比特币源码研读系列1 初始化

为了进一步理解比特币的工作机制，接下来我们将通过阅读其源代码来进一步理解比特币的工作原理。”比特币源码研读”系列文章比较偏专业性，针对的是**具备一定的编码能力且对比特币非常感兴趣**的读者，对于那些只是有兴趣的读者可简单阅读，理解个大概即可。

由于老王专业知识有限，文中如有错误，欢迎指正，非常欢迎这方面的专业人士和老王一起探讨。

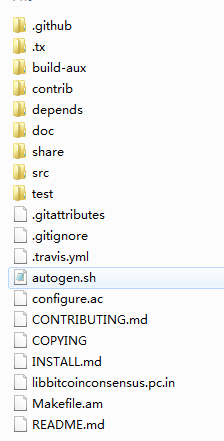
自从中本聪在2009年发布比特币源码以来，比特币的源码经过了多年的完善，引入了很多重要的改进，笔者下载的时候已经是0.15.0版本了，因此下面的解读也是以该版本的代码为依据。

下载地址：github

代码语言：c++

源码版本：0.15.0

目录结构：



这其中有些重要的文件夹，doc文件夹是一些说明文件，包括代码版本说明，编译文档等，src就是源码目录，是我们接下来要解读的文件夹，test文件夹是一些功能测试文件夹。由于代码量非常大，如果全复制出来会淹没文章的主体，因此很多情况我都会采用更易阅读的伪代码的形式来展示，尽量不贴源码，如果想了解具体的细节的读者可以自己下载源码阅读。

好了，源码解读第一篇：比特币环境的初始化过程。

在看代码之前让我们想想，当比特币程序启动时应该需要做哪些事：

1. 初始化网络，准备接收来自其他人的tx（transaction，中文称交易）或block（中文称之为区块，block chain就是中文说的区块链），或者发送存在本机的tx或挖矿刚生成的block
2. 初始化我的钱包，整理出账户下的比特币及每个币的使用情况

下面我们看看比特币的代码是怎么做的：

学过c或c++的读者应该都知道程序启动的入口是main函数，该函数定义在bitcoind.cpp文件中，main函数定义如下：

*int main(int argc, char\* argv[])*

*{*

*SetupEnvironment();*

*// Connect bitcoind signal handlers*

*noui\_connect();*

*return (AppInit(argc, argv) ? EXIT\_SUCCESS : EXIT\_FAILURE);*

*}*

我们挨个看这几个方法，SetupEnvironment和noui\_connect方法都是做一些准备工作，例如设置程序运行环境为windows，linux等，注册日志处理，我们来看AppInit方法。

AppInit方法大概有几个过程：

1. 首先检查参数，文件目录等
2. AppInitBasicSetup方法初始化一些基本设置
3. AppInitSanityChecks检查所有依赖的库是否齐全
4. AppInitMain方法是我们重点关注的

下面我们继续看AppInitMain方法：

1. InitSignatureCache方法将签名缓存初始化，每次的签名会保存在缓存中
2. InitScriptExecutionCache方法将脚本执行缓存初始化
3. AppInitServers方法初始化http server
4. VerifyWallets方法检查钱包数据库的一致性
5. new CBlockTreeDB().writeReIndexing()方法加载blockchain并检查创世块
6. LoadBlockIndex方法从硬盘加载一些block索引到内存中
7. StartWallets方法启动我的钱包，我们需要了解它是如何启动的

下面我们接着看StartWallets方法：

1. ReacceptWalletTransaction方法把钱包关联的tx加载到内存中。它是如何寻找tx并检查的呢？
2. scheduler.scheduleEvery(MaybeCompactWalletDB, 500)定义了一个线程，定期去刷新钱包

下面我们看看ReacceptWalletTransactions源码：

*void CWallet::ReacceptWalletTransactions()*

*{*

*// If transactions aren't being broadcasted, don't let them into local mempool either*

*if (!fBroadcastTransactions)*

*return;*

*LOCK2(cs\_main, cs\_wallet);*

*std::map<int64\_t, CWalletTx\*> mapSorted;*

*// Sort pending wallet transactions based on their initial wallet insertion order*

*for (std::pair<const uint256, CWalletTx>& item : mapWallet)*

*{*

*const uint256& wtxid = item.first;*

*CWalletTx& wtx = item.second;*

*assert(wtx.GetHash() == wtxid);*

*int nDepth = wtx.GetDepthInMainChain();*

*if (!wtx.IsCoinBase() && (nDepth == 0 && !wtx.isAbandoned())) {*

*mapSorted.insert(std::make\_pair(wtx.nOrderPos, &wtx));*

*}*

*}*

*// Try to add wallet transactions to memory pool*

*for (std::pair<const int64\_t, CWalletTx\*>& item : mapSorted)*

*{*

*CWalletTx& wtx = \*(item.second);*

*LOCK(mempool.cs);*

*CValidationState state;*

*wtx.AcceptToMemoryPool(maxTxFee, state);*

*}*

*}*

该方法首先检查是否已设置成可以广播，然后对mapWallet中的tx进行遍历，做如下检查：

1. 检查tx的hash值
2. 检查tx在区块链中的深度。
3. 检查tx是否是coinbase，是否是已丢弃状态。
4. 如果这些检查通过后把这些tx按顺序放入到内存池中（memory pool）

在下一篇进一步介绍前，我先解释这里提到的几个概念：

1. 深度nDepth。深度指的是tx所在block与主链上最新block的间隔block数目。
2. 主链mainchain。主链指的是当前比特币网络中公认的最长的那条区块链，在客户端同时还存在有活跃链chainActive。
3. Coinbase tx。这是tx的一种，在每个block中第一个都是coinbase tx，coinbase tx通常没有关联任何input tx，只有output，最典型的例子就是挖矿得到的系统奖励货币就是放在coinbase中的。
4. Memory pool。在代码中是用mempool表示的，它保存着那些还不在block中的tx，在后面的运行中经常会遇到。

好了，本篇就到这里了，从下一篇开始我们来研读如何创建一个新的tx。