================================================================================

查询交易详情：

eth.getTransaction("Tx hash?")

================================================================================

**交易无法返回结果给web3.js.** 注意这一点！refundTicket函数会返回一个布尔值，但是这在测试中无法检查。因为这个方法是一个交易函数（会改变合约内数据或是发送以太币的调用），而web3.js得到的交易运行结果是一个交易哈希（如果打印出来是一个长长的十六进制/怪怪的字符串）。既然如此为什么还要让refundTicket返回一个值？因为在Solidity合约内可以读到这个返回值，例如当另一个合约调用refundTicket()的时候。也就是说Solidity合约可以读取交易运行的返回值，而web3.js不行。另一方面，在web3.js中你可以用事件机制（Event, 下文会解释）来监控交易运行，而合约不行。合约也无法通过call()来检查交易是否修改了合约内变量的值。

**关于sendTransaction().** 当你通过web3.js调用类似buyTicket()或者refundTicket()的交易函数时（使用web3.eth.sendTransaction），交易并不会立即执行。事实上交易会被提交到矿工网络中，交易代码直到其中一位矿工产生一个新区块把交易记录进区块链之后才执行。因此你必须等交易进入区块链并且同步回本地节点之后才能验证交易执行的结果。用testrpc的时候可能看上去是实时的，因为测试环境很快，但是正式网络会比较慢。

**事件/Event.**在web3.js中你应该监听 事件而不是返回值。我们的智能合约示例定义了这些事件：

<code class="hljs">event Deposit(address \_from, uint \_amount);

event Refund(address \_to, uint \_amount);</code>

它们在buyTicket()和refundTicket()中被触发。触发时你可以在testrpc的输出中看到日志。要监听事件，你可以使用**web.js监听器(listener)**。在写本文时我还不能在truffle测试中记录事件，但是在应用中没问题：

Conference.new({ from: accounts[0] }).then(

function(conference) {

var event = conference.allEvents().watch({}, ''); // or use conference.Deposit() or .Refund()

event.watch(function (error, result) {

if (error) {

console.log("Error: " + error);

} else {

console.log("Event: " + result.event);

}

});

// ...

**过滤器/Filter.** 监听所有事件可能会产生大量的轮询，作为替代可以使用过滤器。它们可以更灵活的开始或是停止对事件的监听。更多过滤器的信息可查看 Solidity文档。

总的来说，使用事件和过滤器的组合比检查变量消耗的Gas更少，因而在验证正式网络的交易运行结果时非常有用。

================================================================================

2022-01-12

web3调试geth私链，配置关键：

“chainid”! 还必须字符串转hex！

-------------------------------------------------------------------------------------------------------------------------------------------

pending: {

0xc12F998223A0b77596519FbA4b1De3bd22Af34be: {

0: "0x6Eba4e39C30164E252938dCED7D82818C635d0Ab: 0 wei + 3141560 gas × **0 we**i"

}

如上的0wei交易也能成功！而且地址来源不是发起者地址！但是转合约代币交易成功完成！

为什么：

1、地址都不是发起者的，怎么签署通过链验证？（子地址概念实现？）

2、交易费0，怎么被挖出来的？（实际有费用但是没有展示出来？合约自己代币交易相当于Call？）

、这个问题的解决就是chainid转hex。但是交易完成的原因还是未知！

================================================================================

关于data参数，可以使用web3.js函数encodeABI()，把contract对象中的智能合约**函数**转换为十六进制表示。如下所示：

const data = contract.methods.myFunc(param1……).encodeABI()

至此我们已经完成了准备工作，接下来就要执行调用了。

================================================================================

、gas可以实时从当前链网上获取：

var gasPrice= **web3.eth.getGasPrice()**

var gasEstimate= contract.myMethod.**estimateGas**(arg1, arg2, **{from:account }** );

、交易的js代码：

const contract = new web3.eth.Contract(contractABI, contractAddress)

web3.eth.getTransactionCount(account1, (err, txCount) => {

// 创建交易对象

const txObject = {

nonce: web3.utils.toHex(txCount),

gasLimit: web3.utils.toHex(8000000),

gasPrice: web3.utils.toHex(web3.utils.toWei('10', 'gwei')),

to: contractAddress,

data: contract.methods.myFunc(param1,param2……).encodeABI()

}

// 签署交易

const tx = new Tx(txObject, { chain: 'ropsten', hardfork: 'petersburg' })

tx.sign(privateKey1)

const serializedTx = tx.serialize()

const raw = '0x' + serializedTx.toString('hex')

// 广播交易

web3.eth.sendSignedTransaction(raw, (err, txHash) => {

console.log('txHash:', txHash)

// 可以去ropsten.etherscan.io查看交易详情

})

})

签署可以不写什么网络和硬分叉：

var tx = new **Tx**(rawTx);

tx.**sign**(privateKey);

var serializedTx = tx.**serialize**();

、

=================================================================================

web3.fromWei(eth.getBalance(eth.accounts[0]),'ether')

==============================================================================

Web调用合约函数传入的附加属性：

**address** - String: The address where the contract is deployed. See options.address.

**jsonInterface** - Array: The json interface of the contract. See options.jsonInterface.

**data** - String: The byte code of the contract. Used when the contract gets deployed.

**from** - String: The address transactions should be made from.

**gasPrice** - String: The gas price in wei to use for transactions.

**gas** - Number: The maximum gas provided for a transaction (gas limit).

**handleRevert** - Boolean: It will otherwise use the default value provided from the Eth module. See handleRevert.

**transactionBlockTimeout** - Number: It will otherwise use the default value provided from the Eth module. See transactionBlockTimeout.

**transactionConfirmationBlocks** - Number: It will otherwise use the default value provided from the Eth module. See transactionConfirmationBlocks.

**transactionPollingTimeout** - Number: It will otherwise use the default value provided from the Eth module. See transactionPollingTimeout.

**chain** - Number: It will otherwise use the default value provided from the Eth module. See defaultChain.

**hardfork** - Number: It will otherwise use the default value provided from the Eth module. See defaultHardfork.

**common** - Number: It will otherwise use the default value provided from the Eth module. See defaultCommon.

==============================================================================

vue安装（Nodejs、VScode、web3.0等已经安装）：

npm install --global vue-cli ;安装vue2.x版本

npm install -g @vue/cli ;安装vue3 版本

创建vue项目：

新建一个空文件夹，进入后，用cmd：

vue create （项目名：showKey etc.）

============================================================================

nodejs可以通过nvm安装多个版本：

NVM（节点版本管理器）是一个bash脚本，可让您基于每个用户管理多个Node.js版本。使用NVM，您可以安装和卸载要使用或测试的任何Node.js版本。

访问nvm GitHub存储库页面并复制curlor wget命令以下载并安装nvm脚本：

**curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash**

不要使用sudo它，因为它将nvm为root用户启用。

该脚本会将项目的存储库从Github复制到~/.nvm目录：

=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM\_DIR="$HOME/.nvm"

[ -s "$NVM\_DIR/nvm.sh" ] && \. "$NVM\_DIR/nvm.sh" # This loads nvm

[ -s "$NVM\_DIR/bash\_completion" ] && \. "$NVM\_DIR/bash\_completion" # This loads nvm bash\_completion

如上输出，你要么关闭并重新打开终端或运行命令的路径添加到nvm脚本到当前的shell会话。您可以做对自己更轻松的事情。

（linux下需要新开CMD窗口，否则旧窗口内没有新内容提示）

将脚本放入您的中后PATH，nvm输入以下命令来验证该脚本是否已正确安装：

nvm --version

输出：

0.35.3

要获取可以通过安装的所有Node.js版本的列表nvm，请运行：

nvm list-remote

该命令将显示所有可用Node.js版本的庞大列表。

要安装Node.js的最新可用版本，请运行：

nvm install node

如果要**更改当前活动的版本**，请输入：

nvm use 12.16.3

输出：

Now using node v12.16.3 (npm v6.14.4)

要更改默认的Node.js版本，请运行以下命令：

nvm alias default 12.16.3

有关如何使用nvm脚本的更多详细信息，请访问项目的GitHub页面。

============================================================================

yarn安装：

nodejs >= 16.10，运行命令：

$ corepack enable

（windows系统需要用admin权限启动终端才能有权限运行。）

nodejs < 16.10，需要单独安装corpack：

$ npm i -g corepack

然后，运行yarn命令就会自动安装yarn。

C:\Windows\system32>yarn

yarn install v1.22.15