今天没事 写一篇用 lua + vscode 制图的基本方法

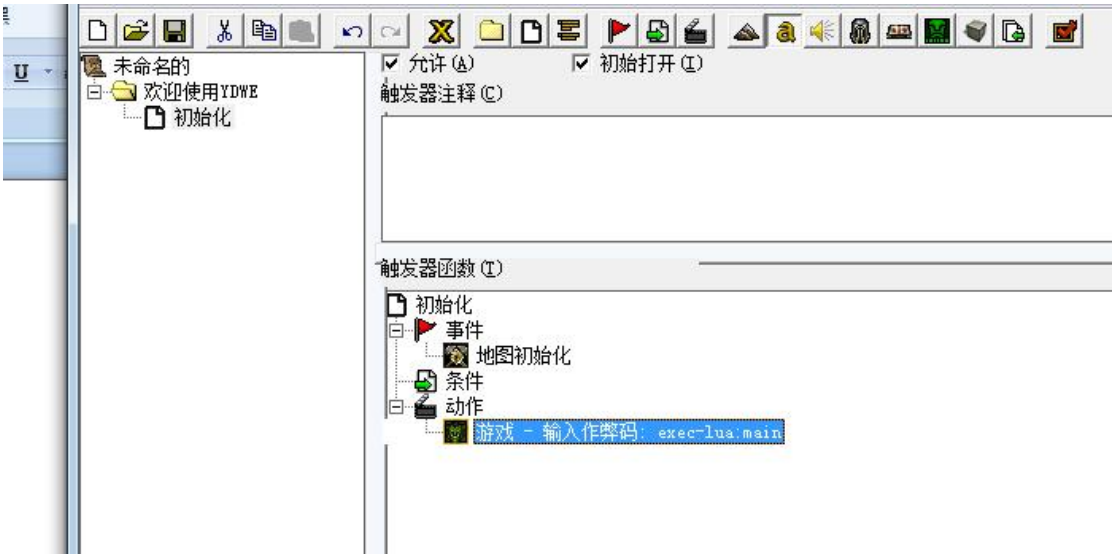
所需工具

YDWE 1.31 版本以上均可 （百度下载）

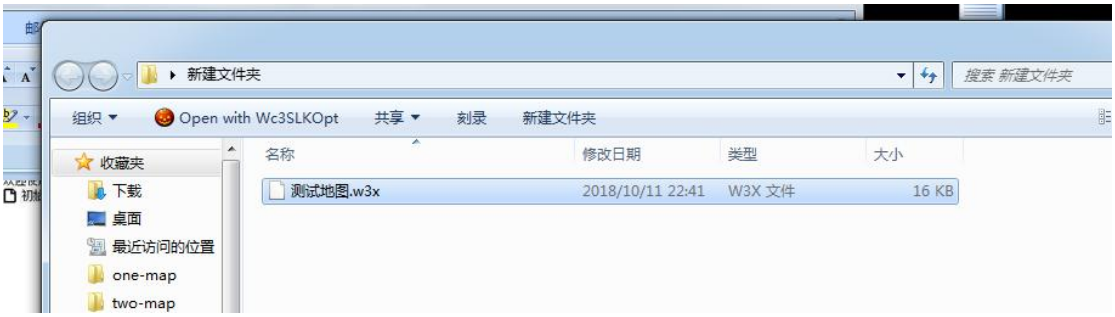
w3x2lni 2.4 版本以上 （贴吧精品里有）

vscode 我用的是 2.0 这个版本无所谓 （百度下载）

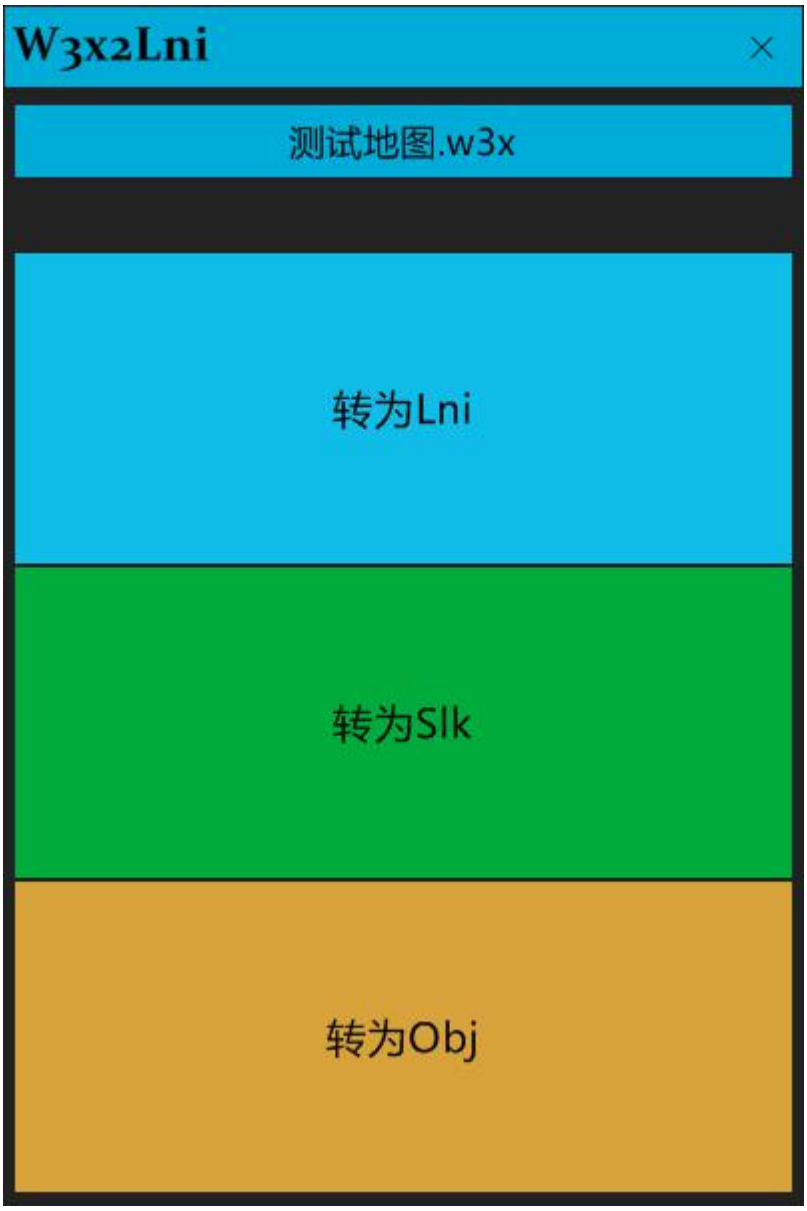
首先用 YDWE 新建一张地图



之后写一个这个的触发 直接保存地图 另存为到一个新建文件夹里 如



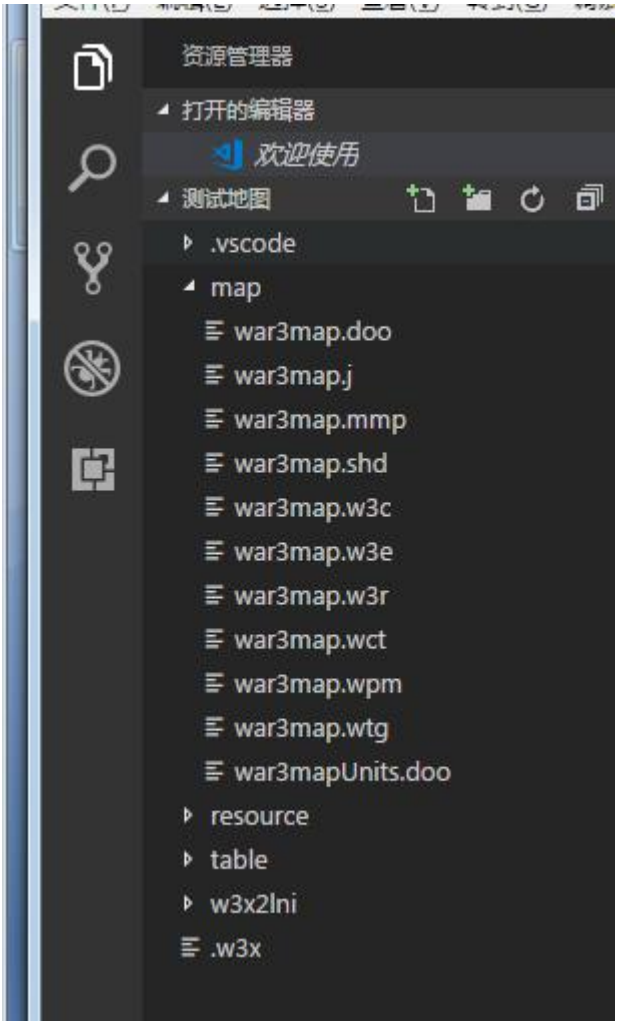
之后 打开 w3x2lni



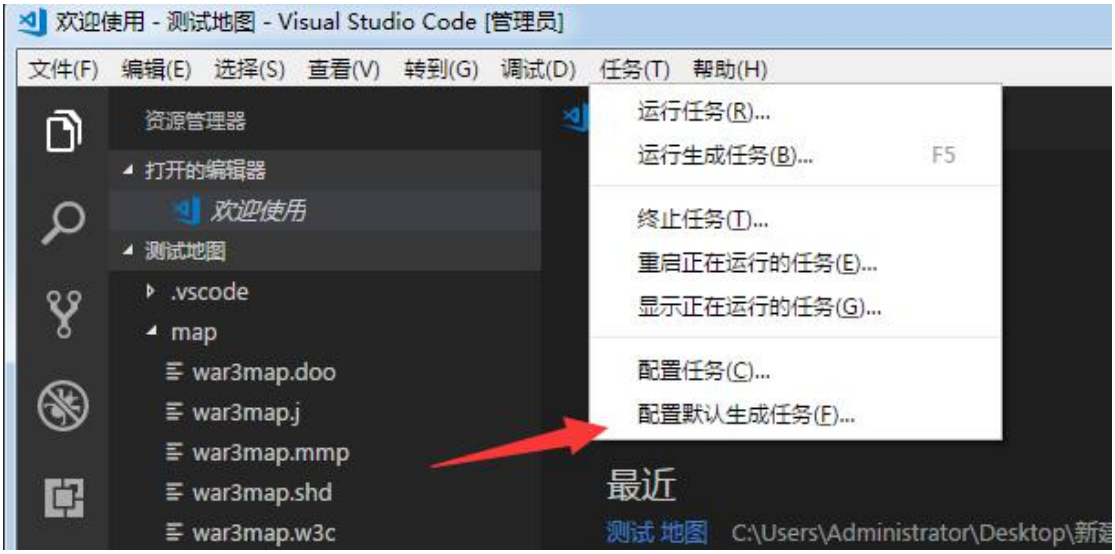
将测试地图转换成 lni 文件夹格式

之后打开 vscode

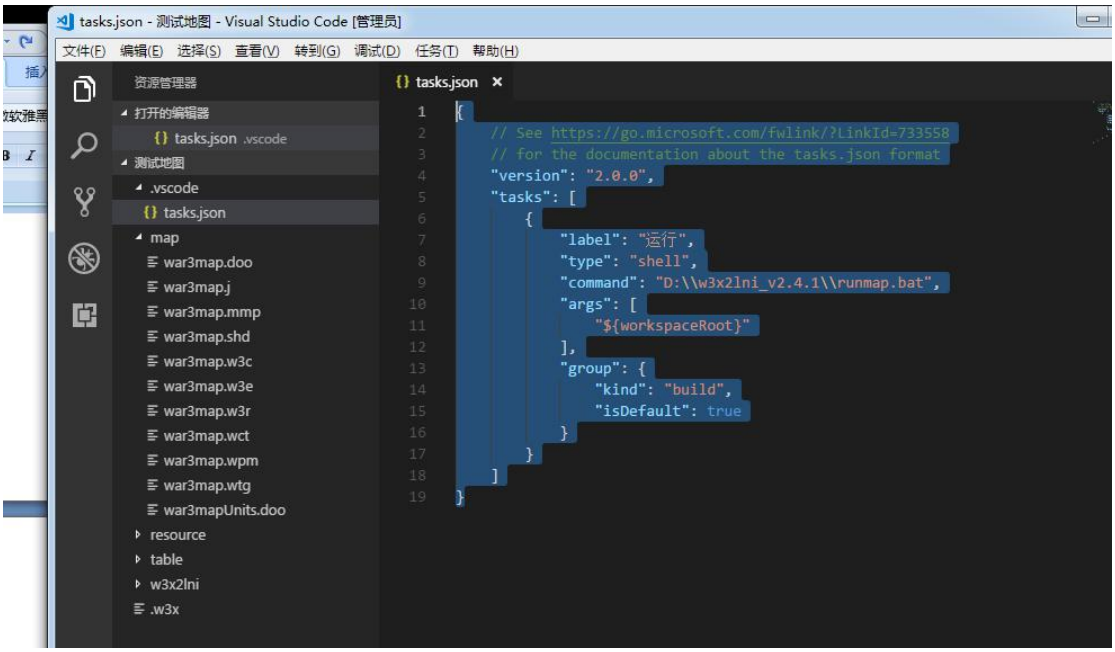
将存放 地图文件夹 拖进 vscode 里 就可以看到 vscode 左边 的文件列表


资源管理器
打开的编辑器
欢迎使用
测试地图
.vscode
map
war3map.doo
war3map.j
war3map.mmp
war3map.shd
war3map.w3c
war3map.w3e
war3map.w3r
war3map.wct
war3map.wpm
war3map.wtg
war3mapUnits.doo
resource
table
w3x2lni
.w3x

之后 点这里


欢迎使用 - 测试地图 - Visual Studio Code [管理员]
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 调试(D) 任务(T) 帮助(H)
资源管理器
打开的编辑器
欢迎使用
测试地图
.vscode
map
war3map.doo
war3map.j
war3map.mmp
war3map.shd
war3map.w3c
运行任务(R)...
运行生成任务(B)... F5
终止任务(T)...
重启正在运行的任务(E)...
显示正在运行的任务(G)...
配置任务(C)...
配置默认生成任务(F)...
最近
测试 地图 C:\Users\Administrator\Desktop\新

点配置默认生成任务 之后会 出现这个文件页面， 将我这段 配置复制进去




```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "运行",
            "type": "shell",
            "command": "D:\\w3x2lni_v2.4.1\\runmap.bat",
            "args": [
                "${workspaceRoot}"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

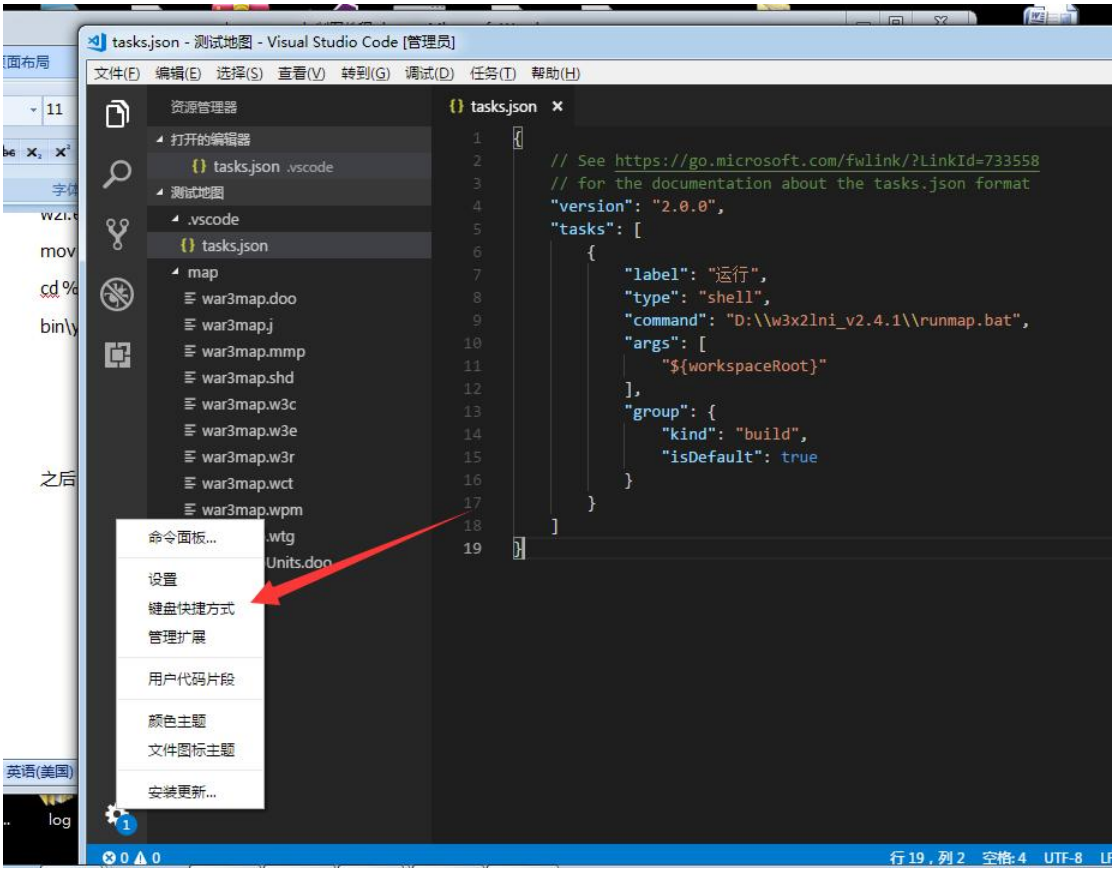
这里 的 D:\\w3x2lni_v2.4.1 是 wx2lni 的工具目录 后面这个 bat 的内容则是 这样的

在 D:\\w3x2lni_v2.4.1 这个工具目录下 右键 新建文件 新建一个 名为 runmap.bat 的文件 将 下面这段代码复制进去

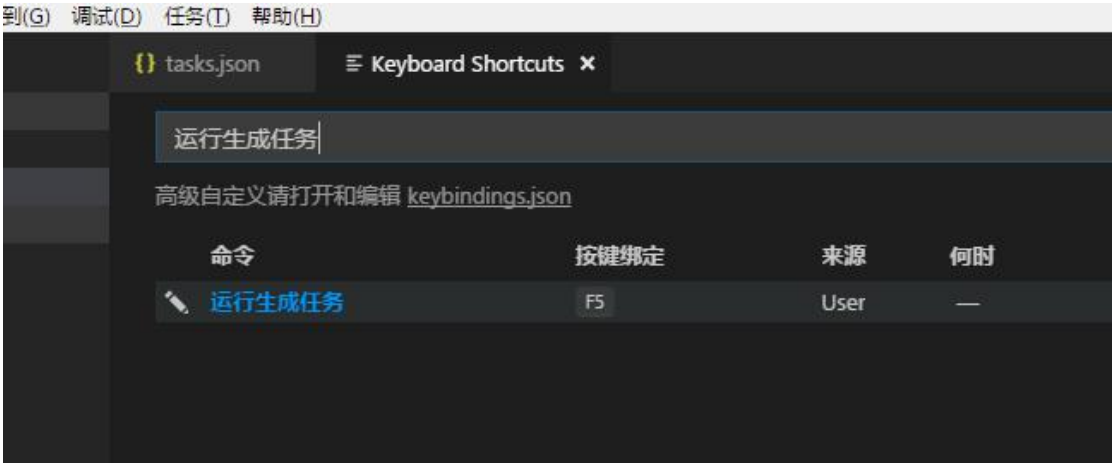
注意这里 ydwePath=ydwe 的路径 换成你的 yd 路径 之后保存文件

```
@echo off
cd /d %~dp0
set ydwePath=D:\YDWE 1.31.8+API 1.0
set mapPath=%~dpn1
set mapName=%~n1
w2l.exe obj "%mapPath%"
move "%mapPath%.w3x" "%ydwePath%\%mapName%.w3x"
cd %ydwePath%
bin\ydweconfig.exe -launchwar3 -loadfile "%mapName%.w3x"
```

之后 点这里　键盘快捷方式 改一下快捷键



在这里输入 运行生成任务 把快捷键 改成 F5 即可



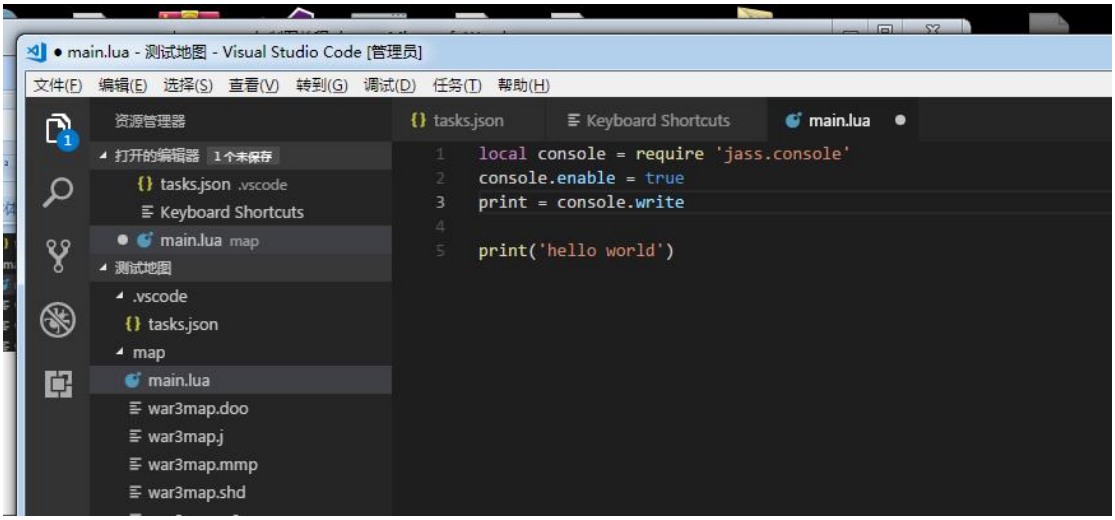
之后按一下 F5 快捷键 就可以立即保存地图 进到游戏里了

之后我们开始编写第一个 lua 文件 还记得我们刚才第一步 在 ydwe 里新建的那个触发吗

右键 map 新建文件 然后新建文件的名字 为 main.lua

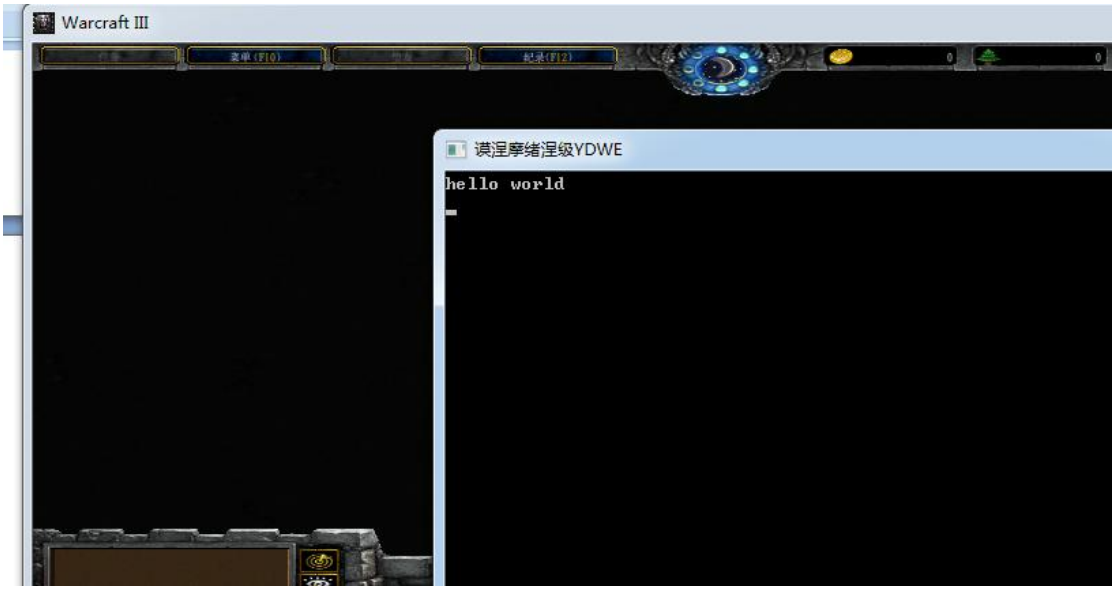
之后 可以开始编写我们的 lua 脚本了

例如

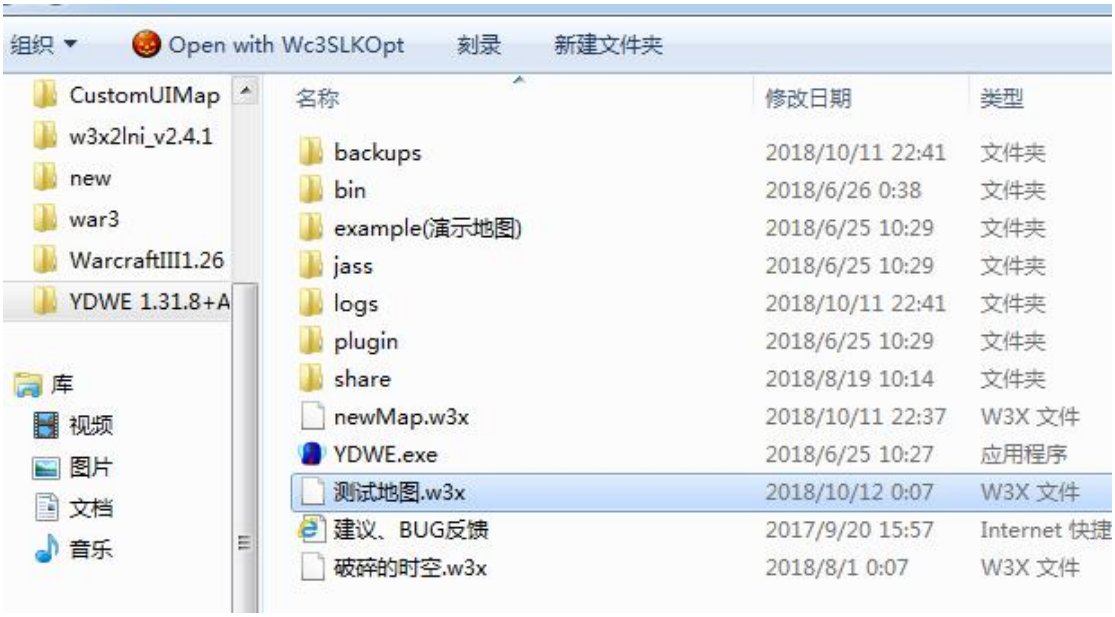


改写完文件后 按 Ctrl + S 键 进行保存

之后再按 F5 就可以立即进到游戏里 查看效果了



每次 F5 都会保存一个地图在 ydwe 目录下 如有需要 可以用 yd 再打开



附上文件 runmap.bat 一份

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

快速开始

W3x2Lni提供了图形界面和命令行两个版本。

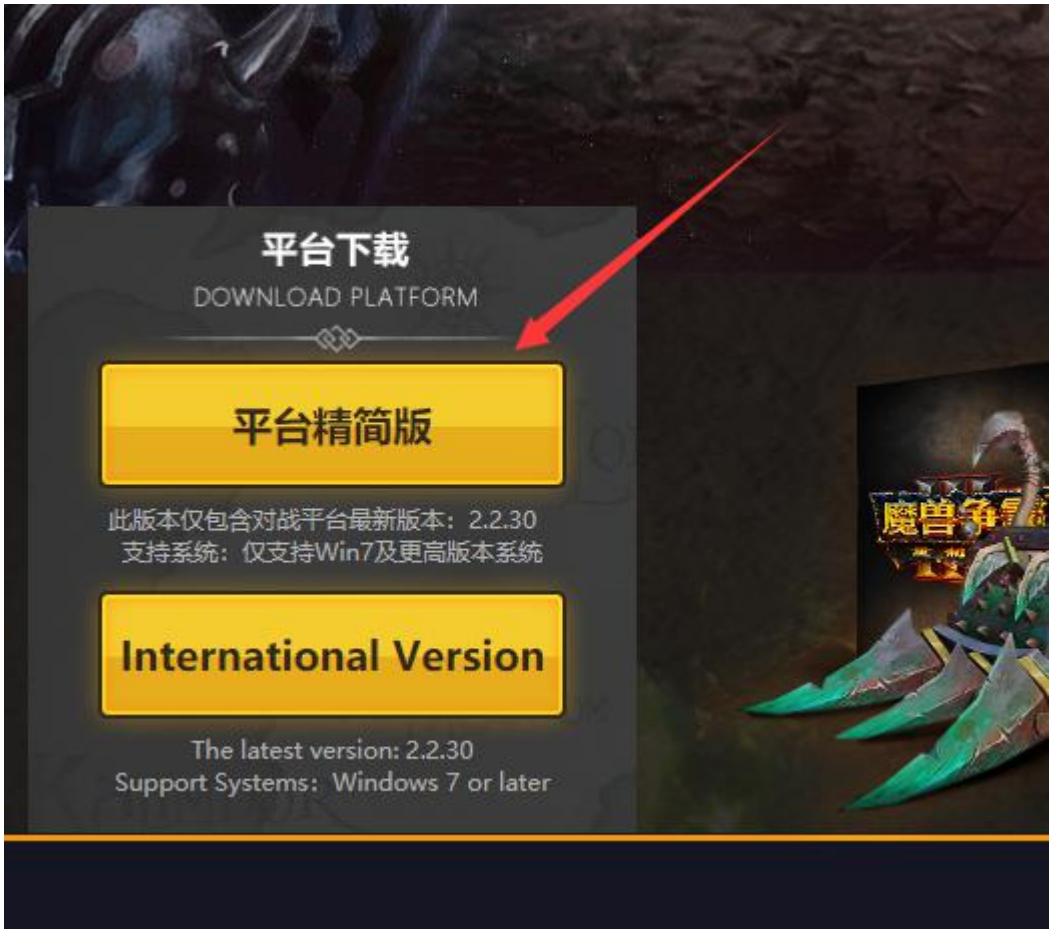
> 图形界面

- 双击`W3x2Lni.exe`
- 拖入地图
- 点击要转化的格式

> 命令行

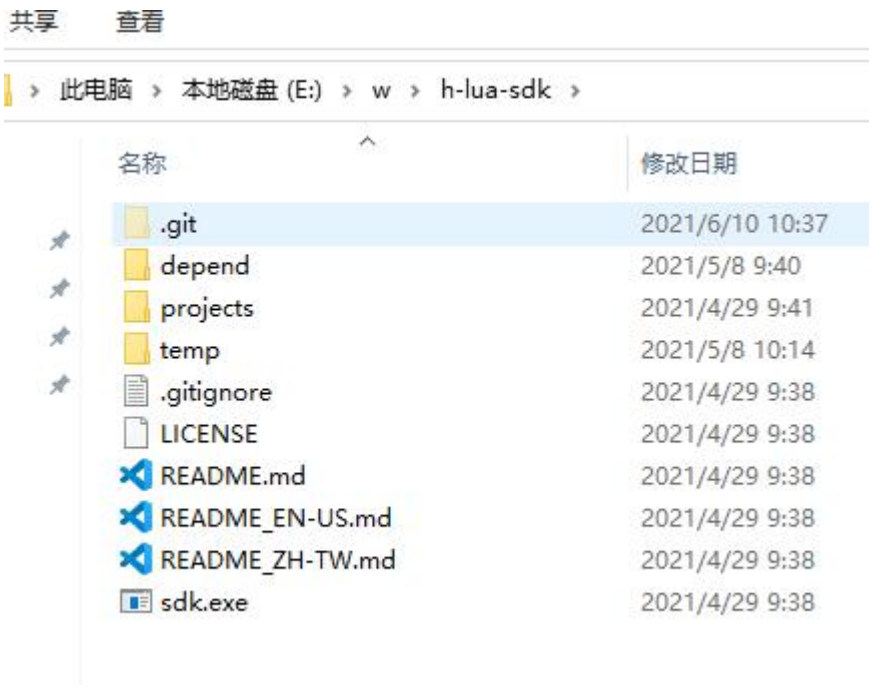
- 运行`w2l.exe help`以获取帮助
- 将`w2l.exe`添加到环境变量，方便写脚本

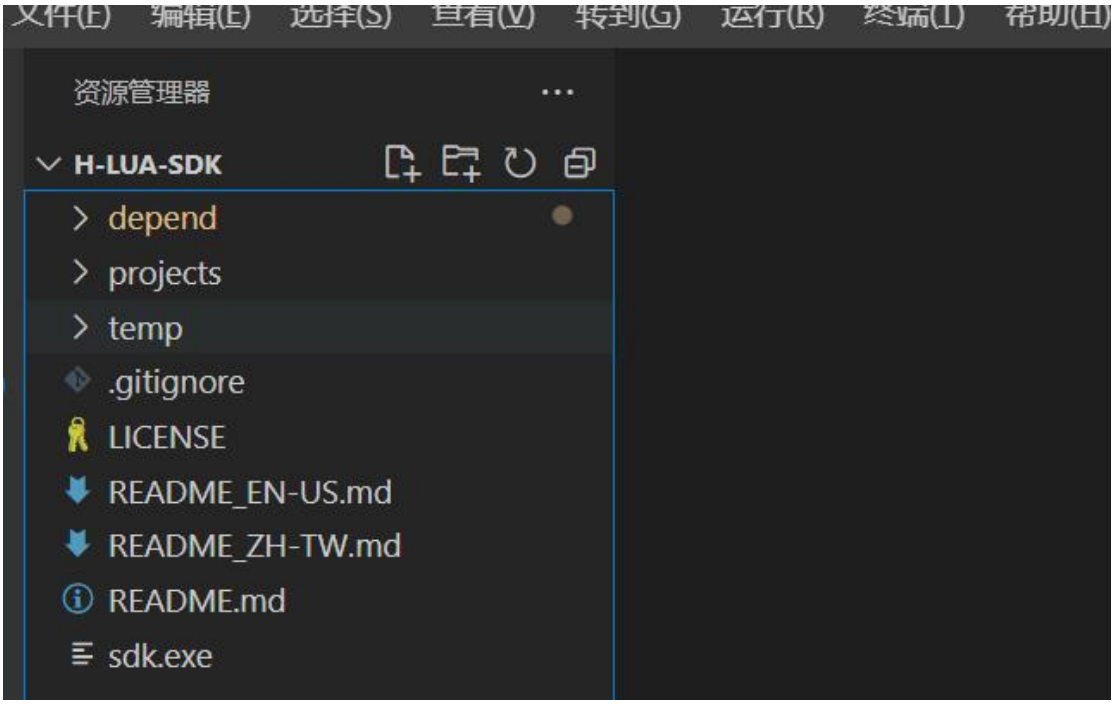
1. 下载对战平台(https://dz.blizzard.cn/)



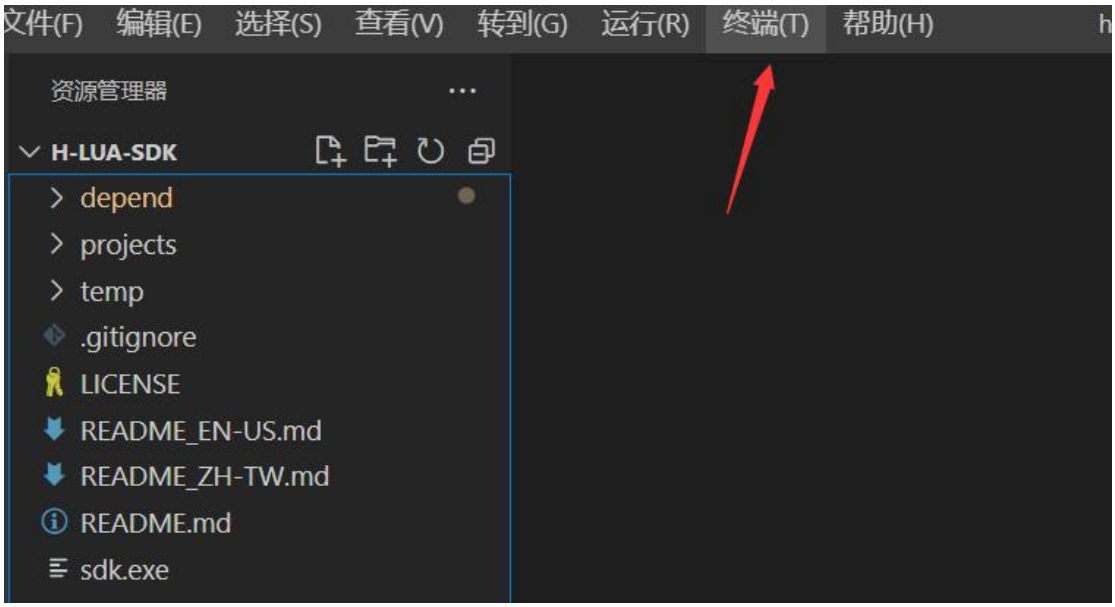
2. 安装对战平台 下载 1.27 客户端
3. 下载 h-lua-sdk https://gitee.com/hunzsig/h-lua-sdk

4. vscode 打开 h-lua-sdk 的目录


共享
查看
此电脑 › 本地磁盘 (E:) › w › h-lua-sdk ›
名称
修改日期
.git 2021/6/10 10:37
depend 2021/5/8 9:40
projects 2021/4/29 9:41
temp 2021/5/8 10:14
.gitignore 2021/4/29 9:38
LICENSE 2021/4/29 9:38
README.md 2021/4/29 9:38
README_EN-US.md 2021/4/29 9:38
README_ZH-TW.md 2021/4/29 9:38
sdk.exe 2021/4/29 9:38

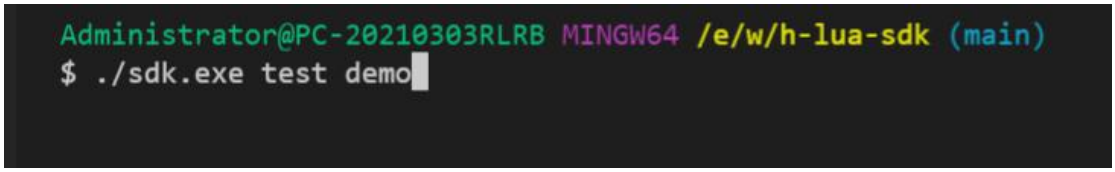

资源管理器
H-LUA-SDK
depend
projects
temp
.gitignore
LICENSE
README_EN-US.md
README_ZH-TW.md
README.md
sdk.exe

5. 打开终端

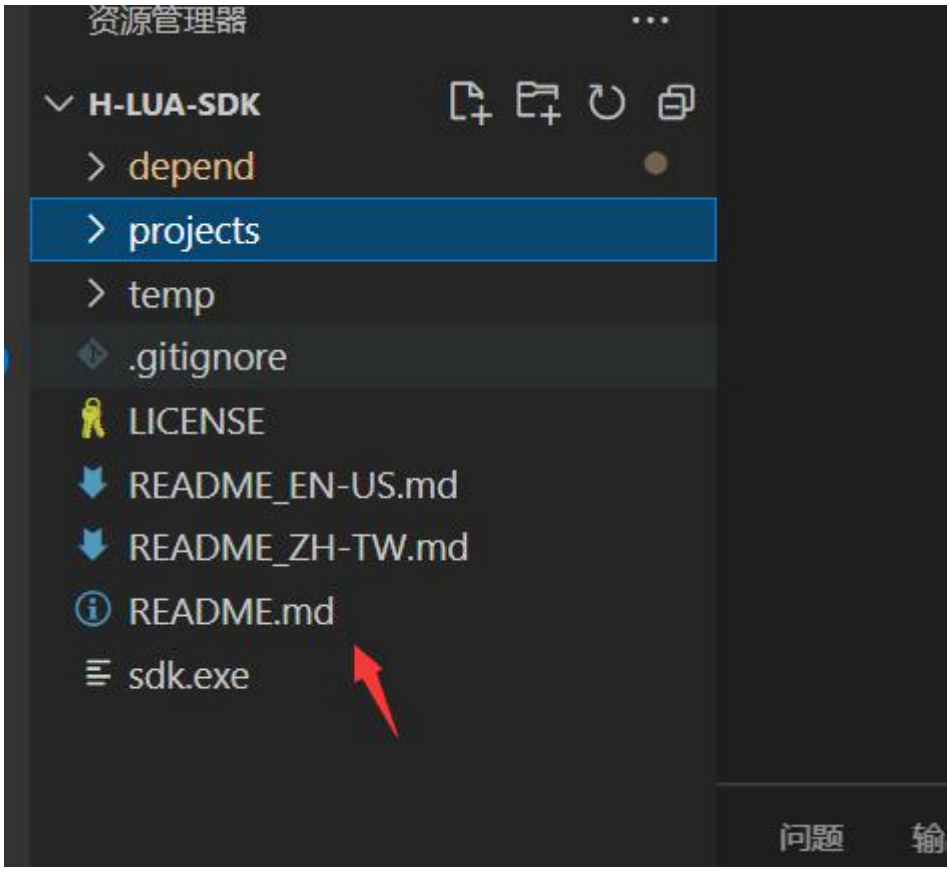


5. 新建地图 输入`./sdk.exe new demo`

6. 测试地图 输入`./sdl.exe test demo`



7. 更多命令 查看`README.md`




```
### 命令行
```
* 必填 ~ 选填
./h-lua-sdk> sdk.exe help //提示cmd工具命令
./h-lua-sdk> sdk.exe new [*PROJECT_NAME] //新建一个地图项目
./h-lua-sdk> sdk.exe ydwe [*PROJECT_NAME] //以YDWE打开地图项目
./h-lua-sdk> sdk.exe model [*PROJECT_NAME] [~CURRENT_PAGE:0] //以YDWE浏览项目模型，一页最大289个，可翻
./h-lua-sdk> sdk.exe clear [*PROJECT_NAME] //清理构建的临时文件
./h-lua-sdk> sdk.exe test [*PROJECT_NAME] //构建测试版本并开启游戏进行调试
./h-lua-sdk> sdk.exe build [*PROJECT_NAME] //构建上线版本并开启游戏进行调试
```

### 命令行的缩写版
```
* 必填 ~ 选填
./h-lua-sdk> sdk.exe -h //提示cmd工具命令
./h-lua-sdk> sdk.exe -n [*PROJECT_NAME] //新建一个地图项目
./h-lua-sdk> sdk.exe -we|-yd [*PROJECT_NAME] //以YDWE打开地图项目
./h-lua-sdk> sdk.exe -m [*PROJECT_NAME] [~CURRENT_PAGE:0] //以YDWE浏览项目模型，一页最大289个，可翻页
./h-lua-sdk> sdk.exe -c [*PROJECT_NAME] //清理构建的临时文件
./h-lua-sdk> sdk.exe -t [*PROJECT_NAME] //构建测试版本并开启游戏进行调试
./h-lua-sdk> sdk.exe -b [*PROJECT_NAME] //构建上线版本并开启游戏进行调试
```
```

插件

w3x2lni支持自定义插件，你可以用插件来修改转换文件的行为。

!> w3x2lni不会对插件行为做安全检查，进行一些错误的操作可能导致转换出错。

快速开始

在w3x2lni根目录新建`plugin`文件夹，在里面放一个`.config`文件，写上想要加载的插件名，w3x2lni便会去加载`plugin\插件名.lua`。

plugin\.config

```
修改单位名
```

plugin\修改单位名.lua

```lua
local mt = {}

mt.info = {
    name = '修改单位名',
    version = 1.0,
    author = '最萌小汐',
    description = '将所有单位的名字加上前缀"被插件修改过的"。',
}

function mt:on_full(w2l)
    for id, obj in pairs(w2l.slk.unit) do
        obj.name = '被插件修改过的' .. obj.name
    end
end

return mt
```

配置

配置文件为`plugin\.config`，在该文件中写上想要加载的插件名，w3x2lni便会去加载相应的插件。如果要加载多个插件，可以每行写一个插件名，w3x2lni会按照顺序去加载这些插件。

```
插件1
插件2
插件3
```

插件

插件为`plugin\插件名.lua`，需要在`plugin\.config`中定义后才会加载。插件应该是一个lua脚本，它需要返回一张表，在表中可以有以下属性或方法：

info

插件的基本信息，是一张拥有下列属性的表：

- `name` 插件的名字，字符串。
- `version` 插件的版本号，数字。
- `author` 插件的作者，字符串。
- `description` 插件的描述，字符串。

```
mt.info = {
    name = '插件名',
    version = 1.0,
    author = '插件作者',
    description = '插件描述',
}
```

on_full

完整数据（Full）事件，关于完整数据的定义见这里。在该事件中可以简单方便的修改物编数据从而修改转换后的结果。

> 完整数据内的数据格式可以参考`data\zhCN-1.24.4\prebuilt\Custom`

```
-- 让所有技能无冷却无消耗
function mt:on_full(w2l)
    for id, skill in pairs(w2l.slk.ability) do
        for i = 1, skill._max_level do
            skill.cost[i] = 0
            skill.cool[i] = 0
        end
    end
end
```

on_mark

引用标记事件，在该事件中可以对对象的引用进行标记，以免转换Slk时对象被当做未使用对象而删除。这个事件期待返回一张表，这张表的所有`key`对应的对象都会被标记为引用。

```
-- 引用L000 - L009的对象，这些对象只在Lua脚本中使用，无法被自动引用
function mt:on_mark()
    local list = {}

    for i = 0, 9 do
```

```
            list['L00'..i] = true
        end

        return list
    end
```

接口

w3x2lni没有为插件准备专用的接口，而是将插件当做了代码的一部分，在调用插件的事件时将当前会话作为参数传入。也就是说，插件可以任意使用w3x2lni内部的函数，任意修改会话状态，你需要自己确保转换不会出错。这里提供一些常用的内部方法（假定传入的会话保存在变量`w2l`中）：

slk

物编数据表，数据结构参考`data\zhCN-1.24.4\prebuilt\Custom`

```
for type, list in pairs(w2l.slk) do
    for id, obj in pairs(list) do
        for key, value in pairs(obj) do
        end
    end
end
```

setting

配置表，除了在`config.ini`中能看到的属性以外，还有以下属性：

- input (filesystem) - 输入路径。
- output (filesystem) - 输出路径。
- target_storage (string) - 输出格式，`mpq`表示打包成地图，`dir`表示生成目录。
- mode (string) - 输出模式，`slk`、`obj`或`lni`。
- version (string) - 地图版本，`Custom`表示自定义地图，`Melee`表示对战地图。

```
if w2l.setting.mode == 'slk' then
end
```

file_save

保存文件

- type (string) - 文件类型，参考Lni后的目录结构
- path (string) - 文件名
- buf (string) - 文件内容

```
w2l:file_save('scripts', 'blizzard.j', '')
```

file_load

读取文件

- type (string) - 文件类型
- path (string) - 文件名

```
local buf = w2l:file_load('script', 'blizzard.j')
```

file_remove

删除文件

- type (string) - 文件类型
- path (string) - 文件名

```
w2l:file_remove('script', 'blizzard.j')
```

input_mode

输入文件模式，`lni`表示输入地图是Lni模式的。

地图内插件

将`plugin`目录放在地图的`w3x2lni`目录中便是地图内插件，这里的地图既可以是目录格式（`Lni`）也可以是MPQ格式（`Obj`）。W3x2lni在转换该地图时便会应用地图内的插件。地图内插件会继续保留在转后的地图内，除非转换模式为`Slk`且启用了`删除只在WE中使用的文件`。

实现原理

本文不是W3x2Lni的使用帮助，只是简单介绍W3x2Lni核心的实现原理。

> Full

除了Obj、Lni、Slk，实际上W3x2Lni还存在第四种地图格式，我们称之为Full。将地图转为某一个格式，都是先转为Full再转到目标格式。例如将地图转为Slk，实际上的流程是 Map -> Full -> Slk。从Map -> Full的转换过程，我们称之为Core Frontend；而从Full -> Obj/Lni/Slk我们称之为Core Backend。

Full实际上即是保留所有信息的一种数据格式，而Obj、Lni、Slk都会因为某种原因忽略掉部分信息。

Core Frontend

> Map

Map指的是一张未知格式的地图。实际上W3x2Lni不会对输入的地图格式做任何的假设，输入的地图可以同时拥有lni、slk、txt、w3u等文件。它们会被W3x2Lni读取并转换为Full。

> 地图的储存形式

W3x2Lni支持读取三种的地图的储存形式，分别是w3x、dir、lni。w3x即是WE和War3通常使用的mpq格式。dir即是将w3x完全解压为文件夹的格式。lni则是W3x2Lni定义的一种储存形式。(注意，和地图格式Lni不是一回事)

W3x2Lni支持任意一种储存形式的地图作为输入，但对于输出，W3x2Lni会使用特定的储存形式。分别对应的是

- Obj 使用 w3x
- Slk 使用 w3x
- Lni 使用 lni

> Metadata

在理解数据如何变为Full前，你需要先知道什么是Metadata。在War3的mpq里存在几个xxxmetadata.slk的文件，WE通过这些文件定义的规则生成Obj格式的数据。但这不是xxxmetadata.slk文件的全部，xxxmetadata.slk还定义了Slk格式数据的规则，但只是定义，War3并不会读取这些文件中的规则来访问Slk数据。War3访问Slk和Obj数据的规则是硬编码在War3内部的。

简单来说，Metadata定义了Slk和Obj数据的构成规则。WE的Metadata是mpq里的xxxmetadata.slk文件。而War3使用的Metadata硬编码在War3内部。WE和War3的Metadata并非完全一致，这也是诸多WE的bug的来源。

在W3x2Lni中，Full格式会使用War3的Metadata。因为我们认为地图的最终目标是在War3上正常运行，所以War3认同的数据才是正确和有意义的数据。如果输入地图的某些数据不符合War3的Metadata，将会导致数据被转义、忽略等情况(请留意日志中的警告和错误)。

> Map -> Full

从Map -> Full可以分为5步。

1. 分别读取slk，obj，lni数据

2. 补全obj数据
3. 补全lni数据
4. 合并obj和slk数据
5. 合并lni和slk数据

抛去存储格式，obj和lni的数据形式几乎一样，而slk则是和full几乎一样。所以这个流程可以粗浅地理解为将obj和lni分别都转为full，然后再将三份full的数据合并在一起。

> Obj/Lni -> Full

Obj数据可以简单地理解为一个补丁。Obj的每一个对象都有一个`parent`，这个`parent`一定是某一个`Slk`里的对象，而Obj里这个对象的其它值都是针对`parent`的差异。所以`Obj`/`Lni` -> `Full`的过程则是，从`Slk`复制一份`parent`并应用所有的补丁。

> Obj/Lni和Slk合并

在经过2~3步的处理后，Obj/Lni/Slk的数据格式已经几乎一致了，我们只需要根据优先级将三份数据合并在一起即可。优先级由高到低的次序为

1. Lni
2. Obj
3. Slk

Core Backend

> Metadata

在`Map` -> `Full`的过程中已经介绍了，什么是`Metadata`，以及`Map` -> `Full`会使用War3的`Metadata`。而在`Core Backend`，当`Full`转为其它格式时也需要使用`Metadata`。不过不同的目标格式会使用不同的`Metadata`。

- Obj 使用WE的`Metadata`
- Slk 使用War3的`Metadata`
- Lni 使用War3的`Metadata`

W3x2Lni会尽可能地使用War3的`Metadata`，因为它是正确的规则。但正确的规则不一定能被WE接受，这就是为什么W3x2Lni会在`Obj`格式中使用WE的`Metadata`。

这样做会导致你的一些正确的数据会被忽略或转义，但不这样做会导致你的WE无法访问到这张地图的这些数据。

如何避免这种情况？两种办法

- 让你的WE使用正确的`Metadata`
- 不要使用`Obj`格式，换句话说，不要使用WE编辑数据。

> Full -> Slk/Obj/Lni

这个实际上只是`Core Frontend`所做事情的逆序，当你理解如何将如何将Slk/Obj/Lni转为Full后，这个也不难明白其实现原理。