短学期项目报告

1131190111-唐川淇，分工：设计、编程、撰写文稿

1131190112-刘玉鹏，分工：查找资料

介绍：

- 区块链（不是比特币，请大家辩证看待）是一种数据结构
- 介绍视频 1：https://space.bilibili.com/9458053/search/video?keyword=%E5%8C%BA%E5%9D%97%E9%93%BE
- 介绍视频 2：https://space.bilibili.com/43276908/channel/collectiondetail?sid=28609
- 如果有时间，也可系统性学习北大课程：https://www.bilibili.com/video/BV1Vt411X7JF

任务：

- 1~2 人一组（在本文档顶部注明分工）
- 完成基于区块链的物流信息存储系统（下图为示意图，仅供参考）
  - 参考链接 1：https://www.bilibili.com/video/BV15L4y1i7FM?vd_source=662217b290c93492394fc14797e13b63
- 完成本文档后面的项目说明
- 课程期间全程参加钉钉会议，不定时随机点名共享屏幕讲解进度
- 7 月 12 日 24:00 前将整个项目代码和本文档提交至钉钉群
- 思考下列开放式问题，形成自己的观点，并写在文档末尾：
  - 是不是万物皆可区块链，区块链到底能用来做什么？
    - 参考链接 1：https://www.bilibili.com/video/BV1V5411p7ye
  - 区块链能真正做到去中心化吗？
  - 区块链能真正做到不可篡改吗？
  - 更多问题可自行发挥 …


物理锚定
条形码
二维码
RFID
NFC
第三方系统
代码质量
代码质量
代码质量
PDA/扫码设备
展现层
溯源小程序
H5/Web
区块链溯源服务平台
码管理
身份管理
品牌商管理
溯源环节管理
溯源链管理
信息上链
溯源查询
溯源统计
区块链BaaS

项目说明

功能/特性	说明	分值	备注（请在此备注是否完成，及其他信息）
基础	必选		
	文档内容充实、排版美观	5	完成
	无编译错误，可顺利运行	5	完成
C++	必选		
	多文件结构	10	完成
	面向对象（类和继承）	10	完成
	设计合理(需在文档中说明)	10	完成
区块链	必选		
	具备区块链基础功能	10	完成
	对区块链的思考	10	完成
区块链++	可选	30	
	去中心化		完成（分布式账本）
	数据安全		完成（挖矿机制）
	不可篡改		完成（GHOST 协议）
	可溯源		完成
	轻节点		完成
	…		
汇报	在线做演示和汇报	10	完成

C++小学期

简介

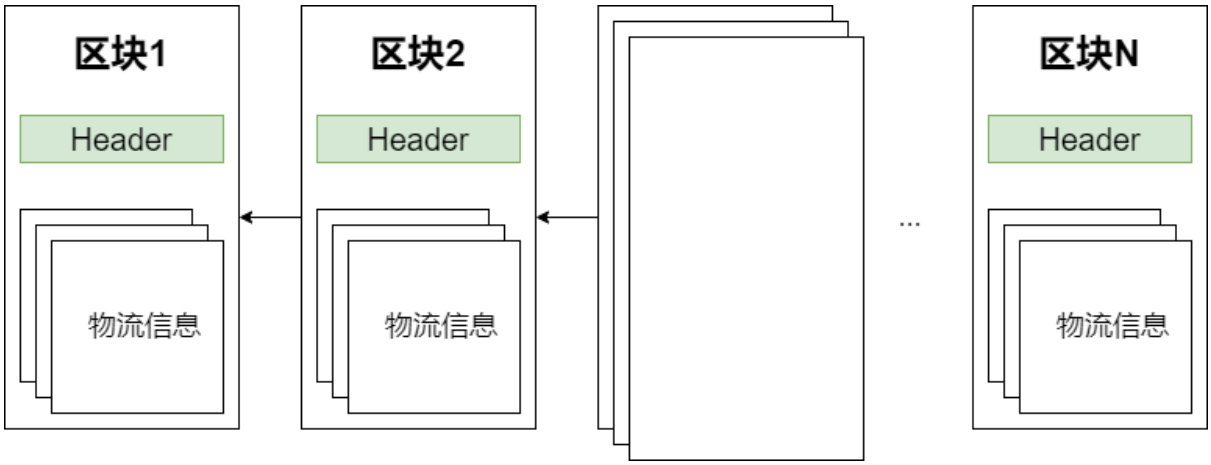
模型说明

该项目本质就是一个基于交易的状态机(transaction-based state machine)。在计算机科学中，一个状态机是指可以读取一系列的输入，然后根据这些输入，会转换成一个新的状态出来的东西。该状态机如下：

```
class BlockChainSys {
  // 指针永远处于最新状态
  Block* Bptr;        // 区块链
  LightBlock* LBptr;  // 轻量级区块链
  // 增量信息
  int dnU, dnLU, dnFU, dnM, dnS, dGas;
  // 信息
  int nU, nLU, nFU, nM, nS;
  string sUser;// 用户头哈希
  string sStuff;// 物品头哈希
```

```
  string sTrade;// 交易头哈希
  vector<FullUser> Vfuser;
  vector<LightUser> Vluser;
  vector<Miner> Vminer;
  vector<Stuff> Vstuff; // 所有的物品
  vector<Trade> Vtrade; // 所有的交易
  int t;                // 系统处于的时间
public:
  BlockChainSys();
  void PrintInformation();    // 打印信息
  void HelpInformation();     // 指令集
  void Create();              // 创建
  void Tradeb();              // 交易
  void AddInformation();      // 加工物品
  void ls();                  // 展示对应表
  void StuffHis();            // 产品溯源
};
```

根据该项目的状态机，我们从创世纪状态开始，当交易被执行后，这个创世纪状态就会转变成最终状态。在任何时刻，这个最终状态都代表着当前的状态，在项目中指针`Bptr`永远指向最新的状态。每个状态可能包含数个物流信息。这些信息都被“组团”到一个区块中。一个区块包含了一系列的物流信息，每个区块都与它的前一个区块链接起来。



挖矿机制

为了让一个状态转换成下一个状态，交易必须是有效的。为了让一个交易被认为是有效的，它必须要经过一个验证过程，此过程也就是挖矿。挖矿就是一组节点用它们的计算资源来创建一个包含有效交易的区块出来。

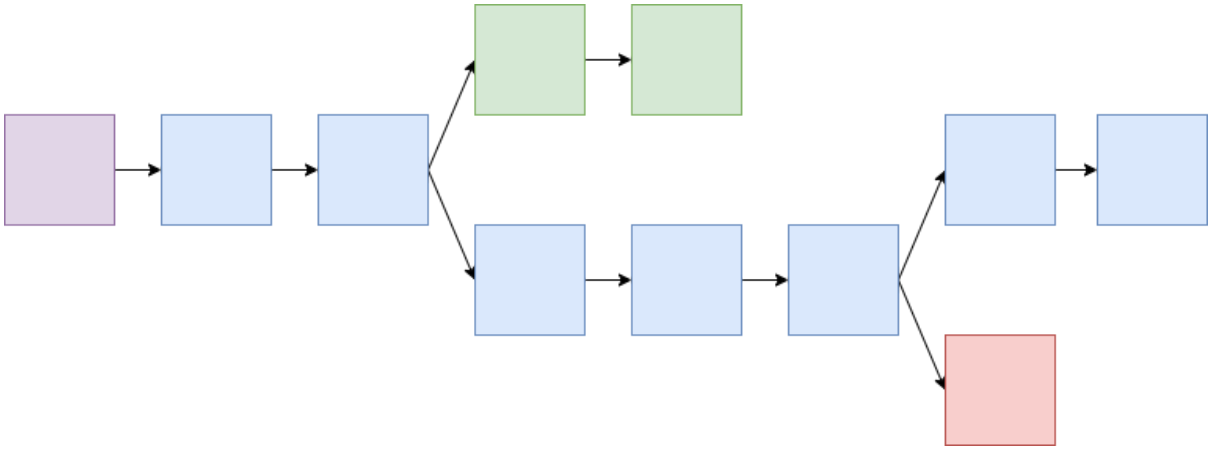
任何在网络上宣称自己是矿工的节点都可以尝试创建和验证区块。很多矿工都在同一时间创建和验证区块。每个矿工在提交一个区块到区块链上的时候都会提供一个数学机制的“证明”，这个证明就像一个保证：如果这个证明存在，那么这个区块一定是有效的。

为了让一个区块添加到主链上，一个矿工必须要比其他矿工更快的提供出这个“证明”。通过矿工提供的一个数学机制的“证明”来证实每个区块的过程称之为工作量证明(proof of work)。

证实了一个新区块的矿工都会被奖励一定价值的奖赏该项目使用一种内在数字代币作为奖赏。每次矿工证明了一个新区块，那么就会产生一个新的货币并被奖励给矿工。

GHOST协议

为了保障数据的安全性以及可靠性，本项目也采用了GHOST协议（Greedy Heaviest Observed Subtree）。简单来说，GHOST协议就是让我们必须选择一个在其上完成计算最多的路径。一个方法确定路径就是使用最近一个区块的区块号，区块号代表着当前路径上总的区块数。区块号越大，路径就会越长，就说明越多的挖矿算力被消耗在此路径上以达到叶子区块。使用这种推理就可以允许我们赞同当前状态的权威版本。



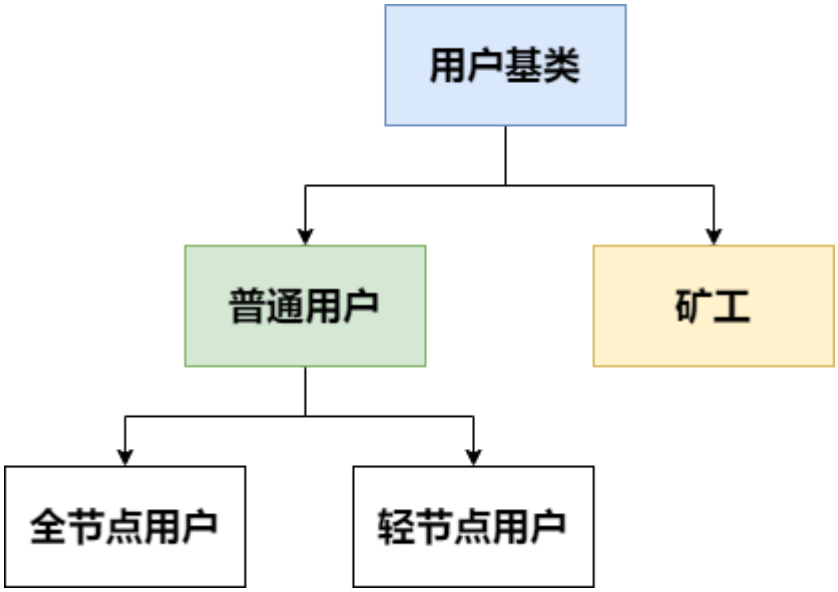
主要构成

该项目主要有以下的部分组成：

1. 账户(accounts)
2. 状态(state)
3. 费用(gas)
4. 交易(transactions)
5. 区块(blocks)
6. 交易执行(transaction execution)
7. 挖矿(mining)
8. 工作量证明(proof of work)

帐户

该项目的全局“共享状态”是有很多小对象来组成的，这些账户可以通过消息传递架构来与对方进行交互。其中分为普通帐户和矿工，普通帐户只具备交易功能，而矿工用户可以挖矿获得报酬。另外，在普通用户中又分为全节点用户和轻节点用户，全节点用户是将所有的区块链进行存储，而轻节点用户只将Merkle树的头节点哈希值进行存储，为什么只存储哈希值呢？这会在后面的部分进行详细讲解。




```
// 用户基类
class User {
  int nonce;         // 执行交易次数
  double balance;    // 持币数量
  string storageRoot;// Merkle树的根节点的值
public:
  User(double bal);
  double GetBalance();         // 得到币
  void ChangeBalance(double b);// 更改币
  void AddNonce();             // 增加交易次数
  string ToString(); // 把所有信息变为string
};
// 完整用户
class FullUser :public User {
  Block* mB;       // 完整区块链
  string name;
public:
  FullUser(double bal,string str);
  void ShowInformation();
  void UpdateBook(Block* nmb);
};
// 轻量级用户
class LightUser :public User {
```

```
  LightBlock* mLB; // 轻量级区块链
  string name;
public:
  LightUser(double bal, string str);
};
// 矿工
class Miner :public User {
  double comp;  // 算力
  Block* mB;    // 存储的区块链
public:
  Miner(double c, double bal);
  double GetComp();
  void UpdateBook(Block* bptr);
};
```

账户状态

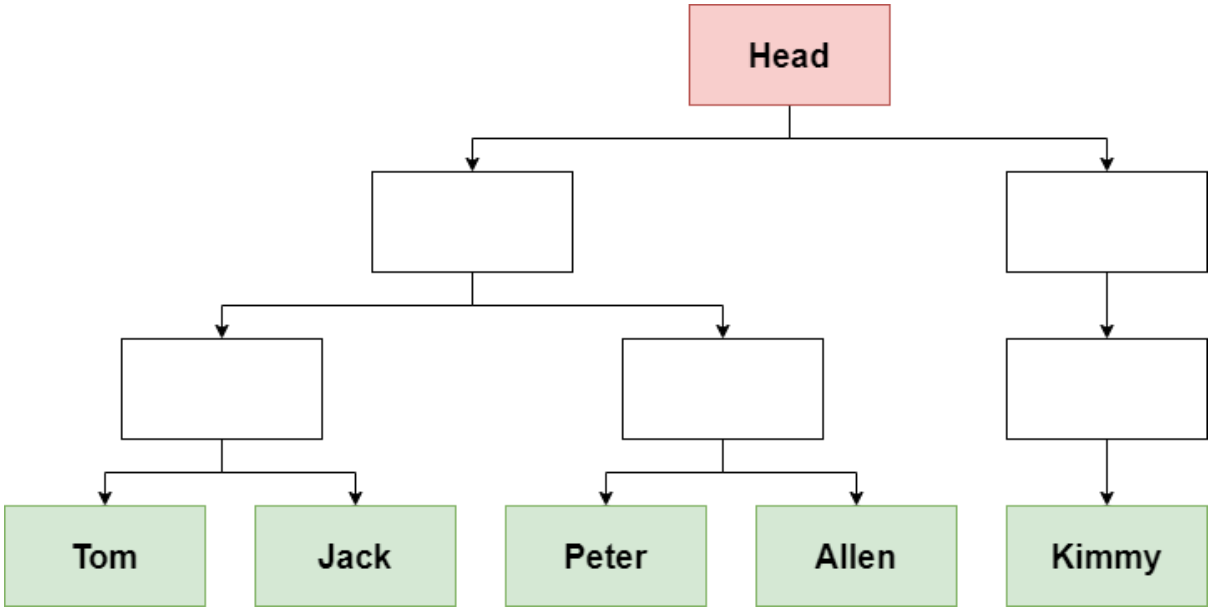
账户状态有三个组成部分，不论账户类型是什么，都存在这三个组成部分：

1. nonce：从此账户地址发送的交易序号。
2. balance： 此地址拥有货币的数量。
3. storageRoot： Merkle树的根节点Hash值。

世界状态

Merkle Tree（也被叫做Merkle trie）是一种由一系列节点组成的二叉树，这些节点包括：

1. 在树的最底层的包含了源数据的大量叶子节点
2. 一系列的中间的节点，这些节点是两个子节点的Hash值
3. 一个根节点，同样是两个子节点的Hash值，代表着整棵树




```
// Merkle
template<class T>
class MerkleTree{
  vector<T> Vdata;                // 存放数据
  vector<vector<string> > MT;     // 存放Merkle树结构
public:
  MerkleTree();                         // 初始化
  MerkleTree(MerkleTree &M);            // 拷贝构造
  void Insert(T data);                  // 插入
  void Change(int indx, T newData);     // 改变节点值
  void Update();                        // 更新
  string GetHead();                     // 返回头节点哈希
  int GetSize();                        // 返回节点个数
};
```

同样的树结构也用来存储交易和物品信息。更具体的说，每个块都保存了三个不同Merkle树结构的根节点的Hash，包括：

1. 状态树
2. 物品树
3. 交易树

```
// 完整区块
class Block {
  int t;                        // 时间
  MerkleTree<User> MerUser;     // 用户树
  MerkleTree<Stuff> MerStuff; // 物品树
  MerkleTree<Trade> MerTrade; // 交易树
  Block* before;
public:
```

```
  Block() { t = 0; before = NULL; };
  Block(Block& b2);           // 深拷贝
  void SetTime(int ti);       // 设置时间
  void InputUser(User u);     // 插入用户
  string GetUserHead();       // 返回用户树头节点哈希
  void InputStuff(Stuff s);   // 插入物品
  string GetStuffHaed();      // 得到物品树头节点哈希
  void InputTrade(Trade t);   // 插入交易
  string GetTradeHead();      // 得到交易树头节点哈希
  void SetBefore(Block* bptr);// 写入前节点
};
```

在Merkle树中存储所有信息的高效性在轻节点中相当的有用。区块链就是一群节点来维持的。广泛的说，有两种节点类型：全节点和轻节点。

全节点通过下载整条链来进行同步，从创世纪块到当前块，执行其中包含的所有交易。矿工会存储全节点，因为他们在挖矿过程中需要全节点。也有可能下载一个全节点而不用执行所有的交易。无论如何，一个全节点包含了整个链。

不过除非一个节点需要执行所有的交易或轻松访问历史数据，不然没必要保存整条链。这就是轻节点概念的来源。比起下载和存储整个链以及执行其中所有的交易，轻节点仅仅下载链的头，从创世纪块到当前块的头，不执行任何的交易或检索任何相关联的状态。由于轻节点可以访问块的头，而头中包含了3个树的Hash，所有轻节点依然可以很容易生成和接收关于交易、事件、余额等可验证的答案。

这个可以行的通是因为在Merkle树中hash值是向上传播的—如果一个恶意用户试图用一个假交易来交换Merkle树底的交易，这个会改变它上面节点的hash值，而它上面节点的值的改变也会导致上上一个节点Hash值的改变，以此类推，一直到树的根节点。

物品

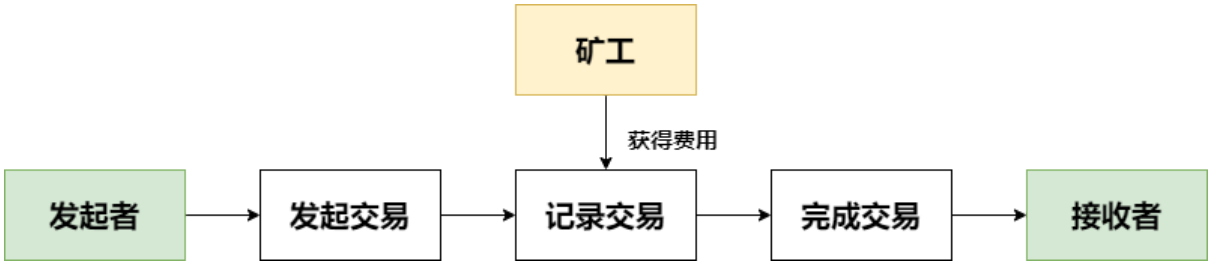
物品的设计结构如下：

```
// 物品
class Stuff {
  int code;         // 唯一编号（Merkle树位置）
  string name;      // 名字
  vector<string> Info; // 产品链
public:
  Stuff(int c, string n);
  Stuff(int c, string n, int u, int t);// 用户创建物品
  void AddInfo(string str);            // 添加信息
  void ShowAllInfo();                  // 展示所有信息
  void ShowBriefInfo();                // 展示概要
  string ToString();
};
```

用户可以创建物品、对物品进行加工，同时用户可以交易该物品到其他用户，在该项目中每一步操作都会被记录到产品信息中，同时会保存在区块链之中，这确保了物品的可溯源性。

费用

区块链可以运作的一个重要方面就是每个网络执行的操作同时也被全节点所影响。然而，计算的操是非常昂贵的。因此，最好是用来执行最简单的任务，比如运行一个简单的业务逻辑或者验证签名和其他密码对象，而不是用于复杂的操作，比如文件存储，电子邮件，或机器学习，这些会给网络造成压力。施加费用防止用户使网络超负荷。



区块

所有的交易都被组成一个”块”。一个区块链包含了一系列这样的链在一起区块。

在该项目中，一个区块包含：

1. 区块头
2. 关于包含在此区块中交易集的信息

轻量级区块

轻节点中只存储了三个Merkle树的哈希值，这样可以大大缩减存储空间。

```
// 轻量级区块
class LightBlock {
  int t;             // 时间
  string LightUser;  // 用户
  string LightStuff; // 物品
  string LightTrade; // 交易
  LightBlock* before;
public:
  LightBlock() { t = 0; before = NULL; };
  LightBlock(LightBlock& lb2);
  void SetTime(int ti);
};
```

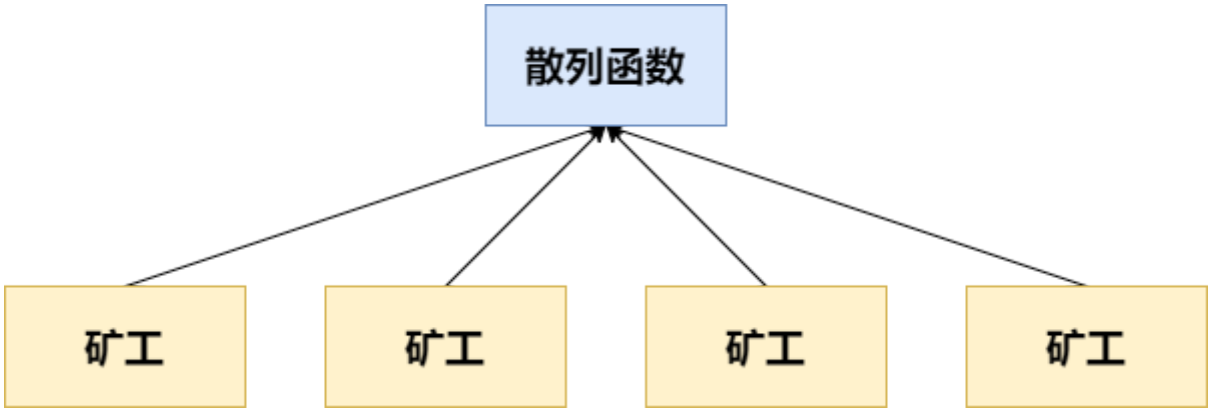
工作量证明挖矿

工作量证明（Proof-of-Work）是一种对应服务与资源滥用、或是拒绝服务攻击的经济对策。

一般要求用户进行一些耗时适当的复杂运算，并且答案能被服务方快速验算，以此耗用的时间、设备与能源做为担保成本，以确保服务与资源是被真正的需求所使用。此概念最早由Cynthia Dwork和Moni Naor于1993年的学术论文提出，而工作量证明一词则是在1999年由Markus Jakobsson与Ari Juels所发表。现时此技术成为了加密货币的主流共识机制之一，如比特币所采用的技术。

工作量证明最常用的技术原理是散列函数。由于输入散列函数h的任意值n，会对应到一个结果$h(n)$，而$h(n)$只要变动一个比特，就会引起雪崩效应，所以几乎无法从$h(n)$反推回n，因此借由指定查找n的特征，让用户进行大量的穷举运算，就可以达成工作量证明。

本项目并没有真正的进行求解MD5哈希的运算，而是使用了模拟的方法。创建若干个数的矿工，每个矿工都有自己的算力，使用随机方法，算力高的矿工有更高的可能求解出答案。



挖矿作为安全机制

总的来说，PoW的目的就是以加密安全的方式证明生成的一些输出是经过了一定量的计算的。因为除了列举所有的可能性，没有更好的其他方法来找到一个低于要求阈值的nonce。重复应用Hash函数的输出均匀分布，所以我们可以确保，在平均值上，找到满足要求的nonce所需时间取决于难度阈值。难度系数越大，所需时间越长。这样的话，PoW算法就给予难度这个概念的意义了：用来加强区块链的安全。

我们所说的区块链的安全又是什么意思？这非常简单：我们想要创造一个每个人都信任的区块链。如果存在超过1条以上的链，用户的信任就会消失，因为他们没有能力合

理的确认哪条链才是“有效的”。为了让一群用户接受存储在区块链中的潜在状态，我们需要有一群人信任的一个权威区块链。

这完完全全就是Pow算法所做的事情：它确保特定的区块链直到未来都一直保持着权威性，让攻击者创造一个新区块来重写某个历史部分（例如清除一个交易或者创建一个假的交易）或者保持一个分叉变得非常困难。为了首先让他们的区块被验证，攻击者需要总是比网络上的其他人要更快的解决掉nonce问题，这样网络就会相信他们的链是最重的链（基于提到的GHOST协议原则）。除非攻击者拥有超过一半的网络挖矿能力，要不然这基本上是不可能的。

挖矿作为财富分配机制

除了提供一个安全的区块链，PoW同样也是分配财富给那些为提供这个安全而花费自己计算力的人的一种方法。一个矿工挖出一个区块的时候会获得奖励，包括：

1. 为“获胜”区块提供的静态区块奖励
2. 区块中的交易在区块内所消耗的gas

系统测试

创建两个用户Jack和Tom：

```
------------------------Full User--------------------------
0 Tom 10
1 Jack 10
-----------------------------------------------------------
```

系统信息：

```
---------------------Information---------------------------
Time: 1
User number: 2
FullUser number: 2
LightUser number: 0
Miner number: 0
Stuff number: 0
Trade number: 0
User Head hash: b01d1bf39268cff383d0909b5afab42a
Stuff Head hash:
Trade Head hash:
The computing power of the whole network: 0
-----------------------------------------------------------
```

用户Tom创建一个物品Book：

```
------------------------Full User--------------------------
0 Tom 10
1 jack 10
-----------------------------------------------------------
--------------------------Stuff----------------------------
0 book
-----------------------------------------------------------
```

用户Tom加工物品Book，查看产品信息：

```
------------------------Stuff book-------------------------
t = 1 user0 create this stuff.
t = 1 user0 Chage It Become Red.
-----------------------------------------------------------
```

用户Tom以0.5gas交易产品至Jack，交易之后用户的信息：

```
------------------------Full User--------------------------
0 Tom 9.5
1 jack 10
-----------------------------------------------------------
--------------------------Stuff----------------------------
0 book
-----------------------------------------------------------
```

此时的区块信息：

```
---------------------Information---------------------------
Time: 2
User number: 2
FullUser number: 2
LightUser number: 0
Miner number: 0
Stuff number: 1
Trade number: 1
User Head hash: 2cfa41c5b19e48583f40d68dd95ce2bb
Stuff Head hash: 02713a11488524e12a11b3085df3c982
Trade Head hash: e6d915807c01280d50d5e18681472780
The computing power of the whole network: 0
-----------------------------------------------------------
```

加入10个随机矿工：

```
---------------------Information---------------------------
Time: 3
```

```
User number: 12
FullUser number: 2
LightUser number: 0
Miner number: 10
Stuff number: 1
Trade number: 1
User Head hash: 2cfa41c5b19e48583f40d68dd95ce2bb
Stuff Head hash: 02713a11488524e12a11b3085df3c982
Trade Head hash: e6d915807c01280d50d5e18681472780
The computing power of the whole network: 4.77807
------------------------------------------------------------
```

用户Jack加工产品Book，最终产品溯源信息：

```
------------------------Stuff book-------------------------
t = 1 user0 create this stuff.
t = 1 user0 Chage It Become Red.
t = 1 user0 trade this to user1 with gas 0.500000.
t = 3 user1 Add A Cover.
------------------------------------------------------------
```

思考问题

区块链能真正做到去中心化吗？

我认为很难做到真正的去中性化，区块链在设计上可以完美的实现去中心化，但是在实际过程中会产生很多的问题。世界上大部分的加密货币交易都是通过Coinbase/GDax、Binance、Bittrex 等中心化交易所完成的，还有一些钱包服务不是去中心化的。我们设计去中心化的基础设施。然而，我们生活在一个中心化的世界。整个数据驻留在集中的数据源上。当我们创建一个新的应用程序时，我们不能后悔以前的数据。那么我们如何设计一个触及中心化资源的去中心化系统呢？我觉得真正实现去中心化还是非常困难的。

区块链是分布式数据库吗？

从形式上看，区块链与预写式日志在设计原理上是高度一致的。WAL是数据库的核心数据结构，记录了从数据库创建之初到当前时刻的所有变更，用于实现主从复制、备份回滚、故障恢复等功能。如果保留了全量的WAL日志，就可以从起点回放WAL，时间旅行到任意时刻的状态，如PostgreSQL的PITR。区块链其实就是这样一份日志，它记录了从创世以来的每笔Transaction。回放日志就可以还原数据库任意时刻的状态。所以区块链当然可以算作某种意义上的数据库，支持数据库有检索。

区块链真的不可篡改吗？

我认为区块链是不可篡改的。区块链的每一个节点都保留有交易记录，这样如果你想修改记录，别的人都可以拿出自己的备份来证明你作弊。这样你就会被踢出网络其实区块链就是一个无法篡改的数据库。就是说一个无法修改的数据记录账本，即区块链上的交易记录是无法篡改的。对于传统数据库而言，几乎都是CRUD数据库，其中四项基本操作为:Create、Read、Update、Delete。而在现在我们所说的区块链数据库中，则是使用CRW数据库，其基本操作只有三个：Create、Read、Write。就是区块链数据库中没有了Update（更新）、Delete（删除）这两步，这正是区块链技术的重大创新之处。没有了“更新”这一步，就意味着被记录到数据库中的数据，无法被修改。没有“删除”这一步，意味着被上传到数据库中的数据无法被擦除。