lab3

PB21051012 刘祥辉

POW部分

```
func (pow *ProofOfWork) prepareData(nonce int) []byte {
    data := bytes.Join(
        [][]byte{
            IntToHex(pow.block.Header.Version),    //版本号
            pow.block.Header.PrevBlockHash[:],     //前一个区块的哈希值
            pow.block.Header.MerkleRoot[:],        //默克尔树根,当前区块数据对应哈希值
            IntToHex(pow.block.Header.Timestamp), //时间戳
            IntToHex(targetBits),                  //难度值
            IntToHex(int64(nonce)),                //随机数
        },
        []byte{},
    )

    return data
}


func (pow *ProofOfWork) Run() (int64, []byte) {
    var hashInt big.Int
    var hash [32]byte
    var nonce  = 0 // 将 nonce 类型改为 int64

    fmt.Printf("Mining a new block")
    for nonce < maxNonce {
        data := pow.prepareData(nonce)

        hash = sha256.Sum256(data)
        hashInt.SetBytes(hash[:])

        if hashInt.Cmp(pow.target) == -1 {
            fmt.Printf("\r%x", hash)
            break
        } else {
            nonce++
        }
    }
    fmt.Print("\n\n")
    return int64(nonce) , hash[:]
}
```

```
func (pow *ProofOfWork) Validate() bool {
    var hashInt big.Int

    data := pow.prepareData(int(pow.block.Header.Nonce))
    hash := sha256.Sum256(data)
    hashInt.SetBytes(hash[:])

    isValid := hashInt.Cmp(pow.target) == -1

    return isValid
}
```

UTXO池部分

```
func (u UTXOSet) FindUnspentOutputs(pubkeyHash []byte, amount int) (int,
map[string][]int) {
    unspentOutputs := make(map[string][]int)
    accumulated := 0
    db := u.Blockchain.db

    _ = db.View(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(utxoBucket))
        c := b.Cursor()

        for k, v := c.First(); k != nil; k, v = c.Next() {
            txID := hex.EncodeToString(k)
            outs := DeserializeOutputs(v)

            for outIdx, out := range outs.Outputs {
                if out.IsLockedWithKey(pubkeyHash) && accumulated < amount {
                    accumulated += out.Value
                    unspentOutputs[txID] = append(unspentOutputs[txID], outIdx)
                }
            }
        }
        return nil
    })

    return accumulated, unspentOutputs
}
```

注意数据库的读取即可

Blockchain部分

```
func (bc *Blockchain) MineBlock(transactions []*Transaction) *Block {
    var prevBlockHash [32]byte
    copy(prevBlockHash[:], bc.tip)
    newBlkHeader := NewBlkHeader(transactions, prevBlockHash)
    newBlkBody := NewBlkBody(transactions)
    newBlk := &Block{newBlkHeader, newBlkBody}

    pow := NewProofOfWork(newBlk)
    nonce, _ := pow.Run()
    newBlk.Header.Nonce = nonce
```

```
    bc.tip = newBlk.CalCulHash()

    bc.db.Update(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(blocksBucket))
        err := b.Put(newBlk.CalCulHash(), newBlk.Serialize())
        if err != nil {
            log.Panic(err)
        }
        err = b.Put([]byte("l"), newBlk.CalCulHash())
        if err != nil {
            log.Panic(err)
        }
        return nil
    })

    return newBlk
}
```

1. **初始化前一个区块的哈希值**：
    - 获取当前区块链的最后一个区块的哈希值，作为新区块的前一个区块哈希值。
2. **创建新的区块头和区块体**：
    - 创建一个新的区块头和区块体，用传入的交易列表和前一个区块的哈希值初始化。
3. **运行工作量证明（PoW）算法**：
    - 执行工作量证明算法，找到符合要求的 `nonce` 值，使得新区块的哈希值满足指定的条件。
4. **更新区块链的顶端（tip）**：
    - 将新区块的哈希值设置为区块链的顶端，以反映最新状态。
5. **将新块保存到数据库**：
    - 将新挖掘的区块序列化并存储到数据库中，并更新数据库中记录的最后一个区块的哈希值。
6. **返回新块**：
    - 返回新挖掘出的区块对象，供调用者进一步处理或展示。

```
func (bc *Blockchain) FindUTXO() map[string]TXOutputs {
    var UTXO = make(map[string]TXOutputs) //创建一个map，key是string，value是[]int，
用来存储交易ID和交易输出索引
    db := bc.db                           //打开数据库
    err := db.View(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(utxoBucket)) //打开bucket
        c := b.Cursor()

        for k, v := c.First(); k != nil; k, v = c.Next() {
            outs := DeserializeOutputs(v) //反序列化交易输出
            txID := hex.EncodeToString(k) //交易ID
            UTXO[txID] = outs
        }
        return nil
    })
    if err != nil {
        log.Panic(err)
    }
    return UTXO
}
```

Transaction部分

```
func NewUTXOTransaction(from, to []byte, amount int, UTXOSet *UTXOSet)
*Transaction {
    var inputs []TXInput
    var outputs []TXOutput

    wallets, err := NewWallets()
    if err != nil {
        log.Panic(err)
    }
    wallet := wallets.GetWallet(from)
    pubkeyHash := HashPublicKey(wallet.PublicKey)
    //查询当前要转出用户的余额
    acc, validOutputs := UTXOSet.FindUnspentOutputs(pubkeyHash, amount)
    //如果余额不足，退出
    if acc < amount {
        log.Panic("ERROR: Not enough funds")
    }

    for txid, outs := range validOutputs {
        txID, err := hex.DecodeString(txid)
        if err != nil {
            log.Panic(err)
        }

        for _, out := range outs {
            input := TXInput{txID, out, nil, wallet.PublicKey}
            inputs = append(inputs, input)
        }
    }

    outputs = append(outputs, *NewTXOutput(amount, to))
    fmt.Println("hash:", HashPublicKey(to))
    fmt.Println("to:", to)
    if acc > amount {
        outputs = append(outputs, *NewTXOutput(acc-amount, from))
    }

    tx := Transaction{nil, inputs, outputs}
    tx.SetID()
    UTXOSet.Blockchain.SignTransaction(&tx, wallet.PrivateKey)
    return &tx

}
```