**缓存算法知识总结**

参考应引用的文献网址：

<http://blog.csdn.net/constant_zyh188/article/details/73350576>

<http://www.cnblogs.com/work115/p/5585341.html>

<http://blog.jobbole.com/30940/>

**一、缓存概念**

**1、什么是缓存？**

凡是位于速度相差较大的两种硬件之间，用于协调两者数据传输速度差异的结构，均可称之为缓存（Cache）。因为内存相对于硬盘读写速度更快，内存可以作为硬盘的缓存；同样的，硬盘读写速度远高于网络数据的读写速度，也可以将硬盘作为网络数据的缓存。在内存和硬盘之间，硬盘与网络之间，都存在某种意义上的Cache。

表现上，缓存载体与被缓存载体总是相对的，缓存设备成本高于被缓存设备，缓存设备速度高于被缓存设备，缓存设备容量远远小于被缓存设备。

缓存可以认为是数据的池子，是存储频繁使用的数据的临时的地方，缓存可以认为是原始数据的子集，它是从原始数据里复制出来的，并且为了能被取回，被加上了**标志**。

**2、命中与回源**

当用户发起一个请求，我们的应用接受这个请求，并且如果是在第一次检查缓存的时候，需要去数据库读取产品信息。如果在缓存中，一个条目通过一个标记被找到了，这个条目就会被使用，我们就叫它缓存命中。

如果没有命中缓存，就需要从原始地址获取，这个步骤叫做“回源”。回源的代价是高昂的，只有尽可能减少回源才能更好的发挥缓存的作用，但受限于缓存设备的成本，不能仅仅增加缓存容量，只能在成本和回源率之间寻求一个平衡点。

**3、缓存未命中(Cache Miss)**

需要注意两点：

* 如果还有缓存的空间，那么，没有命中的对象会被存储到缓存中来。
* 如果缓存满了，而又没有命中缓存，那么就会按照某一种策略，把缓存中的旧对象踢出，而把新的对象加入缓存池。而这些策略统称为替代策略（缓存算法），这些策略会决定到底应该提出哪些对象。

**4、存储成本**

当没有命中时，我们会从数据库取出数据，然后放入缓存。而把这个数据放入缓存所需要的时间和空间，就是存储成本。

**5、失效**

当存在缓存中的数据需要更新时，就意味着缓存中的这个数据失效了。

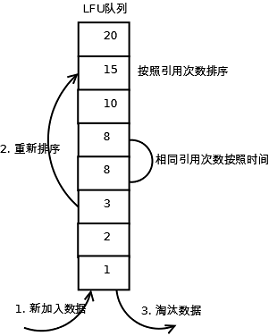
**二、缓存算法原理**

介于缓存只能够使用有限的内存，因此任何Cache系统都需要一个如何淘汰缓存的方案（缓存淘汰算法，等同于页面置换算法）。我们要根据自己的业务需要来选择使用何种算法来淘汰多余的数据，提高命中率。

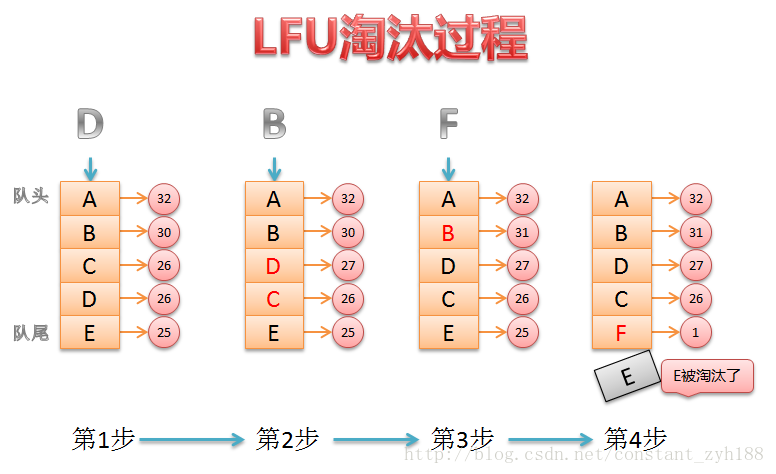
**1、LFU（Least Frequently Used）**

又称“最不经常使用算法”，是根据数据的历史访问频率来淘汰数据，其核心思想是“如果数据过去被访问多次，那么将来被访问的频率也更高”。即使用一个计数器来记录条目被访问的频率。通过使用LFU缓存算法，最低访问数的条目首先被移除。这个方法并不经常使用，因为它无法对一个拥有最初高访问率之后长时间没有被访问的条目缓存负责。

**实现：**LFU的每个数据块都有一个引用计数，所有数据块按照引用计数排序，具有相同引用计数的数据块则**按照时间排序**。



* 1、新加入数据插入到队列尾部（因为引用计数为1）
* 2、队列中的数据被访问后，引用计数增加，队列重新排序
* 3、当需要淘汰数据时，将已经排序的列表最后的数据块删除

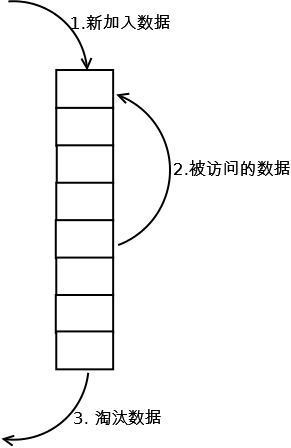


* 1、假设我们的lfu最大的存储空间控制为5个，此时访问D，D现在的访问频率计数是26；
* 2、访问D后，D的频率+1，也就是27了。此时需要调整缓存池数据需要重新排序，D和C交换；
* 3、访问B，B的频率+1，由于A的频率仍然比B大，所以不需要调整；
* 4、当新数据F插入缓存池之前，由于已经空间满了，需要干掉一个！因为E的频率最低，故淘汰E，将F插入缓存池，缓存池重新排序，F放到队尾。

**2、LRU（Least Recently Used）**

又称“最近最少使用算法”，是根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。即将最近使用的条目存放到靠近缓存顶部的位置。当一个新条目被访问时，LRU将它放置到缓存的顶部。当缓存达到极限时，较早之前访问的条目将从缓存底部开始被移除。这里会使用到昂贵的算法，而且它需要记录“年龄位”来精确显示条目是何时被访问的。此外，当一个LRU缓存算法删除某个条目后，“年龄位”将随其他条目发生改变。

最常见的实现是使用一个**链表保存缓存数据**，详细算法实现如下：



* 1、新数据插入到链表头部；
* 2、每当缓存命中（即缓存数据被访问），则将数据移到链表头部；
* 3、当链表满的时候，将链表尾部的数据丢弃。

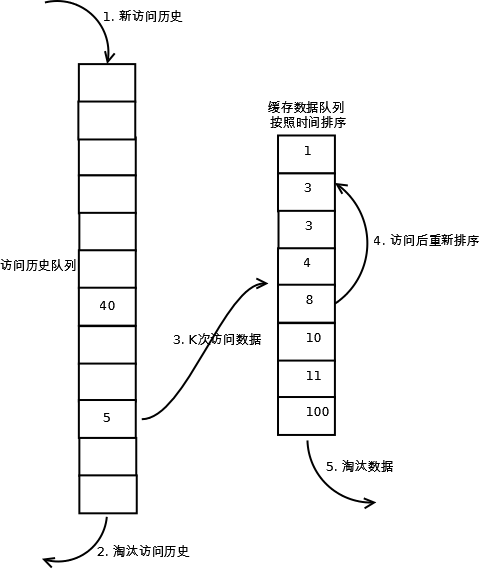
**分析：**

* **命中率**：当存在热点数据时，LRU的效率很好，但偶发性的、周期性的批量操作会导致LRU命中率急剧下降，缓存污染情况比较严重。
* **复杂度**：实现简单
* **代价**：命中时需要遍历链表，找到命中的数据块索引，然后需要将数据移到头部

**3、LRU-K（Least Recently Used K）**

LRU-K中的K代表最近使用的次数，因此LRU可以认为是LRU-1。LRU-K的主要目的是为了解决LRU算法“缓存污染”的问题，其核心思想是将“最近使用过1次”的判断标准扩展为“最近使用过K次”。

相比LRU，LRU-K需要多维护一个队列，用于记录所有缓存数据被访问的历史。只有当数据的访问次数达到K次的时候，才将数据放入缓存。当需要淘汰数据时，LRU-K会淘汰第K次访问时间距当前时间最大的数据。详细实现如下：



* 1、数据第一次被访问，加入到访问历史列表；
* 2、如果数据在访问历史列表里后没有达到K次访问，则按照一定规则（FIFO，LRU）淘汰；
* 3、当访问历史队列中的数据访问次数达到K次后，将数据索引从历史队列删除，将数据移到缓存队列中，并缓存此数据，**缓存队列重新按照时间排序**；
* 4、缓存数据队列中被再次访问后，重新排序；
* 5、需要淘汰数据时，淘汰缓存队列中排在末尾的数据，即：淘汰“倒数第K次访问离现在最久”的数据。

LRU-K具有LRU的优点，同时能够避免LRU的缺点，实际应用中LRU-2是综合各种因素后最优的选择，LRU-3或者更大的K值命中率会高，但适应性差，需要大量的数据访问才能将历史访问记录清除掉。

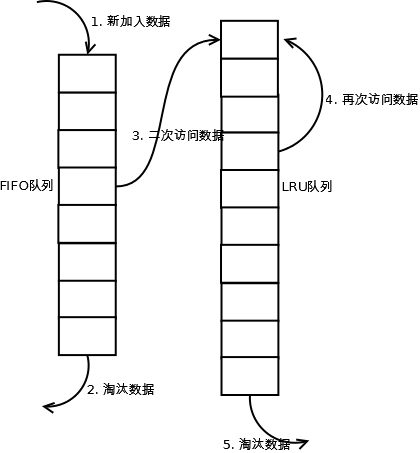
**分析：**

* **命中率**：LRU-K降低了“缓存污染”带来的问题，命中率比LRU要高
* **复杂度**：LRU-K队列是一个优先级队列，算法复杂度和代价比较高
* **代价：**由于LRU-K还需要记录那些被访问过、但还没有放入缓存的对象，因此内存消耗会比LRU要多；当数据量很大的时候，内存消耗会比较可观。LRU-K需要基于时间进行排序（可以需要淘汰时再排序，也可以即时排序），CPU消耗比LRU要高。

**4、2Q（Two queues）**

Two queues（以下使用2Q代替）算法类似于LRU-2，不同点在于2Q将LRU-2算法中的访问历史队列（注意这不是缓存数据的）改为一个FIFO缓存队列，即：2Q算法有两个缓存队列，一个是FIFO队列，一个是LRU队列。

当数据第一次访问时，2Q算法将数据缓存在FIFO队列里面，当数据第二次被访问时，则将数据从FIFO队列移到LRU队列里面，两个队列各自按照自己的方法淘汰数据。详细实现如下：



* 1、新访问的数据插入到FIFO队列；
* 2、如果数据在FIFO队列中一直没有被再次访问，则最终按照FIFO规则淘汰；
* 3、如果数据在FIFO队列中被再次访问，则将数据移到LRU队列头部；
* 4、如果数据在LRU队列再次被访问，则将数据移到LRU队列头部；
* 5、LRU队列淘汰末尾的数据。
* **注：**上图中FIFO队列比LRU队列短，但并不代表这是算法要求，实际应用中两者比例没有硬性规定

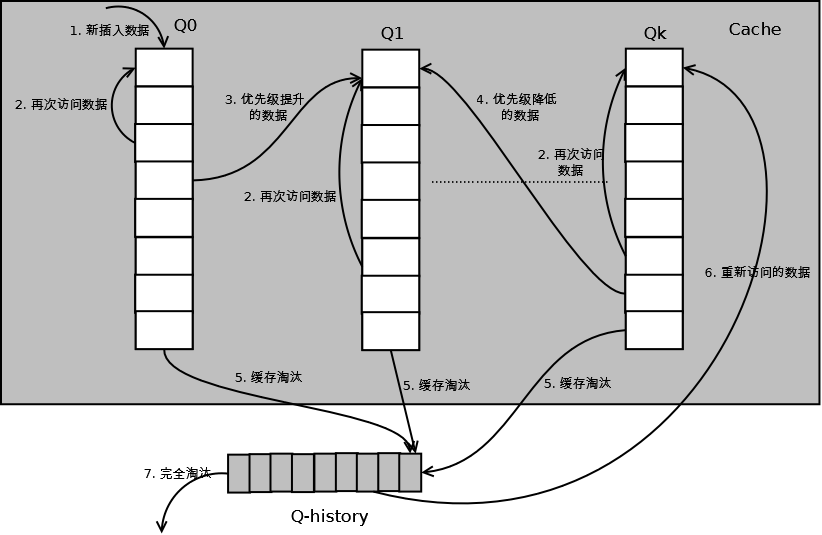
**分析：**

* **命中率**：2Q算法的命中率要高于LRU
* **复杂度**：需要两个队列，但两个队列本身都比较简单
* **代价：**FIFO和LRU的代价之和。2Q算法和LRU-2算法命中率类似，内存消耗也比较接近，但对于最后缓存的数据来说，2Q会减少一次从原始存储读取数据或者计算数据的操作。

**5、MQ（Multi Queue）**

MQ算法根据访问频率将数据划分为多个队列，不同的队列具有不同的访问优先级，其核心思想是：**优先缓存访问次数多的数据**。

MQ算法将缓存划分为多个LRU队列，每个队列对应不同的访问优先级。访问优先级是根据访问次数计算出来的，例如：详细的算法结构图如下，Q0，Q1....Qk代表不同的优先级队列，Q-history代表从缓存中淘汰数据，但记录了数据的索引和引用次数的队列：



如上图，算法详细描述如下：

* 1、新插入的数据放入Q0；
* 2、每个队列按照LRU管理数据；
* 3、当数据的访问次数达到一定次数，需要提升优先级时，将数据从当前队列删除，加入到高一级队列的头部；
* 4、为了防止高优先级数据永远不被淘汰，当数据在指定的时间里访问没有被访问时，需要降低优先级，将数据从当前队列删除，加入到低一级的队列头部；
* 5、需要淘汰数据时，从最低一级队列开始按照LRU淘汰；每个队列淘汰数据时，将数据从缓存中删除，将数据索引加入Q-history头部；
* 6、如果数据在Q-history中被重新访问，则重新计算其优先级，移到目标队列的头部；
* 7、Q-history按照LRU淘汰数据的索引。

**分析：**

* **命中率**：MQ降低了“缓存污染”带来的问题，命中率比LRU要高
* **复杂度：**MQ需要维护多个队列，且需要维护每个数据的访问时间，复杂度比LRU高
* **代价：**MQ需要记录每个数据的访问时间，需要定时扫描所有队列，代价比LRU要高。
* **注：**虽然MQ的队列看起来数量比较多，但由于所有队列之和受限于缓存容量的大小，因此这里多个队列长度之和和一个LRU队列是一样的，因此队列扫描性能也相近。

**6、算法对比**

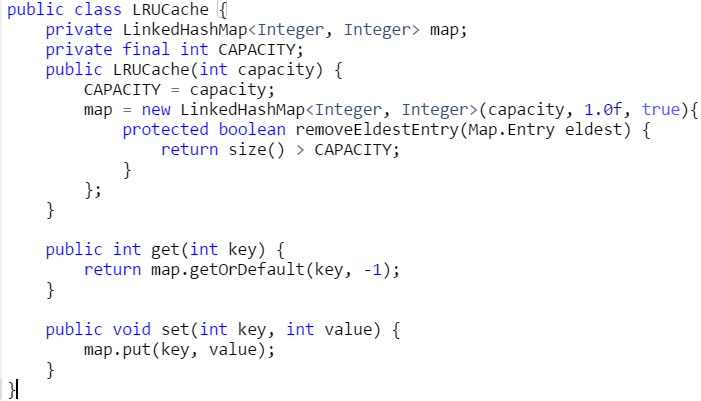
实际应用中需要根据业务的需求和对数据的访问情况进行选择，并不是命中率越高越好。例如：虽然LRU看起来命中率会低一些，且存在”缓存污染“的问题，但由于其简单和代价小，实际应用中反而应用更多。



**三、LRU算法代码实现**

**1、LinkedHashMap**

java中最简单的LRU算法实现，就是利用jdk的LinkedHashMap，覆写其中的removeEldestEntry(Map.Entry)方法即可。LRU算法是通过双向链表来实现，当某个位置被命中，通过调整链表的指向将该位置调整到头位置，新加入的内容直接放在链表头，如此一来，最近被命中的内容就向链表头移动，需要替换时，链表最后的位置就是最近最少使用的位置。



**2、双链表 + HashTable/HashMap**

将Cache的所有位置都用**双连表**连接起来，当一个位置被命中之后，就将通过调整链表的指向，将该位置调整到链表头的位置，新加入的Cache直接加到链表头中。这样，在多次进行Cache操作后，最近被命中的，就会被向链表头方向移动，而没有命中的，而想链表后面移动，链表尾则表示最近最少使用的Cache。当需要替换内容时候，链表的最后位置就是最少被命中的位置，我们只需要淘汰链表最后的部分即可。

