用Go构建区块链——1.基本原型

本篇开始进入"用Go构建区块链"系列，主要对原文进行翻译。对应原文如下：

Building Blockchain in Go. Part 1: Basic Prototype

话不多说，开始进入正文

1、介绍

区块链是21世纪最具革命性的技术之一，它仍在逐步发展中，并且其潜力还未被充分认识。本质上，区块链只是一个分布式数据库而已。但是，它的独特之处在于它不是一个私有的数据库，而是一个公共的，即每个使用它的人都拥有它的全部或部分副本。只有得到其他数据库管理员的同意，新的记录才能被加入。正因为由此区块链，才使得加密货币和智能合约成为可能。

在本系列文章中，我们将构建一个基于简单区块链实现的简单加密货币。

2、区块

我们从"区块链"的"区块"部分开始。在区块链中，它存储有价值的信息。例如，比特币块存储交易信息，这是所有加密货币的本质。除此之外，区块还包含一些技术信息，如版本号、当前时间戳和上一个区块的哈希。

在本文中，我们不会实现区块链中描述的区块，也不会是比特币技术规范中的区块，而是使用简化版本，其中只包含重要信息。这是它的样子：

```
type Block struct {
    Timestamp     int64
    Data          []byte
    PrevBlockHash []byte
    Hash          []byte
}
```

`Timestamp` 是当前时间戳(区块被创建时)，`Data` 是包含在区块中的实际有价值的信息，而 `Hash` 是当前区块的哈希。在比特币技术规范中， `Timestamp` ， `PrevBlockHash` 和 `Hash` 是区块头，它们形成一个单独的数据结构，而交易（在我们的例子中是数据）是一个独立的数据结构。我们这里简单起见，混合在一起了。

那么，怎么计算哈希呢？哈希的计算方式在区块链中是一个非常重要的特性，正是这一特性使得区块链更加安全。问题是计算哈希是一个难以计算的操作，即使在很快的计算机上也需要话费很多时间（这就是为什么人们购买强大的GPU来挖比特币）。这是一个架构上有意为之的设计，这

使得添加新的区块变得困难，从而阻止添加后的修改。我们将在接下来的文章中去讨论和实现这个机制。

现在，我们只取了区块字段，并把它们拼接起来，并在连接的组合上计算SHA-256哈希。 让我们在 `SetHash` 方法中完成这些操作：

```
func (b *Block) SetHash() {
    timestamp := []byte(strconv.FormatInt(b.Timestamp, 10))
    headers := bytes.Join([][]byte{b.PrevBlockHash, b.Data, timestamp}, []by
te{})
    hash := sha256.Sum256(headers)

    b.Hash = hash[:]
}
```

接下来，按照Golang的约定，我们将实现一个将简化创建区块的函数：

```
func NewBlock(data string, prevBlockHash []byte) *Block {
    block := &Block{time.Now().Unix(), []byte(data), prevBlockHash, []byte{}
}
    block.SetHash()
    return block
}
```

好了，这就是区块！

3、区块链

现在我们来实现一个区块链。其本质区块链仅仅是一个具有特定结构的数据库：它是一个有序的，尾部相连的链表。这意味着区块按照插入的顺序来存储，并且每个区块都链接着前一个区块。该结构允许快速获取链中的最新块，并通过其哈希（有效）获取区块。

在Golang中，这个结构可以通过使用array和map来实现：array存储有序的哈希 (Golang中，array是有序的)，并且 map 结构可以保存 `hash -> block` 的匹配信息。但在我们的区块链原型中，我们只会使用一个array，因为我们暂时并不需要通过 哈希来获取区块信息。

```
type Blockchain struct {
  blocks []*Block
}
```

这是我们的第一块区块链！我从未想过它会如此轻松😉

现在，让我们能够给它添加一个区块：

```
func (bc *Blockchain) AddBlock(data string) {
    prevBlock := bc.blocks[len(bc.blocks)-1]
    newBlock := NewBlock(data, prevBlock.Hash)
    bc.blocks = append(bc.blocks, newBlock)
}
```

就这样！还是？

要添加新的区块，我们需要一个已存在的区块，然而我们的区块链上还没有一个区块！因此，在任何区块链中，必须至少有一个区块，而这个区块是链中的第一个区块，称为创世区块。让我们实现一个创建创世区块的函数：

```
func NewGenesisBlock() *Block {
    return NewBlock("Genesis Block", []byte{})
}
```

现在，我们可以实现一个创建包含创世区块的区块链的函数：

```
func NewBlockchain() *Blockchain {
    return &Blockchain{[]*Block{NewGenesisBlock()}}
}
```

现在，让我们来检查一下区块链是否正常工作：

```
func main() {
    bc := NewBlockchain()

    bc.AddBlock("Send 1 BTC to Ivan")
    bc.AddBlock("Send 2 more BTC to Ivan")

    for _, block := range bc.blocks {
        fmt.Printf("Prev. hash: %x\n", block.PrevBlockHash)
        fmt.Printf("Data: %s\n", block.Data)
        fmt.Printf("Hash: %x\n", block.Hash)
        fmt.Println()
    }
}
```

输出：

```
Prev. hash:
Data: Genesis Block
Hash: aff955a50dc6cd2abfe81b8849eab15f99ed1dc333d38487024223b5fe0f1168

Prev. hash: aff955a50dc6cd2abfe81b8849eab15f99ed1dc333d38487024223b5fe0f1168
Data: Send 1 BTC to Ivan
Hash: d75ce22a840abb9b4e8fc3b60767c4ba3f46a0432d3ea15b71aef9fde6a314e1

Prev. hash: d75ce22a840abb9b4e8fc3b60767c4ba3f46a0432d3ea15b71aef9fde6a314e1
Data: Send 2 more BTC to Ivan
Hash: 561237522bb7fcfbccbc6fe0e98bbbde7427ffe01c6fb223f7562288ca2295d1
```

没错，就这样！

4、总结

我们构建了一个非常简单的区块链原型：它只是一个包含有区块的数组，每个区块和前一个相连接。真实的区块链比这个复杂得多。在我们的区块链中，添加新的区块非常简单而且很好，但是在真实的区块链中添加新区块需要做很多工作：一是需要在添加区块前做一些复杂的计算来获取添加区块的权限(这个过程被称为 `工作量证明`)。另外，区块链是一个分布式数据库，其没有单一的决策者。因此，一个新的区块必须得到网络的其他参与者的确认和同意（这种机制被称为共识）。而且，我们的区块链还没有交易！

在以后的文章中，我们将介绍这些功能。

链接：
1.获取源码：https://github.com/Jeiwan/blockchain_go/tree/part_1
2.区块哈希算法：https：//en.bitcoin.it/wiki/Block_hashing_algorithm