

课程回顾补充

%s: []byte可能会出现乱码

%x: 正常展示

终端如果出现乱码，可以使用 `reset` 命令进行重置。

check.sh优化：

```
#!/bin/bash

from=1Fakfxjba4LwEtNVUJnz9erXqjgBeRzvuz
to=1CAu5rZtzWfYnN2KpMUwan9LXhJ75H1eoX
miner=19dpiTubN8ty2Ji5JTTrTspbuYMhnuuq888

./blockchain send $from $to 10 $miner "hello world"

./blockchain getBalance $from #2.5
./blockchain getBalance $to #10
./blockchain getBalance $miner #12.5
```

下标访问注意：

所有涉及到切片下标的操作之前，都要校验一下数组|切片的长度。防止访问越界。

Lsh: left shift :左移

Rsh: right shift 右移

签名相关

需要：

1. 私钥 ==> 付款人的私钥
2. 想要签名的数据src ==> 交易

校验:

1. 公钥 ==》矿工拿着交易里面的携带的公钥
2. 需要校验的数据src ==》通过接收到的交易生成的数据
3. 数字签名 ==》从交易的sig字段中获取

到底对什么进行签名?

1. 签名: 对当前的这笔交易进行
2. 这个交易里面要包含哪些数据呢:
 1. 新生成的output的公钥哈希签名 (这描述了收款人)
 2. output的金额 (描述了转账金额)
 3. 所引用的utxo的锁定的公钥哈希 (这描述了付款人)

校验签名



创建交易副本

```
//创建当前交易的副本 (裁剪)
//Trim 修剪
func (tx *Transaction) TrimmedTransactionCopy() *Transaction {
    //将input的sig和pubkey字段设置成nil
    var inputs []TXInput
    var outputs []TXOutput

    //遍历input
    for _, input := range tx.TxInputs {
```

```

        inputNew := TXInput{
            TXID:      input.TXID,
            Index:     input.Index,
            ScriptSig: nil,
            PubKey:    nil,
        }

        inputs = append(inputs, inputNew)
    }

    //遍历output
    copy(outputs, tx.TXOutputs)

    txCopy := Transaction{
        Txid:      tx.Txid,
        TxInputs:   inputs,
        TXOutputs:  outputs,
        TimeStamp: tx.TimeStamp, //<< 不要使用当前时间，否则矿工校验时的数据一定会改变
    }

    return &txCopy
}

```

Sign函数

```

//具体签名函数
func (tx *Transaction) Sign(priKey *ecdsa.PrivateKey, prevTxs
map[string]*Transaction) bool {
    fmt.Printf("开始具体签名动作: Sign ...\n")
    //所有的签名细节在此处实现
    //TODO

    return true
}

```

SingTransaction

```

//签名相关
func (bc *BlockChain) SignTransaction(priKey *ecdsa.PrivateKey, tx *Transaction)
bool {
    fmt.Printf("开始签名: SignTransaction called!\n")
    if tx.isCoinbaseTx() {
        fmt.Println("发现挖矿交易，不需要签名!")
        return true
    }

    //1. 查到tx所引用的交易的集合

```

```

var prevTxs map[string]*Transaction

//TODO

return tx.Sign(priKey, prevTxs)
}

```

调用:

```

tx := Transaction{
    TxInputs: inputs,
    TXOutputs: outputs,
    TimeStamp: time.Now().Unix(),
}

//3. 设置交易id
tx.SetTxId()

//4. 对当前交易进行签名
bc.SignTransaction(priKey, &tx)

//4. 返回
return &tx, nil
}

```

校验Verify

transaction.go

```

//具体的验证函数
func (tx *Transaction) Verify(prevTxs map[string]*Transaction) bool {
    fmt.Printf("开始具体校验动作: verify ...\n")
    //TODO
    return true
}

```

blockchain.go

```

func (bc *BlockChain) VerifyTransaction(tx *Transaction) bool {
    fmt.Printf("开始校验: verifyTransaction called!\n")
    if tx.isCoinbaseTx() {
        fmt.Println("发现挖矿交易, 不需要校验!")
        return true
    }

    prevTxs := make(map[string]*Transaction)

    //TODO
}

```

```

    return tx.Verify(prevTx)
}

```

调用:

```

//1 <- 2 <-3
//添加区块的方法
func (bc *Blockchain) AddBlock(txs []*Transaction) {
    fmt.Println("AddBlock called!")

    //所有校验通过的交易集合，最终打包到区块 <<=====
    var validTxs []*Transaction

    //对每一条交易进行校验
    for _, tx := range txs {
        if bc.VerifyTransaction(tx) {
            validTxs = append(validTxs, tx)
        } else {
            fmt.Printf("发现签名校验失败的交易:%x\n", tx.Txid)
        }
    }

    //最后一个区块的哈希值
    lastHash := bc.tail

    //1. 创建新的区块
    newBlock := NewBlock(validTxs, lastHash) <<=====

    //...省略
}

```

效果:

```

开始具体签名动作: Sign ...
发现有效的交易，准备添加到区块，txid:75cbd0fc26cc2cabc4e1e98dfbcd91c9e1b67d61434f0c75d3
AddBlock called!
开始签名: SignTransaction called!
发现挖矿交易，不需要校验!
开始签名: SignTransaction called!      开始校验, VerifyTransaction called!
开始具体校验动作: Verify ...
挖矿成功，当前哈希值为:00003635af1f155a83dfda36aeaa7151f2763ee31554238995a400ab4dca7b78
lastHash : 00003635af1f155a83dfda36aeaa7151f2763ee31554238995a400ab4dca7b78
CLI Run called!
getBalance called!

```

FindTransactionByTxid

```

//根据txid找到交易本身
func (bc *Blockchain) FindTransactionByTxid(txid []byte) *Transaction {
    it := NewIterator(bc)

    for {
        block := it.Next()

        for _, tx := range block.Transactions {
            if bytes.Equal(tx.Txid, txid) {
                return tx
            }
        }

        if len(block.PrevHash) == 0 {
            break
        }
    }

    return nil
}

```

实现查找所引用交易的map结构

```

//签名相关
func (bc *Blockchain) SignTransaction(prikey *ecdsa.PrivateKey, tx *Transaction)
bool {
    fmt.Printf("开始签名: SignTransaction called!\n")
    if tx.IsCoinbaseTx() {
        fmt.Println("发现挖矿交易，不需要签名!")
        return true
    }

    //1. 查到tx所引用的交易的集合    <<<=====
    //key: txid
    //value:tx本身
    prevTxs := make(map[string]*Transaction)
    //遍历当前交易的input，通过txid找到每个input的tx，赋值给map

    for _, input := range tx.TxInputs {
        tx := bc.FindTransactionByTxid(input.TXID)
        if tx == nil {
            fmt.Println("没有找到交易，txid:", input.TXID)
            return false
        }

        //将找到的交易放到集合中
        fmt.Printf("找到签名时所引用的交易,txid: %x\n", tx.Txid)
        prevTxs[string(input.TXID)] = tx
    }    <<<=====

    return tx.Sign(prikey, prevTxs)
}

```

```
}
```

效果:

```
FindNeedUtxoInfo called, pubKeyHash :9ff3f16b6374eee60f67f340f8ccd1d4e629bc39, amount:10.000000
FindMyUtxo called, address:9ff3f16b6374eee60f67f340f8ccd1d4e629bc39
找到了属于'9ff3f16b6374eee60f67f340f8ccd1d4e629bc39'的output, index:0, value:12.500000
开始签名: SignTransaction called!
找到签名时所引用的交易,txid: a25784646b52b34e00bcf85b3b87b719cfeba38cf3fccf92eeacab4886f5d967
开始具体签名动作: Sign ...
发现有效的交易, 准备添加到区块, txid:b52f4f974d2d1215ceda324e730ed8d0000cdf3805aabc8e135d2798c73943790
AddBlock called!
开始校验: VerifyTransaction called!
```

VerifyTransaction实现

```
func (bc *Blockchain) VerifyTransaction(tx *Transaction) bool {
    fmt.Printf("开始校验: VerifyTransaction called!\n")
    if tx.isCoinbaseTx() {
        fmt.Println("发现挖矿交易, 不需要校验!")
        return true
    }

    prevTxs := make(map[string]*Transaction)

    for _, input := range tx.TxInputs { //<<<====
        tx := bc.FindTransactionByTxid(input.TXID)
        if tx == nil {
            fmt.Println("没有找到交易, txid:", input.TXID)
            return false
        }

        //将找到的交易放到集合中
        fmt.Printf("找到校验时所引用的交易,txid: %x\n", tx.Txid)
        prevTxs[string(input.TXID)] = tx
    } //<<<====

    return tx.Verify(prevTxs)
}
```

Sign函数实现

1. 获取交易副本txCopy
2. 遍历txCopy里面的input
3. 使用这个input所引用的output来填充每一个input的pubKey字段
4. 对当前交易做哈希处理, 得到需要签名的数据
5. 使用私钥进行签名: sig
6. 将签名赋值给原始交易的input.ScriptSig字段

7. 将当前input的pubKey字段设置成nil

```
//具体签名函数
func (tx *Transaction) Sign(prikey *ecdsa.PrivateKey, prevTxes
map[string]*Transaction) bool {
    fmt.Printf("开始具体签名动作: Sign ...\n")
    //所有的签名细节在此处实现
    //1. 获取交易副本txCopy
    txCopy := tx.TrimmedTransactionCopy()

    //2. 遍历txCopy里面的input
    for i, input := range txCopy.TxInputs {
        prevtx := prevTxes[string(input.TXID)]
        if prikey == nil {
            return false
        }

        //3. 使用这个input所引用的output来填充每一个input的pubkey字段
        output := prevtx.TxOutputs[input.Index] //<< == 不是i变量
        //input.PubKey = output.PubKeyHash <== 这个input是副本，不要对它进行操作
        txCopy.TxInputs[i].PubKey = output.PubKeyHash

        //4. 对当前交易做哈希处理，得到需要签名的数据
        txCopy.SetTxId() //这个函数就是获取了当前交易的哈希值
        hashData := txCopy.Txid

        fmt.Printf("====> 签名内容的哈希值: %x\n", hashData)

        //5. 使用私钥进行签名: sig
        r, s, err := ecdsa.Sign(rand.Reader, prikey, hashData[:])
        if err != nil {
            fmt.Println("ecdsa.Sign err: ", err)
            return false
        }

        //将r, s拼接成一个字节流，生成当前的签名
        signature := append(r.Bytes(), s.Bytes()...)

        //6. 将签名赋值给原始交易的input.ScriptSig字段
        tx.TxInputs[i].ScriptSig = signature

        //7. 将当前input的pubkey字段设置成nil
        txCopy.TxInputs[i].PubKey = nil
        txCopy.Txid = nil //在SetTxId时，已经修改了，需要还原
    }

    fmt.Println("交易签名成功!")

    return true
}
```


TrimmedCopy里面做修改如下:

```
9 //遍历output
9 copy(outputs, tx.TXOutputs)
1
2 txCopy := Transaction{
3     Txid:      nil,
4     TxInputs:  inputs,
5     TXOutputs: outputs,
6     TimeStamp: tx.TimeStamp, //<< 不要使用当前时间, 否则破
7 }
8
9 return &txCopy
0 }
```

```
FindNeedUtxoInfo called, pubkeyHash :9ff3f16b6374eee60f67f340f8ccd1d4e629bc39, amount:10.000000
FindMyUtxo called, address:9ff3f16b6374eee60f67f340f8ccd1d4e629bc39
找到了属于'9ff3f16b6374eee60f67f340f8ccd1d4e629bc39'的output, index:0, value:12.500000
开始签名: SignTransaction called!
找到签名时所引用的交易,txid: e95e22123ce9faec3456cd727fbfdc9ed6535a03104a43cc4263c529917ba3c3
开始具体签名动作: Sign ...
====> 签名内容的哈希值: 367fc58e5f3395b659b52a87befbbc774673db0c96397e5546df4509774e2ff4
交易签名成功!
发现有效的交易, 准备添加到区块, txid:66a24e00933165c0a8df2e4828f0850ddd874f1a3d0e1a7ce53ddf965645
AddBlock called!
开始检验: VerifyTransaction called!
```