**以太坊区块链积分系统示例讲解**

置顶 2018年09月06日 16:27:52 [Anyanyamy](https://me.csdn.net/u011680118) 阅读数：2072

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u011680118/article/details/82462331

[一、背景及基础知识（只需了解即可，也可不看）](https://blog.csdn.net/u011680118/article/details/82462331#%E4%B8%80%E3%80%81%E8%83%8C%E6%99%AF%E5%8F%8A%E5%9F%BA%E7%A1%80%E7%9F%A5%E8%AF%86)

[1. ES6](https://blog.csdn.net/u011680118/article/details/82462331#1.%20ES6)

[2. babel转码器](https://blog.csdn.net/u011680118/article/details/82462331#2.%20babel%E8%BD%AC%E7%A0%81%E5%99%A8)

[3. ESLint](https://blog.csdn.net/u011680118/article/details/82462331#3.%20ESLint)

[二、项目需求及项目结构图](https://blog.csdn.net/u011680118/article/details/82462331#%E4%BA%8C%E3%80%81%E9%A1%B9%E7%9B%AE%E9%9C%80%E6%B1%82%E5%8F%8A%E9%A1%B9%E7%9B%AE%E7%BB%93%E6%9E%84%E5%9B%BE)

[三、源码分析](https://blog.csdn.net/u011680118/article/details/82462331#%E4%B8%89%E3%80%81%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90)

[1. 文件简要解释（弄懂每个文件大概做什么的即可）](https://blog.csdn.net/u011680118/article/details/82462331#2.%20%E6%96%87%E4%BB%B6%E7%AE%80%E8%A6%81%E8%A7%A3%E9%87%8A)

[2. app.js文件注释（重点）](https://blog.csdn.net/u011680118/article/details/82462331#2.%20app.js%E6%96%87%E4%BB%B6%E6%B3%A8%E9%87%8A%EF%BC%88%E9%87%8D%E7%82%B9%EF%BC%89)

[四、试运行项目](https://blog.csdn.net/u011680118/article/details/82462331#%E5%9B%9B%E3%80%81%E8%AF%95%E8%BF%90%E8%A1%8C%E9%A1%B9%E7%9B%AE)

[五、测试合约功能](https://blog.csdn.net/u011680118/article/details/82462331#%E4%BA%94%E3%80%81%E6%B5%8B%E8%AF%95%E5%90%88%E7%BA%A6%E5%8A%9F%E8%83%BD)

**前言**

这个示例是[《区块链技术进阶与实践》](https://www.baidu.com/s?wd=%E3%80%8A%E5%8C%BA%E5%9D%97%E9%93%BE%E6%8A%80%E6%9C%AF%E8%BF%9B%E9%98%B6%E4%B8%8E%E5%AE%9E%E8%B7%B5%E3%80%8B&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)提供的例子，书不错。

**前提**：在学习这个例子前要安装好ganache、truffle等区块链开发环境，对智能合约基本语法和DAPP如何开发有基本概念。

**运行环境**： ganache-cli v6.1.6    truffle v4.1.13  solidity ^0.4.24

本积分系统最原始的源代码在：[https://github.com/Blockchain-book/Ethereum-Score-Hella](https://github.com/Blockchain-book/Ethereum-Score-Hella" \t "_blank)

**建议下载**本人注释后的源代码：<https://download.csdn.net/download/u011680118/10649989>

因为原始的源代码可能过时，编译会报错，我自己进行过修改。

**注意**：下载之后如果直接truffle migrate会报错



把build文件夹删掉，再在本地重新进行编译部署即可。

**附注**：这个项目基于truffle的例子webpack，可以新建一个文件夹，cd到文件夹内，在命令行中输入 truffle unbox webpack，下载并学习这个例子，与本积分例子对比。例子中的前台JavaScript代码很值得学习。

**一、背景及基础知识（只需了解即可，也可不看）**

**1. ES6**

[ECMAScript](https://www.baidu.com/s?wd=ECMAScript&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd) 6.0（以下简称 ES6）是 JavaScript 语言的下一代标准。

ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现。

ES6 既是一个历史名词，也是一个泛指，含义是 5.1 版以后的 JavaScript 的下一代标准，涵盖了 ES2015、ES2016、ES2017 等等

来自 <<http://es6.ruanyifeng.com/#docs/intro>>

**2. babel转码器**

[Babel](https://babeljs.io/) 是一个广泛使用的 ES6 转码器，可以将 ES6 代码转为 ES5 代码，从而在现有环境执行。这意味着，你可以用 ES6 的方式编写程序，又不用担心现有环境是否支持。

Babel 的配置文件是.babelrc，存放在项目的根目录下。使用 Babel 的第一步，就是配置这个文件。

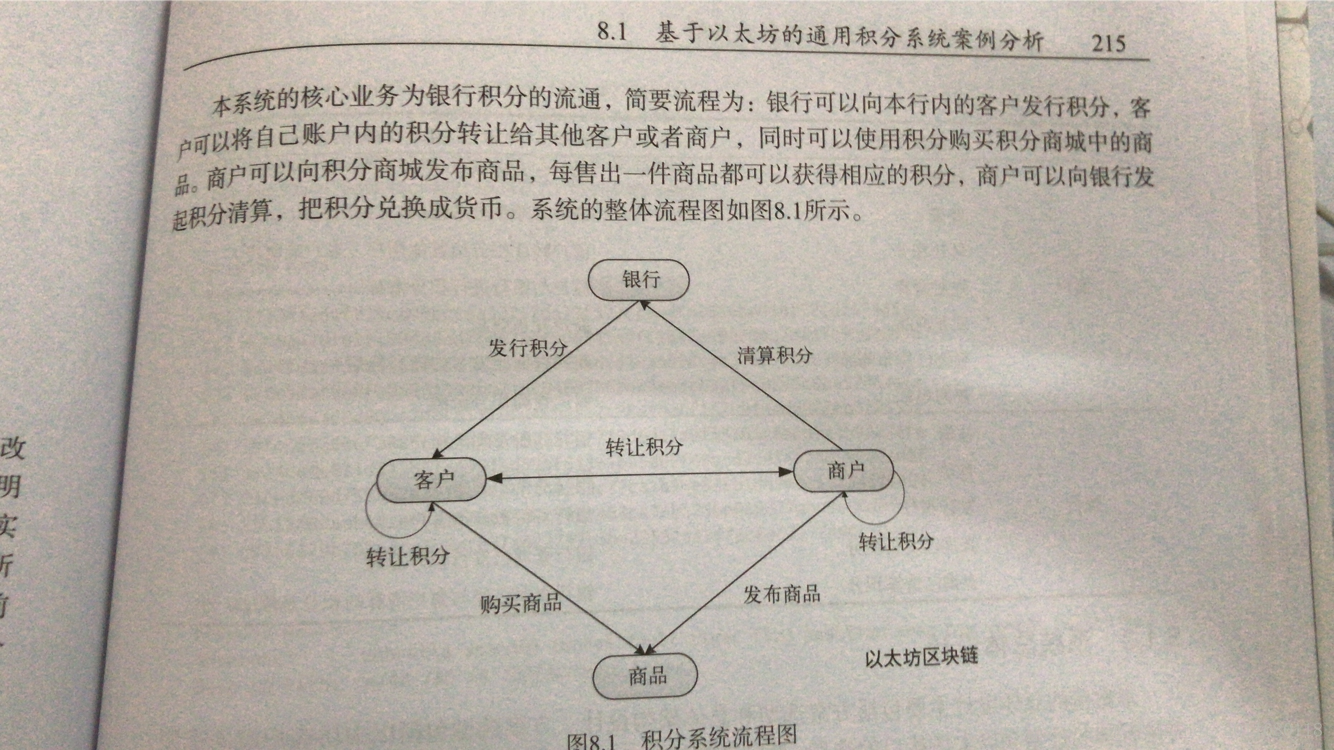


**3. ESLint**

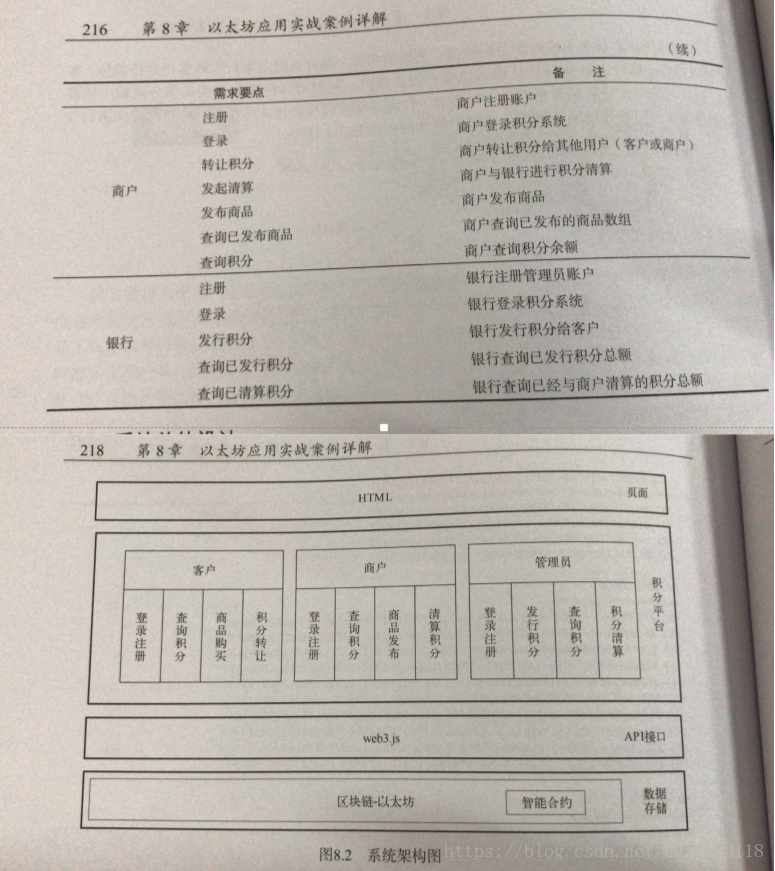


**二、项目需求及项目结构图**

      这个系统有客户、银行、商户组成，进行积分的交易，具体需求如下图：



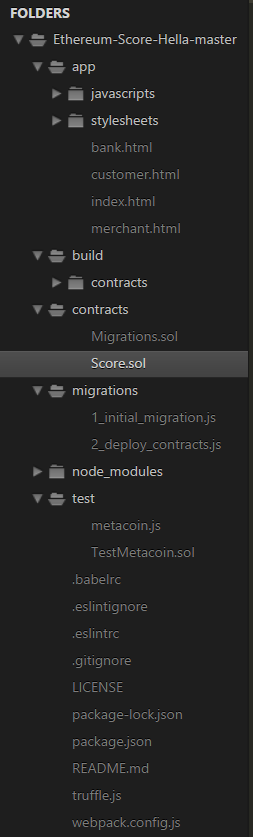




**三、源码分析**

**1. 文件简要解释（弄懂每个文件大概做什么的即可）**

用sublime text打开文件夹，可以看到组织目录



**app**是前端html代码和一些脚本程序

**stylesheets**里面是**app.css**，设定网页各个部件的样式

**html**是前台代码，**index.html**里面有客户注册、登陆、商户注册、登陆、银行登陆和测试功能。

    客户登陆后页面跳转到**customer.html**，页面链接中有当前登陆账户的地址，可以实现查询信息、购买商品、转让积分等功能

    商户登陆后页面跳转到**merchant.html**，页面链接中有当前登陆账户的地址，可以实现查询信息、添加商品、转让积分、积分清算等功能

    银行登陆后页面跳转到**bank.html**，页面链接中有当前登陆账户的地址，可以实现发行积分、查看信息等功能

    JavaScript代码的解释**见代码注释**，主要用到**web3**来与节点通信，获取节点的账户列表等信息，用到**truffle-constract** 来生成合约实例，调用合约的方法。这两个js库都可以在项目目录下 **npm** **install** 进行下载，会根据package.json确定版本

**build** 是在compile之后生成的文件夹

**contracts、migrations、test** 是truffle自动生成的文件夹

**node\_modules**是前端需要的一些依赖文件，在下载web3.js之后有的

**以点号开头的文件**都是用于JavaScript的转码，可以暂时不管

**package.json** 前端开发，测试需要依赖的一些库的配置信息，这个文件是 npm init 时创建的一个文件，会记录当前整个项目中的一些基础信息。package.json里面定义的是版本范围（比如^1.0.0），具体跑npm install的时候安的什么版本，要解析后才能决定，这里面定义的依赖关系树，可以称之为逻辑树（logical tree）。

**package-lock.json** 这个文件却是 node\_modules 文件夹或者 package.json 文件发生变化时自动生成的。这个文件主要功能是确定当前安装的包的依赖，以便后续重新安装的时候生成相同的依赖，而忽略项目开发过程中有些依赖已经发生的更新。

来自 <<http://www.cnblogs.com/tianzun-blog/p/9275931.html>>

node\_modules文件夹下才是npm实际安装的确定版本的东西，这里面的文件夹结构我们可以称之为物理树（physical tree）。

安装过程中有一些去重算法，所以你会发现逻辑树结构和物理树结构不完全一样。

package-lock.json可以理解成对结合了逻辑树和物理树的一个快照（snapshot），里面有明确的各依赖版本号，实际安装的结构，也有逻辑树的结构。

其最大的好处就是能获得可重复的构建（repeatable build），当你在CI（持续集成）上重复build的时候，得到的artifact是一样的，因为依赖的版本都被锁住了。

 链接：https://www.zhihu.com/question/62331583/answer/257972185

**truffle.js**如下：

1. // Allows us to use ES6 in our migrations and tests.
2. require('babel-register')
4. module.exports = {
5. networks: {
6. truffle: {
7. host: '127.0.0.1',
8. port: 9545,
9. network\_id: '\*' // Match any network id
10. },
11. develop: {
12. host: '127.0.0.1',
13. port: 8545,
14. network\_id: '\*' // Match any network id
15. }
16. }
17. }

* require: 用于加载babel模块，实现JavaScript的转码，以兼容。
* networks: 指定在移植(Migration)时使用哪个网络。当在某个特定的网络上编译或运行移植时，合约会缓存起来方便后续使用。这里开发使用的是本地的ganache-cli客户端。

**webpack.config.js**

webpack在执行时，除在命令行传入参数，还可以通过指定的配置文件来执行。默认会搜索当前目录下webpack.config.js。这个文件是一个node.js模块，返回一个json格式的配置对象，或者通过--config选项来指定配置文件。

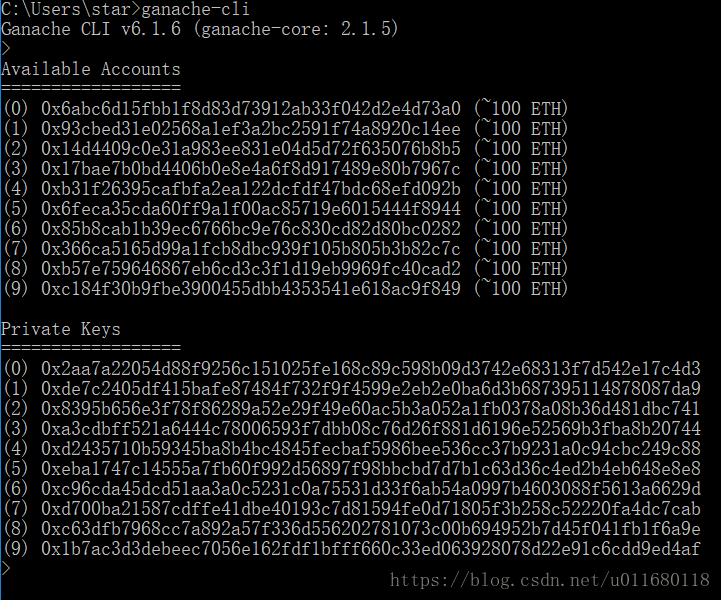
来自 <<https://www.cnblogs.com/zuozuo-blog/p/6560485.html>>

**2. app.js文件注释（重点）**

1. // Import the page's CSS. Webpack will know what to do with it.
2. import '../stylesheets/app.css'
4. //在node中使用babel支持ES6，也仅仅是将ES6转码为ES5再执行，import语法会被转码为require
6. /\*引入customer.js等模块，以使用其中的方法
7. require相当于module.exports的传送门，module.exports后面的内容是什么，require的结果就是什么
8. customer.js等模块中exports的都是方法，可以在这里被调用，如 customer.newCustomer(ScoreInstance, account) \*/
9. const customer = require('./customer')
10. const bank = require('./bank')
11. const merchant = require('./merchant')
13. // Import libraries we need.
14. /\* import是编译时的（require是运行时的），它必须放在文件开头，而且使用格式也是确定的。
15. 它不会将整个模块运行后赋值给某个变量，而是只选择import的接口进行编译，这样在性能上比require好很多。\*/
16. import { default as Web3 } from 'web3' //等同于 import Web3 from 'web3';
17. import { default as contract } from 'truffle-contract' //等同于 import contract from 'truffle-contract';
19. // Import our contract artifacts and turn them into usable abstractions.
20. //导入智能合约编译之后生成的abi的json文件 Score.json
21. import ScoreArtifacts from '../../build/contracts/Score'
23. // ScoreContract is our usable abstraction, which we'll use through the code below.
24. //初始化智能合约，实际上就是为你的智能合约创建一个对应的js对象ScoreContract，方便后续调用
25. //The input to the contract function is a JSON blob defined by truffle-contract-schema.
26. let ScoreContract = contract(ScoreArtifacts)
27. let ScoreInstance
28. // The following code is simple to show off interacting with your contracts.
29. // As your needs grow you will likely need to change its form and structure.
30. // For application bootstrapping, check out window.addEventListener below.
31. let accounts
32. let account
34. window.App = {
35. // 进行初始化
36. init: function () {
37. // 设置智能合约的web3连接，使其能与以太坊节点通信,truffle-contract中的方法
38. ScoreContract.setProvider(window.web3.currentProvider)
39. // 类似于geth客户端查询以太坊账户信息，Get the initial account balance so it can be displayed.
40. // web3.eth.getAccounts(callback(error, result){ ... })
41. window.web3.eth.getAccounts(function (err, accs) {
42. if (err != null) { //查询有问题
43. window.App.setStatus('There was an error fetching your accounts.')
44. return
45. }
47. if (accs.length === 0) { //没有以太坊账户
48. window.App.setStatus('Couldn\'t get any accounts! Make sure your Ethereum client is configured correctly.')
49. return
50. }
51. accounts = accs //把获取的以太坊账户全部存在变量accounts中，连的是ganache-cli的rpc模拟服务，预制了几个有eth的帐号
52. //如果安装了MetaMask插件，应该获得的就是MetaMask里的帐号
53. account = accounts[0] //第一个以太坊账户存在account中
54. })
56. //Create an instance of ScoreContract that represents the default address managed by ScoreContract.
57. //Creates an instance of the contract abstraction representing the contract at its deployed address.
58. // truffle-contract中的方法
59. ScoreContract.deployed().then(function (instance) {
60. ScoreInstance = instance //获取合约的实例，不用重复deploy，都用这一个
61. }).catch(function (e) {
62. console.log(e, null)
63. })
64. },
65. //===================================== 客户方法 ==========================================
66. // 新建客户
67. newCustomer: function () {
68. customer.newCustomer(ScoreInstance, account)
69. },
70. // 客户登录
71. customerLogin: function () {
72. customer.customerLogin(ScoreInstance, account)
73. },
74. // 当前客户信息，currentAccount在url中截取，在status框显示当前登陆账户
75. getCurrentCustomer: function (currentAccount) {
76. customer.showCurrentAccount(currentAccount)
77. },
78. // 当前客户余额
79. getScoreWithCustomerAddr: function (currentAccount) {
80. customer.getScoreWithCustomerAddr(currentAccount, ScoreInstance, account)
81. },
82. // 客户购买商品
83. buyGood: function (currentAccount) {
84. customer.buyGood(currentAccount, ScoreInstance, account)
85. },
86. // 查看已经购买的物品
87. getGoodsByCustomer: function (currentAccount) {
88. customer.getGoodsByCustomer(currentAccount, ScoreInstance, account)
89. },
90. // 客户转让积分
91. transferScoreToAnotherFromCustomer: function (currentAccount) {
92. customer.transferScoreToAnotherFromCustomer(currentAccount, ScoreInstance, account)
93. },
94. //===================================== 商家方法 ==========================================
95. // 商家注册
96. newMerchant: function () {
97. merchant.newMerchant(ScoreInstance, account)
98. },
99. // 商家登录
100. merchantLogin: function () {
101. merchant.merchantLogin(ScoreInstance, account)
102. },
103. // 当前商家账户
104. getCurrentMerchant: function (currentAccount) {
105. merchant.getCurrentMerchant(currentAccount)
106. },
107. // 当前商家余额
108. getScoreWithMerchantAddr: function (currentAccount) {
109. merchant.getScoreWithMerchantAddr(currentAccount, ScoreInstance, account)
110. },
111. // 商家积分转让
112. transferScoreToAnotherFromMerchant: function (currentAccount) {
113. merchant.transferScoreToAnotherFromMerchant(currentAccount, ScoreInstance, account)
114. },
115. // 商家添加商品
116. addGood: function (currentAccount) {
117. merchant.addGood(currentAccount, ScoreInstance, account)
118. },
119. // 显示商家的所有商品
120. getGoodsByMerchant: function (currentAccount) {
121. merchant.getGoodsByMerchant(currentAccount, ScoreInstance, account)
122. },
123. // 商家清算积分
124. settleScoreWithBank: function (currentAccount) {
125. merchant.settleScoreWithBank(currentAccount, ScoreInstance, account)
126. },
127. //===================================== 银行方法 ==========================================
128. // 发行积分
129. sendScoreToCustomer: function () {
130. bank.sendScoreToCustomer(ScoreInstance, account)
131. },
132. // 银行登录
133. bankLogin: function () {
134. bank.bankLogin(ScoreInstance, account)
135. },
136. // 查看已经发行的积分
137. getIssuedScoreAmount: function () {
138. bank.getIssuedScoreAmount(ScoreInstance, account)
139. },
140. // 已经清算积分总数目
141. getSettledScoreAmount: function () {
142. bank.getSettledScoreAmount(ScoreInstance, account)
143. },
144. //===================================== 调试方法 ==========================================
145. // 查询所有的区块链账户
146. allAccounts: function () {
147. let allAccount = ''
148. window.web3.eth.accounts.forEach(e => {
149. allAccount += e + '\n'
150. })
151. window.App.setConsole(allAccount)
152. },
153. // 状态栏显示
154. setStatus: function (message) {
155. const status = document.getElementById('status')
156. status.innerHTML = message
157. },
158. // 显示console
159. setConsole: function (message) {
160. const status = document.getElementById('console')
161. status.innerHTML = message
162. }
163. }
165. //在页面载入后，初始化web3，创建一个基于Http的provider，可以与以太坊节点进行通信
166. //这里用到的就是用ganache-cli启动所提供的对外的rpc服务，因为ganache-cli启动的时候绑定的是localhost，所以测试所使用的浏览器也要在本机。
167. window.addEventListener('load', function () {
168. // 设置web3连接 http://127.0.0.1:8545 即以太坊节点
169. window.web3 = new Web3(new Web3.providers.HttpProvider('http://127.0.0.1:8545'))
170. window.App.init() //调用初始化函数
171. })

**四、试运行项目**

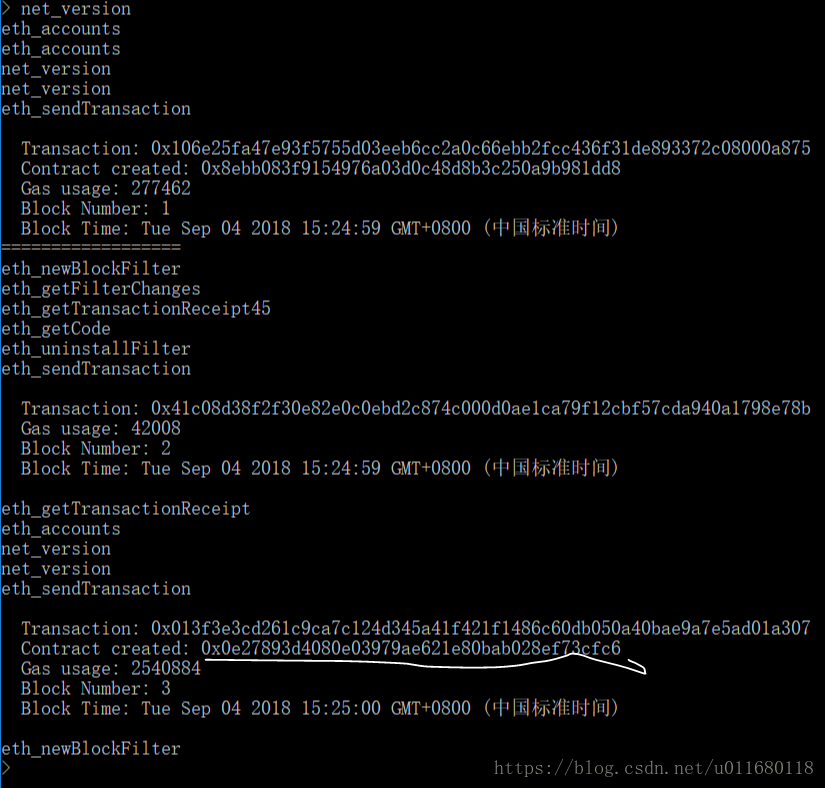
在命令行中打开ganache-cli客户端，命令为：ganache-cli  运行端口是 8545



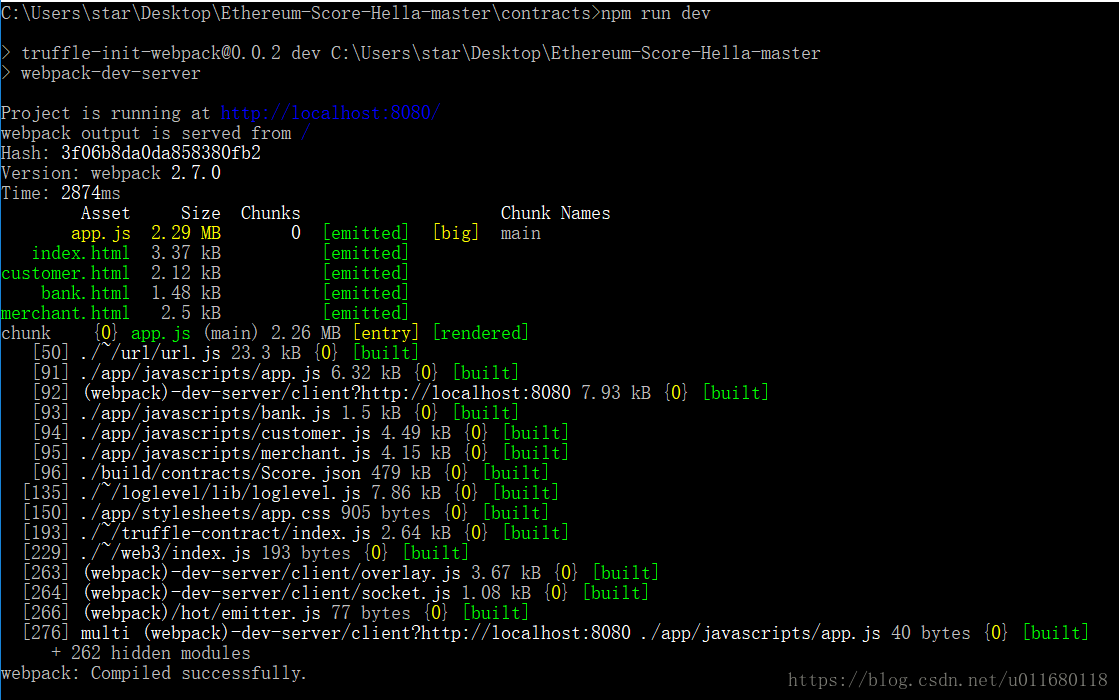
另外开一个命令行，cd到contracts目录下，输入**truffle compile** 和 **truffle migrate --network develop** 命令，把合约部署到区块链上，并使用truffle.js中配置的开发网络



可以看到Score合约地址是0x0e2789开头的。。 查看ganache-cli的命令行，其中打印出调试信息，生成的第三个区块中包含了Score合约地址，也是0x0e2789...



部署成功后，输入 **npm run dev** 命令，会执行 package.json里面的 dev 后面的脚本，即 "webpack-dev-server" 在本地开启[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)，现在可以访问客户端web页面。如果这里报错，说明要安装 webpack-dev-server ，可自行百度如何安装。



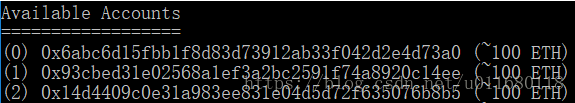
在浏览器中输入网址： http://localhost:8080



**五、测试合约功能**

**注意**：不同电脑账户不同，每次重启ganache-cli生成的账户也不同

注册客户和商户：我们就用第二个和第三个分别注册为客户和商户



商户账户 0x14d4409c0e31a983ee831e04d5d72f635076b8b5

密码 123

登录后跳转，页面链接末尾是商户地址







客户账户 0x93cbed31e02568a1ef3a2bc2591f74a8920c14ee

密码 空

注册成功后，登录之后页面跳转，客户账户在链接末尾



银行账户默认是第一个账户

https://img-blog.csdn.net/20180906161923756?watermark/2/text/aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTE2ODAxMTg=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70

输入银行地址后，密码不用管，后台没有比对密码，点登陆即可跳转到银行界面

0x6abc6d15fbb1f8d83d73912ab33f042d2e4d73a0



在命令行中打开ganache-cli客户端，命令为：ganache-cli  运行端口是 8545

另外开一个命令行，cd到contracts目录下，输入**truffle compile** 和 **truffle migrate --network develop** 命令，把合约部署到区块链上，并使用truffle.js中配置的开发网络

部署成功后，输入 **npm run dev** 命令，会执行 package.json里面的 dev 后面的脚本，即 "webpack-dev-server" 在本地开启[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)，现在可以访问客户端web页面。如果这里报错，说明要安装 webpack-dev-server ，可自行百度如何安装。

在浏览器中输入网址： http://localhost:8080

:

第一个页面：http://localhost:8080/

:

第二个页面：http://localhost:8080/console.html