

第35讲 | 搭建你的迷你区块链（设计篇）

2018-06-13 陈浩

深入浅出区块链

[进入课程 >](#)



讲述：黄洲君

时长 07:07 大小 2.46M



程序员的天赋技能就是通过代码实践自己的想法，完成一个作品会有相当的成就感。

今天我们终于也来到了实践环节。我将以 C++14 的代码为例，和你分享设计并实现一个迷你区块链的例子。

目标和范围

首先我们要知道达成的目标，根据目标划定工作范围。

考虑到我们无法搭建一个类似比特币的庞大 P2P 网络，也没有太多精力实现一个真正意义上的完整功能的全节点钱包，而且完整的全节点过于复杂，会让学习者迷失在细节中。

所以我们的目标是：构建一个包含仅有基础功能的全节点客户端，它可能没有太炫酷的 UI 页面，也没有复杂的命令，它们可以提供下面的功能。

1. 提供 P2P 节点发现和同步区块的功能；
2. 提供创建公私钥对的功能；
3. 提供发送交易的功能；
4. 提供交易查询的功能；
5. 提供余额查询的功能；
6. 提供挖矿的功能，在任意地址上都可以发起单机挖矿；
7. 提供基础日志，方便跟踪监视。

以上 7 个功能基本涵盖了一个区块链全节点的主要功能，但是，由于我的时间有限，代码不能全部实现，主要是讲解设计和实现思路。后续我会逐渐完善代码，你也可以一起参与。

代码开源在：<https://github.com/betachen/tinychain>

技术选型

我们在深入区块链技术专题中说到过，区块链的四个核心技术概念：P2P 网络、账户模型与存储、共识、加密模块。

首先，P2P 网络模块是区块链的最底层模块之一，我们主要考虑方便实现和测试，可选的方案有轻量级消息队列和 WebSocket。考虑到集成的便利性，我们首选 WebSocket，因为至少需要一个 HTTP JSON-RPC Server，我们可以复用 Server 中的 Websocket 服务。

除了通讯协议之外，还要考虑数据交换格式，我们考虑采用易读通用的 JSON 格式，而不是像比特币一样的数据序列化格式，后期更改可以考虑升级到 Protobuf，后者优势主要体现在性能上。而在我们的例子中，性能永远不是首先考虑的，更多是它的易读和易调试性。

其次，我们来说说账户加密部分，由于 ECDSA 非对称加密模块过于复杂，我们选用 OpenSSL 库中的 RSA 算法作为加密模块。而交易模型上，我们考虑使用 UTXO 模型，因为状态模型需要维护状态，可能会带来额外的代码复杂度。

再来说说数据库存储，这个模块需要考虑到易用性和易读性，我们选用 SQLite 3 作为持久化存储。

最后我来谈谈共识算法这一模块，我们选用 PoW 作为共识算法，这是考虑到 PoW 实现起来十分简单，而且交易和区块的哈希计算会涉及 SHA-256，使用 PoW 算法我们就可以复用 SHA-256 的代码，使用 SHA-256 算法作为挖矿算法会降低我们的工作量。

详细功能

有了技术选型之后，我们再对目标功能点进行细分拆解。

1. P2P 网络：节点发现、节点维护、持久化保存、区块同步。
2. 公私钥对：命令行，创建公私钥对并生成地址，提供私钥存储，公私钥验证。
3. 发送交易：命令行，发送成功验证，输入是交易哈希。
4. 交易查询：命令行，JSON 格式的交易查询返回，输入是某个地址。
5. 余额查询：命令行，JSON 格式的余额查询返回，输入是某个地址。
6. 挖矿：命令行、JSON 格式挖矿信息返回，输入是某个地址。
7. 区块共识：编织区块链的算法，包含创世区块以及调整全网挖矿难度。
8. 交易共识：验证单个交易的算法，包含签名验证和 UTXO 验证。
9. 基础日志：用于监控网络，区块验证等操作。
10. 区块持久化存储：□分叉与合并时的一致性，并为查询提供接口。
11. 提供格式化输出交易的功能，这里的格式化主要指 JSON 格式。
12. 有效防止双花交易。

通过详细的功能拆分我们可以发现，功能点多达三十余个，如何设计实现这三十多个功能点是我们接下来首先要解决的问题。问题是这三十多个功能点不是孤立的，而是有相互联系的，我们先从顶层开始设计。

最顶层是一个区块节点，一个完整的可执行程序，我们命名为 Tinychain，而对应的命令行客户端为 cli-tinychain。


Tinychain 的核心程序主要包含以下结构：

 复制代码

```
1 tinychain
2 |— blockchain
3 |— consensus
4 |— database
5 |— network
6 |— http-server
7 |— □node
```

我们以 node 为最顶层，那么 node 会包含其他五个模块，node 启动就会把其他 5 个服务启动。


cli-tinychain 主要包含以下结构：

 复制代码

```
1 cli-tinychain
2 |— □JSON
3 |— □http-client
```

命令行就简单多了，我们把命令行的执行和计算全部都扔到 tinychian 当中，命令行只用一个 http-client 用 JSON 把 API 包起来即可。

通过分析我们知道，以下组件是必不可少的，但是我们不必自己开发，可以直接选取一些现成的开发包直接集成即可。

 复制代码

```
1 基础组件
2 |— log
3 |— JSON-paser
4 |— sha256
5 |— □key-pair
```

区块数据结构设计


有了大致的顶层设计已经分类好，那么接下来我们考虑为每个模块填充一些数据结构。一个区块链最重要的是区块，所以我们从区块开始。

一个区块包含两部分，分别是区块头和区块体，区块头是一个区块的元数据，区块体就是包含交易的列表，所以我们直接设计交易体。

区块头的设计

我们参照比特币的设计，区块头包含了前向区块哈希、默克尔根哈希、时间戳、难度目标、Nonce 值和版本号。

所以我们的结构可能是这样的。

 复制代码

```
1  {
2    "target_bits" : "4575460831240",
3    "hash" :
4    "4a9169e2f4f8673ac9627be0fa0f9e15a9e3b1bc5cd697d96954d25acacd92df",
5    "merkle_tree_hash" : "3d228afc50bc52491f5dd8aa8c416da0d9a16bf829790ea0b7635e5b4d44ab",
6    "nonce" : "3852714822920177480",
7    "height" : 1234567,
8    "previous_block_hash" : "4d2544e044bfd2f342220a711b10842bb6cfae551b1bc1ed6152ff5c7f3",
9    "time_stamp" : 1528070857,
10   "transaction_count" : 1,
11   "version" : 1
12 }
```

target_bits 表示当前区块的目标值；

hash 表示这个区块的哈希；

merkle_tree_hash 表示这个区块当中交易列表的默克尔根；

nonce 表示随机数；

height 表示当前区块的高度；

previous_block_hash 指向前向区块哈希；

time_stamp 表示生产这个区块时的时间戳；

transaction_count 表示这个区块当中包含多少笔交易；

version 表示区块的版本号，不代表交易的版本号。

在这里，我们的区块头大小不是固定的，因为它没有经过序列化，完全以 JSON 表示，所以我们这里就不考虑字节印第安序的问题了，也不考虑固定长度的问题。

有了区块头，我们再看看交易体的设计，由于使用 UTXO 作为交易模型，那么我们先考虑一个输入、一个输出的结构。

```
1 {
2   "hash": "8c14f0db3df150123e6f3dbbf30f8b955a8249b62ac1d1ff16284aefa3d06d87",
3   "version": 1,
4   "input_size": 1,
5   "output_size": 1,
6   "size": 135,
7   "inputs": [{
8     "prev_out": {
9       "hash": "0000000000000000000000000000000000000000000000000000000000000000",
10      "index": 0
11    },
12  }],
13  "out": [{
14    "value": "5000000000",
15    "address": "f3e6066078e815bb2"
16  }],
17 }
```

我们可以按照这种结构来设计交易体。

地址设计

区块链地址都有通常意义上的地址，我们这里将公钥直接算作地址，不再将公钥进行哈希转换。

内存池

内存池是指缓存交易的一块交易缓冲区，这里一个节点的主要处理对象，所以对内存池的管理，是编织区块链的最重要一步。我们这里的内存池使用标准库 STL 中的容器。

哈希计算

区块和交易的哈希计算均使用 SHA-256。

开发环境搭建

由于选取了 C++ 作为实现方式，搭建工程的过程会比较复杂一点。我们用的是 Ubuntu 16.04 开发环境，默认的 gcc 编译器是 gcc-5.4，是支持 C++14 标准的。代码也是全平台可移植的，如果你使用 Mac，也可以尝试搭建。

除了 gcc 之外，我们还需要 Cmake 来构建工程。我们也许需要 Boost 库的支持，例如 Filesystem 和 Datetime 等基础组件。

所以我们的工具链是：

gcc 或 clang

cmake

boost 1.56+ (datetime)

最后我们还需要一个简单好用的轻量级 Httpserver，我选取了元界代码中的 Mongoose 库，这里的 Mongoose 不是 MongoDB，是由 Cesanta 开源的一个 HTTP Server 库，支持 epoll 和 select 两种网络并发机制，也支持 WebSocket。

当然除了 C++ 实现之外，我们也可以使用 Python 来实现，实际上也有不少 Python 实现的 Demo，但我发现用 Python 实现的例子很多是在单进程中模拟区块链的数据结构，并不是真正意义上的分布式节点，所以我采取了使用 C++ 实现的策略。

测试环境搭建

我们知道区块链是一个分布式网络环境，在开始之前，我们需要构造一个简单且容易测试的分布式网络环境。

我们不可能购买大量的云计算资源，所以我们推荐你购买一个基础版的 ECS 节点，2Core 4G 就可以，性能稍好□更好，接着我们选用 Docker 来搭建容器集群，在容器中部署节点，其中宿主机作为编译环境，将编译完成的钱包部署到全部的 Docker 容器中。

总结

今天我大致介绍了实践一个迷你区块链的思路，我们先划定了实践的范围，接着考虑了技术选型，然后细化了详细功能，考虑了一个区块链需要的数据结构，最后考虑了开发环境和测试环境的搭建。今天的问题是，你觉得搭建一个迷你区块链最难的部分是哪一部分呢？

链接：

<https://github.com/cesanta/mongoose>

一些 Python 实现迷你区块链的例子：

1. <https://medium.com/crypto-currently/lets-build-the-tiniest-blockchain-e70965a248b>
2. <https://hackernoon.com/learn-blockchains-by-building-one-117428612f46>
3. <http://adilmoujahid.com/posts/2018/03/intro-blockchain-bitcoin-python/>



深入浅出区块链

你的区块链入门第一课

陈浩 元界 CTO



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第34讲 | 从业区块链需要了解什么？

下一篇 第36讲 | 搭建你的迷你区块链（实践篇）

精选留言 (8)

 写留言



leoxie

2018-07-19

读的人几个技术的英文单词估计搞不懂

展开 ∨

 3



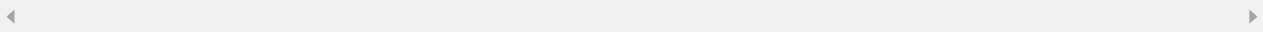
钰胤

2018-06-13

👍 2

陈老师，如果想在自己的笔记本上搭建一个区块链模拟网络，需要什么配置？😊

作者回复: 正常配置即可，有docker就行



沃野阡陌

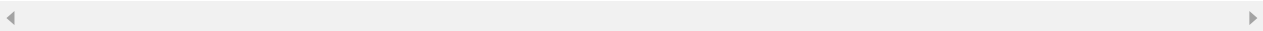
2018-06-16

👍 1

老师，请问什么是共享CDN挖矿？

展开 ▾

作者回复: 你好，这个是营销概念，没有共通性。



Ross 白

2018-11-06

👍

我还在学习 hyperledge golang，好像在常规商业上有很好的用途，请问这个课程是否可以介绍下。 谢谢



谢晋

2018-07-10

👍

谢谢老师的专栏

展开 ▾



krugle

2018-06-20

👍

怎么理解节点，钱包就是节点吗，还是矿工是节点，以太坊的客户端包含节点功能吗



行者

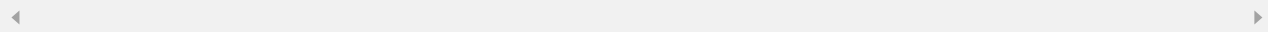
2018-06-14

👍

老师，讲下默克尔根哈希吧，对这个挺困惑的

展开 ▾

作者回复: 好的, 我会在个人私有专栏里写



Ender0224

2018-06-14



终于等到了

展开 ∨

作者回复: 感谢支持, 嘻嘻

