



中山大學
SUN YAT-SEN UNIVERSITY

计算机学院实验报告

课程名称： 区块链原理与技术

任课老师： 黄华威

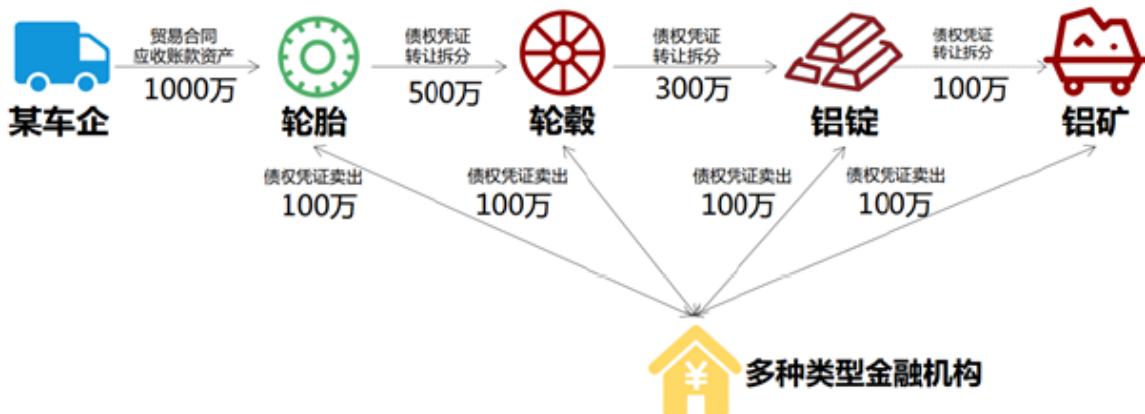
团队成员： 陈家豪 18308013

李骁达 18340095

刘业畅 18340120

1 项目设计说明

1.1 项目背景与需求



传统供应链：

某车企（宝马）因为其造车技术特别牛，消费者口碑好，所以其在同行业中占据绝对优势地位。因此，在金融机构（银行）对该车企的信用评级将很高，认为他有很大的风险承担的能力。在某次交易中，该车企从轮胎公司购买了一批轮胎，但由于资金暂时短缺向轮胎公司签订了1000万的应收账款单据，承诺1年后归还轮胎公司1000万。这个过程可以拉上金融机构例如银行来对这笔交易作见证，确认这笔交易的真实性。在接下来的几个月里，轮胎公司因为资金短缺需要融资，这个时候它可以凭借跟某车企签订的应收账款单据向金融结构借款，金融机构认可该车企（核心企业）的还款能力，因此愿意借款给轮胎公司。但是，这样的信任关系并不会往下游传递。在某个交易中，轮胎公司从轮毂公司购买了一批轮毂，但由于租金暂时短缺向轮胎公司签订了500万的应收账款单据，承诺1年后归还轮胎公司500万。当轮毂公司想利用这个应收账款单据向金融机构借款融资的时候，金融机构因为不认可轮胎公司的还款能力，需要对轮胎公司进行详细的信用分析以评估其还款能力同时验证应收账款单据的真实性，才能决定是否借款给轮毂公司。这个过程将增加很多经济成本，而这个问题主要是由于该车企的信用无法在整个供应链中传递以及交易信息不透明化所导致的。

区块链+供应链金融：

将供应链上的每一笔交易和应收账款单据上链，同时引入第三方可信机构来确认这些信息的交易，例如银行，物流公司等，确保交易和单据的真实性。同时，支持应收账款的转让，融资，清算等，让核心企业的信用可以传递到供应链的下游企业，减小中小企业的融资难度。

1.2 合约设计与功能介绍

1.2.1 合约基础

本次我们基于Table.sol完成智能合约的设计。在合约里面，我们使用两个表管理数据：`company` 表记录企业信息，`receipt` 表记录账款和交易数据。

`company` 表字段如下：

- index: 索引主键，无意义 (string类型)
- name: 企业名字 (string类型)
- address: 企业账户地址 (address类型)
- status: 运营状态: 0为正常运营, 1为破产 (int类型)

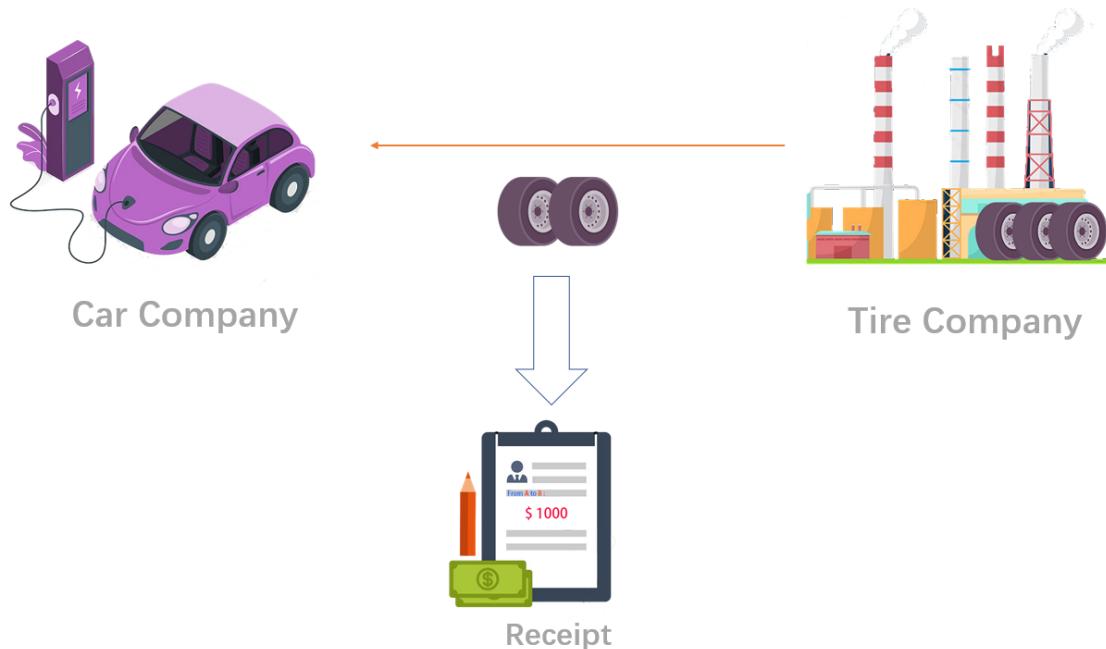
- level: 企业等级: 1为金融机构, 2为普通企业 (int类型)
- credit: 信用值 (int类型)

`receipt` 表字段如下:

- index: 索引主键, 无意义 (string类型)
- id: 账单/款项编号 (int类型)
- from: 债务人/应付方 (string类型)
- to: 债权人/应得方 (string类型)
- cur_bill: 剩余未付金额 (int类型)
- orig_bill: 总金额 (int类型)
- status: 当前状态: 0为待审批, 1为生效, 2为失效 (int类型)
- due_date: 应还日期 (string类型)

1.2.2 基本功能介绍与相关函数

- 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单据。 (基础功能)



需求背景: 在实现交易的过程中, 买方在资金短缺的情况下, 可在金融机构的见证下签订应收账款单据, 来延缓资金支付。在完成交易后, 交易数据会形成账单并完成上链。

实现逻辑: 企业A和企业B之间, 进行商品采购等交易, 交易完成后, 会基于交易完成双方创建账单。在采购商品签订应收款项前, 平台需要对双方企业资质进行审查判断, 确保双方企业正常运营, 交易款项不能超过信用值。所有核验通过后, 系统会基于当前该笔交易创建相应账单并完成上链操作, 提供给金融机构进一步审核。

链端代码:

```

function purchase(string _from, string _to, string _bill, string _dd,
address sender_address) public returns(int)
{ // 采购商品签订应收款项
    if(is_registered(_from) == -1 || is_registered(_to) == -1)// 双方必须
存在且正常运营
        return -4;
    if (verify_account(_from, sender_address)!=1)// from发起purchase
    return -3;
    int count = insert_receipt(_from, _to, _bill, 0,_dd); // 提交账单
    if(count == 1)
        return 0;
    else
        return -1;
}

```

后端代码:

```

def send_addReceipt(request):
    mess={'isSuss': 0, 'info': None}
    global mclient
    try:
        ToBusiness = request.POST.get('name')
        bill = request.POST.get('sum')
        date = request.POST.get('time')
    except:
        mess['isSuss']=-2
        mess['info']="获取参数失败！"
        return JsonResponse(mess)
    mess['isSuss'] = mclient.purchase(ToBusiness, bill, date)
    if mess['isSuss']==-4:
        mess['info']="债务人(债权人)已破产/不存在！"
    elif mess['isSuss']==-3:
        mess['info']="需由债务人提交账单！"
    elif mess['isSuss']==-2:
        mess['info']="债务人信用不足！"
    elif mess['isSuss']==-1:
        mess['info']="账单提交失败！"
    return JsonResponse(mess)

```

```

def purchase(self, To, bill, date):
    """
    发起交易，签订单据
    """
    try:
        self.renew_account_data()
        if int(bill)>self.account_credit:
            return -2
        args=[ self.account_name,
               To,
               str(self.homo_PublicKey.encrypt(int(bill)).ciphertext()),
               date,
               to_checksum_address(self.account_address)]
        res = self.client.call(self.to_address, self.contract_abi,
"purchase", args)
        if res[0]==0:

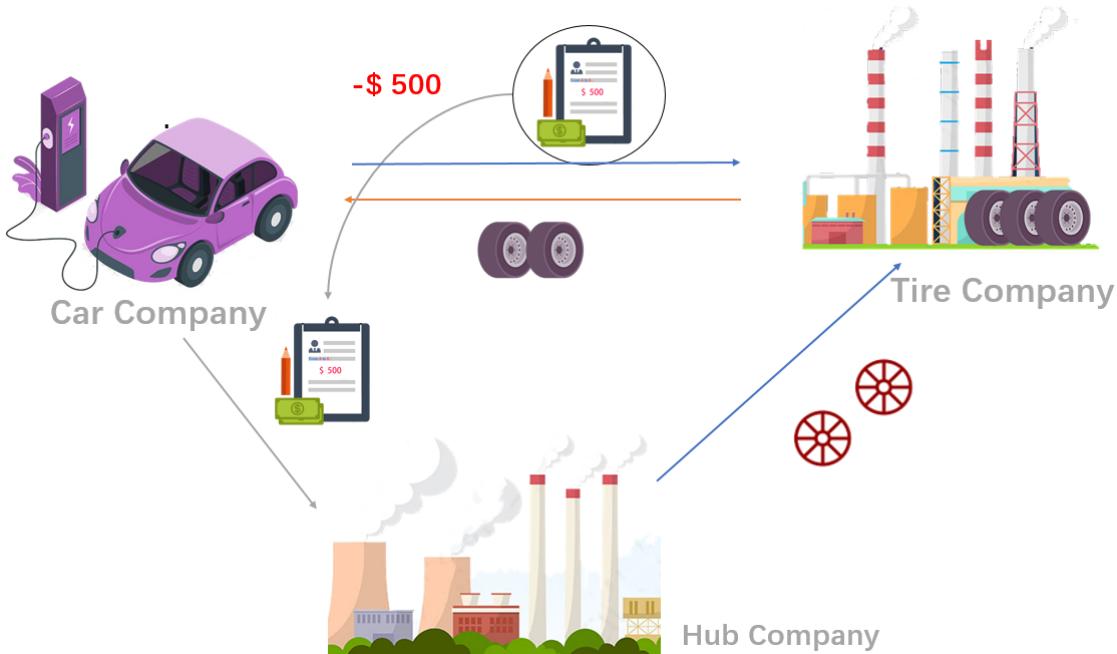
```

```

        receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"purchase", args)
        print(receipt)
        return res[0]
    except BcosException as e:
        print("execute demo_transaction failed ,BcosException for:
{}".format(e))
        traceback.print_exc()
        return -1
    except BcosError as e:
        print("execute demo_transaction failed ,BcosError for:
{}".format(e))
        traceback.print_exc()
        return -1
    except Exception as e:
        traceback.print_exc()
        return -1
    return -1

```

- 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。
(基础功能)



需求背景：企业间进行交易的时候，由于资金周转等问题，企业可以通过转让应收账款账单的方式来完成支付操作。

实现逻辑：假设企业A和企业B之间存在账单ID为01的交易，对应账款1000元。如果企业B与企业C进行交易，那么企业B可以选择直接利用持有的企业A债权对企业C进行支付。在使用企业A债权，完成债权转移时，平台会查询相关企业的企业表信息，基于对企业A和企业B的status运营状态，credit信用值进行核查，避免出现信贷危机和交易风险等情况，审查完毕后，会对01号账单的相关数据进行修改，并新建新的交易账单。

链端代码：

```

function transfer(int _id, string third_party, string update_bill, string
new_bill, int update_status, address sender_address) public returns(int)

```

```

// 转让账单
Table table = openReceiptTable();
Condition condition = table.newCondition();
condition.EQ("id", _id);
condition.EQ("status", 1); // 生效的才能转让
Entries entries = table.select("rec", condition);
if (entries.size()<=0)
    return -5;
Entry entry = entries.get(0);
if(is_registered(entry.getString("from")) == -1 ||
is_registered(entry.getString("to")) == -1 || is_registered(third_party) ==
-1)
    return -4;
if (verify_account(entry.getString("to"), sender_address)!=1)// 应得者才能
转让
    return -3;
update_receipt(_id, entry.getString("from"), entry.getString("to"),
update_bill, entry.getString("orig_bill"), update_status,
entry.getString("due_date"));
insert_receipt(entry.getString("from"), third_party, new_bill, 1,
entry.getString("due_date"));
//renew_credit(entry.getString("to"), third_party, _bill);
return 0;
}

```

后端代码:

```

def send_transfer(request):
mess={'isSuss': 0, 'info': None}
try:
    Id = request.POST.get('id')
    To = request.POST.get('to')
    Sum = request.POST.get('sum')
except:
    mess['isSuss']=-2
    mess['info']="获取参数失败！"
    return JsonResponse(mess)
global mclient
mess['isSuss']=mclient.transfer(Id, To, Sum)
return JsonResponse(mess)

```

```

def transfer(self, Id, To, Sum):
    """
    转让单据
    """
    try:
        # 获取交易内容
        Sum=int(Sum)
        args=[int(Id)]
        res = self.client.call(self.to_address, self.contract_abi,
"select_Receipt_byID", args)
        OriTo = res[1][0]
        cur =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[2][0])))
    
```

```

        update_status=1
        left=cur-Sum
        if left<0:
            return -2
        elif left==0:
            update_status=2
        args = [int(Id), To,
                str(self.homo_PublicKey.encrypt(left).ciphertext()),
                str(self.homo_PublicKey.encrypt(Sum).ciphertext()),
                update_status,
                to_checksum_address(self.account_address)]
        res = self.client.call(self.to_address, self.contract_abi,
"transfer", args)
        if res[0]==0:
            receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"transfer", args)
            print(receipt)
            # 修改双方信用
            args=[OriTo]
            res = self.client.call(self.to_address, self.contract_abi,
"select_company_byName", args)
            OriToCredit =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[3][0])))
            args=[To]
            res = self.client.call(self.to_address, self.contract_abi,
"select_company_byName", args)
            ToCredit =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[3][0])))
            args=[
                OriTo,
                To,
                str(self.homo_PublicKey.encrypt(OriToCredit-
Sum).ciphertext()),
                str(self.homo_PublicKey.encrypt(ToCredit+Sum).ciphertext())
            ]
            res = self.client.call(self.to_address, self.contract_abi,
"renew_credit", args)
            if res[0]==0:
                receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"renew_credit", args)
                print(receipt)
            else:
                return -1
            return 0
        else:
            return -1
    except BcosException as e:
        print("execute demo_transaction failed ,BcosException for:
{}".format(e))
        traceback.print_exc()
        return -1
    except BcosError as e:

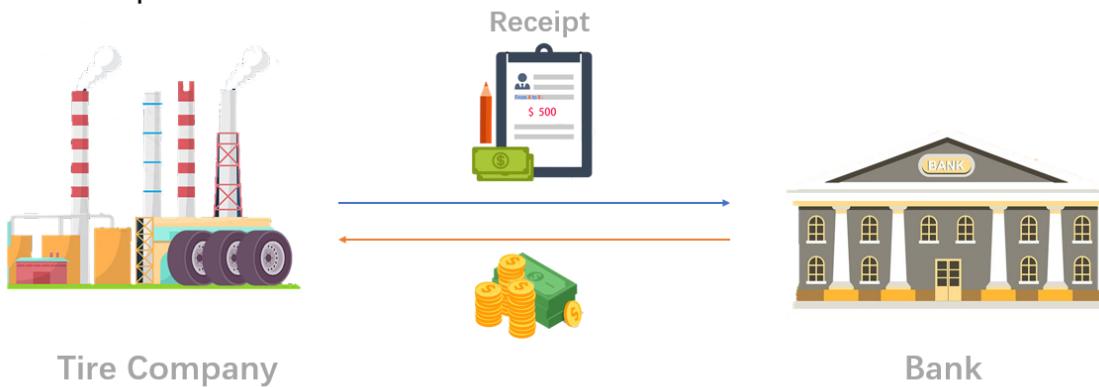
```

```

        print("execute demo_transaction failed ,BcosError for:
        {}".format(e))
        traceback.print_exc()
        return -1
    except Exception as e:
        traceback.print_exc()
        return -1
    return -1

```

- 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。(基础功能)



需求背景：企业运营过程中，可以利用当前应收账款单据向银行发起融资申请。

实现逻辑：企业可以利用目前手上持有的债权(应收账款单据)向银行发起融资请求，银行在收到请求后通过 receipt 表核实账单情况，并通过 company 表对账单上对应的相关企业进行审查。如果发现相关企业存在经营问题则会导致融资失败；如果审查通过，银行则会向企业发放贷款。在融资完成后，平台会更新企业状态信息对应的 credit 信用值。

链端代码：从本质上而言，企业向银行融资的过程相当于企业与银行之间生成一笔应还款项。因此我们通过简单调用 purchase 函数实现融资：

```

function finance(string _from, string _to, string _bill, string _dd, address
sender_address) public returns(int)
{// 使用信用融资
    return purchase(_from, _to, _bill, _dd, sender_address);
}

```

后端代码：

```

def send_finance(request):
    mess={'isSuss': 0, 'info': None}
    try:
        Id = request.POST.get('id')
        To = request.POST.get('to')
        Sum = request.POST.get('sum')
    except:
        mess['isSuss']=-2
        mess['info']="获取参数失败！"
        return JsonResponse(mess)
    global mclient
    mess['isSuss']=mclient.finance(Id, To, Sum)
    return JsonResponse(mess)

```

```

def finance(self, Id, To, Sum):
    """
    融资
    """
    return self.transfer(Id, To, Sum)

```

而金融机构的审查功能在下一章节的附加功能部分介绍。

- 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。（基础功能）



需求背景：企业账期内按时结算账款，结算完成后更新相关企业债务信息。

实现逻辑：假设企业A和企业B之间存在账单ID为01的交易，对应账款1000元。如果企业在账期内如期按时还款，在进行还款前，平台会对账单状态进行查询，确保账单存在且生效，审查完成后，债权方可以进行全额还款和部分还款操作，还款完成后，对还款企业状态信息对应的 credit 信用值进行更新调整。

链端代码：

```

function repay(int _id, string _bill, int new_status, address
sender_address) public returns(int)
{// 偿还款项
Table table = openReceiptTable();
Condition condition = table.newCondition();
condition.EQ("id", _id);
condition.EQ("status", 1);// 账单必须已经生效
}

```

```

Entries entries = table.select("rec", condition);
if (entries.size()<=0)
return -5;
Entry entry = entries.get(0);

if (verify_account(entry.getString("from"), sender_address)!=1)// 自己欠自己还
return -3;

update_receipt(entry.getInt("id"), entry.getString("from"),
entry.getString("to"), _bill, entry.getString("orig_bill"), new_status,
entry.getString("due_date"));
return 0;
}

```

后端代码:

```

def send_repay(request):
mess={'isSuss': 0, 'info': None}
try:
Id = request.POST.get('id')
Sum = request.POST.get('sum')
except:
mess['isSuss']=-2
mess['info']="获取参数失败！"
return JsonResponse(mess)
global mclient
mess['isSuss']=mclient.repay(Id, Sum)
return JsonResponse(mess)

```

```

def repay(self, Id, Sum):
"""
偿还
"""
try:
# 获取交易内容
args=[int(Id)]
res = self.client.call(self.to_address, self.contract_abi,
"select_Receipt_byID", args)
cur =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[2][0])))

if cur<=int(Sum):
new_status=2
else:
new_status=1
args = [ int(Id),
str(self.homo_PublicKey.encrypt(cur-
int(Sum)).ciphertext())),
new_status,
to_checksum_address(self.account_address) ]
res = self.client.call(self.to_address, self.contract_abi,
"repay", args)
if res[0]==0:

```

```

        receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"repay", args)
        print(receipt)

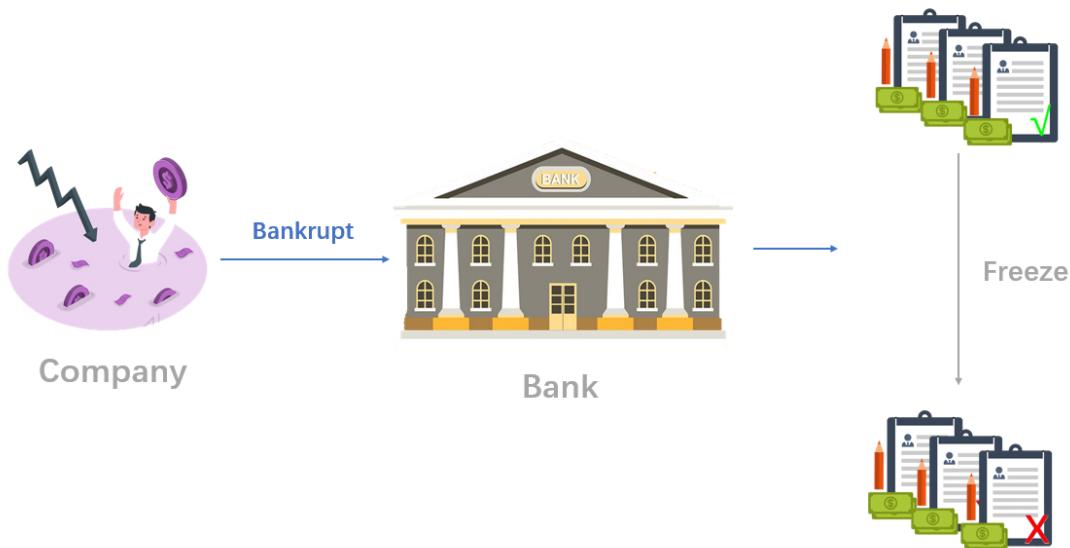
        args = [self.account_name]
        res = self.client.call(self.to_address, self.contract_abi,
"select_company_byName", args)
        credit =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[3][0])))
        self.update_company(self.account_name, res[0][0], res[1][0],
res[2][0], credit+int(sum))
        return 0
    else:
        return -1
except BcosException as e:
    print("execute demo_transaction failed ,BcosException for:
{}".format(e))
    traceback.print_exc()
    return -1
except BcosError as e:
    print("execute demo_transaction failed ,BcosError for:
{}".format(e))
    traceback.print_exc()
    return -1
except Exception as e:
    traceback.print_exc()
    return -1
return -1

```

2 加分项

2.1 实现了额外的功能

- 附加功能一：企业破产申请



需求背景：企业在运营过程中可能存在经营不善，为了减小企业经营不善带来的损失，企业可以向银行提出破产申请。

实现逻辑：企业A因经营不善，造成企业进入无法扭转的债务危机，企业可以向银行提出破产申请，银行根据企业表对企业实际运行状况进行审查，审查通过后，对企业A的 status账单状态变更为破产状态。此后，与该企业相关的账款不能继续流通或进一步操作，避免与企业A相关的债务资产在链上持续流通造成更大的经济损失。此外，金融机构也可以宣布某家企业破产。

链端代码：

```
function set_bankrupt(string _name) public returns(int)
{// 破产
    Table table = openCompanyTable();
    Condition condition1 = table.newCondition();
    condition1.EQ("name", _name);
    condition1.EQ("status", 0);
    Entries entries = table.select("com", condition1);
    if (entries.size()<=0)// 未找到相关企业
        return -1;
    if (verify_account(_name, msg.sender)!=1 && isBank(msg.sender)!=1)// 只能自己或金融机构宣布自己破产
        return -3;
    Entry old_entry = entries.get(0);
    Entry new_entry = table.newEntry();
    new_entry.set("status", int(1));//破产
    new_entry.set("name", _name);
    new_entry.set("address", old_entry.getAddress("address"));
    new_entry.set("level", old_entry.getInt("level"));
    new_entry.set("credit", old_entry.getInt("credit"));
    Condition condition2 = table.newCondition();
    condition2.EQ("name", _name);
    int count = table.update("com", new_entry, condition2);// 更新company表
    emit UpdateResult(count);
    return count;
}
```

后端代码：

```
def broke(request):
    mess={'issuss': 0, 'info': None}
    global mclient
    mess['issuss']=mclient.broke()
    return JsonResponse(mess)
```

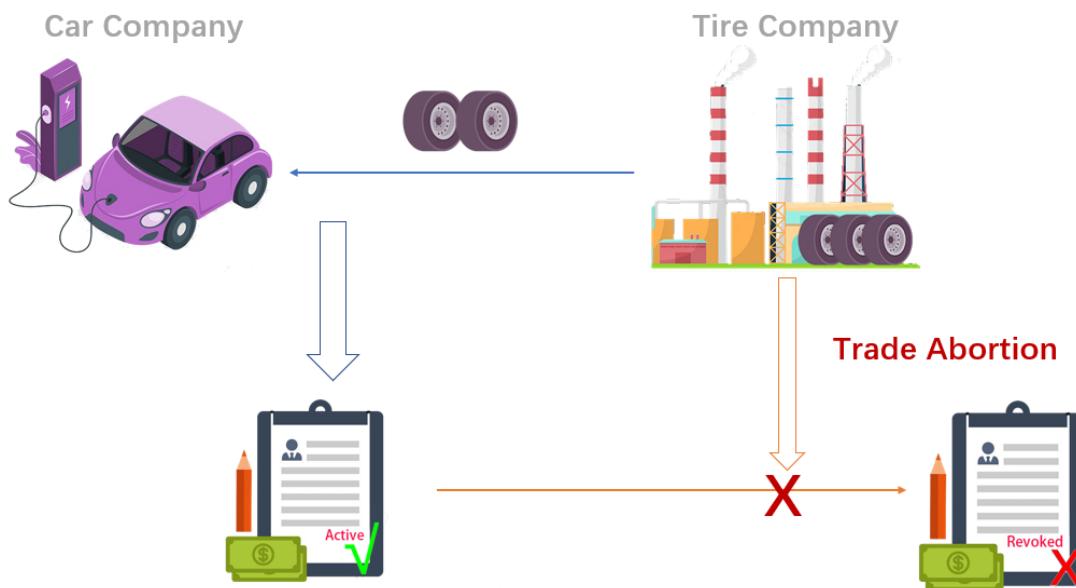
```
def broke(self):
    try:
        args=[self.account_name,
to_checksum_address(self.account_address)]
        res = self.client.call(self.to_address, self.contract_abi,
"set_bankrupt", args)
        print(res)
        if res[0]>=0:
            receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"set_bankrupt", args)
            print(receipt)
            return 0
        else:
            return -1
```

```

        except BcosException as e:
            print("execute demo_transaction failed ,BcosException for:
{}".format(e))
            traceback.print_exc()
            return -1
        except BcosError as e:
            print("execute demo_transaction failed ,BcosError for:
{}".format(e))
            traceback.print_exc()
            return -1
        except Exception as e:
            traceback.print_exc()
            return -1
    return -1

```

- 附加功能二：撤销交易



需求背景: 交易账单(ID=1)创建完成后，交易双方根据自己实际需求可以对交易进行撤销操作。

实现逻辑: 假设企业A和企业B之间存在账单ID为01的交易，对应账款1000元。如果企业双方由于某些原因，需要对交易进行撤销操作，平台接到请求后，需要对交易账单 `status` 账单状态进行审查，如果账单已经失效，那么无法撤销。确保账单状态正常后，需要对交易双方企业状态信息对应的 `status` 信用值进行更新调整，并更新 `receipt` 表的有关账单信息，从而完成整个撤销交易操作。

链端代码:

```

function revoke_receipt(int _id, address sender_address) public returns
(int)
{//撤销交易
Table table = openReceiptTable();
Condition condition = table.newCondition();
condition.EQ("id", _id);
}

```

```

        condition.EQ("status", 1); // 已经失效的不能撤回
        Entries entries = table.select("rec", condition);
        if (entries.size()<=0)
            return -5;
        Entry entry = entries.get(0);
        if (verify_account(entry.getString("from"), sender_address)!=1 &&
        verify_account(entry.getString("to"), sender_address)!=1)
            return -3; // 交易双方才能撤回
        // 原交易失效
        update_receipt(_id, entry.getString("from"), entry.getString("to"),
        entry.getString("cur_bill"), entry.getString("orig_bill"), 2,
        entry.getString("due_date"));
        return 0;
    }
}

```

后端代码:

```

def send_revoke(request):
    mess={'isSuss': 0, 'info': None}
    try:
        Id = request.POST.get('id')
    except:
        mess['isSuss']=-2
        mess['info']="获取参数失败！"
        return JsonResponse(mess)
    global mclient
    mess['isSuss']=mclient.revoke(Id)
    return JsonResponse(mess)
}

```

```

def revoke(self, Id):
    """
    撤销交易
    """
    try:
        args=[int(Id), to_checksum_address(self.account_address)]
        res = self.client.call(self.to_address, self.contract_abi,
        "revoke_receipt", args)
        if res[0]==0:
            receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
        "revoke_receipt", args)
            print(receipt)
            # 获取交易内容
            args=[int(Id)]
            res = self.client.call(self.to_address, self.contract_abi,
        "select_Receipt_byID", args)
            cur =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
        int(res[2][0])))
            ori =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
        int(res[3][0])))
            left = ori-cur
            From = res[0][0]
            To = res[1][0]
            due = res[5][0]
    
```

```

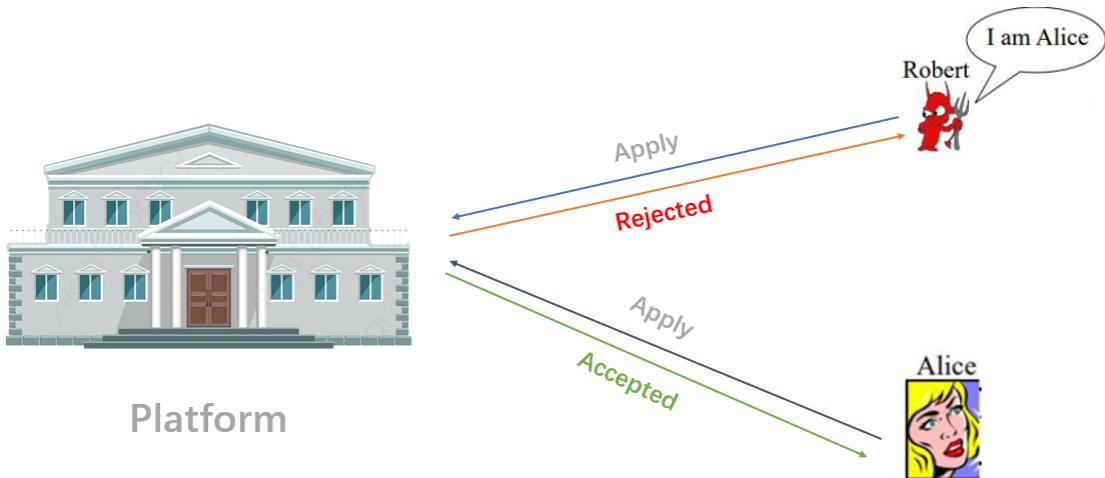
        # 恢复信用
        args=[From]
        res = self.client.call(self.to_address, self.contract_abi,
"select_company_byName", args)
        FromCredit =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[3][0])))
        args=[To]
        res = self.client.call(self.to_address, self.contract_abi,
"select_company_byName", args)
        ToCredit =
self.homo_PrivateKey.decrypt(paillier.EncryptedNumber(self.homo_PublicKey,
int(res[3][0])))
        args=[ To, From,
str(self.homo_PublicKey.encrypt(ToCredit-
left).ciphertext()),

str(self.homo_PublicKey.encrypt(FromCredit+left).ciphertext()) ]
        receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"renew_credit", args)
        print(receipt)
        if left>0:
            args = [ To, From,

str(self.homo_PublicKey.encrypt(left).ciphertext()),
1, due ]
        receipt =
self.client.sendRawTransactionGetReceipt(self.to_address, self.contract_abi,
"insert_receipt", args)
        print(receipt)
        return 0
    else:
        return -1
except BcosException as e:
    print("execute demo_transaction failed ,BcosException for:
{}".format(e))
    traceback.print_exc()
    return -1
except BcosError as e:
    print("execute demo_transaction failed ,BcosError for:
{}".format(e))
    traceback.print_exc()
    return -1
except Exception as e:
    traceback.print_exc()
    return -1
return -1

```

- 附加功能三：身份验证功能



需求背景：为了避免恶意攻击和非法操作，平台需要在企业操作之前进行身份验证。

实现逻辑：假设Alice和Bob都尝试以企业A的身份信息向平台发起操作申请，平台方会对Alice和Bob的身份进行核验，平台方基于数据库存储的企业A的 address 账户地址与Alice和Bob提供的信息进行比对，从而完成身份验证操作。

链端代码：在我们的合约设计中，几乎所有涉及对数据表进行改动的函数都设计了身份验证。例如应收款项交易上链的函数 `purchase` 只能由债务人调用，转让账款函数 `transfer` 函数只能由原账款的债权人调用。我们设计了一个函数用于完成比较：

```
function verify_account(string name, address saddress) private constant
returns(int)
{// 验证登陆账户
Table table = openCompanyTable();
Condition condition = table.newCondition();
condition.EQ("index", "com");
condition.EQ("name", name);
Entries entries = table.select("com", condition);
if (entries.size()<=0)// 不存在该企业
return -1;
Entry entry=entries.get(0);
if (entry.getAddress("address")==saddress)// 账户地址相同
return 1;
return -1;
}
```

在实际使用时，我们只需要用企业名和`msg.sender`调用该函数，通过返回值判断身份是否正确。

此外，在金融机构审批账款、宣布破产时，我们还需要判断调用方是否为金融机构，判断函数如下所示：

```
function isBank(address Myaddress) private constant returns(int)
{// 验证是否是金融机构
Table table = openCompanyTable();
Condition condition = table.newCondition();
condition.EQ("level", 1);
condition.EQ("status", 0);
Entries entries = table.select("com", condition);
```

```

int flag=-1;
for (int i=0;i<entries.size();i++)
{
Entry entry=entries.get(i);
if (entry.getAddress("address")==Myaddress)
{
flag=1;
break;
}
}
return flag;
}

```

后端代码：实际应用中，让用户登录时每次都输入合约地址未免太过繁琐；因此我们采用了更为常见的“账户-密码”的登录方法，允许用户自定义便于记忆的密码；如果输入的密码正确，则视为是合法用户在操作。

```

def loginAccount(request):
mess={'isSuss': 0, 'info': None}
try:
    name = request.POST.get('user')
    password = request.POST.get('pass')
except:
    mess['isSuss']=-2
    return JsonResponse(mess)
global mclient
mclient = myClient(name, password)
if mclient is None:
    mess['isSuss']=-1
elif mclient.isOK==False:
    mess['isSuss']=-1
return JsonResponse(mess)

```

- **附加功能四：部分偿还**

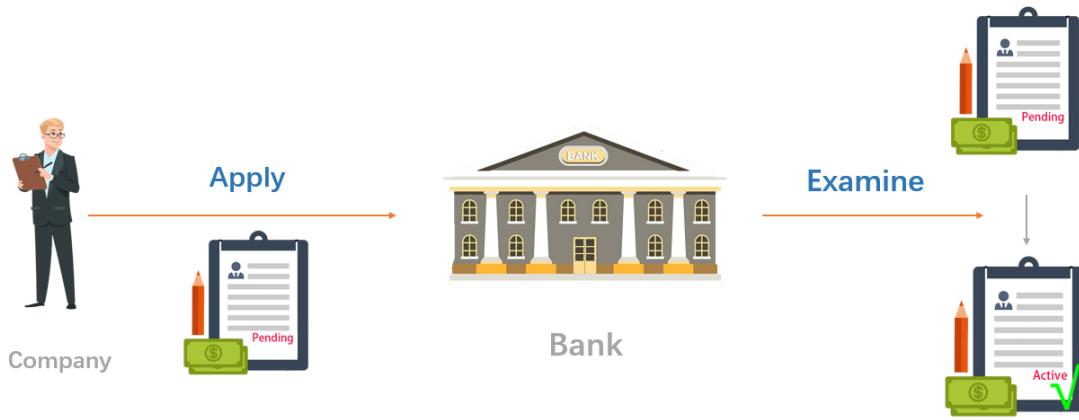


需求背景：企业在偿还过程中，可以根据自身资金状况对账单进行部分偿还。

实现逻辑：企业在进行还款前，平台会对账单状态进行查询，确保账单存在且生效，审查完成后，平台对还款方的还款额与实际欠款额进行对比，确保还款额小于等于欠款额，才能进行还款。还款完成后，还款企业状态信息对应的 credit 信用值进行更新调整。

代码实现：我们将这部分功能集成进偿还函数 `repay` 中。企业可以自定义还款金额，具体代码见前面介绍过的**功能四**。

- 附加功能五：信任机构实时审批



需求背景：企业产生的账款单据都需要经过平台的检查和金融机构的审批，经审核确认后才能生效。

实现逻辑：所有交易产生的应受账款单据初始状态为待审批（`status` 标记为0），必须经过金融机构验证了交易的真实性之后，单据才能正式生效（`status` 标记为1），可以执行转让、偿还等操作。

代码实现：平台的审核操作已经集成在各个函数内，而金融机构的审批操作在函数 `verify_trade` 中实现。本次合约我们实现的是一个简单审批，即金融机构调用该函数后就会通过所有待审批的账单。

```
function verify_trade() public returns(int)
{ // 审批交易数据
    if (isBank(msg.sender)==-1)// 只有金融机构才能审批
        return -3;
    Table table=openReceiptTable();
    Condition condition = table.newCondition();
    condition.EQ("status", 0);
    Entries entries = table.select("rec", condition);
    int count=0;
    for (int i=0;i<entries.size();i++)
    {
        Entry entry=entries.get(i);
        // 全部通过
        update_receipt(entry.getInt("id"), entry.getString("from"),
        entry.getString("to"), entry.getInt("cur_bill"), entry.getInt("orig_bill"),
        1, entry.getString("due_date"));
        renew_credit(entry.getString("from"), entry.getString("to"),
        entry.getInt("cur_bill"));
        count++;
    }
    return count;
}
```

后端代码：由于具体的审批逻辑与同态加密算法密切相关，因此运算部分（即更新交易双方的信用值）的代码在下一小节介绍同态加密的部分再一同展示。

```
def send_AgreeOrNot(request):
    mess={'isSuss': 0, 'info': None}
    try:
```

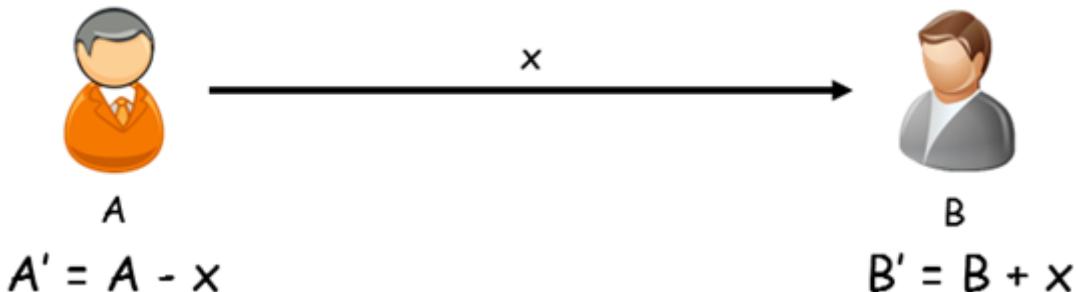
```

id = request.POST.get('id')
choice = request.POST.get('choice')
except:
    mess['isSuss']=-2
    mess['info']="获取参数失败！"
    return JsonResponse(mess)
global mclient
mess['isSuss']=mclient.verify(id, choice)
if mess['isSuss']==-1:
    mess['info']="交互失败！"
elif mess['isSuss']==-2:
    mess['info']="更新信用失败！"
return JsonResponse(mess)

```

2.2 通过同态加密实现链上数据隐私保护

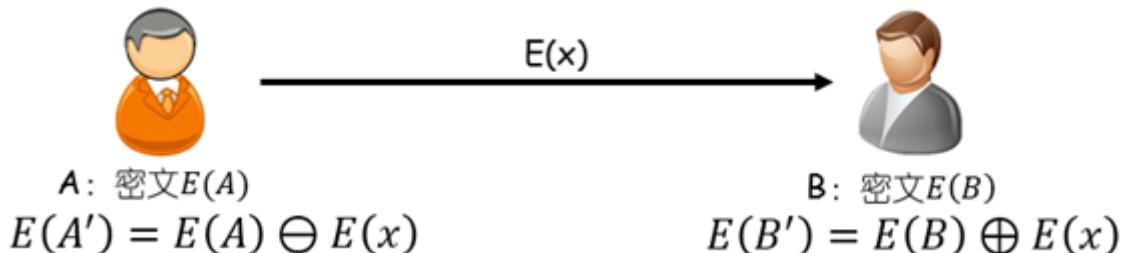
在传统区块链中，数据以明文方式存储在链上，区块链上的验证节点需要利用这些数据来验证交易的合法性。例如下图所示的一个常见的转账场景：



上图中，用户之间的转账金额 x 以明文的形式存储，使得交易的某些关键隐私是完全暴露的。而在实际应用中，用户并不想暴露某些隐私信息。这就需要一种方法将隐私明文映射密文，同时保证对密文的运算与对明文运算后再加密的结果是相同的，即能够保持同态性。同态加密就能够满足这种需求，根据算法所支持的同态运算，同态加密的能力各有不同，这里我们只关注加法同态，记加密函数为 E ，则加法同态指的是：

$$E(a) + E(b) = E(a + b) \quad (1)$$

上链前将关键信息使用 E 进行加密，即可避免隐私的暴露，又通过同态性来支持公开监管，在上述转账场景中引入同态加密后，交易过程如下图所示：



在实现中，我们采用的是 paillier 加密算法，该算法满足加法同态。对链上数据使用 paillier 算法加密后，相关的隐私信息都以同态密文的形式存在，除了拥有私钥的第三方，所有节点都无法获得明文，从而有效地保护了用户的隐私。我们通过调用 python-paillier 软件包（github 地址为：<https://github.com/data61/python-paillier>）实现同态加密。在具体的实现层面，我们在初始化文件 `INIT.py` 中调用软件包的 `generate_paillier_keypair` 函数产生一对长度为 64 位的公私钥，并将他们写入文件中保存起来；后续需要进行加密解密时，再到文件中取出公私钥进行操作，密钥产生以及写入代码如下：

```

# 获取同态加密公私钥对
public_key, private_key = paillier.generate_paillier_keypair(n_length=64)
if (not os.path.isfile("PaillierPublicKey.pkl")) or (not
os.path.isfile("PaillierPrivateKey.pkl")):
    with open("PaillierPublicKey.pkl", 'wb') as f:
        str = pickle.dumps(public_key)
        f.write(str)
    with open("PaillierPrivateKey.pkl", 'wb') as f:
        str = pickle.dumps(private_key)
        f.write(str)

```

为了实现隐私保护，交易金额上链前必须完成加密，这就要求智能合约的数据库中的 `bill` 值为密文。以 `purchase` 函数为例，智能合约中该函数接收的参数 `_bill` 即为交易金额，代表信用的转移。那么在后端调用链端的 `purchase` 函数前，需要对明文的交易金额 `bill` 需要调用公钥类的 `encrypt` 函数进行加密；也正因为链端无法访问到金额的真值，判断交易金额是否超出了买房的剩余信用的操作也必须在后端完成。这部分的后端代码如下（再次附上合约中的 `purchase` 函数头以便参照）：

```

// _from 向 _to 购买价值为 _bill 的商品，还款的最后期限为 _dd，sender_address 用于验证身份
function purchase(string _from, string _to, string _bill, string _dd, address
sender_address) public returns(int)

```

```

# 判断交易金额是否超过买房信用
if int(bill)>self.account_credit:
    return -2

# 调用链端 purchase 函数需要的参数，其中金额需要进行同态加密
args=[ self.account_name,
       To,
       str(self.homo_PublicKey.encrypt(int(bill)).ciphertext()),
       date,
       to_checksum_address(self.account_address)]

# 调用合约中的 purchase 函数
res = self.client.call(self.to_address, self.contract_abi, "purchase", args)

```

由于密文长度过长，我们使用了 `string` 类型来进行金额的存储。金额加密后，合约中任何涉及金额的运算都是在密文上进行的，由于加密算法保持了同态性，我们对运算结果解密的值其实就是真正的金额。因此用户在使用过程中，平台只需要从通过智能合约读取加密的金额，再调用软件包的解密模块形成明文，即可向用户展示其私人的明文信息了，此过程与加密是反向的。需要特别说明的是，原始的 `paillier` 加密算法只针对正整数加法，为了将加密功能扩展到浮点数和负数，软件包借助了编码机制对明文进行了适当的处理，因此加解密过程也会伴随着编码和解码。而编码和解码的引入，会导致密文的运算法则稍微发生变化：

$$D(E(a) \times E(b)) = a + b \quad (2)$$

其中 E 代表先编码再加密， D 代表先解密再解码。因此，明文加法对应于合约中密文的乘法，而减法通过将减数取负即可转化为加法，即：

$$D(E(a) \times E(-b)) = a - b \quad (3)$$

实际上，软件包也保持了乘法同态性，由于合约不涉及乘法，此处不再赘述。真正涉及到密文运算的部分即为企业信用的更新。在实现中，当银行审批账单认为其有效后，会根据交易金额将买房的信用抵押给卖方。在后端，需要借助合约读取出账单对应的交易双方当前的信用密文，再进行同态运算得到更新后的信用，将新信用作为参数调用智能合约中的 `renew_credit` 函数对信用进行更新。整个过程中，智能合约部分的代码均未接触到明文，从后端 `verify` 函数中截取关键代码如下：

```
if int(choice)==1:  
    args=[int(Id)]  
    # 根据 ID 获取从数据库中获取账单信息  
    res = self.client.call(self.to_address, self.contract_abi,  
    "select_Receipt_byID", args)  
    ecredit = int(res[2][0]) # 交易金额密文  
    From = res[0][0] # 买方身份  
    To = res[1][0] # 卖方身份  
  
    args=[From] # 从数据库中获取买家信息  
    res = self.client.call(self.to_address, self.contract_abi,  
    "select_company_byName", args)  
    FromECredit = int(res[3][0]) # 买家信用密文  
  
    args=[To] # 从数据库中获取卖家信息  
    res = self.client.call(self.to_address, self.contract_abi,  
    "select_company_byName", args)  
    ToECredit = int(res[3][0]) # 卖家密文  
    args=[  
        From,  
        To,  
        str(FromECredit * (- ecredit)), # 买家减去信用  
        str(ToECredit * ecredit) # 卖家增加信用  
    ]  
    receipt = self.client.sendRawTransactionGetReceipt(self.to_address,  
    self.contract_abi, "renew_credit", args)
```

2.3 实现了友好高效的用户界面

我们设计了较为美观的网页，并且面向用户在细节上进行了许多考虑：

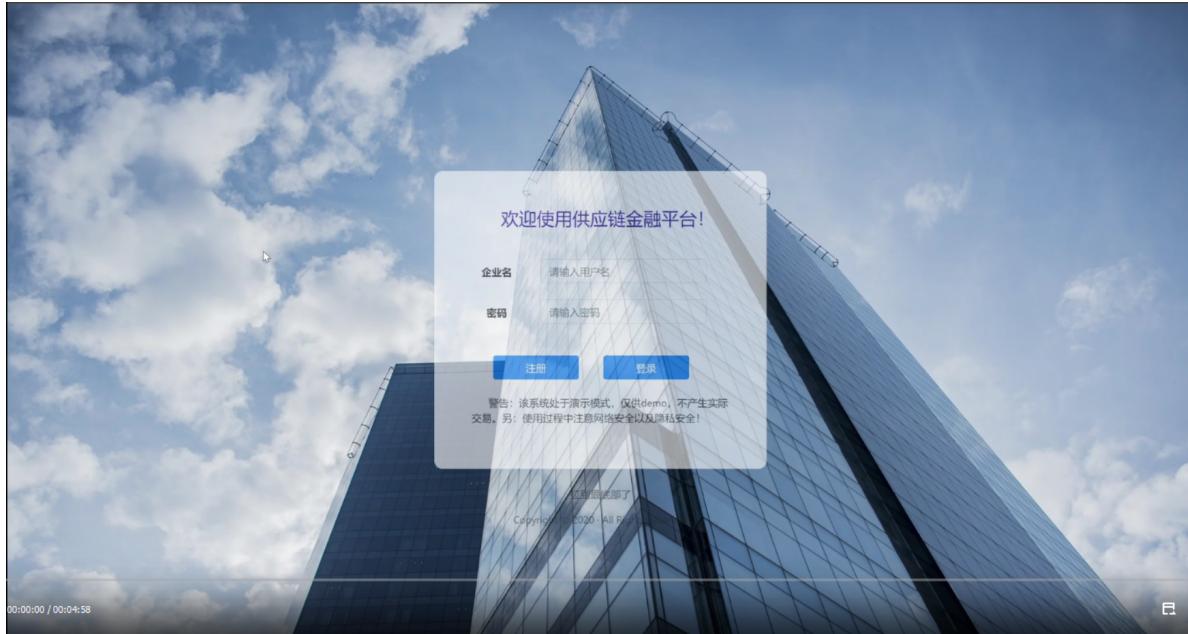
1. 用户分普通企业或金融机构两种身份注册并登陆；
2. 排序展示用户个人历史账单，忽略与用户无关的账单，界面清晰高效；
3. 良好的纠错机制：对用户输入进行判断，对无效输入或者超额交易进行提示，比如对输入的金额进行上下限的提示语显示；

总体来说，最终的供应链金融平台平台在使用体验上能够基本接近当前已实用化的各种平台。具体效果在下一章节通过截图展示。

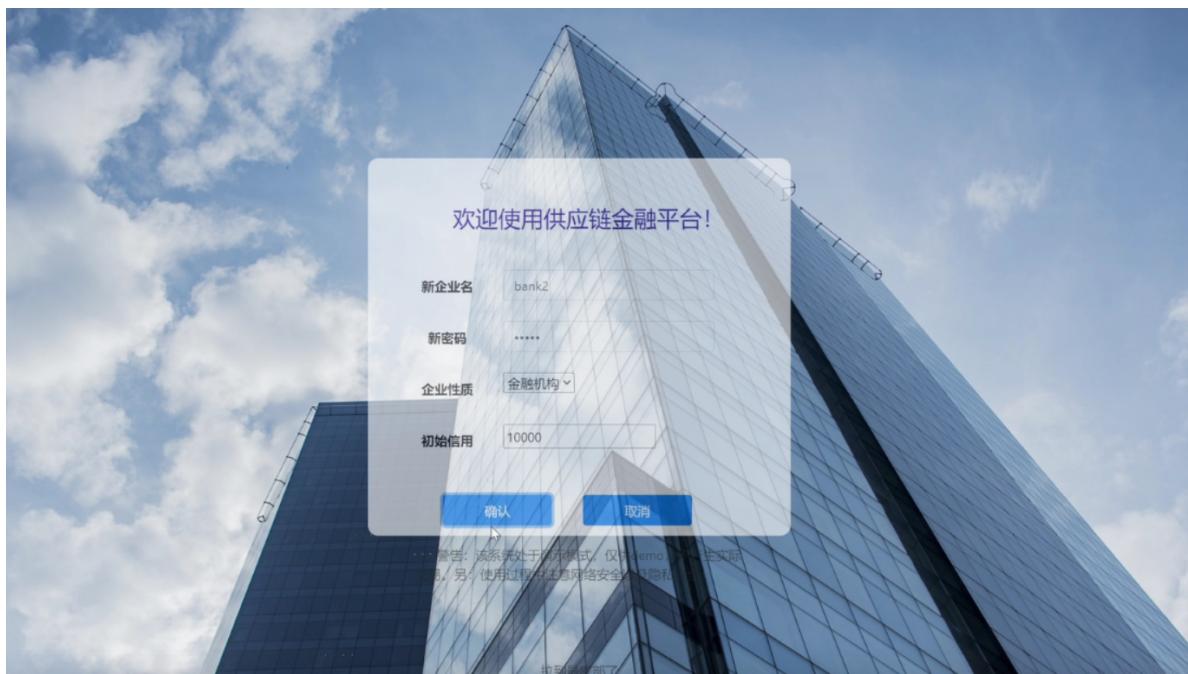
3 应用展示截图

3.1 企业创建与登录

下面展示注册一家金融机构 `bank4`，网址登陆后的初始界面如下：



点击注册，输入账户密码，以及初始信用，点击确认完成登陆如下：



另外，我们也注册了其他几个用户。回到首页登陆 `car_company` 后进入企业主界面，可以看到有 `账单管理` 和 `企业管理` 两个功能模块，如下图所示：



3.2 企业交易并添加账单

车企登陆进入 car_company 后, 假设车企分别向轮胎公司 tire_company 购买了两批货物, 一批金额为 1000, 另一批为 2000; 再想轮毂公司 hub_company 购买金额为 500 的货物, 并不立即完成支付。则车企创建三条账单, 供金融机构审核。车企点击 账单管理 进入账单管理系统, 并选择 应付账单 子栏目, 如下图所示:

A screenshot of the "Bill Management System" interface. The title "账单管理系统" is at the top. Below it, there are two tabs: "应收账款信息" (Receivable Bill Information) and "应付账单信息" (Payable Bill Information), with "应付账单信息" being active. The main area has sections for "基本信息" (Basic Information) with a search bar and three buttons: "搜债务人" (Search Debtor), "搜债权人" (Search Creditor), and "搜账单ID" (Search Bill ID). Below that is a table for "账单管理" (Bill Management) with columns: 账单id (Bill ID), 应还时间 (Due Date), 债务人 (Debtor), 债权人 (Creditor), 剩余未付金额 (Remaining Unpaid Amount), 总金额 (Total Amount), 当前状态 (Current Status), and 操作 (Operation). A "添加" (Add) button is highlighted with a cursor. At the bottom, a message says "拉到最底部了" (Pull to the bottom) and "Copyright © 2020 · All Rights Reserved".

点击 添加, 并按提示输入账单信息, 如下图所示:

A screenshot of the "Bill Management System" interface, similar to the previous one but with different input fields. The title "账单管理系统" is at the top. Below it, there are two tabs: "应收账款信息" (Receivable Bill Information) and "应付账单信息" (Payable Bill Information), with "应付账单信息" being active. The main area has sections for "基本信息" (Basic Information) with a search bar and three buttons: "搜债务人" (Search Debtor), "搜债权人" (Search Creditor), and "搜账单ID" (Search Bill ID). Below that is a table for "账单管理" (Bill Management) with columns: 账单id (Bill ID), 应还时间 (Due Date), 债务人 (Debtor), 债权人 (Creditor), 剩余未付金额 (Remaining Unpaid Amount), 总金额 (Total Amount), 当前状态 (Current Status), and 操作 (Operation). In this screenshot, the "应还时间" (Due Date) field contains "2021/01/29", the "债权人" (Creditor) dropdown is set to "tire_company", and the "剩余未付金额" (Remaining Unpaid Amount) field contains "1000". A "确认" (Confirm) button is highlighted with a cursor. At the bottom, a message says "拉到最底部了" (Pull to the bottom) and "Copyright © 2020 · All Rights Reserved".

点击 **确认** 完成添加，同理，添加剩余的两条账单，最终涉及车企的所有应付账单信息如下：

The screenshot shows the 'Accounts Receivable Management System' interface. At the top, there are tabs for '首页' (Home), '应收账单信息' (Information about receivable bills), and '应付账单信息' (Information about payable bills). The '应收账单信息' tab is selected. Below the tabs is a search bar with placeholder text '请输入关键字' (Please enter keyword) and three buttons: '搜债务人' (Search debtor), '搜债权人' (Search creditor), and '搜账单ID' (Search bill ID). A sidebar on the left shows navigation links: '基本信息' (Basic information), '账单管理' (Bill management), and '退出' (Logout). The main content area displays a table of bills:

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
0	2021-01-29	car_company	tire_company	1000	1000	待审批	
1	2021-01-30	car_company	tire_company	2000	2000	待审批	
2	2021-01-31	car_company	hub_company	500	500	待审批	

A green '添加' (Add) button is located at the bottom right of the table. At the very bottom of the page, it says '拉到最底部了' (Pull to the bottom) and 'Copyright © 2020 · All Rights Reserved'.

此时这些账单的当前状态为 **未审批**，需要等待金融机构审核有效后才能正式生效。

3.3 金融机构实时审批账单

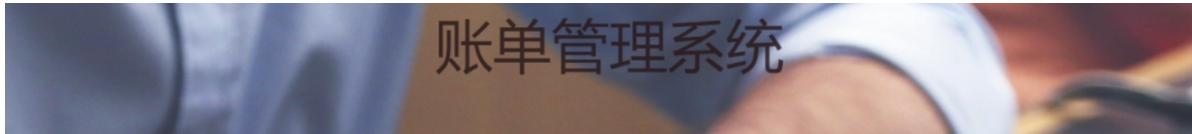
登录 **bank1**，点击 **企业管理** 进入企业管理系统，这里金融机构可以看到企业的名称、账户地址、运营状态（正常或者破产）以及企业信用等等，同时，金融机构有权点击 **修改** 来对某个企业的信息进行修改，如下图所示：

The screenshot shows the 'Enterprise Management System' interface. At the top, there are tabs for '首页' (Home) and '企业状态管理' (Enterprise status management). The '企业状态管理' tab is selected. Below the tabs is a search bar with placeholder text '请输入关键字' (Please enter keyword) and three buttons: '搜企业名称' (Search company name) and '搜账户地址' (Search account address). A sidebar on the left shows navigation links: '基本信息' (Basic information), '账单管理' (Bill management), and '企业管理' (Enterprise management). The main content area displays a table of enterprises:

企业名称	账户地址	企业性质	运营状态	企业信用	操作
car_company	0xc344430fd5e08d5ba9efde218f81fbcbc14b8dd8	普通企业	正常运营	5000	修改
tire_company	0x7234c70600853758c9ef992c711e2b28ab5c92ec	普通企业	正常运营	2000	修改
hub_company	0x3ff7ca3bcda4319299079e9d06912a9bf9d2fbdf	普通企业	正常运营	1000	修改
bank	0x59afa90a1d034ed55d487d6be7fa375cd99c589d	金融机构	正常运营	10000	修改
bank2	0xa9223bb37fc70ed32ab2946629d5e82a1594679f	金融机构	正常运营	10000	修改

企业点击左栏 **账单管理**，即可对上面车企提交的三条账单进行审批，最终认定第二条账单 (id=1) 无效，点击 **否决**，其余有效，分别点击 **通过**，如下图所示：

账单管理系统



基本信息	账单信息							
账单管理	账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
企业管理	0	2021-01-29	car_company	tire_company	1000	1000	待审批	<button>通过</button> <button>否决</button>
退出	1	2021-01-30	car_company	tire_company	2000	2000	待审批	<button>通过</button> <button>否决</button>
	2	2021-01-31	car_company	hub_company	500	500	待审批	<button>通过</button> <button>否决</button>

审核，账单状态变更如下：

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
0	2021-01-29	car_company	tire_company	1000	1000	生效中	
1	2021-01-30	car_company	tire_company	2000	2000	未通过	
2	2021-01-31	car_company	hub_company	500	500	生效中	

3.4 欠款转让与融资

登录 `tire_company`，进入账单管理系统，并查看 `应收账单信息` 子栏目，可以看到目前轮胎公司所持有的款项，如下图所示：

基本信息	应收账单信息							
账单管理	账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
退出	0	2021-01-29	car_company	tire_company	1000	1000	生效中	<button>转让</button> <button>单据融资</button> <button>撤销</button>
	1	2021-01-30	car_company	tire_company	2000	2000	未通过	

可以看到前面被 `bank1` 审核通过的账单状态为 `生效中`，被否决的账单状态为 `未通过`。对合法账单，可以有转让、单据融资、撤销等操作。`tire_company` 将账单 0 拆分出 500 用于向 `hub_company` 支付一批货物，并拆分出 300 向 `bank2` 融资，分别点击 `转让` 和 `单据融资` 完成，转让操作示例如下：

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
0	2021-01-29	car_company	tire_company	1000	1000	生效中	
	2021-01-29	car_company	hub_company	500	1000		<button>确认</button>
1	2021-01-30	car_company	tire_company	2000	2000	未通过	

融资操作示例如下：

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
0	2021-01-29	car_company	tire_company	500	1000	生效中	
	2021-01-29	car_company	bank2	融资金额: 300			确认
1	2021-01-30	car_company	tire_company	2000	2000	未通过	

此时点击查看企业信息，可以看到企业信用（初始为 2000）此时为 2200，多出的 200 即为账单 0 剩余的未付金额。

3.5 偿还账单

`car_company` 决定偿还应还款项。由上述过程可知，`car_company` 在账单 0 中应还 500 给 `tire_company`，同时应还 300 给 `bank2`。使用 `car_company` 的账户登录，查看应付账单信息如下：

首页	应收账单信息	应付账单信息						
基本信息	<input type="text" value="请输入关键字"/> <input type="button" value="搜债务人"/> <input type="button" value="搜债权人"/> <input type="button" value="搜账单ID"/>							
账单管理	账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
退出	0	2021-01-29	car_company	tire_company	200	1000	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
	1	2021-01-30	car_company	tire_company	2000	2000	未通过	
	2	2021-01-31	car_company	hub_company	500	500	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
	3	2021-01-29	car_company	hub_company	500	500	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
	4	2021-01-29	car_company	bank2	300	300	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
								<input type="button" value="添加"/>

点击账单 2 的 `还款` 操作，可以输入还款金额，可以全部偿还，也可以部分偿还，这里是全部偿还，界面如下：

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
0	2021-01-29	car_company	tire_company	200	1000	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
1	2021-01-30	car_company	tire_company	2000	2000	未通过	
2	2021-01-31	car_company	hub_company	500	500	生效中	
				偿还金额: <input type="text" value="500"/>			<input type="button" value="确认"/>
3	2021-01-29	car_company	hub_company	500	500	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
4	2021-01-29	car_company	bank2	300	300	生效中	<input type="button" value="还款"/> <input type="button" value="撤销"/>
							<input type="button" value="添加"/>

偿还完偿还两个账单后，更新的应付账单信息如下：

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
0	2021-01-29	car_company	tire_company	200	1000	生效中	<button>还款</button> <button>撤销</button>
1	2021-01-30	car_company	tire_company	2000	2000	未通过	
2	2021-01-31	car_company	hub_company	300	500	生效中	<button>还款</button> <button>撤销</button>
3	2021-01-29	car_company	hub_company	500	500	生效中	<button>还款</button> <button>撤销</button>
4	2021-01-29	car_company	bank2	0	300	已失效	

[添加](#)

3.8 撤销交易与破产

企业可以撤销与自己有关的当前正在生效的账款。假设 hub_company 想要撤销账单 3，那么我们需要使用 hub_company 的账户登录控制台，在对应编号的账款后面点击 [撤销](#) 操作。

首页	应收账单信息	应付账单信息						
基本信息	<input type="text" value="请输入关键字"/> <button>搜债务人</button> <button>搜债权人</button> <button>搜账单ID</button>							
账单管理	账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
退出	2	2021-01-31	car_company	hub_company	300	500	生效中	<button>转让</button> <button>单据融资</button> <button>撤销</button>
	3	2021-01-29	car_company	hub_company	500	500	生效中	<button>转让</button> <button>单据融资</button> <button>撤销</button>

撤销后的账单信息更新如下：

账单id	应还时间	债务人	债权人	剩余未付金额	总金额	当前状态	操作
2	2021-01-31	car_company	hub_company	300	500	生效中	<button>转让</button> <button>单据融资</button> <button>撤销</button>
3	2021-01-29	car_company	hub_company	500	500	已失效	

再进入基本信息栏，点击 [一键破产](#) 则可以宣布破产，该操作只能由金融机构或者宣布自己破产的企业调用，如下图所示：

首页	企业基本信息
基本信息	企业名: hub_company 企业性质: 普通企业 企业账户地址: 0x3ff7ca3bcda4319299079e9d06912a9bf9d2fbdf 企业状态: 正常运营 一键破产 
账单管理	企业信用: 2000
退出	拉到最底部了 <small>Copyright © 2020 · All Rights Reserved</small>

此后，还需要金融机构对上述操作进行审核，前面已有示例；更全面的效果展示请移步演示视频。

4 小组分工

姓名	学号	分工
李骁达	18340095	编写智能合约完成基本功能，实现同态加密，编写实验报告
陈家豪	18308013	构思平台底层逻辑，完成后端，对平台结构调试优化，完成平台应用化
刘业畅	18340120	设计网页，完成前端工作，实现合约附加功能，编写实验报告