以下内容仅代表Qtum量子链CodeFace个人观点

ICO 众筹合约代码分析

ICO 是以初始产生的数字加密货币作为投资回报的一种筹措资金的方式，它的概念源自证券界的 Initial Public Offering(IPO，首次公开发行)。相较于传统意义上的 IPO，ICO 具有可以缩短投融资链、降低投融资门槛、流动性佳、全球性投资等优势。它可以让项目在最开始阶段就面对投资者，极大降低投融资双方的投资门槛，基于数字货币特性，ICO 可以汇集全球投资者的资金。

本文将对过往知名的 ICO 项目合约代码进行分析，方便大家参考学习。

Augur

Augur 是建立在以太坊平台上的去中心化预测市场平台，也是以太坊平台上第一个众筹的应用，利用 Augur 任何人都可以为任何自己感兴趣的主题创建一个预测市场，并提供初始流动性。Augur 项目于 2015 年 8 月开启代币众筹，一共众筹到 500 万美元，代币名称为 REP。

Augur 众筹既支持比特币也支持以太，众筹合约和 Token 交易合约是分为两个部分，众筹完成后再根据众筹时的以太坊地址分配 Token。

众筹合约源代码地址：https://github.com/AugurProject/token-sale

Token **交易源代码：**https://github.com/AugurProject/augur-core/blob/develop/src/repContract.se

Augur 众筹时的 solidity 语法和现在相比有了很多变化。

```
// Funders 保存了众筹时的每一笔参与情况，amount 记录以太数量、blockNum 记录区块高度、address 记录发送者地址。
data Funders[](address, amount, blockNum)

// Funders 数组长度
data funderNum

// addrToFunderNum 记录了每个账号的记录 ID
data addrToFunderNum[]

// augur 团队的以太坊账号地址
data augurAddress

// 众筹截止时间
data deadlinedef init():
       self.augurAddress = 0xa04fc9bd2be8bcc6875d9ebb964e8f858bcc1b4f
       self.deadline = 432015// 参与众筹调用的函数 def buyRep():
       if(msg.value==0):           return(0)     if(block.number < self.deadline):
          // 将以太发送到 augur 团队的以太坊账号上
              send(self.augurAddress, msg.value)

              // 记录每一笔众筹转账记录
              self.Funders[self.funderNum].amount = msg.value
              self.Funders[self.funderNum].blockNum = block.number
              self.Funders[self.funderNum].address = tx.origin

              // 记录该用户的记录 ID
              self.addrToFunderNum[tx.origin] = self.funderNum
              self.funderNum += 1
              return(1)     else:         return(0)

 // 还有其他函数不展示了。
```

众筹完成后，augur 会统计众筹情况并将 Token 在另一个合约中进行分配。

每个用户的代币数量保存在 reputation 对象中。

```
  data reputation[2**160]
```

在 init 函数中定义代币的名称、符号、小数位、创建者和创建时间。

```
def init():
    self.controller = 0x0
    self.name = "Reputation"
    self.symbol = "REP"
    self.decimals = 18
    self.creator = msg.sender
    self.creation = block.timestamp
```

在 setSaleDistribution 函数中给众筹参与者分配代币 REP，地址和代币数量通过数组参数的形式传递进来。

```
def setSaleDistribution(addresses: address[], balances: uint256
[]):
    i = 0
    numberOfAddresses = len(addresses)
    if(numberOfAddresses != len(balances)):
        ~invalid()
    if(self.seeded or !stillCreating()):
        ~invalid()
    // 确保只有创建者才可以进行代币分配
    if(msg.sender != self.creator):
        ~invalid()

    // 遍历数组，在 reputation 中设置各个用户的代币数，同时更新代币
总数 totalSupply    while(i < numberOfAddresses):        if(!sel
f.reputation[addresses[i]] and !self.seeded):
            self.reputation[addresses[i]] = balances[i]
            self.totalSupply += balances[i]
            log(type = Transfer, 0, addresses[i], balances[i])
        i += 1
    if(self.totalSupply == 11000000 * WEI_TO_ETH):
        self.seeded = 1
    if(self.totalSupply > 11000000 * WEI_TO_ETH):
        ~invalid()    return(1: uint256)
```

用户可以通过 transfer 函数发送 REP，通过 totalSupply 函数获取 REP 总发行数，通过 balanceOf 函数获取某个用户的 REP 数量，通过 approve 函数进行委托，通过 transferFrom 函数执行委托转账，已经和 ERC20 标准很接近了。

FirstBlood

第一滴血是一个可以让玩家随时随地参与竞赛并且获得赏金的去中心化电竞平台。

第一滴血 ICO 于 2016 年 9 月 26 日开始，达成众筹额度仅耗费两分钟，共筹得约 465，313ETH，发放代币 1SF 约 7910 万枚，每枚代币成本约为 0.5 元人民币。

ICO **合约源码地址**：https://github.com/Firstbloodio/token

兑换 Token 的代码如下，第一滴血使用了安全的数学运算函数，保证不会出现溢出等漏洞。

```
function buyRecipient(address recipient, uint8 v, bytes32 r, by
tes32 s) {
        // 可以自己买 也可以替别人购买
        bytes32 hash = sha256(msg.sender);        if (ecrecover
(hash,v,r,s) != signer) throw;
        // 判断是否在设定的众筹区块高度以内
        if (block.number<startBlock || block.number>endBlock ||
 safeAdd(presaleEtherRaised,msg.value)>etherCap || halted) thro
w;
        // 根据价格计算 Token 数量
        uint tokens = safeMul(msg.value, price());
        // 更新 balances 中的记录
        balances[recipient] = safeAdd(balances[recipient], toke
ns);         // 更新代币总数
        totalSupply = safeAdd(totalSupply, tokens);
        // 更新筹集的总以太数
        presaleEtherRaised = safeAdd(presaleEtherRaised, msg.va
lue);         // 将收到的以太立即发送到创建者账号上。
        if (!founder.call.value(msg.value)()) throw;        //
打 log
        Buy(recipient, msg.value, tokens);
    }
```

当有紧急情况发生时，创建者可以调用 halt 函数，停止所有关于资金的操作，unhalt 函数用户接触这种状态。

```
  function halt() {        if (msg.sender!=founder) throw;
        halted = true;
    }    function unhalt() {        if (msg.sender!=founder) thr
ow;
        halted = false;
    }
```

allocateFounderTokens 函数用于众筹完成后给创始人分配 Token，
allocateBountyAndEcosystemTokens 函数用户生产系统中的奖励。
这些生产的 Token 实际上都是放在创始人的账户上。

```
function allocateFounderTokens() {        if (msg.sender!=founder) throw;        if (block.number <= endBlock + founderLockup) throw;        if (founderAllocated) throw;        if (!bountyAllocated || !ecosystemAllocated) throw;
        balances[founder] = safeAdd(balances[founder], presaleTokenSupply * founderAllocation / (1 ether));
        totalSupply = safeAdd(totalSupply, presaleTokenSupply * founderAllocation / (1 ether));
        founderAllocated = true;
        AllocateFounderTokens(msg.sender);
    }    function allocateBountyAndEcosystemTokens() {        if (msg.sender!=founder) throw;        if (block.number <= endBlock) throw;        if (bountyAllocated || ecosystemAllocated) throw;
        presaleTokenSupply = totalSupply;
        balances[founder] = safeAdd(balances[founder], presaleTokenSupply * ecosystemAllocation / (1 ether));
        totalSupply = safeAdd(totalSupply, presaleTokenSupply * ecosystemAllocation / (1 ether));
        balances[founder] = safeAdd(balances[founder], bountyAllocation);
        totalSupply = safeAdd(totalSupply, bountyAllocation);
        bountyAllocated = true;
        ecosystemAllocated = true;
        AllocateBountyAndEcosystemTokens(msg.sender);
    }
```

Token 交易部分，一血遵循了 ERC20 标准。

Golem

Golem 是一个去中心化的全球算力市场。Golem 于 2016 年 11 月 11 日开启众筹，约半个小时完成众筹上限 82 万 ETH，以当日 ETH71 元人民币的收盘价计算，共筹得约 5822 万元人民币，每枚代币成本约为 0.071 元人民币，代币名称为 GNT。

ICO **合约源码地址：** https://github.com/golemfactory/golem-crowdfunding

Golem ICO 众筹合约代码由 Token.sol 和 GNTAllocation.sol 两个文件组成，其中 GNTAllocation.sol 中定义了分配给 Golem 公司和开发者的 Token 数量，主要部分的合约写在 Token.sol 中。

代码首先定义了 Token 的名称、符号、小数位、兑换比率、最大和最小众筹数。

```
 string public constant name = "Golem Network Token";    string
 public constant symbol = "GNT";
    uint8 public constant decimals = 18;  // 18 decimal places,
 the same as ETH.
    uint256 public constant tokenCreationRate = 1000;
    uint256 public constant tokenCreationCap = 820000 ether * t
okenCreationRate;
    uint256 public constant tokenCreationMin = 150000 ether * t
okenCreationRate;
```

然后用两个变量保存众筹开始和结束的区块高度，会在合约创建时以参数的形式传递进来

```
  uint256 public fundingStartBlock;
  uint256 public fundingEndBlock;
```

totalTokens 变量保存总 Token 发行数，balances 字典保存每个账号拥有的 Token 数量

```
uint256 totalTokens;
    mapping (address => uint256) balances;
```

funding 变量初始值设置为真，表明处在发行阶段

```
 bool public funding = true;
```

当众筹开始，用户可以向合约发送以太换取 GNT

```
  function create() payable external {        // 判断是否处在发行
阶段、是否在预设区块高度之内。
        if (!funding) throw;        if (block.number < fundingSt
artBlock) throw;        if (block.number > fundingEndBlock) thr
ow;        // 是否发送以太
        if (msg.value == 0) throw;        // 发送的以太是否超出众
筹额度
        if (msg.value > (tokenCreationCap - totalTokens) / toke
nCreationRate)            throw;
        // 根据比率计算兑换的 Token 数量
        var numTokens = msg.value * tokenCreationRate;        //
 更新 totalTokens 数值
        totalTokens += numTokens;        // 更新 balances 中该用户
的 Token 数量
```

```
        balances[msg.sender] += numTokens;          // 打 Log
        Transfer(0, msg.sender, numTokens);
    }
```

众筹成功结束后，Golem 公司和开发者将会获得 18%的 GNT，同时合约上的以太被发送到 Golem 公司的 ETH 账户上。

Golem 公司和开发者的 GNT 会先保存在 GNTAllocation 合约中，6 个月的锁定期之后才能取回。

```
  function finalize() external {          // 判断众筹是否成功
        if (!funding) throw;          if ((block.number <= funding
EndBlock ||
             totalTokens < tokenCreationMin) &&
            totalTokens < tokenCreationCap) throw;          // 将
发行状态设置为假
        funding = false;          // 计算Golem 公司和开发者应得到的
GNT 数量，并分配。
        uint256 percentOfTotal = 18;
        uint256 additionalTokens =
            totalTokens * percentOfTotal / (100 - percentOfTota
l);
        totalTokens += additionalTokens;
        balances[lockedAllocation] += additionalTokens;
        Transfer(0, lockedAllocation, additionalTokens);
        // 将众筹所得的以太发送到golem 公司账号
        if (!golemFactory.send(this.balance)) throw;
    }
```

若众筹失败，用户可以取回以太

```
 function refund() external {          // 判断众筹是否失败
        if (!funding) throw;          if (block.number <= fundingE
ndBlock) throw;          if (totalTokens >= tokenCreationMin) thr
ow;          // 减去对应的GNT 数量
        var gntValue = balances[msg.sender];          if (gntValue
 == 0) throw;
        balances[msg.sender] = 0;
        totalTokens -= gntValue;
        // 计算应返还的以太数量
        var ethValue = gntValue / tokenCreationRate;
        Refund(msg.sender, ethValue);          // 返回以太
        if (!msg.sender.send(ethValue)) throw;
    }
```

众筹成功结束后，Golem 公司和开发者将会获得 18%的 GNT，同时合约上的以太被发送到 Golem 公司的 ETH 账户上。

Golem 公司和开发者的 GNT 会先保存在 GNTAllocation 合约中，6 个月的锁定期之后才能取回。

```
// CrowdsaleController.sol

    function contributeETH()
        public
        payable
        between(startTime, endTime) // 通过装饰器可以让代码更优雅，逻辑更清晰        returns (uint256 amount)
    {        return processContribution();
    }
    function processContribution() private
        active
        etherCapNotReached(msg.value)
        validGasPrice
        returns (uint256 amount)
    {        //计算 Token 数量
        uint256 tokenAmount = computeReturn(msg.value);
        // 以太转移到指定的受益账号中
        assert(beneficiary.send(msg.value));
        // 更新 token 众筹总数
        totalEtherContributed = safeAdd(totalEtherContributed, msg.value);
        // 通过调用 token 对象的 issue 函数，更新该用户的 Token 数量
        token.issue(msg.sender, tokenAmount); // issue new funds to the contributor in the smart token
        // 通过调用 token 对象的 issue 函数，更新受益账号的 Token 数量
        token.issue(beneficiary, tokenAmount);
        // 打 log
        Contribution(msg.sender, msg.value, tokenAmount);
        return tokenAmount;
    }
```

token 是一个 SmartToken 合约对象，继承自 Owned 合约。

```
// Owned.solcontract Owned is IOwned {
    address public owner;
    address public newOwner;
    // token 对象创建时的所有者是合约的部署者。
    function Owned() {
        owner = msg.sender;
    }
    // 通过 transferOwnership 函数实现隶属关系的变化。
    function transferOwnership(address _newOwner) public ownerOnly {        require(_newOwner != owner);
        newOwner = _newOwner;
```

```
    }
}
```

在部署合约的 JavaScript 代码中完成 transferOwnership 函数的调用

```
// CrowdsaleController.jsasync function initController(account
s, activate, startTimeOverride = startTimeInProgress) {
    token = await SmartToken.new('Token1', 'TKN1', 2);
    tokenAddress = token.address;    let controller = await Tes
tCrowdsaleController.new(tokenAddress, startTime, beneficiaryAd
dress, btcsAddress, realEtherCapHash, startTimeOverride);    le
t controllerAddress = controller.address;    if (activate)
{        await token.transferOwnership(controllerAddress);
        await controller.acceptTokenOwnership();
    }    return controller;
}
```

这样，CrowdsaleController 合约就拥有了对 SmartToken 合约关键函数调用的权利。SmartToken 合约完全基于 ERC20 标准

QTUM

Qtum AAL 量子链账户抽象层，打通了比特币和以太坊生态；Qtum MPOS 量子链互惠权益证明机制，打造了可拓展性的共识机制；Qtum DGP 量子链分布式治理协议，让区块链协议无缝升级。

QTUM 量子链将于 2017 年 6 月 28 日发布测试网络，正式版也将于今年 9 月份上线，届时用户可以在使用内置的合约模版和 Dapp 商店一键在移动端发起自己的 ICO 和发行代币。


扫描或长按识别二维码
添加QTUM群秘
进QTUM 社区