



本科实验报告

课程名称： 区块链与数字货币

实验名称： Book Shop DAPP 系统

姓 名： 应承峻（3170103456）

张雪楠（3170105632）

韩汶东（3170100798）

学 院： 计算机科学与技术学院

专 业： 软件工程

2019/12/22

目录

一、引言.....	4
1.1 设计目的	4
1.2 设计说明	5
二、总体设计	5
2.1 功能模块	5
2.2 系统架构	6
三、详细设计	7
3.1 开发环境	7
3.1.1 Ganache.....	7
3.1.2 Truffle	7
3.1.3 VS Code	8
3.2 智能合约	8
3.2.1 合约中变量	8
3.2.2 书籍购买.....	9
3.2.3 添加书籍.....	10
3.2.4 申请退款与确认退款.....	10
3.2.5 更新书籍信息.....	11
3.3 系统实现	11
3.3.1 部署合约并获得合约地址.....	11
3.3.2 连接以太坊	12
3.3.3 项目启动与初始化.....	12

3.3.4 重新启动项目并在浏览器中运行	13
四、智能合约测试.....	17
4.1 测试用例及初始化	17
4.2 Truffle 框架测试	17
4.3 整体测试设计	18
五、Dapp 测试与运行.....	19
5.1 登录	19
5.2 用户购买书籍	19
5.3 用户申请退款	20
4.4 卖家管理书籍	21
5.5 卖家管理订单	23
六、总结与思考	24
6.1 问题与解决.....	24
6.1.1 Truffle 框架安装.....	24
6.1.2 智能合约中的两类函数	24
6.2 反思与总结.....	27

一、引言

随着居民消费能力持续提升与网上购物习惯逐步养成，中国网络零售市场交易规模保持持续增长。网上店铺零售商的规模随着网上消费者规模的增大持续扩张，除了大型电商平台之外，小型电商平台、小规模电商集群、个体电商网站等的数量也持续增加。然而，这部分市场虽然繁荣，但是鱼龙混杂，随着市场规模的发展，也出现了很多问题，其中对于非大型电商平台而言，遇到最大的问题就是如何取得客户的信任，尤其是网络交易环节。

网络交易环节设计财产账户、财产转移、履行合约等很多部分，所以这部分功能往往比较敏感，大型电商平台有企业做背书，而非大型电商平台想取得消费者的信任往往很难。这是网络电商行业多元化发展所面临的难题。

这个问题产生的原因从根本上看是中心化问题。私人电商店铺的交易系统是中心化的，依靠专有和封闭的软件，这样就会产生很多局限：

- ✧ 交易过程不透明：消费者不能确定财产转移去向，整个过程的真实性、合法性也无法保证；
- ✧ 合约履行无法保证：对于完成交易之后商家是否履行合约无法保证，消费者无法验证整个交易过程已成功完成；
- ✧ 交易生态系统局限：因为没有统一的标准，每个组织者开发自己的用户界面和工具。但这种分散交易系统的质量参差不齐，安全性无法得到保证。

1.1 设计目的

构建一个基于以太坊的去中心化的网上店铺交易系统（Book Shop），解决网上购物交易过程中买卖双方的不信任问题，保障买卖双方的利益。

在此次课程中，我将用所学到的知识，开发一个去中心化的 Book Shop DAPP。一方面加深对于区块链的理解，另一方面也是在实际的应用场景中进行学习，并且该 DAPP 在以后可以进行继续的迭代开发，不断完善并投入使用。

1.2 设计说明

在此次开发过程中，我们小组将使用 truffle 框架、express 框架和 ganache 软件，来进行 Dapp 的开发。

时间轴及工作分配如下：

时间	主要工作	参与人员
12.10~12.14	Truffle 框架的熟悉, 下载 运行了部分 demo	应承峻、张雪楠、韩汶东
12.15~12.19	拍卖系统智能合约编写	应承峻（主负责）、张雪楠、韩汶东
12.20~12.24	拍卖系统 web3.js 调用 智能合约代码编写	应承峻、张雪楠、韩汶东
12.25~12.28	页面前端后端编写	应承峻、张雪楠、韩汶东
12.29~12.30	撰写报告	应承峻、张雪楠、韩汶东

小组成员的贡献占比如下：

姓名	工作量比例
应承峻	40%
张雪楠	30%
韩汶东	30%

二、总体设计

2.1 功能模块

该 DAPP 主要实现了以下功能：

- ✧ 合约范式可升级；
- ✧ 首页展示商品书籍；

- ✧ 消费者可选择商品进行购买；
- ✧ 消费者查看订单；
- ✧ 消费者申请退款；
- ✧ 消费者删除订单；
- ✧ 卖家添加展示商品；
- ✧ 卖家查看订单；
- ✧ 卖家删除订单；
- ✧ 交易过程自动更新交易双方账户余额。

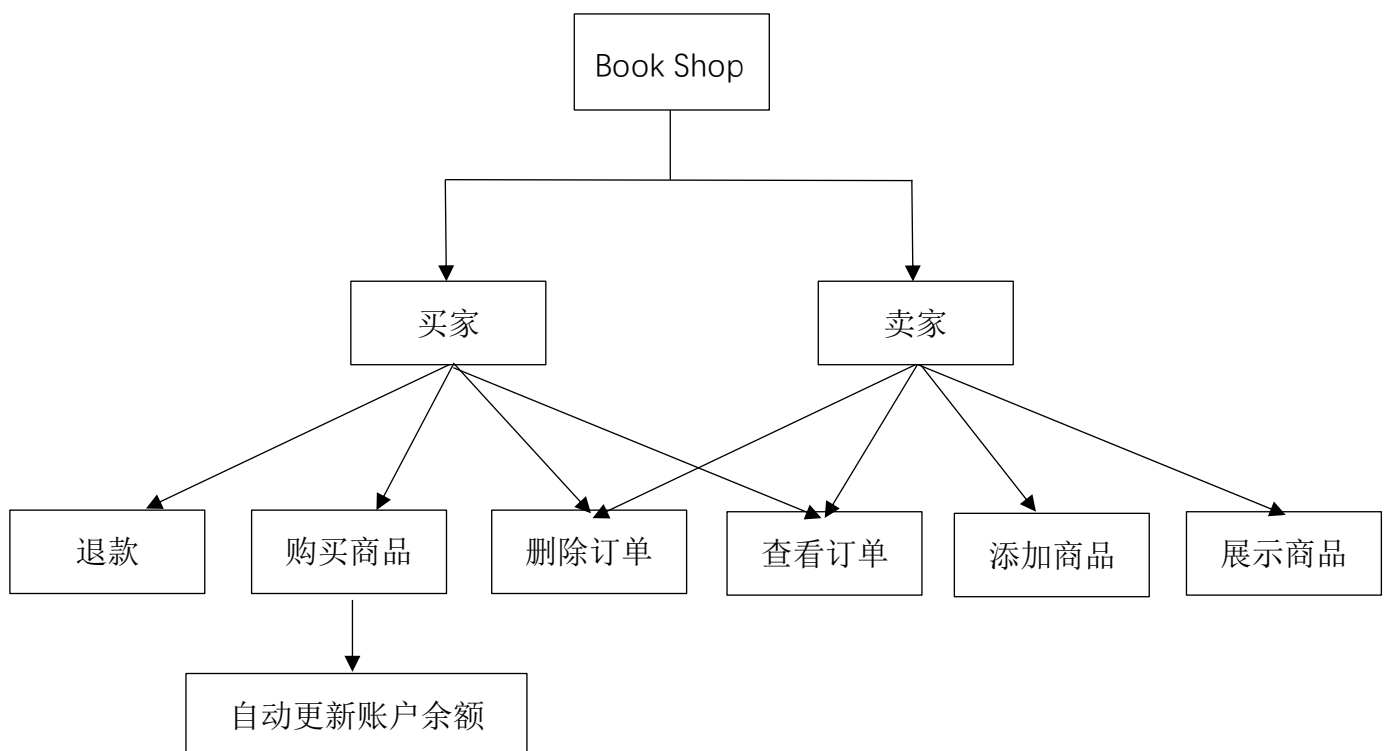


图 1 DAPP 总体功能

2.2 系统架构



三、详细设计

3.1 开发环境

图 2 系统架构图

3.1.1 Ganache

Ganache 是为开发者提供的一个私有 Ethereum 区块链客户端，它可以快速启动个人以太坊区块链，并可以使用它来运行测试、执行命令、检查状态，同时控制链条的运行方式。

使用 Ganache 发布的私有链将会自动生成 10 个账户，每个账户有 100 以太币的余额，同时主界面也将记录交易数、账户等信息。

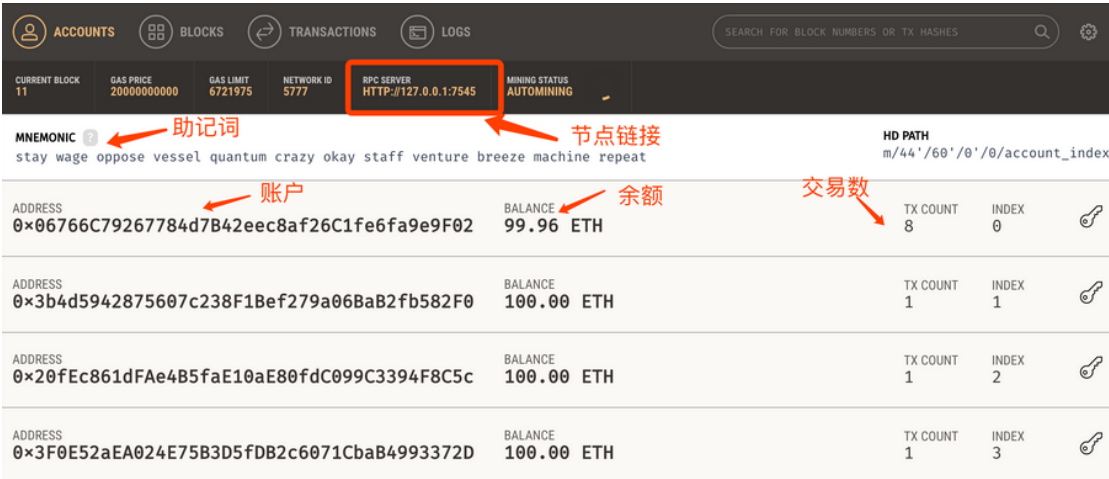


图 3 Ganache 界面

3.1.2 Truffle

truffle 是以太坊（Ethereum）最受欢迎的一个智能合约开发框架，它针对基于以太坊的 Solidity 语言。本身基于 Javascript，支持对合约代码的编译、部署及测试工作，方便开发者的测试与部署，极大推进了工程开发的进展。

此外，truffle 还提供了合约的抽象接口，使用基于 web3.js 的 Ether Pudding 工具包，可在拿到合约对象后，在 Javascript 中直接调用合约中的函数方法，便于对接与实际的开发测试。

windows 10 安装：npm install -g truffle

3.1.3 VS Code

采用 VS Code 集成开发环境，在其中安装最新的 Solidity 拓展后，用于合约及 Dapp 的开发、编译、部署。明晰的代码视图及便捷的拓展安装，将极大缩短项目的开发生命周期，提高开发效率。

3.1.4 Proxy 范式

Proxy 范式采用 Proxy Content 存储所有的状态变量及逻辑合约地址，采用 Logic Contract 来定义具体的逻辑规则。采用 proxy 合约，对外提供的地址是 proxy 合约。真正的合约地址以变量的形式放在 proxy 合约里面。要升级合约时，只需重新部署真正的合约，然后把新的地址更新到 proxy 合约里即可。

3.2 智能合约

3.2.1 合约中变量

```
uint moneyEarned = 0; //售卖书籍的利润
address owner; //用以标记书店主

struct books {
    uint bookID;
    string name;
    string author;
    uint cost;
    uint count;
    string image;
    bool isExist; //懒惰位，用于标志该书籍是否已经被删除
}
```



```

struct purchase {
    uint purchaseID;
    uint bookID;
    string date;    //购买时间
    address from;  //购买者地址
    uint status; //标记订单状态, 0 为订单已支付, 1 为申请退款中, 2 为退款成功
}

struct user {
    string name;
    uint balance;
}

mapping(uint => books) public booksMap;    //书籍映射记录
uint bookCount = 0;    //书籍数量

mapping(uint => purchase) public purchaseMap;    //购买记录映射记录
uint purchaseCount = 0;    //书籍购买列表

mapping(address => user) public userMap;    //用户列表

```

3.2.2 书籍购买

先对需要购买的书籍余量进行判断, 若库存不足, 则提示

“Books are all sold out!”, 接着对用户的账户余额进行判断, 若用户余额小于书本价格, 则提示 “Insufficient balance!” 若上述两个条件均通过, 则将继续更新书籍库存量、用户余额及商家利润, 同时记录下本条购买记录。

```

//购买书籍
function buy(uint bookID, string memory date) public payable {

    require(booksMap[bookID].count > 0, "Books are all sold out!");
    require(userMap[msg.sender].balance >= booksMap[bookID].cost, "Insufficient balance!");

    //减少书籍量
    booksMap[bookID].count = booksMap[bookID].count - 1;
    //计算余额
    userMap[msg.sender].balance -= booksMap[bookID].cost;
    //收益
    userMap[owner].balance += booksMap[bookID].cost;
    //计算利润
    moneyEarned = moneyEarned + booksMap[bookID].cost;
}

```

```

//插入购买记录
addPurchase(bookID, date);

}

```

3.2.3 添加书籍

根据传入的书本名称、作者、数量、封面等信息进行书本的添加。

```

//添加书籍
function addBook(string memory name, string memory author, uint cost, uint count, string memory image) public {

    bookCount++;
    booksMap[bookCount].bookID = bookCount;
    booksMap[bookCount].name = name;
    booksMap[bookCount].author = author;
    booksMap[bookCount].cost = cost;
    booksMap[bookCount].count = count;
    booksMap[bookCount].image = image;
    booksMap[bookCount].isExist = true;

}

```

3.2.4 申请退款与确认退款

买家根据购买账单的编号，申请退款。商家在确认过申请买家的身份、账单id后，确认退款，并更新利润、书籍数量等信息。

```

//买家申请退款
function applyRefund(uint purchaseID) public {
    require(purchaseID > 0 && purchaseID <= purchaseCount && purchaseMap[purchaseID].from == msg.sender, "Invalid Purchase ID");
    purchaseMap[purchaseID].status = 1;
}

//商家确认退款
function agreeRefund(uint purchaseID) public {
    require(msg.sender == owner, "Invalid User!");
    require(purchaseMap[purchaseID].status == 1, "Should apply refund first!");
    require(purchaseID > 0 && purchaseID <= purchaseCount, "Invalid Purchase ID");

    //获取退款额
}

```

```

    uint bookPrice = booksMap[purchaseMap[purchaseID].bookID].cost;
    //返还退款
    userMap[owner].balance -= bookPrice;
    //接收退款
    userMap[purchaseMap[purchaseID].from].balance += bookPrice;
    //扣除利润
    moneyEarned -= bookPrice;
    //恢复书籍数量
    booksMap[purchaseMap[purchaseID].bookID].count++;
    //修改标志位
    purchaseMap[purchaseID].status = 2;
}

```

3.2.5 更新书籍信息

采用位掩码的方式，更安全简洁地对书籍名称、作者、价格、库存、图片等信息进行修改。

```

//使用位掩码标记修改的字段
function updateBook(uint mask, uint index, string memory name, string memory author
, uint cost, uint count, string memory image) public {
    if (mask & 1 > 0) booksMap[index].name = name;
    if (mask & 2 > 0) booksMap[index].author = author;
    if (mask & 4 > 0) booksMap[index].cost = cost;
    if (mask & 8 > 0) booksMap[index].count = count;
    if (mask & 16 > 0) booksMap[index].image = image;
}

```

3.3 系统实现

3.3.1 部署合约并获得合约地址

在 Ganache 中启动一条私有链后，将配置文件 truffle-config.js 中默认账户配置为 Ganache 生成账户中的第一个账户，并使用如下命令得到部署后的合约地址 truffle migrate --reset。

```

2_deploy_bookshop.js
=====

Replacing 'BookShop'
-----
> transaction hash:    0x91cc1db5b5eea4645b001dd56e353036ae2334d50bb59c1d001056d4cb0175a6
> Blocks: 0           Seconds: 0
> contract address:   0x07EaD0747F22E1A03fc97E09380f1bCdB3DB9cE4
> block number:       7
> block timestamp:     1577254379
> account:            0xcAe24EcF9a976dD9366e225890AdBB698A99E903
> balance:            99.87198138
> gas used:           2881190
> gas price:          20 gwei
> value sent:         0 ETH
> total cost:         0.0576238 ETH

> Saving migration to chain.

```

3.3.2 连接以太坊

更改 Dapp 中 models/config.js 的相关参数,使应用获得新部署的合约地址,然后利用 web3.js,按照私有链端口等参数进行配置,而后直接可以调用 web3.eth 下的方法。

```

var Web3 = require('web3');

var web3;

if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
}

else {
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
}

//console.log(web3.eth.accounts);
//var version = web3.version.node;
//console.log(version);

module.exports = web3;

```

3.3.3 项目启动与初始化

利用 router/index.js 中的函数对项目进行初始化,采用 npm start 命令启动项

目，并观察项目初始化的结果。

```
PS C:\Users\lenovo\Desktop\Dapp> npm start

> dapp@0.0.0 start C:\Users\lenovo\Desktop\Dapp
> node ./bin/www

ok
[
  {
```

```
    {
      bookID: 7,
      bookName: '操作系统概念',
      author: '[美]西尔伯查茨',
      bookPrice: 72,
      count: 666,
      image: '/images/book7.jpg',
      isExist: true,
      errMsg: null
    },
    {
      bookID: 8,
      bookName: 'Git学习指南',
      author: '普莱贝尔',
      bookPrice: 44,
      image: '/images/book8.jpg',
      isExist: true,
      errMsg: null
    }
  ]
}
```

如上图，即为项目初始化部分结果。

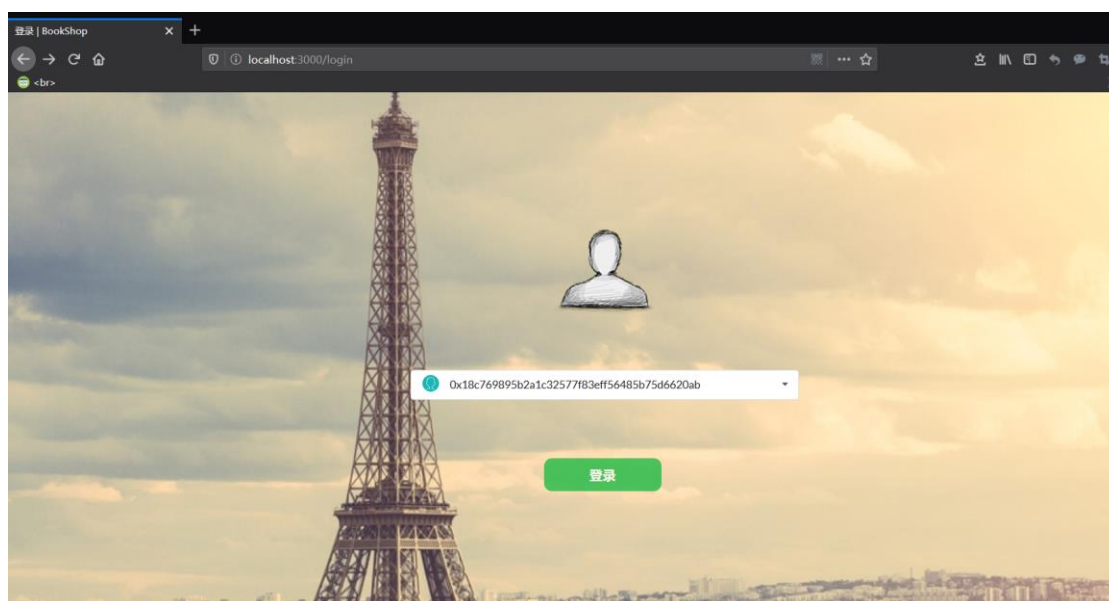
3.3.4 重新启动项目并在浏览器中运行

终止运行上一步的项目，注释掉初始化代码后重新启动项目，并在localhost:3000中运行项目，进行功能体验。

```
终止批处理操作吗(Y/N)? y
PS C:\Users\lenovo\Desktop\Dapp> npm start

> dapp@0.0.0 start C:\Users\lenovo\Desktop\Dapp
> node ./bin/www

GET / 302 10.234 ms - 56
GET /login 200 123.671 ms - 4013
GET /javascripts/jquery.js 200 5.193 ms - 248949
GET /javascripts/semantic.min.js 200 5.773 ms - 274522
GET /stylesheets/semantic.css 200 6.806 ms - 861906
GET /images/login.png 200 2.937 ms - 8453
GET /images/avatar.png 200 3.569 ms - 5593
GET /images/b7.jpg 200 1.183 ms - 349250
```



3.3.5 智能合约升级

为了解决合约智能合约的可迭代需求与其不可更改性之间的矛盾，实现智能合约可升级，我们采用了 proxy 范式来将合约逻辑与状态变量分别存储在两个账号中，当逻辑合约需要更新时，在部署完该新合约后，更新 proxy 中相应的索引，即可实现逻辑合约的升级操作。



相关代码：

Proxy.sol:

```

pragma solidity ^0.5.5;

/**
 * @title Proxy
 * @dev Gives the possibility to delegate any call to a foreign implementation.
 */
contract Proxy {

    /**
     * @dev Tells the address of the implementation where every call will be delegated.
     * @return address of the implementation to which it will be delegated
     */
    function implementation() public view returns (address);

    /**
     * @dev Fallback function allowing to perform a delegatecall to the given implementation.
     * This function will return whatever the implementation call returns
     */
    function () external payable {
        address _impl = implementation();
        require(_impl != address(0), "xxx-msg");
    }
}

```

IRegistry.sol:

```

pragma solidity ^0.5.5;

interface IRegistry {
    /**
     * @dev This event will be emitted every time a new proxy is created
     * @param proxy representing the address of the proxy created
     */
    event ProxyCreated(address proxy);

    /**
     * @dev This event will be emitted every time a new implementation is registered
     * @param version representing the version name of the registered implementation
     * @param implementation representing the address of the registered implementation
     */
    event VersionAdded(string version, address implementation);

    /**
     * @dev Registers a new version with its implementation address
     * @param version representing the version name of the new implementation to be registered
     * @param implementation representing the address of the new implementation to be registered
     */
    function addVersion(string calldata version, address implementation) external;

    /**
     * @dev Tells the address of the implementation for a given version
     * @param version to query the implementation of
     * @return address of the implementation registered for the given version
     */
    function getVersion(string calldata version) external view returns (address);
}

```

UpgradeabilityProxy.sol:

```

contract UpgradeabilityProxy1 is Proxy, UpgradeabilityStorage {

    /**
     * @dev Constructor function
     */
    function UpgradeabilityProxy(string memory _version) public {
        registry = IRegistry(msg.sender);
        upgradeTo(_version);
    }

    /**
     * @dev Upgrades the implementation to the requested version
     * @param _version representing the version name of the new implementation to be set
     */
    function upgradeTo(string memory _version) public {
        _implementation = registry.getVersion(_version);
    }
}

```

UpgradeabilityStorage.sol:

```

contract UpgradeabilityStorage {
    // Versions registry
    IRegistry internal registry;

    // Address of the current implementation
    address internal _implementation;

    /**
     * @dev Tells the address of the current implementation
     * @return address of the current implementation
     */
    function implementation() public view returns (address) {
        return _implementation;
    }
}

```

Upgradeable.sol:

```

contract Upgradeable is UpgradeabilityStorage {
    /**
     * @dev Validates the caller is the versions registry.
     * THIS FUNCTION SHOULD BE OVERRIDDEN CALLING SUPER
     * @param sender representing the address deploying the initial behavior of the contract
     */
    function initialize(address sender) public payable {
        require(msg.sender == address(registry), "xxx-msg");
    }
}

```


四、智能合约测试

4.1 测试用例及初始化

采用下面的代码为一个 TestCase，是整体测试代码一部分，对 getIsDeletebook 功能进行了测试。

```
it('TestCase 2', async () => {
    let instance = await BookShop.deployed();
    await instance.addBook("bookA", "authorA", 80, 10, "imageA");
    let book = await instance.getBook(1);

    console.log(book[0].toNumber(), book[1], book[2], book[3].toNumber(), book[4].toNumber(), book[5], book[6]);

    //掩码位 11 表示修改 name, author, count 三个字段
    await instance.updateBook(11, 1, "bookA Modified", "authorA Modified", 10, 0, "imageA Modified");
    book = await instance.getBook(1);

    console.log(book[0].toNumber(), book[1], book[2], book[3].toNumber(), book[4].toNumber(), book[5], book[6]);

    await instance.deleteBook(1); //删除书籍
    let vvv = instance.getIsDeleteBook(1);
    console.log("vvv = ", vvv.valueOf());

    book = await instance.getBook(1);
    console.log(book[0].toNumber(), book[1], book[2], book[3].toNumber(), book[4].toNumber(), book[5], book[6]);
});
```

4.2 Truffle 框架测试

在命令行，采用 truffle test 命令开始测试，并查看相应的测试结果是否符合预期设定。

```

PS C:\Users\lenovo\Desktop\Solidity> truffle test
Using network 'development'.

Compiling your contracts...
=====
> Compiling .\contracts\BookShop.sol

Contract: BookShop
Username: James Smith      balance 1000
Username: James Smith      Balance 920
Username: Seller            Balance 80
1 2019-12-19 14:15:00 bookA 80 0 0x93fc4D8Bb896A9BA7A59F6eEe7342afc101b5c8c
1 2019-12-19 14:15:00 bookA 80 2 0x93fc4D8Bb896A9BA7A59F6eEe7342afc101b5c8c

```

4.3 整体测试设计

设计了完整的案例测试，基本覆盖了合约中的大部分功能，利用 truffle 进行测试与改进，对与预期结果不符的功能进行了修改与讨论。

如下图为整体的案例测试代码：

```

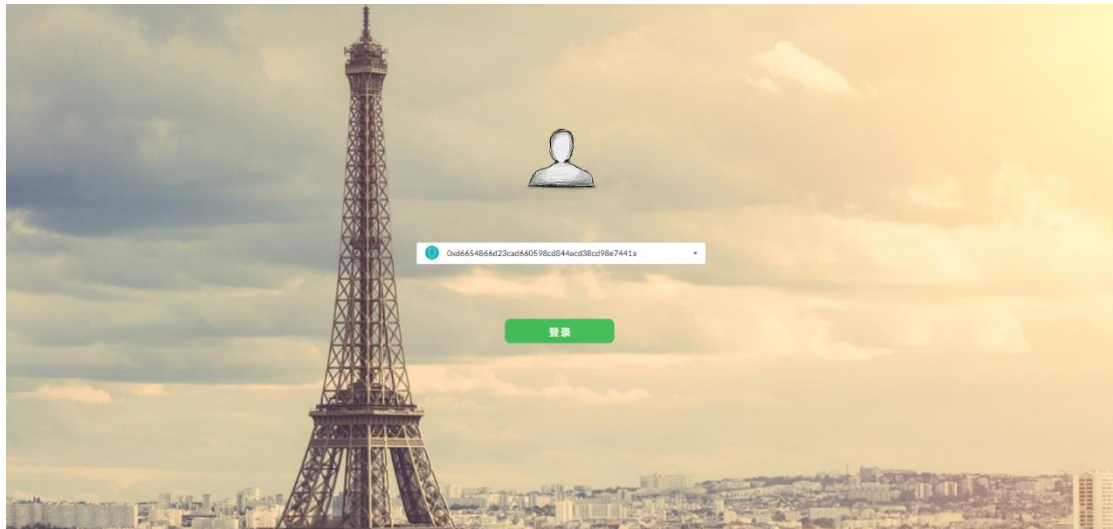
JS BookShop.js x
DApp > Solidity > test > JS BookShop.js > contract('BookShop') callback > it('Test Case 1') callback
1  const BookShop = artifacts.require('BookShop');
2
3
4  contract('BookShop', accounts => {
5    it('Test Case 1', async () => {
6      let instance = await BookShop.deployed();
7      await instance.addBook("bookA", "authorA", 80, 10, "imageA");
8      await instance.addBook("bookB", "authorB", 160, 1, "imageB");
9      await instance.addBook("bookC", "authorC", 240, 4, "imageC");
10     await instance.setUser("Seller", 0); //初始化卖家
11     await instance.setUser("James Smith", 1000, { from: '0x8Ba6C932f80aA818130BE16055d6251e0eBda5bb' }); //初始化买家
12     let buyer = await instance.getUser({ from: '0x8Ba6C932f80aA818130BE16055d6251e0eBda5bb' });
13     console.log('Username: ', buyer[0], '\tBalance ', buyer[1].toNumber()); //获取初始化后用户的数据
14     await instance.buy(1, "2019-12-19 14:15:00", { from: '0x8Ba6C932f80aA818130BE16055d6251e0eBda5bb' }); //购买书籍
15     buyer = await instance.getUser({ from: '0x8Ba6C932f80aA818130BE16055d6251e0eBda5bb' }); //显示购买后买家的数据
16     console.log('Username: ', buyer[0], '\tBalance ', buyer[1].toNumber()); //获取初始化后用户的数据
17     let seller = await instance.getUser(); //显示购买后卖家的信息
18     console.log('Username: ', seller[0], '\tBalance ', seller[1].toNumber()); //获取初始化后用户的数据
19     let moneyEarned = await instance.getMoneyEarned(); //获取利润
20     assert.equal(moneyEarned.toNumber(), 80); //验证利润结果
21     let purchaseCount = await instance.getPurchaseCount();
22     assert.equal(purchaseCount.toNumber(), 1); //验证交易记录数量
23     let purchaseData = await instance.getPurchaseData(1); //获取第一条交易记录
24     console.log(purchaseData[0].toNumber(), purchaseData[1], purchaseData[2], purchaseData[3].toNumber(), purchaseData[4].toNumber());
25     await instance.applyRefund(1, { from: '0x8Ba6C932f80aA818130BE16055d6251e0eBda5bb' }); //申请退款
26     await instance.agreeRefund(1);
27     purchaseData = await instance.getPurchaseData(1); //获取退款后第一条交易记录
28     console.log(purchaseData[0].toNumber(), purchaseData[1], purchaseData[2], purchaseData[3].toNumber(), purchaseData[4].toNumber());
29     moneyEarned = await instance.getMoneyEarned(); //获取退款后的利润
30     assert.equal(moneyEarned.toNumber(), 0); //验证扣除利润的结果
31     buyer = await instance.getUser({ from: '0x8Ba6C932f80aA818130BE16055d6251e0eBda5bb' }); //显示退款后买家的数据
32     console.log('Username: ', buyer[0], '\tBalance ', buyer[1].toNumber()); //获取退款后用户的数据
33     seller = await instance.getUser(); //显示退款后卖家的信息
34     console.log('Username: ', seller[0], '\tBalance ', seller[1].toNumber()); //获取退款后用户的数据
35   });
36   it('Test Case 2', async () => {
37     let instance = await BookShop.deployed();
38     await instance.addBook("bookA", "authorA", 80, 10, "imageA");
39     let book = await instance.getBook(1);
40     console.log(book[0].toNumber(), book[1], book[2], book[3].toNumber(), book[4].toNumber(), book[5], book[6]);
41     await instance.updateBook(1, 1, "bookA Modified", "authorA Modified", 10, 0, "imageA Modified"); //掩码位11表示修改name
42     book = await instance.getBook(1);
43     console.log(book[0].toNumber(), book[1], book[2], book[3].toNumber(), book[4].toNumber(), book[5], book[6]);

```

五、Dapp 测试与运行

5.1 登录

在可选框中选择由 Ganache 私有链生成的任意一个账号进行登录，Ganache 中第一个账户为商家账号，其余的每个账号可被视为买家账号。



5.2 用户购买书籍

选择一个用户账号进行登录后，可见账户余额为 200 元，进入如下界面：



选择 2 本书进行购买，将弹出购买确认窗口



用户若余额不足，则将提示无法购买



查看用户订单及 Ganache 页面, 可见书籍已成功购买, 且相关数据皆已更新:

Book Shop

商品 订单

搜索...

退出

用户: User1, 余额: ¥6

订单编号	商品名称	下单时间	实付金额	当前状态	操作
1	Java核心技术卷I	2019-12-23 19:59:04	¥50	已完成	申请退款
2	Git学习指南	2019-12-23 19:59:29	¥44	已完成	申请退款

ADDRESS

0x738F742a4B4B3a6e8E003A85B20Fd81376b0E9e0

BALANCE

99.99 ETH

TX COUNT

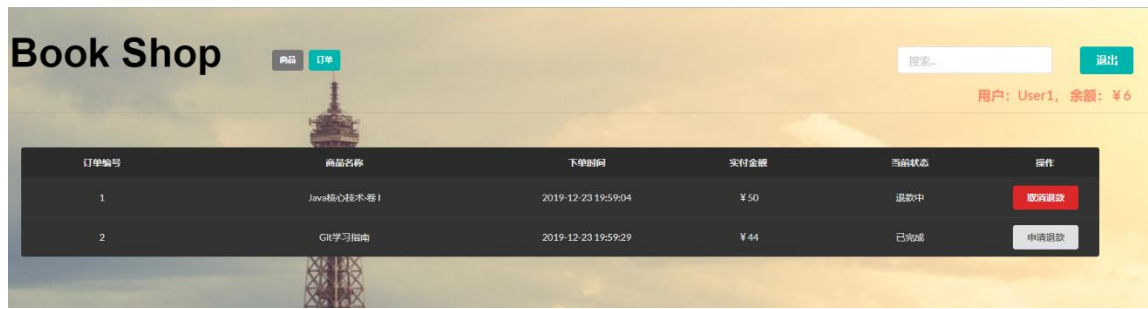
3

INDEX

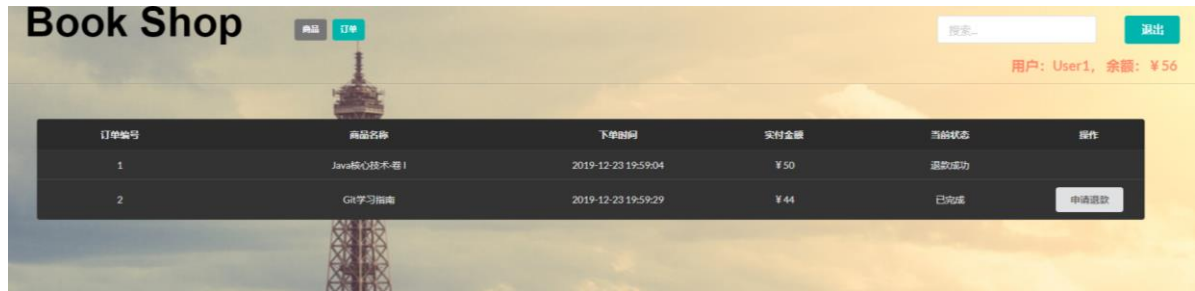
2

5.3 用户申请退款

如图，用户点击“申请退款”按钮后，该条订单进入申请退款状态：



商家确认退款后，该条订单进入退款成功状态，且将观察到用户的余额增加



4.4 卖家管理书籍

卖家添加书籍：

添加书籍

书籍名称
计算机网络 (第4版)

书籍作者
James F. Kurose

书籍封面
/images/book6.jpg

书籍数量
0

出售单价
66

提交 取消

卖家修改图书信息，如图为修改前的图书信息：

修改书籍

书籍编号

9

书籍名称

修改后的书籍

书籍作者

应承峻

书籍封面

images

书籍数量

0

出售单价

100

提交

取消

修改信息后确认，如图，图书信息修改成功

8	Git学习指南	吕荣兴	/images/book8.jpg	15	¥44	修改	删除
9	修改后的书籍	应承峻	images	0	¥100	修改	删除

卖家删除图书，如图，卖家对该种类的书籍进行删除操作，与前面图像的对比得知，该类型图书已经顺利被删除：

Book Shop

商品 订单

搜索

退出

用户: Seller, 余额: ¥0, 收益: ¥0

添加书籍

书籍编号	书籍名称	书籍作者	书籍封面	书籍数量	书籍价格	操作
1	未来简史	韩文东	/images/book1.jpg	80	¥80	修改 删除
2	编译原理	张雪梅	/images/book2.jpg	32	¥120	修改 删除
3	区块链技术及应用	应承峻	/images/book3.jpg	0	¥120	修改 删除
4	算法导论	Thomas H.Cormen	/images/book4.jpg	10	¥88	修改 删除
5	Java核心技术-卷1	凯S.霍斯特曼	/images/book5.jpg	20	¥50	修改 删除
6	计算机组成 (第4版)	James F. Kurose	/images/book6.jpg	0	¥66	修改 删除
7	操作系统概念	兰德尔·伯克曼	/images/book7.jpg	666	¥72	修改 删除
8	Git学习指南	曹福哈尔	/images/book8.jpg	15	¥44	修改 删除

5.5 卖家管理订单

退出用户方登录，重新登入卖家账号，进入后为所有商品及订单的查看，且可对书籍进行相应的增加、删除、修改操作

用户: Seller, 余额: ¥94, 收益: ¥94

添加书籍

书籍编号	书籍名称	书籍作者	书籍封面	书籍数量	书籍价格	操作
1	未来简史	韩江尔	/images/book1.jpg	80	¥80	修改 删除
2	编译原理	张雪峰	/images/book2.jpg	32	¥120	修改 删除
3	区块链技术及应用	应承峻	/images/book3.jpg	0	¥120	修改 删除
4	算法导论	Thomas H.Cormen	/images/book4.jpg	10	¥88	修改 删除
5	Java核心技术-卷I	凯S.霍斯特曼	/images/book5.jpg	19	¥50	修改 删除
6	计算机网络 (第4版)	James F. Kurose	/images/book6.jpg	0	¥66	修改 删除
7	操作系统概念	[美]西尔伯查茨	/images/book7.jpg	666	¥72	修改 删除
8	Git学习指南	曹翠贝尔	/images/book8.jpg	14	¥44	修改 删除

选择查看订单后，可对申请退款的订单进行确认操作

Book Shop

商品 订单

搜索...

退出

用户: Seller, 余额: ¥1122, 收益: ¥1122

订单编号	商品名称	下单用户	下单时间	实付金额	当前状态	操作
1	Java核心技术-卷I	0ed82e62279025493d2104429c9404bc901bbe0	2019-12-23 19:58:04	¥50	退款的申请	
2	Git学习指南	0ed82e62279025493d2104429c9404bc901bbe0	2019-12-23 19:58:29	¥44	已退款	
3	编译原理	0ed8a6c932f90aa818130be1605566251e0ebda5bb	2019-12-23 20:08:35	¥120	退款中	同意退款
4	Java核心技术-卷I	0ed8a6c932f90aa818130be1605566251e0ebda5bb	2019-12-23 20:08:47	¥50	已完成	

确认退款后，该订单进入退款成功状态，商家收益及余额进行更新

Book Shop

商品 订单

搜索...

退出

用户: Seller, 余额: ¥50, 收益: ¥50

订单编号	商品名称	下单用户	下单时间	实付金额	当前状态	操作
1	Java核心技术-卷I	0x738f742a4b4b3a6e8e003a85b20fd81376b0e9e0	2019-12-25 14:34:03	¥50	已完成	
2	Git学习指南	0x738f742a4b4b3a6e8e003a85b20fd81376b0e9e0	2019-12-25 14:34:13	¥44	退款成功	

六、总结与思考

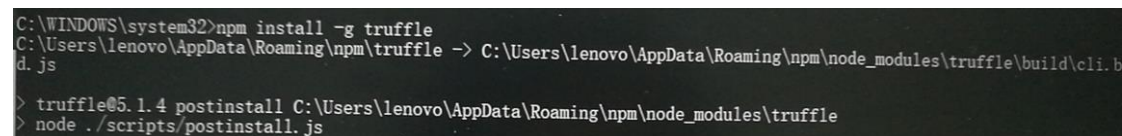
6.1 问题与解决

由于对 truffle 框架和对 Solidity 智能合约编写的一些默认要求的了解不足，我们在项目实际的编写与测试部署的过程中，遇到了许多问题，但最终通过对问题的思考以及向老师助教请教的方式，我们最终解决了这些问题，并由此对 Solidity 编写智能合约以及 truffle 框架有了更为深入的认识与理解。

6.1.1 Truffle 框架安装

问题描述：

在采用命令 `npm install -g truffle` 安装 truffle 时,组员中有出现卡死在 postinstall 步骤，无法正常继续的情况。具体错误如下图所示：



```
C:\WINDOWS\system32>npm install -g truffle
C:\Users\lenovo\AppData\Roaming\npm\truffle -> C:\Users\lenovo\AppData\Roaming\npm\node_modules\truffle\build\cli.js
> truffle@5.1.4 postinstall C:\Users\lenovo\AppData\Roaming\npm\node_modules\truffle
> node ./scripts/postinstall.js
```

问题解决：

在网上查询相关问题及解决方案后，我们发现是由于 ip 的代理问题，因此我们改为选择淘宝的镜像进行下载，采用如下命令，并最终成功安装。

```
cnpm install -g truffle
```

6.1.2 智能合约中的两类函数

问题描述：

在利用 Truffle 框架测试智能合约中的 `deleteBook()` 功能时，我们选择了让该函数返回一个 `uint` 类型变量，以判断删除操作是否成功被执行。但在测试中，我们却发现，原本应该输出的一个 `uint` 值，却变成了一整个区块。错误截图及代码

如下:

测试部分代码:

```
let vvv = await instance.deleteBook(1); //删除书籍
console.log("vvv = ", vvv.valueOf());
```

测试输出截图:

[illegible]

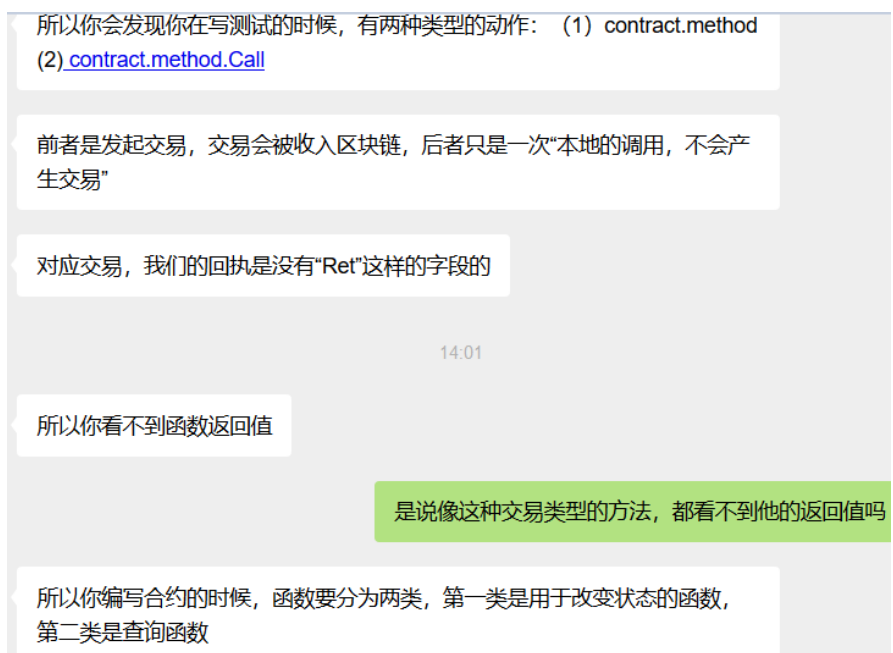
对此，我们上网查询了相关错误信息，仔细分析了代码逻辑，并对合约中其他类似函数进行了测试，发现有部分也出现了类似的错误。最终，我们选择向老师请教，并最终成功解决了问题。

问题解决:

按照以太坊推荐的编程范式，智能合约中成员函数应该被分为两类，一类是状态修改函数，另一类是状态读取函数。状态修改函数通过交易的方式被调用，是全球共识的，为了提高安全性，防止恶意用户利用 `ret` 字段进行垃圾数据的传递，增加区块链节点维护压力，状态修改类的函数回执中不应该有调用信息，仅仅包含一个表示状态执行成功与否的布尔值。对于合约状态的查询，应该采用第二类函数，采取 `contract.method.call(...params)` 的方式，发起本地函数调用来进行合约状态的读取。

在与老师交流后，我们了解到如上信息，并立即对相应的错误代码进行了修改，最终增加一个状态修改函数，并删除原 `deleteBook()`函数的返回值后，测试成功。

交流截图:



6.1.3 智能合约可升级

智能合约可升级是“工程友好性”与“可信程度”在一定程度上的矛盾，为了实现智能合约可升级的操作，我们查阅了 github 上 Proxy 相关源代码，并对该源代码的内部逻辑进行了组内的讨论与学习，在大体了解其代码逻辑与状态变量分离的主要思路，并就相关编程技术问题请教老师后，我们使用了该合约范式，使得我们的合约实现了可升级技术，在有新的合约需要发布时，只需部署新合约，并更改 Proxy 中的索引，即可更简单的实现合约升级。

下图为采用 Proxy 范式后的逻辑：



6.2 反思与总结

在这次的 Dapp 项目编写与部署中，我们在对 truffle 框架和 Solidity 语言的学习过程中更为深入的了解区块链的结构以及其运转过程，对其去中心化的优势认识的更为深刻。

在解决问题的过程中，我们也从状态修改函数和状态读取函数的设计中，体会到了区块链设计的巧妙之处，以及安全问题的重要性。更为惊叹于区块链技术的精巧设计。

但在我们也存在一些需要改进的地方，如项目实现的功能还可以更加丰富，对一些具体功能的处理还可以更细致，如在删除书籍时不是按照类别删除，而是按照具体的一本书来删除。

总而言之，Dapp 去中心化的优势十分显著，它的使用能帮助提高信息的可信度及不可篡改性，让区块中存储的信息更安全可靠。