# BP选举和切换

## 注册成为bp和用户投票

先看数据结构。在eosio.system.hpp里，有定义两个表格，

* struct producer\_info {

account\_name owner;

double total\_votes = 0;

eosio::public\_key producer\_key; /// a packed public key object

bool is\_active = true;

std::string url;

uint32\_t unpaid\_blocks = 0;

uint64\_t last\_claim\_time = 0;

uint16\_t location = 0;

}这是存储producer信息的。

* struct voter\_info {

account\_name owner = 0; /// the voter

account\_name proxy = 0; /// the proxy set by the voter, if any

std::vector<account\_name> producers;

int64\_t staked = 0;

double last\_vote\_weight = 0; /// the vote weight cast the last time the vote was updated

double proxied\_vote\_weight= 0; /// the total vote weight delegated to this voter as a proxy

bool is\_proxy = 0; /// whether the voter is a proxy for others

uint32\_t reserved1 = 0;

time reserved2 = 0;

eosio::asset reserved3;

}这是存储每个账号的投票信息的。

Eosio账号有部署regproducer这个action，调用就可以注册为producer，代码在\contracts\eosio.system\voting.cpp

用户投票是eosio账号的voteproducer接口，直接调用就可以为选中的producer投票。

如上操作会更新上面两个表格的数据。其中voteproducer会修改\_producer表格的total\_votes数值。

给producer投票时票是有权重的，网上描述很多了。

## BP切换

先在整体上看这个过程：

用户选举等改变BP投票排名->每1min会检查一次是否需要修改BP->如果需要，在controller里修改gpo的proposed\_schedule\_block\_num（记录新BP被提名的block num）和 新的BP列表（注意这没有体现在block header上，只是内部数据库变化）。包含该修改的block变为不可逆（约336个block）后，会将提名变为pending，修改区块头中的pending\_schedule\_lib\_num（注意此时才会修改block header），且执行set\_new\_producers，此时会在nodeos的log中看到切换BP，db中也会重置gpo的proposed\_schedule\_block\_num。当该block变为不可逆（又是336个block）后，active\_schedule会被pending schedule替换，整个的BP更换过程完成。

涉及的数据：

Controller.cpp里db->gpo->proposed\_schedule\_block\_num和proposed\_schedule，记录提名过程

Block\_header\_state->pending\_schedule\_lib\_num，在上面的提名block变为不可逆后，修改该header里的数据。

详细：

如chain\_plugin的分析中所说，当controller->start\_block被执行的时候，也就是要将一个block更新到本地状态的时候，会往block里插入一个trx,其action是eosio::system\_contract::onblock,代码在contracts\eosio.system\producer\_pay.cpp。这个action只能eosio调用。它的作用是：

1. 如果系统中抵押的EOS小于总数的15%，返回
2. 对生产该block的producer发工资（不是转账，只是把unpaid\_blocks++做记录）
3. 如果当前block的timestamp比上次的producer更换的时间大于1min，调用update\_elected\_producers看是否需要更新bp
4. 更新namebid相关状态。

这里重点看update\_elected\_producers。它在voting.cpp里。

1. 它把last\_producer\_schedule\_update更新为当前块的timestamp，其实这就是个计时器，控制着系统每1min调用一次update\_elected\_producers。
2. 从\_producers表里取出得票最多的21个producer，按名字排序，调用set\_proposed\_producers看是否需要修改bp。

set\_proposed\_producers有两个实现，一个在wasm\_interface.cpp，一个在controller.cpp里。此处实际调用wasm\_interface里的，只是做些检查，去重等，然后继续调用controller里的。

此处会做一些检查，如是否跟上一个提名一样等。然后修改db里的proposed\_schedule\_block\_num（为当前block）和proposed\_schedule。

在每个block的start\_block时，会做一些判断，这里主要是：当前有提名的bp改变，且proposed\_schedule\_block\_num已经成为不可逆，且没有pending的bp变化，则调用block\_header\_state. set\_new\_producers把bp修改写入pending，修改pending\_schedule\_lib\_num。该过程会修改block\_header，让新bp出现在block\_header里。并且把提名bp时修改的gpo->proposed\_schedule\_block\_num和proposed\_schedule重置为空。

再过336个block，上述第一个写block\_header的块变为不可逆后，active\_schedule会被pending schedule替换，但这个过程我在代码上还没找到。

# 出块过程以及如何变为不可逆

尚未研究producer\_plugin,等研究后再分析