井通科技

智能合约文档

v1.00

目录

1. 概述

井通智能合约以Lua为实现语言，贴合C++实现小型化、配置化和轻量化的智能合约引擎，同时又对系统的改动不大，以及能够保持在C++中实现lua的高效运行。

井通智能合约支持两个智能合约交易，分别是智能合约部署交易和智能合约调用交易。

智能合约部署的curl命令如下：

```
curl -X POST -d '{"method": "submit", "params": [ { "secret":
"snoPBjXtMeMyMHUVTgbuqAfg1SUTb", "tx_json": { "TransactionType":
"ConfigContract", "Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh", "Method":
0, "Amount": "100000000", "Payload":
"726573756c743d7b7d3b202066756e6374696f6e20496e697428742920726573756c743d73634
765744163636f756e74496e666f28742920726573756c742020656e643b20206
6756e6374696f6e20666f6f28742920613d7b7d20726573756c743d736347657441636f756e744
96e666f28742920726573756c742020656e64" } } ]}' 127.0.0.1:5050
```

其中，各个参数含义如下：

参数	含义	备注
TransactionType	交易方法	合约交易固定为ConfigContract
Account	合约发起账号	
Method	合约方法	0为合约部署，1为合约调用
Amount	合约激活SWT	激活合约使用，至少是Reserved SWT金额
Payload	合约代码	合约代码十六进制格式。

合约部署成功，返回一个合约地址，此合约地址即是后续进行合约调用的目标地址。例如：

"ContractState":"**jGjzq74AT9ZLgnduGz4EmKutudpJyJBrZW**"

其中，**jGjzq74AT9ZLgnduGz4EmKutudpJyJBrZW** 为合约地址。

智能合约调用的curl命令如下：

```
curl -X POST -d '{ "method": "submit",  "params": [ { "secret":
"snoPBjXtMeMyMHUVTgbuqAfg1SUTb",  "tx_json": { "TransactionType":
"ConfigContract",  "Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",  "Method":
1,  "ContractMethod": "666f6f",  "Destination":
"jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",  "Args": [ { "Arg": { "Parameter":
"6a486239434a41577942346a7239315652576e3936446b756b473462776474795468" }  }  ] }
} ]}' 127.0.0.1:5050
```

其中，各个参数含义如下：

参数	含义	备注
TransactionType	交易方法	合约交易固定为ConfigContract
Account	交易发起账号	
Method	合约方法	0为合约部署，1为合约调用
ContractMethod	合约方法	调用合约对应调用的合约方法
Destination	调用目标	调用的合约地址
Args	合约参数	合约参数是个对象数组，每个数组对象为一个Arg对象；合约参数为合约方法的参数

合约payload字段说明如下：

```
result={};
function Init(t)
   result = scGetAccountInfo(t);
   return result;
end;
function foo(t)
      result = scGetAccountInfo(t);
      return result;
end;
```

合约脚本中必须包含Init函数，合约执行需要存储的数据在该函数中完成。合约中result={}是固定的，必须包含。合约返回值交由result返回，result中包含两个字段:'state'和'res'。State是bool类型表示接口调用是否成功，'res'是string类型，存储的是接口调用结果或者错误信息。Init函数和foo函数接收参数t是一个lua

语言table类型。用户传递的参数都在这个table中，在lua代码中获取方式如value=t[‘0’]。注：在编写lua代码时不要有换行回车。

井通系统合约引擎对于合约函数的调用，通过lua的talble类型进行参数传递，table中参数顺序为：

1) 按顺序压入用户传入的参数，用户参数在table的key编号从‘0’开始。
2) 最后压入固定的5个参数，分别是合约账号地址(table中对应的key为‘TDstAccountID’)、交易引擎指针(table中对应的key为‘TEnginePointer’)、交易指针(table中对应的key为‘TTxn’)，交易引擎校验值(table中对应的key为‘TEnginePointerHash’)，交易指针校验值(table中对应的key为‘TTxnHash’)。

2. 合约库函数规范

所有合约库函数都以sc开头，sc是smart contract的缩写。信息获取以scGet+获取信息，如获取账号信息函数命名应该是：scGetAccountInfo。功能操作类库函数命名是sc+功能名称，如支付功能：scPayment。

所有合约库函数函数参数都只能是 lua_State *L，返回值只能是 static int，该返回值表示的传出参数的个数。传出参数是一个 table 类型的值，table 中包含两个成员：table['state']和 table['res']。table['state']是一个是 boolean 值，表示库函数执行是否成功，1 表示成功，0 表示失败。table['res']为 string 类型，库函数执行成功返回的是库函数执行结果，失败返回的是错误原因。这个结果记录在交易请求的 contract state 里面。

3. 收费机制

系统以 SWT 为基础的手续费收取，目前收费机制是按照 lua 脚本的字节长度和调用 lua 操作码的难以程度进行双重收费。Lua 脚本长度收费标准：每个字节收取十万分之一 SWT 的费用；lua 操作码的收费标准如下(收费单位是百万分之一 SWT)：

标号	操作码	操作码含义	收费
1	OP_MOVE	R(A) := R(B)	1

2	OP_LOADK	LOADK 将 Bx 表示的常量表中的常量值装载到寄存器 A 中。很多其他指令，比如数学操作指令，其本身可以直接从常量表中索引操作数，所以可以不依赖于 LOADK 指令； R(A) := Kst(Bx)	1
3	OP_LOADKX	LOADKX 是 lua5.2 新加入的指令。当需要生成 LOADK 指令时，如果需要索引的常量 id 超出了 Bx 所能表示的有效范围，那么就生成一个 LOADKX 指令，取代 LOADK 指令，并且接下来立即生成一个 EXTRAARG 指令，并用其 Ax 来存放这个 id。5.2 的这个改动使得一个函数可以处理超过 262143 个常量。	1
4	OP_LOADBOOL	LOADBOOL 将 B 所表示的 boolean 值装载到寄存器 A 中。B 使用 0 和 1 分别代表 false 和 true。C 也表示一个 boolean 值，如果 C 为 1，就跳过下一个指令。R(A) := (Bool)B; if (C) pc++	1
5	OP_LOADNIL	R(A), R(A+1), ..., R(A+B) := nil	2
6	OP_GETUPVAL	R(A) := UpValue[B]，A 是当前函数的外层函数函数的 local value	1
7	OP_GETTABUP	将 B 为索引的 upvalue 当作一个 table，并将 C 做为索引的寄存器或者常量当作 key 获取的值放入寄存器 A。R(A) := UpValue[B][RK(C)]	3
8	OP_GETTABLE	使用 C 表示的 key，将寄存器 B 中的表项值获取到寄存器 A 中。R(A) := R(B)[RK(C)]	3
9	OP_SETTABUP	将 A 为索引的 upvalue 当作一个 table，将 C 寄存器或者常量的值以 B 寄存器或常量为 key，存入 table。UpValue[A][RK(B)] := RK(C)	3
10	OP_SETUPVAL	UpValue[B] := R(A)	2
11	OP_SETTABLE	设置寄存器 A 的表的 B 项为 C 代表的值 R(A)[RK(B)] := RK(C)	3
12	OP_NEWTABLE	在寄存器 A 处创建一个 table 对象。B 和 C 分别用来存储这个 table 数组部分和 hash 部分的初始大小。R(A) := {} (size = B,C)	2
13	OP_SELF	专门为“:”运算符准备的指令。从寄存器 B 表示的 table 中，获取出 C 作为 key 的 closure，存入寄存器 A 中，然后将 table 本身存入到寄存器 A+1 中，为接下来调用这个 closure 做准备。R(A+1) := R(B); R(A) := R(B)[RK(C)]	4
14	OP_ADD	R(A) := RK(B) + RK(C)	1
15	OP_SUB	R(A) := RK(B) - RK(C)	1
16	OP_MUL	R(A) := RK(B) * RK(C)	3

17	OP_DIV	R(A) := RK(B) / RK(C		4
18	OP_MOD	R(A) := RK(B) % RK(C)		4
19	OP_POW	R(A) := RK(B) ^ RK(C)		4
20	OP_UNM	R(A) := -R(B)		1
21	OP_NOT	R(A) := not R(B)		1
22	OP_LEN	直接对应'#'操作符，返回 B 对象的长度，并保存到 A 中 R(A) := length of R(B)		1
23	OP_CONCAT	将 B 和 C 指定范围内的字符串按顺序传接到一起，将结果存入到 A：R(A) := R(B).. ... ..R(C)		2
24	OP_JMP	JMP 执行一个跳转，sBx 表示跳转的偏移位置，被加到当前指向下一指令的指令指针上。如果 sBx 为 0，表示没有任何跳转；1 表示跳过下一个指令；-1 表示重新执行当前指令。如果 A>0，表示需要关闭所有从寄存器 A+1 开始的所有 local 变量。实际执行的关闭操作只对 upvalue 有效。 pc+=sBx; if (A) close all upvalues >= R(A) + 1		2
25	OP_EQ	if ((RK(B) == RK(C)) ~= A) then pc++	关系指令对 RK(B)和 RK(C)进行比较，然后将比较结果与 A 指定的 boolean 值进行比较，来决定最终的 boolean 值。A 在这里为每个关系指令提供了两种比较目标，满足和不满足。比如 OP_LT 何以用来实现“<”和“>”	1
26	OP_LT	f ((RK(B) < RK(C)) ~= A) then pc++		
27	OP_LE	if ((RK(B) <= RK(C)) ~= A) then pc++		
28	OP_TEST	if not (R(A) <=> C) then pc++	逻辑指令用于实现 and 和 or 逻辑运算符，或者在条件语句中判断一个寄存器。	1

29	OP_TEST SET	if (R(B) <=> C) then R(A) := R(B) else pc++	TESTSET 将寄存器 B 转化成一个 boolean 值，然后与 C 进行比较。如果不相等，跳过后面的 JMP 指令。否则将寄存器 B 的值赋给寄存器 A，然后继续执行。TEST 是 TESTSET 的简化版，不需要赋值操作。	
30	OP_CAL L	CALL 执行一个函数调用。寄存器 A 中存放函数对象，所有参数按顺序放置在 A 后面的寄存器中。B－1 表示参数个数 。如果参数列表的最后一个表达式是变长的，则 B 会设置为 0，表示使用 A＋1 到当前栈顶作为参数。函数调用的返回值会按顺序存放在从寄存器 A 开始的 C-1 个寄存器中。如果 C 为 0,表示返回值的个数由函数决定。 A B C R(A), ... ,R(A+C-2) := R(A)(R(A+1), ... ,R(A+B-1))		3
31	OP_TAIL CALL	如果一个 return statement 只有一个函数调用表达式，这个函数调用指令 CALL 会被改为 TAILCALL 指令。TAILCALL 不会为要调用的函数增加调用堆栈的深度，而是直接使用当前调用信息。ABC 操作数与 CALL 的意思一样，不过 C 永远都是 0。TAILCALL 在执行过程中，只对 lua closure 进行 tail call 处理，对于 c closure，其实与 CALL 没什么区别。return R(A)(R(A+1), ... ,R(A+B-1))		4
32	OP_RET URN	RETURE 将返回结果存放到寄存器 A 到寄存器 A＋B－2 中。如果返回的为变长表达式，则 B 会被设置为 0，表示将寄存器 A 到当前栈顶的所有值返回。return R(A), ... ,R(A+B-2)		2
33	OP_CLO SURE	为指定的函数 prototype 创建一个 closure，并将这个 closure 保存到寄存器 A 中。Bx 用来指定函数 prototype 的 id。R(A) := closure(KPROTO[Bx])		3
34	OP_VAR ARG	G 直接对应'...'运算符。VARARG 拷贝 B-1 个参数到从 A 开始的寄存器中，如果不足，使用 nil 补充。如果 B 为 0，表示拷贝实际的参数数量。R(A), R(A+1), ..., R(A+B-2) = vararg		2

35	OP_FOR LOOP	R(A)+=R(A+2); if R(A) <?= R(A+1) then { pc+=sBx; R(A+3)=R(A) }	4
36	OP_FOR PREP	R(A)-=R(A+2); pc+=sBx	4
37	OP_TFO RCALL	R(A+3), ... ,R(A+2+C) := R(A)(R(A+1), R(A+2));	4
38	OP_TFO RLOOP	if R(A+1) ~= nil then { R(A)=R(A+1); pc += sBx }	4

4. 合约库列表

4.1 账号类接口

4.1.1 获得账号信息接口

static int scGetAccountInfo(lua_State* L)

接口简介：查询指定账号的账号信息。

输入参数		类型	简介
t		table	
	‘0’	string	查询账号的地址
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎

	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约调用格式如下，其中 curl 命令中 payload 字段是 lua 脚本的十六进制编码格式，参数也是字符串的十六进制编码格式。合约调用结果会在交易结果中 contractstate 字段中显示。合约部署作用是将生成合约账号，将 payload 中 lua 合约进行存储。合约调用时 destination 字段必须填写部署合约生成的合约账号的地址。

合约	合约 curl 格式

合约部署	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726e20726573756c743d73634765744163636f756e74496e666f287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f28742920613d7b7d20726573756c743d73634765744163636f756e74496e666f28742920207265747572e20726573756c742020656e64",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	
	}
	}

	]} ' 127.0.0.1:4050
Payload 字段 lua 脚本	result={}; function Init(t) result=scGetAccountInfo(t) return result end; function foo(t) a={} result=scGetAccountInfo(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	
	"Method": 1,
	"ContractMethod": "666f6f",
	"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	}
	}
	]}' 127.0.0.1:4050

4.1.2 获得账号挂单和信任线总数接口

scGetAccountOwnerCount（lua_State* L）

接口简介：查询指定账号的 ownercount 数目，ownercount 是指该账号挂单数目和信任线的总和。

输入参数		类型	简介
t		table	
	‘0’	string	查询账号的地址
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎

	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约调用格式如下，其中 curl 命令中 payload 字段是 lua 脚本的十六进制编码格式，参数也是字符串的十六进制编码格式。合约调用结果会在交易结果中 contractstate 字段中显示。合约部署作用是将生成合约账号，将 payload 中 lua 合约进行存储。合约调用时 destination 字段必须填写部署合约生成的合约账号的地址。

合约	合约调用 curl 格式

合约部署	url -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726e20726573756c743d7363476574416363 6f756e744f776e6572436f756e74287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f28742920726573756c743d7363476574416363 6f756e744f776e6572436f756e74287429202072657475726e20726573756c742020656e64",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	}
	}
	]}

	' 127.0.0.1:4050
Payload 字段 lua 脚本	result={}; function Init(t) result=scGetAccountOwnerCount(t) return result end; function foo(t) result=scGetAccountOwnerCount(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 1,
	"ContractMethod": "666f6f",
	"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	}
	}
	]}' 127.0.0.1:4050

4.1.3 获得账号余额接口

static int scGetAccountBalance(lua_State *L)

接口简介：获得指定账户的 swt 余额。

输入参数		类型	简介
t		table	
	‘0’	string	查询账号的地址
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657475726e20726573756c743d7363476574416363 6f756e7442616c616e63652874292020726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657475726e20726573756c743d7363476574416363 6f756e7442616c616e63652874292020726573756c742020656e64",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	}
	}
	]}
	' 127.0.0.1:4050

Payload 字段 lua 脚本	result={}; function Init(t) result=scGetAccountBalance(t) return result end; function foo(t) result=scGetAccountBalance(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",

<table>
<tr><td></td><td>

<pre>
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
"Args": [
{
"Arg": {
"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
}
}
]
}
}
}' 127.0.0.1:4050
</pre>

</td></tr>
</table>

4.1.4 获得账号挂单数目接口

static int scGetAccountOrderCount(lua_State *L)

接口简介：获得指定账号的挂单数目。

输入参数	类型	简介
t	table	
‘0’	string	查询账号的地址
TDstAccountID	string	合约账号地址
TEnginePointer	number	交易引擎
TTxn	number	交易指针
TEnginePointerHash	string	校验码
TTxnHash	string	校验码

输出参数	类型	简介
result	table	接口的返回结果
state	boolean	函数调用是否成功，1 表示成功，0 表示失败
res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

<table>
<tr><td>合约部署</td><td>curl -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 0,
"Amount": "100000000",
"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726e20726573756c743d7363476574416363 6f756e744f72646572436f756e74287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657375726e20726573756c743d7363476574416363 6f756e744f72646572436f756e74287429202072657475726e20726573756c742020656e64",
"Args": [
{
"Arg": {
"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
}
}
]
}
}
]}
' 127.0.0.1:4050</td></tr>
<tr><td>Payload 字段 lua 脚本</td><td>result={}; function Init(t) result=scGetAccountOrderCount(t) return result end; function foo(t) result=scGetAccountOrderCount(t) return result end</td></tr>
<tr><td>合约调用</td><td>curl -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
"Args": [
{</td></tr>
</table>

	<pre>"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	}
	}
	]}' 127.0.0.1:4050</pre>

4.1.5 获得账号指定货币余额接口

static int scGetAccountTum(lua_State *L)

接口简介：获得指定账号的指定货币的余额

输入参数		类型	简介
t		table	
	‘0’	String	查询账号的地址
	‘1’	string	银关账号的地址
	‘2’	String	查询货币的名称
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	<pre>curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {</pre>

	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726c743d7363476574416363 6f756e7454756d28742920272657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f28742920726573756c743d7363476574416363 6f756e7454756d28742920726573756c743d7363476574416363 6f756e7454756d2874292020726574 75726e20726573756c742020656e64",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a723 9315652576e3936446b756b473462776474795468"
	}
	} ,
	{
	"Arg": {
	"Parameter": "6a4547616535715475317777614 4346b7a5a6a4b39556279766e6a725a5673557739"
	}
	},
	{
	"Arg": {
	"Parameter": "434e59"
	}
	}
	]
	}
	}
	]}
	' 127.0.0.1:4050

Payload 字段 lua 脚本	result={}; function Init(t) result=scGetAccountTum(t) return result end; function foo(t) result=scGetAccountTum(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",

<table>
<tr><td></td><td>

<pre>
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4W
g",
"Args": [
{
"Arg": {
"Parameter": "6a486239434a41577942346a723
9315652576e3936446b756b473462776474795468"
}
} ,
{
"Arg": {
"Parameter": "6a4547616535715475317777614
4346b7a5a6a4b39556279766e6a725a5673557739"
}
},
{
"Arg": {
"Parameter": "434e59"
}
}
]
}
}
]}' 127.0.0.1:4050
</pre>

</td></tr>
</table>

4.1.6 获得账号当前交易序列号接口

static int scGetAccountSequence(lua_State *L)

接口简介：获得指定账号当前所做的交易总数。

输入参数		类型	简介
t		table	
	‘0’	string	查询账号的地址
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	url -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657475726e20726573756c742020656e64",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
	}
	}
	]
	}
	}
	]}
	' 127.0.0.1:4050

Payload 字段 lua 脚本	result={}; function Init(t) result=scGetAccountSequence(t) return result end; function foo(t) result=scGetAccountSequence(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {

<table>
<tr><td></td><td><pre>
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
"Args": [
{
"Arg": {
"Parameter": "6a486239434a41577942346a7239315652576e3936446b756b473462776474795468"
}
}
]
}
}
]}' 127.0.0.1:4050
</pre></td></tr>
</table>

4.1.7 获得指定交易接口

static int scGetTransaction(lua_State* L)

接口简介：查询指定交易 ID 的交易内容

输入参数		类型	简介
t		table	
	‘0’	string	查询账号的地址
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

<table>
<tr><td>合约部署</td><td>url -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 0,
"Amount": "100000000",
"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726c743d73634765745472616e73616374696f6e28742920207265747572e20726573756c742020656e643b202066756e6374696f6e20666f6f28742920207265747572e20726573756c743d73634765745472616e73616374696f6e28742920207265747572e20726573756c742020656e64",
"Args": [
{
"Arg": {
"Parameter": "9468B4C1CB3E95F24FD324B41217AAF977F2E9132EF491B55A97C5B2358C2C22"
}
}
]
}
}
]}
' 127.0.0.1:4050</td></tr>
<tr><td>Payload 字段 lua 脚本</td><td>result={}; function Init(t) result=scGetTransaction(t) return result end; function foo(t) result=scGetTransaction(t) return result end</td></tr>
<tr><td>合约调用</td><td>curl -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
"Args": [
{</td></tr>
</table>

	<pre>"Arg": {
	"Parameter": "9468B4C1CB3E95F24FD324B41217AAF977F2E9132EF491B55A97C5B2358C2C22"
	}
	}
	]
	}
	}
	]}' 127.0.0.1:4050</pre>

4.2.账本类接口

4.2.1 获得账本头信息

static int scGetLedgerInfo(lua_State* L)

命令简介：查询指定账本内容信息

输入参数		类型	简介
t		table	
	Ledgerindex	string	指定账本号
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	<pre>curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",</pre>

<table>
<tr><td></td><td>"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 0,
"Amount": "100000000",
"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726c743d73634765744c6564676572496e666f28742920207265747572e20726573756c742020656e643b202066756e6374696f6e20666f6f28742920207265747572e20726573756c743d73634765744c6564676572496e666f28742920207265747572e20726573756c742020656e64" ,
"Args": [
{
"Arg": {
"Parameter": "1000"
}
}
]
}
}
]}
' 127.0.0.1:4050</td></tr>
<tr><td>Payload 字段 lua 脚本</td><td>result={}; function Init(t) result=scGetLedgerInfo(t) return result end; function foo(t) result=scGetLedgerInfo(t) return result end</td></tr>
<tr><td>合约调用</td><td>curl -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
"Args": [
{
"Arg": {
"Parameter": "1000"
}
}
]
}
}
]}' 127.0.0.1:4050</td></tr>
</table>

4.2.2 获得当前账本序列号接口

static int scGetLedgerIndex(lua_State* L)

接口简介：获得当前账本号。

输入参数		类型	简介
t		table	
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726e20726573756c743d73634765744c6564676572496e6465782874292020726573756c743d73634765744c6564676572496e6465782874292020726573756c743d73634765744c6564676572496e646578287429202072657475726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657375726e20726573756c743d73634765744c6564676572496e646578287429202072657475726e20726573756c742020656e64" }
	}
	]}
	' 127.0.0.1:4050

Payload 字段 lua 脚本	result={}; function Init(t) result=scGetLedgerIndex(t) return result end; function foo(t) result=scGetLedgerIndex(t) return result end
合约调用	curl -X POST -d '{

	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 1,
	"ContractMethod": "666f6f",
	"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
	
	}
	}
	]}' 127.0.0.1:4050

4.2.3 获得最新关闭账本序列号

static int scGetLedgerClosedIndex(lua_State* L)

接口简介：获得上一个关闭账本的账本号。

输入参数	类型	简介
t	table	
TDstAccountID	string	合约账号地址
TEnginePointer	number	交易引擎
TTxn	number	交易指针
TEnginePointerHash	string	校验码
TTxnHash	string	校验码

输出参数	类型	简介
result	table	接口的返回结果
state	boolean	函数调用是否成功，1 表示成功，0 表示失败
res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式
合约部署	curl -X POST -d '{ "method": "submit", "params": [

<table>
<tr><td></td><td>
<pre>
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 0,
"Amount": "100000000",
"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657375726e20726573756c742020656e643b202066756e6374696f6e20666f6f287429202072657375726e20726573756c742020656e64"
}
}
]}
' 127.0.0.1:4050
</pre>
</td></tr>
<tr><td>Payload 字段
lua 脚本</td><td>result={}; function Init(t) result=scGetLedgerClosedIndex(t) return result end; function foo(t) result=scGetLedgerClosedIndex(t) return result end</td></tr>
<tr><td>合约调用</td><td>
<pre>
curl -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
}
}
]}' 127.0.0.1:4050
</pre>
</td></tr>
</table>

4.3 交易类接口

4.3.1 支付接口

static int scPayment(lua_State* L)

接口简介：同步支付接口,支付的源账号必须是合约账号或者是交易操作账号。否则无法成功

输入参数		类型	简介
t		tab	
	‘0’	String	支付原账号
	‘1’	string	支付目的账号
	‘2’	string	支付金额
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375756c745b27737461746527 5d203d20312020726573756c745b27726573275d203d202773756363657373272020726574757 26e20726573756c742020656e64202066756e6374696f6e20666f6f287429202097265737 56c743d7363506179d656e74287429202072657475726e20726573756c742020656e64",
	}
	}
	]}' 127.0.0.1:4050

Payload 字段 lua 脚本	result={}; function Init(t) result['state'] = 1 result['res'] = 'success' return result end function foo(t) result=scPayment(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 1,
	"ContractMethod": "666f6f",
	"Destination": "jBjprZbEjNu4NYD68HCMqQ7B5n8mqGZ4Wg",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a34554c5a56313445587542653387537626f7476597371644b46714e4769343548"
	}
	},
	{
	"Arg": {
	"Parameter": "6a7371696a6d446a763265766d6a756764526f484e35446d336a6d576f4736595275"
	}
	},
	{
	"Arg": {
	"Parameter": "32303030302f5553442f6a4263694445385133754a6a663131315665695554e4d373735414d4b484562424c53"
	}
	}
	]
	
	}
	}
	]}' 127.0.0.1:4050

4.4 状态存取类接口

4.4.1 状态存储接口

static int scStateStorage(lua_State* L)

接口简介:存储合约状态，存储合约调用时需要的参数

输入参数		类型	简介
t		table	
	‘0’	String	合约地址账号
	‘1’	String	存储状态的名称
	‘2’	String	存储状态的值
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c743d7b7d3b202066756e6374696f6e20496e6974287429202072657375726c745b277374617465275d203d20312020726573756c745b27726573275d203d202773756363657373272020726574756726e20726573756c742020656e64202066756e6374696f6e20666f6f28742920209726573756c743d7363537461746553746f726167652874292020726574756726e20726573756c742020656e64

<table>
<tr><td></td><td>"
}
}
]}' 127.0.0.1:4050</td></tr>
<tr><td>Payload 字段
lua 脚本</td><td>result={}; function Init(t) result['state'] = 1 result['res'] = 'success' return result end function foo(t) result=scStateStorage(t) return result end</td></tr>
<tr><td>合约调用</td><td>curl -X POST -d '{
"method": "submit",
"params": [
{
"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
"tx_json": {
"TransactionType": "ConfigContract",
"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
"Method": 1,
"ContractMethod": "666f6f",
"Destination": "jffPph792Rp3j1QgvYkj9ptdkHAoLsUAQH",
"Args": [
{
"Arg": {
"Parameter": "6a6666507068373932527033 6a31516776596b6a397074646b48416f4c7355415148"
}
},
{
"Arg": {
"Parameter": "6e616d6532"
}
},
{
"Arg": {
"Parameter": "7468697320697320736 5636f6e64207374617465"
}
}
]
}
}
]}' 127.0.0.1:4055</td></tr>
</table>

4.4.2 状态获取接口

static int scStateGet(lua_State* L)

输入参数		类型	简介
t		table	
	‘0’	String	合约地址账号
	‘1’	String	存储状态的名称
	TDstAccountID	string	合约账号地址
	TEnginePointer	number	交易引擎
	TTxn	number	交易指针
	TEnginePointerHash	string	校验码
	TTxnHash	string	校验码

接口简介:获取存储的合约参数或状态

输出参数		类型	简介
result		table	接口的返回结果
	state	boolean	函数调用是否成功，1 表示成功，0 表示失败
	res	string	成功返回查询账号的信息，是 json 的 string 格式；失败返回错误原因

合约	合约调用 curl 格式

合约部署	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 0,
	"Amount": "100000000",
	"Payload": "726573756c74203d207b7d202066756e6374696f6e20496e6974287429207265737 56c745b277374617465275d203d203120726573756c745b27726573275d3d20277375636365737327207265747572e20726573756c7420656e642066756e6374696f6e20666f6f28742920726573756c74203d207363537461746547657428742920726573756c7420726573756c7420656e64"
	}
	}
	]}
	' 127.0.0.1:4050

Payload 字段 lua 脚本	result = {} function Init(t) result['state'] = 1 result['res']= 'success' return result end function foo(t) result = scStateGet(t) return result end

合约调用	curl -X POST -d '{
	"method": "submit",
	"params": [
	{
	"secret": "snoPBjXtMeMyMHUVTgbuqAfg1SUTb",
	"tx_json": {
	"TransactionType": "ConfigContract",
	"Account": "jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh",
	"Method": 1,
	"ContractMethod": "666f6f",
	"Destination": "jspv6ptFVJPDcEUcS99hDbwuUa41irjtMw",
	"Args": [
	{
	"Arg": {
	"Parameter": "6a6666507068373932527033 6a31516776596b6a397074646b48416f4c7355415148"
	}
	},
	{
	"Arg": {
	"Parameter": "6e616d6532"
	}
	}
	]
	}
	}
	]}' 127.0.0.1:4055

5. lua 基本语法

Lua 是动态类型语言，变量不要类型定义,只需要为变量赋值，值可以存储在变量中，作为参数传递或结果返回。本节中只介绍合约系统用到的 lua 的基本语法。

智能合约采用的是 lua5.2,进过修改剪裁，不支持 io 和多线程操作。不支持 lua 脚本中包含多个 Init（t）函数。

6.1 lua 数据结构

系统用到中有 6 个基本类型分别为：nil、boolean、number、string、function 和 table。

数据结构	描述
nil	这个最简单，只有值 nil 属于该类，表示一个无效值（在条件表达式中相当于 false）。
boolean	包含两个值：true 和 false
number	表示双精度类型的实浮点数
string	字符串由一对双引号或单引号来表示
function	由 C 或 Lua 编写的函数
table	Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是字符串。在 Lua 里，table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。

6.2 lua 变量

变量在使用前，必须在代码中进行声明，即创建该变量，编译程序执行代码之前编译器需要知道如何给语句变量开辟存储区，用于存储变量的值。Lua 变量有三种类型：全局变量、局部变量、表中的域。Lua 中的变量全是全局变量，那怕是语句块或是函数里，除非用 local 显式声明为局部变量。局部变量的作用域为从声明位置开始到所在语句块结束。变量的默认值均为 nil

```
-- test.lua 文件脚本
a = 5                    -- 全局变量 local b = 5          -- 局部变量
function joke()
    c = 5                -- 全局变量
    local d = 6          -- 局部变量 end

joke()print(c,d)             --> 5 nil
do
    local a = 6          -- 局部变量
    b = 6                -- 全局变量
print(a,b);          --> 6 6
end
print(a,b)           --> 5 6
```

赋值语句

赋值是改变一个变量的值和改变表域的最基本的方法

```
a = "hello" .. "world"

t.n = t.n + 1
```

Lua 可以对多个变量同时赋值，变量列表和值列表的各个元素用逗号分开，赋值语句右边的值会依次赋给左边的变量。

```
a, b = 10, 2*x          <-->          a=10; b=2*x
```

遇到赋值语句 Lua 会先计算右边所有的值然后再执行赋值操作，所以我们可以这样进行交换变量的值：

```
x, y = y, x                          -- swap 'x' for 'y'

a[i], a[j] = a[j], a[i]              -- swap 'a[i]' for 'a[j]'
```

当变量个数和值的个数不一致时，Lua 会一直以变量个数为基础采取以下策略：
a. 变量个数 > 值的个数　　按变量个数补足 nil
b. 变量个数 < 值的个数　　多余的值会被忽略
例如：

```
a, b, c = 0, 1print(a,b,c)                  --> 0     1     nil

 a, b = a+1, b+1, b+2        -- value of b+2 is ignoredprint(a,b)                     --> 1     2

 a, b, c = 0print(a,b,c)                     --> 0     nil     nil
```

上面最后一个例子是一个常见的错误情况，注意：如果要对多个变量赋值必须依次对每个变量赋值。

```
a, b, c = 0, 0, 0print(a,b,c)                  --> 0     0     0
```

多值赋值经常用来交换变量，或将函数调用返回给变量：

```
a, b = f()
```

f()返回两个值，第一个赋给 a，第二个赋给 b。

应该尽可能的使用局部变量，有两个好处：
1. 避免命名冲突。
2. 访问局部变量的速度比全局变量更快。

索引
对 table 的索引使用方括号 []。Lua 也提供了 . 操作。

```
t[i]
t.i                    -- 当索引为字符串类型时的一种简化写法
gettable_event(t,i) -- 采用索引访问本质上是一个类似这样的函数调用
```

6.3 lua 循环

很多情况下我们需要做一些有规律性的重复操作，因此在程序中就需要重复执行某些语句。一组被重复执行的语句称之为循环体，能否继续重复，决定循环的终止条件。循环结构是在一定条件下反复执行某段程序的流程结构，被反复执行的程序被称为循环体。循环语句是由循环体及循环的终止条件两部分组成的。

循环类型	描述
While 循环	在条件为 true 时，让程序重复地执行某些语句。执行语句前会先检查条件是否为 true。true 时，让程序重复地执行某些语句。执行语句前会先检查条件是否为 true。
For 循环	重复执行指定语句，重复次数可在 for 语句中控制。

```
while(condition)do
   Statements
end
for var=exp1,exp2,exp3 do
    <执行体> end
```

var 从 exp1 变化到 exp2，每次变化以 exp3 为步长递增 var，并执行一次"执行体"。exp3 是可选的，如果不指定，默认为 1。

泛型 for 循环

泛型 for 循环通过一个迭代器函数来遍历所有值，类似 java 中的 foreach 语句。

Lua 编程语言中泛型 for 循环语法格式:

```
--打印数组 a 的所有值    for i,v in ipairs(a)
   do print(v) end
```

循环控制语句

```
while( a < 20 )do
    print("a 的值为:", a)
    a=a+1
    if( a > 15)
    then
        --[ 使用 break 语句终止循环 --]
       break
    endend
```

6.4 lua 流程控制

Lua 编程语言流程控制语句通过程序设定一个或多个条件语句来设定。在条件为 true 时执行指定程序代码，在条件为 false 时执行其他指定代码。

控制结构的条件表达式结果可以是任何值，Lua 认为 false 和 nil 为假，true 和非 nil 为真。

要注意的是 Lua 中 0 为 true：

语句	描述
If	if 语句 由一个布尔表达式作为条件判断，其后紧跟其他语句组成
If ...else	if 语句 可以与 else 语句搭配使用，在 if 条件表达式为 false 时执行 else 语句代码。
If 嵌套	你可以在 if 或 else if 中使用一个或多个 if 或 else if 语句

```
if(布尔表达式)then
    --[ 在布尔表达式为 true 时执行的语句 --]end
```

```
if(布尔表达式)then
```

```
    --[ 布尔表达式为 true 时执行该语句块 --]else
    --[ 布尔表达式为 false 时执行该语句块 --]end
if( 布尔表达式 1)then
    --[ 布尔表达式 1 为 true 时执行该语句块 --]
    if(布尔表达式 2)
    then
        --[ 布尔表达式 2 为 true 时执行该语句块 --]
    endend
```

6.5 Lua 函数

在 Lua 中，函数是对语句和表达式进行抽象的主要方法。既可以用来处理一些特殊的工作，也可以用来计算一些值。Lua 提供了许多的内建函数，你可以很方便的在程序中调用它们，如 print()函数可以将传入的参数打印在控制台上。Lua 函数主要有两种用途： 1.完成指定的任务，这种情况下函数作为调用语句使用； 2.计算并返回值，这种情况下函数作为赋值语句的表达式使用。

函数定义

```
optional_function_scope function function_name( argument1, argument2, argument3..., argumentn)
  function_body
  return result_params_comma_separated end
```

optional_function_scope: 该参数是可选的制定函数是全局函数还是局部函数，未设置该参数默认为全局函数，如果你需要设置函数为局部函数需要使用关键字 local。

function_name: 指定函数名称。

argument1, argument2, argument3..., argumentn: 函数参数，多个参数以逗号隔开，函数也可以不带参数。

function_body: 函数体，函数中需要执行的代码语句块。

result_params_comma_separated: 函数返回值，Lua 语言函数可以返回多个值，每个值以逗号隔开。

多返回值

```
s, e = string.find("www.runoob.com", "runoob")
```

可变参数

Lua 函数可以接受可变数目的参数，和 C 语言类似在函数参数列表中使用三点（...) 表示函数有可变的参数。

Lua 将函数的参数放在一个叫 arg 的表中，#arg 表示传入参数的个数。

例如，我们计算几个数的平均值：

```
function average(...)
    result = 0
    local arg={...}
    for i,v in ipairs(arg) do
        result = result + v
    end
    print("总共传入 " .. #arg .. " 个数")
    return result/#argend
print("平均值为",average(10,5,3,4,5,6))
```

6.6 运算符

算数运算符

下表列出了 Lua 语言中的常用算术运算符，设定 A 的值为 10,B 的值为 20：

操作符	描述	实例
+	加法	A + B 输出结果 30
-	减法	A - B 输出结果 -10
*	乘法	A * B 输出结果 200
/	除法	B / A 输出结果 2
%	取余	B % A 输出结果 **0**
^	乘幂	A^2 输出结果 100
-	负号	-A 输出结果 -10

逻辑运算符

下表列出了 Lua 语言中的常用逻辑运算符，设定 A 的值为 true，B 的值为 false：

操作符	描述	实例
and	逻辑与操作符。若 A 为 false，则返回 A，否则返回 B。	(A and B) 为 false。
or	逻辑或操作符。若 A 为 true，则返回 A，	(A or B) 为 true。

	否则返回 B。	
not	逻辑非操作符。与逻辑运算结果相反，如果条件为 true，逻辑非为 false。	not(A and B) 为 true。

关系运算符

下表列出了 Lua 语言中的常用关系运算符，设定 A 的值为 10，B 的值为 20：

操作符	描述	实例
==	等于，检测两个值是否相等，相等返回 true，否则返回 false	(A == B) 为 false。
~=	不等于，检测两个值是否相等，相等返回 false，否则返回 true	(A ~= B) 为 true。
>	大于，如果左边的值大于右边的值，返回 true，否则返回 false	(A > B) 为 false。
<	小于，如果左边的值大于右边的值，返回 false，否则返回 true	(A < B) 为 true。
>=	大于等于，如果左边的值大于等于右边的值，返回 true，否则返回 false	(A >= B) 返回 false。
<=	小于等于， 如果左边的值小于等于右边的值，返回 true，否则返回 false	(A <= B) 返回 true。

其他运算符

操作符	描述	实例
..	连接两个字符串 a..b ，其中 a 为 "Hello " ， b 为 "World",	输出结果为 "Hello World"。
#	一元运算符，返回字符串或表的长度。	#"Hello" 返回 5

运算符优先级从高到底的顺序

```
^ not    - (unary) *    / +    - .. <    >    <=    >=    ~=    ==andor
```

6.7 Lua table

table 是 Lua 的一种数据结构用来帮助我们创建不同的数据类型，如：数字、字典等。

Lua table 使用关联型数组，你可以用任意类型的值来作数组的索引，但这个值不能是 nil。

Lua table 是不固定大小的，你可以根据自己需要进行扩容。

Lua 也是通过 table 来解决模块（module）、包（package）和对象（Object）的。 例

如 string.format 表示使用"format"来索引 table string。

table 的构造

构造器是创建和初始化表的表达式。表是 Lua 特有的功能强大的东西。最简单的构造函数是{}，用来创建一个空表。可以直接初始化数组:

```
-- 初始化表
mytable = {}
-- 指定值
mytable[1]= "Lua"
-- 移除引用
mytable = nil-- lua 垃圾回收会释放内存
```

当我们为 table a 并设置元素，然后将 a 赋值给 b，则 a 与 b 都指向同一个内存。如果 a 设置为 nil ，则 b 同样能访问 table 的元素。如果没有指定的变量指向 a，Lua 的垃圾回收机制会清理相对应的内存。

以下列出 Table 操作常用的方法：

序号	方法 & 用途
1	table.concat (table [, sep [, start [, end]]]): concat 是 concatenate(连锁，连接)的缩写. table.concat()函数列出参数中指定 table 的数组部分从 start 位置到 end 位置的所有元素，元素间以指定的分隔符(sep)隔开。
2	table.insert (table, [pos,] value): 在 table 的数组部分指定位置(pos)插入值为 value 的一个元素. pos 参数可选，默认为数组部分末尾.
3	table.remove (table [, pos]) 返回 table 数组部分位于 pos 位置的元素. 其后的元素会被前移. pos 参数可选，默认为 table 长度，即从最后一个元素删起。
4	table.sort (table [, comp]) 对给定的 table 进行升序排序。