介绍

GkcJS 是一个用于在 Gkc 区块链上开发 DApp 的 JavaScript 库。您可以使用此库来开发在浏览器中运行的前端 UI 以及在 NodeJS 中运行的后端脚本。
主要的类有：

类	描述
GkcRPCRaw	使用 JSONRPC1.0 调用合约，直接访问 gkc 的区块链 RPC 服务。
GkcRPC	GkcRPCRaw 的封装，提供像 JSONRPC2.0 这样的接口。
Contract	与智能合约交互的抽象层。使用 ABI encoding/decoding

GkcJS 使用 TypeScript 开发,因此为所有 API 提供了健壮的类型定义。我们建议使用 VSCode 来获得语言支持，例如类型提示和自动完成。
当然，你愿意的话也可以选择使用普通的 JavaScript 和记事本。
本文档是 GkcJS API 及其基本用法的参考文档。

安装 gkcjs

```
npm install git+https://github.com/gkcproject/gkc_js.git
```

Gkc

Gkc 是 GkcJS API 的一个对象.它提供两个主要功能：
•对 gkc RPC 服务的访问.它是 GkcRPC 的子类.
•实例化 Contract 对象的工厂方法,用于与已部署的合约进行交互。

参数	**类型**
url	string
	Gkc RPC 服务的 URL
repoData	IContractsRepoData
	关于 Solidity 合约的信息.

repoData 包含所有已部署合约或库的 ABI 定义，以及它们的部署地址。这些信息用于实例化 Contract 实例。
使用 Gkc 的工厂方法实例化的 Contract 对象能够解码所有在 repoData 里的事件类型.但是手动构建的合约只能解码在其范围内定义的事件类型,这也是 Solidity 编译器输出 ABI 定义的限制。

建议使用 Gkc 来实例化 Contract 对象.

```
const repoData = require("./solar.json")
const gkc = new Gkc("http://gkc:test@localhost:48803", repoData)
```

contract

实例化 Contract 对象的工厂方法，使用了 repoData 中的 ABI 定义和地址。合约对象是使用一个事件 log 解码器来配置的，这个解码器能解码所有 repoData 中已知的事件类型。

参数	类型
name	string
	作为 repoData.contracts map 的 key，用于获取合约信息。

```
const myToken = gkc.contract("contract_name")
```

实例化这个合约使用了这些信息。

rawCall

继承自 GkcRPC#rawcall

Contract

与智能合约交互的抽象层。
这比使用 GkcRPC 直接调用 RPC 的 sendcontract 和 calltocontract 方法更方便。它处理 ABI 编码，转换 JS 和 Solidity 值。
•有 API 用于确认交易。
•有 API 用于调用合约方法，使用 call 或 send .
•有 API 用于获取合约 log 事件。

构造函数

参数	类型	描述
rpc	GkcRPC	RPC 对象，用于与合约进行交互

info	IContractInfo	信息，用于部署合约

建议使用 Gkc#contract 而不是这个构造器。

```
const rpc = new GkcRPC("http://gkc:test@localhost:48803")
const myToken = new Contract(rpc, repo.contracts[
  "contract_name"])
```

call

使用 callcontract 在你本地 gkc 节点"模拟"执行合约方法。这是免费的，实际上并不修改区块链。

这个免费。

参数	类型
method	string
	合约方法名
args	Array<any>
	调用方法的参数
opts	IContractCallRequestOptions
	调用配置项
@return	Promise<IContractCallResult>
	调用结果，带有 ABI 解码的输出

```
async function totalSupply() {
  const result = await myToken.call("totalSupply")

  // supply is a BigNumber instance (see: bn.js)
  const supply = result.outputs[0]

  console.log("supply", supply.toNumber())}
```

实例输出:

```
supply 10000000
```

模拟"balanceOf"调用:

```
const result = await myToken.call("balanceOf",
["c99042ffb7db7ddda4aab04cce545483e32163a5"])
const balance = res.outputs[0]
console.log("balance", balance.toNumber())
```

运行结果:

```
balance: 1234
```

send

创建一个交易，在网络全局执行合约方法，会改变区块链。
这要花费 gas.
对一个合约有 2 个异步步骤
1.你提交交易到网络
2.一旦提交，等待一个指定的确认数
成功确认后，返回带有 ABI 解码的事件日志的交易收据(IContractSendReceipt)

参数	类型
method	string
	合约方法名
args	Array<any>
	所调用方法的参数
opts	IContractSendRequestOptions
	可选发送配置项
@return	Promise<IContractSendResult>
	调用结果，带有 ABI 解码的输出

```
async function transfer(toAddr, amount) {
  const tx = await myToken.send("transfer", [toAddr, amount])
  console.log("transfer tx:", tx.txid)
  console.log(tx)
  const confirmation = tx.confirm(1)
  ora.promise(confirmation, "confirm transfer")
  await confirmation
}
```

实例输出:

```
transfer tx: 27a94c77155813858a3cce7d718a3eff674b81fc4abd38a9843bc850b2254a5b
{ amount: 0,
  fee: -0.0802407,
  confirmations: 0,
  bcconfirmations: 0,
  txid:
   '27a94c77155813858a3cce7d718a3eff674b81fc4abd38a9843bc850b2254a5b',
  walletconflicts: [],
```

```
  time: 1596094914,
  timereceived: 1596094914,
  details:
   [ { account: '',
       category: 'send',
       amount: 0,
       vout: 0,
       fee: -0.0802407 },
     { account: '',
       address: 'GfKrofVYEjWAeaATJreij9k8TVRucrLxmL',
       category: 'send',
       amount: -1.83951806,
       vout: 1,
       fee: -0.0802407 },
     { account: '',
       address: 'GfKrofVYEjWAeaATJreij9k8TVRucrLxmL',
       category: 'receive',
       amount: 1.83951806,
       vout: 1 } ],
  hex:
    '01000000014fef53a6b7c9d1630f071387e1111ec935f0ce80837f020a09b96cec78c2b1e900
0000006b483045022100dae5fe5780d004dd3945bd7ebf46c7246b9fa189ecb070112e689a679e35
21a502202ae7aa7f5dd7e00da3bab869d9f7338d765aee5c885066b5fbe8ab9191fa4268012103ec
a1c7591f46b7bbc3a1c38e88b31c0207166a63e758e233e13476f761d02afdfffffffff0200000000
00000000cc010403400d03012844a9059cbb0000000000000000000000eba34612bcfcb40b25ba
3d83a0e6299f9d932c3e0000000000000000000000000000000000000000000000000000000000
27101428a855f4153b072ffd744725da1b6524860563382103eca1c7591f46b7bbc3a1c38e88b31c
0207166a63e758e233e13476f761d02afd46304402204e028f7fea8bb954a28fa77a60f2213ce8f1
893c56aacf70de92c3599b67d13a0220515a20f9d734ec3cd3f9fb6e0bfa6323690fe244615859f7
d48ef7b26209a725c400bee1f60a000000001976a914eba34612bcfcb40b25ba3d83a0e6299f9d93
2c3e88ac0000000000000000000000000000000000000000000000000000000000000000000000
0000000000',
  method: 'transfer',
  confirm: [Function: confirm] }
√ confirm transfer
```

方法重载

如果没有歧义，使用方法名称来调用/发送方法。如果相同方法名称具有多个定义，请使用方法签名来调用/发送方法。

方法名 foo 可能有多个定义:

```
function foo();
function foo(int256 _a);
```

```
function foo(uint256 _a, uint256 _b);
function foo(int256 _a, int256 _b);
```

foo 方法有 0 个参数和有 1 个参数没有歧义。可以直接调用。

```
contract.call("foo")
contract.call("foo", [1])
```

foo 方法带 2 个参数的有歧义，必须带完整方法签名：

```
contract.call("foo(uint256,uint256)", [1, 2])
contract.call("foo(int256,int256)", [1, 2])
```

logs

获取由合约生成的 Solidity 事件日志。

通过指定 fromBlock 和 toBlock，可以将事件日志查询限制块号范围。 例如，可以查询块 1000 到 1500 之间的事件日志。

此外，你可以使用 minconf 指定事件日志之前确认的最小数量作为结果返回。

参数	实例
opts	IRPCSearchLogsRequest
	事件日志查询参数
@return	Promise<IContractEventLogs>
	日志查询结果，带有 ABI 解码的输出

```
async function getLogs(fromBlock=0, toBlock="latest") {
  const logs = await myToken.logs({
    fromBlock,
    toBlock,
    minconf: 1,
  })

  console.log(JSON.stringify(logs, null, 2))}
```

实例输出:

```
{
  "entries": [
    {
      "blockHash":
"1fcf8fd16c152a2f9aa76dd8d47a24053ddd67dacb25bdda7ec7c48c502c373d",
      "blockNumber": 227461,
      "transactionHash":
"b7f887556ddd881bddb61d89234552019b3a8b1ca113303c3a8b84f01ae6b565",
      "transactionIndex": 2,
```

```
      "from": "c99042ffb7db7ddda4aab04cce545483e32163a5",
      "to": "28a855f4153b072ffd744725da1b652486056338",
      "cumulativeGasUsed": 58385,
      "gasUsed": 58385,
      "contractAddress": "28a855f4153b072ffd744725da1b652486056338",
      "excepted": "0",
      "log": [
        {
          "address": "28a855f4153b072ffd744725da1b652486056338",
          "topics": [
            "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
            "000000000000000000000000c99042ffb7db7ddda4aab04cce545483e32163a5",
            "000000000000000000000000eba34612bcfcb40b25ba3d83a0e6299f9d932c3e"
          ],
          "data": "0000000000000000000000000000000000000000000000000000000005f5e100"
        }
      ],
      "event": {
        "_value": "5f5e100",
        "_from": "c99042ffb7db7ddda4aab04cce545483e32163a5",
        "_to": "eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
        "type": "Transfer"
      }
    },
    {
      "blockHash":
"5d6ba5e381c1232d373af456c8016ddb5f872108515404c937e67fb1506b0066",
      "blockNumber": 227465,
      "transactionHash":
"4f13c59ed2ac2b262193acfee2906a2d96fcfebc963e89d37308d722a8745fee",
      "transactionIndex": 2,
      "from": "c99042ffb7db7ddda4aab04cce545483e32163a5",
      "to": "28a855f4153b072ffd744725da1b652486056338",
      "cumulativeGasUsed": 28449,
      "gasUsed": 28449,
      "contractAddress": "28a855f4153b072ffd744725da1b652486056338",
      "excepted": "0",
      "log": [
        {
          "address": "28a855f4153b072ffd744725da1b652486056338",
          "topics": [
            "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
            "000000000000000000000000c99042ffb7db7ddda4aab04cce545483e32163a5",
            "000000000000000000000000eba34612bcfcb40b25ba3d83a0e6299f9d932c3e"
```

```
        ],
        "data": "0000000000000000000000000000000000000000000000000000000e8ceaf2f00"
      }
    ],
    "event": {
      "_value": "e8ceaf2f00",
      "_from": "c99042ffb7db7ddda4aab04cce545483e32163a5",
      "_to": "eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
      "type": "Transfer"
    }
  },
  {
    "blockHash":
"2422ed6edf01bcf971ed5953892d2c3b5b6305ac5e0b46c622d20411c093a185",
    "blockNumber": 227523,
    "transactionHash":
"e9b1c278ec6cb9090a027f8380cef035c91e11e18713070f63d1c9b7a653ef4f",
    "transactionIndex": 2,
    "from": "eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
    "to": "28a855f4153b072ffd744725da1b652486056338",
    "cumulativeGasUsed": 58321,
    "gasUsed": 58321,
    "contractAddress": "28a855f4153b072ffd744725da1b652486056338",
    "excepted": "0",
    "log": [
      {
        "address": "28a855f4153b072ffd744725da1b652486056338",
        "topics": [
          "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
          "000000000000000000000000eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
          "000000000000000000000000c99042ffb7db7ddda4aab04cce545483e32163a5"
        ],
        "data": "00000000000000000000000000000000000000000000000000000000000004d2"
      }
    ],
    "event": {
      "_value": "4d2",
      "_from": "eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
      "_to": "c99042ffb7db7ddda4aab04cce545483e32163a5",
      "type": "Transfer"
    }
  }
 ],
 "count": 3,
```

```
  "nextblock": 227524
}
```

onlogs

订阅合约新事件。每次收到新事件时都会调用回调。默认情况下，onLog 监听来自区块链顶端的日志。使用 fromBlock 也可以接收较早的事件。

参数	实例
callback	(entry: IContractEventLog) => void
opts	IRPCSearchLogsRequest
	事件日志查询参数

```
myToken.onLog((entry) => {
    console.log(entry)}, { minconf: 1 })
```

logEmitter

使用 EventsEmitter 接口订阅合约新事件。发出的事件是 IContractEventLog 对象。
Solidity 事件名作为发出的事件名使用。
缺失 ABI 定义的事件 (即不能解析) 会发送 "?".

参数	实例
opts	IRPCSearchLogsRequest
	事件日志查询参数

```
this.emitter = myToken.logEmitter({ minconf: 1 })
this.emitter.on("Transfer", (event) => {
  // ...})
this.emitter.on("?", (event) => {
  // all un-decodeable events})
```

receipt

获取已被网络接受的交易收据。 如果交易尚未确认，则返回空值。
交易事件日志是 ABI 编码

参数	实例
txid	string
	交易 ID

@return	Promise<IContractSendReceipt>
	交易收据, 带有事件日志.

```
const txid = "62fecfd27d71ddb260ac48c73c8f0f87e96d0b3a598ed2c2251caa4e6f9a9d97"const
receipt = await qrcToken.receipt(txid)
console.log(JSON.stringify(receipt, null, 2))
```

实例输出:


```
{
  "blockHash": "af37cb8d9905521542243005fadc9f18c1498c9823e35fa277ea1c37174c289a",
  "blockNumber": 83981,
  "transactionHash":
"62fecfd27d71ddb260ac48c73c8f0f87e96d0b3a598ed2c2251caa4e6f9a9d97",
  "transactionIndex": 28,
  "from": "57142e3bcf000f28890b5d979afc7ea90204e1de",
  "to": "49665919e437a4bedb92faa45ed33ebb5a33ee63",
  "cumulativeGasUsed": 37029,
  "gasUsed": 37029,
  "contractAddress": "49665919e437a4bedb92faa45ed33ebb5a33ee63",
  "logs": [
    {
      "type": "Transfer",
      "from": "57142e3bcf000f28890b5d979afc7ea90204e1de",
      "to": "c0ed80283c53c300c31c2bda6eca841e53cb6a21",
      "value": "1ba5add5700"
    }
  ],
  "rawlogs": [
    {
      "address": "49665919e437a4bedb92faa45ed33ebb5a33ee63",
      "topics": [
        "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
        "000000000000000000000000057142e3bcf000f28890b5d979afc7ea90204e1de",
        "000000000000000000000000c0ed80283c53c300c31c2bda6eca841e53cb6a21"
      ],
      "data": "0000000000000000000000000000000000000000000000001ba5add5700"
    }
  ]}
```

GkcRPC

这是一个用于直接访问 Gkc RPC API 的 JSON-RPC 客户端。它不会为你处理任何 ABI 编码或解码。
有需要的话你可以把 RPC 用户名和密码包含到 URL 里。在例子中，用户名是 gkc，密码是 test.
GkcRPC 类有一些在合约抽象内部使用的未公开的方法。 考虑将来可能会发生变化的任何未经证实的不受支持的内容。 现在，rawCall 是唯一的公共 API。 注意:GkcRPC 类有一些没文档的 public 方法在 Contract 抽象层内部使用到了. 你要考虑到之后可能不支持的无文档的内容. 现在 rawCall 是唯一发布的 API.

参数	实例
url	string
	gkc RPC 服务的 URL

```
const rpc = new GkcRPC('http://gkc:test@localhost:48803');
```

rawCall

发起一个 JSON-RPC 1.0 方法调用, 返回调用结果. 如果 JSON API 返回不是 200 HTTP 结果，则抛出错误。

调用 getinfo RPC 方法以获取 Gkc 区块链的基本信息：

```
const info = await rpc.rawCall("getinfo")
console.log(info)
```

getinfo 的输出:

```
{ version: 2040100,
  protocolversion: 70912,
  walletversion: 61000,
  balance: 49.81594884,
  zerocoinbalance: 0,
  blocks: 231853,
  timeoffset: -7,
  connections: 6,
  proxy: '',
  difficulty: 112161.8552178001,
  testnet: false,
  moneysupply: 157403898.84518296,
  zGKCsupply:
   { '1': 0,
     '5': 0,
     '10': 0,
     '50': 0,
     '100': 0,
     '500': 0,
```

```
    '1000': 0,
    '5000': 0,
    total: 0 },
  keypoololdest: 1576304108,
  keypoolsize: 1001,
  paytxfee: 0,
  relayfee: 0.0001,
  'staking status': 'Staking Not Active',
  errors: '' }
```

使用 try...catch 处理错误:

```
async function main() {
  try {
    const result = await rpc.rawCall("unknown-method-hohoho")
  } catch (err) {
    console.log("err", err)
  }}
```

All RPC 方法

gkc 支持的所有 RPC 方法 .

```
== Blockchain ==
callcontract "address" "data" ( address )
findserial "serial"
getaccountinfo "address"
getaddressbalance startheight endheight address1 [address2] [...]
getaddressexplorer address
getbestblockhash
getblock "hash" ( verbose )
getblockbalance blockhash [address] [address] [...]
getblockchaininfo
getblockcount
getblockhash index
getblockhashexplorer index
getblockheader "hash" ( verbose )
getblocksinfoexplorer from to
getchaintips
getdifficulty
getfeeinfo blocks
getinvalid
getmempoolinfo
getqueryexplorer hash
```

```
getrawmempool ( verbose )
getstorage "address"
gettokeninfo "address"
gettransactionreceipt "hash"
gettxexplorer txid
gettxout "txid" n ( includemempool )
gettxoutsetinfo
hashstateandutxo
listcontracts (start maxDisplay)
searchlogs <fromBlock> <toBlock> (address) (topics)
verifychain ( numblocks )

== Control ==
getinfo
help ( "command" )
stop

== Generating ==
getgenerate
gethashespersec
posminer (on/off)
setgenerate generate ( genproclimit )

== Gkc ==
checkbudgets
createmasternodekey
getbudgetinfo ( "proposal" )
getbudgetprojection
getbudgetvotes "proposal-name"
getcrp
getmasternodecount
getmasternodeoutputs
getmasternodescores ( blocks )
getmasternodestatus
getmasternodewinners ( blocks "filter" )
getnextsuperblock
getpoolinfo
listmasternodeconf ( "filter" )
listmasternodes ( "filter" )
masternode "command"...
masternodeconnect "address"
masternodecurrent
masternodedebug
mnbudget "command"... ( "passphrase" )
```

```
mnbudgetrawvote "masternode-tx-hash" masternode-tx-index "proposal-hash" yes|no time
"vote-sig"
mnbudgetvote "local|many|alias" "votehash" "yes|no" ( "alias" )
mnfinalbudget "command"... ( "passphrase" )
mnsync "status|reset"
Obfuscation is not supported any more. Use Zerocoin
preparebudget "proposal-name" "url" payment-count block-start "gkc-address"
monthy-payment
spork <name> [<value>]
startmasternode "local|all|many|missing|disabled|alias" lockwallet ( "alias" )
submitbudget "proposal-name" "url" payment-count block-start "gkc-address"
monthy-payment "fee-tx"

== Mining ==
getblocktemplate ( "jsonrequestobject" )
getmininginfo
getnetworkhashps ( blocks height )
prioritisetransaction <txid> <priority delta> <fee delta>
reservebalance ( reserve amount )
submitblock "hexdata" ( "jsonparametersobject" )

== Network ==
addnode "node" "add|remove|onetry"
banappnode (on/off) (label)
clearbanned
disconnectnode "node"
getaddednodeinfo dns ( "node" )
getconnectioncount
getnettotals
getnetworkinfo
getpeerinfo
listbanned
ping
setban "ip(/netmask)" "add|remove" (bantime) (absolute)

== Rawtransactions ==
createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
decoderawtransaction "hexstring"
decodescript "hex"
fromhexaddress "hexaddress"
gethexaddress "address"
getkeyid "address"
getrawtransaction "txid" ( verbose )
sendrawtransaction "hexstring" ( allowhighfees )
```

```
signrawtransaction "hexstring"
( [{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...]
["privatekey1",...] sighasht
ype )

== Util ==
createmultisig nrequired ["key",...]
estimatefee nblocks
estimatepriority nblocks
validateaddress "gkcaddress"
verifymessage "gkcaddress" "signature" "message"

== Wallet ==
addblackagent <agentid>
addmultisigaddress nrequired ["key",...] ( "account" )
autocombinerewards true|false ( threshold )
backupwallet "destination"
bip38decrypt "gkcaddress"
bip38encrypt "gkcaddress"
createcontract "bytecode" (gaslimit gasprice "senderaddress" broadcast)
createentrustagent (name)
deprive txid
dumpprivkey "gkcaddress"
dumpwallet "filename"
encryptwallet "passphrase"
entrust agentid amount
filteradtx "txid"
getaccount "gkcaddress"
getaccountaddress "account"
getadadmin "gkcaddress"
getaddressdpos gkcaddress
getaddressesbyaccount "account"
getagent agentid [blockheight]
getbalance ( "account" minconf includeWatchonly )
getcurrentseason
getdposnameofblock blockhash
getentrustment
getextenddata height key
getmnemoniccode
getnewaddress ( "account" )
getrawblock blockheight
getrawchangeaddress
getreceivedbyaccount "account" ( minconf )
getreceivedbyaddress "gkcaddress" ( minconf )
```

```
getstakesplitthreshold
getstakingstatus
gettokenbalance <tokenid> <account>
gettransaction "txid" ( includeWatchonly )
getunconfirmedbalance
getwalletinfo
importaddress "address" ( "label" rescan )
importprivkey "gkcprivkey" ( "label" rescan )
importwallet "filename"
keypoolrefill ( newsize )
listaccounts ( minconf includeWatchonly)
listad (offset=0) (count=20)
listaddressgroupings
listadfilter (offset=0) (count=20)
listagents (verbose)
listblackagents
listdeprives
listentrustrewards
listentrusts
listlockunspent
listmyagentid
listreceivedbyaccount ( minconf includeempty includeWatchonly)
listreceivedbyaddress ( minconf includeempty includeWatchonly)
listsinceblock ( "blockhash" target-confirmations includeWatchonly)
listsysmsg (offset=0) (count=20)
listtokentransactions <first> <last> (tokenid)
listtransactions ( "account" count from includeWatchonly)
listunspent ( minconf maxconf  ["address",...] )
lockunspent unlock [{"txid":"txid","vout":n},...]
move "fromaccount" "toaccount" amount ( minconf "comment" )
multisend <command>
resignagent
selectutxotosend address amount minutxo
selectutxotosendmany minutxo {"address":amount,...} ( minconf "comment" )
sendadvertise "title" "author" "text" ("link" "extend")
sendextenddata [{"key":"name","value":"data"},...] senderaddress
sendfrom "fromaccount" "togkcaddress" amount ( minconf "comment" "comment-to" )
sendmany "fromaccount" {"address":amount,...} ( minconf "comment" )
sendsysmessage "message"
sendtoaddress "gkcaddress" amount ( "comment" "comment-to" )
sendtoaddressix "gkcaddress" amount ( "comment" "comment-to" )
sendtocontract "contractaddress" "data" (amount gaslimit gasprice senderaddress
broadcast)
setaccount "gkcaddress" "account"
```

```
setstakesplitthreshold value
settxfee amount
signmessage "gkcaddress" "message"
transfertoken <tokenid> <from> <to> <amount>

== Zerocoin ==
exportzerocoins include_spent ( denomination )
getarchivedzerocoin
getspentzerocoinamount hexstring index
getzerocoinbalance
importzerocoins importdata
listmintedzerocoins
listspentzerocoins
listzerocoinamounts
mintzerocoin <amount>
reconsiderzerocoins
resetmintzerocoin <extended_search>
resetspentzerocoin
spendzerocoin <amount> <mintchange [true|false]> <minimizechange [true|false]>
<securitylevel [1-100]> <address>
```

实例: getblockcount

返回最长的区块链的块数。

```
const result = await rpc.rawCall("getblockcount")
```

结果

```
85687
```

实例: getnewaddress

返回接收付款的新 Gkc 地址。可能对要为用户生成存款地址的交易所有用。

```
const result = await rpc.rawCall("getnewaddress")
```

结果

```
QSnrDTj4UNcRwKdhY8sUZEd74VzwqeAddW
```

实例: gethexaddress

把一个 base58 pubkeyhash 地址转化成 16 进制地址用于智能合约。

```
const result = await rpc.rawCall("gethexaddress", ["GfKrofVYEjWAeaATJreij9k8TVRucrLxmL
"])
```

结果

```
eba34612bcfcb40b25ba3d83a0e6299f9d932c3e
```

实例: gettransactionreceipt

获得确认交易的收据。

```
const txid = "62fecfd27d71ddb260ac48c73c8f0f87e96d0b3a598ed2c2251caa4e6f9a9d97"const
result = await rpc.rawCall("gettransactionreceipt", [txid])
```

结果


```
[
  {
    "blockHash": "af37cb8d99055215422430005fadc9f18c1498c9823e35fa277ea1c37174c289a",
    "blockNumber": 83981,
    "transactionHash":
"62fecfd27d71ddb260ac48c73c8f0f87e96d0b3a598ed2c2251caa4e6f9a9d97",
    "transactionIndex": 28,
    "from": "57142e3bcf000f28890b5d979afc7ea90204e1de",
    "to": "49665919e437a4bedb92faa45ed33ebb5a33ee63",
    "cumulativeGasUsed": 37029,
    "gasUsed": 37029,
    "contractAddress": "49665919e437a4bedb92faa45ed33ebb5a33ee63",
    "log": [
      {
        "address": "49665919e437a4bedb92faa45ed33ebb5a33ee63",
        "topics": [
          "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
          "00000000000000000000000057142e3bcf000f28890b5d979afc7ea90204e1de",
          "000000000000000000000000c0ed80283c53c300c31c2bda6eca841e53cb6a21"
        ],
        "data": "0000000000000000000000000000000000000000000000001ba5add5700"
      }
    ]
```

```
  }]
```

类型词典

IContractInfo

与部署合约交互所需的最少部署信息。

```
export interface IContractInfo {
  /**
   * 合约的 ABI 定义, solc 生成.
   */
  abi: IABIMethod[]

  /**
   * 合约地址
   */
  address: string

  /**
   * 合约所有者的地址
   */
  sender?: string
}
```

IContractCallResult

调用一个合约方法的返回结果，带有解码的输出和日志。
Contract#call 的返回类型.

```
export interface IContractCallResult extends IRPCCallContractResult {
  /**
   * ABI 解码的输出
   */
  outputs: any[]
```

```
  /**
   * ABI 解码的日志
   */
  logs: Array<IDecodedSolidityEvent | null>
}
export interface IRPCCallContractResult {
  address: string
  executionResult: IExecutionResult,
  transactionReceipt: {
    stateRoot: string,
    gasUsed: string,
    bloom: string,
    log: any[],
  }}
export interface IExecutionResult {
  gasUsed: number,
  excepted: string,
  newAddress: string,
  output: string,
  codeDeposit: number,
  gasRefunded: number,
  depositSize: number,
  gasForDeposit: number,
}
```

实例:


```
{
  "address": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
  "executionResult": {
    "gasUsed": 39306,
    "excepted": "None",
    "newAddress": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
    "output": "0000000000000000000000000000000000000000000000000000000000000001",
    "codeDeposit": 0,
    "gasRefunded": 0,
    "depositSize": 0,
    "gasForDeposit": 0
  },
  "transactionReceipt": {
    "stateRoot": "9922edb770bd700a212427d3bc0764a9fed953a987952b2619b8a78dac7498aa",
    "gasUsed": 39306,
    "bloom":
"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
40000000000000000000000000000000000000000000000000000000000000200000000000080000000000000
```

```
0000000000000000000000000000000000000020000000200000000008000000000000040000000010
0000000000000000000000000000000000000000000000000000000000000000000000000000000008000000
0080000000000000000000000000000000000000000000000000000000000201000000000000000000000
0000000000200000000000000000200000000000000000000000000000000000000000000000000020
000000000000000",
    "log": [
      {
        "address": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
        "topics": [
          "0f6798a560793a54c3bcfe86a93cde1e73087d944c0ea20544137d4121396885",
          "000000000000000000000000dcd32b87270aeb980333213da2549c9907e09e94"
        ],
        "data": "00000000000000000000000000000000000000000000000000000000000003e8"
      },
      {
        "address": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
        "topics": [
          "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
          "0000000000000000000000000000000000000000000000000000000000000000",
          "000000000000000000000000dcd32b87270aeb980333213da2549c9907e09e94"
        ],
        "data": "00000000000000000000000000000000000000000000000000000000000003e8"
      }
    ]
  },
  "outputs": [
    true
  ],
  "logs": [
    {
      "type": "Mint",
      "to": "dcd32b87270aeb980333213da2549c9907e09e94",
      "amount": "3e8"
    },
    {
      "type": "Transfer",
      "from": "0000000000000000000000000000000000000000",
      "to": "dcd32b87270aeb980333213da2549c9907e09e94",
      "value": "3e8"
    }
  ]}
```


IContractSendRequestOptions

Contract#send 的配置项

```
/**
 * `send` 合约方法的配置项.
 */
export interface IContractSendRequestOptions {
  /**
   * 要发送的 GKC 数. 例如 0.1, 默认: 0
   */
  amount?: number | string

  /**
   * gasLimit, 默认: 200000, 最大: 40000000
   */
  gasLimit?: number

  /**
   * 每 gas 的 Gkc 价格, 默认: 0.00000001, 最小:0.00000001
   */
  gasPrice?: number | string

  /**
   * 发送者的 gkc 地址
   */
  senderAddress?: string
}
```

IContractSendResult

返回 Contract#send 的值。
confirm 方法用来等待交易确认。
confirm 方法的参数:

参数	类型
n	number
	可选 须等待的确认数
callback	IContractSendConfirmationHandler
	可选 回调函数，每次确认都会调用

回调值为:

参数	类型

updatedTx	IRPCGetTransactionResult
	关于提交给网络的交易的基本信息
receipt	IContractSendReceipt
	关于已确认交易的其他信息

参考

- IRPCGetTransactionResult
- IContractSendReceipt

```
const tx = await contract.send(method, args)await tx.confirm(3, (updatedTx, receipt)
=> {
  /// ...})
```

IRPCGetTransactionResult

关于提交给网络的交易的基本信息。

```
export interface IRPCGetTransactionResult {
  amount: number,
  fee: number,
  confirmations: number,
  blockhash: string,
  blockindex: number,
  blocktime: number,
  txid: string,
  walletconflicts: any[],
  time: number,
  timereceived: number,
  "bip125-replaceable": "no" | "yes" | "unknown",
  details: any[]
  hex: string,
}
```

IContractSendReceipt

Contract#send 的合约收据, 带有解码的事件日志

参考

- IRPCGetTransactionReceiptBase

```
export interface IContractSendReceipt extends IRPCGetTransactionReceiptBase {
  /**
```

```
   * 使用 ABI 解码的日志
   */
  logs: IDecodedLog[],

  /**
   * 未解码的日志
   */
  rawlogs: ITransactionLog[],}
/**
 * 解码的 Solidity 事件日志
 */export interface IDecodedLog {
  /**
   * 事件日志名称
   */
  type: string

  /**
   * 键值映射作为事件日志参数
   */
  [key: string]: any
}
```

实例


```
{
  "blockHash": "3b53ad132c26f9c30e5be9f664573428dad8b52e167becea4428d6903cb32740",
  "blockNumber": 13917,
  "transactionHash":
"79338589bb75e1865be889142890a4e25d3b9dbd454ce3f3c2614587c85e2ed3",
  "transactionIndex": 1,
  "from": "dcd32b87270aeb980333213da2549c9907e09e94",
  "to": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
  "cumulativeGasUsed": 39306,
  "gasUsed": 39306,
  "contractAddress": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
  "logs": [
    {
      "type": "Mint",
      "to": "dcd32b87270aeb980333213da2549c9907e09e94",
      "amount": "7d0"
    },
    {
      "type": "Transfer",
      "from": "0000000000000000000000000000000000000000",
      "to": "dcd32b87270aeb980333213da2549c9907e09e94",
```

```
      "value": "7d0"
    }
  ],
  "rawlogs": [
    {
      "address": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
      "topics": [
        "0f6798a560793a54c3bcfe86a93cde1e73087d944c0ea20544137d4121396885",
        "000000000000000000000000dcd32b87270aeb980333213da2549c9907e09e94"
      ],
      "data": "00000000000000000000000000000000000000000000000000000000000007d0"
    },
    {
      "address": "a778c05f1d0f70f1133f4bbf78c1a9a7bf84aed3",
      "topics": [
        "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
        "0000000000000000000000000000000000000000000000000000000000000000",
        "000000000000000000000000dcd32b87270aeb980333213da2549c9907e09e94"
      ],
      "data": "00000000000000000000000000000000000000000000000000000000000007d0"
    }
  ]}
```


IRPCSearchLogsRequest

```ts
export interface IRPCSearchLogsRequest {
    /**
     * 查找日志的开始块号
     */
    fromBlock?: number | "latest";
    /**
     * 查找日志的停止块号
     */
    toBlock?: number;
    /**
     * 地址或地址列表
     */
    addresses?: string[];

    topics?: Array<string | null>;
    /**
     * 日志返回前的最少确认数
```

```
     */
    minconf?: number;
}
```

IContractEventLogs

查询合约事件日志的结果。
要查询尚未出现的新日志，请在查询事件日志时将 nextblock 用作 startBlock： * IContractEventLog

```
/**
 * 查询合约事件日志的结果。
 */
export interface IContractEventLogs {
  /**
   * 事件日志，ABI 解码.
   */
  entries: IContractEventLog[]

  /**
   * 返回的事件日志数
   */
  count: number

  /**
   * 要开始查询新事件日志的块号
   */
  nextblock: number
}
```

IContractEventLog

一条解码的合约事件日志

```
export interface IContractEventLog extends IRPCGetTransactionReceiptResult {
  /**
   * Solidity 事件，ABI 解码. 如果没有找到 ABI 定义，为 Null
   */
  event?: IDecodedSolidityEvent
}

/**
```

```
 *  gkc 返回的原始日志数据，不是 ABI 解码
 */
export interface IRPCGetTransactionReceiptResult extends IRPCGetTransactionReceiptBase
{
  /**
   * 日志
   */
  log: ITransactionLog[]
}
/**
 * gkc 返回的交易收据
 */
export interface IRPCGetTransactionReceiptBase {
  blockHash: string
  blockNumber: number

  transactionHash: string
  transactionIndex: number

  from: string
  to: string

  cumulativeGasUsed: number
  gasUsed: number

  contractAddress: string
}
```

实例


```
{
      "blockHash":
"2422ed6edf01bcf971ed5953892d2c3b5b6305ac5e0b46c622d20411c093a185",
      "blockNumber": 227523,
      "transactionHash":
"e9b1c278ec6cb9090a027f8380cef035c91e11e18713070f63d1c9b7a653ef4f",
      "transactionIndex": 2,
      "from": "eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
      "to": "28a855f4153b072ffd744725da1b652486056338",
      "cumulativeGasUsed": 58321,
      "gasUsed": 58321,
      "contractAddress": "28a855f4153b072ffd744725da1b652486056338",
      "excepted": "0",
      "log": [
        {
```

```
        "address": "28a855f4153b072ffd744725da1b652486056338",
        "topics": [
          "ddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
          "000000000000000000000000eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
          "000000000000000000000000c99042ffb7db7ddda4aab04cce545483e32163a5"
        ],
        "data": "00000000000000000000000000000000000000000000000000000000000004d2"
      }
    ],
    "event": {
      "_value": "4d2",
      "_from": "eba34612bcfcb40b25ba3d83a0e6299f9d932c3e",
      "_to": "c99042ffb7db7ddda4aab04cce545483e32163a5",
      "type": "Transfer"
    }
}
```


IDecodedSolidityEvent

解码的 Solidity 事件日志。 事件参数存储在键值映射中。

```typescript
/**
 * 一个解码的 Solidity 事件日志
 */
export interface IDecodedSolidityEvent {
  /**
   * 事件名称
   */
  type: string

  /**
   * 键值映射作为事件日志参数
   */
  [key: string]: any
}
```

Example


```
{
  "type": "Transfer",
  "from": "0000000000000000000000000000000000000000",
  "to": "dcd32b87270aeb980333213da2549c9907e09e94",
  "value": "3e8"}
```

IRPCGetTransactionReceiptBase

网络接受的交易收据。它由 gettransactionreceipt RPC 调用返回。

```
export interface IRPCGetTransactionReceiptBase {
  blockHash: string
  blockNumber: number

  transactionHash: string
  transactionIndex: number

  from: string
  to: string

  cumulativeGasUsed: number
  gasUsed: number

  contractAddress: string
}
```

IContractsRepoData

合约相关信息
可以使用开发工具 solar 自动生成。
样例 solar.json.

```
export interface IContractsRepoData {
  /**
   * 部署合约的相关信息
   */
  contracts: {
    [key: string]: IContractInfo,
  },

  /**
   * 部署库的相关信息
   */
  libraries: {
    [key: string]: IContractInfo,
  }
```

```
  /**
   * 部署合约/库引用的合约信息，但未部署
   */
  related: {
    [key: string]: {
      abi: IABIMethod[],
    },
  }}
/**
 * 与部署合约进行交互所需的最少部署信息。
 */export interface IContractInfo {
  /**
   * 合约的 ABI 定义，solc 生成.
   */
  abi: IABIMethod[]

  /**
   * 合约地址
   */
  address: string

  /**
   * 合约所有者的地址
   */
  sender?: string}
export interface IABIMethod {
  name: string,
  type: string,
  payable: boolean,
  inputs: IABIInput[],
  outputs: IABIOutput[],
  constant: boolean,
  anonymous: boolean,
}
```