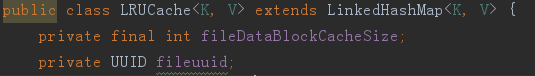
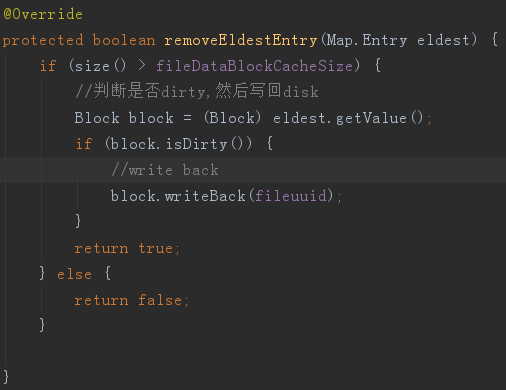
My design of cache

* Cache设计：

cache放于每一个SDFSFileChannel中，于特定的文件请求相绑定，并且实现了LRU的替换策略。

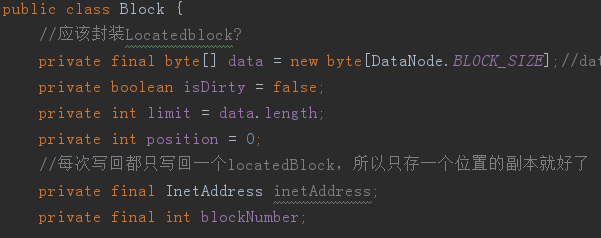


Cache中有cache的block数量的属性fileDataBlockCacheSize，还有对应文件请求的UUID的属性fileuuid。实现的LRUCache是继承了LinkedHashMap，特点为，以LocatedBlock的hashcode为key值，Block类为value值（Block实现后文介绍）；并且可以实现键值对超过规定数量后自动清除时间最远使用的键值对的功能，也就是可以方便的实现LRU替换策略，对应的LRU代码如下：



* Cache实现中的数据块Block的实现：

Cache中的键值对的value是Block对象，其实现为：



由于每次替换只需要写回到一个副本中，所以只存放对应locatedBlock的地址属性，并且有一个位置标记是否为脏；另外为了读写操作，还会有limit和position属性，与blockbuffer中的属性对应。

属性设计：

* data：初始化的时候就设置其大小为DataNode的blockSize的大小，但是block实际代表的数据大小由limit控制。
* isDirty :当改变这个block的内容的时候，就会设置脏位，当channel关闭或者cache替换这个块出去的时候，就会看是否为脏。
* limit：才是真正的block的数据大小，，每次写的时候都会改变，读的时候会以limit为界限防止读过多的零，这样就不需要每次写入更多的数据的时候再动态申请内存大小，影响效率。
* position：游标位置。读写操作的时候，position指向正在读或写的位置，并以position来实现append和overwrite，每次写的时候，都会在position的位置后面去覆盖数据，所以只要设定好游标的位置，就可以实现append操作。

Block实现的主要方法有：

* **int write(ByteBuffer src)**：将src中的数据一位一位写入block的数据属性中，直到src末尾或者写满整个block为止，返回写入的byte数量，position相应的会移动；
* **int read(ByteBuffer dst)**：将block的数据写入dst中，直到dst满了或者数据写完为止，返回写入的byte数量；
* **void writeBack(UUID fileuuid)**：将本block的内容写回dataNode的存储中。应用于替换策略时，或者channel关闭时写回所有脏的数据块。

Describe the problem I met during developing and how you solve it.

1. UUID的使用在文件系统中所起的作用：

UUID标识着一次文件请求，该请求时长从open开始一直到close结束。一开始本以为是和文件名称一样的作用，实现到最后才发现一次请求可以与文件名称无关，于是在NameNode中，实现了对于权限的检验，将UUID和fileNode一起放于hashmap里面;

final Map<UUID, FileNode> readonlyFile = new HashMap<>();

final Map<UUID, FileNode> readwritePFile = new HashMap<>();

1. 利用ByteBuffer对数据进行读写，以此为启发实现缓存数据块Block的读写实现：

本以为ByteBuffer是一个简单数组的包装类，然而发现其功能很多，并且不是简单的包装，不是随便写写代码就可以用好的，在不求甚解乱写了代码之后，终于详细的考察其文档，发现ByteBuffer的posiiton非常方便的可以用来读写。所以用来实现文件系统的append和overwrite就可以用position来方便的实现，不需要每次读写都计算出应该从哪里开始，并且，每个数据块都是固定大小的，那么当一个文件的最后一个块没有填满的话，可以用limit这个属性来限制每次写入的时候的position移动的范围，这也是从ByteBuffer中给出的启示。

1. 缓存与实际存储上的差异：

每次打开文件，会将文件的元信息存储在channel中，那么问题来了，如果增加了一个locatedblock后，文件元信息就会改变，是在channel中改变了，还是马上写回NameNode中，我的实现是，同时去做。因为分配locatedBlock这件事情本来就需要和NameNode交互，那么为了保证channel里面缓存信息的有效性，不如同时都改了。但是文件大小不是立即在nameNode中立刻更改的，而是channel关闭时，与NameNode交互然后写回数据大小。

同样，dataNode的数据就比较统一，在Channel里面改变了，在最后close时或者替换出去时才做最后的写回。

1. 利用测试用例进行需求的明确和系统的完善：

整个文件系统实现下来会非常的享受，原因在于从对系统结构的不明确到基本完全理解，只能在一遍遍手撕代码的过程中获得。但是，整个文件系统还是要明确很多细节上的实现，不能只停留在对于整个系统的把握上，所以与其检查代码的逻辑，不如直接写几个测试用例来的直截了当。所以在SimpleDistributedFileSystem的main函数的注释里面，写了一些测试用例，并打印了系统的状态，才测试中又发现自己对于Block的大小和游标的细节没有完善清楚，导致读出来的结果会有偏差。