搭建最简单的以太坊环境

下面我用geth，带领大家搭建一个基础联盟链环境

一、传世区块

首先创建两个文件夹，分别写入创世区块文件，例如genesis.json，文件内容如下

```
{
  "config": {
        "chainId": 15,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "coinbase" : "0x0000000000000000000000000000000000000000",
    "difficulty" : "0x40000",
    "extraData" : "",
    "gasLimit" : "0xffffffff",
    "nonce" : "0x0000000000000042",
    "mixhash" : "0x0000000000000000000000000000000000000000000000000000000000000000
",
    "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000
000",
    "timestamp" : "0x00",
    "alloc": { }
}
```

分别进入刚才创建genesis.json的目录下，初始化创世区块，前面我们已经写好了两个genesis.json的配置文件，下面就执行一下初始化操作，涉及到操作参数为init。

```
MacBook-Pro:geth zzs$ geth --datadir ./data-init1/ init genesis.json
MacBook-Pro:geth zzs$ geth --datadir ./data-init2/ init genesis.json
```

将初始化json文件放在geth同级目录下，如果放在其他目录下，指定具体的路径即可。同时创建了一个data-init1目录专门存储节点数据，执行完成会发现在该目录下多出两个目录，一个为geth一个为keystore。其中geth里面放数据相关信息，keystore里面放加密过的私钥文件。执行时打印日志如下：

```
WARN [12-28|19:12:03] No etherbase set and no accounts found as default
```

```
INFO [12-28|19:12:03] Allocated cache and file handles         database=/Users/zzs/
develop/eth/geth/data-init1/geth/chaindata cache=16 handles=16
INFO [12-28|19:12:03] Writing custom genesis block
INFO [12-28|19:12:03] Successfully wrote genesis state         database=chaindata
                                            hash=942f59…a2588a
INFO [12-28|19:12:03] Allocated cache and file handles         database=/Users/zzs/
develop/eth/geth/data-init1/geth/lightchaindata cache=16 handles=16
INFO [12-28|19:12:03] Writing custom genesis block
INFO [12-28|19:12:03] Successfully wrote genesis state         database=lightchaind
ata                                           hash=942f59…a2588a
```

二、启动控制台

根据具体的操作系统，开两个窗口来启动两个节点。这里有一点需要注意的是，虽然是两个节点，但他们的启动程序都是geth，只不过datadir目录不同而已。在第一个窗口执行以下命令启动一个节点，注意启动之后不要关闭窗口。

```
geth --datadir ./data-init1/ --networkid 88 --nodiscover console
```

- networkid 指定网路ID，确保不使用1-4，1-4系统内置使用，我们随便写一个端口，比如88。
- nodiscover 此参数确保geth不去寻找peers节点，主要是为了控制联盟链连入的节点。

这里我们需要注意的是在启动第一个节点时并没有指定port参数，因此此处采用了默认的port，也就是30303。以下为执行时打印的日志，并进入控制台。通过日志我们也可以发现端口为30303。

```
WARN [12-28|19:23:16] No etherbase set and no accounts found as default
INFO [12-28|19:23:16] Starting peer-to-peer node               instance=Geth/v1.7.3
-stable-4bb3c89d/darwin-amd64/go1.9.2
INFO [12-28|19:23:16] Allocated cache and file handles         database=/Users/zzs/
develop/eth/geth/data-init2/geth/chaindata cache=128 handles=1024
WARN [12-28|19:23:16] Upgrading database to use lookup entries
INFO [12-28|19:23:16] Database deduplication successful        deduped=0
INFO [12-28|19:23:16] Initialised chain configuration          config="{ChainID: 72
 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil> EIP155: 0 EIP158: 0 Byzant
ium: <nil> Engine: unknown}"
INFO [12-28|19:23:16] Disk storage enabled for ethash caches   dir=/Users/zzs/devel
op/eth/geth/data-init2/geth/ethash count=3
INFO [12-28|19:23:16] Disk storage enabled for ethash DAGs     dir=/Users/zzs/.etha
sh                                 count=2
INFO [12-28|19:23:16] Initialising Ethereum protocol           versions="[63 62]" n
etwork=88
```

```
INFO [12-28|19:23:16] Loaded most recent local header          number=0 hash=942f59
…a2588a td=16384
INFO [12-28|19:23:16] Loaded most recent local full block      number=0 hash=942f59
…a2588a td=16384
INFO [12-28|19:23:16] Loaded most recent local fast block      number=0 hash=942f59
…a2588a td=16384
INFO [12-28|19:23:16] Regenerated local transaction journal    transactions=0 accou
nts=0
INFO [12-28|19:23:16] Starting P2P networking
INFO [12-28|19:23:16] RLPx listener up                         self="enode://aa621c
010c685665ef217044dac4d57f4d1d682c682a5b3f92ca23b40982383240a05b680060ce8b0ce020a96
c49c9c2c3628c4ea3281845211bd4cf4f03b35c@[::]:30303?discport=0"
INFO [12-28|19:23:16] IPC endpoint opened: /Users/zzs/develop/eth/geth/data-init2/g
eth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.3-stable-4bb3c89d/darwin-amd64/go1.9.2
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool
:1.0 web3:1.0

> INFO [12-28|19:23:18] Mapped network port                       proto=tcp extport=
30303 intport=30303 interface="UPNP IGDv1-IP1"
```

如果日志中能显示出“Welcome to the Geth JavaScript console!”，则说明启动成功了。

下面在另外一个窗口，换一个端口，比如30306，再换一下datadir，来启动第二个节点。

```
geth --datadir ./data-init2/ --port 30306 --networkid 88 --nodiscover console
```

执行上面命令，完成节点2的启动，并进入控制台。

三、添加coinbase账户

现在我们就在第一个节点上创建一个账户，具体在控制台操作命令如下：

```
> personal.listAccounts
[]
> personal.newAccount("123456")
"0x60c8abe58c9dbc52a4ee9f8510f1799c432c0f3e"
```

上面的命令先是查看了节点下的地址，结果为空，然后创建了一个秘密为123456的账号。同样的，在另外一个窗口我们执行同样的命令：

```
> personal.listAccounts
[]
> personal.newAccount("123456")
"0x02b7344004c45465796f779b7b95d7912c2ef572"
```

这样，两个节点就拥有了两个地址。同时，在它们的keystore目录下对应生成了加密的私钥文件。
我们也可以再次执行list命令查看添加账户之后的情况。同时可以执行以下命令查看coinbase账号：

```
> eth.coinbase
"0x02b7344004c45465796f779b7b95d7912c2ef572"
```

由于只有一个地址，因此该地址就作为coinbase地址。如果想查看更多的信息可以执行以下命令：

```
> personal.listWallets
[{
    accounts: [{
        address: "0x02b7344004c45465796f779b7b95d7912c2ef572",
        url: "keystore:///Users/zzs/develop/eth/geth/data-init2/keystore/UTC--2017-12-28T11-36-18.185974427Z--02b7344004c45465796f779b7b95d7912c2ef572"
    }],
    status: "Locked",
    url: "keystore:///Users/zzs/develop/eth/geth/data-init2/keystore/UTC--2017-12-28T11-36-18.185974427Z--02b7344004c45465796f779b7b95d7912c2ef572"
}]
```

这里不仅打印了账户信息，还打印出了私钥存储的位置和账户状态等信息。

四、联盟链互通

上面分别是在两个节点上进行的操作，下面我们需要把两个节点之间建立起链接。首先，我们执行以下命令查看以下节点的peers的情况。

```
> admin.peers
[]
```

发现节点并没有链接上任何其他节点，这也是我们的nodiscover参数发挥了效果。下面就通过分享enode地址的方式来让两个节点建立链接。

```
> admin.nodeInfo.enode
"enode://67cd8dc437e06000fe0ff97f37f1ff667c17b35bdcd8da06434c0588b386ac92ee9c7f85440b477690db44ee91d2dfca5c13b9baac3cc4c479813aa7e2397be6@[::]:30303?discport=0"
```

通过上面命令，我们获得了节点2的encode信息。这是geth用来连接到不同节点的enode信息，在这些不同的节点它们能够分享交易和成功挖掘信息。
其实这个信息如果留心的话，在启动节点的打印日志中已经打印出每个节点的encode信息。比如：

```
INFO [12-28|19:23:16] RLPx listener up                         self="enode://aa621c010c685665ef217044dac4d57f4d1d682c682a5b3f92ca23b40982383240a05b680060ce8b0ce020a96c49c9c2c3628c4ea3281845211bd4cf4f03b35c@[::]:30303?discport=0"
```

现在，我们要告知一个节点，另外一个节点的encode信息。首先复制节点2的日志中self等号后面的信息，在节点1的控制台执行以下命令：

```
> admin.addPeer("enode://e4b51e8bf54c82660e3123ff1d996cb4d9234bc1e8312b5144cc6e2d3538b33e8f8f438dad2f08cd968a408e31f5781535eaf1f1e5944e9af7c962ddd05a9594@[::]:30306?discport=0")
true
```

返回true，说明执行成功。再次验证一下：

```
> admin.peers
[{
    caps: ["eth/63"],
    id: "aa621c010c685665ef217044dac4d57f4d1d682c682a5b3f92ca23b40982383240a05b680060ce8b0ce020a96c49c9c2c3628c4ea3281845211bd4cf4f03b35c",
    name: "Geth/v1.7.3-stable-4bb3c89d/darwin-amd64/go1.9.2",
    network: {
      localAddress: "[::1]:49426",
      remoteAddress: "[::1]:30306"
    },
    protocols: {
      eth: {
        difficulty: 16384,
        head: "0x942f596f99dc8879b426b59080824662e1f97587353d087487fea0a0e2a2588a",
        version: 63
      }
    }
}]
```

发现节点1已经有一个peer了，同时我们可以看到remoteAddress: “[::1]:30306”，正是我们节点2的端口信息。在节点2执行admin.peers会发现有类似的信息，指向的peer正是节点1的。

五、查询余额并挖矿

执行查看余额命令：

```
> eth.getBalance(eth.coinbase)
0
```

发现两个节点的账号余额都为0。在节点1执行miner.start()进行挖矿，执行miner.stop()停止挖矿。停止挖矿的时候开业忽略控制台输出，只要正确拼写命令回车即可。当我们在节点1执行挖矿时，我们会发现节点2的控制台出现了这样的日志信息：

六、执行挖矿

```
>miner.start()
> INFO [12-28|20:05:32] Block synchronisation started
INFO [12-28|20:05:33] Imported new state entries                count=1 elapsed=47.6
61µs processed=1 pending=0 retry=0 duplicate=0 unexpected=0
WARN [12-28|20:05:33] Discarded bad propagated block            number=1 hash=ab49ba
…1cf32f
INFO [12-28|20:05:33] Imported new block headers                count=2 elapsed=9.20
8ms  number=2 hash=738225…000e3b ignored=0
INFO [12-28|20:05:33] Imported new chain segment                blocks=2 txs=0 mgas=
0.000 elapsed=1.724ms  mgasps=0.000 number=2 hash=738225…000e3b
INFO [12-28|20:05:33] Fast sync complete, auto disabling
INFO [12-28|20:05:34] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=5.978ms  mgasps=0.000 number=3 hash=b069a9…426060
INFO [12-28|20:05:38] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=6.930ms  mgasps=0.000 number=4 hash=21217e…526253
INFO [12-28|20:05:41] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=6.419ms  mgasps=0.000 number=5 hash=3fa6ff…cf2794
INFO [12-28|20:05:43] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=6.557ms  mgasps=0.000 number=6 hash=4c35b9…78b3ec
INFO [12-28|20:05:45] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=6.514ms  mgasps=0.000 number=7 hash=328e62…1bd3d3
INFO [12-28|20:05:46] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=6.513ms  mgasps=0.000 number=8 hash=12287e…0465b5
INFO [12-28|20:06:19] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=7.048ms  mgasps=0.000 number=9 hash=8e844b…b99d6c
INFO [12-28|20:06:22] Imported new chain segment                blocks=1 txs=0 mgas=
0.000 elapsed=8.156ms  mgasps=0.000 number=10 hash=159b36…d4dde5
```

```
INFO [12-28|20:06:24] Imported new chain segment               blocks=1 txs=0 mgas=
0.000 elapsed=6.549ms  mgasps=0.000 number=11 hash=969100…5658a5
```

也就是说，节点1挖矿，节点2在同步数据信息。停止节点1的挖矿，并查看coinbase地址金额：

```
> miner.stop()
true
> eth.getBalance(eth.coinbase)
140000000000000000000
> eth.coinbase
"0x60c8abe58c9dbc52a4ee9f8510f1799c432c0f3e"
```

这里我们知道了节点一种的地址信息和余额信息，那我们拿节点1的这个地址在节点2的控制台查询一下信息：
> eth.getBalance("0x60c8abe58c9dbc52a4ee9f8510f1799c432c0f3e")
140000000000000000000

很显然，节点2中也能查询到节点1中地址的余额。以上信息说明，节点1和节点2的数据是完全同步的。

七、交易转账

现在我们从节点1的coinbase账户发一笔交易到节点2的coinbase账户。

```
> personal.unlockAccount("0x60c8abe58c9dbc52a4ee9f8510f1799c432c0f3e")
Unlock account 0x60c8abe58c9dbc52a4ee9f8510f1799c432c0f3e
Passphrase:
true
> eth.sendTransaction({from: eth.coinbase, to: '0x02b7344004c45465796f779b7b95d7912
c2ef572', value: 100000000})
INFO [12-28|20:13:21] Submitted transaction                    fullhash=0xc32e40f0f
606a6368aa2c4c6a27db20f5a1d728fb9ef1f50a0410f4889e095a0 recipient=0x02b7344004c4546
5796F779B7b95d7912C2eF572
"0xc32e40f0f606a6368aa2c4c6a27db20f5a1d728fb9ef1f50a0410f4889e095a0"
```

解锁账户并发送1币交易信息。现在查看一下节点2的地址内是否有余额：

```
> eth.getBalance("0x02b7344004c45465796f779b7b95d7912c2ef572")
0
```

发现余额是0，为什么呢？因为虽然我们发起了交易，单并没有矿工挖矿打包交易。再次执行

miner.start()。再次查询，就会发现节点2的coinbase地址已经有金额了。

```
> eth.getBalance("0x02b7344004c45465796f779b7b95d7912c2ef572")
100000000
```

通过以上的操作我们已经建立了一个拥有两个节点的联盟链，如果想建立更多节点的联盟链，可以此添加新的节点。

八、解锁

```
>personal.unlockAccount(eth.accounts[0])
```

九、交易

```
>eth.sendTransaction({from: eth.coinbase, to: "0xa8d003b95ed969e68db9eb077ad333a3df10450a", gas:1000, 'gasPrice': web3.toWei(300, 'gwei'), "value": "1"})
```

写一个简单的智能合约作为测试

```
contract Sample {
uint public value;

      function Sample(uint v){
                value=v;
      }

      function set(uint v){
                value=v;
      }

      function get() constant returns (uint){
                return value;
      }
}

在remix [连接](https://remix.ethereum.org/#version=soljson-v0.4.11+commit.68ef5810.js)得到ABI接口和合约的二进制代码
```

abi为如下格式

```
[
{
    "constant": true,
    "inputs": [],
    "name": "value",
    "outputs": [
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "payable": false,
    "type": "function",
    "stateMutability": "view"
},
{
    "constant": false,
    "inputs": [
        {
            "name": "v",
            "type": "uint256"
        }
    ],
    "name": "set",
    "outputs": [],
    "payable": false,
    "type": "function",
    "stateMutability": "nonpayable"
},
{
    "constant": true,
    "inputs": [],
    "name": "get",
    "outputs": [
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "payable": false,
    "type": "function",
    "stateMutability": "view"
},
{
    "inputs": [
        {
            "name": "v",
            "type": "uint256"
```

```
            }
        ],
        "payable": false,
        "type": "constructor",
        "stateMutability": "nonpayable"
    }
    ]
```

登录如下bejson网站,连接,将刚才的abi去掉空格

[{"constant":true,"inputs":[],"name":"value","outputs":
[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMutability":"view"},
{"constant":false,"inputs":[{"name":"v","type":"uint256"}],"name":"set","outputs":
[],"payable":false,"type":"function","stateMutability":"nonpayable"},{"constant":true,"inputs":
[],"name":"get","outputs":
[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMutability":"view"},
{"inputs":
[{"name":"v","type":"uint256"}],"payable":false,"type":"constructor","stateMutability":"nonpaya
ble"}]

然后在第一个终端输入

```
> abi=[{"constant":true,"inputs":[],"name":"value","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMutability":"view"},{"constant":false,"inputs":[{"name":"v","type":"uint256"}],"name":"set","outputs":[],"payable":false,"type":"function","stateMutability":"nonpayable"},{"constant":true,"inputs":[],"name":"get","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMutability":"view"},{"inputs":[{"name":"v","type":"uint256"}],"payable":false,"type":"constructor","stateMutability":"nonpayable"}]

[{
    constant: true,
    inputs: [],
    name: "value",
    outputs: [{
        name: "",
        type: "uint256"
    }],
    payable: false,
    type: "function"
}, {
    constant: false,
    inputs: [{
        name: "v",
        type: "uint256"
    }],
    name: "set",
    outputs: [],
    payable: false,
    type: "function"
}, {
    constant: true,
    inputs: [],
    name: "get",
    outputs: [{
        name: "",
        type: "uint256"
    }],
    payable: false,
    type: "function"
}, {
    inputs: [{
        name: "v",
        type: "uint256"
    }],
    payable: false,
    type: "constructor"
}]
```

然后在输入

```
> sample=eth.contract(abi)

{
  abi: [{
      constant: true,
      inputs: [],
      name: "value",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      constant: false,
      inputs: [{...}],
      name: "set",
      outputs: [],
      payable: false,
      type: "function"
  }, {
      constant: true,
      inputs: [],
      name: "get",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      inputs: [{...}],
      payable: false,
      type: "constructor"
  }],
  eth: {
    accounts: ["0x4c57e7e9c2f728046ddc6e96052056a241bdbd0a", "0xe82e2f0a5abd8774767
b9751659976f9c4f59181"],
    blockNumber: 6225,
    coinbase: "0x4c57e7e9c2f728046ddc6e96052056a241bdbd0a",
    compile: {
      lll: function(),
      serpent: function(),
      solidity: function()
    },
    defaultAccount: undefined,
    defaultBlock: "latest",
    gasPrice: 18000000000,
    hashrate: 0,
    mining: false,
    pendingTransactions: [],
```

```
    protocolVersion: "0x3f",
    syncing: false,
    call: function(),
    contract: function(abi),
    estimateGas: function(),
    filter: function(fil, callback),
    getAccounts: function(callback),
    getBalance: function(),
    getBlock: function(),
    getBlockNumber: function(callback),
    getBlockTransactionCount: function(),
    getBlockUncleCount: function(),
    getCode: function(),
    getCoinbase: function(callback),
    getCompilers: function(),
    getGasPrice: function(callback),
    getHashrate: function(callback),
    getMining: function(callback),
    getPendingTransactions: function(callback),
    getProtocolVersion: function(callback),
    getRawTransaction: function(),
    getRawTransactionFromBlock: function(),
    getStorageAt: function(),
    getSyncing: function(callback),
    getTransaction: function(),
    getTransactionCount: function(),
    getTransactionFromBlock: function(),
    getTransactionReceipt: function(),
    getUncle: function(),
    getWork: function(),
    iban: function(iban),
    icapNamereg: function(),
    isSyncing: function(callback),
    namereg: function(),
    resend: function(),
    sendIBANTransaction: function(),
    sendRawTransaction: function(),
    sendTransaction: function(),
    sign: function(),
    signTransaction: function(),
    submitTransaction: function(),
    submitWork: function()
  },
  at: function(address, callback),
  getData: function(),
  new: function()
}
```

然后输入合约二进制代码

```
 > SampleHEX="0x6060604052341561000c57fe5b6040516020806101 3a8339810160405280805190602001909190505 05b806000819055505b505b60f9806100416000396000f3006060604052600035 7c0100000000000000000000000000000000000000000000000000000090046 3ffffffff1680633fa4f24514604e57806360fe47b1146071578063 6d4ce63c14608e575bfe5b3415605557fe5b605b60b1565b6040518082815260200191505060405180910390f35b3415607857fe5b608c6004808035906020019091905050606b7565b005b3415609557fe5b609b60c2565b60405180828152602001915050604051809103 90f35b60005481565b806000819055505b50565b600060005490505b905600a165627a7a7230582056284 25e21eefb62faf95e98fbde766001427f1fc9a6ad9856c3ed2ae336a5430029"
```

把合约代码部署上链

```
thesample=sample.new(1,{from:eth.accounts[0],data:SampleHEX,gas:3000000})

{
  abi: [{
      constant: true,
      inputs: [],
      name: "value",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      constant: false,
      inputs: [{...}],
      name: "set",
      outputs: [],
      payable: false,
      type: "function"
  }, {
      constant: true,
      inputs: [],
      name: "get",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      inputs: [{...}],
      payable: false,
      type: "constructor"
  }],
  address: undefined,
  transactionHash: "0xee74bcb4461c9712ec9aca96a5a3a4c3c64be1213854d519fc8e5432b554f7a1"
}
```

挖矿miner.start() 经过一段时间后 miner.stop()停止，然后查看交易细节

```
> samplerecpt=eth.getTransactionReceipt("0xee74bcb4461c9712ec9aca96a5a3a4c3c64be121
3854d519fc8e5432b554f7a1")

{
  blockHash: "0xddba16545af882835fb9a69a0e5f3b9287c61664837d5ea0068b38575cb665c5",
  blockNumber: 6246,
  contractAddress: "0x7504fa9d64ab290844b82660d43b310f8fba0276",
  cumulativeGasUsed: 141836,
  from: "0x4c57e7e9c2f728046ddc6e96052056a241bdbd0a",
  gasUsed: 141836,
  logs: [],
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000",
  root: "0xd1093ecaca9cc0d10e82a533a15feccedf7ff5c79fb3ebd9366ec0b35dbef478",
  to: null,
  transactionHash: "0xee74bcb4461c9712ec9aca96a5a3a4c3c64be1213854d519fc8e5432b554f
7a1",
  transactionIndex: 0
}
```

合约命名

```
> samplecontract=sample.at("0x7504fa9d64ab290844b82660d43b310f8fba0276")

{
  abi: [{
      constant: true,
      inputs: [],
      name: "value",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      constant: false,
      inputs: [{...}],
      name: "set",
      outputs: [],
      payable: false,
```

```
      type: "function"
  }, {
      constant: true,
      inputs: [],
      name: "get",
      outputs: [{...}],
      payable: false,
      type: "function"
  }, {
      inputs: [{...}],
      payable: false,
      type: "constructor"
  }],
  address: "0x7504fa9d64ab290844b82660d43b310f8fba0276",
  transactionHash: null,
  allEvents: function(),
  get: function(),
  set: function(),
  value: function()
}
```

合约查看功能函数get（）,然后调用set()函数，再get()查看时已经改变了

```
> samplecontract.get.call()
1

> samplecontract.set.sendTransaction(9, {from:eth.accounts[0], gas:3000000})

"0x822ee6fb4caceb7e844c533f7f3bc57806f7cb3676fb3066eb848cca46b2f38a"

> samplecontract.get.call()
```

我们再打开一个终端，打开cluster1的peer02的控制台，直接at到上一个终端部署的智能合约地址并进行set操作

```
> abi=[{"constant":true,"inputs":[],"name":"value","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMutability":"view"},{"constant":false,"inputs":[{"name":"v","type":"uint256"}],"name":"set","outputs":[],"payable":false,"type":"function","stateMutability":"nonpayable"},{"constant":true,"inputs":[],"name":"get","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMutability":"view"},{"inputs":[{"name":"v","type":"uint256"}],"payable":false,"type":"constructor","stateMutability":"nonpayable"}]


> sample=eth.contract(abi)
```

```
SampleHEX="0x6060604052341561000c57fe5b6040516020806101 3a8339810160405280805190602001909190505050 5b806000819055505b505b60f9806100416000396000f300606060405260003 57c0100000000000000000000000000000000000000000000000000000000900463ffffffff1680633fa4f24514604e57806360fe47b11460715780636d4ce63c14608e575bfe5b3415605557fe5b605b60b1565b6040518082815260200191505060405180910390f35b3415607857fe5b608c60048080359060200190919050506060b7565b005b3415609557fe5b609b60c2565b60405180828152602001915050604051809103 90f35b60005481565b806000819055505b50565b600060005490505b905600a165627a7a723058205628425e21eefb62faf95e98fbde766001427f1fc9a6ad9856c3ed2ae336a5430029"
```

直接把合约地址赋值并进行set操作

```
samplecontract=sample.at("0x7504fa9d64ab290844b82660d43b310f8fba0276")

> samplecontract.get.call()

9
```