

华东师范大学数据科学与工程学院实验报告

课程名称: 区块链与分享型数据库	年级: 2018	上机实践成绩:
指导教师: 张召	姓名: 池欣宁	学号: 10185501409
上机实践名称: 以太坊部署和编程		上机实践日期:
上机实践编号:	组号:	上机实践时间:

一、实验目的

- (1) 掌握 solidity 结构体、数组、mapping 等数据结构, require 函数及基本代码的编写
- (2) 熟悉 Python 通过以太坊 rpc 接口部署并调用合约的过程
- (3) 熟悉以太坊依赖环境及相关库
- (4) 掌握 solidity 合约的编写、编译和部署

二、实验任务

- (1) 熟悉以太坊依赖环境及相关库
- (2) 掌握 solidity 合约的编写、编译和部署

三、使用环境

以太坊环境: go/Node.js/npm/truffle/solidity,solc/vscode/Geth

四、实验过程

第一部分: 以太坊单节点搭建及合约部署

首先执行 `solc--version` 检查以太坊依赖环境及相关库是否安装完成

```
dase@ubuntu:~$ solc --version
solc, the solidity compiler commandline interface
Version: 0.8.3+commit.8d00100c.Linux.g++
```

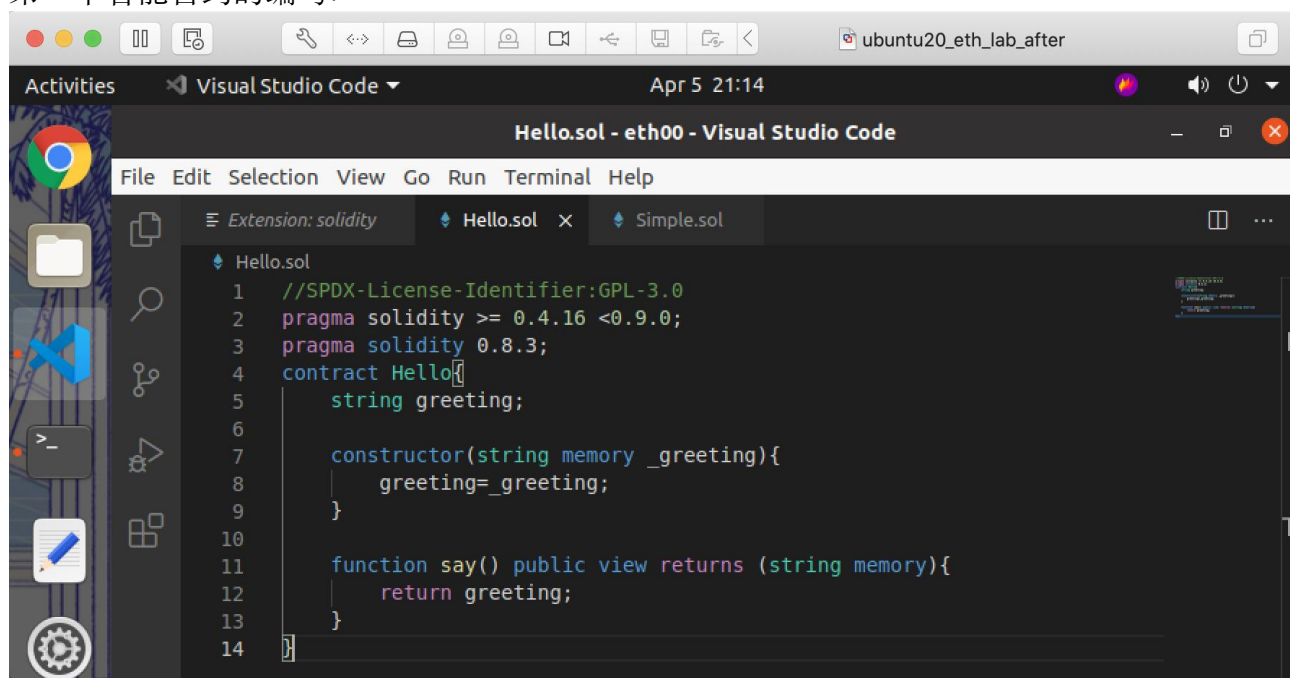
开发模式下以太坊默认账户拥有大量的余额, 如果用该账户来部署合约将无法看到余额变化, 故创建一个新的账户进行下面的实验 (注意, 图中的地址需为实际实验环境下的, 后面也是如此, 不多赘述), “123”是新账户的密码, 实验时可自行设定, 可看到新账户的余额为 0。

```

dase@ubuntu: ~/eth00
> personal.newAccount("123") 新建账户, 密码为123
INFO [03-31|19:05:17.038] Your new key was generated          address=0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a
WARN [03-31|19:05:17.046] Please backup your key file!          path=/home/dase/eth00/testNet/keystore/UTC--2021-04-01T02-05-11.1
50083518Z--1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a
WARN [03-31|19:05:17.046] Please remember your password!
"0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a" 账户地址为0x开头的字符串
> eth.accounts
["0xe03973b8bb000dc28b17652bc58ad944b81fc35e", "0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a"]
> eth.getBalance(eth.accounts[1])
0 查看账户余额
> eth.sendTransaction({from: eth.accounts[0],to:eth.accounts[1],value:web3.toWei(1,"ether")}) 转账, 第一步交易
INFO [03-31|19:10:53.278] Setting new local account          address=0xe03973b8bb000dc28b17652bc58ad944b81fc35e
INFO [03-31|19:10:53.281] Submitted transaction          hash=0x24aaf0578045b2194c755bd370b674606632b85c935a5c6febb8c20749ce0b6c
ce0b6c from=0xe03973b8bb000dc28b17652bc58ad944b81fc35e nonce=0 recipient=0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a value=1000000000
000000000
"0x24aaf0578045b2194c755bd370b674606632b85c935a5c6febb8c20749ce0b6c"
> INFO [03-31|19:10:53.286] Commit new mining work          number=1 sealhash="48b4d6...2fab55" uncles=0 txs=1 gas=21000 fees
=2.1e-14 elapsed=5.445ms
INFO [03-31|19:10:53.290] Successfully sealed new block          number=1 sealhash="48b4d6...2fab55" hash="f1d7f5...752e76" elapsed=4.
227ms
INFO [03-31|19:10:53.290] mined potential block          number=1 hash="f1d7f5...752e76"
INFO [03-31|19:10:53.291] Commit new mining work          number=2 sealhash="a8208e...075905" uncles=0 txs=0 gas=0 fees=0
elapsed="836.448µs"
INFO [03-31|19:10:53.293] Sealing paused, waiting for transactions
INFO [03-31|19:10:53.295] Commit new mining work          number=2 sealhash="a8208e...075905" uncles=0 txs=0 gas=0 fees=0
elapsed=4.518ms
> eth.getBalance(eth.accounts[1])
10000000000000000000
eth.accounts[1]

```

第一个智能合约的编写:



```

Hello.sol - eth00 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Extension: solidity Hello.sol x Simple.sol
Hello.sol
1 //SPDX-License-Identifier:GPL-3.0
2 pragma solidity >= 0.4.16 <0.9.0;
3 pragma solidity 0.8.3;
4 contract Hello{
5     string greeting;
6
7     constructor(string memory _greeting){
8         greeting=_greeting;
9     }
10
11     function say() public view returns (string memory){
12         return greeting;
13     }
14 }

```

调用智能合约账户和调用某个 object 的某个方法是一个道理, 唯一的不同在于, 你所有的信息都要处于交易之中, 像比特币一样, 你也要形成交易, 签名, 发给网络执行。

所以这里 Hello 便是我们编写的第一个智能合约。

注意编译的时候要将上面那一行//SPDX-License-Identifier:GPL-3.0 加上, 不然无法编译通过。

注意这里 `var helloinstance =hellocontract.new("hello world",tx)`

这里即将 hello world 作为字符串参数传入 contract 的 constructor 中

```

> var helloInstance = HelloContract.new( Hello world , tx)
INFO [03-31|19:47:26.214] Setting new local account                address=0x1fB6C7A8c9dd8
9C7D18C746DD494c7f2cD177F4a
INFO [03-31|19:47:26.224] Submitted contract creation            hash=0xc143527d62fe31fd
f6c6815dc05a86e3f11edfc958bbc6d0dbb0ef79ee1b6e76 from=0x1fB6C7A8c9dd89C7D18C746DD494c7f2cD
177F4a nonce=0 contract=0x78D0D81060b01A89a46F6Ab7525Fe6e54FC05047 value=0
undefined
> INFO [03-31|19:47:26.251] Commit new mining work                number=2 sealhash="d1
8598...436291" uncles=0 txs=1 gas=208967 fees=2.08967e-13 elapsed=25.137ms
INFO [03-31|19:47:26.261] Successfully sealed new block          number=2 sealhash="d185
98...436291" hash="32bf4d...6da027" elapsed=10.191ms
INFO [03-31|19:47:26.262] ⚡ mined potential block                number=2 hash="32bf4d...
6da027"
INFO [03-31|19:47:26.271] Commit new mining work                number=3 sealhash="54aa
d6...056880" uncles=0 txs=0 gas=0 fees=0 elapsed=6.654ms
INFO [03-31|19:47:26.271] Sealing paused, waiting for transactions

```

可以通过 `eth.getTransaction` 查看交易信息

```

> eth.getTransaction("0xc143527d62fe31fdf6c6815dc05a86e3f11edfc958bbc6d0dbb0ef79ee1b6e76")
{
  blockHash: "0x32bf4ddb0e23e79622a2f12d5213aa02bd2928123b7975beb4960edf916da027",
  blockNumber: 2,
  from: "0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a",
  gas: 1000000,
  gasPrice: 1,
  hash: "0xc143527d62fe31fdf6c6815dc05a86e3f11edfc958bbc6d0dbb0ef79ee1b6e76",
  input: "0x608060405234801561001057600080fd5b5060405161053738038061053783398181016040528101
019061004892919061004f565b50506102f6565b82805461005b90610224565b90600052602060002090601f0160
82601f1061009657805160ff19168380011785556100c4565b828001600101855582156100c4579182015b82811
00a8565b5b5090506100d191906100d5565b5090565b5b808211156100ee5760008160009055506001016100d65
019b565b905082815260208101848480111561011d57600080fd5b6101288482856101f1565b5093925050505
6101518482602086016100f2565b91505092915050565b60006020828403121561016c57600080fd5b600082015
5b61019284828501610130565b91505092915050565b60006101a56101b6565b90506101b18282610256565b919
fffff8211156101d576101d56102b6565b5b6101e4826102e5565b9050602081019050019050565b60005b8

```

同样的, `simple.sol` 的合约部署也如下图所示:

```

> var SimpleTx = {from:eth.accounts[1],data:SimpleByteCode,gas:"1000000"}
undefined
> personal.unlockAccount(eth.accounts[1],"123")
true
> var simple = SimpleContract.new(SimpleTx)
INFO [03-31|20:09:40.703] Submitted contract creation            hash=0x6cbdbdf3de160b3156a3e5d9a1b0b02f28459291d9c06da27b4df8229b
cf9375 from=0x1fB6C7A8c9dd89C7D18C746DD494c7f2cD177F4a nonce=1 contract=0x68f251491ff7bad62538fe9e826f57393747A276 value=0
undefined
> INFO [03-31|20:09:40.719] Commit new mining work                number=3 sealhash="7b9cec...d01926" uncles=0 txs=1 gas=349492 fee
s=3.49492e-13 elapsed=14.526ms
INFO [03-31|20:09:40.728] Successfully sealed new block          number=3 sealhash="7b9cec...d01926" hash="5d3243...68860f" elapsed=9.
956ms
INFO [03-31|20:09:40.730] ⚡ mined potential block                number=3 hash="5d3243...68860f"
INFO [03-31|20:09:40.735] Commit new mining work                number=4 sealhash="2ed5f9...3a6872" uncles=0 txs=0 gas=0 fees=
0 elapsed=4.097ms
INFO [03-31|20:09:40.735] Sealing paused, waiting for transactions

> simple
{
  abi: [{
    inputs: [{...}],
    name: "add",
    outputs: [],
    stateMutability: "nonpayable",

```

调用 `simpleContract` 中的 `add(6)` 函数:

```

> simple.add(6)
INFO [03-31|20:21:43.535] Submitted transaction            hash=0x5b2062297d980cb906f479c240e3d8c06b9cef39222c2394831f1c7b3e
2ff045 from=0x1fB6C7A8c9dd89C7D18C746DD494c7f2cD177F4a nonce=9 recipient=0x68f251491ff7bad62538fe9e826f57393747A276 value=0
"0x5b2062297d980cb906f479c240e3d8c06b9cef39222c2394831f1c7b3e2ff045"
> INFO [03-31|20:21:43.537] Successfully sealed new block          number=9 sealhash="669254...b3e7ca" hash="cc4347...b41c11" elapsed=
1.344ms
INFO [03-31|20:21:43.537] ⚡ block reached canonical chain        number=2 hash="32bf4d...6da027"
INFO [03-31|20:21:43.537] ⚡ mined potential block                number=9 hash="cc4347...b41c11"
INFO [03-31|20:21:43.535] Commit new mining work                number=9 sealhash="669254...b3e7ca" uncles=0 txs=1 gas=26809 fees=
2.6809e-14 elapsed="355.107µs"
INFO [03-31|20:21:43.543] Commit new mining work                number=10 sealhash="de5aa9...1e9bd5" uncles=0 txs=0 gas=0 fees
=0 elapsed=5.916ms
INFO [03-31|20:21:43.543] Sealing paused, waiting for transactions

```


调用 simpleContract 中的 set(99)函数:

```
> eth.defaultAccount=eth.accounts[1]
"0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a"
> simple.set(99)
INFO [03-31|20:10:28.968] Submitted transaction      hash=0x2ddb291eea449469aa55ffc068d3d1ae98debe84af359c4cf3413e24cd19eccd from=0x1fb6c7a8c9dd89c7d18c746dd494c7f2cd177f4a nonce=2 recipient=0x68f251491ff7bad62538fe9e826f57393747A276 value=0
"0x2ddb291eea449469aa55ffc068d3d1ae98debe84af359c4cf3413e24cd19eccd"
> INFO [03-31|20:10:28.969] Commit new mining work      number=4 sealhash="e849c1...8f93b8" uncles=0 txs=1 gas=43724 fee
s=4.3724e-14 elapsed=1.674ms
INFO [03-31|20:10:28.970] Successfully sealed new block      number=4 sealhash="e849c1...8f93b8" hash="737c46...0ade8d" elapsed=1.
266ms
INFO [03-31|20:10:28.970] mined potential block      number=4 hash="737c46...0ade8d"
INFO [03-31|20:10:28.977] Commit new mining work      number=5 sealhash="558df8...567d3c" uncles=0 txs=0 gas=0 fees=
0 elapsed=5.924ms
INFO [03-31|20:10:28.980] Sealing paused, waiting for transactions
INFO [03-31|20:10:28.981] Commit new mining work      number=5 sealhash="558df8...567d3c" uncles=0 txs=0 gas=0 fees=
0 elapsed=9.294ms
```

第二部分：飞行棋游戏：以太坊多方参与合约

这部分将使用智能合约实现一个有趣的多方参与游戏——飞行棋，同时使用 Python 远程 rpc 调用该合约。

这一部分的主体包括 aeroplane_chess.py 和 aeroplane_chess.sol 函数。

- (1) 在 aeroplane_chess.py 中包含一系列的功能函数：分别为(1)导入包及相应配置(2)通过命令行获取游戏玩家及相应 id;(3)从文件中读取 abi 和 bytecode 并部署合约;(4)保存合约地址，如果文件及地址有效，就读取旧有合约，继续仍在进行的游戏(5)play 进行游戏函数，该游戏可中断后继续进行.以及最后一部分的程序入口:先初始化玩家名称和账户 id，然后设置默认账户，最后参与游戏。

- (2) 飞行棋游戏智能合约设计:因为是使用 Python 远程 rpc 调用合约的函数接口模块，所以可以从 python 的代码逻辑得到合约的接口设计。

- (3) 在此次实验飞行棋的智能合约接口设计中，很好的印证了我们所学习的 solidity 编程的相关知识。下面是在我上一周的飞行棋实验中所写的代码，代码逻辑非常的简单，每一次玩家都向下移动一格，向下移动一格确保一定可以到达终点。
当到达终点时，直接判断结果即可。

```
function play(string memory _str) public{
    require(isPlay[msg.sender]);
    require(!gameover);
    uint step=1;
    uint id =playerId[msg.sender];
    players[id].position+=step;
    if(players[id].position ==destination){
        gameover=true;
        winnerId=id;
    }
}
```

上面代码的运行结果如下:

```
cxn playing, input: f33a0ce2-2768-4a53-b739-247105d26e60
round: 0      random dice: 0  step: 19 -> 20
-----game board-----
0 :
1 :
2 :
3 :
4 :
5 :
6 :
7 :
8 :
9 :
10 :
11 :
12 :
13 :
14 :
15 :
16 :
17 :
18 :
19 :
20 : [ cxn V0 r0 ]
-----game board-----
gameover, the winner is ('0x051c83ACC2d53523053b6900E1bBf470B928f91E', 'cxn', 20, 1, 0, 0)
```

在这一周的实验中，参考了助教学长的代码，实现了一种更为复杂的飞行棋 play 机制：在这个逻辑里，定义了飞行棋游戏的方向等。改动了部分代码，可以实现步长取模的结果：

```
while (true) {
    if (newPosition < 0) {
        newPosition = -newPosition;
        players[id].direction = 1;
    } else if (uint(newPosition) > destination) {
        // newPosition = int(destination) * 2 - newPosition;
        newPosition = newPosition % (int)(destination);
        players[id].direction = -1;
    } else {
        break;
    }
}
```

```
abc playing, input: 3535a3cc-cbe7-423f-8d42-03048ae561e0
round: 5          random dice: 2  step: 18 -> 20
-----game board-----
0 :
1 :
2 :
3 :
4 :
5 :
6 :
7 :
8 :
9 :
10 :
11 :
12 :
13 :
14 :
15 :
16 :
17 :
18 :
19 :
20 : [ abc V2 r5 ]
-----game board-----
gameo 截屏 he winner is ('0xa17330AB4986e38EaE4451c1DFaBe3F9985bb543', 'abc', 20, 1, 2, 5)
```

五、总结

关于智能合约编写的理解:

Reference:(<http://www.gjermundbjaanes.com/understanding-ethereum-smart-contracts/>)

结合三周的上机课和实验, 对智能合约有了如下理解:

- 智能合约只是在区块链上运行的一段代码。
- 区块链就是分布式账本, 由 a list of transactions 组成
- 比特币的激励机制保证了不需要参与双方的互相可信。

像 Reference 里面说的那样:


This distribution coupled with strong monetary incentives removes the need for trust between parties.

这种激励机制保证了, **cheat**(破坏信任)带来的结果还没有 **stay within rules** 高。

下面是智能合约调用的示例:

类型	TO	DATA	FROM	AMOUNT
转账	接受人地址	Any message	You	以太币
创建智能合约账户	Null	智能合约代码	You	你想给你账户初始化的以太币数目
调用智能合约账户中的函数	智能合约合约账户地址	函数名和调用参数	You	0, 或你想让合约为你做什么服务

下图展示了智能合约调用的流程:

 Address: 0x0123456.....



1 Convert function call to HEX

myFunction(parameters) →  → 0xabcdef0123456789.....

2 Put the Information into Transaction object

```
{
  "to": "0x0123456.....",
  "value": 0, // No need to send money here
  "data": "0xabcdef0123456789....."
}
```

3 Sign the Transaction with your Private key

{ ... } +  Private Key →  → 0xfedcba9876....

Signed Transaction
Can only be decrypted with YOUR public key
Only you can have sent this transaction

4 Send the transaction to the Ethereum Network



通过从以太坊部署到智能合约调用，对 solidity 编程有了初步感性的认识。

在飞行棋课上，跟同学组队尝试通过更改 transaction 的地址进行联机的智能合约调用，但无奈因为虚拟机端口的问题，导致没有办法连上，否则可以直接成功执行，就可以更加加深了以太坊就被全世界维护的虚拟机一样的这一步认识。