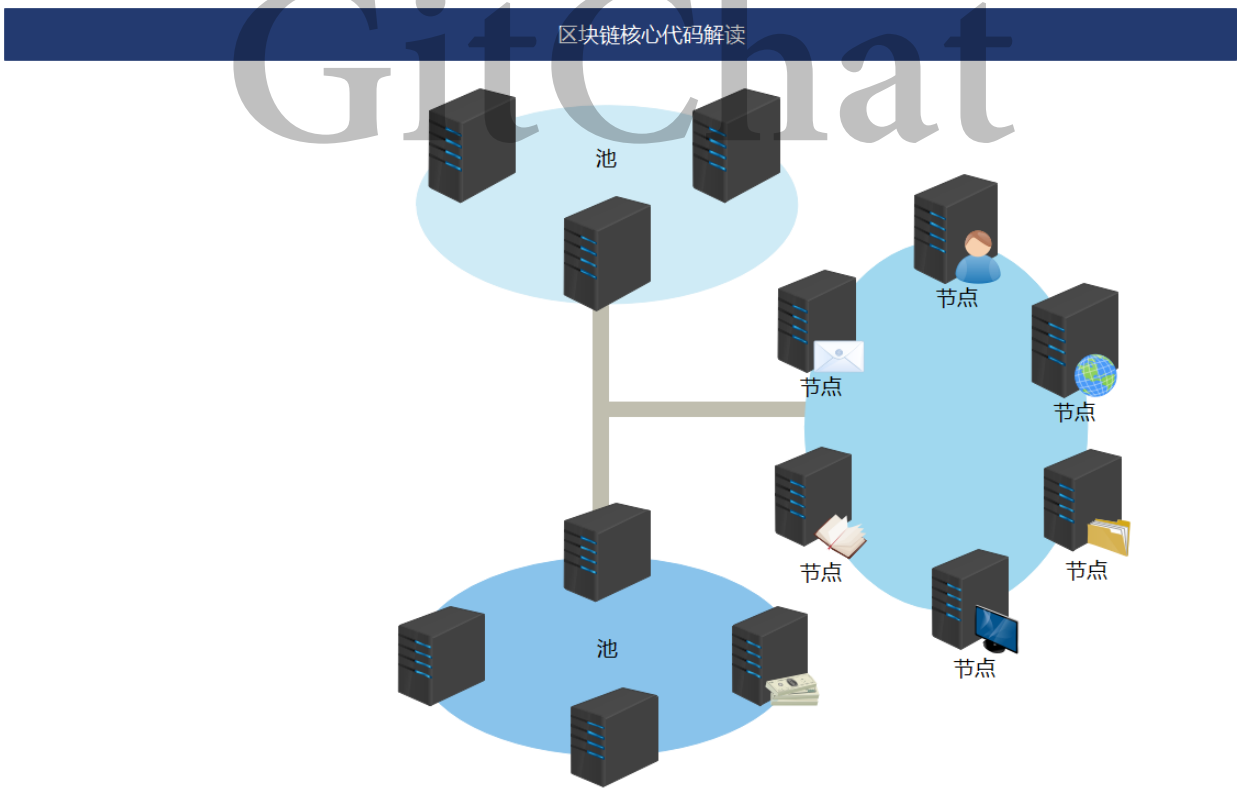
手动做一个自己的 COIN 客户端：附区块链核心代码解读

基本原理和设计

2008年初，中本聪团队发布了一篇名为“比特币：一种点对点的电子现金系统”学术论文，之所以选择在金融危机这年发布，也许别有深意。他认为传统货币最根本的问题在于信任，银行必须让人信任它能帮我们管好钱财，但是银行却在用货币制造信贷泡沫，通货膨胀使人财富缩水。首先中本聪把比特币定义为一种点对点的电子现金系统，目的很明确：希望这套系统不要依赖任何一个中心，比如中央银行，所以这个系统肯定是分布式系统。

区块链的概念是在中本聪设计比特币系统的时候首次提出，也是区块链技术第一次得以落地并且运用得最成功的一次。比特币是一种业务场景，区块链是比特币这种业务场景实现的技术基础，要了解区块链，首先我们看下下图所示的区块链项目基本运行机制。



去中心化的重要性

主要是为了安全与信任，互联网从当初到现集群服务方式，无非就是业务逻辑的需要与数据的可靠性，所以安全与信任是最重要的，之前的模式都基于 B/S 或是 C/S，比如微信支付，当 A 用户向 B 用户支付，事实上是 A 发起请求然后微信支付系统网关进行确认，

然后把 A 的钱转向 B 的帐户。这个时间内，如果微信支付网关服务器一旦出现问题，所有的用户都会受到影响，一旦被黑客攻击，数据就非常容易被修改，而分布式网络中，可以理解为 CDN 机制，比如 Nginx 进行数据缓存，当有100台 CDN，如果中间有几台服务器数据丢了，或是被黑客攻击，这时不会影响全网的服务，因为中心有一个调度机制，可以快速切换到新的或是可靠的 CDN 节点上，分布式系统中，全球每个节点是客户端同时也是服务器，除非同时50.1%以上节点被篡改，如果超过这个值，那就意味调度系统不能正常分析那个是真实的服务器，所以去中心化的机制，所有的数据都是透明，不属于任何一个中心服务器或是客户端，每个人都是节点上的一个贡献者，一个维护者。可能有人会问，本来是服务器上的数据，我们不可见，现在把数据都安装到本地，是不是意味着客户端有存在数据，那是不是更不安全了，事实上是这样，但是这也是区块链项目的一个属性，所有的数据都是通过加密方式存在。

重要概念

接下来，我们看看区块链中的重要概念。

1.P2P 协议

P2P 是一种分布式网络，网络的参与者共享他们所拥有的一部分硬件资源（处理能力、存储能力、网络连接能力），这些共享资源需要由网络提供服务和内容，能被其它对等节点（Peer）直接访问而无需经过中间实体。在此网络中的参与者既是资源（服务和内容）提供者（Server），又是资源（服务和内容）获取者（Client）。

其有两个特点，一是无中央服务器，二是用户之间互联并分享文件或通信。

2.共识机制

共识机制，就是所有记账节点之间如何区域达成共识，来选择和认定记录的真实性和有效性。全网认可的是最长的一条区块链，因为在此之上的工作量最大。如果想要修改某个区块内的交易信息，就必须将该区块和后面所有区块的信息进行修改。这种共识机制既可以作为认定的手段，又可以避免虚假交易和信息篡改。

3.智能合约

智能合约主要基于区块链系统里可信的不可篡改的数据，自动地执行一些预先定义好的规则和条款，如新区块的自动记录。便捷快速智能。

4.非对称加密

非对称加密，即在加密和解密的过程中使用一个“密钥对”，“密钥对”中的两个密钥具有非对称的特点，即在信息发送过程中，发送方通过一把密钥将信息加密，接收方在收到信息后，只有通过配对的另一把密钥才能对信息进行解密。非对称加密使得任何参与者更容易达成共识，将价值交换中的摩擦边界降到最低，还能实现透明数据后的匿名性，保护个人隐私。

挖矿基本原理

区块链网络中，数据以文件的形式被永久记录，我们称之为区块。一个区块是一些或所有最新比特币交易的记录集，且未被其他先前的区块记录。区块可以想像为一个城市记录者的记录本上的单独一页纸（对房地产产权的变更记录）或者是股票交易所的总帐本。在绝大多数情况下，新区块被加入到记录最后（在比特币中的名称为：块链），一旦写上，就再也不能改变或删除。每个区块记录了它被创建之前发生的所有事件。

区块结构如下图所示。

数据项	描述	长度
Magic no	总是0xD9B4BEF9	4 字节
Blocksize	到区块结束的字节长度	4 字节
Blockheader	包含6个数据项	80字节
Transaction counter	正整数 VI = VarInt	1 – 9字节
transactions	交易列表(nonull)	不定大小

每个区块包括一些或所有近期交易、前一个区块的引用，以及其他数据。它还包括一个挖矿难度的答案——该答案对每个区块是唯一的。新区块如果没有正确答案，不能被发送到网络中——“挖矿”的过程本质上是在竞争中“解决”当前区块。每个区块中的数学问题难以解决，但是一旦发现了一个有效解，其他网络节点很容易验证这个解的正确性，对于给定的区块可能有多个有效解，但对于要解决的区块来说只需一个解。

因为每解决一个区块，都会得到新产生的比特币奖励，每个区块包含一个记录，记录中的比特币地址是有权获得比特币奖励的地址。这个纪录被称为生产交易或者 Coinbase 交易，它经常是每个区块的第一个交易。每个区块生产的比特币数量是50个，每产生21万个区块后减少一半（时间大约是4年）。

发送者在网络中广播比特币交易，所有试图解决区块的矿工节点，收集了这些交易记录，把它们加到矿工节点正在解决的区块中。

挖矿难度由比特币网络自动调整，使之实现平均每小时解决6个区块的目标。每2016个区块（大约两周）后，所有客户端把新区块的实际数目与目标数量相比较，并且按照差异的百分比调整目标 Hash 值，来增加（或降低）产生区块的难度。

共识机制

共识机制为区块链应用项目稳定性的一个重要的属性，而事实上没有任何一个机制是完美的。

其主要包括以下模式，很多项目都采用混合机制。如下实例也是通过一个 POS+POW 方式进行的。

POW：Proof of Work，工作证明。

比特币在 Block 的生成过程中使用了 POW 机制，一个符合要求的 Block Hash 由 N 个前导零构成，零的个数取决于网络的难度值。要得到合理的 Block Hash 需要经过大量尝试计算，计算时间取决于机器的哈希运算速度。当某个节点提供出一个合理的 Block Hash 值，说明该节点确实经过了大量的尝试计算，当然，并不能得出计算次数的绝对值，因为寻找合理 Hash 是一个概率事件。当节点拥有占全网 n% 的算力时，该节点即有 n/100 的概率找到 Block Hash。

POS：Proof of Stake，股权证明。

POS，也称股权证明，类似于财产储存在银行，这种模式会根据你持有数字货币的量和时间，分配给你相应的利息。

简单来说，就是一个根据你持有货币的量和时间，给你发利息的一个制度，在股权证明 POS 模式下，有一个名词叫币龄，每个币每天产生1币龄，比如你持有100个币，总共持有了30天，那么，此时你的币龄就为3000，这个时候，如果你发现了一个 POS 区块，你的币龄就会被清空为0。你每被清空365币龄，你将会从区块中获得0.05个币的利息（假定利息可理解为年利率5%），那么在这个案例中，利息 = 3000 * 5% / 365 = 0.41个币，这下就很有意思了，持币有利息。

DPOS：Delegated Proof of Stake，委任权益证明

比特股的 DPoS 机制，中文名叫做股份授权证明机制（又称受托人机制），它的原理是让每一个持有比特股的人进行投票，由此产生101位代表，我们可以将其理解为101个超级节点或者矿池，而这101个超级节点彼此的权利是完全相等的。从某种角度来看，DPOS 有点像是议会制度或人民代表大会制度。如果代表不能履行他们的职责（当轮到他们时，没能生成区块），他们会被除名，网络会选出新的超级节点来取代他们。DPOS 的出现最主要还是因为矿机的产生，大量的算力在不了解也不关心比特币的人身上，类似演唱会的黄牛，大量囤票而丝毫不关心演唱会的内容。

PBFT：Practical Byzantine Fault Tolerance，实用拜占庭容错算法。

PBFT 是一种状态机副本复制算法，即服务作为状态机进行建模，状态机在分布式系统的不同节点进行副本复制。每个状态机的副本都保存了服务的状态，同时也实现了服务的操作。将所有的副本组成的集合使用大写字母 R 表示，使用0到 |R|-1 的整数表示每一个副本。为了描述方便，假设 |R|=3f+1，这里 f 是有可能失效的副本的最大个数。尽管可以存在多于 3f+1 个副本，但是额外的副本除了降低性能之外不能提高可靠性。

以上主要是目前主流的共识算法。

从时间上来看，这个顺序也是按该共识算法从诞生到热门的顺序来定。

对于 POW，直接让比特币成为了现实，并投入使用。而 POS 的存在主要来自经济学上的考虑和创新。而最终由于专业矿工和矿机的存在，让社区对这个标榜去中心化的算法有

了实质性的中心化担忧，即传闻60％～70%的算力集中在中国。因此后来又出现 DPOS，这种不需要消耗太多额外的算力来进行矿池产出物的分配权益方式。但要说能起到替代作用，DPOS 来单独替代 POW，POS 或者 POW + POS 也不太可能，毕竟存在即合理。每种算法都在特定的时间段中有各自的考虑和意义，无论是技术上，还是业务上。

如果跳出技术者的角度，更多结合政治与经济的思考方式在里面，或许还会跳出更多的共识算法，如结合类似 PPP 概念的共识方式，不仅能达到对恶意者的惩罚性质，还能达到最高效节约算力的目的也说不定。

至于说算法的选择，这里引用万达季总的这一段话作为结束。

> 一言以蔽之，共识最好的设计是模块化，例如 Notary，共识算法的选择与应用场景高度相关，可信环境使用 Paxos 或者 Raft，带许可的联盟可使用 PBFT，非许可链可以是 POW、POS、Ripple 共识等，根据对手方信任度分级，自由选择共识机制，这样才是真的最优。

侧链

侧链，是对于某个主链的一个相对概念。英文为 sidechains。侧链概念的提出主要是为了实现比特币和其他数字资产在多个区块链间的转移，简单的说，侧链就是一种使货币在两条区块链间移动的机制。侧链是以融合的方式实现加密货币金融生态的目标，而不是像其它加密货币一样排斥现有的系统。利用侧链，我们可以轻松的建立各种智能化的金融合约，股票、期货、衍生品等等。

侧链的产生

最开始，侧链的出现是为了弥补比特币区块链运行中的一些问题，比如比特币区块链是一个单一原生的数字资产，不能与其他任何资产相兑换，以及在比特币区块链中，由于本身强大的共识机制反而导致交易缓慢等，这些都需要比特币区块链考虑是否进行技术上的升级，来满足人们对区块链更多的需求。

然而比特币区块链对整个系统的完备性和安全性都有很高的要求，对比特币系统本身的升级改造需要很严密的验证以及需要一个安全升级的途径。为了满足更多更新的需求，就需要一种辅助的区块链。首先，能够实现将比特币以一种虚拟的方式转移到侧链中，然后，不同的侧链可以根据对应的需求进行针对性开发，以帮助比特币区块链或者说主链实现其他需求，而不需要主链频繁的更新。同时，在侧链完成操作之后，在侧链中的资产可以随时的转移回比特币区块链或者主链中，从而实现资产的安全回流。由此就产生了开发侧链的需求。

侧链的原理

为了能够实现侧链，首先需要明确清晰的认识到侧链的目的，在实现主链暂时不能胜任的新需求时，能够将主链上的资产无缝的转移到侧链上，而且侧链上的资产不是一种新的独立的币，因为如果每一种侧链都引人一种新的数字货币，那么，在主链向侧链进行资产互通的过程中还要处理大量的汇率转换问题。所以侧链和主链使用统一的数字货币，货币的发行机制以及本身的安全性都由主链来整体维护，侧链只需要关注技术上的创新就行了。

我们从实现的角度出发，当资产从 A 区块链向 B 区块链转移时，我们在 A 链上创建一个锁定资产的交易，然后在 B 链上创建一笔交易，该交易的输入中包含一个密码学证明凭证，来解锁侧链中的等值资产。而反过来操作的话，就是先将 B 链中的资产通过交易锁定，然后在 A 链中将等值资产解锁，来实现货币在不同区块链中的转移。

其中的 A 链我们称之为主链，B 链则称为侧链。在某些模型中，两条链可以对等地来处理。从概念上讲，我们打算将资产从（初始）主链转移到一条侧链，可能会再转移到别的侧链，最终还能转回至主链，并保全初始资产。一般我们把主链看成是比特币系统，侧链是其他区块链中的某一个。当然，侧链中的币也可以在侧链间传递，并非只能与比特币系统进行往来；不过，由于任何一个最初从比特币系统移动的币都可以移回去，所以不管变成什么样，它仍是个比特币。

要完成转移资产方式的实现，有一个难点，就是侧链是如何知道资产已经在主链上完全锁定了。逆向转移同样也需要考虑这个问题。在上面我们说过，要利用一个密码学证明凭证。而这个证明机制其实就是利用之前在比特币区块链中提到的 SPV 证明，SPV 证明即简单支付验证，可以轻量级的验证某一个支付的安全性，而不必获取整个区块链数据。

我们仍然以比特币区块链作为主链来讲解，首先在主链上将主链币发送至一个特定的输出，这个特定的输出包含了侧链信息的特殊地址。然后在侧链中利用 SPV 证明去验证这个支付是可靠的，没有出现双花等安全性问题，之后就会在侧链中产生一个来自主链的输入，接收到这个输入之后，就可以在侧链中自由的使用这个资产了。

在这个两种链之间资产转移的过程中，我们需要两个等待期来同步这两个链。

分叉

因为区块链是去中心化的数据结构，所以不同副本之间不能总是保持一致。区块有可能在不同时间到达不同节点，导致节点有不同的区块链视角。解决的办法是，每一个节点总是选择并尝试延长代表累计了最大工作量证明的区块链，也就是最长的或最大累计难度的链。节点通过将记录在每个区块中的难度加总起来，得到建立这个链所要付出的工作量证明的总量。只要所有的节点选择最长累计难度的区块链，整个比特币网络最终会收敛到一致的状态。分叉即在不同区块链间发生的临时差异，当更多的区块添加到了某个分叉中，这个问题便会迎刃而解。

手动制做自己的 COIN

主要基于 Bitcoin 方式进行创建一个自己的 COIN，接下来我们了解下基本环境要求与编译工具。

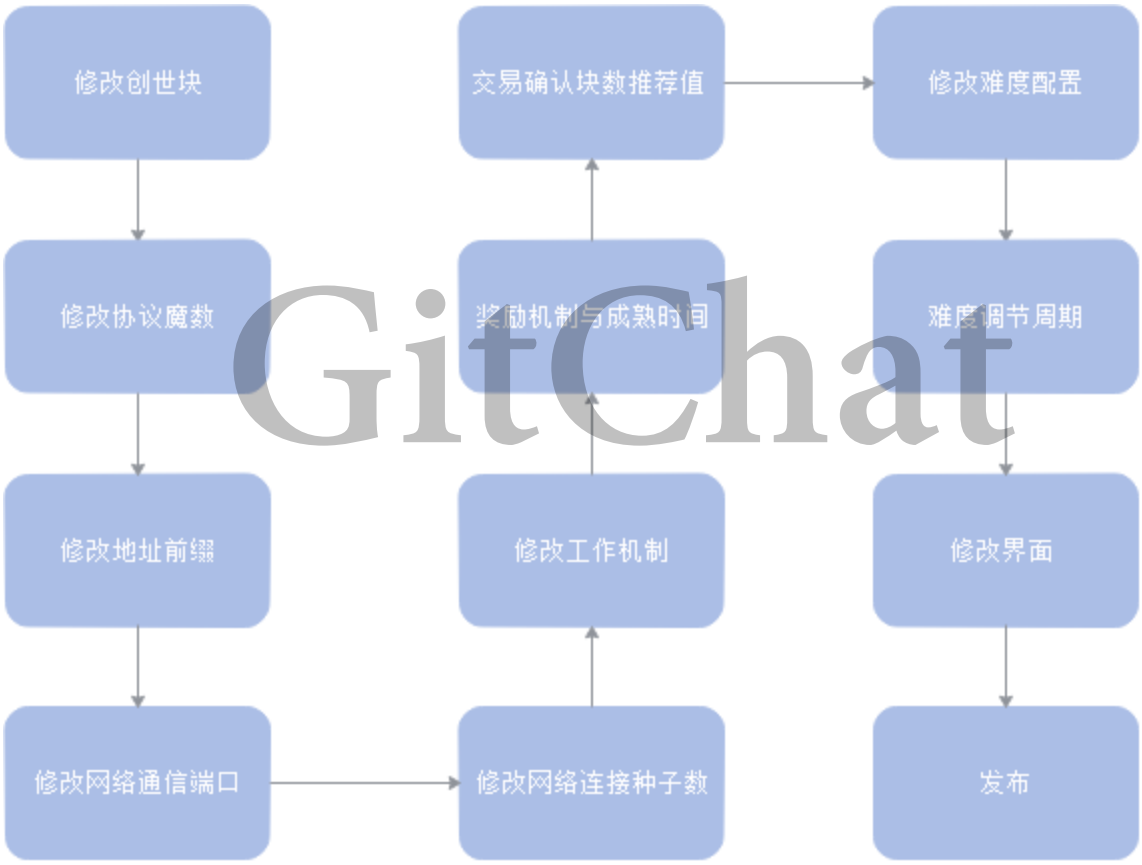
环境要求： Windows 10。

编译环境：使用交叉编译环境（因为 Windows 编译环境相对比较麻烦，所以我们提供 VMware 镜像包文件，用户可以通过镜像包进行一键导入使用。为了保证编译速度，建议使用4G或以上内存进行编译）。

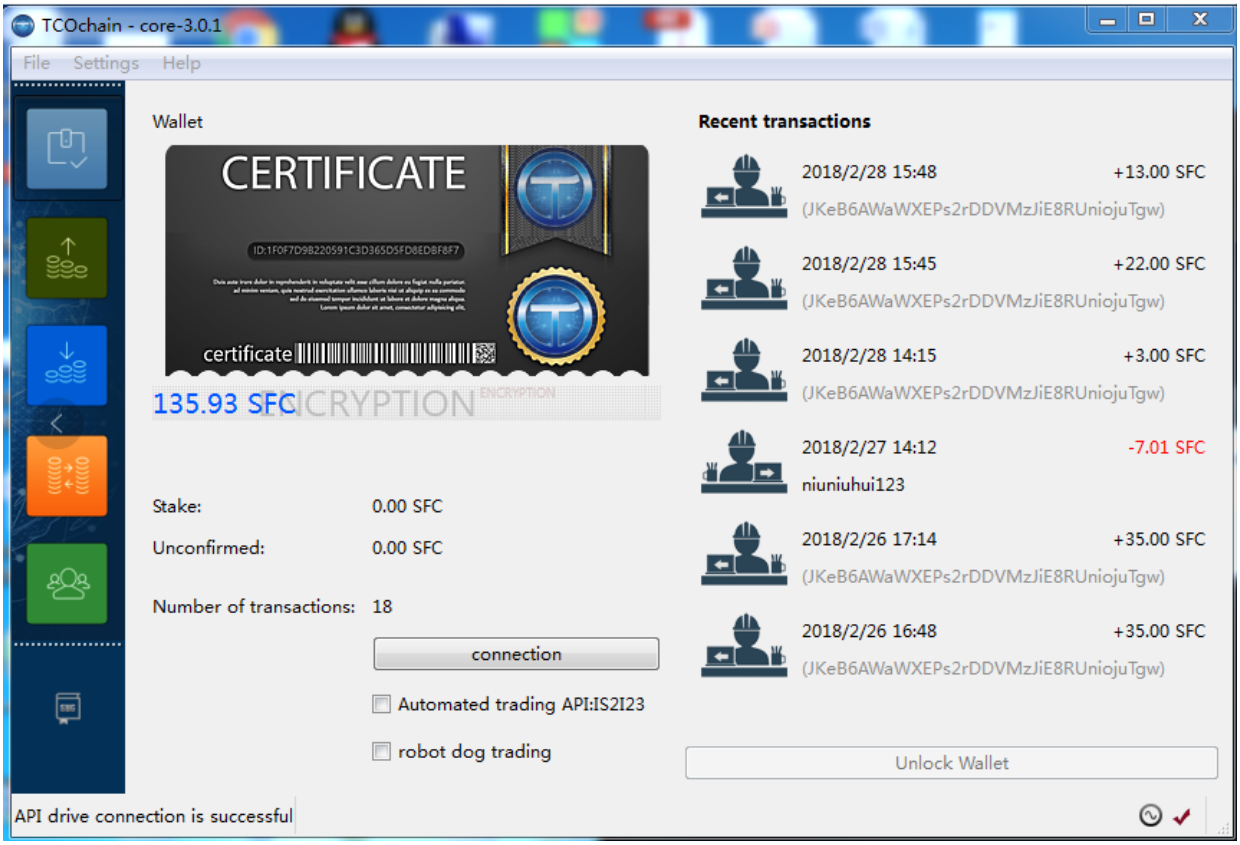
当前编译环境是最新版的，如要升级编译组件建议使用官网的 OpenSSL 与 QR Code 进行升级。

书写工具: Sublime。

制做流程如下图所示。



具体实例我们使用一个现有的项目进行解读，最终钱包图如下所示。



注：以上 TCO 项目为公网源码开放项目，与本教程没有任何关系，具体代码地址可以通过 GitHub 进行下载或是查看学习。

（1）创世块修改。

使用 Sublime 打开 SRC 目录下面的 chainparams.cpp 文件。从156行开始，具体代码如下。

```
  const char* pszTimestamp = "shanghai stock index closed at
2343.57, on 24th Sept., 2014";
        CMutableTransaction txNew;
        txNew.vin.resize(1);
        txNew.vout.resize(1);
        txNew.vin[0].scriptSig = CScript() << 0x1d00ffff <<
CScriptNum(4) << vector<unsigned char>((const unsigned
char*)pszTimestamp, (const unsigned char*)pszTimestamp +
strlen(pszTimestamp));
        txNew.vout[0].nValue = nGenesisSubsidy * COIN;
        txNew.vout[0].scriptPubKey = CScript() <<
ParseHex("049e02fa9aa3c19a3b112a58bab503c5caf797972f5cfe1006275aa
5485a01b48f9f648bc5380ee1e82dc6f474c8e0f7e2f6bbd0de9355f92496e3ea
327ccb19cc") << OP_CHECKSIG;
        genesis.vtx.push_back(txNew);
        genesis.hashPrevBlock.SetNull();
        genesis.hashMerkleRoot = genesis.BuildMerkleTree();
        genesis.nVersion = 1;
        genesis.nTime    = 1411666331;//创世时间
        genesis.nBits    = 0x1d00ffff;
        genesis.nNonce   = 2056985438;
```

```
        hashGenesisBlock = genesis.GetHash();
        assert(hashGenesisBlock ==
uint256S("0x0000000061b1aca334b059920fed7bace2336ea4d23d63428c7ae
e04da49e942"));
        assert(genesis.hashMerkleRoot ==
uint256S("0x7bf229f629a6666596c1ce57117c28d1d29299e8a5303347929bd
70847c49adb"));
```

第一步先将 genesis.nNonce、hashGenesisBlock、genesis.hashMerkleRoot 设为0，然后编译，编译生成 EXE 应用程序后，运行钱包文件后会提示程序错误，此时没有关系，我们打开 C 盘 APPDATA 里面进行查看要 Debug 文件，里面会记录信息，然后把以上三个值的信息以同样的格式写到 chainparams.cpp 文件里面。

（2）修改网络魔数。

网络魔数非常重要，直接关系到区块网络的安全性问题。

具体代码如下。

```
  CMainParams() {
        strNetworkID = "main";
        /**
         * The message start string is designed to be unlikely to
occur in normal data.
         * The characters are rarely used upper ASCII, not valid
as UTF-8, and produce
         * a large 4-byte int at any alignment.
         */
        pchMessageStart[0] = 0x90; //这个前缀内容可以进行修改
        pchMessageStart[1] = 0x0d;
        pchMessageStart[2] = 0xf0;
        pchMessageStart[3] = 0x0d;
        vAlertPubKey =
ParseHex("04e01590abdc5967eb550413fcf04bbd7cead46f13579b58d52ea2f
08d71a1a94196c476cd4fa60c30b51737fe3d9c8c88a04a6bec2282ebb1f22286
130a153b85");
        nDefaultPort = 9999;
        bnProofOfWorkLimit = ~arith_uint256(0) >> 8;
        nSubsidyHalvingInterval = 210000;
        nEnforceBlockUpgradeMajority = 750;
        nRejectBlockOutdatedMajority = 950;
        nToCheckBlockUpgradeMajority = 1000;
        nMinerThreads = 1; // 0 for all available cpus.
        nTargetTimespan = 60 * 60; // re-targeting every one hour
        nTargetSpacing = 1 * 60;  // do new pow every 1 minutes.
        nGenesisSubsidy = 100;
```

比如，比特币的魔数是 0xd9b48ef9 这个值，当用户发生交易或是连接时候，用户可以通过抓包软件进行分析查看当前网络报文前缀内容，所以这个值非常重要，一方面可以区分不同链之类的信息，另一方面可以方便调试人员进行数据抓包调试。

（3）修改前缀地址。

具体代码如下。

```
        base58Prefixes[PUBKEY_ADDRESS] =
boost::assign::list_of(35); // F prefix
        base58Prefixes[SCRIPT_ADDRESS] =
boost::assign::list_of(65); // T prefix
        base58Prefixes[SECRET_KEY] =
boost::assign::list_of(45); // 7 prefix
        base58Prefixes[EXT_PUBLIC_KEY] =
boost::assign::list_of(0x04)(0x88)(0xEE)(0x35);
        base58Prefixes[EXT_SECRET_KEY] =
boost::assign::list_of(0x04)(0x88)(0xEE)(0x45);
```

前缀地址指的是用户收款地址的前缀，一般建议进行修改，具体修改对照表。

用户可以访问这里进行对接转换。

（4）修改网络连接端口。

具体代码如下。

```
        nDefaultPort = 19999; //建议用户进行修改，
        bnProofOfWorkLimit = ~arith_uint256(0) >> 1;
        nEnforceBlockUpgradeMajority = 51;
        nRejectBlockOutdatedMajority = 75;
        nToCheckBlockUpgradeMajority = 100;
        nMinerThreads = 0;
        nTargetTimespan = 14 * 24 * 60 * 60; //! two weeks
        nTargetSpacing = 10 * 60;
```

（5）修改种子连接数。

打开 chainparams.cpp文件，从 240行开始，具体代码如下。

```
 //vSeeds.push_back(CDNSSeedData("alexykot.me", "testnet-
seed.alexykot.me"));
//vSeeds.push_back(CDNSSeedData("bitcoin.petertodd.org",
"testnet-seed.bitcoin.petertodd.org"));
//vSeeds.push_back(CDNSSeedData("bluematt.me", "testnet-
seed.bluematt.me"));
```

```
//vSeeds.push_back(CDNSSeedData("bitcoin.schildbach.de",
"testnet-seed.bitcoin.schildbach.de"));
```

用户可以把注解去掉。这个功能主要用于种子节点的，最好选择 VPS 或云主机，同时保证服务器必须是7X24小时开机的，建议使用二级域名方式。

（6）修改工作模式机制。

打开源码文件 miner.cpp，核心代码如下。

```
        while (true) {
            nNonce++;

            // Write the last 4 bytes of the block header (the nonce)
    to a copy of
            // the double-SHA256 state, and compute the result.
            //CHash256(hasher).Write((unsigned char*)&nNonce,
    4).Finalize((unsigned char*)phash);

            *phash = pblock->ComputePowHash(nNonce);

            // Return the nonce if the hash has at least some zero
    bits,
            // caller will check if it has enough to reach the target
            if (((uint16_t*)phash)[15] == 0)
                return true;

            // If nothing found after trying for a while, return -1
            if ((nNonce & 0xfff) == 0)
                return false;
        }
    }

    CBlockTemplate* CreateNewBlockWithKey(CReserveKey& reservekey)
    {
        CPubKey pubkey;
        if (!reservekey.GetReservedKey(pubkey))
            return NULL;

        CScript scriptPubKey = CScript() << ToByteVector(pubkey) <<
    OP_CHECKSIG;
        return CreateNewBlock(scriptPubKey);
    }
```


函数 BitcoinMiner() 里面创建新的 block，设定时间戳，获取 mempool 里面的最新交易，调用 ScanHash() 搜索随机数。

注意：独立挖矿的条件可以 chainparams.cpp 里面设置。

挖矿代码如下。

```
void static BitcoinMiner(CWallet *pwallet)
{
    LogPrintf("FastCoinMiner started\n");
    SetThreadPriority(THREAD_PRIORITY_LOWEST);
    RenameThread("fastcoin-miner");

    // Each thread has its own key and counter
    CReserveKey reservekey(pwallet);
    unsigned int nExtraNonce = 0;

    try {
        while (true) {
            if (Params().MiningRequiresPeers()) {
                // Busy-wait for the network to come online so we
don't waste time mining
                // on an obsolete chain. In regtest mode we
expect to fly solo.
                while (vNodes.empty())
                    MilliSleep(1000);
            }

            //
            // Create new block
            //
            unsigned int nTransactionsUpdatedLast =
mempool.GetTransactionsUpdated();
            CBlockIndex* pindexPrev = chainActive.Tip();

            auto_ptr<CBlockTemplate>
pblocktemplate(CreateNewBlockWithKey(reservekey));
            if (!pblocktemplate.get())
            {
                LogPrintf("Error in FastCoinMiner: Keypool ran
out, please call keypoolrefill before restarting the mining
thread\n");
                return;
            }
            CBlock *pblock = &pblocktemplate->block;
            IncrementExtraNonce(pblock, pindexPrev, nExtraNonce);

            LogPrintf("Running FastCoinMiner with %u transactions
in block (%u bytes)\n", pblock->vtx.size(),
                ::GetSerializeSize(*pblock, SER_NETWORK,
PROTOCOL_VERSION));

            //
            // Search
            //
            int64_t nStart = GetTime();
```

```
            arith_uint256 hashTarget =
arith_uint256().SetCompact(pblock->nBits);
            uint256 hash;
            uint32_t nNonce = 0;
            while (true) {
                // Check if something found
                if (ScanHash(pblock, nNonce, &hash))
                {
                    if (UintToArith256(hash) <= hashTarget)
                    {
                        // Found a solution
                        pblock->nNonce = nNonce;
                        assert(hash == pblock->GetHash());


SetThreadPriority(THREAD_PRIORITY_NORMAL);
                        LogPrintf("FastCoinMiner:\n");
                        LogPrintf("proof-of-work found  \n  hash:
%s  \ntarget: %s\n", hash.GetHex(), hashTarget.GetHex());
                        ProcessBlockFound(pblock, *pwallet,
reservekey);

SetThreadPriority(THREAD_PRIORITY_LOWEST);

                        // In regression test mode, stop mining
after a block is found.
                        if (Params().MineBlocksOnDemand())
                            throw boost::thread_interrupted();

                        break;
                    }
                }
```

代码预置的 Hash 算法（ sha256、 scrypt等），用户可以通过 block 的版本区分，那么在 scanhash 里面调用 ComputePowHash 的时候，可以选择不同的 POW 算法，甚至你可以把这些算法串联起来进行加密。

（ 7 ）修改奖励机制。

修改挖矿奖励的成熟时间，源代码请访问 main.h。

```
static const int COINBASE_MATURITY = 14;
```

修改交易确认块数推荐值，源代码请访问 qt/transactionrecord.h，核心代码如下。

```
class TransactionRecord
{
public:
```

```
        enum Type
        {
            Other,
            Generated,
            SendToAddress,
            SendToOther,
            RecvWithAddress,
            RecvFromOther,
            SendToSelf
        };

        /** Number of confirmation recommended for accepting a
    transaction */
        static const int RecommendedNumConfirmations = 2;
```

（8）修改难度配置

修改难度配置，请访问源文件 chainparams.cpp，核心代码如下。

```
    class CMainParams : public CChainParams {
    public:
        CMainParams() {
            networkID = CBaseChainParams::MAIN;
            strNetworkID = "main";

            pchMessageStart[0] = 0x90;
            pchMessageStart[1] = 0x0d;
            pchMessageStart[2] = 0xf0;
            pchMessageStart[3] = 0x0d;
            vAlertPubKey =
    ParseHex("04e01590abdc5967eb550413fcf04bbd7cead46f13579b58d52ea2f
    08d71a1a94196c476cd4fa60c30b51737fe3d9c8c88a04a6bec2282ebb1f22286
    130a153b85");
            nDefaultPort = 9999;
            bnProofOfWorkLimit = ~uint256(0) >> 32;
            nSubsidyHalvingInterval = 210000;
            nEnforceBlockUpgradeMajority = 750;
            nRejectBlockOutdatedMajority = 950;
            nToCheckBlockUpgradeMajority = 1000;
     nMinerThreads = 1; // 0 for all available cpus
            nTargetTimespan = 60 * 60; // re-targeting every one hour
            nTargetSpacing = 1 * 60;
            nGenesisSubsidy = 100;  ‘
```


下面解释下几处重要的代码。

A.bnProofOfWorkLimit=~uint256(0) >> 32;是一个大整数（256 bit）表示，前面32个位数是0，这个参数表示全网允许的最小难度，低于这个难度的 block 是不会被挖掘的。

B.nGenesisSubsidy = 100;表示初始津贴，比特币的第一个块的奖励是50个 BTC。

C.nSubsidyHavlingInterval = 210000; 这个参数决定了多少个 block 以后比特币的奖励（补贴，挖矿奖励）会减半。这个参数结合初始奖励（比如比特币50）基本可以估算全网总的货币产量（比如比特币的2100万），这个初始津贴也是可以配置的，如 B 所示。比如比特币，一个等比数列求和公式就可以计算货币总量 50(1/(1 - 0.5))*210000=2100万 BTC。

上面两个参数在查询区块奖励的时候用到，请查看 main.cpp 的 GetBlockValue(height, fees) 找到细节。

币总量，请查看 src\main.cpp 里面的951行处代码。

D.难度调节周期请见以下两处代码。

```
nTargetTimespan = 60 * 60; // re-targeting every one hour, 60
blocks
nTargetSpacing = 1 * 60;  // expect 1 block/minute
```

这里的设置是60分钟（3600秒，因为预期60秒钟1个 block，那么就是60个 blocks 后计算新难度，否则使用上一个 block 的难度），重新评估难度，下一个块的挖掘可能就要使用新的难度设置了。

（9）修改界面。

用户可以通过 QT 进行设计。主要的文件请访问 src\qt，语言包文件为 src\qt\locale。

接下来，我们看看 POS 与 POW 模式切换修改。

取消 POS 挖矿的币量代码如下。

```
int64 GetProofOfStakeReward(int64 nCoinAge, unsigned int nBits,
unsigned int nTime, int nHeight)
{    int64 nRewardCoinYear;
nRewardCoinYear = MAX_MINT_PROOF_OF_STAKE;
if(nHeight < YEARLY_BLOCKCOUNT)
nRewardCoinYear = 10 * MAX_MINT_PROOF_OF_STAKE;
else if(nHeight < (2 * YEARLY_BLOCKCOUNT))
nRewardCoinYear = 8 * MAX_MINT_PROOF_OF_STAKE;
else if(nHeight < (3 * YEARLY_BLOCKCOUNT))
nRewardCoinYear = 6 * MAX_MINT_PROOF_OF_STAKE;
else if(nHeight < (4 * YEARLY_BLOCKCOUNT))
nRewardCoinYear = 4 * MAX_MINT_PROOF_OF_STAKE;
else if(nHeight < (5 * YEARLY_BLOCKCOUNT))
nRewardCoinYear = 2 * MAX_MINT_PROOF_OF_STAKE;
int64 nSubsidy = nCoinAge * nRewardCoinYear / 365;
if (fDebug && GetBoolArg("-printcreation"))
printf("GetProofOfStakeReward(): create=%s nCoinAge=%"PRI64d"
```

```
nBits=%d\n", FormatMoney(nSubsidy).c_str(), nCoinAge, nBits);
  return nSubsidy;}
```

修改成为：

```
int64 GetProofOfStakeReward(int64 nCoinAge, unsigned int nBits,
unsigned int nTime, int nHeight)
{    int64 nSubsidy = 0;
if (fDebug && GetBoolArg("-printcreation"))
printf("GetProofOfStakeReward(): create=%s nCoinAge=%"PRI64d"
nBits=%d\n", FormatMoney(nSubsidy).c_str(), nCoinAge, nBits);
  return nSubsidy;}
```

那就意味着 POS 利息变成0，如果还想使用 POW+POS 这些代码可以不用修改

> 特别声明：本教程通过技术视角进行讲解与说明，任何个人与组织不得通过教程内容进行一些金融或是发布 COIN 等活动。
>
> 注：投资有风险入市需谨慎。没有应用的区块链项目都是高风险的行为。