



区块链课程设计报告

第一阶段准备阶段

学 院 名 称 : 数据科学与计算机学院

专业（班级） : 16 软件工程电子政务方向

学 生 姓 名 : 唐育涛

学 号 : 16340209

时 间 : 2018 年 10 月 30 日



课程任务：

2. 前期热身报告

- 以太坊的安装、私有链创世区块搭建、私有链节点的加入(选做)
- 对 `getBlock` 中所得区块的各个字段进行解释
- 对日志输出进行解释
- 编写简单的智能合约，在 `remix` 下进行调试，并部署在链上进行调用
- 对交易的字段进行解释

写在前面的话：

因为这次的内容是偏向于全新的领域，所以很多知识都是不断的查阅相关知识，整合相关资料，不断探索法索得到的，饮水思源，尊重知识，尊重劳动，这次发掘的学习资料和博客链接如下：

搭建私有链和开创创世块的优秀博客：

<https://blog.csdn.net/sunshine1314/article/details/79692502>

区块链 `web.js` 优秀学习手册：

<http://cw.hubwiz.com/card/c/web3.js-1.0/1/2/18/>

Remix 的使用和简单智能合约的书写：

<https://www.jianshu.com/p/270617283038>

Getblock 相关知识的优秀博客：

<https://blog.csdn.net/shebao3333/article/details/80099216>

查看区块链交易数据博客：

<https://blog.csdn.net/shebao3333/article/details/80078160>



正式开始我们的报告书写：

1. 安装以太坊，创世块建立和私有链结点加入

我选择的是安装的运行环境是 windows，首先安装 geth，可以再官网下载也可以下载国内镜像版本，下载安装包运行 exe 文件安装之后会在 Geth 目录下生成 geth.exe 文件，将 geth.exe 的路径保存到本地的 path 变量中即可（一般会自动加入，如果加入失败就手动加入），之后再 cmd 窗口下输入 geth version 查看安装是否成功

```
C:\>geth version
Geth
Version: 1.8.3-stable
Git Commit: 329ac18ef617d0238f71637bffe78f028b0f13f7
Architecture: amd64
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.10
Operating System: windows
GOPATH=
GOROOT=C:\go
```

如上，安装成功。

接下来建立私有链和写入创世块，

(1) 首先我们进入 geth 的控制台，

输入：`geth --datadir chain1 --nodiscover console`

```
C:\Program Files\Geth>geth --datadir chain1 --nodiscover console
INFO [11-04|02:22:58] Maximum peer count          ETH=25 LES=0 total=25
INFO [11-04|02:22:58] Starting peer-to-peer node  instance=Geth/v1.8.3-stable-329ac18e/windows-amd64/go1.10
INFO [11-04|02:22:58] Allocated cache and file handles database="C:\\Program Files\\Geth\\chain1\\geth\\chaindata"
a" cache=768 handles=1024
INFO [11-04|02:22:58] Initialised chain configuration config="{ChainID: 10 Homestead: 0 DAO: <nil> DAOSupport:
false EIP150: <nil> EIP155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil> Engine: unknown}"
INFO [11-04|02:22:58] Disk storage enabled for ethash caches dir="C:\\Program Files\\Geth\\chain1\\geth\\ethash" count
=3
```

成功进入控制台

(2) 建立两个账户并且查看建立的账户

输入如下：`personal.newAccount("1234")` 输入两次，得到两个 account



建立后使用 `eth.accounts`, 出现如下画面说明建立成功

```
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
> eth.accounts
["0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0", "0x4fca5d4431ad59476cdfb7eab9b4ee4067af0abe"]
> -
```

(3) 有了账户之后我们开始建立创世块, 首先建立一个 json 的文件, 如下:

```
1 {
2   "alloc": {
3     "0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0": { //我建立的第一个账户的地址
4       "balance": "9990000000000000000"
5     }
6   },
7   "config":{
8     "chainId":10,
9     "homesteadBlock":0,
10    "eip155Block":0,
11    "eip158Block":0
12  },
13  "nonce":"0x0000000000000042",
14  "mixhash":"0x0000000000000000000000000000000000000000000000000000000000000000",
15  "difficulty": "0x2000",
16  "coinbase":"0x0000000000000000000000000000000000000000000000000000000000000000",
17  "timestamp": "0x00",
18  "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
19  "extraData": "",
20  "gasLimit":"0xffffffff"
21 }
22 }
```

并且命名为 `genesis.json`

之后使用以下命令来建立创世块

输入: `geth --datadir chain1 init genesis.json`

```
C:\Program Files\Geth>geth --datadir chain1 init genesis.json
INFO [11-04|02:24:34] Maximum peer count          ETH=25 LES=0 total=25
INFO [11-04|02:24:34] Allocated cache and file handles database="C:\Program Files\Geth\chain1\geth\chaindata"
a" cache=16 handles=16
INFO [11-04|02:24:34] Writing custom genesis block nodes=1 size=201.00B time=0s gcnodes=0 gcsiz=0.00B gctim
INFO [11-04|02:24:34] Persisted trie from memory database e=0s livenodes=1 liveness=0.00B
INFO [11-04|02:24:34] Successfully wrote genesis state database=chaindata
hash=ab9b60...e7a532
INFO [11-04|02:24:34] Allocated cache and file handles database="C:\Program Files\Geth\chain1\geth\lightcha
```




(4) 再次进入控制台，查询两个账户余额，并且建立区块链上的第一笔交易

进入控制台：`C:\Program Files\Geth>geth --datadir chain1 --nodiscover console`

查看余额：

```
> eth.getBalance(eth.accounts[0])
9990000000000000000000
> eth.getBalance(eth.accounts[1])
0
> web3.fromWei(eth.getBalance(eth.accounts[0]))
999
> web3.fromWei(eth.getBalance(eth.accounts[1]))
0
```

建立第一笔交易 从账户 0 划去 200 余额给账户 1

```
> a0=eth.accounts[0]
'0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0'
> a1=eth.accounts[1]
'0x4fca5d4431ad59476cdfb7eab9b4ee4067af0abe'
> personal.unlockAccount(a0,"1234")
true
> eth.sendTransaction({from:a0, to:a1, value:web3.toWei(200,"ether")})
INFO [11-04|15:07:34] Submitted transaction fullhash=0xc5e9a790b121bb9000558521d92cac6d8bf2198f294f52cabdc5d8355b675a recipient=0x4fca5d4431ad59476CDFB7EAB9b4eE4067Af0ABe
'0xc5e9a790b121bb9000558521d92cac6d8bf2198f294fba52cabdc5d8355b675a'
```

查看此时的账户

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
999
> web3.fromWei(eth.getBalance(eth.accounts[1]))
0
```

账户余额并没有发生变化，那是因为并没有区块结点去记录当前的交易记录

启动矿工挖矿

```
> miner.start()

> INFO [11-04|02:29:28] Updated mining threads threads=0
INFO [11-04|02:29:28] Transaction pool price threshold updated price=18000000000
INFO [11-04|02:29:28] Starting mining operation
INFO [11-04|02:29:28] Commit new mining work number=1 txs=1 uncles=0 elapsed=0s

> INFO [11-04|02:29:58] Successfully sealed new block number=1 hash=ec2ac5...03bd37
INFO [11-04|02:29:58] □□ mined potential block number=1 hash=ec2ac5...03bd37
INFO [11-04|02:29:58] Commit new mining work number=2 txs=0 uncles=0 elapsed=997.4µs
INFO [11-04|02:30:00] Successfully sealed new block number=2 hash=b91934...2b222f
INFO [11-04|02:30:00] □□ mined potential block number=2 hash=b91934...2b222f
INFO [11-04|02:30:00] Commit new mining work number=3 txs=0 uncles=0 elapsed=997.3µs
INFO [11-04|02:30:01] Successfully sealed new block number=3 hash=c85c30...73bbea
INFO [11-04|02:30:01] □□ mined potential block number=3 hash=c85c30...73bbea
INFO [11-04|02:30:01] Commit new mining work number=4 txs=0 uncles=0 elapsed=0s
INFO [11-04|02:30:01] Successfully sealed new block number=4 hash=46fad2...915c17
INFO [11-04|02:30:01] □□ mined potential block number=4 hash=46fad2...915c17
INFO [11-04|02:30:01] Commit new mining work number=5 txs=0 uncles=0 elapsed=0s
```



会出现上面的一个 DAG 过程，主要是因为我们的矿石块初始化缓冲需要时间，等待缓冲 100%之后就会开始挖矿

```
> INFO [11-04|02:29:28] Updated mining threads threads=0
INFO [11-04|02:29:28] Transaction pool price threshold updated price=18000000000
INFO [11-04|02:29:28] Starting mining operation
INFO [11-04|02:29:28] Commit new mining work number=1 txs=1 uncles=0 elapsed=0s

> INFO [11-04|02:29:58] Successfully sealed new block
INFO [11-04|02:29:58] □□ mined potential block number=1 hash=ec2ac5...03bd37
INFO [11-04|02:29:58] Commit new mining work number=1 hash=ec2ac5...03bd37
INFO [11-04|02:30:00] Successfully sealed new block number=2 txs=0 uncles=0 elapsed=997.4µs
INFO [11-04|02:30:00] □□ mined potential block number=2 hash=b91934...2b222f
INFO [11-04|02:30:00] Commit new mining work number=2 hash=b91934...2b222f
INFO [11-04|02:30:01] Successfully sealed new block number=3 txs=0 uncles=0 elapsed=997.3µs
INFO [11-04|02:30:01] □□ mined potential block number=3 hash=c85c30...73bbea
INFO [11-04|02:30:01] Commit new mining work number=3 hash=c85c30...73bbea
INFO [11-04|02:30:01] Successfully sealed new block number=4 txs=0 uncles=0 elapsed=0s
INFO [11-04|02:30:01] □□ mined potential block number=4 hash=46fad2...915c17
INFO [11-04|02:30:01] Commit new mining work number=4 hash=46fad2...915c17
INFO [11-04|02:30:01] Commit new mining work number=5 txs=0 uncles=0 elapsed=0s
```

停止挖矿，挖矿之后查看我们得账户余额

```
> miner.stop()
true
> web3.fromWei(eth.getBalance(eth.accounts[0]))
869
> web3.fromWei(eth.getBalance(eth.accounts[1]))
200
```

账户 1 余额增加了 200 账户 0 减少了 200，但是因为挖矿获得了部分钱

自此我们完成了我们的私有链创建，并且在上面完成了我们的第一笔交易。接下来我们实现私有链结点加入，（这一部分需要另一台机子，如果用虚拟机做的话只要两个虚拟机即可，但是如果像我一样本地实现的，可以借助借助另一台电脑）

主要过程如下：

1. cmd 输入命令获取两台机器的 ip 并且成功 ping 通

以太网适配器 VMware Network Adapter VMnet1:

```
连接特定的 DNS 后缀 . . . . . :
本地链接 IPv6 地址. . . . . : fe80::7178:a8a0:bffd:8c4d%16
自动配置 IPv4 地址 . . . . . : 169.254.140.77
子网掩码 . . . . . : 255.255.0.0
默认网关. . . . . :
```



2. 能够 ping 通的前提下，将第一台机子中私有链搭建中所新建 genesis.json、keystore 文件夹复制到第二台机子
3. 先在第一台机子中启动节点：`geth --datadir "./" --networkid 989898 -rpc console --port 30304 --rpcport 8546`
4. 在第二台机子中
创建节点：`geth --datadir "./" init genesis.json`
启动节点：`geth --datadir "./" --networkid 989898 -rpc console --port 30304 --rpcport 8546`

```
INFO [11-04|02:04:31] Successfully wrote genesis state      database=lightchaindata
ash=ab9b60...e7a532
C:\Program Files\Geth>geth --datadir "./" --networkid 989898 -rpc console --port 30304 --rpcport 8546
INFO [11-04|02:04:58] Maximum peer count                    ETH=25 LES=0 total=25
INFO [11-04|02:04:58] Starting peer-to-peer node           instance=Geth/v1.8.3-stable-329ac18e/windows-amd64/go1.1
INFO [11-04|02:04:58] Allocated cache and file handles     database="C:\\Program Files\\Geth\\geth\\chaindata" cache=
768 handles=1024
WARN [11-04|02:04:58] Upgrading database to use lookup entries
```

5. 查看节点信息并发给第一台机子动态添加

输入：`admin.nodeInfo.enode`

获得第二台机子的节点信息，将[::]换成第二台机子的 ip 发给第一台机子，第一台机子动态添加

```
> admin.addPeer("enode://38d6eac76c290b8fdb4d9362a6b32711907c4577a1d9ced92dddc84d38313cdc18caa8965ad52fb58ec23aa38613b
f7a4a0f7c22f6ac8a08e0283dfe75625b@[192.168.43.59]:30304")
true
> net.peerCount
```

6. 再次查看节点数目，可以看到现在的兄弟结点为 1

输入：`net.peerCount`

```
> net.peerCount
1
```

私有链结点加入完成。



2. 解释 getblock 的相关字段

使用命令调取某一块的信息，查看返回值我们才能来看这些字段信息

输入 `web3.eth.getBlock (8)`

[illegible]

接下来我们要对其中的数据进行解释

difficulty - BigNumber 类型，记录私有链的挖矿的难度。

extraData - 字符串。当前块的 extra data 字段。

gasLimit - Number, 当前区块允许使用的最大 gas。

gasUsed - 当前区块累计使用的总的 gas。

hash - 字符串，区块的哈希串。当这个区块处于 pending 将会返回 null。

logsBloom- 字符串，区块日志的过滤器。当这个区块处于 pending 将会返回 null。

miner - 字符串, 20 字节。这个区块获得奖励的矿工。（我们这条私聊默认的挖矿工是账户 0, 所以可以看到是我们账户 0 的标志）

nonce - 字符串, 8 字节。POW 生成的哈希。当这个区块处于 pending 将会返回 null。



Number - 区块号。当这个区块处于 pending 将会返回 null。

parentHash - 字符串，32 字节的父区块的哈希值。

sha3Uncles - 字符串，32 字节。叔区块的哈希值。

size - Number。当前这个块的字节大小。

stateRoot - 字符串，32 字节。区块的最终状态前缀树的根。

timestamp - Number。区块打包时的 unix 时间戳。

totalDifficulty - BigNumber 类型。区块链到当前块的总难度，整数。

transactions - 数组。交易对象。或者是 32 字节的交易哈希。

transactionsRoot - 字符串，32 字节，区块的交易前缀树的根。

uncles - 数组。叔哈希的数组。

3. 对日志输出进行解释

这个问题是我整个热身报告做下来最迷的一部分，实际上当我部署好所有的内容并且完成了实验甚至写好了其他部分的实验报告的时候我仍然不知道这一部分该怎么下手，按照我个人的理解，区块链本身的操作实际上是会在我们本地部署的私有链数据存储文件夹里有一个 Log 日志文件的，而这样的一个文件也记录了我们在 web 控制所做的关于我们部署的链的相关操作。但是我顺着这样一个思路去解答的时候发现网上是几乎没有这一块的内容的，更别说清楚的教程。所以导致我很迷。

那么强上，我们首先在本地找到我们的 LOG 日志，他们保存在我们部署的私有链的粗出文件夹的 `« Program Files > Geth > chain1 > geth > chaindata` 中



000014.ldb	2018/11/4 10:09	Microsoft Acces...	11 KB
000015.ldb	2018/11/4 10:25	Microsoft Acces...	2 KB
000018.ldb	2018/11/4 13:13	Microsoft Acces...	7 KB
000019.log	2018/11/4 13:14	文本文档	1 KB
CURRENT	2018/11/4 13:13	文件	1 KB
LOCK	2018/11/4 2:24	文件	0 KB
LOG	2018/11/4 16:06	文件	5 KB

打开日志文件

```
===== Nov 4, 2018 (CST) =====
02:24:34.602951 log@legend F·NumFile S·FileSize N·Entry C·BadEntry B·BadBlock Ke·KeyError D·DroppedEntry L·Level Q·SeqNum T·TimeElapsed
02:24:34.614940 db@open opening
02:24:34.615936 version@stat F·[] S·0B[] Sc·[]
02:24:34.622935 db@janitor F·2 G·0
02:24:34.622935 db@open done T·7.9955ms
===== Nov 4, 2018 (CST) =====
02:25:16.434196 log@legend F·NumFile S·FileSize N·Entry C·BadEntry B·BadBlock Ke·KeyError D·DroppedEntry L·Level Q·SeqNum T·TimeElapsed
02:25:16.434196 version@stat F·[] S·0B[] Sc·[]
02:25:16.434196 db@open opening
02:25:16.435193 journal@recovery F·1
02:25:16.435193 journal@recovery recovering @1
02:25:16.444172 memdb@flush created L0@2 N·11 S·727B "H\xab\x9b...\xe7\xa52,v5":"sec..\xa9a\ac,v1"
02:25:16.446163 version@stat F·[1] S·727B[727B] Sc·[0.25]
02:25:16.458164 db@janitor F·3 G·0
02:25:16.458164 db@open done T·23.968ms
===== Nov 4, 2018 (CST) =====
02:25:31.558671 log@legend F·NumFile S·FileSize N·Entry C·BadEntry B·BadBlock Ke·KeyError D·DroppedEntry L·Level Q·SeqNum T·TimeElapsed
02:25:31.558671 version@stat F·[1] S·727B[727B] Sc·[0.25]
02:25:31.558671 db@open opening
02:25:31.558671 journal@recovery F·1
02:25:31.561691 journal@recovery recovering @3
02:25:31.569642 memdb@flush created L0@5 N·1 S·301B "eth..\xe7\xa52,v13":"eth..\xe7\xa52,v13"
02:25:31.573649 version@stat F·[2] S·1KiB[1KiB] Sc·[0.50]
02:25:31.583639 db@janitor F·4 G·0
02:25:31.583639 db@open done T·24.968ms
===== Nov 4, 2018 (CST) =====
```

可以查看到我们的日志内容。实际上我们日志记录内容主要有以下几部分：

1. 操作的具体时间日期
2. 具体的操作
3. 操作路径
4. 操作时间

4. 编写简单的智能合约，在 remix 下进行调试，并部署在链上进行调用

这个过程是比较简单的，因为我们的 remix 是一个部署在浏览器上的 ide 所以不用我们去下载，当然我们想要下载到自己本机上按照博客操作一遍也是可以的。



(1) 浏览器打开 remix IDE 网址: <http://remix.ethereum.org/>



(2) 左边新建一个我们的 sol 文件, 起名为 hello.sol, 中间编写我们的代码。(万能 hello wrold)

代码如下:

```
1  
2 pragma solidity ^0.4.24;  
3 contract HelloWorld{  
4     function say() public pure returns(string){  
5         return "Hello Wrold";  
6     }  
7 }  
8  
9
```

点击右边的编译, 如果出现绿色界面所以没有任何问题, 如果出现黄色, 红色 warning 都得查看一下自己的代码。



区块链课程作业设计报告


Current
version:0.4.25+commit.59dbf8f1.Emscripten.clang

Select new compiler version ▼


☐ Auto compile ☐ Enable Optimization


☐ Hide warnings


↻ Start to compile


HelloWorld ▼  Swarm

Details

 ABI

 Bytecode

Static Analysis raised 1 warning(s) that requires your attention. Click here to show the warning(s). 

HelloWorld 

(3) 编译通过之后，点击上面的 **Detials** 按钮，下滑选择复制下列一段代码

WEB3DEPLOY

```
var helloworldContract = web3.eth.contract([{"constant":true,"inputs":[],"name":"say","outputs":[{"name":"","type":"string"}],"payable":false,"stat
var helloworld = helloworldContract.new(
{
  from: web3.eth.accounts[0],
  data: '0x608060405234801561001057600080fd5b5061013f806100206000396000f300608060405260043610610041576000357e010000000000000000000000000000000000
  gas: '4700000'
}, function (e, contract){
  console.log(e, contract);
  if (typeof contract.address !== 'undefined') {
    console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract.transactionHash);
  }
})
```

复制等待使用

(4) 进入区块链控制台

```
C:\Program Files\Geth>geth --datadir chain1 --nodiscover console
INFO [11-04|10:09:01] Maximum peer count                      ETH=
INFO [11-04|10:09:01] Starting peer-to-peer node              inst
```

解锁账户



区块链课程作业设计报告

```
> personal.unlockAccount(eth.accounts[0], "1234", 3000);
true
> personal.unlockAccount(eth.accounts[1], "1234", 3000);
true
```

(第三个参数是设置解锁的时间，其实不应该过长，会导致不安全，如果不设置，默认一段时间后就重新上锁)

(5) 将复制好的代码在控制台输入

[illegible]

输入成功我们的任务并没有结束，实际上此时并没有部署成功，我们得先进行挖矿，让我们智能合约能够被记录 and 部署

```
> miner.start();
INFO [11-04|10:12:23] Updated mining threads                      threads=0
INFO [11-04|10:12:23] Transaction pool price threshold updated price=180000000
nluN1F10
> [11-04|10:12:23] Starting mining operation
INFO [11-04|10:12:23] Commit new mining work                      number=1  txs=0
> admin.sleepBlocks(1);
INFO [11-04|10:12:57] Successfully sealed new block                number=1  hash=2db9b4...1d6141
INFO [11-04|10:12:57] □□ mined potential block                    number=1  hash=2db9b4...1d6141
INFO [11-04|10:12:57] Commit new mining work                      number=2  txs=0  uncles=0  elapsed:
true
> miner.stop();
null [object Object]
Contract mined! address: 0x84736b2e26c41acc35bed5cf478f734f6c14c8c0 transactionHash: 0x0e68bcb0f32d686alca9199521c16819def2fc4
true
```



挖矿马上就停止，停止之后实际上我们的部署也就成功了，只要出现我们的部署就成功了

Contract mined!

接下来是调用我们部署的智能合约的内容，操作如下：

输入：（sol 的文件名+呼叫的函数+call（））

```
> hello.sayhello.call();  
"hello!"
```

可以看到我们的智能合约在我们的私有链上部署成功并且能够被调用

5. 对交易的字段进行解释

首先我们要的一笔交易，实际上我们上面部署私有链的时候已经进行了一些交易了，并且由于挖矿他们也被保存和记录了下来，现在来查看一下他们的内容

输入和返回信息如下：

```
> web3.eth.getTransactionFromBlock(1, 2)  
{  
  blockHash: "0xe0d8adea6eb7144ec1b68ffff04a2418d2992990298e9f4e1c201364ede5c5bd",  
  blockNumber: 1,  
  from: "0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0", 账户 0  
  gas: 90000,  
  gasPrice: 18000000000,  
  hash: "0xd105347aa0066da5f831a6f0a3fa07617d9cb286b903a0846320a589f1dde18f",  
  input: "0x",  
  nonce: 2,  
  r: "0xe5c39da3f62dcf6ad8ae3dab39dcb20b1659ad854b71c7c33c3862edf0ade58a",  
  s: "0x1c9583d1b394629d7b77b5aebbb7aefbb5e8f3461d199d8a3b400ed804df55a1",  
  to: "0x4fca5d4431ad59476cdfb7eab9b4ee4067af0abe", 账户 1  
  transactionIndex: 2,  
  v: "0x38",  
  value: 2000000000000000000 转账 200  
}  
> eth.accounts  
["0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0", "0x4fca5d4431ad59476cdfb7eab9b4ee4067af0abe"]
```

这是一个我们的账户 0 向账户 1 发送 200 个币的交易记录

来解释一下这些字段和他们的功能：

返回值是一个对象：



blockHash: String - 32 字节。交易所在区块的哈希值。当这个区块处于 pending 将会返回 null。（账户 0 向账户 1 发起交易记录的区块）

blockNumber: Number - 交易所在区块的块号。当这个区块处于 pending 将会返回 null。

from: String - 20 字节，交易发起者的地址。

gas: Number - 交易发起者提供的 gas。.

gasPrice: BigNumber - 交易发起者配置的 gas 价格，单位是 wei。（一个很小的单位）

hash: String - 32 字节，交易的哈希值。

input: String - 交易附带的数据。

nonce: Number - 交易的发起者在之前进行过的交易数量。

to: String - 20 字节，交易接收者的地址。当这个区块处于 pending 将会返回 null。

value: BigNumber - 交易附带的货币量，单位为 Wei。

至此，我们就完成了我们这个 project 的热身报告。实际上写这份报告的时候是很轻松的，但在做的过程中是不断出现各种问题各种 bug 的，由于自己以前没有接触过，所以时间上 debug 是一个很迷的过程，这个热身报告主要也是在一步步的学习和排坑。

希望接下去的项目能够继续挑战自己，但也是希望自己能够少遇到坑，排坑不容易！

感谢您的时间，项目报告到此结束！