MOACdocs-chn Documentation

MOAC

2019 年 07 月 11 日

目录:

CHAPTER 1

墨客系统介绍

1.1 介绍

区块链技术、加密货币和智能合约都有改变开发人员构建去中心化应用程序（DApp）方式的巨大潜力，并且已经在改变全球商业运作模式。

自 2008 年比特币问世以来，以加密货币身份亮相的区块链技术，已奠定其通过分布式账本技术提供数字化金融交易的身份，并且成为了一种非常有效的价值储存手段。不需要依靠中心化货币当局，比特币使用单一的去中心化共识模型就能验证交易并保障安全性 ()。详情参阅 https://bitcoin.org/bitcoin.pdf 以阅读更多内容。

2015 年，以太坊等平台提出并且实现了在区块链平台上的智能合约，并进一步开创了 Decentralized Application（DApp）的概念。智能合约是一种计算机程序，在特定方或开发者定义的程序条件下，直接控制各方之间进行数字货币或有价资产的转移。详情可参阅 https://github.com/ethereum/wiki/wiki/White-Paper。

第二代区块链技术的 “DApp” 是一种去中心化的应用，不依赖中心服务器运行，但依赖类似 BitTorrent，Napster 和 Kazaa 的点对点（P2P）网络进行互连。P2P 网络要维持来自多个数据源的最佳数据传输。智能合约和 DApp 都使用区块链处理和存储数据，并提供额外的计算功能，与比特币这样的纯粹提供交易功能的第一代区块链技术已大为不同。

自比特币创建以来，区块链技术（即分布式平台和通证）已经得到长足发展，但在功能、性能、安全性和可扩展性方面尚需进一步加强才能真正获得用户接纳，从而最终使得区块链技术可以在广阔的商用领域满足真实商业需求。

墨客项目将试图通过一种开创性解决方案，将现有区块链最佳实践与新型可扩展群链技术相结合，从而显著推进智能合约性能。墨客项目所阐述的技术已不是停留在纸面上的纯理论探讨，而是已经在开发、测试并即将投入生产的真实落地技术。

1.2 墨客区块链

现有技术和区块链平台对于用户来说学习曲线非常陡峭，技术复杂的同时使用费用也很高昂——所有这些都会影响区块链技术的市场接受度以及可扩展性。现有平台交易处理速度很低，共识模型固定，并且无法快速适应开发人员不断增长的需求。迄今，区块链社区的挖矿高度集中，并且由于复杂性和硬件成本的问题，未能有效激励更多的新用户和感兴趣的消费者进入区块链领域。

这些区块链平台也彼此隔离，每个区块链平台上的通证和智能合约与其他区块链底层难以进行有效沟通。于是现有区块链市场根据不同的平台技术形成了各自封闭的圈子，其平台、技术、用户基础和行业也是彼此隔离的。

即使对于有经验的技术开发人员来说，构建新的区块链目前也极具挑战性。导致局面更加复杂的是大多数区块链底层技术很难升级；同时，不同底层技术使得整个区块链的用户被低效的分离到不同的区块链底层平台上。

墨客基金会（MOAC）通过开发分层分片的群链技术解决了现有区块链平台的低效问题。墨客平台采用先进的分层架构，可降低 DApp 开发人员的成本，提供可扩展性并降低开发复杂度，同时使用分片技术提高交易速度和交易量。墨客在其平台内利用群链技术，包括母链（基于工作量证明的底层区块链）和子链（使用多种共识机制的上层区块链），来支持多种共识机制的智能合约。墨客平台还具有跨链功能，不但支持墨客平台上不同子链间的互联，也支持与其他区块链底层及其上的加密货币的互联互通。

墨客首先提出并实现了为每个智能合约提供定制子链的区块链解决方案，比现有智能合约执行的解决方案效率更高，且扩展性更强。

墨客平台使用子链来实现智能合约的业务逻辑，从而避免了在同一条链上同时处理常规的区块链任务（如交易的共识和记录）和与业务紧密相关的逻辑（通过智能合约方式实现）。通过为每个智能合约提供为其定制的子链，开发人员可以自由选择最适合其使用场景的共识算法，并确定分配给智能合约的节点数量，从而可以支持更多的使用场景。智能合约的所有状态都保存在本地子链中，且可根据需要将数据写入母链。

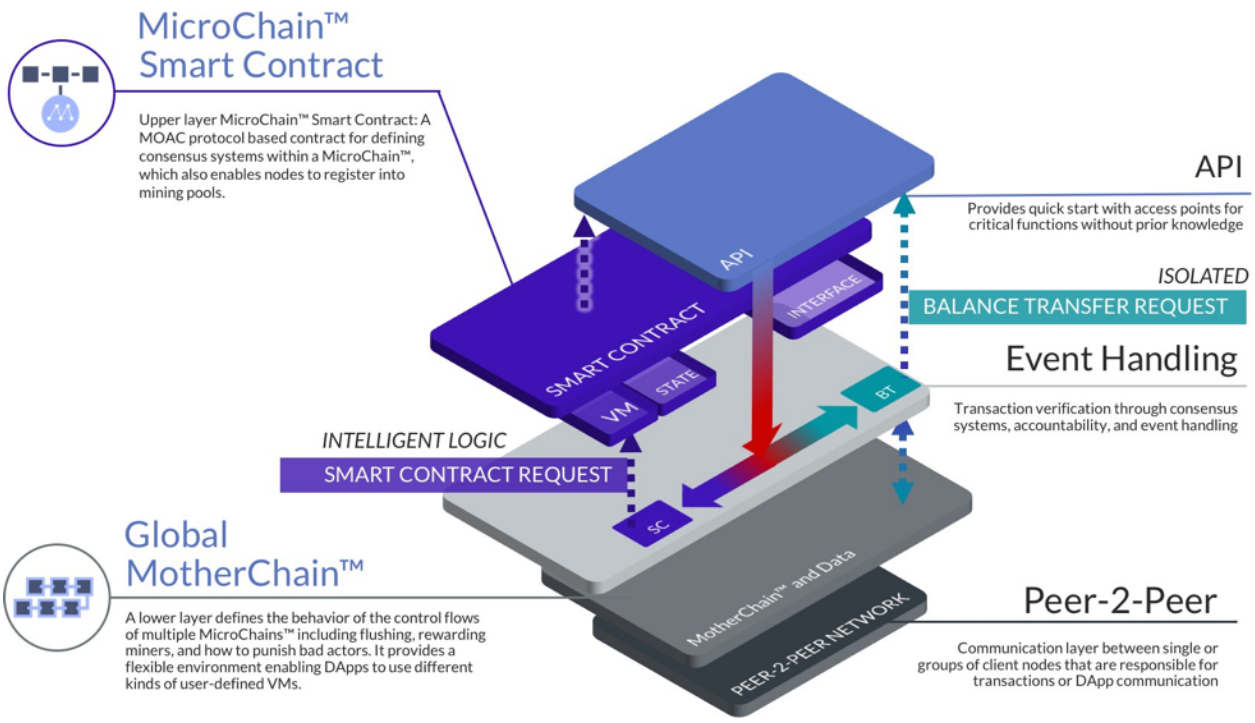
1. 母链
2. 子链

1.3 母链

母链是一个使用工作量证明为共识算法的区块链，可为智能合约和 DApp 解决数据存储和计算处理工作，是墨客的主要部分。

工作量证明 Proof-of-Work（PoW）算法是一种行之有效的措施，可以阻止并最终禁止第三方干扰，包括拒绝服务攻击，其他服务以及网络滥用（如垃圾邮件）。PoW 要求服务请求者提供一些工作量证明，通常是在规定的处理时间内由计算机完成特定处理任务，从而消除错误的系统威胁。在墨客平台上，母链是处理交易和其他区块链操作、共识和数据访问的公共区块链层。墨客还支持使用子链来实现其他共识算法。

Besides the POW consensus on transaction and data store set, each POW node is associated with one or more Smart Contract Server(SCS). SCS node could be local to the POW node, or it could be a remote node. The SCS identity is fully verifiable by the corresponding POW node, or Validating Node (VNODE).



Block is mined every 10s, with reward of 2 MOAC coins per block.

The reward schedule halves every 12,500,000 blocks, equivalent to approx. four years.

After block 15,000,000, the reward will be constant of 0.1 MOAC per block. See below.

We define 1 MOAC = 1,000,000 Sand. 1 Sand = 1,000 Xiao.

Block#	Reward (1 MOAC = 1,000,000 Sand)
1-12,500,000	2 MOAC
12,500,001 - 25,000,000	1 MOAC
25,000,000 - 37,500,000	0.5 MOAC
37,500,001 - 50,00,000	0.25 MOAC
50,000,001 - 62,500,000	0.125 MOAC
> 62,500,001	0.1 MOAC

Transaction fee is paid in two ways. One is through Transaction. The other is for Smart Contract or sub chain. Smart Contract Call cost is set lower than underlying transaction in purpose, thus encouraging the usage of SCS. This can alleviate the pressure on the underlying layer, and also benefit the SCS providers.

1.3.1 母链交易数据结构

墨客母链的交易数据结构与以太坊的相比，加入了三个新变量：

```
type txdata struct {
  AccountNonce   `json:"nonce"    gencodec:"required"`
  SystemContract `json:"syscnt" gencodec:"required"`
            `json:"gasPrice" gencodec:"required"`
  GasLimit       `json:"gas"      gencodec:"required"`
  Recipient      `json:"to"       rlp:"nil"` // nil means contract creation
  Amount         `json:"value"    gencodec:"required"`
  Payload        `json:"input"    gencodec:"required"`
  ShardingFlag   `json:"shardingFlag" gencodec:"required"`
  Via            `json:"via"       rlp:"nil"`

  // Signature values
  V `json:"v" gencodec:"required"`
  R `json:"r" gencodec:"required"`
  S `json:"s" gencodec:"required"`

}
```

1.3.2 母链节点客户端（VNODE）

VNODE client is used to form the MotherChain network. It used a Proof of Work consensus method similar to Ethereum. VNODE not only handles data storage and compute processing for smart contracts but also pass data for the MicroChain Dapps.

1.3.3 外部链接

1. MOAC
2. Mainnet Explorer
3. Testnet Explorer
4. MoacWalletOnline
5. TokenPocket
6. MOACMask

1.4 子链

子链是 MOAC 区块链中非常重要的一个模块。其主要目的在于分流母链中的业务逻辑，把一些比较繁琐的业务操作放在子链中执行。

子链架构于母链上层，每个子链都可以拥有自己独特的共识系统和算法。

例如，想要获得快速高并发的交易效果，您可以创建一个使用股权证明 Proof-of-Stake（POS）共识模式的子链。POS 是区块链网络旨在实现分布式共识的一种算法。

采取股权证明共识的区块链，依赖网络中的验证节点来检验交易，而不像严格的工作量证明 (PoW) 那样需要处理大量数据。在股权证明共识中，下一个区块的创建者是根据诸如持币量或币龄等因素（即股份）的随机算法来选择的。

股权证明系统的优点在于它可以完全扩展到企业量级的交易、能效高并支持多种交易。随着网络中的节点数量增加，其验证能力也会同步提升，在无需不断地访问母链的情况下，允许 DApp 子链开展小额交易。

除了股权证明和工作量证明外，墨客平台还可以支持额外的即插即用共识系统，例如活动证明（Prove of Activity）、销毁证明 (Proof of Burn)、耗时证明（Proof of Elapsed Time），存储证明（Proof of Capacity）等。

由于子链间是隔离的，因此它们可以在一个实例中为智能合约运行不同的虚拟机。这使子链能够启动各种业务逻辑并为 DApp 提供公共服务，包括部署类似 IPFS 的文件系统、构建用于数据存储的传感器网络，甚至还可以通过子链实现人工智能服务。

由于 DApp 部署在开发人员选择的虚拟机中，因此不需要额外编程。墨客可以通过更低的费用运行现有的以太坊智能合约，开发人员可以利用平台的 API 扩展现有的智能合约功能，而无需学习如何编程区块链。

通过使用自己独特的虚拟机和子链隔离每个智能合约，墨客平台可提高智能合约执行效率，并使低成本的处理费用成为可能。这显著降低了开发人员的部署成本，并使他们能够构建基于高交易量的 DApp。

子链本身是以智能合约的方式部署到 MOAC 母链上，其共识方式、节点组成和业务逻辑都在智能合约中定义。

- 子链节点控制合约 microchainprotocolbase，用于定义 SCS 节点共识方式和如何包括 SCS 节点矿工加入子链;
- 子链逻辑控制合约 microchainbase：用于子链控制逻辑，子链生成前和生成后的一系列控制逻辑;
- 子链 DAPP 智能合约：用于部署子链业务逻辑的合约，每个子链只能部署一个 DAPP 合约;

子链的验证过程由合约节点 SCS 完成，SCS 节点随机组合，支持动态增减。

子链支持分片，每个分片都能独立完成业务逻辑。

同时，在主链上，我们增加了代理的 Vnode 节点来保证子链的稳定性，这部分我们会在最后介绍。

当前，按功能分，有如下几种 SCS 节点类型：* 参与业务逻辑的 SCS * 用于业务监控的 SCS * 准备参与业务逻辑的 SCS

1.4.1 子链节点客户端（SCS）

智能合约服务器 Smart Contract Server(SCS) 客户端是支持子链运行的节点软件。SCS 通过 VNODE 代理节点接入墨客母链，每个运行的 SCS 可以支持多条子链，也可以动态接入不同 VNODE。

English tutorials:

Youtube

Youku

1.5 墨客通证

墨客通证的起始供应总量为 1.5 亿枚。墨客基金会本身持有 6300 万，墨客区块链科技公司持有 3100 万个通证。3100 万用于资助正在进行的群链技术发展。墨客通证的价值可以在 CoinMarketcap.com 上查询。

墨客通证最初在 2017 年 7 月以 ERC20 形式在以太坊平台发行。墨客主网在 2018 年 4 月上线以后，已经将以太坊上的 ERC20 合约关闭，按地址将 ERC20 通证全部 1：1 转换成为墨客主网的原生通证。

墨客区块链每年将通过挖矿产生 600 万个通证，进一步增加流通中的供应量。4 年后，产量减半至 300 万，接下来的 4 年再减半。到 2058 年，总供应量将达到 2.1 亿枚。

墨客的小写 moac 或者 mc 是墨客平台上原生通证的名字，如同以太坊的 ether，井通平台的 swt 一样。原生通证被用来支付母链上交易和合约执行的手续费，和子链的运行费用。可以参考*Gas 费用和 mc*.

1.5.1 计量系统

MOAC 具有用作 mc 单位的计量系统。mc 的最小单位，又称为基本单位为 Sha，1 mc = 1e+18 Sha，其它的额度有自己的名称。以下是额度名称的列表他们在沙的价值。按照共同的模式，mc 还指定货币的单位（1e + 18 或一个 quintillion Sha）。

Unit	Sha Value	Sha
sha	1 sha	1
Ksha (femtomc)	1e3 sha	1,000
Msha (picomc)	1e6 sha	1,000,000
Gsha (xiao)	1e9 sha	1,000,000,000
micromc (sand)	1e12 sha	1,000,000,000,000
millimc	1e15 sha	1,000,000,000,000,000
mc (moac)	1e18 sha	1,000,000,000,000,000,000

如何获取墨客通证

用户可以通过交易所交换墨客，目前主要交易所在这里可以找 here.；或者使用矿机参与墨客主网母链挖矿、子链挖矿，也可以通过参与墨客子链的活动；

交易墨客

墨客平台提供了一个在线钱包 MOAC Wallet 来方便用户使用简单的交易功能。

MOAC can also be transferred using the **MOAC console**.

```
> var sender = mc.accounts[0];
> var des = mc.accounts[1];
> var amount = chain3.toSha(0.01, "mc")
> mc.sendTransaction({from:sender, to:des, value: amount})
```

MOAC network, like Ethereum, use mc as a cryptofuel, commonly referred to as "gas". Beyond transaction fees, gas is a central part of every network request and requires the sender to pay for the computing resources consumed. The gas cost is dynamically calculated, based on the volume and complexity of the request and multiplied by the current gas price. Its value as a cryptofuel has the effect of increasing the stability and long-term demand for mc and MOAC as a whole.

Gas 费用和 mc

Gas is supposed to be the constant cost of network resources/utilisation. You want the real cost of sending a transaction to always be the same, so you can't really expect Gas to be issued, currencies in general are volatile.

So instead, we issue mc whose value is supposed to vary, but also implement a Gas Price in terms of MOAC. If the price of mc goes up, the Gas Price in terms of mc should go down to keep the real cost of Gas the same.

Gas has multiple associated terms with it: Gas Prices, Gas Cost, Gas Limit, and Gas Fees. The principle behind Gas is to have a stable value for how much a transaction or computation costs on the Ethereum network.

- Gas Cost is a static value for how much a computation costs in terms of Gas, and the intent is that the real value of the Gas never changes, so this cost should always stay stable over time.
- Gas Price is how much Gas costs in terms of another currency or token like MOAC. To stabilise the value of gas, the Gas Price is a floating value such that if the cost of tokens or currency fluctuates, the Gas Price changes to keep the same real value. The Gas Price is set by the equilibrium price of how much users are willing to spend, and how much processing nodes are willing to accept.
- Gas Limit is the maximum amount of Gas that can be used per block, it is considered the maximum computational load, transaction volume, or block size of a block, and miners can slowly change this value over time.
- Gas Fee is effectively the amount of Gas needed to be paid to run a particular transaction or program (called a contract). The Gas Fees of a block can be used to imply the computational load, transaction volume, or size of a block. The gas fees are paid to the miners (or bonded contractors in PoS).

1.6 项目路线图

1. 盘古 (Pangu)


2017
MOAC
JANUARY 2017
成功募集到$6,998,723 USD
JULY 2017
测试网发布
NOVEMBER 2017 已实现
• 墨客测试网发布
• 异步合约调用上线
2018
FEBRUARY 2018 已实现
• 墨客钱包测试版发布
• 智能合约测试
主网盘古版本上线
APRIL 2018 已实现
• 墨客主网上线
• 主网DAPP
• 母链节点挖矿开始
• 网络浏览器上线
女娲版本上线
AUGUST 2018 已实现
• 实现两种SCS服务: ProcWind（POS）和 FileStorm（IPFS）
• 认证系统上线
• 无币子链解决方案
• ATO模式
子链版本更新
DECEMBER 2018 已实现
• 子链分片系统发布
• 一键发链工具发布
• 完全去中心化的DAPP应用链间（MooDaDa）上线
2019
JUNE 2019
• 高性能存储系统在线
• 新SCS服务上线: PAS
• 数据交换服务上线
伏羲版本发布
DECEMBER 2019
• 功能强化和监测模块
• 物联网（IOT）模块发布
• 真随机数模块发布

内部版本识别号 *(Version)*：*0.8*

Release Date：3/31/2018

Major Progress：

- VNODE mining;
- SCS mining;
- System contract for auto trigger;
- Subchain Protocol contract for SCS miner registration;
- Subchain contract for Dapp configuration and flush control;

Documents:

2. 女娲（Nuwa）

内部版本识别号 *(Version)*：*1.0*

Release Date：7/30/2018

Major Progress：

- Fully functional VNODE to support MicroChain;
- Fully functional SCS server to support MicroChain；
- Enabled VNODE to get rewards from MicroChain mining;
- MicroChain protocol smart contract;
- MicroChain base smart contract that supports POS consensus;
- Fully functional MicroChain supports sharding;
- Supports FileStorm Protocol for IPFS MicroChain;
- Supports MicroChain without token;

3. 伏羲（Fuxi）

内部版本识别号 *(Version)*：*1.1*

Release Date：12/30/2019

Available features：

- Supports MicroChain with cross-chain features;
- Supports MicroChain with IOT mining features;

4. 神农（Shennong）

内部版本识别号 *(Version)*：*1.2*

Release Date：12/30/2020

Available features:

- VNODE consensus protocol upgrade;
- High performance (>10k TPS);
- Zero knowledge support;
- Data store/exchange market;

CHAPTER 2

墨客区块链系统和工具

2.1 安装母链节点

2.1.1 官方发布版本

最新的墨客软件可以从官方的发布地址下载 release link

Debian/Ubuntu/CentOS

1. Untar the file using tar, under the directory, run ./moac

To see the help, use ./moac –help

To enable the console, run: ./moac console

A mainnet directory will be created under $HOME/.moac/ and some info should be seen as:

```
INFO [04-24|11:24:26.506] 86161:IPC endpoint closed: /home/user/.moac/moac.ipc
```

from another terminal, run moac again to attach the running node

```
./moac attach
```

or

```
./moac attach $HOME/.moac/moac.ipc
```

2. from console prompt, create coinbase account

```
>personal.newAccount()
```

3. from console prompt, start mining by running

```
>miner.start()
```

4. from another terminal, run moac again to attach the running node

```
./moac attach
```

5. from prompt, load script

```
>loadScript("mctest.js")
```

6. check if miner has mined any moac by checking:

```
>mc.accounts
```

7. create another account

```
>personal.newAccount()
```

8. try send from one account to another:

```
>Send(mc.accounts[0], '', mc.accounts[1], 0.1)
```

WINDOWS

Unzip the file to a directory, under the directory, run moac.exe

To see the help, use moac.exe –help

To enable the console, run: moac.exe console

A mainnet directory will be created under and some info should be seen as:

2.1 查看 moac 帮助

打开命令（cmd）终端，转到墨客解压目录，在命令行中执行：

```
D:\moacPangu0.8.2-win>moac --help
```

显示帮助信息，包含但不限于以下内容：

查看 moac 帮助

打开命令（cmd）终端，转到墨客解压目录，在命令行中执行：

```
D:\ moacPangu0.8.2-win>moac --help
```

显示帮助信息，包含但不限于以下内容：

```
Start MOAC.... 2
NAME:
 moac - the MOAC-core command line interface
 Copyright 2017 The MOAC Authors
USAGE:
 moac [options] command [command options] [arguments...]
VERSION:
 0.8.2-develop-ed4070bf
MOAC CORE OPTIONS:
--config value                         TOML configuration file
--datadir "C:\Users\[userName]\AppData\Roaming\MoacNode" Data directory for the␣
↪databases and keystore
--keystore                             Directory for the keystore (default = inside the␣
↪datadir)
--nousb                                Disables monitoring for and managing USB hardware␣
↪wallets
--networkid value                      Network identifier (integer, 1=Pangu, 2=Testnet)␣
↪(default: 1)
--testnet                              MOAC test network: pre-configured proof-of-work test␣
↪network
```

2.2 运行节点

打开命令（cmd）终端，转到墨客当前目录，在命令行中执行：

```
D:\ moacPangu0.8.2-win>moac
```

显示如下信息：

至最后三行显示如下：

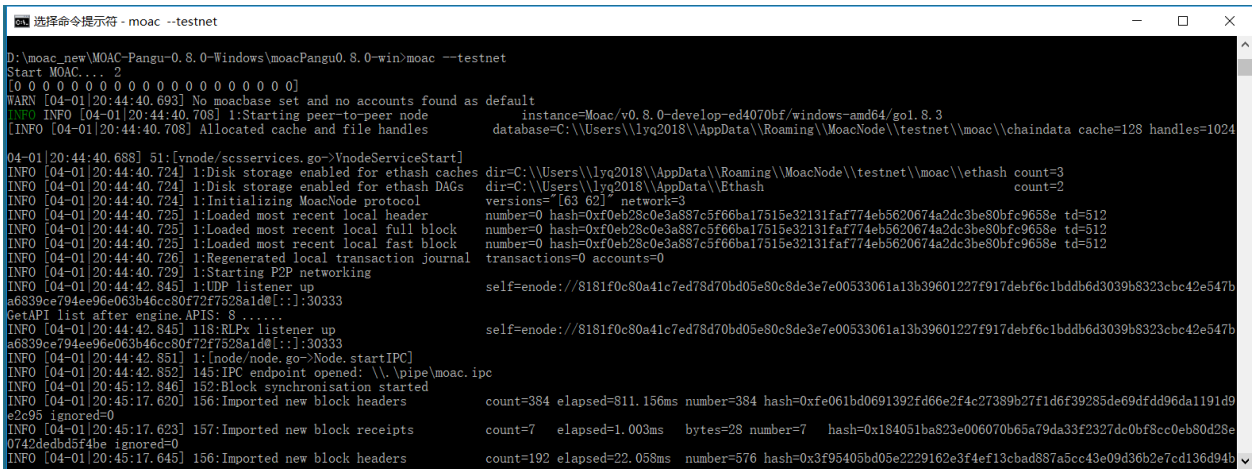


图 1: moac_install_win_0

```
INFO [04-01|20:44:42.851] 1:[node/node.go->Node.startIPC]
INFO [04-01|20:44:42.852] 145:IPC endpoint opened: \\.\pipe\moac.ipc
INFO [04-01|20:45:12.846] 152:Block synchronisation started
```

表示节点安装成功，如果网络正常，就开始同步区块。

系统将 MOAC 节点默认安装在目录:

```
C:\Users\[userName]\AppData\Roaming\MoacNode\
```

该目录下包含两个文件夹：moac 和 keystore。

2.3 进入 MOAC console 界面

系统关机或主动关闭运行中的节点后，如果需要重新启动节点，在命令行中执行:

```
D:\ moacPangu0.8.2-win>moac console
```

之后一直滚屏以同步区块数据。

打开另一个命令（cmd）终端，转到墨客当前目录，在命令行中执行:

```
D:\ moacPangu0.8.2-win>moac attach
```

该命令行不会主动滚屏，而是等待命令。

3. 挖矿

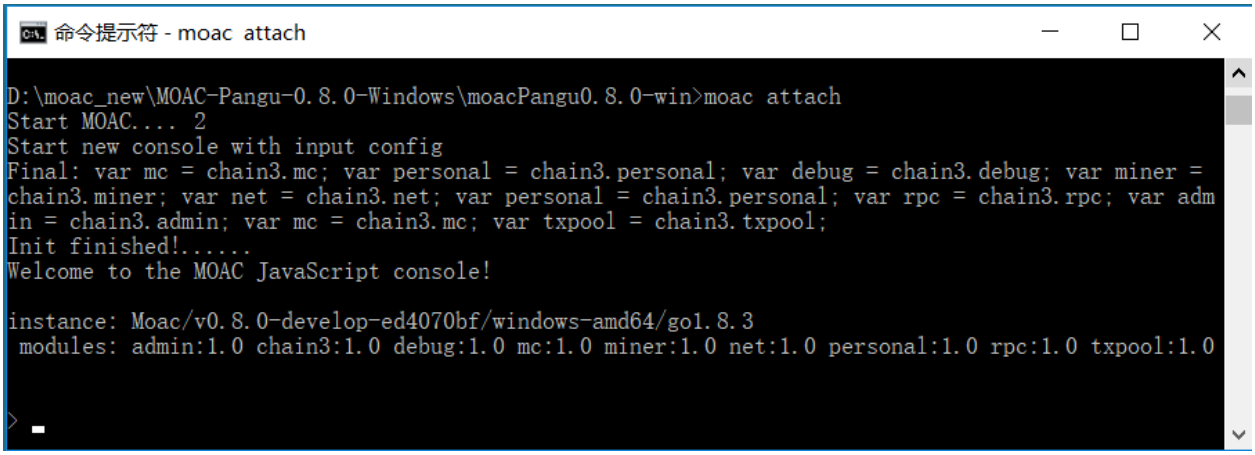


图 2: moac_install_win_1

3.1 建立新账户

挖矿前必须建立一个自己的账户。

进入 MOAC console 界面，执行命令：

```
> personal.newAccount()
```

系统会提示输入一个密码，例如”passwd”，并再次输入相同密码确认后，会显示一个以 0x 开头的字符串，即为 MOAC 帐号的公开地址。

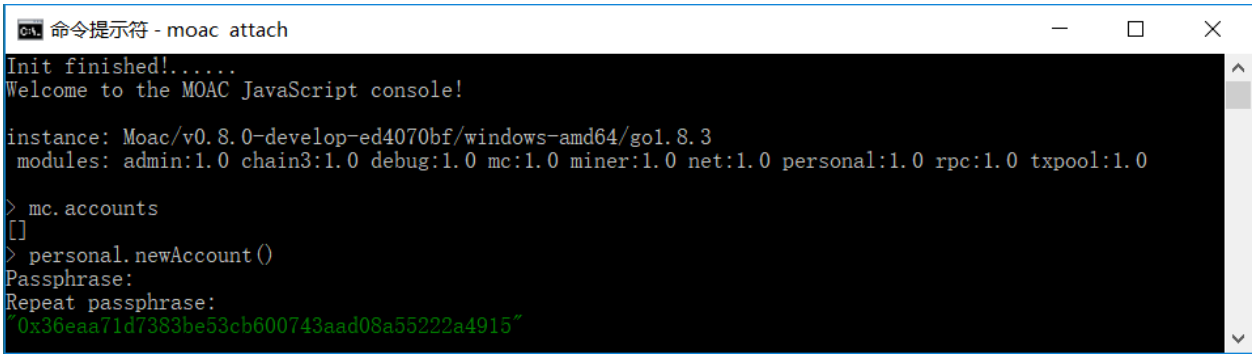


图 3: moac_install_win_2

系统同时会在以下目录：

```
C:\Users\[userName]\AppData\Roaming\MoacNode\testnet\keystore
```

记录一个账号文件。请保存好该文件，并牢记密码，之后用于解密帐号和操作。

3.2 查看账户

进入 MOAC console 界面，执行命令：

```
> mc.accounts
```

可以查看本节点下的所有账号。

3.3 查看账户余额

进入 MOAC console 界面，执行命令：

```
> mc.getBalance(mc.accounts[0])
```

可以查看本节点下的账号余额。0 表示第一个账户，也是默认挖矿账户。

或者：导入“mctest.js”的情况下（见 4.1），执行命令：

```
> checkBalance()
```

该命令用于查看当前节点所有账号的余额。

3.4 查看挖矿状态

进入 MOAC console 界面，执行命令：

```
> mc.mining
```

返回 true 表明节点正在挖矿，false 表明节点没有挖矿。

3.5 开始挖矿

进入 MOAC console 界面，执行命令：

```
> miner.start()
```

挖矿状态下，数据显示有明显不同。

挖到矿之后，可以查看余额

登录墨客区块链浏览器页面：http://explorer.moac.io。

在搜索栏输入你的挖矿账号地址，会显示该账号的余额等信息。

在搜索栏输入你挖到矿的区块号，会显示该区块的信息。

Miner 正是你的账号地址。

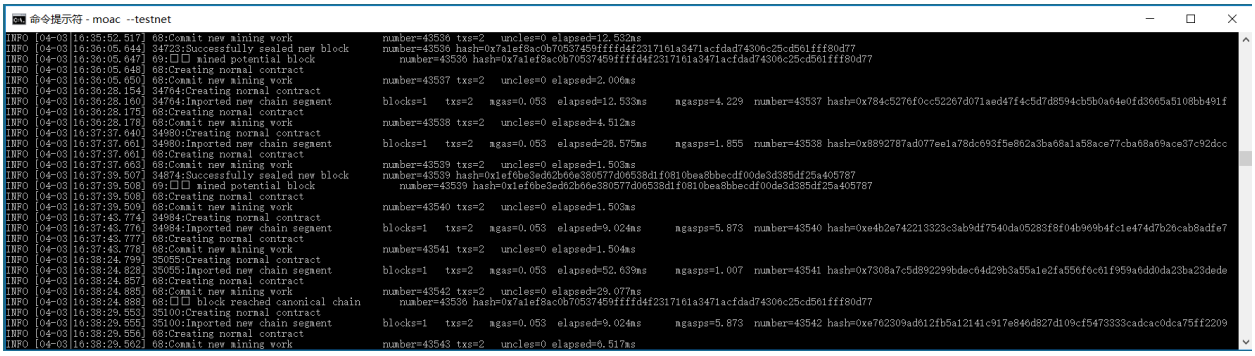


图 4: moac_install_win_4

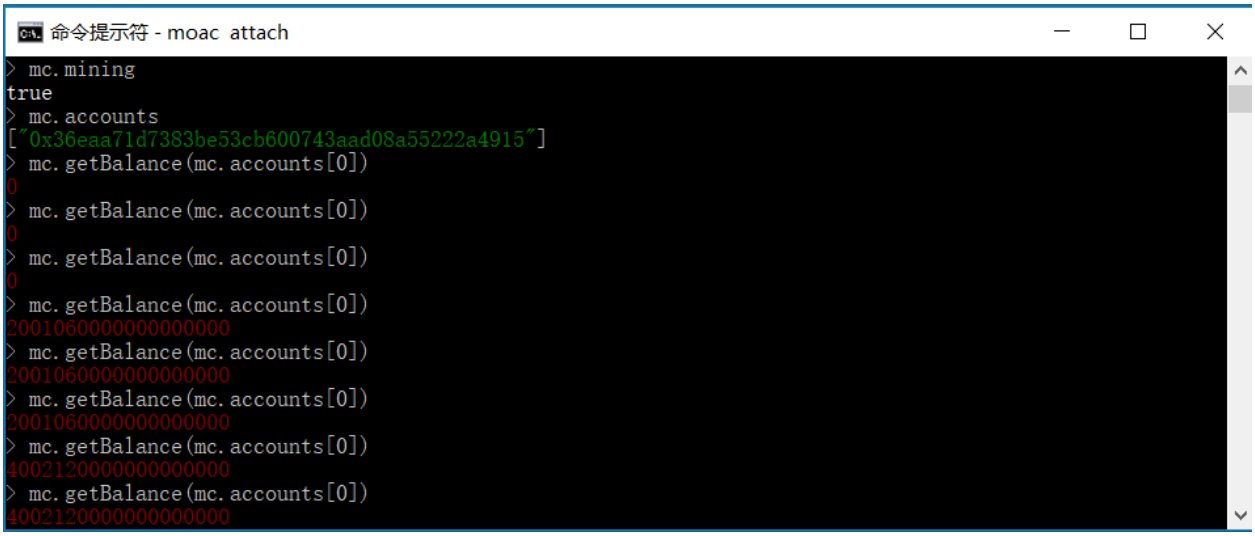


图 5: moac_install_win_5

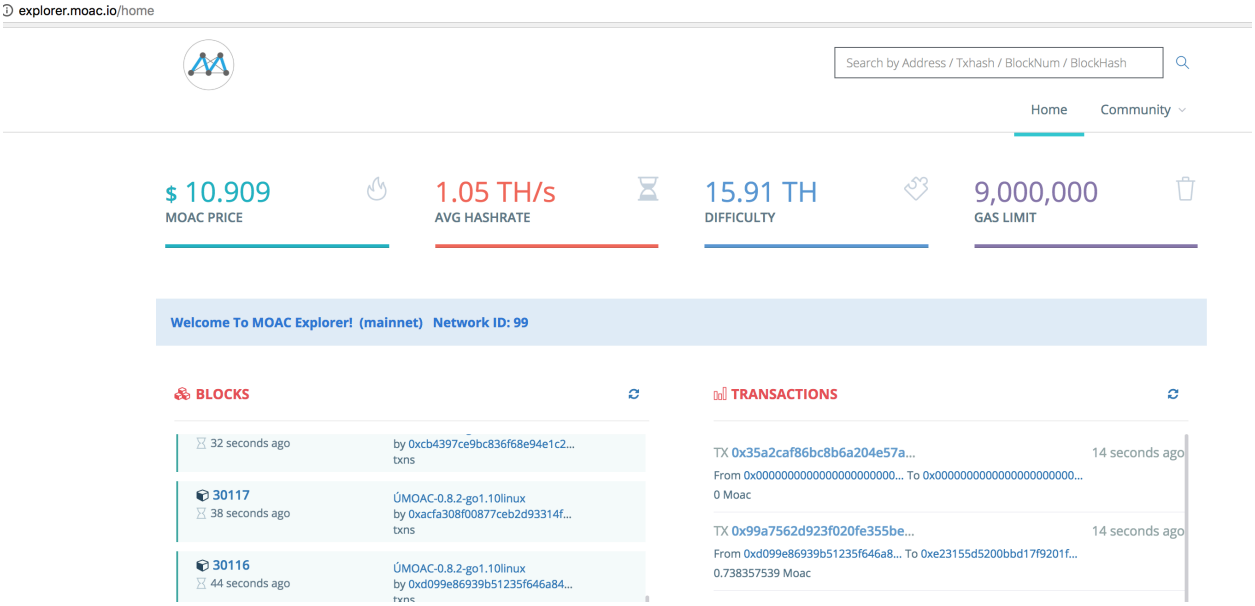


图 6: moac_explorer_0

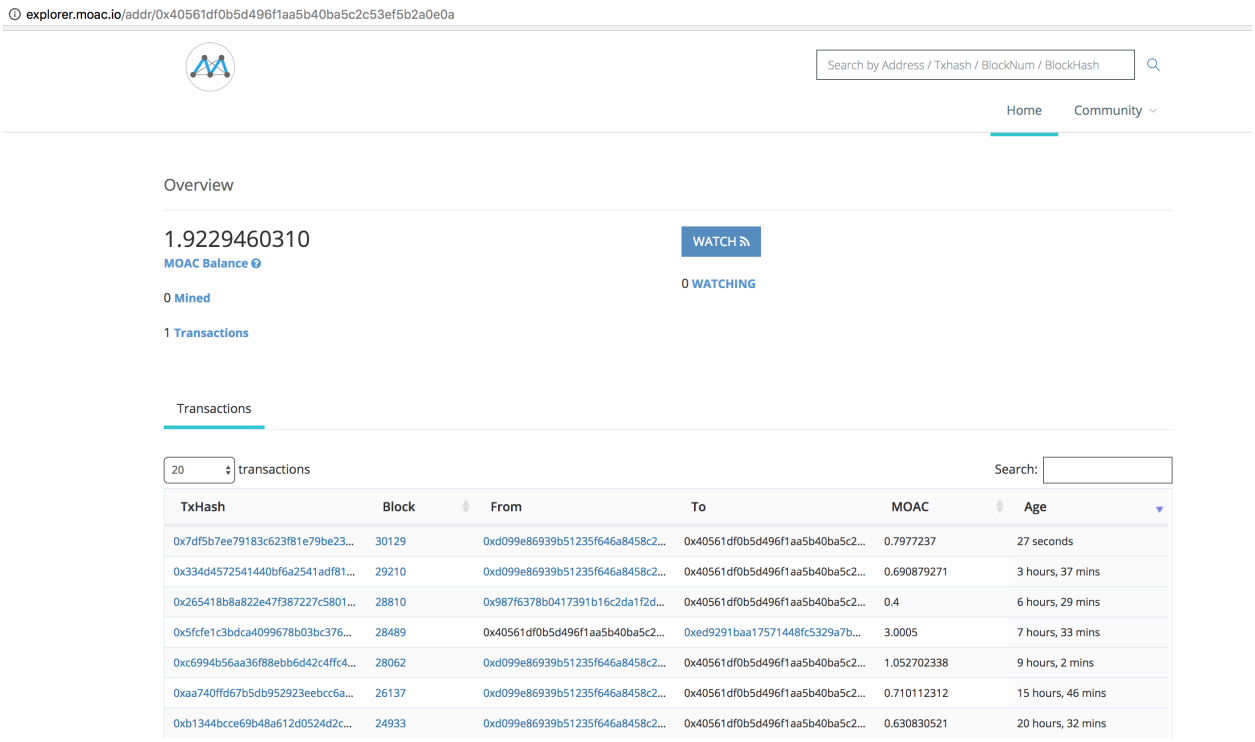


图 7: moac_explorer_1

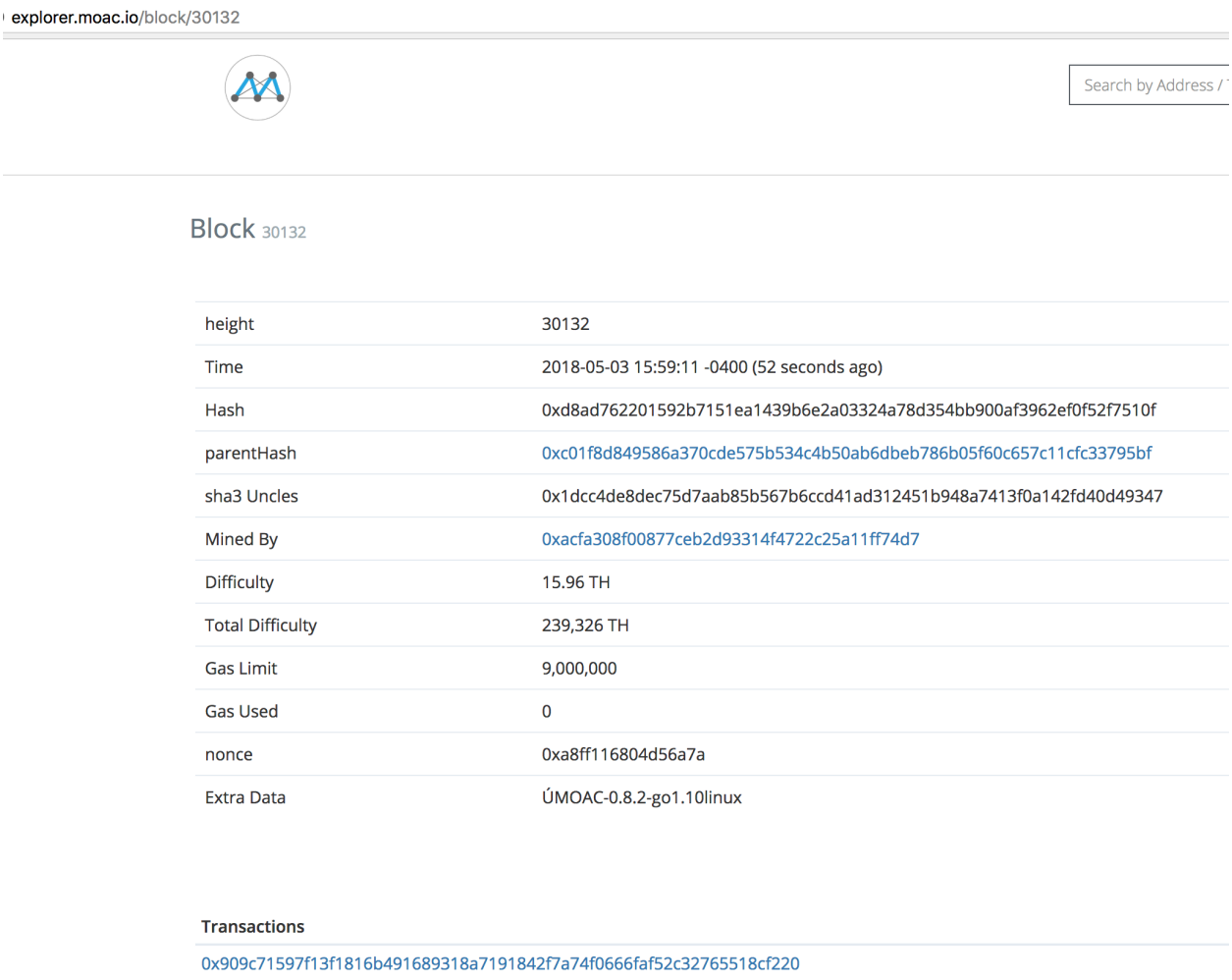


图 8: moac_explorer_2

3.6 停止挖矿

进入 MOAC console 界面，执行命令：

```
> miner.stop()
```

4. 交易

4.1 读入测试函数

部分功能程序存储在 mctest.js 里。

进入 MOAC console 界面，执行命令：

```
> loadScript("mctest.js")
```

4.2 交易条件

为执行交易，需要至少两个帐号，其中一个有足够的 mc。

如果没有目标账号，可以用步骤 2.3.1 的命令创建一个本地账号。并用命令：

```
> mc.accounts
```

显示当前节点中存储的账号，应该至少有一个挖矿账号。

4.3 交易

进入 MOAC console 界面，执行命令：

```
> Send(mc.accounts[0], 'passwd', mc.accounts[1], 0.1)
```

这个过程需要第一个账号的密码。比如’ passwd’，发送额为 0.1 mc。

在系统挖矿的情况下，发送应该在下一个区块产生时完成。

系统显示的是以 **sha（Sand）**为单位的余额，**1 mc = 1e18 sha**。

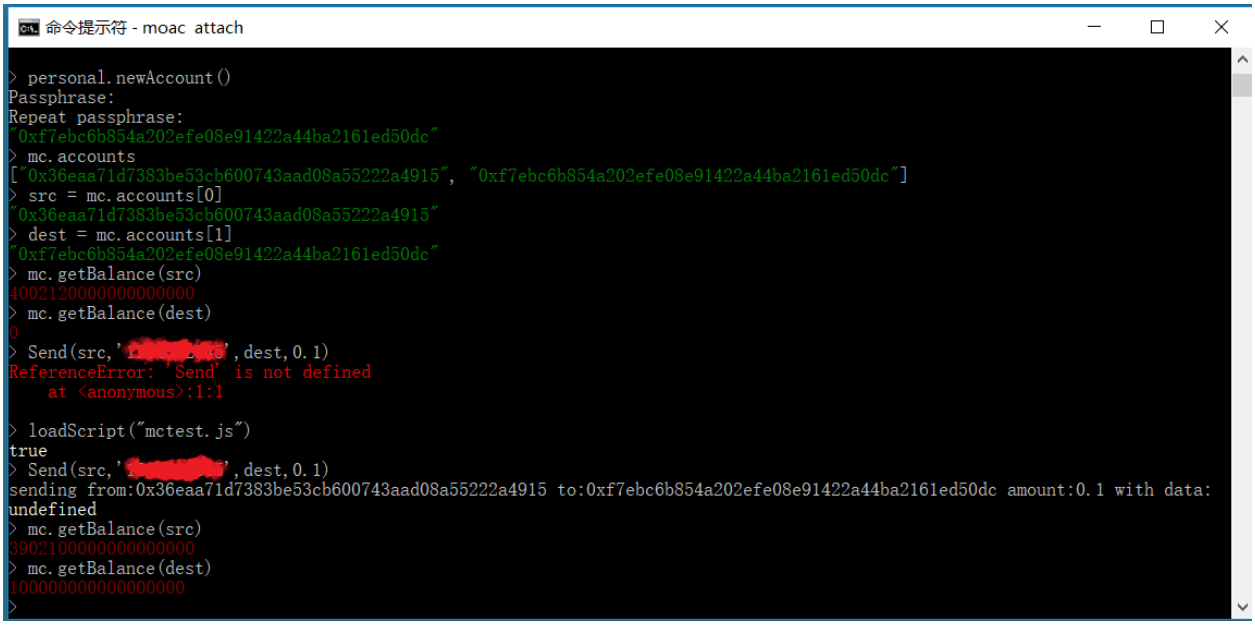


图 9: moac_install_win_6

2.2 母链节点命令行

2.2.1 Most Commonly Used Commands:

```
--testnet: connect to MOAC testnet (networkid = 101);

--rpc: initiate RPC service of HTTP, so as to nonlocally access the specific MOAC node␣
↪service;

--rpcaddr value: default as "localhost", only local accessibility; can be reset to "0.0.
↪0.0", so as to nonlocally access the specific MOAC node service, but the current RPC␣
↪service is based on HTTP using plaintext, so users need to be aware of the safety␣
↪risks;

--rpcport value: default as"8545", normally no changes are needed, users can use the␣
↪default port;

--rpcapi value: command RPC as to which API service will be open, default as "chain3,mc,
↪net", the commonly used will also be configured such as: personal,admin,debug,miner,
↪txpool,db,shh and so on, but because of RPC services' cleartext nature, if users␣
↪choose to use personal, they should be aware of the safety risks;
```

(下页继续)

(续上页)

```
--rpccorsdomain value: in general, if users wish to use this option, they can just use "
↪"; to be more specifical, users are recommended to search up "Cross-origin resource␣
↪sharing" to further understand some key phrases;

--jspath loadScript: default value is ".", the main directory of loadScript when␣
↪installing javascript file;
```

As shown below:

Initiate MOAC VNODEs and connect to mainnet (network ID = 99)

```
./moac
```

Through local ipc port connecting to MOAC mainnet nodes and start the command line

```
./moac attach
```

Initiate MOAC nodes and connect to testnet (network ID = 101)

```
./moac --testnet
```

Initiate MOAC testnet nodes and interactive command line

```
./moac --testnet console
```

Through local ipc port connecting to MOAC nodes

```
./moac attach /xx/xxx/moac.ipc
```

Through HTTP-based rpc port connecting to local or remote MOAC nodes

```
./moac attach http://xxx.xxx.xxx.xxx:8545
```

Initiate MOAC testnet nodes and RPC service

```
./moac --testnet --rpc
```

Initiate MOAC testnet nodes which is nonlocally accessible; personal and debug services can also be nonlocally used while providing cross-domain resource sharing service.

```
./moac --testnet --rpc --rpcaddr=0.0.0.0 --rpcapi="db,mc,net,chain3,personal,debug" --
↪rpccorsdomain=""
```

2.2.2 All command line parameters

On the command line, type:

```
$ ./moac help
```

or

```
$ ./moac --help
```

```
moac - the MOAC-core command line interface

Copyright 2017 The MOAC Authors
```

USAGE:

```
moac [options] command [command options] [arguments...]
```

VERSION:

1.0.9-stable-f98b7fea

```
account     Manage accounts
attach      Start an interactive JavaScript environment (connect to node)
bug         opens a window to report a bug on the moac repo
console     Start an interactive JavaScript environment
dump        Dump a specific block from storage
dumpconfig  Show configuration values
export      Export blockchain into file
import      Import a blockchain file
init        Bootstrap and initialize a new genesis block
js          Execute the specified JavaScript files
license     Display license information
makecache   Generate ethash verification cache (for testing)
makedag     Generate ethash mining DAG (for testing)
monitor     Monitor and visualize node metrics
removedb    Remove blockchain and state databases
version     Print version numbers
wallet      Manage MoacNode presale wallets
help, h     Shows a list of commands or help for one command
```

```
--config value                          TOML configuration file
--datadir "/Users/zpli/Library/MoacNode"  Data directory for the databases and keystore
```

(下页继续)

(续上页)

```
--keystore                                Directory for the keystore (default = inside␣
↪the datadir)
--nousb                                   Disables monitoring for and managing USB␣
↪hardware wallets
--networkid value                         Network identifier (integer, 99=mainnet,␣
↪101=testnet, 100=devnet) (default: 99)
--testnet                                 MOAC test network: pre-configured proof-of-
↪work test network
--dev                                     Developer mode: pre-configured private network␣
↪with several debugging flags
--syncmode "fast"                         Blockchain sync mode ("fast", "full", or "light
↪")
--mcstats value                           Reporting URL of a mcstats service␣
↪(nodename:secret@host:port)
--identity value                          Custom node name
--lightserv value                         Maximum percentage of time allowed for serving␣
↪LES requests (0-90) (default: 0)
--lightpeers value                        Maximum number of LES client peers (default:␣
↪20)
--lightkdf                                Reduce key-derivation RAM & CPU usage at some␣
↪expense of KDF strength
```

```
--ethash.cachedir                     Directory to store the ethash verification caches␣
↪(default = inside the datadir)
--ethash.cachesinmem value            Number of recent ethash caches to keep in memory␣
↪(16MB each) (default: 2)
--ethash.cachesondisk value           Number of recent ethash caches to keep on disk␣
↪(16MB each) (default: 3)
--ethash.dagdir "/Users/zpli/.ethash"  Directory to store the ethash mining DAGs␣
↪(default = inside home folder)
--ethash.dagsinmem value              Number of recent ethash mining DAGs to keep in␣
↪memory (1+GB each) (default: 1)
--ethash.dagsondisk value             Number of recent ethash mining DAGs to keep on␣
↪disk (1+GB each) (default: 2)
```

```
--txpool.nolocals           Disables price exemptions for locally submitted transactions
--txpool.journal value      Disk journal for local transaction to survive node restarts␣
↪(default: "transactions.rlp")
--txpool.rejournal value    Time interval to regenerate the local transaction journal␣
↪(default: 1h0m0s)
```

(下页继续)

(续上页)

```
--txpool.pricelimit value    Minimum gas price limit to enforce for acceptance into the␣
↪pool (default: 1)
--txpool.pricebump value     Price bump percentage to replace an already existing␣
↪transaction (default: 10)
--txpool.accountslots value  Minimum number of executable transaction slots guaranteed␣
↪per account (default: 16)
--txpool.globalslots value   Maximum number of executable transaction slots for all␣
↪accounts (default: 40960)
--txpool.accountqueue value  Maximum number of non-executable transaction slots␣
↪permitted per account (default: 64)
--txpool.globalqueue value   Maximum number of non-executable transaction slots for all␣
↪accounts (default: 1024)
--txpool.lifetime value      Maximum amount of time non-executable transaction are␣
↪queued (default: 3h0m0s)
```

```
--cache value            Megabytes of memory allocated to internal caching (min 16MB /␣
↪database forced) (default: 128)
--trie-cache-gens value  Number of trie node generations to keep in memory (default: 120)
```

```
--unlock value    Comma separated list of accounts to unlock
--password value  Password file to use for non-inteactive password input
```

```
--rpc                    Enable the HTTP-RPC server
--rpcaddr value          HTTP-RPC server listening interface (default: "localhost")
--rpcport value          HTTP-RPC server listening port (default: 8545)
--rpcapi value           API's offered over the HTTP-RPC interface
--ws                     Enable the WS-RPC server
--wsaddr value           WS-RPC server listening interface (default: "localhost")
--wsport value           WS-RPC server listening port (default: 8546)
--wsapi value            API's offered over the WS-RPC interface
--wsorigins value        Origins from which to accept websockets requests
--ipcdisable             Disable the IPC-RPC server
--ipcpath                Filename for IPC socket/pipe within the datadir (explicit paths␣
↪escape it)
--rpccorsdomain value  Comma separated list of domains from which to accept cross origin␣
↪requests (browser enforced)
--jspath loadScript    JavaScript root path for loadScript (default: ".")
--exec value           Execute JavaScript statement
--preload value        Comma separated list of JavaScript files to preload into the␣
↪console
```

(下页继续)

(续上页)

```
--bootnodes value     Comma separated enode URLs for P2P discovery bootstrap (set v4+v5␣
↪instead for light servers)
--bootnodesv4 value   Comma separated enode URLs for P2P v4 discovery bootstrap (light␣
↪server, full nodes)
--bootnodesv5 value   Comma separated enode URLs for P2P v5 discovery bootstrap (light␣
↪server, light nodes)
--port value          Network listening port (default: 30333)
--maxpeers value      Maximum number of network peers (network disabled if set to 0)␣
↪(default: 25)
--maxpendpeers value  Maximum number of pending connection attempts (defaults used if␣
↪set to 0) (default: 0)
--nat value           NAT port mapping mechanism (any|none|upnp|pmp|extip:<IP>)␣
↪(default: "any")
--nodiscover          Disables the peer discovery mechanism (manual peer addition)
--v5disc              Enables the experimental RLPx V5 (Topic Discovery) mechanism
--netrestrict value   Restricts network communication to the given IP networks (CIDR␣
↪masks)
--nodekey value       P2P node key file
--nodekeyhex value    P2P node key as hex (for testing)
```

```
--mine                    Enable mining
--minerthreads value      Number of CPU threads to use for mining (default: 8)
--moacbase value          Public address for block mining rewards (default = first␣
↪account created) (default: "0")
--targetgaslimit value    Target gas limit sets the artificial target gas floor for the␣
↪blocks to mine (default: 9000000)
 --gasprice "18000000000"  Minimal gas price to accept for mining a transactions
--extradata value         Block extra data set by the miner (default = client version)
```

```
--gpoblocks value      Number of recent blocks to check for gas prices (default: 10)
--gpopercentile value  Suggested gas price is the given percentile of a set of recent␣
↪transaction gas prices (default: 50)
```

```
--vmdebug  Record information useful for VM and contract debugging
```

```
--metrics                 Enable metrics collection and reporting
--fakepow                 Disables proof-of-work verification
```

(下页继续)

(续上页)

```
--nocompaction              Disables db compaction after import
--verbosity value           Logging verbosity: 0=silent, 1=error, 2=warn, 3=info, 4=debug,␣
↪5=detail (default: 3)
--vmodule value             Per-module verbosity: comma-separated list of <pattern>=<level>
↪ (e.g. mc/=5,p2p=4)
--backtrace value           Request a stack trace at a specific logging statement (e.g.
↪"block.go:271")
--debug                     Prepends log messages with call-site location (file and line␣
↪number)
--pprof                     Enable the pprof HTTP server
--pprofaddr value           pprof HTTP server listening interface (default: "127.0.0.1")
--pprofport value           pprof HTTP server listening port (default: 6060)
--memprofilerate value      Turn on memory profiling with the given rate (default: 524288)
--blockprofilerate value    Turn on block profiling with the given rate (default: 0)
--cpuprofile value          Write CPU profile to the given file
--trace value               Write execution trace to the given file
```

```
--fast      Enable fast syncing through state downloads
--light     Enable light client mode
--help, -h  show help
```

```
Copyright 2017-2018 MOAC Authors
```

2.3 交互命令行

墨客母链客户端使用了和以太坊类似的交互式命令行。用户可以在命令行（console）中执行内置的 JAVA script 命令或者利用脚本（script），输出结果显示在命令行中。这里使用的 chain3 对象，是 MOAC 参考以太坊，而开发的一套 javascript 库，目的是让应用程序能够与 MOAC 的 VNODE 和 SCS 节点进行通信。注意，这里有两层，moac 启动了一个 MOAC VNODE 节点，console 参数开启了一个 javascript 的控制台，这个控制台注入了 chain3.js 这个库，以使我们可以通过 chain3 对象与 MOAC VNODE 节点做交互。

2.3.1 Interactive use: the JSRE REPL Console

The MOAC VNODE CLI executable moac has a JavaScript console (a Read, Evaluate & Print Loop = REPL exposing the JSRE), which can be started with the console or attach subcommand. The console subcommands starts the moac node and then opens the console. The attach subcommand will not start the moac node but instead tries to open the console on a running moac instance.

```
$ moac console
$ moac attach
```

The attach node accepts an endpoint in case the moac node is running with a non default ipc endpoint or you would like to connect over the rpc interface.

```
$ moac attach ipc:/some/custom/path
$ moac attach http://127.0.0.1:8545
$ moac attach ws://127.0.0.1:8546
```

参数说明 –datadir 数据库和秘钥存储路径 –dev 开发环境，采用权威证明 (PoA)，开发帐号有预存金额，自动封装交易数据到区块链 –rpc 开启 HTTP-RPC 服务，默认 8545 端口 –rpccorsdomain 允许连接到 rpc 的 url 匹配模式*代表任意 console 启动一个交互式的 JavaScript 环境

Note that by default the moac node doesn’ t start the http and weboscket service and not all functionality is provided over these interfaces due to security reasons. These defaults can be overridden when the –rpcapi and –wsapi arguments when the moac node is started, or with admin.startRPC and admin.startWS.

If you need log information, start with:

```
$ moac --verbosity 4 console 2>> /tmp/vnode.log
```

Otherwise mute your logs, so that it does not pollute your console:

```
$ moac console 2>> /dev/null
```

or

```
$ moac --verbosity 0 console
```

Geth has support to load custom JavaScript files into the console through the –preload argument. This can be used to load often used functions, setup chain3 contract objects, or

```
$ moac --preload "/my/scripts/folder/utils.js,/my/scripts/folder/contracts.js" console
```

2.3.2 Non-interactive use: JSRE script mode

It’ s also possible to execute files to the JavaScript interpreter. The console and attach subcommand accept the –exec argument which is a javascript statement.

```
$ moac --exec "mc.blockNumber" attach
```

This prints the current block number of a running moac instance.

Or execute a local script with more complex statements on a remote node over http:

```
$ moac --exec 'loadScript("/tmp/checkbalances.js")' attach http://127.0.0.1:8545
$ moac --jspath "/tmp" --exec 'loadScript("checkbalances.js")' attach http://127.0.0.
↪1:8545
```

Use the –jspath <path/to/my/js/root> to set a libdir for your js scripts. Parameters to loadScript() with no absolute path will be understood relative to this directory.

You can exit the console cleanly by typing exit or simply with CTRL-C.

2.3.3 Caveat

Just like go-ethereum, MOAC' s JSRE uses the Otto JS VM which has some limitations:

"use strict" will parse, but does nothing. The regular expression engine (re2/regexp) is not fully compatible with the ECMA5 specification. Note that the other known limitation of Otto (namely the lack of timers) is taken care of. Ethereum JSRE implements both setTimeout and setInterval. In addition to this, the console provides admin.sleep(seconds) as well as a "blocktime sleep" method admin.sleepBlocks(number).

Since chain3.js uses the bignumber.js library (MIT Expat Licence), it is also autoloded.

2.3.4 Timers

In addition to the full functionality of JS (as per ECMA5), the moac JSRE is augmented with various timers. It implements setInterval, clearInterval, setTimeout, clearTimeout you may be used to using in browser windows. It also provides implementation for admin.sleep(seconds) and a block based timer, admin.sleepBlocks(n) which sleeps till the number of new blocks added is equal to or greater than n, think "wait for n confirmations" .

2.3.5 Management APIs

Beside the official DApp API interface the VNODE has support for additional management API' s. These API' s are offered using JSON-RPC and follow the same conventions as used in the DApp API. The VNODE package comes with a console client which has support for all additional API' s.

2.4 JSON RPC 接口命令

JSON <http://json.org/> is a lightweight data-interchange format. It can represent numbers, strings, ordered sequences of values, and collections of name/value pairs.

JSON-RPC <http://www.jsonrpc.org/specification>'is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over

sockets, over HTTP, or in many various message passing environments. It uses JSON (‘RFC 4627 <http://www.ietf.org/rfc/rfc4627.txt>) as data format.

MOAC JSON-RPC has some compatibility with ETHEREUM JSON-RPC,

Chain3	Web3.js
chain3	web3
mc	eth
net	net
admin	admin
personal	personal
vnode	n/a
scs	n/a

MOAC has two additional JSON-RPC objects (vnode and scs) for VNODE and SCS services. These RPCs are supported by Nuwa 1.0.4 version and later. The most recent RPC is Nuwa 1.0.9.

2.4.1 JSON-RPC Endpoint

默认 JSON-RPC 的接入为:

Client	URL
Go	http://localhost:8545

在 VNODE 和 SCS 监听节点启动时，可以使用 HTTP JSON-RPC 选项 “–rpc“ 命令来开启:

–rpc: 启用 HTTP 的 RPC 服务，以便非本机访问该 MOAC 节点服务;

```
moac --rpc
```

change the default port (8545) and listing address (localhost) with:

```
moac --rpc --rpcaddr <ip> --rpcport <portnumber>
```

If accessing the RPC from a browser, CORS will need to be enabled with the appropriate domain set. Otherwise, JavaScript calls are limit by the same-origin policy and requests will fail:

```
moac --rpc --rpccorsdomain "http://localhost:3000"
```

–rpcapi value: 指定 RPC 要开放的 API 服务，默认为” chain3,mc,net” ，常用的一般还会配置比如 personal,admin,debug,miner,txpool,db,shh 等，但是因为 RPC 服务是明文传输，所以，如果使用 personal 的时候，要注意安全问题;

For SCS:

```
scsserver --rpc
```

2.4.2 HEX value encoding

At present there are two key datatypes that are passed over JSON: unformatted byte arrays and quantities. Both are passed with a hex encoding, however with different requirements to formatting:

When encoding **QUANTITIES** (integers, numbers): encode as hex, prefix with "0x", the most compact representation (slight exception: zero should be represented as "0x0"). Examples: - 0x41 (65 in decimal) - 0x400 (1024 in decimal) - WRONG: 0x (should always have at least one digit - zero is "0x0") - WRONG: 0x0400 (no leading zeroes allowed) - WRONG: ff (must be prefixed 0x)

When encoding **UNFORMATTED DATA** (byte arrays, account addresses, hashes, bytecode arrays): encode as hex, prefix with "0x", two hex digits per byte. Examples: - 0x41 (size 1, "A") - 0x004200 (size 3, "**:raw-latex:‘\0‘B:raw-latex:‘**0‘") - 0x (size 0, "") - WRONG: 0xf0f0f (must be even number of digits) - WRONG: 004200 (must be prefixed 0x)

Currently *MOAC <https://github.com/MOACChain/moac-core/releases>* VNODE server provides JSON-RPC communication over http and IPC (unix socket Linux and OSX/named pipes on Windows). SCS server provides rovides JSON-RPC communication over http only.

2.4.3 The default block parameter

The following methods have an extra default block parameter:

- *mc_getBalance*
- *mc_getCode*
- *mc_getTransactionCount*
- *mc_getStorageAt*
- *mc_call*

When requests are made that act on the state of moac, the last default block parameter determines the height of the block.

The following options are possible for the defaultBlock parameter:

- `HEX String` - an integer block number
- `String "earliest"` for the earliest/genesis block
- `String "latest"` - for the latest mined block
- `String "pending"` - for the pending state/transactions

2.4.4 Curl 命令示例

The curl options below might return a response where the node complains about the content type, this is because the –data option sets the content type to application/x-www-form-urlencoded . If your node does complain, manually set the header by placing -H "Content-Type: application/json" at the start of the call.

The examples assume a local MOAC node is running and connected to testnet (network id = 101). The URL/IP & port combination is localhost:8545 or '127.0.0.1', which must be the last argument given to curl.

2.4.5 JSON-RPC 命令

- chain3
  - *chain3_clientVersion*
  - *chain3_sha3*
- net
  - *net_version*
  - *net_peerCount*
  - *net_listening*
- mc
  - *mc_protocolVersion*
  - *mc_syncing*
  - *mc_coinbase*
  - *mc_mining*
  - *mc_hashrate*
  - *mc_gasPrice*
  - *mc_accounts*
  - *mc_blockNumber*
  - *mc_getBalance*
  - *mc_getStorageAt*
  - *mc_getTransactionCount*
  - *mc_getBlockTransactionCountByHash*
  - *mc_getBlockTransactionCountByNumber*
  - *mc_getUncleCountByBlockHash*

- *mc_getUncleCountByBlockNumber*
- *mc_getCode*
- *mc_sign*
- *mc_sendTransaction*
- *mc_sendRawTransaction*
- *mc_call*
- *mc_estimateGas*
- *mc_getBlockByHash*
- *mc_getBlockByNumber*
- *mc_getTransactionByHash*
- *mc_getTransactionByBlockHashAndIndex*
- *mc_getTransactionByBlockNumberAndIndex*
- *mc_getTransactionReceipt*
- *mc_getUncleByBlockHashAndIndex*
- *mc_getUncleByBlockNumberAndIndex*
- *mc_newFilter*
- *mc_newBlockFilter*
- *mc_newPendingTransactionFilter*
- *mc_uninstallFilter*
- *mc_getFilterChanges*
- *mc_getFilterLogs*
- *mc_getLogs*
- *mc_getWork*
- *mc_submitWork*

- vnode
  - *vnode_address*
  - *vnode_scsService*
  - *vnode_serviceCfg*
  - *vnode_showToPublic*
  - *vnode_vnodeIP*

- scs
  - *scs_directcall*
  - *scs_getblock*
  - *scs_getBlockNumber*
  - *scs_getDappList*
  - *scs_getDappState*
  - *scs_getMicroChainInfo*
  - *scs_getMicroChainList*
  - *scs_getNonce*
  - *scs_getSCSId*
  - *scs_getTransactionByHash*
  - *scs_getTransactionByNonce*
  - *scs_getReceiptByHash*
  - *scs_getReceiptByNonce*
  - *scs_getExchangeByAddress*
  - *scs_getExchangeInfo*
  - *scs_getTxpool*

CHAIN3

Chain3_clientVersion

Returns the current client version.

Parameters

none

Returns

`String` - The current client version

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"chain3_clientVersion","params":[],"id
↪":101}' 'localhost:8545'
```

(下页继续)

(续上页)

```
// Result
{
  "jsonrpc":"2.0",
  "id":101,
  "result":"Moac/v0.8.3-release-ad500ab5/darwin-amd64/go1.10"
}
```

Chain3_sha3

返回输入数据的 Keccak-256 (*not* the standardized SHA3-256) 哈希值（HASH）.

Parameters

1. `DATA` - the data to convert into a SHA3 hash

```
params: [
  "0x68656c6c6f20776f726c64"
]
```

Returns

`DATA` - The SHA3 result of the given string.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"chain3_sha3","params":[
↪"0x68656c6c6f20776f726c64"],"id":101}' localhost:8545

// Result
{
  "jsonrpc":"2.0",
  "id":101,
  "result":"0x47173285a8d7341e5e972fc677286384f802f8ef42a5ec5f03bbfa254cb01fad"
}
```

NET

net_version

返回当前网络 ID，墨客网络主网为 99，测试网为 101，本地调试私网默认为 100.

Parameters

none

Returns

`String` - The current network id. - `"99"`: MOAC Mainnet - `"101"`: MOAC Testnet - `"100"`: Devnet

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_version","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "101"
}
```

---

net_listening

Returns `true` if client is actively listening for network connections.

Parameters

none

Returns

`Boolean` - `true` when listening, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_listening","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":101,
  "jsonrpc":"2.0",
  "result":true
}
```

---

net_peerCount

Returns number of peers currently connected to the client.

Parameters

none

Returns

`QUANTITY` - integer of the number of connected peers.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_peerCount","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":74,
  "jsonrpc": "2.0",
  "result": "0x4" // 4 net peers are connecting
}
```

MC

mc_protocolVersion

Returns the current MOAC protocol version.

Parameters

none

Returns

`String` - The current MOAC protocol version

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_protocolVersion","params":[],"id":101}
↪' localhost:8545

// Result
{
  "id":101,
```

(下页继续)

(续上页)

```
  "jsonrpc": "2.0",
  "result": "0x3f"
}
```

---

mc_syncing

Returns an object with data about the sync status or `false`.

Parameters

none

Returns

`Object|Boolean`, An object with sync status data or `FALSE`, when not syncing:

- `startingBlock`: `QUANTITY` - The block at which the

import started (will only be reset, after the sync reached his head) - `currentBlock`: `QUANTITY` - The current block, same as mc_blockNumber - `highestBlock`: `QUANTITY` - The estimated highest block

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_syncing","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
    startingBlock: '0x384',
    currentBlock: '0x386',
    highestBlock: '0x454'
  }
}
// Or when not syncing
{
  "id":101,
  "jsonrpc": "2.0",
  "result": false
}
```

---

mc_coinbase

Returns the client coinbase address.

Parameters

none

Returns

`DATA`, 20 bytes - the current coinbase address.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_coinbase","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x407d73d8a49eeb85d32cf465507dd71d507100c1"
}
```

---

mc_mining

Returns `true` if client is actively mining new blocks.

Parameters

none

Returns

`Boolean` - returns `true` of the client is mining, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_mining","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": true
}
```

mc_hashrate

Returns the number of hashes per second that the node is mining with.

Parameters

none

Returns

`QUANTITY` - number of hashes per second.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_hashrate","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x38a"
}
```

mc_gasPrice

Returns the current price per gas in sha.

Parameters

none

Returns

`QUANTITY` - integer of the current gas price in sha.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_gasPrice","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":101,
```

(下页继续)

(续上页)

```
  "jsonrpc": "2.0",
  "result": "0x09184e72a000" // 10000000000000
}
```

---

mc_accounts

Returns a list of addresses owned by client.

Parameters

none

Returns

`Array of DATA`, 20 Bytes - addresses owned by the client.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_accounts","params":[],"id":101}'␣
↪localhost:8545

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": ["0x407d73d8a49eeb85d32cf465507dd71d507100c1",
↪"0x87dc9d8014e189b9d32c622a9ad1d02f72383979"]
}
```

---

mc_blockNumber

Returns the number of most recent block.

Parameters

none

Returns

`QUANTITY` - integer of the current block number the client is on.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_blockNumber","params":[],"id":101}'
↪'localhost:8545'
```

(下页继续)

(续上页)

```
// Result
{
  "id":83,
  "jsonrpc": "2.0",
  "result": "0x4b7" // 1207
}
```

---

mc_getBalance

Returns the balance of the account of given address.

Parameters

1. `DATA`, 20 Bytes - address to check for balance.
2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the *default block parameter <#the-default-block-parameter>*

```
params: [
   '0x407d73d8a49eeb85d32cf465507dd71d507100c1',
   'latest'
]
```

Returns

`QUANTITY` - integer of the current balance in sha.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getBalance","params":[
↪"0x87dc9d8014e189b9d32c622a9ad1d02f72383979", "latest"],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x12f8b3a319c000" // 5340000000000000
}
```

---

mc_getStorageAt

Returns the value from a storage position at a given address.

Parameters

1. `DATA`, 20 Bytes - address of the storage.
2. `QUANTITY` - integer of the position in the storage.
3. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the *default block parameter <#the-default-block-parameter>*

Returns

`DATA` - the value at this storage position.

Example

Calculating the correct position depends on the storage to retrieve. Consider the following contract deployed at `0x02701dc451e126316ece6870fd70ea140efa7b15` by address `0xa8863fc8ce3816411378685223c03daae9770ebb`.

```
contract Storage {
    uint pos0;
    mapping(address => uint) pos1;

    function Storage() {
        pos0 = 1234;
        pos1[msg.sender] = 5678;
    }
}
```

Retrieving the value of pos0 is straight forward:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "mc_getStorageAt", "params": [
↪"0x02701dc451e126316ece6870fd70ea140efa7b15", "0x0", "latest"], "id": 101}'␣
↪localhost:8545

{"jsonrpc":"2.0","id":1,"result":
↪"0x00000000000000000000000000000000000000000000000000000000000004d2"}
```

Retrieving an element of the map is harder. The position of an element in the map is calculated with:

```
keccack(LeftPad32(key, 0), LeftPad32(map position, 0))
```

This means to retrieve the storage on pos1[“0x391694e7e0b0cce554cb130d723a9d27458f9298”] we need to calculate the position with:

```
keccak(decodeHex("000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" +
↪"0000000000000000000000000000000000000000000000000000000000000001"))
```

The moac console which comes with the chain3 library can be used to make the calculation:

```
> var key = "000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" +
↪"0000000000000000000000000000000000000000000000000000000000000001"
undefined
> chain3.sha3(key, {"encoding": "hex"})
"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9"
```

Now to fetch the storage:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "mc_getStorageAt", "params": [
↪"0x295a70b2de5e3953354a6a8344e616ed314d7251",
↪"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9", "latest"], "id":␣
↪101}' localhost:8545

{"jsonrpc":"2.0","id":101,"result":
↪"0x000000000000000000000000000000000000000000000000000000000000162e"}
```

---

mc_getTransactionCount

Returns the number of transactions *sent* from an address.

Parameters

1. `DATA`, 20 Bytes - address.
2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the *default block parameter <#the-default-block-parameter>*

```
params: [
   '0x407d73d8a49eeb85d32cf465507dd71d507100c1',
   'latest' // state at the latest block
]
```

Returns

`QUANTITY` - integer of the number of transactions send from this address.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getTransactionCount","params":[
↪"0x57d83802a772adf506a89f5021c93a05749e3c6e","latest"],"id":101}' localhost:8545

// Result
```

(下页继续)

(续上页)

```
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x29" // 41
}
```

---

mc_getBlockTransactionCountByHash

Returns the number of transactions in a block from a block matching the given block hash.

Parameters

1. `DATA`, 32 Bytes - hash of a block

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

`QUANTITY` - integer of the number of transactions in this block.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getBlockTransactionCountByHash",
↪"params":["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id
↪":101}' localhost:8545

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xb" // 11
}
```

---

mc_getBlockTransactionCountByNumber

Returns the number of transactions in a block matching the given block number.

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the *default block parameter <#the-default-block-parameter>*.

```
params: [
   '0xe8', // 232
]
```

Returns

QUANTITY - integer of the number of transactions in this block.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getBlockTransactionCountByNumber",
↪"params":["0xe8"],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0xa" // 10
}
```

---

mc_getUncleCountByBlockHash

Returns the number of uncles in a block from a block matching the given block hash.

Parameters

1. DATA, 32 Bytes - hash of a block

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

QUANTITY - integer of the number of uncles in this block.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getUncleCountByBlockHash","params":[
↪"0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":101}'␣
↪localhost:8545

// Result
```

(下页继续)

(续上页)

```
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

---

mc_getUncleCountByBlockNumber

Returns the number of uncles in a block from a block matching the given block number.

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string "latest", "earliest" or "pending", see the *default block parameter <#the-default-block-parameter>*

```
params: [
   '0xe8', // 232
]
```

Returns

`QUANTITY` - integer of the number of uncles in this block.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getUncleCountByBlockNumber","params":[
↪"0xe8"],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

---

mc_getCode

Returns code at a given address.

Parameters

1. `DATA`, 20 Bytes - address
2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the *default block parameter <#the-default-block-parameter>*

```
params: [
   '0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',
   '0x2'  // 2
]
```

Returns

`DATA` - the code from the given address.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getCode","params":[
↪"0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b", "0x2"],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result":
↪"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078202905091905
↪"
}
```

---

mc_sign

The sign method calculates an MOAC specific signature with: `sign(keccak256("\x19MOAC Signed Message:\n" + len(message) + message))`.

By adding a prefix to the message makes the calculated signature recognisable as an MOAC specific signature. This prevents misuse where a malicious DApp can sign arbitrary data (e.g. transaction) and use the signature to impersonate the victim.

Note the address to sign with must be unlocked.

Parameters

account, message

1. `DATA`, 20 Bytes - address
2. `DATA`, N Bytes - message to sign

Returns

`DATA`: Signature

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_sign","params":[
↪"0x57d83802a772adf506a89f5021c93a05749e3c6e", "0xdeadbeaf"],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result":
↪"0xefa354f816ab378a7da6ec25afe9e6393b72a3c06b2d08550815cc43d1993f6c58afbf228559df6a39ac419bef847ca2dc3
↪"
}
```

An example how to use solidity ecrecover to verify the signature calculated with *:ref:'mc_sign'* can be found *here <https://gist.github.com/bas-vk/d46d83da2b2b4721efb0907aecdb7ebd>*. The contract is deployed on the testnet Ropsten and Rinkeby.

---

mc_sendTransaction

Creates new message call transaction or a contract creation, if the data field contains code.

Note the address to sign with must be unlocked.

Parameters

1. `Object` - The transaction object

- `from`: `DATA`, 20 Bytes - The address the transaction is send from.
- `to`: `DATA`, 20 Bytes - (optional when creating new contract) The address the transaction is directed to.
- `gas`: `QUANTITY` - (optional, default: 90000) Integer of the gas provided for the transaction execution. It will return unused gas.
- `gasPrice`: `QUANTITY` - (optional, default: To-Be-Determined) Integer of the gasPrice used for each paid gas
- `value`: `QUANTITY` - (optional) Integer of the value sent with this transaction
- `data`: `DATA` - The compiled code of a contract OR the hash of the invoked method signature and encoded parameters. MOAC used the solidity ABI. For details see *Ethereum Contract ABI <https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI>*
- `nonce`: `QUANTITY` - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

```
params: [{
  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
  "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
  "gas": "0x76c0", // 30400
  "gasPrice": "0x9184e72a000", // 10000000000000
  "value": "0x9184e72a", // 2441406250
  "data":
↪"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"
}]
```

Returns

`DATA`, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available.

Use *mc_getTransactionReceipt* to get the contract address, after the transaction was mined, when you created a contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_sendTransaction","params":[{see above}
↪],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

---

mc_sendRawTransaction

Creates new message call transaction or a contract creation for signed transactions.

Parameters

1. `DATA`, The signed transaction data.

```
params: [
↪"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"]
```

Returns

`DATA`, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available.

Use *mc_getTransactionReceipt* to get the contract address, after the transaction was mined, when you created a contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_sendRawTransaction","params":[{see␣
↪above}],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

---

mc_call

Executes a new message call immediately without creating a transaction on the block chain.

Parameters

1. `Object` - The transaction call object

- `from`: `DATA`, 20 Bytes - (optional) The address the transaction is sent from.
- `to`: `DATA`, 20 Bytes - The address the transaction is directed to.
- `gas`: `QUANTITY` - (optional) Integer of the gas provided for the transaction execution. mc_call consumes zero gas, but this parameter may be needed by some executions.
- `gasPrice`: `QUANTITY` - (optional) Integer of the gasPrice used for each paid gas
- `value`: `QUANTITY` - (optional) Integer of the value sent with this transaction
- `data`: `DATA` - (optional) Hash of the method signature and encoded parameters. For details see *Ethereum Contract ABI <https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI>*

2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the *default block parameter <#the-default-block-parameter>*

Returns

`DATA` - the return value of executed contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_call","params":[{see above}],"id":101}
↪' localhost:8545
```

(下页继续)

(续上页)

```
// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x"
}
```

---

mc_estimateGas

Generates and returns an estimate of how much gas is necessary to allow the transaction to complete. The transaction will not be added to the blockchain. Note that the estimate may be significantly more than the amount of gas actually used by the transaction, for a variety of reasons including EVM mechanics and node performance.

Parameters

See *mc_call* parameters, expect that all properties are optional. If no gas limit is specified moac uses the block gas limit from the pending block as an upper bound. As a result the returned estimate might not be enough to executed the call/transaction when the amount of gas is higher than the pending block gas limit.

Returns

`QUANTITY` - the amount of gas used.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_estimateGas","params":[{see above}],
↪"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x5208" // 21000
}
```

---

mc_getBlockByHash

Returns information about a block by hash.

Parameters

1. `DATA`, 32 Bytes - Hash of a block.

2. `Boolean` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

```
params: [
   '0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',
   true
]
```

Returns

`Object` - A block object, or `null` when no block was found:

- `number`: `QUANTITY` - the block number. `null` when its pending block.
- `hash`: `DATA`, 32 Bytes - hash of the block. `null` when its pending block.
- `parentHash`: `DATA`, 32 Bytes - hash of the parent block.
- `nonce`: `DATA`, 8 Bytes - hash of the generated proof-of-work. `null` when its pending block.
- `sha3Uncles`: `DATA`, 32 Bytes - SHA3 of the uncles data in the block.
- `logsBloom`: `DATA`, 256 Bytes - the bloom filter for the logs of the block. `null` when its pending block.
- `transactionsRoot`: `DATA`, 32 Bytes - the root of the transaction trie of the block.
- `stateRoot`: `DATA`, 32 Bytes - the root of the final state trie of the block.
- `receiptsRoot`: `DATA`, 32 Bytes - the root of the receipts trie of the block.
- `miner`: `DATA`, 20 Bytes - the address of the beneficiary to whom the mining rewards were given.
- `difficulty`: `QUANTITY` - integer of the difficulty for this block.
- `totalDifficulty`: `QUANTITY` - integer of the total difficulty of the chain until this block.
- `extraData`: `DATA` - the "extra data" field of this block.
- `size`: `QUANTITY` - integer the size of this block in bytes.
- `gasLimit`: `QUANTITY` - the maximum gas allowed in this block.
- `gasUsed`: `QUANTITY` - the total used gas by all transactions in this block.
- `timestamp`: `QUANTITY` - the unix timestamp for when the block was collated.
- `transactions`: `Array` - Array of transaction objects, or 32 Bytes transaction hashes depending on the last given parameter.
- `uncles`: `Array` - Array of uncle hashes.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getBlockByHash","params":[
↪"0x9d88f2a2a348eec743b149f461fddcf0843d3920ddf998896672cee476b99127", true],"id":101}'␣
↪localhost:8545
```

(下页继续)

(续上页)

```
// Result
{
"jsonrpc":"2.0",
"id":101,
"result":{
  "difficulty":"0x9ad5136",
  "extraData":"0xdd854d4f41432d85302e382e332d87676f312e382e338777696e646f7773",
  "gasLimit":"0x895440",
  "gasUsed":"0x0",
  "hash":"0x9d88f2a2a348eec743b149f461fddcf0843d3920ddf998896672cee476b99127",
  "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  "miner":"0x0a2168d2f08161c01745fec4e6e8fe06f314ab41",
  "mixHash":"0xd6e9f8d36f8f0f57abff5ccf6390ece4d5a14eb624779628fd6e6119451589e2",
  "nonce":"0x423e7acb4d05a23c",
  "number":"0x4803d",
  "parentHash":"0xc494dbda1ef4f673fd14fc487a1346a7747a41910d1a1f0e72d6e0e660c0e792",
  "receiptsRoot":"0x9287370eb27f11b0c2188431cbc58a23b685f02dbd851ed4d974f932bd780839",
  "sha3Uncles":"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  "size":"0x24e",
  "stateRoot":"0xff791a26fd0cb6addfe4bec0287b3905bf44898f0fd71e446cb1385f94333e18",
  "timestamp":"0x5b1f3a14",
  "totalDifficulty":"0x325dc638b38e",
  "transactions":[{"blockHash":
↪"0x9d88f2a2a348eec743b149f461fddcf0843d3920ddf998896672cee476b99127","blockNumber":
↪"0x4803d","from":"0x0000000000000000000000000000000000000064","gas":"0x0","gasPrice":
↪"0x4a817c800","hash":
↪"0x52ccd05bdfb137fcbb6154f1cb7c006ba53cd8ce80511841d2e7b3c6324d49df","input":
↪"0xc1c0e9c4","nonce":"0x4803c","syscnt":"0x65","to":
↪"0x0000000000000000000000000000000000000065","transactionIndex":"0x0","value":"0x0","v
↪":"0x0","r":"0x0","s":"0x0","shardingFlag":"0x0"}],
  "transactionsRoot":"0xa29e25d8e3db000e44122a594cc52a7bf050850b3a7e028fd54cb83784fc1bc3
↪",
  "uncles":[]
  }
}
```

mc_getBlockByNumber

Returns information about a block by block number.

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the *default block parameter <#the-default-block-parameter>*.
2. `Boolean` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

```
params: [
   '0x1b4', // 436
   true
]
```

Returns

See *mc_getBlockByHash*

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getBlockByNumber","params":["0x1b4",␣
↪true],"id":101}' localhost:8545
```

Result see *mc_getBlockByHash*

---

mc_getTransactionByHash

Returns the information about a transaction requested by transaction hash.

Parameters

1. `DATA`, 32 Bytes - hash of a transaction

```
params: [
   "0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"
]
```

Returns

`Object` - A transaction object, or `null` when no transaction was found:

- `hash`: `DATA`, 32 Bytes - hash of the transaction.
- `nonce`: `QUANTITY` - the number of transactions made by the sender prior to this one.
- `blockHash`: `DATA`, 32 Bytes - hash of the block where this transaction was in. `null` when its pending.
- `blockNumber`: `QUANTITY` - block number where this transaction was in. `null` when its pending.

- `transactionIndex`: `QUANTITY` - integer of the transactions index position in the block. `null` when its pending.
- `from`: `DATA`, 20 Bytes - address of the sender.
- `to`: `DATA`, 20 Bytes - address of the receiver. `null` when its a contract creation transaction.
- `value`: `QUANTITY` - value transferred in Wei.
- `gasPrice`: `QUANTITY` - gas price provided by the sender in Wei.
- `gas`: `QUANTITY` - gas provided by the sender.
- `input`: `DATA` - the data send along with the transaction.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getTransactionByHash","params":[
↪"0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":101}'␣
↪localhost:8545

// Result
{
"id":101,
"jsonrpc":"2.0",
"result": {
    "hash":"0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b",
    "nonce":"0x",
    "blockHash": "0xbeab0aa2411b7ab17f30a99d3cb9c6ef2fc5426d6ad6fd9e2a26a6aed1d1055b",
    "blockNumber": "0x15df", // 5599
    "transactionIndex":  "0x1", // 1
    "from":"0x407d73d8a49eeb85d32cf465507dd71d507100c1",
    "to":"0x85h43d8a49eeb85d32cf465507dd71d507100c1",
    "value":"0x7f110", // 520464
    "gas": "0x7f110", // 520464
    "gasPrice":"0x09184e72a000",
    "input":"0x603880600c6000396000f300603880600c6000396000f3603880600c6000396000f360",
  }
}
```

mc_getTransactionByBlockHashAndIndex

Returns information about a transaction by block hash and transaction index position.

Parameters

1. `DATA`, 32 Bytes - hash of a block.
2. `QUANTITY` - integer of the transaction index position.

```
params: [
   '0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',
   '0x0' // 0
]
```

Returns

See *mc_getTransactionByHash*

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getTransactionByBlockHashAndIndex",
↪"params":["0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b", "0x0"],
↪"id":101}' localhost:8545
```

Result see *mc_getTransactionByHash*

---

mc_getTransactionByBlockNumberAndIndex

Returns information about a transaction by block number and transaction index position.

Parameters

1. `QUANTITY|TAG` - a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the *default block parameter <#the-default-block-parameter>*.
2. `QUANTITY` - the transaction index position.

```
params: [
   '0x29c', // 668
   '0x0' // 0
]
```

Returns

See *mc_getTransactionByHash*

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getTransactionByBlockNumberAndIndex",
↪"params":["0x29c", "0x0"],"id":101}' localhost:8545
```

Result see *mc_getTransactionByHash*

---

mc_getTransactionReceipt

Returns the receipt of a transaction by transaction hash.

Note That the receipt is not available for pending transactions.

Parameters

1. `DATA`, 32 Bytes - hash of a transaction

```
params: [
   '0x7bb694c3462764cb113e9b742faaf06adc728e70b607f8b7aa95207ee32b1c5e'
]
```

Returns

`Object` - A transaction receipt object, or `null` when no receipt was found:

- `transactionHash`: `DATA`, 32 Bytes - hash of the transaction.
- `transactionIndex`: `QUANTITY` - integer of the transactions index position in the block.
- `blockHash`: `DATA`, 32 Bytes - hash of the block where this transaction was in.
- `blockNumber`: `QUANTITY` - block number where this transaction was in.
- `cumulativeGasUsed`: `QUANTITY` - The total amount of gas used when this transaction was executed in the block.
- `gasUsed`: `QUANTITY` - The amount of gas used by this specific transaction alone.
- `contractAddress`: `DATA`, 20 Bytes - The contract address created, if the transaction was a contract creation, otherwise `null`.
- `logs`: `Array` - Array of log objects, which this transaction generated.
- `logsBloom`: `DATA`, 256 Bytes - Bloom filter for light clients to quickly retrieve related logs.

It also returns *either* :

- `root` : `DATA` 32 bytes of post-transaction stateroot (pre Byzantium)
- `status`: `QUANTITY` either `1` (success) or `0` (failure)

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getTransactionReceipt","params":[
↪"0x7bb694c3462764cb113e9b742faaf06adc728e70b607f8b7aa95207ee32b1c5e"],"id":101}'␣
↪localhost:8545
```

(下页继续)

(续上页)

```
// Result
{
{"jsonrpc":"2.0",
"id":101,
"result":{
"blockHash":"0xad7d2eec610b3e04ea4676c11a3184cb07df43ed7ab717f331d3c952fc1b55cf",
"blockNumber":"0x1b943",
"contractAddress":"0xf2f4eec6c2adfcf780aae828de0b25f86506ffae",
"cumulativeGasUsed":"0x1b337c",
"from":"0x7312f4b8a4457a36827f185325fd6b66a3f8bb8b",
"gasUsed":"0x1b337c",
"logs":[],
"logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
"status":"0x1",
"to":null,
"transactionHash":"0x7bb694c3462764cb113e9b742faaf06adc728e70b607f8b7aa95207ee32b1c5e",
"transactionIndex":"0x1"}
}
```

---

mc_getUncleByBlockHashAndIndex

Returns information about a uncle of a block by hash and uncle index position.

Parameters

1. `DATA`, 32 Bytes - hash a block.
2. `QUANTITY` - the uncle’ s index position.

```
params: [
   '0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b',
   '0x0' // 0
]
```

Returns

See *mc_getBlockByHash*

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getUncleByBlockHashAndIndex","params
↪":["0x696d389aa9b6b44e08af9f3528c51587aac435b75a54ece42f4b2d1289043497", "0x0"],"id
↪":101}' localhost:8545
```

Result see *mc_getBlockByHash*

Note: An uncle doesn't contain individual transactions.

---

mc_getUncleByBlockNumberAndIndex

Returns information about a uncle of a block by number and uncle index position.

Parameters

1. `QUANTITY|TAG` - a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the *default block parameter <#the-default-block-parameter>*.
2. `QUANTITY` - the uncle's index position.

```
params: [
   '0x29c', // 668
   '0x0' // 0
]
```

Returns

See *mc_getBlockByHash*

Note: An uncle doesn't contain individual transactions.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getUncleByBlockNumberAndIndex","params
↪":["0x29c", "0x0"],"id":101}' localhost:8545
```

Result see *mc_getBlockByHash*

---

mc_newFilter

Creates a filter object, based on filter options, to notify when the state changes (logs). To check if the state has changed, call *mc_getFilterChanges*.

A note on specifying topic filters:

Topics are order-dependent. A transaction with a log with topics [A, B] will be matched by the following topic filters: * `[]` "anything" * `[A]` "A in first position (and anything after)" * `[null, B]` "anything

in first position AND B in second position (and anything after)" * `[A, B]` "A in first position AND B in second position (and anything after)" * `[[A, B], [A, B]]` "(A OR B) in first position AND (A OR B) in second position (and anything after)"

Parameters

1. `Object` - The filter options:

- `fromBlock`: `QUANTITY|TAG` - (optional, default: `"latest"`) Integer block number, or `"latest"` for the last mined block or `"pending"`, `"earliest"` for not yet mined transactions.
- `toBlock`: `QUANTITY|TAG` - (optional, default: `"latest"`) Integer block number, or `"latest"` for the last mined block or `"pending"`, `"earliest"` for not yet mined transactions.
- `address`: `DATA|Array`, 20 Bytes - (optional) Contract address or a list of addresses from which logs should originate.
- `topics`: `Array of DATA`, - (optional) Array of 32 Bytes `DATA` topics. Topics are order-dependent. Each topic can also be an array of DATA with "or" options.

```
params: [{
  "fromBlock": "0x1",
  "toBlock": "0x2",
  "address": "0x8888f1f195afa192cfee860698584c030f4c9db1",
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b", null,␣
↪["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b",
↪"0x000000000000000000000000aff3454fce5edbc8cca8697c15331677e6ebccc"]]
}]
```

Returns

`QUANTITY` - A filter id.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_newFilter","params":[{"topics":[
↪"0x12341234"]}],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

---

mc_newBlockFilter

Creates a filter in the node, to notify when a new block arrives. To check if the state has changed, call *mc_getFilterChanges*.

Parameters

None

Returns

`QUANTITY` - A filter id.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_newBlockFilter","params":[],"id":101}
↪' localhost:8545

// Result
{
  "id":101,
  "jsonrpc":  "2.0",
  "result": "0x244afc5c2cb35b24c5b91bba7f7ab19d" // 1
}
```

---

mc_newPendingTransactionFilter

Creates a filter in the node, to notify when new pending transactions arrive. To check if the state has changed, call *mc_getFilterChanges*.

Parameters

None

Returns

`QUANTITY` - A filter id.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_newPendingTransactionFilter","params
↪":[],"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc":  "2.0",
```

(下页继续)

(续上页)

```
  "result": "0x1" // 1
}
```

---

mc_uninstallFilter

Uninstalls a filter with given id. Should always be called when watch is no longer needed. Additonally Filters timeout when they aren't requested with *mc_getFilterChanges* for a period of time.

Parameters

1. `QUANTITY` - The filter id.

```
params: [
  "0xb" // 11
]
```

Returns

`Boolean` - `true` if the filter was successfully uninstalled, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_uninstallFilter","params":["0xb"],"id
↪":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": true
}
```

---

mc_getFilterChanges

Polling method for a filter, which returns an array of logs which occurred since last poll.

Parameters

1. `QUANTITY` - the filter id.

```
params: [
  "0x16" // 22
]
```

Returns

`Array` - Array of log objects, or an empty array if nothing has changed since last poll.

- For filters created with *:ref:'mc_newBlockFilter'* the return are block hashes (`DATA`, 32 Bytes), e.g. `["0x3454645634534..."]`.
- For filters created with *:ref:'mc_newPendingTransactionFilter'* the return are transaction hashes (`DATA`, 32 Bytes), e.g. `["0x6345343454645..."]`.
- For filters created with *:ref:'mc_newFilter'* logs are objects with following params:
- `removed`: `TAG` - `true` when the log was removed, due to a chain reorganization. `false` if its a valid log.
- `logIndex`: `QUANTITY` - integer of the log index position in the block. `null` when its pending log.
- `transactionIndex`: `QUANTITY` - integer of the transactions index position log was created from. `null` when its pending log.
- `transactionHash`: `DATA`, 32 Bytes - hash of the transactions this log was created from. `null` when its pending log.
- `blockHash`: `DATA`, 32 Bytes - hash of the block where this log was in. `null` when its pending. `null` when its pending log.
- `blockNumber`: `QUANTITY` - the block number where this log was in. `null` when its pending. `null` when its pending log.
- `address`: `DATA`, 20 Bytes - address from which this log originated.
- `data`: `DATA` - contains one or more 32 Bytes non-indexed arguments of the log.
- `topics`: `Array of DATA` - Array of 0 to 4 32 Bytes `DATA` of indexed log arguments. (In *solidity*: The first topic is the *hash* of the signature of the event (e.g. `Deposit(address,bytes32,uint256)`), except you declared the event with the `anonymous` specifier.)

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getFilterChanges","params":["0x16"],
↪"id":101}' localhost:8545

// Result
{
  "id":101,
  "jsonrpc":"2.0",
  "result": [{
    "logIndex": "0x1", // 1
    "blockNumber":"0x1b4", // 436
    "blockHash": "0x8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcfdf829c5a142f1fccd7d",
```

(下页继续)

(续上页)

```
    "transactionHash":  "0xdf829c5a142f1fccd7d8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcf
↪",
    "transactionIndex": "0x0", // 0
    "address": "0x16c5785ac562ff41e2dcfdf829c5a142f1fccd7d",
    "data":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "topics": ["0x59ebeb90bc63057b6515673c3ecf9438e5058bca0f92585014eced636878c9a5"]
    },{
      ...
    }]
}
```

---

mc_getFilterLogs

Returns an array of all logs matching filter with given id.

Parameters

1. `QUANTITY` - The filter id.

```
params: [
  "0x16" // 22
]
```

Returns

See *mc_getFilterChanges*

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getFilterLogs","params":["0x16"],"id
↪":101}' localhost:8545
```

Result see *mc_getFilterChanges*

---

mc_getLogs

Returns an array of all logs matching a given filter object.

Parameters

1. `Object` - the filter object, see *mc_newFilter parameters*.

```
params: [{
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]
}]
```

Returns

See *mc_getFilterChanges*

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getLogs","params":[{"topics":[
↪"0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]}],"id":101}'
```

Result see *mc_getFilterChanges*

---

mc_getWork

Returns the hash of the current block, the seedHash, and the boundary condition to be met ("target").

Parameters

none

Returns

`Array` - Array with the following properties: 1. `DATA`, 32 Bytes - current block header pow-hash 2. `DATA`, 32 Bytes - the seed hash used for the DAG. 3. `DATA`, 32 Bytes - the boundary condition ("target"), 2^256 / difficulty.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"mc_getWork","params":[],"id":101}'␣
↪localhost:8545

// Result
{
"jsonrpc":"2.0",
"id":101,
"result":["0x367ebe7ed77dde0a4cc134adaa769f93a833e08a86a0d38ab18e32cc3740b5f8",
↪"0x9b2baad7528ecec612c5751a6bd525905892d7892e155c3b05e61363154a940b",
↪"0x0000001dd67ebb66a2c0fc63b9907077f70b5315e1780a8b74f33d40e2a9a7e4"]
}
```

---

mc_submitWork

Used for submitting a proof-of-work solution.

Parameters

1. `DATA`, 8 Bytes - The nonce found (64 bits)
2. `DATA`, 32 Bytes - The header' s pow-hash (256 bits)
3. `DATA`, 32 Bytes - The mix digest (256 bits)

```
params: [
  "0x0000000000000001",
  "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
  "0xD1FE5700000000000000000000000000D1FE5700000000000000000000000000"
]
```

Returns

`Boolean` - returns `true` if the provided solution is valid, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"mc_submitWork", "params":[
↪"0x0000000000000001",
↪"0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
↪"0xD1GE5700000000000000000000000000D1GE5700000000000000000000000000"],"id":101}'

// Result
{
  "id":101,
  "jsonrpc":"2.0",
  "result": true
}
```

VNODE

vnode_address

Returns the VNODE benificial address. This is needed for SCS to use in the communication.

Parameters

none

Returns

address: DATA, 20 Bytes - address from which the VNODE settings in the vnodeconfig.json file.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"vnode_address","params":[],"id":101}'␣
↪localhost:8545

// Result
{
"jsonrpc":"2.0",
"id":101,
"result":"0xa8863fc8ce3816411378685223c03daae9770ebb"
}
```

---

vnode_scsService

Returns if the VNODE enable the service for SCS servers.

Parameters

none

Returns

Bool - true, enable the SCS service; false, not open.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"vnode_scsService","params":[],"id":101}'
↪'localhost:8545'

// Result
{
  "jsonrpc":"2.0",
  "id":101,
  "result":true
}
```

---

vnode_serviceCfg

Returns the VNODE SCS service ports for connecting with.

Parameters

none

Returns

`String` - The current service port set in the vnodeconfig.json.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"vnode_serviceCfg","params":[],"id":101}'
↪'localhost:8545'

// Result
{
  "jsonrpc":"2.0",
  "id":101,
  "result":":50062"
}
```

---

vnode_showToPublic

Returns if the VNODE enable the public view.

Parameters

none

Returns

`Bool` - true, open to the public; false, not open.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"vnode_showToPublic","params":[],"id":101}
↪' 'localhost:8545'

// Result
{
  "jsonrpc":"2.0",
  "id":101,
  "result":true
}
```

---

vnode_vnodeIP

Returns VNODE IP for users to access. Note for cloud servers, this could be different from the cloud server IP.

Parameters

none

Returns

`String` - The current IP that can be used to access. This is set in the vnodeconfig.json.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"vnode_vnodeIP","params":[],"id":101}'
↪'localhost:8545'

// Result
{
  "jsonrpc":"2.0",
  "id":101,
  "result":"127.0.0.1"
}
```

SCS

scs_directCall

Executes a new constant call of the MicroChain Dapp function without creating a transaction on the MicroChain. This RPC call is used by API/lib to call MicroChain Dapp functions.

Parameters

`Object` - The transaction call object - `from`: `DATA`, 20 Bytes - (optional) The address the transaction is sent from. - `to`: `DATA`, 20 Bytes - The address the transaction is directed to. This parameter is the MicroChain address. - `data`: `DATA` - (optional) Hash of the method signature and encoded parameters. For details see *Ethereum Contract ABI <https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI>*

Returns

`DATA` - the return value of executed Dapp constant function call.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_directCall","params":[{see above}],
↪"id":101}' localhost:8545

// Result
```

(下页继续)

(续上页)

```
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x"
}
```

---

scs_getBlock

Returns information about a block on the MicroChain by block number.

Parameters

1. `String` - the address of the MicroChain that Dapp is on.
2. `QUANTITY|TAG` - integer of a block number, or the string `"earliest"` or `"latest"`, as in the *default block parameter <#the-default-block-parameter>*. Note, scs_getBlock does not support `"pending"`.

Returns

`DATA` - Data in the block on the MicroChain.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getBlock","params":[
↪"0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8","0x1"],"id":101}' localhost:8548

// Result
{"jsonrpc":"2.0","id":101,"result":{"extraData":"0x","hash":
↪"0xc80cbe08bc266b1236f22a8d0b310faae3135961dbef6ad8b6ad4e8cd9537309","number":"0x1",
↪"parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
↪"receiptsRoot":"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
↪"stateRoot":"0x1a065207da60d8e7a44db2f3b5ed9d3e81052a3059e4108c84701d0bf6a62292",
↪"timestamp":"0x0","transactions":[],"transactionsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"}}
```

---

scs_getBlockList

Returns information about multiple MicroChain blocks by block number.

Parameters

1. `String` - the address of the MicroChain that Dapp is on.
2. `QUANTITY` - integer of the start block number.
3. `QUANTITY` - integer of the end block number, need to be larger or equal the start block number.

Returns

`ARRAY` - Array of the block infromation on the MicroChain.

Example

```
// Request
curl -X POST --data curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getBlock","params
↪":["0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8","0x370", "0x373"],"id":101}'␣
↪localhost:8548

// Result
{"jsonrpc":"2.0","id":101,"result":{"blockList":[{"extraData":"0x","hash":
↪"0x56075838e0fffe6576add14783b957239d4f3c57989bc3a7b7728a3b57eb305a","miner":
↪"0xecd1e094ee13d0b47b72f5c940c17bd0c7630326","number":"0x370","parentHash":
↪"0x56352a3a8bd0901608041115817204cbce943606e406d233d7d0359f449bd4c2","receiptsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421","stateRoot":
↪"0xde741a2f6b4a3c865e8f6fc9ba11eadaa1fa04c61d660bcdf0fa1195029699f6","timestamp":
↪"0x5bfb7c1c","transactions":[],"transactionsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"},{"extraData":"0x
↪","hash":"0xbc3f5791ec039cba99c37310a4f30a68030dd2ab79efb47d23fd9ac5343f54e5","miner":
↪"0xecd1e094ee13d0b47b72f5c940c17bd0c7630326","number":"0x371","parentHash":
↪"0x56075838e0fffe6576add14783b957239d4f3c57989bc3a7b7728a3b57eb305a","receiptsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421","stateRoot":
↪"0xde741a2f6b4a3c865e8f6fc9ba11eadaa1fa04c61d660bcdf0fa1195029699f6","timestamp":
↪"0x5bfb7c3a","transactions":[],"transactionsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"},{"extraData":"0x
↪","hash":"0x601be17c47cb4684053457d1d5f70a6dbeb853b27cda08d160555f857f2da33b","miner":
↪"0xecd1e094ee13d0b47b72f5c940c17bd0c7630326","number":"0x372","parentHash":
↪"0xbc3f5791ec039cba99c37310a4f30a68030dd2ab79efb47d23fd9ac5343f54e5","receiptsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421","stateRoot":
↪"0xde741a2f6b4a3c865e8f6fc9ba11eadaa1fa04c61d660bcdf0fa1195029699f6","timestamp":
↪"0x5bfb7c58","transactions":[],"transactionsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"},{"extraData":"0x
↪","hash":"0x8a0bea649bcdbd2b525690ff485e56d5a83443e9013fcdccd1a0adee56ba4092","miner":
↪"0xecd1e094ee13d0b47b72f5c940c17bd0c7630326","number":"0x373","parentHash":
↪"0x601be17c47cb4684053457d1d5f70a6dbeb853b27cda08d160555f857f2da33b","receiptsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421","stateRoot":
↪"0xde741a2f6b4a3c865e8f6fc9ba11eadaa1fa04c61d660bcdf0fa1195029699f6","timestamp":
↪"0x5bfb7c76","transactions":[],"transactionsRoot":
↪"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"}],"endBlk":"0x373
↪","microchainAddress":"0x7D0CbA876cB9Da5fa310A54d29F4687f5dd93fD7","startBlk":"0x370"}}
```

scs_getBlockNumber

Returns the number of most recent block .

Parameters

1. `String` - the address of the MicroChain that Dapp is on.

Returns

`QUANTITY` - integer of the current block number the client is on.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getBlockNumber","params":[
↪"0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8"],"id":101}' 'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x4b7" // 1207
}
```

---

scs_getDappList

Returns the Dapp addresses on the MicroChain. For nuwa 1.0.8 and later version only,

Parameters

1. `String` - the address of the MicroChain that has Dapps.

Returns

`ARRAY` - Array of the DAPP addresses on the MicroChain.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getDappList","params":[],"id":101}'
↪'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
```

(下页继续)

(续上页)

```
  "result": ["0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8",
↪"0x7cfd775c7a97aa632846eff35dcf9dbcf502d0f3"]
}
```

---

scs_getDappState

Returns the Dapp state on the MicroChain.

Parameters

1. `String` - the address of the MicroChain that Dapp is on.

Returns

`QUANTITY` - 0, no DAPP is deployed on the MicroChain; 1, DAPP is deployed on the MicroChain.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getDappState","params":[
↪"0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8"],"id":101}' 'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": 1
}
```

Returns the requested MicroChain information on the connecting SCS. This information is the same as the information defined in the MicroChain contract.

Parameters

1. *String* - the address of the MicroChain on the SCS.

Returns

`Object` A Micro Chain information object as defined in the MicroChain contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getMicroChainInfo","params":[],"id
↪":101}' 'localhost:8545'
```

(下页继续)

(续上页)

```
// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": {"balance":"0x0","blockReward":"0x1c6bf52634000","bondLimit":
↪"0xde0b6b3a7640000","owner":"0xa8863fc8Ce3816411378685223C03DAae9770ebB","scsList":[
↪"0xECd1e094Ee13d0B47b72F5c940C17bD0c7630326",
↪"0x50C15fafb95968132d1a6ee3617E99cCa1FCF059",
↪"0x1b65cE1A393FFd5960D2ce11E7fd6fDB9e991945"],"txReward":"0x174876e800","viaReward":
↪"0x9184e72a000"}
}
```

Returns the list of MicroChains on the SCS that is connecting with.

Parameters

None

Returns

`Array` - A list of Micro Chain addresses on the SCS.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getMicroChainList","params":[],"id
↪":101}' 'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": ["0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8",
↪"0x7cfd775c7a97aa632846eff35dcf9dbcf502d0f3"]
}
```

---

scs_getNonce

Returns the account nonce on the MicroChain.

Parameters

1. `String` - the address of the MicroChain that Dapp is on.
2. `String` - the address of the accountn.

Returns

`QUANTITY` integer of the number of transactions send from this address on the MicroChain;

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getNonce","params":[
↪"0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8",
↪"0x7312F4B8A4457a36827f185325Fd6B66a3f8BB8B"],"id":101}' 'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": 1
}
```

---

scs_getSCSId

Returns the SCS id.

Parameters

None

Returns

1. `String` - SCS id in the scskeystore directory, used for SCS identification to send deposit and receive MicroChain mining rewards.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getSCSId","params":[],"id":101}'
↪'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": "0x9d711986ccc8c89db2dfaf0894acadeb5a383ee8"
}
```

---

scs_getReceiptByHash

Returns the receipt of a transaction by transaction hash. Note That the receipt is not available for pending transactions.

Parameters

1. `String` - The MicroChain address.
2. `String` - The transaction hash.

Returns

`Object` - A transaction receipt object, or null when no receipt was found:.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getReceiptByHash","params":[
↪"0x299afff2da4a57e7e0a0a16bf626f8822b8a3158",
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"],"id":101}'
↪'localhost:8545'

// Result
{
  "id":101,
  "jsonrpc": "2.0",
  "result": {contractAddress: '0x0a674edac2ccd47ae2a3197ea3163aa81087fbd1',
failed: false,"logs":[{"address":"0x2328537bc943ab1a89fe94a4b562ee7a7b013634","topics":[
↪"0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
↪"0x000000000000000000000000a8863fc8ce3816411378685223c03daae9770ebb",
↪"0x0000000000000000000000007312f4b8a4457a36827f185325fd6b66a3f8bb8b"],"data":
↪"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGQ=","blockNumber":0,"transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69","transactionIndex
↪":0,"blockHash":"0x78f092ca81a891ad6c467caa2881d00d8e19c8925ddfd71d793294fbfc5f15fe",
↪"logIndex":0,"removed":false}],"logsBloom":
↪"0x00000000000000000000000000000000008000000000000000000000000000000000000000000000000000000000000000
↪","status":"0x1","transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"}
}
```

scs_getReceiptByNonce

Returns the transaction result by address and nonce on the MicroChain. Note That the nonce is the nonce on the MicroChain. This nonce can be checked using scs_getNonce.

Parameters

1. `String` - The MicroChain address. 1. `String` - The transaction nonce. 1. `QUANTITY` - The nonce of the transaction.

Returns

`Object` - A transaction receipt object, or null when no receipt was found:.

Example

```
  // Request
  curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getReceiptByNonce","params":[
↪"0x299afff2da4a57e7e0a0a16bf626f8822b8a3158",
↪"0xa8863fc8ce3816411378685223c03daae9770ebb", 0],"id":101}' 'localhost:8545'

  // Result
  {
    "id":101,
    "jsonrpc": "2.0",
    "result": {contractAddress: '0x0a674edac2ccd47ae2a3197ea3163aa81087fbd1',
failed: false,"logs":[{"address":"0x2328537bc943ab1a89fe94a4b562ee7a7b013634","topics":[
↪"0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
↪"0x000000000000000000000000a8863fc8ce3816411378685223c03daae9770ebb",
↪"0x0000000000000000000000007312f4b8a4457a36827f185325fd6b66a3f8bb8b"],"data":
↪"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGQ=","blockNumber":0,"transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69","transactionIndex
↪":0,"blockHash":"0x78f092ca81a891ad6c467caa2881d00d8e19c8925ddfd71d793294fbfc5f15fe",
↪"logIndex":0,"removed":false}],"logsBloom":
↪"0x00000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪","status":"0x1","transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"}
  }
```

---

scs_getTransactionByHash

Returns the receipt of a transaction by transaction hash. Note That the receipt is not available for pending transactions.

Parameters

1. `String` - The MicroChain address.
2. `String` - The transaction hash.

Returns

`Object` - A transaction object, or null when no transaction was found.

Example

```
  // Request
  curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getTransactionByHash","params":[
↪"0x299afff2da4a57e7e0a0a16bf626f8822b8a3158",
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"],"id":101}'
↪'localhost:8545'

  // Result
  {
    "id":101,
    "jsonrpc": "2.0",
    "result": {contractAddress: '0x0a674edac2ccd47ae2a3197ea3163aa81087fbd1',
failed: false,"logs":[{"address":"0x2328537bc943ab1a89fe94a4b562ee7a7b013634","topics":[
↪"0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
↪"0x0000000000000000000000000a8863fc8ce3816411378685223c03daae9770ebb",
↪"0x0000000000000000000000007312f4b8a4457a36827f185325fd6b66a3f8bb8b"],"data":
↪"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGQ=","blockNumber":0,"transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69","transactionIndex
↪":0,"blockHash":"0x78f092ca81a891ad6c467caa2881d00d8e19c8925ddfd71d793294fbfc5f15fe",
↪"logIndex":0,"removed":false}],"logsBloom":
↪"0x00000000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪","status":"0x1","transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"}
  }
```

---

scs_getTransactionByNonce

Returns the receipt of a transaction by transaction hash. Note That the receipt is not available for pending transactions.

Parameters

1. `String` - The MicroChain address. 1. `String` - The transaction nonce. 1. `QUANTITY` - The nonce of the transaction.

Returns

`Object` - A transaction receipt object, or null when no receipt was found:.

Example

```
  // Request
  curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getTransactionByNonce","params":[
↪"0x299afff2da4a57e7e0a0a16bf626f8822b8a3158",
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"],"id":101}'
↪'localhost:8545'
```

(下页继续)

(续上页)

```
  // Result
  {
    "id":101,
    "jsonrpc": "2.0",
    "result": {contractAddress: '0x0a674edac2ccd47ae2a3197ea3163aa81087fbd1',
failed: false,"logs":[{"address":"0x2328537bc943ab1a89fe94a4b562ee7a7b013634","topics":[
↪"0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
↪"0x000000000000000000000000a8863fc8ce3816411378685223c03daae9770ebb",
↪"0x000000000000000000000000312f4b8a4457a36827f185325fd6b66a3f8bb8b"],"data":
↪"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGQ=","blockNumber":0,"transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69","transactionIndex
↪":0,"blockHash":"0x78f092ca81a891ad6c467caa2881d00d8e19c8925ddfd71d793294fbfc5f15fe",
↪"logIndex":0,"removed":false}],"logsBloom":
↪"0x00000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪","status":"0x1","transactionHash":
↪"0x67bfaa5a704e77a31d5e7eb866f8c662fa8313a7882d13d0d23e377cd66d2a69"}
```

scs_getExchangeByAddress

Returns the Withdraw/Deposit exchange records between MicroChain and MotherChain for a certain address. This command returns both the ongoing exchanges and processed exchanges. To check all the ongoing exchanges, please use scs_getExchangeInfo.

Parameters

1. *String* - The MicroChain address. 1. *String* - The address to be checked. 1. *Int* - Index of Deposit records >= 0. 1. *Int* - Number of Deposit records extracted. 1. *Int* - Index of Depositing records >= 0. 1. *Int* - Number of Depositing records extracted. 1. *Int* - Index of Withdraw records >= 0. 1. *Int* - Number of Withdraw records extracted. 1. *Int* - Index of Withdrawing records >= 0. 1. *Int* - Number of Withdrawing records extracted.

Returns

`Object` - A JSON format object contains the token exchange info.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getExchangeByAddress","params":[
↪"0x2e4694875de2a7da9c3186a176c52760d58694e4",
↪"0xa8863fc8ce3816411378685223c03daae9770ebb", 0,10,0,10,0,10,0,10],"id":100}'␣
↪localhost:8548
```

(下页继续)

(续上页)

```
// Result
{"jsonrpc":"2.0","id":100,"result":{"DepositRecordCount":2,"DepositRecords":[null,null,{
↪"DepositAmt":"0x18abedda5a37000","Deposittime":"0x5c7f03c4"},{"DepositAmt":
↪"0x2bdbb64bc09000","Deposittime":"0x5c7e8aaa"}],"DepositingRecordCount":0,
↪"DepositingRecords":null,"WithdrawRecordCount":0,"WithdrawRecords":null,
↪"WithdrawingRecordCount":0,"WithdrawingRecords":null,"microchain":
↪"0x2e4694875de2a7da9c3186a176c52760d58694e4","sender":
↪"0xa8863fc8ce3816411378685223c03daae9770ebb"}}
```

---

scs_getExchangeInfo

Returns the Withdraw/Deposit exchange records between MicroChain and MotherChain for a certain address. This command returns both the ongoing exchanges and processed exchanges. To check all the ongoing exchanges, please use scs_getExchangeInfo.

Parameters

1. *String* - The MicroChain address. 1. *String* - The transaction hash. 1. *Int* - Index of Depositing records >= 0. 1. *Int* - Number of Depositing records extracted. 1. *Int* - Index of Withdrawing records >= 0. 1. *Int* - Number of Withdrawing records extracted.

Returns

`Object` - A JSON format object contains the token exchange info.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getExchangeInfo","params":[
↪"0x2e4694875de2a7da9c3186a176c52760d58694e4",0,10,0,10],"id":101}' 'localhost:8545'

// Result

{"jsonrpc":"2.0","id":100,"result":{"DepositingRecordCount":0,"DepositingRecords":null,
↪"WithdrawingRecordCount":0,"WithdrawingRecords":null,"microchain":
↪"0x2e4694875de2a7da9c3186a176c52760d58694e4","scsid":
↪"0x50c15fafb95968132d1a6ee3617e99cca1fcf059"}}
```

---

scs_getTxpool

Returns the ongoing transactions in the MicroChain.

Parameters

1. *String* - The MicroChain address.

Returns

`Object` - A JSON format object contains two fields pending and queued. Each of these fields are associative arrays, in which each entry maps an origin-address to a batch of scheduled transactions. These batches themselves are maps associating nonces with actual transactions.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"scs_getTxpool","params":[
↪"0x2e4694875de2a7da9c3186a176c52760d58694e4"],"id":101}' 'localhost:8545'

// Result

{"jsonrpc":"2.0","id":100,"result":{"pending":{},"queued":{}}}
```

2.5 Chain3 JavaScript 软件库

Chain3 JavaScript 软件库是墨客开发的一套 javascript 库，目的是让应用程序能够使用简便的方式与墨客节点进行通信。注意，这里有两层，moac 启动了一个墨客母链节点，console 参数开启了一个 javascript 的控制台，这个控制台注入了 chain3.js 这个库，以使我们可以通过 chain3 对象与墨客节点的 JSON-RPC 接口做交互。

开发者可以通过 ‘‘chain3‘‘对象来对 DApp 的函数进行调用。chain3 通过 RPC calls 来和母链服务器 VNODE. 但注意要求连接的 VNODE 服务器打开 RPC 接口：

./moac –rpc –rpcport 8545

`chain3` 除了包含 `mc` 对象 - `chain3.mc` 与母链节点互动之外，在 accounts 下面的文件中提供了交易签名功能，以及对消息签名和验证的功能，以方便用户使用。`chain3` 还提供了和子链客户端 SCS 交互的对象 `scs` Over time we’ ll introduce other objects such as SCS for the other chain3 protocols. 更多的例子可以在源文件中的 example 下找到：here.

2.5.1 使用回调函数（CALLBACK）

As this API is designed to work with a local RPC node and all its functions are by default use synchronous HTTP requests.con

If you want to make an asynchronous request, you can pass an optional callback as the last parameter to most functions. All callbacks are using an error first callback style:

```
chain3.mc.getBlock(48, function(error, result){
    if(!error)
        console.log(result)
    else
        console.error(error);
})
```

2.5.2 批处理执行命令

Batch requests allow queuing up requests and processing them at once.

```
var batch = chain3.createBatch();
batch.add(chain3.mc.getBalance.request('0x0000000000000000000000000000000000000000',
↪'latest', callback));
batch.add(chain3.mc.contract(abi).at(address).balance.request(address, callback2));
batch.execute();
```

2.5.3 A note on big numbers in chain3.js

You will always get a BigNumber object for balance values as JavaScript is not able to handle big numbers correctly. Look at the following examples:

```
"1010101003243253453464564564564564564"
// "1010101003243253453464564564564564564"
101010100324325345346456456456456456
// 1.0101010032432535e+38
```

chain3.js depends on the BigNumber Library and adds it automatically.

```
var balance = new BigNumber('123456786979284780000000000');
// or var balance = chain3.mc.getBalance(someAddress);

balance.plus(21).toString(10); // toString(10) converts it to a number string
// "123456786979284780000000021"
```

The next example wouldn' t work as we have more than 20 floating points, therefore it is recommended that you always keep your balance in *sha* and only transform it to other units when presenting to the user:

```
var balance = new BigNumber('13124.234435346456466666457455567456');
```

(下页继续)

(续上页)

```
balance.plus(21).toString(10); // toString(10) converts it to a number string, but can␣
↪only show max 20 floating points
// "13145.234435346456466666646" // you number would be cut after the 20 floating point
```

2.5.4 Chain3 Javascript Đapp API Reference

- *chain3*
- *version*
  - *api*
  - *node*
  - *network*
  - *moac*
- *isConnected()*
- *setProvider(provider)*
- *currentProvider*
- *reset()*
- *sha3(string)*
- *toHex(stringOrNumber)*
- *toAscii(hexString)*
- *fromAscii(textString, [padding])*
- *toDecimal(hexString)*
- *toChecksumAddress(string)*
- *fromDecimal(number)*
- *fromSha(numberStringOrBigNumber, unit)*
- *toSha(numberStringOrBigNumber, unit)*
- *toBigNumber(numberOrHexString)*
- *isAddress(hexString)*
- *net*
  - *listening/getListening*
  - *peerCount/getPeerCount*

- *mc*
  - *defaultAccount*
  - *defaultBlock*
  - *syncing/getSyncing*
  - *isSyncing*
  - *coinbase/getCoinbase*
  - *hashrate/getHashrate*
  - *gasPrice/getGasPrice*
  - *accounts/getAccounts*
  - *mining/getMining*
  - *blockNumber/getBlockNumber*
  - *getBalance(address)*
  - *getStorageAt(address, position)*
  - *getCode(address)*
  - *getBlock(hash/number)*
  - *getBlockTransactionCount(hash/number)*
  - *getUncle(hash/number)*
  - *getBlockUncleCount(hash/number)*
  - *getTransaction(hash)*
  - *getTransactionFromBlock(hashOrNumber, indexNumber)*
  - *getTransactionReceipt(hash)*
  - *getTransactionCount(address)*
  - *sendTransaction(object)*
  - *call(object)*
  - *estimateGas(object)*
  - *filter(array (, options))*
    - *watch(callback)*
    - *stopWatching(callback)*
    - *get()*
  - *contract(abiArray)*

- *contract.myMethod()*
- *contract.myEvent()*
- *contract.allEvents()*
- *encodeParams*
- *namereg*
- *sendIBANTransaction*
- *iban*
- *fromAddress*
- *fromBban*
- *createIndirect*
- *isValid*
- *isDirect*
- *isIndirect*
- *checksum*
- *institution*
- *client*
- *address*
- *toString*

Usage

chain3.version.api

The `chain3` object provides all methods.

Example

```
var Chain3 = require('chain3');
// create an instance of chain3 using the HTTP provider.
var chain3 = new Chain3(new Chain3.providers.HttpProvider("http://localhost:8545"));
```

---

chain3.version.api

```
chain3.version.api
// or async
chain3.version.getApi(callback(error, result){ ... })
```

Returns

`String` - The moac js api version.

Example

```
var version = chain3.version.api;
console.log(version); // "0.2.0"
```

---

chain3.version.node

```
chain3.version.node
// or async
chain3.version.getClient(callback(error, result){ ... })
```

Returns

`String` - The client/node version.

Example

```
var version = chain3.version.node;
console.log(version); // "Moac/v0.1.0-develop/darwin-amd64/go1.9"
```

---

chain3.version.network

```
chain3.version.network
// or async
chain3.version.getNetwork(callback(error, result){ ... })
```

Returns

`String` - The network protocol version.

Example

```
var version = chain3.version.network;
console.log(version); // 54
```

---

chain3.version.moac

```
chain3.version.moac
// or async
chain3.version.getMoac(callback(error, result){ ... })
```

Returns

`String` - The moac protocol version.

Example

```
var version = chain3.version.moac;
console.log(version); // 0x3f
```

---

chain3.isConnected

```
chain3.isConnected()
```

Should be called to check if a connection to a node exists

Parameters

none

Returns

`Boolean`

Example

```
if(!chain3.isConnected()) {

   // show some dialog to ask the user to start a node

} else {

   // start chain3 filters, calls, etc

}
```

---

chain3.setProvider

```
chain3.setProvider(provider)
```

Should be called to set provider.

Parameters

none

Returns

undefined

Example

```
chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545')); // 8545␣
↪for go/moac
```

---

chain3.currentProvider

```
chain3.currentProvider
```

Will contain the current provider, if one is set. This can be used to check if moac etc. set already a provider.

Returns

`Object` - The provider set or `null`;

Example

```
// Check if moac etc. already set a provider
if(!chain3.currentProvider)
    chain3.setProvider(new chain3.providers.HttpProvider("http://localhost:8545"));
```

---

chain3.reset

```
chain3.reset(keepIsSyncing)
```

Should be called to reset state of chain3. Resets everything except manager. Uninstalls all filters. Stops polling.

Parameters

1. `Boolean` - If `true` it will uninstall all filters, but will keep the *chain3.mc.isSyncing()* polls

Returns

`undefined`

Example

```
chain3.reset();
```

---

chain3.sha3

```
chain3.sha3(string [, callback])
```

Parameters

1. `String` - The string to hash using the SHA3 algorithm
2. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - The SHA3 of the given data.

Example

```
var str = chain3.sha3("Some ASCII string to be hashed in MOAC");
console.log(str); // "0xbfa24877cd68e6734710402a2af3e29cf18bd6d2f304aa528ffa3a32fa7652d2"
```

---

chain3.toHex

```
chain3.toHex(mixed);
```

Converts any value into HEX.

Parameters

1. `String|Number|Object|Array|BigNumber` - The value to parse to HEX. If its an object or array it will be `JSON.stringify` first. If its a BigNumber it will make it the HEX value of a number.

Returns

`String` - The hex string of `mixed`.

Example

```
var str = chain3.toHex("moac network");
console.log(str); // '0x6d6f6163206e6574776f726b'

console.log(chain3.toHex({moac: 'test'}));
//'0x7b226d6f6163223a2274657374227d'
```

---

chain3.toAscii

chain3.toAscii

```
chain3.toAscii(hexString);
```

Converts a HEX string into a ASCII string.

Parameters

1. `String` - A HEX string to be converted to ascii.

Returns

`String` - An ASCII string made from the given `hexString`.

Example

```
var str = chain3.toAscii("0x0x6d6f6163206e6574776f726b");
console.log(str); // "moac network"
```

---

chain3.fromAscii

```
chain3.fromAscii(string);
```

Converts any ASCII string to a HEX string.

Parameters

`String` - An ASCII string to be converted to HEX.

Returns

`String` - The converted HEX string.

Example

```
var str = chain3.fromAscii('moac network');
console.log(str); // "0x6d6f6163206e6574776f726b"
```

---

chain3.toChecksumAddress

```
chain3.toChecksumAddress(hexString);
```

Converts a string to the checksummed address equivalent.

Parameters

1. `String` - A string to be converted to a checksummed address.

Returns

`String` - A string containing the checksummed address.

Example

```
var myAddress = chain3.toChecksumAddress('0xa0c876ec9f2d817c4304a727536f36363840c02c');
console.log(myAddress); // '0xA0C876eC9F2d817c4304A727536f36363840c02c'
```

---

chain3.toDecimal

```
chain3.toDecimal(hexString);
```

Converts a HEX string to its number representation.

Parameters

1. `String` - An HEX string to be converted to a number.

Returns

`Number` - The number representing the data `hexString`.

Example

```
var number = chain3.toDecimal('0x15');
console.log(number); // 21
```

---

chain3.fromDecimal

```
chain3.fromDecimal(number);
```

Converts a number or number string to its HEX representation.

Parameters

1. `Number|String` - A number to be converted to a HEX string.

Returns

`String` - The HEX string representing of the given `number`.

Example

```
var value = chain3.fromDecimal('21');
console.log(value); // "0x15"
```

---

chain3.fromSha

```
chain3.fromSha(number, unit)
```

Converts a number of sha into the following moac units:

- `ksha/femtomc`
- `msha/picomc`
- `gsha/nano/xiao`
- `micro/sand`
- `milli`
- `mc`
- `kmc/grand`
- `mmc`
- `gmc`
- `tmc`

Parameters

1. `Number|String|BigNumber` - A number or BigNumber instance.
2. `String` - One of the above moac units.

Returns

`String|BigNumber` - Either a number string, or a BigNumber instance, depending on the given `number` parameter.

Example

```
var value = chain3.fromSha('21000000000000', 'Xiao');
console.log(value); // "21000"
```

---

chain3.toSha

```
chain3.toSha(number, unit)
```

Converts a moac unit into sha. Possible units are:

- ksha/femtomc
- msha/picomc
- gsha/nano/xiao
- micro/sand
- milli
- mc
- kmc/grand
- mmc
- gmc
- tmc

Parameters

1. `Number|String|BigNumber` - A number or BigNumber instance.
2. `String` - One of the above moac units.

Returns

`String|BigNumber` - Either a number string, or a BigNumber instance, depending on the given `number` parameter.

Example

```
var value = chain3.toSha('1', 'mc');
console.log(value); // "1000000000000000000" = 1e18
```

---

chain3.toBigNumber

```
chain3.toBigNumber(numberOrHexString);
```

Converts a given number into a BigNumber instance.

See the *note on BigNumber*.

Parameters

1. `Number|String` - A number, number string or HEX string of a number.

Returns

`BigNumber` - A BigNumber instance representing the given value.

Example

```
var value = chain3.toBigNumber('200000000000000000000001');
console.log(value); // instanceOf BigNumber, { [String: '2.000000000000000000000001e+23']␣
↪s: 1, e: 23, c: [ 2000000000, 1 ] }
console.log(value.toNumber()); // 2.000000000000002e+23
console.log(value.toString(10)); // '200000000000000000000001'
```

---

chain3.net

chain3.net.listening

```
chain3.net.listening
// or async
chain3.net.getListening(callback(error, result){ ... })
```

This property is read only and says whether the node is actively listening for network connections or not.

Returns

`Boolean` - `true` if the client is actively listening for network connections, otherwise `false`.

Example

```
var listening = chain3.net.listening;
console.log(listening); // true of false
```

---

chain3.net.peerCount

```
chain3.net.peerCount
// or async
chain3.net.getPeerCount(callback(error, result){ ... })
```

This property is read only and returns the number of connected peers.

Returns

`Number` - The number of peers currently connected to the client.

Example

```
var peerCount = chain3.net.peerCount;
console.log(peerCount); // 4
```

---

chain3.mc

Contains the MOAC blockchain related methods.

Example

```
var mc = chain3.mc;
```

---

chain3.mc.defaultAccount

```
chain3.mc.defaultAccount
```

This default address is used for the following methods (optionally you can overwrite it by specifying the `from` property):

- *chain3.mc.sendTransaction()*
- *chain3.mc.call()*

Values

`String`, 20 Bytes - Any address you own, or where you have the private key for.

Default is `undefined`.

Returns

`String`, 20 Bytes - The currently set default address.

Example

```
var defaultAccount = chain3.mc.defaultAccount;
console.log(defaultAccount); // 'undefined'

// set the default block
chain3.mc.defaultAccount = '0x8888f1f195afa192cfee860698584c030f4c9db1';
console.log(chain3.mc.defaultAccount);//'0x8888f1f195afa192cfee860698584c030f4c9db1'
```

---

chain3.mc.defaultBlock

```
chain3.mc.defaultBlock
```

This default block is used for the following methods (optionally you can overwrite the defaultBlock by passing it as the last parameter):

- *chain3.mc.getBalance()*
- *chain3.mc.getCode()*
- *chain3.mc.getTransactionCount()*
- *chain3.mc.getStorageAt()*
- *chain3.mc.call()*

Values

Default block parameters can be one of the following:

- `Number` - a block number
- `String` - `"earliest"`, the genisis block
- `String` - `"latest"`, the latest block (current head of the blockchain)

- `String` - `"pending"`, the currently mined block (including pending transactions)

Default is `latest`

Returns

`Number|String` - The default block number to use when querying a state.

Example

```
var defaultBlock = chain3.mc.defaultBlock;
console.log(defaultBlock); // 'latest'

// set the default block
chain3.mc.defaultBlock = 12345;
console.log(chain3.mc.defaultAccount);//'12345'
```

---

chain3.mc.syncing

```
chain3.mc.syncing
// or async
chain3.mc.getSyncing(callback(error, result){ ... })
```

This property is read only and returns the either a sync object, when the node is syncing or `false`.

Returns

`Object|Boolean` - A sync object as follows, when the node is currently syncing or `false`: - `startingBlock`: `Number` - The block number where the sync started. - `currentBlock`: `Number` - The block number where at which block the node currently synced to already. - `highestBlock`: `Number` - The estimated block number to sync to.

Example

```
var sync = chain3.mc.syncing;
console.log(sync);
/*
{
   startingBlock: 300,
```

(下页继续)

(续上页)

```
   currentBlock: 312,
   highestBlock: 512
}
*/
```

---

chain3.mc.isSyncing

```
chain3.mc.isSyncing(callback);
```

This convenience function calls the `callback` everytime a sync starts, updates and stops.

Returns

`Object` - a isSyncing object with the following methods:

- `syncing.addCallback()`: Adds another callback, which will be called when the node starts or stops syncing.
- `syncing.stopWatching()`: Stops the syncing callbacks.

Callback return value

- `Boolean` - The callback will be fired with `true` when the syncing starts and with `false` when it stopped.
- `Object` - While syncing it will return the syncing object:
- `startingBlock`: `Number` - The block number where the sync started.
- `currentBlock`: `Number` - The block number where at which block the node currently synced to already.
- `highestBlock`: `Number` - The estimated block number to sync to.

Example

```
chain3.mc.isSyncing(function(error, sync){
    if(!error) {
        // stop all app activity
        if(sync === true) {
           // we use `true`, so it stops all filters, but not the chain3.mc.syncing␣
↪polling
           chain3.reset(true);
```

(下页继续)

(续上页)

```
        // show sync info
        } else if(sync) {
           console.log(sync.currentBlock);

        // re-gain app operation
        } else {
            // run your app init function...
        }
    }
});
```

---

chain3.mc.coinbase

```
chain3.mc.coinbase
// or async
chain3.mc.getCoinbase(callback(error, result){ ... })
```

This property is read only and returns the coinbase address were the mining rewards go to.

Returns

`String` - The coinbase address of the client.

Example

```
var coinbase = chain3.mc.coinbase;
console.log(coinbase); // "0x407d73d8a49eeb85d32cf465507dd71d507100c1"
```

---

chain3.mc.mining

```
chain3.mc.mining
// or async
chain3.mc.getMining(callback(error, result){ ... })
```

This property is read only and says whether the node is mining or not.

Returns

`Boolean` - `true` if the client is mining, otherwise `false`.

Example

```
var mining = chain3.mc.mining;
console.log(mining); // true or false
```

---

chain3.mc.hashrate

```
chain3.mc.hashrate
// or async
chain3.mc.getHashrate(callback(error, result){ ... })
```

This property is read only and returns the number of hashes per second that the node is mining with.

Returns

`Number` - number of hashes per second.

Example

```
var hashrate = chain3.mc.hashrate;
console.log(hashrate); // 493736
```

---

chain3.mc.gasPrice

```
chain3.mc.gasPrice
// or async
chain3.mc.getGasPrice(callback(error, result){ ... })
```

This property is read only and returns the current gas price. The gas price is determined by the x latest blocks median gas price.

Returns

`BigNumber` - A BigNumber instance of the current gas price in sha.

See the *note on BigNumber*.

Example

```
var gasPrice = chain3.mc.gasPrice;
console.log(gasPrice.toString(10)); // "20000000000"
```

---

chain3.mc.accounts

```
chain3.mc.accounts
// or async
chain3.mc.getAccounts(callback(error, result){ ... })
```

This property is read only and returns a list of accounts the node controls.

Returns

`Array` - An array of addresses controlled by client.

Example

```
var accounts = chain3.mc.accounts;
console.log(accounts); // ["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]
```

---

chain3.mc.blockNumber

```
chain3.mc.blockNumber
// or async
chain3.mc.getBlockNumber(callback(error, result){ ... })
```

This property is read only and returns the current block number.

Returns

`Number` - The number of the most recent block.

Example

```
var number = chain3.mc.blockNumber;
console.log(number); // 2744
```

---

chain3.mc.getBalance

```
chain3.mc.getBalance(addressHexString [, defaultBlock] [, callback])
```

Get the balance of an address at a given block.

Parameters

1. `String` - The address to get the balance of.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *chain3.mc.defaultBlock*.
3. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - A BigNumber instance of the current balance for the given address in sha.

See the *note on BigNumber*.

Example

```
var balance = chain3.mc.getBalance("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
console.log(balance); // instanceof BigNumber
console.log(balance.toString(10)); // '1000000000000'
console.log(balance.toNumber()); // 1000000000000
```

---

chain3.mc.getStorageAt

```
chain3.mc.getStorageAt(addressHexString, position [, defaultBlock] [, callback])
```

Get the storage at a specific position of an address.

Parameters

1. `String` - The address to get the storage from.
2. `Number` - The index position of the storage.

3. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *chain3.mc.defaultBlock*.
4. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - The value in storage at the given position.

Example

```
var state = chain3.mc.getStorageAt("0x407d73d8a49eeb85d32cf465507dd71d507100c1", 0);
console.log(state); // "0x03"
```

---

chain3.mc.getCode

```
chain3.mc.getCode(addressHexString [, defaultBlock] [, callback])
```

Get the code at a specific address. For a contract address, return the binary code of the contract.

Parameters

1. `String` - The address to get the code from.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *chain3.mc.defaultBlock*.
3. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - The data at given address `addressHexString`.

Example

```
var code = chain3.mc.getCode("0x1d12aec505caa2b8513cdad9a13e9d4806a1b704");
console.log(code); //
↪"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078202905091905
↪"
```

chain3.mc.getBlock

```
chain3.mc.getBlock(blockHashOrBlockNumber [, returnTransactionObjects] [, callback])
```

Returns a block matching the block number or block hash.

Parameters

1. `String|Number` - The block number or hash. Or the string `"earliest"`, `"latest"` or `"pending"` as in the *default block parameter*.
2. `Boolean` - (optional, default `false`) If `true`, the returned block will contain all transactions as objects, if `false` it will only contains the transaction hashes.
3. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Object` - The block object:

- `number`: `Number` - the block number. `null` when its pending block.
- `hash`: `String`, 32 Bytes - hash of the block. `null` when its pending block.
- `parentHash`: `String`, 32 Bytes - hash of the parent block.
- `nonce`: `String`, 8 Bytes - hash of the generated proof-of-work. `null` when its pending block.
- `sha3Uncles`: `String`, 32 Bytes - SHA3 of the uncles data in the block.
- `logsBloom`: `String`, 256 Bytes - the bloom filter for the logs of the block. `null` when its pending block.
- `transactionsRoot`: `String`, 32 Bytes - the root of the transaction trie of the block
- `stateRoot`: `String`, 32 Bytes - the root of the final state trie of the block.
- `miner`: `String`, 20 Bytes - the address of the beneficiary to whom the mining rewards were given.
- `difficulty`: `BigNumber` - integer of the difficulty for this block.
- `totalDifficulty`: `BigNumber` - integer of the total difficulty of the chain until this block.
- `extraData`: `String` - the "extra data" field of this block.
- `size`: `Number` - integer the size of this block in bytes.
- `gasLimit`: `Number` - the maximum gas allowed in this block.

- `gasUsed`: `Number` - the total used gas by all transactions in this block.
- `timestamp`: `Number` - the unix timestamp for when the block was collated.
- `transactions`: `Array` - Array of transaction objects, or 32 Bytes transaction hashes depending on the last given parameter.
- `uncles`: `Array` - Array of uncle hashes.

Example

```
var info = chain3.mc.getBlock(0);
console.log(info);
/*
{ difficulty: { [String: '100000000000'] s: 1, e: 11, c: [ 100000000000 ] },
  extraData: '0x31393639415250414e4554373354435049503039425443323031384d4f4143',
  gasLimit: 9000000,
  gasUsed: 0,
  hash: '0x6b9661646439fab926ffc9bccdf3abb572d5209ae59e3390abf76aee4e2d49cd',
  logsBloom:
↪'0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
  miner: '0x0000000000000000000000000000000000000000',
  mixHash: '0x0000000000000000000000000000000000000000000000000000000000000000',
  nonce: '0x0000000000000042',
  number: 0,
  parentHash: '0x0000000000000000000000000000000000000000000000000000000000000000',
  receiptsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
  sha3Uncles: '0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347',
  size: 540,
  stateRoot: '0xe221c9e4ad19514d7ce3e6b0bec3ad7f6cc293336e59c301cda293cfbda83df6',
  timestamp: 0,
  totalDifficulty: { [String: '100000000000'] s: 1, e: 11, c: [ 100000000000 ] },
  transactions: [],
  transactionsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
  uncles: [] }
*/
```

---

chain3.mc.getBlockTransactionCount

```
chain3.mc.getBlockTransactionCount(hashStringOrBlockNumber [, callback])
```

Returns the number of transaction in a given block.

Parameters

1. `String|Number` - The block number or hash. Or the string `"earliest"`, `"latest"` or `"pending"` as in the *default block parameter*.
2. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Number` - The number of transactions in the given block.

Example

```
var number = chain3.mc.getBlockTransactionCount(
↪"0xddfb8508bff841242099e640efe59f5e5252be1a60fa701d333e1a8bfdee6263");
console.log(number); // 1
```

---

chain3.mc.getUncle

```
chain3.mc.getUncle(blockHashStringOrNumber, uncleNumber [, returnTransactionObjects] [,␣
↪callback])
```

Returns a blocks uncle by a given uncle index position.

Parameters

1. `String|Number` - The block number or hash. Or the string `"earliest"`, `"latest"` or `"pending"` as in the *default block parameter*.
2. `Number` - The index position of the uncle.
3. `Boolean` - (optional, default `false`) If `true`, the returned block will contain all transactions as objects, if `false` it will only contains the transaction hashes.
4. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Object` - the returned uncle. For a return value see *chain3.mc.getBlock()*.

Note: An uncle doesn't contain individual transactions.

Example

```
var uncle = chain3.mc.getUncle(500, 0);
console.log(uncle); // see chain3.mc.getBlock
```

---

chain3.mc.getTransaction

```
chain3.mc.getTransaction(transactionHash [, callback])
```

Returns a transaction matching the given transaction hash.

Parameters

1. `String` - The transaction hash.
2. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Object` - A transaction object its hash `transactionHash`:

- `hash`: `String`, 32 Bytes - hash of the transaction.
- `nonce`: `Number` - the number of transactions made by the sender prior to this one.
- `blockHash`: `String`, 32 Bytes - hash of the block where this transaction was in. `null` when its pending.
- `blockNumber`: `Number` - block number where this transaction was in. `null` when its pending.
- `transactionIndex`: `Number` - integer of the transactions index position in the block. `null` when its pending.
- `from`: `String`, 20 Bytes - address of the sender.
- `to`: `String`, 20 Bytes - address of the receiver. `null` when its a contract creation transaction.
- `value`: `BigNumber` - value transferred in Sha.
- `gasPrice`: `BigNumber` - gas price provided by the sender in Sha.
- `gas`: `Number` - gas provided by the sender.
- `input`: `String` - the data sent along with the transaction.

Example

```
var txhash = "0x687751dd47684f4b5df263ae4ec39f54f057d0e2a1dde56f9d52766849c9c7fe";

var transaction = chain3.mc.getTransaction(txhash);
console.log(transaction);
/*
{ blockHash: '0x716c602d49055b4e24fbe7da952c1ad81820ad0401f3cb3ce12e832fbcc368f5',
  blockNumber: 57259,
  from: '0x7cfd775c7a97aa632846eff35dcf9dbcf502d0f3',
  gas: 1000,
  gasPrice: { [String: '200000000000'] s: 1, e: 11, c: [ 200000000000 ] },
  hash: '0xf1c1771204431c1c584e793b49d41586a923c370be93673aac42d66252bc8d0a',
  input: '0x',
  nonce: 840,
  syscnt: '0x0',
  to: '0x3435410589ebd06b74079a1e141759d8502aeb8b',
  transactionIndex: 689,
  value: { [String: '1000000000'] s: 1, e: 9, c: [ 1000000000 ] },
  v: '0xea',
  r: '0x77aa72d0900e436b60f8be377e43dead3567241a85fbeb5517283e2dc8aca2b4',
  s: '0x24477ec130fc4101289b37e6107acfe64026c898994273110faa8f19b8ea6985',
  shardingFlag: '0x0' }
*/
```

---

chain3.mc.getTransactionFromBlock

```
getTransactionFromBlock(hashStringOrNumber, indexNumber [, callback])
```

Returns a transaction based on a block hash or number and the transactions index position.

Parameters

1. `String` - A block number or hash. Or the string `"earliest"`, `"latest"` or `"pending"` as in the *default block parameter*.
2. `Number` - The transactions index position.
3. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Object` - A transaction object, see *chain3.mc.getTransaction*:

Example

```
var transaction = chain3.mc.getTransactionFromBlock(921, 2);
console.log(transaction); // see chain3.mc.getTransaction
```

---

chain3.mc.getTransactionReceipt

```
chain3.mc.getTransactionReceipt(hashString [, callback])
```

Returns the receipt of a transaction by transaction hash.

Note That the receipt is not available for pending transactions.

Parameters

1. `String` - The transaction hash.
2. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Object` - A transaction receipt object, or `null` when no receipt was found:

- `blockHash`: `String`, 32 Bytes - hash of the block where this transaction was in.
- `blockNumber`: `Number` - block number where this transaction was in.
- `transactionHash`: `String`, 32 Bytes - hash of the transaction.
- `transactionIndex`: `Number` - integer of the transactions index position in the block.
- `from`: `String`, 20 Bytes - address of the sender.
- `to`: `String`, 20 Bytes - address of the receiver. `null` when its a contract creation transaction.
- `cumulativeGasUsed`: `Number` - The total amount of gas used when this transaction was executed in the block.
- `gasUsed`: `Number` - The amount of gas used by this specific transaction alone.
- `contractAddress`: `String` - 20 Bytes - The contract address created, if the transaction was a contract creation, otherwise `null`.

- `logs`: `Array` - Array of log objects, which this transaction generated.

Example

```
var receipt = chain3.mc.getTransactionReceipt(
↪'0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b');
console.log(receipt);
{
  "transactionHash": "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b
↪",
  "transactionIndex": 0,
  "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "contractAddress": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
  "cumulativeGasUsed": 314159,
  "gasUsed": 30234,
  "logs": [{
         // logs as returned by getFilterLogs, etc.
     }, ...]
}
```

---

chain3.mc.getTransactionCount

```
chain3.mc.getTransactionCount(addressHexString [, defaultBlock] [, callback])
```

Get the numbers of transactions sent from this address.

Parameters

1. `String` - The address to get the numbers of transactions from.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *chain3.mc.defaultBlock*.
3. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`Number` - The number of transactions sent from the given address.

Example

```
var number = chain3.mc.getTransactionCount("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
console.log(number); // 1
```

---

chain3.mc.sendTransaction

```
chain3.mc.sendTransaction(transactionObject [, callback])
```

Sends a transaction to the network.

Parameters

1. `Object` - The transaction object to send:

- `from`: `String` - The address for the sending account. Uses the *chain3.mc.defaultAccount* property, if not specified.
- `to`: `String` - (optional) The destination address of the message, can be an account address or a contract address, left undefined for a contract-creation transaction.
- `value`: `Number|String|BigNumber` - (optional) The value transferred for the transaction in Sha, also the endowment if it' s a contract-creation transaction.
- `gas`: `Number|String|BigNumber` - (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded).
- `gasPrice`: `Number|String|BigNumber` - (optional, default: To-Be-Determined) The price of gas for this transaction in sha, defaults to the mean network gas price.
- `data`: `String` - (optional) Either a byte string containing the associated data of the message, or in the case of a contract-creation transaction, the initialisation code.
- `nonce`: `Number` - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.
- `shardingFlag`: `Number|String` - (optional) Set to 1 for micro/subchain direct contract-call, 0 or undefine for global contract-creation transaction.

2. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - The 32 Bytes transaction hash as HEX string.

If the transaction was a contract creation use *chain3.mc.getTransactionReceipt()* to get the contract address, after the transaction was mined.

Example

```
// compiled solidity source code using https://chriseth.github.io/cpp-ethereum/
var code =
↪"603d80600c6000396000f3007c0100000000000000000000000000000000000000000000000000000060003504
8060005260206000f3";

chain3.mc.sendTransaction({data: code}, function(err, address) {
  if (!err)
    console.log(address); // "0x7f9fade1c0d57a7af66ab4ead7c2eb7b11a91385"
});
```

---

chain3.mc.call

```
chain3.mc.call(callObject [, defaultBlock] [, callback])
```

Executes a message call transaction, which is directly executed in the VM of the node, but never mined into the blockchain.

Parameters

1. `Object` - A transaction object see *chain3.mc.sendTransaction*, with the difference that for calls the `from` property is optional as well.
2. `Number|String` - (optional) If you pass this parameter it will not use the default block set with *chain3.mc.defaultBlock*.
3. `Function` - (optional) If you pass a callback the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - The returned data of the call, e.g. a codes functions return value.

Example

```
var result = chain3.mc.call({
    to: "0xc4abd0339eb8d57087278718986382264244252f",
    data: "0xc6888fa10000000000000000000000000000000000000000000000000000000000000003"
});
console.log(result); //
↪"0x000000000000000000000000000000000000000000000000000000000000015"
```

---

chain3.mc.estimateGas

```
chain3.mc.estimateGas(callObject [, defaultBlock] [, callback])
```

Executes a message call or transaction, which is directly executed in the VM of the node, but never mined into the blockchain and returns the amount of the gas used.

Parameters

See *chain3.mc.sendTransaction*, expect that all properties are optional.

Returns

`Number` - the used gas for the simulated call/transaction.

Example

```
var result = chain3.mc.estimateGas({
    to: "0xc4abd0339eb8d57087278718986382264244252f",
    data: "0xc6888fa10000000000000000000000000000000000000000000000000000000000000003"
});
console.log(result); // "1465"
```

---

chain3.mc.filter

```
// can be 'latest' or 'pending'
var filter = chain3.mc.filter(filterString);
// OR object are log filter options
var filter = chain3.mc.filter(options);

// watch for changes
```

(下页继续)

(续上页)

```
filter.watch(function(error, result){
  if (!error)
    console.log(result);
});

// Additionally you can start watching right away, by passing a callback:
chain3.mc.filter(options, function(error, result){
  if (!error)
    console.log(result);
});
```

Parameters

1. `String|Object` - The string `"latest"` or `"pending"` to watch for changes in the latest block or pending transactions respectively. Or a filter options object as follows:

- `fromBlock`: `Number|String` - The number of the earliest block (`latest` may be given to mean the most recent and `pending` currently mining, block). By default `latest`.
- `toBlock`: `Number|String` - The number of the latest block (`latest` may be given to mean the most recent and `pending` currently mining, block). By default `latest`.
- `address`: `String` - An address or a list of addresses to only get logs from particular account(s).
- `topics`: `Array of Strings` - An array of values which must each appear in the log entries. The order is important, if you want to leave topics out use `null`, e.g. `[null, '0x00...']`. You can also pass another array for each topic with options for that topic e.g. `[null, ['option1', 'option2']]`

Returns

`Object` - A filter object with the following methods:

- `filter.get(callback)`: Returns all of the log entries that fit the filter.
- `filter.watch(callback)`: Watches for state changes that fit the filter and calls the callback. See *this note* for details.
- `filter.stopWatching()`: Stops the watch and uninstalls the filter in the node. Should always be called once it is done.

Watch callback return value

- `String` - When using the `"latest"` parameter, it returns the block hash of the last incoming block.

- `String` - When using the `"pending"` parameter, it returns a transaction hash of the last add pending transaction.
- `Object` - When using manual filter options, it returns a log object as follows:
  - `logIndex`: `Number` - integer of the log index position in the block. `null` when its pending log.
  - `transactionIndex`: `Number` - integer of the transactions index position log was created from. `null` when its pending log.
  - `transactionHash`: `String`, 32 Bytes - hash of the transactions this log was created from. `null` when its pending log.
  - `blockHash`: `String`, 32 Bytes - hash of the block where this log was in. `null` when its pending. `null` when its pending log.
  - `blockNumber`: `Number` - the block number where this log was in. `null` when its pending. `null` when its pending log.
  - `address`: `String`, 32 Bytes - address from which this log originated.
  - `data`: `String` - contains one or more 32 Bytes non-indexed arguments of the log.
  - `topics`: `Array of Strings` - Array of 0 to 4 32 Bytes `DATA` of indexed log arguments. (In *solidity*: The first topic is the *hash* of the signature of the event (e.g. `Deposit(address,bytes32,uint256)`), except you declared the event with the `anonymous` specifier.)

Note For event filter return values see *Contract Events*

Example

```
var filter = chain3.mc.filter('pending');

filter.watch(function (error, log) {
  console.log(log); //  {"address":"0x0000000000000000000000000000000000000000", "data":
↪"0x0000000000000000000000000000000000000000000000000000000000000000", ...}
});

// get all past logs again.
var myResults = filter.get(function(error, logs){ ... });

...

// stops and uninstalls the filter
filter.stopWatching();
```

---

chain3.mc.contract

```
chain3.mc.contract(abiArray)
```

Creates a contract object for a solidity contract, which can be used to initiate contracts on an address. You can read more about events here.

Parameters

1. `Array` - ABI array with descriptions of functions and events of the contract.

Returns

`Object` - A contract object, which can be initiated as follows:

```
var MyContract = chain3.mc.contract(abiArray);

// instantiate by address
var contractInstance = MyContract.at([address]);

// deploy new contract
var contractInstance = MyContract.new([contructorParam1] [, contructorParam2], {data:
↪'0x12345...', from: myAccount, gas: 1000000});

// Get the data to deploy the contract manually
var contractData = MyContract.new.getData([contructorParam1] [, contructorParam2],
↪{data: '0x12345...'});
// contractData = '0x1234564321345600000000023434234'
```

And then you can either initiate an existing contract on an address, or deploy the contract using the compiled byte code:

```
// Instantiate from an existing address:
var myContractInstance = MyContract.at(myContractAddress);



// Or deploy a new contract:

// Deploy the contract asyncronous:
var myContractReturned = MyContract.new(param1, param2, {
   data: myContractCode,
   gas: 300000,
   from: mySenderAddress}, function(err, myContract){
```

(下页继续)

(续上页)

```
    if(!err) {
       // NOTE: The callback will fire twice!
       // Once the contract has the transactionHash property set and once its deployed␣
↪on an address.

       // e.g. check tx hash on the first call (transaction send)
       if(!myContract.address) {
           console.log(myContract.transactionHash) // The hash of the transaction, which␣
↪deploys the contract

       // check address on the second call (contract deployed)
       } else {
           console.log(myContract.address) // the contract address
       }

       // Note that the returned "myContractReturned" === "myContract",
       // so the returned "myContractReturned" object will also get the address set.
    }
  });

// Deploy contract syncronous: The address will be added as soon as the contract is␣
↪mined.
// Additionally you can watch the transaction by using the "transactionHash" property
var myContractInstance = MyContract.new(param1, param2, {data: myContractCode, gas:␣
↪300000, from: mySenderAddress});
myContractInstance.transactionHash // The hash of the transaction, which created the␣
↪contract
myContractInstance.address // undefined at start, but will be auto-filled later
```

Note When you deploy a new contract, you should check for the next 12 blocks or so if the contract code is still at the address (using *chain3.mc.getCode()*), to make sure a fork didn' t change that.

Example

```
// contract abi
var abi = [{
     name: 'myConstantMethod',
     type: 'function',
     constant: true,
```

(下页继续)

(续上页)

```
    inputs: [{ name: 'a', type: 'string' }],
    outputs: [{name: 'd', type: 'string' }]
}, {
    name: 'myStateChangingMethod',
    type: 'function',
    constant: false,
    inputs: [{ name: 'a', type: 'string' }, { name: 'b', type: 'int' }],
    outputs: []
}, {
    name: 'myEvent',
    type: 'event',
    inputs: [{name: 'a', type: 'int', indexed: true},{name: 'b', type: 'bool', indexed:␣
↪false]
}];

// creation of contract object
var MyContract = chain3.mc.contract(abi);

// initiate contract for an address
var myContractInstance = MyContract.at('0xc4abd0339eb8d57087278718986382264244252f');

// call constant function
var result = myContractInstance.myConstantMethod('myParam');
console.log(result) // '0x25434534534'

// send a transaction to a function
myContractInstance.myStateChangingMethod('someParam1', 23, {value: 200, gas: 2000});

// short hand style
chain3.mc.contract(abi).at(address).myAwesomeMethod(...);

// create filter
var filter = myContractInstance.myEvent({a: 5}, function (error, result) {
  if (!error)
    console.log(result);
    /*
    {
        address: '0x871898638226424425 2fc4abd0339eb8d5708727',
        topics: "0x1234567890123456789012345678901 2",
↪"0x0000000000000000000000000000000000000000000000000000000000000005",
```

(下页继续)

(续上页)

```
        data: "0x0000000000000000000000000000000000000000000000000000000000000001",
        ...
    }
    */
});
```

Contract Methods

```
// Automatically determines the use of call or sendTransaction based on the method type
myContractInstance.myMethod(param1 [, param2, ...] [, transactionObject] [, callback]);

// Explicitly calling this method
myContractInstance.myMethod.call(param1 [, param2, ...] [, transactionObject] [,␣
↪callback]);

// Explicitly sending a transaction to this method
myContractInstance.myMethod.sendTransaction(param1 [, param2, ...] [, transactionObject]␣
↪[, callback]);

// Explicitly sending a transaction to this method
myContractInstance.myMethod.sendTransaction(param1 [, param2, ...] [, transactionObject]␣
↪[, callback]);

// Get the call data, so you can call the contract through some other means
var myCallData = myContractInstance.myMethod.getData(param1 [, param2, ...]);
// myCallData = '0x45ff3ff6000000000004545345345345..'
```

The contract object exposes the contracts methods, which can be called using parameters and a transaction object.

Parameters

- `String|Number` - (optional) Zero or more parameters of the function.
- `Object` - (optional) The (previous) last parameter can be a transaction object, see *chain3.mc.sendTransaction* parameter 1 for more.
- `Function` - (optional) If you pass a callback as the last parameter the HTTP request is made asynchronous. See *this note* for details.

Returns

`String` - If its a call the result data, if its a send transaction a created contract address, or the transaction hash, see *chain3.mc.sendTransaction* for details.

Example

```
// creation of contract object
var MyContract = chain3.mc.contract(abi);

// initiate contract for an address
var myContractInstance = MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

var result = myContractInstance.myConstantMethod('myParam');
console.log(result) // '0x25434534534'

myContractInstance.myStateChangingMethod('someParam1', 23, {value: 200, gas: 2000},␣
↪function(err, result){ ... });
```

Contract Events

```
var event = myContractInstance.MyEvent({valueA: 23} [, additionalFilterObject])

// watch for changes
event.watch(function(error, result){
  if (!error)
    console.log(result);
});

// Or pass a callback to start watching immediately
var event = myContractInstance.MyEvent([{valueA: 23}] [, additionalFilterObject] ,␣
↪function(error, result){
  if (!error)
    console.log(result);
});
```

You can use events like *filters* and they have the same methods, but you pass different objects to create the event filter.

Parameters

1. `Object` - Indexed return values you want to filter the logs by, e.g. `{'valueA': 1, 'valueB': [myFirstAddress, mySecondAddress]}`. By default all filter values are set to `null`. It means, that they will match any event of given type sent from this contract.
2. `Object` - Additional filter options, see *filters* parameter 1 for more. By default filterObject has field 'address' set to address of the contract. Also first topic is the signature of event.
3. `Function` - (optional) If you pass a callback as the last parameter it will immediately start watching and you don't need to call `myEvent.watch(function(){})`. See *this note* for details.

Callback return

`Object` - An event object as follows:

- `args`: `Object` - The arguments coming from the event.
- `event`: `String` - The event name.
- `logIndex`: `Number` - integer of the log index position in the block.
- `transactionIndex`: `Number` - integer of the transactions index position log was created from.
- `transactionHash`: `String`, 32 Bytes - hash of the transactions this log was created from.
- `address`: `String`, 32 Bytes - address from which this log originated.
- `blockHash`: `String`, 32 Bytes - hash of the block where this log was in. `null` when its pending.
- `blockNumber`: `Number` - the block number where this log was in. `null` when its pending.

Example

```
var MyContract = chain3.mc.contract(abi);
var myContractInstance = MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

// watch for an event with {some: 'args'}
var myEvent = myContractInstance.MyEvent({some: 'args'}, {fromBlock: 0, toBlock: 'latest
↪'});
myEvent.watch(function(error, result){
    ...
});

// would get all past logs again.
var myResults = myEvent.get(function(error, logs){ ... });
```

(下页继续)

(续上页)

```
...

// would stop and uninstall the filter
myEvent.stopWatching();
```

Contract allEvents

```
var events = myContractInstance.allEvents([additionalFilterObject]);

// watch for changes
events.watch(function(error, event){
  if (!error)
    console.log(event);
});

// Or pass a callback to start watching immediately
var events = myContractInstance.allEvents([additionalFilterObject,] function(error, log){
  if (!error)
    console.log(log);
});
```

Will call the callback for all events which are created by this contract.

Parameters

1. `Object` - Additional filter options, see *filters* parameter 1 for more. By default filterObject has field 'address' set to address of the contract. Also first topic is the signature of event.
2. `Function` - (optional) If you pass a callback as the last parameter it will immediately start watching and you don't need to call `myEvent.watch(function(){})`. See *this note* for details.

Callback return

`Object` - See *Contract Events* for more.

Example

```
var MyContract = chain3.mc.contract(abi);
var myContractInstance = MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

// watch for an event with {some: 'args'}
var events = myContractInstance.allEvents({fromBlock: 0, toBlock: 'latest'});
events.watch(function(error, result){
    ...
});

// would get all past logs again.
events.get(function(error, logs){ ... });

...

// would stop and uninstall the filter
myEvent.stopWatching();
```

---

chain3.encodeParams

```
chain3.encodeParams
```

Encode a list of parameters array into HEX codes. ##### Parameters

- `types` - list of types of params
- `params` - list of values of params

Example

```
var types = ['int','string'];
var args = [100, '4000'];

var dataHex = '0x' + chain3.encodeParams(types, args);
console.log("encoded params:", dataHex);

// outputs
encoded␣
↪params:0x0000000000000000000000000000000000000000000000000000000000000064000000000000000000000000000000000
```

---

chain3.mc.namereg

```
chain3.mc.namereg
```

Returns GlobalRegistrar object.

Usage

see namereg example

---

chain3.mc.sendIBANTransaction

```
var txHash = chain3.mc.sendIBANTransaction('0x00c5496aee77c1ba1f0854206a26dda82a81d6d8',
↪'XE66MOACXREGGAVOFYORK', 0x100);
```

Sends IBAN transaction from user account to destination IBAN address.

Parameters

- `string` - address from which we want to send transaction
- `string` - IBAN address to which we want to send transaction
- `value` - value that we want to send in IBAN transaction

---

chain3.mc.iban

```
var i = new chain3.mc.iban("XE81ETHXREGGAVOFYORK");
```

---

chain3.mc.iban.fromAddress

```
var i = chain3.mc.iban.fromAddress('0xd814f2ac2c4ca49b33066582e4e97ebae02f2ab9');
console.log(i.toString()); // 'XE72P8O19KRSWXUGDY294PZ66T4ZGF89INT
```

---

chain3.mc.iban.fromBban

```
var i = chain3.mc.iban.fromBban('XE66MOACXREGGAVOFYORK');
console.log(i.toString()); // "XE71XE66MOACXREGGAVOFYORK"
```

chain3.mc.iban.createIndirect

```
var i = chain3.mc.iban.createIndirect({
  institution: "XREG",
  identifier: "GAVOFYORK"
});
console.log(i.toString()); // "XE66MOACXREGGAVOFYORK"
```

chain3.mc.iban.isValid

```
var valid = chain3.mc.iban.isValid("XE66MOACXREGGAVOFYORK");
console.log(valid); // true

var valid2 = chain3.mc.iban.isValid("XE76MOACXREGGAVOFYORK");
console.log(valid2); // false, cause checksum is incorrect

var i = new chain3.mc.iban("XE66MOACXREGGAVOFYORK");
var valid3 = i.isValid();
console.log(valid3); // true
```

chain3.mc.iban.isDirect

```
var i = new chain3.mc.iban("XE66MOACXREGGAVOFYORK");
var direct = i.isDirect();
console.log(direct); // false
```

chain3.mc.iban.isIndirect

```
var i = new chain3.mc.iban("XE66MOACXREGGAVOFYORK");
var indirect = i.isIndirect();
console.log(indirect); // true
```

chain3.mc.iban.checksum

```
var i = new chain3.mc.iban("XE66MOACXREGGAVOFYORK");
var checksum = i.checksum();
console.log(checksum); // "66"
```

chain3.mc.iban.institution

```
var i = new chain3.mc.iban("XE66MOACXREGGAVOFYORK");
var institution = i.institution();
console.log(institution); // 'XREG'
```

chain3.mc.iban.client

```
var i = new chain3.mc.iban("XE66MOACXREGGAVOFYORK");
var client = i.client();
console.log(client); // 'GAVOFYORK'
```

chain3.mc.iban.address

```
var i = new chain3.mc.iban('XE72P8O19KRSWXUGDY294PZ66T4ZGF89INT');
var address = i.address();
console.log(address); // 'd814f2ac2c4ca49b33066582e4e97ebae02f2ab9'
```

chain3.mc.iban.toString

```
var i = new chain3.mc.iban('XE72P8O19KRSWXUGDY294PZ66T4ZGF89INT');
console.log(i.toString()); // 'XE72P8O19KRSWXUGDY294PZ66T4ZGF89INT'
```

2.6 Chain3 GO 软件库

MOAC Go API was built for MOAC chain. It was developed from MOAC RPC API, which can be used to develop Đapp on MOAC chain. It supports both VNODE and SCS JSON RPC API methods in MOAC network.

2.6.1 Chain3Go Installation

setup $GOPATH

“ export GOPATH=/Users/[user]/go “

go get

“bash go get -u github.com/MOACChain/Chain3Go “

2.6.2 MOAC Configuration

Install MOAC

Download latest MOAC Vnode and SCS Releases from here: https://github.com/MOACChain/moac-core/releases

Run MOAC

Run moac vnode on testnet

./moac –testnet

Run moac scs to connect the VNODE locally

./scsserver

Create new accounts and send transactions

```
mc.coinbase
mc.accounts
personal.newAccount()
passphrase:
repeat passphrase:

miner.start()
--wait a few seconds
miner.stop()

personal.unlockAccount("0x18833df6ba69b4d50acc744e8294d128ed8db1f1")
mc.sendTransaction({from: '0x18833df6ba69b4d50acc744e8294d128ed8db1f1', to:
↪'0x2a022eb956d1962d867dcebd8fed6ae71ee4385a', value: chain3.toSha(12, "moac")})
```

Chain3Go Execution

go run main.go

2.6.3 Requirements

go ^1.8.3

[Go installation instructions.](https://golang.org/doc/install)

2.7 Chain3 Java 软件库

chain3j <https://github.com/MOACChain/chain3j> is a lightweight, highly modular, reactive, type safe Java and Android library for working with Smart Contracts and integrating with clients (nodes) on the MOAC network:

This allows you to work with the MOAC blockchain, without the additional overhead of having to write your own integration code for the platform.

2.7.1 特点

- Complete implementation of MOAC' s JSON-RPC client API over HTTP and IPC
- Auto-generation of Java smart contract wrappers to create, deploy, transact with and call smart contracts from native Java code (Solidity and
- Reactive-functional API for working with filters
- Support for MOAC gateway, so you don' t have to run an MOAC client yourself
- Comprehensive integration tests demonstrating a number of the above scenarios
- Command line tools
- Android compatible

It has five runtime dependencies:

- RxJava for its reactive-functional API
- OKHttp for HTTP connections
- Jackson Core for fast JSON serialisation/deserialisation
- Bouncy Castle (Spongy Castle on Android) for crypto
- Jnr-unixsocket for *nix IPC (not available on Android)

It also uses JavaPoet for generating smart contract wrappers.

2.7.2 Commercial support and training

Commercial support and training is available from moac.io.

2.7.3 Quickstart

A chain3j sample project is available that demonstrates a number of core features of MOAC with chain3j, including:

- Connecting to a node on the MOAC network

- Loading an MOAC keystore file
- Sending MOAC from one address to another
- Deploying a smart contract to the network
- Reading a value from the deployed smart contract
- Updating a value in the deployed smart contract
- Viewing an event logged by the smart contract

2.7.4 Getting started

Typically your application should depend on release versions of chain3j, but you may also use snapshot dependencies for early access to features and fixes, refer to the **'Snapshot Dependencies'_** section.

Add the relevant dependency to your project:

2.7.5 Maven

Java 8:

```
<dependency>
  <groupId>io.github.moacchain</groupId>
  <artifactId>chain3j</artifactId>
  <version>0.1.0</version>
</dependency>
```

Android:

```
<dependency>
  <groupId>io.github.moacchain</groupId>
  <artifactId>chain3j</artifactId>
  <version>0.1.0-android</version>
</dependency>
```

2.7.6 Gradle

Java 8:

```
compile ('io.github.moacchain:chain3j:0.1.0')
```

Android:

```
compile ('io.github.moacchain:chain3j:0.1.0-android')
```

2.7.7 Start a client

Start up an MOAC client if you don’ t already have one running, check _:

```
$ ./moac --rpcapi "personal,mc,net,chain3" --rpc --testnet
```

```
Chain3j chain3 = Chain3j.build(new HttpService("http://gateway.moac.io/testnet"));
```

2.7.8 Start sending requests

To send synchronous requests:

```
Chain3j chain3 = Chain3j.build(new HttpService());  // defaults to http://localhost:8545/
Chain3ClientVersion chain3ClientVersion = chain3.chain3ClientVersion().send();
String clientVersion = chain3ClientVersion.getChain3ClientVersion();
```

To send asynchronous requests using a CompletableFuture (Future on Android):

```
Chain3j chain3 = Chain3j.build(new HttpService());  // defaults to http://localhost:8545/
Chain3ClientVersion chain3ClientVersion = chain3.chain3ClientVersion().sendAsync().get();
String clientVersion = chain3ClientVersion.getChain3ClientVersion();
```

To use an RxJava Observable:

```
Chain3j chain3 = Chain3j.build(new HttpService());  // defaults to http://localhost:8545/
chain3.chain3ClientVersion().observable().subscribe(x -> {
    String clientVersion = x.getChain3ClientVersion();
    ...
});
```

Note: for Android use:

```
Chain3j chain3 = Chain3jFactory.build(new HttpService());  // defaults to http://
↪localhost:8545/
...
```

2.7.9 IPC

chain3j also supports fast inter-process communication (IPC) via file sockets to clients running on the same host as chain3j. To connect simply use the relevant *IpcService* implementation instead of *HttpService* when you create your service:

```
// OS X/Linux/Unix:
Chain3j chain3 = Chain3j.build(new UnixIpcService("/path/to/socketfile"));
...

// Windows
Chain3j chain3 = Chain3j.build(new WindowsIpcService("/path/to/namedpipefile"));
...
```

Note: IPC is not currently available on chain3j-android.

2.7.10 Working with smart contracts with Java smart contract wrappers

chain3j can auto-generate smart contract wrapper code to deploy and interact with smart contracts without leaving the JVM.

To generate the wrapper code, compile your smart contract:

```
$ solc <contract>.sol --bin --abi --optimize -o <output-dir>/
```

Then generate the wrapper code using chain3j's *Command line tools*:

```
chain3j solidity generate /path/to/<smart-contract>.bin /path/to/<smart-contract>.abi -o␣
↪/path/to/src/main/java -p com.your.organisation.name
```

Now you can create and deploy your smart contract:

```
Chain3j chain3 = Chain3j.build(new HttpService());  // defaults to http://localhost:8545/
Credentials credentials = WalletUtils.loadCredentials("password", "/path/to/walletfile");

YourSmartContract contract = YourSmartContract.deploy(
        <chain3j>, <credentials>,
        GAS_PRICE, GAS_LIMIT,
        <param1>, ..., <paramN>).send();  // constructor params
```

Alternatively, if you use MOAC wallet, you can make use of its *.json* output files:

```
# Open MOAC wallet and start a local MOAC node
# Click CONTRACTS tab and choose the "DEPLOY NEW CONTRACT" button
# Copy the contract codes to the "SOLIDITY CONTRACT SOURCE CODE"
# The codes will be auto compiled.
```

Then generate the wrapper code using chain3j' s *Command line tools*:

```
$ cd /path/to/your/chain3j/java/project
$ chain3j truffle generate /path/to/<truffle-smart-contract-output>.json -o /path/to/src/
↪main/java -p com.your.organisation.name
```

Whether using *Truffle* or *solc* directly, either way you get a ready-to-use Java wrapper for your contract.

So, to use an existing contract:

```
YourSmartContract contract = YourSmartContract.load(
        "0x<address>|<ensName>", <chain3j>, <credentials>, GAS_PRICE, GAS_LIMIT);
```

To transact with a smart contract:

```
TransactionReceipt transactionReceipt = contract.someMethod(
            <param1>,
            ...).send();
```

To call a smart contract:

```
Type result = contract.someMethod(<param1>, ...).send();
```

To fine control your gas price:

```
contract.setGasProvider(new DefaultGasProvider() {
        ...
        });
```

For more information refer to Smart Contracts.

2.7.11 Filters

chain3j functional-reactive nature makes it really simple to setup observers that notify subscribers of events taking place on the blockchain.

To receive all new blocks as they are added to the blockchain:

```
Subscription subscription = chain3j.blockObservable(false).subscribe(block -> {
    ...
});
```

To receive all new transactions as they are added to the blockchain:

```
Subscription subscription = chain3j.transactionObservable().subscribe(tx -> {
    ...
});
```

To receive all pending transactions as they are submitted to the network (i.e. before they have been grouped into a block together):

```
Subscription subscription = chain3j.pendingTransactionObservable().subscribe(tx -> {
    ...
});
```

Or, if you' d rather replay all blocks to the most current, and be notified of new subsequent blocks being created:

There are a number of other transaction and block replay Observables described in the docs.

Topic filters are also supported:

```
McFilter filter = new McFilter(DefaultBlockParameterName.EARLIEST,
        DefaultBlockParameterName.LATEST, <contract-address>)
             .addSingleTopic(...)|.addOptionalTopics(..., ...)|...;
chain3j.mcLogObservable(filter).subscribe(log -> {
    ...
});
```

Subscriptions should always be cancelled when no longer required:

```
subscription.unsubscribe();
```

Note: filters are not supported on Infura.

For further information refer to Filters and Events and the Chain3jRx interface.

2.7.12 Transactions

chain3j provides support for both working with MOAC wallet files (recommended) and MOAC client admin commands for sending transactions.

To send Mc to another party using your MOAC wallet file:

```
Chain3j chain3 = Chain3j.build(new HttpService());  // defaults to http://localhost:8545/
Credentials credentials = WalletUtils.loadCredentials("password", "/path/to/walletfile");
TransactionReceipt transactionReceipt = Transfer.sendFunds(
        chain3, credentials, "0x<address>|<ensName>",
        BigDecimal.valueOf(1.0), Convert.Unit.MC)
        .send();
```

Or if you wish to create your own custom transaction:

```
Chain3j chain3 = Chain3j.build(new HttpService());  // defaults to http://localhost:8545/
Credentials credentials = WalletUtils.loadCredentials("password", "/path/to/walletfile");

// get the next available nonce
McGetTransactionCount mcGetTransactionCount = chain3j.mcGetTransactionCount(
             address, DefaultBlockParameterName.LATEST).sendAsync().get();
BigInteger nonce = mcGetTransactionCount.getTransactionCount();

// create our transaction
RawTransaction rawTransaction  = RawTransaction.createMcTransaction(
             nonce, <gas price>, <gas limit>, <toAddress>, <value>);

// sign & send out transaction with EIP155 signature
byte[] signedMessage = TransactionEncoder.signTxEIP155(rawTransaction, <chainId>,␣
↪credentials);
String hexValue = Hex.toHexString(signedMessage);
McSendTransaction mcSendTransaction = chain3j.SendRawTransaction(hexValue).send();
// ...
```

Although it's far simpler using chain3j's Transfer for transacting with Mc.

Using an MOAC client's admin commands (make sure you have your wallet in the client's keystore):

```
Admin chain3j = Admin.build(new HttpService());  // defaults to http://localhost:8545/
PersonalUnlockAccount personalUnlockAccount = chain3j.personalUnlockAccount("0x000...",
↪"a password").sendAsync().get();
if (personalUnlockAccount.accountUnlocked()) {
    // send a transaction
}
```

2.7.13 Command line tools

A chain3j fat jar is distributed with each release providing command line tools. The command line tools allow you to use some of the functionality of chain3j from the command line:

- Wallet creation
- Wallet password management
- Transfer of funds from one wallet to another
- Generate Solidity smart contract function wrappers

2.7.14 Further details

In the Java 8 build:

- chain3j provides type safe access to all responses. Optional or null responses are wrapped in Java 8' s Optional type.
- Asynchronous requests are wrapped in a Java 8 CompletableFutures. chain3j provides a wrapper around all async requests to ensure that any exceptions during execution will be captured rather then silently discarded. This is due to the lack of support in *CompletableFutures* for checked exceptions, which are often rethrown as unchecked exception causing problems with detection. See the Async.run() and its associated test for details.

In both the Java 8 and Android builds:

- Quantity payload types are returned as BigIntegers. For simple results, you can obtain the quantity as a String via Response.getResult().
- It' s also possible to include the raw JSON payload in responses via the *includeRawResponse* parameter, present in the HttpService and IpcService classes.

2.7.15 Build instructions

chain3j includes integration tests for running against a live MOAC client. If you do not have a client running, you can exclude their execution as per the below instructions.

To see the compile options:

```
$ ./gradlew tasks
```

To run a full build (excluding integration tests):

```
$ ./gradlew check
```

Sample maven configuration:

```
<repositories>
  <repository>
    <id>sonatype-snasphots</id>
    <name>Sonatype snapshots repo</name>
    <url>https://oss.sonatype.org/content/repositories/snapshots</url>
  </repository>
</repositories>
```

2.7.16 Thanks and credits

- The Web3j project for the framework
- The Nethereum project for the inspiration
- Othera for the great things they are building on the platform
- Finhaus guys for putting me onto Nethereum
- bitcoinj for the reference Elliptic Curve crypto implementation
- Everyone involved in the Etherereum project and its surrounding ecosystem
- And of course the users of the library, who' ve provided valuable input & feedback

2.8 Chain3 Python 软件库

2.8.1 安装

Chain3 python package supports python3.5.3 and up.

Chain3 python can be installed (preferably in a virtualenv) using pip as follows:

```
$ pip install chain3
```

If you run into problems during installation, you might have a broken environment, such as you are on an unsupported version of Python or another package might be installed that has a name or version conflict Often, the best way to guarantee a correct environment is with virtualenv, like:

```
$ which pip || curl https://bootstrap.pypa.io/get-pip.py | python

$ which virtualenv || pip install --upgrade virtualenv

$ sudo pip install virtualenv
```

(下页继续)

(续上页)

```
$ virtualenv -p python3 ~/.venv-py3

$ source ~/.venv-py3/bin/activate

$ pip install --upgrade pip setuptools

$ pip install --upgrade chain3
```

To use the virtualenv next time, run the following command:

```
$ source ~/.venv-py3/bin/activate
```

2.8.2 Using Chain3

```
from chain3 import Chain3
chain3 = Chain3(Chain3.HTTPProvider("http://127.0.0.1:8545", request_kwargs={'timeout':␣
↪30}))
```

2.8.3 Chain3 API

1. chain3.version.network:

Returns the current network id. For MOAC, mainnet network id = 99, testnet network id = 101.

```
print('network id:'+ str(chain3.version.network))
```

2. chain3.HTTPProvider:

Convenience API to access chain3.providers.rpc.HTTPProvider

```
Chain3(Chain3.HTTPProvider("http://127.0.0.1:8545", request_kwargs={'timeout': 30}))
```

3. chain3.sha3(string,options)

string: The string to hash using the SHA3 algorithm. options: optional, need set to hex for HEX string.

```
hash = chain3.sha3("the string to be hashed");
print(hash);
```

(下页继续)

(续上页)

```
hashOfHash = chain3.sha3(hash,{encoding:'hex'});
print(hashOfHash);
```

4. chain3.net.peerCount:

This property is read only and returns the number of connected peers.

```
peerCount = chain3.net.peerCount;
print(peerCount);
```

5. chain3.net.listening:

This property is read only and says whether the node is actively listening for network connections or not.

```
listenState = chain3.net.listening;
print(listenState);
```

6. chain3.mc.coinbase:

Returns the current Coinbase address.

```
nodeCoinbase = chain3.mc.coinbase;
print(nodeCoinbase);
```

7. chain3.mc.mining:

Returns boolean as to whether the node is currently mining.

```
miningState = chain3.mc.mining;
print(miningState);  //true or false
```

8. chain3.mc.accounts:

Returns the list of known accounts.

```
nodeAccounts = chain3.mc.accounts;
print(nodeAccounts);
```

9. chain3.mc.blockNumber:

Returns the number of the most recent block

```
nowBlockNumber = chain3.mc.blockNumber;
print(nowBlockNumber);
```

10. chain3.mc.getBlockTransactionCount(block_identifier):

Returns the number of transactions in the block specified by block_identifier.

Delegates to mc_getBlockTransactionCountByNumber if block_identifier is an integer or one of the predefined block parameters 'latest', 'earliest', 'pending', otherwise delegates to mc_getBlockTransactionCountByHash.

```
transactionCount = chain3.mc.getBlockTransactionCount(96160);
print(transactionCount);
```

11. chain3.mc.getBalance(account, block_identifier=mc.defaultBlock):

Returns the balance of the given account at the block specified by block_identifier.

account may be a hex address or an ENS name

```
balance = chain3.mc.getBalance("0x36eaa71d7383be53cb600743aad08a55222a4915", block_
↪identifier=chain3.mc.defaultBlock);
print("getBalance1" + balance); //instanceof BigNumber
print("getBalance2" + balance.toString(10));
//Result: getBalance1:3.04527226722e+21
//         getBalance2:3045272267220000000000
```

12. chain3.mc.defaultBlock:

The default block number that will be used for any RPC methods that accept a block identifier. Defaults to 'latest'.

```
defultBlock = chain3.mc.defaultBlock;
print("defaultBlock" + defultBlock);
//default is latest,
chain3.mc.defaultBlock = 123;
print("defaultBlock" + defultBlock);
```

13. chain3.mc.gasPrice:

Returns the current gas price in Sha = 1e-18 mc. GasPrice is calculated from most recent blocks.

```
gasPrice = chain3.mc.gasPrice;
print(gasPrice.toString(10));
```

14. chain3.mc.estimateGas(transaction_params=None):

Uses the selected gas price strategy to calculate a gas price. This method returns the gas price denominated in sha. The transaction_params argument is optional however some gas price strategies may require it to be able to produce a gas price.

```
result = chain3.mc.estimateGas({
 to :"0xf7ebc6b854a202efe08e91422a44ba2161ed50dc",
 data: '0x23455654'
    //gas: 11,          //Optional, gaslimit of the TX
    //gasPrice: 11      //Optional, gasPrice
});
print('estimateGas  :'+ result);
//Output: gasprice :20000000000
//        estimateGas :1273
```

15. chain3.mc.getCode(account, block_identifier=mc.defaultBlock):

Returns the bytecode for the given account at the block specified by block_identifier. account may be a hex address or an ENS name

```
code  = chain3.mc.getCode("0x0000000000000000000000000000000000000065");//contract␣
↪address
```

16. chain3.mc.syncing:

Returns either False if the node is not syncing or a dictionary showing sync status.

```
sync = chain3.mc.syncing;
print('syncing  :'+ sync );
//
AttributeDict({
    'currentBlock': 2177557,
```

(下页继续)

(续上页)

```
    'highestBlock': 2211611,
    'knownStates': 0,
    'pulledStates': 0,
    'startingBlock': 2177365,
})
```

17. chain3.mc.getTransaction(transaction_hash):

```
blockHash = "0x6aa4a0db1fc155009bd9ba3a64c1aef109e1418dc05ee241d3e9e3e58d7f3eeb";
transaction = chain3.mc.getTransaction(blockHash);
print('get transaction:'+ str(transaction));

/* Result:
get transaction: AttributeDict({
   'blockHash': HexBytes(
↪'0x77483002572dd29b58640c4ccf5ef30278679037ff17b51cf613f3df562e5e0a'),
   'blockNumber': 815006,
   'from': '0x0000000000000000000000000000000000000064',
   'gas': 0,
   'gasPrice': 20000000000,
   'hash': HexBytes('0x6aa4a0db1fc155009bd9ba3a64c1aef109e1418dc05ee241d3e9e3e58d7f3eeb
↪'),
   'input': '0xc1c0e9c4',
   'nonce': 815005,
   'syscnt': '0x65',
   'to': '0x0000000000000000000000000000000000000065',
   'transactionIndex': 0,
   'value': 0,
   'v': 0, 'r': HexBytes('0x00'), 's': HexBytes('0x00'),
   'shardingFlag': 0})
*/
```

18. chain3.mc.getBlock(block_identifier=mc.defaultBlock, full_transactions=False):

Returns the block specified by block_identifier. Delegates to mc_getBlockByNumber if block_identifier is an integer or one of the predefined block parameters ‘latest’, ‘earliest’, ‘pending’, otherwise delegates to mc_getBlockByHash.

If full_transactions is True then the ‘transactions’ key will contain full transactions objects. Otherwise it will be an array of transaction hashes.

```
getTheBlock = chain3.mc.getBlock(815006);
print('get the block: '+ str(getTheBlock));

/* Result:
get the block({
   'difficulty': 86803583,
   'extraData': HexBytes('0xdd854d4f41432d85312e302e312d87676f312e392e358777696e646f7773
↪'),
   'gasLimit': 9000000,
   'gasUsed': 0,
   'hash': HexBytes('0x77483002572dd29b58640c4ccf5ef30278679037ff17b51cf613f3df562e5e0a
↪'),
   'logsBloom': HexBytes(
↪'0x00000000000000000000000000000000000000000000000000000000000000000000000000000000
          ␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000
          ␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000
          ␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000
          ␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000
           00000000000000000'),
   'miner': '0x0a2168D2f08161c01745fEC4e6E8FE06F314Ab41',
   'mixHash': HexBytes(
↪'0xc154897a85ca63bbbbb76b618a288f6b33f7d2994848dc9c43c6d65e6a5da355'),
   'nonce': HexBytes('0x829f5b23cdf8224f'),
   'number': 815006,
   'parentHash': HexBytes(
↪'0x73c0e4a94b48b41bf5a6a22151e38799a0e17e8b798848af5340f6d725027af1'),
   'receiptsRoot': HexBytes(
↪'0x9287370eb27f11b0c2188431cbc58a23b685f02dbd851ed4d974f932bd780839'),
   'sha3Uncles': HexBytes(
↪'0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347'),
   'size': 590,
   'stateRoot': HexBytes(
↪'0x615d0a39783ae546e11aa0cd6e00c70c2ec989f51316c0f9e07cfc99f1088669'),
   'timestamp': 1535530608,
```

(下页继续)

(续上页)

```
  'totalDifficulty': 136959813601540,
  'transactions': [HexBytes(
↪'0x6aa4a0db1fc155009bd9ba3a64c1aef109e1418dc05ee241d3e9e3e58d7f3eeb')],
  'transactionsRoot': HexBytes(
↪'0x7aba2a9c974693f1cfb96d506e6aa62942a174b4df39c831cf844a35e03249f0'),
  'uncles': []
})
*/
```

19. chain3.personal.unlockAccount(account, passphrase, duration=None):

Unlocks the given account for duration seconds. If duration is None then the account will remain unlocked indefinitely. Returns boolean as to whether the account was successfully unlocked.

```
chain3.personal.unlockAccount(mc.accounts[0], 'password')
```

20. chain3.miner.start(num_threads):

Start the CPU mining process using the given number of threads.

```
chain3.miner.start(2) # number of threads
```

21. chain3.miner.stop:

Stop the CPU mining operation

```
chain3.miner.stop()
```

22. chain3.miner.setGasPrice(gas_price):

Sets the minimum accepted gas price that this node will accept when mining transactions. Any transactions with a gas price below this value will be ignored.

```
chain3.miner.setGasPrice(19999999999)
```

23. chain3.mc.getTransactionReceipt((transaction_hash, timeout=120):

Returns the transaction receipt specified by transaction_hash. If the transaction has not yet been mined returns None

```
txr = chain3.mc.getTransactionReceipt(
↪'0x77483002572dd29b58640c4ccf5ef30278679037ff17b51cf613f3df562e5e0a')
print(txr)

/* Result:
AttributeDict({
   'blockHash': HexBytes(
↪'0x77483002572dd29b58640c4ccf5ef30278679037ff17b51cf613f3df562e5e0a'),
   'blockNumber': 815006,
   'contractAddress': '0x0000000000000000000000000000000000000065',
   'cumulativeGasUsed': 0,
   'from': '0x0000000000000000000000000000000000000064',
   'gasUsed': 0,
   'logs': [],
   'logsBloom': HexBytes(
↪'0x00000000000000000000000000000000000000000000000000000000000000000000000000
      ␣
↪000000000000000000000000000000000000000000000000000000000000000000000000000000000000
      ␣
↪000000000000000000000000000000000000000000000000000000000000000000000000000000000000
      ␣
↪000000000000000000000000000000000000000000000000000000000000000000000000000000000000
      ␣
↪000000000000000000000000000000000000000000000000000000000000000000000000000000000000
      ␣
↪000000000000000000000000000000000000000000000000000000000000000000000000000000000000
      000000000000000000'),
   'status': 1,
   'to': '0x0000000000000000000000000000000000000065',
   'transactionHash': HexBytes(
↪'0x6aa4a0db1fc155009bd9ba3a64c1aef109e1418dc05ee241d3e9e3e58d7f3eeb'),
   'transactionIndex': 0})
*/
```

24. chain3.mc.getTransactionCount((block_identifier):

Returns the number of transactions that have been sent from account as of the block specified by block_identifier.

```
chain3.mc.getTransactionCount('0x87E369172Af1e817ebD8d63bcD9f685A513a6736', block_
↪identifier=chain3.mc.defaultBlock)
```

(下页继续)

(续上页)

25. chain3.mc.sendTransaction(transaction, passphrase):

Signs and sends the given transaction

The transaction parameter should be a dictionary with the following fields.

- from: bytes or text, hex address or ENS name - (optional, default: chain3.mc.defaultAccount) The address the transaction is send from.
- to: bytes or text, hex address or ENS name - (optional when creating new contract) The address the transaction is directed to.
- gas: integer - (optional) Integer of the gas provided for the transaction execution. It will return unused gas.
- gasPrice: integer - (optional, default: To-Be-Determined) Integer of the gasPrice used for each paid gas
- value: integer - (optional) Integer of the value send with this transaction
- data: bytes or text - The compiled code of a contract OR the hash of the invoked method signature and encoded parameters.
- nonce: integer - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

If the transaction specifies a data value but does not specify gas then the gas value will be populated using the estimateGas() function with an additional buffer of 100000 gas up to the gasLimit of the latest block. In the event that the value returned by estimateGas() method is greater than the gasLimit a ValueError will be raised.

- shardingFlag:integer - (optional for Global Transactions), MicroChain flag, default value is 0 for Global TXs. To call MicroChain, this value has to be 1.
- via: bytes or text, hex addres - (optional for Global Transactions), vode beneficial address, default is null for Global TXs. For microChain call

```
chain3.mc.sendTransaction({
    'to':'0xf103BC1c054baBcecD13e7AC1CF34F029647B08C',
    'from':'0x87E369172Af1e817ebD8d63bcD9f685A513a6736',
    'value': 100000,
    'gasPrice': chain3.mc.gasPrice,
    'shardingFlag': 0,
    'via': '0x0000000000000000000000000000000000000000',})
```

26. chain3.mc.sendRawTransaction(raw_transaction):

Sends a signed and serialized transaction. Returns the transaction hash.

```
private_key = '0x94645c7a048771045f90e0b88adf3ddf5afbb5029c2b1b5586d5afa9ba87c8f5'
signed_txn = chain3.mc.account.signTransaction(
    dict(
        nonce=chain3.mc.getTransactionCount(chain3.mc.coinbase),
        gasPrice=chain3.mc.gasPrice,
        gas=100000,
        to='0xf103BC1c054baBcecD13e7AC1CF34F029647B08C',
        value=100000,
        data='0x',
        'chainId': networkid,
        'shardingFlag': 0,
        'via': '0x',
    ),
    private_key,
)
chain3.mc.sendRawTransaction(signed_txn.rawTransaction)

/* Result: tx hash
    '0xd7e3a30f9eec70d5626b70a2082bd2573a2b0a282756479c2f48a57a833204ab'
*/
```

CHAPTER 3

子链

3.1 名词解释

3.1.1 Vnode

墨客主网（母链）节点，用于主网挖矿，主网账本同步，主网交易以及子链数据传输的节点。

3.1.2 Scs

墨客子链节点，用于子链挖矿，子链账本同步以及子链业务逻辑执行的节点，也称为子链矿工

3.1.3 子链矿工池

存储子链矿工的池子，本质上池子是一个智能合约，需要子链节点注册

3.1.4 vnode 代理池

存储 vnode 代理节点，本质上池子是一个智能合约，需要 vnode 节点

3.1.5 子链控制合约

用于控制整个子链的流程

3.1.6 监听节点（Monitor）

监听节点是一个特殊的 Scs 节点，可以用来监听某条子链的运行情况，当一个节点成为监听节点后，其只负责同步该子链的区块信息，不参与子链出块。Dapp 用户可以通过该节点监控子链运行情况

3.1.7 flush

子链的一个特殊操作，每条正常运行的子链每隔一段时间需要向母链进行状态刷新，并且同时完成：Scs 矿工的收益发放；有币子链和母链之间的货币充提等操作。flush 周期可在部署子链控制合约时设置，当子链交易数不活跃时，flush 周期将变大，直到有交易时收敛到设置值

3.1.8 子链多合约基础合约

1.0.8 版本后的新功能，用于在子链上部署多合约的基础控制合约

3.2 子链要点

3.2.1 概述

子链是在母链之上的独立的区块链系统，子链中的每个节点称为 scs，scs 的通讯通过母链，并定期向母链进行数据背书。子链上可以单独跑业务逻辑，母链无需知道。

母链账户地址和子链账户地址是同一体系，可以通用，但在在不同的链上为不同链的货币进行服务，互相之间除非充提否则没有联系。

母链上可以跑多条子链，墨客采用分片技术，随机将 scs 分配给不同的子链。

子链需要依赖于母链来运行，因此，运行一条子链需要 M 个母链节点（vnode）和 N 个子链节点（scs）。

普通子链节点 scs 可以称作子链矿工，其主要负责子链的出块，是维持一条子链稳定安全运行的根本。

子链节点下的母链节点，需要选取一部分注册成 vnode 代理的节点，vnode 代理节点的作用是用于维护子链稳定运行，vnode 代理节点同样有一个代理矿工池

部署子链控制合约时需要指定以上两个池子的地址。

子链合约部署完后，即可调用 registeropen 来召唤池子里的 scs 注册；当数量达到预期后，就可以调用 registerclose 来初始化子链区块，让子链开始运行。

子链需要每隔一段时间发起 flush 操作，将关键状态写入母链进行背书。除此之外，flush 还将完成节点收益分配和有币区块链的母子链充提操作。

可以调用子链控制合约里的 registerasmonitor 来将一个子链节点注册成一个侦听节点。侦听节点只负责信息查询，不参与普通子链节点的服务，适合 dapp 用户部署来监控子链运行状态。

如果普通子链节点状态不稳定或弄虚作假，在 flush 的时候，有几率被没收押金，强制退出这条子链的服务。这时，可以调用子链控制合约的 registeradd 方法，将 scs 池子里的其他 scs 作为备用节点来为这条子链服务。

部署子链方可以调用子链控制合约的 close 方法关闭这条子链。此时会进入清算状态，待所有子链收益结清后，即关闭子链。请注意，子链业务逻辑清算不包含。

子链节点本身可以调用子链控制合约的 withdraw 方法来结束为某一子链服务，并得到返还的押金。

3.2.2 子链中的 MOAC 押金和消耗

为了使子链安全稳定的运行，MOAC 引入的押金机制，主要体现在以下几个方面：

1、每个子链节点在第一次启动后，将会有一个唯一的墨客钱包地址，在部署子链前，需要在向这个地址打入至少 1 个 MOAC 作为运行费用；

2、每个子链节点注册进入子链矿工池时，需要向矿工池缴纳一定的押金，最小值由子链控制合约设置，最大值不限。子链节点每被选中一次，将会扣除一定数额的押金，当押金被扣完后，该节点将不会再参与新的子链，退出子链时，可以调用方法取回押金；

3、当一个子链节点注册成一个监听节点时，需要缴纳一定的押金；当退出子链时可以取回押金；

4、押金一般不会扣除，但在 flush 时，如果有企图作弊的节点，将会按照规则踢出子链，并扣除押金，不再返还；

维护子链的 MOAC 消耗：

首先，调用子链方法不会消耗任何 gas，但是，dapp 运营方需要向子链控制合约地址打入一定量的 MOAC 维持子链运行，这部分 MOAC 将会消耗在给子链矿工费用、子链向母链 flush 状态，以及母链充提 gas 返还上。这个维护消耗可以通过调整子链的 flush 周期来部分改变。

3.2.3 子链部署的注意点

1、子链 scs 的 vnode 需要互相 addPeer 以保证通讯畅通。同时，这些 vnode 建议尽量 add 外面的节点以保证主链高度一致，并且保证时钟同步。

2、子链 scs 的时钟请同步互联网标准时间。

3.3 部署子链前的准备工作

3.3.1 Vnode 节点

墨客主网节点版本来源: https://github.com/MOACChain/moac-core/releases/

此文档采用的版本：1.0.9 环境：windows testnet 浏览器：http://testnet.moac.io/home

在测试环境 testnet 启动节点： moac-windows-4.0-amd64.exe –testnet –rpc –rpcapi "chain3,mc,net,db,personal,admin,miner"

验证：

```
windows command 执行 moac-windows-4.0-amd64.exe attach \\.\pipe\moac.ipc
运行 concole 命令 mc.blockNumber 检查是否同步到最新区块
```

3.3.2 SCS 节点

墨客子链节点版本来源: https://github.com/MOACChain/moac-core/releases/

此文档采用的版本：1.0.9 环境：windows

userconfig.json 配置：

```
VnodeServiceCfg 为代理 vnode 地址: 192.168.10.209:50062  (对应上面部署的 vnode 的 IP)
Beneficiary 为收益账号
```

启动节点： scsserver-windows-4.0-amd64 –password "123456"（生成 scs keystore 的密码）

验证：

```
scs 启动后，并开始从 vnode 同步块号信息。
在 scskeystore 目录内生成的 keystore 文件中生成 scs 账号的 scskeystore 文件。
```

3.3.3 各类账号

可以运行 concole 命令 personal.newAccount() 创建账号；mc.accounts 查看账号；

按序号查询余额：mc.getBalance(mc.accounts[0])

测试环境的公共提币地址：https://faucet.moacchina.com

注意：后续消耗 gas 的操作都需要执行 personal.unlockAccount(mc.accounts[0]) 对应账号进行解锁

准备账号列表：（示例地址参考后续的命令操作）

```
子链操作账号：进行创建合约，发起交易等基本操作： 0x87e369172af1e817ebd8d63bcd9f685a513a6736
主链 vnode 收益账号:      0xf103bc1c054babcecd13e7ac1cf34f029647b08c
子链 scs 收益账号：0xa934198916cd993c73c1aa6e0c0e7b21ce7c735b␣
↪0x2e7c076dbf6e61207a0ddb1b942ef7da8fd139f0
```

3.3.4 chain3 的 nodejs 环境

安装：npm install chain3

验证:

```
> chain3 = require('chain3');
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> chain3.mc.blockNumber  检查是否获得当前区块
```

3.4 子链的部署方法

3.4.1 部署 Vnode 矿池合约

首先部署 vnode 矿池合约，VnodeProtocolBase，如果加入现成的 vnode 矿池，则可以忽略此步骤。

加入矿池的代理 Vnode 节点被用于提供子链调用服务和子链历史数据中转服务的节点。

以下为 nodejs 部署示例：最低保证金为 2 moac

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> solfile = 'VnodeProtocolBase.sol';
> contract = fs.readFileSync(solfile, 'utf8');
> output = solc.compile(contract, 1);
> abi = output.contracts[':VnodeProtocolBase'].interface;
> bin = output.contracts[':VnodeProtocolBase'].bytecode;
> VnodeProtocolBaseContract = chain3.mc.contract(JSON.parse(abi));
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> VnodeProtocolBase = VnodeProtocolBaseContract.new( 2, { from: chain3.mc.accounts[0], ␣
↪data: '0x' + bin,  gas: '5000000'});
> chain3.mc.getTransactionReceipt(VnodeProtocolBase.transactionHash).contractAddress
```

部署完毕后，获得 vnode 矿池合约地址 0x22f141dcc59850707708bc90e256318a5fe0b928

3.4.2 vnode 设置代理并加入矿池

修改 vnode 目录配置文件 vnodeconfig.json: VnodeBeneficialAddress 里设置收益账号: 0xf103bc1c054babcecd13e7ac1cf34f029647b08c

这个账号也作为 vnode 的 address，矿池中对应这个 vnode 的唯一编号

调用 vnode 矿池合约 register 方法加入矿池

参数:

```
from:  子链测试账号
value: 押金，必须大于矿池合约的设置值
to: vnode 矿池合约地址
data: register(address,string)
```

关于 data 传递调用 register 参数说明:

```
根据 ABI chain3.sha3("register(address,string)") =␣
↪0x32434a2e90725ed590daff07a244305001c58c49f7bef73ce5e7249acf69f561
        取前 4 个字节 0x32434a2e
第一个参数 address 传  vnodeconfig.json 的 VnodeBeneficialAddress  (前面补 24 个 0,  凑足
32 个字节)
        000000000000000000000000f103bc1c054babcecd13e7ac1cf34f029647b08c
第二个参数 string 传 vnode 提供给子链的调用地址 192.168.10.209:50062
端口号对应 vnodeconfig.json 的 VnodeServiceCfg
        string 数据类型 + string 数据长度 + string 内容
        string 数据类型:  0000000000000000000000000000000000000000000000000000000000000040
        string 内容:  data = new Buffer('192.168.10.209:50062', 'utf8').toString('hex');
                后面补 0,  凑足 32 个字节  ␣
↪3139322e3136382e31302e3230393a3530303632000000000000000000000000
        string 数据长度:     3139322e3136382e31302e3230393a3530303632        20 字节
                0000000000000000000000000000000000000000000000000000000000000014
        data =
↪'0x32434a2e000000000000000000000000f103bc1c054babcecd13e7ac1cf34f029647b08c0000000000000000000000000000
↪'
也可以调用合约对象的 getData 方法获得 data 参数
        data = VnodeProtocolBase.register.getData(via, '192.168.10.209:50062')
```

调用示例:

```
> amount = chain3.toSha(5,'mc')
> data =
↪'0x32434a2e000000000000000000000000f103bc1c054babcecd13e7ac1cf34f029647b08c0000000000000000000000000000
↪';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction({ from: chain3.mc.accounts[0], value:amount, to:
↪'0x22f141dcc59850707708bc90e256318a5fe0b928', gas: "5000000", gasPrice: chain3.mc.
↪gasPrice, data: data });
```

验证: 访问 Vnode 矿池合约的 vnodeCount

```
> VnodeProtocolBase.vnodeCount()
> chain3.mc.getStorageAt("0x22f141dcc59850707708bc90e256318a5fe0b928",0x02)   // 注意
vnodeCount 变量在合约中变量定义的位置（16 进制）
```

3.4.3 部署子链矿池

目前针对不同的共识协议，可以创建对应的子链矿池，接受对应 SCS 的注册，并缴纳保证金，进入矿池后，成为子链的候选节点

如果加入现成的子链矿池，则可以忽略此步骤

部署 SubChainProtocolBase.sol 示例: 共识:POR 最低保证金: 2 moac

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> solfile = 'SubChainProtocolBase.sol';
> contract = fs.readFileSync(solfile, 'utf8');
> output = solc.compile(contract, 1);
> abi = output.contracts[':SubChainProtocolBase'].interface;
> bin = output.contracts[':SubChainProtocolBase'].bytecode;
> subchainprotocolbaseContract = chain3.mc.contract(JSON.parse(abi));
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> subchainprotocolbase = subchainprotocolbaseContract.new( "POR",  2, { from: chain3.mc.
↪accounts[0],  data: '0x' + bin,  gas: '5000000'});
> chain3.mc.getTransactionReceipt(subchainprotocolbase.transactionHash).contractAddress
```

部署完毕后，获得子链矿池合约地址 0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac

3.4.4 设置启动 scs

这里我们设置三个 scs 节点

确认 userconfig.json 配置

```
VnodeServiceCfg 为代理 vnode 地址: 192.168.10.209:50062
Beneficiary 为收益账号:
        0xa934198916cd993c73c1aa6e0c0e7b21ce7c735b
        0x2e7c076dbf6e61207a0ddb1b942ef7da8fd139f0
        0xea1a118e94344be69f02753d1e6f7fe19dda89ac
```

分别通过命令启动 scsserver-windows-4.0-amd64 –password “123456”（生成 scs keystore 的密码）

然后在 scskeystore 目录内生成的 keystore 文件中分别获得 scs 地址

```
0xd4057328a35f34507dbcd295d43ed0cccf9c368a
0x3e21ba36b396936c6cc9adc3674655b912e5fa54
0x03c74ecc8ad9493a6a3d14f4e48d5eb551fe1be5
```

最后给 scs 转入 moac 以支付必要的交易费用

```
> amount = 20;
> scsaddr = '0xd4057328a35f34507dbcd295d43ed0cccf9c368a';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: scsaddr, gas: "2000000", gasPrice: chain3.mc.gasPrice, data: ''});
> scsaddr = '0x3e21ba36b396936c6cc9adc3674655b912e5fa54';
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: scsaddr, gas: "2000000", gasPrice: chain3.mc.gasPrice, data: ''});
> scsaddr = '0x03c74ecc8ad9493a6a3d14f4e48d5eb551fe1be5';
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: scsaddr, gas: "2000000", gasPrice: chain3.mc.gasPrice, data: ''});
```

可以通过查询余额进行验证

```
> chain3.mc.getBalance('0xd4057328a35f34507dbcd295d43ed0cccf9c368a')
> chain3.mc.getBalance('0x3e21ba36b396936c6cc9adc3674655b912e5fa54')
> chain3.mc.getBalance('0x03c74ecc8ad9493a6a3d14f4e48d5eb551fe1be5')
```

3.4.5 将 scs 加入子链矿池

调用子链矿池合约 register 方法加入矿池

参数:

```
from:  子链测试账号
value: 押金，必须大于矿池合约的设置值
to: 子链矿池合约地址
data: register(address)
```

关于 data 传递调用 register 参数说明:

```
根据 ABI chain3.sha3("register(address)") =␣
↪0x4420e4869750c98a56ac621854d2d00e598698ac87193cdfcbb6ed1164e9cbcd
        取前 4 个字节 0x4420e486
```

(下页继续)

(续上页)

```
参数 address 传 scs 地址    d4057328a35f34507dbcd295d43ed0cccf9c368a  (前面补 24 个 0, ␣
↪凑足 32 个字节)
       000000000000000000000000d4057328a35f34507dbcd295d43ed0cccf9c368a
data = '0x4420e486000000000000000000000000d4057328a35f34507dbcd295d43ed0cccf9c368a'
```

调用示例:

```
> amount = chain3.toSha(5,'mc')
> data = '0x4420e486000000000000000000000000d4057328a35f34507dbcd295d43ed0cccf9c368a';
> chain3.mc.sendTransaction({ from: chain3.mc.accounts[0], value:amount, to:
↪'0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac', gas: "5000000", gasPrice: chain3.mc.
↪gasPrice, data: data });
```

验证: 访问子链矿池合约的 scsCount

```
> subchainprotocolbase.scsCount()
```

同上将另两个 scs 也加入子链矿池

3.4.6 部署子链合约

现在我们可以部署一个子链合约

部署 SubChainBase.sol 示例:

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> input = {'': fs.readFileSync('SubChainBase.sol', 'utf8'), 'SubChainProtocolBase.sol':␣
↪fs.readFileSync('SubChainProtocolBase.sol', 'utf8')};
> output = solc.compile({sources: input}, 1);
> abi = output.contracts[':SubChainBase'].interface;
> bin = output.contracts[':SubChainBase'].bytecode;
> proto = '0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac' ;    // 子链矿池合约
> vnodeProtocolBaseAddr = '0x22f141dcc59850707708bc90e256318a5fe0b928' ;       // Vnode
矿池合约
> min = 1 ;                    // 子链需要 SCS 的最小数量, 当前需要从如下值中选择: 1, 3, 5,
7
> max = 11;             // 子链需要 SCS 的最大数量, 当前需要从如下值中选择: 11, 21, 31, 51,
99
```

(下页继续)

(续上页)

```
> thousandth = 1 ;                          // 千分之几，控制选择 scs 的概率，对于大型子链矿池
才有效
> flushRound = 40 ;              // 子链刷新周期  单位是主链 block 生成对应数量的时间，当前
的取值范围是 40-99
> SubChainBaseContract = chain3.mc.contract(JSON.parse(abi));
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> SubChainBase = SubChainBaseContract.new( proto, vnodeProtocolBaseAddr, min, max,␣
↪thousandth, flushRound,{ from: chain3.mc.accounts[0],  data: '0x' + bin,  gas:'9000000
↪'} , function (e, contract){console.log('Contract address: ' + contract.address + '␣
↪transactionHash: ' + contract.transactionHash); });
```

部署完毕后，获得子链合约地址 0x1195cd9769692a69220312e95192e0dcb6a4ec09

3.4.7 子链开放注册

首先子链合约需要最终提供 gas 费给 scs，需要给子链控制合约发送一定量的 moac，调用合约里的函数 addFund

```
根据 ABI chain3.sha3("addFund()") =␣
↪0xa2f09dfa891d1ba530cdf00c7c12ddd9f6e625e5368fff9cdf23c9dc0ad433b1
        取前 4 个字节 0xa2f09dfa
> amount = 20;
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: subchainaddr, gas: "2000000", gasPrice: chain3.mc.gasPrice, data: '0xa2f09dfa'}
↪);
```

可以通过查询余额进行验证

```
> chain3.mc.getBalance('0x1195cd9769692a69220312e95192e0dcb6a4ec09')
```

然后调用调用合约里的函数 registerOpen 开放注册 (按子链矿池合约中 SCS 注册先后排序进行选取)

```
根据 ABI chain3.sha3("registerOpen()") =␣
↪0x5defc56ce78f178d760a165a5528a8e8974797e616a493970df1c0918c13a175
        取前 4 个字节 0x5defc56c
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: '0x5defc56c'});
```

验证：等待 scs 注册 (vnode 一个 flush 周期后)，可不断访问子链合约的 nodeCount，等待 3 个 scs 注册完成

```
> SubChainBase.nodeCount()
> chain3.mc.getStorageAt(subchainaddr,0x0e)  // 注意 nodeCount 变量在合约中变量定义的位置
(16 进制)
```

3.4.8 子链关闭注册

等到两个 scs 都注册完毕后，即注册 SCS 数目大于等于子链要求的最小数目时，调用子链合约里的函数 registerClose 关闭注册

```
根据 ABI chain3.sha3("registerClose()") =␣
↪0x69f3576fc10c82561bd84b0045ee48d80d59a866174f2513fdef43d65702bf70
        取前 4 个字节 0x69f3576f
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: '0x69f3576f'});
```

验证：SCS 自身完成初始化并开始子链运行，可观察 scs 的 concole 界面，scs 开始出块即成功完成部署子链。

3.5 子链业务逻辑的部署

同主链相同，业务逻辑的实现也通过智能合约的方式。

在 1.0.8 及之后的版本中，加入的子链多合约的功能。子链多合约功能指的是在一条子链上部署多个子链业务逻辑合约，又称 DAPP 的功能。

3.5.1 多合约部署准备工作

需要有一条已经在运行的子链，具体部署方法可参见“子链部署方法”。

假设有两个业务逻辑合约 dapp1.sol 和 dapp2.sol。

从发布链接下载多合约基础合约 dappbase.sol

3.5.2 DAPP 智能合约的部署（多合约版）

DAPP 智能合约也通过主链的 sendTransaction 发送交易到 proxy vnode 的方式进行部署。

参数：

```
to: 子链控制合约 subchainbase 的地址
gas: 不需要消耗 gas 费用，传值：0
shardingflag：表示操作子链， 传值：0x3，请注意，多合约版本部署任何合约 shardingflag 都为 0x3
via: 对应 proxy vnode 的收益地址
```

STEP1：在子链上部署多合约基础合约 DappBase.sol

!!! 特别注意!!!：目前子链原生币支持 moac 和 erc20 两种兑换方式，交易 values 分别对应 subchainbase 的 tokensupply 和 erc20 的 totalsupply，这个值必须对应，否则将会导致 dappbase 部署失败。细节详见母子链货币交互章节

部署示例（以下在 nodeJs console 中进行）：

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> solfile = 'DappBase.sol';
> contract = fs.readFileSync(solfile, 'utf8');
> output = solc.compile(contract, 1);
> abi = output.contracts[':DappBase'].interface;
> bin = output.contracts[':DappBase'].bytecode;
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> amount = tokensupply // 注意：amount 分别对应 subchainbase 的 tokensupply 和 erc20 的
totalsupply，细节详见母子链货币交互章节
> chain3.mc.sendTransaction({from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: subchainaddr, gas:0, shardingFlag: "0x3", data: '0x' + bin, nonce: 0, via: via,
↪ });
```

验证： 合约部署成功后，Nonce 值应该是 1 可调用 monitor 的 rpc 接口 ScsRPCMethod.GetNonce 进行检查，具体详见子链接口调用部分。

dapp 合约地址:6ab296062d8a147297851719682fb5ffe081f1d3 dapp 合约地址可调用 monitor 的 rpc 接口 ScsRPCMethod.GetReceipt，传入对应 Nonce，获得 contractAddress 字段内容

STEP2：在子链上部署业务逻辑合约 dapp1.sol，部署方法和上面雷同 合约地址可调用 monitor 的 rpc 接口 ScsRPCMethod.GetReceipt，传入对应 Nonce，获得 contractAddress 字段内容

STEP3：在子链上部署业务逻辑合约 dapp2.sol，部署方法和上面雷同 合约地址可调用 monitor 的 rpc 接口 ScsRPCMethod.GetReceipt，传入对应 Nonce，获得 contractAddress 字段内容

3.5.3 DAPP 智能合约的调用

DAPP 智能合约的调用也通过主链的 sendTransaction 发送交易到 proxy vnode 的方式进行。

在多合约版本中，调用 dapp 方法前，需要先调用 dappbase 中的 registerDapp 方法来注册每一个 dapp，具体方式如下：

请注意，与母链调用不同，子链的任何调用需要在 data 前加上 dapp 的合约地址!!

dappbase.sol 有个方法 registerDapp(address,address,string)

参数：

```
to: 子链控制合约 subchainbase 的地址
nonce: 调用 monitor 的 rpc 接口 ScsRPCMethod.GetNonce 获得
gas: 0 不需要消耗 gas 费用
shardingflag: 0x1 表示子链调用操作
via: 对应 proxy vnode 的收益地址
data: 调用合约地址 + registerDapp(address,address,string) 对应参数
```

registerDapp 中，第一个参数是想要注册的 dapp 的地址（dapp1 和 dapp2 的地址），可以通过 RPC getReciipt 方法获得部署时 contract address；第二个参数是创建 dappbase 时的 from，也就是只有创建 dappbase 的人才能调用此方法；第三个参数是这个 dapp 的 ABI。

调用示例：

```
> nonce = 3
> addr_dapp = 需要注册 dapp 的合约地址
> abi = 需要注册 dapp 的 abi
> data = dappbase.address + dappbase.registerDapp.getData(addr_dapp, chain3.mc.
↪accounts[0], abi).substring(2)
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { nonce: nonce, from: chain3.mc.accounts[0], value:0, to:␣
↪subchainaddr, gas:0, shardingFlag:'0x1', data: data, via: via,});
```

验证： 每次操作成功后，Nonce 会自动增加 1 或者直接调用 monitor 的 rpc 接口 ScsRPCMethod.GetDappAddrList 获得合约注册列表的方式进行验证。

以部署 dapp1 和 dapp2 为例，需要将这两个业务逻辑合约注册到 dappbase 中去：

STEP4：调用 dappbase 中的 registerDapp 方法来注册 dapp1

STEP5：调用 dappbase 中的 registerDapp 方法来注册 dapp2

STEPX：调用 dapp1 或 dapp2 中的业务逻辑

3.6 子链的监听及子链接口调用

3.6.1 SCS Monitor

Monitor 是一种特殊的子链 SCS 节点，其主要可以用于监控子链的状态和数据。

Monitor 不参与子链的交易共识，只是同步区块数据，提供数据查询

子链启动的方式与 scs 区别在于参数不同，主要定义了 rpc 接口的访问控制

```
scsserver-windows-4.0-amd64 --password "123456" --rpcdebug --rpcaddr 0.0.0.0 --rpcport␣
↪2345 --rpccorsdomain "*"
```

子链运行后，Monitor 可以调用子链控制合约 subchainbase 中的 registerAsMonitor 方法进行注册

调用 registerAsMonitor 参数说明:

```
> data = subchainbase.registerAsMonitor.getData(
↪'0xd135afa5c8d96ba11c40cf0b52952d54bce57363','127.0.0.1:2345')
```

调用示例:

```
> subchainbase = SubChainBaseContract.at('0xb877bf4e4cc94fd9168313e00047b77217760930')
> amount = chain3.toSha(1,'mc')
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> data = subchainbase.registerAsMonitor.getData(
↪'0xd135afa5c8d96ba11c40cf0b52952d54bce57363','127.0.0.1:2345')
> chain3.mc.sendTransaction({ from: chain3.mc.accounts[0], value:amount, to:␣
↪subchainaddr, gas: "5000000", gasPrice: chain3.mc.gasPrice, data: data });
```

验证：观察 SCS monitor concole 界面开始同步子链区块，或者调用子链合约的 getMonitorInfo 方法

```
> subchainbase = SubChainBaseContract.at('0xb877bf4e4cc94fd9168313e00047b77217760930')
> subchainbase.getMonitorInfo.call()
```

3.6.2 子链 RPC 接口

根据子链启动的参数，用户可以访问对应端口，调用接口获取子链相关数据

以调用接口 GetScsId 为列，获得当前的 scs 编号，即 scs keystore 文件中的地址。

关于 rpc http 的接口访问，可以用类似 postman 之类的工具进行快捷测试

```
header 设置:
        Content-Type = application/json
        Accept = application/json

Body 设置:
        {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetScsId","params":{}}

post 返回结果:
        {
                "jsonrpc": "2.0",
                "id": 0,
                "result": "0xd135afa5c8d96ba11c40cf0b52952d54bce57363"
        }
```

同时也可以通过 nodejs 的方式调用

```
> request = require('request');
> url  = "http://127.0.0.1:2345/rpc";
> data = {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetScsId","params":{}};
> request({ url: url, method: "POST", json: true, body: data, headers: {"Content-Type":
↪'application/json', "Accept": 'application/json'}}, function(error, response, result)
↪{if (!error && response.statusCode == 200) {console.log(result)}});
```

以下是几个常用的 RPC 接口及调用 body 示例（v1.0.9）

*** 通用类 ***

此部分接口和子链区块本身相关，业务逻辑无关。

GetNonce：获得子链的 nonce，这是调用子链 DAPP 合约的必要参数之一，每当子链交易发送后会自动加 1

```
SubChainAddr: 子链合约地址
Sender: 查询账号， 每个账号在子链有不同的 nonce
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetNonce",
                        "params":{"SubChainAddr":
↪"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                                "Sender":"0x87e369172af1e817ebd8d63bcd9f685a513a6736"
                        }
          }
```

GetBlockNumber：获得当前子链的区块高度

```
SubChainAddr: 子链合约地址
```

(下页继续)

(续上页)

```
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetBlockNumber",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09"}
          }
```

GetBlock: 获得当前子链的指定的区块信息

```
SubChainAddr: 子链合约地址
Sender: 查询账号
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetBlock",
                "params":{"number":1000,"SubChainAddr":
↪"0x1195cd9769692a69220312e95192e0dcb6a4ec09"}
          }
```

GetBlocks: 获取某一区间内的区块信息

```
SubChainAddr: 子链合约地址
Start: 开始 block
End:  结束 block
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetBlocks",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09"
                        "Start":10, "End":20}
          }
```

GetSubChainInfo：获得当前子链的信息

```
SubChainAddr: 子链合约地址
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetSubChainInfo",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09"}
          }
```

GetTxpool：获得子链交易池信息

```
SubChainAddr: 子链合约地址
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetTxpool",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09"
                 }
          }
```

GetTxpoolCount：获得子链交易池中不同类型交易的数量

```
SubChainAddr: 子链合约地址
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetTxpoolCount",
```

(下页继续)

(续上页)

```
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09"
                 }
          }
```

GetBalance：获得对应账号在子链中的余额

```
SubChainAddr: 子链合约地址
Sender: 查询账号
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetBalance",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                        "Sender":"0x87e369172af1e817ebd8d63bcd9f685a513a6736"
                        }
          }
```

GetDappState：获得子链基础合约合约的状态

```
SubChainAddr: 子链合约地址
Sender: 子链合约地址创建者地址
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetDappState",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                        "Sender":"0x87e369172af1e817ebd8d63bcd9f685a513a6736"
                 }
          }
```

GetDappAddrList：通过 subchainaddr 获取子链内所有多合约的地址列表，需要子链业务逻辑合约调用基础合约 registerDapp 方法后才能生效，具体请参见“母子链货币交互简介”中的示例

```
SubChainAddr: 子链合约地址
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetDappAddrList",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09"
                 }
          }
```

返回 result 中，第零位是 dappbase 的地址，从第一位开始时业务逻辑合约地址

*** 充提类 ***

GetExchangeInfo：获得子链指定数量正在充提的信息

```
SubChainAddr: 子链合约地址
EnteringRecordIndex:  正在充值记录的起始位置 (0)
EnteringRecordSize: 正在充值记录的长度
RedeemingRecordIndex:  正在提币记录的起始位置 (0)
```

(下页继续)

(续上页)

```
RedeemingRecordSize: 正在提币记录的长度
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetExchangeInfo",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                "EnteringRecordIndex":0, "EnteringRecordSize": 10,
                "RedeemingRecordIndex":0, "RedeemingRecordSize", 10
                 }
         }
```

返回中，XXXRecordCount 是指总数量

GetExchangeByAddress：获得子链指定账号指定数量的充提信息

```
SubChainAddr: 子链合约地址
Sender: 需要查询的账号地址
EnterRecordIndex: 已经充值记录的起始位置 (0)
EnterRecordSize: 已经充值记录的长度
RedeemRecordIndex: 已经提币记录的起始位置 (0)
RedeemRecordSize: 已经提币记录的长度
EnteringRecordIndex: 正在充值记录的起始位置 (0)
EnteringRecordSize: 正在充值记录的长度
RedeemingRecordIndex: 正在提币记录的起始位置 (0)
RedeemingRecordSize: 正在提币记录的长度
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetExchangeByAddress",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                "EnterRecordIndex":0, "EnterRecordSize": 10,
                "RedeemRecordIndex":0, "RedeemRecordSize", 10
                "EnteringRecordIndex":0, "EnteringRecordSize": 10,
                "RedeemingRecordIndex":0, "RedeemingRecordSize", 10
                 }
         }
```

返回中，XXXRecordCount 是指总数量

*** 交易类 ***

此部分接口和交易相关，当有子链交易发生（sf>0），即可以在这里查看交易和交易结果

GetTransactionByNonce: 通过账号和 Nonce 获取子链的 tx 信息

```
SubChainAddr: 子链合约地址
Sender: 查询账号
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetTransactionByNonce",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
```

(下页继续)

(续上页)

```
                        "Sender":"0x87e369172af1e817ebd8d63bcd9f685a513a6736",  "Nonce
↪":9,
                        }
        }
```

GetTransactionByHash: 通过交易 hash 获取子链的 tx 信息

```
SubChainAddr: 子链合约地址
Hash: 交易 hash
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetTransactionByHash",
               "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                        "Hash":
↪"0x87e369172af1e817ebd8d63bcd9f685a513a6736fsne3lkgkvu65kkwlcd"
                        }
        }
```

GetReceiptByNonce: 通过账号和 Nonce 获取子链的 tx 执行结果

```
SubChainAddr: 子链合约地址
Sender: 查询账号
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetReceiptByNonce",
               "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                        "Sender":"0x87e369172af1e817ebd8d63bcd9f685a513a6736",  "Nonce":9
                        }
        }
```

注意：如果这是个合约部署的交易，则在 contractAddress 将会显示合约地址；如果是一个有返回值的方法调用，则在 result 中显示调用结果

GetReceiptByHash: 通过交易 hash 获取子链的 tx 执行结果

```
SubChainAddr: 子链合约地址
Sender: 查询账号
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.GetReceiptByHash",
               "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                        "Hash":
↪"0x87e369172af1e817ebd8d63bcd9f685a513a6736fsne3lkgkvu65kkwlcd"
                        }
        }
```

注意：如果这是个合约部署的交易，则在 contractAddress 将会显示合约地址；如果是一个有返回值的方法调用，则在 result 中显示调用结果

*** 业务类 ***

此部分合约需要指明是哪个业务逻辑合约

AnyCall: 获取 dapp 合约函数的返回值，**调用此接口前必须将 dapp 注册入 dappbase**

Params：第一个参数是调用的方法，之后是方法传入参数

```
SubChainAddr: 子链合约地址
Sender: 查询账号
DappAddr: 子链业务逻辑地址
Body: {"jsonrpc":"2.0","id":0,"method":"ScsRPCMethod.AnyCall",
                "params":{"SubChainAddr":"0x1195cd9769692a69220312e95192e0dcb6a4ec09",
                        "DappAddr":"0xcc0D18E77748AeBe3cC6462be0EF724e391a4aD9",
                        "Sender":"0x87e369172af1e817ebd8d63bcd9f685a513a6736",  "Params"␣
↪:["funcA", "param1", param2]
                        }
          }
```

3.7 子链节点的更替

3.7.1 子链节点添加

子链合约提供了 registerAdd 方法来支持子链添加，必须由子链部署账号来发送交易请求。

需要对应 SubChainProtocolBase 矿池合约有等待加入的 scs 节点。

子链收到请求后，在矿池合约选取 scs，开始同步子链区块，等一轮 flush 后生效，正式加入子链。

registerAdd 参数:

```
nodeToAdd:  当前 scs 数 + 需要加入 scs 数
```

调用示例:

```
> data = subchainbase.registerAdd.getData(20)
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: data});
```

验证：scs 对应日志开始同步区块，合约公共变量 codeCount 更新为 scs 最新数量

```
> SubChainBase.nodeCount()
```

3.7.2 子链节点退出

子链节点退出有两种方式：

1. 当子链工作正常时，调用子类合约 requestRelease 方法请求退出子链，等待一轮 flush 后生效。

requestRelease 参数:

```
senderType:      1: scs 发起请求        2: 收益账号发出请求
index:           scs 序号 (参考 ScsRPCMethod.GetSubChainInfo 中 scs 的列表)
```

调用示例:

```
> data = subchainbase.requestRelease.getData(senderType, index)
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: data});
```

验证：等待一轮 flush 后，关注合约公共变量 codeCount 是否变化

```
> SubChainBase.nodeCount()
```

2. 当子链工作不正常时，可以调用子类合约 requestReleaseImmediate 方法请求立即退出子链。

requestReleaseImmediate 参数:

```
senderType:      1: scs 发起请求        2: 收益账号发出请求
index:           scs 序号 (参考 ScsRPCMethod.GetSubChainInfo 中 scs 的列表)
```

调用示例:

```
> data = subchainbase.requestReleaseImmediate.getData(senderType, index)
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: data});
```

验证：合约公共变量 codeCount 是否变化

```
> SubChainBase.nodeCount()
```

3.8 关闭子链

3.8.1 子链关闭请求

子链合约提供了 close 的方法来支持关闭子链，必须由子链部署账号来发送交易请求。

调用示例:

```
根据 ABI chain3.sha3("close()") =␣
↪0x43d726d69bfad97630bc12e80b1a43c44fecfddf089a314709482b2b0132f662
        取前 4 个字节 0x43d726d6
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value:0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: '0x43d726d6'});
```

关闭请求发送后，需等待一轮 flush 后生效，相关子链维护费用也将退回到子链部署账号中。可以通过查询余额进行验证

```
> chain3.mc.getBalance('0x1195cd9769692a69220312e95192e0dcb6a4ec09')
```

3.9 母子链货币交互简介

墨客支持有币子链，并且提供母链货币和子链原生币之间的兑换。

当前，墨客提供三种类型的母子链货币交互，以下一一介绍

3.9.1 母链 MOAC 和子链原生币交互

这个是最基础的一种货币兑换。使用者可以在主链上充值 MOAC，然后最早在下一个 flush 周期在子链上获取子链原生币。同理，使用者可以提出子链原生币，并最早在下一个 flush 周期获得主链 MOAC。

对应 release 中的 ASM 版本包

子链部署准备

参考子链部署章节，完成部署子链的准备工作。

```
子链操作账号: 0x87e369172af1e817ebd8d63bcd9f685a513a6736
vnode 矿池合约地址: 0x22f141dcc59850707708bc90e256318a5fe0b928
vnode 代理地址: 0xf103bc1c054babcecd13e7ac1cf34f029647b08c    192.168.10.209:50062
```

(下页继续)

(续上页)

```
子链矿池合约地址: 0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac
scs0: 0x075447a6df1fde4f39243bc67a945312ff36c193  确保启动并加入子链矿池
scs1: 0x7932f827c90c5f06c0177f642a07edfa73ee3044  确保启动并加入子链矿池
scs2 兼 monitor: 0xa5966600efb221097ce6a8ba1dc6eb1d5b43ef83  确保启动并加入子链矿池
```

subchainbase 部署

注意这个子链合约在官方基础上进行了修改，对应 v1 目录下的 SubChainBase.sol（带 tokensupply 和 exchangerate 参数）。部署 SubChainBase.sol 示例:

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> input = {'': fs.readFileSync('SubChainBase.sol', 'utf8'), 'SubChainProtocolBase.sol':␣
↪fs.readFileSync('SubChainProtocolBase.sol', 'utf8')};
> output = solc.compile({sources: input}, 1);
> abi = output.contracts[':SubChainBase'].interface;
> bin = output.contracts[':SubChainBase'].bytecode;
> proto = '0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac' ;    // 子链矿池合约
> vnodeProtocolBaseAddr = '0x22f141dcc59850707708bc90e256318a5fe0b928' ;       // Vnode
矿池合约
> min = 1 ;                    // 子链需要 SCS 的最小数量，当前需要从如下值中选择: 1, 3, 5,
7
> max = 11 ;            // 子链需要 SCS 的最大数量，当前需要从如下值中选择: 11, 21, 31, 51,
99
> thousandth = 1 ;                      // 千分之几
> flushRound = 40 ;             // 子链刷新周期  单位是主链 block 生成对应数量的时间（10 秒
一个 block）
> tokensupply = 1000;    // 子链原生币总额
> exchangerate = 100;           // 与 moac 兑换比例
> SubChainBaseContract = chain3.mc.contract(JSON.parse(abi));
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> SubChainBase = SubChainBaseContract.new( proto, vnodeProtocolBaseAddr, min, max,␣
↪thousandth, flushRound, tokensupply, exchangerate { from: chain3.mc.accounts[0], ␣
↪data: '0x' + bin,  gas:'9000000'} , function (e, contract){console.log('Contract␣
↪address: ' + contract.address + ' transactionHash: ' + contract.transactionHash); });
```

部署完毕后，获得子链合约地址 0xe9463e215315d6f1e5387a161868d7d0a4db89e8

验证:

访问子链合约的 BALANCE 与 tokensupply 对应

应该是 10 的 21 次方: 即 tokensupply * 10 的 18 次方 (子链原生币 decimals)

调用示例:

```
> subchainaddr = '0xe9463e215315d6f1e5387a161868d7d0a4db89e8';
> SubChainBase = SubChainBaseContract.at(subchainaddr);
> SubChainBase.BALANCE()
```

调用 monitor 的方法 ScsRPCMethod.GetBalance 查询对应部署账号地址的余额，应该也等于 10 的 21 次方

dappbase 合约部署

子链开放注册后，待 scs 开始出块即成功完成部署子链，方法请参见”子链的部署方法”。按照多合约部署步骤，需要首先部署 dappbase 合约，方法请参见”子链业务逻辑的部署”。

注意部署 dappbase 合约的 value 为 1000(tokensupply) * 10 的 18 次方 (子链原生币 decimals)

部署示例:

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> solfile = 'DappBase.sol';
> contract = fs.readFileSync(solfile, 'utf8');
> output = solc.compile(contract, 1);
> abi = output.contracts[':DappBase'].interface;
> bin = output.contracts[':DappBase'].bytecode;
> amount = chain3.toSha(1000,'mc')
> subchainaddr = '0xb877bf4e4cc94fd9168313e00047b77217760930';
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction({from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: subchainaddr, gas:0, gasPrice: 0, shardingFlag: "0x3", data: '0x' + bin,␣
↪nonce:0, via: via });
```

验证:

调用 monitor 的方法 ScsRPCMethod.GetNonce Nonce 值应该是 1

调用 monitor 的方法 ScsRPCMethod.GetBalance 查询对应 dappbase 合约地址的余额，应该等于 10 的 21 次方

调用 monitor 的方法 ScsRPCMethod.GetBalance 查询对应部署账号地址的余额，应该等于 0

调用 monitor 的方法 ScsRPCMethod.GetReceipt 传入对应 Nonce，从 contractAddress 字段内容获得合约地址

dapp 充值

调用 subchainbase 的 buyMintToken 方法充值，用户账号为发出 sendTransaction 的账号数量为 sendTransaction 的 amount 参数

调用示例：

```
根据 ABI chain3.sha3("buyMintToken()") =␣
↪0x6bbded701cd78dee9626653dc2b2e76d3163cc5a6f81ac3b8e69da6a057824cb
        取前 4 个字节 0x6bbded70
> amount = 1;
> subchainaddr = '0xe9463e215315d6f1e5387a161868d7d0a4db89e8';
> chain3.personal.unlockAccount(chain3.mc.accounts[1], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[1], value: chain3.toSha(amount,
↪'mc'), to: subchainaddr, gas:"2000000", gasPrice: chain3.mc.gasPrice, data: '0x6bbded70
↪'});
```

验证：

检查账号的 moac 是否减少: > chain3.mc.getBalance(chain3.mc.accounts[1])

检查子链的 token 是否增加: 调用 monitor 的方法 ScsRPCMethod.GetBalance 获得子链 chain3.mc.accounts[1] 地址对应 token

检查子链 dappbase 合约地址的原生币是否减少: 调用 monitor 的方法 ScsRPCMethod.GetBalance

dapp 提币

请注意 data 前需要加上 dappbase 合约地址

调用 dappbase 合约的 redeemFromMicroChain 方法，用户账号为发出 sendTransaction 的账号数量为 sendTransa

redeemFromMicroChain 方法将用户账号和对应 token 数量加入推送结构体 redeem，等待一轮 flush 后生效

调用示例：

```
根据 ABI chain3.sha3("redeemFromMicroChain()") =␣
↪0x89739c5bf1ef36273bf0e7aeb59ffe71213a58e1f01965e75662cb21b03abb13
取前 4 个字节 0x89739c5b
调用 dapp 合约方法，需要再 data 前加入 dappaddr
> nonce = 1       // 调用 ScsRPCMethod.GetNonce 获得
```

(下页继续)

(续上页)

```
> subchainaddr = '0x1195cd9769692a69220312e95192e0dcb6a4ec09';
> dappbassaddr = dappbase 合约地址
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> amount = 10  // 对应子链原生币  10 * 18 次方    即 0.1 moac
> chain3.personal.unlockAccount(chain3.mc.accounts[1], '123456');
> chain3.mc.sendTransaction( { nonce: nonce, from: chain3.mc.accounts[1], value:chain3.
↪toSha(amount,'mc'), to: subchainaddr, gas:0, shardingFlag:'0x1', data: dappbassaddr +
↪'89739c5b', via: via,});
```

验证:

检查账号的 moac 是否增加: > chain3.mc.getBalance(chain3.mc.accounts[1])

检查子链的 token 是否减少: 调用 monitor 的方法 ScsRPCMethod.GetBalance 获得子链 token

检查子链 dappbase 合约地址的原生币是否增加: 调用 monitor 的方法 ScsRPCMethod.GetBalance

3.9.2 母链 ERC20 和子链原生币交互

这是非常通用的一种货币兑换。使用者可以使用预先已经部署好的 ERC20，或者当场部署一个主链 ERC20，和子链的原生币进行兑换。

对应 release 中的 AST 版本包

子链部署准备

参考子链部署章节，完成部署子链的准备工作。

```
子链操作账号: 0x87e369172af1e817ebd8d63bcd9f685a513a6736
vnode 矿池合约地址: 0x22f141dcc59850707708bc90e256318a5fe0b928
vnode 代理地址: 0xf103bc1c054babcecd13e7ac1cf34f029647b08c    192.168.10.209:50062
子链矿池合约地址: 0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac
scs0:   0xd81043d85c9c959d2925958c54c1a49c7bfd1fc8  确保启动并加入子链矿池
scs1:   0xe767059d768fcef12e527fab63fda68cc13e24b3  确保启动并加入子链矿池
scs2 兼 monitor:        0x0964e5d73d6a40f2fc707aa3e1361028a34923f0 确保启动并加入子链矿池
```

erc20 部署

默认一个标准的 erc20 合约，通过 allowance，transferFrom，balanceOf，transfer 等标准的方法支持货币的转移。

参考官方示例的 erc20 合约 erc20.sol，默认 decimals 为 2，totalSupply 为 10000 乘以 10 的 2 次方。调用示例:

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> solfile = 'erc20.sol';
> contract = fs.readFileSync(solfile, 'utf8');
> output = solc.compile(contract, 1);
> abi = output.contracts[':TestCoin'].interface;
> bin = output.contracts[':TestCoin'].bytecode;
> erc20Contract = chain3.mc.contract(JSON.parse(abi));
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> dtoken = erc20Contract.new( { from: chain3.mc.accounts[0],  data: '0x' + bin,  gas:
↪'9000000'} , function (e, contract){console.log('Contract address: ' + contract.
↪address + ' transactionHash: ' + contract.transactionHash); });
```

部署完毕后, 获得 erc20 合约地址 0x5042086887a86151945d2c2bb60628addf49d48c

验证：调用合约 balanceOf 方法查询部署者的余额，应该是 1000000

```
> contractInstance = erc20Contract.at('0x5042086887a86151945d2c2bb60628addf49d48c')
> contractInstance.balanceOf.call('0x87e369172af1e817ebd8d63bcd9f685a513a6736')
```

subchainbase 部署

注意这个子链合约在官方基础上进行了修改，增加了 erc20 合约地址和兑换比例的参数部署 SubChainBase.sol 示例:

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> input = {'': fs.readFileSync('SubChainBase.sol', 'utf8'), 'SubChainProtocolBase.sol
↪':fs.readFileSync('SubChainProtocolBase.sol', 'utf8')};
> output = solc.compile({sources: input}, 1);
> abi = output.contracts[':SubChainBase'].interface;
> bin = output.contracts[':SubChainBase'].bytecode;
> proto = '0xe42f4f566aedc3b6dd61ea4f70cc78d396130fac' ;    // 子链矿池合约
> vnodeProtocolBaseAddr = '0x22f141dcc59850707708bc90e256318a5fe0b928' ;       // Vnode
矿池合约
> ercAddr = '0x5042086887a86151945d2c2bb60628addf49d48c';     // erc20 合约地址
> ercRate = 10;    // 兑换比率
```

(下页继续)

(续上页)

```
> min = 1 ;                    // 子链需要 SCS 的最小数量，当前需要从如下值中选择：1，3，5，
7
> max = 11 ;            // 子链需要 SCS 的最大数量，当前需要从如下值中选择：11，21，31，51，
99
> thousandth = 1 ;                      // 千分之几
> flushRound = 40 ;             // 子链刷新周期  单位是主链 block 生成对应数量的时间（10 秒
一个 block）
> SubChainBaseContract = chain3.mc.contract(JSON.parse(abi));
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> SubChainBase = SubChainBaseContract.new( proto, vnodeProtocolBaseAddr, ercAddr,␣
↪ercRate, min, max, thousandth, flushRound,{ from: chain3.mc.accounts[0],  data: '0x' +␣
↪bin,  gas:'9000000'} , function (e, contract){console.log('Contract address: ' +␣
↪contract.address + ' transactionHash: ' + contract.transactionHash); });
```

部署完毕后，获得子链合约地址 0xb877bf4e4cc94fd9168313e00047b77217760930

验证：

访问子链合约的 BALANCE 与 ERC20 的 totalsupply 对应

应该是 10 的 23 次方：即 1000000(ERC20 的 totalsupply) * 10(兑换比率) * 10 的 18 次方（子链原生币 decimals) / 10 的 2 次方 (ERC20 的 decimals)

调用示例：

```
> subchainaddr = '0xb877bf4e4cc94fd9168313e00047b77217760930';
> SubChainBase = SubChainBaseContract.at(subchainaddr);
> SubChainBase.BALANCE()
```

调用 monitor 的方法 ScsRPCMethod.GetBalance 查询对应部署账号地址的余额，应该等于 10 的 23 次方

dappbase 合约部署

子链开放注册后，待 scs 开始出块即成功完成部署子链，方法请参见”子链的部署方法”。按照多合约部署步骤，需要首先部署 dappbase 合约，方法请参见”子链业务逻辑的部署”。

注意部署 dappbase 合约的 value 为 ERC20 的 totalsupply * 10(兑换比率) * 10 的 18 次方（子链原生币 decimals) / 10 的 2 次方 (ERC20 的 decimals)

部署示例：

```
> chain3 = require('chain3')
> solc = require('solc')
> chain3 = new chain3();
```

(下页继续)

(续上页)

```
> chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));
> solfile = 'DappBase.sol';
> contract = fs.readFileSync(solfile, 'utf8');
> output = solc.compile(contract, 1);
> abi = output.contracts[':DappBase'].interface;
> bin = output.contracts[':DappBase'].bytecode;
> amount = chain3.toSha(100000,'mc')
> subchainaddr = '0xb877bf4e4cc94fd9168313e00047b77217760930';
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction({from: chain3.mc.accounts[0], value:chain3.toSha(amount,'mc
↪'), to: subchainaddr, gas:0, gasPrice: 0, shardingFlag: "0x3", data: '0x' + bin,␣
↪nonce:0, via: via });
```

验证:

调用 monitor 的方法 ScsRPCMethod.GetBalance 查询对应 dappbase 合约地址的余额，应该等于 10 的 23 次方

调用 monitor 的方法 ScsRPCMethod.GetBalance 查询对应部署账号地址的余额，应该等于 0

调用 monitor 的方法 ScsRPCMethod.GetReceipt 传入对应 Nonce，从 contractAddress 字段内容获得合约地址

dapp 充值

调用 subchainbase 的 buyMintToken 方法充值，用户账号为发出 sendTransaction 的账号，参数分别为子链合约地址和 erc20 的数量。注意：buyMintToken 方法首先调用 erc20 合约的 allowance 检查授权，再调用 transferFrom 方法将 token 从用户账号地址转到合约地址所以要先调用 erc20 的 approve 方法授权对应的 erc20 给 subchainbase 合约地址。

调用示例：

```
> amount = 200
> data = erc20.approve.getData(subchainaddr, amount);
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value: 0, to: erc20.address,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: data});
> subchainaddr = '0xb877bf4e4cc94fd9168313e00047b77217760930';
> SubChainBase = SubChainBaseContract.at(subchainaddr);
> data = SubChainBase.buyMintToken.getData(amount)
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { from: chain3.mc.accounts[0], value: 0, to: subchainaddr,␣
↪gas: "2000000", gasPrice: chain3.mc.gasPrice, data: data});
```

验证：

检查账号的 erc20 token 是否减少 200: 调用 erc20 合约的 balanceOf 方法

检查子链对应账号的原生币是否增加 20000000000000000000: 调用 monitor 的方法 ScsRPCMethod.GetBalance

检查子链 dappbase 合约地址的原生币是否减少 20000000000000000000: 调用 monitor 的方法 ScsRPCMethod.GetBalance

dapp 提币

请注意 data 前需要加上 dappbase 合约地址

调用 dappbase 合约的 redeemFromMicroChain 方法，用户账号为发出 sendTransaction 的账号数量为 sendTransa

redeemFromMicroChain 方法将用户账号和对应 token 数量加入推送结构体 redeem，等待一轮 flush 后，自动会调用子链合约的 redeemFromMicroChain 方法

调用 erc20 合约的 transfer 给用户账号转对应的 token 数量

调用示例：

```
根据 ABI chain3.sha3("redeemFromMicroChain()") =␣
↪0x89739c5bf1ef36273bf0e7aeb59ffe71213a58e1f01965e75662cb21b03abb13
取前 4 个字节 89739c5b
调用 dapp 方法，需要再 data 前加入 dappaddr
> nonce = 5       // 调用 ScsRPCMethod.GetNonce 获得
> subchainaddr = '0xb877bf4e4cc94fd9168313e00047b77217760930';
> dappbassaddr = dappbase 合约地址
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> amount = chain3.toSha(10,'mc')    //   * 10 的 2 次方 (ERC20 的 decimals) / 10(兑换比
率)   100 即为对应 erc20 数量
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { nonce: nonce, from: chain3.mc.accounts[0], value:amount,␣
↪to: subchainaddr, gas:0, shardingFlag:'0x1', data: dappbassaddr + '89739c5b', via: via,
↪});
```

验证：

检查账号的 erc20 token 是否增加 100: 调用 erc20 合约的 balanceOf 方法

等待一轮 flush 后，检查子链对应账号的原生币是否减少 10000000000000000000: 调用 monitor 的方法 ScsRPCMethod.GetBalance

3.9.3 ATO 方式

TODO

3.10 多合约的进阶操作

1.0.8 以后，子链引入了多合约的概念。子链多合约指的是在一条子链中部署多个智能合约，多合约可以将业务逻辑进行拆分，相互调用，合约之间也可以进行升级。

3.10.1 子链业务逻辑合约间的调用

本实例完成一个合约调用另外一个合约中的方法。

STEP0：部署 dappbase.sol

STEP1：部署合约 dapp1.sol，并注册到 dappbase 中去

```
contract Dapp1 {
struct tup {
    uint amount;
    string desc;
}
tup[] public tups;
function addTup(uint amount, string desc) public {
    tups.push(tup(desc));
}
function getString() public view returns (string) {
    return tups[0].desc;
}
function getTup(address addr) public view returns (tup) {
    for (uint i=0; i<tups.length; i++) {
        if (tups[i].owner == addr) {
            return tups[i];
        }
    }
}
```

STEP2：部署合约 dapp2.sol，要求可以调用 dapp1 的 getString 方法

```
contract Dapp1 {
    struct tup {
        uint amount;
```

(下页继续)

(续上页)

```
        string desc;
    }
    function getUint() public view returns (uint);
    function getString() public view returns (string);
    function getTup(address addr) public view returns (tup);
}
contract Dapp2 {
    struct tup {
        uint amount;
        string desc;
    }
    function Dapp2(address dapp1addr) public {
        Dapp1 dp1 = Dapp1(addr1);
    }
    function getString(address addr1) public view returns (string) {
        return dp1.getString();
    }
}
```

调用示例：

```
> nonce = 1   // 调用 ScsRPCMethod.GetNonce 获得
> subchainaddr = '0xb877bf4e4cc94fd9168313e00047b77217760930';
> dappaddr = '0xcc0D18E77748AeBe3cC6462be0EF724e391a4aD9';
> via = '0xf103bc1c054babcecd13e7ac1cf34f029647b08c';
> data = dappaddr + '89ea642f';
> chain3.personal.unlockAccount(chain3.mc.accounts[0], '123456');
> chain3.mc.sendTransaction( { nonce: nonce, from: chain3.mc.accounts[0], value:0, to:␣
↪subchainaddr, gas:0, shardingFlag:'0x1', data: data, via: via,});
```

3.10.2 子链合约的升级（版本迭代）

本实例完成一个合约获取另外一个合约中的数据，进而替代另外一个合约中的功能。

STEP0：部署 dappbase.sol

STEP1：部署合约 dapp1.sol，并注册到 dappbase 中去

```
contract Dapp1 {
    struct tup {
        uint amount;
```

(下页继续)

(续上页)

```
        address addr;
    }
    tup[] public tups;
    tup[] private mytups;

    function addTup(uint amount, address addr) public {
        tups.push(tup(amount, addr));
    }

    function getAmountByAddr(address myaddr) public view returns (tup[]) {
        for (uint i=0;i<tups.length;i++){
            if (tups[i].addr == myaddr){
                mytups.push(tups[i]);
            }
        }
        return mytups;
    }
}
```

STEP2：部署合约 dapp3.sol，要求可以访问 dapp1 中的数据

```
contract Dapp1 {
    struct tup {
        uint amount;
        address addr;
    }
    function getAmountByAddr(address) public view returns (tup[]);
}

contract Dapp3 {
    struct tup {
        uint amount;
        address addr;
    }

    Dapp1 public dp1;
    tup[] public tups;
    tup[] private mytups;
    function Dapp3(address dapp1addr) public{
        dp1 = Dapp1(dapp1addr);
```

(下页继续)

(续上页)

```
    }

    function addTup(uint amount, address addr) public {
        tups.push(tup(amount, addr));
    }

    function getAmountByAddr(address addr) public view returns (tup[]) {
        Dapp1.tup[] memory oldtups = new Dapp1.tup[](10);
        oldtups = dp1.getAmountByAddr(addr);
        for (uint i=0;i<oldtups.length;i++){
            if (oldtups[i].addr == addr){
                mytups.push(tup(oldtups[i].amount, oldtups[i].addr));
            }
        }

        for (i=0;i<tups.length;i++){
            if (tups[i].addr == addr){
                mytups.push(tups[i]);
            }
        }
        return mytups;
    }
}
```

说明：getAmountByAddr 方法将 dapp1 中的老数据放入 dapp3 中的 mytups。

STEP3：部署合约 dapp4.sol，要求可以访问 dapp3 中的数据

```
contract Dapp3 {
        struct tup {
            uint amount;
            address addr;
        }
        function getUint() public view returns (uint);
        function getString() public view returns (string);
        function getAmountByAddr(address) public view returns (tup[]);
}

contract Dapp4 {
    struct tup {
            uint amount;
```

(下页继续)

(续上页)

```
        address addr;
    }

    tup[] public tups;
    Dapp3 dp3;
    tup[] private mytups;
    function Dapp4(address dapp3addr)  public {
        dp3 = Dapp3(dapp3addr);
    }

    function addTup(uint amount, address addr) public {
        tups.push(tup(amount, addr));
    }

    function getAmountByAddr(address addr) public view returns (tup[]) {
        //get dapp3 data
        Dapp3.tup[] memory oldtups = new Dapp3.tup[](10);
        oldtups = dp3.getAmountByAddr(addr);
        for (uint i=0;i<oldtups.length;i++){
            if (oldtups[i].addr == addr){
                mytups.push(tup(oldtups[i].amount, oldtups[i].addr));
            }
        }
        //new data
        for (i=0;i<tups.length;i++){
            if (tups[i].addr == addr){
                mytups.push(tups[i]);
            }
        }
        return mytups;
    }
}
```

说明：getAmountByAddr 方法将 dapp3 中的老数据放入 dapp3 中的 mytups，因为 dapp3 的方法中包含 dapp1 的数据，所以这个方法最终返回 dapp1，dapp3，dapp4 中所有符合 addr 的数据。

CHAPTER 4

Restful API

4.1 Restful API 简介

MOAC 以 Restful API 提供给用户的一种接入方式。包括了对钱包，主链，子链，各交易查询的一系列封装。DAPP 用户当调用客户端 SDK 有困难时，可以通过服务端调用的方式实现对 MOAC 的接入。

4.1.1 URL 设计

使用 http 作为 API 的通信协议，目前采用较多的 POST 方法。

url 格式: http(s)://server.com/api/{module}/{version}/{method}

关于 version，各模块支持多版本，默认 version 为 v1.0

POST 提交格式采用 form 表单参数 Content-Type: application/x-www-form-urlencoded (a=name&b=666)

4.1.2 结构返回

返回体格式采用 json 格式

```
{
        "success": true,
        "message": "",
        "data": "*********"
```

(下页继续)

(续上页)

```
}
success:  true - 成功    false - 失败
message:  失败时的错误信息
data:      接口返回数据
```

相关状态码 200 OK 403 token 权限受限

4.1.3 接口访问控制

API 访问控制，需先请求访问的用户名和密码，先调用 auth 接口请求访问 token。

各 api 接口需要 token 参数调用，不然会返回 403 错误。

token 有过期机制，目前是 2 小时有效。

4.1.4 私钥安全

对于账号私钥的安全提供了两种方案。

1. dapp 注册账户后，收到返回的私钥（privatekey），后续发送交易直接传递私钥
2. dapp 注册账户后，也会返回一个加密串（encode），后续发送交易传递加密串和账户密码，系统会解码获得私钥进行签名。

4.1.5 节点控制

测试环境地址：http://139.198.126.104:8080

测试环境获取 access token，可使用测试账号：test 密码：123456

正式环境地址：https://api.moac.io:8080

正式环境获取 access token，请联系开发团队：moacapi@mossglobal.net。

目前设计需要 vnode 节点的相关 API，可通过参数传入地址和端口的方式指定连接的节点。参数传入为空的情况下，会使用平台默认的节点信息（测试环境对应 testnet 的默认节点，正式环境默认主网节点）。

4.2 Restful API 接口

4.2.1 管理模块

API 认证

请求访问 token，提供权限调用 API 的其他接口

方法：auth

参数:

```
account:  授权账号
pwd:  授权账号密码
```

调用示例：

```
POST: http://139.198.126.104:8080/auth
BODY: account=******&pwd=******
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": "token 内容"
}
```

4.2.2 账户模块

账户注册

方法：register

参数:

```
pwd:  账户密码
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/account/v1.0/register
BODY: pwd=********&token=********************************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "address": 账户地址,
        "encode": 账户加密串,
        "keystore": 账户 keystore 信息,
```

(下页继续)

(续上页)

```
        "privateKey": 账户私钥
}
```

账户登录

方法：login

参数:

```
address:  账户地址
pwd:  账户密码
encode:  账户加密串
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/account/v1.0/login
BODY: address=0x********&pwd=*****&encode=*******&token=************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 账户地址
}
```

账户导入

方法：import 将账户通过 keystore 导入系统

参数:

```
address:  账户地址
pwd:  账户密码
keystore:  账户 keystore
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/account/v1.0/import
BODY: address=0x********&pwd=*****&keystore={*******}&token=************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "address": 账户地址,
        "encode": 账户加密串,
        "privateKey": 账户私钥
}
```

4.2.3 主网模块

账户余额

方法：getBalance

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
address:  账号地址
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/getBalance
BODY: vnodeip=127.0.0.1&vnodeport=8545&address=0x******&token=****************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 账户余额 (单位 moac)
}
```

区块高度

方法：getBlockNumber

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/getBlockNumber
BODY: vnodeip=127.0.0.1&vnodeport=8545&token=***************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 区块高度
}
```

区块信息

方法: getBlockInfo

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
block:  区块号或者区块 hash
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/getBlockInfo
BODY: vnodeip=127.0.0.1&vnodeport=8545&block=2002326&token=******************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 区块信息
}
```

交易明细

方法：getTransactionByHash

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
hash:  交易 hash
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/getTransactionByHash
BODY: vnodeip=127.0.0.1&vnodeport=8545&hash=0x**&token=******************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易明细
}
```

交易详情

方法：getTransactionReceiptByHash

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
hash:  交易 hash
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/getTransactionReceiptByHash
BODY: vnodeip=127.0.0.1&vnodeport=8545&hash=0x**&token=******************
```

返回数据示例

```
{
        "success": true,
```

(下页继续)

(续上页)

```
        "message": "",
        "data": 交易详情
}
```

转账

方法：sendRawTransaction

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
from:  源账号地址
to:  目标账号地址
amount:  数量 (单位 moac)
method:  dapp 合约方法 比如：buyMintToken(uint256)
paramtypes:  dapp 合约方法对应的参数类型 比如：["uint256"]
paramvalues:  dapp 合约方法对应的参数值   比如：[100000000]
privatekey:  源账号私钥 (传 privatekey，可忽略参数 pwd 和 encode，不传 privatekey，则必须传
pwd 和 encode 认证)
pwd:  账户密码
encode：账户加密串
gasprice：可选参数，默认 gasprice 为 chain3 的 gasPrice，当交易堵塞时，需要传原交易的 110%
进行覆盖。
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/sendRawTransaction
BODY: vnodeip=127.0.0.1&vnodeport=8545&from=0x**&to=0x***&amount=10&
↪method=buyMintToken(uint256)&paramtypes=["uint256"]&paramvalues=[100000000]&
↪privatekey=0x**&token=*******
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

调用智能合约

方法：callContract

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
contractaddress:  合约地址
method:  dapp 合约方法 比如：buyMintToken(uint256)
paramtypes:  dapp 合约方法对应的参数类型 比如：["uint256"]
paramvalues:  dapp 合约方法对应的参数值   比如：[100000000]
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/callContract
BODY: vnodeip=127.0.0.1&vnodeport=8545&contractaddress=0x*****&
↪method=buyMintToken(uint256)&paramtypes=["uint256"]&paramvalues=[100000000]0x****&
↪token***************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 调用合约返回结果
}
```

erc20 转账

方法：transferErc

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
from:  源账号地址
to:  目标账号地址
contractaddress:  erc20 合约地址
amount:  erc20 代币数量
privatekey:  源账号私钥（传 privatekey，可忽略参数 pwd 和 encode，不传 privatekey，则必须传
pwd 和 encode 认证）
```

（下页继续）

(续上页)

```
pwd:  账户密码
encode: 账户加密串
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/transferErc
BODY: vnodeip=&vnodeport=&from=0x**&to=0x**&contractaddress=0x**&amount=10&
↪privatekey=0x**&token=*******
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

erc20 余额

方法: getErcBalance

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
address:  账户地址
contractaddress:  erc20 合约地址
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/getErcBalance
BODY: vnodeip=127.0.0.1&vnodeport=8545&address=0x*****&contractaddress=0x**&
↪token=*********
```

返回数据示例

```
{
        "success": true,
        "message": "",
```

(下页继续)

(续上页)

```
        "data": 余额（最小精度，10 进制）
}
```

erc20 授权给子链

方法：ercApprove

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
address:  账户地址
amount:  授权 erc20 数量
privatekey:  账号私钥（传 privatekey，可忽略参数 pwd 和 encode，不传 privatekey，则必须传
pwd 和 encode 认证）
pwd:  账户密码
encode：账户加密串
microchainaddress                          子链地址
contractaddress:  erc20 合约地址
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/ercApprove
BODY: vnodeip=127.0.0.1&vnodeport=8545&address=0x*****&amount=***&privatekey=0x***&
↪microchainaddress=0x***&contractaddress=0x**&token=*********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

充值子链 erc20 兑换子链原生币

方法：buyErcMintToken 注：前提是 erc20 对应数量已经授权给子链

参数:

```
vnodeip: vnode 节点地址
vnodeport: vnode 节点端口
address: 账户地址
privatekey: 源账号私钥 (传 privatekey, 可忽略参数 pwd 和 encode, 不传 privatekey, 则必须传
pwd 和 encode 认证)
pwd: 账户密码
encode: 账户加密串
microchainaddress: 子链地址
method: dapp 合约方法 默认为: buyMintToken(uint256)
paramtypes: dapp 合约方法对应的参数类型 默认为: ["uint256"]
paramvalues: dapp 合约方法对应的参数值 比如: [100000000]
token: auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/buyErcMintToken
BODY: vnodeip=&vnodeport=&address=0x**&privatekey=0x**&microchainaddress=0x**&
↪method=buyMintToken(uint256)&paramtypes=["uint256"]&paramvalues=[100000000]&token=****
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

充值子链 moac 兑换子链原生币

方法: buyMoacMintToken

参数:

```
vnodeip: vnode 节点地址
vnodeport: vnode 节点端口
address: 账户地址
privatekey: 源账号私钥
pwd: 账户密码
encode: 账户加密串
microChainaddress: 子链地址
method: dapp 合约方法 默认为: buyMintToken(uint256)
```

(下页继续)

(续上页)

```
paramtypes:  dapp 合约方法对应的参数类型 默认为: ["uint256"]
paramvalues:  dapp 合约方法对应的参数值   比如: [100000000]
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/vnode/v1.0/buyMoacMintToken
BODY: vnodeip=&vnodeport=&address=0x**&privatekey=0x**&microChainaddress=0x**&
↪method=buyMintToken(uint256)&paramtypes=["uint256"]&paramvalues=[100000000]&token=****
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

4.2.4 子链模块

获得子链区块高度

方法: getBlockNumber

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/getBlockNumber
BODY: microip=127.0.0.1&microport=8546&microchainaddress=0x***&token=***********
```

返回数据示例

```
{
        "success": true,
        "message": "",
```

(下页继续)

(续上页)

```
        "data": 子链区块高度
}
```

获得子链 dapp 地址列表

方法：getDappAddrList

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/micro/v1.0/getDappAddrList
BODY: microip=127.0.0.1&microport=8546&microchainaddress=0x***&token=***********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 子链 dapp 地址列表（按合约注册次序）
}
```

获取子链区块信息

方法：getBlock

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
blocknum:  块号
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/micro/v1.0/getBlock
BODY: microip=127.0.0.1&microport=8546&microchainaddress=0x***&blocknum=*****&
↪token=***********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 子链区块信息
}
```

获得子链对应 Hash 的交易信息

方法：getTransactionByHash

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
hash:  交易 hash
token:  auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/micro/v1.0/getTransactionByHash
BODY: microip=127.0.0.1&microport=8546&microchainaddress=0x***&hash=0x**&
↪token=***********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 子链交易信息
}
```

获得子链对应 Hash 的交易明细

方法：getTransactionReceiptByHash

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
hash:  交易 hash
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/getTransactionReceiptByHash
BODY: microip=127.0.0.1&microport=8546&microchainaddress=0x***&hash=0x**&
↪token=***********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 子链交易明细，其中主要字段描述如下:
                failed: 交易是否成功  false 表示成功
                        result: 如执行合约方法，retrun 的数据
                        transactionHash: 子链 hash
                        contractAddress: 当部署合约时，返回合约地址

}
```

获取子链账户余额

方法: getBalance

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
address:  账户地址
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/getBalance
BODY: vnodeip=&vnodeport=&microip=127.0.0.1&microport=8546&microchainaddress=0x*****&
↪address=0x*****&token=**************
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 账户余额
}
```

子链原生币转账

方法：transferCoin

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
via:  子链收益账号
from:  源账户地址
to:  目标账户地址
amount:  原生币数量
privatekey:  源账号私钥（传 privatekey，可忽略参数 pwd 和 encode，不传 privatekey，则必须传
pwd 和 encode 认证）
pwd:  账户密码
encode: 账户加密串
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/transferCoin
BODY: vnodeip=&vnodeport=&microip=127.0.0.1&microport=8546&microchainaddress=0x**&
↪via=0x**&from=0x**&to=0x**&amount=**&privatekey=0x***&token=*****
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

子链加签交易

方法：sendRawTransaction 调用 dapp 合约涉及修改数据的方法

参数:

```
vnodeip: vnode 节点地址
vnodeport:  vnode 节点端口
microip:  monitor 节点地址
microport:  monitor 节点端口
from: 发送交易账户地址
microchainaddress:  子链 SubChain 地址
via:  子链收益账号
amount:  payable 对应金额
dappaddress:  dapp 合约地址
method:  dapp 合约方法 比如：buyMintToken(uint256)
paramtypes:  dapp 合约方法对应的参数类型 比如：["uint256"]
paramvalues:  dapp 合约方法对应的参数值   比如：[100000000]
privatekey: 源账号私钥（传 privatekey，可忽略参数 pwd 和 encode，不传 privatekey，则必须传
pwd 和 encode 认证）
pwd:  账户密码
encode：账户加密串
token: auth 返回的授权 token
```

调用示例：

```
POST: http://139.198.126.104:8080/api/micro/v1.0/sendRawTransaction
BODY: vnodeip=&vnodeport=&microip=127.0.0.1&microport=8546&from=0x**&
↪microchainaddress=0x***&via=0x**&amount=**&dappaddress=0x***&
↪method=buyMintToken(uint256)&paramtypes=["uint256"]&paramvalues=[100000000]&
↪privatekey=0x***&token=*****
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 子链交易 hash
}
```

子链合约调用

方法：callContract 针对 public 方法和变量，不涉及数据修改

参数:

```
microip:  monitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
dappaddress:  dapp 合约地址
data:  字符串数组，如合约方法 getTopicList(uint pageNum, uint pageSize)，则传入 [
↪"getTopicList", "0", "20"]
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/callContract
BODY: vnodeip=&vnodeport=&microip=127.0.0.1&microport=8546&microchainaddress=0x*****&
↪dappaddress=0x**&data=&token=********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 合约返回结果
}
```

子链 ERC 提币

方法：redeemErcMintToken 原生币转 erc20

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
microipHmonitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
dappbaseaddress:  dappbase 合约地址
via:  子链收益账号
address:  提币账户地址
amount:  提取原生币数量
privatekey:  源账号私钥（传 privatekey，可忽略参数 pwd 和 encode，不传 privatekey，则必须传
pwd 和 encode 认证）
pwd:  账户密码
```

(下页继续)

(续上页)

```
encode: 账户加密串
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/redeemErcMintToken
BODY: vnodeip=&vnodeport=&microip=127.0.0.1&microport=8546&microchainaddress=0x**&
↪dappbaseaddress=0x**&via=0x**&address=0x**&amount=**&data=****&privatekey=0x**&
↪token=********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

子链 MOAC 提币

方法: redeemMoacMintToken 原生币转 moac

参数:

```
vnodeip:  vnode 节点地址
vnodeport:  vnode 节点端口
microipHmonitor 节点地址
microport:  monitor 节点端口
microchainaddress:  子链 SubChain 地址
dappbaseaddress:  dappbase 合约地址
via:  子链收益账号
address:  提币账户地址
amount:  提取原生币数量
privatekey:  源账号私钥 (传 privatekey, 可忽略参数 pwd 和 encode, 不传 privatekey, 则必须传
pwd 和 encode 认证)
pwd:  账户密码
encode: 账户加密串
token:  auth 返回的授权 token
```

调用示例:

```
POST: http://139.198.126.104:8080/api/micro/v1.0/redeemMoacMintToken
BODY: vnodeip=&vnodeport=&microip=127.0.0.1&microport=8546&microchainaddress=0x**&
↪dappbaseaddress=0x**&via=0x**&address=0x**&amount=**&data=****&privatekey=0x**&
↪token=********
```

返回数据示例

```
{
        "success": true,
        "message": "",
        "data": 交易 hash
}
```

CHAPTER 5

SDK

5.1 SDK 简介

为了方便用户接入，MOAC 官方提供 nodejs 版本的 SDK，官方暂不考虑提供其他版本的 SDK。

Node.JS SDK 下载安装

```
npm install moac-api
```

Node.JS SDK 异常处理说明：

应用方根据自己业务逻辑对 sdk 方法进行 try catch 异常处理

示例：

```
var VnodeChain = require("moac-api").vnodeChain;

try{
        var vc = new VnodeChain("http://47.106.69.61:8989");
        var blockNumber = vc.getBlockNumber();
        console.log(blockNumber);
}catch (e){
        console.log(e);
}
```

5.2 SDK 接口

5.2.1 钱包模块

注册

Node.JS Example

参数:

```
pwd: 钱包账户密码
```

代码:

```
var account = require("moac-api").account;
var wallet = account.register(pwd);
```

返回:

```
wallet:
{ address: ' 钱包地址....',
  privateKey: ' 私钥....',
  keyStore: 'keyStore 内容...'
}
```

登录

Node.JS Example

参数:

```
addr: 钱包地址
pwd: 钱包密码
keyStore: keyStore
```

代码:

```
var account = require("moac-api").account;
var status = account.login(addr, pwd, keyStore);
```

返回:

```
status: 0 //登录失败
status: 1 //登录成功
status: 2 //密码错误
```

5.2.2 主链模块

实例化主链对象

Node.JS Example

参数:

```
vnodeAddress: 主链访问地址 //http://47.106.69.61:8989
```

代码:

```
var VnodeChain = require("moac-api").vnodeChain;
var vc = new VnodeChain(vnodeAddress);
```

获取主链区块高度

Node.JS Example

代码:

```
var blockNumber = vc.getBlockNumber();
```

返回:

```
blockNumber: 主链区块高度
```

获取主链某一区块信息

Node.JS Example

参数:

```
hashOrNumber: 区块 hash 或区块高度
```

代码:

```
var blockInfo = vc.getBlockInfo(hashOrNumber);
```

返回:

```
blockInfo: 某一区块信息
```

获取主链交易详情

Node.JS Example

参数:

```
hash: 交易 hash
```

代码:

```
var tradeInfo = vc.getTransactionByHash(hash);
```

返回:

```
tradeInfo: 交易详情
```

获取合约实例

Node.JS Example

参数:

```
microChainAddress: 子链地址
versionKey: 版本号 (默认 0.1 版本)
```

代码:

```
var data = vc.getSubChainBaseInstance(microChainAddress, versionKey);
```

返回:

```
data: 合约实例
```

获取主链账户余额

Node.JS Example

参数:

```
addr: 钱包账户地址
```

代码:

```
var balance = vc.getBalance(addr);
```

返回:

```
balance: 主链账户余额 (单位为 moac)
```

获取主链账户 ERC 代币余额

Node.JS Example

参数:

```
addr: 钱包账户地址
contractAddress: 合约地址
```

代码:

```
var balance = vc.getErcBalance(addr, contractAddress);
```

返回:

```
balance: 账户 ERC 代币余额 (erc20 最小单位)
```

获取主链合约实例

Node.JS Example

参数:

```
abiObj: abi 对象
contractAddress: 合约地址
```

代码:

```
var object = vc.getContractInstance(abiObj, contractAddress);
```

返回:

```
object: 主链合约实例对象
```

获取交易 Data

参数:

```
method: 方法 例 "issue(address,uint256)"
paramTypes: paramTypes 参数类型数组 例 ['address','uint256']
paramValues: paramValues 参数值数组 例 ['0x.....',10000] (如需要传金额的入参为 erc20 最小单位)
```

代码:

```
var data = mc.getData(method,paramTypes,paramValues);
```

返回:

```
data: data 字符串
```

主链加签交易

Node.JS Example

参数:

```
from: 交易发送人
to: 交易接受者 (可以为个人地址, 或者主链上的合约地址)
amount: 交易金额
method: 方法 例 "issue(address,uint256)"
paramTypes: paramTypes 参数类型数组 例 ['address','uint256']
paramValues: paramValues 参数值数组 例 ['0x.....',10000] (如需要传金额的入参为 erc20 最小单位)
privateKey: 交易发起人私钥字符串
gasPrice: gas 费用 (默认为 0, 如返回错误为 gas 过低, 请在返回的 gas 基础上加上整数 gas 重新提交)
```

代码:

```
vc.sendRawTransaction(from, to, amount, method, paramTypes, paramValues, privateKey,␣
↪gasPrice).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

主链 MOAC 转账

参数:

```
from: 转账人地址
to: 收款人地址
amount: 交易金额 (单位为 moac)
privatekey: 转账人私钥
```

代码:

```
vc.transferMoac(from, to, amount, privatekey).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

主链 ERC 代币转账

参数:

```
from: 转账人地址
to: 收款人地址
contractAddress: erc 代币合约地址
amount: 交易金额 (单位为 moac)
privateKey: 转账人私钥
```

代码:

```
vc.transferErc(from, to, contractAddress, amount, privateKey).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

调用主链合约

参数:

```
method: 方法 例 "issue(address,uint256)"
paramTypes: paramTypes 参数类型数组 例 ['address','uint256']
paramValues: paramValues 参数值数组 例 ['0x.....',10000] (如需要传金额的入参为 erc20 最小单位)
contractAddress: 合约地址
```

代码:

```
var callRes = vc.callContract(method, paramTypes, paramValues, contractAddress);
```

返回:

```
callRes: 调用合约返回信息
```

ERC20 充值

参数:

```
addr: 钱包地址
privateKey: 钱包私钥
microChainAddress: 子链地址
method: 方法 "issue(address,uint256)"
paramTypes: paramTypes 参数类型数组 ['address','uint256']
paramValues: paramValues 参数值数组 ['0x.....',10000] (需要传金额的入参为 erc20 最小单位)
```

代码:

```
vc.buyErcMintToken(addr, privateKey, microChainAddress, method, paramTypes, paramValues).
↪then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

MOAC 充值

参数:

```
addr: 钱包地址
privateKey: 钱包私钥
microChainAddress: 子链地址
method: 方法 "issue(address,uint256)"
paramTypes: paramTypes 参数类型数组 ['address','uint256']
paramValues: paramValues 参数值数组 ['0x.....',10000] (金额单位为 moac)
```

代码:

```
vc.buyMoacMintToken(addr, privateKey, microChainAddress, method, paramTypes,␣
↪paramValues).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

5.2.3 子链模块

实例化子链对象

Node.JS Example

参数:

```
vnodeAddress: 主链访问地址 //http://47.106.69.61:8989
monitorAddress: 子链访问地址 //http://47.106.89.22:8546
microChainAddress: 子链地址
via: 子链 via
```

代码:

```
var MicroChain = require("moac-api").microChain;
var mc = new MicroChain(vnodeAddress, monitorAddress, microChainAddress, via);
```

获取子链区块高度

Node.JS Example

代码:

```
mc.getBlockNumber().then((blockNumber) => {
        console.log(blockNumber);
});
```

返回:

```
blockNumber: 子链区块高度
```

获取某一区间内的多个区块信息

Node.JS Example

参数:

```
start: 开始高度
end: 结束高度
```

代码:

```
mc.getBlocks(start, end).then((blockListInfo) => {
        console.log(blockListInfo);
});
```

返回:

```
blockListInfo: 区块信息 List
```

获取子链某一区块信息

Node.JS Example

参数:

```
blockNumber: 区块高度
```

代码:

```
mc.getBlock(blockNumber).then((blockInfo) => {
        console.log(blockInfo);
});
```

返回:

```
blockInfo: 某一区块信息
```

获取子链交易详情

Node.JS Example

参数:

```
transactionHash: 交易 hash
```

代码:

```
mc.getTransactionByHash(transactionHash).then((transactionInfo) => {
        console.log(transactionInfo);
});
```

返回:

```
transactionInfo: 交易详情
```

获取子链账户余额

Node.JS Example

参数:

```
addr: 钱包地址
```

代码:

```
mc.getBalance(addr).then((balance) => {
        console.log(balance);
});
```

返回:

```
data: 子链账户余额 (erc20 最小单位)
```

获取子链详细信息

Node.JS Example

代码:

```
mc.getMicroChainInfo().then((microChainInfo) => {
        console.log(microChainInfo);
});;
```

返回:

```
microChainInfo: 子链信息
```

获取子链 DAPP 状态

Node.JS Example

代码:

```
mc.getDappState().then((state) => {
        console.log(state);
});;
```

返回:

```
state: 1//正常
state: 0//异常
```

获取 Nonce

Node.JS Example

参数:

```
addr: 账户钱包地址
```

代码:

```
mc.getNonce(addr).then((nonce) => {
        console.log(nonce);
});;
```

返回:

```
nonce: 得到的 nonce
```

获取子链 DAPP 合约实例

参数:

```
dappContractAddress: dapp 合约地址
dappAbi: dapp 合约的 Abi 对象
```

代码:

```
var dapp = getDappInstance(dappContractAddress, dappAbi);
```

返回:

```
dapp: dapp 实例
```

获取交易 Data

参数:

```
method: 方法 例 "issue(address,uint256)"
paramTypes: paramTypes 参数类型数组 例 ['address','uint256']
paramValues: paramValues 参数值数组 例 ['0x.....',10000] (如需要传金额的入参为 erc20 最小单位)
```

代码:

```
var data = mc.getData(method,paramTypes,paramValues);
```

返回:

```
data: data 字符串
```

子链加签交易

Node.JS Example

参数:

```
from: 发送方的钱包地址
microChainAddress: 子链地址
amount: 交易金额
dappAddress: dapp 地址
method: 方法 例 "issue(address,uint256)"
```

(下页继续)

(续上页)

```
paramTypes: paramTypes 参数类型数组 例 ['address','uint256']
paramValues: paramValues 参数值数组 例 ['0x.....',10000] (如需要传金额的入参为 erc20 最小单位)
privateKey: 发送方钱包私钥
```

代码:

```
mc.sendRawTransaction(from, microChainAddress, amount, dappAddress, method, paramTypes,␣
↪paramValues, privateKey).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

子链转账

Node.JS Example

参数:

```
from: 发送方的钱包地址
to: 接收方的钱包地址
amount: 交易金额 (erc20 最小单位)
privateKey: 钱包私钥
```

代码:

```
mc.transferCoin(from, to, amount, privateKey).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

调用子链合约

参数:

```
contractAddress: dapp 合约地址
param: 例如合约中存在一个无参的方法 getDechatInfo, 则传入 ["getDechatInfo"];
     存在一个有参的方法 getTopicList(uint pageNum, uint pageSize), 则传入 ["getTopicList",
↪ 0, 20]
```

代码:

```
mc.callContract(contractAddress, param).then((data) => {
        console.log(data);
});
```

返回:

```
data: 调用合约返回信息
```

提币（MOAC）

参数:

```
addr: 钱包地址
amount: 金额 (单位为 moac)
privateKey: 钱包私钥
```

代码:

```
mc.redeemMoacMintToken(addr, amount, privateKey).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

提币（ERC20）

参数:

```
addr: 钱包地址
amount: 金额 (erc20 最小单位)
privateKey: 钱包私钥
```

代码:

```
mc.redeemErcMintToken(addr, amount,privateKey).then((hash) => {
        console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

CHAPTER 6

DAPPs

6.1 去中心化应用（DAPPs）

墨客平台上的去中心化应用 (DAPP) 即可以部署在母链，也可以单独部署一条子链。

6.1.1 母链 DAPP 应用

母链部署的 DAPPs 和以太坊的部署过程相同：

部署和使用 *ERC20* 通证

部署和使用 *ERC721* 通证

如果开发者想把基于以太坊的应用移植到墨客平台，只需做很小的改动，具体可以参考：

移植到墨客平台

6.1.2 子链 DApps 应用

DApp runs on MicroChain means it has its own blockchain that supports transactions, data access, control flow in a layered structure. It creates the framework to allow users to execute Smart Contract in an efficient way. It also provides the architecture to spawn sub blockchains using underlying infrastructure quickly and easily. It is a Blockchain platform with necessary plumbing parts available to sub blockchains, providing solution for idea test, private chain deployment, complex task processing, decentralized applications etc.

6.2 部署和使用 ERC20 通证

6.2.1 Creating your own ERC20 Token on the MOAC blockchain

Before you read this page, you should:

- Understand the following general concepts: blockchain, smart contracts, ERC20 tokens;
- Be able to write a basic Ethereum smart contract in Solidity (see tutorial example);

After you read this page, you will be able to:

- Write your own ERC20 token contract;
- Compile and deploy it on the MOAC blockchain;
- Use the ERC20 token;

6.2.2 Write the ERC20 token smart contract

The ERC20 standard is developed by the Ethereum community.

```
contract ERC20Token {
  string  public name;
  string  public symbol;
  uint    public decimals;
  uint256 public totalSupply;

  function balanceOf(address _owner) public constant returns (uint256 balance);
  function transfer(address _to, uint256 _value) public returns (bool success);
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool␣
↪success);
  function approve(address _spender, uint256 _value) public returns (bool success);
  function allowance(address _owner, address _spender) public constant returns (uint256␣
↪remaining);
  event Transfer(address indexed _from, address indexed _to, uint256 _value);
  event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

User can develop its own ERC20 token. For that, we can either use a ready open source library to help with the task, or write the needed smart contracts directly from scratch. We will show both methods below so you can master your ERC20 token development skills.

Using a ready open source library

Let's see how you can implement this interface by taking advantage of the OpenZeppelin Solidity library, for instance. We will write a TestCoin based on, for example, the PausableToken contract in OpenZeppelin:

```
pragma solidity ^0.4.18;

import "zeppelin-solidity/contracts/token/PausableToken.sol";

contract TestCoin is PausableToken {
  string public name = "Test";
  string public symbol = "TEST";
  uint public decimals = 6;
  uint public INITIAL_SUPPLY = 100000000 * (10 ** decimals);

  function TestCoin() public {
    totalSupply = INITIAL_SUPPLY;
    balances[msg.sender] = INITIAL_SUPPLY;
  }
}
```

To understand better all the steps that the inheritance from PausableToken has achieved in making our TestCoin a useful ERC20-compliant token (and why it can be helpful to start from an open source library like OpenZeppelin), you can follow closely the code of each of the library's files that were imported after each inheritance and import is fully expanded.

For that, browsing the OpenZeppelin ERC20 token github repo will be very helpful: our TestCoin is inheriting from PausableToken, which itself inherits from StandardToken, which inherits from BasicToken and ERC20, and so on all the way to ERC20Basic and the SafeMath library import.

Developing your smart contract directly

Another way of doing this is to simply write the ERC20 interface and your contract implementing it directly from scratch. As you'll see, this is actually a fairly straightforward task:

```
pragma solidity ^0.4.16;

contract ERC20Token {
  uint256 public totalSupply;

  function balanceOf(address _owner) public constant returns (uint256 balance);
  function transfer(address _to, uint256 _value) public returns (bool success);
```

(下页继续)

(续上页)

```
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool␣
↪success);

  function approve(address _spender, uint256 _value) public returns (bool success);

  function allowance(address _owner, address _spender) public constant returns (uint256␣
↪remaining);

  event Transfer(address indexed _from, address indexed _to, uint256 _value);
  event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}

contract TestToken is ERC20Token {

  string  public name = "Test Coin";
  string  public symbol = "TEST";
  uint8   public decimals = 6;
  uint256 public INITIAL_SUPPLY = 100000000 * (10 ** uint256(decimals));

  mapping (address => uint256) balances;
  mapping (address => mapping (address => uint256)) allowed;

  function TestToken() {
    totalSupply = INITIAL_SUPPLY;
    balances[msg.sender] = INITIAL_SUPPLY;
  }

  function transfer(address _to, uint256 _value) public returns (bool success) {
    require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]);
    require(_to != 0x0);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  function transferFrom(address _from, address _to, uint256 _value) public returns (bool␣
↪success) {
    require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
    balances[_to] += _value;
```

(下页继续)

(续上页)

```
    balances[_from] -= _value;
    allowed[_from][msg.sender] -= _value;
    emit Transfer(_from, _to, _value);
    return true;
  }

  function balanceOf(address _owner) public constant returns (uint256 balance) {
    return balances[_owner];
  }

  function approve(address _spender, uint256 _value) public returns (bool success) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  function allowance(address _owner, address _spender) public constant returns (uint256␣
↪remaining) {
    return allowed[_owner][_spender];
  }
}
```

Make sure to save your TestToken contract to a file on your computer (e.g. TestToken.sol).

6.2.3 Compile and deploy ERC20 token smart contract

As a next step, you' ll need to generate the bytecode and ABI for your new smart contract. You can think of the bytecode as basically your contract' s compiled code. The ABI (Application Binary Interface) is a JavaScript Object that defines how to interact with your smart contract.

We will show three ways of achieving this as well - using MOAC wallet, Remix web IDE for Solidity development, or compiling using the solc compiler on your machine (i.e. the command line).

Using MOAC wallet

MOAC wallet is an online free, client-side interface for using MOAC wallets, make transactions and deploy contract. It was developed based on open-source software. To use the service, you need to run a local MOAC node with addition command line arguments:

```
--rpccorsdomain "http://wallet.moac.io"
```

This will allow the access of MOAC node using MOAC wallet.

Otherwise you will see the following error message:

图 1: ERC20_moacwallet01.png

Example command to start a MOAC node connecting with mainnet:

```
moac --rpccorsdomain "http://wallet.moac.io" --rpc --rpcport "8545" --rpcapi "chain3,mc,
net,db"
```

A successful interface connecting to mainnet looks like this:

图 2: ERC20_moacwallet02.png

To compile the contract, click the "CONTRACTS" icon:

图 3: ERC20_moacwallet03.png

Then enter the contract deploy page:

Copy the source code and paste in the "SOLIDITY CONTRACT SOURCE CODE" section. After copying the source code, the compiling process will automatically started. If no errors are not found, the right side should show a "SELECT CONTRACT TO DEPLOY" menu.

Select the contract name "TokenERC20" :

Input the parameters values from the menu:

You need to have enough balance to deploy the contract. You can choose the amount of fee to use when deploying the contract. Click the DEPLOY button:

This is the contract ready to send from Account 1. The Provide gas is estimated by the compiler and we suggest you use it or put a larger number. If gas is not enough, the contract cannot be created. To continue, be sure to unlock the account to create the contract. You can use a console attached to the MOAC to do this:

After deploying, the interface is returned to the main menu and you can see the following transaction is creating.

After 12 confirmations, you can start using the contract by click the admin page link.

MOAC wallet is good for beginners that only need basic contract development needs. It cannot debug contracts. To advanced developers, you can use Remix to work with contracts.

图 4: ERC20_moacwallet04.png

图 5: ERC20_moacwallet05.png

Remix

Remix is an online tool developed by Ethereum community to work with smart contracts. MOAC also supports the deploy of smart contract through Remix.

Open Remix on your browser, create a new file called 'TestToken.sol' and copy paste the code of your smart contract. Make sure you are including all the other Solidity files that your code is referencing with imports, especially if you are using the open source library approach.

Select 'TestToken' in the Compile window then click "Start to Compile" and the Details button next to TestToken. Upon scrolling in the popup details window for TestToken, you should be able to see similar sections to this Remix screenshot for the bytecode and ABI of your smart contract:

If the contract is compiled successfully, remix will show the interface like this:

To deploy the contract, you need to connect REMIX to a local or remote MOAC node. In addition to other arguments, be sure to enable the access of REMIX to the MOAC node with

```
moac --rpccorsdomain "http://remix.ethereum.org" --rpc --rpcport "8545" --rpcapi "chain3,
↪mc,net,db"
```

Click the Run Tab and you should see the following menu:

Choose the Environment menu: JavaScript VM is a simulated environment of Remix, it can be use to debugging the contract without actually deploying the contract to a real network. Injected Web3 is the default web3 connecting to Ethereum network. To deploy MOAC contract, you need to choose Web3 Provider.

After choose "Web3 Provider", you can see a message like this:

Click "OK",

You need to make sure the port is the same as the local running node.

You may see the error message like this:

If you see this error message, check the local node that include both

```
--rpccorsdomain "http://remix.ethereum.org"
```

图 6: ERC20_moacwallet06.png

图 7: ERC20_moacwallet07.png

图 8: ERC20_moacwallet08.png

图 9: ERC20_moacwallet09.png

图 10: ERC20_moacwallet10.png

图 11: ERC20_moacwallet11.png

图 12: ERC20_moacwallet12.png

图 13: ERC20_moacwallet13.png

图 14: ERC20_moacwallet14.png

图 15: ERC20_moacwallet15.png

图 16: ERC20_moacwallet12.png

图 17: ERC20_moacwallet17.png

and

```
--rpcport "8545"
```

If the connection is established, you should see your accounts from the Account List.

图 18: ERC20_moacwallet18.png

Before you deploy the contract, you need to unlock the account that send the contract. You can do the unlock with the MOAC console:

图 19: ERC20_moacwallet09.png

After successfully deployed the contract, you should see the contract address and other information showed in the menu:

图 20: ERC20_moacwallet19.png

Remix is good for developing and debugging smart contracts. It is not very convenient to deploy multiple contracts. If your requires to deploy multiple contracts, you can use the Node.Js packages.

Using the Node.Js packages

You need to install solc package to compile the smart contract, and chain3 package to deploy the contract.

To use the latest stable version of the Solidity compiler via Node.js you can install it via npm:

```
npm install solc
```

```
var solc = require('solc')
var input = 'contract x { function g() {} }'
// Setting 1 as second paramateractivates the optimiser
var output = solc.compile(input, 1)
for (var contractName in output.contracts) {
        // code and ABI that are needed by web3
        console.log(contractName + ': ' + output.contracts[contractName].bytecode)
        console.log(contractName + '; ' + JSON.parse(output.contracts[contractName].
↪interface))
}
```

To deploy the contracts, you need to install the Chain3 package:

```
npm install chain3
```

There is an example file in the package: example/contract_deploy.js

After successfully deploy, you should see the contract is displayed

Succeed!: 0x95d703ea48477f48335ae9c477ce6d986bc68453dfe3d6582714045456b93405

Using solc compiler to generate the ABI and bytecode Another way of generating these two files is to compile your smart contract using the solc compiler on your machine. If you haven' t used solc yet, you can follow these instructions for installing it on your machine.

Open a Terminal window and navigate to your working directory where you have saved your TestToken.sol file. Run the following command to export the 'TestToken.abi' and 'TestToken.bin' files to the bin directory:

```
solc --bin --abi -o bin TestToken.sol
```

As the file extensions suggest, 'TestToken.abi' contains your contract' s ABI, and 'TestToken.bin' contains its bytecode.

If you prefer accessing the solc compiler from within a program' s code to generate the ABI and bytecode files rather than using the command line, you can use the following code instead:

```
var fs =  require ( ' fs ' );
var solc =  requires ( 'solc' );

var cmds = process.argv;
if(cmds != null && cmds.length > 2){
  var file = cmds[2];
  var name = cmds[3];
  var content = fs.readFileSync(file).toString();

  was input = {
    file: content
  };

  var output = solc.compile({sources: input}, 1);
  console.log('contracts', Object.keys(output.contracts));

  var ctt = output.contracts[name];
  if(ctt == null){
      return;
  }
```

(下页继续)

(续上页)

```
  var bytecode = ctt.bytecode;
  var abi = JSON.parse(ctt.interface);

  console.log('bytecode', bytecode);
  console.log('abi', ctt.interface);
}
```

Regardless of which method you followed, you should now have the ABI and bytecode files for your TestToken smart contract. Next, you will be able to deploy your token contract on the MOAC blockchain for others to interact with it.

6.3 部署和使用 ERC721 通证

6.3.1 Creating your own ERC721 Token on MOAC blockchain

Before you read this page, you should:

- Understand the following general concepts: blockchain, smart contracts, ERC721 tokens;
- Be able to write a basic Ethereum smart contract in Solidity (see tutorial example);

After you read this page, you will be able to:

- Write your own ERC721 token contract;
- Compile and deploy it on the MOAC blockchain;
- Use the ERC721 token;

6.3.2 Write the ERC721 token smart contract

The ERC721 standard is developed by the Ethereum community. It is a standard interface for non-fungible tokens(NFT), also known as deeds.

```
contract ERC721 {
    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)␣
↪external payable;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external␣
↪payable;
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
```

(下页继续)

(续上页)

```
    function approve(address _approved, uint256 _tokenId) external payable;
    function setApprovalForAll(address _operator, bool _approved) external;
    function getApproved(uint256 _tokenId) external view returns (address);
    function isApprovedForAll(address _owner, address _operator) external view returns␣
↪(bool);
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
    event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _
↪approved);
}
```

User can develop its own ERC721 token. For that, we can either use a ready open source library to help with the task, or write the needed smart contracts directly from scratch. We will show all three methods below so you can master your ERC721 token development skills.

Using a ready open source library

Let' s see how you can implement this interface by taking advantage of the OpenZeppelin Solidity library, for instance. We will write a TestCoin based on, for example, the PausableToken contract in OpenZeppelin:

```
pragma solidity ^0.4.18;

import "./ERC721Basic.sol";
import "./ERC721Receiver.sol";
import "../../math/SafeMath.sol";
import "../../AddressUtils.sol";

contract ERC721BasicToken is ERC721Basic {
  using SafeMath for uint256;
  using AddressUtils for address;

  // Equals to `bytes4(keccak256("onERC721Received(address,uint256,bytes)"))`
  // which can be also obtained as `ERC721Receiver(0).onERC721Received.selector`
  bytes4 constant ERC721_RECEIVED = 0xf0b9e5ba;

  // Mapping from token ID to owner
  mapping (uint256 => address) internal tokenOwner;

  // Mapping from token ID to approved address
```

(下页继续)

(续上页)

```
  mapping (uint256 => address) internal tokenApprovals;

  // Mapping from owner to number of owned token
  mapping (address => uint256) internal ownedTokensCount;

  // Mapping from owner to operator approvals
  mapping (address => mapping (address => bool)) internal operatorApprovals;

  modifier onlyOwnerOf(uint256 _tokenId) {
    require(ownerOf(_tokenId) == msg.sender);
    _;
      }

modifier canTransfer(uint256 _tokenId) {
require(isApprovedOrOwner(msg.sender, _tokenId));
_;
}

function balanceOf(address _owner) public view returns (uint256) {
require(_owner != address(0));
return ownedTokensCount[_owner];
}

function ownerOf(uint256 _tokenId) public view returns (address) {
address owner = tokenOwner[_tokenId];
require(owner != address(0));
return owner;
}

function exists(uint256 _tokenId) public view returns (bool) {
address owner = tokenOwner[_tokenId];
return owner != address(0);
}

function approve(address _to, uint256 _tokenId) public {
address owner = ownerOf(_tokenId);
require(_to != owner);
require(msg.sender == owner || isApprovedForAll(owner, msg.sender));

if (getApproved(_tokenId) != address(0) || _to != address(0)) {
```

(下页继续)

(续上页)

```
  tokenApprovals[_tokenId] = _to;
  emit Approval(owner, _to, _tokenId);
    }
}

function getApproved(uint256 _tokenId) public view returns (address) {
return tokenApprovals[_tokenId];
}

function setApprovalForAll(address _to, bool _approved) public {
require(_to != msg.sender);
operatorApprovals[msg.sender][_to] = _approved;
emit ApprovalForAll(msg.sender, _to, _approved);
}

function isApprovedForAll(
address _owner,
address _operator
)
public
view
returns (bool)
{
return operatorApprovals[_owner][_operator];
}

function transferFrom(
address _from,
address _to,
uint256 _tokenId
)
public
canTransfer(_tokenId)
{
require(_from != address(0));
require(_to != address(0));

clearApproval(_from, _tokenId);
removeTokenFrom(_from, _tokenId);
addTokenTo(_to, _tokenId);
```

(下页继续)

(续上页)

```
    emit Transfer(_from, _to, _tokenId);
}

function safeTransferFrom(
address _from,
address _to,
uint256 _tokenId
)
public
canTransfer(_tokenId)
{
// solium-disable-next-line arg-overflow
safeTransferFrom(_from, _to, _tokenId, "");
}

function safeTransferFrom(
address _from,
address _to,
uint256 _tokenId,
bytes _data
)
public
canTransfer(_tokenId)
{
transferFrom(_from, _to, _tokenId);
// solium-disable-next-line arg-overflow
require(checkAndCallSafeTransfer(_from, _to, _tokenId, _data));
}

function isApprovedOrOwner(
address _spender,
uint256 _tokenId
)
internal
view
returns (bool)
{
address owner = ownerOf(_tokenId);
// Disable solium check because of
```

(下页继续)

(续上页)

```
// https://github.com/duaraghav8/Solium/issues/175
// solium-disable-next-line operator-whitespace
return (
  _spender == owner ||
  getApproved(_tokenId) == _spender ||
  isApprovedForAll(owner, _spender)
);
}

function _mint(address _to, uint256 _tokenId) internal {
require(_to != address(0));
addTokenTo(_to, _tokenId);
emit Transfer(address(0), _to, _tokenId);
}

function _burn(address _owner, uint256 _tokenId) internal {
clearApproval(_owner, _tokenId);
removeTokenFrom(_owner, _tokenId);
emit Transfer(_owner, address(0), _tokenId);
}

function clearApproval(address _owner, uint256 _tokenId) internal {
require(ownerOf(_tokenId) == _owner);
if (tokenApprovals[_tokenId] != address(0)) {
  tokenApprovals[_tokenId] = address(0);
  emit Approval(_owner, address(0), _tokenId);
}
}

function addTokenTo(address _to, uint256 _tokenId) internal {
require(tokenOwner[_tokenId] == address(0));
tokenOwner[_tokenId] = _to;
ownedTokensCount[_to] = ownedTokensCount[_to].add(1);
}

function removeTokenFrom(address _from, uint256 _tokenId) internal {
require(ownerOf(_tokenId) == _from);
ownedTokensCount[_from] = ownedTokensCount[_from].sub(1);
tokenOwner[_tokenId] = address(0);
}
```

(下页继续)

(续上页)

```
function checkAndCallSafeTransfer(
address _from,
address _to,
uint256 _tokenId,
bytes _data
)
internal
returns (bool)
{
if (!_to.isContract()) {
  return true;
}
bytes4 retval = ERC721Receiver(_to).onERC721Received(
  _from, _tokenId, _data);
return (retval == ERC721_RECEIVED);
}
}
```

To understand better all the steps that the inheritance from PausableToken has achieved in making our TestCoin a useful ERC721-compliant token (and why it can be helpful to start from an open source library like OpenZeppelin), you can follow closely the code of each of the library' s files that were imported after each inheritance and import is fully expanded.

For that, browsing the OpenZeppelin ERC721 token github repo will be very helpful: our TestCoin is inheriting from PausableToken, which itself inherits from StandardToken, which inherits from BasicToken and ERC721, and so on all the way to ERC721Basic and the SafeMath library import.

Developing your smart contract directly

Another way of doing this is to simply write the ERC721 interface and your contract implementing it directly from scratch. As you' ll see, this is actually a fairly straightforward task:

```
pragma solidity ^0.4.16;

contract ERC721Token {
  uint256 public totalSupply;

  function balanceOf(address _owner) public constant returns (uint256 balance);
  function transfer(address _to, uint256 _value) public returns (bool success);
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool␣
↪success);
```

(下页继续)

(续上页)

```
  function approve(address _spender, uint256 _value) public returns (bool success);

  function allowance(address _owner, address _spender) public constant returns (uint256␣
↪remaining);

  event Transfer(address indexed _from, address indexed _to, uint256 _value);
  event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}

contract TestToken is ERC721Token {

  string  public name = "Test Coin";
  string  public symbol = "TEST";
  uint8   public decimals = 6;
  uint256 public INITIAL_SUPPLY = 100000000 * (10 ** uint256(decimals));

  mapping (address => uint256) balances;
  mapping (address => mapping (address => uint256)) allowed;

  function TestToken() {
    totalSupply = INITIAL_SUPPLY;
    balances[msg.sender] = INITIAL_SUPPLY;
  }

  function transfer(address _to, uint256 _value) public returns (bool success) {
    require(balances[msg.sender] >= _value && balances[_to] + _value > balances[_to]);
    require(_to != 0x0);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  function transferFrom(address _from, address _to, uint256 _value) public returns (bool␣
↪success) {
    require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value);
    balances[_to] += _value;
    balances[_from] -= _value;
    allowed[_from][msg.sender] -= _value;
```

(下页继续)

(续上页)

```
    emit Transfer(_from, _to, _value);
    return true;
  }

  function balanceOf(address _owner) public constant returns (uint256 balance) {
    return balances[_owner];
  }

  function approve(address _spender, uint256 _value) public returns (bool success) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

  function allowance(address _owner, address _spender) public constant returns (uint256␣
↪remaining) {
    return allowed[_owner][_spender];
  }
}
```

You can get this file directly from OpenZeppelin' s GitHub repo or copy paste the code above and edit it as needed. Make sure to save your contract to a local file on your computer (e.g. ERC721BasicToken.sol). However, the code is only for demonstration only.

6.3.3 Compile and deploy ERC721 token smart contract

As a next step, you' ll need to generate the bytecode and ABI for your new smart contract. You can think of the bytecode as basically your contract' s compiled code. The ABI (Application Binary Interface) is a JavaScript Object that defines how to interact with your smart contract.

We will show three ways of achieving this as well - using MOAC wallet, Remix web IDE for Solidity development, or compiling using the solc compiler on your machine (i.e. the command line).

Using MOAC wallet

MOAC wallet is an online free, client-side interface for using MOAC wallets, make transactions and deploy contract. It was developed based on open-source software. To use the service, you need to run a local MOAC node with addition command line arguments:

```
--rpccorsdomain "http://wallet.moac.io"
```

This will allow the access of MOAC node using MOAC wallet.

Otherwise you will see the following error message:

图 21: ERC20_moacwallet01.png

Example command to start a MOAC node connecting with mainnet:

```
moac --rpccorsdomain "http://wallet.moac.io" --rpc --rpcport "8545" --rpcapi "chain3,mc,
net,db"
```

A successful interface connecting to mainnet looks like this:

图 22: ERC20_moacwallet02.png

To compile the contract, click the "CONTRACTS" icon:

图 23: ERC20_moacwallet03.png

Then enter the contract deploy page:

Copy the source code and paste in the "SOLIDITY CONTRACT SOURCE CODE" section. After copying the source code, the compiling process will automatically started. If no errors are not found, the right side should show a "SELECT CONTRACT TO DEPLOY" menu. **|ERC20_moacwallet05.png|** Select the contract name "TokenERC721" : **|ERC20_moacwallet06.png|** Input the parameters values from the menu: **|ERC20_moacwallet07.png|** You need to have enough balance to deploy the contract. You can choose the amount of fee to use when deploying the contract. Click the DEPLOY button: **|ERC20_moacwallet08.png|**

This is the contract ready to send from Account 1. The Provide gas is estimated by the compiler and we suggest you use it or put a larger number. If gas is not enough, the contract cannot be created. To continue, be sure to unlock the account to create the contract. You can use a console attached to the MOAC to do this:

After deploying, the interface is returned to the main menu and you can see the following transaction is creating.

|ERC20_moacwallet10.png| After 12 confirmations, you can start using the contract by click the admin page link. **|ERC20_moacwallet11.png|**

MOAC wallet is good for beginners that only need basic contract development needs. It cannot debug contracts. To advanced developers, you can use Remix to work with contracts.

图 24: ERC20_moacwallet04.png

图 25: ERC20_moacwallet09.png

Remix

Remix is an online tool developed by Ethereum community to work with smart contracts. MOAC also supports the deploy of smart contract through Remix.

Open Remix on your browser, create a new file called ‘TestToken.sol’ and copy paste the code of your smart contract. Make sure you are including all the other Solidity files that your code is referencing with imports, especially if you are using the open source library approach.

Select ‘TestToken’ in the Compile window then click “Start to Compile” and the Details button next to TestToken. Upon scrolling in the popup details window for TestToken, you should be able to see similar sections to this Remix screenshot for the bytecode and ABI of your smart contract:

If the contract is compiled successfully, remix will show the interface like this:

|ERC20_moacwallet12.png| To deploy the contract, you need to connect REMIX to a local or remote MOAC node. In addition to other arguments, be sure to enable the access of REMIX to the MOAC node with

```
moac --rpccorsdomain "http://remix.ethereum.org" --rpc --rpcport "8545" --rpcapi "chain3,
↪mc,net,db"
```

Click the Run Tab and you should see the following menu: **|ERC20_moacwallet13.png|**

Choose the Environment menu: JavaScript VM is a simulated environment of Remix, it can be use to debugging the contract without actually deploying the contract to a real network. Injected Web3 is the default web3 connecting to Ethereum network. To deploy MOAC contract, you need to choose Web3 Provider.

图 26: ERC20_moacwallet14.png

After choose “Web3 Provider” , you can see a message like this:

Click “OK” ,

You need to make sure the port is the same as the local running node.

You may see the error message like this:

|ERC20_moacwallet17.png| If you see this error message, check the local node that include both

图 27: ERC20_moacwallet15.png

图 28: ERC20_moacwallet12.png

```
--rpccorsdomain "http://remix.ethereum.org"
```

and

```
--rpcport "8545"
```

If the connection is established, you should see your accounts from the Account List.

图 29: ERC20_moacwallet18.png

Before you deploy the contract, you need to unlock the account that send the contract. You can do the unlock with the MOAC console:

After successfully deployed the contract, you should see the contract address and other information showed in the menu:

Remix is good for developing and debugging smart contracts. It is not very convenient to deploy multiple contracts. If your requires to deploy multiple contracts, you can use the Node.Js packages.

Using the Node.Js packages

You need to install solc package to compile the smart contract, and chain3 package to deploy the contract.

To use the latest stable version of the Solidity compiler via Node.js you can install it via npm:

```
npm install solc
```

```
var solc = require('solc')
var input = 'contract x { function g() {} }'
// Setting 1 as second paramateractivates the optimiser
var output = solc.compile(input, 1)
for (var contractName in output.contracts) {
        // code and ABI that are needed by web3
        console.log(contractName + ': ' + output.contracts[contractName].bytecode)
        console.log(contractName + '; ' + JSON.parse(output.contracts[contractName].
↪interface))
}
```

图 30: ERC20_moacwallet09.png

图 31: ERC20_moacwallet19.png

To deploy the contracts, you need to install the Chain3 package:

```
npm install chain3
```

There is an example file in the package: example/contract_deploy.js

After successfully deploy, you should see the contract is displayed

Succeed!: 0x95d703ea48477f48335ae9c477ce6d986bc68453dfe3d6582714045456b93405

Using solc compiler to generate the ABI and bytecode Another way of generating these two files is to compile your smart contract using the solc compiler on your machine. If you haven' t used solc yet, you can follow the

6.4 FileStorm 使用指南

6.4.1 介绍

星际风暴是一个通过墨客子链将星际文件系统（IPFS）和区块链结合而产生的分布式存储平台。星际风暴构建在墨客子链之上，使得有存储需求的用户可以消耗通证来得到存储空间，而提供存储空间的节点提供者通过竞争来获得通证。这种激励方式可以鼓励大家把闲置的硬盘空间来共享，从而搭建一个可商用的全球化存储空间。更多信息可以访问项目网站（https://www.filestorm.net）。

FileStorm is a decentralized storage platform implemented with IPFS and MOAC. The files are stored on millions of data nodes provided by people around the world in IPFS format in exchange for MOAC. FileStorm is a project that involves three types of users.

1. Storage Provider: Provides the hardware devices used for storage, such as computers with large volume of storage, or customized storage boxes. FileStorm program needs to be installed on these devices so they can connect to Moac FileStorm microchains as well as IPFS network. Storage providers can gain reward in MOAC by offering the storage service.
2. Dapp Developers: The creator and owner of FileStorm microchains. They can create a dedicated microchain used for storage only for their own Dapp. Moac Foundation will also create a public FileStorm microchain. Dapp Developers will have to pay MOAC to deploy and maintain a FileStorm.
3. Storage End Users: The users will use FileStorm through Dapps. The end users do not need any MOAC to use FileStorm, but they may have to pay to use the Dapps.

6.4.2 使用者

存储提供方

Storage providers are the people who run the FileStore MicroChain as the service.

FileStorm MicroChain needs to install the following packages:

- redis - Local database, used to save the map between public HASH and private HASH;
- IPFS Daemon - IPFS platform to save the data files;
- SCSServer - MOAC MicroChain server;
- stormcatcher - Program calls the IPFS program from MOAC MicroChain server;

These packages can be installed separately or through Docker.

Installation

redis:

Ubuntu

```
sudo apt update
sudo apt full-upgrade
sudo apt install build-essential tcl
curl -O http://download.redis.io/redis-stable.tar.gz
tar xzvf redis-stable.tar.gz
sudo apt install redis-server
```

CentOs

:: sudo yum install epel-release sudo yum update sudo yum install redis

ipfs:

IPFS can be downloaded from IPFS

Ubuntu

```
wget https://dist.ipfs.io/go-ipfs/v0.4.17/go-ipfs_v0.4.17_linux-amd64.tar.gz
tar xvfz go-ipfs_v0.4.17_linux-amd64.tar.gz
sudo mv go-ipfs/ipfs /usr/local/bin/ipfs
```

:: sudo yum install epel-release sudo yum update sudo yum install redis

scsserver:

The newest version can be download from release link.

scsserver stormcatcher userconfig.json run_filestorm_scs.sh stop_filestorm_scs.sh

Then you can execute filestorm_install.sh to install filestorm scs.

Configuration

Configuration file: userconfig.json

VnodeServiceCfg can be setup using the Vnode/Port information from Node info - Testnet Protocol Pool. Beneficiary is the public wallet address of SCS owner.

The VNODE server need to open the following ports for outside access: * 4001; * 5001; * 8080;

Running

IPFS file system needs to be initialized before the first usage:

```
ipfs init
```

Then the user can run FileStorm MicroChain using the start script:

```
cd scsserver
./run_filestorm_scs.sh
```

The start script calls the four components in the FileStorm:

```
#!/bin/bash
echo "Starting FileStorm SCS--"

nohup ipfs daemon > ipfs.out 2>&1 &
echo "IPFS Daemon started."

nohup redis-server > ipfs.out 2>&1 &
echo "Redis Server started. "

nohup ./ipfs_monkey > ipfs.out 2>&1 &
echo "IPFS Monkey started."

nohup ./scsserver > scs.out 2>&1 &
echo "SCS started."
```

To stop the service, call the following script:

```
./stop_filestorm_scs.sh
```

The closing script closed the four components:

```
#!/bin/bash
echo "Stoping FileStorm SCS--"
pkill ipfs_monkey
pkill redis-server
pkill ipfs
pkill scsserver
echo "FileStorm SCS Stopped."
```

To make the SCSs be able to work with MicroChain, 0.5 moac depsit needs to be added to each SCS' s address.

Monitring:

To check if the MOAC MicroChain running, check the log file:

```
tail -f scs.out
```

DAPP Developers

DAPP developers deploy the DAPP on the FileStorm MicroChain to let the Storage Users access the data through the DAPP.

To develop a DAPP on the FileStorm platform, you need to: 1. Run a vnode locally to connect to the MOAC mainnet(or testnet for testing). The newest released version is under:

https://github.com/MOACChain/moac-core/releases

2. Start the vnode: To connect with mainnet: `./moac --rpc --rpccorsdomain "http://wallet.moac.io" console` To connect with testnet: `./moac --testnet --rpc --rpccorsdomain "http://wallet.moac.io" console`
3. Use MOAC walletto deploy the MicroChain;
4. DeploySubChainBase.sol
5. Find Node info - Testnet SubChainProtocolBase pool 地址和 Vnodeproxy pool 地址
6. Use MOAC wallet to deploy the FileStormMicroChain.sol;
7. Register the MicroChain;
8. Check the status of MicroChain with MicroChain explorer.

数据存储方

Strage users access the data on the IPFS through DAPP deployed on the FileStorm.

Example:

The following procedures show how to access a data file on the FileStorm testnet.

1. Setup a local VNODE server. The software can be downloaded from https://github.com/MOACChain/moac-core/releases;
2. Running the VNODE: `./moac --testnet console`;
3. Setup IPFS download;

   For ubuntu:

   ```
   wget https://dist.ipfs.io/go-ipfs/v0.4.17/go-ipfs_v0.4.17_linux-amd64.
   ↪tar.gz
   tar xvfz go-ipfs_v0.4.17_linux-amd64.tar.gz
   sudo mv go-ipfs/ipfs /usr/local/bin/ipfs
   ```

4. Generate a local text file for uploading:`vi newtestfile.txt`
5. Add the generated file to the IPFS system：`ifps add newtestfile.txt`
6. Convert the hash to HEX code, which can be done using this web tool：https://codebeautify.org/string-hex-converter. or using the NODEJS tool: “ npm install –save ethereumjs-abi “

   var abi = require(‘ethereumjs-abi’); var original = ‘QmQNe96LqV5TcRQyBz12iQXPZQjemBqkgnpHki3wmKjtd6’ ; var encoded = abi.simpleEncode(‘write(string)’ , original);

   console.log(‘original’ , original);

   console.log(‘encoded’ , encoded.toString(‘hex’));

   7. The HEX code of the HASH should be a HEX code with length 46, total 92 digits. Since the storage of parameters in Solidity only has 32 digits, we used two parameters to store the HEX code of the HASH. The HEX code of the HASH is filled with 0s to make two HEX codes with 64 digits;
   8. Call the three functions to read, write and delete the data on the MicroChain:
   9. from: need to be an unlocked account;
   10. to: DAPP address provided by the DAPP developer or the Storage Provider;
   11. data: Add the HEX code from Step 7 after ‘2e’ ;
   12. After each successful call, the nonce need to increase by 1 for the next call.
   13. via needs to set as the same value of via in the vnodeproxy.json in the VNODE directory.

// write(fileHash) chain3.mc.sendTransaction({ from: chain3.mc.accounts[0], value:chain3.toSha(‘0’ ,’ mc’), to: subchainbaseaddress, gas: “200000” , gasPrice: chain3.mc.gasPrice, shardingflag: 1, data:

‘0xba3835ba0000000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000000000000000
nonce: 1, via: chain3.mc.accounts[0] });

// read(fileHash) chain3.mc.sendTransaction({ from: mc.accounts[0], value:chain3.toSha(‘0’ ,’ mc’), to: subchainbaseaddress, gas: “200000” , gasPrice: chain3.mc.gasPrice, shardingflag: 1, data: ‘0x616ffe830000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000
nonce: 2, via: mc.accounts[0] });

// remove(fileHash) chain3.mc.sendTransaction({ from: mc.accounts[0], value:chain3.toSha(‘0’ ,’ mc’), to: subchainbaseaddress, gas: “200000” , gasPrice: chain3.mc.gasPrice, shardingflag: 1, data: ‘0x80599e4b0000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000
nonce: 3, via: mc.accounts[0] }); “‘

Results：

Write：a file was written to FileStorm MicroChain’ s nodes with a HASH;

Read：A file will be read from FileStorm nodes; Remove：The data file will be removed from every FileStorm node;

6.5 移植到墨客平台

MOAC developed its VNODE client based on the Ethereum project. It also implements a ***javascript runtime environment*** (JSRE) that can be used in either interactive (console) or non-interactive (script) mode. It supports Dapp development through its own **‘chain3 <>‘___** JavaScript API, just like web3.js on Ethereum.

If you want to develop Dapp on MOAC, you should use Solidity. MOAC supports the deployment of smart contracts through Remix, wallet.moac.io and For a Dapp that wants to move to MOAC from Ethereum, please follow these steps:

1. Deploy the smart contract:
   - Remix, a browser compiler for Solidity.
   - wallet.moac.io, a browser Dapp connect with local nodes to deploy contracts.
   - Node.Js Solidity compiler, a JavaScript bindings for the Solidity compiler.

   You can check the two examples in our wiki page:

   ERC20

   ERC721
2. Interact with the smart contract:
   - chain3, developed based on web3.js 0.2.x, is the JavaScript library supported by MOAC.
   - web3.js, only the functions listed in chain3 are workable, such as web3, eth, admin.

6.5.1 Example

Web3.js:

```
var Web3 = require('../index.js');
var web3 = new Web3();

web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545'));

var coinbase = web3.eth.coinbase;
console.log(coinbase);

var balance = web3.eth.getBalance(coinbase);
console.log(balance.toString(10));
```

Chain3:

```
var Chain3 = require('../index.js');
var chain3 = new Chain3();

chain3.setProvider(new chain3.providers.HttpProvider('http://localhost:8545'));

var coinbase = chain3.mc.accounts[0];
console.log(coinbase);

var balance = web3.eth.getBalance(coinbase);
console.log(balance.toString(10));
```

Solidity

- Solidity: a smart contract programming language sharing similarities with Javascript and C++
- Solidity Glossary
- Solidity Features
- Solidity Collections
- Solidity Cheat-sheet

Miscellaneous resources and info

- Safety
- Ethereum ĐApp Developer Resources

- Ethereum JavaScript API
- Ethereum JSON RPC API
- Standardized Contract APIs
- Ethereum development tutorial
- ĐApp using Meteor
- Dapp Insight: dapp statistics

CHAPTER 7

其他

- genindex
- modindex
- search