

西安电子科技大学

物联网安全实验课程 实验报告

实验名称 搭建私有 ETH

物联网工程 1803041 班

姓名 魏红旭 学号 18030400014

同作者

实验日期 2021 年 6 月 2 日

成 绩

指导教师评语：

指导教师：

年月日

实验报告内容基本要求及参考格式

- 一、实验目的
- 二、实验所用仪器（或实验环境）
- 三、实验基本原理及步骤（或方案设计及理论计算）
- 四、实验数据记录（或仿真及软件设计）
- 五、实验结果分析及回答问题（或测试环境及测试结果）

一、实验目的:

利用 Geth 搭建私有 ETH, 深入理解区块链访问控制机理

二、实验所用仪器 (或实验环境)

计算机科学与技术学院实验中心, 可接入 Internet 网台式机 44 台。

三、实验基本原理及要求

实验原理:

第一步 Geth 安装

https://book.ethereum-jp.net/first_use/installing_geth.html

第二步创建一个私有以太网网

以太坊网中包含多个节点, 这些节点在同一区块链上挖矿并传输 ether。请按照以下步骤创建私有以太坊网。

(1) 创建 Genesis 文件

区块链包含多个区块, 第一个块 (块 0) 被称为 genesis 块。描述该 genesis 块信息的文件被称为 genesis 文件。Genesis 文件以 json 格式创建, 并被定义了区块链的一些参数 (e.g., difficulty, gasLimit)。

Genesis 文件的例子

```
{
  "config": { },
  "nonce": "0x0000000000000042",
  "timestamp": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "extraData": "",
  "gasLimit": "0x8000000",
  "difficulty": "0x4000",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x3333333333333333333333333333333333333333333333333333333333333333",
  "alloc": { }
}
```

在任意目录 (例如 /home/EthereumProject/) 中创建 XXX.json (例如, genesis.json), 写入以上内容, 完成 Genesis 文件的创建。

(2) 创建 ether 节点

首先, 创建一个目录来存储与区块链和节点有关的数据。例如, /home/EthereumProject/node_1。接下来执行以下命令, 使用之前创建的 genesis 文件初始化区块链信息。

```
$geth --datadir /home/EthereumProject/node_1 init /home/EthereumProject/genesis.json
```

这样就完成了一个节点的创建, 然后以相同的方式创建多个节点。

(3) 启动以太网节点

执行以下命令启动节点 (e.g., node 1)。

```
$geth --networkid 23 --nodiscover --datadir /home/EthereumProject/node_1 --port
30304 console 2>>/home/EthereumProject/node_1/log
$geth --networkid 23 --nodiscover --datadir data --port 30304 console 2>>data/log
```

每个参数的含义可以参看

<https://geth.ethereum.org/docs/interface/command-line-options>

--networkid 23: 这是创建的私人以太网 ID.启动该网络的多个节点时，必须指定相同的 ID.可以指定任意的 ID，但这里 ID=23。

--nodiscover: 禁用节点发现机制(手动添加节点)

--datadir: 指定节点数据目录。

--port: 这个节点使用的 port 号，虽然默认是 30303，但是这里使用了 30304

console: 启动对话型 JavaScript 控制台

2>>/home/EthereumProject/node_1/log: 关于节点状态的信息保存到
/home/EthereumProject/node_1/log....文件中

其实，每次启动节点时，都必须输入上述命令，相当麻烦。为了解决这个问题，推荐 bash shell script。首先，制作 XXX.sh（例如 startup.sh）文件，在里面添加以下内容

```
#!/bin/sh
```

```
geth --networkid 23 --datadir /home/EthereumProject/node_1/log \
--port 30304 console 2>>/home/EthereumProject/node_1/log
```

使用 chmod777XXX.sh 命令使 XXX.sh 可执行,以 ./XXX.sh 命令执行 XXX.sh 文件,

```
WOLLM GOM JLS S=P
LMS!

=S:
GOM/1.7.3-S/LLO/LSV=S-M64/1.9.2
M-L:S: LMS=:1.0 L-/:1.0 O/:1.0
M=O=:1.0 O/:1.0 P=SS=L:1.0 OP/:1.0 PPL:1.0
VLO3:1.0
```

启动节点. 启动节点后, JavaScript 控制台显示如下

在这个控制台上, 可以使用 web 3.js 的 JavaScript API 和其他管理 API
Web3.js API Link:

<https://github.com/ethereum/wiki/wiki/JavaScript-API#web3js-api-reference>

管理 API Link:

<https://github.com/ethereum/go-ethereum/wiki/Management-APIs>

例如，如果执行 `admin.nodeInfo` 管理 API，可以看到该节点的信息

```
> admin.nodeInfo
{
  enode:"enode://9d8e4e239d2f0d3cec706c41c4391fabf559bc6a942d23ce03cfd8fb2d669358294ef4178a562219c8602d5f1e8ccb15c4aedf3cceded9a5f5183f3c8bc4f398@[::]:30303?discport=0",
  id:"9d8e4e239d2f0d3cec706c41c4391fabf559bc6a942d23ce03cfd8fb2d669358294ef4178a562219c8602d5f1e8ccb15c4aedf3cceded9a5f5183f3c8bc4f398",
  ip::",
  listenAddr:"[::]:30303",
  name:"Geth/v1.7.3-stable/darwin-amd64/go1.9.2",
  ports:{
    discovery:0,
    listener:30303
  },
  protocols:{
    eth:{
      difficulty:16384,
      genesis:"0x7b2e8be699df0d329cc74a99271ff7720e2875cd2c4dd0b419ec60d1fe7e0432",
      head:"0x7b2e8be699df0d329cc74a99271ff7720e2875cd2c4dd0b419ec60d1fe7e0432",
      network:16
    }
  }
}
```

ock (0) 的 web 3. js API, 就能看到 (1) 制作的 Genesis 块的信息

这里可以把 web 3.eth 省略为 eth，同样地，web 3.eth.getBlock (0) 变为 eth.getBlock(0)，另外，也可以把 web 3.net、web3.sh 和 web3.db 省略为 net、shh 和 db，但是 web3.version 不能省略为 version。

(4) 将建立的以太网节点连接到一个以太网上

首先，根据步骤(3)启动多个节点. 这里必须注意的是 1)必须指定相同的 networkid
2) 必须使用不同的 port. 在启动的节点的控制台中执行 admin.pers API，看其他节点的信息。

```
.....
.....
.....
```

什么都看不见?? 是的。因为每个节点还没有添加其他节点.接下来，使用 admin.addPeer (url string) API 添加其他节点 . 参数 string 是 “nodeID@IP_address:Port_Number 的形式。例如，在节点 1 中追加节点 2。

node1 的信息:

```
Q_ <△(M)=∅.∅□△○|∅//□
(E)
  ∅□△△:"∅□△△://9△8○4○239△2//0△3□○□706△41△439
1//<_//559_□6<942△23□○03□//△8//_2△669358294○//417
8<562219△8602△5//1○8□□_15□4<○△//3□□○△○△9<5
//5183//3□8_□4//398(F[::]:30303(ST)△=S□P□◎✓✓=0",
  =△:"9△8○4○239△2//0△3□○□706△41△4391//<_//559_
□6<942△23□○03□//△8//_2△669358294○//4178<562219△86
02△5//1○8□□_15□4<○△//3□□○△○△9<5//5183//3□8_
□4//398",
  =P: ":",
  L=S✓✓○∅A△△◎: "[::]:30303",
  ∅<(M)○:
"GO✓✓✓/1.7.3-S✓✓<_L○/△<◎✓=∅-<(M)△64/△/□1.9.2",
  P□◎✓✓S: E
  △=S□□□□>: 0,
  L=S✓✓○∅◎◎: 30303
(R),
  P○□✓✓□□□L(S): E
  ○✓✓✓: E
  △=//==□-L✓✓>: 16384,
  △/∅∅S=S:"0▽7_2○8_□699△//0△329□□74<99271//77
20○2875△△2△4△△0_419○△60△1//○7○0432",
```

node2 的信息:

```

Q ∠△(M)≡∅.∅□△○∅//□
(E)
○∅□△○:
"○∅□△○://2//15//75□9┘898∠9//┘928□45794□//635┘
△78△△6□37○451006//8//∠○683△△2┘∠○1∠1□18942716
┘0┘┘△81○4∠00992634○64403774528△○89//12△7┘62784△4
△△9○○26(F[:]:30304(ST)△≡$□P□◎∇∇=0",
=△:
"2//15//75□9┘898∠9//┘928□45794□//635┘△78△△6□
37○451006//8//∠○683△△2┘∠○1∠1□18942716┘0┘┘△81○4
∠00992634○64403774528△○89//12△7┘62784△4△△9○○26",
=△:
"::",
L=S∇∇,○∅A△△◎: "[::]:30304",
∅∠(M)○:
"G○∇∇/┘1.7.3-($∇∇┘┘L○/△∠◎∇≡∅-∠(M)△64/∇□1.9.2
",
P□◎∇∇S: (E)
△=S□□□□: 0,
L=S∇∇○∅○◎: 30304
(R),
P○□∇∇□□□L(S): (E)
○∇∇: (E)
△=//=□-L∇∇: 16384,
∇○∅○S≡S:
"0▽7┘2○8┘○699△//0△329□□74∠99271//7720○2875□△

```

节点 1 的控制台执行以下命令，可以添加节点 2

```
admin.addPeer("enode://2f15ff75c9b898a9fb928c45794cf635bd78dd6c37e451006f8fae683dd2bae1a1c18942716b0bd81e4a00992634e64403774528de89f12d7b62784d4dc9ee26@[::]:30304")
```

[::]可以修改为 127.0.0.1。在节点 1 和节点 2 的控制台上再执行一次 admin.pers，就可以看到其他节点的信息。这样，私人的以太网网络的创建就完成了。

但是，重新启动节点的话，其他节点也会看不见。所以，每次启动的时候，必须用 admin.addPeer 追加其他的节点。节点的数量多的话很麻烦。其实，也有在启动节点时追加其他节点的方法。首先，制作 static-nodes.json 文件，写入以下内容。

```
[
    "node_ID1@IP_address1:Port_Number1"
    "node_ID2@IP_address2:Port_Number2"
    "node_ID3@IP_address3:Port_Number3"
]
```

在前面的例子中，节点 1 侧的 static-nodes.json 文件如下

```
[
    "enode://2f15ff75c9b898a9fb928c45794cf635bd78dd6c37e451006f8fae683dd2bae1a1c18942716b0bd81e4a00992634e64403774528de89f12d7b62784d4dc9ee26@[::]:30304"
]
```

节点 2 侧的文件如下所示

```
[
    "enode://9d8e4e239d2f0d3cec706c41c4391fabf559bc6a942d23ce03cfd8fb2d669358294ef4178a562219c8602d5f1e8ccb15c4aedf3cceded9a5f5183f3c8bc4f398@[::]:30303"
]
```

接下来,将各自的 static-nodes.json 文件放在对应的节点的 datadiar/geth 目录上。最后,重新启动所有节点。

(5) 在每个以太网节点上创建账户

为了在区块链上挖掘或执行智能合约,需要以太网账户。请使用各个节点的控制台创建账户,执行 personal.newAccount() API,输入和确认 Passphrase (密码),账户创建完成。请绝对不要忘记 Passphrase。用 eth.accounts 命令确认创建的账户。

接下来,请参照以下链接进行账户之间的汇款和挖掘操作。

<https://enomotodev.hatenablog.com/entry/2018/02/18/182032>

汇款时,需要解除账户锁定.可以使用控制台进行此操作,但启动节点时想要自动解除时,请使用以下命令。

```
$geth --unlock value --password file
```

value: 打算解除锁定的账号的 ID,例如 0, 1; 存储密码的文件,如
/home/EthereumProject/node_1/password:

第三步: 建立智能合约

Solidity 语言创建智能合约,关于 Solidity 的入门资料请参照以下链接。

<https://solidity-cn.readthedocs.io/zh/develop/>

<https://book.ethereum-jp.net/solidity/basic.html>

关于智能合约的编译和向区块链上的部署,请参照以下链接。

https://book.ethereum-jp.net/first_use/contract.html

使用 browser solidity 时,请参照以下链接。

<http://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.1+commit.dfl93b15.js> (contract IDE)

https://book.ethereum-jp.net/first_use/realtimecompiler.html

如果一台机器启动了多个节点,则每个节点必须指定不同的 rpcport。

第四步: 创建 Web3 JavaScript

在步骤 2 中通过控制台知道了操作 geth 的方法,但实际上也可以使用 Web 3.js 操作 geth。需要的模块是 Web 3.js 和 Node.js。

首先,安装 [Node.js](https://nodejs.org/ja/)。https://nodejs.org/ja/

安装 Node.js 后,npm 也会自动安装。然后,使用 npm 安装 Web 3.js。

<https://qiita.com/kolife/items/d936d4aa6cdd23ff65b4>

<https://github.com/ethereum/wiki/wiki/JavaScript-API>

安装完成后,制作简单的 javascript 文件(例如 test.js),用 node 试着执行(命令: node test.js)。

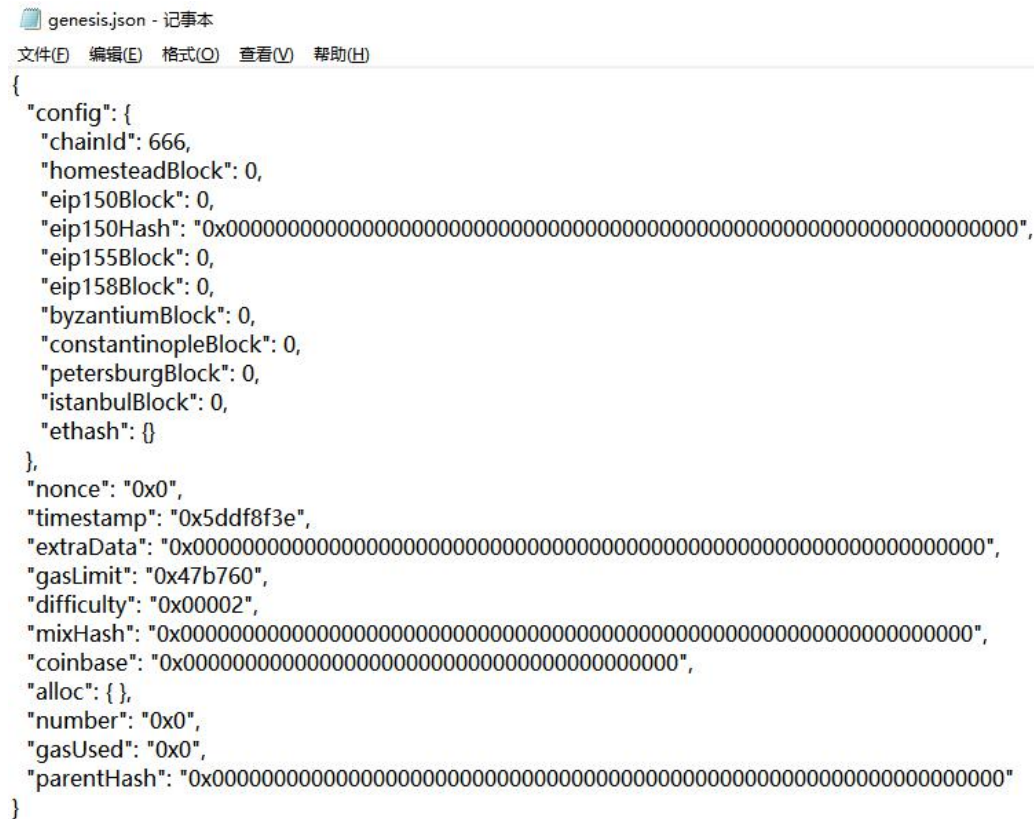
实验要求:

利用 Geth 搭建 ETH,给出实验过程中的必要步骤说明和结果

四、实验步骤及实验数据记录：（要有文字描述和必要截图）

● 创建私有以太坊网：

1) 创建 Genesis 文件



genesis.json - 记事本

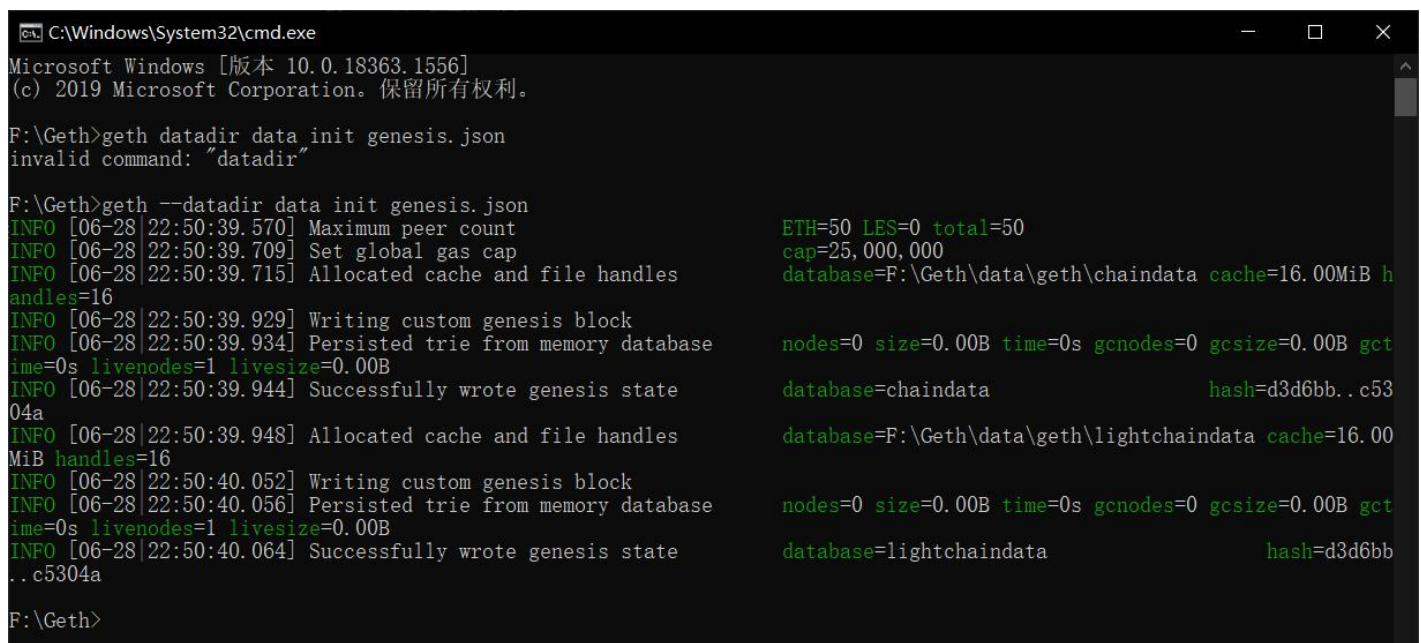
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
{
  "config": {
    "chainId": 666,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "ethash": {}
  },
  "nonce": "0x0",
  "timestamp": "0x5ddf8f3e",
  "extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "gasLimit": "0x47b760",
  "difficulty": "0x00002",
  "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "alloc": {},
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

2) 创建两个节点

```
geth --datadir data init genesis.json
```

```
geth --datadir data1 init genesis.json
```



C:\Windows\System32\cmd.exe

Microsoft Windows [版本 10.0.18363.1556]
(c) 2019 Microsoft Corporation。保留所有权利。

F:\Geth>geth datadir data init genesis.json
invalid command: "datadir"

F:\Geth>geth --datadir data init genesis.json

```
INFO [06-28|22:50:39.570] Maximum peer count          ETH=50 LES=0 total=50
INFO [06-28|22:50:39.709] Set global gas cap          cap=25,000,000
INFO [06-28|22:50:39.715] Allocated cache and file handles database=F:\Geth\data\geth\chaindata cache=16.00MiB handles=16
INFO [06-28|22:50:39.929] Writing custom genesis block
INFO [06-28|22:50:39.934] Persisted trie from memory database nodes=0 size=0.00B time=0s gcnodes=0 gsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [06-28|22:50:39.944] Successfully wrote genesis state database=chaindata hash=d3d6bb..c5304a
INFO [06-28|22:50:39.948] Allocated cache and file handles database=F:\Geth\data\geth\lightchaindata cache=16.00MiB handles=16
INFO [06-28|22:50:40.052] Writing custom genesis block
INFO [06-28|22:50:40.056] Persisted trie from memory database nodes=0 size=0.00B time=0s gcnodes=0 gsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [06-28|22:50:40.064] Successfully wrote genesis state database=lightchaindata hash=d3d6bb..c5304a
```

F:\Geth>


```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.18363.1556]
(c) 2019 Microsoft Corporation。保留所有权利。

F:\Geth>geth --datadir datal init genesis.json
INFO [06-28|23:15:31.356] Maximum peer count
INFO [06-28|23:15:31.472] Set global gas cap
INFO [06-28|23:15:31.477] Allocated cache and file handles
handles=16
INFO [06-28|23:15:31.687] Writing custom genesis block
INFO [06-28|23:15:31.691] Persisted trie from memory database
time=0s livenodes=1 liveness=0.00B
INFO [06-28|23:15:31.699] Successfully wrote genesis state
304a
INFO [06-28|23:15:31.703] Allocated cache and file handles
OMiB handles=16
INFO [06-28|23:15:31.811] Writing custom genesis block
INFO [06-28|23:15:31.815] Persisted trie from memory database
time=0s livenodes=1 liveness=0.00B
INFO [06-28|23:15:31.822] Successfully wrote genesis state
b..c5304a
F:\Geth>
```

3) 启动以太网节点:

启动第一个节点:

```
$geth --networkid 23 --nodiscover --datadir data --port 30304
console 2>>data/log
```

启动第二个节点: 需要注意的是, 命令并不完全相同, 相同时会出现资源被占用, 冲突的情况

```
$geth --networkid 23 --nodiscover --datadir datal --port 30305
--ipcdisable --rpc --rpcaddr "localhost" --rpcport "8546"
--rpccorsdomain "*" --rpcapi "db,eth,net,web3" console
2>>datal/log
```

```
F:\Geth>geth --networkid 23 --nodiscover --datadir data --port 30304 console 2>>data/log
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3
at block: 0 (Thu Nov 28 2019 17:11:26 GMT+0800 (CST))
datadir: F:\Geth\data
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

To exit, press ctrl-d
> _
```

```
C:\Windows\System32\cmd.exe - geth --networkid 23 --nodiscover --datadir datal --port 30305 --ipcdisable --rpc --rpcaddr "localhost"...
Microsoft Windows [版本 10.0.18363.1556]
(c) 2019 Microsoft Corporation。保留所有权利。

F:\Geth>geth --networkid 23 --nodiscover --datadir datal --port 30305 --ipcdisable --rpc --rpcaddr "localhost" --rpcport
"8546" --rpccorsdomain "*" --rpcapi "db,eth,net,web3" console 2>>datal/log
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3
at block: 0 (Thu Nov 28 2019 17:11:26 GMT+0800 (CST))
datadir: F:\Geth\datal
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

To exit, press ctrl-d
```

4) 执行 admin.nodeInfo 管理 API，可以看到该节点的信息

```
> admin.nodeInfo
{
  enode: "enode://d029bc6f18616683347623d9a4c9bed7dbc72014cfbeda359b2eb958317b1cfff48e4bfa1a83b1eb135a976afb79e3ed4977e916dbf1d6a07fedaaa38b846d310127.0.0.1:30304?discport=0",
  enr: "enr:-Ja4QFqVg30-HA0vlpESUmchyuhnlT3i_jGX9xlKamdnR4KFcw10i4AVCXFabjLr4XB4MhxKF87pHiY0WgmF4hXWZckGg2V0aMfGhJPR68WAgmlkgnY0gm1whH8AAAGJc2VjcDI1NmsxoQPQKbxyGGFmgzR2I9mkYb7X28cgFM--2jWbLr1YMXsc_4RzbmFwwIN0Y3CCdmA",
  id: "7612391a1e4a1d296cda327a6934f71d0b6297949b023eb36c5764bd66306ce1",
  ip: "127.0.0.1",
  listenAddr: "[::]:30304",
  name: "Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3",
  ports: {
    discovery: 0,
    listener: 30304
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 0,
        chainId: 666,
        constantinopleBlock: 0,
        eip150Block: 0,
        eip150Hash: "0x0000000000000000000000000000000000000000000000000000000000000000",
        eip155Block: 0,
        eip158Block: 0,
        ethash: {},
        homesteadBlock: 0,
        istanbulBlock: 0,
        petersburgBlock: 0
      },
      difficulty: 2,
      genesis: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
      head: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
      network: 23
    }
  },
  snap: {}
}
```

```
To exit, press ctrl-d
> admin.nodeInfo
{
  enode: "enode://0e9a1e71678cd06ff541c727a351c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee9d6b1240470a88bf558a3a98bc12f7b670127.0.0.1:30305?discport=0",
  enr: "enr:-Ja4QG5yVlitxGhq474o4Ejt7a1_o07hvnY1zXdc258AHc5-NkuJmSOkcfhmSjkeL8nfqUmzftCMvhd2bV14KfnTGzDwFg2V0aMfGhJPR68WAgmlkgnY0gm1whH8AAAGJc2VjcDI1NmsxoQM0mh5xZ4zQb_VBxyejUcJZLXE1ExWfNoy7BGMFxjbJLYRzbmFwwIN0Y3CCdmE",
  id: "b4840d4065712f8cbf4d63aae50fe3240366f4558c9a4b06e077983aff3f456a",
  ip: "127.0.0.1",
  listenAddr: "[::]:30305",
  name: "Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3",
  ports: {
    discovery: 0,
    listener: 30305
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 0,
        chainId: 666,
        constantinopleBlock: 0,
        eip150Block: 0,
        eip150Hash: "0x0000000000000000000000000000000000000000000000000000000000000000",
        eip155Block: 0,
        eip158Block: 0,
        ethash: {},
        homesteadBlock: 0,
        istanbulBlock: 0,
        petersburgBlock: 0
      },
      difficulty: 2,
      genesis: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
      head: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
      network: 23
    }
  },
  snap: {}
}
```

5) 执行 `web3.eth.getBlock(0)` 的 `web3.js` API, 就能看到 (1) 制作的 Genesis 块的信息

```
C:\Windows\System32\cmd.exe - geth --networkid 23 --nodiscover --datadir data --port 30304 console
```

```
{  
  snap: {}  
}  
  
eth.getBlock(0)  
  
{  
  difficulty: 2,  
  extraData: "0x0000000000000000000000000000000000000000000000000000000000000000",  
  gasLimit: 4700000,  
  gasUsed: 0,  
  hash: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbbb1bfd6e59141c2e0bc5304a",  
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000  
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",  
  miner: "0x0000000000000000000000000000000000000000000000000000000000000000",  
  mixHash: "0x0000000000000000000000000000000000000000000000000000000000000000",  
  nonce: "0x0000000000000000",  
  number: 0,  
  parentHash: "0x0000000000000000000000000000000000000000000000000000000000000000",  
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cad001622fb5e363b421",  
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccdd41ad312451b948a7413f0a142fd40d49347",  
  size: 540,  
  stateRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cad001622fb5e363b421",  
  timestamp: 1574932286,  
  totalDifficulty: 2,  
  transactions: [],  
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cad001622fb5e363b421",  
  uncles: []  
}
```

6) 将建立的以太网节点连接到一个以太网上

首先，根据步骤（3）启动多个节点. 这里必须注意的是 1）必须指定相同的 `networkid` 2）必须使用不同的 `port`. 在启动的节点的控制台中执行 `admin.pers API`，看其他节点的信息。

节点 1 的控制台执行以下命令，可以添加节点 2：

```
admin.addPeer("enode://0e9a1e71678cd06ff541c727a351c2592d71351
315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee9d6b12
40470a88bf558a3a98bc12f7b67@127.0.0.1:30305?discport=0")
```

添加完成之后我们通过 `admin.peers`，查看添加的节点信息，如下图所示：


```

> admin.addPeer("enode://0e9a1e71678cd06ff541c727a351c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee> admin.addPeer("
enode://0e9a1e71678cd06ff541c727a351c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee> admin.addPeer("enode://0e9a1e716
78cd06ff541c727a351c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee> admin.addPeer("enode://0e9a1e71678cd06ff541c727a3
51c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee9d6b1240470a88bf558a3a98bc12f7b67@127.0.0.1:30305?discport=0")

true
> admin.peers
[
  {
    caps: ["eth/65", "eth/66", "snap/1"],
    enode: "enode://0e9a1e71678cd06ff541c727a351c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee9d6b1240470a88bf558a3a98bc12f7b67@127.0.0.1:30305?discport=0",
    id: "b4840d4065712f8cbf4d63aae50fe3240366f4558c9a4b06e077983aff3f456a",
    name: "Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3",
    network: {
      inbound: false,
      localAddress: "127.0.0.1:1082",
      remoteAddress: "127.0.0.1:30305",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 2,
        head: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
        version: 66
      },
      snap: {
        version: 1
      }
    }
  }
]
>
>

```

```

C:\Windows\System32\cmd.exe - geth --networkid 23 --nodiscover --datadir c:\windows\system32\cmd.exe - geth --networkid 23 --nodiscover --datadir data1 --port 30305 --ipcdisable --rpc --rpcapi
enode://0e9a1e71678cd06ff541c727a351c2592d71351315ab368cbb046305c636e32dc554ace029aa552826859dd58b333ee9d6b1240470a88bf558a3a98bc12f7b67@127.0.0.1:30305?discport=0",
id: "b4840d4065712f8cbf4d63aae50fe3240366f4558c9a4b06e077983aff3f456a",
name: "Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3",
ports: {
  discovery: 0,
  listener: 30305
},
protocols: {
  eth: {
    config: {
      byzantiumBlock: 0,
      chainId: 666,
      constantinopleBlock: 0,
      eip150Block: 0,
      eip150Hash: "0x0000000000000000000000000000000000000000000000000000000000000000",
      eip155Block: 0,
      eip158Block: 0,
      ethash: {},
      homesteadBlock: 0,
      istanbulBlock: 0,
      petersburgBlock: 0
    },
    difficulty: 2,
    genesis: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
    head: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
    network: 23
  },
  snap: {}
}
}
}
]
> personal.newAccount("aaaaaa")
"0x221d563ba416c987a46ea659e712ea754c0e13f5"
> eth.accounts
["0x221d563ba416c987a46ea659e712ea754c0e13f5"]
>
>
C:\Windows\System32\cmd.exe - geth --networkid 23 --nodiscover --datadir data1 --port 30305 --ipcdisable --rpc --rpcapi
id: "b4840d4065712f8cbf4d63aae50fe3240366f4558c9a4b06e077983aff3f456a",
ip: "127.0.0.1",
listenAddr: "[:]:30305",
name: "Geth/v1.10.3-stable-991384a7/windows-amd64/go1.16.3",
ports: {
  discovery: 0,
  listener: 30305
},
protocols: {
  eth: {
    config: {
      byzantiumBlock: 0,
      chainId: 666,
      constantinopleBlock: 0,
      eip150Block: 0,
      eip150Hash: "0x0000000000000000000000000000000000000000000000000000000000000000",
      eip155Block: 0,
      eip158Block: 0,
      ethash: {},
      homesteadBlock: 0,
      istanbulBlock: 0,
      petersburgBlock: 0
    },
    difficulty: 2,
    genesis: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
    head: "0xd3d6bb893a6e274cab241245d5df1274c58d664fbb1bfd6e59141c2e0bc5304a",
    network: 23
  },
  snap: {}
}
}
}
]
> personal.newAccount("bbbbbb")
"0x0ec1df168f302f4a1d818895889e51350b338107"
> eth.accounts
["0x0ec1df168f302f4a1d818895889e51350b338107"]
>
>

```

7) 在每个以太网节点上创建账户

为了在区块链上挖掘或执行智能合约，需要以太网账户。请使用各个节点的控制台创建账户，执行 `personal.newAccount()` API，输入和确认 Passphrase（密码），账户创建完成。请绝对不要忘记 Passphrase。用

eth.accounts 命令确认创建的账户。

◆ 创建两个账户，并进行验证：

```
>
> personal.newAccount("aaaaaa")
"0x221d563ba416c987a46ea659e712ea754c0e13f5"
> eth.accounts
["0x221d563ba416c987a46ea659e712ea754c0e13f5"]
> personal.newAccount("cccccc")
"0xf92490c1e751166effc269bfde92e258008879fc"
> eth.accounts
["0x221d563ba416c987a46ea659e712ea754c0e13f5", "0xf92490c1e751166effc269bfde92e258008879fc"]
>
_
```

◆ 开始挖矿，eth.mining 可以使用命令检查现在是否已开始挖矿：过一段时间，等挖矿进展到一定程度就停止挖矿；如果挖矿停止 eth.mining，您将在执行命令时 false 收到一条消息。

```
> eth.accounts
["0x221d563ba416c987a46ea659e712ea754c0e13f5", "0xf92490c1e751166effc269bfde92e258008879fc"]
> miner.start(0)
null
> eth.mining
true
> miner.stop()
null
> eth.mining
false
>
_
```

◆ 检查账户余额：使用 eth.getBalance 检查账户余额：

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

◆ 开锁：为了防止由于错误操作而导致的错误汇款，并且在汇款时，必须使用创建帐户时使用的密码进行解锁。

```
> personal.unlockAccount(eth.accounts[0])
Unlock account 0x0ec1df168f302f4a1d818895889e51350b338107
Passphrase:
GoError: Error: account unlock with HTTP access is forbidden at web3.js:6347:37(47)
    at native
    at <eval>:1:24(6)
```

需要注意的是，在新版本中，对操作权限进行了限定，所以我们需要在启动节点时加入--allow-insecure-unlock；加入后我们再次进行尝试，开锁成功；

```

0
> personal.unlockAccount(eth.accounts[0])
Unlock account 0x221d563ba416c987a46ea659e712ea754c0e13f5
Passphrase:
true
>

```

- ◆ 汇款：首先确认汇款目的地 eth.accounts[1] 的余额 0

```

true
> web3.fromWei (eth.getBalance (eth.accounts [1]), "ether")
0
>

```

接下来，尝试 accounts[0] 向 accounts[1] 货币中发送 10 以太币。

```

> eth.sendTransaction ({from: eth.accounts [0], to: eth.accounts [1], value: web3.toWei (10, "ether")})
Error: insufficient funds for transfer
    at web3.js:6347:37(47)
    at web3.js:5081:62(37)
    at <eval>:1:21(21)

```

因为我们两个账户的余额均为 0，所以会显示发送失败：

- ◆ 重新锁定账户：

```

> personal.lockAccount(eth.accounts[0])
true
> exit

```

- 整体过程如下图所示，至此任务二已经全部完成

```

> eth.accounts
["0x221d563ba416c987a46ea659e712ea754c0e13f5", "0xf92490c1e751166effc269bfde92e258008879fc"]
> miner.start(0)
null
> eth.mining
true
> miner.stop()
null
> eth.mining
false
> eth.getBalance(eth.accounts[0])
0
> eth.getBalance(eth.accounts[1])
0
> personal.unlockAccount(eth.accounts[0])
Unlock account 0x221d563ba416c987a46ea659e712ea754c0e13f5
Passphrase:
true
> web3.fromWei (eth.getBalance (eth.accounts [1]), "ether")
0
> eth.sendTransaction ({from: eth.accounts [1], to: eth.accounts [0], value: web3.toWei (10, "ether")})
Error: insufficient funds for transfer
    at web3.js:6347:37(47)
    at web3.js:5081:62(37)
    at <eval>:1:21(21)

> eth.sendTransaction ({from: eth.accounts [0], to: eth.accounts [1], value: web3.toWei (10, "ether")})
Error: insufficient funds for transfer
    at web3.js:6347:37(47)
    at web3.js:5081:62(37)
    at <eval>:1:21(21)

> personal.lockAccount(eth.accounts[0])
true
>

```

- 建立智能合约:

- 1) 安装 node.js, 使用 npm 安装 solc, 需要注意的是, 在安装完成之后, 我们需要手动添加环境变量, 否则会检测不到 solc 命令。

```
管理员: 命令提示符

F:\>cd Geth

F:\Geth>npm install -g solc@0.6.0
D:\Program Files\nodejs\node_global\solcjs -> D:\Program Files\nodejs\node_global\node_modules\solc\solcjs
+ solc@0.6.0
added 25 packages from 15 contributors in 22.738s

F:\Geth>_
```

安装完成后进行检测, 出现以下界面代表安装成功:

```
C:\WINDOWS\system32>solcjs --help
Usage: solcjs [options]

Options:
  -V, --version                output the version number
  --version                    Show version and exit.
  --optimize                   Enable bytecode optimizer.
  --bin                        Binary of the contracts in hex.
  --abi                        ABI of the contracts.
  --standard-json              Turn on Standard JSON Input / Output mode.
  -o, --output-dir <output-directory> Output directory for the contracts.
  -h, --help                   output usage information

C:\WINDOWS\system32>solcjs --version
0.6.0+commit.26b70077.Emscripten.clang
```

- 2) 创建智能合约代码文件, 此代码为实验文档网址中提供的代码:

```
SingleNumRegister.sol - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

pragma solidity ^0.6.0;
contract SingleNumRegister {
    uint storedData;
    function set(uint x) public{
        storedData = x;
    }
    function get() public view returns (uint retVal){
        return storedData;
    }
}
```


3) 使用 solc 编译智能合约文件:

```
F:\Geth>solcjs --abi --bin SingleNumRegister.sol
SingleNumRegister.sol:7:27: ParserError: The state mutability modifier "constant" was removed in version 0.5.0. Use "view" or "pure" instead.
    function get() public constant returns (uint retVal){
                          ^~~~~~
```

编译时出现了提示, 需要修改合约文件中的一个关键词, 我们跟着提示修改就可以, 编译成功后出现了以下文件:

名称	修改日期	类型	大小
SingleNumRegister_sol_SingleNumRegister.abi	2021/6/29 22:15	ABI 文件	1 KB
SingleNumRegister_sol_SingleNumRegister.bin	2021/6/29 22:15	BIN 文件	1 KB
SingleNumRegister.sol	2021/6/29 22:14	SOL 文件	1 KB

SingleNumRegister_sol_SingleNumRegister.bin - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
60806040523480156100115760006000fd5b50610017565b60db806100256000396000f3fe60806040523480156
```

SingleNumRegister_sol_SingleNumRegister.abi - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
[{"inputs":[],"name":"get","outputs":[{"internalType":"uint256","name":"retVal","type":"uint256"}],"stateMutability":"view","type":"function"}]
```

4) 创建“合同”账户, 并访问:

合约代码已经编译完成, 但编译后的代码只在合约创建者的节点上, 以太坊网络中尚无任何人可以访问该合约。只有在将此编译代码发送到以太坊网络并让矿工在区块链上注册后, 其他用户才能访问此合约。可以通过从 EOA 生成和发送交易将创建的合约发送到以太坊网络。启动 geth, 在 geth 的提示下, 首先将编译结果存入一个合适的变量中。

```
> var bin="0x60806040523480156100115760006000fd5b50610017565b60db806100256000396000f3fe60806040523480156100105760006000fd5b506004361060365760003560e01c806360fe47b114603c5780636d4ce63c146068576036565b60006000fd5b60666004803603602081101560515760006000fd5b81019080803590602001909291905050506084565b005b606e6094565b6040518082815260200191505060405180910390f35b8060006000508190909055505b50565b6000600060005054905060a2565b9056fea264697066735822122021c2ffc40ee29c03864dac992661d97c84efb6fca2a90685df8357c0b5e83bd64736f6c63430006000033"
undefined
> var abi=[{"inputs":[],"name":"get","outputs":[{"internalType":"uint256","name":"retVal","type":"uint256"}],"stateMutability":"view","type":"function"}, {"inputs":[{"internalType":"uint256","name":"x","type":"uint256"}],"name":"set","outputs":[],"stateMutability":"nonpayable","type":"function"}]
undefined
> var contract=eth.contract(abi)
undefined
> var myContract = contract.new({ from: eth.accounts[0], data: bin})
Error: insufficient funds for transfer
    at web3.js:6347:37(47)
    at web3.js:5081:62(37)
    at web3.js:3021:48(134)
    at <eval>:1:30(13)
```


● 创建 Web3 JavaScript

制作简单的 javascript 文件(例如 test.js), 用 node 试着执行(命令: node test.js)。

1) 制作简单的 JavaScript 文件:



2) 使用 node 命令进行执行:



五、实验结果分析及实验总结与体会

此次实验的主要任务安装 Geth, 并利用其创建私有以太坊网, 通过此次实验, 我熟悉了创建区块的整体过程, 学习了创建 ether 节点、启动 ether 节点、连接 ether 节点、在节点中创建多个账户、实现账户挖矿、账户之间转账、建立智能合约、利用 node 运行 js 文件的完整步骤, 收获颇丰, 其实在这次实验中遇到的问题有很多, 多数都是有关命令的问题, 部分是关于权限以及安装中出现的问题, 基本上都可以通过查阅资料解决。通过此次实验, 我感受到密码学与区块链之间的紧密来联系, 密码学保障了区块链的安全性和匿名性, 同时我也对于区块链访问控制有了更加深入的理解。