目录

1 平台介绍 2

2 墨玶联盟链 4

3 联盟链中间件平台 10

4 联盟链监控 50

5 联盟链浏览器 53

6 名词解释 56

7 平台介绍 57

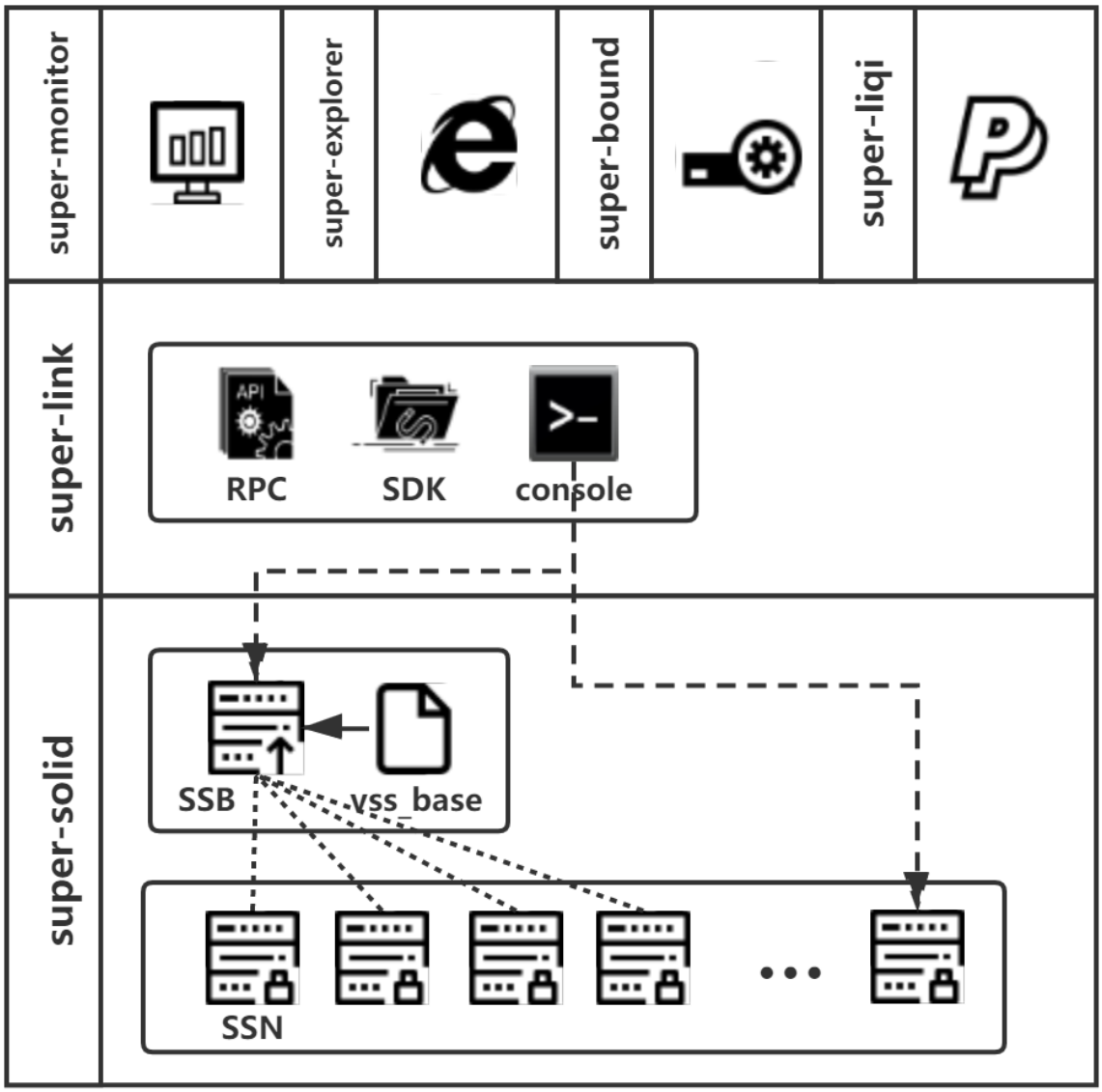
8 Super-Bound 平台部署 58

9 Super-Bound API 接口说明 60

墨玶联盟链平台

1 平台介绍

1.1 什么是墨玶联盟链平台



墨玶联盟链平台，以下简称联盟链平台，是由墨玶网络科技有限公司发布的新一代联盟链产品。由底层联盟链、通用中间间件平台、通用服务和通用产品组成。

底层联盟链（Super-Solid），采用自主研发的双层双控技术，使用 BLS 共识模式，内嵌收费模块，可快速部署，快速使用。具体参见联盟链部署 。

通用中间件平台 (Super-Link)，主要提供联盟链所需交互接口，当前由三种类型的方式：

- console 方式：在联盟链节点服务器上直接打开命令行交互，是一种开发模式的调试方式，具体参见联盟链命令行介绍 。
- rpc 方式：通过 rpc 接口连接联盟链，具体参见联盟链 *RPC* 介绍 。
- sdk 方式：安装联盟链 sdk 安装包，使用客户端方式交互，当前仅提供 nodejs 的 sdk，具体参见联盟链 *SDK* 介绍 。

通用服务指的是显示联盟链内部数据的服务工具，当前主要有：

- 联盟链通用浏览器 (Super-Explorer)：显示联盟链运行数据的通用浏览器，具体参见联盟链浏览器 。
- 联盟链监控平台 (Super-Monitor)：显示联盟链节点数据，并在权限范围内对联盟链进行监控和操作，具体参见联盟链监控 。

通用产品是指基于联盟链开发的通用应用层工具，当前通用产品是墨珩超零界系统（Super-Bound）和墨珩快速支付系统（Super-Liqi），此部分将单独列章介绍。

1.2 面向对象

本文档面向的对象是有一定技术基础的开发人员，我们力求从技术开发的角度将整个联盟脸平台描述清楚。

我们建议您在阅读文档前，首先阅读如下一些区块链基础知识，这将极大的帮助您更好的理解整个平台。

-
-

1.3 章节构成简介

本章主要由联盟链的部署、联盟链的接口、联盟链监控和联盟链浏览器四部分组成。

如果您想自己部署墨珩联盟链，请参见联盟链部署 。

如果您已经部署了联盟链，想要查看联盟链的运行数据，请参见联盟链浏览器 。

如果您已经部署了联盟链，想要监控节点数据，请参见联盟链监控 。

如果您已经部署了联盟链，想要做二次开发，请参见联盟链中间件平台 。

2 墨玶联盟链

2.1 部署前的准备工作

2.1.1 联盟链安装包构成

联盟链本链的安装包可在如下连接中获取: 。

此链接中的节点版本为试用版，部署完成后可使用二十万个块，如果需要后续使用，请参见收费模块介绍 。

安装包构成如下：

文件/文件夹	说明
super-solid-base	SSB，联盟链启动节点
genesis.json	联盟链启动节点配置文件（用户无需修改）
vssbase.js	联盟链启动节点控制脚本（用户有限修改）请参见联盟链部署 *2.1*
super-solid-node	SSN，联盟链主节点
userconfig.json	SSN 配置文件
sendgas.js	发送 gas 的脚本（用户有限修改）请参见联盟链部署 *4.1*
monitor-scripts\	监控脚本目录（用户无需修改）

2.1.2 服务器的建议配置

节点名称 联盟链启动节点 (SSB)

建议数量 1

最低配置 4 核 4G 计算型实例

推荐配置 4 核 8G 计算型实例

硬盘 100G+ 实际业务需求

节点名称 联盟链主节点 (SSN)

建议数量 5，7，9，11

最低配置 2 核 2G 计算型实例

推荐配置 4 核 4G 计算型实例

硬盘 100G+ 实际业务需求

网络带宽方面，服务器相互之间属于同一区域的机房走内网，没有限制，如果相互之间跨区域走外网，推荐至少 4M 带宽来保障数据正常传输，且实际需求需要根据业务量具体调整。

2.2 联盟链部署

在本节中，我们将采用一个启动节点 + 五个主节点的配置来描述如何部署墨珩联盟链。

2.2.1 启动 SSB

1.1 将安装包中的 super-solid-base 和 genesis.json 放在同一目录下，在服务器终端输入

```
super-solid-base  init genesis.json  --datadir "./path/to/chaindata"
```

datadir：表示初始化联盟链数据的目录。

1.2 启动 SSB

```
super-solid-base --verbosity=4, --rpc, --networkid=1510, --datadir=./path/to/
↪chaindata,
--rpcaddr=0.0.0.0, --rpcport=8545, --rpcapi=chain3,mc,net,db
```

可修改的参数说明如下

verbosity：日志级别（debug:4, info:3）。

datadir：表示初始化联盟链数据的目录。

rpcaddr：SSB rpc 地址

rpcport：SSB rpc 端口

1.3 在服务器新窗口中，输入

```
super-solid-base attach
```

此时进入 SSB 节点的命令行模式，这个命令行模式（以下简称 SSB 命令行）将在部署过程中一直使用，请不要关闭，如果异常，请再次 attach 进入。

1.4 创建*owner* 账号。在 SSB 命令行中输入

```
>personal.newAccount("YourPwd")
```

其中，YourPwd 是指 owner 账号的密码，返回是 owner 的地址。

1.5 命令行输入

```
>miner.start(1)
```

此时 SSB 将会进入挖矿状态。

1.6 等待一段时间，在 SSB 命令行输入

```
>mc.blockNumber
```

当输出值大于 10 以后，表示 SSB 启动成功。

2.2.2 部署 SSB 合约 vss_base

2.1 将安装包中 vssbase.js 放到 SSB 服务器上，打开 js 文件，配置联盟链初始化参数

```
var min = 0;                  // 暂时无效
var max = 5 ;                 // 联盟链初始主节点数量
var tokensupply = 1000000 ; // 联盟链原生币数量
var owner = "0x...";          // owner 地址
var pwd = "YourPwd";          // owner 密码
```

max：标识联盟链初始主节点数量，建议是 5，7，9，11。只有达到这个数量，才能建立联盟链。

tokensupply：联盟链原生币数量，建立联盟链后，原生币将会打入 owner 账号。

owner：1.4 中主账号地址。

2.2 SSB 命令行输入

```
>loadScript("your/path/to/vssbase.js")
```

等待结果返回

```
Contract mined! address: 0x...
```

此时标识部署 vss_base 合约成功。记录此地址后续备用。

2.2.3 首次次启动 SSN 节点

3.1 将安装包中 userconfig.json 放到 SSN 服务器上，配置 userconfig.json

```
{
"RpcServiceCfg": "http://127.0.0.1:8545/rpc",
"DataDir": "./ssndata",
"LogPath": "./_logs",
"VssBaseAddr": "0x...",
"ChainId": 1510,                                    //无需修改
"LogLevel": 4,
"SuperSolidName": "myssname"
}
```

RpcServiceCfg：SSB rpc 接口地址，要与 1.2 启动 SSB 相一致

DataDir：SSN 数据路径

LogPath：SSN 日志路径

VssBaseAddr：vss_base 合约地址

LogLevel：节点日志级别（debug:4, info:3）

SuperSolidName：联盟链名称，长度不得超过 32 个字节，**请注意**，所有 SSN 节点的 SuperSolidName 必须一致！

3.2 将安装包中的 super-solid-node 和 userconfig.json 放在同一目录下，在服务器终端输入

```
super-solid-node --rpc --rpcaddr 0.0.0.0 --rpcport 8546 --p2pport 30383
```

rpcaddr：SSN rpc 地址

rpcport：SSN rpc 端口

p2pprot：SSN p2p 端口

第一次启动后，super-solid-node 会自动关闭并提示 ssnId not sufficient funds。

在 super-solid-node 可执行文件路径下找到 **ssnkeystore** 文件夹，获取 ssnid（第一个 keystore 文件的 address），记录这个 ssnid 备用。

在 super-solid-node 可执行文件路径下找到 **ssndata/nodes** 文件夹，文件夹里有 my-static-node.json 文件。

my-static-node.json 文件示例如下

```
["enode://00137f199db5239989d3f2e2c1a2......
↪a96c81a81321c5465682fc240e49a5a4d9999081e08ad@[ip]:30383"]
```

请注意：如果联盟链建立在内网中，可以将 ip 改成内网 ip；如果是外网环境，须将 ip 改成外网 ip。

记录这个 enode 信息备用。

3.3 重复 3.1 和 3.2，将其他 SSN 节点启动起来，并记录各自的 ssnid 和 enode。

请注意：启动的 SSN 数量必须和 2.1 vssbase 中的 max 数量相等。

3.4 将汇总的 enode 信息做成一个总的 static-nodes.json 放到所有 SSN 节点的 **ssndata/nodes/** 下。

总的 static-nodes.json 文件示例如下

```
["enode://00137f199db5239989d3f2e2c1a2......
↪a96c81a81321c5465682fc240e49a5a4d9999081e08ad@[ip]:30383",
"enode://00237f199db5239989d3f2e2c1a2......
↪a96c81a81321c5465682fc240e49a5a4d9999082e08ad@[ip]:30383",
"enode://00337f199db5239989d3f2e2c1a2......
↪a96c81a81321c5465682fc240e49a5a4d9999083e08ad@[ip]:30383",
"enode://00437f199db5239989d3f2e2c1a2......
↪a96c81a81321c5465682fc240e49a5a4d9999084e08ad@[ip]:30383",
"enode://00537f199db5239989d3f2e2c1a2......
↪a96c81a81321c5465682fc240e49a5a4d9999085e08ad@[ip]:30383"]
```

2.2.4 SSN 节点添加 gas

4.1 将安装包中 sendgas.js 放到 SSB 服务器上，打开 js 文件

```
var ssnaddrs=["0x...", "0x...", "0x...", "0x...", "0x..."];
```

4.2 SSB 命令行输入

```
>loadScript("your/path/to/sendgas.js")
```

等待结果返回

```
Success address: 0x..., Balance: 100
Success address: 0x..., Balance: 100
Success address: 0x..., Balance: 100
Success address: 0x..., Balance: 100
Success address: 0x..., Balance: 100
```

如上信息表示添加 gas 成功！

2.2.5 再次启动 SSN 节点

5.1 再次一次启动所有 SSN 节点

```
super-solid-node --rpc --rpcaddr 0.0.0.0 --rpcport 8546 --p2pport 30383
```

此时 SSN 不会退出，将会进入正常的启动流程。

5.2 选择一个 SSN 节点，新开一个服务器窗口，输入

```
super-solid-node attach
```

进入 SSN 命令行模式，等待一段时间，输入

```
> mh.blockNumber
```

当输出值大于 1 以后，表示联盟链启动成功!!

!! 至此联盟链全部部署完成!!

此时，owner 地址中将会有 totalsupply 的货币数额，可在 SSN 的命令行输入如下命令查询

```
> mh.getBalance(yourweneraddr)
```

接下来可继续部署联盟链浏览器 和联盟链监控。

2.2.6 监控相关设置

如果联盟链需要连接监控，需要在 SSB 和 SSN 服务器上做如下设置

STEP1 在一台服务器创建 ssh 账号

```
ssh-keygen -t rsa
```

执行后，将会在当前用户下 ~/.ssh 文件夹中产生一个私钥文件 (id_rsa) 和一个公钥文件 (id_rsa.pub)。

STEP2 将公钥复制到每台 SSB/SSN 的安装目录下

STEP3 将私钥保存下来待用

2.2.7 部署注意点

- 如果您是在云服务器上部署联盟链，请在云服务上开启相关的 rpc 端口
- 所有示例的启动命令没有持久化，如果需要后台执行（守护进程）请加 nohup

2.3 收费模块介绍

墨珩联盟链采用免费试用，付费使用的方式。所有发布的联盟链版本，部署完成后，都会有 **二十万个块高的免费使用**。按十秒的出块速度，试用版可以使用三周左右。试用结束后，区块将停止出块。

如果需要继续使用，需要进行区块高度续块充值。当前我们提供两种方式的续块充值

- 如果联盟链 rpc 对外开放，当您付费后，我们将根据要求直接进行“续块”操作。
- **如果联盟链是内部环境，那么**
  - 如果已经部署了联盟链监控，那我们会提供密钥串，可以在监控的“续块”标签页面输入密钥进行续块。
  - 如果没有监控，我们将提供脚本，可在任意 SSN Console 进行续块。

具体费用，不在本文档范围内，请参见。

3 联盟链中间件平台

3.1 联盟链命令行介绍

Console 是指联盟链开发中的命令行模式，用于开发人员直接通过命令行方式对链上的数据进行调试。当前*SSN* 和 SSB 都有命令行模式。命令行模式适用于联盟链部署和 debug 链上数据，适合开发人员使用。

另外两种*SDK* 和*RPC* 的开发模式，将在本章其他节介绍。

3.1.1 启动 Console 的方式

启动 SSB console 的方式

前提：已经启动 SSB 然后在同一台服务器中输入

```
super-solid-base attach
```

启动 SSN console 的方式

前提：已经启动 SSN 然后在同一台服务器中输入

```
super-solid-node attach
```

3.1.2 SSN console 命令汇总

联盟链主要是在 SSN 上进行命令行调试，以下是当前常用的命令行范例。

本节将按照前缀进行分类。

mh

getSSNId：获取 SSN 节点 id

sendRawTransaction: 发送已加签好的交易

accounts：获取 SSN 节点已创建地址列表

sign：以特殊前缀加签

signTransaction：交易加签

sendTransaction：以未加签方式发送交易

anyCall：调用联盟链已部署合约方法，如何部署联盟链合约，请参见

getBalance：获取联盟链账号余额

getBlock：获取联盟链某一区块信息

getBlockList：获取联盟链某段区块信息

getBlockNumber：获取联盟链区块高度

getContractAddrList：获取联盟链已注册合约列表

getNonce：获得对应账号在联盟链中的 Nonce

getReceiptByHash：通过交易 hash 获取联盟链的 tx 执行结果

getReceiptByNonce：通过账号和 Nonce 获取联盟链的 tx 执行结果

getTransactionByHash：通过交易 HASH 获取联盟链的交易信息

getTransactionByNonce：通过账号和 Nonce 获取联盟链的交易信息

syncing：获取联盟链节点同步信息

getBlockThreshold：获取联盟链下次续费块高度

接口 *mh*.getSSNId

描述 获取 SSN 节点 id

输入参数 无

: 返回：

- SSN 地址

示例:

```
> mh.getSSNId()

"0x76449055dc3cf91090c11f7c544b60363bf896cb"
```

接口 *mh*.sendRawTransaction

描述 发送已加签好的交易

输入参数

- 加签交易数据

返回

- 交易 hash

示例:

```
> mh.sendRawTransaction("0xf86a33808504a817c8008094a......
↪3364503ba2b67599ab218117b3182a30")

"0xa629303563c97821fca......7e2963cb06692704d84fbc05946e5576a5"
```

接口 *mh*.accounts

描述 获取 SSN 节点已创建地址列表

输入参数 无

: 返回：

- SSN 地址列表集

示例:

```
> mh.accounts

["0x76449055dc3cf9109.......c544b60363bf896cb", "0x9441c6707a1......119f27497c8227d"]
```

接口 *mh*.sign

描述

以特定前缀签名, 通过向消息添加前缀，可以将计算得到的签名识别为 moheng 特定的签名。这可以防止恶意的 DApp 对任意数据 (如事务) 进行签名，并使用签名冒充受害者。注意要签名的地址必须解锁。

输入参数

- 加签账号地址
- 加签明文数据

: 返回：

- 签名结果

示例:

```
> mh.sign("0x69f3d468cbec......b203668518dcd18d11b16", "0x53220b11be2c92dbb4......
↪73f93eb95385e92a9ed29bf9d07d79f534b")

"0x8f647ae2563179eed1ceed810300......1784ba8675794a17639e5ae6b52f88d1b"
```

接口 *mh*.signTransaction

描述 对交易进行加签，注意要签名的地址必须解锁。

输入参数

- from：交易发送账号地址，也是交易加签地址
- to：交易指向的账号地址，部署合约交易不填写

- value：交易转出的货币数量，
- data：加签数据
- nonce ： 交易 nonce ， 可不填

: 返回：

- raw: 交易加签结果

示例:

```
> mh.signTransaction({from:mh.accounts[1], to: "0xaa4c98c7efb24......b437cd761df4e648c
↪", value:'0x5f484e5a'})

{
    raw: "0xf8658080808094aa4c98c7efb2......
↪5b7b2a7b6d3e37b29e5eb548aa073e0436ca1318f7f9",
    tx: {
        ......
    }
}
```

接口 *mh*.sendTransaction

描述 发送未加签的交易，注意要签名的地址必须解锁。

输入参数

- from：交易发送账号地址，也是交易加签地址
- to：交易指向的账号地址，部署合约交易不填写
- value：交易转出的货币数量，
- data：交易数据
- nonce ： 交易 nonce ， 可不填

: 返回：

- 交易 hash

示例:

```
> mh.sendTransaction({from:mh.accounts[1], to: "0xaa4c98c7efb244f4......7cd761df4e648c
↪", value:'0x0'})

 "0x06e6153b5878420cedb3850fe3b486......add715edfa24b606b6355b802"
```

接口 *mh*.anyCall

描述 获取 SSN 合约函数的返回值，调用此接口前必须将合约注册入 dappbase

输入参数

- Sender：查询账号
- DappAddr: 联盟链业务逻辑地址
- Params：第一个参数是调用的方法，之后是方法传入参数

: 返回：

- 函数返回值

示例:

```
> mh.anyCall({Sender: "0xc2a0423fac6d......bd8560288edc94c00ee", DappAddr:
↪"0xe5d7da7a37......4c43a7e16575f18e4b746", Params:['getCurNodeList']})

 "{\"nodeList\":[\"0x76449055dc3cf......b60363bf896cb\",\"0xc2a0423fac6d1ae......
↪0288edc94c00ee\"]}"
```

接口 *mh*.getBalance

描述 获得对应账号在联盟链中的货币余额。

输入参数

- 账号地址

: 返回：

- 账号余额，精度 18

示例:

```
> mh.getBalance('0xf180041c895a6aa......277cea3e20c2cbe')

  11000000000000000000
```

接口 *mh*.getBlock

描述 获得联盟链某一区块信息。

输入参数

- 块号

示例:

```
> mh.getBlock(37)

{
  extraData: "0xa77c68b8d817377d61c23bcea32c57eab4db5......
↪256ff80301401c1d7423d512504cc588ab68dd36f90fcd1642b1dfb0a79564f44432311a5fd00ef252dbe79d002929aa42
↪",
  hash: "0xfe8f215cb8b70d43d50c185......de2914c443ac064d436807e39a7",
  miner: "0xc2a0423fac6d......bd8560288edc94c00ee",
  number: "0x25",
  parentHash: "0xc881942213......f9598798343f0a1793006eab6ce84272da5428ce",
  receiptsRoot: "0xa07cd7906ea8b......aa57a7cd068c6f2acd7a165cf599537022b9d8fc4",
  stateRoot: "0xd6f76d1619ae985d......d7c530569d864323d16d248b8cd3f4581e40e",
  timestamp: "0x5e55e4ee",
  transactions: ["0x36249ce5e68.......19c5d3f4e7e63d805416d5c038a611b111cd9a43a9",
↪"0xaaf14ac0152fb12f71cce01fb8.......0c5fc4d0cbb27cc64f3f78b2720bca6b"],
  transactionsRoot: "0xb7ece0cd0fcc......
↪23062dc25fc66529fa5093ae30859f91a11298676bc0f0"
}
```

接口 *mh*.getBlockList

描述 获取联盟链某一区间内的区块信息。

输入参数

- 起始块号
- 结束块号

示例:

```
> mh.getBlockList(36, 37)
{
  blockList: [{
```

(continues on next page)

(continued from previous page)

```
      extraData: "0xf0c9626d45789730f1fcb773907f6bae0d4......
↪1180f4c39f7fd3daf67ee61d79f8e239c3b14ed160f8a606",
      hash: "0xc881942213aad1bad5403c47f95......93006eab6ce84272da5428ce",
      miner: "0xc2a0423fac6d1aee9......8560288edc94c00ee",
      number: "0x24",
      parentHash: "0xc5dd98df039f1d5......1c197b666c1db038da2f225bd8a9772de966d4",
      receiptsRoot: "0x56e81f171bcc55a6......2c0f86e5b48e01b996cadc001622fb5e363b421",
      stateRoot: "0x90766f90ee1d7dfc029......4cfcb94d6f370b78ff522b3066235325",
      timestamp: "0x5e55e4e4",
      transactions: [],
      transactionsRoot: "0x56e81f171bcc55a......f86e5b48e01b996cadc001622fb5e363b421"
  }, {
      extraData: "0xa77c68b8d817377d61c23.......442a62d2b00e",
      hash: "0xfe8f215cb8b70d43d50c185fb0b7......5de2914c443ac064d436807e39a7",
      miner: "0xc2a0423fac6d1aee......d8560288edc94c00ee",
      number: "0x25",
      parentHash: "0xc881942213aa......9598798343f0a1793006eab6ce84272da5428ce",
      receiptsRoot: "0xa07cd7906ea8b......a57a7cd068c6f2acd7a165cf599537022b9d8fc4",
      stateRoot: "0xd6f76d1619ae985......583dd7c530569d864323d16d248b8cd3f4581e40e",
      timestamp: "0x5e55e4ee",
      transactions: ["0x36249ce5e68......719c5d3f4e7e63d805416d5c038a611b111cd9a43a9",
↪ "0xaaf14ac0152fb12f71cce01fb8e244cc......d0cbb27cc64f3f78b2720bca6b"],
      transactionsRoot: "0xb7ece0cd0fcc52......
↪62dc25fc66529fa5093ae30859f91a11298676bc0f0"
  }],
  endBlk: "0x25",
  startBlk: "0x24"
}
```

接口 *mh*.getBlockNumber

描述 获得当前联盟链的区块高度。

: 输入参数：无

: 返回：当前区块高度

示例: :: > mh.getBlockNumber()

2245

接口 *mh*.getContractAddrList

描述 获取联盟链内所有已注册合约的地址列表，需要已部署的合约在 dappbase 中调用 registerDapp 方法后才能生效，具体方法请参见

: 输入参数：无

: 返回：合约地址列表，第一个是 dappbase 地址，之后是已注册合约地址

示例:

```
> mh.getContractAddrList()

["0xe5d7da7a3746......7e16575f18e4b746"]
```

接口 *mh*.getNonce

描述 获得对应账号在联盟链中的 Nonce，这是调用联盟链 DAPP 合约的必要参数之一，每当联盟链交易发送后会自动加 1

- 账号地址

: 返回：该账号 Nonce

示例:

```
> mh.getNonce('0xf180041c895......277cea3e20c2cbe')

1
```

接口 *mh*.getReceiptByHash

描述 通过交易 hash 获取联盟链的 tx 执行结果

: 输入参数：

- 账号地址

示例:

```
> mh.getReceiptByHash('0x9cf04d4f00998ba09c2c6......
↪85b173a406baedde414f08d88df5c992034')
{
  contractAddress: "0x000000000000......00000000000000000",
  failed: false,
  logs: [],
  logsBloom: "0x00000000000000000......00000000000000000000000000000000000000000000",
  queryInBlock: 0,
  result: "",
  transactionHash: "0x9cf04d4f00998ba09c2c6......73a406baedde414f08d88df5c992034"
}
```

接口 *mh*.getReceiptByNonce

描述 通过账号和 Nonce 获取联盟链的 tx 执行结果

: 输入参数：

- 账号地址
- Nonce

示例:

```
> mh.getReceiptByNonce('0xf180041c895a6aA8b2......0A277cEA3e20C2CBE',2)
{
  contractAddress: "0x000000000000000......0000000000000000",
  failed: false,
  logs: [],
  logsBloom: "0x00000000000000000.......00000000000000000",
  queryInBlock: 0,
  result: "",
  transactionHash: "0x9cf04d4f00998ba0.......85b173a406baedde414f08d88df5c992034"
}
```

接口 *mh*.getTransactionByHash

描述 通过交易 HASH 获取联盟链的交易信息

: 输入参数：

- 交易哈希

示例:

```
> mh.getTransactionByHash('0x9cf04d4f00998ba09c......06baedde414f08d88df5c992034')
{
  blockHash: "0xdaf05facc57d218528e6c......51eeed54a9fae9e8be4353c23c58ec6",
  blockNumber: "0x910",
  from: "0xf180041c895a6aa8b2......a277cea3e20c2cbe",
  gas: null,
  gasPrice: null,
  hash: "0x9cf04d4f00998ba09c......5b173a406baedde414f08d88df5c992034",
  input: "0x3876c45f5b6f81......cf5ebcb075e400a571e",
  nonce: "0x2",
  r: "0x5c1aeea440bdf02cdf646......5ed29d42a41c59b299262ffb23a4a5e7",
```

(continues on next page)

(continued from previous page)

```
  s: "0x7dac198898963a33a69af......aab94328c0c03215467032055eb2f26b716d",
  shardingFlag: "0x2",
  syscnt: "0x0",
  to: "0xaa4c98c7efb24......bb437cd761df4e648c",
  transactionIndex: "0x0",
  v: "0xf0",
  value: "0xde0b6b3a7640000"
}
```

接口 *mh*.getTransactionByNonce

描述 通过账号和 Nonce 获取联盟链的交易信息

: 输入参数：

- 账号地址
- Nonce

示例:

```
> mh.getTransactionByNonce('0xf180041c895a6aA......20C2CBE',2)
{
  blockHash: "0xdaf05facc57d218528e6c73df......4a9fae9e8be4353c23c58ec6",
  blockNumber: "0x910",
  from: "0xf180041c895a......500a277cea3e20c2cbe",
  gas: null,
  gasPrice: null,
  hash: "0x9cf04d4f00998ba0......4fd85b173a406baedde414f08d88df5c992034",
  input: "0x3876c45f5b6f81......cf5ebcb075e400a571e",
  nonce: "0x2",
  r: "0x5c1aeea440bdf02cdf6.......3c5ed29d42a41c59b299262ffb23a4a5e7",
  s: "0x7dac198898963a33a69af......b94328c0c03215467032055eb2f26b716d",
  shardingFlag: "0x2",
  syscnt: "0x0",
  to: "0xaa4c98c7efb24......b437cd761df4e648c",
  transactionIndex: "0x0",
  v: "0xf0",
  value: "0xde0b6b3a7640000"
}
```

接口 *mh*.syncing

描述 获取联盟链节点同步信息

: 输入参数：无

: 返回：

- 开始块
- 当前块
- 最高块

示例:

```
> mh.syncing()
{
    startingBlock: '0x384',
    currentBlock: '0x386',
    highestBlock: '0x454'
}
```

接口 *mh*.getBlockThreshold

描述 获取联盟链下次续费块高度

: 输入参数：无

: 返回：

- 当前块号
- 下次续费高度

示例:

```
> mh.getBlockThreshold()
{
  Current: 312,//当前块号
  Threshold: 200000//下次续费块高度
}
```

txpool

content：获得联盟链交易池交易的详细信息

inspect：获得联盟链交易池交易的简要信息

status：获得联盟链交易池交易的交易数量

接口 *txpool*.content

描述 获得联盟链交易池交易的详细信息

: 输入参数：无

: 返回：

- pending：待打包
- queued：队列中

示例:

```
> txpool.content()
{
  pending: {},
  queued: {
    0xf180041c895a6aA8b2F38500A277cEA3e20C2CBE: {
      3: {
        blockHash: "0x00000000000000000......0000000000000000000000000000000000000000
↪",
        blockNumber: null,
        from: "0xf180041c895a......38500a277cea3e20c2cbe",
        gas: null,
        gasPrice: null,
        hash: "0x58eb432ff88f7666......5d92f0a577e6e769712f1a2356fc1c25bde9110",
        input: "0x3876c45f5b6f81......5ebcb075e400a571e",
        nonce: "0x3",
        r: "0x3c79d4f4b1336565db6234af......2c1aee1a3107bf92e97361c0b3",
        s: "0x7f582d62b3e99f5dad7d34dc......37f70811aa90c645a1fdd433f349338",
        shardingFlag: "0x2",
        syscnt: "0x0",
        to: "0xaa4c98c7efb244f41......7cd761df4e648c",
        transactionIndex: "0x0",
        v: "0xf0",
        value: "0x0"
      }
    }
  }
}
```

接口 *txpool*.inspect

描述 获得联盟链交易池交易的简要信息

: 输入参数：无

: 返回：

- pending：待打包
- queued：队列中

示例:

```
 > txpool.inspect()
{
  pending: {},
  queued: {
    0xf180041.......00A277cEA3e20C2CBE: {
      3: "0xAA4c98C7efb24......3bB437CD761Df4e648C: 0 sha + 0 Œ 20000000000 gas"
    }
  }
}
```

接口 *txpool*.status

描述 获得联盟链交易池交易的交易数量

: 输入参数：无

: 返回：

- pending：待打包
- queued：队列中

示例:

```
 > txpool.status()
{
  pending: 0,
  queued: 1
}
```

debug

traceTransaction：获得交易在 EVM 执行期间创建的结构化日志

接口 *debug*.traceTransaction

描述 返回交易在 EVM 执行期间创建的结构化日志，并将它们作为 JSON 对象返回

: 输入参数：

- 交易哈希

示例:

```
>debug.traceTransaction("1a0da52d43d42223c......6aba732a74100f7357f0a0128340",
↪"callTracer")

"{\"type\":\"CALL\",\"from\":\"0xc2a0423fac6d1ae......bd8560288edc94c00ee\",\"to\":\
↪"0xe5d7da7a374663218294c43a7e16575f18e4b746\",\"value\":\"0x0\",\"gas\":\"0x337be36\
↪",\"gasUsed\":\"0x0\",\"input\":\
↪"0x12df9412000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",\"output\":\"0x\",\"time\":\"0s\",......5a6aa8b2f38500a277cea3e20c2cbe\",\"value\
↪":\"0xde0b6b3a7640000\",\"input\":\"0x\"}]}"
```

---

personal

newAccount：创建新的账号

listAccounts: 获取下 SSN 的账号列表

unlockAccount：解锁指定账号

lockAccount：锁定账号

ecRecover：地址验签

接口 *personal*.newAccount

描述 创建新的账号

: 输入参数：

- 账号密码

示例:

```
>personal.newAccount('123456')

"0x69f3d468cbec......203668518dcd18d11b16"
```

接口 *personal*.listAccounts

描述 获取下 SSN 的账号列表

: 输入参数：无

示例:

```
 > personal.listAccounts
["0x76449055dc3cf.......0363bf896cb", "0x9441c6707a1ef......634a119f27497c8227d"]
```

接口 *personal*.unlockAccount

描述 解锁指定账号

- 解锁账号地址
- 解锁账号密码

: 输入参数：- 账号密码

示例:

```
 > personal.unlockAccount(mh.accounts[3], '123456')
true
```

接口 *personal*.lockAccount

描述 锁定账号

- 解锁账号地址

: 输入参数：

- 账号密码

示例:

```
 > personal.lockAccount(mh.accounts[3])
true
```

接口 *personal*.ecRecover

描述 ecRecover 返回用于创建签名的帐户的地址。注意，此函数与 mh.sign 和 personal.sign 兼容。

: 输入参数：

- 加签明文数据

- 加签密文数据

: 返回：

- 账户地址

示例:

```
> personal.ecRecover("0x53220b11be2c92dbb40be6159......92a9ed29bf9d07d79f534b",
↪"0x8f647ae2563......63a203686c0b513822a9935b2c8c51784ba8675794a17639e5ae6b52f88d1b")

  "0x69f3d468cbec148984a......518dcd18d11b16"
```

其他

接口 loadScript

描述 加载并运行 js 脚本文件

: 输入参数：

- 脚本文件位置（绝对路径）

: 返回：

- 是否正常加载

示例:

```
> loadScript('path\to\load.js')
  true
```

3.2 联盟链 RPC 介绍

RPC 是指联盟链对外的 rpc 接口，开发人员可利用这些接口获取联盟链的数据。当前*SSN* 和 SSB 都有 RPC 接口。开发人员可直接利用这些接口做二次开发。

另外两种*Console* 和*SDK* 的开发模式，将在本章其他节介绍。

3.2.1 启动 RPC 的方式

启动 SSB RPC 的方式

在联盟链部署 中，SSB 的启动方式如下

```
super-solid-base --verbosity=4, --rpc, --networkid=1510, --datadir=./path/to/
↪chaindata,
--rpcaddr=0.0.0.0, --rpcport=8545, --rpcapi=chain3,mc,net,db
```

其中，–rpc、–rpcaddr 和 –rpcport 就是指明 RPC 启动的三个参数，开发人员可选择为需要 RPC 服务的 SSB 加上这三个启动参数。

启动 SSN RPC 的方式

同样，在联盟链部署 中，SSN 的启动方式如下

```
super-solid-node --rpc --rpcaddr 0.0.0.0 --rpcport 8546 --p2pport 30383
```

同样，可选择以上三个参数来为 SSN 开启 RPC 服务。

3.2.2 SSN RPC 命令汇总

联盟链主要是在 SSN 上进行二次开发，在开启 SSN RPC 服务后，即可用以下常用的 RPC 服务。

假设当前 rpcaddr=127.0.0.1，rpcport=8546。

mh

getSSNId：获取 SSN 节点 id

sendRawTransaction: 发送已加签好的交易

accounts：获取 SSN 节点已创建地址列表

sign：以特殊前缀加签

signTransaction：交易加签

sendTransaction：以未加签方式发送交易

anyCall：调用联盟链已部署合约方法，如何部署联盟链合约，请参见

getBalance：获取联盟链账号余额

getBlock：获取联盟链某一区块信息

getBlockList：获取联盟链某段区块信息

getBlockNumber：获取联盟链区块高度

getContractAddrList：获取联盟链已注册合约列表

getNonce：获得对应账号在联盟链中的 Nonce

getReceiptByHash：通过交易 hash 获取联盟链的 tx 执行结果

getReceiptByNonce：通过账号和 Nonce 获取联盟链的 tx 执行结果

getTransactionByHash：通过交易 HASH 获取联盟链的交易信息

getTransactionByNonce：通过账号和 Nonce 获取联盟链的交易信息

syncing：获取联盟链节点同步信息

getBlockThreshold：获取联盟链下次续费块高度

接口 *mh*.getSSNId

描述 获取 SSN 节点 id

输入参数 无

返回

- SSN 地址

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getSSNId","params":[]}'␣
↪http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"0xe569b8d860ae0b9......fd66eb24a78de3fed"}
```

接口 *mh*.sendRawTransaction

描述 发送已加签好的交易

输入参数

- 加签交易数据

返回

- 交易 hash

示例:

```
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_sendRawTransaction","params
↪":["0xf8698080808094......ded65da7ff5426d4de44fd634cff969169617c4dca01"]}' http://
↪127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"0x039ace1656a1e2300aadcc......
↪014ae9b605bcf0dc4c991db2faefeb201e"}
```

接口 *mh*.accounts

描述 获取 SSN 节点已创建地址列表

输入参数 无

返回

- SSN 地址列表集

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_accounts","params":[]}'␣
↪http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":["0xe569b8d8......eb24a78de3fed","0x7f5440ce55db40...
↪...0b06d79defc87c7c","0xc4c17b......038af9a2d98309cd81f42b67"]}
```

接口 *mh*.sign

描述

以特定前缀签名, 通过向消息添加前缀，可以将计算得到的签名识别为 moheng 特定的签名。这可以防止恶意的 DApp 对任意数据 (如事务) 进行签名，并使用签名冒充受害者。注意要签名的地址必须解锁。

输入参数

- 加签账号地址
- 加签明文数据

返回

- 签名结果

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_sign","params":[
↪"0x7f5440ce55db4.......c0b06d79defc87c7c", "0xf869808080809......
↪04d754ab038af9a2d98309cd8f"]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":
↪"0x065529a4b5f86ceacff00e10fe18279ba0bd870d2fbab84a0731285deeb4......
↪105ead228cad3991f0ec020b82071b"}
```

接口 *mh*.signTransaction

描述 对交易进行加签，注意要签名的地址必须解锁。

输入参数

- from：交易发送账号地址，也是交易加签地址
- to：交易指向的账号地址，部署合约交易不填写
- value：交易转出的货币数量，
- data：加签数据
- nonce ： 交易 nonce ， 可不填

返回

- raw: 交易加签结果

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_signTransaction","params":[{
↪"from":"0x7f5440ce55......c3ec0b06d79defc87c7c", "to":"0xc4c17b9e04......
↪2d98309cd81f42b67", "value":"0xDE0B6B3A7640000"}]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"raw":"0xf8690180808094c4c17b9e04d......cd4edc2f1b",
↪"tx":{"TxData":{"nonce":1,"syscnt":0,"gasPrice":0,"gas":0,"to":"0xc4c17b9e04......
↪af9a2d98309cd81f42b67","value":1000000000000000000,"input":null,"shardingFlag":0,
↪"via":null,"v":240,"r":898862982657692024547......
↪91826418498030358005919901574114983l0,"s":507781054392481032685727479435646309......
↪27476093179398432947885954831,"hash":null},"DirCallSent":false}}}
```

接口 *mh*.sendTransaction

描述 发送未加签的交易，注意要签名的地址必须解锁。

输入参数

- from：交易发送账号地址，也是交易加签地址
- to：交易指向的账号地址，部署合约交易不填写
- value：交易转出的货币数量，
- data：交易数据
- nonce ：交易 nonce ，可不填

返回

- 交易 hash

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_sendTransaction","params":[{
↪"from":"0x7f5440ce55db403e......b06d79defc87c7c", "to":"0xc4c17b9e04d......
↪d98309cd81f42b67", "value":"0xDE0B6B3A7640000"}]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"0x8c6b2a2c1392839d04223bc3......
↪c7c70e4ebf62abee024a04f47f"}
```

接口 *mh*.anyCall

描述 获取 SSN 合约函数的返回值，调用此接口前必须将合约注册入 dappbase

输入参数

- Sender：查询账号
- DappAddr: 联盟链业务逻辑地址
- Params：第一个参数是调用的方法，之后是方法传入参数

返回

- 函数返回值

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_anyCall","params":[{
↪"DappAddr":"0x974c37d2b3a7......b94285cf5126512a", "Params":["coinName"]}]}' http://
↪127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"\"test coin\""}
```

接口 *mh*.getBalance

描述 获得对应账号在联盟链中的货币余额。

输入参数

- 账号地址

返回

- 账号余额，精度 18，十六进制

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getBalance","params":[
↪"0x7f5440ce55db40......d79defc87c7c"]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"0x21dfe1f5c5363780000"}
```

接口 *mh*.getBlock

描述 获得联盟链某一区块信息。

输入参数

- 块号，十六进制

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getBlock","params":["0x3"]}
↪' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"extraData":"0x18bcb765b41fdc06478c......
↪8d8f09f17f63f1c803","hash":"0x558c038641ca32fe003d6......
↪891571043b53665134cc28542952e7","miner":"0x44268c92ec660......ff1bfcbf9dfd14276c1",
↪"number":"0x3","parentHash":"0x2871f7a292b6b433183f2ee867......ca8bcb5b31eba01b054",
↪"receiptsRoot":"0x56e81f171bcc55a6ff8345e692c0f86e5......c001622fb5e363b421",
↪"stateRoot":"0x5e7b9dc44d4bbe962c1f9c......620ef1c484022a1a69e0a4b","timestamp":
↪"0x5e85821c","transactions":[],"transactionsRoot":"0x56e81f171bcc5......
```

(continues on next page)

(continued from previous page)

接口 *mh*.getBlockList

描述 获取联盟链某一区间内的区块信息。

输入参数

- 起始块号，十六进制
- 结束块号，十六进制

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getBlockList","params":["0x3
↪","0x4"]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"blockList":[{"extraData":
↪"0x18bcb765b41fdc06478c9da......17f63f1c803","hash":"0x558c038641ca32fe003......
↪43b53665134cc28542952e7","miner":"0x44268c92e......bfcbf9dfd14276c1","number":"0x3",
↪"parentHash":"0x2871f7a2......58ca8bcb5b31eba01b054","receiptsRoot":"0x56e81......
↪b996cadc001622fb5e363b421","stateRoot":"0x5e7b9dc44......7d86620ef1c484022a1a69e0a4b
↪","timestamp":"0x5e85821c","transactions":[],"transactionsRoot":"0x56e81f171bcc5....
↪..adc001622fb5e363b421"},{"extraData":"0x46c74da4d4d62e......8724fc009d3b07","hash":
↪"0x42d510......b0f9bb05e2085d2c07ac15caf90","miner":"0xe569b8d860......
↪66eb24a78de3fed","number":"0x4","parentHash":"0x558c038641ca32fe003d6d......
↪134cc28542952e7","receiptsRoot":"0x56e81f171bcc55a6......dc001622fb5e363b421",
↪"stateRoot":"0x5e7b9dc4......f1c484022a1a69e0a4b","timestamp":"0x5e858226",
↪"transactions":[],"transactionsRoot":"0x56e81f171bcc......001622fb5e363b421"}],
↪"endBlk":"0x4","startBlk":"0x3"}}
```

接口 *mh*.getBlockNumber

描述 获得当前联盟链的区块高度。

: 输入参数：无

: 返回: 当前区块高度，十六进制

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getBlockNumber","params":[]}
↪' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"0x33f"}
```

接口 *mh*.getContractAddrList

描述

获取联盟链内所有已注册合约的地址列表，需要已部署的合约在 dappbase 中调用 registerDapp 方法后才能生效，具体方法请参见

: 输入参数：无

: 返回: 合约地址列表，第一个是 dappbase 地址，之后是已注册合约地址

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getContractAddrList","params
↪":[]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":["0x974c37d2b......285cf5126512a"]}
```

接口 *mh*.getNonce

描述 获得对应账号在联盟链中的 Nonce，这是调用联盟链 DAPP 合约的必要参数之一，每当联盟链交易发送后会自动加 1

- 账号地址

: 返回: 该账号 Nonce

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getNonce","params":[
↪"0x7f5440ce55db......6d79defc87c7c"]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":2}
```

接口 *mh*.getReceiptByHash

描述 通过交易 hash 获取联盟链的 tx 执行结果

: 输入参数：

- 账号地址

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getReceiptByHash","params":[
↪"0x8c6b2a2c1392839d04223bc3abf0b31e588ea3c7c70e4ebf62abee024a04f47f"]}' http://127.
↪0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"contractAddress":"0x00000000000......
↪0000000000000000","failed":false,"logs":[],"logsBloom":"0x0000000000000000000......
↪00000000000000000000000000000","queryInBlock":0,"result":"","transactionHash":
↪"0x8c6b2a2c139......e588ea3c7c70e4ebf62abee024a04f47f"}}
```

接口 *mh*.getReceiptByNonce

描述 通过账号和 Nonce 获取联盟链的 tx 执行结果

输入参数

- 账号地址
- Nonce

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getReceiptByNonce","params
↪":["0x7f5440ce55d......6d79defc87c7c", 1]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"contractAddress":"0x00000000......0000000000000000
↪","failed":false,"logs":[],"logsBloom":"0x0000000000000000000000000000000000000000......
↪00000000000000","queryInBlock":0,"result":"","transactionHash":
↪"0x8c6b2a2c1392839d04223b......7c70e4ebf62abee024a04f47f"}}
```

接口 *mh*.getTransactionByHash

描述 通过交易 HASH 获取联盟链的交易信息

输入参数

- 交易哈希

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getTransactionByHash",
↪"params":["0x8c6b2a2c1392839d......4ebf62abee024a04f47f"]}' http://127.0.0.1:8546/
↪rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"blockHash":"0x8593bf7ca8fd20f755598c8b......
↪7b6cfcc936e3d98e3dc1","blockNumber":"0x2d0","from":"0x7f5440ce55db40......
↪79defc87c7c","gas":null,"gasPrice":null,"hash":"0x8c6b2a2c139283......
↪4ebf62abee024a04f47f","input":"0x","nonce":"0x1","syscnt":"0x0","to":
↪"0xc4c17b9e04d7......8309cd81f42b67","transactionIndex":"0x0","value":
↪"0xde0b6b3a7640000","v":"0xef","r":"0x1ea4ea0cd20......6bdf87143292b15af1a","s":
↪"0xd8b7411188aabc29e9bec5......5b7be580e17c435f10955569d80d","shardingFlag":"0x0"}}
```

接口 *mh*.getTransactionByNonce

描述 通过账号和 Nonce 获取联盟链的交易信息

: 输入参数：

- 账号地址
- Nonce

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getTransactionByNonce",
↪"params":["0x7f5440ce55db......defc87c7c", 1]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"blockHash":"0x8593bf7ca8fd20f75......
↪7b6cfcc936e3d98e3dc1","blockNumber":"0x2d0","from":"0x7f5440ce55d......defc87c7c",
↪"gas":null,"gasPrice":null,"hash":"0x8c6b2a2c1......bee024a04f47f","input":"0x",
↪"nonce":"0x1","syscnt":"0x0","to":"0xc4c17b9e04......a2d98309cd81f42b67",
↪"transactionIndex":"0x0","value":"0xde0b6b3a7640000","v":"0xef","r":
↪"0x1ea4ea0cd20658......b72b76f9a5342e6bdf87143292b15af1a","s":
↪"0xd8b7411188aabc29e9bec5bb2......c435f10955569d80d","shardingFlag":"0x0"}}
```

接口 *mh*.syncing

描述 获取联盟链节点同步信息

: 输入参数：无

返回

- 开始块，十六进制
- 当前块，十六进制
- 最高块，十六进制

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_syncing","params":[]}'http:/
↪/127.0.0.1:8546/rpc

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

接口 *mh*.getBlockThreshold

描述 获取联盟链下次续费块高度

输入参数 无

返回

- 当前块号
- 下次续费高度

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"mh_getBlockThreshold","params
↪":[]}' http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"Current":972,"Threshold":200000}}
```

txpool

content：获得联盟链交易池交易的详细信息

inspect：获得联盟链交易池交易的简要信息

status：获得联盟链交易池交易的交易数量

接口 *txpool*.content

描述 获得联盟链交易池交易的详细信息

输入参数 无

返回

- pending：待打包
- queued：队列中

示例:

```
// Request
 curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"txpool_content","params":[]}'␣
↪http://127.0.0.1:8546/rpc

 // Result
 {"jsonrpc":"2.0","id":0,"result":{"pending":{"0x7f5440ce55d......B06d79DEfC87C7c":{"2
↪":{"blockHash":"0x00000000000000000000......00000000000000000000000","blockNumber
↪":null,"from":"0x7f5440ce......ec0b06d79defc87c7c","gas":null,"gasPrice":null,"hash
↪":"0xff5d8......8ac013c9e491","input":"0x","nonce":"0x2","syscnt":"0x0","to":
↪"0xc4c17b9e0......81f42b67","transactionIndex":"0x0","value":"0xde0b6b3a7640000","v
↪":"0xef","r":"0x8494dbb7f3af......78f597406de","s":"0x7d25dd2abc4dafe......
↪5e724c9e66cc8c72e901421ba607c5c9606","shardingFlag":"0x0"}}},"queued":{}}}
```

接口 *txpool*.inspect

描述 获得联盟链交易池交易的简要信息

输入参数 无

返回

- pending：待打包
- queued：队列中

示例:

```
 // Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"txpool_inspect","params":[]}'␣
↪http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"pending":{"0x7f5440ce55db4......B06d79DEfC87C7c":{
↪"3":"0xC4c17b9e04d75......Cd81F42B67: 1000000000000000000 sha + 0 Œ 0 gas"}},"queued
↪":{}}}
```

接口 *txpool*.status

描述 获得联盟链交易池交易的交易数量

: 输入参数：无

: 返回：

- pending：待打包，十六进制
- queued：队列中，十六进制

示例:

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"txpool_status","params":[]}'␣
↪http://127.0.0.1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":{"pending":"0x1","queued":"0x0"}}
```

debug

traceTransaction：获得交易在 EVM 执行期间创建的结构化日志

接口 *debug*.traceTransaction

描述 返回交易在 EVM 执行期间创建的结构化日志，并将它们作为 JSON 对象返回

输入参数

- 交易哈希

示例:

```
 // Request
curl -X POST --data '{"jsonrpc":"2.0","id":0,"method":"debug_traceTransaction","params
↪":["0x1a0da52d43d4222......4100f7357f0a0128340", "callTracer"]}' http://127.0.0.
↪1:8546/rpc

// Result
{"jsonrpc":"2.0","id":0,"result":"{\"type\":\"CALL\",\"from\":\"0xc2a0423fac6......
↪9bd8560288edc94c00ee\",\"to\":\"0xe5d7da7......94c43a7e16575f18e4b746\",\"value\":\
↪"0x0\",\"gas\":\"0x337be36\",\"gasUsed\":\"0x0\",\"input\":\
↪"0x12df941200000000000000000000......0000000000000000000000000000005e562360\",\
↪"output\":\"0x\",\"time\":\"0s\",\"calls\":[{\"type\":\"CALL\",\"from\":\
↪"0xe5d7da7a37466321......7e16575f18e4b746\",\"to\":\"0xf180041c895a6aa......
↪ea3e20c2cbe\",\"value\":\"0xde0b6b3a7640000\",\"input\":\"0x\"}]}"}
```

3.3 联盟链 SDK 介绍

SDK 是作为专为客户端使用的软件开发包，极大方便了开发人员在客户端直接调用区块链相关接口。当前 SDK 可提供 nodejs 版本。SDK 可同时连接*SSN* 和 SSB。

另外两种*Console* 和*RPC* 的开发模式，将在本章其他节介绍。

3.3.1 SDK 的安装

可使用如下命令安装联盟链 nodejs 版本的 SDK

```
npm install super-solid-link
```

3.3.2 nodejs 版 SDK 详述

异常处理

通常，采用如下方式进行异常的捕获

```
var ssb = require("super-solid-link").ssb;

try{
        var ssbobj = new ssb("http://127.0.0.1:8545");
        var blockNumber = ssbobj.getBlockNumber();
        console.log(blockNumber);
}catch (e){
```

(continues on next page)

(continued from previous page)

```
        console.log(e);
}
```

账户注册

参数:

```
pwd: 账户密码
```

代码:

```
var account = require("super-solid-link").account;
var wallet = account.register(pwd);
```

返回:

```
wallet:
{ address: ' 账户地址....',
  privateKey: ' 私钥....',
  keyStore: 'keyStore 内容...'
}
```

账户登录

参数:

```
addr: 账户地址
pwd: 账户密码
keyStore: keyStore 字符串
```

代码:

```
var account = require("super-solid-link").account;
var privateKey = account.login(addr, pwd, keyStore);
```

返回:

```
privateKey:  账户私钥
```

3.3.3 SSB 模块接口

SSB 只介绍部署时需要用到的接口

实例化 SSB 对象

在使用接口前，需要打开一个节点的*RPC* 并允许外部访问。

参数:

```
ssbAddress: 基础链访问地址 http://127.0.0.1:8545
```

代码:

```
var ssb = require("super-solid-link").ssb;
var ssbobj = new ssb(ssbAddress);
```

获取基础链区块高度

代码:

```
var blockNumber = ssbobj.getBlockNumber();
```

返回:

```
blockNumber: 基础链区块高度
```

3.3.4 SSN 模块接口

实例化 ssn 对象

参数:

```
ssnAddress: ssn 访问地址 //http://127.0.0.1:8546
```

代码:

```
var ssn = require("super-solid-link").ssn;
var ssnobj = new ssn(ssnAddress);
```

获取联盟链 ssnId

代码:

```
ssnobj.getSsnId().then((sscId) => {
    console.log(sscId);
});
```

返回:

```
sscId: 联盟链 ssnId
```

获取下次续费块高度

代码:

```
ssnobj.getBlockThreshold().then((data) => {
    console.log(data);
});
```

返回:

```
data.Current: 当前块高度
    data.Threshold: 下次续费块高度
```

获取联盟链区块高度

代码:

```
ssnobj.getBlockNumber().then((blockNumber) => {
    console.log(blockNumber);
});
```

返回:

```
blockNumber: 联盟链区块高度
```

获取某一区间内的多个区块信息

参数:

```
start：开始高度
end：结束高度
```

代码:

```
ssnobj.getBlockList(start, end).then((blockListInfo) => {
    console.log(blockListInfo);
});
```

返回:

```
blockListInfo：区块信息 List
```

获取联盟链某一区块信息

参数:

```
blockNumber：区块高度
```

代码:

```
ssnobj.getBlock(blockNumber).then((blockInfo) => {
    console.log(blockInfo);
});
```

返回:

```
blockInfo：某一区块信息
```

通过交易 HASH 获取联盟链的交易信息

参数:

```
transactionHash：交易 hash
```

代码:

```
ssnobj.getTransactionByHash(transactionHash).then((transactionInfo) => {
    console.log(transactionInfo);
});
```

返回:

```
transactionInfo: 交易详情
```

通过交易 hash 获取联盟链的 tx 执行结果

参数:

```
transactionHash: 交易 hash
```

代码:

```
ssnobj.getTransactionReceiptByHash(transactionHash).then((result) => {
    console.log(result);
});
```

返回:

```
result: 执行结果
```

获取联盟链已注册合约列表

代码:

```
ssnobj.getContractAddrList().then((result) => {
    console.log(result);
});
```

返回:

```
result: 合约列表
```

获取联盟链账户余额

参数:

```
addr: 账户地址
```

代码:

```
ssnobj.getBalance(addr).then((balance) => {
    console.log(balance);
});
```

返回:

```
data：联盟链账户余额（erc20 最小单位）
```

获取联盟链详细信息

代码:

```
ssnobj.getAppChainInfo().then((appChainInfo) => {
    console.log(appChainInfo);
});
```

返回:

```
appChainInfo：联盟链信息
```

获取 Nonce

参数:

```
addr：账户钱包地址
```

代码:

```
ssnobj.getNonce(addr).then((nonce) => {
    console.log(nonce);
});;
```

返回:

```
nonce：得到的 nonce
```

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 = ssnobj.getData(method,paramTypes,paramValues);
```

返回:

```
data: data 字符串
```

联盟链加签交易

参数:

```
from: 发送方的钱包地址
contractAddress: 联盟链合约地址
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: 发送方钱包私钥
```

代码:

```
ssnobj.sendRawTransaction(from, contractAddress, amount, method, paramTypes,␣
↪paramValues, privateKey).then((hash) => {
    console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

调用联盟链合约

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:

```
ssnobj.callContract(contractAddress, param).then((data) => {
    console.log(data);
});
```

返回:

```
data: 调用合约返回信息
```

加签交易

参数:

```
from: 交易发起人
to: 交易接收人
    amount: 交易金额
    strData: 交易备注
    privateKey: 交易发起人私钥
```

代码:

```
ssnobj.sendRawTransactionPrivate(from, to, amount, strData, privateKey).then((hash) =>
↪ {
    console.log(data);
});
```

返回:

```
hash: 交易 hash
```

获取本地加签交易

参数:

```
from: 发送方的钱包地址
contractAddress: 联盟链合约地址
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: 发送方钱包私钥
```

代码:

```
ssnobj.getSignedTx(from, contractAddress, amount, method, paramTypes, paramValues,␣
↪privateKey).then((signedTx) => {
    console.log(signedTx);
});
```

返回:

```
signedTx: 交易加签后交易体
```

发送已加签好的交易

参数:

```
signTx: 交易加签后交易体
```

代码:

```
ssnobj.sendSignTransaction(signTx).then((hash) => {
    console.log(hash);
});
```

返回:

```
hash: 交易 hash
```

4 联盟链监控

4.1 什么是监控

墨珩联盟链监控 (Super-Monitor)，以下简称监控，是墨珩联盟链平台的一个配套服务。在一个已经部署的联盟链上，可以快速部署这样一套监控系统来监控联盟链的数据。

监控系统有如下一些模块

- 账户模块，监控根据不同的登陆账号给出不同的权限
- 数据显示模块
- 图表模块
- 操作模块，主要收费续块和节点相关操作
- 报警模块
- 日志模块等

4.2 服务器配置推荐

推荐 2 核 2G 服务器一台，40G 硬盘。

推荐监控服务器和联盟链在同一网络环境下，可走内网。如果走外网，需要至少 2M 带宽。

4.3 监控部署

强烈建议安装在 Ubuntu 中

4.3.1 部署前需要安装的软件

在部署监控前，以下软件必须安装

- node：https://npm.taobao.org/mirrors/node/v10.19.0/，推荐下载 v10.19.0

- npm：node 自带
- mongodb：https://www.mongodb.org/dl/linux
- pm2

4.3.2 程序包下载

联盟链监控的程序包可在如下连接中获取: 。

4.3.3 依赖包安装

解压程序包到安装目录下，并执行

```
npm install
```

4.3.4 修改配置文件

Mongodb 配置

配置根目录下 Mongodb 文件 config.json

- mongoHost：mongo IP 地址
- mongouname：mongo 用户名
- mongopasswd：mongo 密码
- dbname：mongo 库名称

Monitor 配置

- baseAddress：SSB vss_base 地址，具体参见联盟链部署 *2*
- ssb->list：SSB 服务列表
- ssb->list->name：SSB 显示名称
- ssb->list->host：SSB 服务器 rpc 的 IP，具体参见联盟链部署 *1.2*
- ssb->list->rpcPort：SSB 服务器 rpc 端口
- ssb->list->show：服务器监控信息是否展示（1：展示，0：不展示）
- ssb->list->username：服务器登录用户名

- ssb->list->serverPrivateKey：服务器私钥文件（在项目 routes 目录下，若每台服务器私钥不一样需要对应不同的名称），私钥文件产生方式参见监控相关设置
- ssb->list->scriptPath：SSB 服务器 Monitor 脚本文件夹 (monitor-scripts) 路径
- ssn->list：SSN 服务列表
- ssn->list->name：SSN 显示名称
- ssn->list->host：SSN 服务器 rpc 的 IP，具体参见联盟链部署 *3.1*
- ssn->list->rpcPort：SSN 服务器 rpc 端口
- ssn->list->show：服务器监控信息是否展示（1：展示，0：不展示）
- ssn->list->username：服务器登录用户名
- ssn->list->serverPrivateKey：服务器私钥文件（在项目 routes 目录下，若每台服务器私钥不一样需要对应不同的名称），私钥文件产生方式参见监控相关设置
- ssn->list->scriptPath：SSN 服务器 Monitor 脚本文件夹 (monitor-scripts) 路径
- ssn->list->nodePath：节点文件夹路径
- html->name：页面左菜单显示的名称
- html->chart_url: 页面图表嵌入的网址

4.3.5 启动并查看监控

输入如下命令启动监控

```
pm2 start app.js
```

至此，可在 http: 本机 ip:3002 查看监控。

4.4 监控的信息介绍

监控系统由登陆模块，监控主界面，转账/续块模块，节点投票模块，配置信息和日志模块组成。

监控系统中账号被分成如下几类：

- 联盟链管理员账号：*owner* 账号，拥有最高权限，可以发起添加节点的投票。
- 节点账号：指 SSN 节点的*ssnid* 账号，当 owner 发起投票时，拥有一次投票权。
- 新节点账号：指正在准备加入，在投票阶段的账号。

- 普通账号：监控系统其他注册账号。不能进行投票模块的操作。

4.5 监控的使用介绍

4.5.1 续块

4.5.2 节点添加

联盟链拥有者

老节点方

新节点方

4.5.3 转账

4.5.4 重启节点

5 联盟链浏览器

5.1 什么是联盟链浏览器

墨珩联盟链浏览器 (Super-Explorer)，以下简称浏览器，是墨珩联盟链平台的一个配套服务。在一个已经部署的联盟链上，可以快速部署一套浏览器来显示联盟链的数据。

浏览器有如下模块

- 首页，显示联盟链基本信息、区块信息、节点信息
- 区块信息详情页
- 交易信息详情页
- 账户信息详情页等

5.2 服务器配置推荐

推荐 4 核 4G 服务器一台，100G 硬盘。

推荐浏览器服务器和联盟链在同一网络环境下，可走内网。如果走外网，需要至少 2M 带宽。

5.3 监控部署

强烈建议安装在 Ubuntu 中

5.3.1 部署前需要安装的软件

在部署浏览器前，以下软件必须安装

- node：https://npm.taobao.org/mirrors/node/v10.19.0/，推荐下载 v10.19.0
- npm：node 自带
- node express：http://expressjs.com/en/starter/installing.html
- mongodb：https://www.mongodb.org/dl/linux
- pm2

5.3.2 程序包下载

联盟链浏览器的程序包可在如下连接中获取: 。

5.3.3 依赖包安装

解压程序包到安装目录下，并执行

```
npm install
```

5.3.4 修改配置文件

修改项目中配置文件（路径：/mhExplorer/userconfig.json）

- mongoHost - mongodb 数据库服务器连接方式，格式为 ip:port(如 121.43.129.11:10010)
- mongoUserName - mongodb 数据库服务器用户名

- mongoPwd - mongodb 数据库服务器密码
- dbname - 当前项目数据库名称
- ssbHost - SSB 服务器连接方式，格式为 ip:port(如 120.78.146.128:8545)，相关服务器必须打开 RPC 端口
- baseAddress - vss_base 地址 (大小写通用，页面显示统一处理为小写)
- ssnHosts - SSN 服务器组连接方式，格式为数组（如 [“47.106.34.55:8546”，“47.106.34.56:8546”])，相关服务器组必须打开 RPC 端口
- community_item1 - 页面 header community 部分下拉列表一
- community_item1_url - 下拉列表一链接
- community_item2 - 页面 header community 部分下拉列表二
- community_item2_url - 下拉列表二链接

以上凡涉及 host 格式，请勿在开头添加 http://, 保证“:”前后无空格。

5.3.5 自定义项目 Logo

如需自定义项目 logo，请修改您的 logo 图片名称为 logo.png，并覆盖至/mhExplorer/public/img 路径下。

5.3.6 启动浏览器

以上安装与配置都完成后，可到根目录 (/mhExplorer) 启动项目，如下：

```
node app
```

启动后在浏览器输入：http://localhost:3001, 方可查看项目启动情况, 默认监听端口为 3001，您可在根目录下 app.js 文件中修改端口。除此之外，您还可以根据自己情况，配置 pm2 方式启动，或在第三方 IDE 中启动项目。

5.3.7 启动数据监控脚本

在程序包目录下，有个监控脚本可执行文件，运行

```
./moheng-explorer-script -rpc "" -mdb "" -mongo "" -mpwd "" -muser "" -startBlock ""
```

启动参数说明如下

参数名	类型	说明
mdb	string	mongo 数据库名称
mongo	string	mongodb 数据库服务器连接方式
mpwd	string	mongodb 数据库服务器密码
muser	string	mongodb 数据库服务器用户名
rpc	string	SSN 服务器连接方式
startBlock	int	脚本执行开始高度

数据脚本日志

log.xml 中读取日志配置，会创建 logs 目录，每天生成一个日志文件。

5.4 监控浏览器说明书

6 名词解释

本节内容将会在相关段落中跳转，无需提前阅读

6.1 super-solid-base

SSB，联盟链启动节点

6.2 super-solid-node

SSN，联盟链主节点

6.3 命令行

在 SSB 和 SSN 中，都命令行模式，在开发或部署过程中，在节点启动后，可通过 attach 方式进入命令行模式。命令行的详情请参见联盟链命令行介绍 。

6.4 owner 账号

联盟链拥有者账号，在部署的时候创建，拥有一定的权限，请妥善保存

6.5 ssnid 账号

联盟链节点自动生成账号，在第一次启动 SSN 节点的时候自动生成

墨珩超零界系统

7 平台介绍

7.1 什么是墨珩超零界系统

墨珩超零界系统 (Super-Bound)，以下简称超零界系统，是墨珩网络科技有限公司自主研发的溯源存证平台，利用 cTree 快速存取模块，实现链上链下协同存证溯源。具有存验隔离、数据分片、海量存储、双重校对等特点。

当前，超零界系统需要和联盟链 配合使用，从而实现链上链下的数据交互。

7.2 超零界系统业务逻辑简介

超零界系统的存证溯源主要由如下几个模块：

- 生成 N 个标签。N 原则上小于等于一百万，初始化标签（时间较长）
- 二次开发
  - 访问判定
  - 锁定标签，存证，解锁标签
  - 标签上链（自动）
  - 验证标签
  - 申诉标签

7.3 章节构成简介

本章主要由超零界系统的部署和超零界系统的调用组成。

如果您已经部署了联盟链，想要在链上部署超零界系统，请参见*Super-Bound 部署* 。

如果您已经部署了查拦截系统，想要使用存证溯源的功能并进行二次开发，请参见*Super-Bound 调用* 。

8 Super-Bound 平台部署

8.1 服务介绍

本系统各模块采用分布式部署方式，相互之间通过基于 tcp 的 rpc 服务调用，具体分为如下四个服务：

1. **bcs** : 存证溯源业务服务，包含扫码（锁定标签，标签存证，标签解锁），溯源信息查询，溯源校验等功能模块，可对外提供 api 服务，默认端口 3001。

2. **traceToChain** : 溯源信息上链服务，bcs 服务会定时调用此服务将扫码溯源信息批量上链，默认端口 3002。

3. **merkleMemory** : 公共内存服务，用于构建 ctree 结构, 默认端口 3000。

4. **checkChainResult** : 上链结果确认服务，此服务是定时任务，用来确保上链业务 db 中数据和链上溯源数据的一致性，默认端口 3004。

8.2 准备环境

- 保证您的主机已经安装 node 环境, 版本不低于 v10.0，如还未安装，请参考这里
- 保证您的主机已经安装 node express 环境, 版本不低于 v6.0, 如还未安装，请参考这里
- 保证 **bcs** 服务器已经安装 mongodb, 版本不低于 v3.6，如还未安装，请参考这里
- 保证您有可连接的墨珩联盟链 环境

8.3 依赖安装

环境配置完成后，到各个服务项目的根目录下，执行如下指令安装依赖包。

```
npm install
```

8.4 配置文件

依赖安装完成后，请根据自己需求，修改各服务根目录配置文件 (config.json), 如下为配置文件各属性含义说明：

- mongoHost - mongodb 数据库服务器，格式为 ip:port(如 121.43.129.11:10010)

- mongouname - mongodb 数据库服务器用户名
- mongopasswd - mongodb 数据库服务器密码
- dbname - 当前项目数据库名称
- ssbHost - SSB 服务器连接方式，格式为 ip:port(如 120.78.146.128:8545)
- baseAddress - vss_base 地址
- ssnHost - SSN 服务器连接方式
- rpcHost - merkleMemory 内部 rpc 服务连接方式, 默认为“127.0.0.1:4242”，请勿修改
- from - 交易发送方, 及联盟链合约 owner
- to - 交易接受方
- privateKey - 交易发送方私钥
- env - 运行环境类型，“demo”为前台演示环境，“product”为扫码生产环境
- limitNum - 建议单次上链信息数量，默认为 12000
- pkStrStart - 公钥格式符开头，默认为“—–BEGIN PUBLIC KEY—–”，请勿修改
- pkStrEnd - 公钥格式符结束，默认为“—–END PUBLIC KEY—–”，请勿修改
- openInfo - 出场状态信息，默认为“open”，请勿修改

以上凡涉及 host 格式，请勿在开头添加 http://, 保证“:”前后无空格。

8.5 日志文件

本系统日志文件采用 nodejs log4j 配置，具体配置项可到各个服务根目录下 log4js.json 文件中查看，默认日志文件路径为根目录下/log 文件夹，日志文件名为 server-日期.log。

8.6 启动

以上安装与配置都完成后，请开启各个服务端口，它们默认为：3000，3001，3002，3004，4242，您也可以到各服务项目根目录下 app-*.js 文件中修改端口（实际情况下请将 * 替换为各个服务名称）。分别到各个服务根目录启动项目，如下：

```
node app-*
```

启动后在浏览器输入：http://localhost:3001, 方可查看项目启动情况, 除此之外，您还可以根据自己情况，配置 pm2 方式启动。

9 Super-Bound API 接口说明

在 Super-Bound 部署并初始化标签完毕后，即可使用 API 服务对标签进行存证溯源操作。

以下是可以使用的 API 汇总。

9.1 存证模块

9.1.1 锁定标签 (changeOwener)

简要描述：

- 将一个在初始化状态或解锁的标签锁定。

请求 URL： - `http://localhost:3001/changeOwner`

请求方式： - POST

参数：

参数名	必选	类型	说明
key	是	number	编号
labelPublicKey	是	string	标签公钥
ownerPublicKey	是	string	工厂 (扫码枪) 公钥
signInfo	是	string	加密信息 (标签私钥加签工厂公钥)
userAddr	是	string	工厂 (扫码枪) 地址

返回示例

```
{
  "statusCode": 200,
  "success": true,
  "message": "",
  "data": ""
}
```

返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	string	查询信息

状态码

服务异常状态码：

- 501：用户服务异常
- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码：

- 601：key 不合法
- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息
- 605：信息上链确认中
- 606：入场时候信息不合法，db 中标签 key 的公钥和所传公钥不一致
- 607：扫码操作，参数验签失败

9.1.2 标签存证 (updateInfo)

简要描述：

- 将溯源信息存入锁定的标签中

请求 URL： - `http://localhost:3001/updateInfo`

请求方式： - POST

参数：

参数名	必选	类型	说明
key	是	number	编号
info	是	string	流转信息
signInfo	是	string	加密信息 (工厂私钥加签扫码信息)
userAddr	是	string	当前用户

返回示例

```
{
  "statusCode": 200,
  "success": true,
  "message": "",
  "data": ""
}
```

返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	string	查询信息

状态码

服务异常状态码

- 501：用户服务异常
- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码：

- 601：key 不合法
- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息

- 605：信息上链确认中
- 606：入场时候信息不合法，db 中标签 key 的公钥和所传公钥不一致
- 607：扫码操作，参数验签失败

9.1.3 标签解锁 (open)

简要描述：

- 出场

请求 URL：- `http://localhost:3001/open`

请求方式：- POST

参数：

参数名	必选	类型	说明
key	是	number	编号
userAddr	是	string	当前用户
signInfo	是	string	加密信息 (工厂私钥加签扫码信息)

返回示例

```
{
  "statusCode": 200,
  "success": true,
  "message": "",
  "data": ""
}
```

返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	string	查询信息

状态码

服务异常状态码：

- 501：用户服务异常
- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码：

- 601：key 不合法
- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息
- 605：信息上链确认中
- 606：入场时候信息不合法，db 中标签 key 的公钥和所传公钥不一致
- 607：扫码操作，参数验签失败

9.2 溯源校验模块

9.2.1 查询溯源信息 (getTraceInfo)

简要描述：

- 查询标签溯源信息

请求 URL： - `http://localhost:3001/getTraceInfo`

请求方式： - POST

参数：

参数名	必选	类型	说明
key	是	number	编号

返回示例

```
{
    "success": true,
    "statusCode": 200,
    "message": null,
    "data": {
        "owner": "0x289f7eb882f......829921b92b26aa0",
        "traceInfo": "add trace info",
        "traceValue": "5open&MFwwDQYJKoZIhvcNA",
        "traceArr":      ["MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBALuJxUO+8LPrWf7vAt83c5euY",
↪ "006888b10d274c8aa1c286b30257ea05a73851cf8cb24a1ebb", "add trace info",
↪"8b255422022b369374ea05f74ea352f0d7ff8f7f17f"]
    }
}
```


返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	string	查询信息 (json)

data 参数名	类型	说明
owner	string	标签所属人
traceInfo	string	溯源明文
traceValue	string	溯源信息 db 原文
traceArr	array	溯源信息数组

状态码

服务异常状态码：

- 501：用户服务异常
- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码：

- 601：key 不合法

- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息
- 605：信息上链确认中

9.2.2 公钥校验 (publicKeyCheck)

简要描述：

- 公钥校验，开发模式

请求 URL： - `http://localhost:3001/publicKeyCheck`

请求方式： - POST

参数：

参数名	必选	类型	说明
ownerPublicKey	是	string	用户扫码公钥
checkInfo	是	string	明文
signature	是	string	密文

返回示例

```
{"success":true,"statusCode":200,"message":null,"data":true}
```

返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	boolean	查询信息 (true-成功，false-失败)

状态码

服务异常状态码：

- 501：用户服务异常

- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码：

- 601：key 不合法
- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息
- 605：信息上链确认中

9.2.3 溯源校验 (traceCheck)

简要描述：

- 溯源校验，校验溯源信息是否正确

请求 URL： - `http://localhost:3001/traceCheck`

请求方式： - POST

参数：

参数名	必选	类型	说明
key	是	number	编号
proof	是	string	proof 证明路径

返回示例

```
{
    "success": true,
    "statusCode": 200,
    "message": null,
    "data": {
        "result": 1,
        "rootHash": "0x888b10d274c8aa1c286b30257ea05a7",
        "txHash": "0xcab08ce37f4530ed81960a761ddfc3a0a2c2a4a5d3d251fec1c1286843ea50ba"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	string	查询信息 (json)

data 参数名	类型	说明
result	number	校验结果（1-成功 0-失败）
rootHash	string	merkle 树根 hash
txHash	string	上链交易 hash

状态码

服务异常状态码：

- 501：用户服务异常
- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码

- 601：key 不合法
- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息
- 605：信息上链确认中

9.2.4 溯源申诉 (traceAppeal)

简要描述：

- 溯源申诉，溯源信息不正确，进行申诉修正

请求 URL：- `http://localhost:3001/traceAppeal`

请求方式：- POST

参数：

参数名	必选	类型	说明
key	是	number	编号
proof	是	string	proof 证明路径

返回示例

```
 {
    "success": true,
    "statusCode": 200,
    "message": null,
    "data": {
        "result": 1
    }
}
```

返回参数说明

参数名	类型	说明
statusCode	number	状态码
success	boolean	true / false
message	string	异常说明
data	string	查询信息 (json)

data 参数名	类型	说明
result	number	校验结果（1-申诉成功 0-申诉失败）

状态码

服务异常状态码

- 501：用户服务异常
- 502：rpc 调用异常
- 503：update db 异常
- 504：find db 异常

业务异常状态码

- 601：key 不合法
- 602：状态不合法，不能执行入场，或者不能场内流转
- 603：当前用户无操作权限
- 604：录入重复信息
- 605：信息上链确认中