1. **代码源码**

github地址：<https://github.com/MapleLai/Block-Chain-Project-Contribution/tree/master>

**二、选题背景**

利用智能合约匿名性和不可篡改和匿名的特点，搭建一个去中心化的募捐平台。智能合约的部署者可以发起一次募捐活动，区块链上的账号可以进行捐款。智能合约会公示目前筹集到的捐款，以及最大的捐款数（以账号为单位，多次捐款会累积）。

**三、使用说明**

1. 开发环境

操作系统：Windows 10

Node.js v11.6.6

Webpack v4.28.2

Truffle v0.5.0

1. 运行步骤

本地开发我使用了Truffle这个框架来进行前端的开发，Ethereum客户端则是选择了Ganache，因为在Ganache创建新链时自动给我们创建了10个账号，而且调用智能合约时不需要进行挖矿，与Truffle配合使用进行开发感觉真的良好。Truffle的使用可以参考下面这个链接，挺靠谱的。

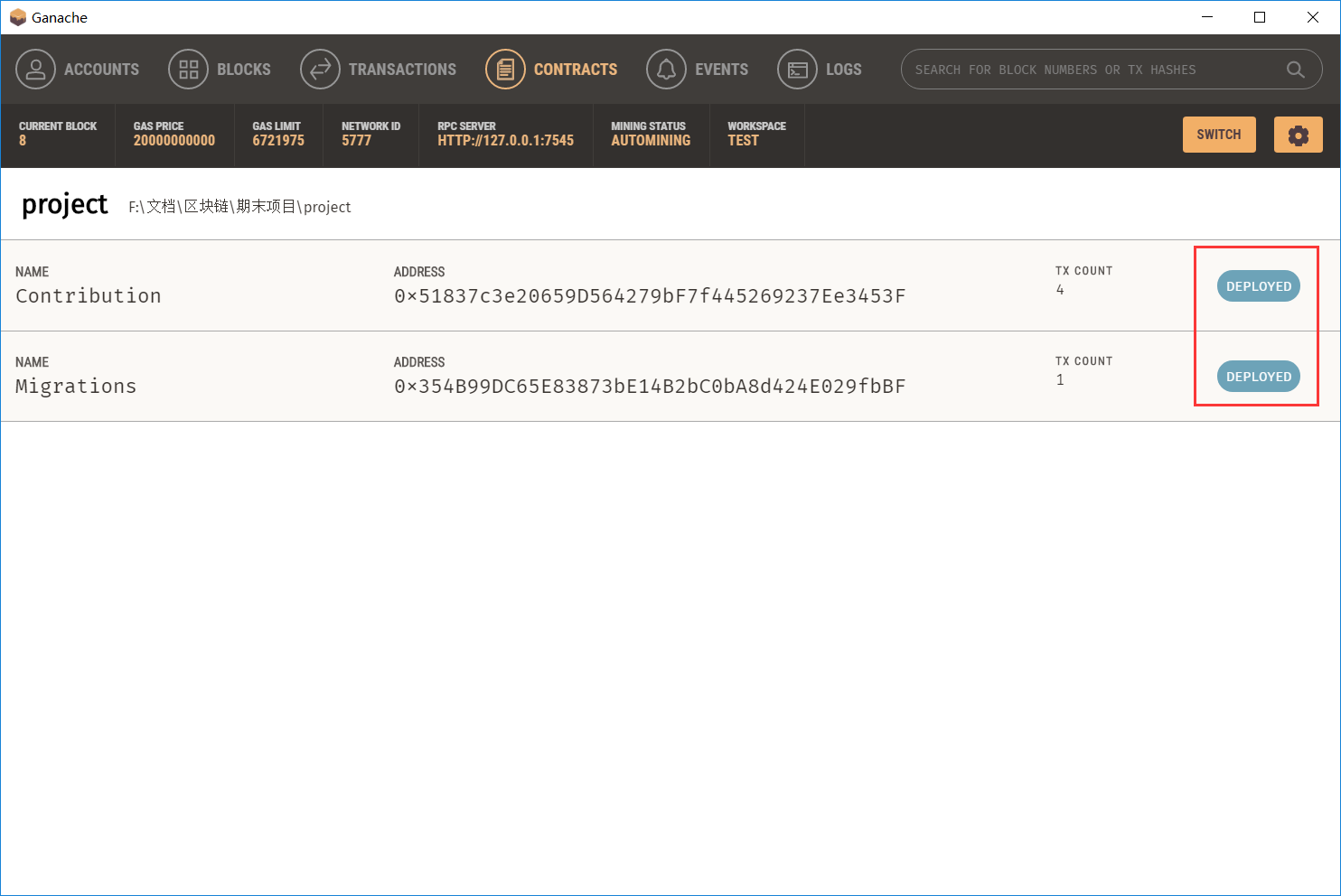
<https://baijiahao.baidu.com/s?id=1597236535233062436&wfr=spider&for=pc>

简单地总结一下步骤大概就是：

①用truffle unbox webpack命令创建新项目，这个过程可能要花比较长的时间，主要是有些模块的下载源在墙外，速度就很慢，如果条件允许的话请挂梯子。

②用truffle compile命令编译智能合约，在编译之前最后打开Ganache，方便查看现在的合约情况。

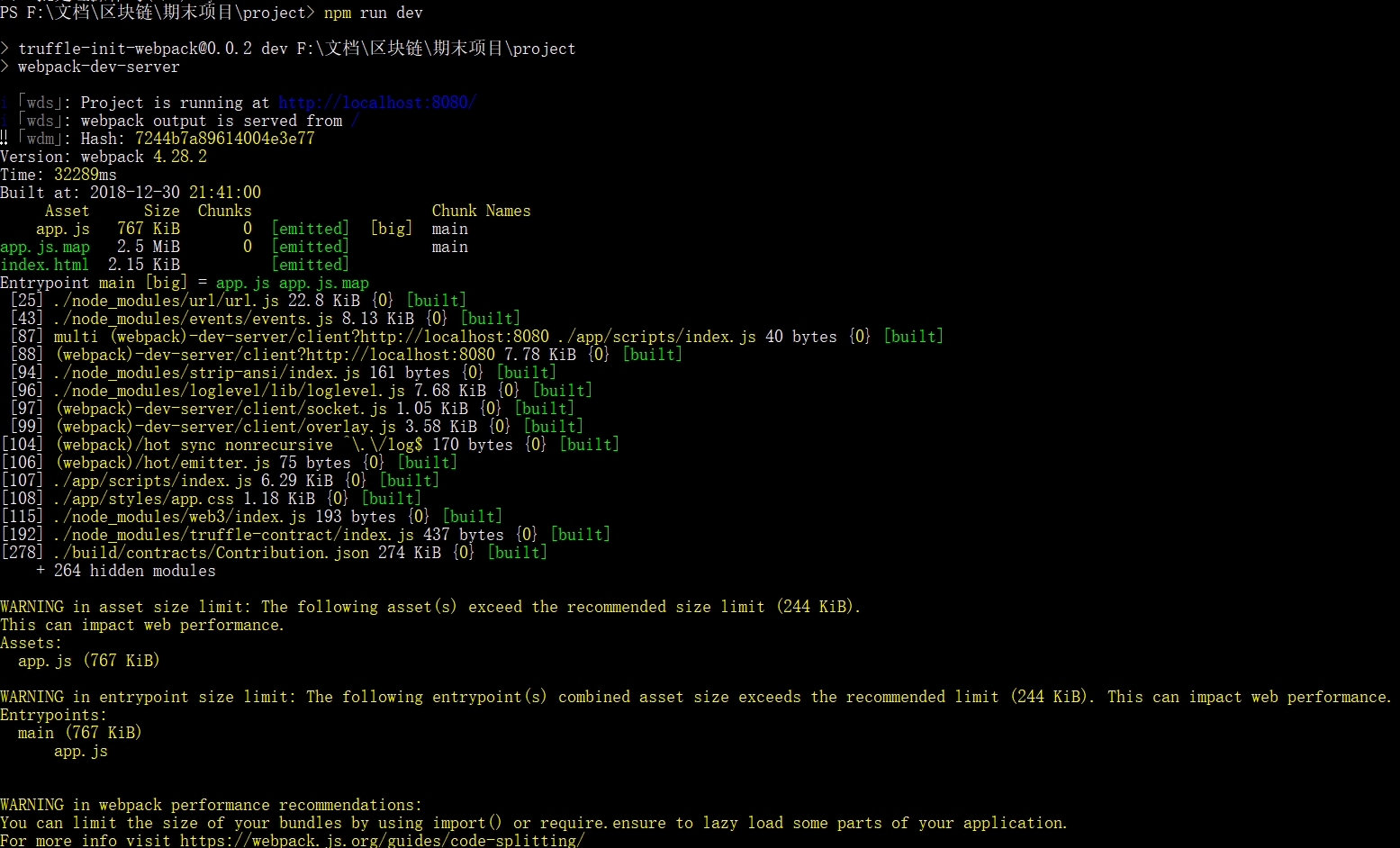
③用truffle migrate命令部署智能合约，可以在Ganache看到部署成功的信息。



④用npm run dev命令运行项目，然后这时候你就会发现系统不停地提醒你缺失了哪个文件......原因就是创建项目时由于墙的原因导致某些模块下载超时，所以这些模块就缺失了，需要手动把缺失的模块安装，建议使用淘宝镜像源：

**npm --registry https://registry.npm.taobao.org install [模块名] -s**

参数-s是把模块安装到项目，如果想要全局安装，改成-g即可。成功运行项目后的窗口如下所示：



⑤成功运行项目后，在浏览器中进入localhost:8080端口就能看到操作页面啦。用户地址必须正确填写，而且用户必须是存在的，否则在console里会有报错提示。通过单选按钮来进行设置目标金额和进行捐款两个操作的转换，使用下方的椭圆按钮输入金额，点击屏幕右方的方形按钮就可以调用智能合约了。





1. 其他注意事项

项目的使用就如之前所说的步骤进行就能成功啦，但智能合约从编译到运行会遇到好多好多的问题，上面提到的模块缺失问题是最麻烦的，下面说一些注意事项，避免遇到坑。

①Windows下输入命令时使用power-shell，不要用cmd，cmd不能执行truffle compile命令。

②github上的项目源码中的node\_modules文件夹是空的，这个文件夹就是存放各个模块的地方，但因为实在是太大了（我的有3GB），我就把所有模块全删了。可以用truffle unbox webpack创建一个新项目，然后用新项目里的node\_modules把空的node\_modules替换掉，编译运行时可能还是会缺了某几个模块，用淘宝镜像源安装即可。

③有个神奇的模块叫node-sass，它与你的Node.js有关系，有时报错说找不到这个模块并不是因为node-sass不存在，而是因为它里面的Node.js版本与你电脑的Node.js版本不一致。解决办法是先用npm uninstall node-sass -s卸载掉原有的node-sass（如果是全局安装就把-s改成-g），再用cnpm install node-sass -s安装新的（全局安装-s改成-g）。

④修改了智能合约后，或者是下载了别人的项目想要在自己的电脑运行时，都需要重新编译部署，记得先把build/contracts/里原有的json文件先删掉。这个巨坑，一开始我以为会自动覆盖掉，找了好久bug才发现是因为编译时如果已经存在同名文件，会直接跳过编译，真的是气死我了。

**四、代码说明**

1. 智能合约代码

1. pragma solidity ^0.5.0;
2. contract Contribution {
4. struct Person{
5. //记录账户的累积捐款金额
6. uint money;
7. }
9. struct Fundraising{
10. //参与募捐的人数，同一个人捐款了两次算作有两个人参与了募捐
11. uint participants;
12. //目前筹集到的捐款
13. uint funds;
14. //募捐的目标金额
15. uint target;
16. }
18. //募捐发起人
19. address fundraiser;
20. //为每个可能的地址存储一个"Person"
21. mapping(address =**>** Person) people;
22. //记录这次募捐信息的一个变量
23. Fundraising public fundraisings;
24. //目前最大的捐款数
25. uint \_max;
26. //最近一次捐款的数目
27. uint public pay;
29. constructor() public {
30. //把智能合约的部署者设置为募捐发起者
31. fundraiser = msg.sender;
32. fundraisings.participants = 0;
33. fundraisings.funds = 0;
34. fundraisings.target = 0;
35. people[fundraiser].money = 0;
36. }
38. //设置目标金额
39. function setTarget(uint \_target) public {
40. require(\_target **>** 0);
41. //只有募捐发起者可以设置目标金额
42. if (msg.sender == fundraiser){
43. fundraisings.target = \_target;
44. }
45. }
47. //进行捐款
48. function contribute(uint \_money) public {
49. //如果此次募捐还没达到目标金额，捐款才能成功
50. if (fundraisings.funds **<** **fundraisings.target**){
51. //如果募捐离目标所需要的金额小于此次捐款数，则只收取需要的金额，退还多余捐款
52. if (fundraisings.target - fundraisings.funds **<** **\_money**) {
53. fundraisingsfundraisings.participants = fundraisings.participants + 1;
54. pay = fundraisings.target - fundraisings.funds;
55. fundraisingsfundraisings.funds = fundraisings.funds + pay;
56. people[msg.sender].money = people[msg.sender].money + pay;
57. }
58. //成功捐款，募捐的筹集金额增加对应数值
59. else {
60. fundraisingsfundraisings.participants = fundraisings.participants + 1;
61. pay = \_money;
62. fundraisingsfundraisings.funds = fundraisings.funds + \_money;
63. people[msg.sender].money = people[msg.sender].money + \_money;
64. }
65. //统计最大捐款数
66. if (people[msg.sender].money **>** \_max){
67. \_max = people[msg.sender].money;
68. }
69. }
70. }
72. //公示最大捐款数
73. function maxContribution() public view returns (uint max) {
74. return \_max;
75. }
77. //获取目标金额
78. function getTarget() public view returns (uint target) {
79. return fundraisings.target;
80. }
82. //获取已筹集资金
83. function getFunds() public view returns (uint funds) {
84. return fundraisings.funds;
85. }
87. //获取参与人次
88. function getParticipants() public view returns (uint participants) {
89. return fundraisings.participants;
90. }
92. //获取最近一次捐款的数目
93. function getPay() public view returns (uint \_pay) {
94. return pay;
95. }
96. }
97. HTML代码

HTML代码中加载css和js代码的方法与平时网页开发时的方式不太一样，所以特别说明一下。

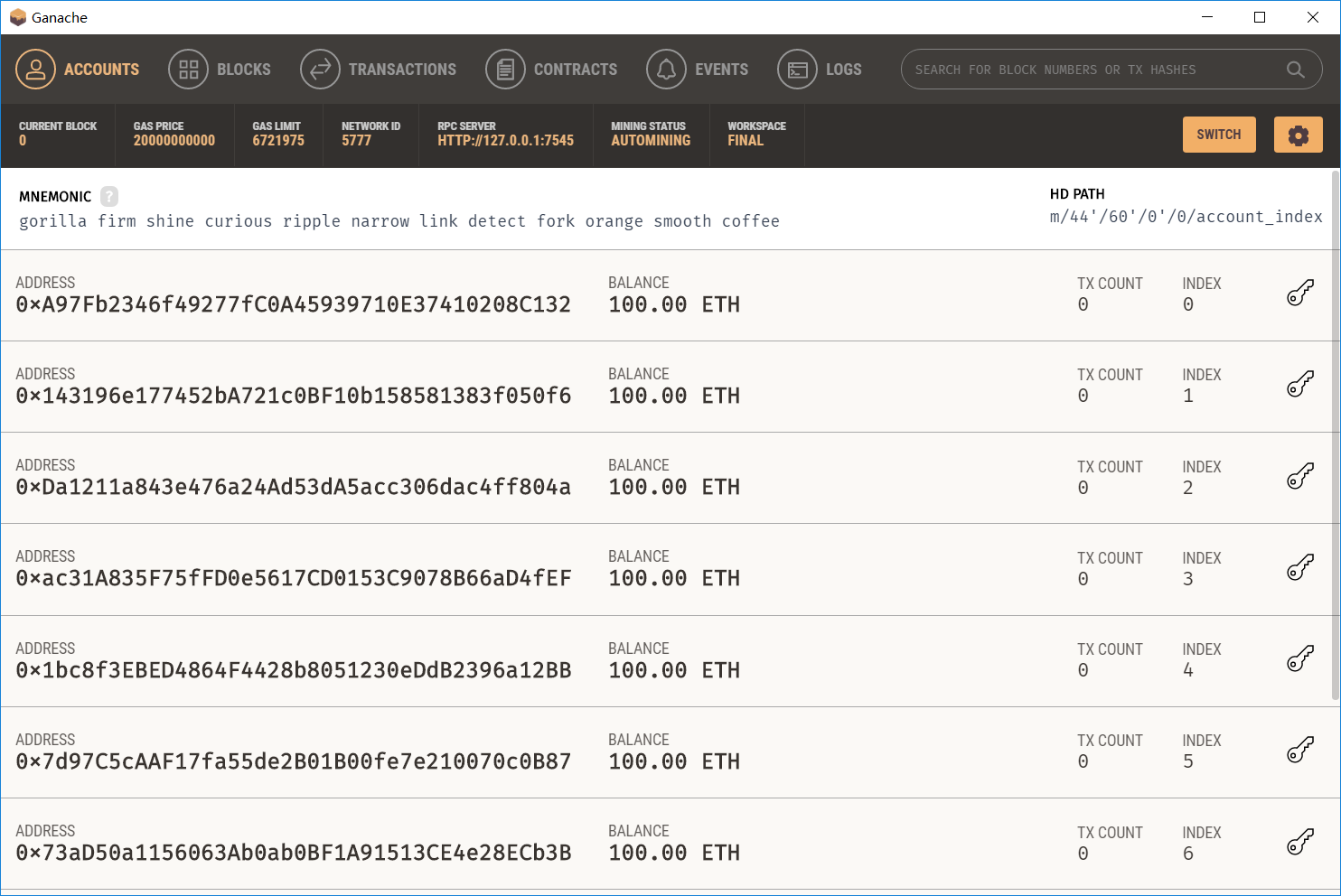
1. <!DOCTYPE html**>**
2. **<html>**
4. **<head>**
5. **<meta** charset="UTF-8"**>**
6. **<title>**Contribution**</title>**
7. **<link** href='https://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'**>**
8. **<script** src="./app.js"**></script>**
9. **</head>**
11. **<body>**
12. ...
13. **</body>**
15. **</html>**
16. Javascript代码

网页的脚本代码主要是为了与智能合约建立联系，使用API来调用智能合约的函数，实现交易和获取变量值更新UI信息等操作，因此这里只对一些与调用智能合约相关的代码进行说明，完整的代码请查看github。

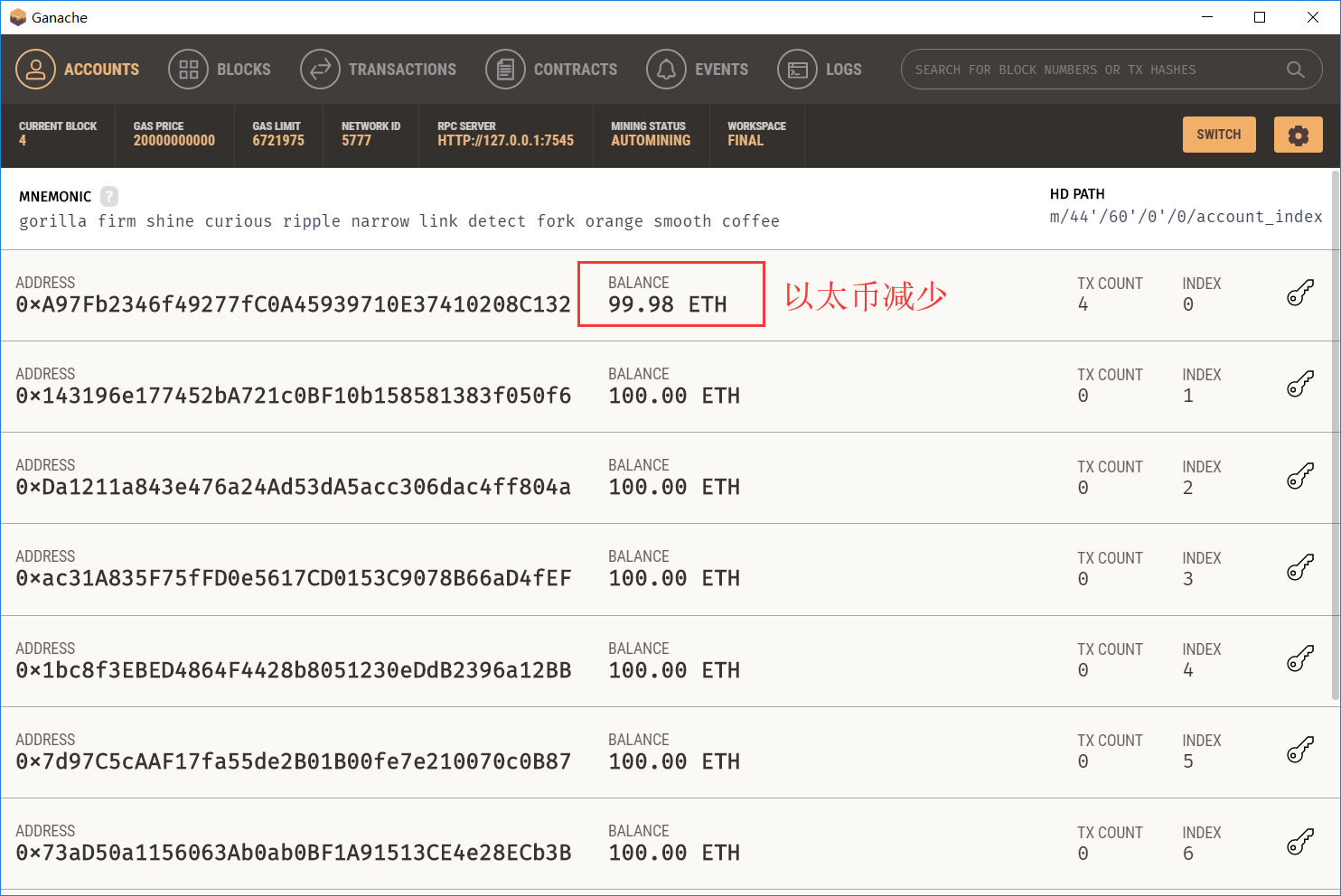
1. // 一些import
2. ...
4. // 获取合约
5. const Contribution = contract(contributionArtifact)
7. let accounts
8. let account
10. const App = {
11. //App初始化
12. start: function () {
13. const self = this
15. // Bootstrap the Contribution abstraction for Use.
16. Contribution.setProvider(web3.currentProvider)
18. // Get the initial account balance so it can be displayed.
19. web3.eth.getAccounts(function (err, accs) {
20. if (err != null) {
21. alert('There was an error fetching your accounts.')
22. return
23. }
24. if (accs.length === 0) {
25. alert("Couldn't get any accounts! Make sure your Ethereum client is configured correctly.")
26. return
27. }
28. accounts = accs
29. account = accounts[0]
30. self.refreshMsg()
31. })
32. },
34. //更新募捐信息
35. refreshMsg: function () {
36. const self = this
38. let meta
39. Contribution.deployed().then(function (instance) {
40. meta = instance
41. return meta.getTarget.call({ from: account })
42. }).then(function (value) {
43. const target = document.getElementById('target')
44. target.innerHTML = "目标金额：" + value.toString()
45. return meta.getFunds.call({ from: account })
46. }).then(function (value) {
47. const funds = document.getElementById('funds')
48. funds.innerHTML = "已筹集数：" + value.toString()
49. return meta.getParticipants.call({ from: account })
50. }).then(function (value) {
51. const participants = document.getElementById('participants')
52. participants.innerHTML = "参与人次：" + value.toString()
53. return meta.maxContribution.call({ from: account })
54. }).then(function (value) {
55. const max = document.getElementById('max')
56. max.innerHTML = "最大捐款：" + value.toString()
57. }).catch(function (e) {
58. console.log(e)
59. console.log('Error getting Message; see log.')
60. })
61. },
63. //判断用户希望捐款还是设置目标金额，并调用相应的智能合约函数
64. send: function () {
65. const self = this
67. const useraddress = document.getElementById('\_useraddress').value
68. const money = parseInt(document.getElementById('money').value)
69. const \_money = document.getElementById('money')
70. const msg = document.getElementById('msg')
71. const op1 = document.getElementById('op1')
73. let meta
74. Contribution.deployed().then(function (instance) {
75. meta = instance
76. if (op1.checked){
77. return meta.contribute(money, { from: useraddress, gas: 4700000 })
78. } else {
79. return meta.setTarget(money, { from: useraddress })
80. }
81. }).then(function () {
82. if (op1.checked){
83. return meta.getPay.call({ from: account })
84. } else {
85. if (useraddress.toUpperCase() != account.toUpperCase()) {
86. msg.innerHTML = "你没有权限进行此项操作"
87. } else {
88. msg.innerHTML = "你已成功设置目标金额为" + money.toString()
89. }
90. }
91. }).then(function (value) {
92. if (op1.checked){
93. msg.innerHTML = "你已成功捐出" + value.toString()
94. }
95. msg.style.visibility = "visible"
96. }).catch(function (e) {
97. console.log(e)
98. console.log('Error sending; see log.')
99. })
101. self.refreshMsg()
102. },
104. //其他与智能合约无关的函数
105. ...
106. }
108. //设置窗口监听器，并初始化各个变量
109. window.App = App
110. window.addEventListener('load', function () {
111. // 检查是否安装了web3，有无使用prprovider插件，进行相应的报错
112. ...
114. App.start()
115. })

**五、测试**

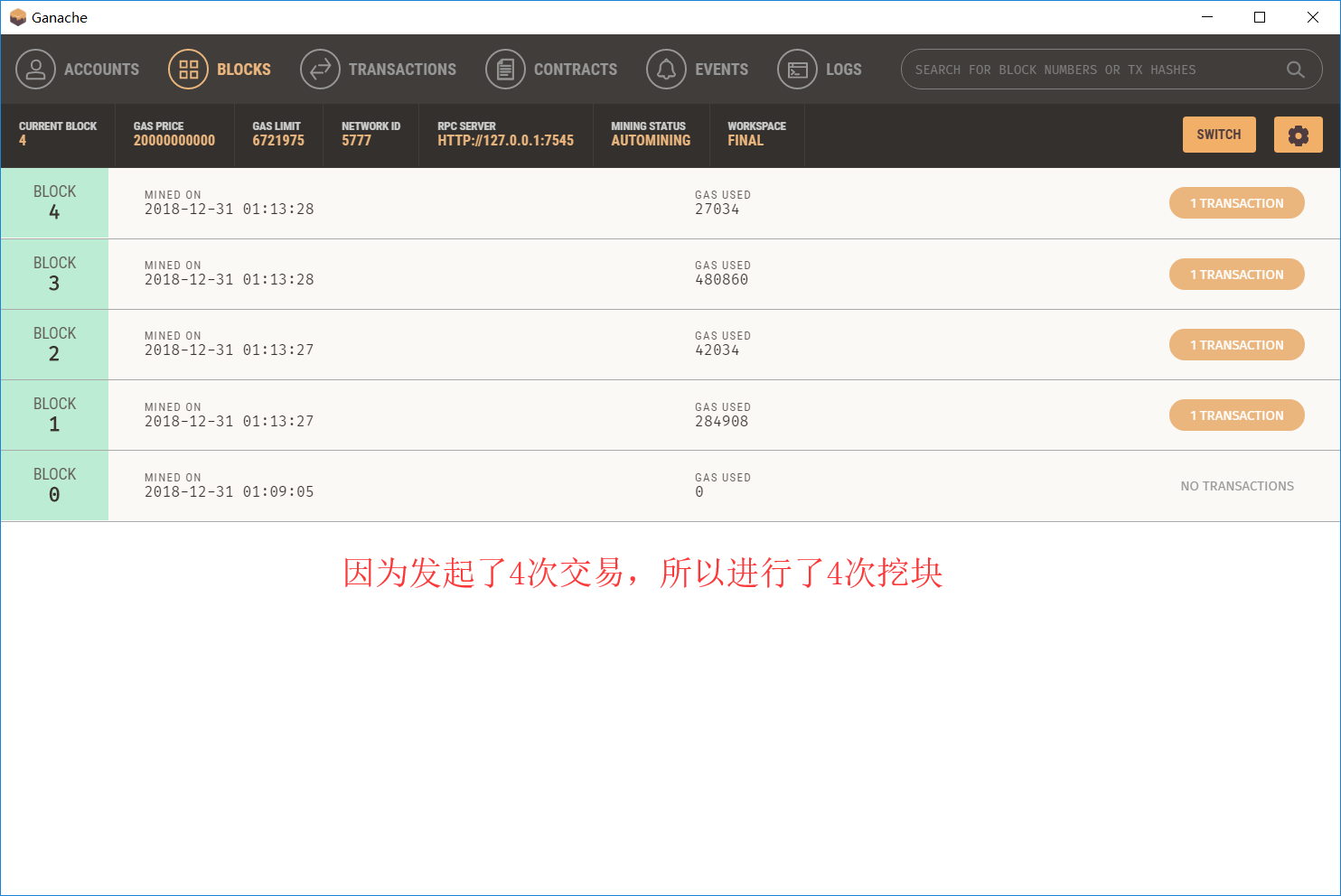
1. 用Ganache开启测试链，有10个账户，每个账户都有100个以太币。



1. 第1个账户消耗以太币来进行合约的部署，要部署的智能合约有2个，所以发起了总计4次的交易和挖块。

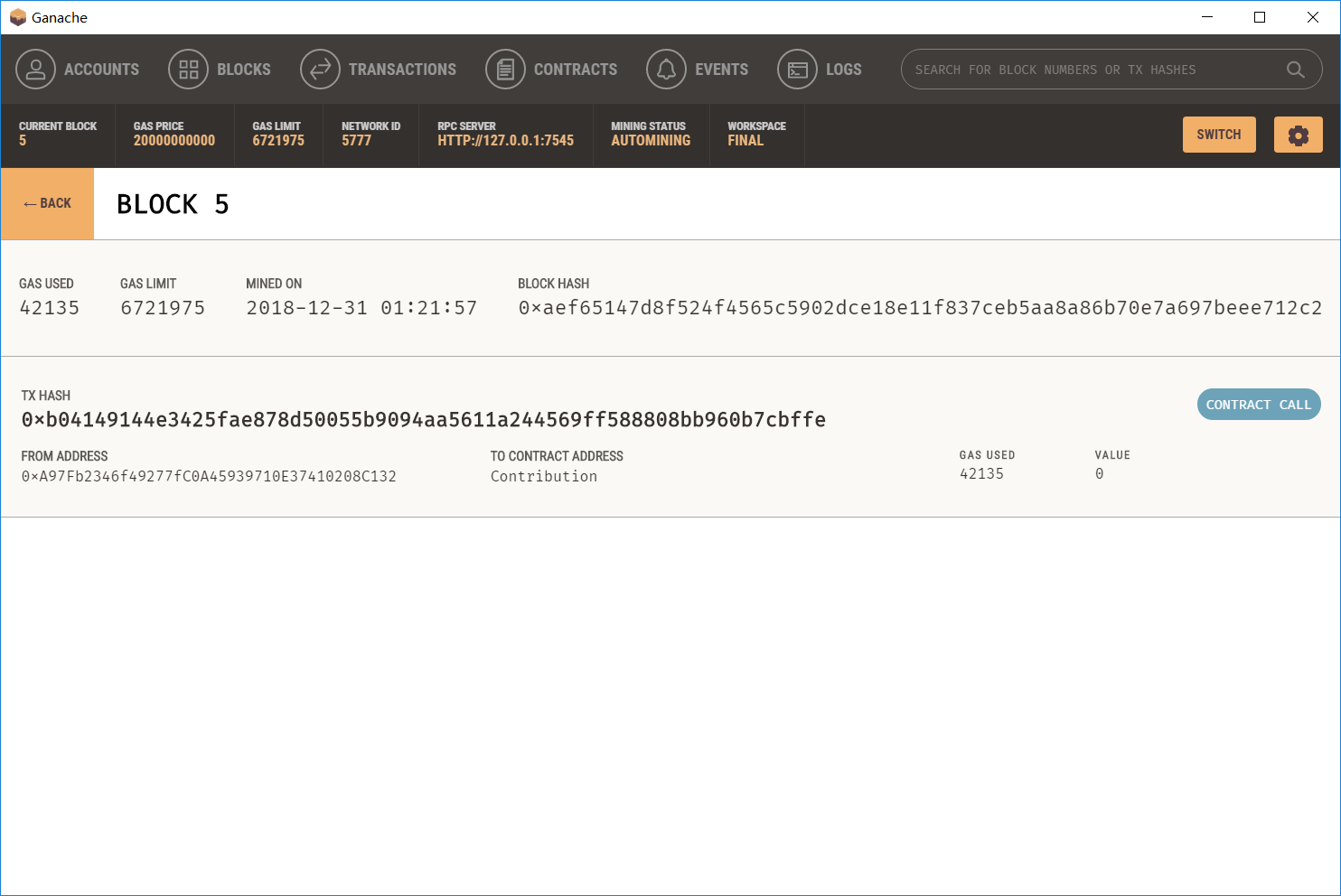






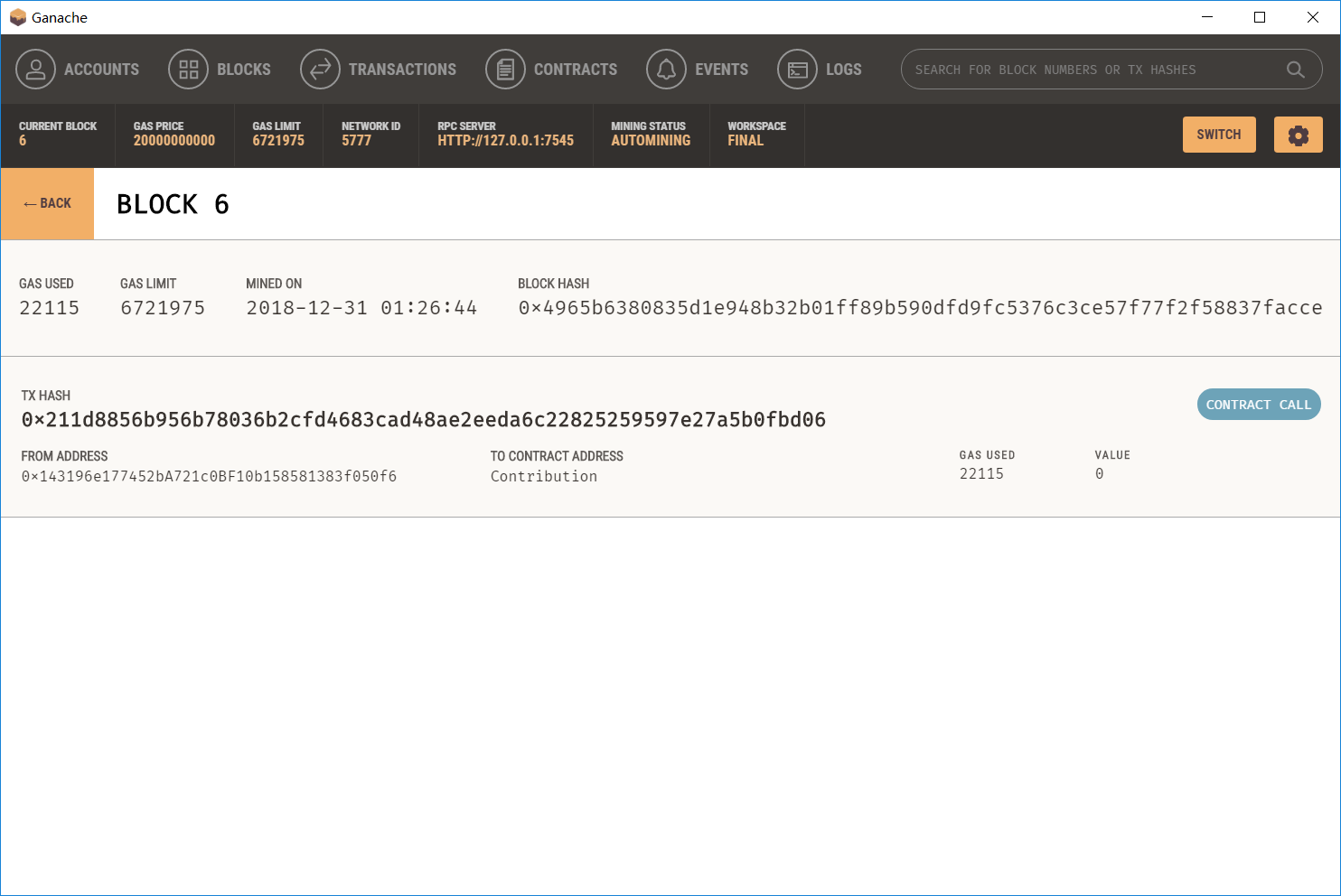
1. 运行项目，尝试用一个账户（智能合约部署者）设置目标金额位100，设置成功，查看Ganache能看到新的区块5。





1. 尝试用第二个用户（非智能合约部署者）设置目标金额为200，设置失败，但因为调用了智能合约，所以还是产生了新区块6。





1. 由于篇幅原因，接下来只展示网页的效果图。第二个用户捐出30，捐款成功，募捐信息进行更新（已筹集数+30，参与人次+1，最大捐款变为30）。



1. 第三个用户捐出100，但由于筹款距离目标金额只相差70，因此只捐出了70，募捐信息进行更新（已筹集数+70，参与人次+1，最大捐款变为70）



1. 第一个用户重新设置目标金额为200，此时以募集的筹款并不会清空，而是算入新一轮的募捐。



1. 第二个用户再次捐出100，捐款成功，募捐信息进行更新（已筹集数+100，参与人次+1，由于第二个用户之前已经捐出30，所以最大捐款变为130）



1. 至此，简单募捐系统的所有功能都测试了一遍，完全按照预期的效果实现了对应的功能。查看Ganache窗口来检查用户和区块信息。

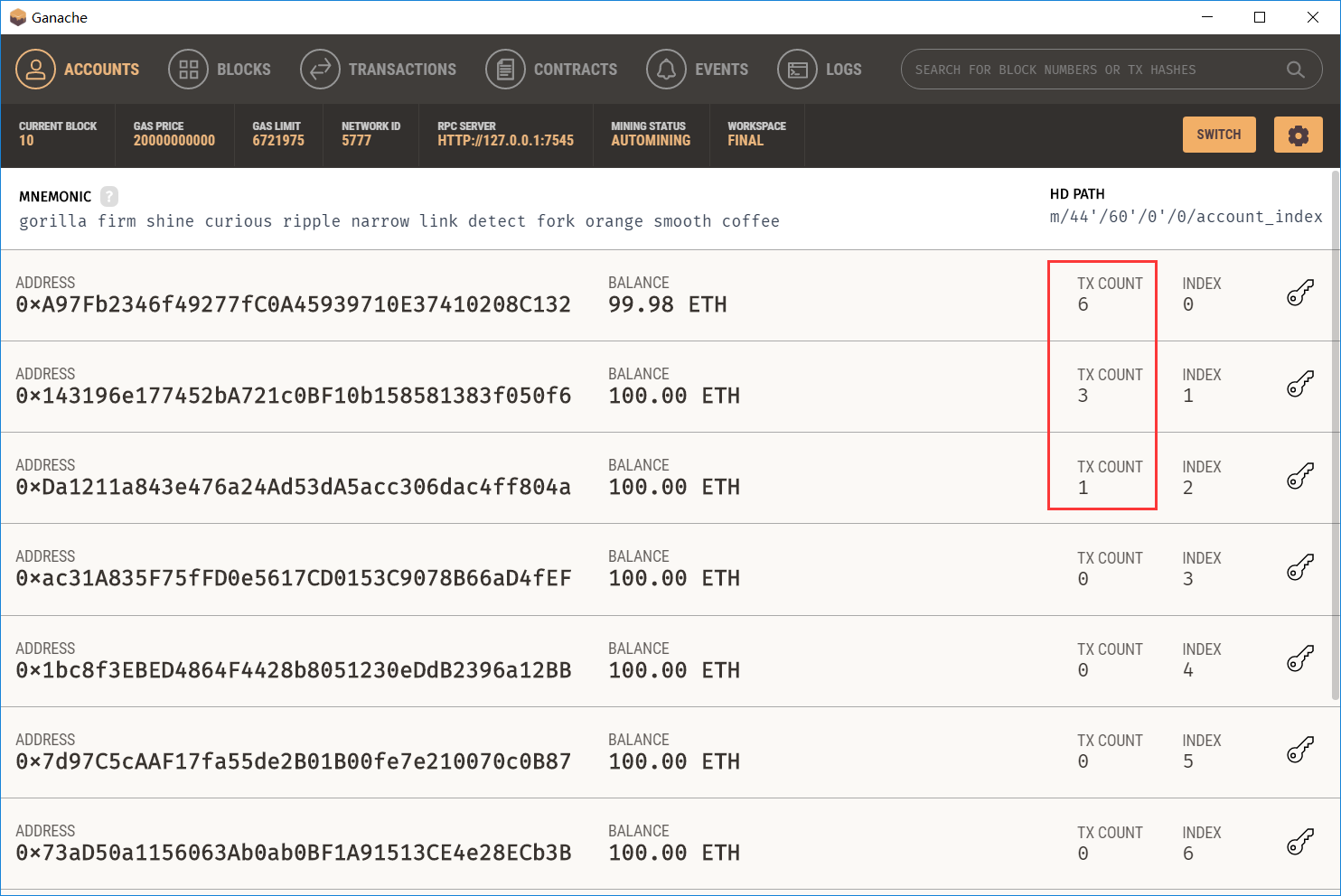
①用户信息

在刚才的测试中一共进行了10次智能合约调用，而对应的用户信息中的相关区块数目是正确的。

第一个用户6个区块（部署智能合约产生4个区块，2次设置目标金额产生2个区块）

第二个用户3个区块（1次设置目标金额产生1个区块，2次捐款产生2个区块）

第三个用户1个区块（1次捐款产生1个区块）



②区块信息

查看区块信息可以看到10个区块，与调用智能合约的次数相同，证明在网页进行操作时正确调用了智能合约，并且每次调用都成功发起交易和挖块，代码逻辑运行正确，测试成功！

