部署智能合约的步骤

1. 启动一个以太坊节点。
2. 使用solc编译智能合约，获得二进制代码。
3. 将编译好的合约部署到网络，获得合约的地址、ABI、bytecode，这一步会消耗以太币。
4. 用web3.js提供的JavaScript API来调用合约。

声明

我们学习的钱包开发、发币、交易所等都只是为了学习当前最前沿技术，禁止用于商业用途进行非法集资。

Token

在Web领域Token用来做身份验证，就是服务端程序每一次请求都需要验证，从而辨别客户端的身份。

那么在区块链技术中的Token是什么呢?

Token翻译有代币的意思，那么区块链中的Token就代表着用于流通的代替货币的虚拟币。如：BTC、ETC、EOS等。

在区块链中，代币指的是加密数字货币，不依靠法定货币机构发行，不受央行管控。

代币主要分为两种类型：

- 有自己区块链项目型的代币：比特币、莱特币、瑞波币、以太币等，主要用于矿工奖励和防止垃圾交易。
- 基于以太坊去中心化智能合约平台的平台型代币，市面上绝大部分属于该分类：QTUM、OmiseGo等。

EIP20 Token 标准

当前有许多基于以太坊的代币，为了让它们更加兼容，因此以太坊出了一个Token标准叫做ERC20，最近ERC20更名为了EIP20。

```
contract EIP20Interface {
    // 获取总的支持量
    uint256 public totalSupply;
```

```
    // 获取其他地址的余额
    function balanceOf(address _owner) public view returns (uint256 balance);
    // 调用者向_to地址发送_value数量的token
    function transfer(address _to, uint256 _value) public returns (bool success);
    //从_from地址向_to地址发送_value数量的token
    function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success);
    //允许_spender从自己的账户转出_value数量的token，调用多次会覆盖可用量。
    function approve(address _spender, uint256 _value) public returns (bool
success);
    // 返回_spender仍然允许从_owner获取的余额数量
    function allowance(address _owner, address _spender) public view returns
(uint256 remaining);

    // token转移完成后触发
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    // approve调用后触发
    event Approval(address indexed _owner, address indexed _spender, uint256
_value);
}
```

Token智能合约

注意以下四个状态变量必须公开，并且变量名不能变化，如decimals，不能命名为decimal。同时实现方法必须遵守EIP20 Token 标准接口，否则与其它钱包应用不能兼容。

```
uint256 public totalSupply;
uint8 public decimals;
string public name;
string public symbol;
```

在转账时一定要注意验证，一般都使用`transfer`主动发起转账，在使用`transferFrom`收账时，很容易被攻击，因为任何人都可以点用它指定`_from`和`_to`，进而可以将任何人*(from)的余额转到自己*(to)的账户。部署到主网后，任何人都可以查询到该token合约源码，因此代码一定要严谨，避免给别人有机可乘。

Token智能合约完整代码如下

```


contract EIP20Interface {
    // 获取总的支持量
    uint256 public totalSupply;
    // 获取其他地址的余额
    function balanceOf(address _owner) public view returns (uint256 balance);
```

```
    // 调用者向_to地址发送_value数量的token
    function transfer(address _to, uint256 _value) public returns (bool success);
    //从_from地址向_to地址发送_value数量的token
    function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success);
    //允许_spender从自己的账户转出_value数量的token，调用多次会覆盖可用量。
    function approve(address _spender, uint256 _value) public returns (bool
success);
    // 返回_spender仍然允许从_owner获取的余额数量
    function allowance(address _owner, address _spender) public view returns
(uint256 remaining);

    // token转移完成后触发
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    // approve调用后触发
    event Approval(address indexed _owner, address indexed _spender, uint256
_value);
}

contract MyToken is EIP20Interface {
    //注意以下四个状态变量必须公开，并且变量名不能变化，如decimals，命名为decimal。否则
与其它钱包应用不能兼容。
    uint256 public totalSupply;
    uint8 public decimals;
    string public name;
    string public symbol;

    mapping(address=>uint256) public balances;
    mapping(address=>mapping(address=>uint256)) public allowed;

    function MyToken(
        uint256 _totalSupply,
        uint8 _decimal,
        string _name,
        string _symbol) {

        totalSupply = _totalSupply;
        decimals = _decimal;
        name = _name;
        symbol = _symbol;

        balances[msg.sender] = totalSupply;
    }

    // 获取其他地址的余额
    function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }
```

```
    // 调用者向_to地址发送_value数量的token
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balances[msg.sender] >= _value && _value > 0);
        require(balances[_to] + _value > balances[_to]);
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        Transfer(msg.sender, _to, _value);
        return true;
    }

    //从_from地址向_to地址发送_value数量的token
    function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success) {
        uint256 allow = allowed[_from][_to];
        require(_to == msg.sender && allow >= _value && balances[_from] >=
_value);
        require(balances[_to] + _value > balances[_to]);
        allowed[_from][_to] -= _value;
        balances[_from] -= _value;
        balances[_to] += _value;
        Transfer(_from, _to, _value);
        return true;
    }

    //允许_spender从自己的账户转出_value数量的token，调用多次会覆盖可用量。
    function approve(address _spender, uint256 _value) public returns (bool
success) {
        require(balances[msg.sender] >= _value && _value > 0 );
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }

    // 返回_spender仍然允许从_owner获取的余额数量
    function allowance(address _owner, address _spender) public view returns
(uint256 remaining) {
        return allowed[_owner][_spender];
    }

}
```


区块链部落
专注于区块链技术
识别图中二维码关注我们