## History Plugin描述

History plugin大家都不陌生，它有什么具体功能？如何设计的呢？

EOS上的action们都有个global seq num，记录该action在EOS历史上是第几笔action。这个可以作为action记录时的索引。

History plugin有主要有4个表格：

1. Action\_history\_obj，主表，记录所有的action.  
   包括global seq num，action内容，block\_num, block\_time,trx\_id
2. Account\_history\_obj，记录action和account之间的对应关系，  
   action的global seq num, 对应的account(sender或者receiver，会被分别记录)，该action在该account下的序号（每个account有自己的，从第一个到最近的）。  
   这个表格主要是把action和account对应起来，这样就能按account查找action了。
3. Public\_key\_history\_obj,记录pubkey和account之间的对应关系
4. Account\_control\_history\_obj，记录授权控制关系

History plugin的内容在执行chain\_plugin执行push\_trx时更新。History plugin注册了applied\_transaction的监听，当chain\_plugin中执行push\_trx完成，调用emit时会把action的信息发送过来，触发on\_applied\_transaction的执行。此时会触发对上面4个表格的更新。具体流程不复杂。

History\_plugin对外提供几个接口：

* Get\_actions
* Get\_transaction
* Get\_block\_detail
* Get\_key\_accounts(从pub key查找account)
* Get\_controlled\_accounts

## 改进目标

History\_plugin运行过程中需要耗费大量内存，对节点们的资源造成巨大消耗。大部分节点无法承担运行full history plugin的成本。因此期望提供方法降低节点运行history\_plugin的成本，如只保存最近N天的history

在config.ini里添加一个设置项，设置多久之前的history\_plugin的数据可以自动清除掉。就像这样：

History\_plugin\_retain\_time = 3 #unit: days

该设置在下文中记录为“clear\_span”

## 影响范围

Action\_history\_obj, Account\_history\_obj是需要清除的，Public\_key\_history\_obj, Account\_control\_history\_obj不能被清除

## 应用情景和实现方法

在实际使用中，主要有2个情景：

1. 新节点同步数据，赶上当前出块；或执行replay  
    history\_plugin的写入发生在执行push\_transaction时，当执行成功时，chain\_plugin会emit该trx到history\_plugin，调起history\_plugin.on\_applied\_transaction，从而将该trx内的actions插入到history\_plugin。  
    此情景下需要根据当前时间，计算出需要开始保存history的时间，然后该时间之前的1,2表格内容不记录，直接跳过。但是需注意3,4表格的数据还是需要记录的，所以不能直接把整个action跳过，而是只跳过记录1,2表格的部分。否则get\_key\_accounts和get\_controlled\_account两个操作会出问题。

另外同步块/replay可能会持续几天，所以可能导致在其操作完成后，history\_plugin的实际保存数据量超过预期的。如预期只保存最近3天的，但是同步过程持续了2天，最终可能会导致history\_plugin里实际保存了5天的数据。这个问题应该是难以避免的，但是在第二个情景中，会将其修正，因此问题不大

1. 节点已经同步到最新区块，正常运行中  
    此时需要开启一个timer，每隔clear\_span就执行一次，执行时计算出哪个时间点之前的history需要被清除，记为x-time。该x-time应该从当前的不可逆块起计算。然后根据当前最新的不可逆块block\_num，减去x-time时长可能产生的块个数 ，得到预计的需要清除哪个块之前的数据，然后根据chain.get\_block得到该block的block\_time，与x-time对比，可能需要前后调整从而找到确切的Block\_num.(如果精度要求不高，这一步可以省略，直接使用计算出来的block\_num)。使用类似history\_plugin.get\_block\_detail的方法，可以得到1,2表格里的该block对应的action\_sequence\_num（即global的action序号），遍历1,2表格，删除小于该序号的所有记录即可。
2. 1,2表格里有action\_sequence\_num，这个表示从创世开始的每个action的序号，这个序号从1开始，每个action就加1  
   还有account\_sequence\_num，这个对每个account单独记录，每个account的都从1开始，每执行一个action就加1.  
   上述1,2的操作中，会导致上述两个sequence\_num较早的数据丢失，而且在1中由于是直接跳过，所以实际开始记录时开始的数还是1，而2中由于是删除，所以实际记录开始时记录可能不是1。即2的过程，在查询时有可能可以提供删了几条history，而1不能。但由于get\_actions等操作都是用pos和offset操作的，所以这个序号差别对用户来说感受不太到。

另外在get\_actions等操作时，对于取数据的start和end的操作，之前默认是全部数据都在的，现在有些数据被删了，所以该部分操作会变复杂，要注意控制。

## 影响

在get\_action,get\_transaction，get\_block\_detail等需要history\_plugin的操作时，由于只保存最近几天的数据，所以返回结果与用户预期的有差别。而且可能无法告诉用户有几条操作被忽略了。  
get\_keys\_account和get\_controlled\_accounts不会被影响。