

中山大学数据科学与计算机学院本科生实验报告

(2019 年秋季学期)

课程名称 : 区块链原理与技术

任课教师 : 郑子彬

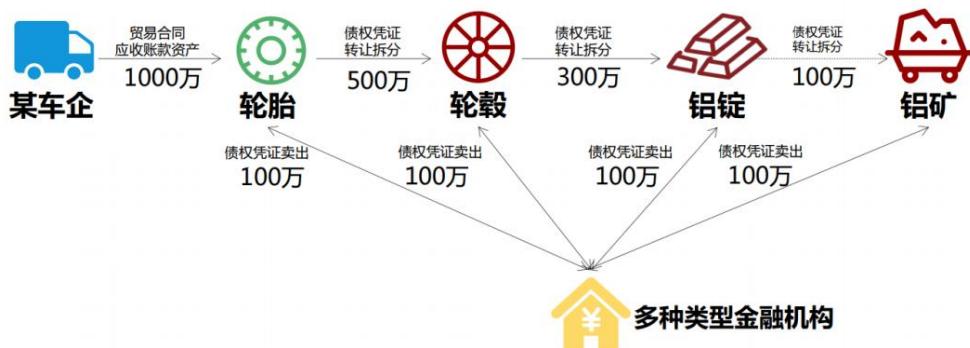
年级	2017	专业(方向)	软件工程
学号	17343006	姓名	曾峥
电话	15881447310	Email	962910588@qq.com
开始日期	2019.10.26	完成日期	2019.12.14

一、项目背景

基于已有的开源区块链系统 FISCO-BCOS(<https://github.com/FISCO-BCOS/FISCO-BCOS>),以联盟链为主,开发基于区块链或区块链智能合约的供应链金融平台,实现供应链应收账款资产的溯源、流转。



场景 :



实现功能：

功能一：实现采购商品一签发应收账款交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。

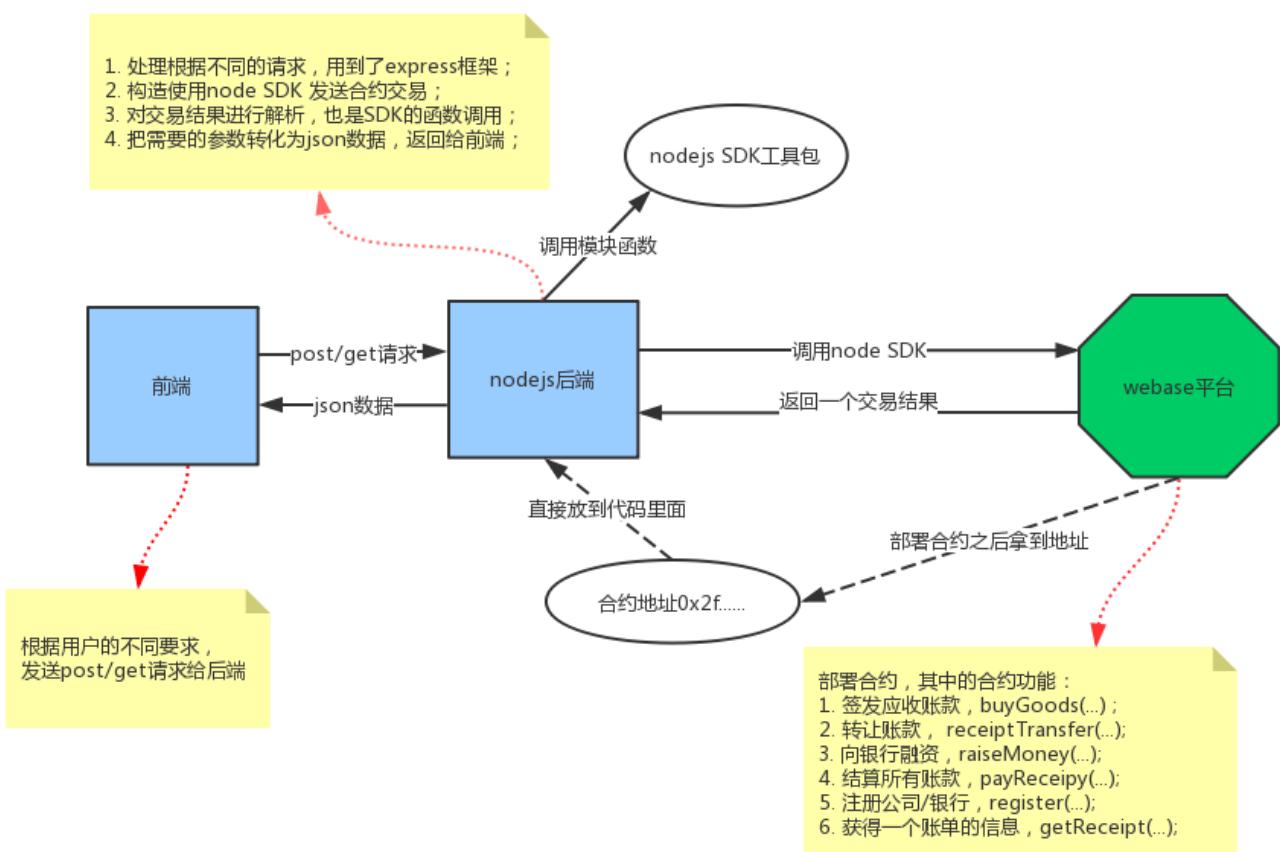
功能二：实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将与车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。

功能三：利用应收账款向银行融资上链，供应链上所有可以利用应收账款单据向银行申请融资。

功能四：应收账款支付结算上链，应收账款单据到期时核心企业向下游企业支付相应的欠款。

二、 方案设计

整个项目的逻辑如下：



接下来，说明每一个模块的设计，这包括智能合约、前端、后端。

2.1 智能合约

我的账单数据结构如下：

```
struct Receipt {
    string From;          //欠款人
    string To;            //收款人
    uint Type;           //代表账单的类型，0 买卖、1 转让、3 融资、4 已结算
    uint receiptAmount;   //账款金额
    uint BuildDate;       //创建日期
    uint dueDate;         //还款日
    string information;  //其他信息
}
```

其中 uint Type 代表账单的类型，比如正常购买物资生成的账单类型为 0

如果这个账单是有别的公司转让应收账款而来的为 1；

如果公司 B 将这个账单取银行融资，这个账单类型为 2；

如果这个账单的欠款人公司 A 已经结算了这个账单，那么这个账单类型为 3，可以理解为无效。

公司结构如下：

```
struct Organization {
    string Name;          //组织名称
    string password;      //密码
    bool Type;            //company1 or bank0
}
```

关于账单、公司的存储：

由于需要后期遍历，以及查询的遍历，我单独设置了参数，代表账单/公司的数目，然后用 map 映射这个账单/公司的编号 index 以及对应的实体 Receipt/Organization,如下所示：

```

uint O_len = 0; //组织（公司以及银行）数目
mapping(uint => string) orgas; //方便根据公司名称得到公司
mapping(string => Organization) organizations;//组织

uint R_len = 0; //账单数目
mapping(uint => Receipt) receipts; //所有账单，方便遍历

```

4个功能的算法如下：

2.1.1 生成账单

伪算法如下：

<p>任务：完成两个公司的交易，生成一个账单</p> <p>输入：两个组织、交易金额、账单的有效期限、账单额外信息；</p> <p>输出：如果交易成功输出这个编号，如果失败则返回异常；</p> <p>步骤：</p> <ol style="list-style-type: none"> 1. 判断输入组织是否都是公司； 2. 判断两个组织是否不相同 3. 创立新的账单； 4. 将新账单加入账单合集。 5. 返回这个账单的编号 index
--

代码如下：

<pre> function buyGoods(string seller, string buyer, uint money, uint daysAfter, string information) payable public returns(uint) { //buyGoods Failed! Please input company names! if (organizations[seller].Type && organizations[buyer].Type && !isEqual(buyer, seller)) { //buyGoods Success! create a new receipt Receipt memory rNew = Receipt(buyer, seller, 0, money, now, now + daysAfter * 1 minutes, information); receipts[R_len++] = rNew; return R_len - 1; } return 99999999; //buyGoods Failed! } </pre>
--

2.1.2 转账

伪算法如下：

任务：完成两个公司的应收账款转让；遍历所有符合规则的账单，转让金额 money，如果账单上的金额 amount 不够，则只转让 amount，遵循“有多少转多少”的原则。

输入：两个组织、转让金额（公司 A 以及公司 B，转让金额 money）；

输出：修改的账单数目 count 以及已经转让的金额 sum

步骤：

1. 判断两个组织是否都是公司（假设需要 A 转让 B，金额为 money）

2. 遍历所有账单，找到收款人 To 为 A 且不为 B 的账单，对于这每一个账单：

(1) 如果账单金额 amount == money：

 直接修改账单收款人 A 为 B;

 Count++, sum += money;

 结束，返回(count, sum)。

(2) 如果账单金额 amount > money：

 减少这个账单的金额为 amount-money，收款人 To 仍然是 A，账单类型 Type 仍然是 0，代表买卖账单；

 重新生成一张账单，金额为 money, 收款人 To 为 B，账单类型为 1，代表转让账单。

 Count++, sum += money;

 结束，返回(count, sum)。

(3) 如果金额 amount < money:

 这张账单的收款人改为 B，账单类型改为 1；

 Money = money - amount；

 Count ++, sum += amount

 继续寻找账单。

3. 返回 (count, sum) 这个步骤应该没什么必要，因为逻辑上在 (1)、(2) 步中会返回。

代码如下：

```

//2. transferReceipt
function transferReceipt(string company1, string company2, uint money)  public
returns(uint, uint) {
    //check if it is enough
    uint count = 0;
    uint sum = 0;
    if (!organizations[company1].Type || !organizations[company1].Type) {
        return (99999999, 0);
    }
    //transferReceipt Failed! Please input two company name!";
    for (uint i = 0; i < R_len; i++) {
        if (isEqual(receipts[i].To, company1) && !isEqual(receipts[i].To,
company2) && (receipts[i].Type == 0 || receipts[i].Type == 1)) {
            if (receipts[i].receiptAmount == money) {
                receipts[i].To = company2;
                sum += money;
                count += 1;
                return (count, sum);
            } //all
            else if (receipts[i].receiptAmount > money) {
                receipts[i].receiptAmount -= money;
                sum += money;
                count += 1;
                //new receipt
                Receipt memory rNew = Receipt(receipts[i].From, company2, 1,
money, receipts[i].BuildDate, receipts[i].dueDate, receipts[i].information);
                receipts[R_len++] = rNew;

                return (count, sum);
            } //partition
            else {
                receipts[i].To = company2;
                money -= receipts[i].receiptAmount;
                sum += receipts[i].receiptAmount;
                count += 1;
                continue;
            }
        }
    }

    if (sum == 0 || count == 0) {
        return (99999999, 0);
    }
}

```

2.1.3 融资

任务：完成一个公司 `company` 向一个银行 `Bank` 的融资，融资这个公司的所有应收账款。

输入：两个组织；

输出：修改的账单数目 `count` 以及已经融资的金额 `sum`

步骤：

1. 检查收入的两个组织是否是个公司以及银行；
2. 找到收款人 `To` 为 A、未过期、账单类型为 0/1 的账单：
 - ① 账单类型改为 2，代表已经融资；
 - ② `Count++`, `sum += amount` (这个账单的金额)
3. 返回 (`count, sum`)

代码如下：

```
//3. raiseMoney
function raiseMoney(string company, string bank)    public returns(uint,
uint) {
    uint count = 0;
    uint sum = 0;
    if (!organizations[company].Type || organizations[bank].Type) {
        return(99999999, 0);
    }//return "raiseMoney Failed! Please input right names!";

    for (uint i = 0; i < R_len; i++)
    {
        if (isEqual(receipts[i].To, company) && receipts[i].dueDate >= now
&& (receipts[i].Type == 0 || receipts[i].Type == 1)) {
            receipts[i].To = bank;
            receipts[i].Type = 2;
            sum += receipts[i].receiptAmount;
            count += 1;
        }
    }
    if (sum == 0 || count == 0) {
        return (99999999, 0);
        //transferReceipt failed! No receipt can be raised";
    }
    return (count, sum); // raiseMoney success!;
}
```

2.1.4 结算账单

任务：完成一个公司 `company` 的所有账单偿还。

输入：一个组织；

输出：修改的账单数目 `count` 以及已经偿还的金额 `sum`

步骤：

1. 检查这个组织是否为公司
2. 对于未偿还 (Type != 3)、已过期账单：
 - a) 修改 Type = 3;
 - b) Count++, sum += amount
3. 返回 (count, sum)

代码如下：

```
//4. payReceipt      //duedate
function payReceipt(string company)  public returns(uint, uint) {
    uint count = 0;
    uint sum = 0;
    if (!organizations[company].Type)
    {
        return (99999999, 0);
        //return "payReceipt Failed! Please input right names!";
    }
    //traverse, change receipt's type
    for (uint i = 0; i < R_len; i++) {
        if ((now <= receipts[i].dueDate) && (receipts[i].Type != 3) &&
isEqual(receipts[i].From, company)) {
            receipts[i].Type = 3;
            sum += receipts[i].receiptAmount;
            count += 1;
        }
    }
    if (sum == 0 || count == 0) {
        return (99999999, 0);
        //没有账单可偿还"payReceipt failed!";
    }
    else return  (count, sum);
}
```

2.1.5 额外功能

由于考虑到和一个组织的有些操作只能由对应的组织进行操作（比如说公司 A 不能对公司 B 的应收账款进行融资操作），设置了组织注册以及登陆的功能。

当一个组织使用系统进行交易时，需要：

- 1) 拥有合法的账号，即注册账号；
- 2) 登陆系统；
- 3) 操作。

在登陆系统之后，这个组织才可以进行一些操作。

1. 注册新的公司/银行

任务：完成一个组织（公司/银行）的注册

输入：注册的名称、密码、组织的类型（1 代表公司、0 代表银行）；

输出：这个组织的编号。

步骤：

1. 检查输入是否合法；
2. 检测这个组织名称是否已被注册；
3. 以上都通过，则注册账号：
 - a) 记录组织信息，系统中存储组织的数目的参数 `o_len++`
 - b) 返回参数，代表这个组织的标号。
4. 结束。

代码如下：

```

//register a company/bank, return the index of organisations
function register(string name, string password, string tpye)    public
returns(uint) {
    if (isEqual(name, "") || isEqual(password, "") || isEqual(tpye, ""))
        return 99999999;
    if (isExist(name)) {
        return 99999999; //Register Failed! This name has been used";
    }

    organizations[name].Name = name;
    organizations[name].password = password;
    if (isEqual(tpye, "1")) organizations[name].Type = true;
    else organizations[name].Type = false;
    orgas[0_len++] = name;
    return 0_len;
}

```

2. 登陆公司/银行 (用于前端)

任务: 完成一个组织 (公司/银行) 的登陆

输入: 注册的名称、密码;

输出: 一个 bool 值, 代表是否成功登陆。

步骤:

1. 检查输入是否合法;
2. 检测这个密码是否是该组织的密码;
3. 以上都通过, 则登陆成功, 返回 true, 否则返回 false;
4. 结束。

代码如下 :

```

//check the name and password for web login
function login(string name, string password) public returns(bool) {
    if (isEqual(name, "") || isEqual(password, ""))
        return false;
    if (isEqual(organizations[name].password, password))
        return true;
    else return false;
}

```

3. 查询组织

这个功能很简单，直接返回这个组织的信息（名称、组织类型）

```
//用到辅助函数getOrgName(i)，返回这个编号为i的公司名称
function getCompany(uint i) view public returns(string, bool) {
    return (getOrgName(i), organizations[getOrgName(i)].Type);
}
```

4. 查询账单

```
//输出这个账单的所有信息，
function getReceipt(uint i) view public returns(string, string, uint, uint, uint, string) {
    return (
        getFrom(i),           //欠款人
        getTo(i),             //收款人
        getType(i),           //该账单的类型
        getAmount(i),          //交易金额
        getBuildDate(i),       //交易生成时间
        getDueDate(i),         //账单截止日期
        getInformation(i));   //交易额外信息
}
```

5. 查询数目

查询已有的公司数目、所有的账单

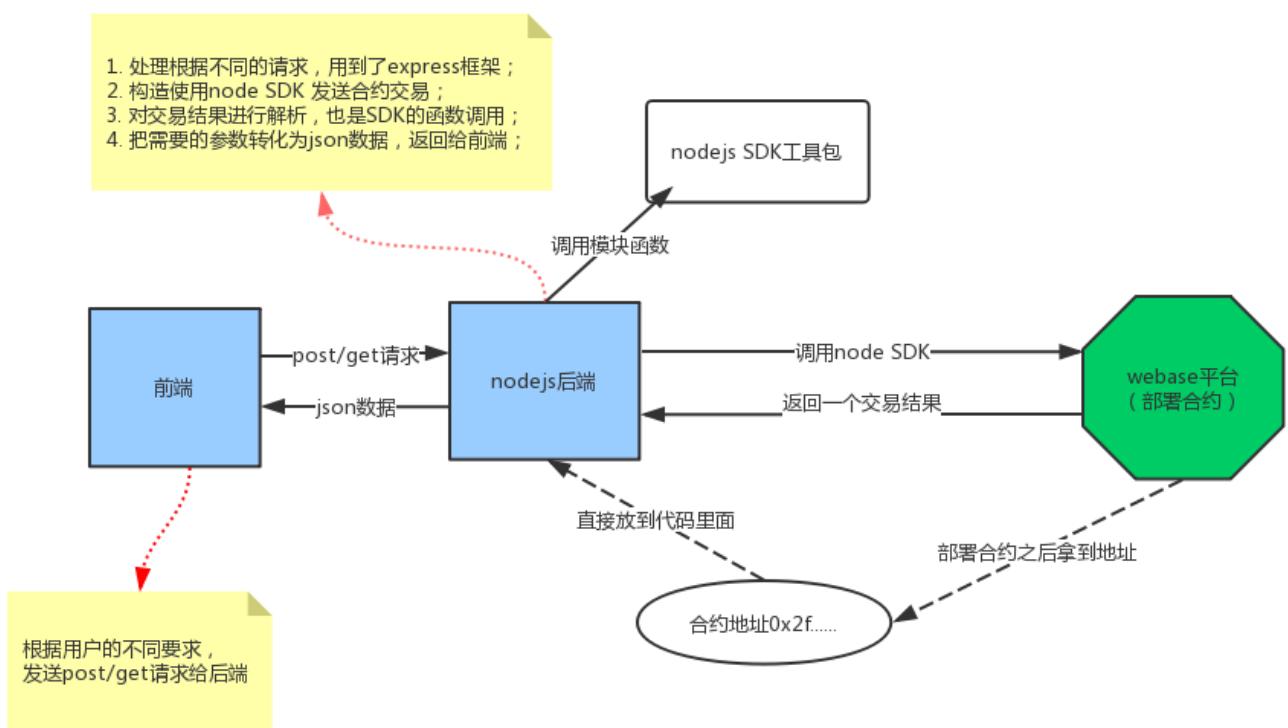
```
function getCountOs() public view returns(uint) {
    return O_len;
}
function getCountRs() public view returns(uint) {
    return R_len;
}
```

2.2 前后端

2.2.1 框架

现在写一个网页实现对这个智能合约的相关操作，形成良好的用户体验。

先梳理一下整个前端的操作，我的理解为：



- 由于我不会前端的高级操作，比如框架什么的，因此是裸写 HTML 网页、CSS 样式、js 交互，js 代码向后端发送 post/get 请求。
 - 写一个 HTML 文件、css 文件、前端 js 文件；
- 后端使用了 express 框架处理前端发送过来的 post/get 请求，然后使用实现配置好的 nodejs SDK，发送交易
 - 写一个后端 js 文件；
- Webbase 平台部署合约。

最后，整个网页设计的文件结构如下：

```
pro
├── css
│   ├── index.css
│   └── menu.css
├── images
│   ├── bg.jpg
│   └── pie_icon.gif
└── js          //前端 js 文件夹
    ├── index.js
    ├── jquery.js
    └── menu.js
├── index.html      //前端登陆/注册页面
├── menu.html       //前端功能页面
├── node_modules    //引用 nodejsSDK 时生成
├── package-lock.json
└── server.js        //后端代码
```

下面以此介绍不同模块的设计与实现：

2.2.2 前端

我设计了两个页面，分别：

- 登陆/注册页面 index.html；
- 登陆成功之后，用户进入的功能页面 menu.html，进行交易。

HTML/CSS 文件没什么好说的，主要是 js 前端操作；

伪算法：

1. 通过检测到某一个按钮的变化时，判断这个表格中内容是否合法；
2. 如果合法，通过 ajax 提交表单到后端；
3. 得到后端返回的结果，进行一些判断；
4. 输出到页面上。

比如，在功能签发应收账款的时候，HTML 如下：

```
<!-- func1 -->
<div id="func1" class="function4 span-inline">
  <h2>func1:buyGoods</h2>
  <form id="func1_form">
    <label>
      <span>seller</span>
      <input type="text" value="" id="func1_seller" name="seller"/></label>
    <label>
      <span>amount ($)</span>
      <input type="number" value="" id="func1_amount"
name="amount"/></label>
    <label>
      <span>delayTime (day)</span>
      <input type="number" value="" id="func1_delaytime"
name="time"/></label>
    <label>
      <span>Extra Information</span>
      <input type="text" value="" id="func1_info" name="info"/></label>
    <button type="button" class="submit"
id="func1_button">submit</button>
  </form>
</div>
```

前端 js 获取表单信息 , 使用 ajax 异步提交到后端 ;

- 路由 url 为 /func1, 这个在后端的时候再深入 , 这里看可以理解为后端接受不同的 post/get 请求会有不同的处理方式 , 那么这个请求对的处理方式为 func1 ;
- 请求方式 type 为 post, 代表要提交数据 ;
- 提交的数据 data 为 id = func1_form 的表单的序列化表单值 ;
- 如果成功的话 , 执行匿名函数 function(result) ;
- 如果失败的话 , 抛出异常。

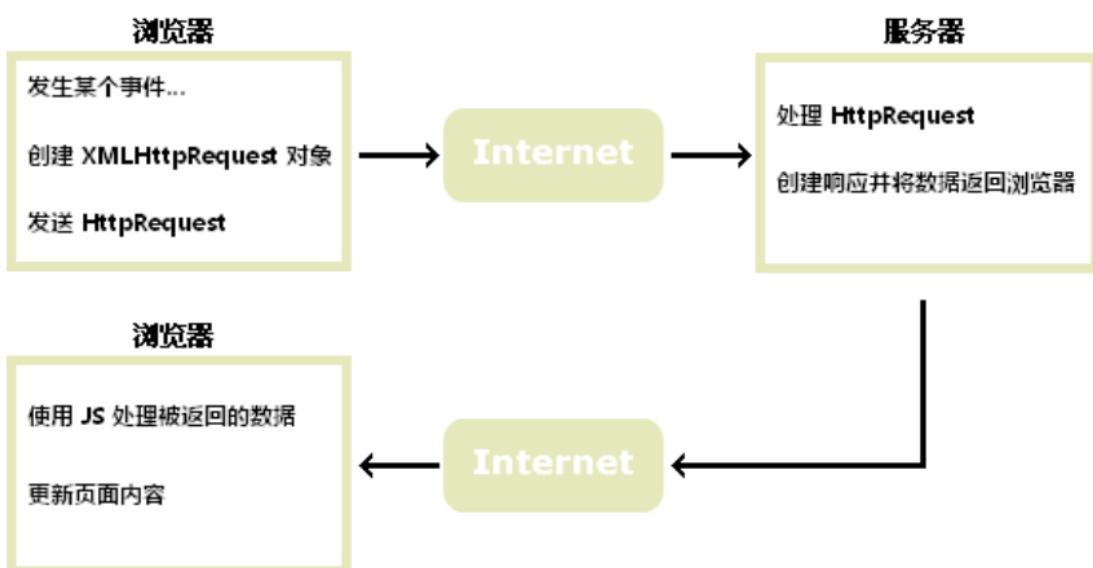
```

$( "#func1_button" ).click(function() {
    func1();
});

function func1() {
    if (isValidForfunc1()) {
        $.ajax({
            url: '/func1',
            type: 'post',
            dataType: 'json',
            data: $('#func1_form').serialize(),
            success: function(result) {
                //打印服务端返回的数据(调试用)
                if (result.status == "ok") {
                    result.index = '0x' + result.index
                    $("#resultText").html('buyGoods success! Receipt
index : ' + parseInt(result.index));
                } else $("#resultText").html('buyGoods failed!');
            },
            error: function(err) {
                $("#resultText").html('failed to get post-returns!');
            }
        });
    }
}

```

其中，用到了辅助函数 `isValidfunc1`，在前端将输入合法化，以免传入一些错误的输入的调用合约，比如名称或者密码为空格。另外，`ajax` 的工作原理如下（图片来自[菜鸟教程](#)）：



代码如下：

```
function isValidForfunc1() {
    //获取 name/amount/time/info
    var seller = $("#func1_seller").val();
    var amount = $("#func1_amount").val();
    var time = $("#func1_delaytime").val();
    var info = $("#func1_info").val();

    //检验是否合法
    if (seller == "") {
        alert("Please input seller name!")
        console.log('invalid input.');
        return false;
    }
    if (amount <= 0 || time <= 0) {
        alert("Please input amount and delay time for payback!")
        console.log('invalid input.');
        return false;
    }
    return true;
}
```

其他的功能的逻辑与此相同，最后整个 js 前端的函数为：

Index.js，有登陆和注册两个功能：

```
function register()          //注册
    function isValidForRegister() //检验注册的信息是否合法
function login()           //登陆，需要输入的信息较少，因此直接在这里检验了
    function menu()            //跳转到功能界面
```

Menu.js，有基础的 4 个功能以及一个扩展功能。

```
function func1 ()      //签发应收账款
function func2 ()      //转让应收账款
function func3 () //向银行融资
function func4 () //结算，不需要辅助函数
function func5 () //获得账单信息

//辅助函数：
function isValidForfunc1()
function isValidForfunc2()
function isValidForfunc3()
function isValidForfunc5()
function showReceipt(result) //func5 得到账单后的输出
```

2.2.3 后端

- 后端用到了 express 框架，引入框架如下：

```
const express = require('express');
var bodyParser = require('body-parser');
// 创建 application/x-www-form-urlencoded 编码解析
var urlencodedParser = bodyParser.urlencoded({ extended: false })
const app = express();           // 一个 express 实例
app.use(express.static('.'));   // 代表静态路由
//
.....不同路由的代码
//

//监听端口号
app.listen(8080, function() {
    console.log('Server running at http://127.0.0.1:8080/');
});
```

后面再对不同的**路由写函数**。

路由用于确定应用程序如何响应对特定端点的客户机请求，包含一个 URI（或路径）和一个特定的 HTTP 请求方法（GET、POST 等）。

每个路由可以具有一个或多个处理程序函数，这些函数在路由匹配时执行。路由定义采用以下结构：

app.**METHOD**(PATH, HANDLER)

来自 <https://expressjs.com/zh-cn/starter/basic-routing.html>

- Nodejs SDK 中模块的引用如下：

```
'use strict';

var __dirname = '/home/fisco-bcos/nodejs-sdk';
var path = require('path');

//给 api 配置私钥和证书
const config = require(path.join(__dirname, "./packages/api/common/configuration")).Configuration;
config.setConfig(path.join(__dirname, "./packages/cli/conf/config.json"));

//调用 API: sendRawTx
const web3jService = require(path.join(__dirname, "./packages/api/web3j")).Web3jService
const web3 = new web3jService;

//调用 decodeParams 函数，后面才改用的 decodeMethod，但是前面没时间改了
const util = require(path.join(__dirname, "./packages/api/common/web3lib/utils"));
//调用 decodeMethod 函数，不需要自己拼签名
const utils = require(path.join(__dirname, "./packages/api/common/utils"));
//合约地址
var contractAddress = '0xfd2d15cb5fabaf05d1e148f4052e67776a77ed3';
```

- 处理 post/get 请求

这里只写关键的 4 个函数：

- 前提，login 的时候，如果登陆成功，后端会记录当前登陆者的名称 `currentUser`。

伪算法：

1. 路由为 `/funcX (X = 1, 2, 3, 4, 5)`；
2. 取出前端传过来的参数，将需要调用合约的函数的参数放入一个数组 `params`；
3. 异步调用了一个辅助函数 `funcX`；
4. 收到结果之后，将其返回给前端。

- 异步函数

API调用约定

- 使用服务之前，首先需要初始化全局的 `Configuration` 对象，用以为各个服务提供必要的配置信息。`Configuration` 对象位于 `nodejs-sdk/packages/api/common/configuration.js`，其初始化参数为一个配置文件的路径或包含配置项的对象。配置文件的配置项说明见[配置说明](#)
- 如无特殊说明，Node.js SDK提供的API均为异步API。异步API的实际返回值是一个包裹了API返回值的Promise对象，开发者可以使用`async/await`语法或`then...catch...finally`方法操作该Promise对象以实现自己的程序逻辑
- 当API内部出现错误导致逻辑无法继续执行时（如合约地址不存在），均会直接抛出异常，所有异常均继承自`Error`类

因此，需要掌握使用 promise 的正确姿势。这里是[参考教程](#)。

■ 功能 1，签发应收账款

```
// func1,buyGoods
app.post('/func1', urlencodedParser, (request, response) => {
  console.log("主页 func1-POST 请求");
  // 打包函数参数
  var buyer = currentUserName
  var seller = request.body.seller;
  var amount = parseInt(request.body.amount)
  var time = parseInt(request.body.time)
  var info = request.body.info;
  var params = new Array();
  params.push(buyer);
  params.push(seller);
  params.push(amount);
  params.push(time);
  params.push(info);
  console.log(params)
  //promise, get true/false
  const promise = func1(params);
  promise.then(function(temp) {
    console.log('returns to js web: ' + JSON.stringify(temp));
    response.send(temp)
  });
});
```

其中，异步辅助函数 `func1` 如下：

```
async function func1(params) {
  var functionName = "buyGoods(string,string,uint256,uint256,string)"
```

```

var item = {
    "constant": false,
    "inputs": [{
        "name": "buyer",
        "type": "string"
    }, {
        "name": "seller",
        "type": "string"
    }, {
        "name": "money",
        "type": "uint256"
    }, {
        "name": "daysAfter",
        "type": "uint256"
    }, {
        "name": "information",
        "type": "string"
    }],
    "name": "buyGoods",
    "outputs": [{
        "name": "",
        "type": "uint256"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}

return web3.sendRawTransaction(contractAddress, functionName, params).then(result => {
    let txHash = result.transactionHash;
    let status = result.status;
    let ret = {
        transactionHash: txHash,
        status: status
    };
    let output = result.output;
    if (output !== '0x') {
        ret.output = utils.decodeMethod(item, output);
        var index = ret.output[0] // {"0":"f"}
        if (index != 99999999) {
            return { status: "ok", index: index }
        } else return { status: "err", index: index }
    } else return { status: "Error", index: index }
}

```

```
    });
}
```

注意这里解析输出的时候，使用的是 decodeMethod，不需要自己构造签名。

在我的代码中，需要 uint256 参数的函数合约调用，我用了 decodeMethod，其他的调用我用的是 decodeParams 进行输出数据 output 的解析。

因为最开始我是用的 decodeParams，自己构造签名，但是一旦输入参数有 uint 类型就解析不了 output，后面改用了 decodeMethod 才可以。

但是前面的我没改，web3 的技术大佬们反正说的是 decodeMethod 比较好。

■ 功能 2，转让应收账款

```
// func2,transfer
app.post('/func2', urlencodedParser, (request, response) => {
  console.log("主页 func2-POST 请求");
  // 输入数据
  var from = currentUserName
  var to = request.body.to;
  var amount = parseInt(request.body.amount)

  var params = new Array();
  params.push(from);
  params.push(to);
  params.push(amount);

  console.log(params)
  //promise, get true/false
  const promise = func2(params);
  promise.then(function(temp) {
    console.log('returns to js web: ' + JSON.stringify(temp));
    response.send(temp)
  });
});
```

调用的异步辅助函数 func2 如下：

```
async function func2(params) {
  var functionName = "transferReceipt(string,string,uint256)"
  var item = {
    "constant": false,
```

```

    "inputs": [
        {
            "name": "company1",
            "type": "string"
        },
        {
            "name": "company2",
            "type": "string"
        },
        {
            "name": "money",
            "type": "uint256"
        }
    ],
    "name": "transferReceipt",
    "outputs": [
        {
            "name": "",
            "type": "uint256"
        },
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}

return web3.sendRawTransaction(contractAddress, functionName, params).then(result => {
    let txHash = result.transactionHash;
    let status = result.status;
    let ret = {
        transactionHash: txHash,
        status: status
    };
    let output = result.output;
    if (output !== '0x') {
        ret.output = utils.decodeMethod(item, output);
        console.log(ret.output)
        var countReceipts = ret.output[0]
        var sum = ret.output[1]
        if (sum != 0) {
            console.log(sum + ' ' + typeof(sum))
            return { status: "ok", count: countReceipts, sum: sum }
        } else return { status: "err" }
    } else return { status: "Error" }
    // { transactionHash: '0xd96a12...' ,

```

```
//      status: '0x0',
//      output: Result { '0': <BN: f> } } //array
});
}
```

■ 功能 3 , 融资

```
// func3 raisemoney
app.post('/func3', urlencodedParser, (request, response) => {
  console.log("主页 func3-POST 请求");
  // 输入数据
  var company = currentUserName
  var bank = request.body.bank;

  var params = new Array();
  params.push(company);
  params.push(bank);

  console.log(params)
  //promise, get true/false
  const promise = func3(params);
  promise.then(function(temp) {
    console.log('returns to js web: ' + JSON.stringify(temp));
    response.send(temp)
  });
});
```

调用的异步辅助函数 func3 如下：

```
async function func3(params) {
  var result = await web3.sendRawTransaction(
    contractAddress,
    "raiseMoney(string,string)",
    params
  ).catch((err) => { console.error(err) });
  console.log('part 1 web3 sendRTx');

  var output = result.output;
  console.log(result)
  var types = new Array();
  types.push('uint');
  types.push('uint');
  if (output != '0x') {
    var decodeResult = util.decodeParams(types, output);
```

```

console.log(decodeResult)
var count = decodeResult[0]
var sum = decodeResult[1]
if (count != 99999999 && sum != 0) {
    return { status: "ok", count: count, sum: sum }
} else return { status: "err", sum: 0 }
} else return { status: "Err" }; //invalid transaction
}

```

可以看到是调用的 `decodeParams` 解析输出结果。

■ 功能 4 , 结算

```

// function4,payback
app.get('/func4', (request, response) => {
    console.log("主页 func4-POST 请求");
    var company = currentUserName
    var params = new Array();
    params.push(company);

    const promise = func4(params);
    promise.then(function(temp) {
        console.log('returns to js web: ' + JSON.stringify(temp));
        response.send(temp)
    });
})

```

由于这个在前端不需要输入数据，因此使用的是 get 请求。

调用的异步辅助函数 `func4` 如下：

```

async function func4(params) {
    var result = await web3.sendRawTransaction(
        contractAddress,
        "payReceipt(string)",
        params
    ).catch((err) => { console.error(err) });
    console.log('part 1 web3 sendRTx');

    var output = result.output;
    console.log(result)
    var types = new Array();
    types.push('uint');
    types.push('uint');
}

```

```

var status = ""
if (output != '0x') {
    var decodeResult = util.decodeParams(types, output);
    var count = decodeResult[0]
    var sum = decodeResult[1]
    if (count != 99999999 && sum != 0) {
        return { status: "ok", count: count, sum: sum }
    } else return { status: "err", sum: 0 }
}
return { status: "err", count: 0, sum: 0 };
}

```

■ 功能 5 , 查询账单

```

//get the receipt information
app.post('/getReceipt', urlencodedParser, (request, response) => {
    console.log("receipt GET 请求");
    var ReceiptIndex = parseInt(request.body.index)
    console.log('received index = ' + ReceiptIndex)

    if (ReceiptIndex >= 0) {
        var params = new Array();
        params.push(ReceiptIndex);

        const promise = getReceipt(params);
        promise.then(function(temp) {
            console.log('returns to js web: ' + JSON.stringify(temp));
            temp.index = ReceiptIndex
            response.send(temp)
        });
    }
});

```

调用的异步辅助函数 func2 如下 :

```

async function getReceipt(params) {
    var functionName = "getReceipt(uint256)"
    var item = {
        "constant": true, //true
        "inputs": [
            {
                "name": "i",
                "type": "uint256"
            },
        ]
    }
}

```

```
"name": "getReceipt",
"outputs": [
    {
        "name": "",
        "type": "string"
    },
    {
        "name": "",
        "type": "string"
    },
    {
        "name": "",
        "type": "uint256"
    },
    {
        "name": "",
        "type": "uint256"
    },
    {
        "name": "",
        "type": "uint256"
    },
    {
        "name": "",
        "type": "string"
    }
],
"payable": false,
"stateMutability": "view", //!nonpayable
"type": "function"
}

return web3.call(contractAddress, functionName, params).then(result => {
    console.log(JSON.stringify(result.result))

    let txHash = result.transactionHash;
    let status = result.status;
    let ret = {
```

```
        transactionHash: txHash,
        status: status
    );
    let output = result.result.output;
    if (output !== '0x') {
        ret = utils.decodeMethod(item, output); //return array
        console.log(ret)
        if (ret[0] != "") {
            return {
                status: "ok",
                from: ret[0],
                to: ret[1],
                type: ret[2],
                amount: ret[3],
                buildDate: ret[4],
                dueDate: ret[5],
                info: ret[6]
            }
        } else return { status: "err" }
    }
});
}
```

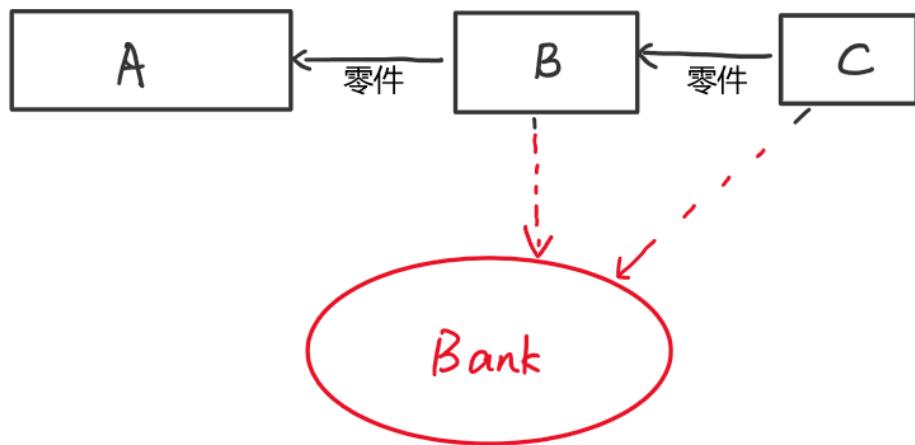
由于要传入 uint 类型参数给合约，因此这个用的是 decodeMethod。

三、 功能测试

这里测试合约、后端、前端。每一个功能将展示不同端的过程。我假设了一个情景，然后在后面的测试中我是根据这个情景的逻辑进行相关的检验。

3.1 一个情景

如果 C - B - A 是一条供应链，Bank 是一家银行。

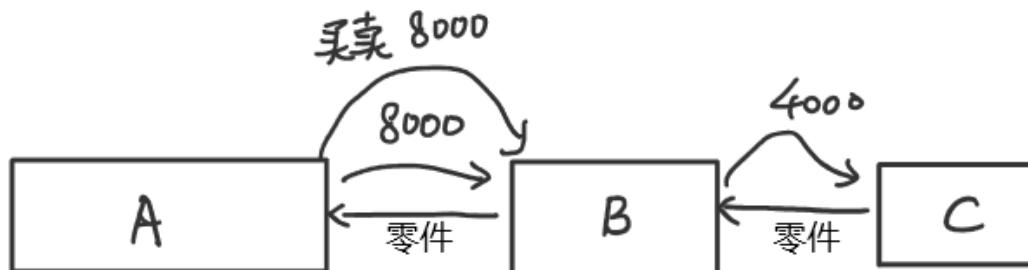


(1) **Register** 注册 4 个组织账号，为 3 个公司 A、B、C 以及一个银行 Bank。

(2) **BuyGoods** 如果 A 向先后两次采购了价值为 8000 的商品，由于一般来说签订合约的时间不一样，那么 `duetime` 也不一样，因此生成两个账单，`receipt0` 以及 `receipt1`；
B 向 C 采购了价值为 4000 的商品，生成账单 `receipt2` 主要信息如下：

即，调用两次 `buyGoods("B", "A", 8000...")`

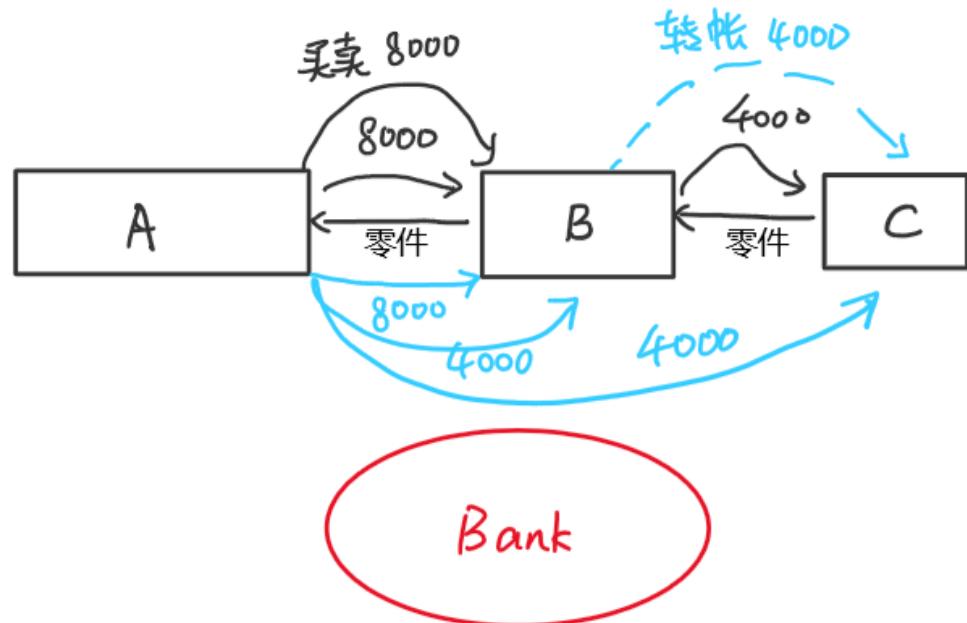
再调用一次 `buyGoods("C", "B", 4000....");`



账单如下：

Receipt2	Receipt1	Receipt0
From: B	From: A	From: A
To: C	To: B	To: B
Type: 0	Type: 0	Type: 0
Uint: 4000	Uint: 8000	Uint: 8000
DueTime: date2	DueTime: date1	DueTime: date0

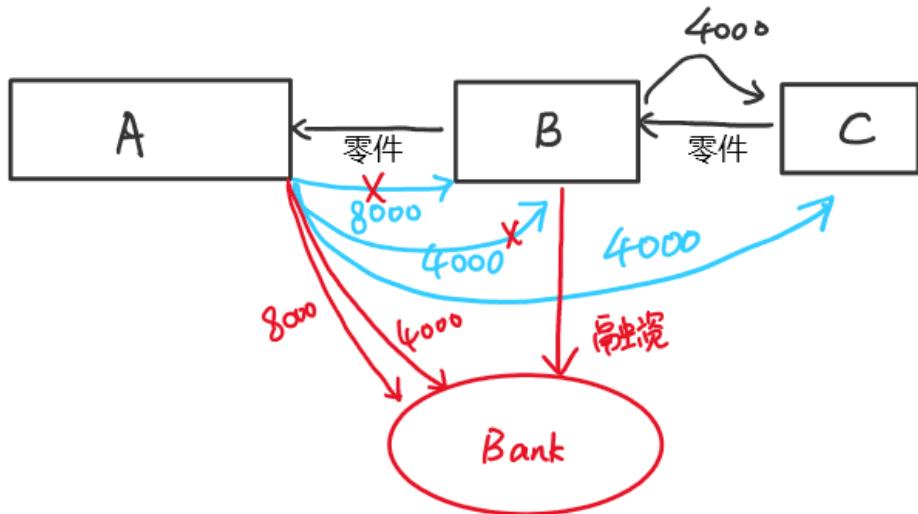
(3) transferReceipt, 此时 B 向 C 转 4000, 即 transferReceipt("B", "C", 4000);



账单变为：

Receipt0 From: A To: B Type: 0 Uint: 4000 DueTime: date0	Receipt1 From: A To: B Type: 0 Uint: 8000 DueTime: date1	Receipt2 From: B To: C Type: 0 Uint: 4000 DueTime: date2
Receipt3 From: A To: C Type: 1 Uint: 4000 DueTime: date0		

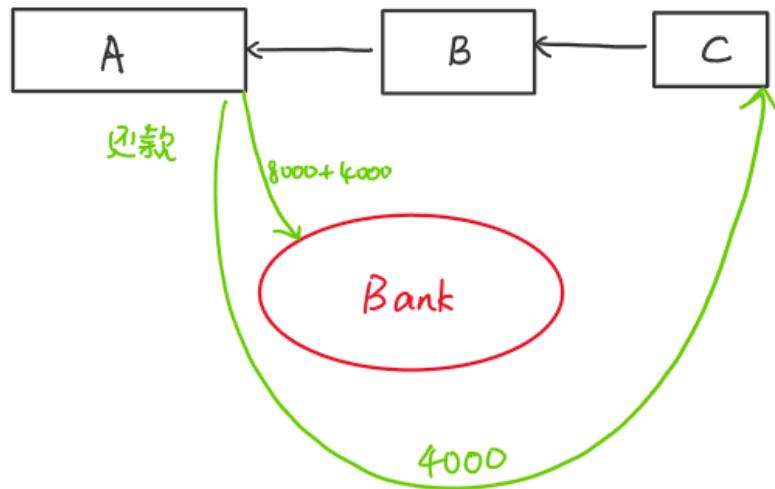
(4) RaiseMoney, 此时 B 向银行 Bank 融资, 将所有关于收账人是 B 的有效账单给银行, 得到资金。



此时的账单如下：

Receipt0 From: A To: Bank Type: 2 Uint: 4000 DueTime: date0	Receipt1 From: A To: Bank Type: 2 Uint: 8000 DueTime: date1	Receipt2 From: B To: C Type: 0 Uint: 4000 DueTime: date2
Receipt3 From: A To: C Type: 0 Uint: 4000 DueTime: date0		

(5) PayReceipt, 此时 A 偿还已过期的合约，由于实验效果，我设置的账单有效期均比较短，只有几分钟。因此在 A 的账单均已过期，结算 A 的所有欠款。



账单如下：

Receipt0	Receipt1	Receipt2
From: A To: Bank Type: 3 Uint: 4000 DueTime: date0	From: A To: Bank Type: 3 Uint: 8000 DueTime: date1	From: B To: C Type: 0 Uint: 4000 DueTime: date2

Receipt3
From: A To: C Type: 3 Uint: 4000 DueTime: date0

3.2 测试合约

合约测试在 remix 中进行，测试过程如下：

(1) Register

测试函数如下：

```

function test_register()public {
    if(register("A", true) && register("B", true)&&
        register("C", true) && register("Bank", false)){
        assert(isExist("A"));
        assert(isExist("B"));
        assert(isExist("C"));
        assert(isExist("Bank"));
    }
}

```

调用函数：

The screenshot shows a list of functions on the left and a transaction details panel on the right.

Functions (Left):

- buyGoods string seller, string buyer
- payReceipt string company
- raiseMoney string company, string b
- register string name, bool tpye
- test_buyGoods
- test_main
- test_register** (highlighted with a red box)

Transaction Details (Right):

- decoded output: -
- logs: []
- value: 0 wei
- transact to con.test_register pending ...
- [vm] from:0x583...40225 to:con.test_register() 0x832...8cd32 (highlighted with a red box)
- status: 0x1 Transaction mined and ex
- transaction hash: 0x6498fe1e5e07f242c1d9289021

成功。

(2) buyGoods

测试函数如下：

```

function test_buyGoods_0()public {
    uint i = buyGoods("B", "A", 8000, 15, "An Apple");
    assert(i == 0 );

    assert(isEqual(getFrom(i), "A"));
    assert(isEqual(getTo(i), "B"));
    assert(getAmount(i) == 8000);
    assert(getType(i) == 0);
}

function test_buyGoods_1()public {
    uint j = buyGoods("B", "A", 8000, 15, "An Apple");
    assert(j == 1);

    assert(isEqual(getFrom(j), "A"));
    assert(isEqual(getTo(j), "B"));
    assert(getAmount(j) == 8000);
    assert(getType(j) == 0);
}

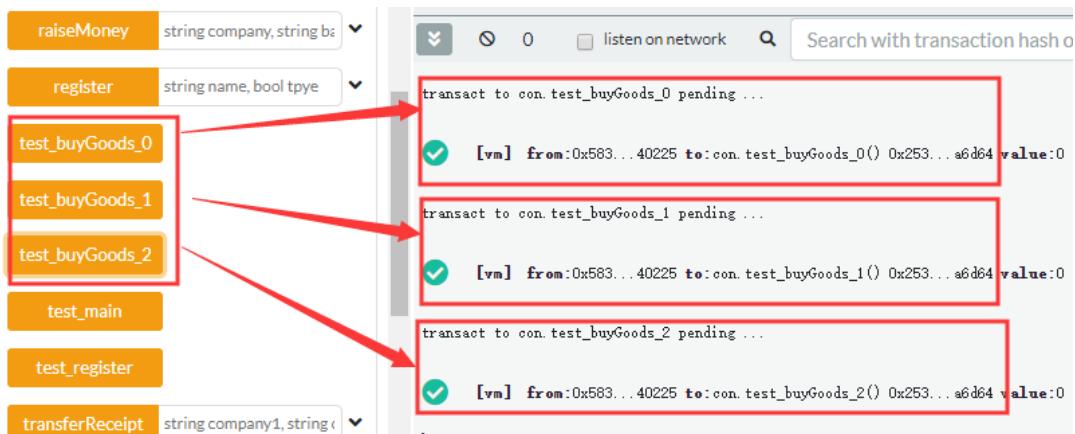
function test_buyGoods_2()public{
    uint k = buyGoods("C", "B", 4000, 15, "A banana");
    assert(k == 2);

    assert(isEqual(getFrom(k), "B"));
    assert(isEqual(getTo(k), "C"));
    assert(getAmount(k) == 4000);
    assert(getType(k) == 0);
}

```

调用函数：

3 个 func1 函数，分别调用如下：



成功。

(3) Transfer

```
function test_transfer() public returns(uint){  
    uint [] memory index_arr;  
    uint size;  
    uint j;  
    (size, index_arr)=transferReceipt("B", "C", 4000);  
  
    assert(size == 1);  
  
    assert(isEqual(getFrom(j), "A"));  
    assert(isEqual(getTo(j), "C"));  
    assert(getAmount(3) == 4000);  
    assert(getType(3) == 1);  
    assert(getAmount(0) == 4000);  
}
```

调用测试函数如下：



成功。

(4) RaiseMoney

代码如下：

```

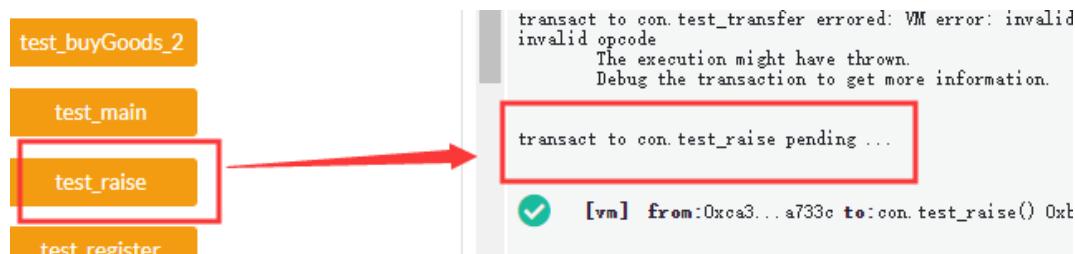
function test_raise() public {
    uint [] memory index_arr;
    uint size;
    (size, index_arr) = raiseMoney("B", "Bank");
    assert(size == 2);

    uint i = 0 ;
    uint j = 1;
    assert(getType(i) == 2 && getType(j) == 2);

    //再次测试，没有任何结果
    (size, index_arr) = raiseMoney("B", "Bank");
    assert(size == 1 && index_arr[0] == 99999999);
}

```

测试结果如下：



成功。

(5) payReceipt

代码如下：

```

function test_pay()public{
    uint [] memory index_arr;
    uint size;
    (size, index_arr) = payReceipt("A");
    assert(size == 3);
    assert(index_arr[0] == 0 && index_arr[1] == 1 && index_arr[2] == 3);
    assert(getType(0) == 3 && getType(1) == 3 && getType(2) == 3);
}

```

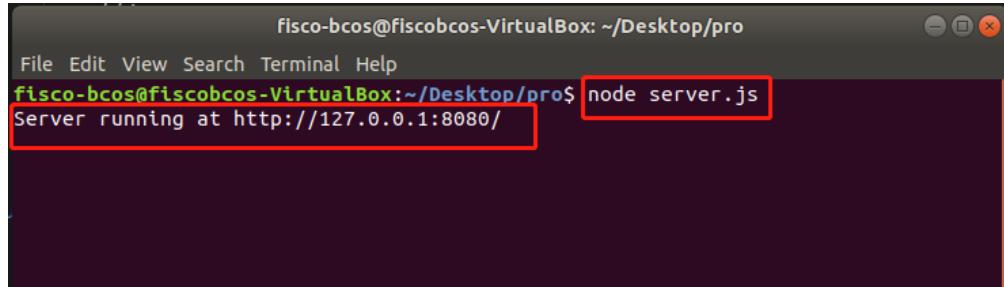
测试结果如下：



成功。

3.3 测试前后端

后端 node 启动如下：



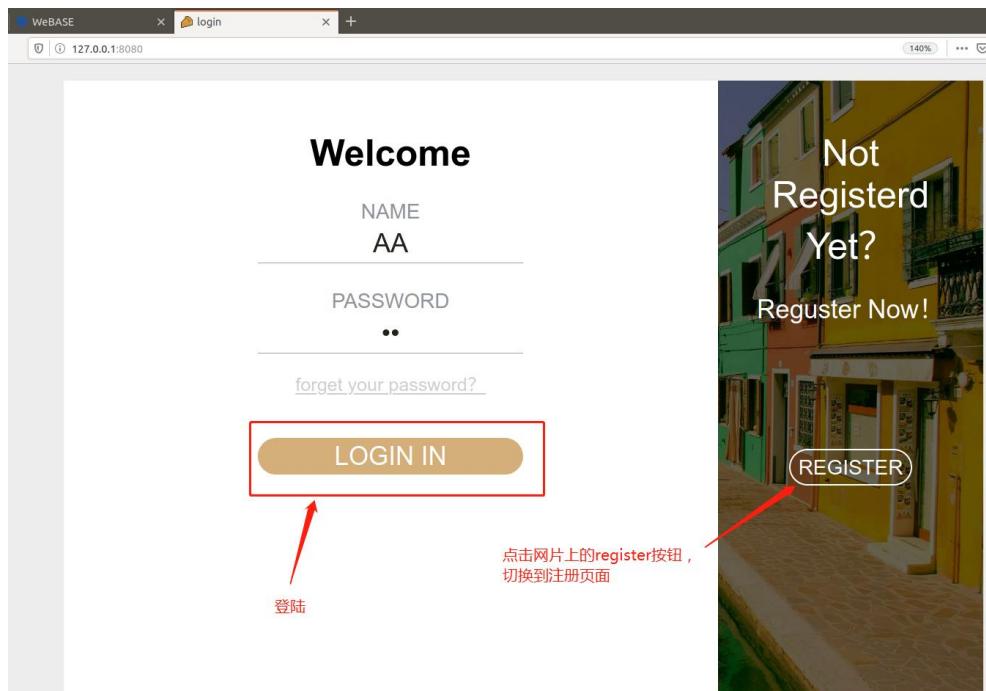
```
Fisco-bcos@fiscobcos-VirtualBox: ~/Desktop/pro
File Edit View Search Terminal Help
fisco-bcos@fiscobcos-VirtualBox:~/Desktop/pro$ node server.js
Server running at http://127.0.0.1:8080/
```

开始监听前端的请求，监听的端口为 8080

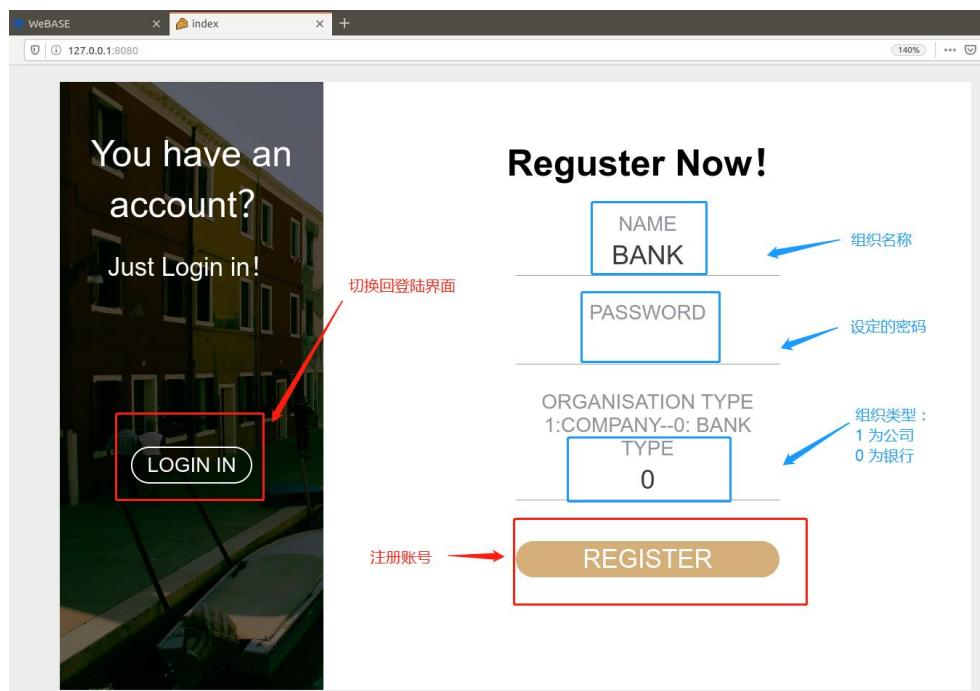
3.3.1 登陆界面

- 浏览器中打开网址 <http://127.0.0.1:8080/>

设计完成之后的界面应该是这样的，一个可以滑动的组件，可以来回切换登陆/注册：

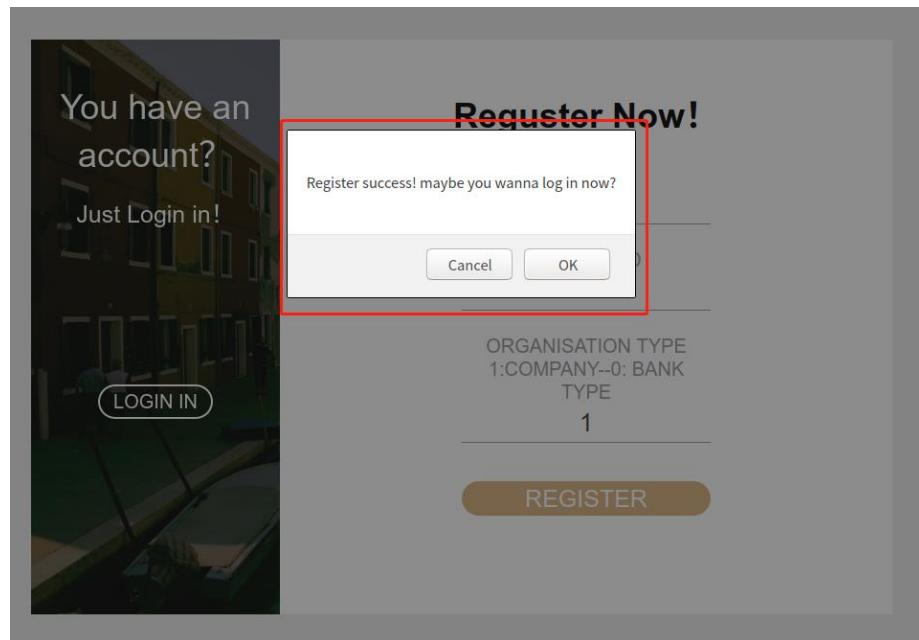


切换到注册页面之后，是这样子：



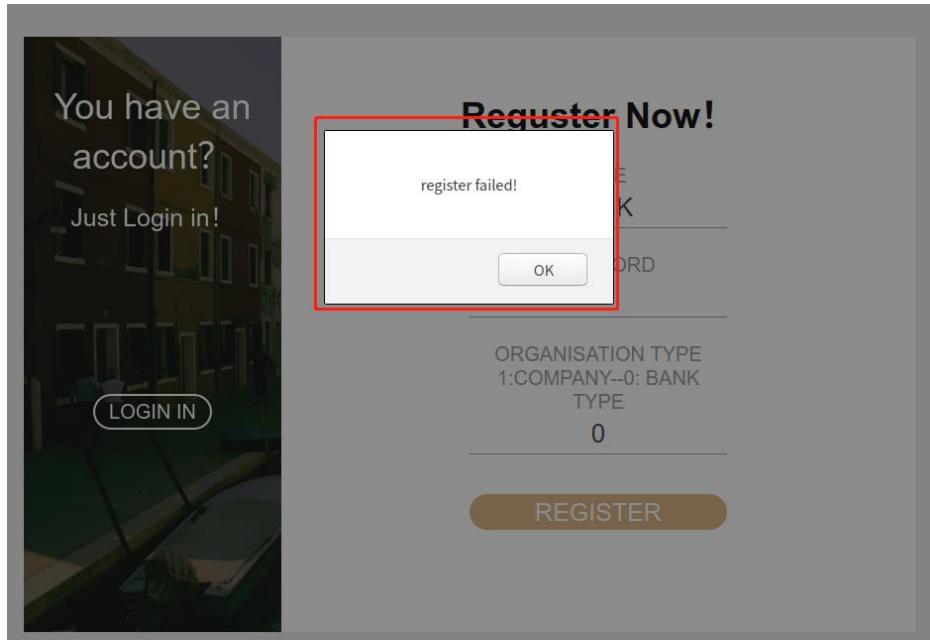
注册成功之后，弹出消息：

- 如果这个名称没有被注册过，注册成功：



根据反馈决定是否要回到登陆界面，进行登陆，或者继续在这个页面上进行账户的注册，方便批量注册用户。

- 这个名称已经被注册过，当前操作失败：



■ 后端输出如下：

1. 收到来自前端的请求，请求方式是 post，因为要提交用户注册的信息；
 2. 中间的 object 是调用 nodejs SDK 之后，得到的返回的结果，是交易成功的区块信

息；

3. 其中 output 是我们想要的信息，即合约中这个函数的返回值；
4. 解析之后，我们拿到这个参数，验证这个参数是否有效（如果注册失败应该改返回一个无穷大的数）；
5. 返回这个参数给前端。

3.1.2 功能界面

登陆之后，系统会记录当前登陆用户的名称，进入功能页面如下：

The screenshot shows a web browser window titled "func page" at the URL "127.0.0.1:8080/menu.html". The page displays a "Welcome, AA!" message and a "LOG OUT" button. Below this, there are five function buttons:

- func1:buyGoods**:
 - SELLER:
 - AMOUNT(\$):
 - DELAYTIME(DAY):
 - EXTRA INFORMATION:
 - SUBMIT
- func2:transferReceipt**:
 - TO:
 - AMOUNT(\$):
 - SUBMIT
- func3:raiseMoney**:
 - BANK NAME:
 - SUBMIT
- func4:payReceipt**:
 - SUBMIT
- func5:getReceipt**:
 - RECEIPT INDEX:
 - SUBMIT

Annotations with red arrows point to specific fields and buttons:

- 指向 "func1:buyGoods" 的 "SELLER" 字段的箭头标注为 "功能1：向seller买东西，签发应收账款"
- 指向 "func2:transferReceipt" 的 "TO" 字段的箭头标注为 "功能2：转让amount金额的账款给公司TO"
- 指向 "func3:raiseMoney" 的 "BANK NAME" 字段的箭头标注为 "功能3：向银行BANK融资"
- 指向 "func4:payReceipt" 的 "SUBMIT" 按钮的箭头标注为 "功能4：结算这个公司的所有欠款"
- 指向 "func5:getReceipt" 的 "RECEIPT INDEX" 字段的箭头标注为 "附加功能：查询编号为index的账单信息"

- 功能 1：A 公司向 B 公司进行两次金额为 8000 的交易，签发两个应收账款。

The screenshot shows the same "func page" interface after performing a transaction. The "func1:buyGoods" form is highlighted with a red border, showing the following data:

SELLER	B
AMOUNT(\$)	8000
DELAYTIME(DAY)	8000
EXTRA INFORMATION	an apple
SUBMIT	

A red box highlights the "func1:buyGoods" section. To the right, the other four functions are visible. At the bottom center, a red-bordered box contains the message "buyGoods success! Receipt index : 6".

Welcome, A!

LOG OUT

func1:buyGoods SELLER B AMOUNT(\$) 8000 DELAYTIME(DAY) 8000 EXTRA INFORMATION a pear <input type="button" value="SUBMIT"/>	func2:transferReceipt TO AMOUNT(\$) <input type="button" value="SUBMIT"/> <input type="button" value="SUBMIT"/>	func3:raiseMoney BANK NAME <input type="button" value="SUBMIT"/>	func4:payReceipt <input type="button" value="SUBMIT"/>	func5:getReceipt RECEIPT INDEX <input type="button" value="SUBMIT"/>
--	--	---	--	---

buyGoods success! Receipt index : 7

可见，生成了 2 个账单，编号分别是 0x6、0x7，现在用**功能 5** 查询这两个账单：

注意，这个账单编号是 16 进制。

func1:buyGoods SELLER B AMOUNT(\$) 8000 DELAYTIME(DAY) 8000 EXTRA INFORMATION a pear <input type="button" value="SUBMIT"/>	func2:transferReceipt TO AMOUNT(\$) <input type="button" value="SUBMIT"/> <input type="button" value="SUBMIT"/>	func3:raiseMoney BANK NAME <input type="button" value="SUBMIT"/>	func4:payReceipt <input type="button" value="SUBMIT"/>	func5:getReceipt RECEIPT INDEX 6 <input type="button" value="SUBMIT"/>
--	--	---	--	--

Receipt: 6

From: A
To: B
Receipt Type: 0
Receipt Amount: 8000
buildDate: 16f03dd3de5
dueDate: 16f2d101de5
extra information: an apple

func1:buyGoods SELLER B AMOUNT(\$) 8000 DELAYTIME(DAY) 8000 EXTRA INFORMATION a pear <input type="button" value="SUBMIT"/>	func2:transferReceipt TO AMOUNT(\$) <input type="button" value="SUBMIT"/> <input type="button" value="SUBMIT"/>	func3:raiseMoney BANK NAME <input type="button" value="SUBMIT"/>	func4:payReceipt <input type="button" value="SUBMIT"/>	func5:getReceipt RECEIPT INDEX 7 <input type="button" value="SUBMIT"/>
--	--	---	--	--

Receipt: 7

From: A
To: B
Receipt Type: 0
Receipt Amount: 8000
buildDate: 16f03de0874
dueDate: 16f2d10e874
extra information: a pear

由于交易信息一样，因此查询出来的账单除了时间上、额外信息不一样，其他相同。

后端输出：

```
number
[ 'A', 'B', 8000, 8000, 'an apple' ]
returns to js web: {"status":"ok","index":"6"}  
menu getName-get 请求
主页 func1-POST 请求
number
[ 'A', 'B', 8000, 8000, 'a pear' ]
returns to js web: {"status":"ok","index":"7"}  
... - - -请求
```

这表示收到前端 func1 的请求、返回给这个请求的参数。

其中，查询账单的后端输出如下：

收到来自前端的 func5 请求、返回给前端的结果。

■ 功能 2：B 公司向 C 公司转让 4000 的金额

登陆 B 账号，结果如下：

Welcome, B!

LOG OUT

func1:buyGoods SELLER _____ AMOUNT(\$) _____ DELAYTIME(DAY) _____ EXTRA INFORMATION _____ <input type="button" value="SUBMIT"/>	func2:transferReceipt TO C _____ AMOUNT(\$) 4000 <input type="button" value="SUBMIT"/>	func3:raiseMoney BANK NAME _____ <input type="button" value="SUBMIT"/>	func4:payReceipt <input type="button" value="SUBMIT"/>	func5:getReceipt RECEIPT INDEX _____ <input type="button" value="SUBMIT"/>
---	--	--	--	--

transfer receipt success! count:1total Amount: 4000

根据之前的逻辑，还会分裂成一个新的账单 0x8,现在查询已有的 3 个订单：

<div style="border: 1px solid #ccc; padding: 10px; width: 100%;"> <p style="text-align: center;">Receipt: 6</p> <p>From: A To: B Receipt Type: 0 Receipt Amount: 4000 buildDate: 16f03dd3de5 dueDate: 16f2d101de5 extra information: an apple</p> </div>	<div style="border: 1px solid #ccc; padding: 10px; width: 100%;"> <p style="text-align: center;">Receipt: 7</p> <p>From: A To: B Receipt Type: 0 Receipt Amount: 8000 buildDate: 16f03de0874 dueDate: 16f2d10e874 extra information: a pear</p> </div>
<div style="border: 1px solid #ccc; padding: 10px; width: 100%;"> <p style="text-align: center;">Receipt: 8</p> <p>From: A To: C Receipt Type: 1 Receipt Amount: 4000 buildDate: 16f03dd3de5 dueDate: 16f2d101de5 extra information: an apple</p> </div>	

可以看到账单 6 由于已经转让了 4000，现在的金额变成 $8000 - 4000 = 4000$ ；

操作的金额为 4000，账单 6 的金额已经足够，因此账单 7 没有任何改动；

新生成一个账单 8，这个收款人是 C，类型是 1 代表这个账单是别人转过来的。

操作成功。后端如下：

```
主页 func2-POST 请求
[ 'B', 'C', 4000 ]
Result { '0': <BN: 1>, '1': <BN: fa0> }
4000 object
returns to js web: {"status":"ok","count":"1","sum":"fa0"}
```

表示收到来自前端 func2 的请求、返回的结果 (fa0 是 16 进制的 4000)。

■ 功能 3：B 公司向银行 Bank 进行融资

Welcome, B!

LOG OUT

func1:buyGoods	func2:transferReceipt	func3:raiseMoney	func4:payReceipt	func5:getReceipt
SELLER C	TO C	BANK NAME Bank	SUBMIT	RECEIPT INDEX 8
AMOUNT(\$) 4000	AMOUNT(\$) 4000	SUBMIT		SUBMIT
DELAYTIME(DAY)	SUBMIT			
EXTRA INFORMATION				
SUBMIT				

账单 6 中：4 0 0 0
账单 7 中：8 0 0 0

raiseMoney success! Receipt count: 2 total amount: 12000

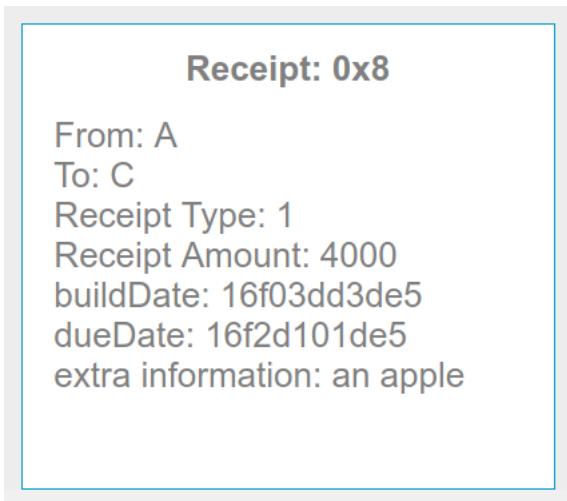
现在再查询这几个账单，收款人为 B 的账单，也就是刚刚的 6、7。

Receipt: 6

From: A
To Bank
Receipt Type 2
Receipt Amount: 4000
buildDate: 16f03dd3de5
dueDate: 16f2d101de5
extra information: an apple

Receipt: 7

From: A
To Bank
Receipt Type 2
Receipt Amount: 8000
buildDate: 16f03de0874
dueDate: 16f2d10e874
extra information: a pear



可以看到，收款人已经变成了银行 Bank，代表 A 欠 Bank；

账单类型变为 2，代表这是融资的账单；

账单 0x8 的收款人是 C，因此 B 融资与此无关，没有变动。

后端：

■ 功能 4：A 公司结算所有的应收账款

Welcome, A!

LOG OUT

func1:buyGoods SELLER AMOUNT(\$) DELAYTIME(DAY) EXTRA INFORMATION SUBMIT	func2:transferReceipt TO AMOUNT(\$) SUBMIT	func3:raiseMoney BANK NAME SUBMIT	func4:payReceipt SUBMIT	func5:getReceipt RECEIPT INDEX 6 SUBMIT
--	--	--	---------------------------------------	---

payReceipt success!Receipt count = 3, total amount = 16000

现在查看这 3 个账单：

Receipt: 0x6

From: A
To: Bank
Receipt Type: 3
Receipt Amount: 4000
buildDate: 16f03dd3de5
dueDate: 16f2d101de5
extra information: an apple

Receipt: 0x7

From: A
To: Bank
Receipt Type: 3
Receipt Amount: 8000
buildDate: 16f03de0874
dueDate: 16f2d10e874
extra information: a pear

Receipt: 0x8

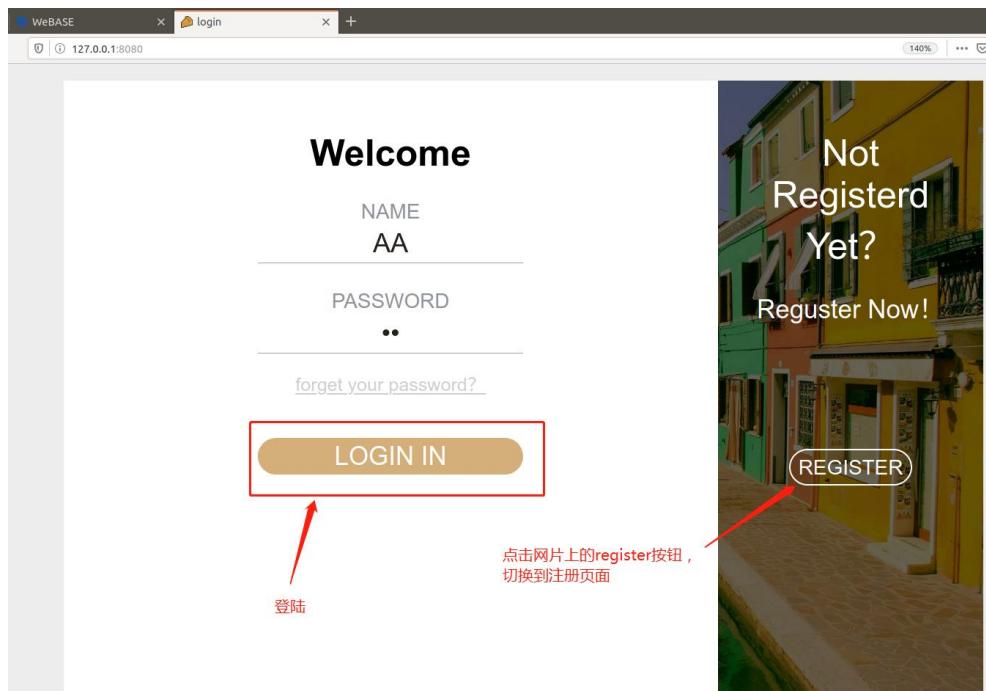
From: A
To: C
Receipt Type: 3
Receipt Amount: 4000
buildDate: 16f03dd3de5
dueDate: 16f2d101de5
extra information: an apple

这 3 个欠款人为 A 的账单都改变了账单类型，代表这个账单已经结算，无效。

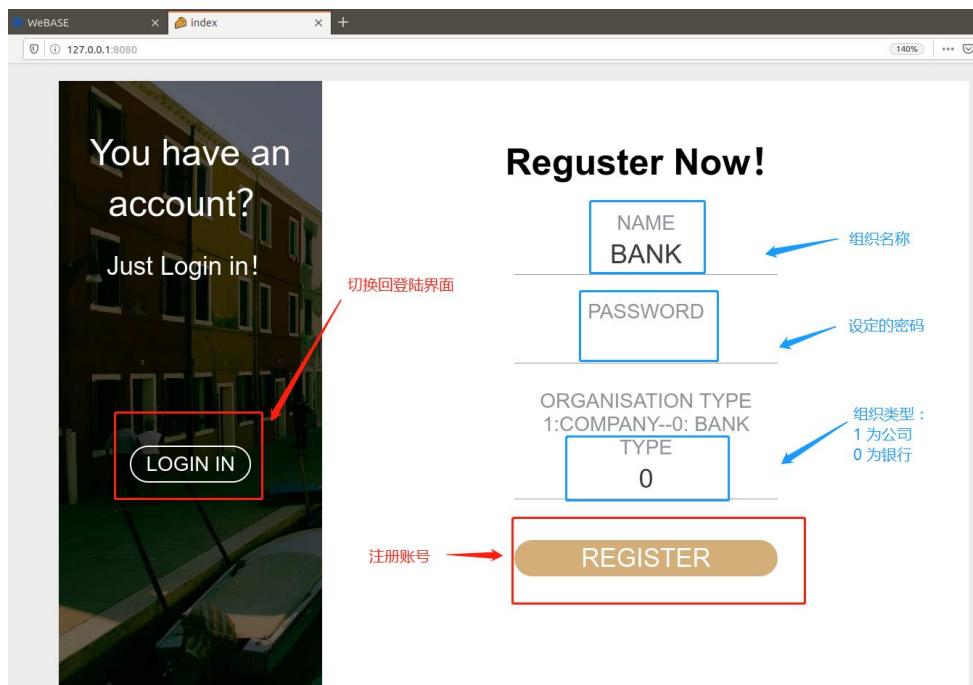
后端：

四、 界面展示

- 登陆：



- 注册：



- 功能界面：

func page x 127.0.0.1:8080/menu.html

Welcome, AA!

[LOG OUT](#) [登出](#)

func1:buyGoods	func2:transferReceipt	func3:raiseMoney	func4:payReceipt	func5:getReceipt
SELLER	TO	BANK NAME	SUBMIT	RECEIPT INDEX
AMOUNT(\$)	AMOUNT(\$)			SUBMIT
DELAYTIME(DAY)		SUBMIT		
EXTRA INFORMATION				
SUBMIT	SUBMIT	SUBMIT		

功能1：向seller买东西，签发应收账款
功能2：转让amount金额的账款给公司TO
功能3：向银行BANK融资
功能4：结算这个公司的所有欠款
附加功能：查询编号为index的账单信息

五、心得体会

整个项目花了很多精力和时间，最后合约所占的比例却不是很大，全部用于写网页了。

三天速成前端后端，但是还是学到了很多东西，写了一个力所能及的最完善的前后端。

不过合约部分其实我觉得还可以更完善一些，比如增加公司的信誉，如果账款到期但是很久没有结算的话，就减少信誉这个样子。

最后，由于很多东西都是现学现卖（比如前后端请求处理），遇到了很多 bug，我将这些 bug 和解决方案都列在了附录中。

六、附录

包括本文档的目录、webbase 管理台配置、nodejs SDK 配置、提交在 GitHub 上的文件列表以及在实验中参考的一些资料。

a) 目录

目录

一、	项目背景	1
二、	方案设计	2
2.1	智能合约	3
2.1.1	生成账单	4
2.1.2	转让账款	5
2.1.3	融资	7
2.1.4	结算账单	8
2.1.5	额外功能	9
2.2	前端	12
2.2.1	框架	12
2.2.2	前端	13
2.2.3	后端	17
三、	功能测试	27
3.1	一个情景	27
3.2	测试合约	31
3.3	测试前端	35
3.3.1	登陆界面	35
3.3.2	功能界面	38
四、	界面展示	47
五、	心得体会	48
六、	附录	48
a)	目录	49
b)	GitHub 提交列表	50
c)	Webase 管理配置	50
d)	Nodejs SDK 配置	54
e)	bugs 以及解决方案	56

b) GitHub 提交列表

code -- 网页前端、后端代码、solidity 合约代码、合约测试代码

zz_17343006 -- 演示视频

report.pdf -- 实验报告

c) Webase 管理配置

本实验用 webase 管理平台部署合约，详见[参考教程](#)，环境配置实验截图如下：

环境为 webase 银行提供的 Ubuntu 虚拟机



(1) 拉取部署脚本

```
sco-bcos@fiscobcos-VirtualBox:~$ unzip webase-deploy.zip
Archive: webase-deploy.zip
  creating: webase-deploy/
  inflating: webase-deploy/build_chain.sh
  creating: webase-deploy/comm/
  inflating: webase-deploy/common.properties
  inflating: webase-deploy/comm/build.py
  inflating: webase-deploy/comm/check.py
  inflating: webase-deploy/comm/log.py
  inflating: webase-deploy/comm/mysql.py
  inflating: webase-deploy/comm/nginx.conf
  inflating: webase-deploy/comm/utils.py
  inflating: webase-deploy/comm/__init__.py
  inflating: webase-deploy/deploy.py
  inflating: webase-deploy/install.md
  extracting: webase-deploy/nodeconf
  inflating: webase-deploy/telnet.py
fisco-bcos@fiscobcos-VirtualBox:~$
```

(2) 修改配置

① 数据库部署

```
fisco-bcos@fiscobcos-VirtualBox:~/webase-deploy$ sudo systemctl start mariadb.service
fisco-bcos@fiscobcos-VirtualBox:~/webase-deploy$ sudo systemctl enable mariadb.service
fisco-bcos@fiscobcos-VirtualBox:~/webase-deploy$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none): [REDACTED]

fisco-bcos@fiscobcos-VirtualBox:~/webase-deploy$ sudo -i
root@fiscobcos-VirtualBox:~# mysql -uroot -p -h localhost -P 3306
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 56
Server version: 10.1.41-MariaDB-0ubuntu0.18.04.1 Ubuntu 18.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> [REDACTED]

MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'zz962910588'
' WITH GRANT OPTION;
Query OK, 0 rows affected (0.11 sec)

MariaDB [(none)]> flush PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> [REDACTED]

MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'test'@localhost IDENTIFIED BY 'zz969
10588' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> flush PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

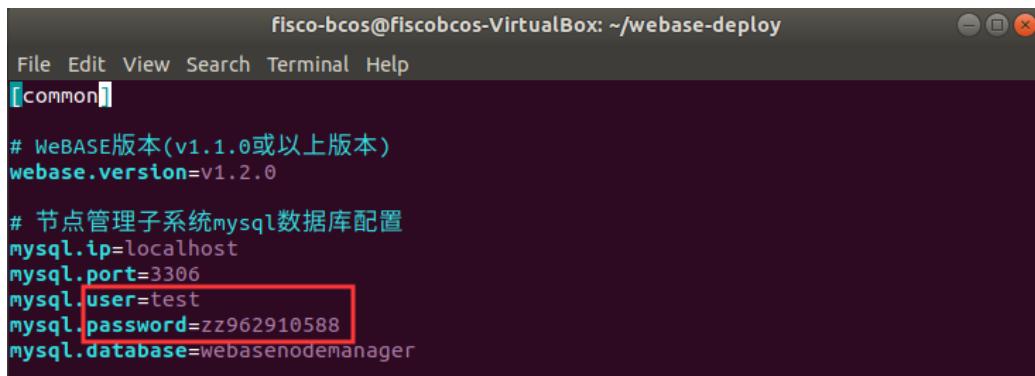
```
root@fiscobcos-VirtualBox:~# mysql -utest -pzz962910588 -h localhost -P 3306
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 67
Server version: 10.1.41-MariaDB-0ubuntu0.18.04.1 Ubuntu 18.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> 
MariaDB [(none)]> create database webasenodemanager;
Query OK, 1 row affected (0.00 sec)
```

② 修改配置文件



```
fisco-bcos@fiscobcos-VirtualBox: ~/webase-deploy
File Edit View Search Terminal Help
[common]
# WeBASE版本(v1.1.0或以上版本)
webase.version=v1.2.0

# 节点管理子系统mysql数据库配置
mysql.ip=localhost
mysql.port=3306
mysql.user=test
mysql.password=zz962910588
mysql.database=webasenodemanager
```

(3) 部署

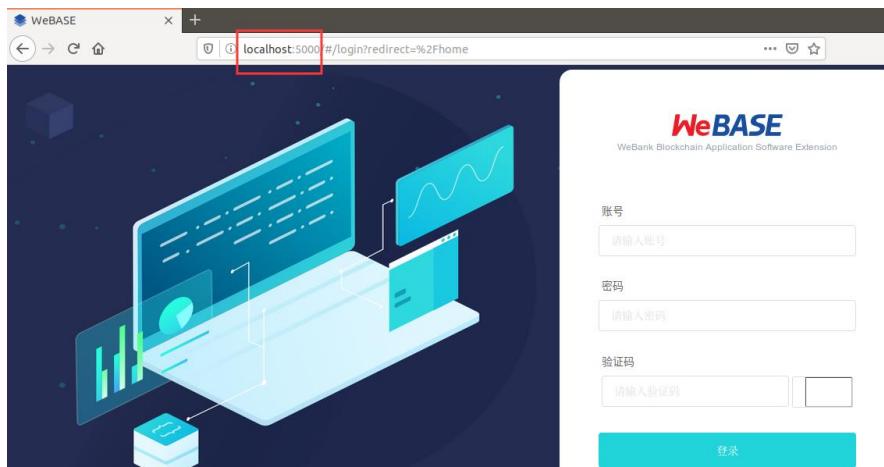
```
fisco-bcos@fiscobcos-VirtualBox:~/webase-deploy$ vi common.properties
fisco-bcos@fiscobcos-VirtualBox:~/webase-deploy$ python deploy.py installAll
=====
[REDACTED]
=====

=====
===== environment check... =====
check git...
check finished sucessfully.
check openssl...
check finished sucessfully.
check curl...
check finished sucessfully.
check nginx...
check finished sucessfully.
check java...
```

```
fisco-bcos@fiscobcos-VirtualBox: ~/webase-deploy
File Edit View Search Terminal Help
check finished sucessfully.
check WeBASE-Front port...
check finished sucessfully.
check database connection...
check finished sucessfully.
===== environent ready... =====
===== deploy start... =====
===== FISCO-BCOS install... =====
FISCO-BCOS节点目录nodes已经存在。是否重新安装？[y/n]y
[INFO] Downloading fisco-bcos binary from https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.0.0/fisco-bcos.tar.gz ...
% Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
          Dload  Upload Total Spent   Left Speed
100  608     0  608     0      0  383      0  --::--  0:00:01  --::--   383
  2 7908k    2 220k     0      0 35067      0  0:03:50  0:00:06  0:03:44 52019
curl: (28) Operation too slow. Less than 204800 bytes/sec transferred the last 3
seconds
```

全部回车，如果想要保留之前部署的合约就在数据库的地方输入 n.

(4) 进入网页 localhost:5000



初始账户为 admin，密码为 Abcd1234

(5) 输入配置

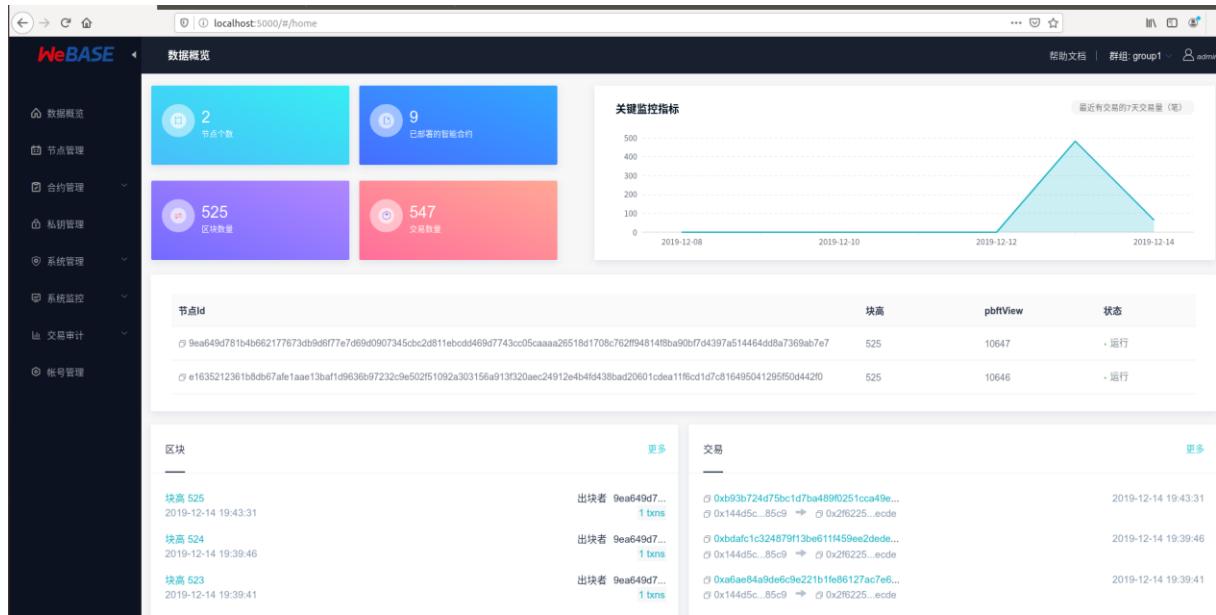
节点前置配置

ip

前置端口

所属机构

成功进入管理页面。



(实验结束才截得这个图)

d) Nodejs SDK 配置

[参考教程](#)为 fisco 官方文档，部分截图如下：

(1) Node.js 开发环境

```
fisco-bcos@fiscobcos-VirtualBox:~/git$ nvm install 8
v8.16.2 is already installed.
Now using node v8.16.2 (npm v6.4.1)
fisco-bcos@fiscobcos-VirtualBox:~/git$ 

fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ 
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ npm config set registry https://registry.npm.taobao.org
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
added 698 packages from 379 contributors in 27.938s
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ 
```

(2) 拉取源代码、使用 npm 安装依赖项

```
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ npm run repoclean
> nodejs-sdk@0.9.0 repoclean /home/fisco-bcos/nodejs-sdk
> lerna clean && npm run clean-solc-0.4 && npm run clean-solc-0.5

lerna notice cli v3.19.0
lerna info Removing the following directories:
lerna info clean packages/api/node_modules
lerna info clean packages/cli/node_modules
? Proceed? Yes
lerna info clean removing /home/fisco-bcos/nodejs-sdk/packages/api/node_modules
lerna info clean removing /home/fisco-bcos/nodejs-sdk/packages/cli/node_modules
lerna success clean finished

> nodejs-sdk@0.9.0 clean-solc-0.4 /home/fisco-bcos/nodejs-sdk
> rm -rf ./packages/api/common/solc-0.4/node_modules

> nodejs-sdk@0.9.0 clean-solc-0.5 /home/fisco-bcos/nodejs-sdk
> rm -rf ./packages/api/common/solc-0.5/node_modules
```

```
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ npm run bootstrap
> nodejs-sdk@0.9.0 bootstrap /home/fisco-bcos/nodejs-sdk
> lerna bootstrap && npm run install-solc-0.4 && npm run install-solc-0.5

lerna notice cli v3.19.0
lerna info Bootstrapping 2 packages
lerna info Installing external dependencies
lerna info hoist Installing hoisted dependencies into root
lerna info hoist Finished bootstrapping root
lerna info hoist Pruning hoisted dependencies
lerna info hoist Finished pruning hoisted dependencies
lerna info Symlinking packages and binaries
lerna success Bootstrapped 2 packages

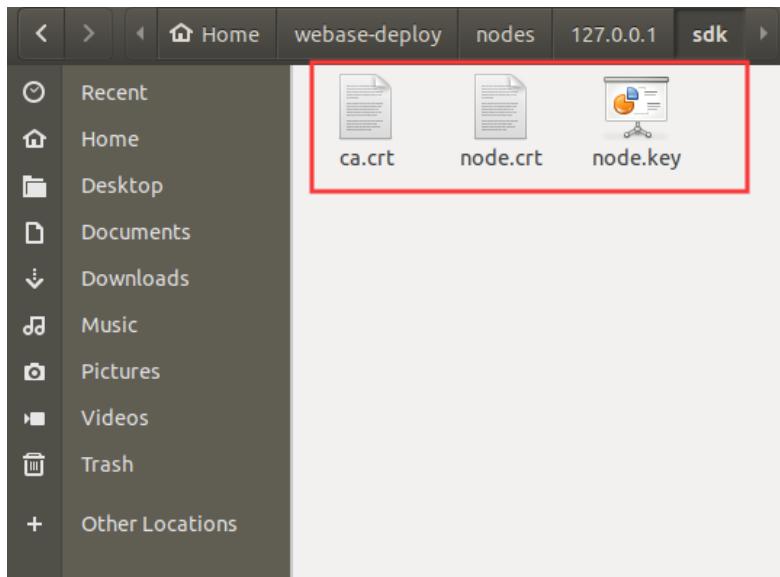
> nodejs-sdk@0.9.0 install-solc-0.4 /home/fisco-bcos/nodejs-sdk
> cd ./packages/api/common/solc-0.4/ && npm install

npm notice created a lockfile as package-lock.json. You should commit this file.
added 66 packages from 35 contributors in 3.816s

> nodejs-sdk@0.9.0 install-solc-0.5 /home/fisco-bcos/nodejs-sdk
> cd ./packages/api/common/solc-0.5/ && npm install

npm notice created a lockfile as package-lock.json. You should commit this file.
added 25 packages from 15 contributors in 4.98s
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$
```

(3) 配置证书及 Channel 端口



这 3 个文件夹的路径写入配置文件，如下：

config.json

```
{
  "privateKey": {
    "type": "pem",
    "value": "./0x144d5ca47de35194b019b6f11a56028b964585c9.pem"
  },
  "nodes": [
    {
      "ip": "127.0.0.1",
      "port": "20200"
    }
  ],
  "authentication": {
    "key": "/home/fisco-bcos/webase-deploy/nodes/127.0.0.1/sdk/node.key",
    "cert": "/home/fisco-bcos/webase-deploy/nodes/127.0.0.1/sdk/node.crt",
    "ca": "/home/fisco-bcos/webase-deploy/nodes/127.0.0.1/sdk/ca.crt"
  },
  "groupId": 1,
  "timeout": 10000
}
```

(4) 效果

修改前

```
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk/packages/cli$ ./cli.js getBlockNumber
error: ENOENT: no such file or directory, open '/path/to/sdk/key'
  at Object.fs.openSync (fs.js:646:18)
  at Object.fs.readFileSync (fs.js:551:33)
  at createNewSocket (/home/fisco-bcos/nodejs-sdk/packages/api/common/channelPromise.js:72:17)
  at channelPromise (/home/fisco-bcos/nodejs-sdk/packages/api/common/channelPromise.js:193:25)
  at Web3jService.getBlockNumber (/home/fisco-bcos/nodejs-sdk/packages/api/web3j/web3jService.js:48:16)
  at interfaces.push.produceSubCommandInfo (/home/fisco-bcos/nodejs-sdk/packages/cli/interfaces/web3j.js:35:29)
  at Object.handler (/home/fisco-bcos/nodejs-sdk/packages/cli/interfaces/base.js:29:27)
  at Object.yargs.command.needHelp [as handler] (/home/fisco-bcos/nodejs-sdk/packages/cli/cli.js:110:28)
  at Object.runCommand (/home/fisco-bcos/nodejs-sdk/node_modules/yargs/lib/command.js:242:26)
  at Object.parseArgs [as _parseArgs] (/home/fisco-bcos/nodejs-sdk/node_modules/yargs/yargs.js:107:30)
```

修改后

```
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk/packages/cli$ ./cli.js getBlockNumber
{"id":1,"jsonrpc":"2.0","result":"0x2"}
```

e) bugs 以及解决方案

- i. 调用不了 nodejs SDK，后端显示如下：

修改前

```
fisco-bcos@fiscobcos-VirtualBox:~/Desktop/login
File Edit View Search Terminal Help
at bootstrap_node.js:625:3
fisco-bcos@fiscobcos-VirtualBox:~/Desktop/login$ node server.js
Server running at http://127.0.0.1:8080/
主页 register-POST 请求
object
3
{ name: '213', password: '121', type: '1' }
(node:4979) UnhandledPromiseRejectionWarning: ConfigurationError: invalid configuration object or file path
  at new Configuration (/home/fisco-bcos/nodejs-sdk/packages/api/common/configuration.js:48:19)
  at Function.getInstance (/home/fisco-bcos/nodejs-sdk/packages/api/common/configuration.js:34:38)
  at Web3jService.get [as config] (/home/fisco-bcos/nodejs-sdk/packages/api/common/service/Base.js:28:50)
  at Web3jService.sendRawTransaction (/home/fisco-bcos/nodejs-sdk/packages/api/web3j/web3jService.js:391:42)
  at func (/home/fisco-bcos/Desktop/login/server.js:56:29)
  at app.post (/home/fisco-bcos/Desktop/login/server.js:40:18)
  at Layer.handle [as handle_request] (/home/fisco-bcos/Desktop/login/node_modules/express/lib/router/layer.js:95:5)
  at next (/home/fisco-bcos/Desktop/login/node_modules/express/lib/router/route.js:137:13)
  at /home/fisco-bcos/Desktop/login/node_modules/body-parser/lib/read.js:130:5
  at invokeCallback (/home/fisco-bcos/Desktop/login/node_modules/raw-body/index.js:224:16)
(node:4979) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing inside of an async function without a catch block, or by rejecting a promise which was not handled with .catch(). (rejection id: 3)
(node:4979) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process with a non-zero exit code.
```

解决方案：没有配置证书，修改配置文件，[参考教程](#)：

API调用约定

- 使用服务之前，首先需要初始化全局的 `Configuration` 对象，用以为各个服务提供必要的配置信息。`Configuration` 对象位于 `nodejs-sdk/packages/api/common/configuration.js`，其初始化参数为一个配置文件的路径或包含配置项的对象。配置文件的配置项说明见[配置说明](#)
- 如无特殊说明，Node.js SDK提供的API均为异步API。异步API的实际返回值是一个包裹了API返回值的Promise对象，开发者可以使用`async/await`语法或`then...catch...finally`方法操作该Promise对象以实现自己的程序逻辑
- 当API内部出现错误导致逻辑无法继续执行时（如合约地址不存在），均会直接抛出异常，所有异常均继承自Error类

ii. Node 报错为：

```
fisco-bcos@fiscobcos-VirtualBox:~/Desktop/login$ node ser.js
module.js:550
    throw err;
  ^

Error: Cannot find module '/home/fisco-bcos/Desktop/login/ser.js'
  at Function.Module._resolveFilename (module.js:548:15)
  at Function.Module._load (module.js:475:25)
  at Function.Module.runMain (module.js:694:10)
  at startup (bootstrap_node.js:204:16)
  at bootstrap_node.js:625:3
```

由于对 node 不熟悉，因此这个错误我蒙了很久，后面才发现是文件路径搞错了。。。

iii. 提交一个表单的时候，页面会一直不停刷新，一直 load

这是由于 HTML 的默认部件属性，认为如果一个部件的 `type = submit` 的话，会进入下一个页面，而我只有一个页面，因此会一直不停刷新，正在这个页面上的数据也没了。

解决方案：将 `type` 的 `submit` 更改为 `button`（其他属性应该也可以，不过我没试过）；