 21](#_Toc433006972)

[6.8 lrt\_tpfree 21](#_Toc433006973)

[6.9 lrt\_tpterm 21](#_Toc433006974)

[7 流程控制 21](#_Toc433006975)

[7.1 判断字符串出现次数 21](#_Toc433006976)

[8 待补充函数 22](#_Toc433006977)

[9 参考文档 23](#_Toc433006978)

版本变更

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **要点** | **修改人** |
| 2015-10-12 | 1.0 | Loadrunner函数总结 |  |
|  |  |  |  |
|  |  |  |  |

# 基本函数

## web\_save\_timestamp\_param

web\_save\_timestamp\_param 函数(stamp=标志)

　　函数原型：int web\_save\_timestamp\_param( const char \*tmstampParam, LAST );

　　返回值： LR\_PASS(0)成功 ,LR\_FAIL (1) 失败

　　作用：**得到当前的时间戳(13位)**

　　参数解释(一个)：tmstampParam 得到时间戳后，存储到的变量名字 该变量不需要提前定义。函数执行成功后，直接只用lr\_eval\_string 取出使用。

使用例子：

web\_save\_timestamp\_param("tStamp", LAST );

lr\_output\_message("%s",lr\_eval\_string("{tStamp}"));

结果：Action.c(7): 1445241312223

## web\_save\_param\_length

web\_save\_param\_length函数

　　函数原型：int web\_save\_param\_length( const char \*Param, const char \*Base, LAST );

　　返回值：LR\_PASS (0) 成功 , LR\_FAIL(1) 失败

　　作用：**得到指定参数的长度大小。**

参数解释(两个)：Param 需要获得的参数名称，Base 参数大小的格式包括Hexadecimal(十六进制) 和 Decimal(十进制)

Action()

{

lr\_save\_string("Aqwe位个", "Param");

//loadrunner：取参数长度，可以按照10进制和16进制进行取长度

web\_save\_param\_length("Param", "Base=Decimal", LAST );

lr\_output\_message ("获取参数长度： %s",lr\_eval\_string ("{Param\_Length}"));

return 0;

}

Action.c(8): 获取参数长度： 8

## web\_reg\_save\_param/ web\_reg\_save\_param\_ex

作用：关联，从web资源response信息提取数据，并保存到param中；

语法：

int web\_reg\_save\_param(const char \*ParamName, <list of Attributes>, LAST);

web\_reg\_save\_param\_ex(

    "ParamName=newParam",

    "LB/IC=ll",

    "RB/BIN/RE=RR",

    "Ordinal=all",

    "SaveLen=-1",

    "DFEs=UrlEncoding",

    SEARCH\_FILTERS,

        "Scope=body",

LAST);

参数说明**:**

ParamName:  存放所获取动态内容的参数名

list of Attributes: 其属性，包括：

Notfound、 LB、 RB、 RelFrameID、 Search、 ORD、 SaveOffset、 Convert、 SaveLen。属性值不分大小写。

|  |  |  |
| --- | --- | --- |
| List of Attributes | 参数使用说明 | 备注 |
| Notfound | 当在返回信息中找不到要找的内容时应该怎么处理。  Notfound=error: 当在返回信息中找不到要找的内容时，发出一个错误讯息。  Notfound=warning: 当在返回信息中找不到要找的内容时，只发出警告，脚本也会继续执行下去不会中断。 | 缺省值：=error  非必填项 |
| LB( Left Boundary ) | 返回信息的左边界字串。该属性必须有，并且区分大小写。 |  |
| RB( Right Boundary ) | 返回信息的右边界字串。该属性必须有，并且区分大小写。 |  |
| RelFrameID | 相对于URL而言，欲查找的网页的Frame。此属性可以是All或是数字，该属性可有可无。 |  |
| Search | 返回信息的查找范围。可以是Headers，Body，Noresource，All(缺省) | All(缺省)  非必填项 |
| ORD/Ordinal | 说明第几次出现的左边界子串的匹配项才是需要的内容。如为All，则将所有找到的内容储存起来。 | 缺省值是1  非必填项 |
| SaveOffset | 当找到匹配项后，从第几个字元开始存储到参数中。该属性不能为负数。 | 缺省值为0 |
| SaveLen | 当找到匹配项后，偏移量之后的几个字元存储到参数中。缺省值是-1，表示一直到结尾的整个字串都存入参数。 |  |
| Convert | 可取的值有以下两种:  HTML\_TO\_URL : 将 HTML-encoded 资料转成 URL-encoded 资料格式  HTML\_TO\_TEXT : 将 HTML-encoded 资料转成纯文字资料格式 |  |

例如：

web\_reg\_save\_param("yanzhengma", //变量名

"LB=f7f7f7\">", //左边界

"RB=<", //右边界

"Ord=1", //从第几次出现的位置取值

"RelFrameId=1", //从哪个frame中取值

"Search=Body", //窗口中的头还是body中获取内容

LAST);

**w**eb\_reg\_save\_param\_ex(

    "ParamName= yanzhengma ",

    "LB/IC= f7f7f7\">",

    "RB/BIN/RE=<",

    "Ordinal=all",

    "SaveLen=-1",

    "DFEs=UrlEncoding",

    SEARCH\_FILTERS, //查找过虑

        "Scope=body",

LAST);

以上的函数追加是从response中找到需要替换的内容，以上的内容是从以下的response中获取的：

<td width="30%" bgcolor=#EFF1F3 class=tdc align="right">输入验证码：</td>

<td width="70%" bgcolor=#EFF1F3 class=tdc>

&nbsp;<input type="text" name="cpbcode" size="20" class="bdtj3"> 请在左边框输入:<B><span style="background-color: #**f7f7f7">7790<**/span></b></td>

</tr>

可以将得到的内容存入日志用于检查

lr\_log\_message("getvalue : %s",lr\_eval\_string ("{yanzhengma}"));

if ( lr\_eval\_string ("{yanzhengma}") == "正常"){

//------------------

}else{

//-----------------

}

这样可以将不同的用户导航到不同的页面上，而且也可以对订单的状态进行判断，然后根据订单的状态流转到不同的页面上。

1. 经验总结  
   1)为了便于脚本的调试，将返回的数据都写入日志是个好办法；  
   2)为了验证取得的数据是否是自己期望的，可以将取得的数据写入日志中进行验证，  
   例：lr\_log\_message("getvalue : %s",lr\_eval\_string ("{oldstate}"));  
   3)因为它是一个注册函数，必须在返回信息前使用，所以注册的位置必须正确，否则很可能得到类似如下错误：  
   4)vuser\_init.c(734): Error -27190: No match found for the requested parameter "oldstate".   
   Check whether the requested boundaries exist in the response data. Also,   
   if the data you want to save exceeds 1024 bytes,   
   use web\_set\_max\_html\_param\_len to increase the parameter size [MsgId: MERR-27190]  
   5)vuser\_init.c(734): Error -27187: The above "not found"   
   error(s) may be explained by header and body byte counts being 0 and 0,   
   respectively. [MsgId: MERR-27187]  
   6)vuser\_init.c(734):   
   web\_concurrent\_end highest severity level was "ERROR" [MsgId: MMSG-27181]  
   7)所以使用手工方法，右键页面确定在代码中哪个位置之前注册函数至关重要  
   8)如果脚本中中文为乱码，可能是因为源文件的字符集和操作系统字符集不匹配。需要修改字符集或者设置录制脚本的编码方式。

## lr\_paramarr\_random

lr\_paramarr\_random实例(关联参数随机取值) ,和web\_reg\_save\_param一直使用

char \* FlightVal;

web\_reg\_save\_param("outFlightVal",

    "LB=outboundFlight value=", "RB=>",

    "ORD=ALL",

    "SaveLen=18",

    LAST );

web\_submit\_form("reservations.pl",

    "Snapshot=t4.inf",

    ITEMDATA,

    "Name=depart", "Value=London", ENDITEM,

    "Name=departDate", "Value=11/20/2003", ENDITEM,

    "Name=arrive", "Value=New York", ENDITEM,

    "Name=returnDate", "Value=11/21/2003", ENDITEM,

    "Name=numPassengers", "Value=1", ENDITEM,

    "Name=roundtrip", "Value=<OFF>", ENDITEM,

    "Name=seatPref", "Value=None", ENDITEM,

    "Name=seatType", "Value=Coach", ENDITEM,

    "Name=findFlights.x", "Value=83", ENDITEM,

    "Name=findFlights.y", "Value=16", ENDITEM,

    LAST );

/\*

The result of the web\_reg\_save\_param having been called before the web\_submit\_form is:

Notify: Saving Parameter "outFlightVal\_1 = 230;378;11/20/2003"

Notify: Saving Parameter "outFlightVal\_2 = 231;337;11/20/2003"

Notify: Saving Parameter "outFlightVal\_3 = 232;357;11/20/2003"

Notify: Saving Parameter "outFlightVal\_4 = 233;309;11/20/2003"

Notify: Saving Parameter "outFlightVal\_count = 4"

\*/

FlightVal = **lr\_paramarr\_random**("outFlightVal");

## lr\_convert\_string\_encoding（from, LR\_ENC\_SYSTEM\_LOCALE, LR\_ENC\_UTF8, "strUTF8"）

//将字符串编码转为UTF-8

lr\_convert\_string\_encoding(lr\_eval\_string("{keyword}"), LR\_ENC\_SYSTEM\_LOCALE, LR\_ENC\_UTF8, "strUTF8");

//去掉C数组后面的null终止符

lr\_save\_string(lr\_eval\_string("{strUTF8}"), "keyword");

//将UTF-8字符串URL encode

web\_convert\_param("keyword", "SourceEncoding=PLAIN", "TargetEncoding=URL", LAST);

## lr\_save\_string

1. **lr\_save\_string("字符串变量","参数名")** //将字符变量里的值传递给指定参数。通过该函数来改变DataFile类型参数的数值

例子：lr\_save\_string( WeekID,"GetWeekID" ); lr\_save\_string("777", "emp\_id")--》 将777保存到emp\_id这个参数中。（1和2配套使用）

## lr\_eval\_string

1. **lr\_eval\_string("{参数名}")**  //取得参数的数值。可将参数转为变量，返回参数的内容

例子：lr\_eval\_string("The row count is: {row\_cnt}"));

lr\_output\_message ("weeklistID: %s",lr\_eval\_string ("{weeklistID\_1}"));

## lr\_advance\_param

9、lr\_advance\_param //取下个参数值 （Advances to the next available parameter value）

例子：if(strlen(lr\_eval\_string("<Cardno6>"))!=19)

{

lr\_output\_message("LR未知错误，卡号:%s",lr\_eval\_string("<Cardno6>"));

lr\_advance\_param("Cardno6");

}

## sprintf

int sprintf(char \* string , const char\*format\_string[,args]);

 格式化数字字符串

sprintf 能够将整数打印到字符串中（ 类似itoa）。

如：

 把整数123 打印成一个字符串保存在s 中。

sprintf(s, " %d ", 123); //产生" 123 "

 可以指定宽度，不足的左边补空格：

sprintf(s, "%8d%8d", 123, 4567); //产生：" 123 4567"

 连接字符串

 sprintf 能够一次连接多个字符串（优于strcat ）

 例：

char s[64];

sprintf(s, "%s love %s.", "I", " ocean"); //产生： " I love ocean. "

lr\_output\_message("%s",lr\_eval\_string(s));

 例：参数名称格式化输出到变量中

sprintf(str2,"{Param\_%d}",1);

lr\_message("str2: %s",lr\_eval\_string(str2));

## lr\_decrypt()：解密函数

**Eg：**

lr\_output\_message("解密后数据为%s",lr\_decrypt("4d9d6013dcbe726b"));

**OutPut：**

解密后数据为isis

## time()

一般使用time函数，获取当前**[unix](javascript:;" \t "_self)**时间戳

   lr程序如下：

  int t1;

   char a[20];

   t1=time();//获取当前系统时间，是10位数才到S，要到毫秒呢？？

   //根据不同情况，将时间存储成不同的参数类型

   lr\_save\_int(t1,"time1");//将t1存成整形参数time1

   sprintf(a,"%d",t1);//将t1存进数组a

   lr\_save\_string(a,"time2");//将数组a存成字符串参数time2

   lr\_output\_message("当前时间是%s",lr\_eval\_string("{time1}"));

   lr\_output\_message("当前时间是%s",lr\_eval\_string("{time2}"));

   //后面的程序，可以将time1和time2代入不同的表单项中

The following example prints out the system time. It then prints the same time value in UNIX-style format using **ctime**.

Note that when accessing system functions from VuGen, don't include system header files (e.g., time.h). However, type declarations (here time\_t) are required.

typedef long time\_t;

time\_t t;

// Get UNIX-style time and display as number and string

lr\_message ("Time in seconds since 1/1/70: %ld\n", time(&t));

lr\_message ("UNIX-style time and date: %s", **ctime**(&t));

**Output**:  
Time in seconds since 1/1/70: 1019382960  
UNIX-style time and date: Sun Apr 21 11:56:00 2002

## 日志相关函数

### lr\_error\_message

lr\_error\_message //输出错误信息

例子：if(position==NULL){

lr\_end\_transaction("预制卡开卡",LR\_FAIL);

lr\_error\_message("返回报文格式错误,收到响应报文: %s\n\n, 发送的请求是:%s, ",ReturnData, buf); }

### 其它日志函数

表H-5 日志相关函数

|  |  |
| --- | --- |
| 函 数 | 功 能 描 述 |
| lr\_debug\_message | 输出一条调试信息的函数 |
| lr\_error\_message | 发送一条错误信息到输出窗口的函数 |
| lr\_get\_debug\_message | 返回当前日志设置信息的函数 |

续表

|  |  |
| --- | --- |
| 函 数 | 功 能 描 述 |
| lr\_log\_message | 发送一条消息到Vuser日志文件的函数 |
| lr\_message | 发送一条消息到Vuser日志并输出到窗口的函数 |
| lr\_output\_message | 将输出消息直接发送到output.txt文件，此文件位于Vuser脚本目录中。该函数有助于防止输出消息干扰TCP/IP通信的函数 |
| lr\_set\_debug\_message | 设置输出消息类别的函数 |
| lr\_vuser\_status\_message | 发送一条虚拟用户状态消息到Controller的函数 |

## return

return 0; 函数返回0表示成功完成本函数

return -1;执行该函数失败！

# 字符串函数

## Rand()%a+b -后面的是起始数，前面的是个数

* **Rand()：得到一个整数的随机值（0-32767）**

**Eg：**

//如果需要获取指定长度内的一个随机数，比如从X到Y之间的随机数，可以使用一下算法

Num=rand()%(Y-X+1)+X //这样就能得到包含X到Y之间的任何数（Num=rand()%7+3表示得到包含3到7+3-1（9）之间的任何数）（后面的数是起始数，前面的个数）

srand(time(NULL));

lr\_output\_message ("随即抽取0-99之间整数：%d\n",rand()%100);

lr\_output\_message ("随即抽取0-99之间整数：%d\n",rand()%10+100);// 表示得到包含100到109之间的任何数(一共10个)

**OutPut：**

随即抽取0-99之间整数：70

每运行一次函数就会随即抽取一次。

## memset

**memset**( void \**buffer*, int *c*, size\_t *n* **);**

功能：将缓冲区设置为指定字符的字节（Sets *n* bytes of a buffer to a given character）

memset()函数原型是extern void \*memset(void \*buffer, int c, int count)

buffer：为指针或是数组,

c：是赋给buffer的值,

count：是buffer的长度.

这个函数在socket中多用于清空数组.如:原型是memset(buffer, 0, sizeof(buffer))

Memset用来对一段内存空间全部设置为某个字符，一般用在对定义的字符串进行初始化为‘ ’或‘/0’；

void \***memset**( void \**buffer*, int *c*, size\_t *n* **);**

|  |  |
| --- | --- |
| *buffer* | Pointer to block of data to be filled with *c*. |
| *c* | The character placed in *buffer*. |
| *n* | The number of bytes in *buffer* set as *c*. |

The following example uses **memset** to fill a structure, about\_app, with zeros, before initializing it.

struct about\_app {  
  
     char \*version;  
     int field1;  
     int field2;  
};  
struct about\_app a; /\* Global scope \*/  
  
     a.field1 = 12;  
     a.field2 = 13;  
     **memset**(&a, 0, sizeof(struct about\_app)); /\* Zero all fields \*/  
     /\* Show that fields have been zeroed \*/  
     lr\_output\_message("After memset field1=%d field2=%d", a.field1, a.field2);  
     /\* Now initialize the structure \*/  
     a.version = "app version 1.0";  
     a.field1 = 5;  
     a.field2 = 6;  
     lr\_output\_message("Now, field1=%d field2=%d", a.field1, a.field2);

**Output**:  
Action.c(18): After memset field1=0 field2=0  
Action.c(23): Now, field1=5 field2=6

## strstr

原型：char \*strstr(const char \*str1, const char \*str2); //查找

用法：#include<string.h>

功能：找出str2字符串在str1字符串中第一次出现的位置（不包括str2的串结束符）。返回该位置的指针，如找不到，返回空指针。

## strcat

功能：链接两个字符串。

原型：char \*strcat(char \*dest,const char \*src); //追加

用法：#include<string.h>

功能：把src所指字符串添加到dest结尾处(覆盖dest结尾处的'\0')并添加'\0'。src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。

char \*strcat ( char \*to, const char \*from );   
　　例子：   
　　这个例子是用strcat链接字符串：Leeing和 @163.com.cn  
　　脚本如下：   
char test[1024], \*a = "@163.com.cn";   
strcpy(test, "Cheers\_Lee");   
strcat(test, a);   
lr\_output\_message("We can see %s",test);   
　　运行后在executon　log中看到如下语句：   
Starting action Action.

Action.c(16): We can see [Leeing@163.com.cn](mailto:Leeing@163.com.cn)

## strnct

char \*strncat ( char \*to\_string, const char \*from\_string, size\_t n );   
　　**功能：**把一个字符串连接到另一个字符串后面。   
　　**例子：**   
　　在这里，我随便写了两个字符串，用此函数把他们连接起来，并打印出来。   
　　**脚本如下：**   
char str1[]="Cheers is ";   
char str2[]="a tester.";   
lr\_output\_message("What can we see?");   
lr\_output\_message("The str1 is %s.",str1);   
strncat(str1,str2,20);   
lr\_output\_message("The str1 is %s.",str1);   
　　运行后在log中看到如下语句：   
Action.c(9): What can we see?   
Action.c(10): The str1 is Cheers is .   
Action.c(13): The str1 is Cheers is a tester..   
　　注：我们可以看到，没有连接前的str1是：Cheers is，连接后的字符串是：Zee is a tester。也可以看看strcat函数。

## strchr &strrchr

char \*strchr ( const char \*string, int c );   
　　功能：返回第一个匹配字符串中指定字符后面的字符串。

char \*strrchr ( const char \*string, int c );//常量字符串const char \*string，int c是指定字符！！

功能：返回最后一次匹配字符串中指定字符后面的字符串。    
　　例子：   
　　这个例子是返回第一个出现e字符以后所有的字符，和最后一次出现e字符以后所有的字符。   
　　脚本如下：   
char \*string = "Cheers is a tester";   
char \*first\_e, \*last\_e;   
first\_e = (char \*)strchr(string, 'e'); //返回第一个出现e字符以后所有的字符  
lr\_output\_message("We can see the first occurrence of e: %s",first\_e);   
last\_e = (char \*)strrchr(string, 'e'); //最后一次出现e字符以后所有的字符。   
lr\_output\_message("We can see the last occurrence of e: %s", last\_e);

## memcpy

原型：extern void \*memcpy(void \*dest, void \*src, unsigned int count); //复制+strlen(n)

用法：#include <string.h>

功能：由src所指内存区域复制count个字节到dest所指内存区域。

## strlen

6、原型：extern int strlen(char \*s); //打印长度

用法：#include <string.h>

功能：计算字符串s的(unsigned int型）长度

strlen //返回字符串的长度

例子：if(strlen(lr\_eval\_string("<Cardno6>"))!=19)

{

lr\_output\_message("LR未知错误，卡号:%s",lr\_eval\_string("<Cardno6>"));

lr\_advance\_param("Cardno6");

}

**Sizeof与strlen的区别：**

一、**strlen**  
    strlen(...)是函数，要在运行时才能计算。参数必须是字符型指针（char\*）。当数组名作为参数传入时，实际上数组就退化成指针了。

**参数必须是字符型指针（char\*）, 且必须是以'\0'结尾的，**不计算为'\0'的数组元素**。**  
    它的功能是：返回字符串的长度。该字符串可能是自己定义的，也可能是内存中随机的，该函数实际完成的功能是从代表该字符串的第一个地址开始遍历，直到遇到结束符NULL。返回的长度大小不包括NULL。

二、**sizeof**

sizeof(...)是运算符，在头文件中typedef为unsigned int，其值在编译时即计算好了，参数可以是数组、指针、类型、对象、函数等。

它的功能是：获得保证能容纳实现所建立的最大对象的字节大小。

由于在编译时计算，因此sizeof不能用来返回动态分配的内存空间的大小。实际上，用sizeof来返回类型以及静态分配的对象、结构或数组所占的空间，返回值跟对象、结构、数组所存储的内容没有关系。

具体而言，当参数分别如下时，sizeof返回的值表示的含义如下：

数组——编译时分配的数组空间大小；

指针——存储该指针所用的空间大小（存储该指针的地址的长度，是长整型，应该为4）；

类型——该类型所占的空间大小；

对象——对象的实际占用空间大小；

函数——函数的返回类型所占的空间大小。函数的返回类型不能是void。

## sizeof

一、sizeof的概念

　　sizeof是C语言的一种单目操作符，如C语言的其他操作符++、--等。

它并不是函数。

二、sizeof的使用方法

　 用于数据类型

　　sizeof使用形式: sizeof(type)

　　数据类型必须用括号括住: sizeof(int)

三、sizeof的主要用途

　　1、主要用途是与存储分配和I/O系统那样的例程进行通信。

例如: void　\*malloc(size\_t　size);

　　 size\_t　fread(void　\*ptr, size\_t　size, size\_t　nmemb, FILE　\*　stream);

　　2、另一个的主要用途是计算数组中元素的个数。

例如: void　\*memset(void　\*s, int　c, sizeof(s));

## strcpy &strncpy

char \*strcpy ( char \*dest, const char \*source );   
　　**功能：**复制一个字符串到另一个字符串中。 把从src地址开始且含有NULL结束符的字符串复制到以dest开始的地址空间，src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。返回指向dest的指针。  
char \***strncpy(** char \**dest*, const char \**source*, size\_t *n* **);**

|  |  |
| --- | --- |
| dest | The destination（目标） string to which *n* characters are copied. |
| source | The source string from which *n* characters are copied. |
| n | The number of characters copied. |

**功能：**复制一个字符串的前C个字符到另一个字符串中。把src所指由NULL结束的字符串的前n个字节复制到dest所指的数组中。

**例子：**   
　　复制一个字符串到字符数组中，并打印出来。   
　　**脚本如下：**   
char test[1024];   
strcpy(test, "what can we see?");   
lr\_output\_message("%s", test);   
　　运行后在executon　log中看到如下语句：   
Starting action Action.   
Action.c(10): what can we see?

**如下源函数**

strcpy

原型：extern char \*strcpy(char\* dest, const char \*src); //复制

用法：#include <string.h> 和 #include <stdio.h>

功能：把从src地址开始且含有NULL结束符的字符串复制到以dest开始的地址空间，src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。返回指向dest的指针。

strncpy

原型：extern char \*strncpy(char \*dest, char \*src, int n);

用法：#include <string.h>

功能：把src所指由NULL结束的字符串的前n个字节复制到dest所指的数组中。

说明：

如果src的前n个字节不含NULL字符，则结果不会以NULL字符结束。

如果src的长度小于n个字节，则以NULL填充dest直到复制完n个字节。

src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。

返回指向dest的指针。

举例：

// strncpy.c

#include <syslib.h>

#include <string.h>

main()

{

char \*s="Golden Global View";

char \*d="Hello, GGV Programmers";

char \*p=strdup(s);

clrscr();

textmode(0x00); // enable 6 lines mode

strncpy(d,s,strlen(s));

printf("%s\n",d);

strncpy(p,s,strlen(d));

printf("%s",p);

getchar();

return 0;

}

相关函数：memccpy,memcpy,stpcpy,strcpy

## Strcmp-/stricmp/strncmp--比较两个字符串是否相等

int result;

char tmp[20];

char string1[] = "The quick brown dog jumps over the lazy fox";

char string2[] = "The quick brown dog jumps over one lazy fox";

result = strcmp( string1, string2); // Case-sensitive comparison

if(result > 0)

strcpy(tmp, "greater than");

else if(result < 0)

strcpy(tmp, "less than");

else

strcpy(tmp, "equal to");

lr\_output\_message ("strcmp: String 1 is %s string 2", tmp);

**result = stricmp(string1, string2 ); // Case-insensitive comparison**

if( result > 0 )

strcpy( tmp, "greater than" );

else if( result < 0 )

strcpy( tmp, "less than" );

else

strcpy( tmp, "equal to" );

lr\_output\_message( "stricmp: String 1 is %s string 2", tmp );

result = **strncmp**( string1, string2 , 30); // Compare 30 chars

if (result > 0 )

strcpy(tmp, "greater than");

else if (result < 0)

strcpy(tmp, "less than");

else

strcpy(tmp, "equal to");

lr\_output\_message ("strncmp: String 1 is %s string 2", tmp);

**Output**:  
Action.c(17): strcmp: String 1 is greater than string 2  
Action.c(28): stricmp: String 1 is equal to string 2

Returns a value indicating the lexicographical relation between the strings:

|  |  |
| --- | --- |
| **Return value** | **Description** |
| <0 | *string1* is less than *string2* |
| 0 | *string1* is the same as *string2* |
| >0 | *string1* is greater than *string2* |

5、strcmp //比较两个字符串是否相等

例子：if(strcmp(lr\_eval\_string("{getTaskMsg}"),"新增周任务成功！")==0)

{

lr\_end\_transaction("周任务新增",LR\_PASS);

}

else

{

lr\_end\_transaction("周任务新增",LR\_FAIL);

lr\_error\_message("新增周任务失败，未关联到服务器返回成功信息,错误提示：%s",lr\_eval\_string("getTaskMsg"));

}

## **Strupr**

**strupr**.

    int id;

    char \* groupname\_static, \* groupname;

    // Get the groupname from VuGen

    lr\_whoami (&id, &groupname\_static, NULL);

    lr\_output\_message ("groupname=%s", groupname\_static);

    // Make a copy of groupname\_static so we can change it

    groupname = (char \*)strdup(groupname\_static);

    groupname = (char \*)**strupr**(groupname);

    lr\_output\_message ("Upper case groupname=%s", groupname);

    free(groupname);

**Output**:  
Action.c(8): groupname=None  
Action.c(14): Upper case groupname=NONE

## strdup &strlwr

char \*strdup ( const char \*string );   
　　**功能：**复制一个字符串。   
　　char \*strlwr ( char \*string );   
　　**功能：**转换成小写字母。   
　　**例子：**   
　　在这个例子中，Vuser的组名被转换为小写字母。但是lr\_whoami把组名作为静态buffer返回。这样的buffer不能被操作。如果有操作需要，就复制这个静态buffer。

**脚本如下：**   
int id;   
char \*groupname\_static, \*groupname;   
lr\_whoami(&id, &groupname\_static, NULL);   
lr\_output\_message("groupname=%s", groupname\_static);   
groupname = (char \*)strdup(groupname\_static);   
groupname = (char \*)strlwr(groupname);   
lr\_output\_message("lower case groupname=%s", groupname);   
free(groupname);   
　　上述脚本用vugen保存为：CHANGE   
　　在controller中运行（设置为总是发送消息）   
　　运行后在log中看到如下语句：   
Starting action Action. [MsgId: MMSG-15919]   
Action.c(11): groupname=CHANGE [MsgId: MMSG-17999]   
Action.c(16): lower case groupname=change [MsgId: MMSG-17999]

# 转换函数

## int **atoi**

Converts a string to an integer value.

|  |  |
| --- | --- |
| string | The string to convert. |

The following example converts the initial portion of the string, s, to an integer.

int i;

char \* s = "7 dollars";

i = **atoi**(s);

lr\_output\_message ("Price $%d", i);

**Output**:  
vuser\_init.c(7): Price $7

## int **itoa**

Converts an integer to a string. Windows only.

* **itoa函数**
  + 根据给定的进制，转换一个整型数据为字符串

      int Cint = 100;

char Cstring[100], cstring[100];

itoa(Cint,Cstring,10);

itoa(Cint,cstring,16);

lr\_output\_message(" Cstring = %s, cstring = %s ",Cstring, cstring);

例子：len=strlen(str\_Body);

lr\_output\_message("%d",len);

itoa(len,temp,10); //len 是存放整型 temp是存放字符型 10是10进制。

|  |  |
| --- | --- |
| string | The string to convert. |

## lrs\_decimal\_to\_hex\_string

lrs\_decimal\_to\_hex\_string

将十进制整数转换为十六进制字符串

# 动态内存分配函数

## calloc

原型：extern void \*calloc(int num\_elems, int elem\_size);

用法：#include <alloc.h>

功能：为具有num\_elems个长度为elem\_size元素的数组分配内存

说明：如果分配成功则返回指向被分配内存的指针，否则返回空指针NULL。

当内存不再使用时，应使用free()函数将内存块释放。

举例：

// calloc.c

#include <syslib.h>

#include <alloc.h>

main()

{

char \*p;

clrscr(); // clear screen

p=(char \*)calloc(100,sizeof(char));

if(p)

printf("Memory Allocated at: %x",p);

else

printf("Not Enough Memory!\n");

free(p);

getchar();

return 0;

}

## malloc

原型：extern void \*malloc(unsigned int num\_bytes);

用法：#include <alloc.h>

功能：分配长度为num\_bytes字节的内存块

说明：如果分配成功则返回指向被分配内存的指针，否则返回空指针NULL。

当内存不再使用时，应使用free()函数将内存块释放。

举例：

// malloc.c

#include <syslib.h>

#include <alloc.h>

main()

{

char \*p;

clrscr(); // clear screen

p=(char \*)malloc(100);

if(p)

printf("Memory Allocated at: %x",p);

else

printf("Not Enough Memory!\n");

free(p);

getchar();

return 0;

}

Allocates a block of memory. //分配内存给变量

void \***malloc(** size\_t *num\_bytes* **);**

|  |  |
| --- | --- |
| *num\_bytes* | The size of the block of memory that is allocated. |

The following example uses **malloc** to allocate a buffer of length 1024.

char \* buf;

if ((buf = (char \*)**malloc**(1024 \* sizeof(char))) == NULL) {

lr\_output\_message ("Insufficient memory available");

return -1;

}

lr\_output\_message ("Memory allocated . Buffer address = %.8x", buf);

// Do something with the buffer here ...

// Now free the buffer

free(buf);

**Output**:  
Action.c(11): Memory allocated. Buffer address = 007b9d88

## free

原型：extern void free(void \*p);

用法：#include <alloc.h>

功能：释放指针p所指向的的内存空间。

说明：p所指向的内存空间必须是用calloc,malloc,realloc所分配的内存。

如果p为NULL或指向不存在的内存块则不做任何操作。

举例：

// free.c

#include <syslib.h>

#include <alloc.h>

main()

{

char \*p;

clrscr(); // clear screen

textmode(0x00);

p=(char \*)malloc(100);

if(p)

printf("Memory Allocated at: %x",p);

else

printf("Not Enough Memory!\n");

getchar();

free(p); // release memory to reuse it

p=(char \*)calloc(100,1);

if(p)

printf("Memory Reallocated at: %x",p);

else

printf("Not Enough Memory!\n");

free(p); // release memory at program end

getchar();

return 0;

}

相关函数：free,malloc,realloc

# Socket常用函数

Socket常用的strcmp/strcat/

## lrs\_create\_socket

15、lrs\_create\_socket //初始化socket

例子：lrs\_create\_socket("socket0", "TCP", "LocalHost=0", "RemoteHost=10.0.193.140:8869", LrsLastArg);

## lrs\_send

lrs\_send //发送报文

例子：lrs\_send("socket0", "buf0",LrsLastArg);

## lrs\_receive

20、lrs\_receive （Receives data from a datagram or stream socket.）

例子：lrs\_receive("socket0", "buf1",LrsLastArg);

## lrs\_set\_send\_buffer

16、lrs\_set\_send\_buffer //指定发送的buffer

例子：lrs\_set\_send\_buffer("socket0",buf,strlen(buf));//指定发送的缓冲区,并设置发送端口，发送长度

## lrs\_get\_received\_buffer

21、lrs\_get\_received\_buffer （Gets the last received buffer or a part of it.）

例子： ReturnData=lrs\_get\_received\_buffer("socket0",0,-1,"ascii");

## Socket关联函数

**综合比较：**

前两个函数都是根据**偏移量**和查找的数据的**长度**来定位所需要的数据，**所以只适合于返回内容基本固定**。只是所**需要的数据动态变化，而且长度不变的情况，第三个函数是根据左右边界来定位要查找的数据**，适用于**数据长度变化**的情况。lrs\_save\_searched\_string与**[web](javascript:;" \t "_self)**协议中的web\_reg\_save\_param函数作用基本一样，所不同的只是web\_reg\_save\_param要放在需要关联的请求前，而lrs\_save\_searched\_string是放在请求后。

### **lrs\_save\_param**

1、/\***lrs\_save\_param**将静态或接收到的缓冲区保存到参数中\*/  
lrs\_save\_param (char \*s\_desc, char \*buf\_desc, char \*param\_name, int offset, int param\_len);  
参数:  
**s\_desc:**套接字标识符  
**buf\_desc:**缓冲区标识符  
**param\_name:**存入缓存数据的参数名称  
**offset:**被保存到参数中的缓存区偏移量  
**param\_len:**要保存到参数中的字节数

**例如：**lrs\_save\_param("socket0",NULL,"Resultscode",1575,6);//把右偏移1575个字符，后的6个字符，保存到 Resultscode 中

### **lrs\_save\_param\_ex**

2、/\***lrs\_save\_param\_ex**将用户、静态或接收到的缓冲区保存到参数中\*/  
lrs\_save\_param\_ex (char \*s\_desc, char \*type, char \*buff, int offset, int length, char \*encoding, char \*param);  
参数:  
**s\_desc:**套接字标识符  
**type:**保存到参数中的缓冲区类型，有"user"(用户缓冲区)、"static"(data.ws 中的静态缓冲区)和"received"(最后接收的缓冲区数据)三种  
**buff:**和type的值有关，如果type的值是"user"则buff的值为指定的用户缓冲 区，如果type的值是"static"则buff的值为指定的静态缓冲区，如果type的值是"received"则buff参数可以设为NULL  
**offset:**缓存区偏移量  
**length:**要保存到参数中的字节数  
**encoding:**编码方式可以指定为"ascii"或"ebcdic"，如果是用户缓冲区则NULL默认为"ascii"，如果type为"static"或"received"则NULL默认为客户端编码方式  
**param:**参数名称

**例如：**lrs\_save\_param\_ex("socket0","static","buf0",0,10240,"ascii","RecvBuf"); //将会把第0个字符后的10240个字符，保存到 RecvBuf 中

### **lrs\_save\_searched\_string**

3、/\***lrs\_save\_searched\_string**将静态或接收到的缓冲区中搜索出现的字符串，将出现字符串的缓冲区部分保存到参数中\*/  
lrs\_save\_param (char \*s\_desc, char \*buf\_desc, char \*param\_name, char \*left\_boundary, char \*right\_boundary, int ordinal, int offset, int param\_len);

lrs\_save\_searched\_string("socket1", LRS\_LAST\_RECEIVED, "Parameter1", "LB/BIN=\*", "RB/BIN=#", 2, 0, -1);  
参数:  
**s\_desc:**套接字标识符  
**buf\_desc:**缓冲区标识符  
**param\_name:**保存缓冲区数据的参数名称  
**left\_boundary:**标识要搜索缓冲区部分的左边界的字符串，格式为"LB=XXX"  
**right\_boundary:**标识要搜索缓冲区部分的右边界的字符串，格式为"RB=XXX"  
**ordinal:**表示从第几次出现的左边界字符串开始搜索，如果指定了左边界则ordinal的值一定大于0，如果没有指定左边界则将ordinal设为-1  
**offset:**要开始搜索的偏移量。如果指定了左边界则此偏移量相对于左边界计算，否则就从缓冲区的开始计算偏移量  
**param\_len:**要保存到参数中的缓冲区数据字节数。适用于没有指定右边界的情况，如果指定了右边界则设param\_len为-1  
**小白版：例子**

lrs\_save\_searched\_string("socket0", NULL, "evalid", "LB=SimpleEngineTest/", "RB=\"", 1,0,-1);

这个函数的用法基本与web\_reg\_save\_param()类似，一样是通过左右边界来进行数据查找，函数的**8个参数依次代表**：

“socket线程名”、“buf名称”（这里传NULL表示从上面一个buf取值）、“参数名称”（需要将数据保存的参数名称）、

“左边界”、“右边界”、“第几次出现”、“位移的数量”、“参数的长度”。线程名和buf名一定不可以传错，最后三个数据

只要传（1 , 0，-1）就可以了。

## Socket超时函数

### lrs\_set\_recv\_timeout

18、lrs\_set\_recv\_timeout //设置获取返回值的超时时长

（Sets a timeout for receiving the expected data on a socket）

例子：lrs\_set\_recv\_timeout(120,0);

### lrs\_set\_recv\_timeout2

lrs\_set\_recv\_timeout2

（Sets a timeout for receiving data on a socket after a connection was established）

例子：lrs\_set\_recv\_timeout2(120,0);

## 缓冲区函数

lrs\_free\_buffer 释放分配给缓冲区的内存

lrs\_get\_buffer\_by\_name 从数据文件中获取缓冲区及其大小

lrs\_get\_last\_received\_buffer 获取套接字上接收到的最后的缓冲区及其大小

lrs\_get\_last\_received\_buffer\_size 获取套接字上接收到的最后一个缓冲区的大小

lrs\_get\_received\_buffer 获取最后接收到的缓冲区或其一部分

lrs\_get\_static\_buffer 获取静态缓冲区或其一部分

lrs\_get\_user\_buffer 获取套接字的用户数据的内容

lrs\_get\_user\_buffer\_size 获取套接字的用户数据的大小

lrs\_set\_send\_buffer 指定要在套接字上发送的缓冲区

# Java函数

# Tuxedo常用函数

## lrt\_set\_env\_list

26、lrt\_set\_env\_list 设置环境变量（Sets a list of environment variables）

例子：lrt\_set\_env\_list(env\_allow\_array);

## lrt\_tuxputenv

27、lrt\_tuxputenv 修改或增加环境变量值

例子：lrt\_tuxputenv("TUXDIR=C:\\bea\\tuxedo8.0"); lrt\_tuxputenv("WSNADDR=//{IP}");

## lrt\_tpinitialize

28、lrt\_tpinitialize 初始化，使客户端连接到目标服务器的TUXEDO应用上 （Enables a client to join a System/T application）

例子：tpresult\_int = lrt\_tpinitialize(LRT\_END\_OF\_PARMS);

## lrt\_abort\_on\_error

29、lrt\_abort\_on\_error 当tuxedo出现ERROR终止脚本

例子：Aborts the current transaction if the previous Tuxedo function resulted in an error.

## lrt\_tpalloc

30、char \*lrt\_tpalloc 分配一个新的buffer （Allocates a new buffer）

例子：pFml = (FBFR32 \*)lrt\_tpalloc("FML32",0,4096); 其中pFml是这样定义的：FBFR32 \*pFml;

## lrt\_tpcall

31、lrt\_tpcall 发送报文，并接收返回结果 （Sends a service request and awaits its reply）

例子：ret = lrt\_tpcall("CDMSVC",(char \*)pFml,0,(char \*\*)&pFml,&rcvlen,0); ret定义为整型，CDMSVC是TUXEDO应用用于接收报文的服务名称，

rcvlen的定义为long rcvlen;

## lrt\_save32\_fld\_val

32、lrt\_save32\_fld\_val 把an FML32 buffer中一个值保存到一个参数（Saves the current value of an FML32 buffer to a parameter. ）

例子：lrt\_save32\_fld\_val((FBFR32 \*)pFml, "id=33554464", 0,"host\_stat");

## lrt\_tpfree

33、lrt\_tpfree 释放buffer

例子：lrt\_tpfree((char \*)pFml);

## lrt\_tpterm

34、lrt\_tpterm( ); 断开客户端与TUXEDO应用的链接

例子：lrt\_tpterm();

# 流程控制

## 判断字符串出现次数

// Check result

if (atoi(lr\_eval\_string("{StatusCodeCount }")) > 0){ //判断如果Welcome字符串出现次数大于0

lr\_output\_message("Send out the comment successfully."); } //在日志中输出Send out the comment successfully

else{ //如果出现次数小于等于

lr\_error\_message("Send out the comment unsuccessfully."); //在日志中输出Send out the comment successfully

# 待补充函数

lr\_get\_master\_host\_name 返回运行 LoadRunner Controller 的计算机名  
lr\_eval\_string 用参数的当前值替换参数  
lr\_save\_string 将以 NULL 结尾的字符串保存到参数中  
lr\_save\_var 将变长字符串保存到参数中  
lr\_save\_datetime 将当前日期和时间保存到参数中  
lr \_advance\_param 前进到下一个可用参数  
lr\_debug\_message 将调试信息发送到输出窗口  
lr\_error\_message 将错误消息发送到输出窗口  
lr\_get\_debug\_message 检索当前消息类  
lr\_log\_message 将消息发送到日志文件  
lr\_output\_message 将消息发送到输出窗口  
lr\_set\_debug\_message 设置调试消息类  
lr\_message 将消息发送到 Vuser 日志和输出窗口 lr\_load\_dll 加载外部 DLL  
lr\_continue\_on\_error 指定处理错误的方法 lr\_continue\_on\_error (0）  
web\_custom\_request 允许您使用 HTTP 支持的任何方法来创建自定义 HTTP 请求  
web\_image 在定义的图像上模拟鼠标单击 web\_link 在定义的文本链接上模拟鼠标单击   
web\_submit\_data 执行"无条件"或"无上下文"的表单  
web\_submit\_form 模拟表单的提交  
web\_url 加载由"URL"属性指定的 URL  
web\_find 在 HTML 页内搜索指定的文本字符串  
web\_image\_check 验证指定的图像是否存在于 HTML页内  
web\_reg\_find 在后面的 HTTP 请求中注册对 HTML源或原始缓冲区[中文](http://action.vogate.com/c/c.php?r=http%3A//www.baidu.com/s%3Fbs%3D%25C8%25ED%25BC%25FE+%25D0%25D4%25C4%25DC%25B2%25E2%25CA%25D4%26f%3D8%26wd%3Dloadrunner+%25BA%25AF%25CA%25FD&aid=5841&sid=6235007045049672&click=1&url=http%3A//www.seawww.com/style/info/domain.asp&v=0&k=%u4E2D%u6587&s=http%3A//old.blog.edu.cn/user2/35840/archives/2008/2119208.shtml&rn=119586)本字符串的搜索   
web\_add\_cookie 添加新的 Cookie 或修改现有的 Cookie  
web\_cleanup\_cookies 删除当前由 Vuser 存储的所有 Cookie  
web\_remove\_cookie 删除指定的 Cookie

# 参考文档

网页版部分函数

<http://www.kuqin.com/clib/>

sizeof用法汇总

<http://www.cnblogs.com/chengxin1982/archive/2009/01/13/1374575.html>

c语言详解sizeof

<http://www.cnblogs.com/wangkangluo1/archive/2011/09/27/2193072.html>

LoadRunner函数学习笔记

<http://blog.itpub.net/9934490/viewspace-933440/>

lr\_paramarr\_random实例(关联参数随机取值)

<http://blog.sina.com.cn/s/blog_6090a6ba01010wmp.html>