从0到1简易区块链开发手册V0.1

Author: brucefeng

Email: brucefeng@brucefeng.com

微信ID: brucefeng1991

编程语言: Golang

前言

这是我这段时间学习区块链开发以来打造的第一个区块链平台，之所以叫做简易区块链，是因为它确实比较简易，仅仅是实现了底层的一些功能，不足以作为一个真正的公链使用，但通过学习，我们能够通过代码更加理解比特币白皮书中描述的各种比特币原理，区块链世界，无论是研究理论的，还是实战开发的，甚至炒币玩资本的，都离不开比特币的影子，区块链技术毕竟是从比特币中剥离抽象而来，所以，作为一个技术人员，无论是研究以太坊，超级账本，甚至是各种公链，包括某些山寨公链，都需要先去理解比特币原理，而对于开发者而言，理解原理最好的方式就是将其通过代码实现，当然，我们这里实现的一些原理只是应用实战范围之内可以实现的，比如椭圆加密算法，我们要实现的只是使用椭圆加密去实现某些加密功能，而非用代码去实现一个完整的椭圆加密代码库，这个不再本文的讨论范围内，所以本文面向的群体是:

- 对比特币原理不了解，但没时间看太多的资料文献的初学者
- 对比特币原理有所了解，但是停留在理论阶段的研究者
- 没有对比特币进行研究，想直接研究以太坊，超级账本的实战者(大神除外)
- 对Golang熟悉，但是不知道如何入手区块链的开发者或者是像我一样的运维 :-)

本文中，我们先通过命令行的方式演示区块链的工作流程以及相关原理，涉及到比较重要的内容，比如Sha256哈希，椭圆加密，Base58编码等内容，我会根据时间以及后期的工作情况进行适当调整，这注定是一个短期内没有结尾的故事。

> 为表尊敬，写在前面，建议先阅读该文档
>
> 本文的学习资料来自这位**liuxhengxu**前辈翻译的资料
>
> 能将资料翻译得如此完美，相比其技术功能也是相当深厚的，感谢分享

建议大家可以先看该资料后再来看我的这系列文章，否则可能会有一些难度，由于该资料是通过循序渐进的方式进行版本迭代，慢慢引导开发者不断对原有的代码进行优化，拓展，非常认真并细心，希望大家时间充裕的时候以及对某些本文并未写清楚的地方，强烈建议阅读该资料。

本文在此基础上进行了一些修改(谈不上改进)，我摒弃一些过于基础的以及后期需要大量重构的代码，直接通过该项目的执行流程进行代码分析，这样可以稍微节省一些大家的时间，把有限的精力放在对业务有更大提升的技术研究上。

一. 功能描述

```
Usage:
        createwallet
                        -- 创建钱包
        getaddresslists
                        -- 获取所有的钱包地址
        createblockchain -address address
                        -- 创建创世区块
        send -from SourceAddress -to DestAddress -amount Amount
                        -- 转账交易
        printchain
                        -- 打印区块
        getbalance -address address
                        -- 查询余额
```

本文围绕着几个功能进行讲解

- 创建钱包

  通过椭圆加密算法创建钱包地址

- 获取钱包地址

  获取区块链中所有的钱包地址

- 创建创世区块

  实现创世区块的创建，并生成区块链

- 实现转账交易

  通过转账交易，生成区块，并存入区块链

- 打印区块

  打印出所有的区块信息，实现转账交易的溯源

- 查询余额

  查询出对应钱包的余额状态

随着代码的不断完善，我们将会对以上进行改进，并提供更多的功能点进行分析探讨，我们先通过下图简单演示一下如上功能

关于效果图，大家先大致看下即可，不需要刻意研究，在后期的课程中都会涉及。

二. 实现命令行功能

1.定义结构体

定义一个空结构体

```
type CLI struct {

}
```

2.结构体方法

> 重要！初学者必看
>
> 这里提到的结构体方法并不是真正实现功能的方法，而是命令行对象的方法，这些方法中会调用实际的功能对象方法进行功能实现,在本章节中，创建结构体方法即可，功能代码可以为空，如:

```
例子：
func (cli *CLI) CreateWallet() {
}
func (cli *CLI) GetAddressLists() {
}
.....
```

其他的可以在后期逐步实现，为了让有基础的同学对项目整体提前有些印象，所以，代码内容我直接复制粘贴进来，不做删减，在后期的内容中，会逐步涉及到每个调用的对象方法或者函数的作用。

2.1 创建钱包

```
func (cli *CLI) CreateWallet() {
    _, wallets := GetWallets() //获取钱包集合对象
    wallets.CreateNewWallets() //创建钱包集合

}
```

2.2 获取钱包地址

```
func (cli *CLI) GetAddressLists() {
    fmt.Println("钱包地址列表为:")
        //获取钱包的集合，遍历，依次输出
    _, wallets := GetWallets() //获取钱包集合对象
    for address, _ := range wallets.WalletMap {
        fmt.Printf("\t%s\n", address)
    }
}
```

2.3 创建创世区块

```
func (cli *CLI) CreateBlockChain(address string) {
    CreateBlockChainWithGenesisBlock(address)
    bc :=GetBlockChainObject()
    if bc == nil{
        fmt.Println("没有数据库")
        os.Exit(1)
    }
    defer bc.DB.Close()
    utxoSet:=&UTXOSet{bc}
    utxoSet.ResetUTXOSet()
}
```

2.4 创建转账交易

```
func (cli *CLI) Send(from, to, amount []string) {
    bc := GetBlockChainObject()
    if bc == nil {
        fmt.Println("没有BlockChain，无法转账。。")
        os.Exit(1)
    }
    defer bc.DB.Close()

    bc.MineNewBlock(from, to, amount)
    //添加更新
    utsoSet :=&UTXOSet{bc}
    utsoSet.Update()
}
```

2.5 查询余额

```
func (cli *CLI) GetBalance(address string) {
    bc := GetBlockChainObject()
    if bc == nil {
        fmt.Println("没有BlockChain，无法查询。。")
        os.Exit(1)
    }
    defer bc.DB.Close()
    //total := bc.GetBalance(address,[]*Transaction{})
    utxoSet :=&UTXOSet{bc}
    total:=utxoSet.GetBalance(address)


    fmt.Printf("%s,余额是：%d\n", address, total)
}
```

2.6 打印区块

```
func (cli *CLI) PrintChains() {
    //cli.BlockChain.PrintChains()
    bc := GetBlockChainObject() //bc{Tip,DB}
    if bc == nil {
        fmt.Println("没有BlockChain，无法打印任何数据。。")
        os.Exit(1)
    }
    defer bc.DB.Close()
    bc.PrintChains()
}
```

3. 相关函数

3.1 判断参数是否合法

```
func isValidArgs() {
    if len(os.Args) < 2 {
        printUsage()
        os.Exit(1)
    }
}
```

判断终端命令是否有参数输入，如果没有参数，则提示程序使用说明，并退出程序

3.2 程序使用说明

```
func printUsage() {
    fmt.Println("Usage:")
    fmt.Println("\tcreatewallet\n\t\t\t-- 创建钱包")
    fmt.Println("\tgetaddresslists\n\t\t\t-- 获取所有的钱包地址")
    fmt.Println("\tcreateblockchain -address address\n\t\t\t-- 创建创世区块")
    fmt.Println("\tsend -from SourceAddress -to DestAddress -amount
Amount\n\t\t\t-- 转账交易")
    fmt.Println("\tprintchain\n\t\t\t-- 打印区块")
    fmt.Println("\tgetbalance -address address\n\t\t\t-- 查询余额")
}
```

3.3 JSON解析的函数

```
func JSONToArray(jsonString string) []string {
    var arr [] string
    err := json.Unmarshal([]byte(jsonString), &arr)
    if err != nil {
        log.Panic(err)
    }
    return arr
}
```

通过该函数将JSON字符串格式转成字符串数组，用于在多笔转账交易中实现同时多个账户进行两两转账的功能。

3.4 校验地址是否有效

```
func  IsValidAddress(address []byte) bool {

    //step1：Base58解码
    //version+pubkeyHash+checksum
    full_payload := Base58Decode(address)

    //step2：获取地址中携带的checkSUm
    checkSumBytes := full_payload[len(full_payload)-addressCheckSumLen:]
```

```
    versioned_payload := full_payload[:len(full_payload)-
addressCheckSumLen]

    //step3：versioned_payload，生成一次校验码
    checkSumBytes2 := CheckSum(versioned_payload)

    //step4：比较checkSumBytes，checkSumBytes2
    return bytes.Compare(checkSumBytes, checkSumBytes2) == 0

}
```

以下三个功能实现之前需要先调用该函数进行地址校验

- 创建创世区块
- 转账交易
- 查询余额

4.命令行主要方法Run

```
Usage:
        createwallet
                        -- 创建钱包
        getaddresslists
                        -- 获取所有的钱包地址
        createblockchain -address address
                        -- 创建创世区块
        send -from SourceAddress -to DestAddress -amount Amount
                        -- 转账交易
        printchain
                        -- 打印区块
        getbalance -address address
                        -- 查询余额
```

我们将如上功能展示的实现功能写在Run方法中，实现命令行功能的关键是了解**os.Args**与**flag**

> 关于这两个功能，此处不再赘述，否则篇幅会无限臃肿。

代码块均在方法体Run中，下文将分步骤对代码实现进行体现

```
func (cli *CLI) Run() {
}
```

4.1 判断命令行参数是否合法

```
isValidArgs()
```

4.2 创建flagset命令对象

```
    createWalletCmd := flag.NewFlagSet("createwallet", flag.ExitOnError)
    getAddresslistsCmd := flag.NewFlagSet("getaddresslists",
flag.ExitOnError)
    CreateBlockChainCmd := flag.NewFlagSet("createblockchain",
flag.ExitOnError)
    sendCmd := flag.NewFlagSet("send", flag.ExitOnError)
    printChainCmd := flag.NewFlagSet("printchain", flag.ExitOnError)
    getBalanceCmd := flag.NewFlagSet("getbalance", flag.ExitOnError)
    testMethodCmd := flag.NewFlagSet("test", flag.ExitOnError)
```

如上，通过flag.NewFlagSet方法创建命令对象，如createwallet,getaddresslists,createblockchain等命令对象

固定用法，掌握即可。

4.3 设置命令后的参数对象

```
    flagCreateBlockChainData := CreateBlockChainCmd.String("address",
"GenesisBlock", "创世区块的信息")
    flagSendFromData := sendCmd.String("from", "", "转账源地址")
    flagSendToData := sendCmd.String("to", "", "转账目标地址")
    flagSendAmountData := sendCmd.String("amount", "", "转账金额")
    flagGetBalanceData := getBalanceCmd.String("address", "", "要查询余额的账
户")
```

通过命令对象的String方法为命令后的参数对象

- createblockchain命令后的参数对象：address
- send命令后的参数对象: from | to |amount
- getbalance命令后的参数对象: address

其中createwallet，getaddresslists，printchain命令没有参数对象。

4.4 解析命令对象

```
switch os.Args[1] {
    case "createwallet":
        err := createWalletCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    case "getaddresslists":
        err := getAddresslistsCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    case "createblockchain":
        err := CreateBlockChainCmd.Parse(os.Args[2:])
        if err != nil {
```

```
            log.Panic(err)
        }
    case "send":
        err := sendCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    case "getbalance":
        err := getBalanceCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    case "printchain":
        err := printChainCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    case "test":
        err := testMethodCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    default:
        printUsage()
        os.Exit(1)

    }
```

匹配对应的命令，用命令对象的Parse方法对**os.Args[2:]**进行解析。

4.5 执行对应功能

```
    //4.1 创建钱包--->交易地址
    if createWalletCmd.Parsed() {
        cli.CreateWallet()
    }
    //4.2 获取钱包地址
    if getAddresslistsCmd.Parsed() {
        cli.GetAddressLists()
    }
    //4.3 创建创世区块
    if CreateBlockChainCmd.Parsed() {
        if !IsValidAddress([]byte(*flagCreateBlockChainData)) {
            fmt.Println("地址无效，无法创建创世前区块")
            printUsage()
            os.Exit(1)
        }
        cli.CreateBlockChain(*flagCreateBlockChainData)
    }
```

```
    //4.4 转账交易
    if sendCmd.Parsed() {
        if *flagSendFromData == "" || *flagSendToData == "" ||
*flagSendAmountData == "" {
            fmt.Println("转账信息有误")
            printUsage()
            os.Exit(1)
        }
        //添加区块
        from := JSONToArray(*flagSendFromData)     //[]string
        to := JSONToArray(*flagSendToData)         //[]string
        amount := JSONToArray(*flagSendAmountData) //[]string
        for i := 0; i < len(from); i++ {
            if !IsValidAddress([]byte(from[i])) ||
!IsValidAddress([]byte(to[i])) {
                fmt.Println("地址无效，无法转账")
                printUsage()
                os.Exit(1)
            }
        }

        cli.Send(from, to, amount)
    }
    //4.5 查询余额
    if getBalanceCmd.Parsed() {
        if !IsValidAddress([]byte(*flagGetBalanceData)) {
            fmt.Println("查询地址有误")
            printUsage()
            os.Exit(1)
        }
        cli.GetBalance(*flagGetBalanceData)
    }

    //4.6 打印区块信息
    if printChainCmd.Parsed() {
        cli.PrintChains()
    }
```

5. 测试代码

> 在main.go中中添加测试代码

```
package main
func main() {
    cli:=BLC.CLI{}
    cli.Run()
}
```

编译运行

```
$ go build -o mybtc main.go
```

> 测试思路
>
> 1. 查看命令行列表是否可以正常显示
> 2. 输入非法字符查看是否有错误提示
>
> 业务功能此处暂未实现，测试时忽略。

下一篇文章将介绍如何实现钱包/地址的生成功能，即**创建钱包**。