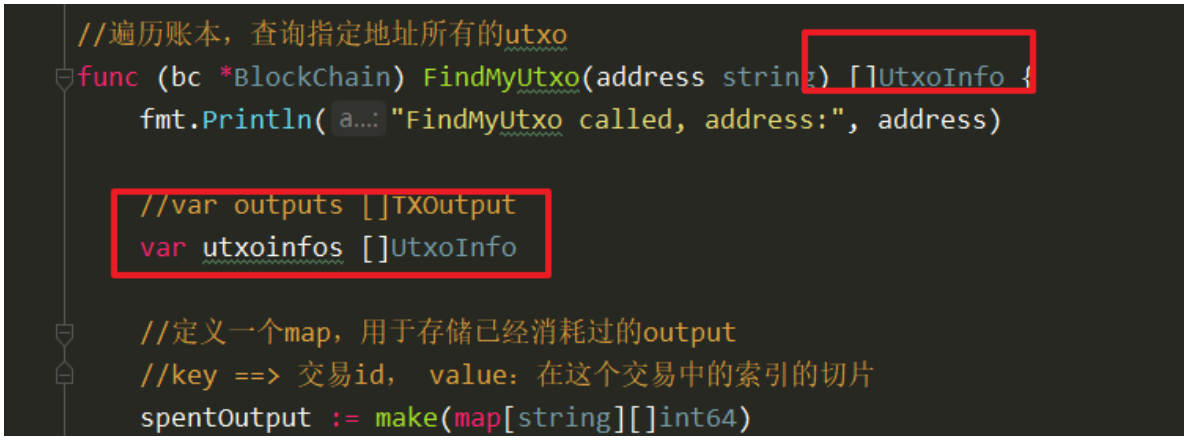
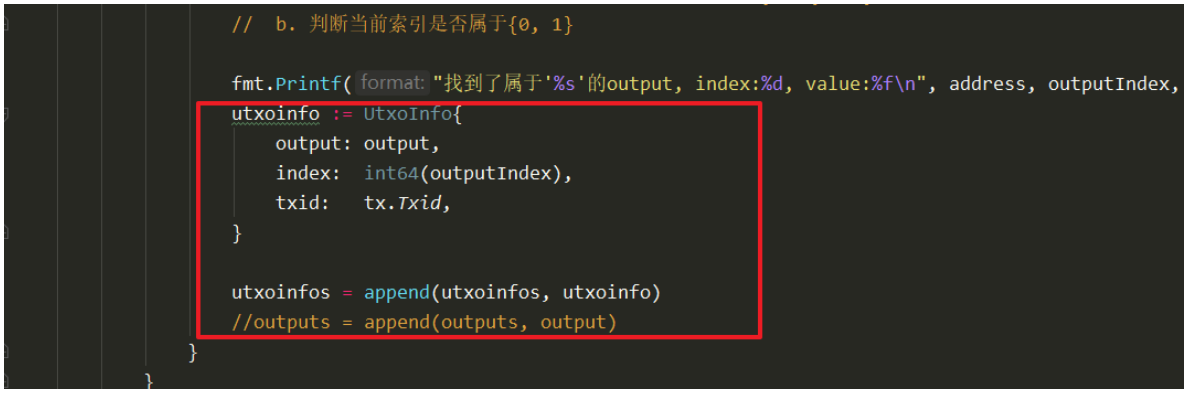
1. 使用UTXOInfo改写程序

```
//定义一个结构，同时包含output和它的位置信息
type UtxoInfo struct {
    output TXOutput
    index  int64
    txid   []byte
}
```





通过返回值，返回数据结构

FindNeedUtxoInfo

FindNeedUtxoInfo(付款人，支付的金额）[ ]UtxoInfo

1. 找到满足转账需求所需要的钱，找到后立刻返回
2. 如果没有找到，则返回空切片已经0，创建交易前，会校验返回的金额是否满足转账需求，如果不满足则创建交易失败

```go
func (bc *BlockChain) FindNeedUtxoInfo(address string, amount float64)
([]UtxoInfo, float64) {
    fmt.Printf("FindNeedUtxoInfo called, address :%s, amount:%f\n", address,
amount)

    //1. 遍历账本，找到所有address（付款人）的utxo集合
    utxoinfos := bc.FindMyUtxo(address)

    //返还的utxoinfo里面包含金额
    var retValue float64
    var retUtxoInfo []UtxoInfo

    //2. 筛选出满足条件的数量即可，不要全部返还
    for _, utxoinfo := range utxoinfos {
        retUtxoInfo = append(retUtxoInfo, utxoinfo)
        retValue += utxoinfo.output.Value

        if retValue >= amount {
            //满足转账需求，直接返回
            break
        }
    }

    return retUtxoInfo, retValue
}
```

2. 创建普通交易

分析：

1. 参数：
    1. 付款人
    2. 收款人
    3. 付款金额
    4. 矿工
    5. bc *Blockchain
2. 逻辑分析：
    1. 找到付款人能够支配的合理的钱，返回金额和utxoinfo
    2. 判断返回金额是否满足转账条件，如果不满足，创建交易失败。
    3. 拼接一个新的交易
        1. 拼装inputs
            1. 遍历返回的utxonifo切片，逐个转成input结构
        2. 拼装outputs
            1. 拼装一个属于收款人的output
            2. 判断一下是否需要找零，如果有，拼装一个属于付款方output
        3. 设置交易id
        4. 返回

4.

代码实现：

```
//普通交易
func NewTransaction(from, to string, amount float64, bc *BlockChain)
(*Transaction, error) {

    //1. 1. 找到付款人能够支配的合理的钱，返回金额和utxoinfo
    utxoinfos, value := bc.FindNeedUtxoInfo(from, amount)

    //2. 判断返回金额是否满足转账条件，如果不满足，创建交易失败。
    if value < amount {
        return nil, errors.New("付款人金额不足!")
    }

    //3. 拼接一个新的交易
    var inputs []TXInput
    var outputs []TXOutput

    //1. 拼装inputs
    for _, utxoinfo := range utxoinfos {
        input := TXInput{
            TXID:      utxoinfo.txid,
            Index:     utxoinfo.index,
            ScriptSig: from,
        }

        inputs = append(inputs, input)
    }
    //1. 遍历返回的utxonifo切片，逐个转成input结构
    //2. 拼装outputs
    //1. 拼装一个属于收款人的output
    output := TXOutput{
        LockScript: to,
        Value:      amount,
    }
    outputs = append(outputs, output)

    //2. 判断一下是否需要找零，如果有，拼装一个属于付款方output
    if value > amount {
        //找零
        output1 := TXOutput{
            LockScript: from,
            Value:      value - amount,
        }

        outputs = append(outputs, output1)
    }

    tx := Transaction{
        TxInputs:  inputs,
        TXOutputs: outputs,
```

```
        TimeStamp: time.Now().Unix(),
    }

    //3. 设置交易id
    tx.SetTxId()

    //4. 返回
    return &tx, nil
}
```

在比特币系统中，手续费没有单独的字段来定义。

每一笔交易的总inputs与总outputs的差值就是手续费。

3. send命令实现

4. 优化程序

- IsFileExist
- IsCoinbase
- HashTransaction

5. 钱包相关

椭圆曲线介绍

ecdsa介绍demo

6. 钱包关系

Wallet

- 创建秘钥对
- 根据私钥生成地址

WalletManager

- 定义结构
- 创建结构
- 添加创建地址命令
- 存储
- 加载
- 所有地址打印

- //排序, 升序
  sort.Strings(addresses)