
海南大学网络空间安全学院（密码学 院）

2023-2024 学年度第二学期

区块链安全（Y00310） 课程论文



报告题目： 基于以太坊的杂货铺 Dapp 开发

姓名： 甄五四

学号： 20213006839

授课教师： 张亮

完成时间： 2024 年 7 月 14 日

摘要

本文介绍了基于以太坊区块链的杂货铺 DApp 的开发过程及相关实验。共享经济的兴起推动了基于区块链技术的去中心化应用（DApp）的发展，本文设计的杂货铺 DApp 旨在通过智能合约实现用户之间的物品共享和交换。文章首先介绍了区块链的基本原理及以太坊平台的特点，重点阐述了智能合约的概念和 Solidity 编程语言的应用。在实验环境部分，详细描述了以太坊环境的搭建过程，包括 Node.js、geth、Truffle 等工具的配置和使用。通过编写智能合约代码，实现了杂货铺 DApp 的核心功能，包括商品添加、领取及归属查询等。随后，通过测试文件验证了合约的功能正确性，并展示了在 Ganache 私链上部署合约和与之交互的过程。最后，使用 MetaMask 作为前端工具，演示了用户如何在 DApp 中进行商品添加和认领操作。本文通过实验验证了杂货铺 DApp 的设计与实现，展示了区块链技术在共享经济场景中的应用潜力，为后续基于以太坊的 DApp 开发提供了实践经验和参考。

关键词：区块链技术；智能合约；以太坊；杂货铺 DApp

Abstract

This paper presents the development process and related experiments of a grocery store DApp based on the Ethereum blockchain. The rise of the sharing economy has driven the development of decentralized applications (DApps) based on blockchain technology. The designed grocery store DApp aims to facilitate item sharing and exchange among users through smart contracts. The paper begins with an introduction to the basic principles of blockchain and the characteristics of the Ethereum platform, focusing on the concept of smart contracts and the application of the Solidity programming language. In the experimental setup section, it details the setup of the Ethereum environment, including the configuration and use of tools such as Node.js, geth, and Truffle. By writing smart contract code, the core functionalities of the grocery store DApp are implemented, including item listing, claiming, and ownership querying. Subsequently, the functionality of the contract is validated through test scripts, demonstrating the deployment of the contract on the Ganache private chain and the interaction process. Finally, using MetaMask as a frontend tool, it illustrates how users can add and claim items within the DApp. This paper validates the design and implementation of the grocery store DApp through experiments, showcasing the potential application of blockchain technology in the sharing economy and providing practical experience and references for future DApp development on the Ethereum platform.

Keywords: Blockchain technology; Smart contracts; Ethereum; Grocery store DApp

基于以太坊的杂货铺 DApp 开发

1. 问题背景和相关知识介绍

1.1 问题背景

在当今社会，共享经济的概念越来越流行。基于区块链技术的分布式应用（DApp）提供了一种新的方式来实现共享经济的理念，使得参与者可以直接在无需第三方干预的情况下进行交换和共享。以太坊是目前最流行的智能合约平台之一，它允许开发者编写基于区块链的去中心化应用程序。智能合约是一种在区块链上运行的自动化合约，可以执行预先编码的规则和逻辑，而无需信任第三方。本课设是基于以太坊的杂货铺 DApp，旨在允许用户分享和交换不再需要的物品，以此来获得其他人的物品，简单来说就是共享自己无用的物品。

1.2 区块链相关知识

1.2.1 区块链

区块链是一种分布式账本技术，它可以在没有中心化管理机构的情况下，记录和验证各种类型的交易。它的本质是一个由多个节点组成的分布式网络，每个节点都可以看到整个网络上的交易记录。每个交易都被加密并打包成一个块，然后通过共识机制被添加到整个链中。这个过程使得区块链的交易记录具有防篡改性和去中心化的特性，因为每个节点都具有对交易记录的完整拷贝。简单理解：区块链是一个链条，这个链上一旦发生了交易这个链上的所有人都知道并且记录，相当于一个账本。区块链是一个分布式网络，每个人都可以部署自己的节点加入到区块链网络中，成为其中一个节点。以太坊客户端是以太坊网络中的节点程序，运行客户端后节点加入以太坊网络，同时这个节点程序可以完成如创建账号、发起交易、部署合约、执行合约、挖掘区块等工作。节点生成区块的过程称为挖矿，挖矿就是节点打包一批交易数据成一个新区块，然后把这个区块广播到区块链网络，由其他节点去验证有效性，验证成功后，各节点将其作为新区块存储在本地，然后基于新区块继续竞争挖矿。挖矿成功的节点矿工会得到数字货币的激励，因此很多节点会加入生产区块的竞争。

1.2.2 共识算法

在挖矿的过程中，矿工需要付出算力来查找一个满足条件的数字，这种算力付出是无法伪造的。当前某节点运算出该数字后，其他节点可以很快验证该数字的有效性。这种为了持续生成区块而被所有网络节点认可的方案就叫做共识算法，而付出算力来证明自己工作的共识算法被称为 PoW（Proof of Work）共识算法。在区块链中，共识算法是确保网络安全和有效运行的关键机制之一。Proof of Work (PoW) 是最早和最知名的共识算法之一，通过节点的算力来竞争解决数学难题，验证交易并创建新的区块。这种方法确保了每个节点付出实际的计算资源证明其工作的难以伪造性，但由于其需要大量能源消耗和交易验证时间长的特点，限制了交易处理能力和速度。随后，为了解决 PoW 的问题，出现了 Proof of Stake (PoS) 共识算法。在 PoS 中，节点的选择和新区块的创建不再依赖于算力竞赛，而是依据节点持有的数字货币数量（即权益）。验证者（Validators）通过持有一定数量的货币，参与网络验证和安全维护，并有机会被选中创建下一个区块。这种方法降低了能源消耗，并加快了交易确认速度，但也引入了一些新的挑战 and 复杂性，如节点的经济激励和安全性保障。

1.2.3 以太坊和智能合约

以太坊（Ethereum）是一种开源的区块链平台，于 2015 年由 Vitalik Buterin 等人创建。相较于比特币等早期区块链，以太坊不仅支持加密货币交易，还引入了智能合约的概念，使得开发者能够在其上构建和部署各种去中心化应用（DApps）。以太坊的关键特点包括以太坊虚拟机（Ethereum Virtual Machine, EVM），这是一个可在所有参与节点上运行的分布式计算机。智能合约是以太坊上的核心功能，它们是自动执行的合约或程序，由 Solidity 等编程语言编写，用于处理和执行特定条件下的交易和逻辑。智能合约是一个运行在以太坊链上的一个程序。它是位于以太坊区块链上一个特定地址的一系列代码（函数）和数据（状态）。智能合约也是一个以太坊帐户，我们称之为合约帐户。这意味着它们有余额，可以成为交易的对象。但是，他们无法被人操控，他们是被部署在网络上作为程序运行着。个人用户可以通过提交交易执行智能合约的某一个函数来与智能合约进行交互。智能合约能像常规合约一样定义规则，

并通过代码自动强制执行。默认情况下，你无法删除智能合约，与它们的交互是不可逆的。以太坊通过其去中心化的特性和智能合约的灵活性，推动了区块链技术的发展和应用场景的扩展。它成为了许多区块链初创公司和大企业的首选平台，用于构建各种应用，从金融服务到供应链管理，以及数字身份验证等广泛领域的解决方案。

1.2.4 Solidity 语言

Solidity 是一种智能合约编程语言，被广泛用于以太坊平台上的去中心化应用（DApps）开发。它类似于 JavaScript，但是具有更多的安全特性和区块链特定的功能。Solidity 语言的主要特点包括：

- 安全性：**Solidity 是为了编写智能合约而设计的，因此它具有很多内置的安全特性，可以帮助开发人员避免一些常见的漏洞和攻击。
- 可编程性：**Solidity 是一种高级编程语言，它提供了诸如条件语句、循环、函数等常见的编程结构，使得开发人员可以轻松地编写复杂的智能合约。
- 与以太坊集成：**Solidity 是以太坊平台上最流行的智能合约语言，它与以太坊虚拟机（EVM）集成，可以与以太坊上的其他智能合约、代币和其他 DApps 进行交互。
- 可读性：**Solidity 语言的语法结构类似于 JavaScript，使得开发人员可以更容易地阅读和理解现有的代码。

2. 实验环境

本文以 Ubuntu 22.04.4 LTS 为基础环境。在开始前，先运行 `sudo apt update` 更新系统组件。在此系统上安装部署了以太坊环境，具体的实验配置及版本如下表所示：

Name	Version
Node.js	20.15.1
geth	1.14.7-stable
Go	1.22.5
Yarn	1.22.22
Ganache	2.7.1

Truffle	5.11.5
MetaMask	11.12.2
Solidity	0.5.16
Web3.js	1.10.0

3. 关键源码功能的注释

完整代码已同步到 [ZhenWusi/DAPP \(github.com\)](https://github.com/ZhenWusi/DAPP)，本节仅对关键代码功能注释

3.1 编写智能合约

contracts/Grocery.sol 如下，代码逻辑比较简单，每个人都可以添加商品，并且获得领取次数，已经被领取的商品不允许被再次领取。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.0; // 指定了 Solidity 编译器的版本范围。^0.5.0 表示使用 0.5.0 版本及以上的编译器
pragma experimental ABIEncoderV2;
// 这是一个实验性特性的声明。ABIEncoderV2 是 Solidity 中的一个特性，
// 允许函数接受和返回任意类型的数组和结构体。由于其实验性质，需要显式声明以启用该功能。
contract Grocery {
    // 商品结构体
    struct Goods {
        string name; // 商品名称
        string img; // 商品图片链接
        bool isClaim; // 是否已被领取
    }
    mapping(uint => address) goodsMap; // 商品的归属地址映射
    Goods[] public goodsList; // 所有商品列表
    mapping(address => uint[]) claimList; // 用户领取的商品列表
    mapping(address => int) claimNum; // 用户已领取数量统计
    int maxClaim = 10; // 最大领取数量，可以通过添加商品来增加领取机会
    // @notice 添加商品
    function AddGoods(string memory _name, string memory _img) public returns (uint){
        uint id = goodsList.push(Goods(_name, _img, false)) - 1; // 创建新商品并添加到列表中
```

```

        goodsMap[id] = msg.sender; // 将商品与发布者关联
        claimNum[msg.sender] -= 1; // 每添加一个商品，用户领取机会减一
        return id; // 返回商品 ID
    }
    // @notice 领取商品
    function Claim(uint _id) public {
        require(claimNum[msg.sender] <= maxClaim, "您已经领取的足够多了"); // 检查用户是否达到领取上限
        require(!goodsList[_id].isClaim, "该商品已经被认领"); // 检查商品是否已被领取
        goodsMap[_id] = msg.sender; // 将商品分配给领取者
        goodsList[_id].isClaim = true; // 标记商品已被领取
        claimList[msg.sender].push(_id); // 将商品 ID 添加到用户的领取列表中
        claimNum[msg.sender] += 1; // 增加用户已领取数量
    }
    // @notice 获取所有的商品
    function GetAllGoods() public view returns (Goods[] memory){
        return goodsList; // 返回所有商品列表
    }
    // @notice 获取商品归属地址
    function GoodsOf(uint _id) public view returns (address) {
        return goodsMap[_id]; // 返回商品的归属地址
    }
}

contracts/Migrations.sol 用于迁移合约的状态
// SPDX-License-Identifier: MIT
// 指定合约的许可证标识为 MIT 许可证
pragma solidity >=0.4.22 <0.8.0;
// 指定 Solidity 编译器的版本范围，要求版本大于等于 0.4.22 且小于 0.8.0
contract Migrations {
    address public owner = msg.sender;
    // 合约的所有者地址，默认为部署合约的账户地址
    uint public last_completed_migration;
    // 记录最后一次完成的迁移版本号
    modifier restricted() {
        // 修饰符：限制调用者必须是合约的所有者
        require(
            msg.sender == owner,
            "This function is restricted to the contract's owner"
        );
        _; // 执行函数体
    }
    function setCompleted(uint completed) public restricted {
        // 设置最后完成的迁移版本号，只有合约所有者可以调用
        last_completed_migration = completed;
    }
}

```

```
}  
}
```

3.2 编写部署脚本

migrations/1_initial_migration.js 如下

```
const Migrations = artifacts.require("Migrations");  
//Truffle 的部署器对象 deployer 来部署名为 Migrations 的 Solidity 合约。  
//artifacts.require("Migrations") 用于加载合约的 ABI 和地址信息。  
//然后, deployer.deploy(Migrations) 将合约部署到区块链上。  
module.exports = function(deployer) {  
  deployer.deploy(Migrations); // 部署 Migrations 合约  
};
```

migrations/2_initial_grocery.js 如下

```
const Grocery = artifacts.require("Grocery");  
module.exports = function (deployer) {  
  // 部署 Grocery 合约, 并设置 gas 和 gasPrice  
  deployer.deploy(Grocery, {  
    gas: 3000000, // 设置 gas 上限  
    gasPrice: 10000000000 // 设置 gasPrice, 单位为 wei  
  });  
};
```

3.3 编写测试文件

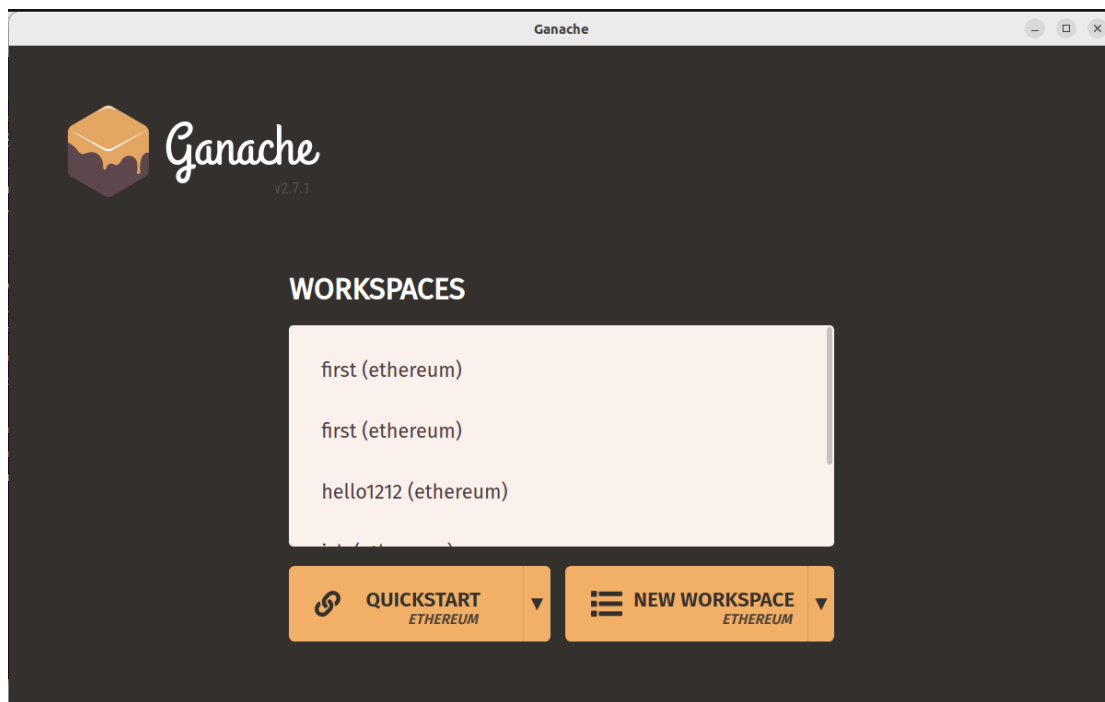
```
pragma solidity ^0.5.0;  
import "truffle/Assert.sol"; // 导入 Truffle 断言库  
import "truffle/DeployedAddresses.sol"; // 导入 Truffle 部署地址库  
import "../contracts/Grocery.sol"; // 导入要测试的 Grocery 合约  
contract TestGrocery {  
  Grocery grocery = Grocery(DeployedAddresses.Grocery());  
  // 发布商品测试  
  function testAddGoods() public {  
    // 调用合约的 AddGoods 函数来发布商品, 获取商品的 id  
    uint id = grocery.AddGoods("GoLang", "https://img-blog.csdnimg.cn/20190629154954578.png?x-oss-process=image/resize,m_fixed,h_64,w_64");  
    uint expected = 0; // 预期的商品 id 应为 0  
    // 使用断言库来验证实际返回的 id 是否与预期相符  
    Assert.equal(id, expected, "Goods id is not 0");  
  }  
  // 领取商品测试  
  function testClaim() public {  
    // 预期领取者的地址为当前测试合约的地址, 因为交易由测试合约发起
```

```
        address expected = address(this);
        // 调用 Grocery 合约的 Claim 函数来领取商品 id 为 0 的商品
        grocery.Claim(0);
        // 获取领取后商品 id 为 0 的实际所有者地址
        address claimAddr = grocery.GoodsOf(0);
        // 使用断言库验证实际领取者地址是否与预期相符
        Assert.equal(claimAddr, expected, "Owner of goods id 0 should be recorded.");
    }
}
```

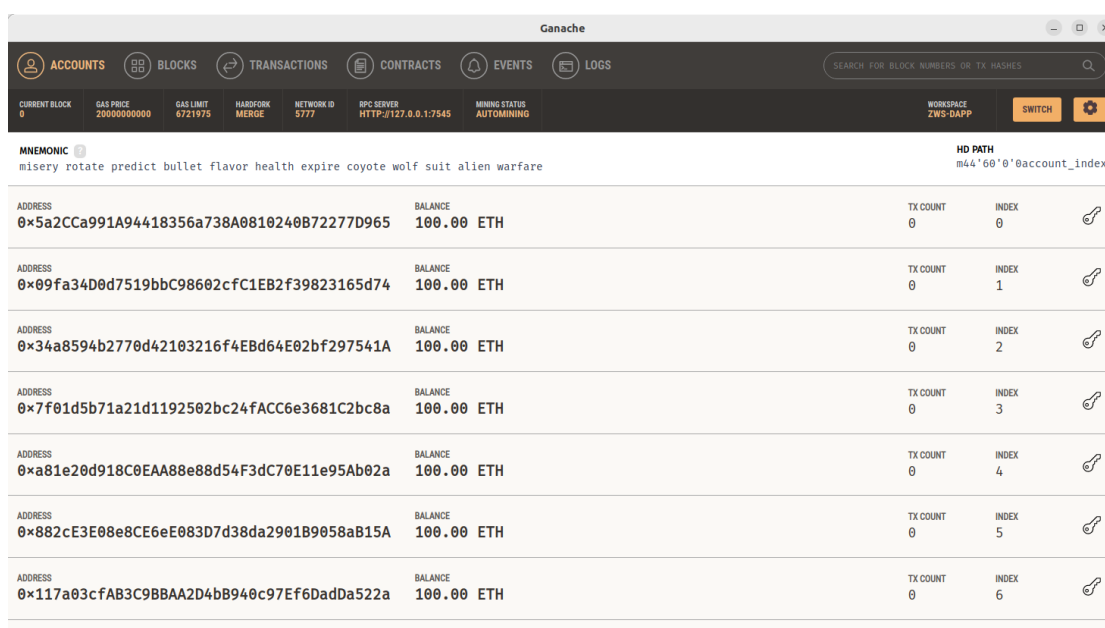
4. 程序运行截图及解释

4.1 配置网络

我们可以将上面创建的智能合约发布到以太坊公链上去。但这个操作其实没有任何意义，只浪费了以太坊的资源。而且还奇慢无比，而且还浪费钱(以太坊)。所以，在进行编程、调试及测试时一般都把智能合约发布到一个和以太坊公链相似的私链上，这样就能又快又省钱的进行程序开发。我使用 **Ganache** 来创建私链，它是一个以太坊的个人开发环境，你可以在上面部署合约、开发程序和进行测试。它有桌面版本和命令行工具版本，同时提供对 windows、Mac 和 Linux 的支持。打开 ganache，启动区块链，默认会生成 10 个有 100 个 ETH 的账户。默认的 PRC SERVER 为 `http://127.0.0.1:7545`。为了将我们的智能合约发布到 Ganache 私链上，我们还需要对 `truffle-config.js` 进行修改，一般情况下只要取消注释就可以。



左边的按钮是快速启动，它的数据不会保存，每次启动后都是全新的开发环境。右边的按钮是保存当前的数据到相应的工作空间中，可以有多个工作空间。我们选择右边的 new workspace。端口设为 7545，其余默认。



这个界面最上面一栏是导航菜单，接下来一行是各种信息，右边是切换工作空间和设置按钮。第三行是助记词，所有的地址都是根据它生成的。界面主体有 10 个账号，分别列出了每个账号的地址，余额，已经完成的交易数量、账号数组中的索引、显示私钥按钮等。点击那个钥匙图标，就会显示它的私钥。点击导航上的 Blocks，可以看到目前 Block 是 0。这是因为 Ganache 挖矿机制决定的，每

一个交易产生一个 block，所以一个 block 也只会有一笔交易。在我们交易后就会产生一个 block，目前还没有。点击 Transactions，会显示没有交易。点击 CONTRACTS，因为他们工作空间并没有添加 truffle 工程，所以这里也是空的。

4.2 部署合约并进行交互

编译合约：命令行执行 `truffle compile`。成功后有如下图输出，并且项目会多个 build 目录，里面有编译好的 json 合约文件

```
zws@zws-virtual-machine:~/zws-dapp$ truffle compile

Compiling your contracts...
=====
> Compiling ./contracts/Grocery.sol
> Compilation warnings encountered:

    project:/contracts/Grocery.sol:3:1: Warning: Experimental features are turned on. Do not use experimental features on live deployments
    pragma experimental ABIEncoderV2;
    ^
    ~~~~~

> Artifacts written to /home/zws/zws-dapp/build/contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
```

部署合约：`truffle migrate`，`truffle migrate --reset` // 如果合约修改了需要重新部署，则需要添加 `reset` 参数。成功后有如下图输出，创建了两个合约

```
> account:      0x5a2CCa991A94418356a738A0810240B72277D965
> balance:      99.995762657029439902
> gas used:     193243 (0x2f2db)
> gas price:    3.037909716 gwei
> value sent:   0 ETH
> total cost:   0.000587054787248988 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:   0.000587054787248988 ETH

2_initial_grocery.js
=====

Replacing 'Grocery'
-----
> transaction hash: 0xdca109fd54b1ff7bd2b14b99cafedd8d2fc1c332b85112927a6f718011551324
> Blocks: 0
> contract address: 0x14014F6D9eCbd4a51c7422E4877812fdb1cd8240
> block number: 7
> block timestamp: 1720894286
> account: 0x5a2CCa991A94418356a738A0810240B72277D965
> balance: 99.986940767658283588
> gas used: 868584 (0xd40e8)
> gas price: 10 gwei
> value sent: 0 ETH
> total cost: 0.00868584 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.00868584 ETH

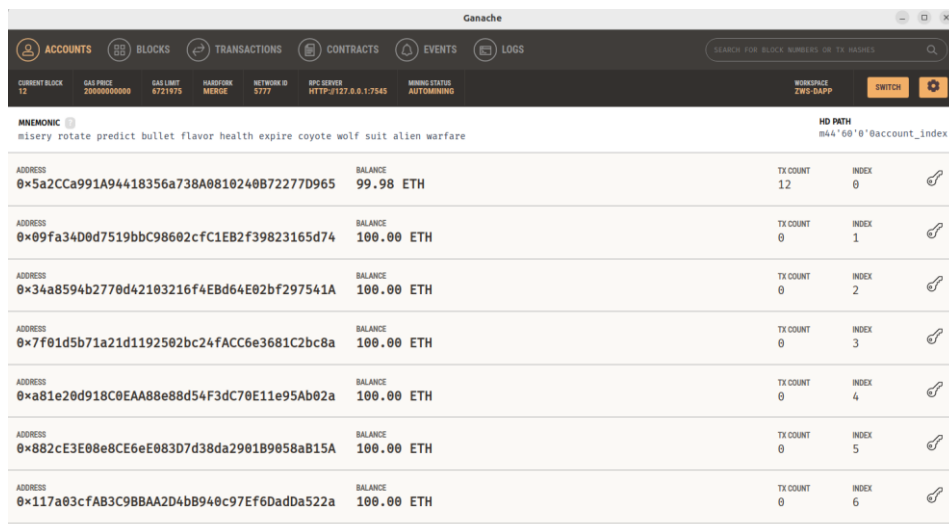
Summary
=====
> Total deployments: 2
> Final cost: 0.009272894787248988 ETH
```

参数解释：

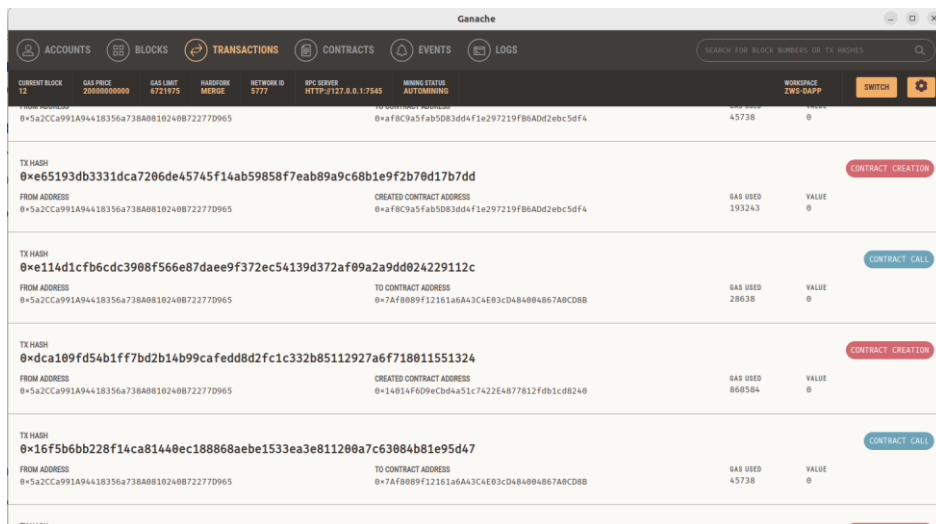
- Transaction Hash: 交易的哈希值，唯一标识此次交易。

- **Blocks:** 区块数，指该交易被确认的区块数量。通常，交易确认数越多，表示该交易越安全，因为它已经被多个区块确认过了。
- **Contract Address:** 合约地址，智能合约部署后在区块链上的地址。这个地址用于唯一标识你的智能合约，其他用户可以通过这个地址与你的合约进行交互。
- **Block Number:** 区块号，交易所在区块的编号。每个区块都有一个唯一的编号，这个编号可以确定交易的位置和顺序。
- **Block Timestamp:** 区块时间戳，指示交易发生的时间。这对于记录交易的时间顺序和验证交易的时间戳是非常有用的。
- **Account:** 发起交易的账户地址。
- **Gas Used:** 消耗的 Gas 数量，Gas 是执行智能合约操作所需的单位。Gas 是以太坊中执行交易和操作的计量单位，每个操作会消耗一定数量的 gas。部署合约是一种消耗 gas 的操作，消耗量取决于合约的复杂性和初始化过程中的计算量。
- **Gas Price:** Gas 的价格，以 Gwei 为单位，用于计算交易的费用。Gas Price 表示每单位 gas 的价格，以 gwei (giga wei) 为单位。在以太坊上执行交易和操作时，需要支付一定数量的 gas 作为手续费，Gas Price 决定了你愿意为每单位 gas 支付多少以太币。较高的 Gas Price 可能会加快交易的确认速度，但也会增加交易的成本。
- **Value Sent:** 发送的 ETH 数量。
- **Total Cost:** 总费用，以 ETH 表示。

可以在打开的 Ganache 中看到区块变化，默认花费第一个帐户的 ETH。然后转到 TRANSACTIONS 标签中会发现已经有多个事件。



ADDRESS	BALANCE	TX COUNT	INDEX
0x5a2CCa991A94418356a738A0810240B72277D965	99.98 ETH	12	0
0x09fa34D0d7519bbC98602cfc1EB2f39823165d74	100.00 ETH	0	1
0x34a8594b2770d42103216f4EBd64E02bf297541A	100.00 ETH	0	2
0x7f01d5b71a21d1192502bc24fACC6e3681C2bc8a	100.00 ETH	0	3
0xa81e20d918C0EAA08e88d54F3dC70E11e95Ab02a	100.00 ETH	0	4
0x882cE3E08e8CE6eE083D7d38da2901B9058aB15A	100.00 ETH	0	5
0x117a03cfAB3C9BBA2D4b8940c97Ef6DadDa522a	100.00 ETH	0	6



4.3 测试

4.3.1 控制台测试

truffle console 进入 truffle 控制台，看到 truffle(development) 表示连接上了 development 环境

let accounts = await web3.eth.getAccounts() // 获取当前所有的账户，也就是初始化的是个地址

accounts // 打印地址

let instance = await Grocery.deployed() // 获取合约实例

```
zws@zws-virtual-machine:~/zws-dapps$ truffle console
truffle(development)> let accounts = await web3.eth.getAccounts() // 获取当前所有的账户，也就是初始化的是个地址
undefined
truffle(development)> accounts // 打印地址
[
  '0x5a2CcA991A94418356a738A0810240B72277D965',
  '0x09fa34D0d7519bbC98602cfc1EB2f39823165d74',
  '0x34a8594b2770d42103216f4EB064E02bf297541A',
  '0x7f01d5b71a21d1192502bc24fACC6e3681C2bc8a',
  '0xa81e20d918C0EAA88e88d54f3dC70E11e95Ab02a',
  '0x882cE3E08e8CE6eE083D7d38da2901B9058aB15A',
  '0x117a03cfAB3C9BBAA2D4bB940c97F6D0ad0a522a',
  '0x279a039EF7FFaBA30b7B181B00aB364129F0c0cd',
  '0xFee88C005346663Ad30e0E82354624E46a04f88',
  '0xeeef4ebf1F07aAbF09B55B8716753bbEDEC84D365'
]
truffle(development)> let instance = await Grocery.deployed() // 获取合约实例
undefined
truffle(development)>
```

用第二个账户去添加商品添加商品，成功后返回交易详情。并且 Ganache 也会记录事件。

```
instance.AddGoods("zombie","https://dss0.bdstatic.com/70cFvHSh_Q1YnxGkpoWK1HF6hhy/it/u=3994931373,529771369&fm=26&gp=0.jpg", {from: accounts[1]})
```

[illegible]

用第三个账户取认领 `instance.Claim(0, {from: accounts[2]})` 成功后返会交易详情
 获取商品归属 `instance.GoodsOf(0)` 成功后返回地址。获取所有商品列表
`instance.GetAllGoods()`

4.3.2 编写 js/sol 测试文件来测试

```
truffle(development)> test
Using network 'development'.

Compiling your contracts...
=====
> Compiling ./test/TestGrocery.sol
> Compilation warnings encountered:

   project:/contracts/Grocery.sol:3:1: Warning: Experimental features are turned on. Do not use experimental features on live deployment
   s.
   pragma experimental ABIEncoderV2;
   ^-----^

> Artifacts written to /tmp/test--110764-9IRucn01SgKw
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang

TestGrocery
  ✓ testAddGoods (366ms)
  ✓ testClaim (471ms)

2 passing (15s)
```

4.4 前端测试

使用 MetaMask 添加本地测试网，点击 MetaMask 右上角网络选择自定义 Rpc 加入本地测试网，然后点击导入账户，在 Ganache 中找个有 ETH 的私钥导入。

ACCOUNT INFORMATION

ACCOUNT ADDRESS

0x5a2CCa991A94418356a738A0810240B72277D965

PRIVATE KEY

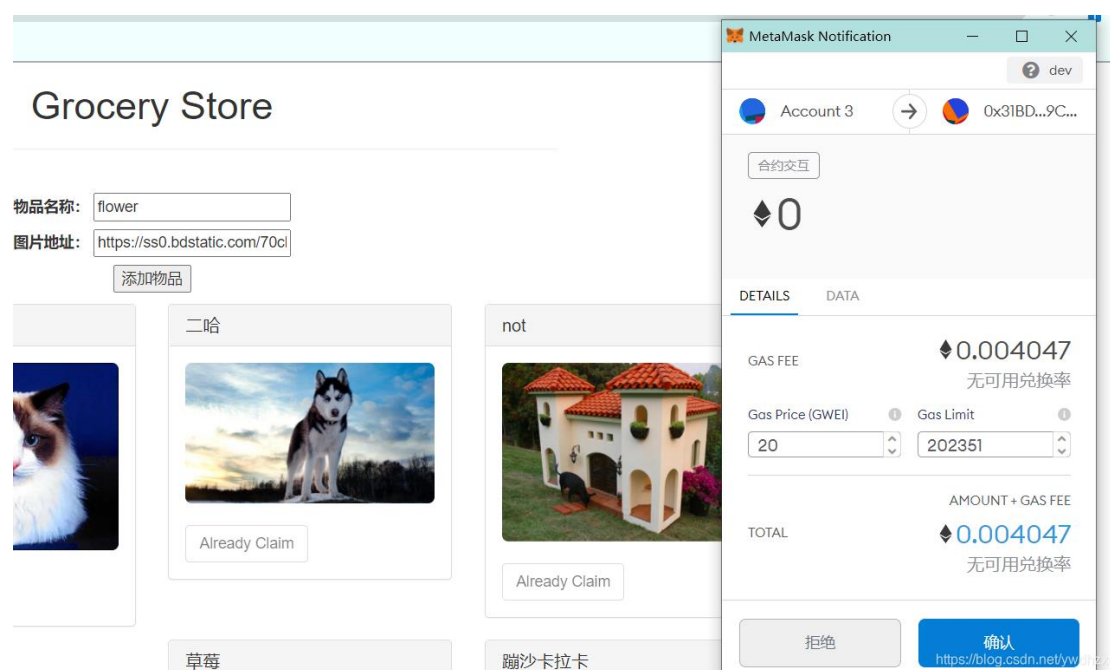
0xd278c8c082e7e54765c67c06a3fc7c42c1afb0b2d7e6a1b2e92aeafcefefab94

Do not use this private key on a public blockchain; use it for development purposes only!

DONE

这样就可以用 MetaMask 进行添加和认领物品的确认操作了，yarn run dev 启动 lite-server，输入图片网址和物品名称可以进行物品添加，之后还可以使用 MetaMask 进行转账。

。



5. 课程心得

5.1 实验心得

在本次以太坊区块链环境和智能合约开发的实验过程中,我获得了宝贵的经验和深刻的理解。在开始实验之前,我首先学习了如何搭建和配置以太坊开发环境。这包括安装和设置以太坊客户端(如 Ganache)、Solidity 智能合约编程语言的基础知识,并学会了如何使用 Truffle 框架来简化合约的开发和测试过程。通过这些步骤,我能够快速开始编写和部署智能合约。在实验中,我编写了几个简单但功能强大的智能合约。通过实践,我学会了如何定义合约的数据结构、编写函数来处理数据、以及如何使用事件来记录合约的状态变化。部署合约到测试网络(如 Ropsten)并进行测试是学习过程中不可或缺的一部分,这帮助我验证了合约的功能和逻辑正确性。我开发了一个简单的 DApp(去中心化应用),在这个应用中,用户可以共享和交换未使用的物品。通过这个项目,我学会了如何设计前端界面与智能合约进行交互,以及如何利用 Web3.js 库来连接用户的浏览器和区块链网络。对于智能合约的优点也有了更深入的理解,智能合约是预先编写好的代码,一旦部署在区块链上,便能自动执行其中设定的规则和逻辑。这种自动化执行消除了传统合同中需要的人工干预,大大提高了交易效率和可靠性。区块链智能合约在去中心化网络上运行,不依赖于单一的中央机构或第三方的信任。

合约的执行结果被记录在不同节点共享的分布式账本上，确保了数据的透明性和安全性。一旦智能合约部署在区块链上，其代码和执行过程将被永久记录，不可更改。这种特性确保了合约的不可篡改性，防止任何一方在未经允许的情况下篡改合约内容或执行结果。区块链智能合约通过密码学技术和分布式共识机制保证了高度的安全性。合约的执行过程经过多个节点验证和授权，减少了被恶意攻击或篡改的可能性，确保了合约的可信度和安全性。智能合约通过消除中介和简化流程，降低了交易的成本和时间。传统交易可能需要多个中介和复杂的程序，而智能合约可以在几分钟内自动执行，节省了时间和费用。区块链智能合约的执行过程对所有参与者都是透明的，并且能够追溯到合约的初始状态和历史操作。参与者可以查看合约的代码和所有交易记录，增强了交易的透明度和可追溯性。智能合约的灵活性和可编程性使其在多个行业和场景中得以应用，如金融服务、供应链管理、投票系统等。它创新提供了平台，可以实现复杂的业务逻辑和自动化流程。

这个过程不仅加深了我对以太坊生态系统的理解，还提升了我开发区块链应用的能力。我也学习了智能合约开发中的安全性问题和最佳实践。这包括如何避免重入攻击、如何处理异常情况、以及如何优化合约以减少 gas 费用等。这些知识对于确保合约的安全性和性能至关重要。在实验过程中我也发现了诸多问题，如对区块链环境配置的不了解，致使我在配置以太坊环境时花费大量时间。理论和实际的不结合，让我在实际操作运用以太坊时手足无措。以及刚刚开始学习 Solidity 语言，对于语法，版本等问题学习不透彻，好在参考多个开源代码有了一定的了解。对于 web3.js, 使用 HTTP 或 IPC 连接来和本地或远程以太坊节点进行交互，我的运用还是不太清楚，之后还需要继续研究。

总之，通过这些实验，我不仅学到了区块链技术和智能合约的具体实现方式，还培养了解决问题和创新的能力。我期待将这些经验应用到未来的项目中，并继续深入研究区块链技术的发展和应用。

5.2 课程学习心得

说起区块链，最先我了解到的就是比特币，那时便对区块链技术产生了好奇。之后在学校的多个讲座以及去年在海口的区块链技术与应用高峰论坛报告的学

习，让我对区块链技术有了一个更具体的理解，好奇与兴趣也更深了。这学期很幸运能够选修《区块链安全》这门课程。在本学期区块链安全的课程学习中，跟着张老师的带领，对于区块链的技术有了一定的了解与掌握，算是初步入门了，并且对于区块链技术的神奇之处产生了浓厚的兴趣，这也将我今后学习乃至读研的一个重要方向。区块链技术作为一种分布式数据库，其去中心化的特性使得数据的存储和传输更加安全和透明。通过学习，我了解到区块链如何通过共识机制确保数据的不可篡改性，以及智能合约如何在没有中介的情况下自动执行合同条款。智能合约是区块链的核心之一，它们是通过编程代码实现的自动化合同。在本次作业中，我学习了 Solidity 编程语言，并通过编写简单的智能合约来实现各种功能，如资产交换、投票系统等。这些合约不仅能够减少交易的复杂性，还能提高执行的效率和透明度。区块链技术不仅限于加密货币，它在供应链管理、物联网、金融服务等各个领域都有广泛的应用潜力。通过课程中的案例分析和实际项目，我了解到区块链可以解决传统中心化系统中存在的信任问题，降低交易成本，并提升数据安全性。区块链的优点包括去中心化、公开透明性，集体维护性、一致性、不可篡改性、匿名性等特性。然而，其也面临着扩展性、法律法规、能源消耗等方面的挑战。在今后的学习中，我将深入研究如何通过技术创新来解决这些问题。

通过课程的学习和实践，我深刻理解了区块链技术的核心概念和应用场景。未来，我希望能够进一步探索区块链在更多行业中的应用，并通过开发实际项目来加深对区块链技术的理解和掌握。

参考文献

- [1] 沈鑫, 裴庆祺, 刘雪峰. 区块链技术综述[J]. 网络与信息安全学报, 2016, 2(11): 11-20.
- [2] 邵奇峰, 金澈清, 张召, 等. 区块链技术: 架构及进展[J]. 计算机学报, 2018, 41(5): 969-988.
- [3] 袁勇, 倪晓春, 曾帅, 等. 区块链共识算法的发展现状与展望[J]. 自动化学报, 2018, 44(11): 2011-2022.
- [4] 欧阳丽炜, 王帅, 袁勇, 等. 智能合约: 架构及进展[J]. 自动化学报, 2019, 45(3): 445-457.
- [5] 黄秋波, 安庆文, 苏厚勤. 一种改进 PBFT 算法作为以太坊共识机制的研究与实现[J]. 计算机应用与软件, 2017, 34(10): 288-293.
- [6] 林冠宏. 区块链以太坊 DApp 开发实战[M]. 清华大学出版社 (崧博), 2019.
- [7] 林冠宏. 区块链 DApp 开发: 基于以太坊和比特币公链[M]. BEIJING BOOK CO. INC., 2020.

参考文档

[使用 Truffle 开发 Dapp - 简书 \(jianshu.com\)](https://jianshu.com/p/1a1a1a1a)[Truffle 翻译说明及概述 | Truffle 中文文档 - DApp 开发框架 | 深入浅出区块链 \(learnblockchain.cn\)](https://learnblockchain.cn/docs/truffle-overview.html)

[第二十三课 如何部署 TRUFFLE 智能合约到以太坊主网\(以宠物商店为例\)1, 摘要 2, 操作内容 3, 常见问题和解决方法 4, 参考-腾讯云开发者社区-腾讯云 \(tencent.com\)](#)

[Truffle & Web3.js 教程: 教你开发、部署第一个去中心化应用\(Dapp\) - 宠物商店 | 登链社区 | 区块链技术社区 \(learnblockchain.cn\)](#)

[快速入门 Truffle | Truffle 中文文档 - DApp 开发框架 | 深入浅出区块链 \(learnblockchain.cn\)](#)

[从零开始编写智能合约-手工打造 - YINHUI's BLOG \(yinhui1984.github.io\)](#)

[智能合约简介 | ethereum.org](#)