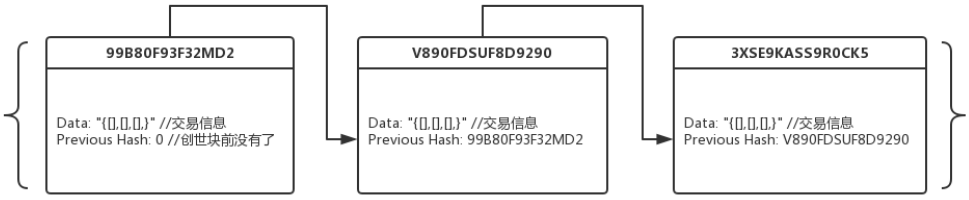
Java实现简单区块链 1

读完了 <区块链 领导干部读本> , 以及看了一部分<区块链原理,设计与应用>,
通过对区块链的一些了解跟随网络博客完成demo

创建区块链
区块链就是一串或者是一系列区块的集合,类似于链表的概念,每个区块都指向于后面一个区块,然后顺序的连接在一起. 那么每个区块中内容是? 区块链中的每一个区块都存放了很多有价值的信息, 只要包括3个部分 : 自己的数字签名, 上一个区块的数字签名, 还有一切需要加密的数据 (这些数据在比特币中就相当于是交易的信息,它是加密货币的本质). 每个数字签名不但证明了自己是特有的一个区块, 而且指向了前一个区块的来源,让所有的区块在链条中可以串起来,而数据就是一些特定的信息, 你可以按照业务逻辑来保存业务数据.



这里的hash指的就是数字签名
所以每一个区块不仅包含前一个区块的hash值, 同时包含自身的一个hash值, 自身的hash值是通过之前的hash值和数据data通过hash计算出来的. 如果前一个区块的数据一旦被篡改了, 那么前一个区块的hash值也会同样发生变化 (因为数据也被计算在内) ,这样也就导致了所有后续的区块中的hash值, 所以计算和对比hash值会让我们检查到当前的区块链是否是有效的, 也就避免了数据被恶意篡改的可能性, 因为篡改数据就会改变hash值并破坏整个区块链.

定义区块链的类Block:

```
package com.sha256.sha256.bean;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

Data
NoArgsConstructor
@AllArgsConstructor
public class Block {
```

```
    private String hash; // our signature
    private String previousHash; // the hash of previous block
    private String data; //our data will be a simple message.
    private long timeStamp; //as number of milliseconds since 1/1/1970.

    //Block Constructor
    public Block(String data,String previousHash){
        this.data = data;
        this.previousHash = previousHash;
        this.timeStamp = new Date().getTime();
    }
}
```

String hash是我们的数字签名, 变量previousHash保存前一个区块的hash值, String data是保存我们区块的数据(比如交易转账信息).

创建数字签名

熟悉加密算法的朋友,Java方式可以实现的加密方式很多, 例如BASE, MD, RSA ,SHA 等等, 我在这里选用了SHA256这种加密方式, SHA (Secure Hash Algorithm) 安全散列算法, 这种算法的特点是数据的少量更改会在Hash值中产生不可预知的大量更改, hash值用作表示大量数据的固定大小的唯一值, 而SHA256算法的hash值大小为256位. 之所以选用SHA256是因为它的大小正合适, 一方面产生重复hash值的可能性很小, 另一方面在区块链实际应用过程中, 有可能会产生大量的区块, 而使得信息量很大, 那么256位的大小就比较恰当了.

第5章　密码学与安全技术

工程领域从来没有黑科技；密码学不仅是工程。

密码学相关的安全技术在整个信息技术领域的重要地位无需多言。如果没有现代密码学和信息安全的研究成果，人类社会根本无法进入信息时代。区块链技术大量依赖了密码学和安全技术的研究成果。

实际上，密码学和安全领域所涉及的知识体系十分繁杂，本章将介绍密码学领域中跟区块链相关的一些基础知识，包括Hash算法与数字摘要、加密算法、数字签名、数字证书、PKI体系、Merkle树、布隆过滤器、同态加密等。读者通过阅读本章可以了解如何使用这些技术保护信息的机密性、完整性、认证性和不可抵赖性。

> 节选<区块链原理,设计与应用>

测试SHA256加密:

```
package com.sha256.sha256.test;
```

```
import com.sha256.sha256.utils.SHA256Util;

public class TestSHA256 {

    public static void main(String[] args) {
        String message0 = "我是要被加密的信息";
        String message1 = "我是要被加密的信息";
        String message2 = "我是要被加密的信息.";
        String encryptionMessage0 = SHA256Util.applySha256(message0);
        String encryptionMessage1 = SHA256Util.applySha256(message1);
        String encryptionMessage2 = SHA256Util.applySha256(message2);
        System.out.println(encryptionMessage0);
        System.out.println(encryptionMessage1);
        System.out.println(encryptionMessage2);
    }
}


>>>>>输出:
2d7641299aba44f11e8b567dc55f9a45c5218e20bdb65d1306020bfb09fe2f31
2d7641299aba44f11e8b567dc55f9a45c5218e20bdb65d1306020bfb09fe2f31
2a6588b9fd3b412176b4cf499c23f1aa06b35843e6082ca0ab2227f4129bc805
```

Hash算法与数字摘要:
Hash定义:
Hash(哈希或散列)算法是非常基础也非常重要的计算机算法,它能将任意长度的二进制明文串映射为较短的(通常是固定长度的)二进制串(Hash值), 并且不同的明文很难映射为相同的Hash值.

这意味着对于某个文件,无需查看其内容,只要其SHA-256 Hash计算后结果同样为:
2d7641299aba44f11e8b567dc55f9a45c5218e20bdb65d1306020bfb09fe2f31,
则说明文件内容极大概率上就是 –> 我是要被加密的信息 几个字.

Hash值在应用中又常被称为指纹(fingerprint)或摘要(digest). Hash算法的核心思想也经常被应用到基于内容的编址或命名算法中.

一个优秀的Hash算法将能实现如下功能:

- 正向快速 : 给定明文和Hash算法, 在有限时间和有限资源内能计算得到Hash值.
- 逆向困难 : 给定(若干)Hash值, 在有限时间内很难(基本不可能)逆推出明文;
- 输入敏感 : 原始输入信息发生任何改变,新产生的Hahs值都应该出现很大不同;(见上面的三个字符串的比较)
- 冲突避免 : 很难找到两端内容不同的明文,使得它们的Hash值一致(发生碰撞).

冲突避免有时候又称为"抗碰撞性", 分为"弱抗碰撞性"和"强抗碰撞性". 如果给定明文前提下, 无法找到与之碰撞的其他明文, 则算法具有"弱抗碰撞性", 如果无法找到任意两个发生Hash碰撞的明文, 则称算法具有"强抗碰撞性".

很多场景下,也往往要求算法对于任意长的输入内容,可以输入定长的Hash值结果.

常见算法

目前常见的Hash算法包括MD5和SHA系列算法。

MD4（RFC 1320）是MIT的Ronald L.Rivest在1990年设计的，MD是Message Digest的缩写。其输出为128位。MD4已被证明不够安全。

MD5（RFC 1321）是Rivest于1991年对MD4的改进版本。它对输入仍以512位进行分组，其输出是128位。MD5比MD4更加安全，但过程更加复杂，计算速度要慢一点。MD5已被证明不具备“强抗碰撞性”。

SHA（Secure Hash Algorithm）并非一个算法，而是一个Hash函数族。NIST（National Institute of Standards and Technology）于1993年发布其首个实现。目前知名的SHA-1算法在1995年面世，它的输出为长度160位的Hash值，抗穷举性更好。SHA-1设计时模仿了MD4算法，采用了类似原理。SHA-1已被证明不具备“强抗碰撞性”。

为了提高安全性，NIST还设计出了SHA-224、SHA-256、SHA-384和SHA-512算法（统称为SHA-2），跟SHA-1算法原理类似。SHA-3相关算法也已被提出。

目前，MD5和SHA1已经被破解，一般推荐至少使用SHA2-256或更安全的算法。

提示: MD5是一个经典的Hash算法,和SHA-1算法一起都被认为安全性已不足应用于商业场景.

调用工具类的SHA256算法:

```
package com.sha256.sha256.utils;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA256Util {
    //Applies SHA256 to a string and returns the result
    //SHA256 encryption
    public static String applySha256(String input){
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            //Applies sha256 to our input
            byte[] hash = digest.digest(input.getBytes("UTF-8"));
            //This will contain hash as hexidecimal
            StringBuffer hexString = new StringBuffer();
            for(int i=0;i<hash.length;i++){
                String hex = Integer.toHexString(0xff & hash[i]);
                if(hex.length()==1){
                    hexString.append('0');
                }
                hexString.append(hex);
            }
```

```
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

强化Block实体类:
对hash值进行赋值:

```
package com.sha256.sha256.bean;

import com.sha256.sha256.utils.SHA256Util;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Block {

    private String hash; // our signature
    private String previousHash; // the hash of previous block
    private String data; //our data will be a simple message.
    private long timeStamp; //as number of milliseconds since 1/1/1970.

    //Block Constructor
    public Block(String data,String previousHash){
        this.data = data;
        this.previousHash = previousHash;
        this.timeStamp = new Date().getTime();
        this.hash = SHA256Util.calculateHash(this); //Making sure we do t
his after we set the other values.
    }
}
```

同时在工具类中:
加入新的方法 calculateHash:

```
package com.sha256.sha256.utils;

import com.sha256.sha256.bean.Block;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA256Util {
    //Applies SHA256 to a string and returns the result
    //SHA256 encryption
    public static String applySha256(String input){
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            //Applies sha256 to our input
            byte[] hash = digest.digest(input.getBytes("UTF-8"));
            //This will contain hash as hexidecimal
            StringBuffer hexString = new StringBuffer();
            for(int i=0;i<hash.length;i++){
                String hex = Integer.toHexString(0xff & hash[i]);
                if(hex.length()==1){
                    hexString.append('0');
                }
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            return null;
        }
    }

    //calculate the hash use previoushash , timestamp , data
    public static String calculateHash(Block block){
        String calculateHash = SHA256Util.applySha256(block.getPreviousHa
sh() + Long.toString(block.getTimeStamp()) + block.getData());
        return calculateHash;
    }

}
```

测试:

```
package com.sha256.sha256.test;

import com.sha256.sha256.bean.Block;
```

```
import com.sha256.sha256.utils.SHA256Util;

public class TestSHA256 {

    public static void main(String[] args) {
        //test1 测试三个被加密字符串 加密后的hash值的差别
        /**
         * 虽然第三条信息仅仅多一个".",但加密后的数据hash相差极大
         */
        String message0 = "我是要被加密的信息";
        String message1 = "我是要被加密的信息";
        String message2 = "我是要被加密的信息.";
        String encryptionMessage0 = SHA256Util.applySha256(message0);
        String encryptionMessage1 = SHA256Util.applySha256(message1);
        String encryptionMessage2 = SHA256Util.applySha256(message2);
        System.out.println(encryptionMessage0);
        System.out.println(encryptionMessage1);
        System.out.println(encryptionMessage2);

        //创建区块链逻辑, 因为第一个块没有上一个块的hash头部值,所以输入0 作为前一个
块的previous hash
        /**
         * 由于在{@link SHA256Util#calculateHash(Block)}
         * 中对同时产生的new Date().getTime() (timestamp)
         * 也加入进行了hash加密,所以固有的message (data)及
         * previoushash之和进行了加密.
         */
        Block genesisBlock = new Block("这是第一个区块中的要被加密的信息和交易信
息","0");
        String hash1 = genesisBlock.getHash();
        System.out.println("Hash for block 1 : "+hash1);

        Block secondBlock = new Block("这是第二个区块,以及其中信息!!!它的前区块
头部hash我们拿上一个的来使用",hash1);
        String hash2 = secondBlock.getHash(); //
        System.out.println("Hash for block 2 : "+hash2);

        Block thirdBlock = new Block("这是第三个区块,它的hash应该已经被前两个的
信息纳入进来了,它的hash如果对不上,那么说明前面的信息被改动过了",hash2);
        String hash3 = thirdBlock.getHash();
        System.out.println("Hash for block 3 : "+hash3);

    }
}
```

运行结果:

* 由于在{@link SHA256Util#calculateHash(Block)}
* 中对同时产生的new Date().getTime() (timestamp)
* 也加入进行了hash加密,所以固有的message (data)及
* previoushash之和进行了加密.

```
2d7641299aba44f11e8b567dc55f9a45c5218e20bdb65d1306020bfb09fe2f31
2d7641299aba44f11e8b567dc55f9a45c5218e20bdb65d1306020bfb09fe2f31
2a6588b9fd3b412176b4cf499c23f1aa06b35843e6082ca0ab2227f4129bc805
Hash for block 1 : cdb1bb85e8f2394f3cee57d82800f5413848fa6c981fefa0fd2044
97f853c8b4
Hash for block 2 : fad4bc33a9b9f5fc5053fe3583b6bf366be9ea518936ce37d58b91
6e2c4699be
Hash for block 3 : 558ff9aac60aea20da1936a78a863195cbe23748f08fa34219bb3a
bc66078b65
```

注意: 每次 Hash for block * 的产生的值是不同的,因为每次对timestamp进行了计算

每一个区块都必须要有自己的数据签名即hash值,这个hash值依赖于自身的信息(data)和上一个区块的数字签名(previousHash), 但这个还不是区块链, 下面让我们存储区块到数组中, 这里我会引入gson包, 目的是可以用json方式查看整个一条区块链结构.
看test3

```
package com.sha256.sha256.test;

import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.google.gson.GsonBuilder;
import com.sha256.sha256.bean.Block;
import com.sha256.sha256.utils.SHA256Util;

import java.util.ArrayList;

public class TestSHA256 {

    //声明一个区块链,用于添加Block实体
    public static ArrayList<Block> blockChain = new ArrayList<>();

    public static void main(String[] args) {
        //test1 测试三个被加密字符串 加密后的hash值的差别
        /**
         * 虽然第三条信息仅仅多一个".",但加密后的数据hash相差极大
         */
        String message0 = "我是要被加密的信息";
        String message1 = "我是要被加密的信息";
        String message2 = "我是要被加密的信息.";
        String encryptionMessage0 = SHA256Util.applySha256(message0);
        String encryptionMessage1 = SHA256Util.applySha256(message1);
        String encryptionMessage2 = SHA256Util.applySha256(message2);
        System.out.println(encryptionMessage0);
        System.out.println(encryptionMessage1);
        System.out.println(encryptionMessage2);

        //test2 创建区块链逻辑, 因为第一个块没有上一个块的hash头部值,所以输入0 作为
前一个块的previous hash
        /**
```

```
         * 由于在{@link SHA256Util#calculateHash(Block)}
         * 中对同时产生的new Date().getTime() (timestamp)
         * 也加入进行了hash加密,所以固有的message (data)及
         * previoushash之和进行了加密.
         */
        Block genesisBlock = new Block("这是第一个区块中的要被加密的信息和交易信息","0");
        String hash1 = genesisBlock.getHash();
        System.out.println("Hash for block 1 : "+hash1);

        Block secondBlock = new Block("这是第二个区块,以及其中信息!!!它的前区块头部hash我们拿上一个的来使用",hash1);
        String hash2 = secondBlock.getHash(); //
        System.out.println("Hash for block 2 : "+hash2);

        Block thirdBlock = new Block("这是第三个区块,它的hash应该已经被前两个的信息纳入进来了,它的hash如果对不上,那么说明前面的信息被改动过了",hash2);
        String hash3 = thirdBlock.getHash();
        System.out.println("Hash for block 3 : "+hash3);

        //test3 add our blocks to the blockchain ArrayList :
        blockChain.add(new Block("区块链上第一小节","0"));
        blockChain.add(new Block("区块链第二小节",blockChain.get(blockChain.size()-1).getHash()));
        blockChain.add(new Block("区块链第三小节",blockChain.get(blockChain.size()-1).getHash()));

//        JSONArray blockChainJson1 = (JSONArray)JSONArray.toJSON(blockChain); //JSONArray是不排版的
//        System.out.println(blockChainJson1);
        String blockChainJson = new GsonBuilder().setPrettyPrinting().create().toJson(blockChain);

        System.out.println(blockChainJson);
    }
}
```

改善,加入区块链的index下标,用来记录第几个区块.

```
package com.sha256.sha256.bean;

import com.sha256.sha256.utils.SHA256Util;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Data
@NoArgsConstructor
```

```
@AllArgsConstructor
public class Block {

    private long index;
    private String hash; // our signature
    private String previousHash; // the hash of previous block
    private String data; //our data will be a simple message.
    private long timeStamp; //as number of milliseconds since 1/1/1970.

    //Block Constructor
    public Block(long index,String data,String previousHash){
        this.index = index;
        this.data = data;
        this.previousHash = previousHash;
        this.timeStamp = new Date().getTime();
        this.hash = SHA256Util.calculateHash(this); //Making sure we do t
his after we set the other values.
    }
}
```

改善: 测试循环加入的区块链,最大下标24

```
package com.sha256.sha256.test;

import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.google.gson.GsonBuilder;
import com.sha256.sha256.bean.Block;
import com.sha256.sha256.utils.SHA256Util;

import java.util.ArrayList;
import java.util.Random;

public class TestSHA256 {

    //声明一个区块链,用于添加Block实体
    public static ArrayList<Block> blockChain = new ArrayList<>();

    public static void main(String[] args) {
        //test1 测试三个被加密字符串 加密后的hash值的差别
        /**
         * 虽然第三条信息仅仅多一个".",但加密后的数据hash相差极大
         */
//        String message0 = "我是要被加密的信息";
//        String message1 = "我是要被加密的信息";
//        String message2 = "我是要被加密的信息.";
//        String encryptionMessage0 = SHA256Util.applySha256(message0);
//        String encryptionMessage1 = SHA256Util.applySha256(message1);
//        String encryptionMessage2 = SHA256Util.applySha256(message2);
//        System.out.println(encryptionMessage0);
```

```
//        System.out.println(encryptionMessage1);
//        System.out.println(encryptionMessage2);
//
//        //test2 创建区块链逻辑, 因为第一个块没有上一个块的hash头部值,所以输入0 作为前一个块的previous hash
//        /**
//         * 由于在{@link SHA256Util#calculateHash(Block)}
//         * 中对同时产生的new Date().getTime() (timestamp)
//         * 也加入进行了hash加密,所以固有的message (data)及
//         * previoushash之和进行了加密.
//         */
//        Block genesisBlock = new Block(0,"这是第一个区块中的要被加密的信息和交易信息","0");
//        String hash1 = genesisBlock.getHash();
//        System.out.println("Hash for block 1 : "+hash1);
//
//        Block secondBlock = new Block(1,"这是第二个区块,以及其中信息!!!它的前区块头部hash我们拿上一个的来使用",hash1);
//        String hash2 = secondBlock.getHash(); //
//        System.out.println("Hash for block 2 : "+hash2);
//
//        Block thirdBlock = new Block(2,"这是第三个区块,它的hash应该已经被前两个的信息纳入进来了,它的hash如果对不上,那么说明前面的信息被改动过了",hash2);
//        String hash3 = thirdBlock.getHash();
//        System.out.println("Hash for block 3 : "+hash3);
//
////        test3 add our blocks to the blockchain ArrayList :
//        blockChain.add(new Block(0,"区块链上第一小节","0"));
//        blockChain.add(new Block(1,"区块链第二小节",blockChain.get(blockChain.size()-1).getHash()));
//        blockChain.add(new Block(2,"区块链第三小节",blockChain.get(blockChain.size()-1).getHash()));
        int chainNumber = 24;
        int index = 0;
        while (chainNumber > 0) {
            System.out.println("blockChain.size():" + blockChain.size());
            if (blockChain.size() == 0) {
                blockChain.add(new Block(0, "创世块", "0"));
            }
            index++;
            blockChain.add(new Block(index, "区块内容" + blockChainMessage(index), blockChain.get(blockChain.size() - 1).getHash()));
            chainNumber--;
        }

//        JSONArray blockChainJson1 = (JSONArray)JSONArray.toJSON(blockChain); //JSONArray是不排版的
//        System.out.println(blockChainJson1);
        String blockChainJson = new GsonBuilder().setPrettyPrinting().create().toJson(blockChain);

        System.out.println(blockChainJson);
```

```
    }


    //模拟一些交易信息
    private static String blockChainMessage(int getNumber) {
        Random random = new Random(getNumber);
        long l = random.nextLong();
        System.out.println("blockChainMessage:" + l);
        return String.valueOf(l);
    }
}
```

输出:


```
[
  {
    "index": 0,
    "hash": "cdfb9b95804568519f7b77783d3ace883c5310ca9aa25e1f30be1394e306
8065",
    "previousHash": "0",
    "data": "创世块",
    "timeStamp": 1564793820934
  },
  {
    "index": 1,
    "hash": "40e456dbce2531668337c72454300a5401dbabe6aa1fa5e791fe5f32c8e0
a180",
    "previousHash": "cdfb9b95804568519f7b77783d3ace883c5310ca9aa25e1f30be
1394e3068065",
    "data": "区块内容-49644209488930660 24",
    "timeStamp": 1564793820946
  },
  {
    "index": 2,
    "hash": "3b7174358b7458ab3c545bbde5e21e634b0fec9a440d790e2c76b0f828a0
2bc3",
    "previousHash": "40e456dbce2531668337c72454300a5401dbabe6aa1fa5e791fe
5f32c8e0a180",
    "data": "区块内容-49594634992430136 40",
    "timeStamp": 1564793820946
  },
  {
    "index": 3,
    "hash": "76bd7513fd9a03a8977d9e0b7f08cbd4a22436e9f1164b1732917af37ab7
dfef",
    "previousHash": "3b7174358b7458ab3c545bbde5e21e634b0fec9a440d790e2c76
b0f828a02bc3",
    "data": "区块内容-49611159867546650 64",
    "timeStamp": 1564793820947
  },
```

```
  {
    "index": 4,
    "hash": "9b2bb99b8f67b8f3e50bb0083481a63ca3cd922b0c12b8d900aff0f13820
f252",
    "previousHash": "76bd7513fd9a03a8977d9e0b7f08cbd4a22436e9f1164b173291
7af37ab7dfef",
    "data": "区块内容-49693784028380857O4",
    "timeStamp": 1564793820947
  },
  {
    "index": 5,
    "hash": "5b5c6759585f513a4f8c0cbb6932d4ed0d720de6c65789f8b3d5d4832dbe
779d",
    "previousHash": "9b2bb99b8f67b8f3e50bb0083481a63ca3cd922b0c12b8d900af
f0f13820f252",
    "data": "区块内容-49710308860547698 32",
    "timeStamp": 1564793820947
  },
  {
    "index": 6,
    "hash": "7aa770343259bf62951d94a834dc8a3dbb470b638b85375d395118b7fa2e
d428",
    "previousHash": "5b5c6759585f513a4f8c0cbb6932d4ed0d720de6c65789f8b3d5
d4832dbe779d",
    "data": "区块内容-49660734321097501 52",
    "timeStamp": 1564793820947
  },
  {
    "index": 7,
    "hash": "0a5b5e9ca3fa4dca4a0d85dd3e1123693b5d4d2a1b1c14a53f3aec8c443e
812b",
    "previousHash": "7aa770343259bf62951d94a834dc8a3dbb470b638b85375d3951
18b7fa2ed428",
    "data": "区块内容-49677259196214015 76",
    "timeStamp": 1564793820947
  },
  {
    "index": 8,
    "hash": "04b44886cfa92848abd979eb922fd5bfe8ee8eef15d4e6e44f6173921183
d590",
    "previousHash": "0a5b5e9ca3fa4dca4a0d85dd3e1123693b5d4d2a1b1c14a53f3a
ec8c443e812b",
    "data": "区块内容-49759883399997895 12",
    "timeStamp": 1564793820947
  },
  {
    "index": 9,
    "hash": "8e6dba359895255b0d19b1427ab684e5727608d98117e93442670a2641b1
6ca2",
    "previousHash": "04b44886cfa92848abd979eb922fd5bfe8ee8eef15d4e6e44f61
73921183d590",
    "data": "区块内容-49776408232164736 40",
```

```
    "timeStamp": 1564793820947
  },
  {
    "index": 10,
    "hash": "a634302bad568a7d3f7361b8bdb258e6f67eef0b177102b28ea3cdca37b2
d044",
    "previousHash": "8e6dba359895255b0d19b1427ab684e5727608d98117e9344267
0a2641b16ca2",
Disconnected from the target VM, address: '127.0.0.1:51122', transport:
'socket'
    "data": "区块内容-49726833692714539 60",
    "timeStamp": 1564793820947
  },
  {
    "index": 11,
    "hash": "4aae9efccbe1fae9e8cc36727ba61d9666531b5ba3fdae8ec094fb89ba01
3c27",
    "previousHash": "a634302bad568a7d3f7361b8bdb258e6f67eef0b177102b28ea3
cdca37b2d044",
    "data": "区块内容-4974335856783105384",
    "timeStamp": 1564793820948
  },
  {
    "index": 12,
    "hash": "1543f183edcf1b6b8e7be35a64f354f9dcd92b8287d9f29c3365ed504186
876c",
    "previousHash": "4aae9efccbe1fae9e8cc36727ba61d9666531b5ba3fdae8ec094
fb89ba013c27",
    "data": "区块内容-49825982728665260 24",
    "timeStamp": 1564793820948
  },
  {
    "index": 13,
    "hash": "6c45067b6dd2cde2c2d6d61b7d47871a5d1d5ba2e41290b30d520678f981
b4e1",
    "previousHash": "1543f183edcf1b6b8e7be35a64f354f9dcd92b8287d9f29c3365
ed504186876c",
    "data": "区块内容-49842507560832101 52",
    "timeStamp": 1564793820948
  },
  {
    "index": 14,
    "hash": "9826c226a0c1278315a0a01f4ffdb9ee6a40419e3e49a73024830e5c0d39
4ba0",
    "previousHash": "6c45067b6dd2cde2c2d6d61b7d47871a5d1d5ba2e41290b30d52
0678f981b4e1",
    "data": "区块内容-49792933064331577 68",
    "timeStamp": 1564793820948
  },
  {
    "index": 15,
```

```
    "hash": "f4ed82e810da2e2f09aca7af2c1f21485d6a787d5007090c162595d19b5d
2a76",
    "previousHash": "9826c226a0c1278315a0a01f4ffdb9ee6a40419e3e49a7302483
0e5c0d394ba0",
    "data": "区块内容-4980945789649841896",
    "timeStamp": 1564793820948
  },
  {
    "index": 16,
    "hash": "dbd73819fcec84f9e7ee7fefb92d7d695f2db15546d50ec9bb99085b74a8
f560",
    "previousHash": "f4ed82e810da2e2f09aca7af2c1f21485d6a787d5007090c1625
95d19b5d2a76",
    "data": "区块内容-49363287256195012 55",
    "timeStamp": 1564793820951
  },
  {
    "index": 17,
    "hash": "31c79ce44fbc3eb8a18935455e6bd1612ea66e140a1e965bd6925071f18f
bd49",
    "previousHash": "dbd73819fcec84f9e7ee7fefb92d7d695f2db15546d50ec9bb99
085b74a8f560",
    "data": "区块内容-4937981208836185383",
    "timeStamp": 1564793820951
  },
  {
    "index": 18,
    "hash": "565786f85d117792a2de85199c54a0c6c7c7a484fcceda03575613220371
aec6",
    "previousHash": "31c79ce44fbc3eb8a18935455e6bd1612ea66e140a1e965bd692
5071f18fbd49",
    "data": "区块内容-4933023754891165703",
    "timeStamp": 1564793820951
  },
  {
    "index": 19,
    "hash": "8d60646b234897b980e1438a7d3f0913df8117b4c503fa63363b1ef74057
2371",
    "previousHash": "565786f85d117792a2de85199c54a0c6c7c7a484fcceda035756
13220371aec6",
    "data": "区块内容-4934676238107849831",
    "timeStamp": 1564793820951
  },
  {
    "index": 20,
    "hash": "166364c27c5959f36e602b55da52abe89b5005bcf957f7957fd967c37b9a
3fac",
    "previousHash": "8d60646b234897b980e1438a7d3f0913df8117b4c503fa63363b
1ef740572371",
    "data": "区块内容-49429386627812050 63",
    "timeStamp": 1564793820951
  },
```

```
  {
    "index": 21,
    "hash": "06b8c4cf94f2fe4942d6b1f4ddbc91554c9cbc90b73b6dcabaac1da6b8e5
c99e",
    "previousHash": "166364c27c5959f36e602b55da52abe89b5005bcf957f7957fd9
67c37b9a3fac",
    "data": "区块内容-49445911459978891 91",
    "timeStamp": 1564793820952
  },
  {
    "index": 22,
    "hash": "827b21529d8cd5d533f7b08681a63f54481b11915289057ebd5ab87b3788
bf96",
    "previousHash": "06b8c4cf94f2fe4942d6b1f4ddbc91554c9cbc90b73b6dcabaac
1da6b8e5c99e",
    "data": "区块内容-4939633692052869511",
    "timeStamp": 1564793820952
  },
  {
    "index": 23,
    "hash": "409c8728599af789190d4b7d191b28d7965258d15b2fbf3faa47c74c7d00
b60d",
    "previousHash": "827b21529d8cd5d533f7b08681a63f54481b11915289057ebd5a
b87b3788bf96",
    "data": "区块内容-4941286175269553639",
    "timeStamp": 1564793820952
  },
  {
    "index": 24,
    "hash": "bd51148ae6e686c16136b89f2b1846fd1836de419183e32733201577a6a4
c367",
    "previousHash": "409c8728599af789190d4b7d191b28d7965258d15b2fbf3faa47
c74c7d00b60d",
    "data": "区块内容-4949548595647941576",
    "timeStamp": 1564793820952
  }
]
```


blockChain.get(blockChain.size() - 1).getHash()
这个是获取上一个区块的hash值.
而本块的hash计算是通过: (在Block块实体中的SHA256Util.calculateHash(this))来进行的加密:

```
    //Block Constructor
    public Block(long index,String data,String previousHash){
        this.index = index;
        this.data = data;
        this.previousHash = previousHash;
        this.timeStamp = new Date().getTime();
        this.hash = SHA256Util.calculateHash(this); //Making sure we do t
his after we set the other values.
```

```
    }
```

这时的this已经是赋值好的index,data,previousHash,timeStamp等一系列数据(大概和比特币类似)
再看一下SHA256Util下的加密方法:

```
package com.sha256.sha256.utils;

import com.sha256.sha256.bean.Block;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA256Util {
    //Applies SHA256 to a string and returns the result
    //SHA256 encryption
    public static String applySha256(String input){
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            //Applies sha256 to our input
            byte[] hash = digest.digest(input.getBytes("UTF-8"));
            //This will contain hash as hexidecimal
            StringBuffer hexString = new StringBuffer();
            for(int i=0;i<hash.length;i++){
                String hex = Integer.toHexString(0xff & hash[i]);
                if(hex.length()==1){
                    hexString.append('0');
                }
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
            return null;
        }
    }

    //calculate the hash use previoushash , timestamp , data
    public static String calculateHash(Block block){
        String calculateHash = SHA256Util.applySha256(block.getPreviousHa
sh() + Long.toString(block.getTimeStamp()) + block.getData());
        return calculateHash;
    }

}
```

第一个是applySha256加密方法,是用来对信息进行sha256加密.返回加密值.
第二个calculateHash是对区块本身的一些数据进行加密,其中的返回值calculateHash也是

```
String calculateHash = SHA256Util.applySha256(block.getPreviousHash() + L
ong.toString(block.getTimeStamp()) + block.getData());
```

调用了applySha256加密方法,将区块的 previousHash + 区块的时间戳 + 区块的数据 进行字符串相加, 进行sha256加密后得出最后的加密结果. 作为返回值,并作为了当前区块的 hash 头部值.

2019-8-3