实现一个可管理、增发、兑换、冻结等高级功能的代币

作者：守护平井一夫【链客】

写在前面

在上一篇：一步步教你创建自己的数字货币（代币）进行 ICO 中我们实现一个最基本功能的代币，本文将在上一遍文章的基础上，讲解如果添加更多的高级功能。

实现代币的管理者

虽然区块链是去中心化的，但是实现对代币（合约）的管理，也在许多应用中有需求，为了对代币进行管理，首先需要给合约添加一个管理者。

我们来看看如果实现，先创建一个 **owned** 合约。

```
contract owned {
    address public owner;

    function owned() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    // 实现所有权转移
    function transferOwnership(address newOwner) onlyOwner {
        owner = newOwner;
    }
}
```

这个合约重要的是加入了一个函数修改器（Function Modifiers）**onlyOwner**，函数修改器是一个合约属性，可以被继承，还能被重写。它用于在函数执行前检查某种前置条件。

关于函数修改器可进一步阅读 Solidity 教程系列 10 - 完全理解函数修改器

> 如果熟悉 Python 的同学，会发现函数修改器的作用和 Python 的装饰器很相似。

然后让代币合约继承 owned 以拥有 **onlyOwner** 修改器，代码如下：

```
contract MyToken is owned {
    function MyToken(
        uint256 initialSupply,
        string tokenName,
        uint8 decimalUnits,
        string tokenSymbol,
        address centralMinter
        ) {
        if(centralMinter != 0 ) owner = centralMinter;
    }
}
```

代币增发

实现代币增发，代币增发就如同央行印钞票一样，想必很多人都需要这样的功能。

给合约添加以下的方法：

```
function mintToken(address target, uint256 mintedAmount) onlyOwner {
        balanceOf[target] += mintedAmount;
        totalSupply += mintedAmount;
        Transfer(0, owner, mintedAmount);
        Transfer(owner, target, mintedAmount);
    }
```

注意 **onlyOwner** 修改器添加在函数末尾，这表示只有 **ower** 才能调用这用函数。

他的功能很简单，就是给指定的账户增加代币，同时增加总供应量。

资产冻结

有时为了监管的需要，需要实现冻结某些账户，冻结后，其资产仍在账户，但是不允许交易，之道解除冻结。

给合约添加以下的变量和方法（可以添加到合约的任何地方，但是建议把 mapping 加到和其他 mapping 一起，event 也是如此）：

```
mapping (address => bool) public frozenAccount;
event FrozenFunds(address target, bool frozen);

function freezeAccount(address target, bool freeze) onlyOwner {
    frozenAccount[target] = freeze;
    FrozenFunds(target, freeze);
}
```

单单以上的代码还无法冻结，需要把他加入到 transfer 函数中才能真正生效，因此修改 transfer 函数

```
function transfer(address _to, uint256 _value) {
        require(!frozenAccount[msg.sender]);
        ...
}
```

这样在转账前，对发起交易的账号做一次检查，只有不是被冻结的账号才能转账。

代币买卖（兑换）

可以自己的货币中实现代币与其他数字货币（ether 或其他 tokens）的兑换机制。有了这个功能，我们的合约就可以在一买一卖中赚利润了。

先来设置下买卖价格

```
uint256 public sellPrice;
uint256 public buyPrice;

```

```
function setPrices(uint256 newSellPrice, uint256 newBuyPrice) onlyOwner {
    sellPrice = newSellPrice;
    buyPrice = newBuyPrice;
}
```

setPrices()添加了 **onlyOwner** 修改器，注意买卖的价格单位是 wei（最小的货币单位： 1 eth = 1000000000000000000 wei)

添加来添加买卖函数:

```
function buy() payable returns (uint amount){
    amount = msg.value / buyPrice;                    // calculates the amount
    require(balanceOf[this] >= amount);               // checks if it has enough to sell
    balanceOf[msg.sender] += amount;                  // adds the amount to buyer's balance
    balanceOf[this] -= amount;                        // subtracts amount from seller's balance
    Transfer(this, msg.sender, amount);               // execute an event reflecting the change
    return amount;                                    // ends function and returns
}

function sell(uint amount) returns (uint revenue){
    require(balanceOf[msg.sender] >= amount);         // checks if the sender has enough to sell
    balanceOf[this] += amount;                        // adds the amount to owner's balance
    balanceOf[msg.sender] -= amount;                  // subtracts the amount from seller's balance
    revenue = amount * sellPrice;
    msg.sender.transfer(revenue);                     // sends ether to the seller: it's important to do this last to prevent recursion attacks
    Transfer(msg.sender, this, amount);               // executes an event reflecting on the change
    return revenue;                                   // ends function and returns
}
```

加入了买卖功能后，要求我们在创建合约时发送足够的以太币，以便合约有能力回购市面上的代币，否则合约将破产，用户没法先合约卖代币。

实现 Gas 的自动补充

以太坊中的交易时需要 gas（支付给矿工的费用，费用以 ether 来支付）。而如果用户没有以太币，只有代币的情况（或者我们想向用户隐藏以太坊的细节），就需要自动补充 gas 的功能。这个功能将使我们代币更加好用。

自动补充的逻辑是这样了，在执行交易之前，我们判断用户的余额（用来支付矿工的费用），如果用户的余额非常少（低于某个阈值时）可能影响到交易进行，合约自动售出一部分代币来补充余额，以帮助用户顺利完成交易。

先来设定余额阈值：

```
uint minBalanceForAccounts;

    function setMinBalance(uint minimumBalanceInFinney) onlyOwner {
         minBalanceForAccounts = minimumBalanceInFinney * 1 finney;
    }
```

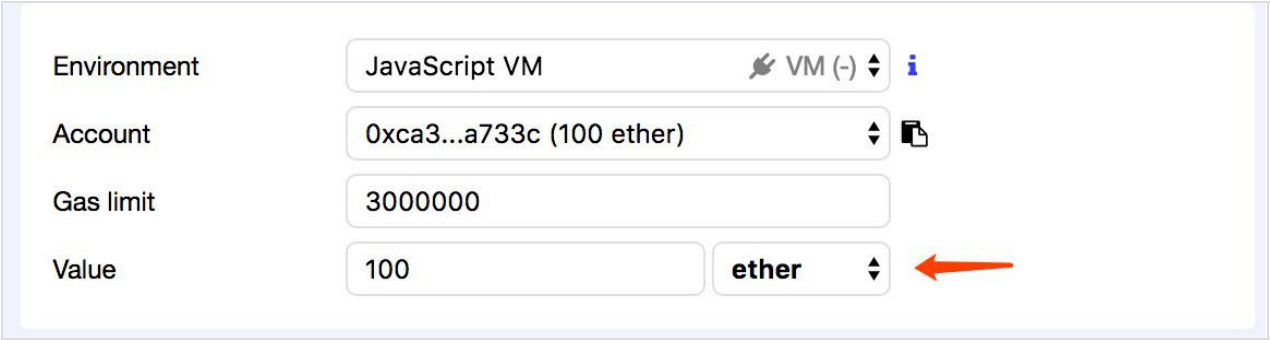
finney 是货币单位 1 finney = 0.001eth

然后交易中加入对用户的余额的判断。

```
 function transfer(address _to, uint256 _value) {
    ...
    if(msg.sender.balance < minBalanceForAccounts)
        sell((minBalanceForAccounts - msg.sender.balance) / sellPrice);
    if(_to.balance<minBalanceForAccounts)   // 可选，让接受者也补充余额，以便接受者使用代币。
        _to.send(sell((minBalanceForAccounts - _to.balance) / sellPrice));
 }
```

代码部署

高级功能完整代码请前往我的小专栏， 项目的完整的部署方法参考上一篇，不同的是创建



合约时需要预存余额，如图：

专栏已经有多篇文章介绍 Remix Solidity IDE 的使用，这里就不一一截图演示了，请大家自己测试验证。

如何创建代币发行代币，现在也录制了对应的**视频教程:通过代币学以太坊智能合约开发**，现在我们在**招募体验师**，可以点击链接了解详情。

如果你在创建代币的过程中遇到问题，欢迎到我的**知识星球**提问，作为星球成员福利，成员可加入区块链技术付费交流群。

参考文档

- Create your own crypto-currency with ethereum

我是【链客】六级算力等级《守护平井一夫》 为各位解答区块链技术问题，欢迎加入。

链客区块链技术问答社区，有问必答！！

国内域名：www.liankexing.com　　复制网址至浏览器即可进入社区

国际域名：www.lk.wiki　　QQ 群: 725414372