毕业设计（论文）

**西安邮电大学**

**毕业设计（论文）**

题目：基于区块链的去中心化皮肤交易系统设计与实现

学院： 网络空间安全学院

专业： 信息安全

班级： 安全1704

学生姓名： 许冠行

学号： 03173145

导师姓名： 罗维 职称： 讲师

起止时间： 2021 年 3 月 6 日 至 2021 年 6 月 8 日

毕业设计（论文）承诺书

本人所提交的毕业论文《基于区块链的去中心化皮肤交易系统设计与实现》是本人在指导教师指导下独立研究、写作的成果，论文中所引用他人的文献、数据、图件、资料均已明确标注；对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式注明并表示感谢。

本人深知本承诺书的法律责任，违规后果由本人承担。

签名：

日期： 年 月 日

西安邮电大学本科毕业设计(论文)选题审批表

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 申报人 | 罗维 | | | 职称 | 讲师 | | 学院 | | 网络空间安全学院 | | | |
| 题目名称 | 基于区块链的去中心化皮肤交易系统设计与实现 | | | | | | | | | | | |
| 题目来源 | 科研 | | **√** | | | | 教学 | |  | 其它 | |  |
| 题目类型 | 硬件  设计 | |  | 软件  设计 | **√** | | 论文 | |  | 艺术  作品 |  | |
| 题目性质 | 应用研究 | | | **√** | | | 理论研究 | | |  | | |
| 题目  简述 | 区块链是分布式数据存储、点对点传输、共识机制、加密算法等计算机技术的新型应用模式。区块链技术作为21世纪的重大创新技术，已经得到了全世界范围内的认可。国家对区块链技术的发展极为重视，要求加快推动区块链技术和产业创新发展，积极推进区块链和经济社会融合发展，提高运用和管理区块链技术能力，使区块链技术在建设网络强国、发展数字经济、助力经济社会发展等方面发挥更大作用。为了保证数字虚拟资产在流通过程中的安全性、交易的不可抵赖性，设计一种基于区块链的数字虚拟资产交易系统具有一定实践价值。 | | | | | | | | | | | |
| 对学  生知  识与  能力  要求 | 1、有较好的数学、信息安全与密码学基础知识；  2、熟悉区块链技术的基础知识，对相关的算法有一定程度的了解；  3、熟练掌握一种以上的编程软件如Python、C等；  4、对新知识有较快的接受能力；  5、有一定的英文文献检索和阅读能力。 | | | | | | | | | | | |
| 具体  任务  以及  预期  目标 | 1、熟悉课题要求，能够利用网络资源检索区块链技术的相关资料，掌握其背景和研究现状。制定工作计划、完成开题报告。  2、熟悉区块链系统的基本原理和主要思想，熟练掌握基于区块链的交易系统的技术实现，选择合适的编程工具对其进行实现。  3、对设计的基于区块链的交易系统进行分析，评价其性能指标并探讨未来的发展前景以及对该技术领域发展的影响。  4、综合基于区块链的交易系统的基础知识、典型算法和分析结果撰写毕业论文，写作规范，并能够熟练阐述和讲解论文的主要内容。  5、掌握毕业设计和论文撰写流程，能够科学管理和周密安排每个环节的工作进度。 | | | | | | | | | | | |
| 时间  进度 | 3月6日至3月26日：学习区块链技术的基础知识，完成开题报告。  3月27日至4月16日：了解基于区块链的交易系统的基本思想，熟练掌握几种典型的基于区块链的交易系统。  4月17日至5月1日：编程实现基于区块链的交易系统，并分析其性能。  5月2日至5月20日：撰写论文，完成论文初稿。  5月21日至5月31日：修改并完善毕业论文，审阅验收论文成果。  6月1日至6月8日 ：答辩并完成后续工作。 | | | | | | | | | | | |
| 系（教研室）主任  签字 | | 年 月 日 | | | | 主管院长  签字 | | 年 月 日 | | | | |

西安邮电大学本科毕业设计（论文）开题报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 | 许冠行 | 学号 | 03173145 | 专业班级 | 安全1704 |
| 指导教师 | 罗维 | 题目 | 一个基于区块链的去中心化皮肤交易系统 | | |
| 选题目的（为什么选该课题）  随着虚拟物品的价值属性被社会大众的观念与各国的法律所认可，虚拟资产的安全变得越发地重要，传统的中心化几张方式存在着许多的安全隐患，数字虚拟资产再流通过程中的安全性，个人信息的保密性，交易的不可抵赖性没有得到有效的保护。而区块链与分布式数据库技术可以很好地解决这一问题。而现有的大部分区块链方案更多地专注于“代币”这一金融功能，忽视了区块链再版权，数字资产认定等领域的优势。本题旨在研究如何开发一套去中心化的皮肤交易系统。 | | | | | |
| 前期基础（已学课程、掌握的工具、资料积累、软硬件条件等）  已学课程：《密码学基础》《信息安全数学基础》《网络安全技术》《数据结构与算法》《密码学》  掌握的语言：c/c++,c+,rust,JavaScript,html,css,python,perl,solidity,go  资料：《Bitcoin：A Peer-to-peer ElectronicCash System》(Satoshi Nakamoto)  《区块链技术指南》 （张海宁）《solidity编程》（modi）  软件条件：pycharm, vscode, webstorm; | | | | | |
| 要研究和解决的问题（做什么）  研究产生用户名与密钥的ecc算法，研究用于验证区块完成行的哈希函数。设计区块数据结构与交易过程。搭建区块生成平台、前端web界面、web后端、p2p网络和整体的架构。设计共识机制的选取和记账权的分配方式。测试平台的性能，发现潜在的安全缺陷与性能问题。思考如何应对双花攻击、拜占庭缺陷、51%算力攻击等区块链的常见问题。因此，现有的区块链方案存在着一些问题与挑战，所以设计实现一个去中心化的皮肤交易方案，是一个需要解决的问题。 | | | | | |
| 工作思路和方案（怎么做）  第一阶段：熟悉学习区块链所需要的核心技术，了解全世界主流的区块链方案，研究国内外对于数字货币等区块链应用的政策，了解区块链相关技术的研究背景与意义思考平台的需求，确定整体的架构，选定相应的算法。重点学习《密码学基础》中关于ecc和hash的章节。学习所要使用到的语言与框架。  第二阶段：使用python编写第一版架构，编写非关系数据库，p2p，加密等模块。  第三阶段：使用flask后端，原生html前端完成GUI。  第四阶段：内网固定网段udp协议p2p测试。  第五阶段：重构代码，使用go，solidity语言在以太坊虚拟机上重现功能。  第六阶段：24小时测试，修改错误。  第七阶段：根据测试结果进行分析，评价各项性能指标与安全指标，并探讨该技术的未来前景与对该领域的影响。  第八阶段：完成论文初稿，向指导老师寻求意见，优化论文结构，润色语句，修改不当之处，补充不足之处，并能够熟练阐述和讲解论文的主要内容。  第九阶段：掌握毕业设计和论文编写流程，能够科学管理和周密安排每个环节工作进度，论文资料整合，修改并完善毕业论文，审阅验收论文结果。 | | | | | |
| 指导教师意见  签字 年 月 日 | | | | | |

西安邮电大学毕业设计 (论文)成绩评定表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学生姓名 |  | 性别 |  | | 学号 |  | | 专业  班级 | |  |
| 课题名称 |  | | | | | | | | | |
| 指导  教师  意见 | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 支撑指标点/赋分 | 2.1/30 | 5.1/20 | 10.1/30 | 11.2/20 | 合计 | | 得分 |  |  |  |  |  |   评分（百分制）： 指导教师**(**签字**)：** 年 月 日 | | | | | | | | | | | |
| 评阅  （验收）  意见 | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 支撑指标点/赋分 | 2.1/30 | 5.3/40 | 10.1/10 | 10.2/20 | 合计 | | 得分 |  |  |  |  |  |   评分（百分制）： 评阅教师(签字)： 年 月 日 | | | | | | | | | | | |
| 答辩  小组  意见 | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 支撑指标点/赋分 | 5.3/30 | 7.2/20 | 10.1/30 | 10.2/20 | 合计 | | 得分 |  |  |  |  |  |   评分（百分制）： 答辩小组组长(签字)： 年 月 日 | | | | | | | | | | |
| 评分比例 | 指导教师评分 20 (％) 评阅（验收）评分 40 (％) 答辩小组评分 40 (％) | | | | | | | | | |
| 学生总评  成绩 | 百分制成绩 | | |  | | | 等级制成绩 | |  | |
| 答辩委员会意见 | 毕业论文(设计)最终成绩(等级)**：**  学院答辩委员会主任**(**签字、学院盖章**)：** 年 月 日 | | | | | | | | | | |

摘 要

数字资产与版权的管理与认证在互联网时代变成了国家与企业重点关注的问题。传统的数字资产管理系统使用一个第三方的服务中心完成，而在这种中心化的系统中，交易与资产的透明性，不可抵赖性，维权等问题无法得到保证。而区块链这一数据结构的安全性，不可篡改，不可逆，信息公开，去中心化等特性可以很好的解决传统数字资产与版权管理系统中存在的问题。

本系统给出了一个区块链的区中心化的皮肤交易系统，运用区块链本身的安全性，不可篡改性等特点，去解决传统方案的问题。

本系统使用python语言开发，后端使用Flask架构，前端使用原生HTML，数据库使用Mongodb。实现了皮肤资产上传，交易，管理，溯源等功能。

关键词：皮肤交易；区块链；python；

# ABSTRACT

The management and authentication of digital assets and copyrights has become a key concern for the state and businesses in the Internet era. Traditional digital asset management systems use a third-party service centre, but in such a centralised system, the transparency, non-repudiation and rights of transactions and assets cannot be guaranteed. Blockchain, however, is a data structure that is secure, immutable, irreversible, open and decentralised, and can solve the problems of traditional digital asset and copyright management systems.

This system is a blockchain-based decentralised skin trading system that uses the security and tamper-evident features of the blockchain itself to solve the problems of traditional solutions.

The system is developed using python language, Flask architecture for the back-end, native HTML for the front-end, and Mongodb for the database, enabling the uploading, trading, management and traceability of skin assets.

**Key words：**skin trading；blockchain；python；

目录

第一章 绪论 …………………………………………………………1

1.1 选题原因 ……………………………………………………………… 1

1.2 研究内容 ……………………………………………………………… 2

1.3 本文结构 ……………………………………………………………… 2

第二章 技术与环境 ………………………………………………3

2.1 技术 ……………………………………………………………………… 3

2.2 环境 ……………………………………………………………………… 3

第三章 需求与可行性分析 …………………………………………4

3.1 创建账号 ……………………………………………………………… 4

3.2 登录账号 ……………………………………………………………… 4

3.3 皮肤上传 ……………………………………………………………… 5

3.4 皮肤交易 ……………………………………………………………… 5

3.5 交易历史 ……………………………………………………………… 5

3.6 搜索皮肤 ……………………………………………………………… 5

第四章 结构设计 …………………………………………………… 6

4.1 系统结构 ……………………………………………………………… 6

4.2 数据结构 ……………………………………………………………… 8

4.3 对象结构 ……………………………………………………………… 10

4.4 工具 …………………………………………………………………… 18

4.5 Web 设计 ……………………………………………………………… 24

第五章 安全与缺陷 ………………………………………………… 34

5.1 区块设计的安全与缺陷 ……………………………………………… 34

5.2 web设计的安全与缺陷 ……………………………………………… 35

结束语 *…………………………………………………………*……… 36

致谢 *…………………………………………………………………* 37

参考文献 *……………………………………………………*……… 38

附录 *…………………………………………………………*……… 39

第一章 绪论

1.1选题原因

游戏皮肤这一概念初始于1986年的初版《塞尔达传说》，这是世界上第一款能够自定义玩家的衣着与造型的游戏。在进入二十一世纪后，一些游戏厂家将皮肤作为DLC进行售卖，皮肤这一概念被作为商品开始在游戏历史上出现。在随后的网络技术与个人电脑的发展，能够连接互联网的网络游戏逐渐取代了单机游戏，游戏账号的概念开始出现，皮肤作为玩家的资产与玩家的账号所绑定。但是，玩家在购买皮肤之后只能使用，不能够再次出售，使得皮肤只能被玩家所租赁而不能完全地被拥有，在二十一世纪第二个十年，游戏厂家Value设计出了一个独特皮肤系统，玩家可以自由的购买别的玩家的皮肤，也可以出售自己的皮肤，让皮肤系统不再是单纯地向游戏公司租用，而是变成了一个自由市场。

传统的皮肤交易必须依赖于游戏公司提供的中心化服务，任何玩家与玩家之间的交易都相当于发送方先和游戏公司交易，再由游戏公司与接收方进行交易。这种不透明的和中心化的交易方式让玩家维权变得困难，也给了游戏公司恶意操纵皮肤市场，收取高额手续费以可乘之机。

但区块链技术的出现，为传统的中心化交易系统带来了发展的方向。中本聪于2008年写下了一篇名为《bitcoin ： a peer-to-peer electronic cash system》，首次提出了“区块链”这一概念，在2009年使用c++开发出了比特币系统，创造了第一个区块。在这篇论文中，中本聪描述了一个去中心化的交易系统。区块链源自比特币却又高于区块链技术。

区块链是比特币数字加密货币体系的核心技术，通过运用密码学，数据加密，时间戳，分布式共识系统，实现不需要信任的点对点去中心化交易，解决了传统中心化机构普遍存在的成本过高，低效，数据存储安全等问题。区块链是使用密码学原理而不是基于信任和权威的交易系统，玩家可以直接交易而不需要任何一个第三方，皮肤的交易由密码学原理保护，保障交易双方不被欺诈，保障用户拥有皮肤财产的合法性，保障交易不被撤销，不被修改。同时区块链记录了所有的交易，使得市场公开透明。

目前，基于区块链的数字资产交易的研究成果与应用依然是一片空白，本文将使用区块链技术完成一个皮肤交易系统。

1.2研究内容

传统的皮肤交易必须依赖于游戏公司提供的中心化服务，任何玩家与玩家之传统的皮肤交易系统已经使用发展了多年，是一项成熟的技术，但也有很多为问题没有解决。而区块链技术可以解决这些问题。本系统将研发出一套基于区块链的皮肤交易系统，实现皮肤上传登记，皮肤交易，历史查看，信息上链等功能。

1.3 本文结构

本文前两章主要讲了线上皮肤交易的现状与问题，描述了区块链技术的基本情况与本系统所需要的一些技术。第三章对本系统的需求与可行性进行分析。第四章则描述了实现系统地设计与解决方法。第五章分析了系统的安全性与存在的缺陷。

第二章 技术与环境

2.1技术

2.1.1 Flask

Flask是一个基于python的web微型框架，前端模板引擎使用Jinja2。是一个成熟稳定的轻量级可定制的框架。相较于同python的Django，Java的spring全家桶等更为灵活、轻便、安全且容易上手。可以很好地结合MVC模式进行开发。

2.1.2 Mongodb

Mongod是一个基于分布式文件的数据库，由c++编写，属于NoSql非关系数据库，所支持的结构使用类似json的bson格式，而本系统区块链的数据结构基于json，再使用python第三方bson库，便能实现将Mongod数据库抽象为区块链对象

2.1 环境

……….

……..

……..

……..

……..

……..

……..

……..

……..

系统环境：Windows10

语言开发环境：python3.9

数据库： MongoDB 4.2.0

IDE：PyCharm Community Edition 2020.3.1 Microsoft VS Code

第三章 需求与可行性分析

描述

本系统是一个基于区块链的皮肤交易系统，主要有登录模块，注册模块，皮肤上传模块，皮肤交易模块，能够在前端完成皮肤上传、交易、皮肤查看、查看历史记录等功能。

3.1 创建账号

3.1.1 需求

用户在主页点击注册，系统将生成一个密钥对，作为用户名与私钥。

3.1.2 可行性

后端使用ed25519库，生成一个ecc密钥对，这个密钥对是一个ed25519对象，将其格式化，使用base64编码，生成一个易于保存的字符串并返回前端进行显示。

3.2 登录账号

3.2.1 需求

用户点击登录，跳转到登录页面，用户输入自己的账户密码，点击登录，即可获取自己账号的全部权限。登录成功后跳转到用户主页。

3.2.2 可行性

后端再接收到前端的账号密码后，进行格式化，生成ed25519对象，再使用ed25519库的验证功能进行验证，验证通过后将密码存到后端的session中。后端对链进行解析，获取用户的资产列表历史记录等并跳转到用户主页。

3.3 皮肤上传

3.3.1 需求

用户点击选择文件，选择一张JPEG或JPG格式的图片，在下方的消息输入栏输入要上链的消息，点击上传。

3.3.2 可行性

前端先检查图片的格式，消息的字符类型，并传入后端。将上传的图片保存在temp文件夹，计算该文件的HASH,在coins文件夹新建以该文件HASH为文件夹名的文件夹。再将图片移动到该文件夹。通过文件HASH、时间戳、消息等创建新的区块，检查之后再上链。

3.4 皮肤交易

3.4.1 需求

用户选择一个自己拥有的皮肤，再消息栏输入交易消息，输入皮肤发送对象的地址，点击发送。

3.4.2 可行性

前端检查表单，通过文件HASH、时间戳、消息等创建新的区块，检查之后再上链。

3.5 搜索

3.5.1 需求

用户输入皮肤或用户名，查询相应的信息。

3.5.2 可行性

前端检查表单，传入后端后，根据长度判断查询的是用户名还是皮肤，重定向到相应的界面。

3.6 历史与皮肤

3.6.1 需求

获取区块链上所有的历史信息，包括创建与交易的信息。获取在链上所有上传过的皮肤。

3.7.2 可行性

后端同构区块链对象实现遍历整个链的功能，封装成接口后供前端使用。

第四章 结构设计

4.1 系统结构

本系统使用python语言flask框架。

config.py存放系统的设置，Genesis\_Block.json保存创世块的json，main.py为flask视图文件，src存放源代码，static存放前端静态文件，包括本地的皮肤图片文件与css文件，template存放前端模板html文件。

1. 2021/05/07  08:01    <DIR>          .
2. 2021/05/07  08:01    <DIR>          ..
3. 2021/05/08  04:36    <DIR>          .idea
4. 2021/05/07  00:14               427 config.py
5. 2021/05/07  04:46               601 Genesis\_Block.json
6. 2021/05/07  05:27                96 log.log
7. 2021/05/07  05:31             8,641 main.py
8. 2021/04/17  22:53                11 README.md
9. 2021/05/07  04:54    <DIR>          src
10. 2021/05/07  00:14    <DIR>          static
11. 2021/05/07  05:31    <DIR>          templates
12. 2021/04/22  12:28    <DIR>          test
13. 2021/05/07  03:46               170 test.py
14. 2021/05/07  04:32               543 user.json
15. 2021/05/07  02:22    <DIR>          \_\_pycache\_\_
16. 9 个文件      2,056,371 字节
17. 8 个目录 18,184,835,072 可用字节

在src文件夹中，mvc.py存放flask后端相应的功能代码，object文件夹存放该系统的所有对象，包括用户对象（account.py），区块对象（block.py），区块链对象（blockchain.py)。

在bin文件夹中存放该系统所需要的工具，包括加密相关，日志相关，数据库相关，网络相关，OS相关。

1. C:.
2. │  mvc.py
3. │  user.json
4. │  \_\_init\_\_.py
5. │
6. ├─bin
7. │  │  encrypt.py
8. │  │  log.py
9. │  │  mongodb.py
10. │  │  net.py
11. │  │  os.py
12. │  │  \_\_init\_\_.py
13. │
14. ├─object
15. │  │  account.py
16. │  │  block.py
17. │  │  blockchain.py
18. │  │  \_\_init\_\_.py

templates存放前端模板，分别对应登录界面，首页，交易界面，上传界面，用户主页等界面等。

1. C:.
2. base.html
3. coin.html
4. coins.html
5. create.html
6. history.html
7. login.html
8. root.html
9. search.html
10. showcoins.html
11. showtrans.html
12. succesetran.html
13. succeseupload.html
14. test.html
15. tran.html
16. upload.html
17. user.html
18. users.html

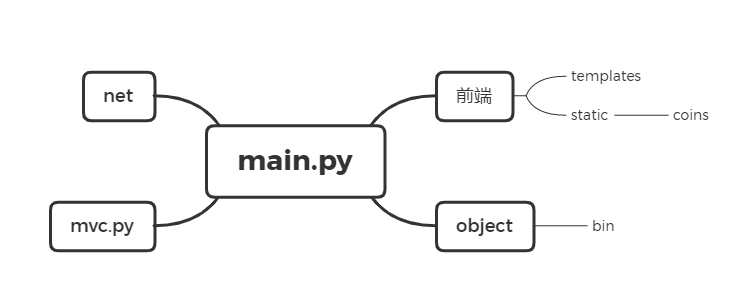


图4.1 系统架构

4.2 数据结构

4.2.1 区块

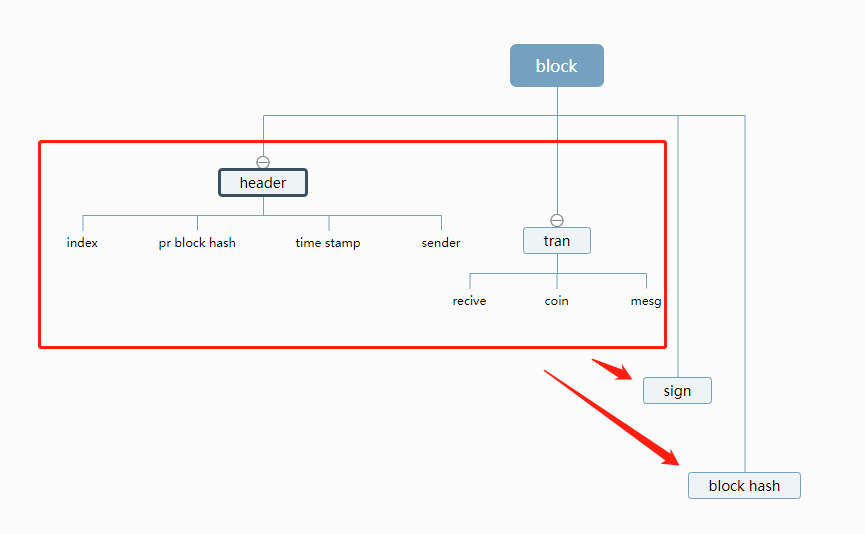


图4.2 区块结构

1. {
2. "header": {
3. "index": 0,
4. "pr\_block\_hash": "0000000000000000000000000000000000000000000000000000000000000000",
5. "timestamp": "1619191723.2759726",
6. "sender": "eb1d2b43aad8cf60ff9911a894f80ec4f4befccc6c61901d2cc72aac1b1d89d2"
7. },
8. "tran": {
9. "recive": "eb1d2b43aad8cf60ff9911a894f80ec4f4befccc6c61901d2cc72aac1b1d89d2",
10. "coin": "1f0e6968fc800be658bbb21ef27c18af",
11. "mesg": "Genesis\_Block"
12. },
13. "sign": "kFC7si5HDshII9m44G7ucfgrsfy0yqD1Qab42+DpO0Qp90Vs1R3/rEtVqQEW864LzMOkCFqBHAJrDtThRn5nBQ",
14. "block\_hash": "c5bf4a85dc0849b09e47bf0de2de5b5c1dad8c58eaa948e583ecdb38a125be44"
15. }

区块中含有三个部分，header是区块头，index表示区块的高度、索引。pr\_block\_hash,为上一个区块的HASH值，timestamp为创建区块时的时间戳。Sender为发送者的地址。

Tran是交易内容，recive为皮肤接受者的地址，coin为皮肤文件HASH值，mesg为额外的消息。

Sign是区块的签名，使用header和tran的字符串和用户私钥进行签名。

1. self.sign\_value = sign(self.header.\_\_str\_\_() + self.tran.\_\_str\_\_(), sender\_key)

Block\_hash为该区块的HASH值，使用header和tran的字符串，用sha256进行运算。

4.3 对象结构

4.3.1 数据库对象

数据库使用mongodb，非关系型文件数据库，其中存储文件的数据结构为bson，类似与区块链对象的json，使用第三方库json\_util将json转为bson。

设计了插入块，删除块，清空数据库，获取链的长度，获取特定块等方法。

1. import json
2. import pymongo
3. from bson import json\_util

区块链存放在mongodb的skin\_chain库chain表中，若不存在该表则创建。数据库再本地的27017端口，若数据库不在本机，则替换localhost为相应的IP地址。

1. try:
2. myclient = pymongo.MongoClient("mongodb://localhost:27017/")
3. dblist = myclient.list\_database\_names()
4. if "skin\_chain" not in dblist:
5. Log().info("数据库不存在，正在创建--------")
6. print("数据库不存在，正在创建--------")
7. mydb = myclient["skin\_chain"]
8. Log().info("已创建skin\_chain数据库")
9. print("已创建skin\_chain数据库")
10. else:
11. mydb = myclient["skin\_chain"]
12. self.mycol = mydb["chain"]
13. except Exception as e:
14. print("数据库创建失败：" + str(e))
15. Log().error("数据库创建失败：" + str(e))

对数据库对象进行封装，创建相应的方法，包括：插入，删除，清空，获取最新记录，获取集合数量，根据用户地址或皮肤HASH查询等。

4.3.2 区块链

区块链对象继承与数据库对象。

除了继承自数据库的方法，新定义了几个针对区块链的方法。

通过用户名获取该用户的历史记录，使用父对象中的get\_block\_list\_by\_user方法，获取所有含有该用户的区块列表，遍历这个列表，进行分类，转换时间戳。返回该用户的操作类型，时间，发送方或接受方的地址，皮肤的HASH值及在区块中的消息。

1. def get\_user\_history(self, user):
2. block\_list = self.get\_block\_list\_by\_user(user)
3. if not block\_list:
4. return False
5. # print(block\_list)
6. user\_history\_list = []
7. for block in block\_list:
8. if block["header"]["sender"] == user and block["tran"]["recive"] != user:
9. user\_history\_list.append({
10. "opt": "send",
11. "timestamp": int(float(block["header"]["timestamp"])),
12. "recive": block["tran"]["recive"],
13. "coin": block["tran"]["coin"],
14. "mesg": block["tran"]["mesg"]
15. })
16. elif block["tran"]["recive"] == user and block["header"]["sender"] != user:
17. user\_history\_list.append({
18. "opt": "recive",
19. "timestamp": int(float(block["header"]["timestamp"])),
20. "sender": block["header"]["sender"],
21. "coin": block["tran"]["coin"],
22. "mesg": block["tran"]["mesg"]
23. })
24. elif block["tran"]["recive"] == user and block["header"]["sender"] == user:
25. user\_history\_list.append({
26. "opt": "create",
27. "timestamp": int(float(block["header"]["timestamp"])),
28. "creater": block["header"]["sender"],
29. "coin": block["tran"]["coin"],
30. "mesg": block["tran"]["mesg"]
31. })
32. user\_history\_list = sorted(user\_history\_list, key=lambda i: i['timestamp'])
33. for index, coin in enumerate(user\_history\_list):
34. user\_history\_list[index]["timestamp"] = time.asctime(time.localtime(int(float(coin["timestamp"]))))
35. return user\_history\_list

4.3.3 用户对象

新建用户的密钥对，使用第三方库ed25519，分别创建私钥和公钥对象，再使用对象的to\_ascii方法，获取方便存放和传输的字符串。

将生成的密钥对进行裁切，做成方面写入区块的字符串。

1. import ed25519
2. class Account(object):
3. """用户对象"""
4. def \_\_init\_\_(self):
5. self.signing\_key, self.verifying\_key = ed25519.create\_keypair()
6. self.PublicKey = self.verifying\_key.to\_ascii(encoding="hex")
7. self.PrivateKey = self.signing\_key.to\_ascii(encoding="hex")
8. self.SigningKey = self.signing\_key
9. self.VerifiyingKey = self.verifying\_key
10. self.Username = str(self.PublicKey)[2:-1]
11. self.Password = str(self.PrivateKey)[2:-1]
12. def create\_account(self):
13. return self.Username, self.Password

用户对象继承自BlockChain对象。除了继承自父对象的方法外，AccountOpertion定义了：获取用户的所有皮肤，获取用户所拥有皮肤的数量，查看用户交易记录，查看用户所有上传的皮肤，交易，上传等方法。

获取一个用户的所有资产，需要先取得该用户的历史，从头开始遍历，查看每笔交易，最终生成一个字典。该字典的键为该用户曾拥有的所有皮肤，键值为1的表示该用户正拥有这个皮肤，为0代表该用户曾经拥有这个皮肤，但已经被交易出去。最后统计键值为1的键，即可获取该用户的资产列表。

获取一个用户曾经上传的所有皮肤，先获取该用户的交易历史，查看获取历史列表中的字段含有creater的记录。以同样的方法，统计键值为1的键，获取该用户的上传列表。

4.4 工具

4.4.1加密与散列

对字符串取HASH，使用第三方库hashlib，散列算法使用sha256。

1. def hash\_sha256(string):
2. """哈希"""
3. sha = hashlib.sha256()
4. sha.update(string)
5. return sha.hexdigest()

对文件取HASH， 使用md5算法。设置一次读取的长度为1024byte

1. def file\_hash(file\_path):
2. """对文件取哈希值"""
3. byte = 1024
4. md5 = hashlib.md5()
5. with open(file\_path, 'rb') as f:
6. while True:
7. data = f.read(byte)
8. if data:
9. md5.update(data)
10. else:
11. break
12. return md5.hexdigest()

验证区块的HASH，做完整性验证。

1. def check\_hash(string, hash\_value):
2. """验证哈希值"""
3. sha = hashlib.sha256()
4. sha.update(string)
5. return hash\_value == sha.hexdigest()

验证区块的签名：签名与需要验签的内容为utf-8的字符串，先进行格式转化，转成字节流，再使用hex编码，转成能被ed25519对象识别的格式。

1. def check\_sign(string, sig, vkey\_hex):
2. """验证签名"""
3. sig = bytes(sig, encoding="utf8")
4. vkey\_hex = bytes(vkey\_hex, encoding="utf8")
5. verifying\_key = ed25519.VerifyingKey(vkey\_hex, encoding="hex")
6. try:
7. verifying\_key.verify(sig, bytes(string, encoding='UTF-8'), encoding="base64")
8. result = True
9. except ed25519.BadSignatureError:
10. result = False
11. return result

检查用户的账户密码是否为密钥对。由椭圆曲线密码的特性，私钥可以生成相应的公钥，但公钥生成不了私钥，检查由私钥生成的公钥是否与用户地址相对应，即可验证账户密码是否正确。

1. def check\_password(username, password):
2. """检查账户密码是否正确，返回bool"""
3. sig = bytes(password, encoding="utf8")
4. vkey\_hex = bytes(username, encoding="utf8")
5. signature\_key = ed25519.SigningKey(sig, encoding="hex")
6. verifying\_key = ed25519.VerifyingKey(vkey\_hex, encoding="hex")
7. return verifying\_key == signature\_key.get\_verifying\_key()

4.4.1文件

移动文件，用于上传皮肤文件。

1. def move\_file(srcfile, dstpath):
2. """复制文件到指定文件夹"""
3. try:
4. if not os.path.exists(dstpath):
5. os.mkdir(dstpath)
6. if os.path.exists(srcfile):
7. return True
8. shutil.move(srcfile, dstpath)
9. return True
10. except Exception as e:
11. Log().error("文件不存在：" + str(e))
12. print("文件不存在：" + str(e))
13. return False

获取图片的文件流

1. def return\_img\_stream(img\_local\_path):
2. """获取图片文件流"""
3. img\_stream = ''
4. with open(img\_local\_path, 'rb') as img\_f:
5. img\_stream = img\_f.read()
6. img\_stream = base64.b64encode(img\_stream)
7. return img\_stream

4.4.2设置

指定端口、网段、皮肤文件存储文件夹、最大文件大小及flask的各种设置等。存放在该工程的根目录的config.py中，创建Config类。

1. class Config:
2. PORT = 9999
3. SHACKPORT = 9998
4. SYNCPORT = 9996
5. FILEPORT = 9997
6. IPPOOL = "192.168.0.255"
7. SECRET\_KEY = "SECRETKEY"
8. UPLOAD\_FOLDER = "C:\\Users\\Administrator.DESKTOP-35V3OQH\\SkinChain\\static\\images\\coins\\"
9. MAX\_CONTENT\_LENGTH = 1024 \* 1024 \* 1024

4.4.3 检查区块

检查通过HASH值检查区块的完整性，通过发送者的地址验证签名，检查上一区块的HASH是否正确。

验证区块分为这几步：

1. 验证签名，获取块里面的签名值与，被签名的内容。
2. 验证区块的哈希值是否正确。
3. 检查区块的索引值是否符合可以入库的标准。
4. 检查区块的pr\_block\_hash的值是否等于上一个区块的哈希值。
5. 检测交易发起者是否拥有这个皮肤。
6. def check\_block(block):
7. """检查收到的块"""
8. result = True
9. try:
10. string = block["header"].\_\_str\_\_() + block["trans\_list"].\_\_str\_\_()
11. sig = block["sign"]
12. user\_name = block["header"]["sender"]
13. hash\_value = block["block\_hash"]
14. if not check\_sign(string, sig, user\_name):
15. result = False
16. if not check\_hash(string, hash\_value):
17. result = False
18. block\_index = int(block["header"]["index"])
19. db\_top\_index = BlockChain().get\_top\_block\_index()
20. if block\_index != db\_top\_index + 1:
21. result = False
22. if block["header"]["pr\_block\_hash"] != BlockChain().get\_top\_block\_hash():
23. result = False
24. sender = block["header"]["sender"]
25. coin = block["tran"]["coin"]
26. sender\_obj = AccountOpertion(sender)
27. coin\_list = sender\_obj.show\_coins()
28. if coin in coin\_list:
29. return False
30. except Exception as e:
31. Log().error("检查区块合法性失败：" + str(e))
32. result = False
33. finally:
34. return result

4.4.4 网络

读取config文件中设置的ip池，获取该ip池里上线的主机。

1. def \_\_ping(self, ip):
2. cmd = "ping -n 1 " + ip + '|findstr TTL'
3. res = os.popen(cmd)
4. if res.readlines():
5. self.ip\_alive\_list.append(ip)
6. 先对网段中所有ip地址执行ping命令，最初使用for循环进行轮询，但是速度过慢，所以使用多线程加快轮询速度。
7. for ip in self.ip\_pool:
8. thread\_list.append(threading.Thread(target=self.\_\_ping, args=(ip,)))
9. for thread in thread\_list:
10. thread.start()
11. for thread in thread\_list:
12. thread.join()

2.查找网段中的节点：先查找网段中在线的主机，再检查在线的主机上是否存在相应的数据库与表，通过操作网段中存活的数据库完成同步。

4.5 web设计

4.5.1 主页

介绍平台的基本信息，给出进行各种操作的连接



图4.4 系统首页

所有前端模板继承于base.html。页面的头给了首页、登录、注册、交易、上传、搜索等页面的入口。给了该项目的仓库地址。

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <title>SkinChain</title>
6. <script src="https://cdn.static.runoob.com/libs/jquery/1.10.2/jquery.min.js"></script>
7. <link rel=stylesheet type="text/css" href="{{ url\_for('static', filename='css/base.css')}}">
8. </head>
9. <body>
10. <div class='pg-header'>
11. <a href="/">SkinChain</a>
12. <a href="/">首页</a>
13. <a href="/login/">登录</a>
14. <a href="/create/">注册</a>
15. <a href="/user/">用户中心</a>
16. <a href="/tran/">交易</a>
17. <a href="/upload/">上传</a>
18. </div>
19. {% block content %}
20. {% endblock %}
21. <div  class='pg-footer'></div>
22. </body>
23. </html>

4.5.2 登录



图4.5 登录表单

检查表单，用户名与密码应为64位的数字与小写字母。

1. function check\_form(){
2. var username = document.getElementById("username").value;
3. var password = document.getElementById("password").value;
4. if (username.length == 0){
5. alert("请输入用户名");
6. return false;
7. }
8. if(password.length == 0){
9. alert("请输入密码");
10. return false;
11. }
12. if (username.length != 64){
13. alert("用户名长度应为64");
14. return false;
15. }
16. if (password.length != 64){
17. alert("密码长度应为64");
18. return false;
19. }
20. reg\_str = "^[a-z0-9]+$";
21. var reg = new RegExp(reg\_str);
22. if (!reg.test(username)){
23. alert("用户名应为数字与小写字母");
24. return false;
25. }
26. if (!reg.test(password)){
27. alert("密码应为数字与小写字母");
28. return false;
29. }
30. }

检查后端session是否已经有账户与密码，若没有，则使用check\_password（）函数检查密码是否正确。若正确，则将账户密码存入session，并将前端重定向到用户界面。

1. @app.route('/login/', methods=['POST', 'GET'])
2. def login():
3. status = "请输入账户密码"
4. if session.get("username") and session.get("password"):
5. return redirect('/user/' + session.get("username") + '/')
6. if request.method == 'POST':
7. username = request.form.get("username")
8. password = request.form.get("password")
9. if check\_password(username, password):
10. session["username"] = username
11. session["password"] = password
12. return redirect('/user/' + username + '/')
13. else:
14. status = "账户/密码错误"
15. return render\_template('login.html',
16. status=status)

4.5.3 退出登录

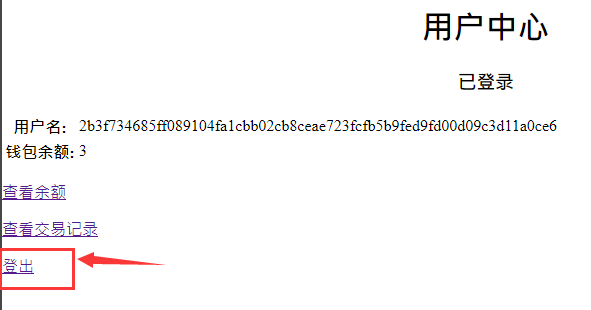


图4.6 用户页面

点击登出后，后端清除session，并重定向到登录界面

1. @app.route('/logout/', methods=['POST', 'GET'])
2. def logout():
3. session.clear()
4. return redirect('/login/')

4.5.3 创建新用户



图4.7 注册用户

生成密钥对作为账户与密码并返回前端。传统web应用用户的注册需要将用户数据存在后端数据库中。但由于区块链系统的特殊性，用户的注册可以完全不需要中心机构完成，只需要根据协议的算法生成相应的密钥对，再进行相应的格式化与裁剪后即可，在没有开始交易时，也不需要上链。

1. @app.route('/create/', methods=['POST', 'GET'])
2. def create():
3. username, password = Account().create\_account()
4. return render\_template('create.html',
5. username=username,
6. password=password)



图4.8 新账户生成页面

4.5.4 交易

从后台链中获取该用户的所有皮肤，返回给前端。用户选择一个皮肤，输入要发送的用户和额外的消息，点击发送，在前端检查之后将表单传给后端，后端使用用户对象的send\_coin（）方法发送皮肤。若发送成功。重定向到成功页面。

图4.9 交易界面



图4.10 交易成功界面

1. @app.route('/user/<user>/tran/', methods=['POST', 'GET'])
2. def tran(user):
3. user\_obj = AccountOpertion(user)
4. coin\_list = user\_obj.show\_coins()
5. pic\_path\_list = []
6. for coin in coin\_list:
7. pic\_path\_list.append({
8. "coin": coin,
9. "path": get\_pic\_path(coin)
10. })
11. if request.method == 'POST':
12. obj\_username = request.form.get("obj\_username")
13. coin = request.form.get("coin")
14. message = request.form.get("message")
15. index = user\_obj.send\_coin(recive=obj\_username, coin=coin, mesg=message, sender\_key=session.get("password"))
16. return redirect('/user/' + user + "/tran/succese/" + index + "/")
17. return render\_template("tran.html",
18. pic\_path\_list=pic\_path\_list)

4.5.5 上传皮肤



图4.11

选择本地的皮肤文件上传。前端将文件与额外的信息传给后端，后端进行相应的检查后上链。



图4.12

1. @app.route('/uploader/', methods=['POST', 'GET'])
2. def uploader():
3. username = session.get("username")
4. img = request.files.get("photo")
5. mesg = request.form.get("mesg")
6. file\_path = Config.UPLOAD\_FOLDER + "temp\\" + img.filename
7. img.save(file\_path)
8. file\_hash\_value = file\_hash(file\_path)
9. dst\_path = Config.UPLOAD\_FOLDER + file\_hash\_value + "\\"
10. dst\_file = dst\_path + img.filename
11. if move\_file(file\_path, dst\_path):
12. # send\_file(dst\_file)
13. user\_obj = AccountOpertion(username)
14. index = user\_obj.create\_coin(coin=file\_hash\_value, mesg=mesg, sender\_key=session.get("password"))
15. return redirect("/uploader/" + str(index) + "/")

4.5.6 搜索



图4.13

在输入栏中输入皮肤或地址，将这个字符串传到后端。后端对字符串长度进行判断，如果长度为64，则为地址，若长度为32，则为皮肤。分别重定向到皮肤或用户的界面。

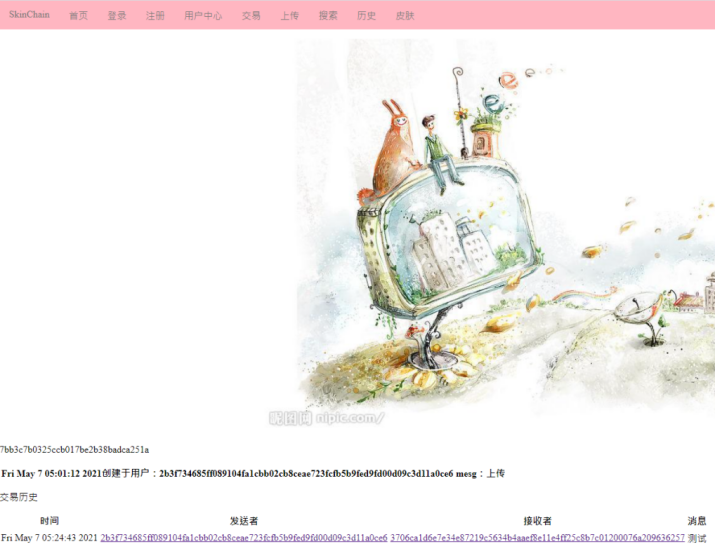


图4.14

1. @app.route('/search/', methods=['POST', 'GET'])
2. def search():
3. if request.method == 'POST':
4. username = request.form.get("username")
5. if len(username) == 64:
6. return redirect("/users/" + username + "/")
7. elif len(username) == 32:
8. return redirect("/coin/" + username + "/")
9. return render\_template("search.html")

4.5.7 皮肤页面

皮肤页面显示皮肤图片、文件HASH、上传时的信息与交易记录。

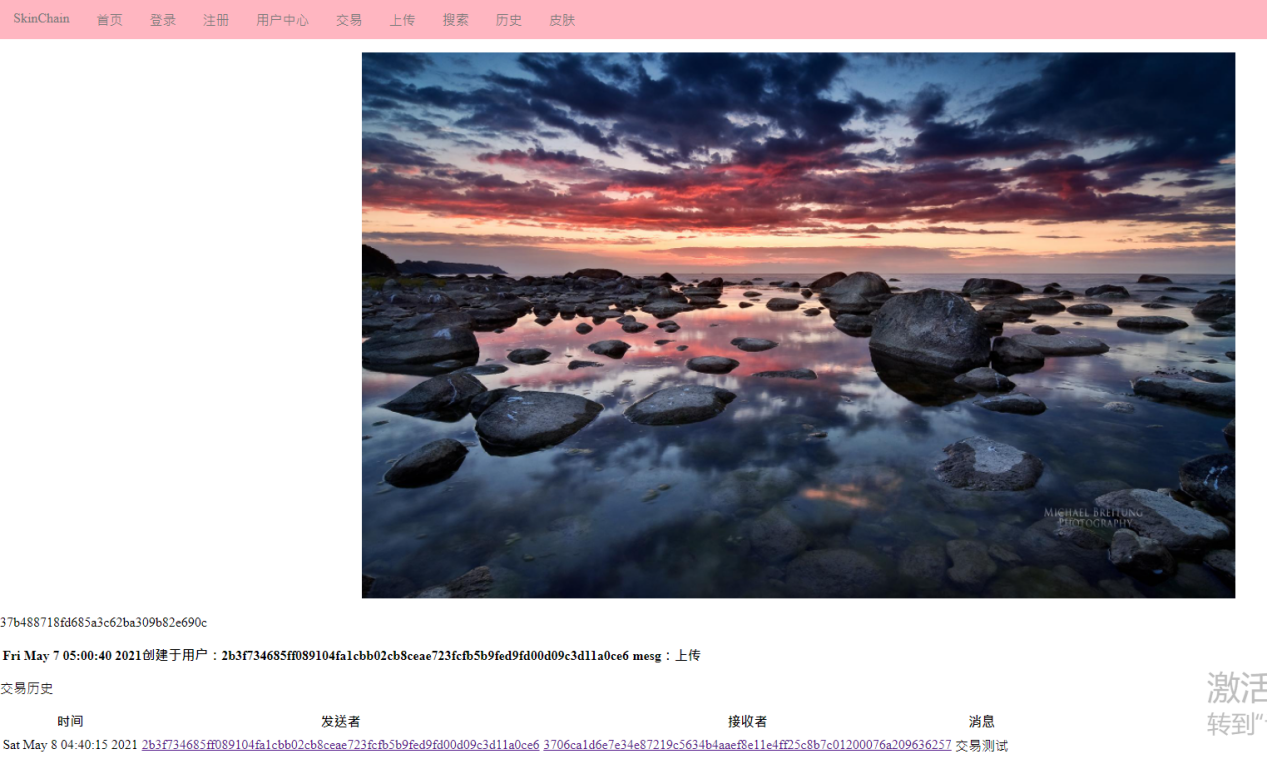


图4.15

1. @app.route('/coin/<coin>/', methods=['POST', 'GET'])
2. def coin(coin):
3. create, coin\_history = BlockChain().get\_coin\_history(coin)
4. img\_url = "/display/img/" + coin
5. return render\_template("coin.html",
6. coin\_history=coin\_history,
7. img\_url=img\_url,
8. coin=coin,
9. create=create)

4.5.7 用户页面

显示用户地址，余额，提供跳转到该用户余额和交易记录的链接。由于区块链上的所有交易与内容均可见，则每个地址的用户界面对所有人可见。

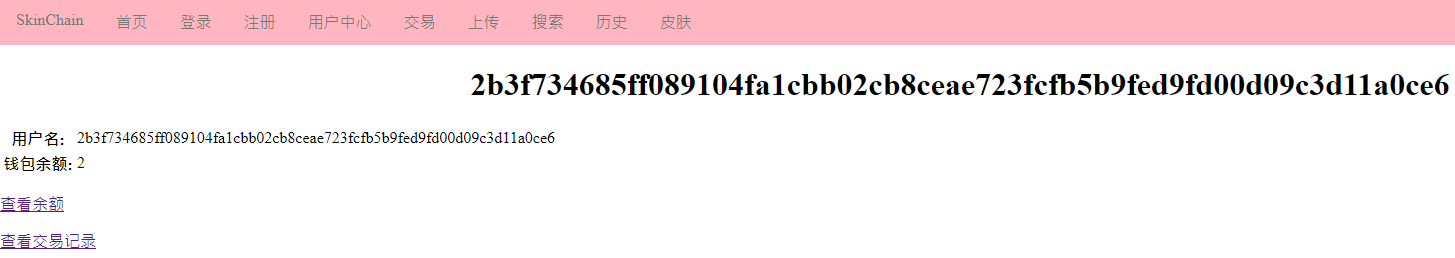


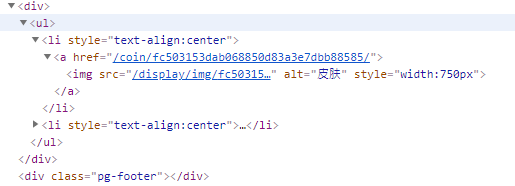
图4.16

1. @app.route('/users/<user>/', methods=['POST', 'GET'])
2. def users(user):
3. user\_obj = AccountOpertion(user)
4. username = user
5. coin\_count = user\_obj.show\_coin\_count()
6. return render\_template('users.html',
7. username=username,
8. coin\_count=coin\_count)

4.5.8 显示皮肤图片

通过GET中的URL参数得到皮肤的HASH，在本地进行查找相应文件，获取文件流并返回前端。

1. @app.route('/display/img/<coin>', methods=['POST', 'GET'])
2. def display\_img(coin):
3. if coin is None:
4. pass
5. else:
6. image\_data = open(get\_pic\_path(coin), "rb").read()
7. response = make\_response(image\_data)
8. response.headers['Content-Type'] = 'image/jpg'
9. return response

图4.17

第五章 安全与缺陷

5.1区块设计的安全与缺陷

整个区块这一数据结构采用json格式，其中含有四个部分，分别是header、tran、sign、block\_hash。其中header中存储了该区块所在的高度index，采用int数据类型，迁移个区块的哈希值，生成区块时的时间戳，和发送者的地址。与传统的比特币、以太坊、狗狗币等挖矿货币不同，本系统是谁交易，谁公布块，所以发送者地址只存在交易头中，而接收者地址存放再交易中，交易部分还存储着交易皮肤的哈希值与一段额外的信息。签名部分存放着用发送者的私钥头和交易的签名。最后保存着整个区块的哈希值。

区块中的签名能够证明该区块对发送者的合法性，保证该笔交易由发送者发起，只要使用区块中的头、交易与头中的发送者的地址进行验签。若有第三方想要篡改交易，则会导致签名无法通过验证。

若发送者自己想要篡改区块内容，则会改变区块的哈希值，而下一个区块的区块头中存储着该哈希值，若前一区块哈希值发生变化，则会使得区块链断开。若要让区块链重新连上，则必须修改下一区块的头哈希值、签名与哈希值，则又会导致下一区块断裂，则攻击者必须修改之后全部区块。若之后的区块存在不是使用自己地址的，则无法修改区块签名及内容以保证区块链不断开。

每个想要交易的用户打包区块，并广播到全网，全网会对这个新广播的区块进行验证，验证这个块是否比在该节点最大高度的块还要大，验证签名与上一区块的哈希值，验证发送者是否拥有这一资产，验证通过后便加入区块链。再全网中的每个节点拥有共识，节点总是认可拥有最长长度的链。而每次发布区块不一定要公布到全部节点，如果某一节点收到的区块的高度比节点最高高度值相差超过1，则会在全网的节点中进行请求，获取全网最长的链与拥有该链的节点的ip地址，并请求进行节点同步。

每个节点只会选择最长的分支进行保存，若攻击者使用双花攻击，则区块链会根据时间戳而保留最早的交易，舍弃掉其他的交易。对于因为网络原因没有接收到正确区块的节点，当这笔交易的6个区块后，这笔交易基本能够确定。

本系统所有设计细节与区块的传输均为开源与明文传输，但并没有真正做到零信任。因为没有采用pow或pos进行工作量或权力证明，使得伪造区块变得相对容易，增加区块交易被篡改地风险。

比特币的区块数据结构使用Merkel根精简区块结构，目前整个比特币的所有区块大小已超过100GB，而使用Merkel根精简过的轻钱包可以存放在手机等移动设备中。本系统每一笔交易对应一个区块，计算余额的方式需要遍历整个数据库，在交易数量变大时造成性能问题。

5.2 web设计的安全与缺陷

5.2.1 接口与路由保护

本系统使用flask web架构，没有使用如nigix等web服务器，只作为一个GUI使用。若flask中的接口暴露到了内网或公网，会带来敏感信息泄露等问题。

解决方法：

1. 使用flask自带的装饰器，禁用掉除了本机以外的所有ip的请求。
2. from flask import abort, request
3. @app.before\_request
4. def limit\_remote\_addr():
5. if request.remote\_addr != '192.168.0.1':
6. abort(403)  *# Forbidden*
7. 使用防火墙进行包过滤，将所有到达端口5000的通信丢弃，本机ip除外
8. /sbin/iptables -A INPUT -s ! 192.168.0.1 --dport 80 -j DROP

5.2.2 敏感文件隐藏

对于本机中存放的密钥文件，设置文件等做好隐藏与加密。可以使用trueencrypt对敏感文件进行加密。

防火墙使用白名单模式，仅保留几个必须使用的端口。关闭掉3389远程桌面， 22ssh，21FTP，80http，443ssl，445打印机等危险端口。

5.2.3 输入输出漏洞

在前端提交表单时进行检测，使用白名单模式，只允许字母汉字数字等写入区块，防止如引号等特殊字符进行注入破坏

结束语

本文设计了一个基于区块链的皮肤交易系统。他抛弃了传统的中心化交易方案，让交易处于完全对等点对点的交易双方，使用基于共识的模型，代替了原先的基于第三方信任的交易方案。每笔交易由发送者发起，并向全网公布区块，该区块最大限度地公布到全网。每个节点重新加入网络时，会自动接受最长的合法链最为该节点离线时全网发生一切交易的证明。由于HASH和数字签名的存在，攻击者若要篡改交易记录则要修改全网大部分的节点的记录，并修改需要篡改交易记录后的所有区块。

结合python、flask、web前端等技术，实现了用户只需要点击鼠标就可以完成该系统的所有功能包括皮肤上传，交易，查看记录等。通过区块链技术，实现了交易记录存放在分布式数据库中，使用特殊的算法，可以通过区块链查询到交易记录，余额等信息。

致 谢

参考文献

[1] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. Whitepaper, 2009.

[2] A. Back，Hashcash - a denial of service counter-measure, 2002.

附录

1. class Block(object):
2. """区块对象"""
3. def \_\_init\_\_(self, index, sender, sender\_key, recive, pr\_block\_hash, coin, mesg):
4. self.header = {
5. "index": index,
6. "pr\_block\_hash": pr\_block\_hash,
7. "timestamp": str(time.time()),
8. "sender": sender
9. }
10. self.tran = {
11. "recive": recive,
12. "coin": coin,
13. "mesg": mesg
14. }
15. self.sign\_value = sign(self.header.\_\_str\_\_() + self.tran.\_\_str\_\_(), sender\_key)
16. self.block\_hash = hash\_sha256((self.header.\_\_str\_\_() + self.tran.\_\_str\_\_() + self.sign\_value).encode("utf8"))
17. self.block = {
18. "header": self.header,
19. "tran": self.tran,
20. "sign": self.sign\_value,
21. "block\_hash": self.block\_hash
22. }
23. self.block\_dict = json.loads(self.block.\_\_str\_\_().replace("'", '"'))
24. def print\_block(self):
25. print(self.block)
26. def add\_tran(self, trans\_list):
27. self.block["trans\_list"].append(trans\_list)
28. class Db(object):
30. def \_\_init\_\_(self):
31. try:
32. myclient = pymongo.MongoClient("mongodb://localhost:27017/")
33. dblist = myclient.list\_database\_names()
34. if "skin\_chain" not in dblist:
35. Log().info("数据库不存在，正在创建--------")
36. print("数据库不存在，正在创建--------")
37. mydb = myclient["skin\_chain"]
38. Log().info("已创建skin\_chain数据库")
39. print("已创建skin\_chain数据库")
40. else:
41. mydb = myclient["skin\_chain"]
42. self.mycol = mydb["chain"]
43. except Exception as e:
44. print("数据库创建失败：" + str(e))
45. Log().error("数据库创建失败：" + str(e))
46. def test(self, user):
47. block\_list = json\_util.dumps(self.mycol.find({"header.sender": user}, {"\_id": False}))
48. return block\_list
49. def insert(self, block):
50. """在数据库插入新的块"""
51. try:
52. self.mycol.insert\_one(block.block\_dict)
53. return True
54. except Exception as e:
55. Log().error("区块插入失败：" + str(e))
56. print(e)
57. return False
58. def insert\_dict(self, block):
59. self.mycol.insert\_one(block)
60. def delete(self, index):
61. """根据索引删除块"""
62. try:
63. self.mycol.delete\_one({"header.index": index})
64. except Exception as e:
65. Log().error("删除失败：" + str(e))
66. def clean(self):
67. """清空数据库"""
68. try:
69. self.mycol.delete\_many({})
70. except Exception as e:
71. Log().error("清空数据库失败：" + str(e))
72. def get\_top\_block(self):
73. """获取最新的块"""
74. block = self.mycol.find({}, {"\_id": False}).sort("\_id", -1).limit(1)
75. return json.loads(json\_util.dumps(block))[0]
76. def get\_chain\_len(self):
77. """获取链的长度"""
78. return self.mycol.count()
80. def get\_block\_by\_index(self, index):
81. """通过索引获取最新的块"""
82. try:
83. return json.loads(json\_util.dumps(self.mycol.find({"header.index": index}, {"\_id": False})))[0]
84. except Exception as e:
85. Log().error("获取最新块失败：" + str(e))
86. def get\_block\_list\_by\_user(self, user):
87. """获取含有某一用户的所有块"""
88. try:
89. block\_list\_raw = json.loads(json\_util.dumps(self.mycol.find({"header.sender": user}, {"\_id": False}))) \
90. + json.loads(json\_util.dumps(self.mycol.find({"tran.recive": user}, {"\_id": False})))
91. block\_list = []
92. for item in block\_list\_raw:
93. if item not in block\_list:
94. block\_list.append(item)
95. return block\_list
96. except Exception as e:
97. Log().error("通过用户获取块失败：" + str(e))
98. def get\_block\_list\_by\_coin(self, coin):
99. """获取含有某一资产的所有块"""
100. try:
101. return json.loads(json\_util.dumps(self.mycol.find({"tran.coin": coin}, {"\_id": False})))
102. except Exception as e:
103. Log().error("通过资产获取块失败：" + str(e))
104. class BlockChain(Db):
105. def insert\_block(self, block):
106. return self.insert(block)
107. def get\_user\_history(self, user):
108. block\_list = self.get\_block\_list\_by\_user(user)
109. if not block\_list:
110. return False
111. *# print(block\_list)*
112. user\_history\_list = []
113. for block in block\_list:
114. if block["header"]["sender"] == user and block["tran"]["recive"] != user:
115. user\_history\_list.append({
116. "opt": "send",
117. "timestamp": int(float(block["header"]["timestamp"])),
118. "recive": block["tran"]["recive"],
119. "coin": block["tran"]["coin"],
120. "mesg": block["tran"]["mesg"]
121. })
122. elif block["tran"]["recive"] == user and block["header"]["sender"] != user:
123. user\_history\_list.append({
124. "opt": "recive",
125. "timestamp": int(float(block["header"]["timestamp"])),
126. "sender": block["header"]["sender"],
127. "coin": block["tran"]["coin"],
128. "mesg": block["tran"]["mesg"]
129. })
130. elif block["tran"]["recive"] == user and block["header"]["sender"] == user:
131. user\_history\_list.append({
132. "opt": "create",
133. "timestamp": int(float(block["header"]["timestamp"])),
134. "creater": block["header"]["sender"],
135. "coin": block["tran"]["coin"],
136. "mesg": block["tran"]["mesg"]
137. })
138. user\_history\_list = sorted(user\_history\_list, key=lambda i: i['timestamp'])
139. for index, coin in enumerate(user\_history\_list):
140. user\_history\_list[index]["timestamp"] = time.asctime(time.localtime(int(float(coin["timestamp"]))))
141. return user\_history\_list
142. def get\_coin\_history(self, coin):
143. block\_list = self.get\_block\_list\_by\_coin(coin)
144. if not block\_list:
145. return False
146. coin\_list = []
147. for block in block\_list:
148. coin\_list.append({
149. "timestamp": block["header"]["timestamp"],
150. "sender": block["header"]["sender"],
151. "recive": block["tran"]["recive"],
152. "mesg": block["tran"]["mesg"]
153. })
154. coin\_list = sorted(coin\_list, key=lambda i: i['timestamp'])
155. for index, coin in enumerate(coin\_list):
156. coin\_list[index]["timestamp"] = time.asctime(time.localtime(int(float(coin["timestamp"]))))
157. return coin\_list
158. def get\_top\_block\_index(self):
159. top\_block = self.get\_top\_block()
160. return int(top\_block["header"]["index"])
161. def get\_top\_block\_hash(self):
162. top\_block = self.get\_top\_block()
163. return top\_block["block\_hash"]
164. class AccountOpertion(BlockChain):
165. """用户操作对象"""
166. def \_\_init\_\_(self, user):
167. BlockChain.\_\_init\_\_(self)
168. self.user\_history = self.get\_user\_history(user)
169. self.user = user
170. def show\_coins(self):
171. """展示一个用户的所有资产"""
172. coin\_dict = {}
173. user\_history = self.user\_history
174. if not user\_history:
175. return []
176. for history in self.user\_history:
177. coin = history["coin"]
178. if "recive" in history.keys():
179. coin\_dict[coin] = coin\_dict[coin] + 1 if coin in coin\_dict else 1
180. elif "sender" in history.keys():
181. try:
182. coin\_dict[coin] = coin\_dict[coin] - 1
183. except:
184. print("err")
185. elif "creater" in history.keys():
186. coin\_dict[coin] = coin\_dict[coin] + 1 if coin in coin\_dict else 1
187. coin\_list = []
188. for coin in coin\_dict:
189. if coin\_dict[coin] == 1:
190. coin\_list.append(coin)
191. return coin\_list
192. def show\_coin\_count(self):
193. return len(self.show\_coins())
194. def show\_trans\_history(self):
195. """查看用户的历史纪录"""
196. history\_list = self.user\_history
197. user\_history = []
198. if not user\_history:
199. return []
200. for history in history\_list:
201. if history["opt"] == "send":
202. user\_history.append({
203. "timestamp": time.asctime(time.localtime(history["timestamp"])),
204. "coin": history["coin"],
205. "desc": "发送给" + history["recive"]
206. })
207. elif history["opt"] == "recive":
208. user\_history.append({
209. "timestamp": time.asctime(time.localtime(history["timestamp"])),
210. "coin": history["coin"],
211. "desc": "收到从" + history["recive"]
212. })
213. elif history["opt"] == "create":
214. user\_history.append({
215. "timestamp": time.asctime(time.localtime(history["timestamp"])),
216. "coin": history["coin"],
217. "desc": "创建"
218. })
219. return user\_history
220. def show\_create(self):
221. """查看用户作品"""
222. coin\_dict = {}
223. for history in self.user\_history:
224. coin = history["coin"]
225. if "creater" in history.keys():
226. coin\_dict[coin] = coin\_dict[coin] + 1 if coin in coin\_dict else 1
227. coin\_list = []
228. for coin in coin\_dict:
229. if coin\_dict[coin] == 1:
230. coin\_list.append(coin)
231. return coin\_list
232. def send\_coin(self, recive, coin, mesg, sender\_key):
233. """向别人交易"""
234. index = BlockChain().get\_top\_block\_index() + 1
235. sender = self.user
236. pr\_block\_hash = BlockChain().get\_top\_block\_hash()
237. Client().sync\_chain(index=index - 1)
238. block = Block(index, sender, sender\_key, recive, pr\_block\_hash, coin, mesg)
239. if BlockChain().insert(block):
240. Client().push\_blocks([block])
241. return index
242. def create\_coin(self, coin, sender\_key, mesg=None):
243. """上传作品"""
244. index = BlockChain().get\_top\_block\_index() + 1
245. sender = self.user
246. recive = self.user
247. Client().sync\_chain(index=index - 1)
248. *# time.sleep(3)*
249. pr\_block\_hash = BlockChain().get\_top\_block\_hash()
250. block = Block(index, sender, sender\_key, recive, pr\_block\_hash, coin, mesg)
251. if BlockChain().insert(block):
252. block\_empty\_list = list()
253. print(block.block\_dict)
254. block\_empty\_list.append(block.block\_dict)
255. Client().push\_blocks(block\_empty\_list)
256. else:
257. print("shibai")
258. return index
259. class Node(object):
260. """获取存活节点"""
261. def \_\_init\_\_(self):
262. IPPOOL = Config.IPPOOL
263. conf = IPPOOL.split(".")
264. ip\_prefix = ""
265. for count in range(0, 3):
266. if conf[count] != "255":
267. ip\_prefix = ip\_prefix + conf[count] + "."
268. self.ip\_pool = list()
269. for count in range(1, 255):
270. self.ip\_pool.append(ip\_prefix + str(count))
271. self.ip\_alive\_list = list()
272. self.ip\_list = list()
273. def \_ping(self, ip):
274. cmd = "ping -n 1 " + ip + '|findstr TTL'
275. res = os.popen(cmd)
276. if res.readlines():
277. self.ip\_alive\_list.append(ip)
278. def \_ping\_mongod(self, ip):
279. try:
280. myclient = pymongo.MongoClient(host=ip, port=27017, serverSelectionTimeoutMS=50, socketTimeoutMS=50)
281. dblist = myclient.list\_database\_names()
282. if dblist:
283. self.ip\_list.append(ip)
284. except:
285. pass
286. def find(self):
287. thread\_list = list()
288. for ip in self.ip\_pool:
289. thread\_list.append(threading.Thread(target=self.\_ping, args=(ip,)))
290. for thread in thread\_list:
291. thread.start()
292. for thread in thread\_list:
293. thread.join()
294. thread\_mongod\_list = list()
295. for ip in self.ip\_alive\_list:
296. thread\_mongod\_list.append(threading.Thread(target=self.\_ping\_mongod, args=(ip,)))
297. self.ip\_list = list()
298. print(self.ip\_alive\_list)
299. for thread in thread\_mongod\_list:
300. thread.start()
301. for thread in thread\_mongod\_list:
302. thread.join()
303. return self.ip\_list
304. class Sync\_mongod(object):
305. """同步数据库"""
306. def \_\_init\_\_(self):
307. self.ip\_list = Node().find()
308. self.local\_db = Mongodb("localhost")
309. self.db\_dict = {}
310. for ip in self.ip\_list:
311. self.db\_dict[ip] = Mongodb(ip)
312. def get\_max\_index(self):
313. index\_dict = {}
314. for db in self.db\_dict.values():
315. index\_dict[db.ip] = db.get\_max\_index()
316. print(index\_dict)
317. max\_ip = max(index\_dict, key=index\_dict.get)
318. max\_index = index\_dict[max\_ip]
319. return max\_ip, max\_index
320. def get\_block\_list(self, index1, index2, ip):
321. db = self.db\_dict[ip]
322. block\_list = list()
323. for index in range(index1 + 1, index2 + 1):
324. block\_list.append(db.get\_block\_by\_index(index))
325. return block\_list
326. def add\_block\_list\_to\_net(self, ip, block\_list):
327. db = self.db\_dict[ip]
328. for block in block\_list:
329. if check\_net\_block(block, ip):
330. db.insert(block)
331. def sync(self):
332. local\_index = self.local\_db.get\_max\_index()
333. max\_ip, max\_index = self.get\_max\_index()
334. if local\_index > max\_index:
335. for ip in self.db\_dict.keys():
336. block\_list = self.get\_block\_list(max\_index, local\_index, "local\_host")
337. self.add\_block\_list\_to\_net(ip, block\_list)
338. elif local\_index > max\_index:
339. block\_list = self.get\_block\_list(local\_index, max\_index, max\_ip)
340. self.add\_block\_list\_to\_net("localhost", block\_list)
341. def server\_block():
342. """UDP接收端(接收区块)"""
343. udp\_server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
344. udp\_server\_sock.bind((get\_host\_ip(), Config.PORT))
345. udp\_server\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)
346. while True:
347. mesg, addr = udp\_server\_sock.recvfrom(65535)
348. mesg = json.loads(mesg.decode('utf-8'))
349. request = mesg["request"]
350. data = mesg["data"]
351. if request == "chuanSongKuai":
352. block\_list = data
353. for block in block\_list:
354. BlockChain().insert\_block(block)
355. elif request == "sync":
356. host\_index = BlockChain().get\_top\_block\_index()
357. index = data
358. if host\_index > index:
359. block\_list = []
360. for sync\_index in range(index, host\_index + 1):
361. block\_list.append(BlockChain().get\_block\_by\_index(sync\_index))
362. Client().push\_blocks(block\_list=block\_list, addr=addr)
363. def find\_node():
364. """查找网络上的节点"""
365. PORT = Config.SHACKPORT
366. ip\_pool = Config.IPPOOL
367. address = (ip\_pool, PORT)
368. udp\_cli\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
369. udp\_cli\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)
370. ip\_list = []
371. for count in range(100):
372. udp\_cli\_sock.sendto("nodefinder".encode('utf-8'), address)
373. mesg, addr = udp\_cli\_sock.recvfrom(1024)
374. if mesg == "Node".encode('utf-8'):
375. ip\_list.append(addr)
376. ip\_list = set(ip\_list)
377. return ip\_list
378. def response\_find\_node():
379. """监听并相应获取上线请求(守护进程)"""
380. udp\_server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
381. udp\_server\_sock.bind((get\_host\_ip(), Config.SHACKPORT))
382. udp\_server\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)
383. while True:
384. mesg, addr = udp\_server\_sock.recvfrom(1024)
385. if mesg == "nodefinder".encode('utf-8'):
386. udp\_server\_sock.sendto(bytes("Node".encode('utf-8')), addr)
387. def request\_sync(index=int):
388. """根据索引请求同步链"""
389. PORT = Config.SYNCPORT
390. udp\_cli\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
391. udp\_cli\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)
392. ip\_pool = find\_node()
393. for ip in ip\_pool:
394. address = (ip, PORT)
395. mesg = {
396. "request": "sync",
397. "data": index
398. }
399. for count in range(5):
400. udp\_cli\_sock.sendto(json.dumps(mesg).encode('utf-8'), address)
401. time.sleep(0.2)
402. def response\_sync():
403. """同步请求(守护进程)"""
404. udp\_server\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
405. udp\_server\_sock.bind((get\_host\_ip(), Config.SYNCPORT))
406. udp\_server\_sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)
407. while True:
408. mesg, addr = udp\_server\_sock.recvfrom(1024)
409. mesg = json.loads(mesg.decode('utf-8'))
410. try:
411. request = mesg["request"]
412. if request == "sync":
413. request\_index = int(mesg["data"])
414. host\_index = BlockChain().get\_top\_block\_index()
415. if request\_index < host\_index:
416. block\_list = []
417. for index in range(request\_index, host\_index):
418. block\_list.append(BlockChain().get\_block\_by\_index(index+1))
419. Client().push\_blocks(block\_list)
420. except Exception as e:
421. Log().error("同步请求响应失败：" + str(e))
422. pass
423. def check\_block(block):
424. """检查收到的块"""
425. result = True
426. try:
427. string = block["header"].\_\_str\_\_() + block["trans\_list"].\_\_str\_\_()
428. sig = block["sign"]
429. user\_name = block["header"]["sender"]
430. hash\_value = block["block\_hash"]
431. if not check\_sign(string, sig, user\_name):
432. result = False
433. if not check\_hash(string, hash\_value):
434. result = False
435. block\_index = int(block["header"]["index"])
436. db\_top\_index = BlockChain().get\_top\_block\_index()
437. if block\_index != db\_top\_index + 1:
438. result = False
439. if block["header"]["pr\_block\_hash"] != BlockChain().get\_top\_block\_hash():
440. result = False
441. sender = block["header"]["sender"]
442. coin = block["tran"]["coin"]
443. sender\_obj = AccountOpertion(sender)
444. coin\_list = sender\_obj.show\_coins()
445. if coin in coin\_list:
446. return False
447. except Exception as e:
448. Log().error("检查区块合法性失败：" + str(e))
449. result = False
450. finally:
451. return result