

(/apps/redirect?utm_source=side-banner-click)

(译)使用Go语言从零编写PoS区块链



Surface聊技术 (/u/47bf63bd97d2) + 关注

2018.03.26 13:30* 字数 4536 阅读 305 评论 0 喜欢 0

(/u/47bf63bd97d2)

原文发布日期： 2018-03-26
原文链接：<https://medium.com/@mycoralhealth/code-your-own-proof-of-stake-blockchain-in-go-610cd99aa658> (<https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fcode-your-own-proof-of-stake-blockchain-in-go-610cd99aa658>)

转载请在文章开头注明作者和出处
作者: ChainGod(孙飞)
原文链接: <http://chaingod.io/article/16> (<https://link.jianshu.com?t=http%3A%2F%2Fchaingod.io%2Farticle%2F16>)

PoS简介

在上一篇文章 (<https://link.jianshu.com?t=%28https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fcode-your-own-blockchain-mining-algorithm-in-go-82c6a71aba1f%29>)中，我们讨论了工作量证明(Proof of Work)，并向您展示了如何编写自己的工作量证明区块链。当前最流行的两个区块链平台，比特币和Ethereum都是基于工作量证明的。

但是工作证明的缺点是什么呢?其中一个主要的问题是电力能源的消耗。为了挖掘更多的比特币，就需要建立更多的挖矿硬件池，现在在世界各地，挖矿池都在不断建立中，而且呈现出规模越来越大的趋势。例如以下这张照片（仅仅是矿池的一角）：



挖矿场的一角.png

挖矿工作需要耗费大量的电力，仅比特币开采耗费的能源就超过了159个国家的电力能源消耗总和!! 这种能源消耗是非常非常不合理的，而且，从技术的角度来看，工作量证明还有其他不足之处：随着越来越多的人参与到挖矿工作中，共识算法的难度就需要提



高，难度的提高意味着需要更多、更长时间的挖矿，也意味着区块和交易需要更长的时间才能得到处理，因此能源的消耗就会越发的高。总之，工作量证明的方式就是一场竞赛，你需要更多的计算能力才能有更大的概率赢得比赛。

有很多区块链学者都试图找到工作量证明的替代品，到目前为止最有希望的就是PoS(权益证明或者股权证明，Proof of Stake)。目前在生产环境，已经有数个区块链平台使用了PoS，例如Nxt (<https://link.jianshu.com?t=https%3A%2F%2Fnxtplatform.org%2Fwhat-is-nxt%2F>) 和Neo (<https://link.jianshu.com?t=https%3A%2F%2Fneo.org%2F>)。以太坊Ethereum在不远的未来也很可能会使用PoS——他们的Casper项目已经在测试网络上运行和测试了。

(/apps/redirect?utm_source=side-banner-click)

那么，到底什么才是股权证明PoS呢？

在PoW中，节点之间通过hash的计算力来竞赛以获取下一个区块的记账权，而在PoS中，块是已经铸造好的(这里没有“挖矿”的概念，所以我们不用这个词来证明股份)，铸造的过程是基于每个节点(Node)愿意作为抵押的令牌(Token)数量。这些参与抵押的节点被称为验证者(Validator)，**注意在本文后续内容中，验证者和节点的概念是等同的！**令牌的含义对于不同的区块链平台是不同的，例如，在以太坊中，每个验证者都将Ether作为抵押品。

如果验证者愿意提供更多的令牌作为抵押品，他们就有更大的机会记账下一个区块并获得奖励。你可以把奖励的区块看作是存款利息，你在银行存的钱越多，你每月的利息就会越高。

因此，这种共识机制被称为股权证明PoS。

PoS的缺陷是什么？

您可能已经猜到，一个拥有大量令牌的验证者会在创建新块时根据持有的令牌数量获得更高的概率。然而，这与我们在工作量证明中看到的并没有什么不同：比特币矿场变得越来越强大，普通人在自己的电脑上开采多年也未必能获得一个区块。因此，许多人认为，使用了PoS后，区块的分配将更加民主化，因为任何人都可以在自己的笔记本上参与，而不需要建立一个巨大的采矿平台，他们不需要昂贵的硬件，只需要一定的筹码，就算筹码不多，也有一定概率能获得区块的记账权，希望总是有的，你说呢？

从技术和经济的角度来看，还有其他不利因素。我们不会一一介绍，但这里(<https://link.jianshu.com?t=https%3A%2F%2Fbitcoin.stackexchange.com%2Fquestions%2F25743%2Fwhat-are-the-downsides-of-proof-of-stake>)有一个很好的介绍。在实际应用中，PoS和PoW都有自己的优点和缺点，因此以太坊的Casper具有两者混合的特征。

像往常一样，了解PoS的方法是编写自己的代码，那么，我们开始吧！

编写PoS代码

我们建议在继续之前看一下200行Go代码编写区块链Part2 (<https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fpart-2-networking-code-your-own-blockchain-in-less-than-200-lines-of-go-17fe1dad46e1>)，因为在接下来的文章中，一些基础知识不再会介绍，因此这篇文章能帮助你回顾一下。

注意

我们将实现PoS的核心概念，然后因为文章长度有限，因此一些不必要的代码奖省去！

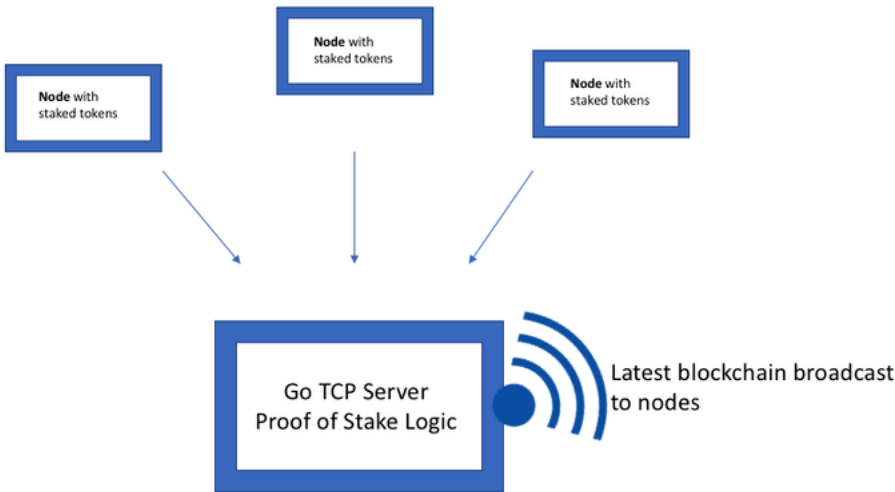
- P2P网络的实现。文中的网络是模拟的，区块链状态只在其中一个中心化节点持有，而不是每个节点，同时状态通过该持有节点广播到其它节点



- 钱包和余额变动。本文没有实现一个钱包，持有的令牌数量是通过stdin（标准输入）输入的，你可以输入你想要的任何数量。一个完整的实现会为每个节点分配一个hash地址，并在节点中跟踪余额的变动

(/apps/redirect?utm_source=side-banner-click)

架构图



架构.png

- 我们将有一个中心化的TCP服务节点，其他节点可以连接该服务器
- 最新的区块链状态将定期广播到每个节点
- 每个节点都能提议建立新的区块
- 基于每个节点的令牌数量，其中一个节点将随机地(以令牌数作为加权值)作为获胜者，并且将该区块添加到区块链中

设置和导入

在开始写代码之前，我们需要一个环境变量来设置TCP服务器的端口，首先在工作文件夹中创建.env文件，写入一行配置：

ADDR=9000

我们的Go程序将读取该文件，并且暴露出9000端口。同时在工作目录下，再创建一个main.go文件。

```
package main

import (
    "bufio"
    "crypto/sha256"
    "encoding/hex"
    "encoding/json"
    "fmt"
    "io"
    "log"
    "math/rand"
    "net"
    "os"
    "strconv"
    "sync"
    "time"

    "github.com/davecgh/go-spew/spew"
    "github.com/joho/godotenv"
)
```

- spew 可以把我们的区块链用漂亮的格式打印到终端terminal中
- godotenv 允许我们从之前创建的.env文件读取配置

快速脉搏检查



如果你读过我们的其他教程，就会知道我们是一家医疗保健公司，目前要去收集人体脉搏信息，同时添加到我们的区块上。把两个手指放在你的手腕上，数一下你一分钟能感觉到多少次脉搏，这将是您的BPM整数，我们将在接下来的文章中使用。

(/apps/redirect?utm_source=side-banner-click)

全局变量

现在，让我们声明我们需要的所有全局变量(main.go中)。

```
// Block represents each 'item' in the blockchain
type Block struct {
    Index      int
    Timestamp  string
    BPM        int
    Hash       string
    PrevHash   string
    Validator  string
}

// Blockchain is a series of validated Blocks
var Blockchain []Block
var tempBlocks []Block

// candidateBlocks handles incoming blocks for validation
var candidateBlocks = make(chan Block)

// announcements broadcasts winning validator to all nodes
var announcements = make(chan string)

var mutex = &sync.Mutex{}

// validators keeps track of open validators and balances
var validators = make(map[string]int)
```

- Block是每个区块的内容
- Blockchain是我们的官方区块链，它只是一串经过验证的区块集合。每个区块中的PrevHash与前面块的Hash相比较，以确保我们的链是正确的。tempBlocks是临时存储单元，在区块被选出来并添加到BlockChain之前，临时存储在这里
- candidateBlocks是Block的通道，任何一个节点在提出一个新块时都将它发送到这个通道
- announcements也是一个通道，我们的主Go TCP服务器将向所有节点广播最新的区块链
- mutex是一个标准变量，允许我们控制读/写和防止数据竞争
- validators是节点的存储map，同时也会保存每个节点持有的令牌数

基本的区块链函数

在继续PoS算法之前，我们先来实现标准的区块链函数。如果你之前看过200行Go代码编写区块链 ([https://link.jianshu.com/t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fcode-your-own-blockchain-in-less-than-200-lines-of-go-e296282bcffc%3Fsource%3Duser_profile-----13-----](https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fcode-your-own-blockchain-in-less-than-200-lines-of-go-e296282bcffc%3Fsource%3Duser_profile-----13-----)),那接下来应该更加熟悉。

main.go

```
// SHA256 hasing
// calculateHash is a simple SHA256 hashing function
func calculateHash(s string) string {
    h := sha256.New()
    h.Write([]byte(s))
    hashed := h.Sum(nil)
    return hex.EncodeToString(hashed)
}

//calculateBlockHash returns the hash of all block information
func calculateBlockHash(block Block) string {
    record := string(block.Index) + block.Timestamp + string(block.BPM) + block.PrevHash
    return calculateHash(record)
}
```



这里先从hash函数开始，calculateHash函数会接受一个string，并且返回一个SHA256 hash。calculateBlockHash是对一个block进行hash，将一个block的所有字段连接到一起后，再进行hash。

main.go

```
func generateBlock(oldBlock Block, BPM int, address string) (Block, error) {

    var newBlock Block

    t := time.Now()

    newBlock.Index = oldBlock.Index + 1
    newBlock.Timestamp = t.String()
    newBlock.BPM = BPM
    newBlock.PrevHash = oldBlock.Hash
    newBlock.Hash = calculateBlockHash(newBlock)
    newBlock.Validator = address

    return newBlock, nil
}
```

generateBlock是用来创建新块的。每个新块都有的一个重要字段是它的hash签名(通过calculateBlockHash计算的)和上一个连接块的PrevHash(因此我们可以保持链的完整性)。我们还添加了一个Validator字段，这样我们就知道了该构建块的获胜节点。

main.go

```
// isBlockValid makes sure block is valid by checking index
// and comparing the hash of the previous block
func isBlockValid(newBlock, oldBlock Block) bool {
    if oldBlock.Index+1 != newBlock.Index {
        return false
    }

    if oldBlock.Hash != newBlock.PrevHash {
        return false
    }

    if calculateBlockHash(newBlock) != newBlock.Hash {
        return false
    }

    return true
}
```

isBlockValid会验证Block的当前hash和PrevHash，来确保我们的区块链不会被污染。

节点 (验证者)

当一个验证者连接到我们的TCP服务，我们需要提供一些函数达到以下目标：

- 输入令牌的余额（之前提到过，我们不做钱包等逻辑）
- 接收区块链的最新广播
- 接收验证者赢得区块的广播信息
- 将自身节点添加到全局的验证者列表中（validators）
- 输入Block的BPM数据- BPM是每个验证者的人体脉搏值
- 提议创建一个新的区块

这些目标，我们用 handleConn 函数来实现

main.go

(/apps/redirect?
utm_source=side-
banner-click)



```

func handleConn(conn net.Conn) {
    defer conn.Close()

    go func() {
        for {
            msg := <-announcements
            io.WriteString(conn, msg)
        }
    }()
    // validator address
    var address string

    // allow user to allocate number of tokens to stake
    // the greater the number of tokens, the greater chance to forging a new block
    io.WriteString(conn, "Enter token balance:")
    scanBalance := bufio.NewScanner(conn)
    for scanBalance.Scan() {
        balance, err := strconv.Atoi(scanBalance.Text())
        if err != nil {
            log.Printf("%v not a number: %v", scanBalance.Text(), err)
            return
        }
        t := time.Now()
        address = calculateHash(t.String())
        validators[address] = balance
        fmt.Println(validators)
        break
    }

    io.WriteString(conn, "\nEnter a new BPM:")

    scanBPM := bufio.NewScanner(conn)

    go func() {
        for {
            // take in BPM from stdin and add it to blockchain after conducting necessary checks
            for scanBPM.Scan() {
                bpm, err := strconv.Atoi(scanBPM.Text())
                // if malicious party tries to mutate the chain with a bad input, delete the block
                if err != nil {
                    log.Printf("%v not a number: %v", scanBPM.Text(), err)
                    delete(validators, address)
                    conn.Close()
                }

                mutex.Lock()
                oldLastIndex := Blockchain[len(Blockchain)-1]
                mutex.Unlock()

                // create newBlock for consideration to be forged
                newBlock, err := generateBlock(oldLastIndex, bpm, address)
                if err != nil {
                    log.Println(err)
                    continue
                }
                if isBlockValid(newBlock, oldLastIndex) {
                    candidateBlocks <- newBlock
                }
                io.WriteString(conn, "\nEnter a new BPM:")
            }
        }
    }()

    // simulate receiving broadcast
    for {
        time.Sleep(time.Minute)
        mutex.Lock()
        output, err := json.Marshal(Blockchain)
        mutex.Unlock()
        if err != nil {
            log.Fatal(err)
        }
        io.WriteString(conn, string(output)+"\n")
    }
}

```

(/apps/redirect?
utm_source=side-
banner-click)

第一个Go协程接收并打印来自TCP服务器的任何通知，这些通知包含了获胜验证者的通知。



io.WriteString(conn, "Enter token balance:")允许验证者输入他持有的令牌数量，然后，该验证者被分配一个SHA256地址，随后该验证者地址和验证者的令牌数被添加到验证者列表validators中。

接着我们输入BPM，验证者的脉搏值，并创建一个单独的Go协程来处理这块儿逻辑，下面这一行代码很重要：

```
delete(validators, address)
```

如果验证者试图提议一个被污染（例如伪造）的block，例如包含一个不是整数的BPM，那么程序会抛出一个错误，我们会立即从我们的验证器列表validators中删除该验证者，他们将不再有资格参与到新块的铸造过程同时丢失相应的抵押令牌。

正式因为这种抵押令牌的机制，使得PoS协议是一种更加可靠的机制。如果一个人试图伪造和破坏，那么他将被抓住，并且失去所有抵押和未来的权益，因此对于恶意者来说，是非常大的威慑。

接着，我们用generateBlock函数创建一个新的block，然后将其发送到candidateBlocks通道进行进一步处理。将Block发送到通道使用的语法：

```
candidateBlocks <- newBlock
```

上面代码中最后一段的循环会周期性的打印出最新的区块链，这样每个验证者都能获知最新的状态

选择获胜者

这里是PoS的主题逻辑。我们需要编写代码以实现获胜验证者的选择;他们所持有的令牌数量越高，他们就有可能被选为胜利者。

为了简化代码，我们只会让提出新块儿的验证者参与竞争。在传统的PoS，一个验证者即使没有提出一个新的区块，也可以被选为胜利者。切记，PoS不是一种确定的定义（算法），而是一种概念，因此对于不同的平台来说，可以有不同的PoS实现。

下面来看看pickWinner函数：

```
main.go
```

(/apps/redirect?utm_source=side-banner-click) ✕



```

// pickWinner creates a lottery pool of validators and chooses the validator who gets
// by random selecting from the pool, weighted by amount of tokens staked
func pickWinner() {
    time.Sleep(30 * time.Second)
    mutex.Lock()
    temp := tempBlocks
    mutex.Unlock()

    lotteryPool := []string{}
    if len(temp) > 0 {

        // slightly modified traditional proof of stake algorithm
        // from all validators who submitted a block, weight them by the number of
        // in traditional proof of stake, validators can participate without submitting
    OUTER:
        for _, block := range temp {
            // if already in lottery pool, skip
            for _, node := range lotteryPool {
                if block.Validator == node {
                    continue OUTER
                }
            }

            // lock list of validators to prevent data race
            mutex.Lock()
            setValidators := validators
            mutex.Unlock()

            k, ok := setValidators[block.Validator]
            if ok {
                for i := 0; i < k; i++ {
                    lotteryPool = append(lotteryPool, block.Validator)
                }
            }
        }

        // randomly pick winner from lottery pool
        s := rand.NewSource(time.Now().Unix())
        r := rand.New(s)
        lotteryWinner := lotteryPool[r.Intn(len(lotteryPool))]

        // add block of winner to blockchain and let all the other nodes know
        for _, block := range temp {
            if block.Validator == lotteryWinner {
                mutex.Lock()
                Blockchain = append(Blockchain, block)
                mutex.Unlock()
                for _ = range validators {
                    announcements <- "\nwinning validator: " + lotteryWinner + "\n"
                }
                break
            }
        }
    }

    mutex.Lock()
    tempBlocks = []Block{}
    mutex.Unlock()
}

```

(/apps/redirect?
utm_source=side-
banner-click)

每隔30秒，我们选出一个胜利者，这样对于每个验证者来说，都有时间提议新的区块，参与到竞争中来。接着创建一个 `lotteryPool`，它会持有所有验证者的地址，这些验证者都有机会成为一个胜利者。然后，对于提议块的暂存区域，我们会通过 `if len(temp) > 0` 来判断是否已经有了被提议的区块。

在 `OUTER FOR` 循环中，要检查暂存区域是否和 `lotteryPool` 中存在同样的验证者，如果存在，则跳过。

在以 `k, ok := setValidators[block.Validator]` 开始的代码块中，我们确保了从 `temp` 中取出来的验证者都是合法的，即这些验证者在验证者列表 `validators` 已存在。若合法，则把该验证者加入到 `lotteryPool` 中。

那么我们怎么根据这些验证者持有的令牌数来给予他们合适的随机权重呢？

首先，用验证者的令牌填充 `lotteryPool` 数组，例如一个验证者有100个令牌，那么在 `lotteryPool` 中就会有100个元素填充；如果有1个令牌，那么将仅填充1个元素。



然后，从 `lotteryPool` 中随机选择一个元素，元素所属的验证者即是胜利者，把胜利验证者的地址赋值给 `lotteryWinner`。这里能够看出来，如果验证者持有的令牌越多，那么他在数组中的元素也越多，他获胜的概率就越大；同时，持有令牌很少的验证者，也是有概率获胜的。

接着我们把获胜者的区块添加到整条区块链上，然后通知所有节点关于胜利者的消息：

```
announcements <- "\nwinning validator: " + lotteryWinner + "\n"。
```

最后，清空 `tempBlocks`，以便下次提议的进行。

以上便是PoS一致性算法的核心内容，该算法简单、明了、公正，所以很酷！

收尾

下面我们把之前的内容通过代码都串联起来

main.go

```
func main() {
    err := godotenv.Load()
    if err != nil {
        log.Fatal(err)
    }

    // create genesis block
    t := time.Now()
    genesisBlock := Block{}
    genesisBlock = Block{0, t.String(), 0, calculateBlockHash(genesisBlock), "", ""}
    spew.Dump(genesisBlock)
    Blockchain = append(Blockchain, genesisBlock)

    // start TCP and serve TCP server
    server, err := net.Listen("tcp", ":"+os.Getenv("ADDR"))
    if err != nil {
        log.Fatal(err)
    }
    defer server.Close()

    go func() {
        for candidate := range candidateBlocks {
            mutex.Lock()
            tempBlocks = append(tempBlocks, candidate)
            mutex.Unlock()
        }
    }()

    go func() {
        for {
            pickWinner()
        }
    }()

    for {
        conn, err := server.Accept()
        if err != nil {
            log.Fatal(err)
        }
        go handleConn(conn)
    }
}
```

这里从.env文件开始，然后创建一个创世区块 `genesisBlock`，形成了区块链。接着启动了Tcp服务，等待所有验证者的连接。

启动了一个Go协程从 `candidateBlocks` 通道中获取提议的区块，然后填充到临时缓冲区 `tempBlocks` 中，最后启动了另外一个Go协程来完成 `pickWinner` 函数。

最后面的for循环，用来接收验证者节点的连接。

这里是所有的源代码：[mycoralhealth/blockchain-tutorial \(https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fmycoralhealth%2Fblockchain-tutorial%2Fblob%2Fmaster%2Fproof-stake%2Fmain.go\)](https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fmycoralhealth%2Fblockchain-tutorial%2Fblob%2Fmaster%2Fproof-stake%2Fmain.go)

(/apps/redirect?utm_source=side-banner-click)



结果

下面来运行程序，打开一个终端窗口，通过 `go run main.go` 来启动整个TCP程序，如我们预料，首先创建了创世区块 `genesisBlock`。

```
proof-stake $go run main.go
(main.Block) {
  Index: (int) 0,
  Timestamp: (string) (len=53) "2018-03-25 12:50:32.51411551 -0700 PDT m=+0.001543317",
  BPM: (int) 0,
  Hash: (string) (len=64) "96a296d224f285c67bee93c30f8a309157f0daa35dc5b87e410b78630a09cfc7",
  PrevHash: (string) "",
  Validator: (string) ""
}
```

(/apps/redirect?utm_source=side-banner-click)

接着，我们启动并连接一个验证者。打开一个新的终端窗口，通过linux命令 `nc localhost 9000` 来连接到之前的TCP服务。然后在命令提示符后输入一个持有的令牌数额，最后再输入一个验证者的脉搏速率BPM。

```
proof-stake $nc localhost 9000
Enter token balance:5
Enter a new BPM:60
```

然后观察第一个窗口（主程序），可以看到验证者被分配了地址，而且每次有新的验证者加入时，都会打印所有的验证者列表

```
map[d711ab00b23fbc000fd2d38fc0a0632184e34fcab9a08865b05c69524e2b8b21:5 c6d8d6a0741e6bd633c92681b2b59caf77b4cb4f31e5e8039b13d83d031e9030:12]
```

稍等片刻，检查下你的新窗口（验证者），可以看到正在发生的事：我们的程序在花费时间选择胜利者，然后Boom一声，一个胜利者就诞生了！

```
winning validator: d711ab00b23fbc000fd2d38fc0a0632184e34fcab9a08865b05c69524e2b8b21
```

再稍等一下，boom! 我们看到新的区块链被广播给所有的验证者窗口，包含了胜利者的区块和他的BPM信息。很酷吧！

```
[{"Index":0,"Timestamp":"2018-03-25 12:50:32.51411551 -0700 PDT m=+0.001543317",
"BPM":0,"Hash":"96a296d224f285c67bee93c30f8a309157f0daa35dc5b87e410b78630a09cfc7",
"PrevHash":"","Validator":""},{Index:1,"Timestamp":"2018-03-25 12:50:50.265450636 -0700 PDT m=+17.752783098",
"BPM":60,"Hash":"00e7f03fea6e32e85202f0eacebe78be324bc3eb4823904e581b65ebc6fb3df86",
"PrevHash":"96a296d224f285c67bee93c30f8a309157f0daa35dc5b87e410b78630a09cfc7",
"Validator":"d711ab00b23fbc000fd2d38fc0a0632184e34fcab9a08865b05c69524e2b8b21"}]
```

BPM of winning validator

下一步做什么

你应该为能通过本教程感到骄傲。大多数区块链的发烧友和许多程序员都听说过PoS的证明，但他们很多都无法解释它到底是什么。你已经做得更深入了，而且实际上已经从头开始实现了一遍，你离成为下一代区块链技术的专家又近了一步！

因为这是一个教程，我们可以做更多的事情来让它成为区块链，例如：

- 阅读我们的PoW，然后结合PoS，看看你是否可以创建一个混合区块链
- 添加时间机制，验证者根据时间块来获得提议新区快的概率。我们这个版本的代码让验证者可以在任何时候提议新的区块。
- 添加完整的点对点的功能。这基本上意味着每个验证者将运行自己的TCP服务器，并连接到其他的验证者节点。这里需要添加逻辑，这样每个节点都可以找到彼此，这里 (<https://link.jianshu.com/>)



t=https%3A%2F%2Fbitcoin.stackexchange.com%2Fquestions%2F3536%2Fhow-do-bitcoin-clients-find-each-other)有更多的内容。

或者你可以学习一下我们其它的教程：

(/apps/redirect?utm_source=side-banner-click) ✕

- 200行Go代码编写区块链 (https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fcode-your-own-blockchain-in-less-than-200-lines-of-go-e296282bcffc)
- 使用IPFS在区块链上存储文件 (https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Flearn-to-securely-share-files-on-the-blockchain-with-ipfs-219ee47df54c)
- 区块链网络 (https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fpart-2-networking-code-your-own-blockchain-in-less-than-200-lines-of-go-17fe1dad46e1)
- 从零编写PoW (https://link.jianshu.com?t=https%3A%2F%2Fmedium.com%2F%40mycoralhealth%2Fcode-your-own-blockchain-mining-algorithm-in-go-82c6a71aba1f)

如果我写的任何文章曾在你的心里荡起涟漪，那至少说明在逝去的岁月里，我们在某一刻，共同经历着一样的技术探索之路。
有时候，虽然素未谋面，却已相识很久，很微妙也很知足。

想学习区块链技术，可以搜索公众号**优优区块链课堂**或者添加公众微信号**uuleson**



优优区块链课堂

小礼物走一走，来简书关注我

赞赏支持

区块链 (/nb/23712098)

举报文章 © 著作权归作者所有



Sunface聊技术 (/u/47bf63bd97d2) ♂

写了 46179 字，被 109 人关注，获得了 92 个喜欢

(/u/47bf63bd97d2)

+ 关注

大道至简，技术如歌。本博客关注区块链、微服务、容器云、分布式架构等方向。本博客已经迁移到github...

喜欢



更多分享





下载简书 App ▶

随时随地发现和创作内容



(/apps/redirect?utm_source=note-bottom-click)

(/apps/redirect?utm_source=side-banner-click)





登录 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-comment-form)


评论

智慧如你，不想发表一点想法 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-nocomments-text) 详~

被以下专题收入，发现更多相似内容

- 


区块链研习院 (/c/2d3920d837b8?utm_source=desktop&utm_medium=notes-included-collection)
- 

区块链研习社 (/c/b17f09dc2831?utm_source=desktop&utm_medium=notes-included-collection)
- 

程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)

一文看懂比特币与区块链：从原理到应用 (/p/a43ea6038f9b?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

1 货币的演变——从贝壳到比特币 当社会分工产生之后，人类就产生了商品交换的需求。在货币被发明之前，人类是以以物换物的方式进行的。但显然以物换物存在着商品价值无法精确衡量，效率低下的问题。...

 longlee (/u/bbaf4c459693?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/214acff6c9bd?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)




三种网络模式：中心化、分布式、去中心化

utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

区块链从基础概念到技术实现的进阶攻略 (/p/214acff6c9bd?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

讲师自我介绍：菜根科技技术总监那有涛 大家好，近年来，“区块链”技术迅速走红，其去中心化、去信任的机制得到全球市场的认同，并有望成为下一代“价值互联网”的基础协议。因此，借这样的机会，首先给大...



区块链教育张权 (/u/5e507591d8f1?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/e591e1f9d030?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

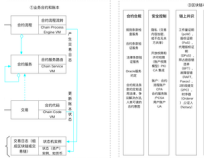
“3点钟无眠区块链”六天干货合辑-帅初、蔡文胜、李笑来、点付大头、薛蛮...

帮大家把连续六天120小时的聊天记录，整理成合集，让各位一次读完。阅读需要大约1小时。请合理安排专门补课时间。欢迎留言探讨。想必区块链所有参与者和关注者都被这个叫“3点钟无眠区块链”的群内马拉松...

 元一同学 (/u/dbf7e90e6d53?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation) (/apps/redirect?utm_source=side-banner-click)


(/p/d93b66410144?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

区块链架构、跨链和演进 (/p/d93b66410144?utm_campaign=maleskine&...

文/庄鹏 本文是基于作者近几年来对各种区块链平台理念和技术的研究，结合作者过去十多年的 IT 经验，审慎思考的结果，文章仅代表作者个人观点。作者会假设读者对各种区块链平台有一定的认知，不会对具体...

 简闻 (/u/0b36d5e1b5fb?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/0ff6b5c17c4d?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

斑马 斑马 (/p/0ff6b5c17c4d?utm_campaign=maleskine&utm_content=n...

许子谦戒烟了，两年前，为了那个站在江边吹口琴的姑娘。好像是在一个四月天吧，疏星点点的夜晚。晚风掠过江面，带来江流的呢喃。岸上行人两三，香樟树影不安分的摇曳。许子谦沿江漫步，低沉而悲伤的旋...

 思雪 (/u/db93f3b00cb6?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/104a95bef3dd?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

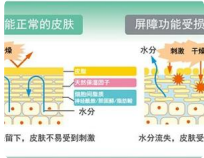
#周检视韩杰#2017年10月第3周（10.16-10.22）1年不到，晋升“能量女神”...

下午好，亲爱的小伙伴 周检视韩杰#2017年10月第3周（10.16-10.22）没有反思的人生不值得过 嘉许自己的每一天的努力 好习惯养成：1，每周运动3次，每次3km或1万步2/33，周检视（4次/月，3/4）4，月检视...

 JIE胭脂雪 (/u/4e7a0d0ef2cc?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/e79bacb00dd8?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

皮肤过敏和敏感皮肤的应对方法（二） (/p/e79bacb00dd8?utm_campaign=...

前篇说到过敏是指当皮肤受到各种刺激后，正在发生的肌肤状态，如湿疹、瘙痒、刺痛、炎症、发红、极度干燥等症状。任何肌肤在某些特定的环境条件或自身防御机制有问题时，都有可能出现过敏症状，不加重...

 ziyou (/u/58973a431a08?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/57b3139223e8?)

utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

【诗】我喜欢的这山间的风 (/p/57b3139223e8?utm_campaign=maleskin...

山间的风吹着广阔的田野，稻田里的谷子连绵起伏。这来自山间的风啊，像温柔的姑娘抚摸我的脸颊，带来的凉爽已让我深深沉陷。山间的风啊，吹着一...



枳春风 (/u/55b6c013c8e5?



(/apps/redirect?
utm_source=side-
banner-click)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

