实验环境

在 windows 下的 VMware Workstation Pro 虚拟机中搭建 Ubuntu 系统进行实验。

以太坊环境搭建

安装 go 语言环境

解压 go 安装包，并在 home 目录下新建 go 文件夹：

```
sudo tar -C /usr/local -xzf go1.7.1.linux-amd64.tar.gz
mkdir $HOME/go
```

在.bashrc 文件中进行如下环境配置：

```
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

安装配置完成，测试查看 go 版本：

```
qhh0809@ubuntu:~$ go version
go version go1.11.1 linux/amd64
```

安装 node 和 npm，solc

安装 node 和 npm 命令：

```
sudo apt-get install nodejs-legacy nodejs
sudo apt-get install npm
sudo apt-get install solc
```

安装完成查看版本号：

```
qhh0809@ubuntu:~$ node -v
v4.2.6
qhh0809@ubuntu:~$ npm -v
3.5.2
```

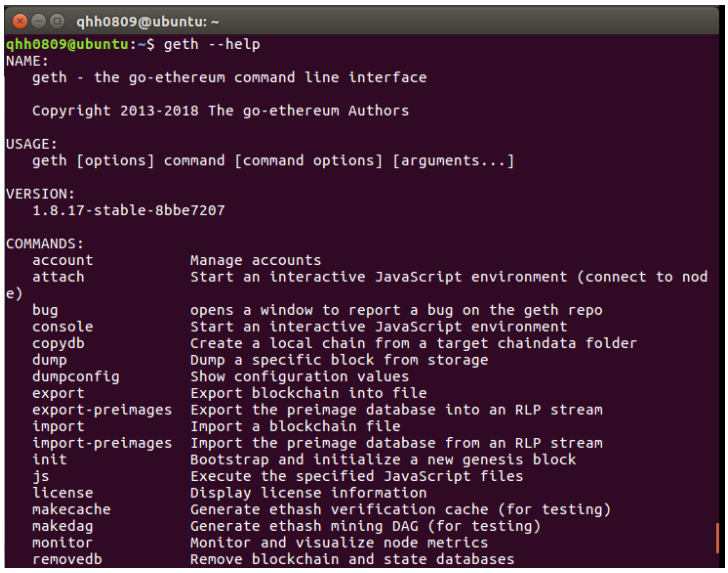
安装 Ethereum

安装以太坊 Ethereum 命令：

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -y ppa:ethereum/ethereum
```

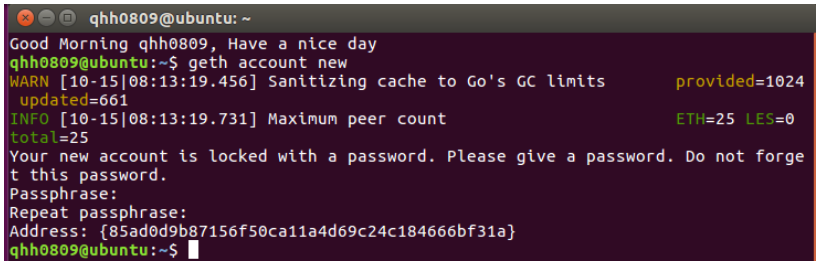
```
$ sudo apt-get update
$ sudo apt-get install ethereum
```

安装完成后，使用 geth –help 命令进行检查安装情况：



搭建以太坊私有链

创建账号： geth account new



在根目录（~/）下编写创世块文件 genesis.json：

```
{
  "nonce":"0x0000000000000042",
  "mixhash":"0x0000000000000000000000000000000000000000000000000000000000000000",
  "difficulty": "0x4000",
  "alloc": {},
  "coinbase":"0x0000000000000000000000000000000000000000",
  "timestamp": "0x00",
  "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
  "extraData": "0x00000000",
  "gasLimit":"0xffffffff",
  "config" :{
```

```
    "chainId": 68,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  }
}
```

```
qhh0809@ubuntu:~$ cat genesis.json
{
  "nonce":"0x0000000000000042",
  "mixhash":"0x0000000000000000000000000000000000000000000000000000000000000000"
,
  "difficulty": "0x4000",
  "alloc": {},
  "coinbase":"0x0000000000000000000000000000000000000000",
  "timestamp": "0x00",
  "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000
00",
  "extraData": "0x00000000",
  "gasLimit":"0xffffffff",
  "config" :{
    "chainId": 68,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  }
}
```

初始化创世块文件：

```
geth --datadir "~/.ethereum" init ./genesis.json
```

```
qhh0809@ubuntu:~$ geth --datadir “~/.ethereum” init ./genesis.json
WARN [10-15|08:41:02.692] Sanitizing cache to Go's GC limits       provided=1024
 updated=661
INFO [10-15|08:41:02.694] Maximum peer count                       ETH=25 LES=0
total=25
INFO [10-15|08:41:02.696] Allocated cache and file handles         database=/hom
e/qhh0809/“~/.ethereum”/geth/chaindata cache=16 handles=16
INFO [10-15|08:41:02.713] Writing custom genesis block
INFO [10-15|08:41:02.714] Persisted trie from memory database      nodes=0 size=
0.00B time=31.151µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-15|08:41:02.714] Successfully wrote genesis state         database=chai
ndata                                     hash=fd6d40…aee892
INFO [10-15|08:41:02.714] Allocated cache and file handles         database=/hom
e/qhh0809/“~/.ethereum”/geth/lightchaindata cache=16 handles=16
INFO [10-15|08:41:02.721] Writing custom genesis block
INFO [10-15|08:41:02.721] Persisted trie from memory database      nodes=0 size=
0.00B time=2.868µs  gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-15|08:41:02.721] Successfully wrote genesis state         database=ligh
tchaindata                                hash=fd6d40…aee892
```

初始化完成后，就有了创世区块，就可以启动以太坊：

```
geth --datadir "~/.ethereum" console
```

```
qhh0809@ubuntu: ~
INFO [10-15|08:44:05.120] Loaded most recent local header          number=0 hash
=fd6d40…aee892 td=16384 age=49y5mo4w
INFO [10-15|08:44:05.120] Loaded most recent local full block      number=0 hash
=fd6d40…aee892 td=16384 age=49y5mo4w
INFO [10-15|08:44:05.121] Loaded most recent local fast block      number=0 hash
=fd6d40…aee892 td=16384 age=49y5mo4w
INFO [10-15|08:44:05.121] Regenerated local transaction journal    transactions=
0 accounts=0
INFO [10-15|08:44:05.155] Starting P2P networking
INFO [10-15|08:44:07.289] UDP listener up                          self=enode://
1012f3d9ebd46875df95afd01ee75abc5bfafb7bc0dd5f436a27f1408aad2147e2c7a36afc841ca9
7d7af6904bdf96e7eb8e5a233401f69ed4172c328bdf16a6@[::]:30303
INFO [10-15|08:44:07.292] RLPx listener up                         self=enode://
1012f3d9ebd46875df95afd01ee75abc5bfafb7bc0dd5f436a27f1408aad2147e2c7a36afc841ca9
7d7af6904bdf96e7eb8e5a233401f69ed4172c328bdf16a6@[::]:30303
INFO [10-15|08:44:07.324] IPC endpoint opened                      url=/home/qhh
0809/“~/.ethereum”/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.17-stable-8bbe7207/linux-amd64/go1.10
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0
rpc:1.0 txpool:1.0 web3:1.0

>
```

如上：已成功进入 geth 命令行。

开始挖矿，输入 miner.start()命令，出现如下错误：

```
> miner.start()
INFO [10-15|08:45:48.662] Updated mining threads                   threads=2
INFO [10-15|08:45:48.662] Transaction pool price threshold updated price=1000000
000
ERROR[10-15|08:45:48.662] Cannot start mining without etherbase    err="etherbas
e must be explicitly specified"
Error: etherbase missing: etherbase must be explicitly specified
    at web3.js:3143:20
    at web3.js:6347:15
    at web3.js:5081:36
    at <anonymous>:1:1

>
```

经查，是因为没有账户，可能之前创建的账户路径不对，eth.accounts 仍无内容，所以重新创建账户：

```
personal.newAccount("qhh666888")
```

查看 eth.accounts,可得下图：

```
> eth.accounts
[]
> personal.newAccount("qhh666888")
"0x4e53abca6a1012a8bde75040819f11b98db28a2b"
> eth.accounts
["0x4e53abca6a1012a8bde75040819f11b98db28a2b"]
>
```

创建账户成功后，重新开始挖矿 miner.start()：

```
> miner.start()
INFO [10-15|08:53:36.944] Updated mining threads                   threads=2
INFO [10-15|08:53:36.944] Transaction pool price threshold updated price=1000000
000
INFO [10-15|08:53:36.944] Etherbase automatically configured       address=0x4E5
3abca6a1012A8bDe75040819F11B98Db28A2b
null
> INFO [10-15|08:53:36.962] Commit new mining work                   number=1 se
alhash=cfe59d…85ed01 uncles=0 txs=0 gas=0 fees=0 elapsed=17.709ms
INFO [10-15|08:53:40.916] Generating DAG in progress               epoch=0 perce
ntage=0 elapsed=3.052s
INFO [10-15|08:53:45.302] Generating DAG in progress               epoch=0 perce
ntage=1 elapsed=7.438s
INFO [10-15|08:53:48.926] Generating DAG in progress               epoch=0 perce
ntage=2 elapsed=11.062s
INFO [10-15|08:53:51.968] Generating DAG in progress               epoch=0 perce
ntage=3 elapsed=14.104s
INFO [10-15|08:53:55.198] Generating DAG in progress               epoch=0 perce
ntage=4 elapsed=17.334s
INFO [10-15|08:53:59.014] Generating DAG in progress               epoch=0 perce
ntage=5 elapsed=21.150s
INFO [10-15|08:54:06.569] Generating DAG in progress               epoch=0 perce
ntage=6 elapsed=28.706s
```

挖矿成功：

```
> miner.stop(INFO [10-15|09:23:32.935] Successfully sealed new block
number=102 sealhash=829340…43339d hash=f9d89d…e33447 elapsed=2.151s
INFO [10-15|09:23:32.935] 🔗block reached canonical chain          number=95  ha
sh=884ede…31f3dd
INFO [10-15|09:23:32.935] 🔨mined potential block                  number=102 ha
sh=f9d89d…e33447
INFO [10-15|09:23:32.935] Commit new mining work                   number=103 se
alhash=dc71a7…19a371 uncles=0 txs=0 gas=0 fees=0 elapsed=108.358µs
> miner.stop()
null
> eth.blockNumber
102
> eth.getBalance(eth.accounts[0])
5100000000000000000
>
```

JSON-RPC 部署调用合约

合约编写编译

编写一个简单的智能合约：

```
contract SimpleTest {
    uint data;
    function set(uint n) public {
        data = n*3;
    }
    function get() public returns (uint) {
        return data;
    }
}
```

编译合约：

```
curl --data '{"jsonrpc":"2.0","method": "eth_compileSolidity", "params":
["contract SimpleTest {uint data;function set(uint n) public {data =
n*3;}function get() public returns (uint) {return data;}}"], "id": 5}'
localhost:8545
```

同时，Geth 启动以太坊进行挖矿，但是报如下错误：

```
qhh0809@ubuntu:~$ curl --data '{"jsonrpc":"2.0","method": "eth_compileSolidity",
 "params": ["contract SimpleTest {uint data;function set(uint n) public {data =
n*3;}function get() public view returns (uint) {return data;}}"], "id": 5}' loca
lhost:8545
curl: (7) Failed to connect to localhost port 8545: Connection refused
```

查询原因，因为端口 8545 未打开，按以下命令重新启动 geth：

```
geth --datadir "~/.ethereum" --networkid 16 --rpc console
```

重新启动后在此编译，又出现了如下错误：

```
qhh0809@ubuntu:~$ curl --data '{"jsonrpc": "2.0","method": "eth_compileSolidity"
,"params": ["contract SimpleTest {uint data;function set(uint n) public {data =
n*3;}function get() public returns (uint) {return data;}}"],"id": 5}' localhost:
8545
invalid content type, only application/json is supported
```

在编译命令中增加请求头，改为：

```
curl -X POST -H "Content-Type":application/json --data
'{"jsonrpc":"2.0","method": "eth_compileSolidity", "params": ["contract
SimpleTest {uint data;function set(uint n) public {data = n*3;}function
get() public returns (uint) {return data;}}"], "id": 5}' localhost:8545
```

再次编译，结果又报如下错误：

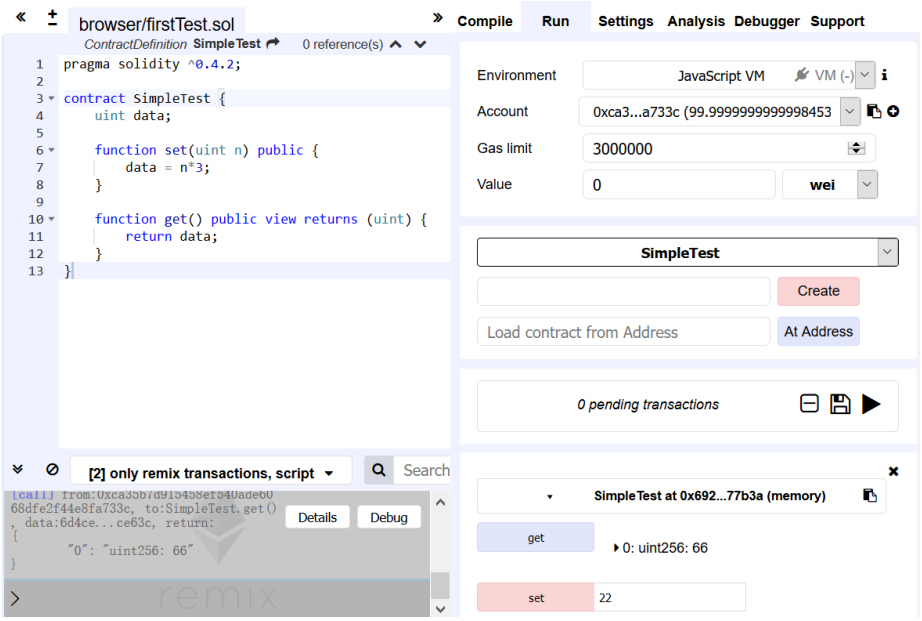
```
qhh0809@ubuntu:~$ curl -X POST -H "Content-Type":application/json --data '{"json
rpc":"2.0","method": "eth_compileSolidity", "params": ["contract SimpleTest {uin
t data;function set(uint n) public {data = n*3;}function get() public returns (u
int) {return data;}}"], "id": 5}' localhost:8545
{"jsonrpc":"2.0","id":5,"error":{"code":-32601,"message":"The method eth_compile
Solidity does not exist/is not available"}}
```

经查，这是因为在 geth 1.6 版之后就废弃了“eth_compileSolidity”这个函数，所以我采取了在线编译的方法进行编译。

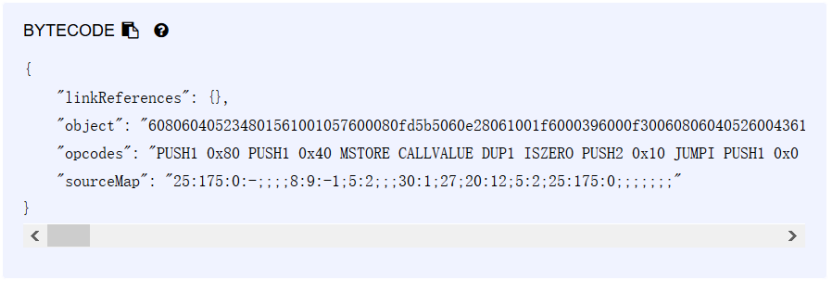
在线编译智能合约

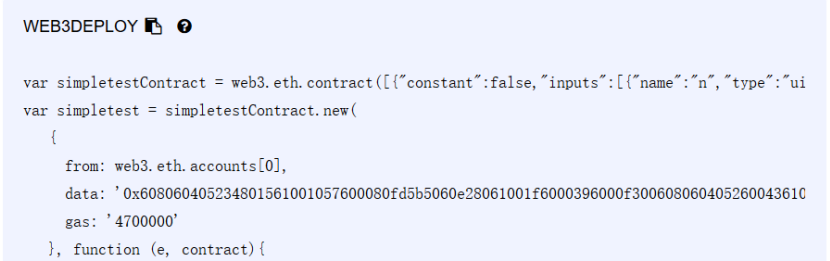
这里使用 https://ethereum.github.io/browser-solidity/进行在线编写智能合约的代码。

编译运行成功，并进行测试，截图如下：



并且获得编译后的字节码和 abi 文件，如下图：





其中字节码为：

```
0x6080604052348015610010576000080fd5b5060e28061001f6000396000f3006080604052
600436106049576000357c0100000000000000000000000000000000000000000000000000000
000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5b
34801560595760008fd5b506076600480360381019080803590602001909291905050506
0a0565b005b34801560835760008fd5b50608a60ad565b60405180828152602001915050606
040518091039f35b6003810260008190555050565b60008054905090560a165627a7a7230
582010be0b1da73b1ecf4cbced8c6ff8eb9b29d01034260ab552932800bd69a575fa0029
```

abi 文件内容为：

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

web3 deploy 内容为：

```
var simpletestContract =
web3.eth.contract([{"constant":false,"inputs":[{"name":"n","type":"uint256
"}],"name":"set","outputs":[],"payable":false,"stateMutability":"nonpayabl
e","type":"function"},{"constant":true,"inputs":[],"name":"get","outputs":
[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","t
ype":"function"}]); var simpletest = simpletestContract.new( { from:
web3.eth.accounts[0], data:
'0x6080604052348015610010576000080fd5b5060e28061001f6000396000f300608060405
260043610604957600357c0100000000000000000000000000000000000000000000000000000
0000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5
b34801560595760008fd5b506076600480360381019080803590602001909291905050506
0a0565b005b34801560835760008fd5b50608a60ad565b60405180828152602001915050606
040518091039f35b6003810260008190555050565b60008054905090560a165627a7a723
0582010be0b1da73b1ecf4cbced8c6ff8eb9b29d01034260ab552932800bd69a575fa0029'
, gas: '4700000' }, function (e, contract){ console.log(e, contract); if
(typeof contract.address !== 'undefined') { console.log('Contract mined!
address: ' + contract.address + ' transactionHash: ' +
contract.transactionHash); } })
```

智能合约部署

首先，要获得调用账户：

```
curl -H "Content-Type":application/json --data
'{"jsonrpc":"2.0","method":"eth_coinbase", "id":1}' localhost:8545
```

```
qhh0809@ubuntu:~$ curl -H "Content-Type":application/json --data '{"jsonrpc":"2.
0","method":"eth_coinbase", "id":1}' localhost:8545
{"jsonrpc":"2.0","id":1,"result":"0x6815515f32312f65718bc0b1ee8d900c27167d85"}
qhh0809@ubuntu:~$
```

所以，0x6815515f32312f65718bc0b1ee8d900c27167d85 就是要进行交易的账户。

然后，在 B 终端中执行以下命令，params 参数中的 from 就是发起该笔交易的账户，data 参数中就是该合约的字节码。

```
curl -H "Content-Type":application/json --data '{"jsonrpc":"2.0","method":
"eth_sendTransaction", "params":
[{"from":"0x6815515f32312f65718bc0b1ee8d900c27167d85", "gas": "0xbb8a9",
"data":
"0x6080604052348015610010576000800fd5b5060e28061001f6000396000f300608060405
260043610604957600035 7c0100000000000000000000000000000000000000000000000000000000
0000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5
b34801560595760008 0fd5b50607660048036038101908080359060200190929190505050506
0a0565b005b34801560835760008 0fd5b50608a60ad565b6040518082815260200191505060405180910390f35b60038102600
0405180910390f35b60038102600081905550505 65b60008054905090560 0a165627a7a723
0582010be0b1da73b1ecf4cbced8c6ff8eb9b29d01034260ab552932800bd69a575fa0029"
}],"id": 6}' localhost:8545
```

在 B 终端执行上述命令后，在 A 终端中执行”miner.start()”启动挖矿。

在 B 终端报如下错误：

```
qhh0809@ubuntu:~$ curl -H "Content-Type":application/json --data '{"jsonrpc":"2.
0","method": "eth_sendTransaction", "params": [{"from":
> "0x85ad0d9b87156f50ca11a4d69c24c184666bf31a", "gas": "0xb8a9", "data":
> "0x6080604052348015610010057600080fd5b5060e28061001f6000396000f3006080604052600
4361060495760003 57c0100000000000000000000000000000000000000000000000000000000
0000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5b348015605957600
080fd5b506076600480360381019080803590602001909291905050506 0a0565b005b34801560835
7600080fd5b50608a60ad565b6040518082815260200191505060405180910390f35b60038102600
08190555050565b6000805490509056 00a165627a7a723058 2010be0b1da73b1ecf4cbced8c6ff8e
b9b29d01034260ab552932800bd69a575fa0029"}],
> "id": 6}' localhost:8545
{"jsonrpc":"2.0","id":6,"error":{"code":-32000,"message":"authentication needed:
 password or unlock"}}
qhh0809@ubuntu:~$
```

所以在 A 终端先将该账户进行解锁：

```
personal.unlockAccount(eth.accounts[0],"qhh666888")
```

```
> miner.stop()
null
> eth.mining
false
> personal.unlockAccount(eth.accounts[0])
Unlock account 0x85ad0d9b87156f50ca11a4d69c24c184666bf31a
Passphrase:
true
```

解锁成功后再次部署，得到下图：

```
qhh0809@ubuntu:~$ curl -H "Content-Type":application/json --data '{"jsonrpc":"2.
0","method": "eth_sendTransaction", "params": [{"from":
> "0x6815515f32312f65718bc0b1ee8d900c27167d85", "gas": "0xbb8a9", "data":
> "0x6080604052348015610010057600080fd5b5060e28061001f6000396000f3006080604052600
4361060495760003 57c0100000000000000000000000000000000000000000000000000000000
0000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5b348015605957600
080fd5b506076600480360381019080803590602001909291905050506 0a0565b005b34801560835
7600080fd5b50608a60ad565b6040518082815260200191505060405180910390f35b60038102600
08190555050565b6000805490509056 00a165627a7a723058 2010be0b1da73b1ecf4cbced8c6ff8e
b9b29d01034260ab552932800bd69a575fa0029"}],
> "id": 6}' localhost:8545
{"jsonrpc":"2.0","id":6,"result":"0x35b30794b23c55754f271cfe97a2692f3001e9a36c2a
5734a5b80ffdf1790068"}
```

由上图可知，本次交易（部署合约）的交易 Hash 为：
0x35b30794b23c55754f271cfe97a2692f3001e9a36c2a5734a5b80ffdf1790068

在 A 终端显示成功提交合约：

```
INFO [10-15|18:58:28.076] Submitted contract creation              fullhash=0x35
b30794b23c55754f271cfe97a2692f3001e9a36c2a5734a5b80ffdf1790068 contract=0x671b14
461CC576e682F32738f5D51448f56Ce822
```

结果为：
fullhash=0x35b30794b23c55754f271cfe97a2692f3001e9a36c2a5734a5b80ffdf1790068
contract=0x671b14461CC576e682F32738f5D51448f56Ce822

至此，合约部署完成。

调用合约

首先，在 A 终端中计算 payload 中的方法选择符对应的字节，进行十六进制编码：

```
web3.sha3("set(uint256)").substring(0, 10)
```

执行上述命令会得到："0x60fe47b1"，即下图：

```
> web3.sha3("set(uint256)").substring(0, 10)
"0x60fe47b1"
>
```

然后将其作为参数进行方法的调用：

```
curl -H "Content-Type":application/json --data '{"jsonrpc":"2.0","method":
"eth_sendTransaction", "params":
[{"from":"0x6815515f32312f65718bc0b1ee8d900c27167d85",
"to":"0x671b14461CC576e682F32738f5D51448f56Ce822",
"data":"0x60fe47b1000000000000000000000000000000000000000000000000000000
00000064"}], "id": 8}' localhost:8545
```

```
qhh0809@ubuntu:~$ curl -H "Content-Type":application/json --data '{"jsonrpc":"2.
0","method": "eth_sendTransaction", "params": [{"from":"0x6815515f32312f65718bc0
b1ee8d900c27167d85", "to":"0x671b14461CC576e682F32738f5D51448f56Ce822", "data":"
0x60fe47b1000000000000000000000000000000000000000000000000000000000000064"}], "
id": 8}' localhost:8545
{"jsonrpc":"2.0","id":8,"result":"0x3cec2231b930aa2d9001e5021f1ebba4479ada0afafc
b5a9d34c5f62c6969a26"}
qhh0809@ubuntu:~$
```

执行完 set 方法后，用同样的方法调用 get 方法：

```
curl -H "Content-Type":application/json --data '{"jsonrpc":"2.0","method":
"eth_sendTransaction", "params":
[{"from":"0x6815515f32312f65718bc0b1ee8d900c27167d85",
"to":"0x671b14461CC576e682F32738f5D51448f56Ce822", "data":"0x6d4ce63c"}],
"id": 8}' localhost:8545
```

```
qhh0809@ubuntu:~$ curl -H "Content-Type":application/json --data '{"jsonrpc":"2.
0","method": "eth_sendTransaction", "params": [{"from":"0x6815515f32312f65718bc0
b1ee8d900c27167d85", "to":"0x671b14461CC576e682F32738f5D51448f56Ce822", "data":"
0x6d4ce63c"}], "id": 8}' localhost:8545
{"jsonrpc":"2.0","id":8,"result":"0x613fa507d89634b4b0fb9f1e037ba3576190897b929d
3c3c46c58c048a1aec93"}
qhh0809@ubuntu:~$
```

最后得到 get 的交易 hash 为：
0x613fa507d89634b4b0fb9f1e037ba3576190897b929d3c3c46c58c048a1aec93

利用 get 的交易 hash 值查询交易结果：

```
curl -H "Content-Type":application/json --
data '{"jsonrpc":"2.0","method":"eth_getTransactionReceipt","params":["0x6
13fa507d89634b4b0fb9f1e037ba3576190897b929d3c3c46c58c048a1aec93"],"id":1}'
localhost:8545
```

```
qhh0809@ubuntu:~$ curl -H "Content-Type":application/json --data '{"jsonrpc":"2.
0","method":"eth_getTransactionReceipt","params":["0x613fa507d89634b4b0fb9f1e037
ba3576190897b929d3c3c46c58c048a1aec93"],"id":1}' localhost:8545
{"jsonrpc":"2.0","id":1,"result":{"blockHash":"0xd72d3e9c3306eded072fc714d997ca9
93b224ed96feef1e01668404144471fcc","blockNumber":"0x90","contractAddress":null,"
cumulativeGasUsed":"0x54c0","from":"0x6815515f32312f65718bc0b1ee8d900c27167d85",
"gasUsed":"0x54c0","logs":[],"logsBloom":"0x00000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000","r
oot":"0xfc12a5a53ed4382f2442309beabfd8fb07cad2fb622c9234891817cd094020e3","to":"
0x671b14461cc576e682f32738f5d51448f56ce822","transactionHash":"0x613fa507d89634b
4b0fb9f1e037ba3576190897b929d3c3c46c58c048a1aec93","transactionIndex":"0x0"}}
qhh0809@ubuntu:~$
```

但是此处看不到结果，log 为空，真正的结果在 decoded output 之中，答案为 300.

JavaScript API 部署调用智能合约

编译合约

先定义 source 存放智能合约代码：

```
var source = 'contract SimpleTest {uint data;function set(uint n) public
{data = n*3;}function get() public returns (uint) {return data;}}'
```

```
> var source = 'contract SimpleTest {uint data;function set(uint n) public {data
 = n*3;}function get() public returns (uint) {return data;}}'
undefined
> source
"contract SimpleTest {uint data;function set(uint n) public {data = n*3;}functio
n get() public returns (uint) {return data;}}"
>
```

调用 web3 中的方法编译上面的 source 变量：

```
var compiled = web3.eth.compile.solidity(source);
```

运行代码后报如下错误：

```
> var compiled = web3.eth.compile.solidity(source);
Error: The method eth_compileSolidity does not exist/is not available
    at web3.js:3143:20
    at web3.js:6347:15
    at web3.js:5081:36
    at <anonymous>:1:16
```

同理，Geth1.8 没有 eth_compileSolidity 方法。所以依然采用在线编译。并将字节码和 abi 分别存放至 code 和 abi 变量之中，便于调用。

```
var code =
'0x6080604052348015610010576000080fd5b5060e28061001f6000396000f300608060405
```

```
260043610604957600035 7c0100000000000000000000000000000000000000000000000000000000
0000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5
b34801560595760008 0fd5b506076600480360381019080803590602001909291905050506
0a0565b005b34801560835760 0080fd5b50608a60ad565b6040518082815260200191505060405180910390f35b6003810260008
190555050565b6000805490509056 00a165627a7a723 0582010be0b1da73b1ecf4cbced8c6ff8eb9b29d01034260ab552932800bd69a575fa0029'
var abi = '[ { "constant": true, "inputs": [], "name": "get", "outputs":
[ { "name": "", "type": "uint256" } ], "payable": false,
"stateMutability": "view", "type": "function" }, { "constant": false,
"inputs": [ { "name": "n", "type": "uint256" } ], "name": "set",
"outputs": [], "payable": false, "stateMutability": "nonpayable", "type":
"function" } ]'
```

```
> code
"0x6080604052348015610010576000 80fd5b5060e28061001f6000396000f300608060405260043
6106049576000357c0100000000000000000000000000000000000000000000000000000000900463
3fffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5b34801560595760008
0fd5b506076600480360381019080803590602001909291905050506a0565b005b3480156083576
00080fd5b50608a60ad565b6040518082815260200191505060405180910390f35b6003810260008
190555050565b6000805490509056 00a165627a7a7230582010be0b1da73b1ecf4cbced8c6ff8eb9
b29d01034260ab552932800bd69a575fa0029"
> abi
"[ { \"constant\": true, \"inputs\": [], \"name\": \"get\", \"outputs\": [ { \"n
ame\": \"\", \"type\": \"uint256\" } ], \"payable\": false, \"stateMutability\":
 \"view\", \"type\": \"function\" }, { \"constant\": false, \"inputs\": [ { \"na
me\": \"n\", \"type\": \"uint256\" } ], \"name\": \"set\", \"outputs\": [], \"pa
yable\": false, \"stateMutability\": \"nonpayable\", \"type\": \"function\" } ]"
```

部署合约

使用上述生成的 code 和 abi 把合约部署到以太坊上。需要挖矿来确认该笔交易。

```
web3.eth.contract(abi).new({from:"0x6815515f32312f65718bc0b1ee8d900c27167d
85",data:code})
```

```
> eth.contract(abi).new({from:"0x6815515f32312f65718bc0b1ee8d900c27167d85",data:
code})

TypeError: 'filter' is not a function
    at web3.js:2830:12
    at web3.js:3001:21
    at <anonymous>:1:1
```

发现这个方法也被废除，故而我们采取 web3deploy 进行部署，将 web3deploy 放入命令行进行定义：

```
var simpletestContract =
web3.eth.contract([{"constant":false,"inputs":[{"name":"n","type":"uint256
"}],"name":"set","outputs":[],"payable":false,"type":"function","stateMuta
bility":"nonpayable"},{"constant":false,"inputs":[],"name":"get","outputs"
:[{"name":"","type":"uint256"}],"payable":false,"type":"function","stateMu
tability":"nonpayable"}]); var simpletest =
simpletestContract.new( { from: web3.eth.accounts[0], data:
'0x606060405234156100 0c57fe5b5b60c98061001b6000396000f3006060604052600035 7
c0100000000000000000000000000000000000000000000000000000000900463ffffffff1
6806360fe47b11460445780636d4ce63c146061575bfe5b3415604b57fe5b605f600480803
```

```
590602001909190505060845 65b005b3415606857fe5b606e6092565b60405180828152602
001915050604051809103 90f35b600381026000819055505b50565b600060005490505b905
600a165627a7a723058208181481e86008df4345b2647d9ae52b4f3506a91e10859601bcf0
4d2a0ef30660029', gas: '4700000' }, function (e, contract){ console.log(e,
contract); if (typeof contract.address !== 'undefined')
{ console.log('Contract mined! address: ' + contract.address + '
transactionHash: ' + contract.transactionHash); } })
```

```
> var simpletestContract = web3.eth.contract([{"constant":false,"inputs":[{"name
":"n","type":"uint256"}],"name":"set","outputs":[],"payable":false,"type":"funct
ion","stateMutability":"nonpayable"},{"constant":false,"inputs":[],"name":"get",
"outputs":[{"name":"","type":"uint256"}],"payable":false,"type":"function","stat
eMutability":"nonpayable"}]);
undefined
```

```
> var simpletest = simpletestContract.new(
   {
     from: web3.eth.accounts[0],
     data: '0x6060604052341561000c57fe5b5b60c98061001b6000396000f300606060405260
00357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16
806360fe47b11460445780636d4ce63c146061575bfe5b3415604b57fe5b605f6004808035906020
019091905050608456 5b005b3415606857fe5b606e6092565b6040518082815260200191505060040
5180910390f35b6003810260008190555505b50565b600060005490505b905600a165627a7a723058
20263a84bdc57c5b6e140ff4310b36d76ceb005418d34e7c0c454dff69116dd3880029',
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + ' transact
ionHash: ' + contract.transactionHash);
    }
 })
INFO [10-16|10:03:33.272] Submitted contract creation              fullhash=0x7c
b81d51b8b390804ef7ddcb4b99a2847591699bcb466a7259ee882b86525f9e contract=0x914989
4f935571995AEe5fdf0Dbe4cc9d6743624
null [object Object]
undefined
```

之后启动 miner.start()挖矿来确认交易的进行，得到下图，证明合约部署成功：

```
null [object Object]
Contract mined! address: 0x9149894f935571995aee5fdf0dbe4cc9d6743624 transactionH
ash: 0x7cb81d51b8b390804ef7ddcb4b99a2847591699bcb466a7259ee882b86525f9e
```

查看合约名，得到下图：

```
> simpletest
{
  abi: [{
      constant: false,
      inputs: [{...}],
      name: "set",
      outputs: [],
      payable: false,
      stateMutability: "nonpayable",
      type: "function"
  }, {
      constant: false,
      inputs: [],
      name: "get",
      outputs: [{...}],
      payable: false,
      stateMutability: "nonpayable",
      type: "function"
  }],
  address: "0x9149894f935571995aee5fdf0dbe4cc9d6743624",
  transactionHash: "0x7cb81d51b8b390804ef7ddcb4b99a2847591699bcb466a7259ee882b86
525f9e",
  allEvents: function(),
  get: function(),
  set: function()
}
```

所以可以确认合约部署成功，合约地址为：
0x9149894f935571995aee5fdf0dbe4cc9d6743624

合约调用

合约部署成功之后，调用 set 方法，因为算一个交易，所以采用 sendTransaction 进行赋值，并指定支付 gas 的用户，然后可以调用 txpool 中有一个待写入的交易，然后进行挖矿，将交易写入区块。

```
simpletest.set.sendTransaction(22,{from:eth.accounts[0]})
```

```
> simpletest.set.sendTransaction(22,{from:eth.accounts[0]})
INFO [10-16|19:57:46.016] Submitted transaction                    fullhash=0x4e
5745dddb808d88ad272fad54214c9cb072d85e28362db9148d0f9d92f436aa recipient=0x653Fc
dd5CcC3565384536230b082F4C29f5DE03E
"0x4e5745dddb808d88ad272fad54214c9cb072d85e28362db9148d0f9d92f436aa"
> txpool.status
{
  pending: 1,
  queued: 0
}
```

调用完 set 方法后，调用 get 方法的 call 查看结果是否正确，set 值为 22，乘以 3 后结果为 66，可见结果调用正确。

```
simpletest.get.call()
```

```
> simpletest.get.call()
66
>
```