

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
}
```

```
//遍历账本，查询指定地址所有的utxo
func (bc *Blockchain) FindMyUtxo(address string) []UtxoInfo {
    fmt.Println(a...: "FindMyUtxo called, address:", address)

    //var outputs []TXOutput
    var utxoinfos []UtxoInfo

    //定义一个map，用于存储已经消耗过的output
    //key ==> 交易id, value: 在这个交易中的索引的切片
    spentOutput := make(map[string][]int64)
```

```
// b. 判断当前索引是否属于{0, 1}

    fmt.Printf( format: "找到了属于'%s'的output, index:%d, value:%f\n", address, outputIndex,
    utxoinfo := UtxoInfo{
        output: output,
        index:  int64(outputIndex),
        txid:   tx.Txid,
    }

    utxoinfos = append(utxoinfos, utxoinfo)
    //outputs = append(outputs, output)
}
```

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

```

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)

    //返回的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

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

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)