ydwe lua引擎使用说明

来源 github/actboy168/jass2lua

简介

ydwe lua引擎(以下简称lua引擎)是一个嵌入到《魔兽争霸III》(以下简称魔兽)中的一个插件，它可以让魔兽可以执行lua并且调用魔兽的导出函数(在common.j内定义的函数)，就像使用jass那样。本说明假定你已经掌握了jass和lua的相关语法，有关语法的问题不再另行解释。

入口

在jass内调用 `call Cheat("exec-lua: hello")`，这等价于在lua里调用了 `require 'hello'` 。lua引擎已经把地图内的文件加载到搜索路径，所以地图内的hello.lua将会得到执行。

lua引擎对标准lua的修改

为了适合在魔兽内使用lua引擎对lua略有修改。

1. math.randomseed改为使用jass函数SetRandomSeed实现。
2. math.random改为使用jass函数GetRandomReal实现。
3. table元素随机化种子依赖于魔兽内部的随机种子。
4. 屏蔽了部分被认为不安全的函数

内置库

lua引擎一共有12个内置库，可以通过"require '库名'"调用。

- jass.common
- jass.ai
- jass.globals
- jass.japi
- jass.hook
- jass.runtime
- jass.slk
- jass.console
- jass.debug
- jass.log
- jass.message
- jass.bignum

jass.common

jass.common库包含common.j内注册的所有函数。 (不包括BJ)

```
local jass = require 'jass.common'
print(jass.GetHandleId(jass.Player(0)))
```

jass.ai

jass.ai库包含common.ai内注册的所有函数。

```
local jass = require 'jass.common'
local ai = require 'jass.ai'
print(ai.UnitAlive(jass.GetTriggerUnit()))
```

jass.globals

jass.globals库可以让你访问到jass内的全局变量。

> 你可以使用此库访问预设在大地图的对象。

```
local cg = require 'jass.globals'
print(cg.udg_i) --获取jass中定义的i整数
```

jass.japi

jass.japi库当前已经注册的所有japi函数。（包含dz函数）

```
local jass = require 'jass.common'
local japi = require 'jass.japi'
japi.EXDisplayChat(jass.Player(0), 0, "Hello!")
```

japi函数不同环境下可能会略有不同，你可以通过pairs遍历当前的所有japi函数

```
for k, v in pairs(require 'jass.japi') do
    print(k, v)
end
```

jass.hook

jass.hook库可以对common.j内注册的函数下钩子。注：jass.common库不会受到影响。

同时，为了避免jass和lua之间传递浮点数时产生误差，通过jass.hook传递到lua中的浮点数，并不是number类型，而是userdata。当你需要**精确**地操纵浮点数时，也请注意这点。

```
local hook = require 'jass.hook'
function hook.CreateUnit(pid, uid, x, y, face, realCreateUnit)
    -- 当jass内调用CreateUnit时，就会被执行
```

```
        print('CreateUnit')
        print(type(x))
        return realCreateUnit(pid, uid, x, y, face)
    end
```

jass.slk

jass.slk库可以在地图运行时读取地图内的slk/w3*文件。

```
    local slk = require 'jass.slk'
    print(slk.ability.AHbz.Name)
```

你也可以遍历一个表或者一个物体（不建议方式）

```
    local slk = require 'jass.slk'
    for k, v in pairs(slk.ability) do
        print(k, v)
    end
    for k, v in pairs(slk.ability.AHbz) do
        print(k, v)
    end
```

slk包含

- unit
- item
- destructable
- doodad
- ability
- buff
- upgrade
- misc

与你物体编辑器中的项目一一对应。

获取数据时使用的索引你可以在物体编辑器中通过Ctrl+D来查询到

注意，当访问正确时返回值永远是字符串。如果你获取的是某个单位的生命值，你可能需要使用tonumber来进行转换。当访问不正确时将返回nil。

jass.runtime

jass.runtime库可以在地图运行时获取lua引擎的信息或修改lua引擎的部分配置。

```
    local runtime = require 'jass.runtime'
```

runtime.console(默认为false)

赋值为true后会打开一个cmd窗口，print与console.write函数可以输出到这里

```
runtime.console = true
```

runtime.version

返回当前lua引擎的版本号

```
print(runtime.version)
```

runtime.error_handle

当你的lua脚本出现错误时将会调用此函数。

runtime.error_handle有一个默认值，等价于以下函数

```
runtime.error_handle = function(msg)
    print("Error: ", msg, "\n")
end
```

你也可以让它输出更多的信息，比如输出错误时的调用栈

```
runtime.error_handle = function(msg)
    print("----------------------------------------")
    print("               LUA ERROR!!              ")
    print("----------------------------------------")
    print(tostring(msg) .. "\n")
    print(debug.traceback())
    print("----------------------------------------")
end
```

注意，注册此函数后lua脚本的效率会降低(即使并没有发生错误)。

runtime.handle_level(默认为0)

lua引擎处理的handle的安全等级，有效值为0~2，注，等级越高，效率越低，安全性越高、

0: handle直接使用number，jass无法了解你在lua中对这个handle的引用情况，也不会通过增加引用计数来保护这个handle

```

```

```lua
local t = jass.CreateTimer()
print(t) -- 1048000
type(t) -- "number"
```

1: handle封装在lightuserdata中，保证handle不能和整数相互转换，同样不支持引用计数

```lua
local t = jass.CreateTimer()
print(t) -- "handle: 0x10005D"
type(t) -- "userdata"
jass.TimerStart(t, 1, false, 0) -- ok
```

```lua
local t = jass.CreateTimer()
local h1 = jass.CreateTimer()
jass.DestroyTimer(h1)
jass.TimerStart(t, 1, false,
    function()
        local h2 = jass.CreateTimer()
        print(h1) -- "handle: 0x10005E"
        print(h2) -- "handle: 0x10005E"
    end
)
```

2: handle封装在userdata中，lua持有该handle时将增加handle的引用计数。lua释放handle时会释放handle的引用计数。

```lua
local t = jass.CreateTimer()
local h1 = jass.CreateTimer()
jass.DestroyTimer(h1)
jass.TimerStart(t, 1, false,
    function()
        local h2 = jass.CreateTimer()
        print(h1) -- "handle: 0x10005E"
        print(h2) -- "handle: 0x10005F"
    end
)
```

runtime.sleep(默认为false)

common.j中包含sleep操作的函数有4个，TriggerSleepAction/TriggerSyncReady/TriggerWaitForSound/SyncSelections。当此项为false时，lua引擎会忽略这4个函数的调用，并给予运行时警告。当此项为true时，这4个函数将会得到正确的执行。

但请注意此项为true时将降低lua引擎的运行效率，即使你没有使用这4个函数。

```
local trg = jass.CreateTrigger()
local a = 1
jass.TriggerAddAction(trg, function()
    jass.TriggerSleepAction(0.2)
    print(a) -- 2
end)
jass.TriggerExecute(trg)
a = 2
```

runtime.catch_crash(默认为true)

调用jass.xxx/japi.xxx发生崩溃时，会生产一个lua错误，并忽略这个崩溃。你可以注册jass.runtime.error_handle来获得这个错误。注：开启此项会略微增加运行时消耗（即使没有发生错误）。

runtime.debugger

启动调试器并监听指定端口。需要使用VSCode并安装Lua Debug。

```
runtime.debugger = 4279
```

jass.console

jass.console与控制台相关

console.enable(默认为false)

赋值为true后会打开一个cmd窗口，print与console.write函数可以输出到这里

```
console.enable = true
```

console.write

将utf8编码的字符串转化为ansi编码后输出到cmd窗口中，如果你需要输出魔兽中的中文，请使用该函数而不是print

console.read

将控制台中的输入传入魔兽中(会自动转换编码)

首次调用console.read后将允许用户在控制台输入，输入完成后按回车键提交输入。

用户提交完成后，传入一个函数f来调用console.read，将会调用函数f，并将用户的输入作为参数传入(已转换为utf8编码)。

推荐的做法是每0.1秒运行一次console.read，见下面的例子：

```
    local jass    = require 'jass.common'
    local console = require 'jass.console'

    console.write('测试开始...')

    --开启计时器,每0.1秒检查输入
    jass.TimerStart(jass.CreateTimer(), 0.1, true,
        function()

            --检查CMD窗口中的用户输入,如果用户有提交了的输入,则回调函数(按回车键提交输入).否则不做任何动作
            console.read(
                function(str)
                    --参数即为用户的输入.需要注意的是这个函数调用是不同步的(毕竟其他玩家不知道你输入了什么)
                    jass.DisplayTimedTextToPlayer(jass.Player(0), 0, 0, 60, '你在控制台中输入了:' .. str)
                end
            )
        end
    )
```

需要注意的是控制台输入是不同步的。

jass.debug

jass.debug库能帮助你更深入地剖析lua引擎的内部机制。

- functiondef jass.common或者jass.japi函数的定义

```
    local jass = require 'jass.common'
    local dbg = require 'jass.debug'
    print(table.unpack(dbg.functiondef(jass.GetUnitX)))
```

- globaldef jass.globals内值的定义
- handledef handle对应对象的内部定义
- currentpos 当前jass执行到的位置
- handlemax jass虚拟机当前最大的handle
- handlecount jass虚拟机当前的handle数
- h2i/i2h handle和integer的转换，当你runtime.handle_level不是0时，你可能会需要它
- handle_ref 增加handle的引用
- handle_unref 减少handle的引用

- ~~gchash~~（已废弃） 指定一张table的gchash，gchash会决定了在其他table中这个table的排序次序
  在默认的情况下，lua对table的排序次序是由随机数决定的，不同玩家的lua生成的随机数不一致，所以下面的代码在不同的玩家上执行的次序是不一致的，这可能会引起不同步掉线

jass.log

日志库

- path 日志的输出路径
- level 日志的等级，指定等级以上的日志才会输出
- 日志有6个等级 trace、debug、info、warn、error、fatal

```lua
local log = require 'jass.log'
log.info('这是一行日志')
log.error('这是一行', '日志')
```

jass.message

- keyboard 一张表，魔兽的键盘码
- mouse 本地玩家的鼠标坐标(游戏坐标)
- button 本地玩家技能按钮的状态
- hook 魔兽的消息回调，可以获得部分鼠标和键盘消息
- selection 获得本地玩家当前选中单位
- order_immediate 发布本地命令，无目标
- order_point 发布本地命令，点目标
- order_target 发布本地命令，单位目标
- order_enable_debug 开启后，会在控制台打印当前的本地命令，调试用

jass.bignum

大数库