

区块链阶段三报告

第 11 小组

18342069 罗炜乐

18342066 鲁沛

18342071 马靖成

一、链段设计说明

公司

```
struct Company {  
    string name;           //名称  
    address addr;          //地址  
    uint asset;            //资产  
    Debt[] debts;          //欠条  
    Receipt[] receipts;    //收据  
}
```

公司采用结构体的形式定义，有名称，地址，资产，欠条集（应付账款）和收据集（应收账款）。所有账户都可以注册公司，并设置自己的初始资产。

银行

```
struct Bank {  
    string name;  
    address addr;  
}
```

银行是唯一的，由最初部署合约时的账户自动调用构造函数形成，只有银行具有创建账单（第三方可信机构确认交易），发行货币和融资的功能。

账单

一张账单由**收据**和**欠条**两部份组成，分别被**债权人**和**债务人**所持有。

收据：id 用于在债权人的收据集中索引该账单，borrer_id 用于在债务人的欠条集中索引该账单。

```
struct Receipt {  
    uint id;  
    uint borrrer_id;      //此账单在对应欠债方的欠条中的索引  
    address borrrer;  
    uint amount;  
}
```

欠条：id 用于在债务人的欠条集中索引该账单，creditor_id 用于在债权人的收据集中索引该账单。

```
struct Debt {  
    uint id;  
    uint creditor_id;     //此账单在对应债务方的收据中的索引  
    address creditor;  
    uint amount;  
}
```

由于 creditor_id 和 borrrer_id 的存在，使得收据和欠条可以一一对应起来并互相索引，形成一张完整的账单。

变量

```
mapping(address => Company) public companies;  
Bank public bank;
```

companies 是由地址到公司的映射，bank 则是合约中唯一的中央银行，拥有最高的权限，在构造函数中被初始化：

```
constructor() {  
    bank.name = "CentralBank";  
    bank.addr = msg.sender;  
}
```

事件

8 个事件作为后续函数的日志，分别是查询总负债额，查询总放债额，注册公司，发行货币，创建账单，应收账款转移，融资和结算。

```
event Debt_query(string name, uint amount);  
event Receipt_query(string name, uint amount);  
event Register_company(string name, address addr, uint asset);  
event Issue(address to, uint amount);  
event Create_bill(address from, address to, uint amount);  
event Transfer_bill(address from, address to, uint from_id, uint to_id, uint amount);  
event Finance(address to, uint amount);  
event Settle(address to, uint bill_id);
```

函数

```
function bank_info() public view returns(string memory, address)
```

说明：获取银行信息

```
function register_company(string memory name, uint asset) public
```

说明：注册公司

```
function getAsset() public returns(uint)
```

说明：获取公司总资产

```
function getDebt(uint bill_id) public returns(address, uint)
```

说明：根据欠条 id 获取欠条

```
function getReceipt(uint bill_id) public returns(address, uint)
```

说明：根据收据 id 获取收据

```
function get_total_debt() public returns(uint)
```

说明：获取一个公司总的应还账款（总负债额）

```
function get_total_receipt() public returns(uint)
```

说明：获取一个公司总的应收账款（总放债额）

```
function issue(address to, uint amount) public
```

说明：向指定方发行货币

```
function create_bill(address from, address to, uint amount) public
```

说明：签发账单

```
function transfer_bill(uint from_id, uint to_id, address to, uint amount) public
```

说明：应收账款转移，涉及到三个人和两个账单

```
function finance(address addr, uint amount) public
```

说明：融资

```
function settle(address to, uint bill_id)
```

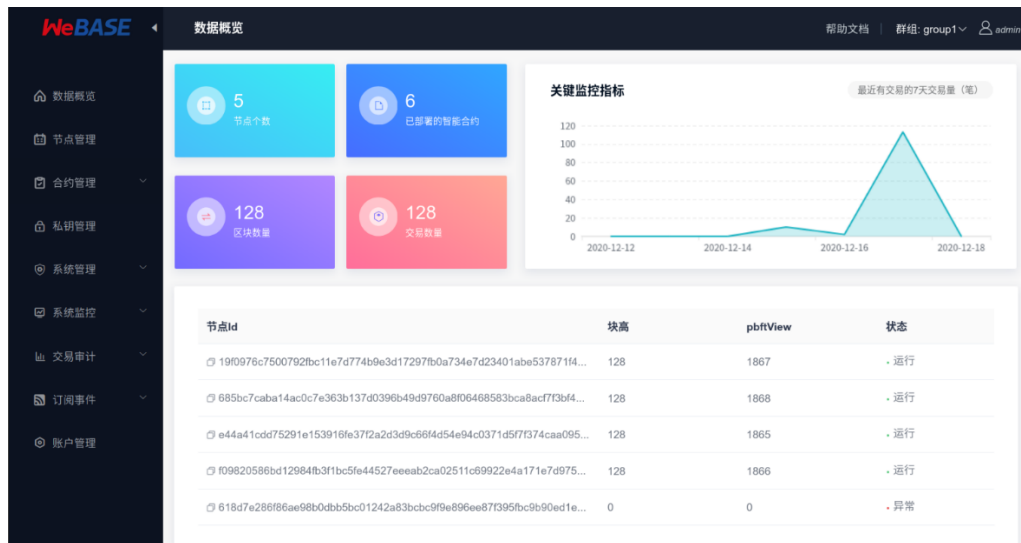
说明：结算

总体说明

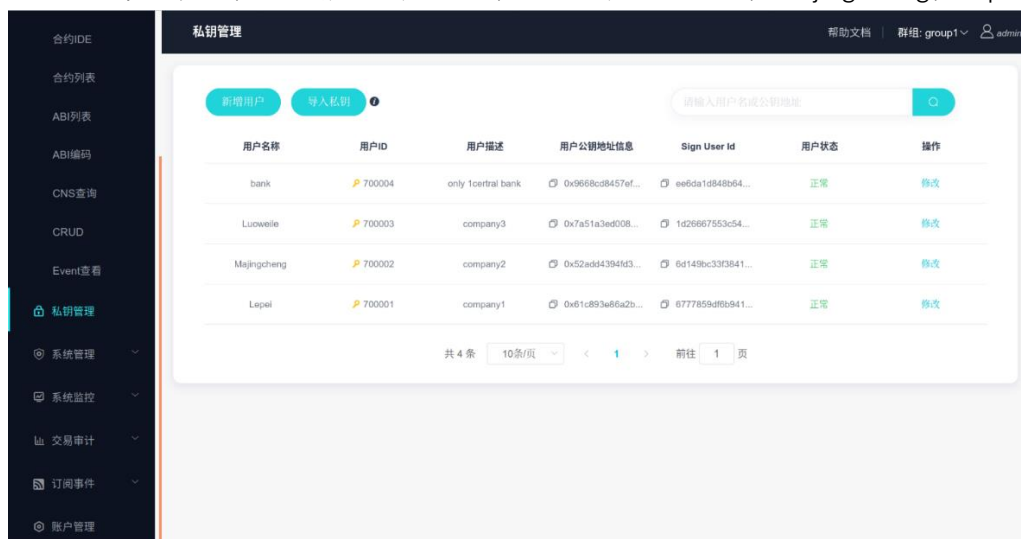
提前创建一个银行账户和几个公司账户，使用银行账户来部署合约。合约部署完成后，使用公司账户注册公司，然后即可开始创建账单等一系列操作。由于区块链的不可篡改特性和账本特性，可以解决需求中提出的问题。

二、链段功能测试

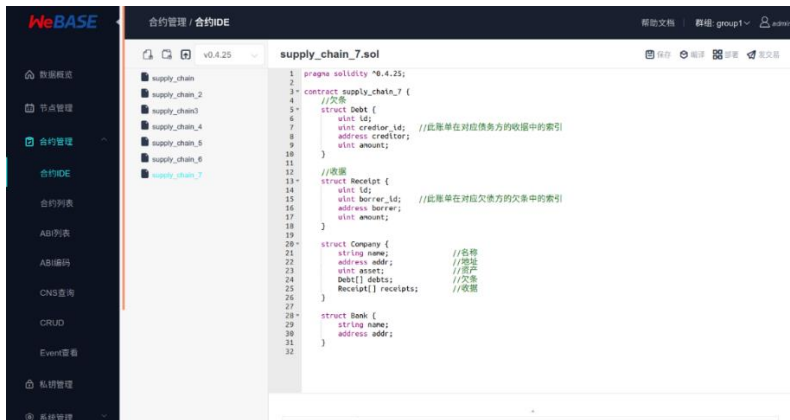
准备工作：首先安装 WeBASE 管理平台：



进入私钥管理页面，创建银行账户 bank，公司账户 Luoweile, Majingcheng, Lupei:



进入合约管理的合约 IDE:



上传合约，保存，编译，部署（使用 bank 账户），即可开始操作。

使用账户 Luoweile 创建汽车公司 Car：

发送交易

×

合约名称: supply_chain_9

合约地址: 0xbbb418507ef9e3c8f75d104c17ebf

用户: Luoweile

方法: function register_comp

参数:

name	Car
asset	1000

❗ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如: ["aaa","bbb"]和[100,101]; 如果数组参数包含双引号，需转义，例如: ["aaa\"bbb\",\"ccc"]。

取消 确定

使用账户 Majingcheng 创建轮胎公司 Tyre：

发送交易

×

合约名称: supply_chain_9

合约地址: 0xbbb418507ef9e3c8f75d104c17ebf

用户: Majingcheng

方法: function register_comp

参数:

name	Tyre
asset	1000

❗ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如: ["aaa","bbb"]和[100,101]; 如果数组参数包含双引号，需转义，例如: ["aaa\"bbb\",\"ccc"]。

取消 确定

利用账户 Lupei 创建轮毂公司 Hub：

发送交易

×

合约名称: supply_chain_9

合约地址: 0xbbb418507ef9e3c8f75d104c17ebf

用户: Lupei

方法: function register_comp

参数:

name	Hub
asset	1000

❗ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如: ["aaa","bbb"]和[100,101]; 如果数组参数包含双引号，需转义，例如: ["aaa\"bbb\",\"ccc"]。

取消 确定

功能一：实现采购商品—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。

实现函数：create_bill

实现思路：首先检查函数调用者是否是银行（只有银行才能签发账单），获取欠条在债务人的欠条集中的位置 borrar_id 和收据在债权人的收据集中的位置 creditor_id，然后利用 push 函数在债务人的欠条集和债权人的收据集中分别插入收据和欠条组成一张完整的收据。

```
function create_bill(address from, address to, uint amount) public {
    require(msg.sender == bank.addr, "only bank could create bill");
    uint borrar_id = companies[to].debts.length;
    uint credior_id = companies[from].receipts.length;

    companies[from].receipts.push(
        Receipt(credior_id, borrar_id, to, amount)
    );
    companies[to].debts.push(
        Debt(borrar_id, credior_id, from, amount)
    );
    emit Create_bill(from, to, amount);
}
```

使用 create_bill 函数创建 Car 公司欠 Tyre 公司 100 的账单：

发送交易

×

合约名称: supply_chain_7

合约地址: 0x4a2f9e2715edb93954906270f2ecc ⓘ

用户: bank

方法: function create_bill

参数:

from	0x52add4394fd35f454524
to	0x7a51a3ed00889da54dedt
amount	100

ⓘ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如: ["aaa","bbb"]和[100,101]; 如果数组参数包含双引号，需转义，例如: ["aaa\"bbb\",\"ccc"]。

取消

确定

交易回执

address: 0x4a2f9e2715edb93954906270f2ecd355d42e12c4
eventName : Create_bill(address from,address to,uint256 amount)
data:

name	data
from	0x52Ad4d4394fD35F45452450e6C36...
to	0x7a51A3ED00889da54dedB744B7C.
amount	100

还原

使用 getDebt 函数可以在 Car 公司查到欠 Tyre 公司 100 的欠条 0:

的

公

7

```

function transfer_bill(uint from_id, uint to_id, address to, uint amount) public {
    address before_addr = companies[msg.sender].receipts[from_id].borrer;
    //from_id <=> borrrer_id, to_id <=> credior_id
    uint borrrer_id = companies[msg.sender].receipts[from_id].borrer_id;
    uint credior_id = companies[msg.sender].debts[to_id].credior_id;

    // 若A欠B50, B欠C100, 转移50后, A欠C50, B欠C50, A不再欠B
    if(amount <= companies[msg.sender].receipts[from_id].amount) {
        companies[before_addr].debts[borrrer_id].amount -= amount;
        if(companies[before_addr].debts[borrrer_id].amount == 0)
        {
            delete companies[before_addr].debts[borrrer_id];
        }

        companies[msg.sender].receipts[from_id].amount -= amount;
        if(companies[msg.sender].receipts[from_id].amount == 0)
        {
            delete companies[msg.sender].receipts[from_id];
        }

        companies[msg.sender].debts[to_id].amount -= amount;
        if(companies[msg.sender].debts[to_id].amount == 0) {
            delete companies[msg.sender].debts[to_id];
        }

        companies[to].receipts[credior_id].amount += amount;
        if(companies[to].receipts[credior_id].amount == 0) {
            delete companies[to].receipts[credior_id];
        }

        // 新增一张A欠C50的账单
        companies[before_addr].debts.push(
            Debt(companies[before_addr].debts.length, companies[to].receipts.length, to, amount)
        );
        companies[to].receipts.push(
            Receipt(companies[to].receipts.length, companies[before_addr].debts.length, before_addr, amount)
        );
        emit Transfer_bill(msg.sender, to, from_id, to_id, amount);
    }
}

```

Tyre 调用 transfer_bill，将 Car 欠 Tyre 的 100 转给 Hub：

发送交易

×

合约名称: supply_chain_7

合约地址: 0x4a2f9e2715edb93954906270f2ecc ⓘ

用户: Majingcheng ▾

方法: function ▾ transfer_bill ▾

参数:

from_id 0

to_id 0

to 0x61c893e86a2bb5665247d

amount 100

ⓘ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如：["aaa","bbb"]和[100,101]；如果数组参数包含双引号，需转义，例如：["aaa\"bbb\",\"ccc"]。

取消 确定

交易回执

×

address: 0x4a2f9e2715edb93954906270f2ecd355d42e12c4

eventName : Transfer_bill(address from,address to,uint256 from_id,uint256 to_id,uint256 a mount)

data:

name

data

from

0x52Add4394fD35F45452450e6C3..

to

0x61c893e86a2BB5665247De0F92..

from_id

0

还原

此时的账单状态变更为：Car 不再欠 Tyre 钱 100，Tyre 不再欠 Hub 100，而 Car 欠 Hub 100。

Car 调用 getDebt 函数查看欠条 0：

交易回执

message: "success"
from: "0x7a51a3ed00889da54dedb744b7cebc294938c910"
to: "0x4a2f9e2715edb93954906270f2ecd355d42e12c4"
input: "0xc9dd1f3c00"
output: function getDebt(uint256 bill_id) returns(address, uint256)

data:	name	type	data
	address		0x00000000...
	uint256		0

Car不再欠Tyre100

还原

Tyre 调用 getDebt 函数查看欠条 0:

交易回执

message: "success"
from: "0x52add4394fd35f45452450e6c36c055ff3d93d0f"
to: "0x4a2f9e2715edb93954906270f2ecd355d42e12c4"
input: "0xc9dd1f3c00"
output: function getDebt(uint256 bill_id) returns(address, uint256)

data:	name	type	data
	address		0x00000000...
	uint256		0

Tyre不再欠Hub100

还原

Car 调用 getDebt 函数查看欠条 1:

交易回执

message: "success"
from: "0x7a51a3ed00889da54dedb744b7cebc294938c910"
to: "0x4a2f9e2715edb93954906270f2ecd355d42e12c4"
input: "0xc9dd1f3c0004"
output: function getDebt(uint256 bill_id) returns(address, uint256)

data:	name	type	data
	address		0x61c893e8...
	uint256		100

Car新增欠Hub100的欠条

还原

Tyre 调用 getReceipt 函数查看收据 1:

交易回执

message: "success"
from: "0x61c893e86a2bb5665247de0f9241d4f06bf9e93a"
to: "0x4a2f9e2715edb93954906270f2ecd355d42e12c4"
input: "0xb63e6ac30001"
output: function getReceipt(uint256 bill_id) returns(address, uint256)

data:	name	type	data
	address		0x7a51A3E...
	uint256		100

Hub新增Car欠其100的收据

还原

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。

实现函数：finance

实现思路：只有银行有资格融资（给公司发行货币），将公司总的应收账款作为其信用额度，申请的融资金额要小于信用额度。

```
function finance(address addr, uint amount) public {
    require(msg.sender == bank.addr, "only bank could finance");
    // 计算公司全部应收账款作为信用额度
    uint credit = 0;
    for(uint i = 0; i < companies[addr].receipts.length; ++i)
    {
        credit += companies[addr].receipts[i].amount;
    }
    require(credit >= amount, "credit of company is not enough");
    companies[addr].asset += amount;
    emit Finance(addr, amount);
}
```

Hub 向银行申请 50 的融资：

发送交易

×

合约名称: supply_chain_9

合约地址: 0xbbb418507ef9e3c8f75d104c17eb5

用户: bank

方法: function finance

参数: addr 47de0f9241d4f06bf9e93a
amount 50

❗ 如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如: ["aaa","bbb"]和[100,101]; 如果数组参数包含双引号，需转义，例如: ["aaa\"bbb","ccc"]。

取消 确定

交易回执

×

data:

name	data
to	0x61c893e86a2bb5665247De0F924..
amount	50

还原

Hub 调用 getAsset 查询自己的总资产，从 1000 变成了 1050：

交易回执

×

contractAddress: "0x00"
root: "0x00"
status: 0x0
message: "success"
from: "0x61c893e86a2bb5665247de0f9241d4f06bf9e93a"
to: "0xbbb418507ef9e3c8f75d104c17eb5ccb481819a5"
input: "0x5c222bad"
output: function getAsset() returns(uint256)

data:

name	type	data
	uint256	1050

还原

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。

实现函数：settle

实现思路：根据债务人的欠条 id 找到欠条中记录的债权人的收据 id (creditor_id)，然后检查债务人的资产是否足够结算，若足够，则将欠条和收据都删除掉，即删除此账单，并且从债务人的总资产中扣除相应金额。

```
function settle(address to, uint bill_id) {
    uint creditor_id = companies[msg.sender].debts[bill_id].creditor_id;
    uint amount = companies[msg.sender].debts[bill_id].amount;
    require(companies[msg.sender].asset >= amount, "asset is not enough to pay");
    companies[msg.sender].asset -= amount;
    delete companies[msg.sender].debts[bill_id];
    delete companies[to].receipts[creditor_id];
    emit Settle(to, bill_id);
}
```

Car 调用 settle 函数，结算欠 Hub 的 100:

发送交易

×

合约名称: supply_chain_9

合约地址: 0xbbb418507ef9e3c8f75d104c17ebf

用户: bank

方法: function finance

参数:

addr	0x61c893e86a2bb566524
amount	50

如果参数类型是数组，请按照以下格式输入，以逗号分隔，非数值和布尔值须使用双引号，例如: ["aaa","bbb"]和[100,101]; 如果数组参数包含双引号，需转义，例如: ["aaa\"bbb","ccc"]。

取消 确定

交易回执

×

address: 0xbbb418507ef9e3c8f75d104c17eb5ccb481819a5

eventName : Settle(address to,uint256 bill_id)

data:

name	data
to	0x61c893e86a2BB5665247De0F924...
bill_id	1

还原

Car 调用 getAsset 函数，查询自己的总资产，由于结算了 100，总资产从 1000 变为 900，而 Hub 的总资产从 1050 变为 1150:

交易回执

×

status: 0x0

message: "success"

from: "0x7a51a3ed00889da54dedb744b7cebc294938c910"

to: "0xbbb418507ef9e3c8f75d104c17eb5ccb481819a5"

input: "0x5c222bad"


output: function getAsset() returns(uint256)

data:

name	type	data
	uint256	900

交易回执



```
status: 0x0
message: "success"
from: "0x61c893e86a2bb5665247de0f9241d4f06bf9e93a"
to: "0x0bbb418507ef9e3c8f75d104c17eb5ccb481819a5"
input: "0x5c222bad"
output: function getAsset() returns(uint256)
```

name	type	data
	uint256	 1150

Car 查询自己的欠条 1, 已经被清空:

交易回执



[illegible]

name	type	data
	address	 0x00000000...
	uint256	 0

Hub 查询自己的收据 1, 同样被清空:

交易回执

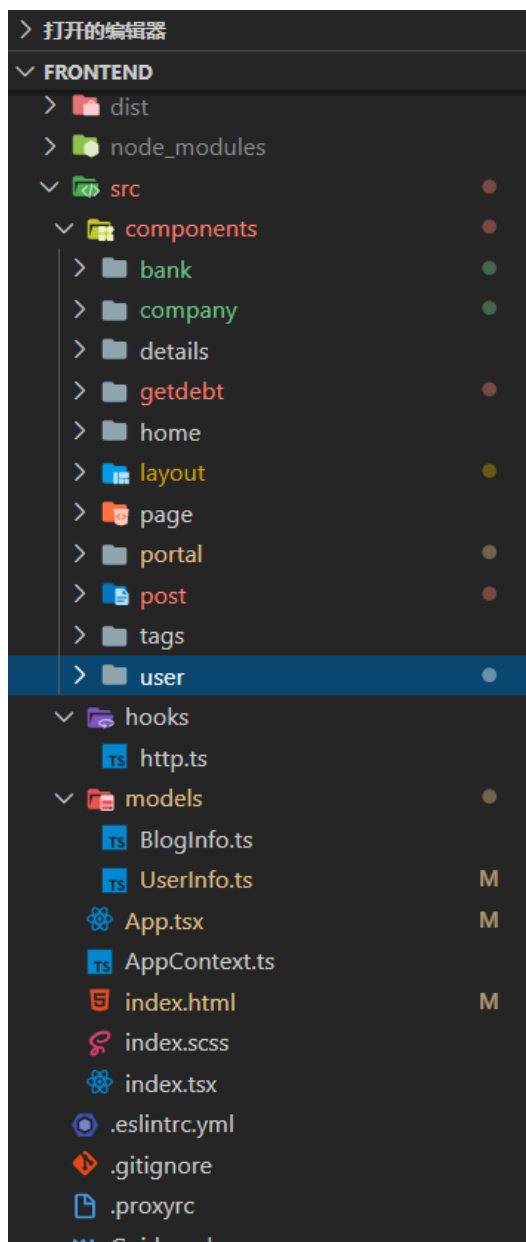
```
message: "success"  
from: "0x61c893ce86a2bb5665247de0f9241d4f06bf9e93a"  
to: "0xed02aab085bc16e33f3aa0a2fdaf7818011e9c2e"  
input: "0xb63eac3000000000000000000000000000000000000000000000000000000000000000000000  
0001"  
output: function getReceipt(uint256 bill_id) returns(address, uint256)
```

name	type	data
	address	 0x00000000...
	uint256	 0

则结算成功。

三、前端部分

(1)、项目框架：本次前端采用的是 JavaScript 的 react 框架。UI 库使用 Fluent UI。对于 UI 库的使用，参考 <https://developer.microsoft.com/en-us/fluentui#/controls/web> 项目结构组织如下：



我将链上的功能分解为了六个模块，利用侧边栏（layout）来管理：

- 1、“我的”（portal）部分，这个部分用于管理用户的登录、注册、登出。
- 2、“查看欠条与收据”（getdebt）部分，这个部分有四个功能，用户根据 id 查找自己的欠条、收据以及查看自己公司的总负债、总放债数额。
- 3、“个人公司”（company）部分，功能包括用户创建、修改自身公司的名字、资产。以及查看公司总资产。
- 4、“银行”（bank）部分，包括仅允许银行用户操作的两个功能：签发账单和融资。
- 5、“个人金融操作”（user）部分，包括应收账款转移和结算（还钱）两个功能
- 6、还额外提供了一个直接操作合约的平台以及对合约的使用的解释说明。

(2)、前端效果展示：

1、注册页面：

Block Chain

注册账户

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

用户名

密码

注册

登录

2、登录页面

Block Chain

登录账户

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

用户名

密码

登录

注册

3、登录后

Block Chain

欢迎, lp!

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

你的地址: 0xab796f5616fba683709e7fbbeff7f043e5f6f1a3

退出

4、查看欠条与收据页面

Block Chain

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

查看收据或欠条

按id查看收据或欠条

选择需要查看的类型

查看收据

ID

确认ID

当前D: 0

返回结果:

债权人/债务人地址: 0xef45426e88d42d8cb8e6ea62bb22208dc82d8577

金额: 99

按id查看

查看公司总负债或放债

选择需要查看的类型

选择查看类型

返回结果:

总金额: 99

查看总数

5、个人公司页面

Block Chain

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

个人公司

创建/修改公司名字与资产

每个用户只能最多有一个公司，每次操作都会覆盖之前的值

公司名字

CAR

公司总资产

250

修改公司名字与资产

查看公司总负债或放债

返回结果:

总资产: 250

查看当前总资产

6、银行页面

Block Chain

银行

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

签发账单

让两个用户之间建立债务关系

债权人地址

债务人地址

债务金额

签发账单

融资

融资金额需小于融资者的总放债额度，融资后公司资产增加相应数值

融资人地址

融资金额

融资

7、个人金融操作部分：

Block Chain

个人金融操作

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

应收账款转移

将自己持有的收据里的钱转移到自己的欠条里。举例：你欠A100元(记录该关系的欠条id为1)，B欠你200元(记录该关系的收据id为2)。transfer_bill(2,1,A的地址，100)，就会让B欠你100，你不欠A的钱

收据id

欠条id

债权人地址

转移金额

债务转移

结算(还钱)

从自身资产中扣除对应欠条的数额并删去欠条

债权人地址

欠条id

结算

8、直接调用合约部分

Block Chain

我的

查看欠条与收据

个人公司

银行(仅银行账户可以使用)

个人金融操作

直接操作合约及合约说明

调用合约

选择需要调用的合约

bank_info

参数

添加参数

已加参数:

返回结果:

合约使用说明

1. bank_info, 获取银行信息。不需要参数。返回值包含两项, 含义分别为: (1)、中央银行的名字。(2)中央银行的地址

2. register_company, 注册个人公司, 需注意个人最多只能有一个公司, 多次使用会覆盖之前的结果。需要两个参数, 分别为公司的名字和初始资金。无返回值

3. getAsset, 获取公司总资产。不需要参数。返回值包括一项, 就是公司的总资产

4. getDebt, 根据欠条 id 获取欠条。需要一个参数即欠条id, 欠条id从0开始。返回值有两项, 分别为债权人的address, 以及欠他的数额

5. getReceipt, 根据收据 id 获取收据。需要一个参数即收据id, 欠条id从0开始。返回值有两项, 分别为债务人的address, 以及他欠的数额

6. get_total_debt, 获取一个公司总的应还款项(总负债额)。不需要参数。返回值有一项, 为总负债数额

7. get_total_receipt, 获取一个公司总的应收账款(总放债额)。不需要参数。返回值有一项, 为总放债数额

8. create_bill, 签发账单, 只有银行账户可以操作, 让两个用户之间建立债务关系。需要三个参数, 分别为(1)债权人的地址(2)债务人的地址(3)借款数额。无返回值

9. transfer_bill, 应收账款转移, 将自己持有的收据里的钱转移到自己的欠条里。需要四个参数, 分别为(1)收据的id(2)欠条的id(3)欠条的债权人(4)转移的数值。无返回值。

transfer_bill使用举例, 你欠A100元(记录该关系的欠条id为1), B欠你200元(记录该关系的收据id为2), transfer_bill(2,1,A的地址, 100), 就会让B欠你100, 你不欠A的钱

10. finance, 融资, 只有银行账户可以操作, 需要两个参数即融资者的地址和融资额度(需小于融资者的总放债额度), 融资后公司资产增加相应数值。无返回值

11. settle, 结算(还钱), 从自身资产中扣除对应欠条的数额并删去欠条。需要两个参数即还款人地址和欠条id。无返回值。

运行合约

(3)、前端实现关键部分代码:(实现过程的较为详细教学放在前端的 guide.md, 这里只说最关键的部分)

1、将输入框中的内容绑定为状态, 便于改动页面上的元素且便于作为参数传递。

以 Login 页面中的用户名和密码为例, 演示这一过程的重要性。

一、这部分是定义状态和改变状态的函数, 以及如何利用状态来作为参数发送 http 请求

```
const Login: React.FunctionComponent = () => {
  const { setUser } = React.useContext(AppContext);
  const [name, setName] = React.useState<string>(); // 用户名的状态
  const [password, setPassword] = React.useState<string>(); // 密码的状态
  const loginRequest = useHttp<{ status: string; }>("/api/user/login", "POST"); // 与后端交互的url
  const userInfoRequest = useHttp<UserInfo>("/api/user/self", "GET"); // 与后端交互的url
  // 利用绑定的状态作为参数求发post与get请求
  React.useEffect(() => {
    if (userInfoRequest.data && !userInfoRequest.loading && setUser) {
      console.log(userInfoRequest.data);
      setUser({
        address: userInfoRequest.data.address,
        username: userInfoRequest.data.username,
      });
    }
  }, [userInfoRequest.loading, userInfoRequest.data]);

  React.useEffect(() => {
    if (!loginRequest.loading) {
      if (loginRequest.data?.status === "success") {
        userInfoRequest.fire();
        setError(undefined);
      }
      else if (!loginRequest.data || !loginRequest.ok) {
        // setError("登录失败");
        // alert("密码或者账户名错误");
      }
      setType("not defined");
    }
  }, [loginRequest.loading, loginRequest.data, loginRequest.ok]);
};
```

二、在这个部分中, 将输入框中的内容绑定在状态中。

注意到我们调用了上面的 setname 和 setpassword

```
<Stack.Item styles={{ root: { paddingTop: 10, width: 300 } }}>
  <TextField label="用户名" defaultValue={name} onChange={(_, v) => setName(v)} />
</Stack.Item>
<Stack.Item styles={{ root: { paddingTop: 10, width: 300 } }}>
  <TextField label="密码" canRevealPassword={true} type="password" defaultValue={password} onChange={(_, v) => setPassword(v)} />
</Stack.Item>
```

三、以“直接调用合约”部分的选择合约的多选框为例, 演示多选框内容以及多个参数的绑定


```

const Post: React.FunctionComponent = () => {
  const { user, setSelectedKey } = React.useContext(AppContext);
  const [title, setTitle] = React.useState<string>(); //TITLE 即为合约名
  const [currTags, setCurrTags] = React.useState<string>();
  const [tags, setTags] = React.useState<string[]>([]); // 合约参数
  const [results, setresults] = React.useState<string[]>([]);
  const [text, setText] = React.useState<string>();
  const [type, setType] = React.useState<string>();
  const [selectedItem, setSelectedItem] = React.useState<IDropdownOption>();

  React.useEffect(() => {
    setSelectedKey && setSelectedKey("post");
  }, [])

  React.useEffect(() => {
    // console.log("title:", title);
    console.log("tags:", tags);
    // console.log("text:", text);
  }, [tags]);

  const post = () => { // 发送http请求, 参数包括合约名和合约参数
    fetch(postUrl, {
      method: "POST",
      credentials: "include",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ funcName: title, funcParam: tags, text: text })
    }).then(res => res.json()).then(data => {
      //console.log(data);
      //console.log(data.result);
      console.log(data.result[0]);
      setresults(data.result);
      setTags([]);
      setType(data.result);
    });
  };
};

```

多选框的内容定义如下（就是我们的所有合约）：

```

01.   const options: IDropdownOption[] = [
02.     { key: 'bank_info', text: 'bank_info' },
03.     { key: 'register_company', text: 'register_company' },
04.     { key: 'getAsset', text: 'getAsset' },
05.     { key: 'getDebt', text: 'getDebt' },
06.     { key: 'getReceipt', text: 'getReceipt' },
07.     { key: 'get_total_debt', text: 'get_total_debt' },
08.     { key: 'get_total_receipt', text: 'get_total_receipt' },
09.     { key: 'create_bill', text: 'create_bill' },
10.     { key: 'transfer_bill', text: 'transfer_bill' },
11.     { key: 'finance', text: 'finance' },
12.     { key: 'settle', text: 'settle' },
13.   ];

```

然后我们在多选框中调用 `onChange` 函数记录当前选中的 `title`(合约名)，在“添加参数”按钮中记录当前的参数列表。

```

<Dropdown
  placeholder="选择合约"
  label="选择需要调用的合约"
  selectedKey={selectedItem ? selectedItem.key : undefined}
  // defaultSelectedKeys={['apple', 'banana', 'grape']}
  options={options}
  styles={dropdownStyles}
  onChange={onChange}
/>
<Stack.Item styles={{ root: { paddingTop: 10, width: 1000 } }}>
  <Stack horizontal>
    <Stack.Item>
      <TextField value={currTags} label="参数" onChange={(_, v) => setCurrTags(v)} />
    </Stack.Item>
    <Stack.Item styles={{ root: { paddingLeft: 10, paddingTop: 30 } }}>
      <PrimaryButton text="添加参数" onClick={() => {
        if (currTags) {
          setTags([...tags, currTags!]);
          setCurrTags("");
        }
      }} />
    </Stack.Item>
  </Stack>
  <Stack.Item styles={{ root: { paddingTop: 40, paddingLeft: 20 } }}>
    <i>已加参数: </i>
    {
      tags.map(
        (tag, index) => {
          return <i key={index}>{ " " + tag}</i>;
        }
      )
    }
  </Stack.Item>
</Stack>
</Stack.Item>

```

2、演示发送 http 请求和解析的过程

这个部分应该是前后端交互的重中之重，react 中有两种发送请求的方法。一种是提前预设好请求类型和返回体，然后利用 fire 函数去发送

```
01. const loginRequest = useHttp<{ status: string; }>("/api/user/login", "POST");
02. const login = () => {
03.   loginRequest.fire({
04.     username: name,
05.     password: password
06.   });
07.   setType("login");
08. }
```

另一种则是在发送时定义变量、类型等，我们以“个人公司”页面中的两个功能为例，一个是改公司名字与资产的请求 post1(需要两个参数，在代码中命名为 title, title2.).另一个是查看公司总资产的请求 post2（无参数）。

```
01. const post = () => {
02.   fetch(postUrl, {
03.     method: "POST",
04.     credentials: "include",
05.     headers: { "Content-Type": "application/json" },
06.     body: JSON.stringify({ funcName: "register_company", funcParam: [title, title2], text: text })
07.   }).then(res => res.json()).then(data => {
08.     setTitle('');
09.     setTitle2('');
10.
11.     console.log(data.result[0]);
12.
13.   });
14. };
15. const post2 = () => {
16.   fetch(postUrl, {
17.     method: "POST",
18.     credentials: "include",
19.     headers: { "Content-Type": "application/json" },
20.     body: JSON.stringify({ funcName: "getAsset", funcParam: [], text: text })
21.   }).then(res => res.json()).then(data => {
22.     //console.log(data);
23.     //console.log(data.result);
24.     console.log(data.result[0]);
25.     setMoney(data.result[0].data);
26.     setTags([]);
27.     setType(data.result);
28.   });
29. }
```

可以看到我们在发出时才定义了 http 请求的类型，和 content-type 以及参数的 key-value 值。

然后是 http 返回参数的解析，react 将返回参数解析为字典对象。所以一种解析方法就像上图中的，直接取 data.result[0]，就取到返回值中 key 为 result 的字段。

另外一种情况下，我们也许需要将 data 中的参数都完整的展示出来，这个时候我们就需要用到 JSON.parse，将返回的字典对象再解析为 json 字符串，然后输出。在“直接调用合约”部分我们就是这样展示结果的。如下：

```
30. <Stack.Item styles={{ root: { paddingTop: 20, paddingLeft: 0 } }}>
31.   <Label>返回结果: </Label>
32.   {
33.     results.map(
34.       (result, index) => {
35.         return <i key={index}>{" " + JSON.stringify(result)}</i>;
36.       }
37.     )
38.   }
39. </Stack.Item>
```

3、演示发送 react 如何将 http 元素和 JavaScript 元素结合，到达渲染和绑定同时进行。

在本次项目中，我使用了 fluentUI 的 stack 来简单的组织元素。以个人公司的元素为例

个人公司

创建/修改公司名字与资产

每个用户只能最多有一个公司，每次操作都会覆盖之前的值

公司名字

公司总资产

修改公司名字与资产

查看公司总负债或放债

返回结果:

总资产:

查看当前总资产

代码组织如下:

```
const BeforePost =
  <Stack>
    <Stack.Item>
      <Text variant="xxLarge">个人公司</Text>
    </Stack.Item>
    <br></br>
    <Stack.Item>
      <Text variant="large">创建/修改公司名字与资产</Text>
    </Stack.Item>
    <Label>每个用户只能最多有一个公司，每次操作都会覆盖之前的值</Label>

    <Stack.Item styles={{ root: { paddingTop: 10, width: 300 } }}>
      <TextField label="公司名字" onChange={(_, v) => setTitle(v)} />
    </Stack.Item>
    <Stack.Item styles={{ root: { paddingTop: 10, width: 300 } }}>
      <TextField label="公司总资产" onChange={(_, v) => setTitle2(v)} />
    </Stack.Item>

    <Stack.Item styles={{ root: { paddingTop: 10, width: 300 } }}>
      <PrimaryButton text="修改公司名字与资产" onClick={post} />
    </Stack.Item>

    <br></br>
    <br></br>
    <Stack.Item>
      <Text variant="large">查看公司总负债或放债</Text>
    </Stack.Item>

    <Stack.Item styles={{ root: { paddingTop: 20, paddingLeft: 0 } }}>
      <Label>返回结果: </Label>
      <Label>总资产: {money}</Label>

    </Stack.Item>

    <Stack.Item styles={{ root: { paddingTop: 10, width: 300 } }}>
      <PrimaryButton text="查看当前总资产" onClick={post2} />
    </Stack.Item>
```

四、后端部分

所有 API 的使用方法在 backend guide.md 文件中

(1)、项目框架: 本次后端采用的是 go 语言的 gin 框架。原因: Gin 是一个 golang 的微框架, 封装比较优雅, API 友好, 源码注释比较明确, 具有快速灵活, 容错方便等特点。并且对于 golang 而言, web 框架的依赖要远比 Python, Java 之类的要小。自身的 net/http 足够简单, 性能也非常不错。借助框架开发, 可以省去很多常用的封装带来的时间。

(2)、数据库: 本次项目使用的是 BoltDB, Bolt 是一个纯粹 Key/Value 模型的程序。该项目的目标是为不需要完整数据库服务器 (如 Postgres 或 MySQL) 的项目提供一个简单, 快

速,可靠的数据库。BoltDB 只需要将其链接到你的应用程序代码中即可使用 BoltDB 提供的 API 来高效的存取数据。而且 BoltDB 支持完全可序列化的 ACID 事务,让应用程序可以更简单的处理复杂操作。其源码地址为:<https://github.com/boltdb/bolt>

(3)、基于 token 的鉴权机制

(4)、本后端实现没用使用 sdk,而是直接使用 webase-front 链端 API 和链端打交道

首先,初始化数据库。并且确保中央银行一定在数据库中

```
1. func dbInit() {
2.   db, err := bolt.Open("blockchain.db", 0600, nil)
3.   if err != nil {
4.     log.Fatal(err)
5.   }
6.   db.Update(func(tx *bolt.Tx) error {
7.     _, err1 := tx.CreateBucketIfNotExists([]byte("account"))
8.     if err1 != nil {
9.       log.Fatal(err)
10.    }
11.    return nil
12.  })
13.  db.Update(func(tx *bolt.Tx) error {
14.    _, err2 := tx.CreateBucketIfNotExists([]byte("address"))
15.    if err2 != nil {
16.      log.Fatal(err)
17.    }
18.    return nil
19.  })
20.  db.Close()
21.  fmt.Println(dbSearch("account", "bank"))
22.  if dbSearch("account", "bank") == "" {
23.    dbInsert("account", "bank", "bank")
24.    dbInsert("address", "bank", "0xa49a7036e0eeb1190918798b446c8be1
    59b0b8bc")
25.  }
26. }
```

jwt token 的使用

```
1. var (
2.   Secret      = "blockchain"
3.   ExpireTime = 3600
4. )
5.
6. type JWTClaims struct {
```

```

7.  jwt.StandardClaims
8.  Password string `json:"password"`
9.  UserName string `json:"username"`
10. Address string `json:"address"`
11.}
12.
13.func getToken(claims *JWTClaims) (string, error) {
14. token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
15. signedToken, err := token.SignedString([]byte(Secret))
16. if err != nil {
17. return "", err
18. }
19. return signedToken, nil
20.}
21.
22.func verifyToken(strToken string) (*JWTClaims, error) {
23. token, err := jwt.ParseWithClaims(strToken, &JWTClaims{}, func(t
oken *jwt.Token) (interface{}, error) {
24. return []byte(Secret), nil
25. })
26. if err != nil {
27. return nil, err
28. }
29. claims, ok := token.Claims.(*JWTClaims)
30. if !ok {
31. return nil, err
32. }
33. if err := token.Claims.Valid(); err != nil {
34. return nil, err
35. }
36. return claims, nil
37.}

```

注册操作，如果在数据库中查询不到该用户，则向数据库插入用户名密码，并调用链端 API 注册

```

1. func register(c *gin.Context) {
2. var registerInfo registerModel
3. c.Bind(&registerInfo)
4.
5. if registerInfo.UserName == "bank" {
6. c.JSON(http.StatusForbidden, gin.H{
7. "status": "cannot register bank",
8. })

```

```

9.     return
10. }
11.
12. if dbSearch("account", registerInfo.UserName) != "" {
13.     c.JSON(http.StatusForbidden, gin.H{
14.         "status": "username already exists",
15.     })
16. } else {
17.     dbInsert("account", registerInfo.UserName, registerInfo.Password)
18.     chainReturn := get("http://localhost:5002/WeBASE-Front/privateKey?type=0&userName=" + registerInfo.UserName)
19.     var data map[string]string
20.     json.Unmarshal([]byte(chainReturn), &data)
21.     dbInsert("address", registerInfo.UserName, data["address"])
22.     claims := &JWTClaims{
23.         UserName: registerInfo.UserName,
24.         Password: registerInfo.Password,
25.         Address:  data["address"],
26.     }
27.     claims.IssuedAt = time.Now().Unix()
28.     claims.ExpiresAt = time.Now().Add(time.Second * time.Duration(ExpirationTime)).Unix()
29.     signedToken, _ := getToken(claims)
30.     c.SetCookie("jwt-token", signedToken, 3600, "/", "", false, true)
31.     c.JSON(http.StatusOK, gin.H{
32.         "status": "success",
33.     })
34. }
35. }

```

登陆操作，直接向数据库查询是否存在 token，如果存在就设置 token

```

1. func login(c *gin.Context) {
2.     var loginInfo loginModel
3.     c.Bind(&loginInfo)
4.     status := "not defined"
5.     password := dbSearch("account", loginInfo.UserName)
6.     if password != loginInfo.Password {
7.         // 如果未查询到对应字段则...
8.         status = "not found"
9.     } else {
10.        status = "success"

```

```

11. }
12.
13. claims := &JWTClaims{
14.   UserName: loginInfo.UserName,
15.   Password: loginInfo.Password,
16.   Address: dbSearch("address", loginInfo.UserName),
17. }
18. claims.IssuedAt = time.Now().Unix()
19. claims.ExpiresAt = time.Now().Add(time.Second * time.Duration(Ex
   expireTime)).Unix()
20. signedToken, _ := getToken(claims)
21. c.SetCookie("jwt-
   token", signedToken, 3600, "/", "", false, true)
22.
23. c.JSON(http.StatusOK, gin.H{
24.   "status": status,
25. })
26.}

```

通过后端使用链端合约的方法时, 登录后则直接使用当前账户调用合约, 不需要多余的参数。该设计使得前端和后端的交互更简单快捷。

```

1. func trans(c *gin.Context) {
2.   var transInfo transModel
3.   c.Bind(&transInfo)
4.
5.   strToken, err := c.Cookie("jwt-token")
6.   claims, err := verifyToken(strToken)
7.   if err != nil {
8.     c.String(401, err.Error())
9.     return
10.  }
11.   claims.ExpiresAt = time.Now().Unix() + (claims.ExpiresAt - claim
   s.IssuedAt)
12.   signedToken, err := getToken(claims)
13.   if err != nil {
14.     c.String(500, err.Error())
15.     return
16.  }
17.
18.   c.SetCookie("jwt-
   token", signedToken, 3600, "/", "", false, true)
19.

```

```
20. abiString := "[{"constant":false,"inputs":[],"name":"get_
total_receipt","outputs":[{"name":"total_receipt","type"
:"uint256"}],"payable":false,"stateMutability":"nonpayable
","type":"function"},{"constant":false,"inputs":[{"name
":"to","type":"address"},{"name":"bill_id","type":"
uint256"}],"name":"settle","outputs":[],"payable":false,
"stateMutability":"nonpayable","type":"function"},{"cons
tant":false,"inputs":[{"name":"from","type":"address"}
,{"name":"to","type":"address"},{"name":"amount","ty
pe":"uint256"}],"name":"create_bill","outputs":[],"paya
ble":false,"stateMutability":"nonpayable","type":"functio
n"},{"constant":true,"inputs":[{"name":"","type":"add
ress"}],"name":"companies","outputs":[{"name":"name",\
"type":"string"},{"name":"addr","type":"address"},{"n
ame":"asset","type":"uint256"}],"payable":false,"stateM
utability":"view","type":"function"},{"constant":false,\
"inputs":[],"name":"getAsset","outputs":[{"name":"asset
","type":"uint256"}],"payable":false,"stateMutability":"\
nonpayable","type":"function"},{"constant":false,"inputs
":[],"name":"get_total_debt","outputs":[{"name":"total_
debt","type":"uint256"}],"payable":false,"stateMutability
":"nonpayable","type":"function"},{"constant":true,"inp
uts":[],"name":"bank","outputs":[{"name":"name","type
":"string"},{"name":"addr","type":"address"}],"payabl
e":false,"stateMutability":"view","type":"function"},{"
constant":false,"inputs":[{"name":"addr","type":"addres
s"},{"name":"amount","type":"uint256"}],"name":"finan
ce","outputs":[],"payable":false,"stateMutability":"nonpa
yable","type":"function"},{"constant":false,"inputs":[{"\
name":"to","type":"address"},{"name":"amount","type\
":"uint256"}],"name":"issue","outputs":[],"payable":fal
se,"stateMutability":"nonpayable","type":"function"},{"c
onstant":false,"inputs":[{"name":"name","type":"string\
"},{"name":"asset","type":"uint256"}],"name":"register
_company","outputs":[],"payable":false,"stateMutability":"\
nonpayable","type":"function"},{"constant":false,"inputs
":[{"name":"from_id","type":"uint256"},{"name":"to_id
","type":"uint256"},{"name":"to","type":"address"},{\
name":"amount","type":"uint256"}],"name":"transfer_bi
ll","outputs":[],"payable":false,"stateMutability":"nonpa
yable","type":"function"},{"constant":false,"inputs":[],
"name":"bank_info","outputs":[{"name":"bank_name","typ
e":"string"},{"name":"bank_address","type":"address"}]
,"payable":false,"stateMutability":"nonpayable","type":"
```



```

function\"},{\"constant\":false,\"inputs\": [{\"name\": \"bill_id\",
\"type\": \"uint256\"}], \"name\": \"getReceipt\", \"outputs\": [{\"n
ame\": \"borrer\", \"type\": \"address\"}, {\"name\": \"amount\", \"typ
e\": \"uint256\"}], \"payable\": false, \"stateMutability\": \"nonpaya
ble\", \"type\": \"function\"}, {\"constant\": false, \"inputs\": [{\"n
ame\": \"bill_id\", \"type\": \"uint256\"}], \"name\": \"getDebt\", \"o
utputs\": [{\"name\": \"creditor\", \"type\": \"address\"}, {\"name\":
\"amount\", \"type\": \"uint256\"}], \"payable\": false, \"stateMutabi
lity\": \"nonpayable\", \"type\": \"function\"}, {\"inputs\": [], \"pay
able\": false, \"stateMutability\": \"nonpayable\", \"type\": \"constr
uctor\"}, {\"anonymous\": false, \"inputs\": [{\"indexed\": false, \"na
me\": \"name\", \"type\": \"string\"}, {\"indexed\": false, \"name\": \"
amount\", \"type\": \"uint256\"}], \"name\": \"Debt_query\", \"type\":
\"event\"}, {\"anonymous\": false, \"inputs\": [{\"indexed\": false, \"
name\": \"name\", \"type\": \"string\"}, {\"indexed\": false, \"name\":
\"amount\", \"type\": \"uint256\"}], \"name\": \"Receipt_query\", \"ty
pe\": \"event\"}, {\"anonymous\": false, \"inputs\": [{\"indexed\": fal
se, \"name\": \"name\", \"type\": \"string\"}, {\"indexed\": false, \"na
me\": \"addr\", \"type\": \"address\"}, {\"indexed\": false, \"name\": \"
asset\", \"type\": \"uint256\"}], \"name\": \"Register_company\", \"t
ype\": \"event\"}, {\"anonymous\": false, \"inputs\": [{\"indexed\": fa
lse, \"name\": \"to\", \"type\": \"address\"}, {\"indexed\": false, \"na
me\": \"account\", \"type\": \"uint256\"}], \"name\": \"Issue\", \"type
\": \"event\"}, {\"anonymous\": false, \"inputs\": [{\"indexed\": false
, \"name\": \"from\", \"type\": \"address\"}, {\"indexed\": false, \"nam
e\": \"to\", \"type\": \"address\"}, {\"indexed\": false, \"name\": \"am
ount\", \"type\": \"uint256\"}], \"name\": \"Create_bill\", \"type\": \"
event\"}, {\"anonymous\": false, \"inputs\": [{\"indexed\": false, \"n
ame\": \"from\", \"type\": \"address\"}, {\"indexed\": false, \"name\":
\"to\", \"type\": \"address\"}, {\"indexed\": false, \"name\": \"from_id
\", \"type\": \"uint256\"}, {\"indexed\": false, \"name\": \"to_id\", \"
type\": \"uint256\"}, {\"indexed\": false, \"name\": \"amount\", \"typ
e\": \"uint256\"}], \"name\": \"Transfer_bill\", \"type\": \"event\"},
{\"anonymous\": false, \"inputs\": [{\"indexed\": false, \"name\": \"to
\", \"type\": \"address\"}, {\"indexed\": false, \"name\": \"amount\", \"
type\": \"uint256\"}], \"name\": \"Finance\", \"type\": \"event\"}, {\"
anonymous\": false, \"inputs\": [{\"indexed\": false, \"name\": \"to\"
, \"type\": \"address\"}, {\"indexed\": false, \"name\": \"bill_id\", \"
type\": \"uint256\"}], \"name\": \"Settle\", \"type\": \"event\"}]\"

```

```

21. var tempABI interface{}
22. err = json.Unmarshal([]byte(abiString), &tempABI)
23. if err != nil {
24.     fmt.Println(err)
25. }

```

```

26.
27. toChain := make(map[string]interface{})
28. toChain["user"] = claims.Address
29. toChain["contractName"] = "supply_chain"
30. toChain["contractAddress"] = "0xdcdf32d05308d16bbd84564b7b9a5818
    ff7c4256"
31. toChain["contractAbi"] = tempABI
32. toChain["groupId"] = "1"
33. toChain["funcName"] = transInfo.FuncName
34. toChain["funcParam"] = transInfo.FuncParam
35.
36. toChainData, _ := json.Marshal(toChain)
37. txReturn := post("http://localhost:5002/WeBASE-
    Front/trans/handle", toChainData, "application/json")
38.
39. tempToDecode := make(map[string]string)
40. err = json.Unmarshal([]byte(txReturn), &tempToDecode)
41. if err != nil {
42.     fmt.Println(err)
43. }
44.
45. if tempToDecode["status"] != "0x0" {
46.     c.JSON(http.StatusForbidden, gin.H{
47.         "result": "forbidden",
48.     })
49.     return
50. }
51.
52. toChainDecode := make(map[string]interface{})
53. toChainDecode["input"] = tempToDecode["input"]
54. toChainDecode["output"] = tempToDecode["output"]
55. toChainDecode["abiList"] = tempABI
56. toChainDecode["decodeType"] = 2
57. toChainDecode["returnType"] = 2
58. toChainDecodeString, _ := json.Marshal(toChainDecode)
59.
60. decodeData := post("http://localhost:5002/WeBASE-
    Front/tool/decode", toChainDecodeString, "application/json")
61.
62. tempResult := make(map[string]interface{})
63. err = json.Unmarshal([]byte(decodeData), &tempResult)
64. if err != nil {
65.     fmt.Println(err)
66. }

```

```
67. c.JSON(http.StatusOK, gin.H{
68.   "result": tempResult["result"],
69. })
70.}
```