区块链共识协议理论与实战

基于 golang 开发

作者：艾振东
组织：组织机构
时间：April 25, 2021
版本：1.0

ElegantLaTeX Program

温柔正确的人总是难以生存，因为这世界既不温柔，也不正确。——比企谷八幡

前言

这里是前言，介绍本书的动机、内容范围、受众、其他提示等。

艾振东

二零二一年四月二十五日

目录

版本更新历史

第一章 区块链简介

内容提要

Listing 1.1: hello.go

```go
// package main main表示包名
package main

// 导入"fmt"这个包
import "fmt"

// 程序的入口 main函数
func main() {
  // 调用"fmt"包下的Println()函数向标准输出打印"hello world!"字符串
  fmt.Println("hello, world!")
}
```

1.1 本章导读

1.2 区块链定义

1.3 区块链分类

1.3.1 公有链

1.3.2 联盟链

1.4 区块链发展

1.4.1 区块链 1.0——可编程数字货币

1.4.2 区块链 2.0——可编程金融

1.4.3 区块链 3.0——可编程社会

1.4.4 区块链技术瓶颈与解决方案

1.5 区块链技术架构

1.5.1 区块链四层技术架构

1.5.2 区块链中的加密学

1.5.3 数据层——区块链中的数据结构

1.5.4 网络层——区块链中的 P2P

1.5.5 共识层——区块链中的共识协议

1.6 本章小结

第二章 共识协议介绍

2.1 本章导读

这里是导读

2.2 共识协议基础

2.2.1 CAP 定理

2.2.2 拜占庭将军问题

2.2.3 区块链不可能三角

2.2.4 共识协议内涵

2.2.5 共识协议节点分类

2.2.6 共识协议分类

2.2.7 共识协议一般流程

2.3 典型共识协议分析

2.3.1 工作量证明 PoW

2.3.2 权益证明 PoS

2.3.3 股份授权证明 DPoS

2.3.4 权威证明 PoA

2.3.5 经过时间证明 PoET

2.3.6 实用拜占庭容错协议 PBFT

2.3.7 Raft 协议

2.4 共识协议设计关键与思路

2.5 本章小结

第三章 Go语言基础

3.1 本章导读

3.2 Go语言基础介绍

3.2.1 为什么使用Go语言？

3.2.2 Go语言的数据类型

3.2.3 Go语言的控制语句

3.2.4 Go语言的编译与运行

3.3 Go语言的网络编程

3.3.1 网络编程基础

3.3.2 net库

3.3.3 TCP服务端与客户端编写

3.4 Go语言的并发编程

3.4.1 goroutine与go关键字

3.4.2 Mutex/RWMutex——goroutine间的资源互斥

3.4.3 channel——基于消息传递而非资源共享的goroutine间通信

3.5 实现一个TCP并发P2P节点

3.6 实现命令行工具

3.7 本章小结

第 四 章　一个简单的区块链

第五章 bcc 库框架

5.1 本章导读

前面我们介绍了区块链与共识协议的基础，也简单介绍了阅读本文所需的 Go 语言基础，并且带领读者编写了一个简单的区块链程序，相信读者已经对区块链底层开发有一定的认识和了解。我们本书的目的是实现一个自己的共识协议，这要求我们编写一个通用的、模块可替换的区块链程序，从而对提出的共识协议进行测试、实验。为了方便起见，我们称这样的程序为 bcc 库（blockchain-consensus library），基于 bcc 库，可以快速建立自己的。本章我们将介绍 bcc 库的设计与实现。

5.2 bcc 库的模块架构

通过前面实现的简单区块链程序，可以发现，一个完整的区块链节点程序需要具备以下一些模块/内容：

1. 基础的数据结构。指帐号（或钱包地址）、节点信息、交易、区块、消息等数据结构。
2. P2P 通信模块（网络模块）。节点程序需要具备与其他节点进行双向通信的能力。对于基于连接的通信协议（如 TCP 协议）而言，P2P 通信模块还需要负责保持与其他节点的连接的维护工作。
3. 账本模块。帐本模块负责存储节点程序收集的区块、交易数据，同时负责对区块进行检验、排序。账本的概念源于比特币等基于区块链的数字货币，某种程度上我们可以将之视为区块与交易的存储模块。
4. 共识协议。节点间的共识正是通过彼此的消息往来实现，同时它的运行也依赖于本地帐本模块的状态。也就是说，共识协议依托于网络模块与帐本模块。在前面实现的简单区块链中，共识协议并没有明显的划分为模块，这样的做法不利于共识协议的替换。

现在，我们希望编写一个通用的、模块可替换的 bcc 库，我们希望将这些模块/内容尽可能地进行职能上的分离：基础数据结构具备足够的通用性、账本模块仅提供区块与交易相关的查询更新操作、网络模块仅提供通信能力、共识协议部分封装为使用其他模块的共识模块以达到对于上层应用而言的可替换性。此外，对于节点账户相关的信息，我们新增一个节点信息表模块用于对其进行管理。如此一来，无论是网络模块还是共识模块，均无需直接维护邻居节点信息，进一步降低了单个模块的代码复杂度。我们给出如下的模块结构图：

出于模块化、通用、便捷开发的设计考虑，本文所提出的 bcc 库组件结构如图5.1所示。图中箭头表示模块的调用关系。图中“bcc 库”部分为 bcc 库实际所包含的组件结构，而虚线框部分则是 bcc 库使用者需要提供的模块。图中的所有模块均可以自定义实现以

替换，各个模块的简单介绍如下：

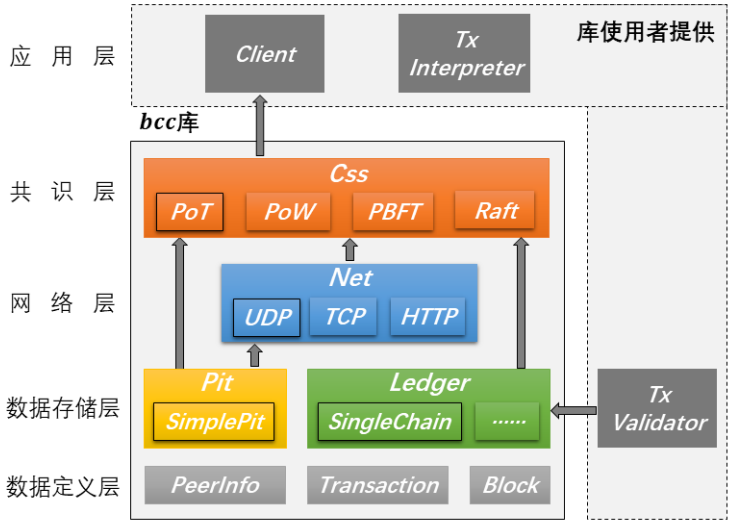

图 5.1: PoT 协议模块结构

1. *Css*。*Css* 模块即共识模块，是整个 bcc 库的核心。共识模块调用存储（*Pit*、*Ledger*）与网络（*Net*）模块完成消息处理、状态转换、区块共识等任务。*Css* 的具体实现可以有 PoT、PoW、PBFT、Raft 等，目前默认提供了 PoT 实现。
2. *Net*。*Net* 模块即网络模块，负责与其他节点的网络通信，包括消息格式的封装与解封、基础的验证工作等。*Net* 模块可以基于 UDP、TCP、HTTP 等多种协议实现。
3. *Pit*。*Pit* 即节点信息表（PeerInfoTable），负责存储集群所有节点必要的信息（如节点 ID、节点网络地址等），并向 *Css* 和 *Net* 模块提供增删改查能力。*Pit* 的默认实现是 *SimplePit*，对于需要记录节点信用历史、余额等的区块链项目可以自定义更丰富的 *Pit* 模块。
4. *Ledger*。*Ledger* 即账本存储模块，负责区块与交易的增删改查操作，与 *Tx Validator* 交互以验证新区块和新交易。*Ledger* 的默认实现是 *SimpleChain*（一个简单的区块链单链结构）。
5. *Client & Tx Interpreter & Tx Validator*。*Client*（客户端模块）负责用户交互（构建交易、访问区块链信息等）。*Tx Interpreter*（交易解释器）负责解释一个交易的内容。*Tx Validator*（交易验证器）负责提供对交易的内容验证能力。这三项是基于 bcc 库构建区块链程序必须进行自定义的模块，并且交易验证器必须按照 bcc 库给出的接口定义进行实现，其他两个模块则是逻辑意义上的，由用户自行决定。

5.3 通用账户结构设计

5.4 基础数据结构——节点信息、交易、区块与消息

在 bcc 库中，基础的数据结构指账户唯一标识（ID）、节点信息（PeerInfo）、交易（Transaction）、区块（Block）、消息（Message）。下面对其一一介绍。

Listing 5.1: xxx

```
put your code here!
```

Listing 5.2: xxx

```
put your code here!
```

5.4.1 账户 ID

我们已经知道，区块链中使用数字签名算法生成密钥对作为帐号，但是为了方便记忆与使用，通常会对密钥对中的公钥进行编码，得到一串较短的、可反编码为公钥的字符串，并以此为账户 ID。在 bcc 库中，我们不太关心一个 ID 是如何生成的（也即采用了何种数字签名算法、经过了怎样的编码过程）。我们仅使用字符串作为 ID 的类型。但要注意的是，我们需要为其添加一些约定，例如：ID 具备签名与验证签名的能力

5.4.2 节点信息

节点信息

xxx

5.4.3 交易

5.4.4 区块

5.4.5 消息

5.5 网络模块的接口定义与实现

5.5.1 BNet 接口

5.5.2 TCPNet 实现

Listing 5.3: xxx

```
put your code here!
```

Listing 5.4: xxx

```
put your code here!
```

Listing 5.5: xxx

```
put your code here!
```

5.5.3 UDPNet 实现

5.6 节点表模块的接口定义与实现

5.6.1 Pit 接口

5.6.2 SimplePit 实现

5.7 账本模块的接口定义与实现

5.7.1 Ledger 接口

代码5.11

5.7.2 SimpleChain 实现

5.8 共识模块的接口定义

5.8.1 Consensus 接口

5.9 本章小结

本章

Listing 5.6: xxx

```
put your code here!
```

Listing 5.7: xxx

```
put your code here!
```

Listing 5.8: xxx

```
put your code here!
```

Listing 5.9: xxx

```
put your code here!
```

Listing 5.10: xxx

```
put your code here!
```

Listing 5.11: modules/ledger/ledger.go

```
type LedgerType uint8

const (
  LedgerType_SimpleChain LedgerType = iota
)

// New
func New(ledgertype LedgerType, id string) (requires.BlockChain, error) {
  switch ledgertype {
  case LedgerType_SimpleChain:
    return simplechain.New(id)
  default:
    return nil, errors.New("unsupport ledger type")
  }
}

// Ledger 账本。比如区块链账本
type Ledger interface {
  requires.BlockChain
}
```

Listing 5.12: modules/ledger/simplechain/simplechain.go

```
xxx
```

Listing 5.13: modules/consensus/css.go

```
type ConsensusType uint8

const (
  ConsensuType_PoT ConsensusType = iota
  ConsensuType_PoW
  ConsensuType_PBFT
  ConsensuType_Raft
)

// Consensus 共识接口。事实上一个Consensus实例代表一个基于该共识协议的共识节点
type Consensus interface {
  Init() error  // Init 初始化(运行)。 New之后需要Init
  Inited() bool // Inited 判断是否初始化好
  Ok() bool     // Ok 检查Net所依赖的对象是否初始化好
  Close() error // Close 关闭状态机服务，执行一些必要的清理工作
  Closed() bool

  /*不可调用，只是为了约束Consensus该有的实现*/

  // HandleMsg [不可调用] 共识节点必需有处理各类消息的能力
  HandleMsg(msg *defines.Message) error
  // StateMachineLoop [不可调用] 状态机循环，负责状态的切换 go
  StateMachineLoop()
  // MsgHandleLoop [不可调用] 消息处理循环 go
  MsgHandleLoop()
  // SetMsgInChan [不建议调用] 设置接收消息的channel，需要将该chan移交给网络模块去写消息
  // Consensus模块在内部循环读该chan，处理Message
  SetMsgInChan(bus chan *defines.Message)
}

// New 新建一个共识状态机
func New(typ ConsensusType,
  id string, duty defines.PeerDuty,
  pit pitable.Pit, bc requires.BlockChain,
  net bnet.BNet, msgchan chan *defines.Message,
  monitorId, monitorHost string,
  genesisData ...string) (Consensus, error) {

  switch typ {
  case ConsensuType_PoT:
    return pot.New(id, duty, pit, bc, net, msgchan, monitorId, monitorHost,
        genesisData...)
  default:
    return nil, errors.New("unknown consensus type")
  }
}
```

第六章 PoT 协议与实现

6.1 本章导读

6.2 PoT 协议介绍

6.3 PoT 协议的情景分析与特性分析

6.4 PoT 协议在 bcc 库中的实现

6.5 本章小结

第七章 bcc 库进一步完善以及协议测试

7.1 本章导读

7.2 准备交易生成器

7.3 PoT 运行测试

7.4 增加性能监视器

7.5 自动测试协议性能

7.6 本章小结

第 八 章 bcc 库的简单应用——一个分布式 KV 存储

// 先不写

8.1 本章导读

用来测试 bcc 库使用

8.2 增删改查相关交易设计

8.3 自定义账本模块

8.4 增加一个命令行交互工具

8.5 本机部署及运行

8.6 本章小结

第 九 章　基于 PoT 的安全数据共享

参考文献