

从"斗地主"开始一步一步学习区块链

区块链由浅入深（第 1 部分）：从记账开始

"If you can not explain it simply, you don't understand it."

"如果你不能把一个技术很简单的讲出来，实际上是你没有吃透它。"

前言

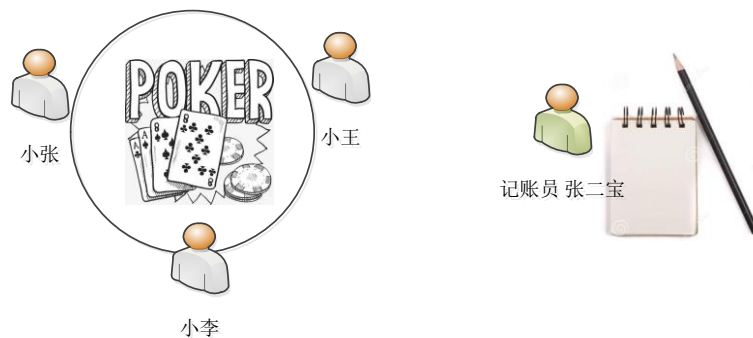
区块链技术是最近几年的热门技术，许多人开始了解学习它，但是它毕竟是一个有一定难度的技术，要深刻理解需要一定的融会贯通能力，要不就只能“空学习了一大堆名词，而难探其究”。

本文试图从最简单的生活场景开始，一步一步通过例子来描述区块链的原理；

对于想准确理解的技术性读者，会将用到的技术中英文属于对照列出，避免很多文章中翻译不一致造成的概念混淆。

对于有一定编码能力的读者，我们提供了示例性的代码，你可以通过下载运行，一步一步更加直观的理解区块链在计算机上的实现方式，完成你的第一个可以实际运行的区块链代码示例。

先从一个打牌游戏开始



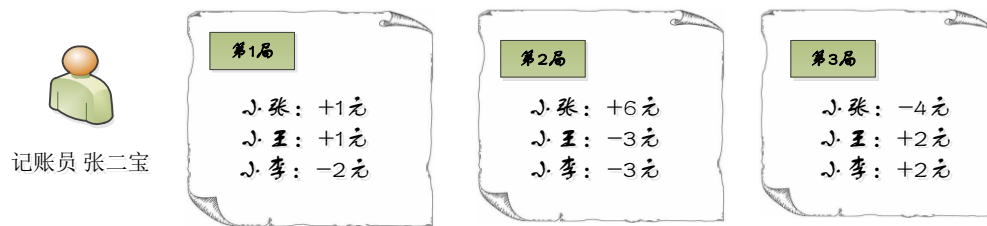
话说张王李三个好基友最喜欢的娱乐就是“欢乐斗地主”，今天周末他们又聚在一起，准备好好干个天昏地暗。

规则还是老样子：输赢基数一元钱，炸弹翻倍。

打牌是在小张的家里，每次小张都是让自己的上小学弟弟张二宝过来记账，张二宝虽小，但是是比较细心，字写得好看，冰雪聪明。

第 1 局，小李是地主，结果输掉了，他就分别给小张小王各 1 元钱； 开始第 2 局.....

三局之后，张二宝在纸上的记账如下：



记账的问题来了

三局过后，小张看到账本一盘算，还是自己赢的多，上次也是自己赢了，禁不住手舞足蹈起来，甚至还在别人洗牌间隙表演起来了自己在抖音上新的“嘟嘟舞”，然而意想不到的事情发生了：她竟然把刚刚泡好的一杯咖啡给打翻了，更重要的是这杯咖啡竟然就倒在了记账的便签纸上，一下子糊掉了，什么都看不清了...

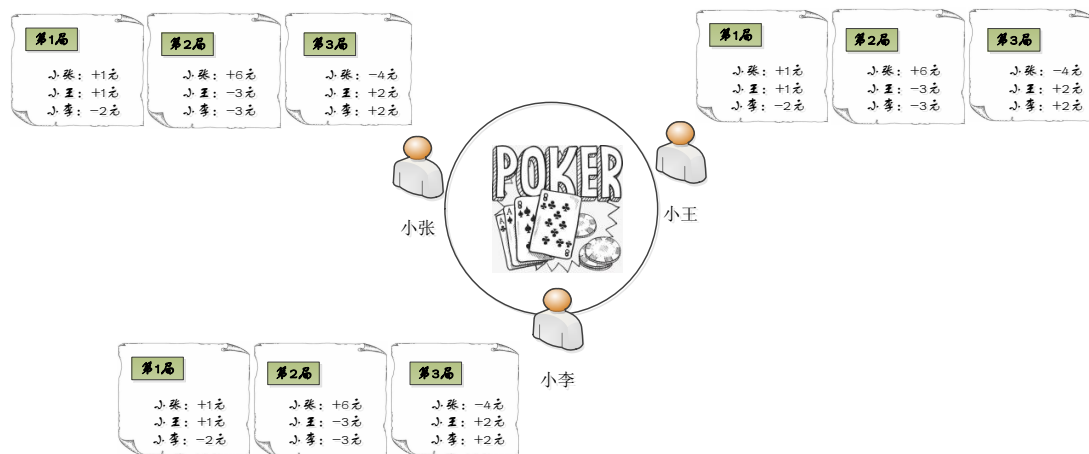
在一阵慌乱之后，大家都开始了思索：

小李在心里嘀咕了：为什么我每次打牌都是输呢？是不是记账的张二宝同学有舞弊嫌疑，给哥哥记录假账的可能？

小王是个没有记性的人，每次打几把，他都要问记账的把账单拿过来看看，心里琢磨自己赢多少或者输多少？次数多了，有点被记账的张二宝嫌弃，也有点不爽。

记账员张二宝也不开心了，每次都给他们服务，只能得到一个棒棒糖的奖赏，而且又一次自己去上个厕所漏记了一次，被他们发现教训，心里不爽，借着这个机会，就推掉算了。

小张是个机灵的人，他早也看到了两位的心思，突然想起网上有关区块链的介绍，“区块链的基础核心是分布式记账系统，具备不可篡改、不可抵赖，公开透明的特点”，为什么不用用区块链技术呢？想到这里他开始说给大家如此这般介绍一下，大家一听，都拍手称赞，新的记账模式开始了：



小结：他们开始各自记账

原来的模式下：所有的记账都是一个集中式的，由张二宝同学集中记账，所有的记录以中央记录的账本为准，每一个参与者（打牌的人）要想知道记账信息，就要向集中的账本查询。在异常的情况下会出现账本丢失损毁，所有信息就会丢失；还会出现集中记账系统出现错误时候（漏记，记错，或者有意篡改数据的可能）。

新的各自记账的模式：取消了集中记账的角色（张二宝），在每局打完之后，三个人就**分头各自记账并各自保存账本**。这样的两个最明显的益处是：可以防范集中记账奔溃的问题，而且每个人的账本完全一样，公开透明，每个人可以随时查看他自己的信息。

blockchain is a distributed, decentralized, public ledger.
区块链是一种分布式的，去中心化的，公开的记账系统。

读到这里，恭喜你，你已经了解大致区块链是怎么运作了。

接下来我们将看看新的模式下有什么新的问题，如何解决它。

区块链由浅入深（第 2 部分）：新的问题和解决之道

“A solution of the problem may also bring new problems.”

“一个问题的解决也可能随之带来新的问题。”

记录更多的信息

为了让信息更加准确和完备，每局打完牌后我们可以记录更多的信息，如下是第一局结束后的记录：

No:1
Xiaozhang = 1; Xiaowang = 1; Xiaoli = -2;
时间： 2020-03-26 16:42:08
计数器: 59
哈希（Hash）:005f83c135...
前哈希（PreviousHash）:00719d3db9...

其中：

“No” 来标记这是第几局，是第一局，所以写 “No:1”；这个数据有利于准确的记录局数，也便于最终的核对

接下来是具体的输赢数据，第一局，小张赢了 1 元，小李赢了 1 元，小李输了 2 元，依次记录；

“时间” 用来记录这局结束后记账的时间，我们精确到秒，例如 “2020-03-26 16:42:08”

“计数器” 记录的是节点计算机需要计算多少次哈希函数，才能得到符合设定目标的哈希值（后面详解）。

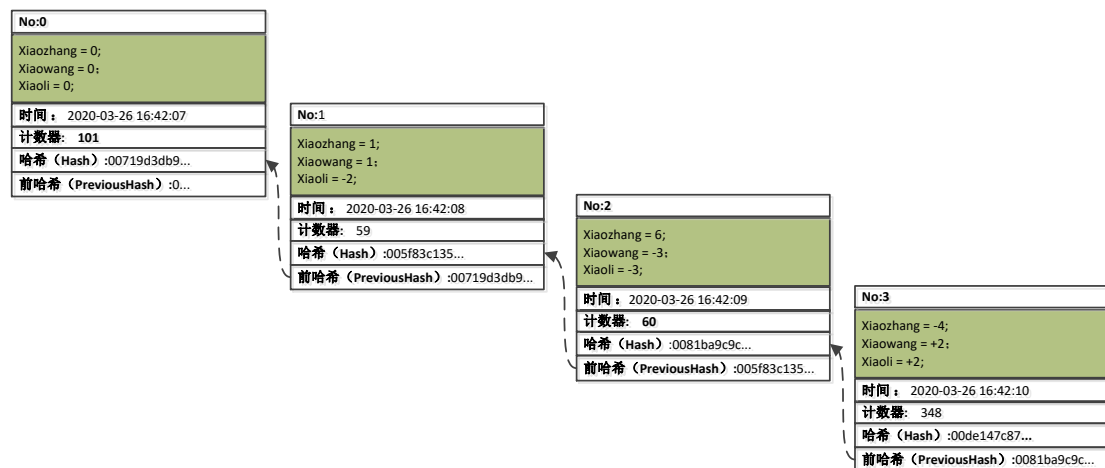
“哈希” 记录的是本局（本区块）记账信息的一个哈希计算结果（后面详解）。

“前哈希” 记录的是上一局（上一个区块）记账信息的一个哈希计算结果（后面详解）。

每个区块通过记录前面一个区块中的哈希值，把整个区块链接起来，形成了“区块链”。

注意：后面将要提到的 “区块” 对应到这里就是每一局结束后的交易记录（记账信息）。

注意：后面将要提到的 “节点” 对应到这里就是每一个单独的记账者（本例子中有 3 个人各自记账，就有 3 个节点，节点数也可以和交易参与者数量不一致）。



哈希 (Hash) 加密算法

哈希函数 (Hash Function)，也称为散列函数或杂凑函数。哈希函数是一个公开函数，可以将任意长度的消息 M 映射成为一个长度较短且长度固定的值 $H(M)$ ，称 $H(M)$ 为哈希值、散列值 (Hash Value)、杂凑值或者消息摘要 (Message Digest)。它是一种**单向密码体制**，即一个从明文到密文的不可逆映射，只有加密过程，没有解密过程。

通俗的讲，就是无论输入是什么数字格式、文件有多大，输出都是固定长度的比特串。例如 SHA256 算法为例，无论输入是什么数据文件，输出就是 256bit (64 位 16 进制数)。

理论上讲，计算机上的任意一段信息，都可以利用哈希函数来生成一个 64 位的不重复的 16 进制字符串。只要稍微更改一下信息，生成的哈希字符串就不一样了，例如：

Charles 2020 哈希值是：d93a6a24e8dee43aa1b3303a54f518420405294c4296851e5634e9532ef179c8

charles2020 哈希值是：14a3ab2a3f64dee43b5f1f37ae911e6aae86ffc501cbec00f12f01f928b9a73

“《人民日报》截至北京时间 25 日 17 时，中国以外新冠肺炎确诊病例达到 332331 例” 这句话的哈希值是：a579eedb7cd380b6c80ce417449a8fea8d62f4a0bd37266e852ee4b5dd1292df



我们再回到我们前面的区块记录中：

哈希值 记录的是当前块中数据字符转拼接起来，通过哈希算法，得到当前区块的哈希值，这个哈希值也可以叫做当前区块的签名，具备唯一性。就是当区块中数据如果发生了任何变化（例如被篡改等）那么重新计算出来的哈希函数值就会和表中记录的哈希值不相等，从而引起当前区块处于“无效”。

如果有的节点想篡改数据，他先修改了区块中的数据，然后按照哈希算法再算出新的哈希值，再用新计算的哈希值更新当前块的哈希值，使得当前块看起来“有效”，但是由于后面每个区块都会依次记录前一个区块的哈希值，现在发现前面的区块哈希值发生了变化，就会引起从这个区块及以后的所有的区块标识为“无效”。

共识机制：找出能把当前区块链接到链上的人

刚打完一局，例如第一局中，小李自己作为地主，输了 2 元，需要给小张和小王各自 1 元。这一点在大家没有异议的情况下，可以分头记录在自己的账本上。

但是如果参与者众多（成千上万的情况），最好的方式是，**先由其中一个人例如小张记账，然后把记账拿给大家看，大家照着这个记账再抄写一遍到自己的账本上。**我们再这里权且称这个人（小张）为本区块（局）**“有权记账者”**

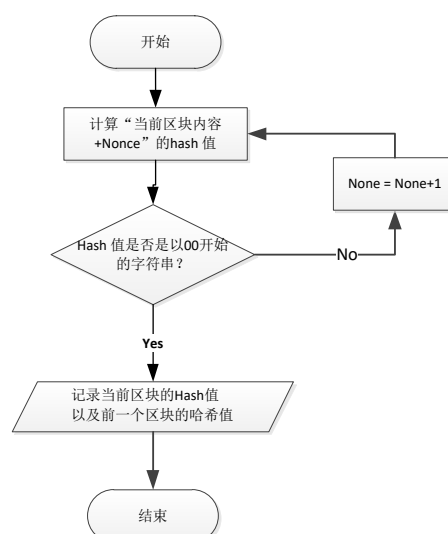
那么问题来了？这个有权记账者是怎么产生的？

要知道在区块链中每个节点的位置是完全相同的，那么说有权记账者是怎么选出来？这个可以有各种办法，例如抽签，例如投票，例如按照年龄，也可以设计一个简单问答来选出“有权记账者”但是在计算机实现上，考虑到计算机最大的资源是“计算机的运算能力”，我们就设定一个规则如下：

每一局结束，谁的计算能力最强，谁来做**“有权记账者”**，这个就是把这个块链接到已经有的链上，让当前这个块有效。

要考察谁的计算能力最强，我们出的题目是：**“计算每个区块的哈希值，为了增加难度，哈希值需要前面 N 位为 0”**

如果假定 N 设定为 2。则计算过程为：



谁先计算出来，谁就是把这个区块链接到主链上的人。链接后区块有效，系统通知其他人员复制到自己的账本，本次记账完成，开始下一局打牌。

如果该节点计算了 101 次（None 值=100），他的工作量就是 100。

这就是区块链中的共识机制中的 **POW (Proof of Work) 工作量证明**。

PoW 工作量证明 及其它

在区块链系统当中，没有一个像银行一样的中心化记账机构，保证每一笔交易在所有记账节点上的一致性，即让全网达成共识至关重要。区块链共识机制解决的就是这个问题。

目前区块链主要的共识机制有工作量证明机制 PoW (Proof of Work)和权益证明机制 PoS (Proof of stack)。

PoW 通过评估你的工作量来决定你获得记账权的机率，工作量越大，就越有可能获得此次记账机会。

PoS 通过评估你持有代币的数量和时长来决定你获得记账权的机率。这就类似于股票的分红制度，持有股权相对多的人能够获得更多的分红。

还有一种是 DPOS，它与 POS 原理相似，只是选了一些“代表”。与 PoS 的主要区别在于节点选举若干代表人，由代表验证和记账。

随着技术的发展，未来可能还会诞生更先进的共识机制。

本章小结

共识机制 (consensus)：区块链中的节点由于在同一时间会有时间上的延迟和动作的不同，需要一套公平的规则来规范这些节点，使得整个区块链系统顺利地运行下去。从本质上来讲，共识机制就是决定了谁在区块链系统中有权负责某一个新区块链接到主链上的作用。

区块链由浅入深（第 3 部分）：简单的区块链代码

“Genuine knowledge comes from practice”

“实践出真知”

通过前面两个章节，我们已经对于区块链的概念以及运行机制做了大致了解，接下来是实际动手通过一个编程来实际体验了。

需要具备的编码知识

- 面向对象的编码基本知识。
- Java 编程，编译及运行
- 加密的相关知识

代码实现：区块类-PokerBlock

“区块类”中存放每次区块的信息（记账信息，解密信息，链接信息等），我们在该类中存放 6 个值：

data - 当前区块的说明信息，第几局。
xiaozhang - 记录小张在当前局中的输赢数量
xiaowang - 记录小王在当前局中的输赢数量
xiaoli - 记录小李在当前局中的输赢数量
strDateTime - 记录当前交易发生的时间
nonce - 记录当前区块的工作量(即通过多少次 hash 运算最后得到了符合条件的当前区块的哈希值)
hash - 当前区块的哈希值
previousHash - 前一个区块的 hash 值

```
/**
 * @Description:    简单的区块链示例:扑克牌斗地主记账。
 *
 * @author         Charles (yonglin_guo@hotmail.com)
 * @version        V1.0
 * @Date           03/19/2020
 */
package com.janny.pokerblockchain;

import com.janny.pokerblockchain.JannyUtil;
/**
 * Block 区块类
 * @author Charles
 */
public class PokerBlock {

    //每个区块存放的数据信息，这里我们存放的是第几局，以及三个人的在当前牌局的输赢数量。
    private String data;
    private int xiaozhang;
    private int xiaowang;
    private int xiaoli;

    //时间字符串
    private String strDateTime;

    /*           挖矿者的工作量证明 PoW。

```



```

        *           在这里指需要经过多少次哈希运算才能得到满足条件的哈希值
        *
        */
private int nonce;

// 当前区块的哈希值;
public String hash;

// 前一个区块的 hash 值;
public String previousHash;

//构造方法
public PokerBlock(String data,int xiaozhang, int xiaowang, int xiaoli, String previousHash ) {
    this.data = data;
    this.xiaozhang = xiaozhang;
    this.xiaowang = xiaowang;
    this.xiaoli = xiaoli;

    this.previousHash = previousHash;
    this.strDateTime = (new JannyUtil()).getCurrentDateStr();
    this.hash = calculateHash();
}

//根据当前区块的信息内容计算新的哈希值
public String calculateHash() {
    String calculatedhash = (new JannyUtil()).applySha256(
        data +
        Integer.toString(xiaozhang) +
        Integer.toString(xiaowang) +
        Integer.toString(xiaoli) +
        strDateTime +
        Integer.toString(nonce) +
        previousHash
    );

    return calculatedhash;
}

/* “挖矿”过程
在这里指需要经过多少次哈希运算才能得到满足条件的哈希值
条件有参数 difficulty 设定
例如：如果 difficulty 值是 2,则要求计算出来的哈希值前 2 位字节为“00”
如果计算出来不是，则 nonce 值加 1，继续计算
直到算出来的哈希值满足条件，则结束
*/
public void mineBlock(int difficulty) {
    //难度值，difficulty 越大，计算量越大
    String target = (new JannyUtil()).getDifficultyString(difficulty);
    //difficulty 如果为 4，那么 target 则为 0000
    while(!hash.substring( 0, difficulty).equals(target)) {
        nonce ++;
        hash = calculateHash();
        System.out.println("!PoW 工作中..." + "nonce:" + nonce + ".hash: " + hash);
    }
    System.out.println("!!!当前节点 PoW 完成，新的区块被创建:" + hash);
}
}

```

代码实现：公共工具类-JannyUtil

“公共工具类”中是要用到的各种加密算法，字符串转换算法等的公共函数。注意为了让区块对象以字符串的方式打印出来方便观察，我们可以把一个区块对象装换为一个 Json 字符串对象。

注意: 请确保在类的编译路径中下载并包含 gson-2.8.6.jar 公共类库

```

/**
 * @Description:    简单的区块链示例:扑克牌斗地主记账。
 *
 * @author         Charles (yonglin_guo@hotmail.com)
 * @version        V1.0
 * @Date           03/18/2020

```

```

*/

package com.janny.pokerblockchain;

import java.text.SimpleDateFormat;

import java.util.Date;
import java.util.ArrayList;
import java.util.Base64;

import java.security.Key;
import java.security.MessageDigest;

//需要下载并确保 gson-2.8.6.jar 在类路径中
import com.google.gson.GsonBuilder;

/**
 * 工具类
 * 获得当前时间的字符串格式，对于字符串的数字签名、将一个对象装换为 JSON 格式数据、返回难度字符串目标
 * @author Charles
 */
public class JannyUtil {

    //返回系统当前时间的字符串格式
    public static String getCurrentDateStr() {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String dateString = formatter.format(new Date());
        return dateString;
    }

    /**计算一个字符串的的 Hash 值
     * MessageDigest.getInstance("MD5"); MD5 产生一个 128 位(16 字节)的散列值(hash value)
     * MessageDigest.getInstance("SHA-1"); SHA-1 产生一个 160(20 字节)位字符串
     * MessageDigest.getInstance("SHA-256"); SHA-256 产生一个 256 位(32 字节)字符串
     */
    public static String applySha256(String input){

        try {

            MessageDigest digest = MessageDigest.getInstance("SHA-256");

            byte[] hash = digest.digest(input.getBytes("UTF-8"));

            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(Exception e) {
            throw new RuntimeException(e);
        }

    }

    //返回 JSON 格式数据，供打印用。
    public static String getJson(Object o) {
        return new GsonBuilder().setPrettyPrinting().create().toJson(o);
    }

    //返回难度字符串目标，例如难度 5 将返回 “00000”
    public static String getDifficultyString(int difficulty) {
        return new String(new char[difficulty]).replace('\0', '0');
    }

    //返回难度字符串目标，例如难度 5 将返回 “00000”
    public static String getStringFromKey(Key key) {
        return Base64.getEncoder().encodeToString(key.getEncoded());
    }

    public static void main(String[] args) {

        System.out.println("Datetime..... "+getCurrentDateStr());

    }
}

```

代码实现：区块链类- PokerBlockChain

区块链类，以此产生区块，并将其加入到区块链中，也可以判断当前区块是否有效。

```
/**
 * @Description:    简单的区块链示例:扑克牌斗地主记账。
 *
 * @author         Charles (yonglin_guo@hotmail.com)
 * @version        V1.0
 * @Date           03/19/2020
 */

package com.janny.pokerblockchain;

import java.util.ArrayList;
import java.util.Base64;

import java.security.Security;

//需要下载并确保 gson-2.8.6.jar 在类路径中
import com.google.gson.GsonBuilder;

/**
 * 创建区块链
 * @author Charles
 *
 */
public class PokerBlockChain{
    //存放所有的区块集合
    public static ArrayList<PokerBlock> blockchain = new ArrayList<PokerBlock>();

    //挖矿的难度，数字越大越难
    public static int difficulty = 2;

    public static void main(String[] args) {

        String title = "创世块";
        int xiaozhang = 0;
        int xiaowang = 0;
        int xiaoli = 0;
        System.out.println("开始创建第 0 个区块链:创世块..... ");
        addBlock(new PokerBlock("第 0 个区块",xiaozhang,xiaowang,xiaoli,"0"));//创世块

        title = "第 1 局";
        xiaozhang = 1;
        xiaowang = 1;
        xiaoli = -2;
        System.out.println("正在创建第 1 个区块..... ");
        addBlock(new PokerBlock("第 1 个区块
",xiaozhang,xiaowang,xiaoli,blockchain.get(blockchain.size()-1).hash));

        title = "第 2 局";
        xiaozhang = 6;
        xiaowang = -3;
        xiaoli = -3;
        System.out.println("正在创建第 2 个区块..... ");
        addBlock(new PokerBlock("第 2 个区块
",xiaozhang,xiaowang,xiaoli,blockchain.get(blockchain.size()-1).hash));

        title = "第 3 局";
        xiaozhang = -4;
        xiaowang = 2;
        xiaoli = 2;
        System.out.println("正在创建第 3 个区块..... ");
        addBlock(new PokerBlock("第 3 个区块
",xiaozhang,xiaowang,xiaoli,blockchain.get(blockchain.size()-1).hash));
```

```

        System.out.println("区块链是否有效的: " + isChainValid());

        String blockchainJson = (new JannyUtil()).getJson(blockchain);
        System.out.println(blockchainJson);

    }

    /**
     * 检查区块链的完整性
     * 从第一个区块开始逐次检查两个内容：
     *     1，当前的区块哈希值是否正确？（通过重新计算结果，和区块中保存的区块比较）
     *     2，当前区块中保存的“前区块哈希值”是不是和前一个区块中的实际的哈希值相等
     */
    public static Boolean isChainValid() {
        PokerBlock currentBlock;
        PokerBlock previousBlock;
        String hashTarget = new String(new char[difficulty]).replace('\0', '0');

        //循环区块链检查哈希值:
        for(int i=1; i < blockchain.size(); i++) {
            currentBlock = blockchain.get(i);
            previousBlock = blockchain.get(i-1);

            //比较当前区块中的哈希值是否和重新计算出来的相等？
            if(!currentBlock.hash.equals(currentBlock.calculateHash())){
                System.out.println("当前区块哈希值校验错误...");
                return false;
            }
            //当前区块中保存的“前区块哈希值”是不是和前一个区块中的实际的哈希值是否相等？
            if(!previousBlock.hash.equals(currentBlock.previousHash)) {
                System.out.println("当前区块的‘前哈希值’校验错误...");
                return false;
            }
            //检查当前区块是否已经加到区块链上？
            if(!currentBlock.hash.substring( 0, difficulty).equals(hashTarget)) {
                System.out.println("这个区块还没有被加到区块链上...");
                return false;
            }

        }

        return true;
    }

    /**
     * 在区块链上增加一个新的区块
     * @param newBlock
     */
    public static void addBlock(PokerBlock newBlock) {
        newBlock.mineBlock(difficulty);
        blockchain.add(newBlock);
    }
}

```

运行和测试

代码运行后，开始以此创建 4 个区块。

每个区块在创建时候，先做 PoW 工作（在比特币中叫做挖矿），挖矿结束后，将当前的区块链接到区块链上。再开始下一个区块的创建。

为了演示方便，将难度系数设定为 2，这样，只需要做几十到上百次 Hash 算法就可以完成。

以下由于篇幅限制省略了部分程序输出，只显示前面三个计算的哈希值以及后面两个计算的哈希值。

```

开始创建第 0 个区块链:创世块.....
!PoW 工作中...nonce:1:hash: c40a984d0d31b85640cd67decf7dbe3ab684765b6bf36b38e243dc428b2d3fb5
!PoW 工作中...nonce:2:hash: 0eef49b73b88396e474899b69baeaf6f8089c4c6f07417446b8894d720c6d2a4

```

```
!PoW 工作中...nonce:3:hash: e0d6bfb8f222fc886802047f62217c0683570f403ca1c6ae1409ac8ba359f150
...

!PoW 工作中...nonce:100:hash: b9ee001c09cc9dfa8884fb4f1b9447f75e30d7fcea09edf75d71eabaa4b97827
!PoW 工作中...nonce:101:hash: 00719d3db941650ef3ccdace527c3deed836ddb0cf6304e759cab4ba237db6ee
!!!当前节点 PoW 完成, 新的区块被创建:00719d3db941650ef3ccdace527c3deed836ddb0cf6304e759cab4ba237db6ee
正在创建第 1 个区块.....
!PoW 工作中...nonce:1:hash: 18215996f7b6f7041d20b8ea5067040af4eeb2370773de9713a1eca1206a5d3e
!PoW 工作中...nonce:2:hash: 01987723af672357a6f6f50da35c32c996c01b6349c860417776144e0b4c6e91
!PoW 工作中...nonce:3:hash: f12614d3c9745d00a5631e3a620cbb472454a92214f27eb3f40a8cb0e9faee88
...

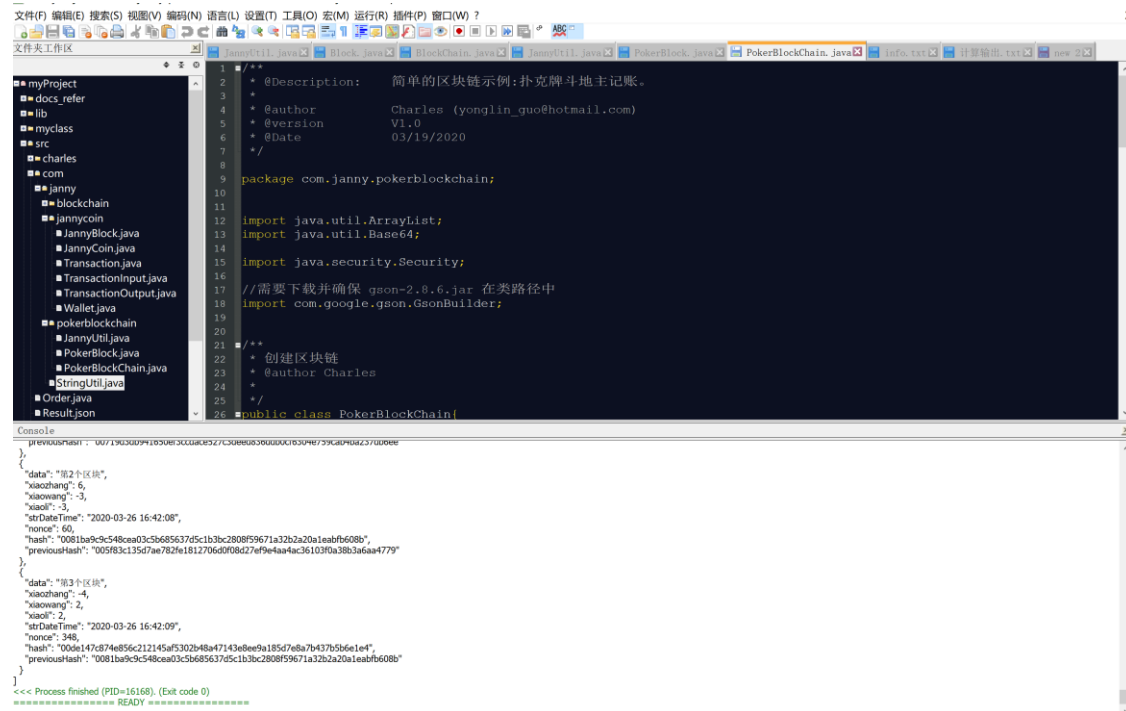
!PoW 工作中...nonce:58:hash: 6929737b2d514807aad983b020112699491c82fda46bb6000903046d6607050e
!PoW 工作中...nonce:59:hash: 005f83c135d7ae782fe1812706d0f08d27ef9e4aa4ac36103f0a38b3a6aa4779
!!!当前节点 PoW 完成, 新的区块被创建:005f83c135d7ae782fe1812706d0f08d27ef9e4aa4ac36103f0a38b3a6aa4779
正在创建第 2 个区块.....
!PoW 工作中...nonce:1:hash: 3f45344b335827d278643dd7eaa43bb255c2095200b27dabdc288470e2ded533
!PoW 工作中...nonce:2:hash: cc32db0f15df81d0eb95780175540d45ab20676375ce29e7f8779500819be1f1
!PoW 工作中...nonce:3:hash: f505f201630e52e19807d08337b65142f9679854f72fed7657b8266a64e5e254
...

!PoW 工作中...nonce:59:hash: f3b29379d64e8b6c467efb7aca05df5b0e0dcff4eae2c281ba24a5775ac27ef3
!PoW 工作中...nonce:60:hash: 0081ba9c9c548cea03c5b685637d5c1b3bc2808f59671a32b2a20a1eabfb608b
!!!当前节点 PoW 完成, 新的区块被创建:0081ba9c9c548cea03c5b685637d5c1b3bc2808f59671a32b2a20a1eabfb608b
正在创建第 3 个区块.....
!PoW 工作中...nonce:1:hash: eda9b63fcd8e3a72d604cc9dda9ebc26e8eaaa314a9999ac44afc725b919c7c7
!PoW 工作中...nonce:2:hash: 061b3386cdb975311cf48a899ac931371da683bac276823e1726741da4a964db
!PoW 工作中...nonce:3:hash: 5a9b65e96f470cfcfec5466fd6f5a687c0a9e59d697ad0fb24d723f7c6b3584c
...

!PoW 工作中...nonce:347:hash: 42727f99cbb4c90e622f9c1e73663555a8adff4b4bcd13fa4375a4b5184a255
!PoW 工作中...nonce:348:hash: 00de147c874e856c212145af5302b48a47143e8ee9a185d7e8a7b437b5b6e1e4
!!!当前节点 PoW 完成, 新的区块被创建:00de147c874e856c212145af5302b48a47143e8ee9a185d7e8a7b437b5b6e1e4
区块链是否有效的: true
[
  {
    "data": "第 0 个区块",
    "xiaozhang": 0,
    "xiaowang": 0,
    "xiaoli": 0,
    "strDateTime": "2020-03-26 16:42:07",
    "nonce": 101,
    "hash": "00719d3db941650ef3ccdace527c3deed836ddb0cf6304e759cab4ba237db6ee",
    "previousHash": "0"
  },
  {
    "data": "第 1 个区块",
    "xiaozhang": 1,
    "xiaowang": 1,
    "xiaoli": -2,
    "strDateTime": "2020-03-26 16:42:08",
    "nonce": 59,
    "hash": "005f83c135d7ae782fe1812706d0f08d27ef9e4aa4ac36103f0a38b3a6aa4779",
    "previousHash": "00719d3db941650ef3ccdace527c3deed836ddb0cf6304e759cab4ba237db6ee"
  },
  {
    "data": "第 2 个区块",
    "xiaozhang": 6,
    "xiaowang": -3,
    "xiaoli": -3,
    "strDateTime": "2020-03-26 16:42:08",
    "nonce": 60,
    "hash": "0081ba9c9c548cea03c5b685637d5c1b3bc2808f59671a32b2a20a1eabfb608b",
    "previousHash": "005f83c135d7ae782fe1812706d0f08d27ef9e4aa4ac36103f0a38b3a6aa4779"
  },
  {
    "data": "第 3 个区块",
    "xiaozhang": -4,
    "xiaowang": 2,
    "xiaoli": 2,
    "strDateTime": "2020-03-26 16:42:09",
    "nonce": 348,
    "hash": "00de147c874e856c212145af5302b48a47143e8ee9a185d7e8a7b437b5b6e1e4",
    "previousHash": "0081ba9c9c548cea03c5b685637d5c1b3bc2808f59671a32b2a20a1eabfb608b"
  }
]
```

]

我们设定 $\text{difficulty} = 2$ ，来运行，结果如下：



The screenshot shows an IDE with a project named 'myProject' on the left. The main editor displays the 'PokerBlockChain.java' file. The code includes a package declaration, imports for ArrayList, Base64, and Gson, and a public class 'PokerBlockChain' with a 'main' method. The console at the bottom shows the output of the program, which is a JSON array of two blocks. Each block contains metadata like 'data', 'xiao Zhang', 'xiao Wang', 'xiao Li', 'str Date Time', 'nonce', 'hash', and 'previous Hash'.

```
1 /**
2  * @Description: 简单的区块链示例:扑克牌斗地主记账。
3  *
4  * @author Charles (yonglin_guo@hotmail.com)
5  * @version V1.0
6  * @Date 03/19/2020
7  */
8
9 package com.janny.pokerblockchain;
10
11
12 import java.util.ArrayList;
13 import java.util.Base64;
14
15 import java.security.Security;
16
17 //需要下载并确保 gson-2.8.6.jar 在类路径中
18 import com.google.gson.GsonBuilder;
19
20
21 /**
22  * 创建区块链
23  * @author Charles
24  */
25
26 public class PokerBlockChain {
```

```
{
  "data": "第2个区块",
  "xiao Zhang": 6,
  "xiao Wang": -3,
  "xiao Li": -3,
  "str Date Time": "2020-03-26 16:42:08",
  "nonce": 60,
  "hash": "0081ba9c548cea03c5b685637d5c1b3bc2808f59671a32b2a20a1eabf6608b",
  "previous Hash": "005f83c135d7ae782fe1812706d0f08d27ef9e4a4ac36103f0a38b3a6aa4779"
},
{
  "data": "第3个区块",
  "xiao Zhang": 4,
  "xiao Wang": 2,
  "xiao Li": 2,
  "str Date Time": "2020-03-26 16:42:09",
  "nonce": 348,
  "hash": "00de147d874e856c212145af5302b48a47143e8ee9a185d7e8a7b437b5b6e1e4",
  "previous Hash": "0081ba9c548cea03c5b685637d5c1b3bc2808f59671a32b2a20a1eabf6608b"
}
}
<<< Process finished (PID=16168). (Exit code 0)
***** READY *****
```

实际运行时后可以 通过调整挖矿难度，来体验，如果设定 $\text{difficulty} = 4$ ，计算机需要几分钟时间几万条的哈希计算才能完成一个区块的上链。

获得示例

从 github 中下载: <https://github.com/ecustjanny/PokerBlockChain>

本章小结

通过实际编码，你已经掌握了一个简单的区块链代码，学会了创建区块，并通过工作证明机制，让你的区块链链接到主链上，也可以用来测试数据被篡改后的有效性...

更多...

如果你确认对于这里讲的区块链已经能理解，并希望更深入的了解高级技术，请继续阅读后面的章节。

任何疑问或者反馈，请在评论区给我留言，或者邮件给我 yonglin_guo@hotmail.com 一起探讨。

如果你想转载，请注明出处，谢谢。

区块链由浅入深（第 4 部分） 高级篇：智能合约

"If you can not explain it simply, you don' t understand it."

"如果你不能把一个技术很简单的讲出来，实际上是你没有吃透它。"

区块链由浅入深（第 5 部分）高级篇：加密技术

"If you can not explain it simply, you don' t understand it."

"如果你不能把一个技术很简单的讲出来，实际上是你没有吃透它。"

区块链由浅入深（第 6 部分）高级篇：动手创建我的比特币

"If you can not explain it simply, you don' t understand it."

"如果你不能把一个技术很简单的讲出来，实际上是你没有吃透它。"

区块链由浅入深（第 7 部分）高级篇：我的比特币代码实现

"If you can not explain it simply, you don' t understand it."

"如果你不能把一个技术很简单的讲出来，实际上是你没有吃透它。"