# 节点API

### web3\_sha3

返回指定数据的Keccak-256（不同于标准的SHA3-256算法）哈希值。.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"web3\_sha3","params":["0x68656c6c6f20776f726c64"],"id":67} |
| GET | api.php?action=sha3&data=0x68656c6c6f20776f726c64 |

##### 参数

1. DATA - 要计算SHA3哈希的数据

##### 返回

DATA - 指定字符串的SHA3结果.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"web3\_sha3","params":["0x68656c6c6f20776f726c64"],"id":64}'

// Result

{

"id":64,

"jsonrpc": "2.0",

"result": "0x47173285a8d7341e5e972fc677286384f802f8ef42a5ec5f03bbfa254cb01fad"

}

### net\_listening

检测节点主动侦听网络连接的状态，则返回 true.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"net\_listening","params":[""],"id":67} |
| GET | api.php?action=listening |

##### 返回

Boolean - 如果节点正在主动侦听网络连接，则返回true否则 false.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"net\_listening","params":[],"id":67}'

// Result

{

"id":67,

"jsonrpc":"2.0",

"result":true

}

### net\_peerCount

返回当前节点所连接的端对端节点数量.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"net\_peerCount","params":[],"id":67} |
| GET | api.php?action=peercount |

##### 返回

QUANTITY -  连接PEER的数量

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"net\_peerCount","params":[],"id":74}'

// Result

{

"id":74,

"jsonrpc": "2.0",

"result": "0x2" // 2

}

### fwl\_protocolVersion

返回软件版本.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_protocolVersion","params":[""],"id":67} |
| GET | api.php?action=protocolVersion |

##### 返回

String -当前软件版本

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_protocolVersion","params":[],"id":67}'

// Result

{

"id":67,

"jsonrpc": "2.0",

"result": "54"

}

### fwl\_syncing

对于已经同步的客户端，该调用返回一个描述同步状态的对象；对于未同步客户端，返回false。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_syncing","params":[""],"id":67} |
| GET | api.php?action=syncing |

##### 返回

Object|Boolean，同步状态对象或false。同步对象的结构如下：

* startingBlock：QUANTITY- 导入开始的块
* currentBlock：QUANTITY- 当前块
* highestBlock：QUANTITY- 估计的最高区块

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_syncing","params":[],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": {

startingBlock: '0x384',

currentBlock: '0x386',

highestBlock: '0x454'

}

}

// Or when not syncing

{

"id":1,

"jsonrpc": "2.0",

"result": false

}

### fwl\_coinbase

返回节点coinbase地址.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_coinbase","params":[""],"id":67} |
| GET | api.php?action=coinbase |

##### 返回

DATA，20个字节 - 当前coinbase地址.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_coinbase","params":[],"id":64}'

// Result

{

"id":64,

"jsonrpc": "2.0",

"result": "0x407d73d8a49eeb85d32cf465507dd71d507100c1"

}

### fwl\_hashrate

返回节点每秒可算出的哈希数量.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_hashrate","params":[""],"id":67} |
| GET | api.php?action=hashrate |

##### 返回

QUANTITY -每秒的哈希数.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_ashrate","params":[],"id":71}'

// Result

{

"id":71,

"jsonrpc": "2.0",

"result": "0x38a"

}

### fwl\_gasPrice

返回当前的gas价格，单位：wei.

##### 调用方式：（注意RPC参数中gasPrice的大小写）

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_gasPrice","params":[""],"id":67} |
| GET | api.php?action=gasprice |

##### 返回

QUANTITY -当前GAS价格的整数.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_gasPrice","params":[],"id":73}'

// Result

{

"id":73,

"jsonrpc": "2.0",

"result": "0x09184e72a000" // 10000000000000

}

### fwl\_accounts

返回节点持有的地址列表.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_accounts","params":[""],"id":67} |
| GET | api.php?action=accounts |

##### 返回

Array of DATA，20字节 - 节点拥有的地址.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_accounts","params":[],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": ["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]

}

### fwl\_blockNumber

返回最近块的块号.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_blockNumber","params":[""],"id":67} |
| GET | api.php?action=blocknumber |

##### 返回

QUANTITY -节点当前的块号.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_blockNumber","params":[],"id":83}'

// Result

{

"id":83,

"jsonrpc": "2.0",

"result": "0x4b7" // 1207

}

### fwl\_getBalance

返回给定地址帐户的余额.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getBalance","params":["0x593b932397926a9787e4350eadd6952668c84de1","latest"],"id":67} |
| GET | api.php?action=getBalance&address=0x593b932397926a9787e4350eadd6952668c84de1 |

##### 参数

1. DATA，20字节 - 地址。
2. QUANTITY|TAG- 整数块号或字符串"latest"，"earliest"或者"pending"

params: [

'0x407d73d8a49eeb85d32cf465507dd71d507100c1',

'latest'

]

##### 返回

QUANTITY -  当前余额的整数.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getBalance","params":["0x407d73d8a49eeb85d32cf465507dd71d507100c1", "latest"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x0234c8a3397aab58" // 158972490234375000

}

### fwl\_getStorageAt

返回指定地址存储位置的值.

##### 调用方式：（注意：RPC调用中fwl\_getStorageAt的大小写）

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getStorageAt","params":[""],"id":67} |
| GET | api.php?action=getstorageat&param1=data1& param2=data2& param 3=data3(注释：这里用param1、param2、param3表示三个参数) |

##### 参数

1. address，20字节 - 存储地址。
2. QUANTITY - 存储位置的整数。
3. QUANTITY|TAG- 整数块号或字符串"latest"，"earliest"或者"pending"。

##### 返回

DATA -  指定位置的存储的值.

##### 例子

根据要提取的存储计算正确的位置。考虑下面的合约，由0x391694e7e0b0cce554cb130d723a9d27458f9298 部署在地址0x295a70b2de5e3953354a6a8344e616ed314d7251.

contract Storage {

uint pos0;

mapping(address => uint) pos1;

function Storage() {

pos0 = 1234;

pos1[msg.sender] = 5678;

}

}

提取pos0的值很直接：

curl -X POST --data '{"jsonrpc":"2.0", "method": "fwl\_getStorageAt", "params": ["0x295a70b2de5e3953354a6a8344e616ed314d7251", "0x0", "latest"], "id": 1}' localhost:8545

响应结果：

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

要提取映射表中的成员就难一些了。映射表中成员位置的计算如下：

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

这意味着为了提取pos1["0x391694e7e0b0cce554cb130d723a9d27458f9298"]的值，我们需要如下计算：

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

节点控制台自带的web3库可以用来进行这个计算：

> var key = "000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" + "0000000000000000000000000000000000000000000000000000000000000001"

undefined

> web3.sha3(key, {"encoding": "hex"})

"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9"

现在可以提取指定位置的值了：

curl -X POST --data '{"jsonrpc":"2.0", "method": "fwl\_getStorageAt", "params": ["0x295a70b2de5e3953354a6a8344e616ed314d7251", "0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9", "latest"], "id": 1}' localhost:8545

相应结果如下：

{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000000000000162e"}

### fwl\_getUncleCountByBlockHash

返回指定块的叔伯数量，使用哈希指定块。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getUncleCountByBlockHash","params":[""],"id":67} |
| GET | api.php?action=getUncleCountByBlockHash |

##### 参数

1. DATA，32字节 - 块的散列

params: [

'0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'

]

##### 返回

QUANTITY -指定块的叔伯数量，整数.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getUncleCountByBlockHash","params":["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x1" // 1

}

### fwl\_getUncleCountByBlockNumber

返回指定块的叔伯数量，使用块编号指定块.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getUncleCountByBlockNumber","params":[""],"id":67} |
| GET | api.php?action=getUncleCountByBlockNumber |

##### 参数

1. QUANTITY|TAG - QUANTITY|TAG- 块号的整数，或字符串"latest", "earliest" or "pending"

params: [

'0xe8', // 232

]

##### 返回

QUANTITY -指定块的叔伯数量

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getUncleCountByBlockNumber","params":["0xe8"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x1" // 1

}

### fwl\_getCode

返回指定地址的代码.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_ getCode ","params":[""],"id":67} |
| GET | api.php?action=getCode |

##### 参数

1. DATA，20字节 - 地址
2. QUANTITY|TAG- 整数块号或字符串"latest"，"earliest"或者"pending"

params: [

'0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',

'0x2' // 2

]

##### 返回

DATA -给定地址的代码

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getCode","params":["0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b", "0x2"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078202905091905056"

}

### fwl\_call

立刻执行一个新的消息调用，无需在区块链上创建交易.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_ call,"params":[""],"id":67} |
| GET | api.php?action=call |

##### 参数

1. Object - 交易调用对象

* from：DATA，20字节 - （可选）交易发送地址。
* to：DATA，20字节 - 交易处理的地址。
* gas：QUANTITY - （可选）为交易执行提供的GAS整数。fwl\_call消耗零点GAS，但某些执行可能需要此参数。
* gasPrice：QUANTITY - （可选）用于每种付费GAS的gasPrice的整数
* value：QUANTITY - （可选）与此交易一起发送的值的整数
* data：DATA - （可选）方法签名和编码参数的哈希值。

1. QUANTITY|TAG- 整数块号或字符串"latest"，"earliest"或者"pending"

##### 返回

DATA -所执行合约的返回值

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_call","params":[{see above}],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x"

}

### fwl\_estimateGas

执行并估算一个交易需要的gas用量。该次交易不会写入区块链。注意，由于多种原因，例如EVM的机制 及节点旳性能，估算的数值可能比实际用量大的多。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_ estimategas,"params":[""],"id":67} |
| GET | api.php?action=estimategas |

##### 参数

参考fwl\_call调用的参数，所有的属性都是可选的。如果没有指定gas用量上限，geth将使用挂起块的gas上限。 在这种情况下，返回的gas估算量可能不足以执行实际的交易。

##### 返回

QUANTITY - 使用的GAS量

.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_estimateGas","params":[{see above}],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x5208" // 21000

}

### fwl\_getBlockByHash

返回具有指定哈希的块。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getBlockByHash","params":[""],"id":67} |
| GET | api.php?action=getBlockByHash&param1=0x0000000&param2=true |

##### 参数

1. Param1：DATA，32字节 - 块的散列。
2. Param2：Boolean-为true时返回完整的交易对象，否则仅返回交易哈希.

params: [

'0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',

true

]

##### 返回

Object- 匹配的块对象，如果未找到块则返回null，结构如下：

* number: QUANTITY - 块编号，挂起块为null
* hash: DATA, 32 Bytes - 块哈希，挂起块为null
* parentHash: DATA, 32 Bytes - 父块的哈希
* nonce: DATA, 8 Bytes - 生成的pow哈希，挂起块为null
* sha3Uncles: DATA, 32 Bytes - 块中叔伯数据的SHA3哈希
* logsBloom: DATA, 256 Bytes - 快日志的bloom过滤器，挂起块为null
* transactionsRoot: DATA, 32 Bytes - 块中的交易树根节点
* stateRoot: DATA, 32 Bytes - 块最终状态树的根节点
* receiptsRoot: DATA, 32 Bytes - 块交易收据树的根节点
* miner: DATA, 20 Bytes - 挖矿奖励的接收账户
* difficulty: QUANTITY - 块难度，整数
* totalDifficulty: QUANTITY - 截止到本块的链上总难度
* extraData: DATA - 块额外数据
* size: QUANTITY - 本块字节数
* gasLimit: QUANTITY - 本块允许的最大gas用量
* gasUsed: QUANTITY - 本块中所有交易使用的总gas用量
* timestamp: QUANTITY - 块时间戳
* transactions: Array - 交易对象数组，或32字节长的交易哈希数组
* uncles: Array - 叔伯哈希数组

.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getBlockByHash","params":["0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331", true],"id":1}'

// Result

{

"id":1,

"jsonrpc":"2.0",

"result": {

"number": "0x1b4", // 436

"hash": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",

"parentHash": "0x9646252be9520f6e71339a8df9c55e4d7619deeb018d2a3f2d21fc165dde5eb5",

"nonce": "0xe04d296d2460cfb8472af2c5fd05b5a214109c25688d3704aed5484f9a7792f2",

"sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",

"logsBloom": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",

"transactionsRoot": "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",

"stateRoot": "0xd5855eb08b3387c0af375e9cdb6acfc05eb8f519e419b874b6ff2ffda7ed1dff",

"miner": "0x4e65fda2159562a496f9f3522f89122a3088497a",

"difficulty": "0x027f07", // 163591

"totalDifficulty": "0x027f07", // 163591

"extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",

"size": "0x027f07", // 163591

"gasLimit": "0x9f759", // 653145

"gasUsed": "0x9f759", // 653145

"timestamp": "0x54e34e8e" // 1424182926

"transactions": [{...},{ ... }]

"uncles": ["0x1606e5...", "0xd5145a9..."]

}

}

### fwl\_getBlockByNumber

返回指定编号的块。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getBlockByNumber","params":["0x10",true],"id":67} |
| GET | api.php?action=getBlockBynumber &param1=0x10&param2=true |

##### 参数

1. QUANTITY|TAG - 整数块编号，或字符串"earliest"、"latest" 或"pending"
2. Boolean - 为true时返回完整的交易对象，否则仅返回交易哈希.

params: [

'0x1b4', // 436

true

]

##### 返回

见 fwl\_getBlockByHash

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getBlockByNumber","params":["0x1b4", true],"id":1}'

### fwl\_getUncleByBlockHashAndIndex

返回具有指定哈希的块具有指定索引位置的叔伯。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_getUncleByBlockHashAndIndex ","params":[""],"id":67} |
| GET | api.php?action=getUncleByBlockHashAndIndex &param1=&param2= |

##### 参数

1. DATA, 32字节 - 块哈希
2. QUANTITY - 叔伯索引位置.

params: [

'0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b',

'0x0' // 0

]

##### 返回

见 fwl\_getBlockByHash

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getUncleByBlockHashAndIndex","params":["0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b", "0x0"],"id":1}'

### fwl\_getUncleByBlockNumberAndIndex

返回具有指定编号的块内具有指定索引序号的叔伯。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_ getUncleByBlockNumberAndIndex ","params":[""],"id":67} |
| GET | api.php?action=getUncleByBlockNumberAndIndex |

##### 参数

* QUANTITY|TAG - 整数块编号，或字符串"earliest"、"latest" 或"pending"
* QUANTITY - 叔伯在块内的索引序号

params: [

'0x29c', // 668

'0x0' // 0

]

##### 返回

见 fwl\_getBlockByHash

##### 注意：叔伯块内不包含交易。

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getUncleByBlockNumberAndIndex","params":["0x29c", "0x0"],"id":1}'

### fwl\_compileSolidity

返回编译后的solidity代码。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_compileSolidity ","params":[""],"id":67} |
| GET | api.php?action=compileSolidity &param1=xxx |

##### 参数

1. String -源代solidity源代码.

params: [

"contract test { function multiply(uint a) 返回(uint d) { return a \* 7; } }",

]

##### 返回

DATA -编译后的源代码.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_compileSolidity","params":["contract test { function multiply(uint a) 返回(uint d) { return a \* 7; } }"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": {

"code": "0x605880600c6000396000f3006000357c010000000000000000000000000000000000000000000000000000000090048063c6888fa114602e57005b603d6004803590602001506047565b8060005260206000f35b60006007820290506053565b91905056",

"info": {

"source": "contract test {\n function multiply(uint a) constant 返回(uint d) {\n return a \* 7;\n }\n}\n",

"language": "Solidity",

"languageVersion": "0",

"compilerVersion": "0.9.19",

"abiDefinition": [

{

"constant": true,

"inputs": [

{

"name": "a",

"type": "uint256"

}

],

"name": "multiply",

"outputs": [

{

"name": "d",

"type": "uint256"

}

],

"type": "function"

}

],

"userDoc": {

"methods": {}

},

"developerDoc": {

"methods": {}

}

}

}

### fwl\_compileLLL

返回编译后的LLL代码。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"fwl\_compileLLL","params":[""],"id":67} |
| GET | api.php?action=fwl\_compileLLL &param1=xxx |

##### 参数

1. String -  LLL源代码.

params: [

"(returnlll (suicide (caller)))",

]

##### 返回

DATA -  编译后的LLL源代码.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_compileLLL","params":["(returnlll (suicide (caller)))"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b6021600435602b565b8060005260206000f35b600081600702905091905056" // the compiled source code

}

### fwl\_compileSerpent

返回编译后的Serpent代码.

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":" compileSerpent ","params":[""],"id":67} |
| GET | api.php?action=compileSerpent&param1=xxx |

##### 参数

1. String - Serpent源代码.

params: [

"/\* some serpent \*/",

]

##### 返回

DATA -  编译后的Serpent代码.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_compileSerpent","params":["/\* some serpent \*/"],"id":1}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b6021600435602b565b8060005260206000f35b600081600702905091905056" // the compiled source code

}

### fwl\_newFilter

基于给定的选项创建一个过滤器对象，接收状态变化时的通知。要检查状态是否变化， 请调用fwl\_getFilterChanges。

关于特定主题过滤器的说明：主题是顺序相关的。如果一个交易的日志有主题[A, B]，那么将被 以下的主题过滤器匹配：

* [] 任何主题
* [A] 先匹配A主题
* [null, B] 先匹配其他主题，再匹配B主题
* [A, B] 先匹配A主题，再匹配B主题，最后匹配其他主题
* [[A, B], [A, B]] "先匹配A主题或B主题，再匹配A主题或B主题，最后匹配其他主题

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":" newFilter ","params":[""],"id":67} |
| GET | api.php?action=newFilter |

##### 参数

Object -过滤器选项对象：

* fromBlock: QUANTITY|TAG - 可选，默认值："latest"。整数块编号，或字符串"latesr"表示最后挖出的块，"pending"或"earliest"用于未挖出的交易。
* toBlock: QUANTITY|TAG - 可选，默认值："latest"。整数块编号，或字符串"latesr"表示最后挖出的块，"pending"或"earliest"用于未挖出的交易。
* address: DATA|Array, 20字节 - 可选，合约地址或生成日志的一组地址
* topics: Array of DATA, - 可选，32字节主题数组，每个主题可以是数组或使用or选项连接.

params: [{

"fromBlock": "0x1",

"toBlock": "0x2",

"address": "0x8888f1f195afa192cfee860698584c030f4c9db1",

"topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b", null, ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b", "0x0000000000000000000000000aff3454fce5edbc8cca8697c15331677e6ebccc"]]

}]

##### 返回

QUANTITY -过滤器编号.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_newFilter","params":[{"topics":["0x12341234"]}],"id":73}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x1" // 1

}

### fwl\_newBlockFilter

在节点中创建一个过滤器，以便当新块生成时进行通知。要检查状态是否变化， 请调用fwl\_getFilterChanges。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"newBlockFilter","params":[""],"id":67} |
| GET | api.php?action=newBlockFilter |

##### 参数

没有

##### 返回

QUANTITY -过滤器编号

.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_newBlockFilter","params":[],"id":73}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x1" // 1

}

### fwl\_newPendingTransactionFilter

在节点中创建一个过滤器，以便当产生挂起交易时进行通知。 要检查状态是否发生变化，请调用fwl\_getFilterChanges。。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":" newPendingTransactionFilter ","params":[""],"id":67} |
| GET | api.php?action=newPendingTransactionFilter |

##### 参数

没有

##### 返回

QUANTITY -过滤器编号

.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_newPendingTransactionFilter","params":[],"id":73}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": "0x1" // 1

}

### fwl\_uninstallFilter

写在具有指定编号的过滤器。当不在需要监听时，总是需要执行该调用。另外，过滤器 如果在一定时间内未接收到fwl\_getFilterChanges调用会自动超时。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"uninstallFilter","params":[""],"id":67} |
| GET | api.php?action=uninstallFilter &param1=xxxx |

##### 参数

1. QUANTITY -  过滤器编号

params: [

"0xb" // 11

]

##### 返回

Boolean- 如果成功卸载则返回true，否则返回false.

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_uninstallFilter","params":["0xb"],"id":73}'

// Result

{

"id":1,

"jsonrpc": "2.0",

"result": true

}

### fwl\_getFilterChanges

轮询指定的过滤器，并返回自上次轮询之后新生成的日志数组。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":" getFilterChanges ","params":[""],"id":67} |
| GET | api.php?action=getFilterChanges &param1=xxx |

##### 参数

1. QUANTITY - 过滤器ID.

params: [

"0x16" // 22

]

##### 返回

Array - 日志对象数组，如果没有新生成的日志，则返回空数组

使用fwl\_newBlockFilter创建的过滤器将返回块哈希（32字节），例如["0x3454645634534..."]。

使用fwl\_newPendingTransactionFilter创建的过滤器将返回交易哈希 (32字节)，例如["0x6345343454645..."]。

使用fwl\_newFilter创建的过滤器，日志对象具有如下参数：

* removed: TAG - 如果日志已被删除则返回true，如果是有效日志则返回false
* logIndex: QUANTITY - 日志在块内的索引序号。对于挂起日志，该值为null
* transactionIndex: QUANTITY - 创建日志的交易索引序号，对于挂起日志，该值为null
* transactionHash: DATA, 32字节 - 创建该日志的交易的哈希。对于挂起日志，该值为null
* blockHash: DATA, 32字节 - 该日志所在块的哈希。对于挂起日志，该值为null
* blockNumber: QUANTITY - 该日志所在块的编号。对于挂起日志，该值为null
* address: DATA, 20字节 - 该日志的源地址
* data: DATA - 包含该日志的一个或多个32字节无索引参数
* topics: Array of DATA -0~4个32字节索引日志参数的数据。在solidity中，第一个主题是事件签名，例如Deposit(address,bytes32,uint256)，除非你声明的是匿名事件

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getFilterChanges","params":["0x16"],"id":73}'

// Result

{

"id":1,

"jsonrpc":"2.0",

"result": [{

"logIndex": "0x1", // 1

"blockNumber":"0x1b4" // 436

"blockHash": "0x8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcfdf829c5a142f1fccd7d",

"transactionHash": "0xdf829c5a142f1fccd7d8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcf",

"transactionIndex": "0x0", // 0

"address": "0x16c5785ac562ff41e2dcfdf829c5a142f1fccd7d",

"data":"0x0000000000000000000000000000000000000000000000000000000000000000",

"topics": ["0x59ebeb90bc63057b6515673c3ecf9438e5058bca0f92585014eced636878c9a5"]

},{

...

}]

}

### fwl\_getFilterLogs

返回指定编号过滤器中的全部日志。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":"getFilterLogs","params":[""],"id":67} |
| GET | api.php?action=getFilterLogs&param1=xxx |

##### 参数

1. QUANTITY - 过滤器ID.

params: [

"0x16" // 22

]

##### 返回

见fwl\_getFilterChanges

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getFilterLogs","params":["0x16"],"id":74}'

### fwl\_getWork

返回当前块的哈希、种子哈希、以及要满足的边界条件，即目标。

##### 调用方式：

| **节点** | **调用方法** |
| --- | --- |
| RPC | {"jsonrpc":"2.0","method":" getWork ","params":[""],"id":67} |
| GET | api.php?action=getWork |

##### 参数

没有

##### 返回

Array - 数组，具有如下成员：

* DATA, 32字节 - 当前块头的pow-hash
* DATA, 32字节 - 用于DAG的种子哈希
* DATA, 32字节 - 边界条件，目标， 2^256 / difficulty..

##### 例子

// Request

curl -X POST --data '{"jsonrpc":"2.0","method":"fwl\_getWork","params":[],"id":73}'

// Result

{

"id":1,

"jsonrpc":"2.0",

"result": [

"0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",

"0x5EED00000000000000000000000000005EED0000000000000000000000000000",

"0xd1ff1c01710000000000000000000000d1ff1c01710000000000000000000000"

]

}

# 节点服务部安装操作指南

## 1 、安装环境要求

CenOS7 操作系统；硬件要求4G内存、300G以上的硬盘、CPU双核3G；网络带宽10M以上。

建议环境如下：

|  |  |
| --- | --- |
| **操作系统环境** | |
| 操作系统 | Centos 7.4 64位 |
| 内核 | 3.10.0-693.2.2.el7.x86\_64 |
| CPU | Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz |
| 内存 | 8G |
| 存储空间 | 500GB |

|  |  |
| --- | --- |
| **软件环境准备** | |
| gethfw | 私链客户端程序 |
| Golang | V1.9.4 |
| Nodejs | V8.7.0 |
| Httpd | V2.4.6 |
| Php | V3.0.0 |
| Mongod | V3.4.15 |

## 2 、安装规划

需要规划如下参数：1、每个服务器节点数量；节点端口、RPC端口、浏览器端口、GET端口；2、创世区块的预置数字资产配置；3、区块的网络ID。

单服务器节点规划试例如下：

在服务器数量有限的情况下，可以在单服务器上同时部署多个节点服务，通过端口号进行区分，以下以单服务器上开启2个节点客户端为例：

|  |  |  |
| --- | --- | --- |
| 系统服务 | 端口号 | 备注 |
| gethfw | 8145、8546 | RPC端口号，其中8145对外服务，8546用于区块链浏览器解析区块数据。 |
| gethfw | 8801、8802 | 节点间内部通信端口号 |
| Node | 8080 | 区块浏览器服务端口号 |
| Sshd | 22 | 远程接入管理端口号 |
| Mongod | 27017 | 区块浏览器解析区块数据后写入mongodb数据库供查询 |
| Httpd | 80 | API接口服务端口号 |

## 3 、 安装包文件结构说明

提供的nodeinit.tgz 包解压后目录结构

node

---- autocheck 服务启停脚本

---- start.sh 服务启动脚本

---- autocheck.sh 服务运行期间监控脚本

---- start8145.sh 启动本服务器第一个节点脚本

---- start8546.sh 启动本服务器第二个节点脚本

---- node8001 节点1

---- keystore 存放初创账户目录

---- node8002 节点2

---- init 初始化配置文件目录

---- genesis.json 初创文件配置

---- init.sh 节点初始化

---- install.sh 软件安装

---- config/sshd\_conf 视个人情况调整SSH配置,需要重启服务,解决连接后一断时间没操作ssh客户端自动关闭问题

---- config/httpd.conf mod\_rewirte开启

## 4 、安装步骤

第一步，执行系统环境安装脚本（目的安装：apache、gcc、mongodb、php、golang）

node/init/install.sh ，安装脚本完成节点的安装，详细执行如下：

yum install httpd

yum install npm

yum install bzip2 bzip

yum install -y gcc make gcc-c++ openssl-devel wget

rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm

yum install php70w

yum install php70w-pecl-mongodb

cp /node/init/config/httpd.conf /etc/httpd/conf/

cp /node/init/config/sshd\_config /etc/ssh/

第二步，下载安装node-v8.7.0

cd /root

wget https://npm.taobao.org/mirrors/node/v8.7.0/node-v8.7.0-linux-x64.tar.xz

tar -xvf node-v8.7.0-linux-x64.tar.xz

ln -s /root/node-v8.7.0-linux-x64/bin/node /usr/local/bin/node

ln -s /root/node-v8.7.0-linux-x64/bin/npm /usr/local/bin/npm

第三步，安装mongodb

将我们提供的mongodb-org-3.4.repo文件上传到服务器的//etc/yum.repos.d/目录然后通过yum进行安装。

yum install -y mongodb-org

mongo start

第四步，安装go，并编译geth

将下载好的golang包上传到root目录。

tar zxvf go1.10.linux-amd64.tar.gz

cp /root/go/bin/\* /usr/bin –rf

mv go /usr/local

进入go-ethereum目录，运行make，运行完毕后在build/bin目录下生成geth

第四步，拷贝区块链节点程序

gethfw是区块链节点主程序，将其拷贝到/usr/bin目录下

cp build/bin/geth /usr/bin/gethfw

第五步，节点初创和静态节点部署

1.初创账号生成方式：

创建数据存储目录，第一台为8001-8002，第二台为8003-8004以此类推

mkdir /node/node8001

mkdir /node/node8002

gethfw account new - -datadir "/node/node8001"

执行后会要求设置账户的unlock口令,**请记住配置的口令**.

2.修改创世文件，配置网络ID、区块链ID等内容，初创文件genesis.json注意事项：

\* chainId，链id，该区块链的链id，以和其他区块链区别，可以采用创建的时间作为id；

\* 私链连接的节点必须采用同一个genesis.json文件进行 init 初始化操作;

\* 添加"byzantiumBlock"说明，数字12表示后续的区块高度超过了12个块，表示永久确认,才能正确返回接口的status，如txlist;

\* alloc 初创资产配置,给某个创世账户预配置一笔创世的资产;

\* timestamp 为LINUX时间秒数转十六进制，表示第0块的产生时间

\* difficulty 初始的难度序数

{

"config":{

"chainId":1807030325,

"homesteadBlock":0,

"byzantiumBlock": 12,

"eip155Block":0,

"eip158Block":0

},

"nonce":"0x0000000000000042",

"mixhash":"0x0000000000000000000000000000000000000000000000000000000000000000",

"difficulty":"0x0001",

"alloc":{"0xfccb43b1f140ee10122941ce78e35d4144455b88":{"balance":"900000000000000000000000000000"}},

"timestamp":"0x5b3bcd2f",

"parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000",

"extraData":"",

"gasLimit":"0xffffffff"

}

3.初始化节点

#!/bin/sh

/usr/bin/gethfw --datadir /node/node8001 init ./genesis.json

/usr/bin/gethfw --datadir /node/node8002 init ./genesis.json

node8001 ~ 8002为节点数据存储目录，可以根据实际情况配置，如：三台服务器，每台配置2个节点客户端，第一台配置node8001~8002，第二台配置8003~8004，第三台配置8005~8006，更多服务器的话以此类推.

4.修改启动脚本和监测脚本

修改/node/autocheck/start.sh以及start8145.sh、start8546.sh修改其中的IP地址和网络ID（用于区别与其他区块链的地址，所有节点必须一致才能互联，区块ID和网络ID请保持一致），只修改这两项即可

#!/bin/sh

/usr/bin/gethfw --targetgaslimit 4294967295 --identity "node8001" --datadir /node/node8001 --port 8801 --rpc --rpcaddr "192.168.10.215" --rpcport 8145 --rpccorsdomain "http://192.168.10.215:8080" --rpcapi "admin,debug,eth,net,personal,shh,txpool,web3" --networkid 1807030325002 --nodiscover --mine 2>/dev/null &

/usr/bin/gethfw --targetgaslimit 4294967295 --identity "node8002" --datadir /node/node8002 --port 8802 --rpc --rpcaddr "192.168.10.215" --rpcport 8546 --rpccorsdomain "http://192.168.10.215:8081" --rpcapi "admin,debug,eth,net,personal,shh,txpool,web3" --networkid 1807030325002 --nodiscover 2>/dev/null &

修改完毕后，启动节点

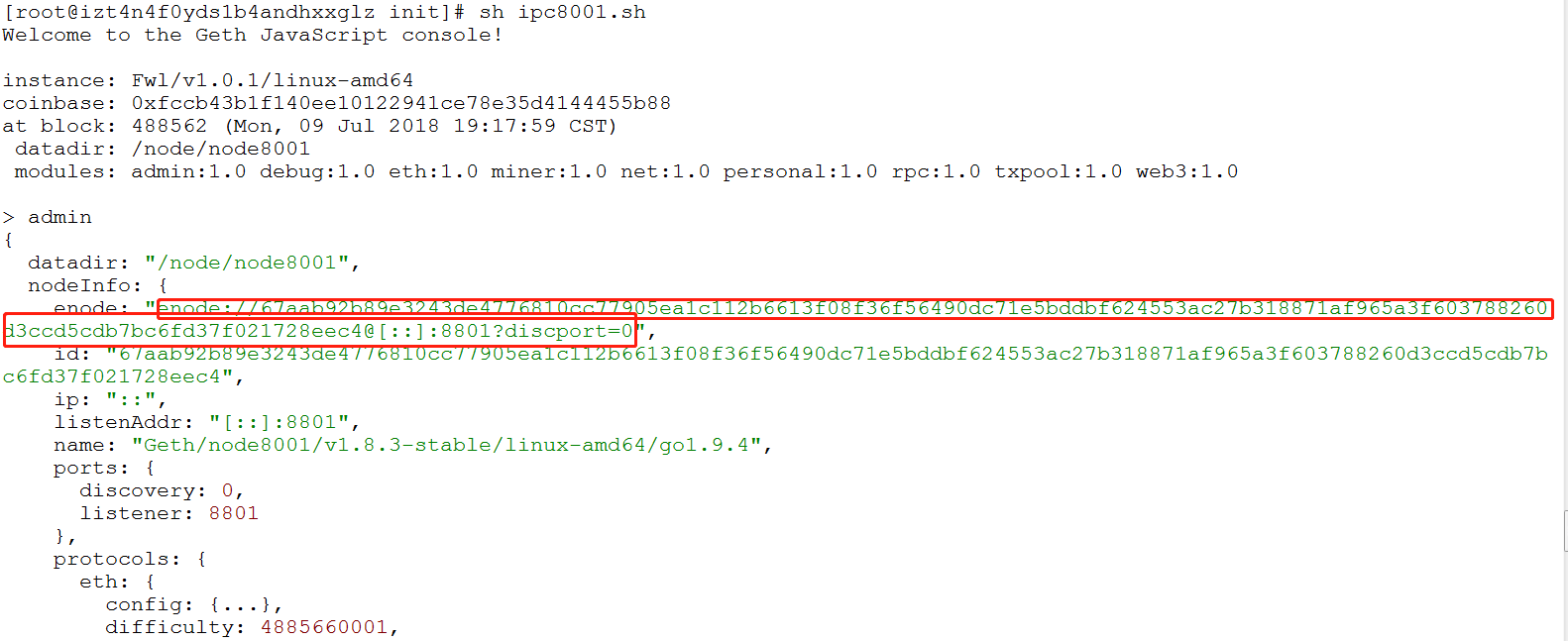
/node/autocheck/shart.sh

第六步 添加静态节点连接

Node的信息查看：（通过ipc在console下面用admin就可以查看，如下信息,）

查看8001节点：

/usr/bin/gethfw attach ipc:/node/node8001/geth.ipc



采用相同的方法，将几台服务器上所有节点的enode信息记录到static-nodes.json文件上(注:enode中没有IP地址，需要修改一下，将IP填入，替换[::]部分)。

静态节点添加方式：（节点配置文件：路径/node/node8001/static-nodes.json；其他节点类似均要放如下配置文件）

内容如下：（当前六个节点的静态配置）

[

"enode://67aab92b89e3243de4776810cc77905ea1c112b6613f08f36f56490dc71e5bddbf624553ac27b318871af965a3f603788260d3ccd5cdb7bc6fd37f021728eec4@172.21.206.48:8801?discport=0",

"enode://a0826a95944a7e518d18d2f07ef1d511254a5430693ec5076dd3cfd8464f5c6ac1005e852408b4304723e8983acffc16ca95c6b19fbad7d593f853ed4125f1a1@172.21.206.48:8802?discport=0",

"enode://ee6d1b72325e1a661e3acd2dd7005309ea0c9a7ec1a2bd03668c66b32782cdff3e3a07a70f95f86f99c7e30fc0af1f44d12370279d3dae1766f4d4cb5a0e1a35@172.21.206.49:8801?discport=0",

"enode://83a78e9df113c8f1a9b8fc77bde9608666a5445da5e10bfb9557593924c34b05aae6ac7e8c19c4fad02c41a516aea264f7e8c827c19e0414079d7c29080e4308@172.21.206.49:8802?discport=0"

]

通过ipc可以查看节点运行后的连接情况

admin.peers



第七步，其他工具部署（安装包：/root/api\_explorer\_remix.tgz）

mv /root/api\_explorer\_remix.tgz /var/www

cd /var/www

tar zxvf api\_explorer\_remix.tgz

第四步，服务配置

1. HTTP\_GET接口服务配置：修改/var/www/html/api.php,修改其中url地址，端口为任何RPC服务器的端口。
2. 浏览器服务器的配置

\* 修改/var/www/explore/config.json修改:

{

"nodeAddr": "172.21.206.49",#修改这里

"gethPort": 8546, #修改这里

"startBlock": 0,

"quiet": true,

"syncAll": true,

"patch": true,

"patchBlocks": 100,

"bulkSize": 100,

"settings": {

"symbol": "ACT",

"name": "ACT",

"title": "ACT",

"author": "ACT",

"contact": "mailto:ACT@ACT.com",

"about": "ACT",

"rss": "#",

"reddit": "#",

"twitter": "#",

"linkedin": "#",

"github": "#",

"logo": "/img/explorer-logo.png", #浏览器首页的logo图片显示名称

"customCss": "green-haze.min.css",

"copyright": "2018 &copy; woutla.",

"useEthFiat": false,

"miners": {

"0xfccb43b1f140ee10122941ce78e35d4144455b88": "ACT",#修改别名,显示在统计，算力分布图位置,以下几个预留做配置更新.

"0xc91716199ccde49dc4fafaeb68925127ac80443f": "F2Pool",

"0x9eab4b0fc468a7f5d46228bf5a76cb52370d068d": "NanoPool",

"0x8c5535afdbdeea80adedc955420f684931bf91e0": "MiningPoolHub",

"0x4750e296949b747df1585aa67beee8be903dd560": "UUPool",

"0xef224fa5fad302b51f38898f4df499d7af127af0": "91pool",

"0x00d29bfdf5f8d2d0466da4b948f37692ca50867a": "2miners",

"0x4c2b4e716883a2c3f6b980b70b577e54b9441060": "ETCPool PL",

"0xd144e30a0571aaf0d0c050070ac435deba461fab": "Clona Network"

}

}

}

\* 修改/var/www/explorer/app.js

找到 app.set('port', process.env.PORT || 8080); // 8080为启动的端口号

\* 修改/var/www/explorder/tools/stats.js

var web3 = new Web3(new Web3.providers.HttpProvider('http://172.21.206.49:8546')); 访问的节点RPC端口，需要修改IP

\* 修改IP /var/www/explorer/tools/patcher.js

var web3 = new Web3(new Web3.providers.HttpProvider('http://172.21.206.49:' +

config.gethPort.toString()));

\* 修改IP /var/www/explorer/tools/blockHelper.js

var web3 = new Web3(new Web3.providers.HttpProvider('http://172.21.206.49:' +

config.gethPort.toString()));

1. 配置系统自启动脚本：（chmod +x /etc/rc.d/rc.local）

其中rc.local中需要增加如下：（实现开机自动启动）

sh /node/autocheck/start.sh &

sh /node/autocheck/autocheck.sh &

配置服务自启动（启动服务，执行一次就可以）

systemctl enable mongod

systemctl enable httpd

1. 配置或关闭防火墙：

systemctl disable firewalld.service

第五步，服务启动及监控配置

/node/autocheck/autocheck.sh

第六步

如果是在原有节点上重新部署（重新初始化），应在初始化完毕后清空mongoDB的内容。

#mongo

#use blockDB

#db.dropDatabase()

下载钱包，锁仓BIK 成为节点