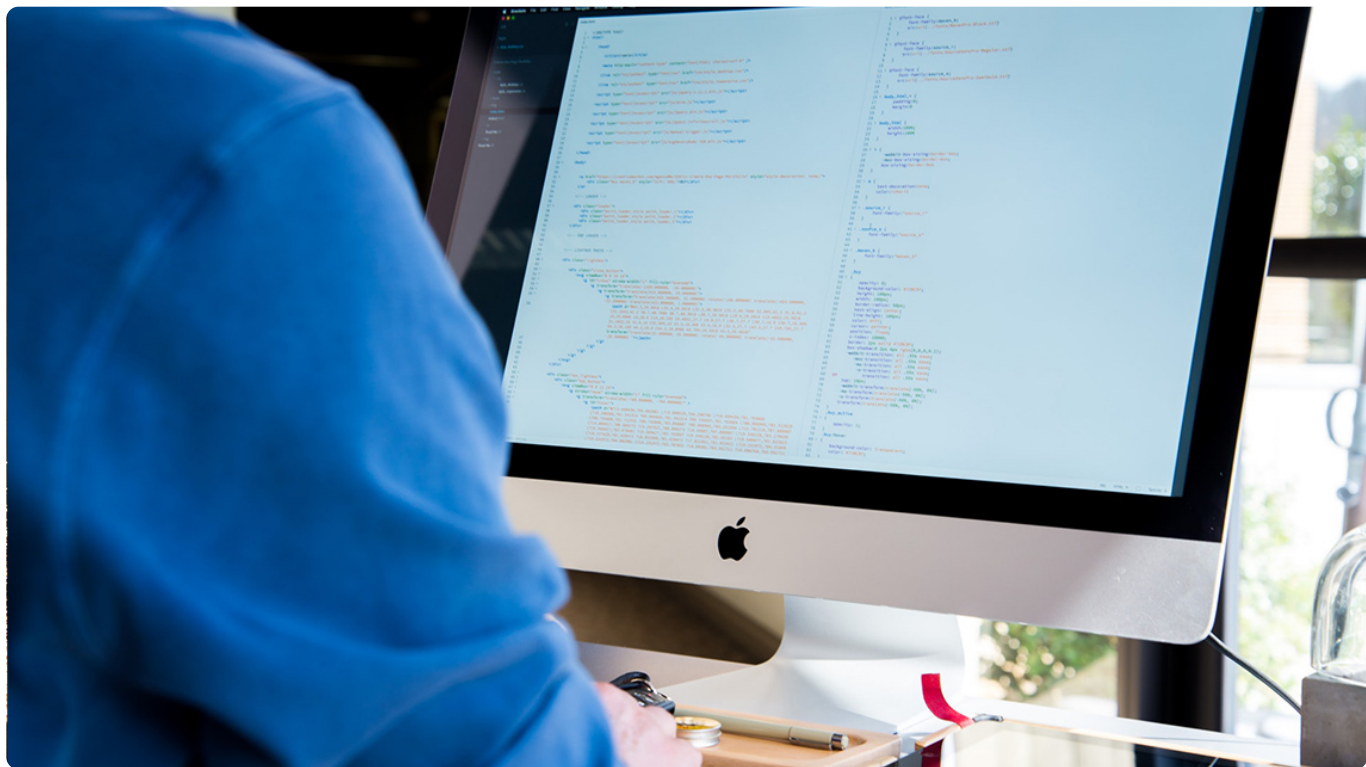


第36讲 | 搭建你的迷你区块链（实践篇）

2018-06-15 陈浩

深入浅出区块链

[进入课程 >](#)



讲述：黄洲君

时长 02:31 大小 3.47M



上一篇文章中，我们介绍了实现一个迷你区块链的大致思路。今天，我们将通过代码编写，以及简单的功能测试，来完成我们的迷你区块链 Tinychain。

除了正常的测试案例之外，我们还可以构造一些极端测试案例，来观察 Tinychain 的分叉合并，挖矿难度调整等情况。


代码编写

通过前文的分析，□我们已经了解到了实践一个迷你区块链的大致思路。接下来，我将从顶层到底层来搭建区块链。

代码编写 1 Server

从链的顶层设计来看，我们需要一个入口，那么我们就从入口开始：我需要先为整个服务做一些基础设置，最后再来 `Server.run()`。

所以，我们的代码大概是这样的。

 复制代码

```
1 // server setup
2 node my_node;
3 mgbubble::RestServ Server{"webroot", my_node};
4 auto& conn = Server.bind("0.0.0.0:8000");
5 mg_set_protocol_http_websocket(&conn);
6 log::info("main")<<"httpserver started";
7 Server.run();
```

我们首先生成一个 `node` 实例，然后被 `Server` 装载进去，最后设置好 `Server` 启动。

这个 `Server` 主要有两个功用，第一是向本地用户服务，也就是接受命令行，接受本地 `RPC` 调用；第二是接受外部网络传送进来是的新交易，和新的区块。所以 `Server` 是整个节点的入口。

代码编写 2 node

那么这里的 `node` 其实就是区块链的 `node`，里面包含了区块链的基本设置，这些一般都是硬编码在代码中的，例如一般区块链都有个“魔法数”，实际上就是区块链 `ID`，这个 `ID` 会被放在所有消息的开头，如果区块链 `ID` 不匹配，则抛弃接收到的消息。


这里的区块链 `ID` 我们设置在这里。

 复制代码


```
1 blockchain(uint16_t id = 3721):id_(id) {
2     id_ = id;
3     create_genesis_block();
4 }
```

代码中所展示的 `id_` 就是区块链 `ID`，在 `Tinychain` 的案例中，我也是硬编码的。

在一个 node 当中，至少要包含 network、blockchain、miner 三个模块。

 复制代码

```
1 public:
2     void miner_run(address_t address);
3     blockchain& chain() { return blockchain_; }
4     network& p2p() { return network_; }
```

 复制代码


```
1 private:
2     network network_;
3     blockchain blockchain_;
4     miner miner_{blockchain_};
```

network 也就是 P2P 网络类，blockchain 是区块链的核心类，miner 是共识模块下的核心类，三者被聚合到 node 中。

同时，node 也会提供的一些 blockchain 和 miner 的接口，方便 Server 层调用。

代码编写 3 blockchain

一个 blockchain 实例，应当包含下面的内容。


 复制代码

```
1     uint16_t id_;
2     block genesis_block_;
3     chain_database chain_;
4     key_pair_database key_pair_database_;
5     memory_pool_t pool_;
```

genesisblock 就是创世区块，这个是预先生成好的。genesis_block 的信息也是被硬编码在代码中，我在 Tinychain 的例子为了方便测试，每个 genesis_block 都是可以自行生成的。

chaindatabase 的 chain 是相对于 memorypool 而言的，chain 就是已经经过确认，并且在本地持久化存储的区块数据（由于时间有限，Tinychain 的案例中还未实现持久化存储，可以后续升级替换）。

memory_pool 是指还未经过确认，暂时驻留在内存中的交易池，交易池中的交易会在挖矿时，被导入到新的区块中。


 复制代码

```
1 // 装载交易
2 new_block.setup(pool);
```

这里的 pool 就是交易池。

key_pair_database 是指专门存储用户的私钥的数据库，同时提供私钥管理。

同时 blockchain 也负责统一对外提供上述功能的接口。


 复制代码

```
1 // 获取当前节点高度
2 uint64_t height() { return chain_.height(); }
3 // 获取当前节点最新区块
4 block get_last_block();
5 // 查询指定区块
6 bool get_block(sha256_t block_hash, block& out);
7 // 查询指定交易
8 bool get_tx(sha256_t tx_hash, tx& out);
9 // 查询目标地址的余额
10 bool get_balance(address_t address, uint64_t balance);
11 // 获取当前区块链的 ID
12 auto id() {return id_;}
13 // 获得交易池数据
14 memory_pool_t pool() { return pool_; }
15 // 区块打包成功以后，用于清空交易池
16 void pool_reset() { pool_.clear(); }
17 // 从网络中收集未确认的交易到交易池
18 void collect(tx& tx) {
19     pool_.push_back(tx);
20 }
21
22 void merge_replace(block_list_t& block_list);
```


除了上述接口之外，blockchain 还负责当发现自己处于较短的分叉链上时，自动合并到最长链。

代码编写 4 network

在 network 中，可用的地址簿代表了可用的其他对等节点，至少是连接过成功一次的。

 复制代码

```
1 public:
2     void broadcast(const block& block);
3     void broadcast(const tx& transaction);
4     void process(event_t ev, func_t f);
```

 复制代码

```
1 private:
2     endpoint_book_t book_;
3     channels_t channels_;
```

地址簿会随着网络的变化进行更新，实时状态的地址簿是驻留在内存中的，当节点关闭是，会被刷到持久化存储中。


channels 代表了已经激活的连接，这些连接可以被 broadcast 接口使用，当本地节点产生新的区块和交易时，会调起这些 channels。

当 P2P 网络产生了新的事件时，会通过 process 接口处理新到达的交易和区块，这一事件会传导给 blockchain 模块。

代码编写 5 consensus


consensus 的含义为共识，共识会在两种情况下产生，第一是对本地生产的交易进行验证，第二是外来的区块和交易进行验证。

无论是哪种情况，他们遵循的验证规则是一样的。validate_tx 和 validate_block 分别承担了这样的功能。

 复制代码

```
1 bool validate_tx(const tx& new_tx) ;
2
3 bool validate_block(const tx& new_block) ;
```


除了验证区块之外，还涉及到提供基础挖矿设施。我们知道挖矿分为两种，一种叫做 solo 挖矿，另外一种叫做联合挖矿。其实无论哪种挖矿类型，都必须用到 miner 类。

 复制代码

```
1 public:
2     // 开始挖矿
3     void start(address_t& addr);
4     inline bool pow_once(block& new_block, address_t& addr);
5     // 填写自己奖励—coinbase
6     tx create_coinbase_tx(address_t& addr);
7
8 private:
9     blockchain& chain_;
```

miner 类展示了在 solo 挖矿情况下，支持开始挖矿以及计算自己的 coinbase 的过程。

实际 pow_once 的挖矿代码如下，pow_once 被 start 调用，start 里面是一个死循环，死循环里面包了 pow_once 函数。

 复制代码

```
1 bool miner::pow_once(block& new_block, address_t& addr) {
2
3     auto&& pool = chain_.pool();
4
5     auto&& prev_block = chain_.get_last_block();
6
7     // 填充新块
8     new_block.header_.height = prev_block.header_.height + 1;
9     new_block.header_.prev_hash = prev_block.header_.hash;
10
11     new_block.header_.timestamp = get_now_timestamp();
```

```

12
13     new_block.header_.tx_count = pool.size();
14
15     // 难度调整:
16     // 控制每块速度, 控制最快速度, 大约 10 秒
17     uint64_t time_peroid = new_block.header_.timestamp - prev_block.header_.timestamp;
18     //log::info("consensus") << "target:" << ncan;
19
20     if (time_peroid <= 10u) {
21         new_block.header_.difficulty = prev_block.header_.difficulty + 9000;
22     } else {
23         new_block.header_.difficulty = prev_block.header_.difficulty - 3000;
24     }
25     // 计算挖矿目标值, 最大值除以难度就目标值
26     uint64_t target = 0xffffffffffffffff / prev_block.header_.difficulty;
27
28     // 设置 coinbase 交易
29     auto&& tx = create_coinbase_tx(addr);
30     pool.push_back(tx);
31
32     // 装载交易
33     new_block.setup(pool);
34     // 计算目标值
35     for ( uint64_t n = 0; ; ++n) {
36         // 尝试候选目标值
37         new_block.header_.nonce = n;
38         auto&& jv_block = new_block.to_json();
39         auto&& can = to_sha256(jv_block);
40         uint64_t ncan = std::stoull(can.substr(0, 16), 0, 16); // 截断前 16 位, 转换 uint
41
42         // 找到了
43         if (ncan < target) {
44             //log::info("consensus") << "target:" << ncan;
45             //log::info("consensus") << "hash  :" << to_sha256(jv_block);
46             new_block.header_.hash = can;
47             log::info("consensus") << "new block :" << jv_block.toStyledString();
48             log::info("consensus") << "new block :" << can;
49             return true;
50         }
51     }

```


上面的代码从一开始到 for 循环之前, 都可以提取出来, 做成叫做 getblocktemplate 的接口, getblocktemplate 是一种 JSON-RPC 调用。

通过这个调用, 就可以把挖矿的状态信息分享给其他矿机, 矿机拿到 blocktemplate 以后直接进行 nonce 部分暴力搜索即可。

代码编写 6 database

database 是偏底层的接口，主要的功能有两个，第一是提供区块和私钥的持久化存储，第二是提供交易和区块的查询接口。

上文 blockchain 中的 blockchain_database 和 keypair_database 都是从 database 派生过来的。

 复制代码

```
1
2 key_pair_database
3
4 // 相当于是本地钱包的私钥管理
5 class key_pair_database
6 {
7 public:
8     key_pair get_new_key_pair();
9     const key_pair_database_t& list_keys() const;
10 private:
11     key_pair_database_t key_pair_database_;
12 };
13
14
15 blockchain_database
16
17 public:
18     uint64_t height();
19
20     auto get_last_block();
21
22     bool get_block (const sha256_t block_hash, block& b);
23
24     bool get_tx (const sha256_t tx_hash, tx& t);
25
26     bool push_block (const block& b);
27
28     bool pop_block (cconst sha256_t block_hash);
29
30 private:
31     chain_database_t chain_database_;
```

代码编写 7 commands

commands 提供了开发者命令行交互接口。


```
1
2     bool exec(Json::Value& out);
3
4     static const vargv_t commands_list;
5
6 private:
7     vargv_t vargv_;
8     node& node_;
```

首先得有一个可识别的命令列表，接着是执行接口，例如命令行发起生成新 key_pair 的过程，执行 getnewkey 命令。

先被 command 解析，接着执行 exec，执行的时候需要用到 node 对象。

实际上 command 类比较繁琐，因为一个功能复杂的钱包，维护的命令和种类可能多达几十种。

同时命令又可以被 JSON-RPC 调用，所以一般命令行客户端本身就是一个轻量级的 http-client。

```
1
2     std::string url{"127.0.0.1:8000/rpc"};
3     // HTTP request call commands
4     HttpRequest req(url, 3000, reply_handler(my_impl));
```

代码编写 8 基础类

基础类是实际生成公私钥对、构建交易 tx 的基本单元类，构建区块的基本单元类。

```
1 key_pair:
2 class key_pair
3 {
4 public:
5     key_pair() {
6         private_key_ = RSA::new_key();
```

```

7         public_key_ = private_key_.public_key();
8     }
9
10    address_t address();
11    sha256_t public_key() const;
12    uint64_t private_key() const;
13
14    // ... 一些序列化接口 (tinychain 中是 Json)
15 private:
16     private_key_t private_key_;
17     public_key_t public_key_;
18
19 tx:
20 public:
21     input_t inputs() const { return inputs_; }
22     output_t outputs() const { return outputs_; }
23     sha256_t hash() const { return hash_; }
24
25 private:
26     input_t inputs_;
27     output_t outputs_;
28     sha256_t hash_;
29
30 block:
31 class block
32 {
33 public:
34     typedef std::vector<tx> tx_list_t;
35
36     struct blockheader {
37         uint64_t nonce{0};
38         uint64_t height{0};
39         uint64_t timestamp{0};
40         uint64_t tx_count{0};
41         uint64_t difficulty{0};
42         sha256_t hash;
43         sha256_t merkel_root_hash; //TODO
44         sha256_t prev_hash;
45
46     };
47     // ... 一些其他接口和序列化函数
48     std::string to_string() {
49         auto&& j = to_json();
50         return j.toStyledString();
51     }
52
53     sha256_t hash() const { return header_.hash; }
54
55     void setup(tx_list_t& txs) {tx_list_.swap(txs);}
56
57 private:
58     blockheader header_;

```

首次运行

我们编写完基础类和基本结构的代码之后，就可以运行试一试。

编译成功是这样子的。

```

1. chenhao@chenhaodeMacBook-Pro: ~/workspace/tinychain/build (zsh)
x ..tinychain/src (zsh) x2 x ..tinychain/build (zsh) x3 x ..tinychain/build/bin (zsh) x4
[ 91%] Linking CXX executable ../bin/tinychain
cd /Users/chenhao/workspace/tinychain/build/src && /usr/local/Cellar/cmake/3.11.3/bin/cmake -E cmake_link_script CMakeFiles/tinychain.dir/link.txt --verbose=1
/Library/Developer/CommandLineTools/usr/bin/c++ -std=c++14 -fstrict-aliasing -fvisibility=hidden -Wall -Werror -Wstrict-aliasing=2 -Wno-unused-parameter -Wno-unused-variable -Wno-type-limits -Wno-deprecated-register -Wno-reorder -g -Wl,-search_paths_first -Wl,-headerpad_max_install_names CMakeFiles/tinychain.dir/blockchain.cpp.o CMakeFiles/tinychain.dir/commands.cpp.o CMakeFiles/tinychain.dir/consensus.cpp.o CMakeFiles/tinychain.dir/database.cpp.o CMakeFiles/tinychain.dir/lib/Mongoose.cpp.o CMakeFiles/tinychain.dir/lib/RestServ.cpp.o CMakeFiles/tinychain.dir/lib/exception/Error.cpp.o CMakeFiles/tinychain.dir/lib/exception/Exception.cpp.o CMakeFiles/tinychain.dir/lib/exception/Instances.cpp.o CMakeFiles/tinychain.dir/lib/logging.cpp.o CMakeFiles/tinychain.dir/lib/sha256.cpp.o CMakeFiles/tinychain.dir/lib/utility/Stream.cpp.o CMakeFiles/tinychain.dir/lib/utility/Stream_buf.cpp.o CMakeFiles/tinychain.dir/main.cpp.o CMakeFiles/tinychain.dir/network.cpp.o CMakeFiles/tinychain.dir/node.cpp.o CMakeFiles/tinychain.dir/tinychain.cpp.o -o ../bin/tinychain /usr/local/lib/libboost_date_time-mt.a ../lib/libmongoose.a
[ 91%] Built target tinychain
/Library/Developer/CommandLineTools/usr/bin/make -f cli-tinychain/CMakeFiles/cli-tinychain.dir/build.make cli-tinychain/CMakeFiles/cli-tinychain.dir/depend
cd /Users/chenhao/workspace/tinychain/build && /usr/local/Cellar/cmake/3.11.3/bin/cmake -E cmake_depends "Unix Makefiles" /Users/chenhao/workspace/tinychain /Users/chenhao/workspace/tinychain/cli-tinychain /Users/chenhao/workspace/tinychain/build /Users/chenhao/workspace/tinychain/build/cli-tinychain /Users/chenhao/workspace/tinychain/build/cli-tinychain/CMakeFiles/cli-tinychain.dir/DependInfo.cmake --color=
/Library/Developer/CommandLineTools/usr/bin/make -f cli-tinychain/CMakeFiles/cli-tinychain.dir/build.make cli-tinychain/CMakeFiles/cli-tinychain.dir/build
[ 95%] Building CXX object cli-tinychain/CMakeFiles/cli-tinychain.dir/main.cpp.o
cd /Users/chenhao/workspace/tinychain/build/cli-tinychain && /Library/Developer/CommandLineTools/usr/bin/c++ -DBCX_STATIC=1 -DBOOST_NO_AUTO_PTR=1 -DBOOST_NO_TYPEID=1 -DMVS_DEBUG=1 -I/usr/local/include -I/Users/chenhao/workspace/tinychain/contrib -I/Users/chenhao/workspace/tinychain/include -std=c++14 -fstrict-aliasing -fvisibility=hidden -Wall -Werror -Wstrict-aliasing=2 -Wno-unused-parameter -Wno-unused-variable -Wno-type-limits -Wno-deprecated-register -Wno-reorder -Wno-deprecated-declarations -g -o CMakeFiles/cli-tinychain.dir/main.cpp.o -c /Users/chenhao/workspace/tinychain/cli-tinychain/main.cpp
[100%] Linking CXX executable ../bin/cli-tinychain
cd /Users/chenhao/workspace/tinychain/build/cli-tinychain && /usr/local/Cellar/cmake/3.11.3/bin/cmake -E cmake_link_script CMakeFiles/cli-tinychain.dir/link.txt --verbose=1
/Library/Developer/CommandLineTools/usr/bin/c++ -std=c++14 -fstrict-aliasing -fvisibility=hidden -Wall -Werror -Wstrict-aliasing=2 -Wno-unused-parameter -Wno-unused-variable -Wno-type-limits -Wno-deprecated-register -Wno-reorder -Wno-deprecated-declarations -g -Wl,-search_paths_first -Wl,-headerpad_max_install_names CMakeFiles/cli-tinychain.dir/main.cpp.o -o ../bin/cli-tinychain ../lib/libmongoose.a ../lib/libjsoncpp.a
[100%] Built target cli-tinychain
/usr/local/Cellar/cmake/3.11.3/bin/cmake -E cmake_progress_start /Users/chenhao/workspace/tinychain/build/CMakeFiles 0
chenhao@chenhaodeMacBook-Pro ~/workspace/tinychain/build master

```

我们可以看到有 Tinychain 和 Cli-tinychain。

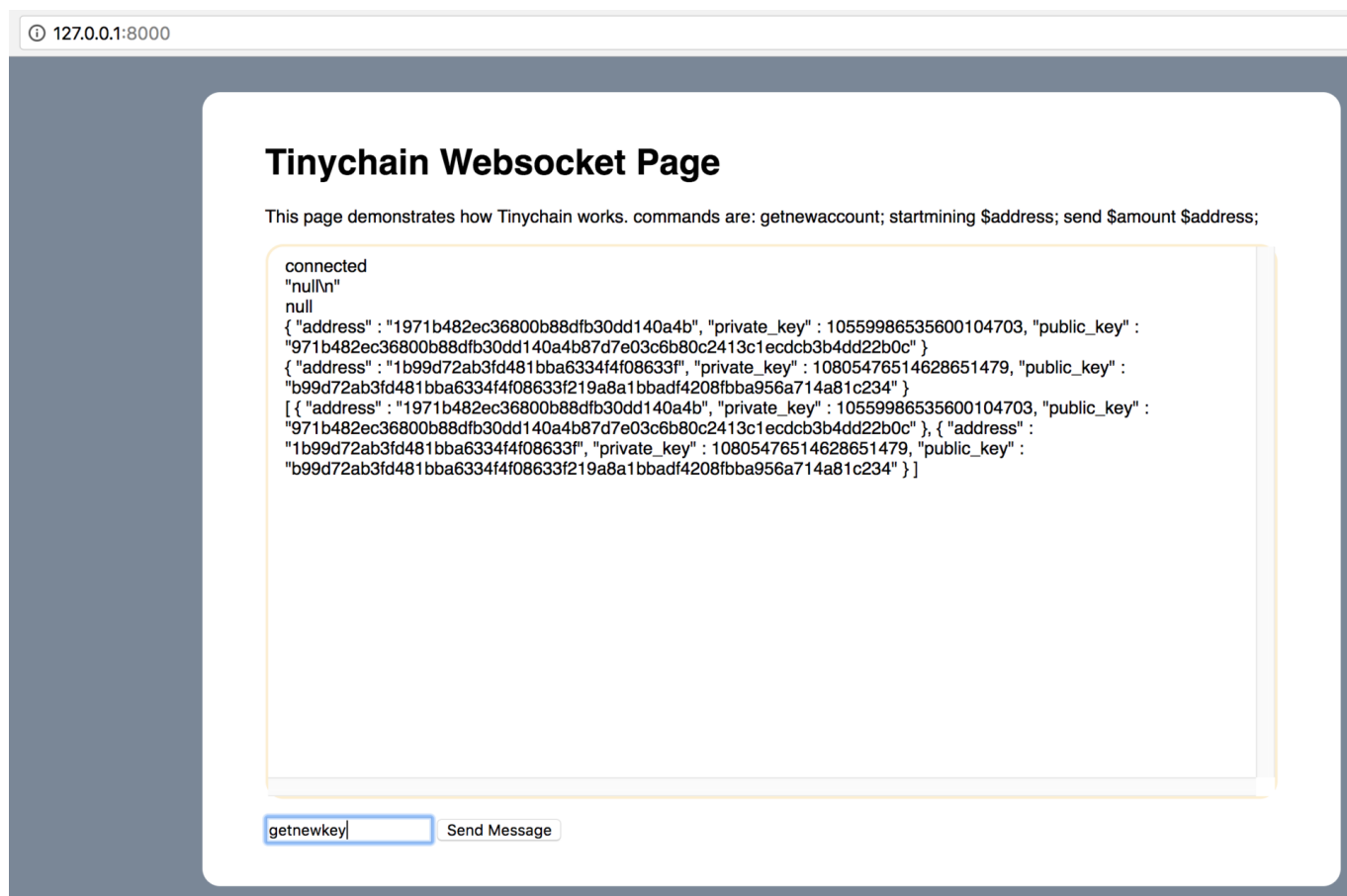
```

chenhao@chenhaodeMacBook-Pro ~/workspace/tinychain/build/bin master l
total 5240
-rwxr-xr-x  1 chen hao  staff   868852 Jun 10 23:14 cli-tinychain
-rw-r--r--  1 chen hao  staff    67389 Jun 10 18:25 debug.log
-rw-r--r--  1 chen hao  staff     318 Jun 10 17:59 error.log
-rwxr-xr-x  1 chen hao  staff  1736480 Jun 10 23:14 tinychain
drwxr-xr-x  3 chen hao  staff     96 Jun  9 13:20 webroot

```

Tnychain 就是我们的核心程序，cli-tinychain 就是我们的命令行客户端。

实际上我在 Server 里还嵌入了一个可视化的 Websocket 界面。



只需要在 Tinychain 可执行文件同目录下创建 webroot 文件夹，将 etc 底下的 index 放入 webroot 下，接着打开浏览器 127.0.0.1:8000 就可以看到了。

实际上这个页面我想做成区块的监视页面，只是还没改造完成，目前支持发送命令。

我们开始首次运行 Tinychain。


```
chenhao@chenhaodeMacBook-Pro ~/workspace/tinychain/build/bin master • ./tinychain
20180610T182223 INFO [main] started
20180610T182223 INFO [node] node started
20180610T182223 INFO [main] httpserver started
```

运行后，等 node 和 server 全部 started，就可以开始操作命令行了。

也可以通过日志进行监视，但是需要在代码处详细打桩，这次我偷懒了，没有好好打，所以不多，直接查看同目录下 debug.log 和 error.log 即可。

首次挖矿

我们通过./tinychain 启动之后，开始第一次挖矿。

 复制代码

```
1  X chenhao@chenhaodeMacBook-Pro ~/workspace/tinychain/build/bin  master  ./tinychain
2  20180610T232347 INFO [main] started
3  20180610T232347 INFO [node] node started
4  20180610T232347 INFO [main] httpserver started
5  20180610T232356 INFO [consensus] new block :{
6    "header" :
7    {
8      "difficulty" : 9001,
9      "hash" : "",
10     "height" : 1,
11     "merkel_header_hash" : "",
12     "nonce" : 0,
13     "prev_hash" : "00b586611d6f2580e1ea0773ec8b684dc4acf231710519e6272ed7d0c61ed43e",
14     "timestamp" : 1528644236,
15     "tx_count" : 0
16   },
17   "txs" :
18   [
19     {
20       "hash" : "cddf6e838eff470d81155cb4c26fd3a7615b94a00e82f99b1fd9f583d7bc0659",
21       "inputs" :
22       [
23         {
24           "hash" : "00000000000000000000000000000000",
25           "index" : 0
26         }
27       ],
28       "outputs" :
29       [
30         {
31           "address" : "122b03d11a622ac3384904948c4d808",
32           "value" : 1000
33         }
34       ]
35     }
36   ]
37 }
38
39 20180610T232356 INFO [consensus] new block :0de5c36420aab2f7fc9413cfbd21bece697a3491067
40 20180610T232357 INFO [consensus] new block :{
41   "header" :
42   {
43     "difficulty" : 18001,
44     "hash" : "",
45     "height" : 2,
46     "merkel_header_hash" : "",
47     "nonce" : 6048,
```

```

48  "prev_hash" : "0de5c36420aab2f7fc9413cfbd21bece697a349106771dc58b25a6a099d6aa86",
49  "timestamp" : 1528644236,
50  "tx_count" : 0
51 },
52 "txs" :
53 [
54 {
55   "hash" : "cddf6e838eff470d81155cb4c26fd3a7615b94a00e82f99b1fd9f583d7bc0659",
56   "inputs" :
57   [
58     {
59       "hash" : "00000000000000000000000000000000",
60       "index" : 0
61     }
62   ],
63   "outputs" :
64   [
65     {
66       "address" : "122b03d11a622ac3384904948c4d808",
67       "value" : 1000
68     }
69   ]
70 }
71 ]
72 }

```



刚开始挖矿会比较快，随着难度提升，会趋向于稳定到 10 秒种左右一个块，如果长时间不出块，难度会自动降下来。曾经元界的代码在难度调整上有缺陷，遭受了严重的“难度坠落”攻击。

我们可以通过这个位置观察难度调整的情况。

```

20180610T232658 INFO [consensus] new block :{
  "header" :
  {
    "difficulty" : 111001,
    "hash" : "",
    "height" : 19,
    "merkel_header_hash" : "",
    "nonce" : 122340,
    "prev_hash" : "000075971364dc56475e934c30280c4fb7fe8e2a228f97fd4602d9268d734ce6",
    "timestamp" : 1528644401,
    "tx_count" : 0
  },
  "txs" :
  [

```

第一笔交易

我们保持挖矿，接下来发送一笔交易。我们先通过 `getnewkey` 命令获得一个新公私钥对以及对应的地址。

```
chenhao@chenhaodeMacBook-Pro ~/workspace/tinychain/build/bin master ./cli-tinychain getnewkey
{
  "address" : "13389c7ae2872cc16b6561e6067ce02",
  "private_key" : 1129220089613184720,
  "public_key" : "3389c7ae2872cc16b6561e6067ce026b73b3b5384b22c34ce0bb66a90d2e8c03"
}
```

接着发送第一笔交易。

```
20180611T004052 INFO [blockchain-pool] new tx:{
  "hash" : "cae3c831e1e650c19244bc2867845c42cdddb77533736f7572f4b36ec08c3470",
  "inputs" :
  [
    {
      "hash" : "a2875bfce6e140d04e91b66180e36e2d8529963e16c23c6b3ef19a010b70f918",
      "index" : 0
    }
  ],
  "outputs" :
  [
    {
      "address" : "13389c7ae2872cc16b6561e6067ce02",
      "value" : 88888
    }
  ]
}
```

探测到接下来被打包到区块中。

```

20180611T004104 INFO [consensus] new block :{
  "header" :
  {
    "difficulty" : 51001,
    "hash" : "",
    "height" : 7,
    "merkel_header_hash" : "",
    "nonce" : 36795,
    "prev_hash" : "000124118b25b71c7ea66e4bbdfe192e9cdc313e908c24eafc3a643809091bc5",
    "timestamp" : 1528648855,
    "tx_count" : 1
  },
  "txs" :
  [
    {
      "hash" : "cae3c831e1e650c19244bc2867845c42cdddb77533736f7572f4b36ec08c3470",
      "inputs" :
      [
        {
          "hash" : "a2875bfce6e140d04e91b66180e36e2d8529963e16c23c6b3ef19a010b70f918",
          "index" : 0
        }
      ],
      "outputs" :
      [
        {
          "address" : "13389c7ae2872cc16b6561e6067ce02",
          "value" : 88888
        }
      ]
    },
    {
      "hash" : "485d7bed91ef3c9febf5045e19ede40ba4d772b7d1996410d4e79d2c6ee55e8a",
      "inputs" :

```

分叉与合并

区块链分叉是数据全网不一致的表现，通常是矿工□节点行为不一致导致的，常见的有网络分区和协议不兼容，如果同时产生，那么必然会出现两条比较长的分叉链。

在现实情况中，分叉 1 个是最常见的，2 个已经非常罕见了，3 个以上基本是网络分区造成的。

如果我们要在 Tinychain 中实践网络分区和分叉，我们需要构建局域网多节点私链环境，可以通过 docker 来试验。

通过本文，你可以看到即使是搭建一个迷你区块链，它的工作量也是巨大的，□其中不仅仅只是组合几个基础组件那么简单，还要涉及各个模块的设计和交互等详细的工作。

由于在短时间内全部搭建以及实现 Tinychain 所有功能是不可行的，在这里，我只为你提供了一些实践的思路。

目前 Tinychain 缺失了 P2P 网络实现、RSA 公私钥对集成、共识模块的交易和区块的验证等内容，我会在后续逐渐完善，你也可以跟我一起补充。

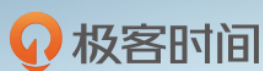
总结

好了，通过今天的代码实践，我们实现了迷你区块链 Tinychain，并且，通过运行与测试 Tinychain，我们了解到了一个最简单区块链的运行原理，希望通过今天的文章，可以帮你加深对区块链技术的理解。

区块链技术只是作为基础设施，服务于广大的开发者和业务需求。目前区块链的发展远远不止 Tinychain 中所展现的样子，我们还需要去考虑区块链 2.0 智能合约，如何设计 Token 经济等一些问题。

随着区块链的发展和应用规模，区块链安全问题也日益突出，所以今天的问题是，如果要攻击 Tinychain，可以采取什么手段呢？你可以给我留言，我们一起讨论。

感谢你的收听，我们下次再见。



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35讲 | 搭建你的迷你区块链（设计篇）

下一篇 第36讲 | 搭建你的迷你区块链（部署篇）

精选留言 (4)

写留言



huangshao...

2018-06-16

2

老师，对应的代码可以放到您的github上吗

展开

作者回复: 你好，已经投放哦。 github.com/betachen/tinychain



许星昊

2018-06-29

1

老师，可以推荐几本区块链相关的书籍好吗？

展开



Eric

2018-12-13

1

陈老师，p2p network类，例子里是不是还没实现好？

展开



肖水平

2018-06-20

1

老师，代码在mac上编译不通过

展开