区块链算法和实现思路报告

2151133 孙韩雅

一、区块链算法概述

区块链是一种分布式的、不可变的数据结构，由一个个被称为区块的数据块按照时间顺序连接而成。每个区块包含了一组交易数据以及与之相关的元数据，例如时间戳和前一个区块的哈希值。这种链式结构使得区块链具备了去中心化、透明、安全和可验证等特性。

在区块链中，每个区块都有一个唯一的哈希值，该哈希值是通过对区块中的数据进行哈希运算得到的。每个区块的哈希值也会包含前一个区块的哈希值，这样就形成了一个链式结构。当区块链中的数据发生变化时，前后区块的哈希值也会随之改变，从而确保数据的完整性和安全性。

在实现区块链时，需要使用哈希函数对区块进行哈希运算来生成唯一的哈希值。哈希函数是一种将任意长度的数据映射为固定长度哈希值的函数。在我们的实现中将使用 SHA-256 哈希函数。

SHA-256（Secure Hash Algorithm 256-bit）是一种密码学安全哈希函数，它接受任意长度的输入并生成固定长度的 256 位哈希值，逻辑如下：

1. 初始化常量：定义一组初始的 256 位常量，这些常量是预定义的，并且与 SHA-256 算法相关。

2. 数据预处理：对输入数据进行预处理，包括填充和附加长度信息。填充是指在输入数据的末尾添加比特位，使其达到满足特定长度要求（例如 512 位块）。附加长度信息是指将输入数据的原始长度编码为二进制，并附加到填充后的数据末尾。

3. 划分数据块：将预处理后的数据分割为 512 位的数据块。如果数据块不足 512 位，则将其填充至 512 位。

4. 初始化哈希值：定义一个 256 位的哈希值，作为 SHA-256 算法的初始值。这个初始哈希值是根据 SHA-256 算法的规范确定的。

5. 处理数据块：对每个数据块进行处理。

   5.1 初始化工作变量：定义一组工作变量，这些变量是在处理每个数据块时使用的中间结果。

   5.2 压缩函数：应用压缩函数对当前数据块和工作变量进行操作，并产生新的工作变量。

   5.3 更新哈希值：根据压缩函数的输出更新当前的哈希值。

6. 输出结果：当处理完所有数据块后，将最终的哈希值作为 SHA-256 算法的输出结果。

二、区块链实现思路

基于以上的区块链算法概述，我们使用 JavaScript 来实现一个简单的区块链逻辑。下面是具体的实现思路：

1. 创建区块类（Block）

区块类用于表示区块链中的每个区块，具有如下属性：

索引（index）：标识区块在链中的位置；

时间戳（timestamp）：记录区块的创建时间；

数据（data）：存储区块中的交易数据或其他相关信息；

前一个区块的哈希值（previousHash）：记录前一个区块的哈希值，用于确保链的完整性；

当前区块的哈希值（hash）：通过对区块的索引、时间戳、数据和前一个区块的哈希值进行哈希运算而得到；

```
    class Block {
      constructor(index, timestamp, data, previousHash) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.previousHash = previousHash;
        this.hash = this.calculateHash();
      }

      calculateHash() {
        const data = this.index + this.timestamp + JSON.stringify(this.data) +
this.previousHash;
        const hash = CryptoJS.SHA256(data).toString();
        return hash;
      }
    }
```

2. 创建区块链类（Blockchain）

区块链类用于管理和操作区块链。包含以下方法：

createGenesisBlock ：用于创建创世块（即第一个区块）。创世块是链的起点，其前一个哈希值为固定值。在您的代码中，创世块的数据包含了一个示例数据对象。

getLatestBlock ：用于获取区块链中的最新区块。

addBlock ：用于向区块链中添加新的区块。在添加新区块时，需要更新新区块的前一个哈希值和当前区块的哈希值。具体而言，通过调用 getLatestBlock 方法获取最新区块，将其哈希值赋值给新区块的 previousHash 属性，然后使用新区块的数据计算哈希值并赋值给 hash 属性。

updataBlock ：用于更新区块链中特定索引位置的区块的数据。函数首先检查给定的索引是否在合法范围内（大于等于 0 且小于链的长度）。如果索引有效，则将该索引对应的区块的 data 属性更新为提供的新数据 newData。

isChainValid ：用于验证整个区块链的有效性。该方法通过遍历链数组中的每个区块，逐个比较当前区块的哈希值和前一个区块的哈希值，确保链的完整性和安全性。

```
    class Blockchain {
      constructor() {
        this.chain = [this.createGenesisBlock()];
      }

      createGenesisBlock() {
        const productId = this.calculateHash(0, new Date().toISOString(), 'Genesis Block',
'0');
        return new Block(0, new Date().toISOString(), {
          productId,
          producer: '',
          location: '',
          responsiblePerson: '',
          carrier: {
            origin: '无',
            destination: '无',
            transporter: '无'
          }
        }, '0');
      }

      getLatestBlock() {
        return this.chain[this.chain.length - 1];
      }

      addBlock(newBlock) {
        newBlock.previousHash = this.getLatestBlock().hash;
        newBlock.hash = newBlock.calculateHash();
        this.chain.push(newBlock);
```

```
    }

    updateBlockData(index, newData) {
      if (index >= 0 && index < this.chain.length) {
        this.chain[index].data = newData;
      }
    }

    isChainValid() {
      for (let i = 1; i < this.chain.length; i++) {
        const currentBlock = this.chain[i];
        const previousBlock = this.chain[i - 1];

        if (currentBlock.previousHash !== previousBlock.hash) {
          return false;
        }
      }

      return true;
    }

    calculateHash(index, timestamp, data, previousHash) {
      const dataToHash = index + timestamp + JSON.stringify(data) + previousHash;
      const hash = CryptoJS.SHA256(dataToHash).toString();
      return hash;
    }
  }
```

3. 使用区块链

导入区块链类，并创建一个区块链实例。

使用 addBlock 方法向区块链中添加新的区块。每个区块都包含了特定的数据和相关信息。

在需要验证区块链的有效性时，调用 isChainValid 方法。该方法会检查每个区块的哈希值和前一个区块的哈希值是否匹配，以确保整个链的完整性。

4. 哈希函数

在实现区块链时，需要使用哈希函数对区块进行哈希运算来生成唯一的哈希值。哈希函数是一种将任意长度的数据映射为固定长度哈希值的函数。在 calculateHash 函数中，我首先将区块的索引(index)、时间戳(timestamp)、数据(data)和前一个区块的哈希值(previousHash)拼接成一个字符串(data)。然后使用 CryptoJS.SHA256 函数将该字符串作为输入，计算其 SHA-256 哈希值。最后用.toString()将哈希值转换为字符串并返回。

5. 产品供应链

基于茅台的产业链模拟出关键角色，分别为生产商、运输商和消费者。

生产商是指负责生产茅台酒的企业或酒厂。他们负责从原材料采购到酿造和灌装茅台酒。生产商在生产过程中需要记录关键信息：生产商、生产地、生产负责人等，以及关键性的生产时间和防伪码（哈希码）。

运输商是负责茅台酒从生产商到消费者之间的物流运输的企业或物流公司。他们负责将茅台酒从生产地运送到销售点或消费者手中。运输商可以利用区块链技术来记录物流信息：发货地、目的地、运输员等。通过区块链的不可篡改性和透明性，运输商可以提高物流过程的可信度和透明度。

消费者是最终购买和使用茅台酒的个人或机构。通过区块链溯源技术，消费者可以扫描产品上的二维码或访问相关平台，获取茅台酒的生产信息、运输信息和验证信息，本系统中我们通过防伪码的搜索来进行验证。

三、系统实现与说明

源代码：https://github.com/Duck800/BlockChainMAOTAI

本系统采用 Vue 3 作为前端框架，使用 JavaScript 语言进行开发。所有的数据存储保存在 localStorage 以模拟数据库。

实现茅台酒供应链系统主要实现两个关键功能。

①用户界面：使用 Vue 3 构建用户界面，包括生产商、运输商和消费者的不同视图。每个角色都有自己的操作界面，以便于他们管理和查看相关数据。

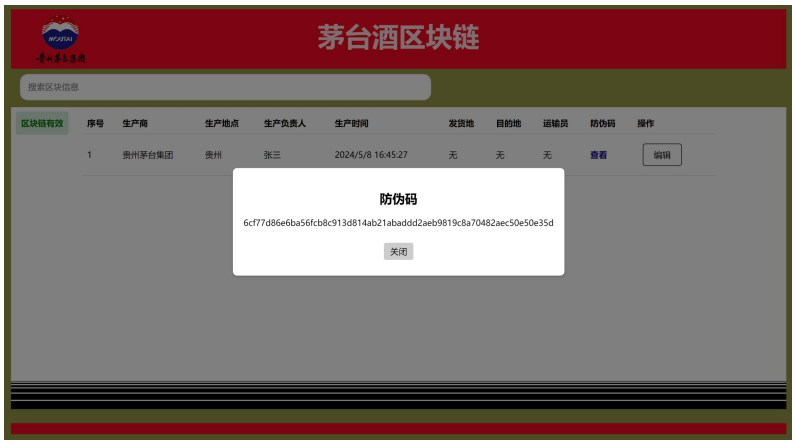
②溯源功能：利用区块链技术实现茅台酒供应链的溯源功能。每个关键操作（生产、运输、验证）都使用区块链进行记录，确保数据的不可篡改性和可追溯性。

1. 页面详解

- 首页：根据角色选择对应的按钮入口，分别进入生产商/运输商/消费者的界面。

- 生产商页面：提供添加数据按钮，可以填写新商品信息（生产商、生产地点、负责人）；也可在区块链表格中对相应的商品进行数据的修改。

- 运输商页面：与生产商略有不同，无法添加新的数据，因为运输商只负责运输，介于区块链的安全性，不开放运输商对于茅台商品添加的功能。因此运输商页面智能够对商品进行编辑或查看；运输商的信息比生产商多一部分（发货地、目的地、运输员），是他们负责填写的部分；点击防伪码下的查看，即可看到详细的哈希码。



- 消费者页面：消费者一般以搜索防伪码的形式进行使用，因此不会对商品进行修改操作。在示例图中，我们搜索了包含关键词的商品防伪码，最终得出两个结果。用户可以点击防伪码进行查看与验证。

2. 系统运行

进入根目录下，在终端输入命令：`npm install`，等待处理完成后，再输入 `npm run dev` 运行系统，并在 http://localhost:5173/中打开系统。