Dice!嵌入模块开发手册

（原Shiki Plugin Manual）

简介

本手册是对Dice!2.4.2(build570)新增的自定义指令功能、Dice!2.5.1(build577)新增的自定义任务、Dice!2.6.0(build577)支持调用Lua的关键词回复、Dice!2.6.4(build612)支持的事件处理等所作的说明，**当前对应最新版本Dice!2.6.6rc(638)**。通过在mod目录安装扩展模块，在plugin目录放入lua脚本或toml配置文件，Dice!将监听特定消息或事件并做出响应，也可基于时刻或循环处理定时任务，从而实现骰主对骰娘的深度化定制。扩展模块旨在以下方面实现对Dice!内建功能的补充：

1. 过于客制化而无法使用现有数据结构表现的功能，如：基于D20结果的数值分段回复；
2. 同人性质或版本各异的规则，如：JOJO团、圣杯团、骑士团、方舟团等；
3. 因受众过偏不适合写入骰娘源代码的规则；
4. 非骰娘功能，如：好感度系统；

从零编写lua脚本需要对lua有一定的了解，**建议使用VS Code等工具编辑脚本**。如果实在脚本苦手，手册附录准备了现成的样例脚本，你也**可以在Shiki官方群或论坛内找到Shiki免费发布共享的脚本**。Shiki的桌游指令集已收录：DND检定、ShadowRun、双重十字、永夜后日谈（命中检定）、祸不单行、山屋惊魂。**如果你只要安装现成脚本，只需记住在存档目录下创建plugin文件夹，将lua文件放入后启动或.system load**。如果你实在有想实现的指令，没办法修改现成脚本实现，也确实找不到人白嫖的话，可以联系Shiki定制。愿所有人能从Dice!骰娘处获得更好的用户体验，拥有自己独一无二的骰娘。

Shiki的开发官方群:【一群】928626681【二群】1029435374

——安研色Shiki

版权声明

Dice!使用AGPLv3许可，如果你将Dice!与Lua脚本结合用于分发或通过网络提供服务，那么Dice!与所搭载脚本将作为整体使用AGPLv3许可，任何接收者或用户基于AGPLv3许可，均完全享有获取并公开完整代码的自由。

同样地，本手册使用AGPLv3许可，如果您通过付费方式获取到本手册或本手册附带的Lua脚本，那么您显然付出了不必要的代价。

特别地，若您将Dice!源代码从C++重写为其他语言，这种翻译行为属于著作权法意义上的修改，因此您翻译的代码一旦分发或用于网络交互，也强制使用AGPLv3协议且必须保留Dice!开发者的署名。AGPLv3文本格式参考：

```
--[[
Copyright (C) 2018-2021 w4123溯洄
Copyright (C) 2019-2021 String.Empty
This program is free software: you can redistribute it
and/or modify it under the terms of the GNU Affero General
Public License as published by the Free Software
Foundation, either version 3 of the License, or (at your
option) any later version.

This program is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU Affero General Public License for
more details.

You should have received a copy of the GNU Affero General
Public License along with this program. If not, see
<http://www.gnu.org/licenses/>.
]]
```

模块Mod

模块是Dice!外置可热插拔的扩展文件集，远程装卸、开关的单元。Mod采用类Paradox风格结构，由Dice目录/mod/下的json文件及其同名文件夹构成：mod_name.json写有mod的标题、作者、版本信息、说明文本等；同名mod_name目录下子目录存放不同类型文件。

阅读说明：下文中形如(build609+)的注释用于说明该特性在何时被添加或最后编辑，开发者应当在制作mod时，将dice_build设定为满足mod内所有特性要求的最小build。

```
[DiceData]
|-- mod
    |-- mod_name.json
    |-- mod_name
        |-- event
        |   |-- good_morning.lua
        |
        |-- reply
        |   |-- good_morning.lua
        |
        |-- script
        |   |-- event_good_morning.lua
        |   |-- reply_good_morning.lua
        |
        |-- speech
        |   |-- rlobal_msg.yaml
        |
        |-- image
        |-- audio
```

Mod主文件

```
{
    "title":"角色卡栏位扩展",
    "ver":"1.0.0",
    "author":"安研色Shiki",
    "dice_build":606,
    "brief":"角色卡栏位扩展（状态栏、物品栏、法术栏、专长栏、笔记栏等）",
```

```
    "require":[],
    "comment":"",
    "helpdoc":{
        "title":"模块名称",
        "ver":"mod版本号",
        "author":"作者署名",
        "dice_build":"支持mod运行的最低Dice!版本号，低于此项的
Dice!将放弃读取该mod",
        "brief":"会在Dice!中展示的模块简介",
        "require":"mod所需前置mod，若未满足则将放弃加载",
        "comment":"不会写入，仅用作文件内的注释项",
        "helpdoc":"帮助文档，其中的项目可被.help获取"
    }
}
```

mod主文件读取成功后，Dice!将尝试进一步读取同名子目录(build599+)。

Mod子目录

event

以lua(build609+)或toml(build636+)形式向表event写入事件。load时读入，修改后需要`system load`应用。

lua格式

```
event.listen_friend_request = { --该事件的ID，唯一对应，同名覆盖
    title = "好友审核", --事件标题
    trigger = { --触发方式，无则只能直接通过ID调用
        hook = "FriendRequest"  --代理内置的好友申请事件
    },
    --调用script/listen_friend_request.lua
    action = { lua = "listen_friend_request" }
}
event.good_morning = {
    title = "早安",
    trigger = {
        clock = { --在每日指定时点触发，可设置多个时点
            {hour=7,minute=30}
        }
```

```
    },
    --调用script/daily/good_morning.lua
    action = { lua = "daily.good_morning" }
}
event.heartbeat = {
    title = "心跳",
    trigger = {
        cycle = { --计时循环,所有cycle事件在初始化后立刻执行第一次
            minute=5,second=30 --循环执行的间隔
        }
    },
    action = { lua = "heartbeat" }
}
```

toml格式

```
[event.listen_friend_request]
title = "好友审核"
trigger.hook = "FriendRequest"
action.lua = "listen_friend_request"

[event.good_morning]
title = "早安"
trigger.clock = [{hour=7,minute=30}]
action.lua = "daily.good_morning"

[event.heartbeat]
title = "心跳"
trigger.cycle = { minute=5,second=30 }
action.lua = "heartbeat"
```

reply

以lua(build603+)或toml(build636+)形式向表msg_reply（toml为reply）写入关键词回复。load时读入，修改后需要system load应用。

- keyword: 触发该回复的关键词，可以有多种触发形式和多个关键词。
- limit: 触发限制条件。table键值表示条件类型=条件内容。

- echo: 回复内容。值为文本时，回复该文本；值为数组时，作为牌堆回复；值为表且形如`{lua=文件名}`时，调用lua文件。

lua格式

```
msg_reply.good_morning = {   --该条msg_reply的id，唯一对应，同名覆盖
    keyword = {
        match = "早",
        prefix = {"早安","早上好"},
    },
    limit = {
        today = { user = 1 },    --每用户当日触发次数
    },
    echo = {
        lua = "reply_good_morning"   --调用文件名对应lua
    }
}
msg_reply.good_night = {
    keyword = {
        prefix = "晚安",
    },
    limit = {
        cd = { user = 60 },
        today = { user = 2 },
    },
    echo = "{reply_good_night}" --直接回复文本
}
```

toml格式

```
[reply.good_morning]
keyword = { match = "早", prefix = ["早安","早上好"] }
limit.today.user = 1
echo.lua = "reply_good_morning"

[reply.good_night]
keyword.prefix = "晚安"
limit = { cd = { user = 60 }, today = { user = 2 }}
echo = "{reply_good_night}"
```

script

script目录中的lua文件名（不含后缀）可作为loadLua函数的参数，也可直接作为关键词回复Lua形式的回复内容。推荐script文件名及文件内不出现中文字符。

script中的文件不会被预加载，而是调用时实时读取，因此热更新后不需要使用.system load加载。

```
---reply_good_morning.lua
clock_now = now = os.date("*t")
if(now.hour>12)then
    msg.hour = now.hour
    return "{reply_good_morning_late}"
elseif(now.hour<5)then
    return "{reply_good_morning_early}"
else
    return "{reply_good_morning}"
end
```

speech

(build603+)台词speech是自定义回执文本的上位，可直接由花括号转义。每项条目可存多条文本，等效于单抽放回的牌堆或{sample}。load时读入，修改后需要system load应用。

```
reply_good_morning:
 - "早上好啊{nick}"
 - "那{nick}也早安哦"
reply_good_morning_late: "{hour}点的早安？{nick}起得可真早呢"
reply_good_morning_early: "早安，{nick}的睡眠还够吗？"
reply_good_night:
 - "晚安呐"
 - "也祝{nick}晚安"
strRollDice: "{pc}掷骰：{res}"
strRollRegularSuccess:
 - 成功了哦
 - "成功 看来{self}还是护佑着{nick}的"
```

image

(build633+)图片文件夹，会在mod加载时复制到/data/image文件夹中用于在消息中发送。

audio

(build633+)语音文件夹，会在mod加载时复制到/data/record文件夹中用于发送语音。

Lua快速教程

如果你已经了解过脚本语言lua，请跳过该部分.由于function使用小括号()输入参数，下文使用中括号[]表示参数可省略

Lua基本介绍

Lua是一种用标准C语言编写并开源的嵌入式脚本语言。

Lua语句不使用分号或其它标点作结尾，不使用大括号或缩进表示作用域，使用"end"作为函数、条件判断、while循环的结束。单行注释以"--"开头，与或非运算符使用"and""or""not"。Lua字符串与数组的首位索引均为1，而不是一般编程语言的0。

Lua标识符

- 逻辑运算: 与and; 或or; 非not
- 运算符号: 赋值=; 等号==; 不等号~=; 小于号< ; 小于等于号<=; 四则运算+-*/; 取余%; 乘幂^; 取整除法//; 连接..;
- 取长度#: #table 或 #string
- 函数参数():function(args)
- 表索引键值（index）[]: table[key]
- 字符串（不可混用）:"字符串"/'字符串'/[[不转义换行等符号的字符串]]
- 单行注释 --

```
-- lua单行注释
```

- 多行注释 -[[]]

```
--[[
lua多行注释
]]
```

Lua数据类型

Lua中的变量一共有8种基本数据类型，具体类型在赋值时自动判断而不需要声明。编写Dice!脚本需要了解其中6种：空(nil)、逻辑型(boolean)、双精度浮点数(number)、字符串(string)、函数(function) 和 表(table)。使用type()函数可以返回一个表示变量类型的文本值，如type(msg)=="table"。

空值nil

nil类型只有nil一种值，表示无效值，比如一个没有赋值过的变量，访问表中不存在的key，缺失的函数返回值。函数不返回相当于返回nil，对变量赋值nil等于删除该变量。例：试图用string.match匹配字符串时，如果匹配失败，则只返回nil。nil不能视作空字符串参与运算，所以需要预先考虑变量为nil的情况。

逻辑值（布尔）boolean

boolean类型有两种值：真true，假false。所有变量都可以自动转换为逻辑值，非空变量均视为真，空变量(nil)视为假。

数值number

lua中所有的数都可视为双精度浮点数(double)。

string与number进行算术运算时（如"6"*5），若string可转换为数字，将进行自动类型转换，否则报错；string与number进行取相等运算时自动为false（如"123"~=123）；string与number用不等号比较时必定报错（如"6">5）

文本值（字符串）string

Lua中的string可以用三种方式表示:"双引号"、'单引号'、[[双层中括号]]，其中双层中括号内引号、换行符等特殊字符不需要转义。string不能使用加减符号，可以通过".."连接。在string前加"#"表示该字符串的长度。

- **str1..str2**

  连接字符串str1与str2，若连接到数字，将数字自动保留6位小数打印。
- **string.match(str, pattern[, init])**

  从str的init位置起，寻找正则匹配pattern的子串，成功返回匹配到的string，失败返回nil。init默认为1。

  例:`string.match(msg.fromMsg,"%d*",#order_name+1)`从指令名的后一位开始，从消息中匹配数字。
- **string.len(str)**

  返回str的长度，等效于`#str`。
- **string.format(...)**

  将特定类型变量按格式转化为string。

  例: `string.format("%.0f",die)`将掷骰结果按保留0位小数的浮点数打印。
- **string.sub(str, init [, end])**

  从str中截取从init位置到end的子串，end默认-1（截取到最后）

例:`string.sub(msg.fromMsg,#order_name+1)`截取消息从指令名后一位开始的文本

- **string.find (str, substr, [init, [end]])**

  在str从init到end的位置间从左至右寻找匹配substr的子串。如匹配成功，返回str中子串的首位，否则返回nil。
- **string.upper(str)** 将str中所有字母全部大写。
- **string.lower(str)** 将str中所有字母全部小写。
- **string.rep(str, n)** 返回将str重复n遍后连接的string。

  例:`string.rep("木大",8)`返回"木大木大木大木大木大木大木大木大"。

表table

table是存储变量的容器(关联数组)，可以通过不同的索引访问对应的值，格式为`table[key]`。

当table的键值为纯ANSI字符时（非纯数字、无中文等多字节字符），可用'.'代替[""]格式，如`msg.fromQQ`等价于`msg["fromQQ"]`，但不能用`msg_order.抱Shiki`来代替`msg_order["抱Shiki"]`

- **table.concat(tab[, sep])**

  将表tab中所有元素作为string连接，使用sep分隔。

  例:`table.concat(die_pool,'+')`将所有骰目通过加号连接。
- **table.insert(tab, [pos,]value)**

  在table的pos位置插入value，默认插入在末尾。

  例:`table.insert(die_pool,die)`将本次骰目加入结果池中
- **table.sort (tab [, comp])**

  将tab按comp函数进行排序，默认进行升序排序。

  例: `table.sort(die_pool,function(a,b) return (a>b) end)`当需要为掷骰结果取最大的若干个时，就需要预先为骰目降序排序。

```
--初始化空表
msg_order = {}
--向表中插入键值对
msg_order[".duel"] = mdice_duel
```

函数function

函数在定义时需要声明function 函数名(参数列表)，使用return返回，使用end作为函数体的结尾。函数不需要规定返回值的类型和数量，甚至可以在不同的位置返回不同的值。返回多个值时，值之间以','分隔。不要在return的下一行接一般语句。

```
function max_min(a,b)
    if(a>b)then
        return a,b
    else
        return b,a
    end
end
```

Lua流程控制

if()then...elseif()then..else...end

if(语句为真)then [执行语句] end。可使用多个elseif()then来进行判定语句为假时的后续判定。

注意：elseif与else if是两种格式，后者表示else作用域内额外嵌套一层if结构，因此比前者额外多一个end。

```
if(favor < 20)then
    return "{nick}的好感度还不够哟"
elseif(favor < 60)then
    return "只给{nick}一下哦~就一下"
else
    return "那就随{nick}的便啦"
end
```

for循环(数值)

```
for i=1, cnt_dice do    --表示i初始值为1，每次循环+1，大于cnt_dice时停止循环，相当于执行cnt_dice次
    local dice_point = ranint(1,6)
    table.insert(dice_pool,dice_point)
    sum = sum + dice_point
end
```

while()do 循环

```
while(cnt_dice > 0) do  --条件为真则继续循环，相当于执行cnt_dice次
    local dice_point = ranint(1,6)
    table.insert(dice_pool,dice_point)
    sum = sum + dice_point
    cnt_dice = cnt_dice - 1
end
```

repeat...until 循环

```
repeat  --第一次无条件执行，相当于执行cnt_dice次但至少为1次
    local dice_point = ranint(1,6)
    table.insert(dice_pool,dice_point)
    sum = sum + dice_point
    cnt_dice = cnt_dice - 1
until( cnt_dice < 1 )   --条件为真则跳出循环
```

好了，现在你已经了解了lua的全部基本语法，可以自己动手自定义几乎任何跑团规则的掷骰指令，或是自制牌堆/好感互动指令了~

前缀指令脚本

Dice!在收到消息时，将先匹配基础指令(.authorize/.dismiss/.warning/.master/.bot/.helpdoc/.help)，然后匹配自定义指令(Type=Order类reply)，之后再执行内建指令，再匹配自定义回复(Type=Reply)。（因此自定义指令可用于替换原有指

令）根据匹配到的前缀，Dice!将取到加载时注册的脚本文件和指令函数名，执行对应函数，并将消息未匹配的部分存为`msg.suffix`。

指令函数

指令函数的参数msg可看做一张记录消息语境的表（实际为Lua类型userdata，以Dice定义的Context为元表），msg.fromMsg、msg.uid、msg.gid分别表示消息文本、来源群、来源用户。

指令函数的返回值可以有0到2个，如有，第一个返回值表示直接回复的语句，第二个返回值表示私聊回复的语句（用于暗骰/暗抽/暗检定，会同时发送给ob用户）。由于Dice!在发送消息时会将换页符'\f'视为消息分段发送，实际上可以通过一个返回值返回多段回复。

指令注册

lua脚本读取目录为[存档目录]/plugin/（plugin文件夹不会自动创建）

在启动或.system load时，Dice!会遍历plugin目录下的所有lua文件，读取msg_order表中的键值对。当消息文本前缀匹配到触发词时，Dice!将读取记录的文件名并调用指定函数。因此，可在运行期间实时修改脚本文件，**只有增删文件、修改触发词或函数名需要.system load**，只修改其他内容不需要重新加载，注意修改后保存即可。

脚本调试

作为一种脚本语言，lua的代码错误要到运行报错时才暴露。但lua虚拟机的报错并不会终结掉宿主C++程序，且Dice!会将Lua打印的报错信息发送到通知窗口。脚本可能在两个时点报错：**加载脚本文件**时和**运行指令函数**时。由于指令注册时会将整个lua文件作为脚本运行，因此基本的语法错误会导致此时立即报错，如多了个end，少了个括号，return之后没有end。通过在Lua在线运行工具上试跑脚本，可以提前发现第一类错误。但由于此时并不会真正执行函数体，所以更多更复杂的错误要在函数执行时报错（几乎全部是函数参数类型不正确），出现后需要第一时间热修复。

扩展说明

如果看不懂这部分内容，可以略过

指令注册时会将整个lua文件运行一遍。如果担心函数体过于复杂影响读取，建议将主体放置到plugin的子文件夹，使用loadLua调用。指令注册时不会递归遍历子文件夹，因此可以降低不必要的开销。例如，在Lua实现了Maid角色卡模板功能，保存在PC文件夹下Maid.lua，这样其他脚本可使用loadLua("PC/Maid")重复调用。

样例模板

```
--声明表msg_order,初始化时会遍历plugin文件，读取msg_order表中的指令名
--msg_order中的键值对表示 前缀指令->函数名
msg_order = {}
--声明触发词，允许多个触发词对应一个函数
order_word = "触发词"
order_word_1 = "触发词1"
order_word_2 = "触发词2"
--声明指令函数主体，函数名可自定义
function custom_order(msg)
    return "回复语句"
end
msg_order[order_word] = "custom_order"  --value为字符串格式且与指令函数名一致
msg_order[order_word_1] = "custom_order"
msg_order[order_word_2] = "custom_order"
--注意：本手册所提供脚本样例并非固定，仅基于模板化考虑选定该格式。
```

定制任务脚本

关于plugin的通用机制参见指令脚本部分

Dice!在当日时间（时:分）达到预设的时点，会触发定时任务（.admin clock设置）。根据设置中的任务名，Dice!将取到加载时注册的脚本文件和任务函数名，执行对应函数。

任务函数

任务函数没有输出参数，返回值也没有意义。定时任务可以用于某些扩展功能定时发送消息或每日重置状态。

任务注册

在启动或.system load时，Dice!会遍历plugin目录下的所有lua文件，读取task_call表中的键值对。当定时任务匹配到任务名时，Dice!将读取记录的文件名并调用指定函数。

样例模板

```
--声明表task_call,初始化时会遍历plugin文件，读取task_call表中的指令名
--task_call中的键值对表示 任务名->函数名
task_call = {}
--声明任务名，允许多个任务名对应一个函数
task_name = "任务名"
--声明任务函数主体，函数名可自定义
function custom_task()
    --函数执行内容
end
task_call[task_name] = "custom_task"    --value为字符串格式且与任务函数名一致
--注意：本手册所提供脚本样例并非固定，仅基于模板化考虑选定该格式。
```

开发说明

lua文件的字符编码问题

Windows系统一般使用GBK字符集。Dice!支持utf-8及GBK两种字符集的lua文件，在读写字符串时将自动检测utf-8编码并转换。而出现以下情况时，编码并非二者皆可：

- lua文件相互调用或读写其他文本文件，且字符串含有非ASCII字符时，**关联文件字符集应保持一致**；

- lua文件中需要调用http函数时，**应当与目标网页的编码一致（基本是UTF8）**
- lua文件使用require或os等以文件名为参数的函数，且路径含有非ASCII字符时，**必须使用GBK**；

附录：Dice!预置的lua元表

Context

get(self, field, defaultVal)

取语境中变量，如#3省略，则`context:get(field)`可等效于`context[field]`。

输入参数	变量类型	说明
语境	Context	形如context:get()可省略#1
变量字段	string	字段会先进行一次转义
候补值	任意	如未定义该变量则返回该值

返回值类型	说明
任意	待取变量

format(self, rawString)

在该语境下转义文本。

输入参数	变量类型	说明
语境	Context	形如msg:format()可省略#1
待转义文本	string	

返回值类型	说明
string	转义后文本

echo(self, replyMsg, isRaw)

回复消息，有来源聊天窗口的事件也可以用`event:echo(replyMsg)`。

输入参数	变量类型	说明
语境	Context	形如msg:echo()可省略#1
待回复消息	string	
是否禁用转义	boolean	可省略，默认转义

inc(self, field, val)

自增变量，仅对数字有效。

输入参数	变量类型	说明
语境	Context	形如context:set()可省略#1
变量字段	string	字段会先进行一次转义
自增量	number	可省略，表示+1

附录：Dice!预置的lua函数

调用前注意Dice版本是否匹配！Dice!2.6.6总计预置19条全局函数，4条http函数，4条context方法。

类型为number的参数，一般也可传入可数字化的字符串，如msg.fromGroup.

log(info[,notice_level])

(build598+)发送日志

输入参数	变量类型	说明
日志内容	string	待输出日志内容
通知窗口级别	number	选填，若空则只输出到框架日志界面

loadLua(scriptName)

运行Lua文件，返回目标脚本的返回值

参数使用相对路径且无后缀，根目录为plugin文件夹(build575+)或mod内script文件夹(build575+)

与Lua自带的require函数不同，目标文件定义的变量会保持生命周期

```
loadLua("PC/COC7")
```

输入参数	变量类型	说明
lua文件名	string	待调用mod/script/文件或plugin/文件

返回值类型	说明
同文件内返回值类型	执行指定文件后的返回值

ranint(low,high)

取随机数

输入参数	变量类型	说明
随机区间下限	number	整数
随机区间上限	number	整数
返回值类型	说明	
number	生成随机数	

getDiceQQ()

取Dice账号

返回值类型	说明
string	取骰子自身账号

getDiceDir()

取Dice存档目录，用于自行读写文件

返回值类型	说明
string	取Dice存档目录

eventMsg(msg, gid, uid)

虚构一条消息进行处理，不计入指令频度。可使用参数列表`eventMsg(msg, gid, uid)`或(build608+)参数包形式`eventMsg(pkg)`.

```
eventMsg(".rc Rider Kick:70 踢西鹿", msg.gid, msg.uid)
eventMsg({
        fromMsg = ".rc Rider Kick:70 踢西鹿",
        gid = msg.gid,
        uid = msg.uid,
})
```

输入参数/pkg子项	变量类型	说明
消息文本/fromMsg	string	
来源群/gid	number	可以为空
发送者/uid	number	

sendMsg发送消息

可使用参数列表`sendMsg(msg, gid, uid)`或(build619+)参数包形式`sendMsg(pkg)`发送.

```
sendMsg("早安哟", msg.fromGroup, msg.fromQQ)
```

输入参数/pkg子项	变量类型	说明
fwdMsg	string	待发送消息
gid	number	私聊时为空
uid	number	群聊时可以为空
chid	number	频道id，仅参数包可用

getUserToday(userID, keyConf, defaultVal)

取用户今日数据项。特别地，配置项为"jrrp"时，所取值同`.jrrp`结果。所有当日数据会在系统时间24时清空。

```
getUserToday(msg.uid, "jrrp")
```

输入参数	变量类型	说明
用户账号	number	
配置项	string	待取配置项
候补值	任意	配置项不存在时返回该值，为空则返回0

返回值类型	说明
任意	待取值

setUserToday(userID, keyConf, val)

存用户今日数据项

输入参数	变量类型	说明
用户账号	number	
配置项	string	待存配置项
配置值	任意	待存入数据

getUserConf(userID, keyConf, defaultVal)

取用户配置，配置项带*标记表示会另行计算而非调用存储数据。(build613+)参数1可以为空，此时遍历所有**记录了该属性**的用户并返回以账号=属性值为键值对的table。

```
getUserConf(msg.fromQQ, "favor", 0)
getUserConf(nil, "favor") --返回所有用户的favor列表
```

输入参数	变量类型	说明
用户账号	number	
配置项	string	待取配置项
候补值	任意	配置项不存在时返回该值

返回值类型	说明
任意	待取值

特殊配置项	说明
trust	用户信任（仅4以下可编辑）
firstCreate	用户记录创建（初次使用）时间 [时间戳，秒]
lastUpdate	用户记录最后更新时间 [时间戳，秒]
name*	用户账号昵称（只读）
nick*	全局称呼（备取账号昵称）
nick#`群号`*	特定群内的称呼（备取群名片->全局称呼->账号昵称）
nn*	全局nn
nn#`群号`*	特定群内的nn

setUserConf(userID, keyConf, val)

存用户配置项

输入参数	变量类型	说明
用户账号	number	
配置项	string	待存配置项
配置值	任意	待存入数据

getGroupConf(groupID, keyConf, defaultVal)

取群配置，配置项带*标记表示会另行计算而非调用存储数据。(build613+)
群号可以为空，此时遍历所有**记录了该属性**的群并返回以群号=属性值为键值对的table。

```
getGroupConf(msg.fromQQ, "rc房规", 0)
```

输入参数	变量类型	说明
群号	number	
配置项	string	待取配置项
候补值	任意	配置项不存在时返回该值

返回值类型	说明
任意	待取值

特殊配置项	说明
name*	群名称（只读）
size*	群人数（只读）
maxsize*	群规模（只读）
firstCreate	用户记录创建（初次使用）时间 [时间戳，秒]
lastUpdate	用户记录最后更新时间 [时间戳，秒]
members	群用户列表
admins	群管理列表
card#`群员账号`*	群名片
auth#`群员账号`*	群权限（只读） 1-群员;2-管理;3-群主
lst#`群员账号`*	最后发言时间（只读） [时间戳，秒]

setGroupConf(groupID, keyConf, val)

存群配置项

输入参数	变量类型	说明
群号	number	
配置项	string	待存配置项
配置值	任意	待存入数据

getPlayerCard(userID, groupID)

取指定群内绑定的角色卡（整张）

输入参数	变量类型	说明
用户账号	number	
群号	string	为空或未绑定则取默认卡

getPlayerCardAttr(userID, groupID, keyAttr, defaultVal)

取角色卡属性

```
getPlayerCardAttr(msg.fromQQ, msg.fromGroup, "理智",
val_default)
```

输入参数	变量类型	说明
用户账号	number	
群号	number	
属性名	string	待取属性
候补值	任意	属性不存在时返回该值

返回值类型	说明
任意	待取属性

setPlayerCardAttr(userID, groupID, keyConf, val)

存角色卡属性

输入参数	变量类型	说明
用户账号	number	
群号	number	
属性名	string	待存属性
属性值	任意	待存数据

mkDirs(pathDir)

输入参数	变量类型	说明
文件夹路径	string	递归创建该文件夹

sleepTime(ms)

输入参数	变量类型	说明
等待毫秒数	number	

http

(build590+)

http.get

输入参数	变量类型	说明
待访问url	string	若含须转义字符须用urlEncode转义

返回值	变量类型	说明
连接是否成功	boolean	
网页返回内容	string	访问失败则返回错误原因

http.post

输入参数	变量类型	说明
待访问url	string	若含须转义字符须用urlEncode转义
post数据	string	(build634+)如为table则自动序列化为json格式
headerstring		(build606+)可省略，默认Content-Type: application/json，(build629+)如为table将自动拼接

返回值	变量类型	说明
连接是否成功	boolean	
网页返回内容	string	访问失败则返回错误原因

http.urlEncode

将url中须转义的字符进行转义。

输入参数	变量类型	说明
待编码url	string	

返回值	变量类型	说明
编码后url	string	

http.urlDecode

还原url中转义的字符。

输入参数	变量类型	说明
待解码url	string	
返回值	变量类型	说明
解码后url	string	

附录：自定义指令常用的正则匹配

解析参数时，可使用string.match的第三个参数来略过指令前缀部分，从指令长度+1的位置开始匹配。注意单个汉字的长度大于1且受字符编码影响，不提倡手动数中文指令长度。

```
--前缀匹配且指令参数为消息余下部分时，去除前后两端的空格
local rest = string.match(msg.suffix,"^[%s]*(.-)[%s]*$")

--指令参数为单项整数时，直接用%d+表示匹配一个或多个数字，未输入数字时匹配失败返回nil
local cnt = string.match(msg.fromMsg,"%d+")

--同上，%d*表示匹配0或任意个数字，未输入数字时匹配成功返回空字符串""
--该匹配模式需要确保数字之前的其他字符已被排除
local cnt = string.match(msg.fromMsg,"%d*")

--参数使用空格分隔且不限数目，遍历匹配
local item,rest = "",string.match(msg.fromMsg,"^[%s]*(.-)[%s]*$",#order_select_name+1)
if(rest == "")then
    return "请输入参数"
end
local items = {}
repeat
    item,rest = string.match(rest,"^([^%s]*)[%s]*(.-)$")
    table.insert(items, item)
until(rest=="")
```

附录：lua报错信息说明

```
module '%s' not found:
    no field package.preload['%s']
    no file
-- 没有把被引用的lua或dll文件放在指定位置（多见于require与loadLua）
-- 解决方式：把所需文件放入Dice存档目录/plugin/或Diceki/lua/，dll文件或require对象必须置于后者
attempt to call a nil value (global '%s')
-- 将空变量%s用作函数（global表示被调用的是全局变量，local表示本地变量，method表示索引方法）
attempt to index a nil value (global '%s')
-- 对空变量%s使用索引（只有table等结构可以索引，形如msg.fromMsg）
attempt to concatenate a nil value (local '%s')
-- 使用..连接字符串时连接了一个空变量%s
bad argument #1 to '%s' (string expected, got nil)
-- 函数%s的第1个参数类型错误，要求类型为string，但实际传入的参数为nil。特别地，got nil表示输入参数为nil或缺少参数
value expected
-- 要求参数，但没有传入
attempt to perform arithmetic on a nil value (global '%s')
-- 将一个空变量%s当做数字表示
bad argument #5 to 'format'(number has no integer representation)
-- 函数format的第5个参数类型错误，要求是整数，但传入是小数，或者是其余类型不能化为整数
'}' expected (to close '{' at line 169) near '%s'
-- 脚本第169行的左花括号缺少配对的右花括号。此错误也可以由表格内缺少逗号分隔、括号外的中文等原因造成
'end' expected (to close 'function' at line 240) near <eof>
-- 脚本第240行的function缺少收尾的end，<eof>表示文件结束（找到文件末也没找到）
'then' expected near 'end'
-- if then end逻辑结构缺少then
unexpected symbol near '%s'
-- 符号%s边有无法识读的符号，比如中文字符
attempt to get length of a nil value (local 'tab')
-- 对空变量tab作取长度运算（#）
```

```
attempt to add a 'string' with a 'string'
-- 对(不能化为数字的)字符串用加法'+'（字符串只能用连接'..'）
attempt to compare number with string
-- 对数字和(不能化为数字的)字符串用比较运算符
error loading module '%s' from file
-- 使用require "%s"时加载出错
no visible label '%s' for <goto> at line
-- 在循环结构中跳转不存在的节点
invalid option '%s'
-- 传入的参数不是string或不在给定的字符串列表中
malformed number near '86400..'
-- 数字与字符连接符粘连导致误判为小数点，使得格式解析错误，需要插入空格
```

附录：常用语法在线校验器

- json: 在线JSON校验格式化工具
- yaml: YAML、YML在线编辑器(格式化校验)
- lua: 在线运行Lua
- toml: 在线TOML转YAML工具

附录：事件event样例

代理事件

时点	说明
StartUp	启动完成后
FriendRequest	好友申请后，处理前
FriendAdd	自动或被同意添加好友后，回复前
GroupRequest	受群邀请后，处理前
GroupKicked	被踢出群后，拉黑前
GroupBanned	被禁言后，拉黑前
GroupAuthorize	群申请许可后，处理前
LogEnd	日志完成后，上传前
WhisperIgnored	私聊不识别为指令且未触发回复
DayEnd	（时差修正后）0点后刷新每日数据前
DayNew	刷新每日数据后

被踢出群不含解散，被禁言不含全群禁言

GroupAuthorize、LogEnd、WhisperIgnored时点在消息处理时，对应Event为Message，其余事件Event为其自身

event参数	说明
uid	当事用户
gid	当事群
fromMsg	消息文本/好友申请文本
aimGroup	GroupAuthorize中待审核群
AttachInfo	GroupAuthorize中申请文本
log_file/log_path	LogEnd中日志文件名及路径
blocked	[回传]是否拦截该事件后续处理
approval	[回传]是否同意申请
ignored	[回传]是否忽略算入指令计数

```
--event/hook.lua
event.listen_startup = {
    title = "启动完成",
    trigger = {
        hook = "StartUp"
    },
    action = { lua = "listen_startup" }
}
event.listen_friend_request = {
    title = "好友申请处理",
    trigger = {
        hook = "FriendRequest"
    },
    action = { lua = "listen_friend_request" }
}
event.listen_friend_add = {
    title = "好友添加事件",
    trigger = {
        hook = "FriendAdd" --主动通过好友申请时不触发
    },
    action = { lua = "listen_friend_add" }
}
```

代理好友申请

```
--script/listen_friend_requst.lua
answer = event.fromMsg
--如果验证方式为回答问题，则需要截去问题部分
-answer = string.sub(answer,string.find(answer,"答
案：")+#"答案：")
true_answer = "正确回案"
if string.find(answer,"true_answer") then
    log("收到"..getUserConf(event.uid,"name").."
("..event.fromUser.."的好友请求:\n"..answer.."\n回答通过√",1)
    event.approval = true --通过申请
else
    log("收到"..getUserConf(event.uid,"name").."
("..event.fromUser.."的好友请求:\n"..answer.."\n回答错误×",1)
    event.approval = false
end
event.blocked = true --终止连锁，不执行原生申请处理流程
```

代理新增好友

```
--script/listen_friend_add.lua
answer = event.fromMsg
--如果验证方式为回答问题，则需要截去问题部分
-answer = string.sub(answer,string.find(answer,"答
案：")+#"答案：")
true_answer = "正确回案"
if string.find(answer,"true_answer") then
    log("收到"..getUserConf(event.fromUser,"name").."
("..event.fromUser.."的好友请求:\n"..answer.."\n回答通过√",1)
    event.approval = true --通过申请
else
    log("收到"..getUserConf(event.fromUser,"name").."
("..event.fromUser.."的好友请求:\n"..answer.."\n回答错误×",1)
    event.approval = false
end
event.blocked = true --终止连锁，不执行原生申请处理流程
```

代理公骰审核

```
--script/listen_group_authorize.lua
answer = event.attachText
--如果验证方式为回答问题，则需要截去问题部分
-answer = string.sub(answer,string.find(answer,"答
案：")+#"答案：")
true_answer = "正确回案"
if string.find(answer,"true_answer") then
    log("收到"..getUserConf(event.fromUser,"name").."
("..event.fromUser.."的好友请求:\n"..answer.."\n回答通过√",1)
    event.approval = true --通过申请
else
    log("收到"..getUserConf(event.fromUser,"name").."
("..event.fromUser.."的好友请求:\n"..answer.."\n回答错误×",1)
    event.approval = false
end
event.blocked = true --终止连锁，不执行原生申请处理流程
```

附录：DND规则.rdc指令

.rdc(B/P) (+/-[加值]) ([检定理由]) ([成功阈值])
参数[优/劣势骰]:可选，B=2D20取大，P=2D20取小
参数[加值]:可选，加值最后修正到D20结果上
参数[检定理由]:可选，将显示在回执语句中
参数[成功阈值]:可选，将与掷骰结果比较，返回成功或失败
出于及时向用户提供指令帮助的考虑，空参将返回帮助

```
msg_order = {}

function intostring(num)
    if(num==nil)then
        return ""
    end
    return string.format("%.0f",num)
end

--DND
```

```
function roll_d20(msg)
    local rest=string.match(msg.fromMsg,"^[%s]*(.-)
[%s]*$", 5)
    if(rest=="")then
        return [[
D20检定:.rdc
.rdc(B/P) (+/-[加值]) ([检定理由]) ([成功阈值])
参数[优/劣势骰]:可选，B=2D20取大，P=2D20取小
参数[加值]:可选，加值最后修正到D20结果上
参数[检定理由]:可选，将显示在回执语句中
参数[成功阈值]:可选，将与掷骰结果比较，返回成功或失败
空参返回本段说明文字]]
    end
    local bonus,sign,rest = string.match(rest,"^[%s]*
([bBpP]?)[%s]*([+-]?)(.*)$")
    local addvalue = 0
    local modify = ""
    --加减符号判断有无加值
    if(sign ~= "")then
        modify,rest = string.match(rest,"^([%d]+)[%s]*
(.*)$")
        if(modify==nil or modify=="")
        then
            return "请{pc}输入正确的加值"
        end
        addvalue = tonumber(modify)
        if(sign=='-')
        then
            addvalue = addvalue*-1;
        end
    end
    local skill,dc = string.match(rest,"^[%s]*(.-)[%s]*
([%d]*)$")
    local dc = tonumber(dc)
    if(dc~=nil)then
        if(dc<1)then
            return "这你要怎么才能失败啊？"
        elseif(dc>99)then
            return "这你要怎么才能成功啊？"
        end
```

```
    end
    local die = ranint(1,20)
    local selected = die
    local res
    --考虑优势骰/劣势骰
    if(bonus=="")then

 res="D20"..sign..modify.."="..intostring(die)..sign..modi
fy
    else
        local another=ranint(1,20)
        bonus=string.upper(bonus)
        if((another>die and bonus=="B")or(another<die and
bonus=="P"))then
            selected=another
        end
        res=bonus.."
("..intostring(die)..','..intostring(another)..")"..sign..
modify.."->"..intostring(selected)..sign..modify
    end
    local strReply = "{pc}进行"..skill.."检定:"..res
    local final=selected+addvalue
    if(addvalue~=0)then
        strReply=strReply.."="..intostring(final)
    end
    if(dc=="")then
        --对成功与否不做判定
    elseif(final>dc)
    then
        strReply=strReply..'>'..intostring(dc).."
{strRollRegularSuccess}"
    elseif(final<dc)
    then
        strReply=strReply..'<'..intostring(dc).."
{strRollFailure}"
    else
        strReply=strReply..'='..intostring(dc).."
{strRollRegularSuccess}"
    end
    return strReply
```

```
end
msg_order[".rdc"] = "roll_d20"
```

附录：ShadowRun规则.rsr指令

ShadowRun掷骰规则：掷X粒六面骰，点数为5或6记一次命中，过半数骰目出1记失误。自定义指令格式.rsr X。

```
msg_order = {}

function intostring(num)
    if(num==nil)then
        return ""
    end
    return string.format("%.0f",num)
end

--shadowrun
function roll_shadowrun(msg)
    local cnt =
tonumber(string.match(msg.fromMsg,"%d+",5))
    if(not cnt)
    then
        return "shadowrun检定:.rsr[骰数] 出5/6视为命中\n出1过半
视为失误"
    end
    local res="{pc}掷骰"..cnt.."D6: "
    cnt = tonumber(cnt)
    if(cnt<1)
    then
        return "{strZeroDiceErr}"
    elseif(cnt>100)
    then
        return "{strDiceTooBigErr}"
    else
        local cntSuc=0
        local cntFail=0
        local pool={}
        local glitch=""
```

```
        for times=1,cnt do
            local die=ranint(1,6)
            if(die>=5)then
                cntSuc = cntSuc + 1
            elseif(die==1)then
                cntFail = cntFail + 1
            end
            table.insert(pool,intostring(die))
        end
        if(cntFail > cnt/2)then
            glitch = " {strGlitch}"
        end
        local reply=res..table.concat(pool,'+')..">"..intostring(cntSuc)..glitch
        return reply
    end
end
msg_order[".rsr"] = "roll_shadowrun"
```

附录：仿MDice的.duel指令

```
msg_order = {}

function intostring(num)
    if(num==nil)then
        return ""
    end
    return string.format("%.0f",num)
end

--惠惠给每个得分都写了专属回复，这里作为样例仅使用随机选取
reply_duel_win = {
    "不服？不服你就正面赢我啊",
    "你请我吃顿饭这次就算你赢，怎么样？",
    "哦吼？我觉得我甚至可以让你一个骰子",
    "（面露笑意地看着你）",
    "你觉得能赢我，嘿嘿，那是个幻觉，美丽的幻觉。",
    "再试也没用啦，会输一次就会输一百次哦",
}
```

```
reply_duel_lose = {
    "？！居然趁我一时大意。。。"
}
reply_duel_even = {
    "哼哼，平分秋色嘛"
}

function table_draw(tab)
    return tab[ranint(1,#tab)]
end

--惠系决斗
function mdice_duel(msg)
    local isDark=string.sub(msg.fromMsg,1,5)==".dark"
    local poolSelf, poolPlayer = {},{}
    local resSelf, resPlayer, resTotal, resEval =
"","","",""
    local sumSelf, sumPlayer, sumTotal=0,0,0
    for times=1, 5 do
        local die = ranint(1,100)
        sumSelf = sumSelf + die
        table.insert(poolSelf,intostring(die))
    end
    resSelf="\n{self}\n"..table.concat(poolSelf,'、')..."\n
打点"..intostring(sumSelf)
    for times=1, 5 do
        local die = ranint(1,100)
        sumPlayer = sumPlayer + die
        table.insert(poolPlayer,intostring(die))
    end

 resPlayer="\n{pc}\n"..table.concat(poolPlayer,'、')..."\n打
点"..intostring(sumPlayer)
    sumTotal = sumPlayer-sumSelf;
    resTotal="\n———\n
计"..intostring(sumTotal).."/10="..intostring(sumTotal/10).
."分"
    if(sumSelf>sumPlayer)then
        if(isDark)then
            resEval = "发起黑暗决斗的你，应该做好觉悟了吧？"
```

```
            --好孩子不要用 eventMsg(".group ".. msg.fromGroup
.." ban ".. msg.fromQQ .." "..intostring(sumSelf), 0,
getDiceQQ())
        else
            resEval = table_draw(reply_duel_win)
        end
    elseif(sumSelf<sumPlayer)then
        resEval = table_draw(reply_duel_lose)
    else
        resEval = table_draw(reply_duel_even)
    end
    local reply = "看起来{pc}想向我发起决斗，呼呼，那我就接下了\n
胜负的结果是——"..resSelf..resPlayer..resTotal.."\n"..resEval
    return reply --如有返回值表示回复文本，如有第二返回值表示暗骰文
本
end
msg_order[".dark duel"] = "mdice_duel"
msg_order[".duel"] = "mdice_duel"
--指令注册格式 key->指令前缀 value->脚本内函数名
```

附录：仿SitaNya的.clock指令

```
msg_order = {}

function intostring(num)
    if(num==nil)then
        return ""
    end
    return string.format("%.0f",num)
end

function print_chat(msg)
    if(msg.fromGroup == "0")then
        return "qq "..msg.fromQQ
    end
    return "group "..msg.fromGroup
end

function Sleep(n)
```

```
    --575+实现方式
    if n > 0 then sleepTime(n*1000) end
end

function sitanya_clock(msg)
    local seconds = string.match(msg.fromMsg,"%d+",7)
    if(not seconds)then
        return [[clock    计时器
出于技术验证仿塔骰写的指令，请勿滥用该指令。
用法：.clock [秒]
示例：.clock 60]]
    end
    if(tonumber(seconds)>600)then
        return "设置{self}定时失败:设置时长超过10分钟×"
    end
    eventMsg(".send "..print_chat(msg).." 您的定时器:<"..
seconds .."秒已开启计时",0,getDiceQQ())
    Sleep(seconds)
    return "{nick}的定时器:<".. seconds .."秒已到期"
end
msg_order[".clock"] = "sitanya_clock"
```