

分账与空头

一、分账

分账就是按照一定比例分钱。在现实中，经常会有“分赃不均”的事情发生；而在区块链的世界里，`Code is Law`，我们可以事先把每个人应分的比例写在智能合约中，获得收入后，再由智能合约来进行分账。



分账合约

分账合约(PaymentSplit)具有以下几个特点：

1. 在创建合约时定好分账受益人 `payees` 和每人的份额 `shares`。
2. 份额可以是相等，也可以是其他任意比例。
3. 在该合约收到的所有 `ETH` 中，每个受益人将能够提取与其分配的份额成比例的金額。
4. 分账合约遵循 `Pull Payment` 模式，付款不会自动转入账户，而是保存在此合约中。受益人通过调用 `release()` 函数触发实际转账。

- `// SPDX-License-Identifier: MIT`
`pragma solidity ^0.8.4;`

`/**`

分账合约

@dev 这个合约会把收到的ETH按事先定好的份额分给几个账户。收到ETH会存在分账合约中，需要每个受益人调用release()函数来领取。

`*/`

`contract PaymentSplit{`

事件

分账合约中共有 3 个事件：

- `PayeeAdded`：增加受益人事件。
- `PaymentReleased`：受益人提款事件。
- `PaymentReceived`：分账合约收款事件。

```
1 // 事件
2 event PayeeAdded(address account, uint256 shares); // 增加受益人事件
3 event PaymentReleased(address to, uint256 amount); // 受益人提款事件
4 event PaymentReceived(address from, uint256 amount); // 合约收款事件
```

状态变量

分账合约中共有 5 个状态变量，用来记录受益地址、份额、支付出去的 ETH 等变量：

- `totalShares`：总份额，为 `shares` 的和。
- `totalReleased`：从分账合约向受益人支付出去的 ETH，为 `released` 的和。
- `payees`：`address` 数组，记录受益人地址
- `shares`：`address` 到 `uint256` 的映射，记录每个受益人的份额。
- `released`：`address` 到 `uint256` 的映射，记录分账合约支付给每个受益人的金额。

```
1 uint256 public totalShares; // 总份额
2 uint256 public totalReleased; // 总支付
3
4 mapping(address => uint256) public shares; // 每个受益人的份额
```

```

5     mapping(address => uint256) public released; // 支付给每个受益人的金额
6     address[] public payees; // 受益人数组

```

函数

分账合约中共有 6 个函数：

- 构造函数：始化受益人数组 `_payees` 和分账份额数组 `_shares`，其中数组长度不能为0，两个数组长度要相等。`_shares`中元素要大于0，`_payees`中地址不能为0地址且不能有重复地址。
- `receive()`：回调函数，在分账合约收到 `ETH` 时释放 `PaymentReceived` 事件。
- `release()`：分账函数，为有效受益人地址 `_account` 分配相应的 `ETH`。任何人都可以触发这个函数，但 `ETH` 会转给受益人地址 `account`。调用了`releasable()`函数。
- `releasable()`：计算一个受益人地址应领取的 `ETH`。调用了 `pendingPayment()` 函数。
- `pendingPayment()`：根据受益人地址 `_account`，分账合约总收入 `_totalReceived` 和该地址已领取的钱 `_alreadyReleased`，计算该受益人现在应分的 `ETH`。
- `_addPayee()`：新增受益人函数及其份额函数。在合约初始化的时候被调用，之后不能修改。

```

1     /**
2      * @dev 初始化受益人数组_payees和分账份额数组_shares
3      * 数组长度不能为0，两个数组长度要相等。_shares中元素要大于0，_payees中地址不能为0
      地址且不能有重复地址
4      */
5     constructor(address[] memory _payees, uint256[] memory _shares) payable {
6         // 检查_payees和_shares数组长度相同，且不为0
7         require(_payees.length == _shares.length, "PaymentSplitter: payees and
      shares length mismatch");
8         require(_payees.length > 0, "PaymentSplitter: no payees");
9         // 调用_addPayee，更新受益人地址payees、受益人份额shares和总份额totalShares
10        for (uint256 i = 0; i < _payees.length; i++) {
11            _addPayee(_payees[i], _shares[i]);
12        }
13    }
14
15    /**
16     * @dev 回调函数，收到ETH释放PaymentReceived事件
17     */
18    receive() external payable virtual {
19        emit PaymentReceived(msg.sender, msg.value);
20    }
21

```

```

22  /**
23  * @dev 为有效受益人地址_account分帐，相应的ETH直接发送到受益人地址。任何人都可以触
    发这个函数，但钱会打给account地址。
24  * 调用了releasable()函数。
25  */
26  function release(address payable _account) public virtual {
27      // account必须是有效受益人
28      require(shares[_account] > 0, "PaymentSplitter: account has no shares");
29      // 计算account应得的eth
30      uint256 payment = releasable(_account);
31      // 应得的eth不能为0
32      require(payment != 0, "PaymentSplitter: account is not due payment");
33      // 更新总支付totalReleased和支付给每个受益人的金额released
34      totalReleased += payment;
35      released[_account] += payment;
36      // 转账
37      _account.transfer(payment);
38      emit PaymentReleased(_account, payment);
39  }
40
41  /**
42  * @dev 计算一个账户能够领取的eth。
43  * 调用了pendingPayment()函数。
44  */
45  function releasable(address _account) public view returns (uint256) {
46      // 计算分账合约总收入totalReceived
47      uint256 totalReceived = address(this).balance + totalReleased;
48      // 调用_pendingPayment计算account应得的ETH
49      return pendingPayment(_account, totalReceived, released[_account]);
50  }
51
52  /**
53  * @dev 根据受益人地址_account，分账合约总收入_totalReceived和该地址已领取的钱
    _alreadyReleased，计算该受益人现在应分的ETH。
54  */
55  function pendingPayment(
56      address _account,
57      uint256 _totalReceived,
58      uint256 _alreadyReleased
59  ) public view returns (uint256) {
60      // account应得的ETH = 总应得ETH - 已领到的ETH
61      return (_totalReceived * shares[_account]) / totalShares -
        _alreadyReleased;
62  }
63
64  /**

```

```
65      * @dev 新增受益人_account以及对应的份额_accountShares。只能在构造器中被调用，不能
      修改。
66      */
67      function _addPayee(address _account, uint256 _accountShares) private {
68          // 检查_account不为0地址
69          require(_account != address(0), "PaymentSplitter: account is the zero
            address");
70          // 检查_accountShares不为0
71          require(_accountShares > 0, "PaymentSplitter: shares are 0");
72          // 检查_account不重复
73          require(shares[_account] == 0, "PaymentSplitter: account already has
            shares");
74          // 更新payees, shares和totalShares
75          payees.push(_account);
76          shares[_account] = _accountShares;
77          totalShares += _accountShares;
78          // 释放增加受益人事件
79          emit PayeeAdded(_account, _accountShares);
80      }
```

Remix 演示

5. 部署 PaymentSplit 分账合约，并转入 1 ETH

在构造函数中，输入两个受益人地址，份额为 1 和 3。

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (London)

ACCOUNT: 0x5B3...eddC4 [98.249999%]

GAS LIMIT: 3000000

VALUE: 0 Ether

CONTRACT: PaymentSplit - 42_PaymentSplit/Pay

DEPLOY: [0x5B38D6a701c568545] [1,3] **transact**

Transactions recorded: 4

Deployed Contracts: PAYMENTSPLIT AT 0xD7A...F77

release address_account

payees uint256

pendingPay... address_account uint

releasable address_account uint

released address

shares address

totalReleased

PaymentSplit.sol

```

1 // 分拆合约
2 // @dev 这个合约会把收到的ETH按照事先定好的份额分给几个账户。收到ETH会存在分拆合约中，需要每个受益人调用release()函数来领取。
3
4 contract PaymentSplit{
5     // 事件
6     event PayeeAdded(address account, uint256 shares); // 增加受益人事件
7     event PaymentReleased(address to, uint256 amount); // 受益人提款事件
8     event PaymentReceived(address from, uint256 amount); // 合约收款事件
9
10    uint256 public totalShares; // 总份额
11    uint256 public totalReleased; // 总支付
12
13    mapping(address => uint256) public shares; // 每个受益人的份额
14    mapping(address => uint256) public released; // 支付给每个受益人的金额
15    address[] public payees; // 受益人数组
16
17    /**
18     * @dev 初始化受益人数组_payees和分拆份额数组_shares
19     * 数组长度不能为0，两个数组长度要相等，_shares中元素要大于0，_payees中地址不能为0地址且不能有重复地址
20     */
21    constructor(address[] memory _payees, uint256[] memory _shares) payable {
22        // 检查 _payees和 _shares数组长度相同，且不为0
23    }
24
25    // 调用
26    call to PaymentSplit.payees 查看第一位受益人地址
27    call to PaymentSplit.releasable 查看第一位受益人能领到的ETH数额
28    call to PaymentSplit.released 查看第一位受益人已领到的ETH数额
29    call to PaymentSplit.shares 查看第一位受益人的份额
30    call to PaymentSplit.totalReleased 查看合约约定总ETH数额
31    call to PaymentSplit.totalShares 查看合约的总份额
32
33    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.payees(uint256) data: 0x630...00000
34    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.releasable(address) data: 0x63F...eddC4
35    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.released(address) data: 0x6B6...eddC4
36    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.shares(address) data: 0x0e7...eddC4
37    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.totalReleased() data: 0x633...b7dC3
38    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.totalShares() data: 0x3D9...8eF39

```

6. 查看受益人地址、份额、应分到的 ETH

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (London)

ACCOUNT: 0x5B3...eddC4 [98.249999%]

GAS LIMIT: 3000000

VALUE: 0 Ether

CONTRACT: PaymentSplit - 42_PaymentSplit/Pay

DEPLOY: [0x5B38D6a701c568545] [1,3] **transact**

Transactions recorded: 4

Deployed Contracts: PAYMENTSPLIT AT 0xD7A...F77

release address_account

payees uint256

pendingPay... address_account uint

releasable address_account uint

released address

shares address

totalReleased

PaymentSplit.sol

```

1 // 分拆合约
2 // @dev 这个合约会把收到的ETH按照事先定好的份额分给几个账户。收到ETH会存在分拆合约中，需要每个受益人调用release()函数来领取。
3
4 contract PaymentSplit{
5     // 事件
6     event PayeeAdded(address account, uint256 shares); // 增加受益人事件
7     event PaymentReleased(address to, uint256 amount); // 受益人提款事件
8     event PaymentReceived(address from, uint256 amount); // 合约收款事件
9
10    uint256 public totalShares; // 总份额
11    uint256 public totalReleased; // 总支付
12
13    mapping(address => uint256) public shares; // 每个受益人的份额
14    mapping(address => uint256) public released; // 支付给每个受益人的金额
15    address[] public payees; // 受益人数组
16
17    /**
18     * @dev 初始化受益人数组_payees和分拆份额数组_shares
19     * 数组长度不能为0，两个数组长度要相等，_shares中元素要大于0，_payees中地址不能为0地址且不能有重复地址
20     */
21    constructor(address[] memory _payees, uint256[] memory _shares) payable {
22        // 检查 _payees和 _shares数组长度相同，且不为0
23    }
24
25    // 调用
26    call to PaymentSplit.payees 查看第一位受益人地址
27    call to PaymentSplit.releasable 查看第一位受益人能领到的ETH数额
28    call to PaymentSplit.released 查看第一位受益人已领到的ETH数额
29    call to PaymentSplit.shares 查看第一位受益人的份额
30    call to PaymentSplit.totalReleased 查看合约约定总ETH数额
31    call to PaymentSplit.totalShares 查看合约的总份额
32
33    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.payees(uint256) data: 0x630...00000
34    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.releasable(address) data: 0x63F...eddC4
35    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.released(address) data: 0x6B6...eddC4
36    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.shares(address) data: 0x0e7...eddC4
37    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.totalReleased() data: 0x633...b7dC3
38    CALL [call] from: 0x5B38D6a701c568545 to: PaymentSplit.totalShares() data: 0x3D9...8eF39

```

DEPLOY & RUN TRANSACTIONS

VALUE
0 Ether

CONTRACT
PaymentSplit - 42_PaymentSplitPay

DEPLOY

_PAYEES: 0x5B38Da6a701c568545

_SHARES: [1,3]

transact

☐ Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

▼ PAYMENTSPLIT AT 0xD7A...F77

release address_account

payees 1

0: address: 0xAb843F64d9C6d1Ae677d0331593140272D8780

pendingPay... address_account, uint

releasable 0xAb843F64d9C6d1

0: uint256: 7500000000000000000

released 0xAb843F64d9C6d1

0: uint256: 0

shares 0xAb843F64d9C6d1

0: uint256: 3

totalReleased

0: uint256: 0

uintShares

0: uint256: 4

Low level interactions 1

```

4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

```

ContractDefinition PaymentSplit 1 reference(s)

☐ listen on all transactions

CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit payees(uint256) data: 0x830...0001

CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit releasable(address) data: 0x83F...35b82

CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit released(address) data: 0x95...35b82

CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit shares(address) data: 0x0e7...35b82

release payees pendingPay... releasable released shares totalReleased uintShares

0: uint256: 7500000000000000000 0: uint256: 0 0: uint256: 3 0: uint256: 0 0: uint256: 4

CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit payees(uint256) data: 0x830...0001
CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit releasable(address) data: 0x83F...35b82
CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit released(address) data: 0x95...35b82
CALL [call] from: 0x5B38Da6a701c568545dE6f803FdB07585b66d434 to: PaymentSplit shares(address) data: 0x0e7...35b82

7. 函数领取 ETH

The screenshot shows the Remix IDE interface during the deployment of a `PaymentSplit` contract. The left sidebar displays the contract's state with fields like `_PAYEEs`, `_SHARES`, and a `release` button. The main editor shows the Solidity code for the `PaymentSplit` contract, including the `release` function. The right sidebar shows the transaction log, highlighting the `release` function call and the resulting state changes.

Contract State (Left Sidebar):

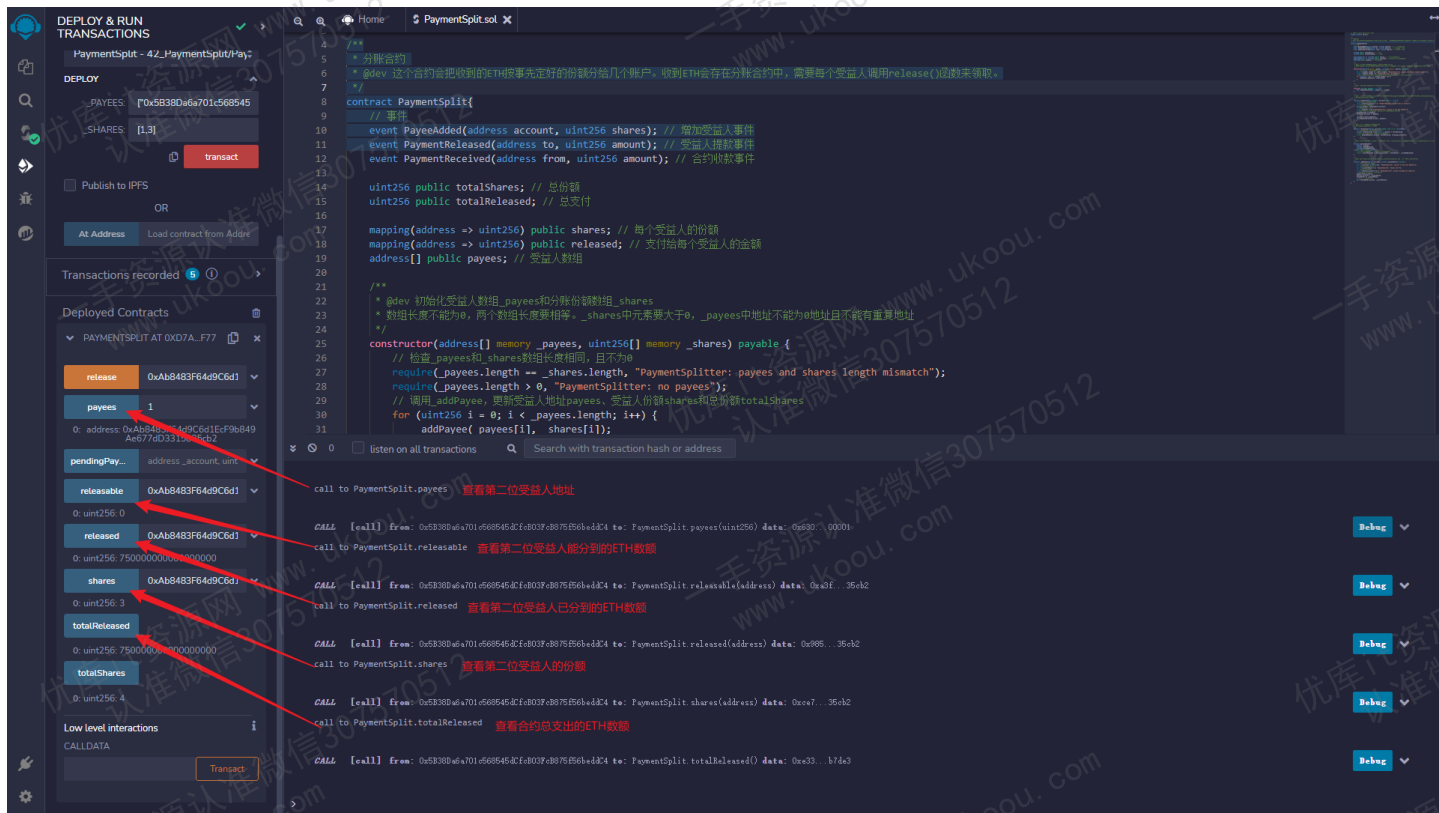
- `_PAYEEs`: `0x5B38Da6a701c568545`
- `_SHARES`: `[1,3]`
- `release` button: `transaction`
- `payees`: `1`
- `address`: `0xAb8483F64d9C6d1E4F9b849Ae677d3315835cb2`
- `pendingPay`: `address_account_uint`
- `releasable`: `0xAb8483F64d9C6d1E4F9b849Ae677d3315835cb2`
- `released`: `0xAb8483F64d9C6d1E4F9b849Ae677d3315835cb2`
- `shares`: `0xAb8483F64d9C6d1E4F9b849Ae677d3315835cb2`
- `totalReleased`: `0xAb8483F64d9C6d1E4F9b849Ae677d3315835cb2`
- `totalShares`: `0xAb8483F64d9C6d1E4F9b849Ae677d3315835cb2`

Contract Code (Main Editor):

```

4 // 分拆合约
5 // @dev 这个合约会把收到的ETH按照事先定好的份额分给几个账户。收到ETH会存在分拆合约中，需要每个受益人调用release()函数来领取。
6
7 contract PaymentSplit{
8     // 事件
9     event PayeeAdded(address account, uint256 shares); // 增加受益人事件
10    event PaymentReleased(address to, uint256 amount); // 受益人提款事件
11    event PaymentReceived(address from, uint256 amount); // 合约收款事件
12
13    uint256 public totalShares; // 总份额
14    uint256 public totalReleased; // 总支付
15
16    mapping(address => uint256) public shares; // 每个受益人的份额
17    mapping(address => uint256) public released; // 支付给每个受益人的金额
18    address[] public payees; // 受益人数组
19
20    // ... (other code)
21
22    function release(address to) public {
23        // ... (release logic)
24    }
25
26    // ... (other functions)
27
28    // ... (other code)
29
30    // ... (other code)
31
32    // ... (other code)
33
34    // ... (other code)
35
36    // ... (other code)
37
38    // ... (other code)
39
40    // ... (other code)
41
42    // ... (other code)
43
44    // ... (other code)
45
46    // ... (other code)
47
48    // ... (other code)
49
50    // ... (other code)
51
52    // ... (other code)
53
54    // ... (other code)
55
56    // ... (other code)
57
58    // ... (other code)
59
60    // ... (other code)
61
62    // ... (other code)
63
64    // ... (other code)
65
66    // ... (other code)
67
68    // ... (other code)
69
70    // ... (other code)
71
72    // ... (other code)
73
74    // ... (other code)
75
76    // ... (other code)
77
78    // ... (other code)
79
80    // ... (other code)
81
82    // ... (other code)
83
84    // ... (other code)
85
86    // ... (other code)
87
88    // ... (other code)
89
90    // ... (other code)
91
92    // ... (other code)
93
94    // ... (other code)
95
96    // ... (other code)
97
98    // ... (other code)
99
100   // ... (other code)
101
102   // ... (other code)
103
104   // ... (other code)
105
106   // ... (other code)
107
108   // ... (other code)
109
110   // ... (other code)
111
112   // ... (other code)
113
114   // ... (other code)
115
116   // ... (other code)
117
118   // ... (other code)
119
120   // ... (other code)
121
122   // ... (other code)
123
124   // ... (other code)
125
126   // ... (other code)
127
128   // ... (other code)
129
130   // ... (other code)
131
132   // ... (other code)
133
134   // ... (other code)
135
136   // ... (other code)
137
138   // ... (other code)
139
140   // ... (other code)
141
142   // ... (other code)
143
144   // ... (other code)
145
146   // ... (other code)
147
148   // ... (other code)
149
150   // ... (other code)
151
152   // ... (other code)
153
154   // ... (other code)
155
156   // ... (other code)
157
158   // ... (other code)
159
160   // ... (other code)
161
162   // ... (other code)
163
164   // ... (other code)
165
166   // ... (other code)
167
168   // ... (other code)
169
170   // ... (other code)
171
172   // ... (other code)
173
174   // ... (other code)
175
176   // ... (other code)
177
178   // ... (other code)
179
180   // ... (other code)
181
182   // ... (other code)
183
184   // ... (other code)
185
186   // ... (other code)
187
188   // ... (other code)
189
190   // ... (other code)
191
192   // ... (other code)
193
194   // ... (other code)
195
196   // ... (other code)
197
198   // ... (other code)
199
200   // ... (other code)
201
202   // ... (other code)
203
204   // ... (other code)
205
206   // ... (other code)
207
208   // ... (other code)
209
210   // ... (other code)
211
212   // ... (other code)
213
214   // ... (other code)
215
216   // ... (other code)
217
218   // ... (other code)
219
220   // ... (other code)
221
222   // ... (other code)
223
224   // ... (other code)
225
226   // ... (other code)
227
228   // ... (other code)
229
230   // ... (other code)
231
232   // ... (other code)
233
234   // ... (other code)
235
236   // ... (other code)
237
238   // ... (other code)
239
240   // ... (other code)
241
242   // ... (other code)
243
244   // ... (other code)
245
246   // ... (other code)
247
248   // ... (other code)
249
250   // ... (other code)
251
252   // ... (other code)
253
254   // ... (other code)
255
256   // ... (other code)
257
258   // ... (other code)
259
260   // ... (other code)
261
262   // ... (other code)
263
264   // ... (other code)
265
266   // ... (other code)
267
268   // ... (other code)
269
270   // ... (other code)
271
272   // ... (other code)
273
274   // ... (other code)
275
276   // ... (other code)
277
278   // ... (other code)
279
280   // ... (other code)
281
282   // ... (other code)
283
284   // ... (other code)
285
286   // ... (other code)
287
288   // ... (other code)
289
290   // ... (other code)
291
292   // ... (other code)
293
294   // ... (other code)
295
296   // ... (other code)
297
298   // ... (other code)
299
300   // ... (other code)
301
302   // ... (other code)
303
304   // ... (other code)
305
306   // ... (other code)
307
308   // ... (other code)
309
310   // ... (other code)
311
312   // ... (other code)
313
314   // ... (other code)
315
316   // ... (other code)
317
318   // ... (other code)
319
320   // ... (other code)
321
322   // ... (other code)
323
324   // ... (other code)
325
326   // ... (other code)
327
328   // ... (other code)
329
330   // ... (other code)
331
332   // ... (other code)
333
334   // ... (other code)
335
336   // ... (other code)
337
338   // ... (other code)
339
340   // ... (other code)
341
342   // ... (other code)
343
344   // ... (other code)
345
346   // ... (other code)
347
348   // ... (other code)
349
350   // ... (other code)
351
352   // ... (other code)
353
354   // ... (other code)
355
356   // ... (other code)
357
358   // ... (other code)
359
360   // ... (other code)
361
362   // ... (other code)
363
364   // ... (other code)
365
366   // ... (other code)
367
368   // ... (other code)
369
370   // ... (other code)
371
372   // ... (other code)
373
374   // ... (other code)
375
376   // ... (other code)
377
378   // ... (other code)
379
380   // ... (other code)
381
382   // ... (other code)
383
384   // ... (other code)
385
386   // ... (other code)
387
388   // ... (other code)
389
390   // ... (other code)
391
392   // ... (other code)
393
394   // ... (other code)
395
396   // ... (other code)
397
398   // ... (other code)
399
400   // ... (other code)
401
402   // ... (other code)
403
404   // ... (other code)
405
406   // ... (other code)
407
408   // ... (other code)
409
410   // ... (other code)
411
412   // ... (other code)
413
414   // ... (other code)
415
416   // ... (other code)
417
418   // ... (other code)
419
420   // ... (other code)
421
422   // ... (other code)
423
424   // ... (other code)
425
426   // ... (other code)
427
428   // ... (other code)
429
430   // ... (other code)
431
432   // ... (other code)
433
434   // ... (other code)
435
436   // ... (other code)
437
438   // ... (other code)
439
440   // ... (other code)
441
442   // ... (other code)
443
444   // ... (other code)
445
446   // ... (other code)
447
448   // ... (other code)
449
450   // ... (other code
```

8. 查看总支出、受益人余额、应分到的 ETH 的变化



总结

这一讲，我们介绍了分账合约。在区块链的世界里，**Code is Law**，我们可以事先把每个人应分的比例写在智能合约中，获得收入后，由智能合约来进行分账，避免事后“分赃不均”。

二、空投 Airdrop

在币圈，最开心的一件事就是领空投，空手套白狼。这一讲，我们将学习如何使用智能合约发送 **ERC20** 代币空投。

空投是币圈中一种营销策略，项目方将代币免费发放给特定用户群体。为了拿到空投资格，用户通常需要完成一些简单的任务，如测试产品、分享新闻、介绍朋友等。项目方通过空投可以获得种子用户，而用户可以获得一笔财富，两全其美。

因为每次接收空投的用户很多，项目方不可能一笔一笔的转账。利用智能合约批量发放 **ERC20** 代币，可以显著提高空投效率。

空投代币合约

Airdrop 空投合约逻辑非常简单：利用循环，一笔交易将 **ERC20** 代币发送给多个地址。合约中包含两个函数

- `getSum()` 函数: 返回 `uint` 数组的和。

```

1 // 数组求和函数
2 function getSum(uint256[] calldata _arr) public pure returns(uint sum)
3 {
4     for(uint i = 0; i < _arr.length; i++)
5         sum = sum + _arr[i];
6 }

```

- `multiTransferToken()` 函数: 发送 ERC20 代币空投, 包含 3 个参数:

- `_token`: 代币合约地址 (`address` 类型)
- `_addresses`: 接收空投的用户地址数组 (`address[]` 类型)
- `_amounts`: 空投数量数组, 对应 `_addresses` 里每个地址的数量 (`uint[]` 类型)
- 该函数有两个检查: 第一个 `require` 检查了 `_addresses` 和 `_amounts` 两个数组长度是否相等; 第二个 `require` 检查了空投合约的授权额度大于要空投的代币数量总和。

```

1 /// @notice 向多个地址转账ERC20代币, 使用前需要先授权
2 ///
3 /// @param _token 转账的ERC20代币地址
4 /// @param _addresses 空投地址数组
5 /// @param _amounts 代币数量数组 (每个地址的空投数量)
6 function multiTransferToken(
7     address _token,
8     address[] calldata _addresses,
9     uint256[] calldata _amounts
10 ) external {
11     // 检查: _addresses和_amounts数组的长度相等
12     require(_addresses.length == _amounts.length, "Lengths of Addresses and
    Amounts NOT EQUAL");
13     IERC20 token = IERC20(_token); // 声明IERC合约变量
14     uint _amountSum = getSum(_amounts); // 计算空投代币总量
15     // 检查: 授权代币数量 >= 空投代币总量
16     require(token.allowance(msg.sender, address(this)) >= _amountSum, "Need
    Approve ERC20 token");
17
18     // for循环, 利用transferFrom函数发送空投
19     for (uint8 i; i < _addresses.length; i++) {
20         token.transferFrom(msg.sender, _addresses[i], _amounts[i]);
21     }
22 }

```

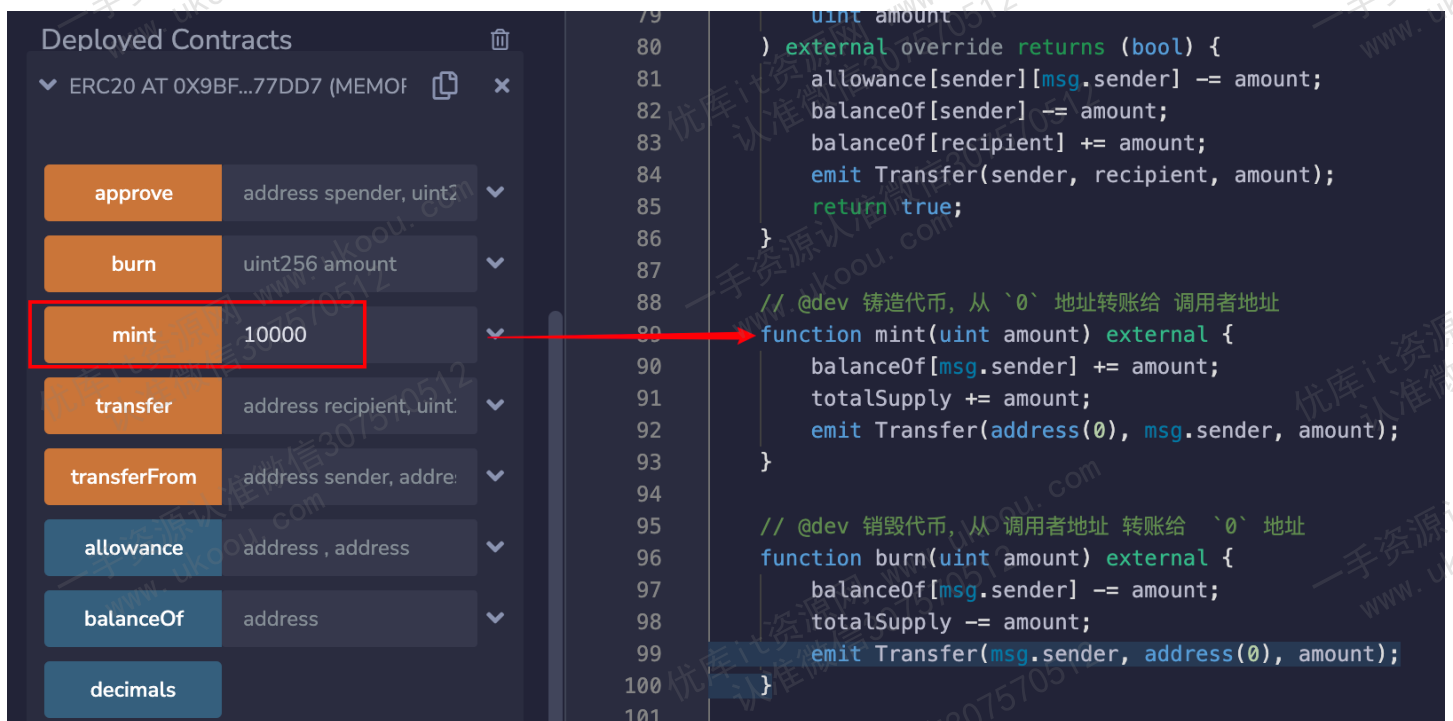
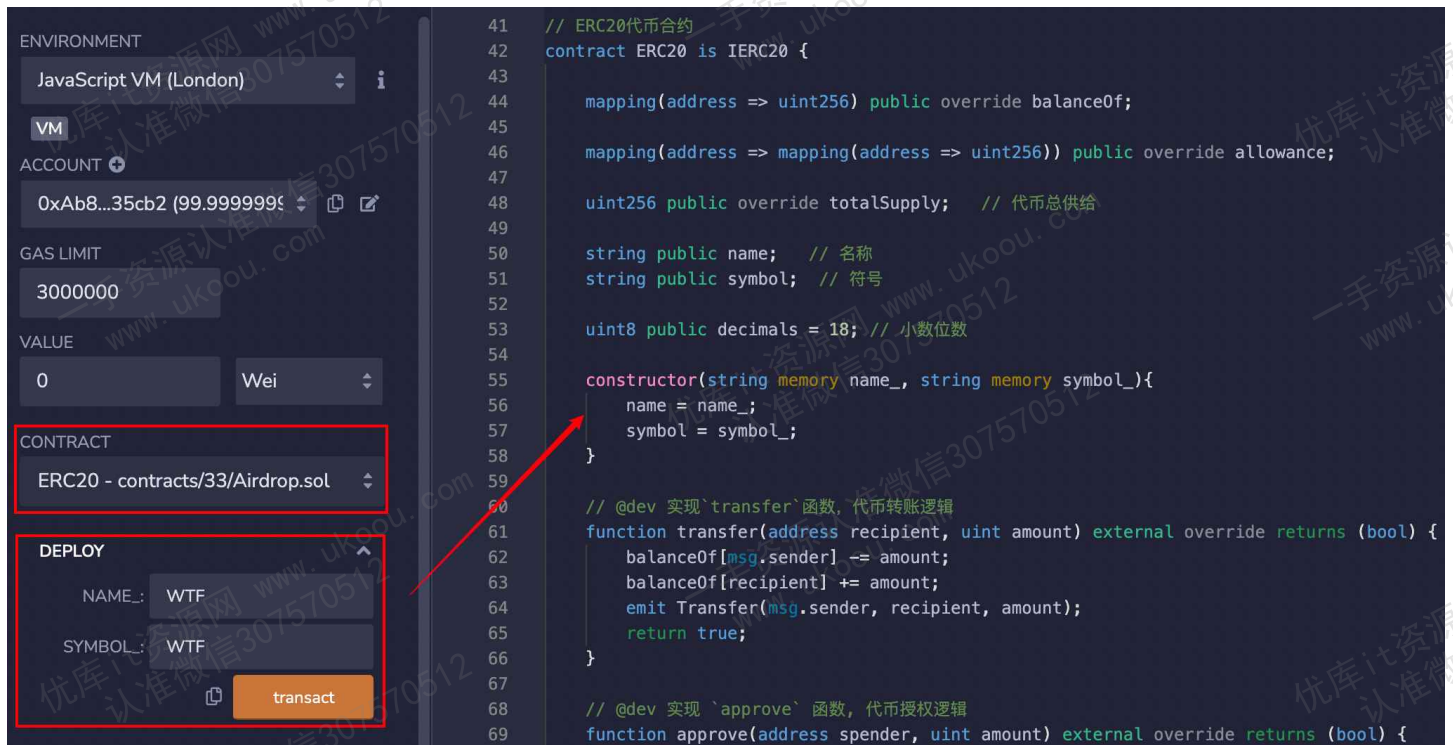
- `multiTransferETH()` 函数: 发送 `ETH` 空投, 包含 `2` 个参数:

- `_addresses`: 接收空投的用户地址数组 (`address[]` 类型)
- `_amounts`: 空投数量数组, 对应 `_addresses` 里每个地址的数量 (`uint[]` 类型)

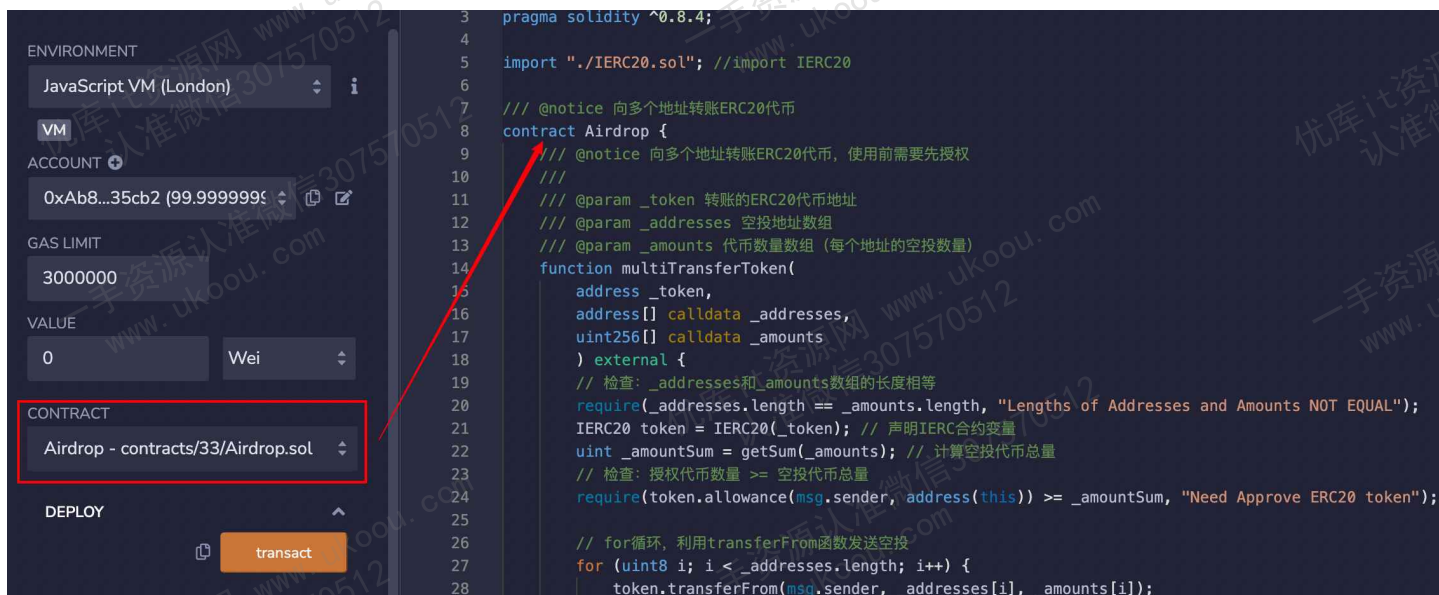
```
1  /// 向多个地址转账ETH
2  function multiTransferETH(
3      address payable[] calldata _addresses,
4      uint256[] calldata _amounts
5  ) public payable {
6      // 检查: _addresses和_amounts数组的长度相等
7      require(_addresses.length == _amounts.length, "Lengths of Addresses and
      Amounts NOT EQUAL");
8      uint _amountSum = getSum(_amounts); // 计算空投ETH总量
9      // 检查转入ETH等于空投总量
10     require(msg.value == _amountSum, "Transfer amount error");
11     // for循环, 利用transfer函数发送ETH
12     for (uint256 i = 0; i < _addresses.length; i++) {
13         _addresses[i].transfer(_amounts[i]);
14     }
15 }
```

空投实践

1. 部署 `ERC20` 代币合约, 并给自己 `mint` 10000 单位代币。



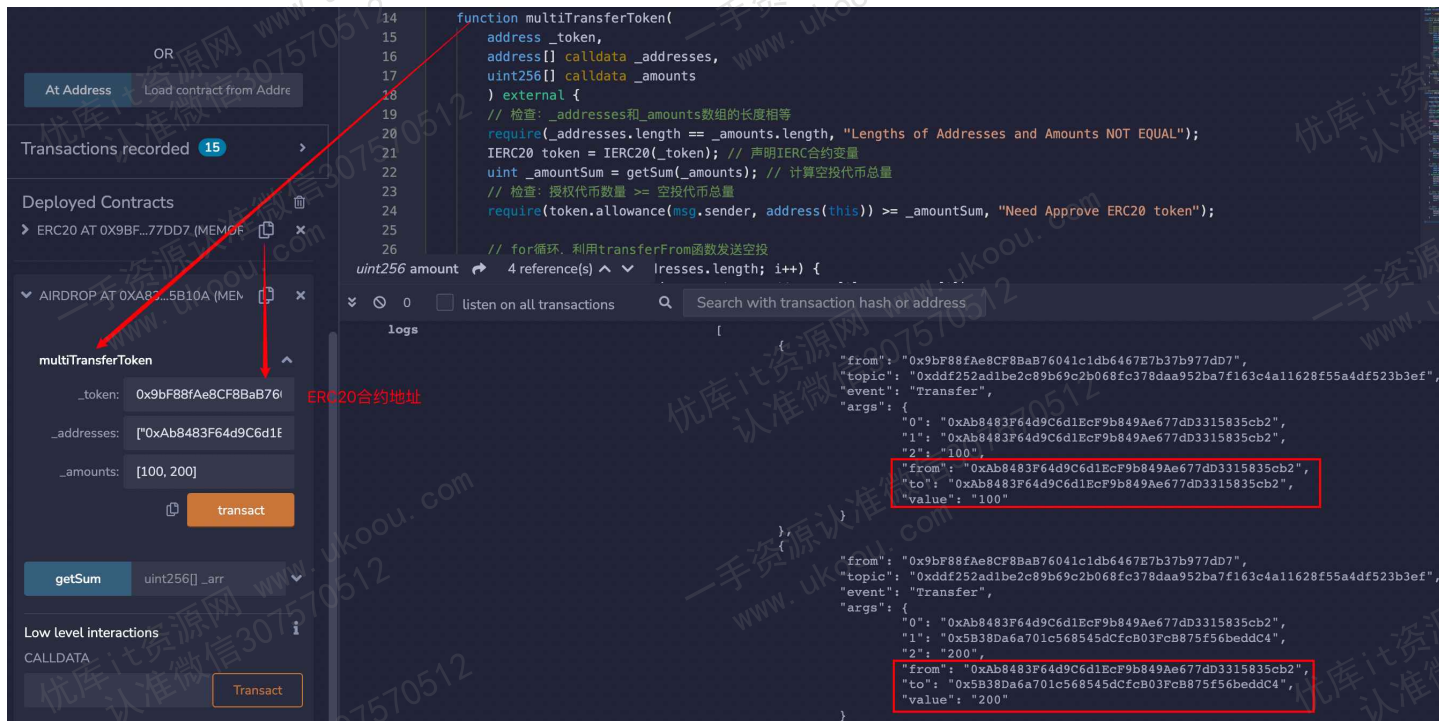
1. 部署 Airdrop 空投合约。



1. 利用 ERC20 代币合约中的 approve() 函数, 给 Airdrop 空投合约授权10000 单位代币。

1. 执行 Airdrop 合约的 multiTransferToken() 函数进行空投, _token 填 ERC20 代币地址, _addresses 和 _amounts 按照以下填写

```
1 // _addresses填写
2 ["0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
  "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"]
3
4 // _amounts填写
5 [100, 200]
```

1. 利用 ERC20 合约的 `balanceOf()` 函数查询上面用户地址的代币余额, 成功变为 100 和 200 , 空投成功!

总结

这一讲, 我们介绍了如何使用 `solidity` 写 ERC20 代币空投合约, 极大增加空投效率。我撸空投收获最大的一次是 ENS 空投, 你们呢?