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，玩家可以自由地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。

针对这个系统研究可行性、性能、安全性、存在的问题等。

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（python）

Flask是一个基于python的web微型框架，使用Bsd授权，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，c#的Asp.net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。因为其轻量级的核心，Flask没有默认的数据库与窗体套件，数据库orm，身份认证， 文件上传等功能可以自己实现。

Flask提供了丰富的web组件，如：render\_template，request， request,rediret, session, 等

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。

2.1.3 P2P网络

即对等网络，网络上的所有参与者节点能够被访问，或访问别的节点而不需要一个中心化的实体。与传统的“客户端-服务端”网络不同，在这种网络上的参与节点既是网络资源的提供者，同时又是可以获取资源与服务的客户端。彼此连接的多个节点处于一个平等的地位，如图2.1。若有少数节点下线也不会影响整个P2P网的正常运行。P2P网同时还拥有非常强的可拓展性和健壮性，任何计算机都可以加入或退出节点。

对于本系统，通过区块链技术使得所有在网络上运行的计算机节点拥有在一定规则下相同的数据库，若有黑客想要修改交易记录，则必须要攻击掉全网51%的计算节点，相比与传统的中心化服务器或服务器集群，极大地提高了作恶的成本，保护网络的正常运行。

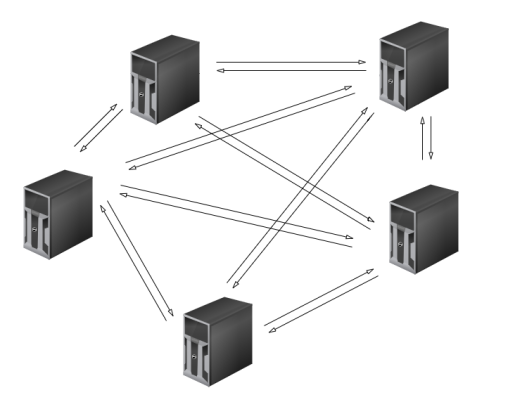


图2.1 点对点网络

2.1.4 区块链技术

区块链由一个一个区块由一定的顺序连接而成，顺序由区块中的时间戳决定，而这个时间戳代表区块生成的时间，每个区块由上一个区块的哈希值相连接如图2.2，由于区块链的特性，所有被记录到区块链中的交易记录就无法被篡改或删除。

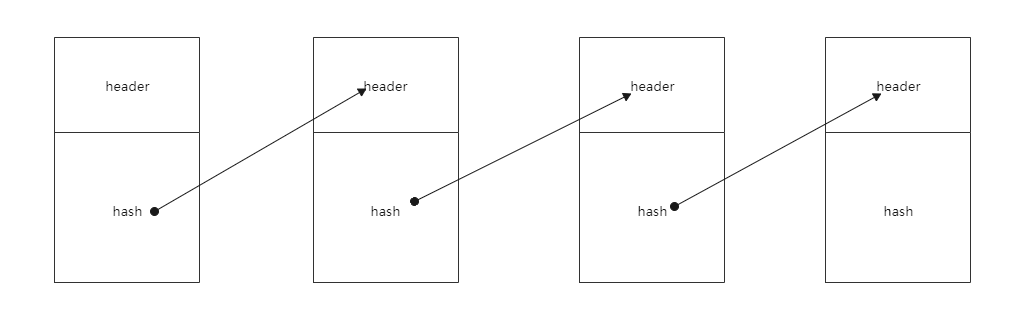


图2.2 区块链结构

区块链主要的数据格式是json，即JavaScript Object Notation，是一种轻量级的数据交换格式，广泛地应用于web地数据传输中，能够非常简洁地存储具有层次结构的键值对，非常适合存储区块链数据结构，中本聪地比特币也是采用json格式。

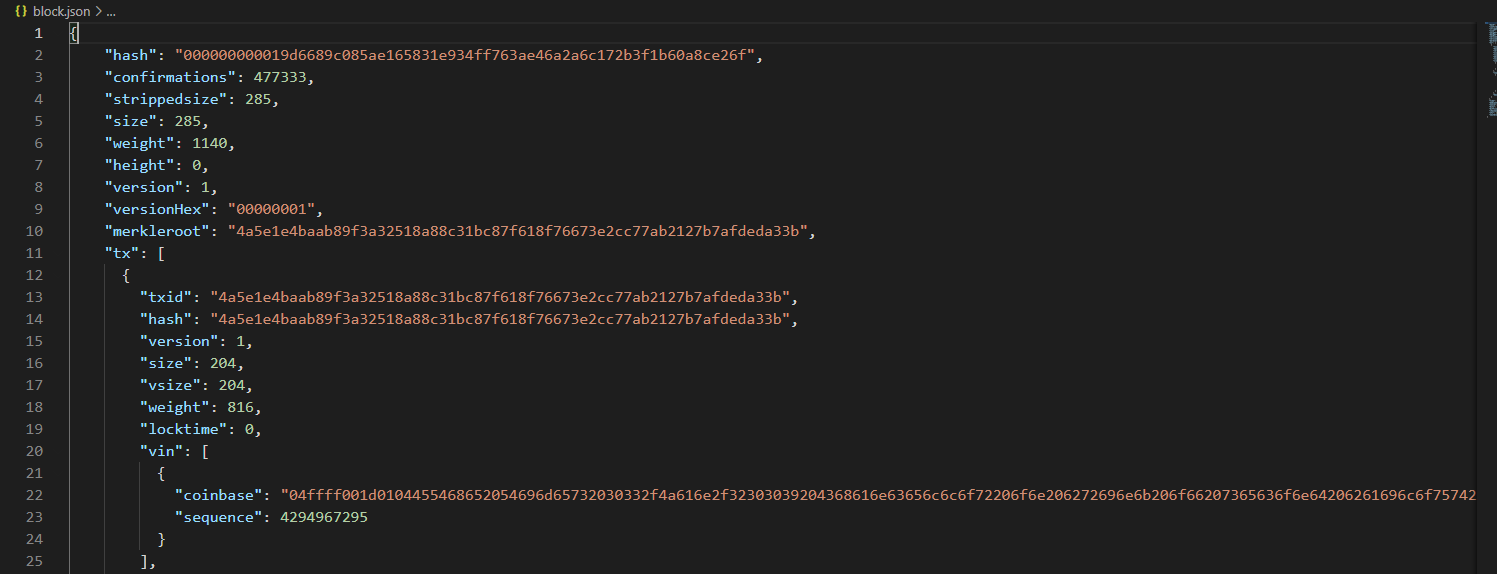


图2.3 比特币创世块的部分

区块链本身存储在数据库中，数据库分为关系型与非关系型数据库。关系型数据库的种类很多，但都遵循着一个标准化语言，数据以关系的形式存在表中。而非关系型数据库主要有四种类型：键值对存储，列存储，文档数据存储和图形数据存储。本系统使用的是Mongod文件数据存储非关系型数据库，区块以时间戳形成的索引进行排序。



图2.4 本系统的区块链存放在数据库的形式

2.1.5 ed25519 签名算法

curve25519、ed25519、x25519 是著名密码学家 Daniel J. Bernstein 在 2006 年独立设计的椭圆曲线加密、签名 、DH算法，和现有的任何椭圆曲线算法都完全独立，其中Ed25519用于签名，可在区块链中进行签名。同时该曲线的相关算法也被应用到了5G的安全标准中。

ed25519是一个基于椭圆曲线的数字签名算法，无论是签名与验签的性能都极为优秀，是目前速度最快的椭圆曲线加密算法，同时安全性极高，基本等价于4096位的RSA，签名过程不依赖随机数生成器和哈希本身的防碰撞性，没有时间通道攻击的问题。

ed25519生成的签名和公钥都很小，分别只有64和32字节。对于减小区块的大小有很大帮助。

网络中的每一个用户使用ed25519生成的公钥作为自己的地址，生成的私钥作为密码，在每笔交易被打包后，使用私钥对打包后的内容进行签名，以此证明这次交易的合法性与不可抵赖性。公钥地址与签名作为两个字段存放在区块中。在验证区块的合法性中，其中重要的一步就是使用用户地址所代表的公钥去验证签名。

其中，ed25519对象含有生成密钥对方法，生成公钥与私钥。同时，ed25519对象还有一个方法可以通过私钥生成公钥，但不能通过公钥获取私钥，如图2.5。那么，利用这个方法，在验证账户地址与密码时，通过对比用户的公钥和由私钥生成的私钥，即可完成认证。

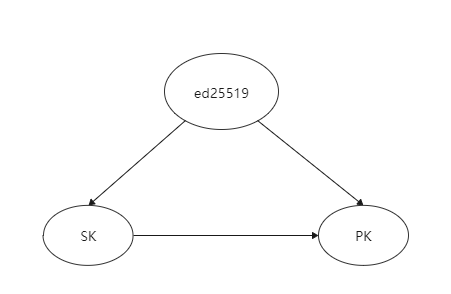


图2.5 ed25519对象

2.1.6 HASH

哈希函数也被称为散列，他能够将任意长度的输入经过一个算法而变成固定长度的输出。哈希算法可以确保传递真实的信息，也被称作完整性。

在本系统中，它可以确保交易被打包成区块后就无法被修改。同时，一个区块的哈希值也会被记录到下一个区块的头中，若该区块的内容被篡改，则该区块的哈希值也会发生改变，与下一区块头中的前哈希字段无法匹配，如图2.6在规则上，我们认为区块链断裂，这个区块后的所有内容将被作废。本系统确保区块完整性的哈希算法为SHA-256，生成256位的散列值。

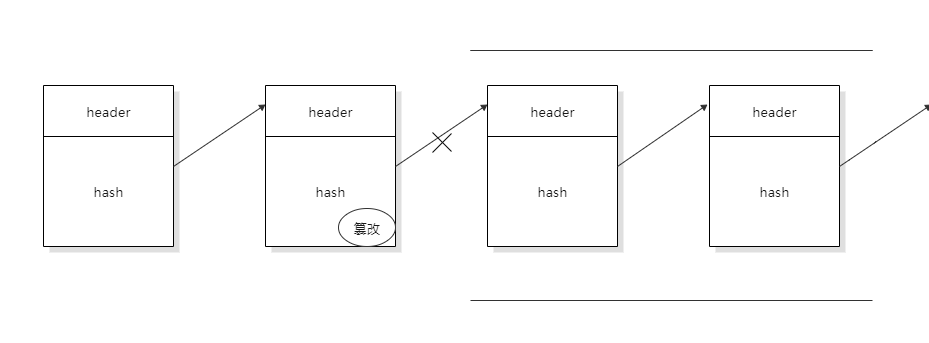


图2.6 区块链断裂

哈希函数具有抗碰撞性，这意味着找出两个哈希值完全相同的输入在概率上是非常难的，利用这个特性，我们将要上传的文件先进行编码，再使用散列函数生成一段散列值，并将这个值最为该文件的标识符。本系统使用MD5算法，将图片文件作为输入，输出一个128位的散列值。

2.2环境

……….

……..

……..

……..

……..

……..

……..

……..

……..

系统环境：Windows10，manjaro

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

需求分析是系统构思与软件实际开发阶段的中间桥梁。需求分析是对系统封最初构思的总结与拓展，也同时是之后软件设计，系统实现，前端设计，测试等的重要基础。精确而合适的需求分析可以尽早解决或避免引入问题，有效降低维护开发成本，提高系统整体水平。

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

后端在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，在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。

**系统功能汇总**

|  |  |  |
| --- | --- | --- |
| **需求名称** | **方案** | **对象** |
| 生成用户签名密钥 | 使用ed25519曲线生成密钥 | 用户 |
| 生成交易地址 | 使用签名密钥生成公钥 | 用户 |
| 登录 | 后端验证 | 用户 |
| 退出登录 | 后端清除sessions | 用户 |
| 上传皮肤 | 后端打包交易上链 | 皮肤上传者 |
| 发送皮肤所有权 | 后端打包交易上链 | 皮肤发送者 |
| 查看一个地址下的所有皮肤 | 遍历数据库，用特殊算法获取皮肤列表 | 用户 |
| 查看用户的交易记录 | 遍历数据库，用特殊算法获取交易记录 | 用户 |
| 查看皮肤的交易记录 | 遍历数据库，用特殊算法获取皮肤交易记录 | 皮肤 |
| 搜索 | 在数据库中查询 | 皮肤/用户 |

表2.1 系统功能汇总

3.8 安全性

本系统既使用了传统的web技术，也同时使用了新型的区块链技术。既要保障传统的web安全。

3.9 稳定性

稳定性对一个面对普通用户的系统非常重要。同时，此平台与资产交易相关。在这种与金钱相关的重要操作中，保障系统的稳定性至关重要。

1. 结构设计

在平台系统的开发过程中，结构设计是将需求分析进行实现，设计相应的平台架构与专用的数据结构。

首先，先对平台的结构进行设计，依据系统功能分析，将负载的系统分模块，确定各个模块的结构和互相的调用关系。

其次，是对系统的数据结构进行设计，包括区块、区块链设计以及数据库的设计。

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。

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。

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。

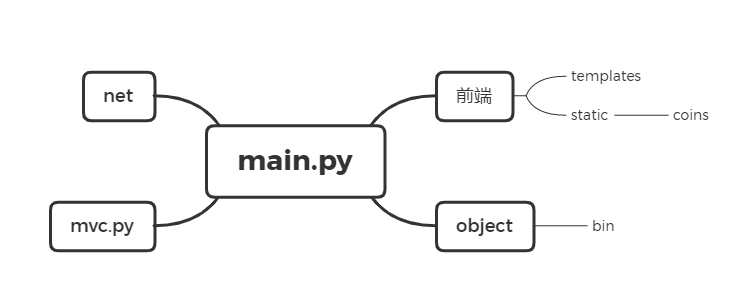


图4.1 系统架构

4.2 数据结构

4.2.1 区块

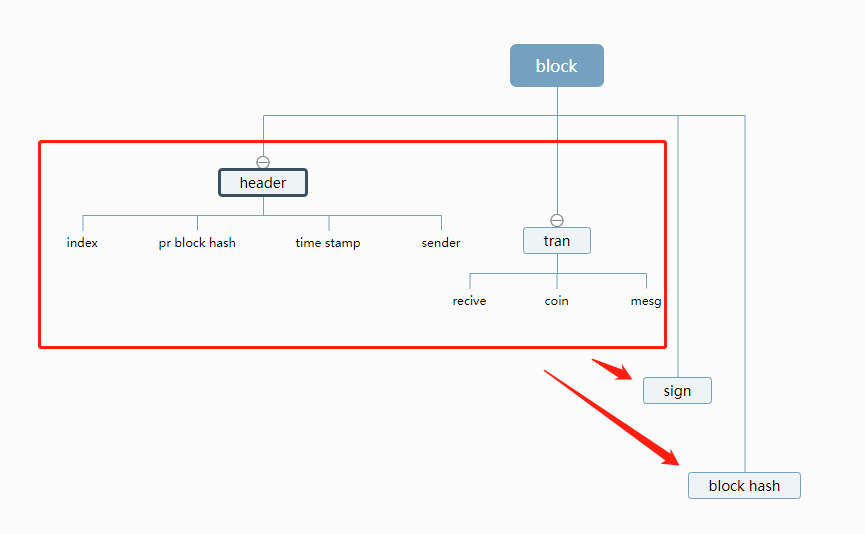


图4.2 区块结构

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. }

区块是整个区块链系统中最核心的结构，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。区块头中含有区块的关键信息。

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。

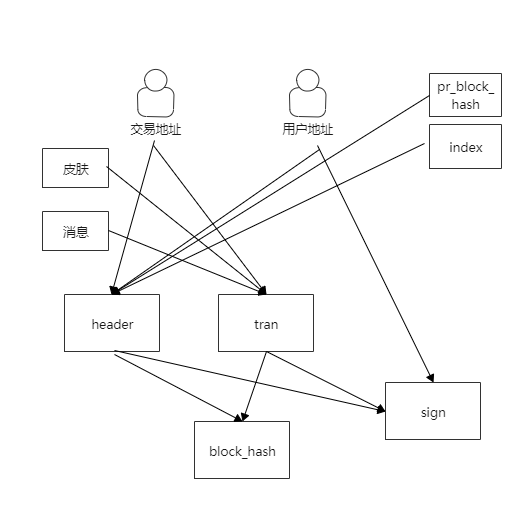


图4.3 区块生成流程

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。

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

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。

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

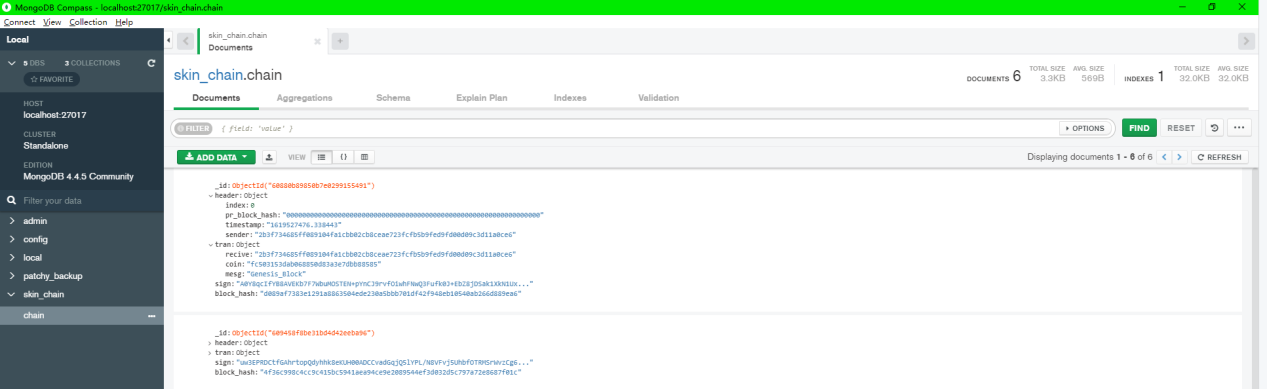


图4.4 新建用户流程

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。

**区块链对象方法**

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 参数 | 描述 | 返回值 |
| Insert\_block | Block:\_\_block\_object | 在区块链对象中插入新的区块 | None |
| get\_user\_history | user:str | 获取用户的交易记录 | user\_history\_list:list |
| get\_coin\_history | coin:str | 获取皮肤交易记录 | create:str  coin\_list:list |
| get\_top\_block\_index | None | 获取区块链高度 | index:int |
| get\_top\_block\_hash | None | 获取区块链的最新区块的哈希值 | Hash:str |
| get\_all\_coin | None | 获取记录在区块链中的所有皮肤 | coin\_list:list |
| get\_block\_by\_user\_coin | user:str  coin:str | 根据地址和皮肤定位到相应的去区块 | history:list |
| get\_history | None | 获取区块链的 历史 | history:list |

图2.2 区块链对象方法

4.3.3 用户对象

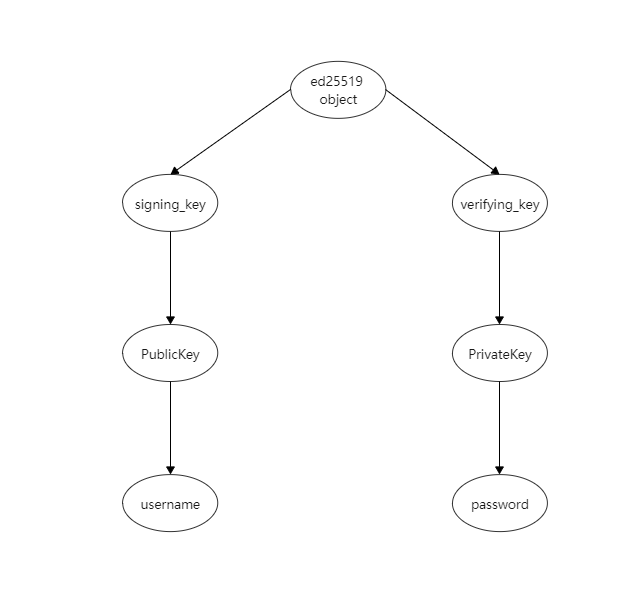
新建用户的密钥对，使用第三方库ed25519，分别创建私钥和公钥对象，再使用对象的to\_ascii方法，获取方便存放和传输的字符串。将生成的密钥对进行裁切

图4.4 新建用户流程

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。

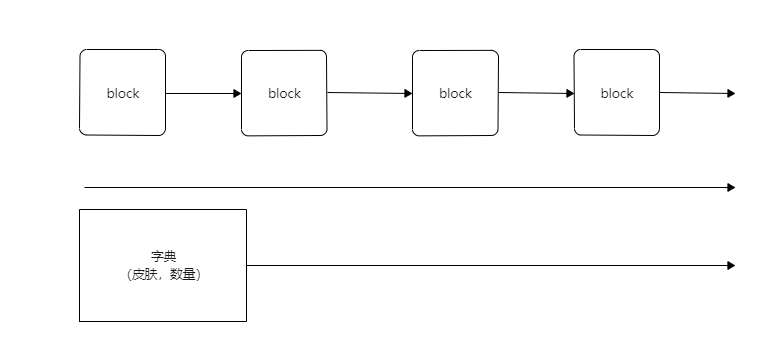


图4.6 遍历区块链

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。

**用户对象**

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 参数 | 描述 | 返回值 |
| \_\_init\_\_ | user:str | 初始化 | None |
| show\_coin | None | 展示一个用户的所有资产 | coin\_list:list |
| show\_coin\_count | None | 获取一个用户的资产数量 | coin\_count:int |
| show\_trans\_history | None | 获取用户的交易历史 | user\_history:list |
| show\_create | None | 获取用户所有上传的皮肤 | coin\_list:list |
| send\_coin | Recive:str  coin:str  mesg:str  sender\_key:str | 向别的用户发送皮肤 | index:int |
| create\_coin | coin:str  sender\_key:str  mesg:str | 上传皮肤 | index:int |

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。用于区块的散列计算。

对文件取HASH， 使用md5算法。设置一次读取的长度为1024byte。输入文件的路径，将文件进行序列化，最终输出32位散列值，作为皮肤文件的资源值

4.4.1文件

移动文件，用于上传皮肤文件。前端在将文件上传到后端之后，后端先将图片文件存放在temp临时文件夹中，再对图片文件进行序列化，计算md5值。 创建以该md5码位文件夹名的文件夹，再将皮肤文件从temp临时文件夹移动到该文件夹，该文件夹位于static的pic中，方便之后前端显示。

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。

只要有任何一个步骤的验证没有通过，则区块验证失败

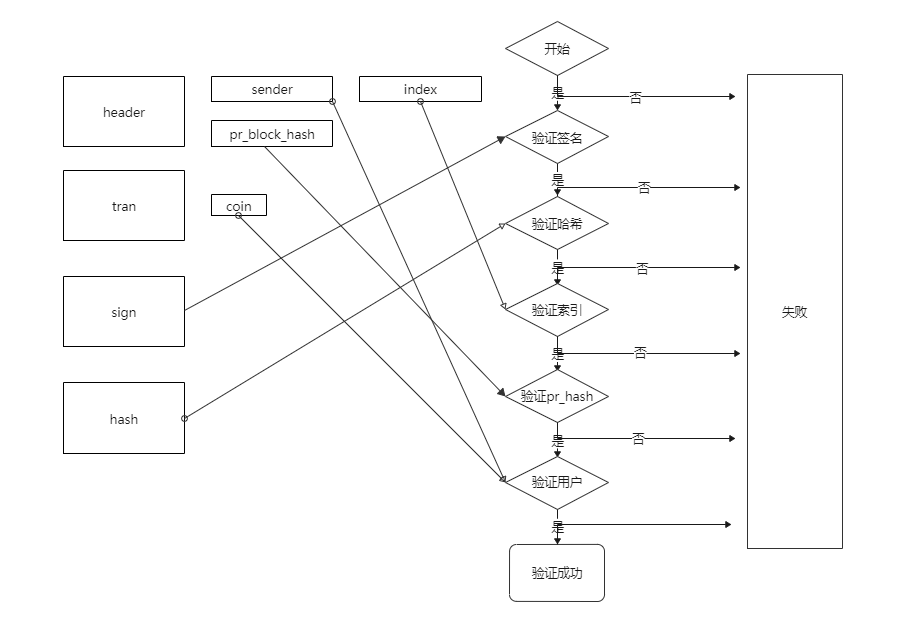


图4.8 验证区块流程

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 后端接口设计

后端接口存放在main.py中

4.5.1 根路由 <http://127.0.0.1:5000/> 【GET】

渲染主页

4.5.2 登录 <http://127.0.0.1:5000/login/> 【POST， GET】

{

Username: str,

Password: str

}

查看后端的session的状态，若session字段的username与password字段仍未失效，表明已经登录则重定向到用户界面。

若session未含有username和password字段，则表明未登录，先进行验证，验证通过，存入session，重定向到用户界面。

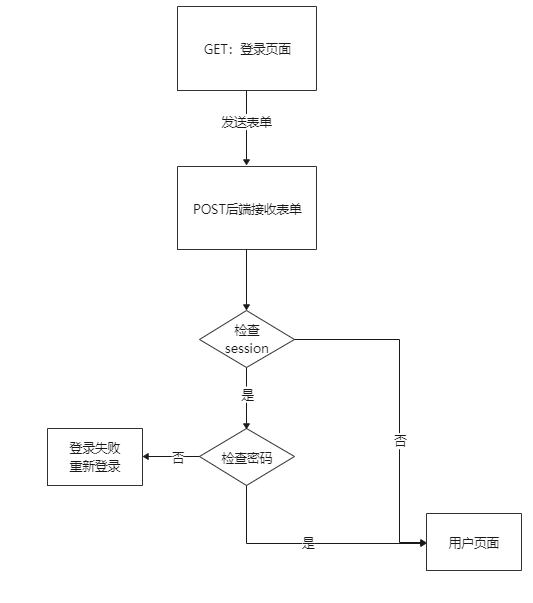


图4.9 登录流程

4.5.3 退出登录 [http://127.0.0.1:5000/logout/](http://127.0.0.1:5000/login/) 【GET】

清除session，以达到退出登录的效果。

4.5.4 创建地址 [http://127.0.0.1:5000/create/](http://127.0.0.1:5000/login/) 【GET】

使用Account对象的create\_account()方法，创建密钥对，返回前端。

{

Username: str,

Password: str

}

4.5.5 用户页面 [http://127.0.0.1:5000/user/](http://127.0.0.1:5000/login/)<user> 【GET】

user字段为用户地址，该页面是通用的用户页面，对所有人可见，会显示该用户的资产数量。通过user字段初始化用户对象，再使用该对象的show\_coin\_count()方法获取该用户的资产数量。最后返回前端。

4.5.6 交易 [http://127.0.0.1:5000/user/](http://127.0.0.1:5000/login/)<user>/tran/ 【GET, POST】

1.GET：user字段为用户地址，先查看session的username与password字段，查看是否登录，若未登录则重定向到登录界面，确认登录之后，通过user字段初始化用户对象，再使用该对象的shop\_coins()方法获取该用户所拥有的资产列表，通过这些资产进行拼接，获取这些图片资源的链接并返回前端。

2.POST：

{

Username:str,

Pic:str,

Messge:str

}

使用用户对象的send\_coin（）方法，传入的参数有接收方的姓名，资产名称，消息与签名私钥，其中签名私钥从session的password字段中获取，该方法的返回值为产生的新区块的高度，重定向到交易成功界面。

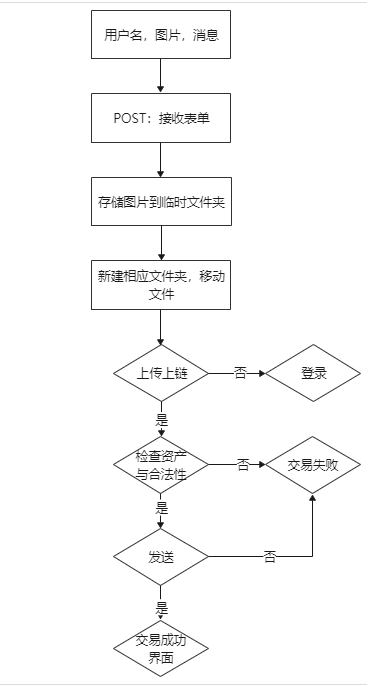


图4.10 交易流程

4.5.7 交易成功 [http://127.0.0.1:5000/user/](http://127.0.0.1:5000/login/)<user>/tran/succese/<index> 【GET】

字段index为交易生成区块的高度，使用BlockChain对象的get\_block\_by\_index()的方法获取相应的区块。从区块的tran中的recive字段获取接收者地址，从header的timestamp字段获取时间戳，从区块的tran的coin字段获取皮肤，加工后获取该皮肤的链接，并将这些数据返回前端。

4.5.8 交易历史 [http://127.0.0.1:5000/user/](http://127.0.0.1:5000/login/)<user>/showtrans/ 【GET】

字段user为用户地址，使用该字段初始化用户对象，使用用户对象的show\_trans\_history（）方法获取用户的所有交易历史并返回前端。

4.5.9 用户财产 [http://127.0.0.1:5000/user/](http://127.0.0.1:5000/login/)<user>/showcoins/ 【GET】

字段user为用户地址，使用该字段初始化用户对象，使用用户对象的show\_coins（）方法获取用户的所有皮肤资产并返回前端。

4.5.10 上传皮肤 [http://127.0.0.1:5000/uploader/](http://127.0.0.1:5000/login/) 【POST】

{

Username:str,

Photo: str,

Mesg:str，

Img:byte

}

从表单分别获取用户名。图片文件与消息，先将文件存在temp临时文件，调用哈希函数得到哈希值，再static文件夹的img中新建以该哈希值为文件名的文件夹，再将该文件移动到该文件夹。移动完成后再使用用户名地址生成用户对象，调用create\_coin()方法上传资产，上传完成后重定向到上传成功界面

若未成功则重定向到上传失败页面。

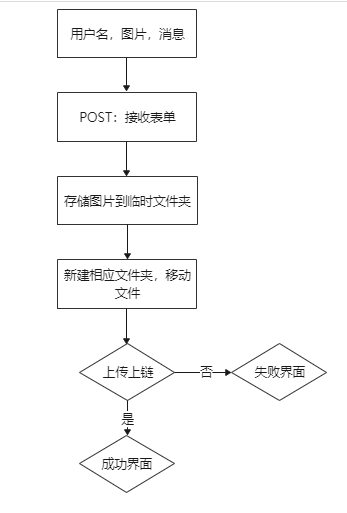


图4.11 上传流程

4.5.11 皮肤页面 [http://127.0.0.1:5000/coin/](http://127.0.0.1:5000/login/)<coin>/ 【GET】

本结构用于显示皮肤的详情页，coin字段为皮肤的MD5值，通过该字段，使用BlockChain对象的get\_coiun\_history()方法获取该皮肤的交易历史记录与上传人，并根据coin字段拼接出图片链接，将历史记录，图片链接，上传者返回前端。

4.5.11 区块链上的所有皮肤 [http://127.0.0.1:5000/coins/](http://127.0.0.1:5000/login/) 【GET】

使用BlockChain对象的get\_all\_coins()方法，获取区块链所有的皮肤的列表，将这个列表进行拼接，获取相应的图片链接的列表。

4.5.12 搜索 [http://127.0.0.1:5000/search/](http://127.0.0.1:5000/login/) 【POST】

该接口同时支持对皮肤和用户的搜索，从表单获取username字段，检查该字段的长度，若长度为64，则该字段表示用户名，根据该字段重定向到用户主页。若长度为32，则重定向到皮肤主页。

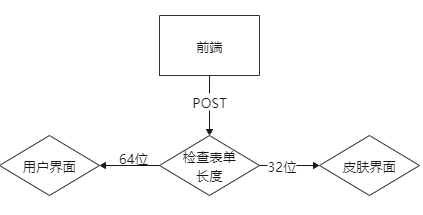


图4.12 搜索流程

4.5.13 搜索 [http://127.0.0.1:5000/display/img/](http://127.0.0.1:5000/login/)<coin> 【GET】

一个显示图片的接口，其中coin字段为皮肤md5码。通过这个coin字段获取相应的存放路径, 通过这个路径，获取文件流，并返回前端。

4.6 web设计

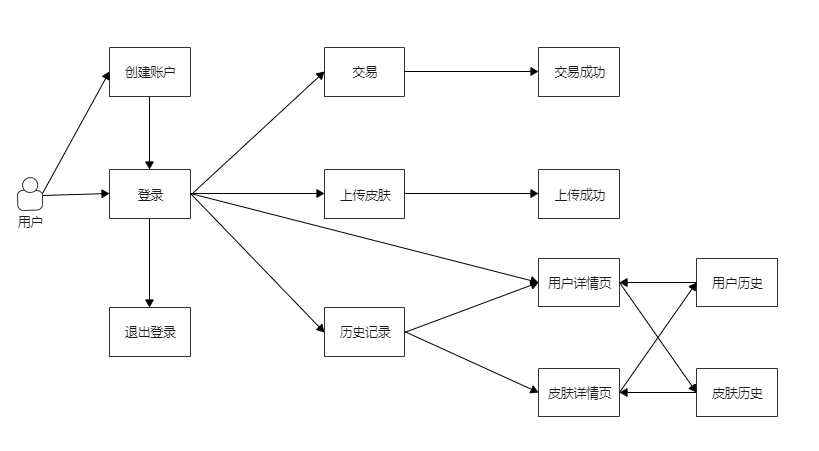


图4.13 用户操作逻辑

4.6.1 主页

介绍平台的基本信息，给出进行各种操作的连接



图4.14 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。

其中图标与首页的链接指向这个首页，在其他页面点击这个链接会直接跳转到首页，登录链接会跳转到登录页面。注册链接会跳转到新用户界面。点击用户中心链接，如果已经登录，则会跳转到用户中心，如果没有登录，则会跳转到登录界面。点击历史，会取得区块链中的所有交易历史，点击皮肤链接，则会显示区块链上的所有资产。

4.6.2 登录



图4.15 登录表单

在前端中，单独给出了一个div对象来放置表单。检查表单，用户名与密码应为64位的数字与小写字母。

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。

4.6.3 退出登录

退出登录的按键在用户中心，点击按钮后，会调用logout后端接口。

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/')

4.6.3 创建新用户



图4.17 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。



图4.18 新账户生成页面

4.6.4 交易

从后台链中获取该用户的所有皮肤，返回给前端。前端上根据这个皮肤列表渲染出一个表。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。

成功界面先显示这个皮肤的哈希识别码，其次下面是该皮肤的图片文件。再下面是该交易的详细信息，包括皮肤发送者，接收者，与交易发生的时间。

4.6.5 上传皮肤

若要上传皮肤资产，则点击页眉上的上传链接，则跳转到上传皮肤界面。页面上显示该用户的地址，下面有选择方档案的按钮，点击按钮就可以选择存放在本地的图片，选择并上传至后端。上传完成后，在消息栏填入要上链的额外消息，点击提交后就完成了交易。

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。

上传成功后，跳转到上传成功界面。该界面的标题为上传成功，下面为交易成功的皮肤资产的皮肤文件，下面是该交易的具体信息，包括哈希值，交易时间。最后在下方是一个链接，点击后可以继续交易。

4.6.6 搜索



图4.23

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。

这个功能在最初的设计时，进行过多次方案设想。有如下几种：

1. 使用两个单独的界面，分别进行皮肤搜索和用户搜索。分别使用不同的后端接口进行查询。
2. 在同一界面集成这两种功能，在前面的选择框先进性选择，在皮肤与用户查询中选择一个。
3. 最终选定的方案为只用统一的搜索框与接口，因为用户的地址和皮肤的字符长度不同。用户地址为64位，密码为32位。后端对前端传进的字符串的长度进行判定，若长度位64位，则认定用户查询的是用户，后端直接根据这个用户地址重定向到用户页面。若长度位32，则认定用户查询的是皮肤，则后端根据这个皮肤识别码重定向到皮肤主页。

4.6.7 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。

4.6.7 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。

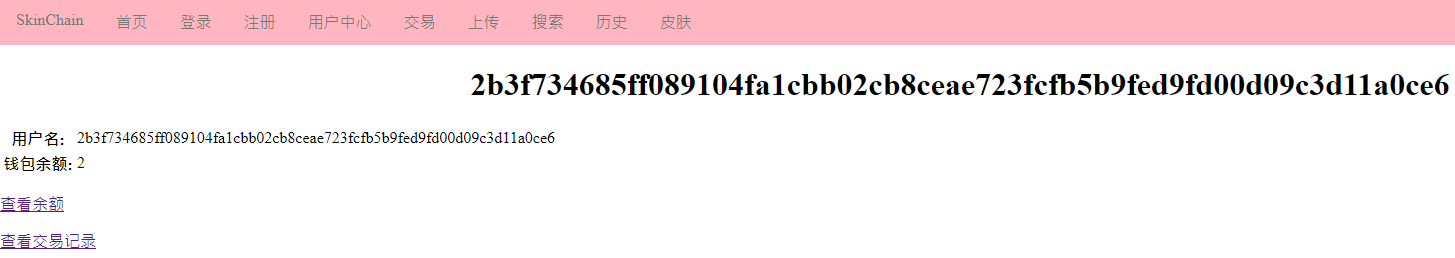


图4.25

4.6.8 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。

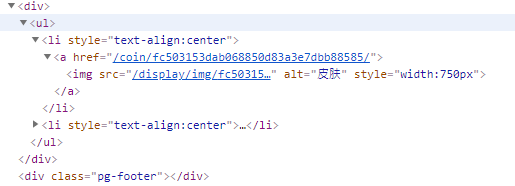


图4.26

1. 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，而接收者地址存放在交易中，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

查看git log，本系统于2021/4/15 at 9:32创建仓库， 到2021/5/20 at 16:49最后一次提交，中间一共commit了28次。设计论文于2021/4/28 at 8:00创建，完成于2021/5/20 at 16:49。在这段系统开发的过程中我要首先感谢罗维导师这段时间对我的帮助，没有导师对我疑问的解答与讲解，我是无法完成这套系统于编写完设计论文。其次我要感谢帮助我解答疑问的室友于同学。感谢开源框架flask和开源数据库mongod及其开发者，感谢他们为我设计这套系统提供必要的工具。最后，感谢中本聪先生，以及所有孕育了区块链的密码学，点对点网络等技术，感谢他们对我的启迪。