

4 版权保护模块——出版合约 代码说明

191830090 刘璐

1、运行说明

publish.sol文件为出版合约。

出版即部署，部署时需填入正确的构造方法参数，格式参考如下：

_addrs为股东地址列表、_names为股东信息/签名列表、_weight为股东权重列表（总和需为100）、_pid代指软件、_price为软件的价钱

注：price的单位为100 wei

```
/// 构造方法
/// 参数：股东发布出版合约时必须填写的内容
constructor(address payable[] memory _addrs, string[] memory _names, uint[] memory _weight, uint256 _pid, uint _price) {
    require((_addrs.length == _names.length) && (_names.length == _weight.length));
}
```

输入示例：

CONTRACT (Compiled By Remix)

Publish - publish.sol

DEPLOY

_ADDRS:	["0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"]
_NAMES:	["shareholder1"]
_WEIGHT:	[100]
_PID:	1
_PRICE:	10


Calldata ... transact

部署成功得到如下信息：



```
✓ [vm] from: 0x5B3...eddC4 to: Publish. (constructor) value: 0 wei data: 0x608...00064 logs: 0 hash: 0x19b...9796f

status      true Transaction mined and execution succeed
transaction hash  0x19b382f143ba7be131451ac69563bdc574aec3ae50389d7ca5147b8d1a89796f
from        0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to          Publish. (constructor)
gas         665670 gas
transaction cost  578843 gas
execution cost   578843 gas
input        0x608...00064
decoded input  {
  "address[] _addrs": [
    "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
  ],
  "string[] _names": [
    "shareholder1"
  ],
  "uint256[] _weight": [
    "100"
  ],
  "uint256 _pid": "1",
  "uint256 _price": "10"
}
decoded output  -
logs            []
```

用户购买时，设置正确的金额(price的100倍wei)，点击buy方法：

ACCOUNT 

0xAb8...35cb2 (99.99999999)





GAS LIMIT



3000000

VALUE

1000


Wei 

Deployed Contracts 

▼ PUBLISH AT 0XD91...39138 (MEMORY) 

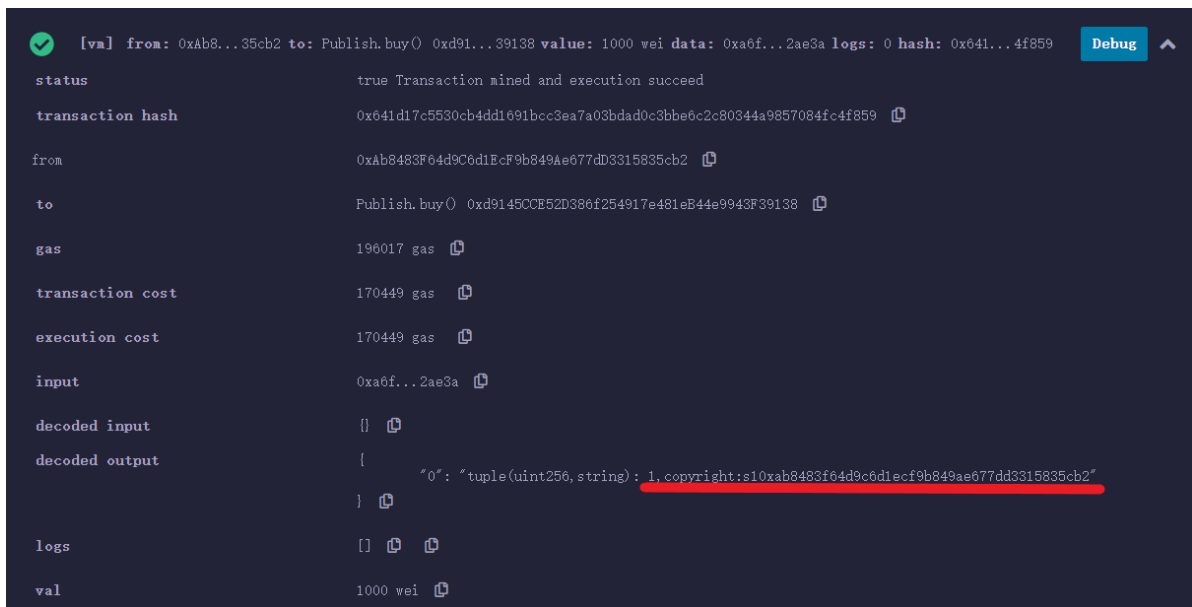
Balance: 0 ETH

buy

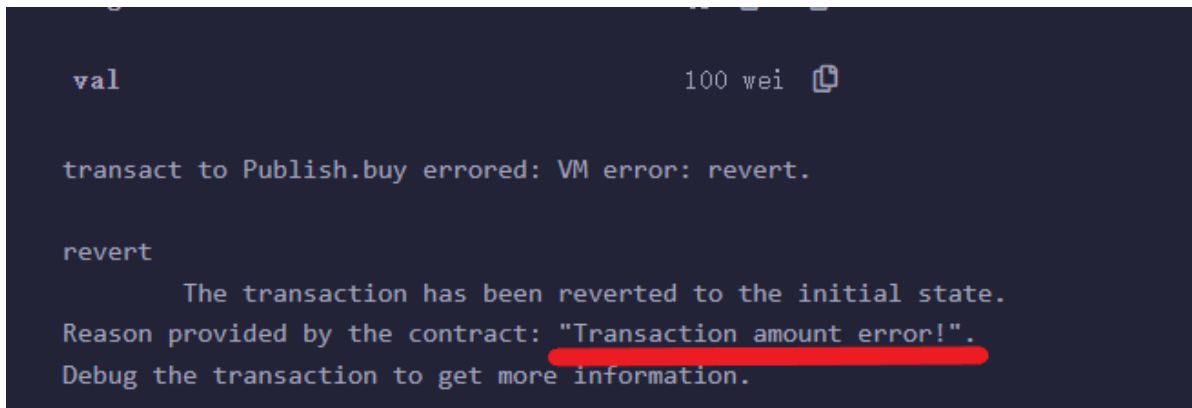
Low level interactions 

CALLDATA

购买成功后会返回带水印的软件（软件，水印）：



如用户转入金额错误则有相应提示：



2、代码解释

为股东、软件、用户创建三个结构体类型：

```
struct Shareholder{
    address payable addr; // 股东的收款地址
    string name;
    uint weight; // 股东权重。小于100,和为100
}

struct Product{
    uint256 id; // 代表软件
    string waterMark; // 水印
}

struct User{
    address addr;
    uint256 buyTime;
}
```

声明出版合约的storage类型变量：

```

address payable owner; // 合约发起者
Product product;      // 合约作用的软件

mapping(uint => Shareholder) holderMap; // 股东列表
mapping(uint => User) userMap;          // 购买该软件的用户列表

uint price; // 软件的售价（单位：100 wei）
uint holderNum; // 股东的总数量
uint userNum; // 软件购买者的总数量

```

出版合约发布时，由发布者填入发布合约所必须的填写的内容，包括股东信息（收款地址、个人信息、权重）、软件、以及软件售价。

生成带有所有股东信息的软件水印。

```

// 构造方法
// 参数：股东发布出版合约时必须填写的内容
constructor(address payable[] memory _addrs, string[] memory _names, uint[]
memory _weight, uint256 _pid, uint _price) {
    // 验证股东信息的完整
    require((_addrs.length == _names.length) && (_names.length ==
_weight.length));
    price = _price;
    holderNum = _addrs.length;
    userNum = 0;

    string memory waterMark = "copyright:";
    // 在水印中存储带所有股东的版权信息
    for(uint i=0; i<holderNum; i++){
        holderMap[i] = Shareholder(_addrs[i], _names[i], _weight[i]);
        waterMark = strConcat(waterMark, _names[i]);
    }

    product = Product(_pid, waterMark);

    owner = payable(msg.sender);
}

```

用户购买软件方法，由用户发起支付，moneyCheck方法检验用户支付金额，若金额错误提示错误信息，金额正确则继续执行交易。

合约将用户支付的金额按权重转入各股东账户，在合约的购买者列表中记录购买者信息，并生成带有购买者个人信息水印的软件副本返还给购买者。

```

//验证用户输入金额是否合适
modifier moneyCheck(uint _price){
    require(msg.value == _price*100,
        "Transaction amount error!");
    _;
}

// 用户购买软件
function buy() payable public moneyCheck(price) returns(Product memory){
    // 按股东权重分配金额
}

```

```

    for(uint i=0; i<holderNum; i++){
        shareholder memory temp = holderMap[i];
        address payable sowner = temp.addr;
        uint weight = temp.weight;
        // 将金额从合约账户转到股东账户
        sowner.transfer(price*weight);
    }

    // 记录购买者
    userMap[userNum++] = User(msg.sender, block.timestamp);

    // 生成发给购买用户的软件副本（带有购买者水印）
    Product memory rproduct = product;
    string memory infoMark = addrToStr(msg.sender);
    rproduct.waterMark = strConcat(product.waterMark, infoMark);

    return rproduct;
}

```

处理水印过程中涉及的一些字符串处理方法：

```

// 地址转换成字符串 (address -> string)
function addrToStr(address addr) internal pure returns (string memory) {
    return bytesToStr(abi.encodePacked(addr));
}

// Bytes转换成字符串 (bytes -> string)
function bytesToStr(bytes memory data) internal pure returns (string memory)
{
    bytes memory alphabet = "0123456789abcdef";

    bytes memory str = new bytes(2 + data.length * 2);
    str[0] = "0";
    str[1] = "x";
    for (uint256 i = 0; i < data.length; i++) {
        str[2 + i * 2] = alphabet[uint256(uint8(data[i] >> 4))];
        str[3 + i * 2] = alphabet[uint256(uint8(data[i] & 0x0f))];
    }
    return string(str);
}

// 字符串拼接
function strConcat(string memory _a, string memory _b) internal pure
returns(string memory){
    bytes memory _ba = bytes(_a);
    bytes memory _bb = bytes(_b);

    string memory ret = new string(_ba.length+_bb.length);
    bytes memory bret = bytes(ret);

    uint k=0;
    for(uint i=0; i<_ba.length; i++){
        bret[k++] = _ba[i];
    }
    for(uint i=0; i<_bb.length; i++){

```

```
        bret[k++] = _bb[i];  
    }  
  
    return string(ret);  
}
```