Grampus Chain
Web3.js API 手册

一、综述

1:Web3.js 是以太坊提供的一个库，它封装了以太坊的 RPC 通信 API，提供了一系列与区块链交互方法，使程序与以太坊交互变得简单。Grampus chain 继承了该库的大部分功能。

2:以太坊节点通过 JSON-RPC 提供外部访问接口，Web3.js 是基于 Java 和 Node.js 的 JSON-RPC 封装。JSON-RPC 是一个无状态、轻量级的远程过程调用协议（RPC），与传输协议无关，可使用 socket、HTTP 或者其它协议，它使用 JSON（RFC4627）作为数据格式。

二、安装方法

npm install web3js --g

三、接口手册

目前用的接口手册根据 web3@1.0.0-beta.36（https://github.com/ethereum/web3.js/tree/1.0）进行改造。为了规避风险，禁止访问了一些函数，在手册里也会一并给出。为了简便起见，该文档省略了部分 API，主要以涉及交易相关和链相关，如有需要，可以单独联系。

grampus 测试网地址：http://119.3.43.136:8203

测试公私钥：

私钥 1： e9534f981ce834ffb36751d0d73dc480071c286b4a93854ec4388b96a4b95240

私钥 2: a06979342ac35f4b3997dc6796a964ad2759e13addd2c91ab9d6f13417b479f5

账户 1（对应私钥 1）： 0x6C2fD03faF77Be34B33356f40BC9bD837da38434

账户 2（对应私钥 2）： 0x41A42d58D08220aC70E41b35818281099fc4b976

Web3

```
var Web3 = require('web3');
```

setProvider

```
web3.setProvider(myProvider)
```

这个模块的设置会改变连接的服务器.

参数

1. Object - myProvider: 有效的 Provider.

返回

Boolean

测试用例（1.1）

```
var Web3 = require('web3');var web3 = new Web3('http://localhost:8545');   // orvar web3 = new Web3(new Web3.providers.HttpProvider('http://localhost:8545'));
```

web3.eth

getCoinbase

```
getCoinbase([callback])
```

获取 token 的存储地址——coinbase

返回

Promise 返回 String - bytes 20: 挖矿奖励的存储地址

测试用例（2.1）

```
web3.eth.getCoinbase().then(console.log);> "0x14ca04ff85747def87d6c6c566db84cc24e4643b"
```

isMining

```
web3.eth.isMining([callback])
```

测试是否挖矿

返回

Promise 返回 Boolean: true 或者 false.

测试用例（2.2）

```
web3.eth.isMining().then(console.log);> true
```

getGasPrice

```
web3.eth.getGasPrice([callback])
```

返回目前的 gas 费用. gasPrice 由最近几个块的平均产生.

返回

Promise 返回 String - Number string of the current gas price in wei.

测试用例 （2.3）

```
web3.eth.getGasPrice().then(console.log);> 33333333333 //单位是 wei
```

getAccounts

```
web3.eth.getAccounts([callback])
```

获取节点上的所有的账户信息

返回

Promise 返回 Array - An array of addresses controlled by node.

测试用例 （2.4）

```
web3.eth.getAccounts().then(console.log);
[ '0x14CA04Ff85747DEF87d6c6C566dB84Cc24e4643b',
  '0x349118dD4764b6335055582949a24A1d76DDad15',
  '0x31E9A02b34D54061Ac98EAcBa7cB214fbd392004',
  '0x619f889c5699394B9c5033BC85028eb4Af11Faa1',
  '0xcc42B2640e3325Ec69E075Ab4a1Bd006D82D51Da',
  '0xa04FCe8906c870a47A8A7F5F7547dfc4a62caE7a',
  '0x6d55C5cb3Dc1D183930ae046D044fDB89b862371',
  '0x84b7B529dc9289bc8445D999FDb6312874Eb1C42',
  '0xBCcBb5cab0946575366F6baCc50de3FCa8601D20',
  '0xcad7999e5dfF568848f8647E311aD45f04CfE0f1',
  '0x5602F505E9Aa8658A6083F06560ae4FD08c0F0e7',
  '0x68b60B35C787dE682c558D785cC727c24567b76B',
  '0x9fAaF288798b0cd16bA61E9868f09c4de29ef442',
  '0xa401314b5d6D4E7E7682b49B2755E3182714EBa8',
  '0x3d2c7340b5dc2D8852fB10b48319528f7E784347',
  '0x3DE74e01B740a772067Ae3a6FF3d0ef3d65435da',
  '0x0028949597e389b2a0EbF98c177B1D7B12E57075',
  '0x6C2B65c525814c68BF26a566b69d56237072d0e0',
  '0x0E6fdfa77ff38c8A24F1Ef4cA1c8ee55faa8E4ba',
  '0x620360EBD2a260E56173e701C9236629f916FAbf',
  '0x8829981b88180C9235f63dA74Aa5fcEa71790cbC' ]
```

getBlockNumber

```
web3.eth.getBlockNumber([callback])
```

返回目前的块编号(高度)

返回

Promise 返回 Number - 最新块编号（高度）.

测试用例 (2.5)

```
web3.eth.getBlockNumber().then(console.log);> 2935460
```

getBalance

```
web3.eth.getBalance(address [, defaultBlock] [, callback])
```

获取指定地址的账户余额.

参数

1. String - 需要获取的余额的地址.

返回

Promise 返回 String - 返回账户余额用 wei 来表示.

测试用例 (2.6)

```
web3.eth.getBalance('0x5602F505E9Aa8658A6083F06560ae4FD08c0F0e7').then(console.log)
```

getBlock

```
web3.eth.getBlock(blockHashOrBlockNumber [, 返回 TransactionObjects] [, callback])
```

获取区块信息

参数

1. String|Number - 区块数或者区块哈希. 也可以使用 "genesis", "latest" or "pending".

返回

Promise 返回 Object - The block object:

- number - Number: 块高度. null 表示还在 pending.
- coinbase - String: 表示挖矿所得所存储的地址
- hash 32 Bytes - String: 块哈希值. null 表示还在 pending.
- parentHash 32 Bytes - String: 父区块哈希.
- nonce 8 Bytes - String: Hash of the generated proof-of-work. null when its pending block.
- sha3Uncles 32 Bytes - String: 叔块的 sha3.
- logsBloom 256 Bytes - String: The bloom filter for the logs of the block. null when its pending block.
- transactionsRoot 32 Bytes - String: The root of the transaction trie of the block
- stateRoot 32 Bytes - String: The root of the final state trie of the block.
- miner - String: The address of the beneficiary to whom the mining rewards were given.
- difficulty - String: 难度.
- extraData - String: The “extra data” 值.
- size - Number: 块大小.
- gasLimit - Number: 块能容纳的最大 gas 费用.
- gasUsed - Number: 块中已经包含的 gas 费用.
- timestamp - Number: 时间戳.
- transactions - Array: 由哈希组成的交易记录 depending on the 返回 TransactionObjects 测试用例.
- uncles - Array:叔区块哈希.
- dposContest: dpos 信息，包括投票根 hash，候选人根 hash，周期根哈希等

测试用例 (2.7)

```
web3.eth.getBlock(33151).then(console.log);

> { coinbase: '0x14ca04ff85747def87d6c6c566db84cc24e4643b',
  difficulty: '1',
```

dposContext:
{ candidateRoot:
'0xd501ead71a77e72352758349555b015d2e1def3d067ad14a20dff7766672b774',
delegateRoot:
'0xa9de49e72847d31bd55e8be8035f59e7c50e16a07773633a36f2d8c1cd4cc445',
epochRoot:
'0x1d93d1f40fafe042a690f76588195b6e1fbb89b247983c749029654a1bb4d045',
mintCntRoot:
'0xfb193c13aff13085f5315db157670e3dc5f17b57a3d76255a4573ae23fd9049c',
voteRoot:
'0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421' },
extraData:
'0xd883010803846765746888676f312e31322e35856c696e757800000000000000a8e0a3b9928a162c30935f965a7da555166edbf5bdb9c0787a6c
fd688c5b4a36486c8d0a254f2d1e6616573c3b799ff388048e158a1160e1f501ee1dfd73018800',
gasLimit: 75398208,
gasUsed: 0,
hash:
'0xc7fa9b531864959f6810c8d20184884d9f5062d8ba4f0f82516d239fc7fa73b8',
logsBloom:
'0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000',
miner: '0x14CA04Ff85747DEF87d6c6C566dB84Cc24e4643b',
mixHash:
'0x0000000000000000000000000000000000000000000000000000000000000000',
nonce: '0x0000000000000000',
number: 33151,
parentHash:
'0x79d761eaa1d46908b29b52265f4d57a663badf0995c834199b2b96394a755a84',
receiptsRoot:
'0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
sha3Uncles:
'0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347',
size: 797,
stateRoot:
'0x72dd9892c0049c585cae891291974769f281b0f9029deeb77c596b84fc068cae',
timestamp: 1564576071,
totalDifficulty: '164223',
transactions: [],
transactionsRoot:

'0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',

uncles: [] }

getBlockTransactionCount

web3.eth.getBlockTransactionCount(blockHashOrBlockNumber [, callback])

返回在指定块里的交易数量.

参数

1. String|Number - 块高度或者块哈希. 也可以写成 "genesis", "latest" or "pending"在 default block 测试用例 .
2. Function - (optional) Optional callback, 返回 an error object as first 测试用例 and the result as second.

返回

Promise 返回 Number - 返回在这个指定块中的交易数量.

测试用例 (2.8)

web3.eth.getBlockTransactionCount('0xc7fa9b531864959f6810c8d20184884d9f5062d8ba4f0f82516d239fc7fa73b8').then(console.log);> 0

getTransaction

web3.eth.getTransaction(transactionHash [, callback])

获取指定哈希的交易.

参数

1. String - 指定哈希.
2. Function - (optional) Optional callback

返回

Promise 返回 Object - 返回一个哈希对应的交易 transactionHash:

- hash 32 Bytes - String: 交易 hash.
- nonce - Number: 交易计数.
- blockHash 32 Bytes - String: 区块 hash.
- blockNumber - Number: 被打包进块的高度.
- transactionIndex - Number: 在块中的第几个交易（从 0 开始）
- from - String: 发起者.
- to - String: 接受者
- value - String: 转账金额.
- gasPrice - String: gas 费用
- gas - Number: gas 数量.
- input - String: 输入（如不为空，可能是合约交易）.
- v,r,s: - String: 签名数据

测试用例 (2.9)

web3.eth.getTransaction('0xec56a2dc51cd10b64a8721f0f915ecdce9a765cafb0bc3c03819e25fd97bd8f4').then(console.log);

>{ blockHash:

'0xf097faff0930da7cbb3468db0fc4dbb32bad4fa040b9da0ed7e6e43930e2f676',

blockNumber: 2935884,

from: '0x6C2fD03faF77Be34B33356f40BC9bD837da38434',

gas: 21000,

gasPrice: '10000000000',

```
  hash:
   '0xec56a2dc51cd10b64a8721f0f915ecdce9a765cafb0bc3c03819e25fd97bd8f4',
  input: '0x',
  nonce: 0,
  to: '0x41A42d58D08220aC70E41b35818281099fc4b976',
  transactionIndex: 0,
  value: '10000000000000000000',
  v: '0x4594',
  r:
   '0x2e94014821d9b84125d46da7b3dad090879716bbf7fc75f5e9b428fdd254bf19',
  s:
   '0x208e446687e71a218f1118abf307283359b1eb73595ee46473878fb3383d30fa' }
```

getTransactionReceipt

```
web3.eth.getTransactionReceipt(hash [, callback])
```

通过输入交易哈希返回交易凭证.

参数

1. String - 交易哈希.

返回

Promise 返回 Object -返回交易 Object, 如果没有找到 Receipt 则返回 null :

- status - Boolean: TRUE 表示成功, FALSE, 表示失败.
- blockHash 32 Bytes - String: 交易所在的块哈希.
- blockNumber - Number: 交易所在的区块高度.
- transactionHash 32 Bytes - String: 交易哈希.
- transactionIndex- Number: 交易所在的块中位置（序号）.
- from - String: 发送者.
- to - String: 接受者。如果为空，是合约交易.
- contractAddress - String: 如果是合约，则会显示合约地址，如果是普通转账，则显示 null
- cumulativeGasUsed - Number: 执行完毕后所需要的所有 gas
- gasUsed- Number: 仅此交易产生的 gas 费用
- logs - Array: 日志记录.

测试用例 (2.10)

```
var receipt =
web3.eth.getTransactionReceipt('0xec56a2dc51cd10b64a8721f0f915ecdce9a765cafb0bc3c03819e25fd97bd8f4').then(console.log);
> { blockHash:
   '0xf097faff0930da7cbb3468db0fc4dbb32bad4fa040b9da0ed7e6e43930e2f676',
  blockNumber: 2935884,
  contractAddress: null,
  cumulativeGasUsed: 21000,
  from: '0x6c2fd03faf77be34b33356f40bc9bd837da38434',
  gasUsed: 21000,
  logs: [],
  logsBloom:
```

```
'0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000',
  status: true,
  to: '0x41a42d58d08220ac70e41b35818281099fc4b976',
  transactionHash:
   '0xec56a2dc51cd10b64a8721f0f915ecdce9a765cafb0bc3c03819e25fd97bd8f4',
  transactionIndex: 0 }
```

getTransactionCount

`web3.eth.getTransactionCount(address [, defaultBlock] [, callback])`

获取某个地址上的交易数.

参数

1. String - 需要获得交易数的地址.
2. Number|String - (optional) 如果不填这个参数，则默认用 web3.eth.defaultBlock.
3. Function - (optional) Optional callback, 返回 an error object as first 测试用例 and the result as second.

返回

Promise 返回 Number - 返回给定地址上的交易数.

测试用例（2.11）

```
web3.eth.getTransactionCount("0x6c2fd03faf77be34b33356f40bc9bd837da38434").then(console.log);> 1
```

sendSignedTransaction

`web3.eth.sendSignedTransaction(signedTransactionData [, callback])`

发送一个已经被签署交易，如何签署交易？请参见 using web3.eth.accounts.signTransaction.

参数

1. String - 签署的 16 进制数据
2. Function - (optional) Optional callback, 返回 an error object as first 测试用例 and the result as second.

返回

PromiEvent: A promise combined event emitter. Will be resolved when the transaction receipt is available.

Please see the 返回 values for web3.eth.sendTransaction for details.

测试用例（2.12）

```
web3.eth.sendSignedTransaction("0xf86509841000000082521894f0109fc8df283027b6285cc889f5aa624eac1f556400824594a01368a3f4f526
bd2ed13cae40544dc14809321a4b1a4e95be3ef0fdc7ce3e5eeda033823191dd2deb025a536ebee554a52423e477f8499d098d69de5d46c814a4d3
",function(err,hash){
  if (err){
console.log(err)}
else {
console.log(hash);}
```

```
})
>0xca1ba84a932b5000a064f03833d4e20b96957ae1b32f846e0e49935fc15b6234
```

web3.eth.accounts

web3.eth.accounts 包含了生成以太坊账户，签名.

create

web3.eth.accounts.create([entropy]);

Generates an account object with private key and public key.

参数

1. entropy - String (optional): 随机字符串来生成账户，如果不填写，则会随机生成

返回

Object - The account object with the following structure:

- address - string: 账户地址
- privateKey - string: 私钥，请勿分享私钥

测试用例（3.1）

```
var account = web3.eth.accounts.create();     //Creates the account (is an object)
console.log(account);     //show the object in the console
{ address: '0x76e79422Aa6B33D77C704121d059Bff7FF516474',
  privateKey: '0xfccfb081b354fa289ed55ed4931b78450be3e3d892cf2d0a83beedd4f51ae2d4',
  signTransaction: [Function: signTransaction],
  sign: [Function: sign],
  encrypt: [Function: encrypt] }
```

privateKeyToAccount

web3.eth.accounts.privateKeyToAccount(privateKey);

从私钥计算账户.

参数

1. privateKey - String: 需要计算的私钥.

返回

Object - 返回账户对象 structure seen here.

测试用例（3.2）

```
var toAccount = web3.eth.accounts.privateKeyToAccount('0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709');
console.log(toAccount);  //show the  ACCOUNT
{ address: '0xb8CE9ab6943e0eCED004cDe8e3bBed6568B2Fa01',
  privateKey: '0x348ce564d427a3311b6536bbcff9390d69395b06ed6c486954e971d960fe8709',
  signTransaction: [Function: signTransaction],
  sign: [Function: sign],
  encrypt: [Function: encrypt] }
```

signTransaction

web3.eth.accounts.signTransaction(tx, privateKey [, callback]);

用私钥签署一个交易.

参数

- nonce - String: (可选) nonce 用来签署交易. 如果不填，默认是 web3.eth.getTransactionCount().请特别注意，每次签署的 nonce 必须加一，如果填写了 nonce，但是小于 web3.eth.getTransactionCount().则会报错
- chainId - String: (可选) chainID 是链的标识，默认是 web3.eth.net.getId().

- to - String: (可选) 接收方的地址，如果发送是智能合约，则可以不填.
- input- String: (可选) 附带数据，如果是普通交易，而非智能合约，则可以不填写.
- value - String: (可选) 用 wei 来表征的转账金额.
- gasPrice - String: (可选) 设定的 gas 费用，如果不填写，则默认 web3.eth.gasPrice()
- gas - String: 交易提供的 gas 值.

1. privateKey - String: 用来签署的私钥.
2. callback - Function: (optional) Optional callback, 返回 an error object as first 测试用例 and the result as second.

返回

Promise 返回是一组 RLP 编码的数据:

- messageHash - String: 信息的哈希.
- r - String: 签名的前 32 位
- s - String: 签名的接下来 32 位
- v - String: 恢复值 + 27
- rawTransaction - String: RLP 编码，参见 web3.eth.sendSignedTransaction.

测试用例 (3.3)

```
var Web3 = require('web3');var web3 = new Web3('http://119.3.43.136:8203');


web3.eth.accounts.signTransaction({
                to: '0xf0109fc8df283027b6285cc889f5aa624eac1f55',
            value: '0x64',
                gasPrice: '0x10000000',
            gas: '0x5218',
                nonce: '0x9',
            chainId: '0x22b8',
                input : '0x0'
}, 'e9534f981ce834ffb36751d0d73dc480071c286b4a93854ec4388b96a4b95240')
.then(console.log);
>{ messageHash:
   '0x8d06c4d1e8eeb86769768b2fdb1e70ae062797ff0fd44a0fedaf8ed3d18c74a3',
  v: '0x4594',
  r:
   '0x1368a3f4f526bd2ed13cae40544dc14809321a4b1a4e95be3ef0fdc7ce3e5eed',
  s:
   '0x33823191dd2deb025a536ebee554a52423e477f8499d098d69de5d46c814a4d3',
  rawTransaction:

'0xf865098410000000082521894f0109fc8df283027b6285cc889f5aa624eac1f556400824594a01368a3f4f526bd2ed13cae40544dc14809321a4b1
a4e95be3ef0fdc7ce3e5eeda033823191dd2deb025a536ebee554a52423e477f8499d098d69de5d46c814a4d3',
  transactionHash:
   '0xca1ba84a932b5000a064f03833d4e20b96957ae1b32f846e0e49935fc15b6234' }
```

发送该交易可参见 web3.eth.sendSignedTransaction

encrypt

web3.eth.accounts.encrypt(privateKey, password);

用密码把私钥加密成 web3 keystore.

参数

1. privateKey - String: 私钥.
2. password - String: 用于加密的密码.

返回

Object: keyStore 对象.

测试用例（3.4）

```
web3.eth.accounts.encrypt('0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318', 'test!')> {
    version: 3,
    id: '04e9bcbb-96fa-497b-94d1-14df4cd20af6',
    address: '2c7536e3605d9c16a7a3d7b1898e529396a65c23',
    crypto: {
        ciphertext: 'a1c25da3ecde4e6a24f3697251dd15d6208520efc84ad97397e906e6df24d251',
        cipherparams: { iv: '2885df2b63f7ef247d753c82fa20038a' },
        cipher: 'aes-128-ctr',
        kdf: 'scrypt',
        kdfparams: {
            dklen: 32,
            salt: '4531b3c174cc3ff32a6a7a85d6761b410db674807b2d216d022318ceee50be10',
            n: 262144,
            r: 8,
            p: 1
        },
        mac: 'b8b010fff37f9ae5559a352a185e86f9b9c1d7f7a9f1bd4e82a5dd35468fc7f6'
}}
```

decrypt

web3.eth.accounts.decrypt(keystoreJsonV3, password);

用密码解 keystore.

参数

1. encryptedPrivateKey - String: 加密后的 keystore 密钥.
2. password - String: 密码.

返回

Object: 解密账户.

测试用例（3.5）

```
web3.eth.accounts.decrypt({
    version: 3,
    id: '04e9bcbb-96fa-497b-94d1-14df4cd20af6',
    address: '2c7536e3605d9c16a7a3d7b1898e529396a65c23',
    crypto: {
        ciphertext: 'a1c25da3ecde4e6a24f3697251dd15d6208520efc84ad97397e906e6df24d251',
        cipherparams: { iv: '2885df2b63f7ef247d753c82fa20038a' },
        cipher: 'aes-128-ctr',
```

```
            kdf: 'scrypt',
            kdfparams: {
                dklen: 32,
                salt: '4531b3c174cc3ff32a6a7a85d6761b410db674807b2d216d022318ceee50be10',
                n: 262144,
                r: 8,
                p: 1
            },
            mac: 'b8b010fff37f9ae5559a352a185e86f9b9c1d7f7a9f1bd4e82a5dd35468fc7f6'
    }}, 'test!');> {
    address: "0x2c7536E3605D9C16a7a3D7b1898e529396a65c23",
    privateKey: "0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318",
    signTransaction: function(tx){...},
    sign: function(data){...},
    encrypt: function(password){...}}
```