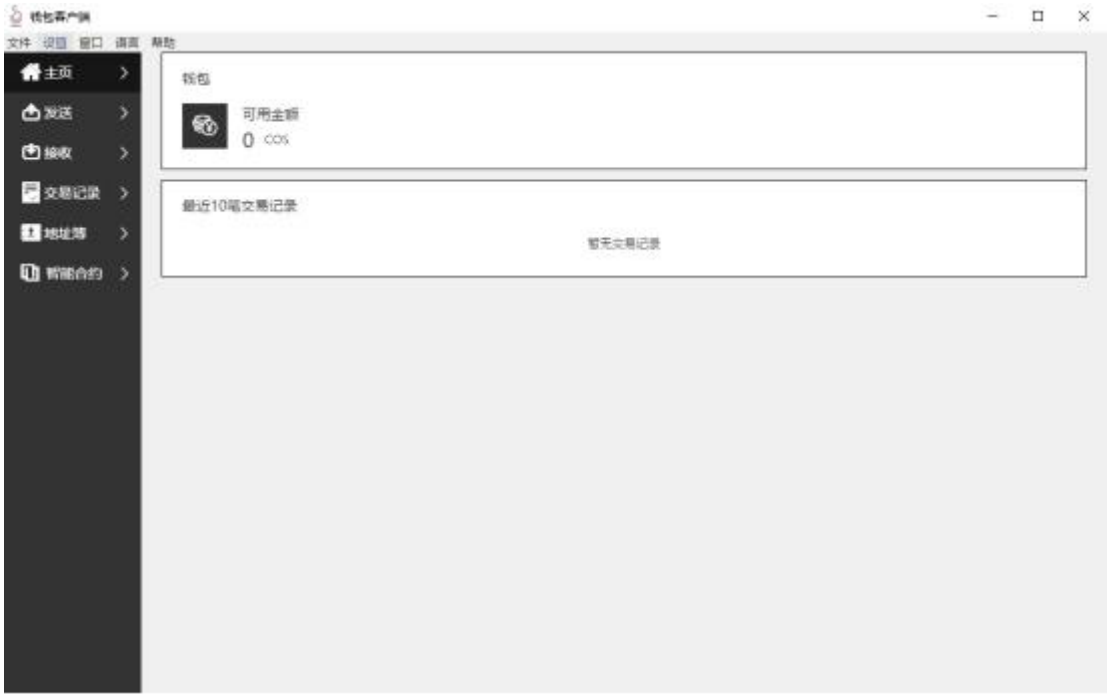
ICO 合约案例说明

1、 功能说明

通过区块链智能合约项目 ico，用户可通过向智能合约里面发送区块链代币，合约会根据参与者的地址与参与金额，合约会自动把参与者发送的金额转到指定的地址，并给其参与者分配相应的权益份额，用于后续权益兑换。

2、 合约开发部署流程

A、 启动一个区块链节点/钱包



B、编写智能合约代码

代码如下：

```
mylib = require "mylib"

function Unpack(t, i)
    i = i or 1
    if t[i] then
        return t[i], Unpack(t, i + 1)
    end
end

function GetValueFromContract(tbl, start, length)
    assert(start > 0, "GetValueFromContract start err")
    assert(length > 0, "GetValueFromContract length err")
    local newTab = {}
    local i
```

```
    for i = 0, length - 1 do
        newTab[1 + i] = tbl[start + i]
    end
    return newTab
end

OPER_TYPE = {
    ENUM_ADD_FREE = 1,
    ENUM_MINUS_FREE = 2
}

APP_OPERATOR_TYPE = {
    ENUM_ADD_FREE_OP = 1,
    ENUM_SUB_FREE_OP = 2,
    ENUM_ADD_FREEZED_OP = 3,
    ENUM_SUB_FREEZED_OP = 4
}

ADDR_TYPE = {
    ENUM_REGID = 1,
    ENUM_BASE58 = 2
}

local funs = {}

-- 众筹
funs[1] = function()
    local moneyTbl = { mylib.GetCurTxPayAmount() }
    local money = mylib.ByteToInteger(Unpack(moneyTbl))
    assert(money > 0, 'invalid money num')
    print('金额：', money)

    local appRecharge = { operatorType = APP_OPERATOR_TYPE.ENUM_ADD_FREE_OP, outHeight = 0, moneyTbl = moneyTbl, userIdLen = 34, userIdTbl = { mylib.GetBase58Addr(mylib.GetCurTxAccount()) }, fundTagLen = 0, fundTagTbl = {} }
    assert(mylib.WriteOutAppOperate(appRecharge), "failed to zc")

    local appRecharge = { operatorType = APP_OPERATOR_TYPE.ENUM_ADD_FREE_OP, outHeight = 0, moneyTbl = moneyTbl, userIdLen = 6, userIdTbl = { mylib.GetScriptID() }, fundTagLen = 0, fundTagTbl = {} }
    assert(mylib.WriteOutAppOperate(appRecharge), "failed to add app")

    local dataTbl = {
        addrType = ADDR_TYPE.ENUM_REGID,
```

```
        operatorType = OPER_TYPE.ENUM_MINUS_FREE,
        accountIdTbl = {mylib.GetScriptID()},
        moneyTbl = moneyTbl,
        outHeight = 0
    }
    assert(mylib.WriteOutput(dataTbl), 'failed to minus')

    local dataTbl2 = {
        addrType = ADDR_TYPE.ENUM_BASE58,
        operatorType = OPER_TYPE.ENUM_ADD_FREE,
        accountIdTbl = {0x54, 0x46, 0x67, 0x47, 0x4d, 0x64, 0x65, 0x74, 0x37, 0x75, 0x43, 0x70,
0x54, 0x36, 0x5a, 0x46, 0x4a, 0x67, 0x50, 0x6d, 0x43, 0x4b, 0x6e, 0x73, 0x6d, 0x39, 0x70, 0x41,
0x48, 0x33, 0x34, 0x5a, 0x4b, 0x6b},
        moneyTbl = moneyTbl,
        outHeight = 0
    }
    assert(mylib.WriteOutput(dataTbl2), 'failed to add')
end

-- 返回结果
print('合约执行完成')
return funs[contract[2]] and funs[contract[2]]() or true
```

C、在智能合约界面部署合约

注意：这一步会消耗一定量的币，返回一个交易 ID，等到下一个区块产生后，可通过这个交易 ID 查询合约 ID，所有跟合约相关的操作都会用到这个合约 ID。

部署：

获取合约 ID：

参与众筹：

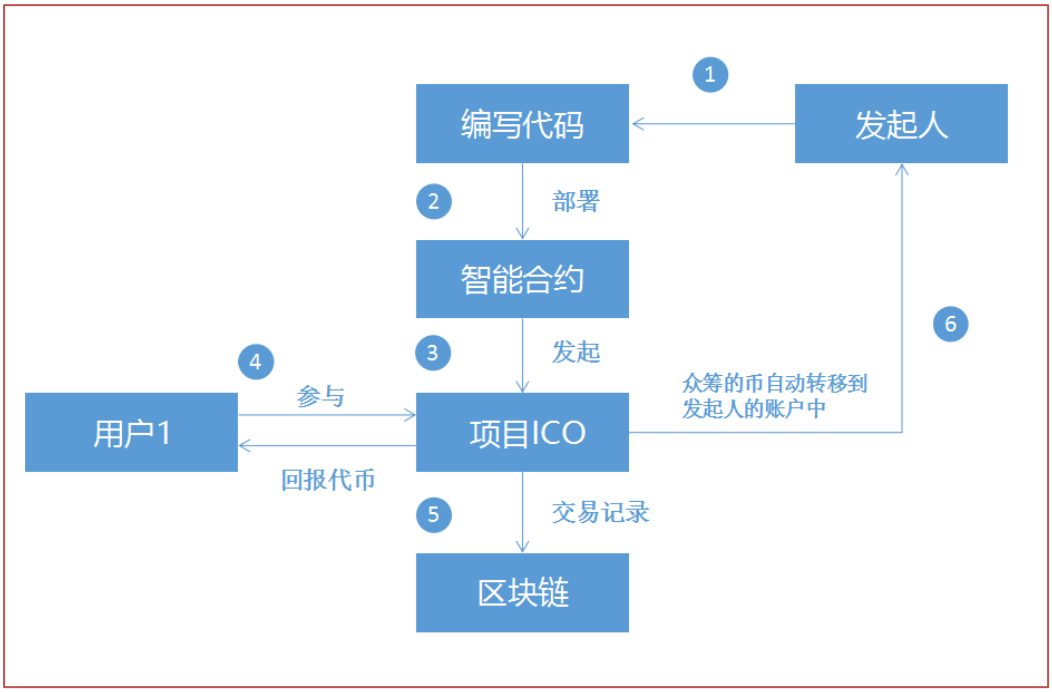
指定调用的智能合约，也就是之前得到的合约 ID，参与人的地址，参与金额，合约内容：ff01 表示参与众筹，点击创建，即会创建一笔合约交易，如果参与者有足够的金额，这笔交易会在下一个区块被确认。

查询份额：

通过合约 ID、地址，可查询参与 ico 所得份额，返回的 json 字符串，FreeValues 为用户所持份额。


文件 设置 窗口 语言 帮助
主页
发送
接收
交易记录
地址簿
智能合约
智能合约
步骤一
步骤二
步骤三
应用余额
查询余额
地址：
TVKBtZ6PMf2GuWTn2XBj9PjWQg6Kig45Y
合约id：
010000000100
查询
PS:注册合约以及运行合约之后需要进行挖矿来更新区块高度。
挖矿
当前区块高度：2

流程图如下：


1
编写代码
发起人
2
部署
智能合约
3
发起
4
参与
用户1
项目ICO
回报代币
众筹的币自动转移到
发起人的账户中
6
5
交易记录
区块链

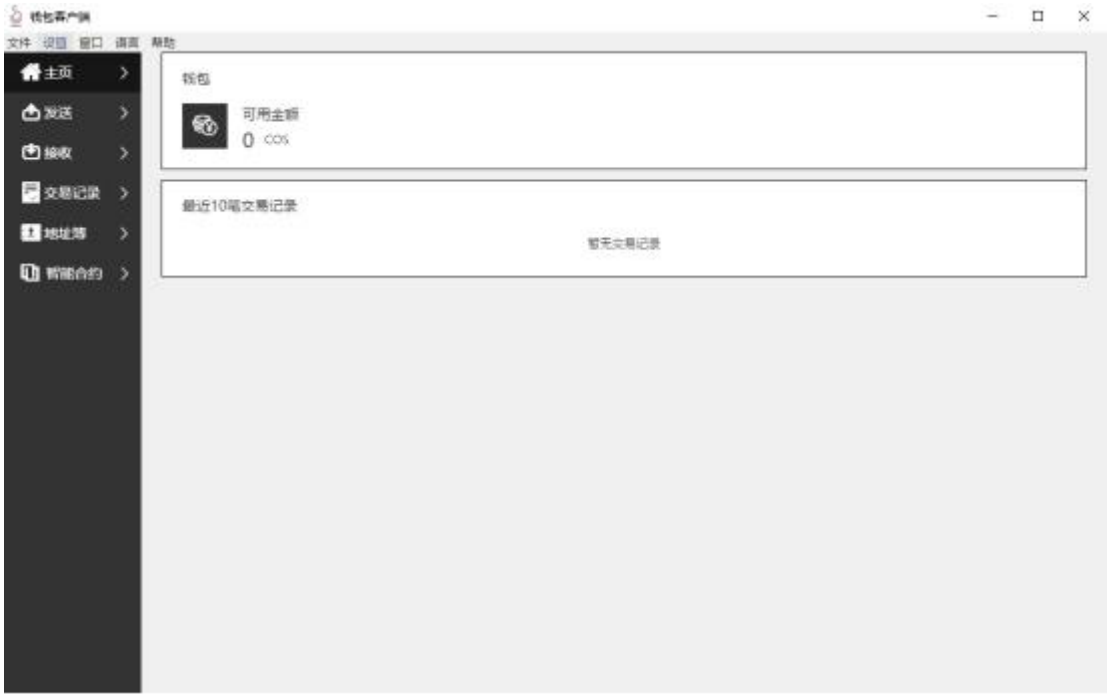
证券发行合约案例说明

1、功能说明：

证券发行，资产冻结，分期解冻。

3、 合约开发部署流程：

A、启动一个区块链节点/钱包



B、编写智能合约代码

代码如下：

```
mylib = require "mylib"

LOG_TYPE =
{
    ENUM_STRING = 0,        ENUM_NUMBER = 1, };

OPER_TYPE =
{
    ENUM_ADD_FREE = 1,        ENUM_MINUS_FREE = 2 }

APP_OPERATOR_TYPE =
{
    ENUM_ADD_FREE_OP    =    1,                ENUM_SUB_FREE_OP    =    2,
ENUM_ADD_FREEZED_OP = 3,        ENUM_SUB_FREEZED_OP = 4 }

ADDR_TYPE =
{
    ENUM_REGID = 1,        ENUM_BASE58 = 2, }

TX_TYPE =
```

```
{
	TX_WITHDRAW = 1,		TX_RECHARGE = 2, }

FREEZE_MONTH_NUM = 20 FREEZE_PERIOD = 5
MAX_MONEY = 100000000000000000 FREE_MONEY = 10000000000000000
gCheckAccount = true
gSendAccountTbl = {
	0x00, 0x00, 0x00, 0x00, 0x01, 0x00
}

function TableIsEmpty(t)
	return _G.next(t) == nil
end

function TableIsNotEmpty(t)
	return false == TableIsEmpty(t)
end
function LogPrint(aKey, bLength, cValue)
	assert(bLength >= 1, "LogPrint bLength invlaid")
	if (aKey == LOG_TYPE.ENUM_STRING) then
		assert(type(cValue) == "string", "LogPrint cValue invlaid0")
	elseif (aKey == LOG_TYPE.ENUM_NUMBER) then
		assert(TableIsNotEmpty(cValue), "LogPrint cValue invlaid1")
	else
		error("LogPrint aKey invlaid")
	end

	local logTable = {
		key = LOG_TYPE.ENUM_STRING,
		length = 0,
		value = nil
	}
	logTable.key = aKey
	logTable.length = bLength
	logTable.value = cValue
	mylib.LogPrint(logTable)
end
function Unpack(t, i)
	i = i or 1
	if t[i] then
		return t[i], Unpack(t, i + 1)
	end
end
function MemCmp(tDest, tSrc, start1)
```

```
    assert(TableIsNotEmpty(tDest), "tDest is empty")
    assert(TableIsNotEmpty(tSrc), "tSrc is empty")
    local i
    for i = #tDest, 1, -1 do
        if tDest[i] > tSrc[i + start1 - 1] then
            return 1
        elseif tDest[i] < tSrc[i + start1 - 1] then
            return -1
        else
        end
    end
    return 0
end

function GetValueFromContract(tbl, start, length)
    assert(start > 0, "GetValueFromContract start err")
    assert(length > 0, "GetValueFromContract length err")
    local newTab = {}
    local i
    for i = 0, length - 1 do
        newTab[1 + i] = tbl[start + i]
    end
    return newTab
end
function Recharge()
                                    local toAddrTbl = {}
    toAddrTbl = GetValueFromContract(contract, 3, 34)
    local moneyTbl = {}
    moneyTbl = GetValueFromContract(contract, 37, 8)
    local money = mylib.ByteToInteger(Unpack(moneyTbl))
    assert(money > 0, "money <= 0")
    local freeMoneyTbl = {}
    freeMoneyTbl = GetValueFromContract(contract, 45, 8)
    local freeMoney = mylib.ByteToInteger(Unpack(freeMoneyTbl))
    assert(freeMoney > 0, "freeMoney <= 0")
    local freeMonthMoneyTbl = {}
    freeMonthMoneyTbl = GetValueFromContract(contract, 53, 8)
    local freeMonthMoney = mylib.ByteToInteger(Unpack(freeMonthMoneyTbl))
    assert(freeMonthMoney > 0, "freeMonthMoney <= 0")
    local payMoneyTbl = {}
    payMoneyTbl = { mylib.GetCurTxPayAmount() }
    assert(TableIsNotEmpty(payMoneyTbl), "GetCurTxPayAmount error")
    local payMoney = mylib.ByteToInteger(Unpack(payMoneyTbl))
    assert(payMoney > 0, "payMoney <= 0")
```

```
        assert(money == payMoney, "充值金额不正确 1")
        if money < 1
            or money < freeMoney
            or money < freeMonthMoney then
        LogPrint(LOG_TYPE.ENUM_STRING, string.len("充值金额不正确 2"), "充值金额不正确 2");
        error("充值金额不正确 2")
    end
        if money >= MAX_MONEY
            or freeMoney >= FREE_MONEY
            or freeMonthMoney >= FREE_MONEY then
        LogPrint(LOG_TYPE.ENUM_STRING, string.len("充值金额不正确 3"), "充值金额不正确 3");
        error("充值金额不正确 3")
    end
    local freezeNum = FREEZE_MONTH_NUM - 1
    assert(freezeNum > 0, "月冻结总数不正确")
        local freezeMoney = freeMonthMoney * freezeNum
    if freezeMoney < freezeNum
            or freezeMoney < freeMoney
            or money < freezeMoney then
        LogPrint(LOG_TYPE.ENUM_STRING, string.len("充值金额不正确 4"), "充值金额不正确 4");
        error("充值金额不正确 4")
    end
        freezeMoney = freezeMoney + freeMoney
    if money ~= freezeMoney then
        LogPrint(LOG_TYPE.ENUM_STRING, string.len("充值金额不正确 5"), "充值金额不正确 5");
        error("充值金额不正确 5")
    end
            local writeOutputTbl =
    {
        addrType = 1,           accountIdTbl = {},          operatorType = 0,
outHeight = 0,          moneyTbl = {}       }

    writeOutputTbl.addrType = ADDR_TYPE.ENUM_BASE58
    writeOutputTbl.operatorType = OPER_TYPE.ENUM_ADD_FREE
    writeOutputTbl.accountIdTbl = { Unpack(toAddrTbl) }
    writeOutputTbl.moneyTbl = { Unpack(freeMoneyTbl) }
    assert(mylib.WriteOutput(writeOutputTbl), "WriteOutput err1")
    local curHeight = 0
    curHeight = mylib.GetCurRunEnvHeight()
        local appOperateTbl = {
```

```
            operatorType = 0,           outHeight = 0,          moneyTbl = {},
            userIdLen = 0,              userIdTbl = {},         fundTagLen = 0,
fundTagTbl = {}         }
        appOperateTbl.operatorType = APP_OPERATOR_TYPE.ENUM_ADD_FREEZED_OP
        appOperateTbl.userIdLen = 34
        appOperateTbl.userIdTbl = toAddrTbl
        appOperateTbl.moneyTbl = freeMonthMoneyTbl
        for i = 1, freezeNum do
            appOperateTbl.outHeight = curHeight + FREEZE_PERIOD * i
            assert(mylib.WriteOutAppOperate(appOperateTbl),     "WriteOutAppOperate
err1")
        end
            writeOutputTbl.addrType = ADDR_TYPE.ENUM_REGID
        writeOutputTbl.operatorType = OPER_TYPE.ENUM_MINUS_FREE
        writeOutputTbl.accountIdTbl = { mylib.GetScriptID() }
        assert(mylib.WriteOutput(writeOutputTbl), "WriteOutput err2")
        return true
    end
    function WriteWithdrawal(accTbl, moneyTbl)
            local writeOutputTbl =
        {
            addrType = 1,           accountIdTbl = {},          operatorType = 0,
outHeight = 0,          moneyTbl = {}       }
            assert(TableIsNotEmpty(accTbl), "WriteWithDrawal accTbl invlaid1")
        assert(TableIsNotEmpty(moneyTbl), "WriteWithDrawal moneyTbl invlaid1")
            writeOutputTbl.addrType = ADDR_TYPE.ENUM_REGID
        writeOutputTbl.operatorType = OPER_TYPE.ENUM_ADD_FREE
        writeOutputTbl.accountIdTbl = { Unpack(accTbl) }
        writeOutputTbl.moneyTbl = { Unpack(moneyTbl) }
        assert(mylib.WriteOutput(writeOutputTbl), "WriteWithDrawal WriteOutput err0")
            writeOutputTbl.operatorType = OPER_TYPE.ENUM_MINUS_FREE
        writeOutputTbl.accountIdTbl = { mylib.GetScriptID() }
        assert(mylib.WriteOutput(writeOutputTbl), "WriteWithDrawal WriteOutput err1")
        return true
    end
    function Withdraw(addrType)
            if addrType ~= ADDR_TYPE.ENUM_REGID and addrType ~=
ADDR_TYPE.ENUM_BASE58 then
            error("In function Withdraw, addr type err")
        end
            local accountTbl = { 0, 0, 0, 0, 0, 0 }
        accountTbl = { mylib.GetCurTxAccount() }
        assert(TableIsNotEmpty(accountTbl), "GetCurTxAccount error")
        local idTbl =
```

```
        {
            idLen = 0,
            idValueTbl = {}
        }
        if addrType == ADDR_TYPE.ENUM_REGID then
            idTbl.idLen = 6
            idTbl.idValueTbl = accountTbl
        else
            local base58Addr = {}
            base58Addr = { mylib.GetBase58Addr(Unpack(accountTbl)) }
            assert(TableIsNotEmpty(base58Addr), "GetBase58Addr error")
            idTbl.idLen = 34
            idTbl.idValueTbl = base58Addr
        end
            local freeMoneyTbl = { mylib.GetUserAppAccValue(idTbl) }
        assert(TableIsNotEmpty(freeMoneyTbl), "GetUserAppAccValue error")
        local freeMoney = mylib.ByteToInteger(Unpack(freeMoneyTbl))
        assert(freeMoney > 0, "Account balance is 0.")
            local appOperateTbl = {
            operatorType = 0,           outHeight = 0,          moneyTbl = {},
            userIdLen = 0,              userIdTbl = {},             fundTagLen = 0,
fundTagTbl = {}         }
        appOperateTbl.operatorType = APP_OPERATOR_TYPE.ENUM_SUB_FREE_OP
        appOperateTbl.userIdLen = idTbl.idLen
        appOperateTbl.userIdTbl = idTbl.idValueTbl
        appOperateTbl.moneyTbl = freeMoneyTbl
        assert(mylib.WriteOutAppOperate(appOperateTbl), "WriteOutAppOperate err1")
            assert(WriteWithdrawal(accountTbl, freeMoneyTbl), "WriteWithdrawal err")
        return true
    end


    function Main()
        assert(#contract >= 2, "contract length err.")
        assert(contract[1] == 0xff, "Contract identification error.")

        if contract[2] == TX_TYPE.TX_RECHARGE then
            assert(#contract == 60, "recharge contract length err.")
            Recharge()
        elseif contract[2] == TX_TYPE.TX_WITHDRAW then
            assert(#contract == 11, "withdraw contract length err.")
            Withdraw(contract[3])
        else
            error("RUN_SCRIPT_DATA_ERR")
```

```
        end
    end

    Main()
```

C、在智能合约界面部署合约

注意：这一步会消耗一定量的币， 返回一个交易 ID，等到下一个区块产生后，可通过这个交易 ID 查询合约 ID，所有跟合约相关的操作都会用到这个合约 ID。

部署：

获取合约 ID：

发行资产：

指定调用的智能合约，也就是之前得到的合约 ID，发起人的地址，发行金额，合约内容：

ff02544667474d6465743775437054365a464a67506d434b6e736d3970414833345a4b6b00e40b54020000000065cd1d000000000065cd1d0000000 表示冻结资产到某个地址，点击创建，即会创建一笔合约交易，如果参与者有足够的金额，这笔交易会在下一个区块被确认。

账户查询：

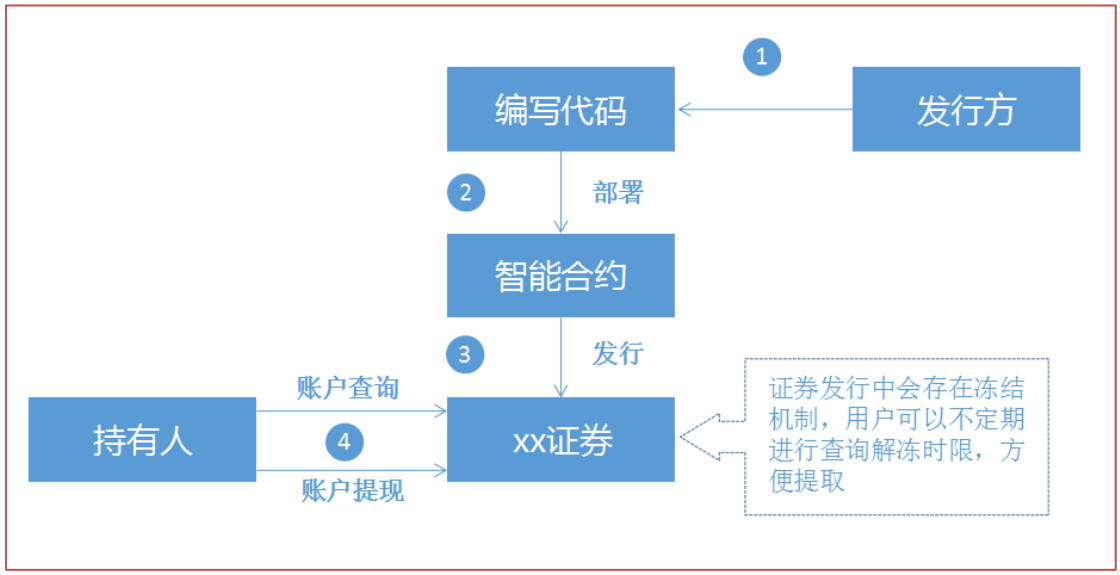
通过合约 ID、地址，可查询冻结，解冻情况，返回的 json 字符串，FreeValues 为用户已解冻份额；m_vcFreezedFund 为冻结的资产，value 表示冻结金额，height 表示解冻的区块高度，这里有多个{Value: .., height: ..} ，表示多期解冻，可以看到最近一个即将解冻的资产将在第 9 个区块解冻，当前区块是 4，需要挖矿，等待新区块产生。


文件 设置 窗口 语言 帮助
主页
发送
接收
交易记录
地址簿
智能合约
智能合约
步骤一
步骤二
步骤三
应用余额
查询余额
地址：
TFgGMdet7uCpT6ZFJgPmCKnsm9pAH34Z
合约id：
030000000100
查询
PS:注册合约以及运行合约之后需要进行挖矿来更新区块高度。
挖矿
当前区块高度：4

账户提现：


文件 设置 窗口 语言 帮助
主页
发送
接收
交易记录
地址簿
智能合约
智能合约
步骤一
步骤二
步骤三
应用余额
运行合约
地址：
TFgGMdet7uCpT6ZFJgPmCKnsm9pAH34Z
合约id：
030000000100
操作金额：
0
合约内容：
ff01020065cd1d00000000
手续费：
1
创建
PS:注册合约以及运行合约之后需要进行挖矿来更新区块高度。

流程图如下：


1
编写代码
发行方
2
部署
智能合约
3
发行
账户查询
持有人
4
xx证券
账户提现
证券发行中会存在冻结机制，用户可以不定期进行查询解冻时限，方便提取