区块链课程设计报告

第三阶段准备阶段

学 院 名 称 ： 数据科学与计算机学院

专业（班级）： 16 软件工程电子政务方向

学 生 姓 名 ： 唐育涛

学 号 16340209

时 间 ： 2018 年 11 月 11 日

区块链课程作业设计报告

课程任务：

4. 提交合约部署报告
   - 提交合约部署的情况如部署截图，相关接口的解释，命名为学号+姓名，以 pdf 文件的形式，提交到超算习堂
   - 智能合约代码需有明确清晰的执行与逻辑，与题目相符合

写在前面的话：

软件题目：部署 token 货币测试发行，实现接口化、自动化、可众筹、可升级的 token，如果有可能完善其他功能

这一过程想做的很多，但是遇到的困难也很多，Linux 下有一套牛逼的框架叫 truffle，编译、调试、部署一条龙，一条——龙，但我自己是在 windows 下实现，多次想放弃转战 linux，但仍坚持着走下去。饮水思源，先贴上参考及推荐的博客：

solidity 语法知识快速入门博客：

https://blog.csdn.net/JohnnyMartin/article/details/79565875

ECR20 标准 token 发布的参考博客及代码：

https://blog.csdn.net/JohnnyMartin/article/details/79642784

区块链相关操作：

https://blog.csdn.net/sunshine1314/article/details/79692502

下列开始代码书写和合约部署：

凡是 ethereum 上的 token，都必须要符合 ERC20 标准。该标准共有 9 个函数：

- Name symbol decimals totalSupply balanceOf
- Allowance **transfer** **transferFrom** **approve**

前 6 个函数可以利用 solidity 的语法糖：默认给 public 的 storage 变量生成一个同名 getter，我们只需给前 6 个函数定义同名的 public 变量即可。

首先是前六个函数，都是能够简单实现完成的。

```
string public name = "myToken";                                  //token 名称
string public symbol = "MTC";                                    //token 符号
uint256 public decimals = 18;                                    //token 小数位数
uint256 public totalSupply = 10000 * 10 ** (uint256(18)); //总发行量
mapping (address => uint256) public balanceOf;                   //存放账户余额的map
```

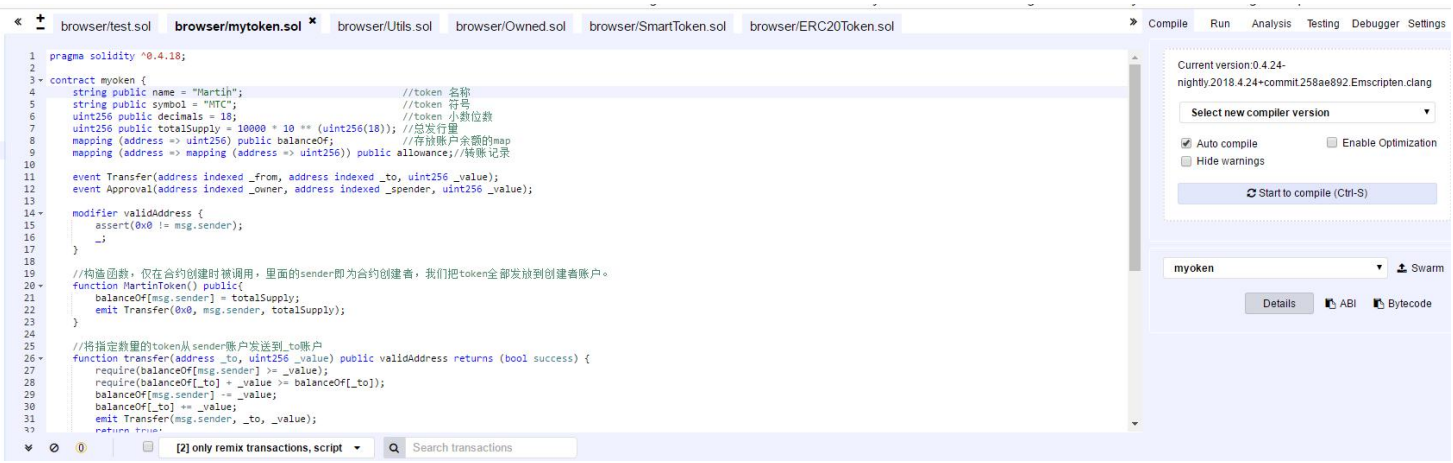
接下去实现其他三个函数，我们主要是先走一遍最简单的 token 发布，之后我们会完善它们的功能，实现接口化，自动化等。

```
//将指定数量的token从sender账户发送到_to账户
function transfer(address _to, uint256 _value) public validAddress returns (bool success) {
    require(balanceOf[msg.sender] >= _value);
    require(balanceOf[_to] + _value >= balanceOf[_to]);
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
    emit Transfer(msg.sender, _to, _value);
    return true;
}

//将A账户的N个token转移到B的账户中，前提是事前A‘批准’给B账户M个token，M >= N。
function transferFrom(address _from, address _to, uint256 _value) public validAddress returns (bool success) {
    require(balanceOf[_from] >= _value);
    require(balanceOf[_to] + _value >= balanceOf[_to]);
    require(allowance[_from][msg.sender] >= _value);
    balanceOf[_to] += _value;
    balanceOf[_from] -= _value;
    allowance[_from][msg.sender] -= _value;
    emit Transfer(_from, _to, _value);
    return true;
}

//sender批准给制定的账户一定量的token
function approve(address _spender, uint256 _value)public validAddress returns (bool success) {
    require(_value == 0 || allowance[msg.sender][_spender] == 0);
    allowance[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

同时需要一个构造函数和一个辅助判断函数

```
//判断函数，需要鉴别是否是正确的发起者
modifier validAddress {
    assert(0x0 != msg.sender);
    _;
}

//构造函数，仅在合约创建时被调用，里面的sender即为合约创建者，我们把token全部发放到创建者账户。
function MartinToken() public{
    //账户创建者得到全部的资产
    balanceOf[msg.sender] = totalSupply;
    emit Transfer(0x0, msg.sender, totalSupply);
}
```

完成我们的代码后我们就接着将我们的合约在 remix 进行编写和尝试着编译部署



编译成功之后我们需要在本地私有链部署，首先进入私有链控制台

```
C:\>cd "Program Files"

C:\Program Files>cd Geth

C:\Program Files\Geth>geth --datadir chain1 --nodiscover console
INFO [11-26|17:59:18] Maximum peer count                       ETH=25 LES=0 total=25
INFO [11-26|17:59:18] Starting peer-to-peer node               instance=Geth/v1.8.3-stable-329ac18e/windows-amd64/go1.10

INFO [11-26|17:59:18] Allocated cache and file handles         database="C:\\Program Files\\Geth\\chain1\\geth\\chaindat
a" cache=768 handles=1024
INFO [11-26|17:59:19] Initialised chain configuration          config="{ChainID: 10 Homestead: 0 DAO: <nil> DAOSupport:
false EIP150: <nil> EIP155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil> Engine: unknown}"
INFO [11-26|17:59:19] Disk storage enabled for ethash caches   dir="C:\\Program Files\\Geth\\chain1\\geth\\ethash" count
=3
INFO [11-26|17:59:19] Disk storage enabled for ethash DAGs     dir=C:\\Users\\14156\\AppData\\Ethash               count
=2
INFO [11-26|17:59:19] Initialising Ethereum protocol           versions="[63 62]" network=1
WARN [11-26|17:59:19] Head state missing, repairing chain      number=47 hash=a94152…86dff6
INFO [11-26|17:59:19] Rewound blockchain to past state         number=16 hash=2f5fdb…48eb3c
INFO [11-26|17:59:19] Loaded most recent local header          number=47 hash=a94152…86dff6 td=6196096
INFO [11-26|17:59:19] Loaded most recent local full block      number=16 hash=2f5fdb…48eb3c td=2112064
```

2. 查看一下当前的账户并且解锁账户

```
> eth.accounts
["0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0", "0x4fca5d4431ad59476cdfb7eab9b4ee4067af0abe"]
> a0=eth.accounts[0]
"0x7c2dbaa840b93f988ff6b90ddc4f212705a82eb0"
>
>
>
> personal.unlockAccount(a0,"1234")
true
```

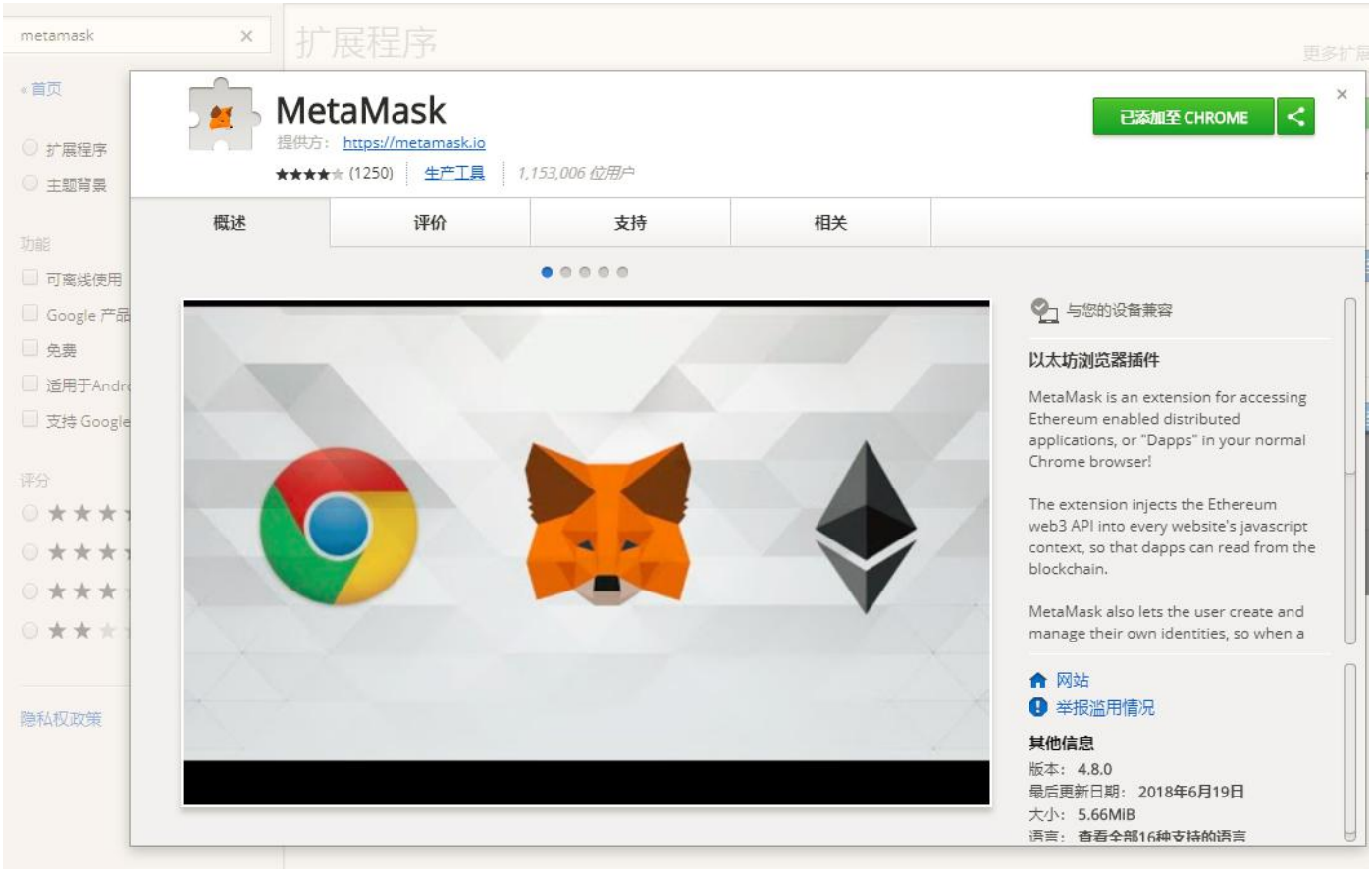
3. 合约部署

将 remix 下编译好的部署合约中 detail 关于合约部分复制到当前的控制台上，进行部署

```
4815600a165627a7a723058209b8afd12e9c4d88a262daf19a6d4dfb979d50233aa1de6edc79ab2a781976bba0029',
......      gas: '4700000'
......    }, function (e, contract){
......     console.log(e, contract);
......     if (typeof contract.address !== 'undefined') {
.........          console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract.transact
ionHash);
.........      }
......  })
INFO [11-26|18:04:02] Submitted contract creation              fullhash=0x959edcb1946672d207fab1ad064cfa23c8f4d94b0f1746
2d221505d026b4a897 contract=0x6370a93E9E0ad0Fe2acf7D8309e901AE3585b043
null [object Object]
undefined
```

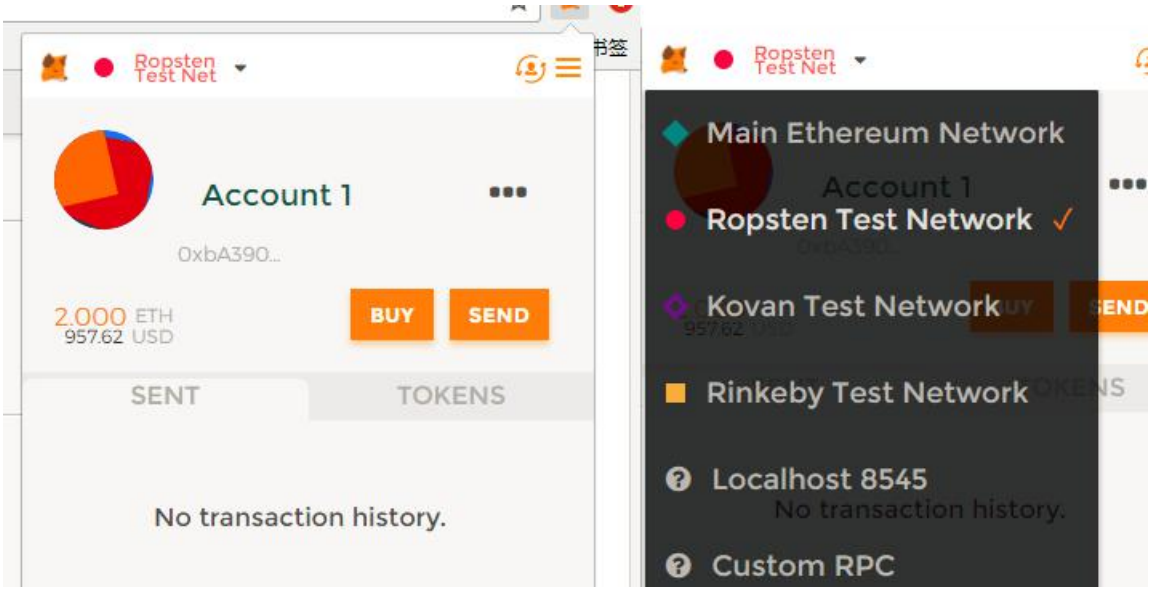
部署成功

当然可以尝试在 remix 直接部署，前提是要有 MetaMask 插件。



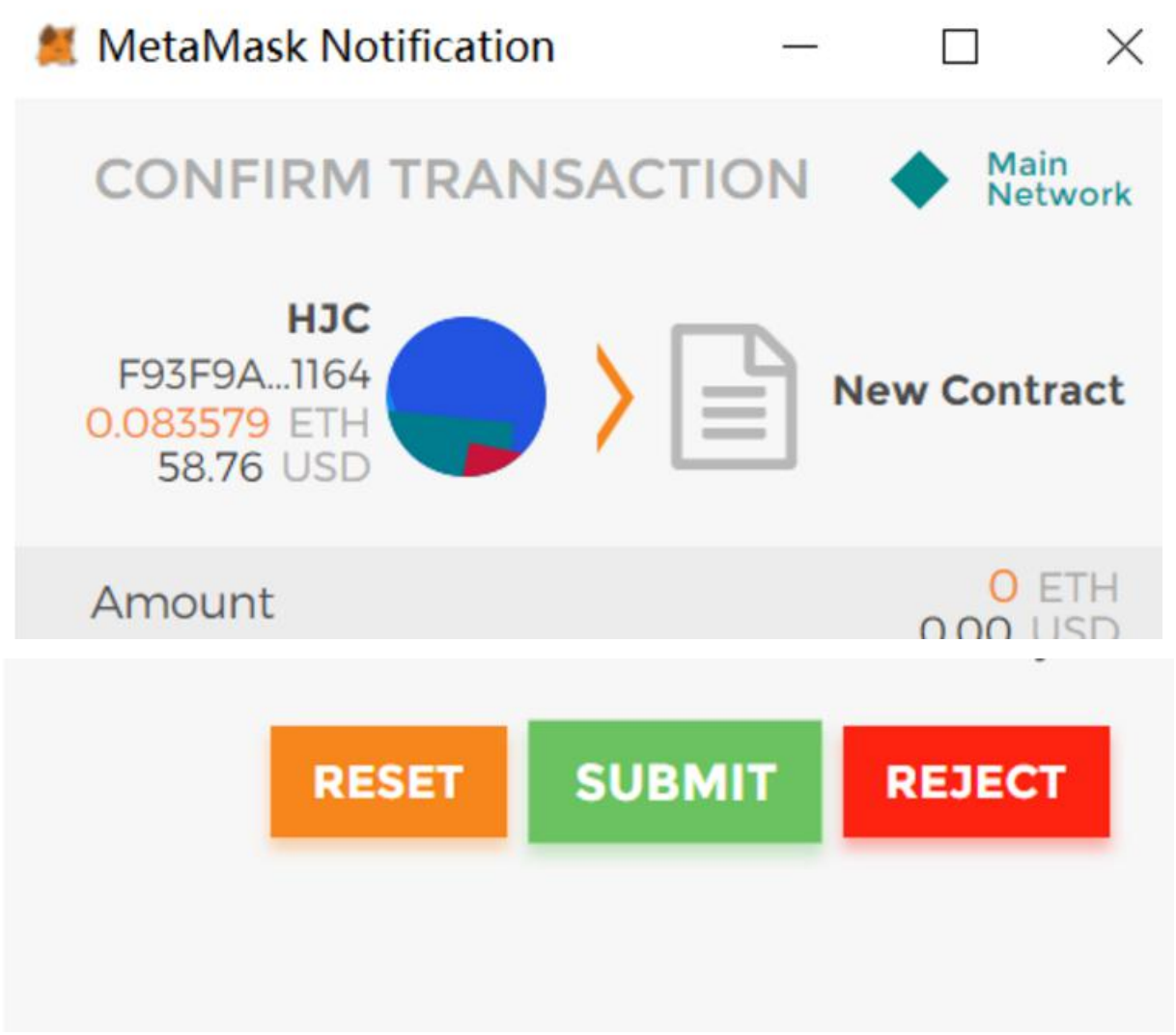
1. 在 MetaMask 中，切换到 Ropsten Test Network，并创建钱包

打开 MetaMask 钱包，进行注册和登录。



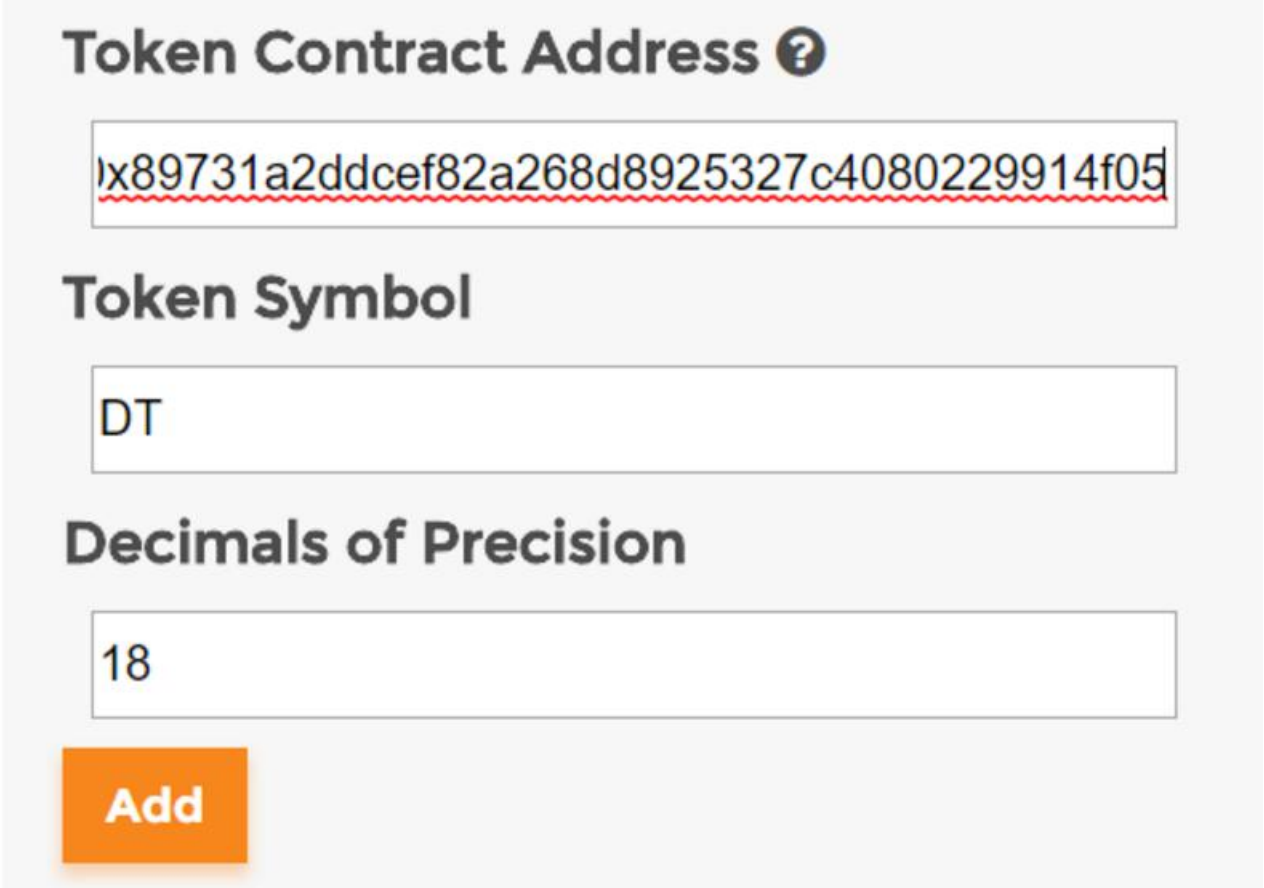
然后，从 Main Ethereum Network 切换到测试网络 Ropsten Test Network 连接成功。

MetaMask会弹出，确认交易，点击submit



交易被确认，会出现一个红框所示的合约地址

再进入 MetaMask，点击 Tokens——AddToken，粘贴复制的合约地址，点击 Add，将上面复制得到的合约地址加入，这样也实现了部署，这个这个页面操作更强。



自此，一个简单的 token 合约部署完成，但是这样的一个 token 合约是非常简单的，也存在着许多的缺点，明显不适用于我们实际的生活：

举个例子：我想买 N 个 MTC token，需要给 creator 转一定量的 ether，或者用支付宝转一定的 RMB 给他，他再往我的账户地址上转 N 个 token——流通效率非常低。而且这其中还有不可避免地信任问题：我转了 RMB 给他，他却没有给我 token，或者少给了 token。

为此我们希望能改善这种 token 的发布方式，提供来解决这些问题：

有两种思路可以解决以上问题：

1. 交易所，creator 把一定量的 token approve 给交易所的账户，有交易所进行 token 的售卖。
2. 走类似 Bancor 的思路，把 token 的发行逻辑放到合约代码里，将代码开源，接受大家监督。

为此我们改进了上述代码：接下来分模块去讲解一个接口化、自动化、可众筹、可升级的 token，也是合约看起来更加的结构化，自动化。

1. 首先把常用的 modifier、函数提取到一个 Util 中，以备重用。

```
pragma solidity ^0.4.18;

contract Utils {
    function Utils() public{
    }
    modifier greaterThanZero(uint256 _amount) {
        require(_amount > 0);
        _;
    }
    modifier validAddress(address _address) {
        require(_address != 0x0);
        _;
    }
    modifier notThis(address _address) {
        require(_address != address(this));
        _;
    }
    function safeAdd(uint256 _x, uint256 _y) internal pure returns (uint256) {
        uint256 z = _x + _y;
        assert(z >= _x);
        return z;
    }
    function safeSub(uint256 _x, uint256 _y) internal pure returns (uint256) {
        assert(_x >= _y);
        return _x - _y;
    }
    function safeMul(uint256 _x, uint256 _y) internal pure returns (uint256) {
        uint256 z = _x * _y;
        assert(_x == 0 || z / _x == _y);
        return z;
    }
}
```

2. 接下来是提取 ERC20 接口，实现一个通用的 ERC20 基类

```
pragma solidity ^0.4.18;
contract ERC20Token{
    string public name = '';
    string public symbol = '';
    uint8 public decimals = 0;
    uint256 public totalSupply = 0;
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);

    function ERC20Token(string _name, string _symbol, uint8 _decimals) public{
        //检验参数的合法性
        require(bytes(_name).length > 0 && bytes(_symbol).length > 0);

        name = _name;
        symbol = _symbol;
        decimals = _decimals;
    }

    function transfer(address _to, uint256 _value)public
        validAddress(_to)
        returns (bool success)
    {
        balanceOf[msg.sender] = safeSub(balanceOf[msg.sender], _value);
        balanceOf[_to] = safeAdd(balanceOf[_to], _value);
        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    function transferFrom(address _from, address _to, uint256 _value)
        public
        validAddress(_from)
        validAddress(_to)
        returns (bool success)
    {
        allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
        balanceOf[_from] = safeSub(balanceOf[_from], _value);
        balanceOf[_to] = safeAdd(balanceOf[_to], _value);
        emit Transfer(_from, _to, _value);
        return true;
    }

    function approve(address _spender, uint256 _value)
        public
        validAddress(_spender)
        returns (bool success)
    {
        require(_value == 0 || allowance[msg.sender][_spender] == 0);

        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }
```

3. 然后引入 owner 与 owned 两个概念，也就是控制合约与数据合约的概念。数据合约用来存放用户余额等信息，一经发布便不能再变动，所以数据合约的代码要慎之又慎。控制合约用来处理代币的集体逻辑，例如众筹，根据逻辑操纵数据合约。

首先是 owned 合约

```
//一般来说，数据合约要继承这个基类
contract Owned{
    address public owner;
    address public newOwner;

    event OwnerUpdate(address _prevOwner, address _newOwner);

    function Owned() public{
        owner = msg.sender;
    }

    modifier ownerOnly {
        assert(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public ownerOnly {
        require(_newOwner != owner);
        newOwner = _newOwner;
    }

    function acceptOwnership() public {
        require(msg.sender == newOwner);
        owner = newOwner;
        newOwner = 0x0;
        emit OwnerUpdate(owner, newOwner);
    }
}
```

接着是 owner 合约

```
//一般来说，控制合约要继承这个基类
contract Owner{
    address public creator;
    address public ownedDataContract;
    Owned   public dataContract;

    //_dataContract must be specified when creating the owner contract.
    function Owner(Owned _dataContract) public{
        assert(address(_dataContract) != address(0));
        creator = msg.sender;
        dataContract = _dataContract;
    }

    modifier creatorOnly{
        assert(msg.sender == creator);
        _;
    }

    function transferTokenOwnership(address _newOwner) public creatorOnly {
        dataContract.transferOwnership(_newOwner);
    }

    function acceptTokenOwnership() public creatorOnly {
        dataContract.acceptOwnership();
    }
}
```

4. 接下来引入一个 SmartContract 基类，用来处理 token 的发行与销毁。

```
contract SmartToken is Owned, ERC20Token {
    event NewSmartToken(address _token);
    event Issuance(uint256 _amount);
    event Destruction(uint256 _amount);

    function SmartToken(string _name, string _symbol, uint8 _decimals)
        ERC20Token(_name, _symbol, _decimals) public
    {
        emit NewSmartToken(address(this));
    }

    //只有数据合约的owner才有资格使用issue方法给某个账户发行一定数量的token
    function issue(address _to, uint256 _amount)
        public
        ownerOnly
        validAddress(_to)
        notThis(_to)
    {
        totalSupply = safeAdd(totalSupply, _amount);
        balanceOf[_to] = safeAdd(balanceOf[_to], _amount);

        emit Issuance(_amount);
        Transfer(this, _to, _amount);
    }

    function destroy(address _from, uint256 _amount) public {
        require(msg.sender == _from || msg.sender == owner); // validate input

        balanceOf[_from] = safeSub(balanceOf[_from], _amount);
        totalSupply = safeSub(totalSupply, _amount);

        emit Transfer(_from, this, _amount);
        emit Destruction(_amount);
    }
}
```

6. 现在基础设施已经构建完毕，我们要实现一个数据合约、一个控制合约

```
40  //架构实现之后的简单合约
41  contract MartinToken is SmartToken {
42      string public version = '0.1';
43      function MiningSharesToken()
44          SmartToken("MartinToken", "MTC", 18) public
45      {
46
47      }
48
49      function() public payable{
50
51      }
52  }
53
```

7. 实现众筹合约

```
54  contract CrowdContract is Owner, Utils{
55      address public tokenAddr;
56
57      function CrowdContract(address token) Owner(Owned(token)) public{
58          tokenAddr = token;
59      }
60      function HandleContribute(address to, uint256 amount){
61          //假设我们众筹阶段，按照1:1000的比例收取eth。假设A给此合约转账1个eth，则发行100个MTC给A账户
62          //众筹伴随着经济利益产生，相当于投资，这里做的比较浅
63          SmartToken mtToken = SmartToken (tokenAddr);
64          mtToken.issue(msg.sender, amount * 100);
65      }
66      function() public payable{
67          HandleContribute(msg.sender, msg.value);
68      }
```

这样一个完整的合约代码就已经产生了。

但是由于我自己环境操作的原因，实际上在 remix 能够正常编译的合约代码在部署的时候一直产生各种 bug，我自己费了很多时间，一直在努力在 bug 实现解决方式。由于自己一开始选择了 windows 平台，很多软件也用不了，而且能够查到的 bug 介绍比较少，在截止时间之前我没有能把改善后的合约给成功部署调用，下一次我会更努力，这次我也尽力了，花了很多时间，也走了很多弯路，这一部分只能继续努力提交最终制品。谢谢师兄师姐老师的时间，祝生活愉快！