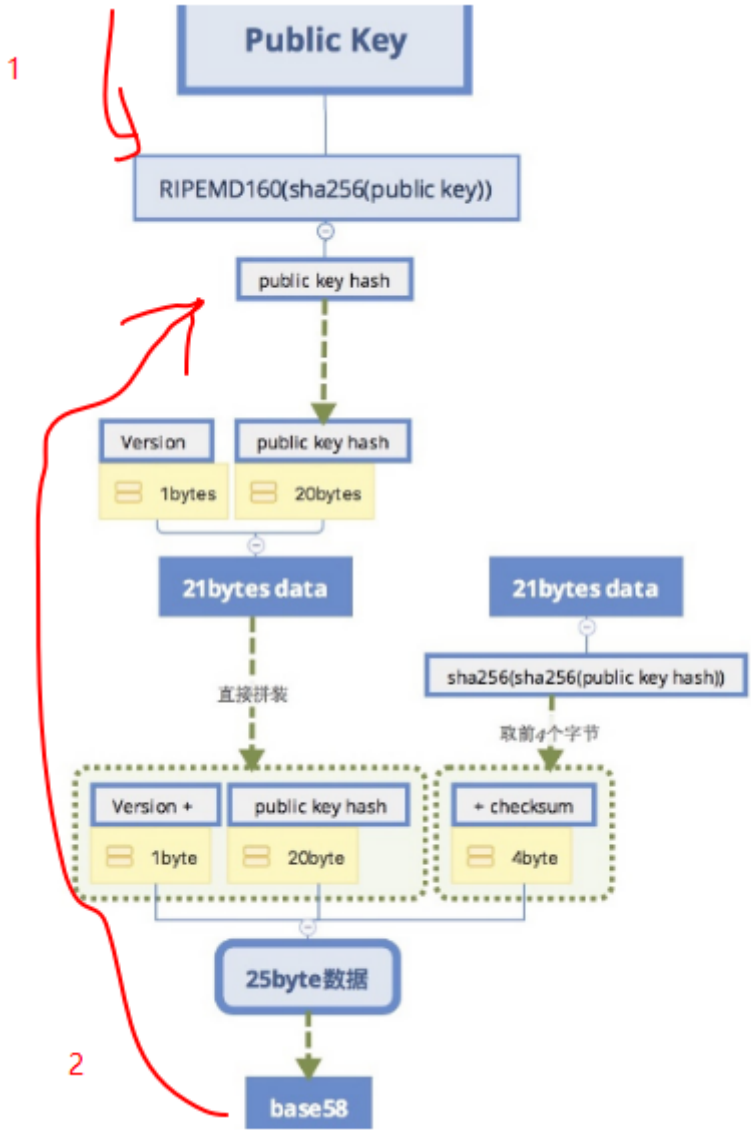
地址改写程序

1. 通过公钥， 获取公钥哈希
    1. getPubKeyHashFromPubKey(公钥)
2. 通过地址，获取公钥哈希
    1. getPubKeyHashFromAddress（地址）



实现三个函数：

```
//1. 通过公钥， 获取公钥哈希
func getPubKeyHashFromPubKey(pubKey []byte) []byte {
    //一、第一次哈希
    firstHash := sha256.Sum256(pubKey)
    //第二次哈希
    hasher := ripemd160.New()
```

```
    hasher.Write(firstHash[:])
    pubKeyHash := hasher.Sum(nil)

    return pubKeyHash
}

//2. 通过地址，获取公钥哈希
func getPubKeyHashFromAddress(address string) []byte {
    //1. base58解码，得到25字节数据
    decodeInfo := base58.Decode(address)

    if len(decodeInfo) != 25 {
        fmt.Println("地址长度无效，应该为25字节，当前字节为:", len(decodeInfo))
        return nil
    }

    //2. 截取出中间的20字节
    pubKeyHash := decodeInfo[1:21]

    //3. 返回
    return pubKeyHash
}

func checkSum(payload []byte) []byte {
    f1 := sha256.Sum256(payload)
    second := sha256.Sum256(f1[:])

    //checksum := second[:] //作闭右开
    checksum := second[:4] //作闭右开
    return checksum
}
```

改写地址函数

```
func (w *Wallet) getAddress() string {
    //通过公钥， 获取公钥哈希
    pubKeyHash := getPubKeyHashFromPubKey(w.PubKey)

    //二、在前面添加1个字节的版本号
    payload := append([]byte{byte(00)}, pubKeyHash...)

    //三、做两次哈希运算，截取前四个字节，作为checksum，
    checksum := checkSum(payload)

    //四、拼接25字节数据
    payload = append(payload, checksum...)

    //五、base58处理，得到地址
    address := base58.Encode(payload)
    return address
}
```

技巧：Ctrl+R，输入“cre”，此时会显出最近的含义cre开头的命令，

如果不是想要的，可以继续ctrl +R，此时会查找另外一个cre开头的命令。

```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/03-比特币/v5-a-wallet
(reverse-i-search)`':
```

终端常用操作：

1. ctrl + a => 移动到行首
2. ctrl + e => 移动到行尾

1. ctrl + f =》 向前移动一个字符
2. ctrl + b=》向后移动一个字符

1. ctrl + h=》向前删除一个字符
2. ctrl + d=》向后删除一个字符

1. ctrl + u =》 删除当前光标前的所有字符
2. ctrl + k=》 删除当前光标后的所有字符
3. ctrl + l =》 清屏

改写交易

```
//交易输入
type TXInput struct {
    //1. 所引用的output所在的交易id
    TXID []byte
    //2. 所引用的output的索引值
    Index int64
    //3. 解锁脚本：
    //ScriptSig string //先使用string代替，后续会改成签名

    //1. 私钥签名
    ScriptSig []byte   <<=====
    //2. 公钥
    PubKey []byte  <<=====
}

//交易输出
type TXOutput struct {
    //1. 锁定脚本
    //LockScript string
```

```
    //1收款人的公钥哈希
    PubKeyHash []byte  <<====
    //2. 转账金额
    Value float64
}
```

提供NewTXOutput方法

```
//收款人给付款人地址，锁定的时候不是使用地址锁定的，而是使用公钥哈希锁定的
//提供一个生成output的方法
func NewTXOutput(value float64, address string) TXOutput {
    //计算公钥哈希
    pubKeyHash := getPubKeyHashFromAddress(address)

    output := TXOutput{
        PubKeyHash: pubKeyHash,
        Value:      value,
    }

    return output
}
```

改写挖矿交易：

```
func NewCoinbaseTx(miner string, data string) *Transaction {
    intputs := []TXInput{{
        TXID:      nil,
        Index:     -1,
        ScriptSig: []byte(data),  <<===
        PubKey:    nil, <<===
    }}

    output := NewTXOutput(reward, miner) <<===
    outputs := []TXOutput{output} <<===

    //outputs := []TXOutput{{
    //  LockScript: miner,
    //  Value:      reward,
    //}}

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

    //设置交易id
    tx.SetTxId()

    return tx
}
```

改写NewTransaction

```go
func NewTransaction(from, to string, amount float64, bc *BlockChain) (*Transaction, error) {

    //1. 打开钱包
    wm := NewWalletManager()
    if wm == nil {
        return nil, errors.New( text: "打开钱包失败!")
    }

    //2. 找到付款方对应的私钥和公钥
    w, ok := wm.Wallets[from]
    if !ok {
        return nil, fmt.Errorf( format: "没有找到: '%s'对应的钱包!", from)
    }
    //创建input的时候需要私钥签名和公钥
    //priKey := w.PrivKey //TODO 汇通在使用
    pubKey := w.PubKey

    //付款人的公钥哈希
    pubKeyHash := getPubKeyHashFromPubKey(pubKey)

    //1. 1. 找到付款人能够支配的合理的钱，返回金额和utxoinfo
```

```go
//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(pubKeyHash, 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( text: "付款人金额不足!")
}
```

```go
    //1. 拼装inputs
    for _, utxoinfo := range utxoinfos {
        input := TXInput{
            TXID:      utxoinfo.txid,
            Index:     utxoinfo.index,
            ScriptSig: nil,    //钱包再交易创建的最后再处理 TODO
            PubKey:    pubKey, //付款人的公钥
        }

        inputs = append(inputs, input)
    }
```

```
    //1. 拼装一个属于收款人的output
    output := NewTXOutput(amount, to)

    //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,
        //}
        output1 := NewTXOutput(value-amount, from)
        outputs = append(outputs, output1)
    }
```

修改遍历函数

```
//参数1：付款人的公钥哈希值
func (bc *BlockChain) FindNeedUtxoInfo(pubKeyHash []byte, amount float64) ([]UtxoInfo, float64) {
    fmt.Printf( format: "FindNeedUtxoInfo called, pubKeyHash :%x, amount:%f\n", pubKeyHash, amount)

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

    //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 {
```

```
//遍历账本，查询指定地址所有的utxo
func (bc *BlockChain) FindMyUtxo(pubKeyHash []byte) []UtxoInfo {
    fmt.Printf( format: "FindMyUtxo called, address:%x\n", pubKeyHash)

    //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)
    //map[0x2222] = {0}
    //map[0x3333] = {0, 1}
```

```
block := it.Next()
//2. 遍历交易
for _, tx := range block.Transactions {
LABEL1:
    //3. 遍历output
    for outputIndex, output := range tx.TXOutputs {
        //判断当前的output是否是目标地址锁定的
        //if output.LockScript == address {

        if bytes.Equal(output.PubKeyHash, pubKeyHash) {
            //再添加之前进行过滤 ，依据：spentOutput集合
            //1. 先查看当前交易（0x3333）是否已经存在于spentOutput容器中
            currTxId := string(tx.Txid)
            //{0, 1}
            indexArr := spentOutput[currTxId]
```

```
//遍历inputs， 得到一个map
if !tx.isCoinbaseTx() {
    //如果不是挖矿交易，才有必要遍历innputs
    for _, input := range tx.TxInputs {

        pubKeyHash1 := getPubKeyHashFromPubKey(input.PubKey)
        if bytes.Equal(pubKeyHash1, pubKeyHash) {

            spentKey := string(input.TXID) //这个input的来源
            spentOutput[spentKey] = append(spentOutput[spentKey], input.Index)

            //下面是错误的，无法将数据写到spentOutput中
            //indexArray := spentOutput[spentKey]
            //indexArray = append(indexArray, input.Index)
```

修改创世块地址：临时的

> 1FakfxjbaLwEtNVUJnz9erXqjgBeRzvuz

```
//写入创世块
//创建一个挖矿交易，里面写入创世语
coinbaseTx := NewCoinbaseTx( miner: "1FakfxjbaLwEtNVUJnz9erXqjgBeRzvuz", genesisInfo)
genesisBlock := NewBlock([]*Transaction{coinbaseTx}, prevHash: nil)
```

```go
func (cli *CLI) getBalance(address string) {
    //utxos := cli.bc.FindMyUtxo(address)

    //通过地址获取公钥哈希
    pubKeyHash := getPubKeyHashFromAddress(address)

    utxoinfos := cli.bc.FindMyUtxo(pubKeyHash)
    var total float64

    for _, utxoinfo := range utxoinfos {
        total += utxoinfo.output.Value
    }
```

写脚本测试：check.sh

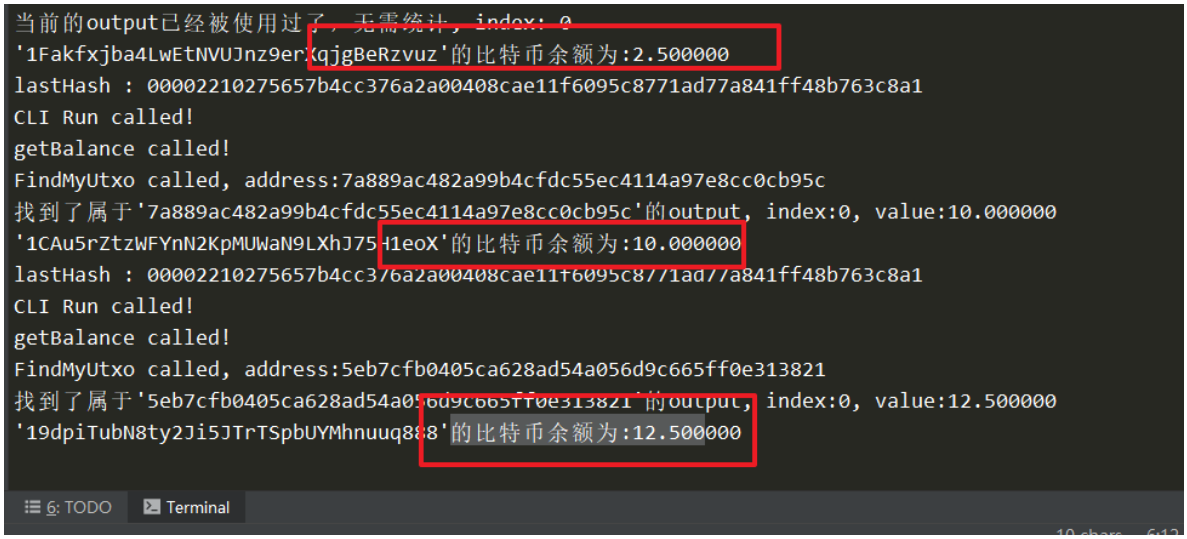
```
#!/bin/bash

./blockchain send 1Fakfxjba4LwEtNVUJnz9erXqjgBeRzvuz
1CAu5rZtzWFYnN2KpMUWaN9LXhJ75H1eoX 10 19dpiTubN8ty2Ji5JTrTSpbUYMhnuuq888 "hello
world"

./blockchain getBalance 1Fakfxjba4LwEtNVUJnz9erXqjgBeRzvuz #2.5
./blockchain getBalance 1CAu5rZtzWFYnN2KpMUWaN9LXhJ75H1eoX #10
./blockchain getBalance 19dpiTubN8ty2Ji5JTrTSpbUYMhnuuq888 #12.5
```

```
付款人：

address: 1Fakfxjba4LwEtNVUJnz9erXqjgBeRzvuz


收款人：

address: 1CAu5rZtzWFYnN2KpMUWaN9LXhJ75H1eoX

矿工：

address: 19dpiTubN8ty2Ji5JTrTSpbUYMhnuuq888
```

chmod.exe +x check.sh

./check.sh

效果：



校验地址有效性

签名相关

校验签名