

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(prevTxs)
}

```

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:75cbd0fc26cc2cab4e1e98dfbcd91c9e1b67d61434f0c75d3
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!
```