# MyLuaForX64Dbg

对 x64Dbg 支持 lua 的开发文档

**2019 年 8 月 4 日**
**CR32**

# MyLuaForX64Dbg API 文档

## 1. api 模块

## 注: 除 Global 函数之外其它所有的函数都在 api.lua 中定义

### api.ctx 模块

Api 模块内部模块，提供对被调试进程寄存器的操作

### Fun eax

获取或者修改被调试进程的 eax 的值

函数原型:

```
function api.ctx.eax(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 eax 的值

返回值:

val 为 nil 则返回 EAX 的内容、否则返回更改 eax 是否成功还是失败 true/false

### Fun ebx

获取或者修改被调试进程的 ebx 的值

函数原型:

```
function api.ctx.ebx(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 ebx 的值

返回值:

val 为 nil 则返回 EBX 的内容、否则返回更改 ebx 是否成功还是失败 true/false

## Fun ecx

获取或者修改被调试进程的 ecx 的值

函数原型：

```
function api.ctx.ecx(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 ecx 的值

返回值:

val 为 nil 则返回 EcX 的内容、否则返回更改 ecx 是否成功还是失败 true/false

## Fun edx

获取或者修改被调试进程的 edx 的值

函数原型：

```
function api.ctx.edx(val)
```

参数:

val - 类型(numder、16 进制的字符串)、可空参数、不填则代表获取 edx 的值

返回值:

val 为 nil 则返回 EdX 的内容、否则返回更改 edx 是否成功还是失败 true/false

## Fun ebp

获取或者修改被调试进程的 ebp 的值

函数原型：

```
function api.ctx.ebp(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 ebp 的值

返回值:

val 为 nil 则返回 EBP 的内容、否则返回更改 ebp 是否成功还是失败 true/false

## Fun esp

获取或者修改被调试进程的 esp 的值

函数原型：

```
function api.ctx.esp(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 esp 的值

返回值:

val 为 nil 则返回 ESP 的内容、否则返回更改 esp 是否成功还是失败 true/false

## Fun esi

获取或者修改被调试进程的 esi 的值

函数原型：

```
function api.ctx.esi(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 esi 的值

返回值:

val 为 nil 则返回 ESI 的内容、否则返回更改 esi 是否成功还是失败 true/false

## Fun edi

获取或者修改被调试进程的 edi 的值

函数原型：

```
function api.ctx.edi(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 edi 的值

返回值:

val 为 nil 则返回 EDI 的内容、否则返回更改 edi 是否成功还是失败 true/false

## Fun eip

获取或者修改被调试进程的 eip 的值

函数原型：

```
function api.ctx.eip(val)
```

参数:

val - 类型(number、16 进制的字符串)、可空参数、不填则代表获取 eip 的值

返回值:

val 为 nil 则返回 EIP 的内容、否则返回更改 eip 是否成功还是失败 true/false

## Fun stepinto

调试器单步步入功能

函数原型：

```
function api.stepinto(count)
```

参数:

count - 类型(number、16 进制的字符串)、可空参数、默认为 1、单步重复的次数

返回值:

无

## Fun stepover

调试器单步步过功能

函数原型：

```
function api.stepover(count)
```

参数:

count - 类型(number、16 进制的字符串)、可空参数、默认为 1、单步重复的次数

返回值:

无

## Fun stepout

调试器单步跳出功能

函数原型:

```
function api.stepOut(count)
```

参数:

count - 类型(number、16 进制的字符串)、可空参数、默认为 1、单步重复的次数

返回值:

无

## Fun wait

等待调试事件到来, 阻塞到调试事件到来才会触发

函数原型:

```
function api.wait()
```

参数:

无

返回值:

无

## Fun pause

暂停调试

函数原型：

```
function api.pause()
```

参数:

无

返回值:

无

## Fun stop

停止调试

函数原型：

```
function api.stop()
```

参数:

无

返回值:

无

## Fun isrunning

是否正在运行状态

函数原型：

```
function api.isrunning()
```

参数:

无

返回值:

类型(boolean) 运行返回 true，否则 false

## Fun clearbp

清除所有断点

函数原型：

```
function api.clearbp()
```

参数:

无

返回值:

无

## Fun setbp

设置软件断点

函数原型：

```
function api.setbp(addr)
```

参数:

Addr - 类型(number, 16 进制字符串, API 名称 ) 需要下断点的位置

返回值:

类型(boolean) true/false

## Fun delbp

删除软件断点

函数原型：

```
function api.delbp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean) true/false

## Fun enablebp

启用已被禁用的软件断点

函数原型：

```
function api.enablebp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean) true/false

## Fun disablebp

禁用已被启用的软件断点

函数原型：

```
function api.enablebp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean) true/false

## Fun sethbp

设置硬件断点

函数原型：

```
function api.sethbp(addr, t, size)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

T - 类型(string) r w x 读 写 执行 默认 x

Size - 类型(number) 1 2 4 (8 只能在 64 位下) 默认 1

返回值:

类型(boolean)  true/false

## Fun delhbp

删除硬件断点

函数原型：

```
function api.delhbp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean)  true/false

## Fun enablehbp

启用硬件断点

函数原型：

```
function api.enablehbp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean)  true/false

## Fun disablehbp

禁用硬件断点

函数原型：

```
function api.disablehbp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean)  true/false

## Fun setmbp

设置内存断点

函数原型:

```
function api.setmbp(addr, time, t)
```

参数:

Addr -  类型(number, 16 进制地址, API 名称)  位置

time -  类型(boolean)  是一次性断点还是永久断点  默认永久

t -  类型(string)  a r w x  r 读  w 写  x 执行  a(rwx)

返回值:

类型(boolean)  true/false

## Fun delmbp

删除内存断点

函数原型:

```
function api.delmbp(addr)
```

参数:

Addr -  类型(number, 16 进制地址, API 名称)  位置

返回值:

类型(boolean)  true/false

## Fun enablembp

启用内存断点

函数原型:

```
function api.enablembp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean) true/false

## Fun disablembp

禁用内存断点

函数原型:

```
function api.disablembp(addr)
```

参数:

Addr - 类型(number, 16 进制地址, API 名称) 位置

返回值:

类型(boolean) true/false

## Fun go

运行被调试进程

函数原型:

```
function api.go()
```

参数:

无

返回值:

无

## Fun log

在调试器日志窗口输出调试信息

函数原型:

```
function api.log(str)
```

参数:

Str - 类型(string) 要输出的文本

返回值:

无

## Fun msgbox

弹出消息框

函数原型:

```
function api.msgbox(title, msg)
```

参数:

Title - 类型(string) 标题

Msg- 类型(string) 消息

返回值:

无

## Fun cls

清屏调试器日志窗口

函数原型:

```
function api.cls()
```

参数:

无

返回值:

无

## Fun curtid

取当前调试的线程 ID

函数原型:

```
function api.curtid()
```

参数:

无

返回值:

类型(number) 当前线程的 ID

## Fun curpid

取当前调试进程的进程 ID

函数原型:

```
function api.curpid()
```

参数:

无

返回值:

类型(number) 当前进程的 ID

## Fun peb

取当前进程的 PEB

函数原型:

```
function api.peb()
```

参数:

无

返回值:

类型(number) 当前进程的 PEB

## Fun teb

取当前线程的 TEB

函数原型:

```
function api.teb()
```

参数:

无

返回值:

类型(number) 当前线程的 TEB

## Fun cmd

执行调试器命令行

函数原型:

```
function api.cmd(strCmd)
```

参数:

strCmd - 类型(string) 命令

返回值:

无

## Fun isdebugging

是否调试状态

函数原型:

```
function api.isdebugging()
```

参数:

无

返回值:

类型(boolean) true/false

## Fun disasm

反汇编一条指令

函数原型：

```
function api.disasm(addr)
```

参数:

Addr- 被调试进程的地址，可以是整数或者是十六进制的文本地址

返回值:

类型(table) 返回一个对象，格式如下

```
-- {
--     "ins" : "mov eax, ecx",
--     "instr_size" : 2,
--     "args" : {
--         "count" : 2,
--         0 : "eax",
--         1 : "ecx"
--     }
-- }
```

### Fun input

文本输入框 注意 ： 慎用 此函数有 bug,与输入法冲突可能会引起程序崩溃

函数原型：

```
function api.input(label, title)
```

参数:

Label - 类型(string) 提示文本

Title - 类型(string) 标题文本

返回值:

类型(string) 返回用户输入的文本

## 2. mod 模块

### Fun mainoep

获取被调试进程主模块的 EntryPoint

函数原型：

```
function mod.mainoep()
```

参数:

无

返回值:

类型(number) 返回主模块 OEP

## Fun mainbase

获取被调试进程主模块的基地址

函数原型:

```
function mod.mainbase()
```

参数:

无

返回值:

类型(number) 返回主模块基地址

## Fun mainsize

获取被调试进程主模块的大小

函数原型:

```
function mod.mainsize()
```

参数:

无

返回值:

类型(number) 返回主模块大小

## Fun mainname

获取被调试进程主模块的名称

函数原型：

```
function mod.mainname()
```

参数:

无

返回值:

类型(number) 返回主模块名称

## Fun mainpath

获取被调试进程主模块的名称

函数原型：

```
function mod.mainname()
```

参数:

无

返回值:

类型(number) 返回主模块名称

## Fun getbasebyaddr

根据地址查找属于该模块的基址

函数原型：

```
function mod.getbasebyaddr(addr)
```

参数:

Addr - 类型(number, 16 进制文本地址) 被调试进程地址

返回值:

类型(number) 返回对应模块的基址

## Fun getbasebyname

根据模块名称查找属于该模块的基址

函数原型：

```
function mod.getbasebyname(name)
```

参数:

Name - 类型(string) 模块名称

返回值:

类型(number) 返回对应模块的基址

## Fun getsizebyaddr

根据地址查找属于该模块的大小

函数原型：

```
function mod.getsizebyaddr(addr)
```

参数:

Addr - 类型(number, 16 进制文本地址) 被调试进程地址

返回值:

类型(number) 返回对应模块的大小

## Fun getsizebyname

根据模块名称查找模块大小

函数原型：

```
function mod.getsizebyname(name)
```

参数:

Name - 类型(string) 模块名称

返回值:

类型(number) 返回对应模块的大小

## Fun getoepbyname

根据模块名称查找模块 OEP

函数原型：

```
function mod.getoepbyname(name)
```

参数:

Name - 类型(string) 模块名称

返回值:

类型(number) 返回对应模块的 OEP

## Fun getoepbyaddr

根据地址查找属于该模块的 OEP

函数原型：

```
function mod.getoepbyaddr(addr)
```

参数:

Addr - 类型(number, 16 进制文本地址) 被调试进程地址

返回值:

类型(number) 返回对应模块的 OEP

## Fun getprocaddr

查询被调试进程 WIN API 地址

函数原型：

```
function mod.getprocaddr(md, api)
```

参数:

md- 类型(string) 模块名称

Api- 类型(string) 函数名称

返回值:

类型(number) 返回对应函数的地址

## Fun getmdlist

获取被调试进程模块列表

函数原型：

```
function mod.getmdlist()
```

参数:

无

返回值:

类型(table) 返回模块列表对象，失败返回 nil

返回对象结构:

```
-- {
--  count : 2,
--  list : [
--      {
--          "base" : 401000,
--          "oep" : 401000,
--          "size" : 3000,
--          "name" : "xx.exe",
--          "path" : "C:/xx.exe",
--          "sectionCount" : 3
--      },
--      {
--          "base" : 401000,
--          "oep" : 401000,
--          "size" : 3000,
--          "name" : "xx.dll",
--          "path" : "C:/xx.dll",
--          "sectionCount" : 3
--      }
--  ]
-- }
```

例子:

```
-- 遍历模块
local tb = mod.getmdlist()
if (tb ~= nil) then
    local count = tb['count']
    local list = tb['list']
    for i = 0, count - 1 do
        printf("base = %08X, name = %s \r\n", list[i]['base'],
list[i]['name'])
    end
end
```

# 3. mem 模块

## Fun byte

以 byte 为单位读写被调试进程的内存

函数原型:

```
function mem.byte(addr, b)
```

参数:

Addr - 类型(number、16 进制的字符串) 被调试进程 32 位地址

b - 类型(number) 可空参数 不写则读 byte 值，否则写入目标进程的 byte 值，如果超过 byte 则截断、如 0x1234，取 0x34

返回值:

类型(number、boolean) b 若为 ni，则返回读取到目标进程的 byte 值，否则返回写入成功或者失败 true/false

## Fun word

以 word 为单位读写被调试进程的内存

函数原型:

```
function mem.word(addr, w)
```

参数:

Addr - 类型(number、16 进制的字符串) 被调试进程 32 位地址

w - 类型(number) 可空参数 不写则读 word 值，否则写入目标进程的 word 值，如果超过 word 则截断

返回值:

类型(number、boolean) b 若为 ni，则返回读取到目标进程的 word 值，否则返回写入成功或者失败 true/false

## Fun dwrod

以 dword 为单位读写被调试进程的内存

函数原型：

```
function mem.dword(addr, dw)
```

参数:

Addr - 类型(number、16 进制的字符串) 被调试进程 32 位地址

dw - 类型(number) 可空参数 不写则读 dword 值，否则写入目标进程的 dword 值，如果超过 dword 则截断、如 0x1234，取 0x34

返回值:

类型(number、boolean) b 若为 ni，则返回读取到目标进程的 dword 值，否则返回写入成功或者失败 true/false

## Fun getjmpaddr

此函数可获取被调试进程跳转指令的目标地址、只可用于长跳转

函数原型：

```
function mem.getjmpaddr(eip)
```

参数:

Eip - 类型(number、16 进制字符串) jxx 指令的地址

返回值:

类型(number) jxx 指令的目标地址

## Fun isvalid

判断被调试进程地址是否有效

函数原型：

```
function mem.isvaild(addr)
```

参数:

addr - 类型(number、16 进制字符串) 被调试进程的地址

返回值:

类型(boolean) true 表示有效，false 无效

## Fun read

读取调试器进程的内存

函数原型：

```
function mem.read(dstva, buf, size)
```

参数:

addr - 类型(number、16 进制字符串) 被调试进程的地址
Buf - 类型(number) 缓冲区地址，可以是 malloc 申请的空间
Size- 类型(number) 需要读取数据的尺寸

返回值:

类型(boolean) true 表示有效，false 无效, nil 表示参数错误

## Fun write

写被调试进程的内存

函数原型：

```
function mem.write(dstva, buf, size)
```

参数:

addr - 类型(number、16 进制字符串)  被调试进程的地址
Buf - 类型(number) 写入缓冲区的数据地址，可以是 malloc 申请的空间
Size- 类型(number) 需要写入数据的尺寸

返回值:
类型(boolean)  true 表示有效，false 无效, nil 表示参数错误

## Fun remotealloc

在被调试进程中申请内存

函数原型：
```
function mem.remotealloc(size)
```

参数:
Size - 类型(number)  需要申请内存的空间大小

返回值:
类型(number)  返回目标进程的内存地址，nil 表示参数错误

## Fun remotefree

释放被调试进程中申请的内存

函数原型：
```
function mem.remotefree(addr)
```

参数:
addr - 类型(number、16 进制字符串)  被调试进程的地址

返回值:
类型(boolean)  true 表示有效，false 无效, nil 表示参数错误

## Fun localmemhex

从调试器进程中获取一段内存，转化为 hex string

函数原型：

```
function mem.localmemhex(addr, size, step, spacer)
```

参数:

Addr - 类型(number, 16 进制文本地址) x64dbg 进程的地址，一般是 malloc 申请的地址
Size - 类型(number) 需要读取的大小，最大长度 255
Step- 类型(number) 每一项的长度，可以是 1, 2, 4, 默认为 1
Spacer- 类型(string) 分隔符，默认是一个空格

返回值:

类型(string) 十六进制字符串 例如: 6F 73 A4 31 32 33 34

## Fun localmemhex

从被调试器进程中获取一段内存，转化为 hex string

函数原型：

```
function mem.remotememhex(addr, size, step, spacer)
```

参数:

Addr - 类型(number, 16 进制文本地址) 被调试进程的地址
Size - 类型(number) 需要读取的大小，最大长度 255
Step- 类型(number) 每一项的长度，可以是 1, 2, 4, 默认为 1
Spacer- 类型(string) 分隔符，默认是一个空格

返回值:

类型(string) 十六进制字符串 例如: 6F 73 A4 31 32 33 34

# 4. Global 函数

注: Global 函数是由 C 直接提供的函数，不依赖任何的其它 lua 模块

## Fun printf

此函数用于在调试器日志窗口格式化输出调试信息

函数原型：

```
void printf(const char* format, ...);
```

参数:

请参照 C 语言同名函数

返回值:

无

## Fun DynamicAddFunc

为了插件的灵活性，此函数可以在运行时动态增加函数，一般用于调用 Windows Api, 详情请参见 winapi.lua 例子

参数:

szFuncName- 类型(string) 给函数取一个名称
lpFuncAddr- 类型(number) 目标函数地址

返回值:

类型(boolean) true 表示添加成功, false 表示函数名称已存在

## Fun GetEFlag

获取被调试进程标志寄存器

函数原型：

```
int GetEFLAGS();
```

参数:

无

返回值:

类型(number) 状态寄存器的值

## Fun SetEFlag

设置被调试进程标志寄存器

函数原型：

```
bool SetEFLAGS(int value);
```

参数:

Value - 新的标志值

返回值:

成功失败

## Fun Push

往当前被调试线程栈压入一个值

函数原型：

```
int Push(int value);
```

参数:

Value - 32 位 DWORD 值

返回值:

返回之前栈顶的值

## Fun Pop

从当前被调试线程栈栈顶弹出一个值

函数原型：

```
int Pop();
```

参数:

无

返回值:

栈顶的值

## Fun fopen

同 C 库函数

## Fun fread

同 C 库函数

## Fun fwrite

同 C 库函数

## Fun fputs

同 C 库函数

## Fun fseek

同 C 库函数

## Fun fflush

同 C 库函数

## Fun feof

同 C 库函数

## Fun fclose

同 C 库函数

## Fun malloc

同 C 库函数

## Fun free

同 C 库函数

## Fun memset

同 C 库函数

## Fun memcpy

同 C 库函数

## Fun memcmp

同 C 库函数

# Win32 API

## Fun Sleep

休眠一段时间

函数原型：

```
WINBASEAPI
VOID
WINAPI
Sleep(
    __in DWORD dwMilliseconds
    );
```

参数:

请参照 Windows 同名函数

返回值:

请参照 Windows 同名函数

## Fun MessageBoxA

请参照 Windows 同名函数

## Fun LoadLibrary

请参照 Windows 同名函数

## Fun GetModuleHandle

请参照 Windows 同名函数

## Fun GetProcAddress

请参照 Windows 同名函数