

区块链原理与实践

实验八 使用Go语言编程实现常见共识机制

实验目的

通过自主设计实验，体验典型共识机制的原理及运作过程，感受区块链系统的奥妙。实验内容主要分为：

- 1. 实验1：工作量证明（PoW）算法实现
- 2. 实验2：权益证明（PoS）算法实现
- 3. 实验3：委托权益证明（DPoS）算法实现

实验步骤

实验1：工作量证明（PoW）算法实现

实验1.1 区块结构

在实验文档所给出的区块结构基础上，我们实现结构体，完成了区块内容的创建，具体而言有结构如下：

```
type block struct {
    Lasthash string //上一个区块的Hash
    Hash string //本区块Hash
    Data string //区块存储的数据（比如比特币UTXO模型 则此处可用于存储交易）
    Timestamp string //时间戳
    Height int //区块高度
    DiffNum uint //难度值
    Nonce int64 //随机数
}
```

实验1.2 设计共识

在已有结构体的基础上，我们完善mine挖矿代码：

```
func mine(data string) Block {
    if len(blockchain) < 1 {
        log.Panic("还未生成创世区块！")
    }

    lastBlock := blockchain[len(blockchain)-1]
    newBlock := Block{
        Lasthash: lastBlock.Hash,
        Data: data,
        Timestamp: time.Now().String(),
        Height: lastBlock.Height + 1,
        DiffNum: 2, // 假设难度值为 2
    }
}
```

```
for {
    newBlock.Nonce++
    newBlock.Hash = newBlock.getHash()
    // 将 newBlock.DiffNum 转换为 int
    if strings.HasPrefix(newBlock.Hash, strings.Repeat("0",
int(newBlock.DiffNum))) {
        break
    }
}

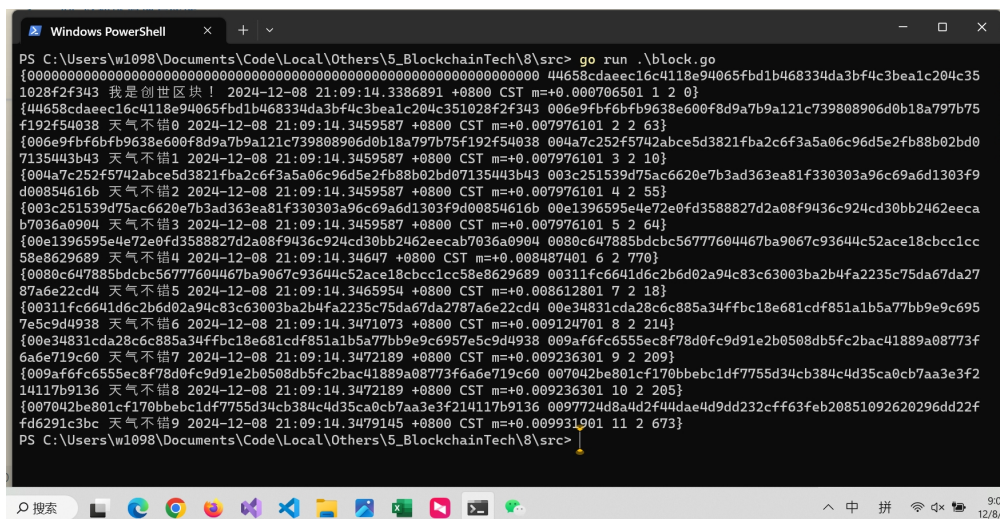
return newBlock
}
```

mine函数首先进行创世区块判断，若已存在区块链，则创立新block并进行相关赋值，并调用如下gethash函数，结合区块信息生成区块哈希。值得注意的是，我们在for循环中不断刷新Nonce随机数、生成新哈希并进行判断。如果满足条件要求（在此为哈希前两位为0），则跳出循环，生成该区块。

```
func (b *Block) getHash() string {
    hashData := fmt.Sprintf("%s%s%s%d%d%d", b.Lasthash, b.Data, b.Timestamp,
b.Height, b.DiffNum, b.Nonce)
    hash := sha256.Sum256([]byte(hashData))
    return hex.EncodeToString(hash[:])
}
```

练习1：根据pow共识原理，完善代码，形成可运行的pow共识模型。

见如上实验报告及压缩包内所附代码



```
PS C:\Users\w1098\Documents\Code\Local\Others\5_BlockchainTech\8\src> go run .\block.go
{0000000000000000000000000000000000000000000000000000000000000000 44658cdaee16c4118e94065fbd1b468334da3bf4c3bealc204c35
1028f2f343 我是创世区块！ 2024-12-08 21:09:14.3386891 +0800 CST m=+0.000706501 1 2 0}
{44658cdaee16c4118e94065fbd1b468334da3bf4c3bealc204c351028f2f343 006e9fbf6bf9638e600f8d9a7b9a121c739808906d0b18a797b75
f192f54038 天气不错0 2024-12-08 21:09:14.3459587 +0800 CST m=+0.007976101 2 2 63}
{006e9fbf6bf9638e600f8d9a7b9a121c739808906d0b18a797b75f192f54038 004a7c252f5742abce5d3821fba2c6f3a5a06c96d5e2fb88b02bd0
7135443b43 天气不错1 2024-12-08 21:09:14.3459587 +0800 CST m=+0.007976101 3 2 10}
{004a7c252f5742abce5d3821fba2c6f3a5a06c96d5e2fb88b02bd07135443b43 003c251539d75ac6620e7b3ad363ea81f330303a96c69a6d1303f9
d00854616b 天气不错2 2024-12-08 21:09:14.3459587 +0800 CST m=+0.007976101 4 2 55}
{003c251539d75ac6620e7b3ad363ea81f330303a96c69a6d1303f9d00854616b 00e1396595e4e72e0fd3588827d2a08f9436c924cd30bb2462eecc
b7036a0904 天气不错3 2024-12-08 21:09:14.3459587 +0800 CST m=+0.007976101 5 2 64}
{00e1396595e4e72e0fd3588827d2a08f9436c924cd30bb2462eeccab7036a0904 0080c647885bdbc56777604467ba9067c93644c52ace18cbcc1cc
58e8629689 天气不错4 2024-12-08 21:09:14.34647 +0800 CST m=+0.008487401 6 2 770}
{0080c647885bdbc56777604467ba9067c93644c52ace18cbcc1cc58e8629689 00311fc6641d6c2b6d02a94c83c63003ba2b4fa2235c75da67da27
87a6e22cd4 天气不错5 2024-12-08 21:09:14.3465954 +0800 CST m=+0.008612801 7 2 18}
{00311fc6641d6c2b6d02a94c83c63003ba2b4fa2235c75da67da2787a6e22cd4 00a34831cda28c6c885a34ffbc18e681cdf851a1b5a77bb9e9c695
7e5c944038 天气不错6 2024-12-08 21:09:14.3471073 +0800 CST m=+0.009124701 8 2 214}
{00a34831cda28c6c885a34ffbc18e681cdf851a1b5a77bb9e9c6957e5c944038 009af6fc6555ec8f78d0fc9d91e2b0508db5fc2bac41889a08773f
6a6e719c60 天气不错7 2024-12-08 21:09:14.3472189 +0800 CST m=+0.009236301 9 2 209}
{009af6fc6555ec8f78d0fc9d91e2b0508db5fc2bac41889a08773f6a6e719c60 007042be801cf170bbebc1df7755d34cb384c4d35ca0cb7aa3e3f2
14117b9136 天气不错8 2024-12-08 21:09:14.3472189 +0800 CST m=+0.009236301 10 2 205}
{007042be801cf170bbebc1df7755d34cb384c4d35ca0cb7aa3e3f214117b9136 0097724d8a4d2f44dae4d9dd232cfff63feb20851092620296dd22f
f66291c3bc 天气不错9 2024-12-08 21:09:14.3479145 +0800 CST m=+0.009931901 11 2 673}
PS C:\Users\w1098\Documents\Code\Local\Others\5_BlockchainTech\8\src>
```

练习2：pow的本质就是看谁能先解出一道数学题，同学们可以按照这种思想设计一种pow类型的共识算法，并且上述代码风格，使用golang实现pow类型的共识算法。

比如我们更换数学难题，命令哈希值对20241208取模后小于difficulty（在此为2024），代码实现如下：

```
// PoW 算法
func (b *Block) mine() {
    const targetMod = 20241208
    const targetValue = 2024

    for {
        b.Nonce++
        b.Hash = b.getHash()
        hashInt, err := strconv.ParseInt(b.Hash, 16, 64)
        if err != nil {
            fmt.Println("哈希转换错误:", err)
            return
        }

        if hashInt%targetMod < targetValue {
            fmt.Printf("成功挖矿! Nonce: %d, Hash: %s\n", b.Nonce, b.Hash)
            break
        }
    }
}
```

实验报告思考题：回答以下问题：

- 如何调整挖矿难度？

可以通过增加或减少 DiffNum 的值来调整：

增大难度值: 要求的前缀 0 位数增加，这会显著降低生成符合条件的哈希的概率，使得挖矿时间变长。

减小难度值: 要求的前缀 0 位数减少，会增加生成符合条件的哈希的概率，使得挖矿时间缩短。

- Pow 挖矿对CPU要求高还是内存要求高？

PoW 挖矿对 CPU 要求高，而对内存要求相对较低，因为挖矿主要依赖于哈希运算，而哈希算法（如 SHA-256）对 CPU 性能有很高要求。

- 你能想出哪些其他的工作量证明的共识实现？

我们可以通过更换数学难题产生不同的PoW机制，如：

素数问题: 找到一个大于某数的第 n 个素数。

哈希反向碰撞: 给定哈希值 H，找到一个消息 M 使得 SHA256(M) 接近 H。

求解方程: 找到满足方程（如多元一次方程组）解的特定数值。

实验2 PoS共识的实现

在原区块添加了 `address string //挖出本块的地址` 的基础上，新增了挖矿节点，使用以下数据，为概率池提供基础支持

Tokens：节点持有的代币数量，决定了节点的持币量。

Days: 质押时间，表示节点已经质押了多长时间。

Address: 节点的地址，用于标识和记录挖矿节点

```
//代表挖矿节点
type node struct {
    tokens int    //代币数量
    days int      //质押时间
    address string //节点地址
}
```

与概率池的初始化：人工添加两个节点后进行初始化随机概率池，具体而言，我们遍历mineNodesPool中的每个节点v，计算其在 probabilityNodesPool 中的出现次数。其中v.tokens * v.days 计算出一个值，这个值决定了节点在 probabilityNodesPool 中的插入次数。通过这种方式，节点在 probabilityNodesPool 中出现的次数与其持币量和质押时间成正比，从而决定了它们在挖矿过程中的概率。持币量和质押时间越高的节点，在概率池中出现的次数越多，被选中的概率也越高。

```
//初始化
func init() {
    //手动添加两个节点
    mineNodesPool = append(mineNodesPool, node{1000, 1, "AAAAAAAAAA"})
    mineNodesPool = append(mineNodesPool, node{100, 3, "BBBBBBBBBB"})
    //初始化随机节点池（挖矿概率与代币数量和币龄有关）
    for _, v := range mineNodesPool {
        for i := 0; i <= v.tokens*v.days; i++ {
            probabilityNodesPool = append(probabilityNodesPool, v)
        }
        fmt.Println(len(probabilityNodesPool))
    }
}
```

我们实现挖矿过程如下：

```
// 获取随机节点地址
func getMineNodeAddress() string {
    rand.Seed(time.Now().UnixNano())
    index := rand.Intn(len(probabilityNodesPool))
    return probabilityNodesPool[index]
}
```

首先初始化随机数种子，使用当前时间的纳秒数 (time.Now().UnixNano()) 作为随机数种子，确保每次运行时的随机性。

然后随机选择节点，使用 rand.Intn 生成一个在 probabilityNodesPool 长度范围内的随机索引。从 probabilityNodesPool 中获取该索引对应的节点地址。

最后根据概率池返回节点地址，通过随机选择 probabilityNodesPool 中的一个节点地址，基于节点在池中的权重（代币数量和质押时间），决定哪一个节点有机会挖出新区块。权重越高的节点在池中出现的次数越多，被选中的概率也越高。

练习1：根据pos共识原理，完善代码，形成可运行的pos共识模型。

具体实现见如上说明及压缩包内代码

```
Windows PowerShell
PS C:\Users\w1098\Documents\Code\Local\Others\5_BlockchainTech\8\src> go run .\block2.go
随机节点池长度: 1300
创世区块: {0000000000000000000000000000000000000000000000000000000000000000 8780f209ee717a4a26a7093deba3266da66073ddf49ca36d43d130a2280ef7 2024-12-08 21:56:45 我是创世区块 1 0000000000}
新区块: {8780f209ee717a4a26a7093deba3266da66073ddf49ca36d43d130a2280ef7 42795f5d67500e0c46fd67b1d2c7b699ba95f2f30311d5ce5b048fb5a8f0941c 2024-12-08 21:56:46 我是第 1 个区块 2 BBBB888888}
新区块: {42795f5d67500e0c46fd67b1d2c7b699ba95f2f30311d5ce5b048fb5a8f0941c 48328a4b815475a57b1db0b69b80871ccb057ce67ed38d97b467bd34896b8c65 2024-12-08 21:56:47 我是第 2 个区块 3 AAAAAAAAAA}
新区块: {48328a4b815475a57b1db0b69b80871ccb057ce67ed38d97b467bd34896b8c65 d190aa53a202956808c6cce33634b85b59868dd663d31e6a1cdc030cc4 4a8cf 2024-12-08 21:56:48 我是第 3 个区块 4 AAAAAAAAAA}
新区块: {d190aa53a202956808c6cce33634b85b59868dd663d31e6a1cdc030cc4 4a8cf 0f703dfb27d1fb1b80066386d5e7a8c52f074e8c76946897cdf47d0669b168a 2024-12-08 21:56:49 我是第 4 个区块 5 AAAAAAAAAA}
新区块: {0f703dfb27d1fb1b80066386d5e7a8c52f074e8c76946897cdf47d0669b168a 93ac69423117f8b27dadf269a198b520e33e7aa1119a865359046b4277e430be 2024-12-08 21:56:50 我是第 5 个区块 6 AAAAAAAAAA}
新区块: {93ac69423117f8b27dadf269a198b520e33e7aa1119a865359046b4277e430be 8c19006a0daab15018f3f95ca0c07dbf89eba5f95f2524ae0a96f0d2b34a94 2024-12-08 21:56:51 我是第 6 个区块 7 AAAAAAAAAA}
新区块: {8c19006a0daab15018f3f95ca0c07dbf89eba5f95f2524ae0a96f0d2b34a94 f9e0da237fee7ef3560cfe59b73c160a6b8b402899f56660d43cd577602243ce 2024-12-08 21:56:52 我是第 7 个区块 8 BBBB888888}
新区块: {f9e0da237fee7ef3560cfe59b73c160a6b8b402899f56660d43cd577602243ce 11982520c897751a538e7bbbd29ce867ee43ae79652c93fbc177bf58c5745a27 2024-12-08 21:56:53 我是第 8 个区块 9 AAAAAAAAAA}
新区块: {11982520c897751a538e7bbbd29ce867ee43ae79652c93fbc177bf58c5745a27 d32294b0e604a77771804db43a862f4de535e225db354a449022a838832f3b02 2024-12-08 21:56:54 我是第 9 个区块 10 AAAAAAAAAA}
新区块: {d32294b0e604a77771804db43a862f4de535e225db354a449022a838832f3b02 230af1beaf8cfe905cf9e1e1026f1bed1183998570baee25642ccb81a438be29 2024-12-08 21:56:55 我是第 10 个区块 11 BBBB888888}
新区块: {230af1beaf8cfe905cf9e1e1026f1bed1183998570baee25642ccb81a438be29 e29e3a43444e623d43c48d8979082e9ee31229de927aef10abe65570e2a9a181 2024-12-08 21:56:56 我是第 11 个区块 12 AAAAAAAAAA}
新区块: {e29e3a43444e623d43c48d8979082e9ee31229de927aef10abe65570e2a9a181 7a307ac78d2f55067f00acf0de94f3c6cd9e536f72f19879e20a90cc466a4c93 2024-12-08 21:56:57 我是第 12 个区块 13 AAAAAAAAAA}
新区块: {7a307ac78d2f55067f00acf0de94f3c6cd9e536f72f19879e20a90cc466a4c93 ed42c6d7c72f8e8c05c2adb4aa116b8dc61180ec0223b63e26dad4866304127a 2024-12-08 21:56:58 我是第 13 个区块 14 AAAAAAAAAA}
新区块: {ed42c6d7c72f8e8c05c2adb4aa116b8dc61180ec0223b63e26dad4866304127a 1f98da9a22fd177373e6f11e62cea20a2b47945168bb566913273f67991b0c8f 2024-12-08 21:56:59 我是第 14 个区块 15 BBBB888888}
```

实验报告思考题：回答以下问题：

- 为什么出块概率需要和质押时间有关？

将质押时间与出块概率挂钩可以激励用户长期持有其资产，而不是频繁买卖。这有助于提高网络的稳定性和安全性。同时如果质押时间越长，出块概率就越高，那么攻击者需要锁定大量资产，且长期保持质押状态，这样可以减少短期攻击的可能性，增强网络的安全性。

- pos 共识相比pow共识具备哪些优点？

PoS 不需要大量的计算能力和电力来解决复杂的数学问题，从而显著降低了能源消耗。同时PoS无需控制挖矿难度，能够实现更快的块生成时间，进而提高交易确认的速度，适合高频交易和应用场景。

实验3 DPoS共识的实现

具体化示例代码如下，我们将得票人从高到低进行排序并选出前十名进行投票：

```
// 按票数选出见证人
func electWitnesses() {
    // 排序，票数从高到低
    sort.Slice(users, func(i, j int) bool {
        return users[i].Votes > users[j].Votes
    })
    // 取前10名作为见证人
    witnessCandidates = users[:10]
    fmt.Println("选出的见证人：")
}
```

```
    for _, witness := range witnessCandidates {  
        fmt.Printf("Address: %s, Votes: %d\n", witness.Address, witness.Votes)  
    }  
}
```

然后在接下来的函数中进行区块生成，并输出生成人相关信息：

```
func shuffleAndMine() {  
    rand.Seed(time.Now().UnixNano())  
    rand.Shuffle(len(witnessCandidates), func(i, j int) {  
        witnessCandidates[i], witnessCandidates[j] = witnessCandidates[j],  
witnessCandidates[i]  
    })  
    fmt.Println("见证人随机顺序: ")  
    for _, witness := range witnessCandidates {  
        fmt.Printf("Address: %s, Votes: %d\n", witness.Address, witness.Votes)  
    }  
  
    // 模拟区块生成  
    for i, witness := range witnessCandidates {  
        fmt.Printf("见证人 %s 打包了区块 #%d\n", witness.Address, i+1)  
    }  
}
```

作为相关辅助函数，我们还实现了用户投票与撤销投票函数：

```
// 用户投票  
func vote(voterIndex, candidateIndex int, voteTokens int) {  
    if voteTokens > users[voterIndex].Tokens {  
        fmt.Println("投票失败：投票数量超过持有代币数量")  
        return  
    }  
  
    // 扣除投票者代币  
    users[voterIndex].Tokens -= voteTokens  
    // 增加候选人票数  
    users[candidateIndex].Votes += voteTokens  
    fmt.Printf("%s 投票给 %s, 票数增加 %d\n", users[voterIndex].Address,  
users[candidateIndex].Address, voteTokens)  
}  
  
// 用户撤销投票  
func revokeVote(voterIndex, candidateIndex int, revokeTokens int) {  
    // 减少候选人票数  
    if revokeTokens > users[candidateIndex].Votes {  
        fmt.Println("撤票失败：撤票数量超过候选人的票数")  
        return  
    }  
    users[candidateIndex].Votes -= revokeTokens
```



```
// 退回给投票者代币
users[voterIndex].Tokens += revokeTokens
fmt.Printf("%s 撤销对 %s 的投票, 票数减少 %d\n", users[voterIndex].Address,
users[candidateIndex].Address, revokeTokens)
}
```

最后我们在main函数中进行示例投票，并依次调用相关函数进行DPoS模拟：

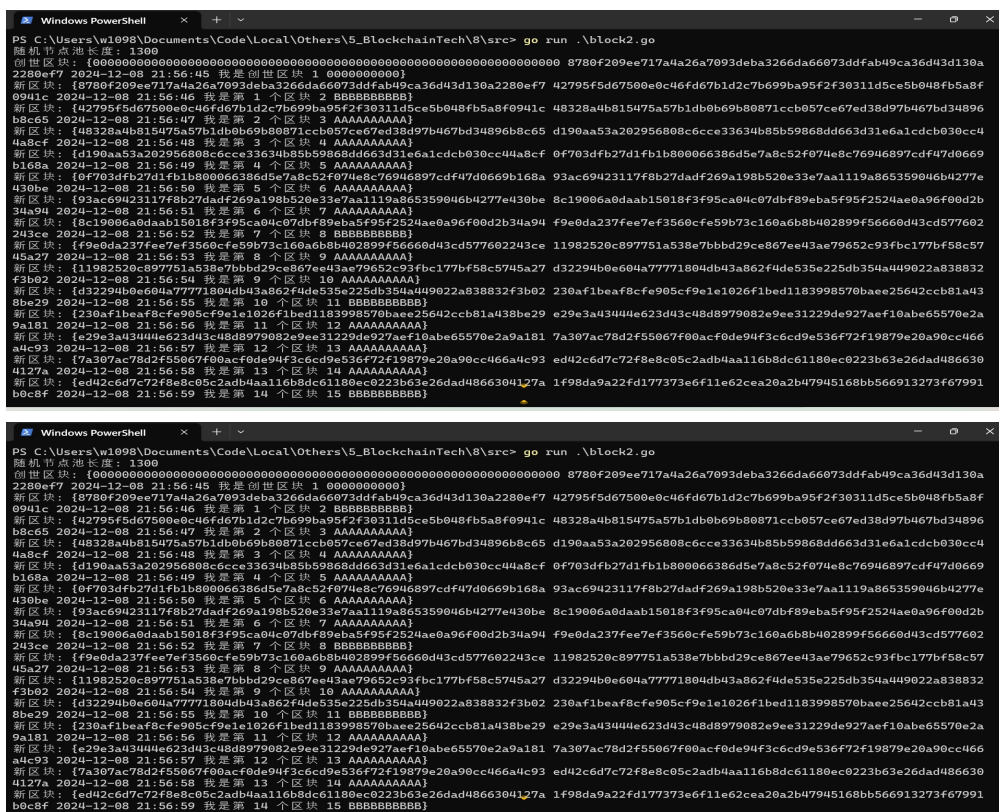
```
func main() {
    // 初始化用户
    initUsers()

    // 模拟投票
    vote(0, 3, 300)
    vote(1, 4, 200)
    vote(2, 3, 500)
    vote(3, 5, 800)

    // 模拟撤票
    revokeVote(0, 3, 100)

    // 选举见证人
    electWitnesses()

    // 随机化见证人顺序并挖矿
    shuffleAndMine()
}
```



```
PS C:\Users\w1098\Documents\Code\Local\Others\5_BlockchainTech\8\src> go run .\block2.go
随机节点池长度: 1300
创世区块: {0000000000000000000000000000000000000000000000000000000000000000 8780f209ee717a4a26a7093deba3266da66073ddfab49ca36d43d130a
2280ef7 2024-12-08 21:56:45 我是创世区块 1 0000000000}
新区块: {8780f209ee717a4a26a7093deba3266da66073ddfab49ca36d43d130a2280ef7 42795f5d67500e0c46fd67b1d2c7b699ba95f2f30311d5ce5b048fb5a8f
0941c 2024-12-08 21:56:46 我是第 1 个区块 2 BBBB888888}
新区块: {42795f5d67500e0c46fd67b1d2c7b699ba95f2f30311d5ce5b048fb5a8f0941c 48328a4b815475a57b1db0b69b80871ccb857ce67ed38d97b467bd34896
b8c65 2024-12-08 21:56:47 我是第 2 个区块 3 AAAAAAAAAA}
新区块: {48328a4b815475a57b1db0b69b80871ccb857ce67ed38d97b467bd34896b8c65 d190aa53a202956808c6cce33634b85b59868dd663d31e6a1cdcb030cc4
4a8cf 2024-12-08 21:56:48 我是第 3 个区块 4 AAAAAAAAAA}
新区块: {d190aa53a202956808c6cce33634b85b59868dd663d31e6a1cdcb030cc44a8cf 0f703dfb27d1fb1b800066386d5e7a8c52f074e8c76946897cdf47d0669
b168a 2024-12-08 21:56:49 我是第 4 个区块 5 AAAAAAAAAA}
新区块: {0f703dfb27d1fb1b800066386d5e7a8c52f074e8c76946897cdf47d0669b168a 93ac69423117f8b27daf269a198b520e33e7aa1119a865359046b4277e
430be 2024-12-08 21:56:50 我是第 5 个区块 6 AAAAAAAAAA}
新区块: {93ac69423117f8b27daf269a198b520e33e7aa1119a865359046b4277e430be 8c19006a0daab15018f3f95ca04c07dbf89eba5f95f2524ae0a96f00d2b
34a94 2024-12-08 21:56:51 我是第 6 个区块 7 AAAAAAAAAA}
新区块: {8c19006a0daab15018f3f95ca04c07dbf89eba5f95f2524ae0a96f00d2b34a94 f9e0da237fee7ef3560cfe59b73c160a6b8b402899f56660d43cd577602243ce
243ce 2024-12-08 21:56:52 我是第 7 个区块 8 BBBB888888}
新区块: {f9e0da237fee7ef3560cfe59b73c160a6b8b402899f56660d43cd577602243ce 11982520c897751a538e7bbbd29ce867ee43ae79652c93fbc177bf58c57
45a27 2024-12-08 21:56:53 我是第 8 个区块 9 AAAAAAAAAA}
新区块: {11982520c897751a538e7bbbd29ce867ee43ae79652c93fbc177bf58c5745a27 d32294b0e604a77771804db43a862f4de535e225db354a449022a838832f3b02
f3b02 2024-12-08 21:56:54 我是第 9 个区块 10 AAAAAAAAAA}
新区块: {d32294b0e604a77771804db43a862f4de535e225db354a449022a838832f3b02 230af1beaf8cfe905cf9e1e026f1bed1183998570baee25642ccb81a43
8be29 2024-12-08 21:56:55 我是第 10 个区块 11 BBBB888888}
新区块: {230af1beaf8cfe905cf9e1e026f1bed1183998570baee25642ccb81a438be29 e29e3a43444e623d43c48d8979082e9ee31229de927aef10abe65570e2a
9a181 2024-12-08 21:56:56 我是第 11 个区块 12 AAAAAAAAAA}
新区块: {e29e3a43444e623d43c48d8979082e9ee31229de927aef10abe65570e2a9a181 7a307ac78d2f55867f00acf0de94f3c6cd9e536f72f19879e20a90acc466
a4c93 2024-12-08 21:56:57 我是第 12 个区块 13 AAAAAAAAAA}
新区块: {7a307ac78d2f55867f00acf0de94f3c6cd9e536f72f19879e20a90acc466a4c93 ed42c6d7c72f8e8c05c2adb4aa116b8dc61180ec0223b63e26dad486630
4127a 2024-12-08 21:56:58 我是第 13 个区块 14 AAAAAAAAAA}
新区块: {ed42c6d7c72f8e8c05c2adb4aa116b8dc61180ec0223b63e26dad4866304127a 1f98da9a22fd177373e6f11e62cea20a2b47945168bb566913273f67991
b0c8f 2024-12-08 21:56:59 我是第 14 个区块 15 BBBB888888}
```