部署智能合约的多种方式

这里介绍三种，推荐使用第一种

1. Remix编译+MetaMask部署

编译器结合插件进行部署。支持主网、测试网、私网部署。

2. 使用Ethereum Wallet部署

使用Mist客户端部署。支持主网、测试网(除Kovan网络)、私网部署。

3. solc编译+web3部署

需要自己编码实现编译和部署。经测试，支持私网部署。

solc编译+web3部署Token

一、solc编译智能合约

solidity编写的以太坊智能合约可通过命令行编译工具solc来进行编译，成为以太坊虚拟机中的代码。

项目初始化

1. cd到项目跟路径
2. 使用npm初始化项目

```
npm init
```

3. 安装solc

```
npm i solc
```

4. 在根目录新建文件夹build、contract
5. 将Mytoken.sol复制到contract文件夹中

solc

所用技术：使用 `solc.compile` 方法即可编译sol文件，输出智能合约对象数组，智能合约元素中包含 `bytecode` 、 `interface` 等字段。简单用法如下

```
var solc = require('solc')
var input = 'contract x { function g() {} }'
// Setting 1 as second paramateractivates the optimiser
var output = solc.compile(input, 1)
for (var contractName in output.contracts) {
    // code and ABI that are needed by web3
    console.log(contractName + ': ' + output.contracts[contractName].bytecode)
    console.log(contractName + '; ' +
JSON.parse(output.contracts[contractName].interface))
}
```

整体流程：将MyToken.sol使用solc编译后的json文件写到build文件夹下，以合约名称命名json文件名称。

fs是nodejs的文件系统模块，fs模块中的方法一般都支持异步和同步，例如读取文件内容的函数有异步的 fs.readFile() 和同步的 fs.readFileSync()。异步的方法函数最后一个参数为函数类型，是回调函数。简单用法如下

fs

```
var fs = require("fs");
var filepath = "/../..";
```

```
// 同步读取
var data = fs.readFileSync(filepath, "utf8");
console.log("同步读取: " + data.toString());

// 异步读取
fs.readFile(filepath, "utf8", function (err, data) {
   if (err) {
       return console.error(err);
   }
   console.log("异步读取: " + data.toString());
});
```

编译

以下是compile.js完整代码

```
const solc = require('solc');
const fs = require('fs');
const path = require('path')

function main() {
    const filepath = path.join(__dirname, 'contract', 'MyToken.sol')
    console.log(filepath);
    const contractFile = fs.readFileSync(filepath, 'utf8');

    var output = solc.compile(contractFile, 1)
    console.log(output)
    const contracts = output.contracts;
    for (let contract in contracts) {
        console.log(contract)
        console.log(contract + ': ' + contracts[contract].bytecode)
        console.log(contract + '; ' + JSON.parse(contracts[contract].interface))

        let newfilepath = path.join(__dirname, 'build', contract.replace(':', '')
+ '.json');
        fs.writeFileSync(newfilepath, JSON.stringify(contracts[contract], null,
4));
    }
}

main()
```

运行命令编译

```
node compile.js
```

二、web3.js部署智能合约

web3.js介绍

web3.js是一个库集合，允许您使用HTTP或IPC连接与本地或远程以坊节点进行交互，包含以太坊生态系统的特定功能。

web3模块主要连接以太坊暴露出来的RPC层。开发者利用web3连接RPC层，可以连接任何暴露了RPC接口的节点，从而与区块链交互。

- 这`web3-eth`与以太坊区块链和智能合约之间的交互。
- 的`web3-shh`是用于协议进行通信的P2P和广播。
- 的`web3-bzz`是对于群协议，分散的文件存储。
- 它`web3-utils`包含Dapp开发人员的有用辅助函数。

github地址：web3

文档：web3

deploy.js完整源码如下

```
const fs = require('fs');
const path = require('path')
const Web3 = require('web3');
const web3 = new Web3('http://127.0.0.1:8545');

const filepath = path.resolve(__dirname,'build','kongyixueyuan.json')
let contractabi = JSON.parse(fs.readFileSync(filepath, 'utf8'));
let publisher = "0x33e41da7197A0B71Aa618b003aa5e5f629B1d46F"
let deploy = () => {
    let contract = new web3.eth.Contract(JSON.parse(contractabi.interface));

    contract.deploy({
        data: '0x' + contractabi.bytecode,
        arguments: [1000000000000000, "Tether", 5, "USDT"] // 合约构造函数参数
    })
    .send({
        from:publisher,
        gas: 1500000,
        gasPrice: web3.utils.toWei("0.0000002", "ether")
    })
    .then(async d => {
        if(d.options.address){
            console.log("部署智能合约成功,合约地址:" + d.options.address);

            d.methods.name().call().then(r => {
                console.log(r);
            });
```

```
            var name = await d.methods.name().call()
            var decimals = await d.methods.decimals().call()
            var symbol = await d.methods.symbol().call()
            var totalSupply = await d.methods.totalSupply().call()
            let params = [name, symbol, decimals, totalSupply]
            console.log("params:"+params)
        }
    }).catch(console.log)

    console.log("等待矿工确认")
}

deploy();
```

部署到私链

1. 安装web3.js

```
npm i web3
```

2. 启动私链节点

```
geth --datadir ~/privatechain/data0 --networkid 110  --rpc console
```

3. 运行脚本

```
node deploy.js
```

此时会报错如下

```
Error: Returned error: authentication needed: password or unlock
```

因为部署脚本需要解锁账号去认证。

4. 解锁账号

```
personal.unlockAccount("0x33e41da7197A0B71Aa618b003aa5e5f629B1d46F");
```

会提示输入密码，验证成功会返回 `true` 。

如下

```
> personal.unlockAccount("0x33e41da7197A0B71Aa618b003aa5e5f629B1d46F");
Unlock account 0x33e41da7197A0B71Aa618b003aa5e5f629B1d46F
Passphrase:
true
>
```

5. 再运行脚本后会处于等待状态

因为任务将会到提交到交易池中，可通过命令`txpool.status`查看是否有一个pending任务未处理，但是需要通过挖矿确认部署的合约，运行如下命令

```
miner.start(1);admin.sleepBlocks(1);miner.stop();
```

6. 部署成功会输出如下

```
$ node deploy.js
等待矿工确认
部署智能合约成功,合约地址:0xe4c8a74a63413e0B3c9fb2aE7C35538f75944a8E
Tether
params:Tether,USDT,5,1000000000000000
```