<https://mp.weixin.qq.com/s/uNVB1sgXYccSWHBdTdvL-A>

# 3种堆内缓存算法

要说计算机系统里，什么技术把tradeoff体现的淋漓尽致，那肯定是缓存无疑。为了协调高速部件和低速部件的速度差异，加入一个中间缓存层，是解决这种冲突最有效的方案。

其中，JVM堆内缓存是缓存体系中重要的一环，最常用的有FIFO/LRU/LFU三种算法。

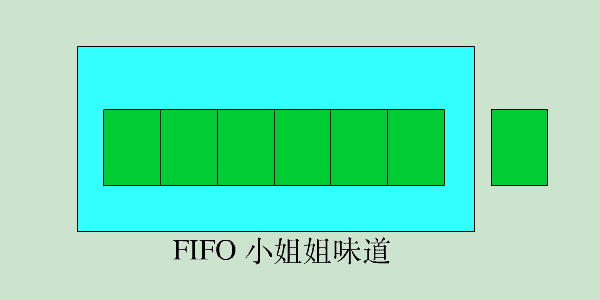
1. FIFO是简单的队列，先进先出。
2. LRU是最近最少使用，优先移除最久未使用的数据。是时间维度。
3. LFU是最近最不常用，优先移除访问次数最少的数据。是统计维度。

由于过期也是缓存的一个重要特点。所有在设计这三种缓存算法时，需要额外的存储空间去存储这个过期时间。

以下将讨论这三种缓存算法的操作和设计要点，但****暂未考虑高并发环境****。

## **FIFO**

先进先出，如果缓存容量满，则优先移出最早加入缓存的数据；其内部可以使用队列实现。



### **操作**

* ****Object get(key)**** ：获取保存的数据，如果数据不存在或者已经过期，则返回null。
* ****void put(key,value,expireTime)****：加入缓存。 无论此key是否已存在，均作为新key处理（移除旧key）；如果空间不足，则移除已过期的key，如果没有，则移除最早加入缓存的key。过期时间未指定，则表示永不自动过期。
* ****注意**** ，我们允许key是有过期时间的，这一点与普通的FIFO有所区别，所以在设计此题时需要注意。（也是面试考察点，偏设计而非算法）

普通的FIFO或许大家都能很简单的写出，增加了过期时间的考虑之后，在设计时需要多考虑。如下示例，为暂未考虑并发环境的FIFO设计。

### **设计思路**

1）用普通的hashMap保存缓存数据。  
2）需要额外的map用来保存key的过期特性，例子中使用了TreeMap，将“剩余存活时间”作为key，利用TreeMap的排序特性。

## **LRU**

least recently used，最近最少使用，是目前最常用的缓存算法和设计方案之一，其移除策略为“当缓存（页）满时，优先移除最近最久未使用的数据”，优点是易于设计和使用，适用场景广泛。算法可以参考leetcode 146 （LRU Cache）。

### **操作**

* ****Object get(key)****：从cache中获取key对应的数据，如果此key已过期，移除此key，并则返回null。
* ****void put(key,value,expired)****：设置k-v，如果容量不足，则根据LRU置换算法移除“最久未被使用的key”。 需要注意，根据LRU优先移除已过期的keys，如果没有，则根据LRU移除未过期的key。如果未设定过期时间，则认为永不自动过期。
* 这里的设计关键是过期时间特性，这与常规的LRU有所不同。

### **设计思路**

* LRU的基础算法，需要了解；每次put、get时需要更新key对应的访问时间，我们需要一个数据结构能够保存key最近的访问时间且能够排序。
* 既然包含过期时间特性，那么带有过期时间的key需要额外的数据结构保存。
* 暂时不考虑并发操作；尽量兼顾空间复杂度和时间复杂度。
* 此题仍然偏向于设计题，而非纯粹的算法题。

此题代码与FIFO基本相同，唯一不同点为get()方法，对于LRU而言，get方法需要重设访问时间（即调整所在cache中顺序）

## **LFU**

最近最不常用，当缓存容量满时，移除 ****访问次数**** 最少的元素，如果访问次数相同的元素有多个，则移除最久访问的那个。设计要求参见leetcode 460（ LFU Cache）

## **End**

本文力求比较三个基本的缓存算法，以便对缓存建设之路有一个比较笼统的感觉。

更加易用的cache，可以参考guava的实现。谨希望这三个代码模版，能够对你有所帮助。