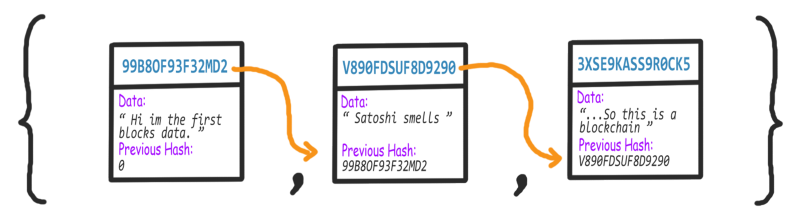
**[java开发区块链只需150行代码](https://www.cnblogs.com/helloworld2018/p/9011369.html)**

本文目的是通过java实战开发教程理解区块链是什么。将通过实战入门学习，用Java自学开发一个很基本的区块链，并在此基础上能扩展如web框架应用等。这个基本的java区块链也实现简单的工作量证明系统。本文用一个java例子,演示了开发一个区块链应用的过程,涉及到所有区块链的概念和基本实现方法。本文阅读对象,主要是希望和即将从事区块链开发的项目架构师。需要指出的是，我们用150行java代码构建的区块链达不到生产级别的，它只是为了帮助你更好的理解区块链的概念。

Java源代码保存在[Github](https://github.com/CryptoKass/NoobChain-Tutorial-Part-1)

## 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。那么每个区块中的内容是什么呢？在区块链中的每一个区块都存放了很多很有价值的信息，主要包括三个部分：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。所以计算和比对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：

import java.util.Date;

public class Block {

public String hash;

public String previousHash;

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();

}

}

正如你可以看到我们的基本块包含String hash，它将保存我们的数字签名。变量previoushash保存前一个块的hash和String data来保存我们的块数据

## 创建数字签名

熟悉加密算法的朋友们，Java方式可以实现的加密方式有很多，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。

下面我创建了一个StringUtil方法来方便调用SHA256算法

import java.security.MessageDigest;

public class StringUtil {

//Applies Sha256 to a string and returns the result.

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"));

StringBuffer hexString = new StringBuffer(); // This will contain hash as hexidecimal

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(Exception e) {

throw new RuntimeException(e);

}

}

}

或许你不完全理解上述代码的含义，但是你只要理解所有的输入调用此方法后均会生成一个独一无二的hash值（数字签名），而这个hash值在区块链中是非常重要的。

接下来让我们在Block类中应用 方法 applySha256 方法，其主要的目的就是计算hash值，我们计算的hash值应该包括区块中所有我们不希望被恶意篡改的数据，在我们上面所列的Block类中就一定包括previousHash，data和timeStamp，

public String calculateHash() {

String calculatedhash = StringUtil.applySha256(

previousHash +

Long.toString(timeStamp) +

data

);

return calculatedhash;

}

然后把这个方法加入到Block的构造函数中去

public Block(String data,String previousHash ) {

this.data = data;

this.previousHash = previousHash;

this.timeStamp = new Date().getTime();

this.hash = calculateHash(); //Making sure we do this after we set the other values.

}

## 测试

在主方法中让我们创建一些区块，并把其hash值打印出来，来看看是否一切都在我们的掌控中。

第一个块称为创世块，因为它是头区块，所以我们只需输入“0”作为前一个块的previous hash。

public class NoobChain {

public static void main(String[] args) {

Block genesisBlock = new Block("Hi im the first block", "0");

System.out.println("Hash for block 1 : " + genesisBlock.hash);

Block secondBlock = new Block("Yo im the second block",genesisBlock.hash);

System.out.println("Hash for block 2 : " + secondBlock.hash);

Block thirdBlock = new Block("Hey im the third block",secondBlock.hash);

System.out.println("Hash for block 3 : " + thirdBlock.hash);

}

}

打印输出结果：

Hash for block 1: f6d1bc5f7b0016eab53ec022db9a5d9e1873ee78513b1c666696e66777fe55fb

Hash for block 2: 6936612b3380660840f22ee6cb8b72ffc01dbca5369f305b92018321d883f4a3

Hash for block 3: f3e58f74b5adbd59a7a1fc68c97055d42e94d33f6c322d87b29ab20d3c959b8f

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。

import java.util.ArrayList;

import com.google.gson.GsonBuilder;

public class NoobChain {

public static ArrayList<Block> blockchain = new ArrayList<Block>();

public static void main(String[] args) {

//add our blocks to the blockchain ArrayList:

blockchain.add(new Block("Hi im the first block", "0"));

blockchain.add(new Block("Yo im the second block",blockchain.get(blockchain.size()-1).hash));

blockchain.add(new Block("Hey im the third block",blockchain.get(blockchain.size()-1).hash));

String blockchainJson = new GsonBuilder().setPrettyPrinting().create().toJson(blockchain);

System.out.println(blockchainJson);

}

}

这样的输出结构就更类似于我们所期待的区块链的样子。

## 检查区块链的完整性

在主方法中增加一个isChainValid()方法，目的是循环区块链中的所有区块并且比较hash值，这个方法用来检查hash值是否是于计算出来的hash值相等，同时previousHash值是否和前一个区块的hash值相等。或许你会产生如下的疑问，我们就在一个主函数中创建区块链中的区块，所以不存在被修改的可能性，但是你要注意的是，区块链中的一个核心概念就是去中心化，每一个区块可能是在网络中的某一个节点中产生的，所以很有可能某个节点把自己节点中的数据修改了，那么根据上述的理论数据改变会导致整个区块链的破裂，也就是区块链就无效了。

public static Boolean isChainValid() {

Block currentBlock;

Block previousBlock;

//loop through blockchain to check hashes:

for(int i=1; i < blockchain.size(); i++) {

currentBlock = blockchain.get(i);

previousBlock = blockchain.get(i-1);

//compare registered hash and calculated hash:

if(!currentBlock.hash.equals(currentBlock.calculateHash()) ){

System.out.println("Current Hashes not equal");

return false;

}

//compare previous hash and registered previous hash

if(!previousBlock.hash.equals(currentBlock.previousHash) ) {

System.out.println("Previous Hashes not equal");

return false;

}

}

return true;

}

任何区块链中区块的一丝一毫改变都会导致这个函数返回false，也就证明了区块链无效了。

在比特币网络中所有的网络节点都分享了它们各自的区块链，然而最长的有效区块链是被全网所统一承认的，如果有人恶意来篡改之前的数据，然后创建一条更长的区块链并全网发布呈现在网络中，我们该怎么办呢？这就涉及到了区块链中另外一个重要的概念工作量证明，这里就不得不提及一下hashcash，这个概念最早来自于Adam Back的一篇论文，主要应用于邮件过滤和比特币中防止双重支付。

## 挖矿

这里我们要求挖矿者做工作量证明，具体的方式是在区块中尝试不同的参数值直到它的hash值是从一系列的0开始的。让我们添加一个名为nonce的int类型以包含在我们的calculatehash（）方法中，以及需要的mineblock（）方法。

import java.util.Date;

public class Block {

public String hash;

public String previousHash;

private String data; //our data will be a simple message.

private long timeStamp; //as number of milliseconds since 1/1/1970.

private int nonce;

//Block Constructor.

public Block(String data,String previousHash ) {

this.data = data;

this.previousHash = previousHash;

this.timeStamp = new Date().getTime();

this.hash = calculateHash(); //Making sure we do this after we set the other values.

}

//Calculate new hash based on blocks contents

public String calculateHash() {

String calculatedhash = StringUtil.applySha256(

previousHash +

Long.toString(timeStamp) +

Integer.toString(nonce) +

data

);

return calculatedhash;

}

public void mineBlock(int difficulty) {

String target = new String(new char[difficulty]).replace('\0', '0'); //Create a string with difficulty \* "0"

while(!hash.substring( 0, difficulty).equals(target)) {

nonce ++;

hash = calculateHash();

}

System.out.println("Block Mined!!! : " + hash);

}

}

mineBlock()方法中引入了一个int值称为difficulty难度，低的难度比如1和2，普通的电脑基本都可以马上计算出来，我的建议是在4-6之间进行测试，普通电脑大概会花费3秒时间，在莱特币中难度大概围绕在442592左右，而在比特币中每一次挖矿都要求大概在10分钟左右，当然根据所有网络中的计算能力，难度也会不断的进行修改。

我们在NoobChain类 中增加difficulty这个静态变量。

public static int difficulty = 5;

这样我们必须修改主方法中让创建每个新区块时必须触发mineBlock()方法，而isChainValid()方法用来检查每个区块的hash值是否正确，整个区块链是否是有效的。

import java.util.ArrayList;

import com.google.gson.GsonBuilder;

public class NoobChain {

public static ArrayList<Block> blockchain = new ArrayList<Block>();

public static int difficulty = 5;

public static void main(String[] args) {

//add our blocks to the blockchain ArrayList:

blockchain.add(new Block("Hi im the first block", "0"));

System.out.println("Trying to Mine block 1... ");

blockchain.get(0).mineBlock(difficulty);

blockchain.add(new Block("Yo im the second block",blockchain.get(blockchain.size()-1).hash));

System.out.println("Trying to Mine block 2... ");

blockchain.get(1).mineBlock(difficulty);

blockchain.add(new Block("Hey im the third block",blockchain.get(blockchain.size()-1).hash));

System.out.println("Trying to Mine block 3... ");

blockchain.get(2).mineBlock(difficulty);

System.out.println("\nBlockchain is Valid: " + isChainValid());

String blockchainJson = new GsonBuilder().setPrettyPrinting().create().toJson(blockchain);

System.out.println("\nThe block chain: ");

System.out.println(blockchainJson);

}

public static Boolean isChainValid() {

Block currentBlock;

Block previousBlock;

String hashTarget = new String(new char[difficulty]).replace('\0', '0');

//loop through blockchain to check hashes:

for(int i=1; i < blockchain.size(); i++) {

currentBlock = blockchain.get(i);

previousBlock = blockchain.get(i-1);

//compare registered hash and calculated hash:

if(!currentBlock.hash.equals(currentBlock.calculateHash()) ){

System.out.println("Current Hashes not equal");

return false;

}

//compare previous hash and registered previous hash

if(!previousBlock.hash.equals(currentBlock.previousHash) ) {

System.out.println("Previous Hashes not equal");

return false;

}

//check if hash is solved

if(!currentBlock.hash.substring( 0, difficulty).equals(hashTarget)) {

System.out.println("This block hasn't been mined");

return false;

}

}

return true;

}

}

打印：

Connected to the target VM, address: '127.0.0.1:61863', transport: 'socket'

Trying to Mine block 1...

Block Mined!!! : 0000016667d4240e9c30f53015310b0ec6ce99032d7e1d66d670afc509cab082

Trying to Mine block 2...

Block Mined!!! : 000002ea55735bea4cac7e358c7b0d8d81e8ca24021f5f85211bf54fd4ac795a

Trying to Mine block 3...

Block Mined!!! : 000000576987e5e9afbdf19b512b2b7d0c56db0e6ca49b3a7e638177f617994b

Blockchain is Valid: true

[

{

"hash": "0000016667d4240e9c30f53015310b0ec6ce99032d7e1d66d670afc509cab082",

"previousHash": "0",

"data": "first",

"timeStamp": 1520659506042,

"nonce": 618139

},

{

"hash": "000002ea55735bea4cac7e358c7b0d8d81e8ca24021f5f85211bf54fd4ac795a",

"previousHash": "0000016667d4240e9c30f53015310b0ec6ce99032d7e1d66d670afc509cab082",

"data": "second",

"timeStamp": 1520659508825,

"nonce": 1819877

},

{

"hash": "000000576987e5e9afbdf19b512b2b7d0c56db0e6ca49b3a7e638177f617994b",

"previousHash": "000002ea55735bea4cac7e358c7b0d8d81e8ca24021f5f85211bf54fd4ac795a",

"data": "third",

"timeStamp": 1520659515910,

"nonce": 1404341

}

]

经过测试增加一个新的区块即挖矿必须花费一定时间，大概是3秒左右，你可以提高difficulty难度来看，它是如何影响数据难题所花费的时间的。如果有人在你的区块链系统中恶意篡改数据：

* 他们的区块链是无效的。
* 他们无法创建更长的区块链
* 网络中诚实的区块链会在长链中更有时间的优势

因为篡改的区块链将无法赶上长链和有效链，除非他们比你网络中所有的节点拥有更大的计算速度，可能是未来的量子计算机或者是其他什么。

你已经完成了你的基本区块链！