比特币源码研读系列4 BIP141隔离见证SegWit

在介绍挖矿之前，我们先来介绍一个重要的更新，也是BIP141，BIP是Bitcoin Improvement Proposals的简写，指的是bitcoin的一些升级意见。

在比特币中，升级也称之为fork，中文叫分叉，分叉有两种：softfork，软分叉，表示向后兼容；hardfork，硬分叉，不向后兼容。Bitcoin Cash比特币黄金就是硬分叉的结果。

本篇我们介绍一个非常重要的升级建议BIP141：Segragated Witness，中文称之为隔离见证。我们先来回忆下tx的inputs是一个引用tx的数组，每个被引用的tx会保留下面这些数据：

|  |
| --- |
| **CTxIn** |
| COutPoint prevout; |
| CScript scriptSig; |
| uint32\_t nSequence; |
| CScriptWitness scriptWitness |

在non-SegWit的tx中，prevout大概占用32+4=36B，scriptSig占用1B，nSequence占用4B，scriptWitness占用107B，因此一个vin通常会占用32+4+1+107+4=148B。

可以看出来scriptWitness占了大部分数据，它是个什么数据呢？主要是付款人的签名授权数据，表示该vin一定是本人签名spend的，就像国内信用卡消费时的签字一样。BIP141的SegWit升级考虑的是就是把scriptWitness数据单独剥离出来，在vin中只保留引用的指针信息，因此就可以减少tx的大小，通常一个SegWit的vin会占用

32 + 4 + 1 + 107/4 + 4字节。

下面我们再来看看比特币是如何判断是否升级成功的。

**IsWitnessEnabled**方法。该方法用于判断当前是否已启用SegWit功能

*bool IsWitnessEnabled(const CBlockIndex\* pindexPrev, const Consensus::Params& params)*

*{*

*LOCK(cs\_main);*

*return (VersionBitsState(pindexPrev, params, Consensus::DEPLOYMENT\_SEGWIT, versionbitscache) == THRESHOLD\_ACTIVE);****//SEGWIT更新是否是Active活跃状态***

*}*

*ThresholdState AbstractThresholdConditionChecker::GetStateFor(const CBlockIndex\* pindexPrev, const Consensus::Params& params, ThresholdConditionCache& cache) const*

*{*

***//通常一个升级会有写上升级起始，截止时间，以及如何判断本次升级是否成功的标准。nPeroids表示的校验周期，一般是2016个block。***

***nThreshold表示是要升级成功必须达到的限定值，比特币要求升级成功必须达到95%的认可，因此nThreshold=0.95\*2016=1916.***

***nTimeStart本次升级的起始时间，例如nStartTime = 1479168000表示从日期20161115开始***

***nTimeTimeout本次升级的结束时间。例如nTimeout = 1510704000表示到日期20171115结束。***

*int nPeriod = Period(params);*

*int nThreshold = Threshold(params);*

*int64\_t nTimeStart = BeginTime(params);*

*int64\_t nTimeTimeout = EndTime(params);*

*// A block's state is always the same as that of the first of its period, so it is computed based on a pindexPrev whose height equals a multiple of nPeriod - 1.*

*if (pindexPrev != nullptr) {*

*pindexPrev = pindexPrev->GetAncestor(pindexPrev->nHeight - ((pindexPrev->nHeight + 1) % nPeriod));****//假设当前主链高度=2016\*n+m，则pindexPrev=2016\*n-1。意思是2016个block为一个周期，pindexPrev指向当前周期的起点。***

*}*

*// Walk backwards in steps of nPeriod to find a pindexPrev whose information is known*

*std::vector<const CBlockIndex\*> vToCompute;*

***//下面这个循环用于将当前周期的block都放到vToCompute数组。***

*while (cache.count(pindexPrev) == 0) {*

*if (pindexPrev == nullptr) {*

*// The genesis block is by definition defined.*

*cache[pindexPrev] = THRESHOLD\_DEFINED;*

*break;*

*}*

*if (pindexPrev->GetMedianTimePast() < nTimeStart) {****//未到升级的起始日期***

*// Optimization: don't recompute down further, as we know every earlier block will be before the start time*

*cache[pindexPrev] = THRESHOLD\_DEFINED;*

*break;*

*}*

*vToCompute.push\_back(pindexPrev);*

*pindexPrev = pindexPrev->GetAncestor(pindexPrev->nHeight - nPeriod);*

*}*

*// At this point, cache[pindexPrev] is known*

*assert(cache.count(pindexPrev));*

*ThresholdState state = cache[pindexPrev];*

*// Now walk forward and compute the state of descendants of pindexPrev*

*while (!vToCompute.empty()) {****//计算vToCompute数组有多少个block是更新了该功能的block，超过nThreshold阀值则说明本次更新成功。***

*ThresholdState stateNext = state;*

*pindexPrev = vToCompute.back();*

*vToCompute.pop\_back();*

*switch (state) {*

*case THRESHOLD\_DEFINED: {*

*if (pindexPrev->GetMedianTimePast() >= nTimeTimeout) {*

*stateNext = THRESHOLD\_FAILED;*

*} else if (pindexPrev->GetMedianTimePast() >= nTimeStart) {*

*stateNext = THRESHOLD\_STARTED;*

*}*

*break;*

*}*

*case THRESHOLD\_STARTED: {*

*if (pindexPrev->GetMedianTimePast() >= nTimeTimeout) {*

*stateNext = THRESHOLD\_FAILED;*

*break;*

*}*

*// We need to count*

*const CBlockIndex\* pindexCount = pindexPrev;*

*int count = 0;*

*for (int i = 0; i < nPeriod; i++) {*

*if (Condition(pindexCount, params)) {*

*count++;*

*}*

*pindexCount = pindexCount->pprev;*

*}*

*if (count >= nThreshold) {*

*stateNext = THRESHOLD\_LOCKED\_IN;*

*}*

*break;*

*}*

*case THRESHOLD\_LOCKED\_IN: {*

*// Always progresses into ACTIVE.*

*stateNext = THRESHOLD\_ACTIVE;*

*break;*

*}*

*case THRESHOLD\_FAILED:*

*case THRESHOLD\_ACTIVE: {*

*// Nothing happens, these are terminal states.*

*break;*

*}*

*}*

*cache[pindexPrev] = state = stateNext;*

*}*

*return state;*

*}*

到这里我们差不多可以看到，通过检测当前周期产生的block是否满足95%的通过率来判断当前是否升级成功。

好了，本篇我们就介绍到这里。