https://mp.weixin.qq.com/s/Q8Mg\_EhDvPVRIaDv6m0Nlg

# LRU Cache

## **LRU Cache**

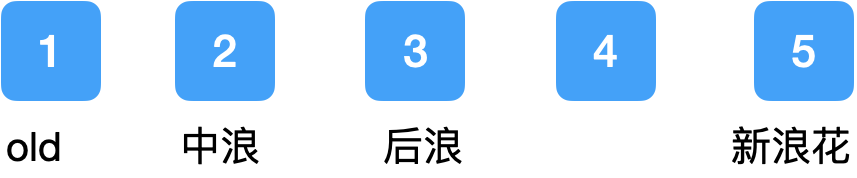
### **LRU 是什么**

LRU = Least Recently Used 最近最少使用  
它是一种缓存逐出策略 **cache eviction policies[1]**

LRU 算法是假设最近最少使用的那些信息，将来被使用的概率也不大，所以在容量有限的情况下，就可以把这些不常用的信息踢出去，腾地方。

比如有热点新闻时，所有人都在搜索这个信息，那刚被一个人搜过的信息接下来被其他人搜索的概率也大，就比前两天的一个过时的新闻被搜索的概率大，所以我们把很久没有用过的信息踢出去，也就是 Least Recently Used 的信息被踢出去。

举个例子：我们的内存容量为 5，现在有 1-5 五个数。



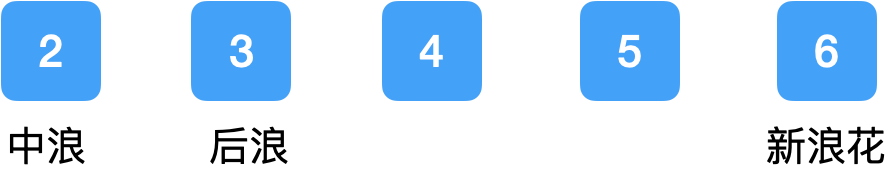
我们现在想加入一个新的数：6  
可是容量已经满了，所以需要踢出去一个。

那按照什么规则踢出去，就有了这个缓存逐出策略。比如：

* FIFO (First In First Out) 这个就是普通的先进先出。
* LFU (Least Frequently Used) 这个是计算每个信息的访问次数，踢走访问次数最少的那个；如果访问次数一样，就踢走好久没用过的那个。这个算法其实很高效，但是耗资源，所以一般不用。
* LRU (Least Recently Used) 这是目前最常用了。

LRU 的规则是把很长时间没有用过的踢出去，那它的隐含假设就是，认为最近用到的信息以后用到的概率会更大。

那我们这个例子中就是把最老的 1 踢出去，变成：



不断迭代...

### **Cache 是什么？**

简单理解就是：把一些可以重复使用的信息存起来，以便之后需要时可以快速拿到。

那至于它存在哪里就不一定了，最常见的是存在内存里，也就是 memory cache，但也可以不存在内存里。

使用场景就更多了，比如 Spring 中有 @Cacheable 等支持 Cache 的一系列注解。上个月我在工作中就用到了这个 annotation，当然是我司包装过的，大大减少了 call 某服务器的次数，解决了一个性能上的问题。

再比如，在进行数据库查询的时候，不想每次请求都去 call 数据库，那我们就在内存里存一些常用的数据，来提高访问性能。

这种设计思想其实是遵循了著名的“二八定律”。在读写数据库时，每次的 I/O 过程消耗很大，**但其实 80% 的 request 都是在用那 20% 的数据**，所以把这 20% 的数据放在内存里，就能够极大的提高整体的效率。

总之，Cache 的目的是存一些可以复用的信息，方便将来的请求快速获得。

### **LRU Cache**

那我们知道了 LRU，了解了 Cache，合起来就是 LRU Cache 了：

当 Cache 储存满了的时候，使用 LRU 算法把老家伙清理出去。

## **思路详解**

说了这么多，Let's get to the meat of the problem!

这道经典题大家都知道是要用 HashMap + Doubly Linked List，或者说用 Java 中现成的 LinkedHashMap，但是，为什么？你是怎么想到用这两个数据结构的？面试的时候不讲清楚这个，不说清楚思考过程，代码写对了也没用。

和在工作中的设计思路类似，没有人会告诉我们要用什么数据结构，一般的思路是先想有哪些 operations，然后根据这些操作，再去看哪些数据结构合适。

### **分析 Operations**

那我们来分析一下对于这个 LRU Cache 需要有哪些操作：

1. 首先最基本的操作就是能够从里面读信息，不然之后快速获取是咋来的；
2. 那还得能加入新的信息，新的信息进来就是 most recently used 了；
3. 在加新信息之前，还得看看有没有空位，如果没有空间了，得先把老的踢出去，那就需要能够找到那个老家伙并且删除它；
4. 那如果加入的新信息是缓存里已经有的，那意思就是 key 已经有了，要更新 value，那就只需要调整一下这条信息的 priority，它已经从上一次被使用升级为最新使用的了。

### **找寻数据结构**

那第一个操作很明显，我们需要一个能够快速查找的数据结构，非 HashMap 莫属，还不了解 HashMap 原理和设计规则的在公众号内发消息「HashMap」，送你一篇爆款文章；

可是后面的操作 HashMap 就不顶用了呀。。。

来来来，我们来数一遍基本的数据结构：  
Array, LinkedList, Stack, Queue, Tree, BST, Heap, HashMap

在做这种数据结构的题目时，就这样把所有的数据结构列出来，一个个来分析，有时候不是因为这个数据结构不行，而是因为其他的数据结构更好。

怎么叫更好？忘了我们的衡量标准嘛！时空复杂度，赶紧复习递归那篇文章，公众号内回复「递归」即可获得。

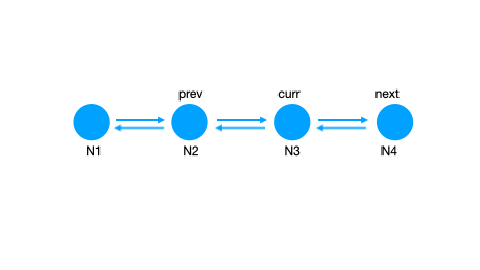
那我们的分析如下：

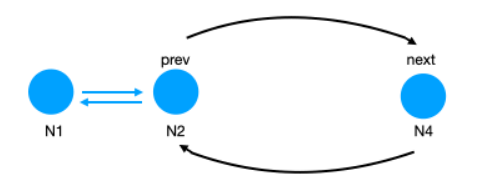
Array, Stack, Queue 这三种本质上都是 Array 实现的（当然 Stack, Queue 也可以用 LinkedList 来实现。。），一会插入新的，一会删除老的，一会调整下顺序，array 不是不能做，就是得 O(n) 啊，用不起。

BST 同理，时间复杂度是 O(logn).

Heap 即便可以，也是 O(logn).

LinkedList，有点可以哦，按照从老到新的顺序，排排站，删除、插入、移动，都可以是 O(1) 的诶！但是删除时我还需要一个 previous pointer 才能删掉，所以我需要一个 Doubly LinkedList.





那么我们的数据结构敲定为：  
HashMap + Doubly LinkedList

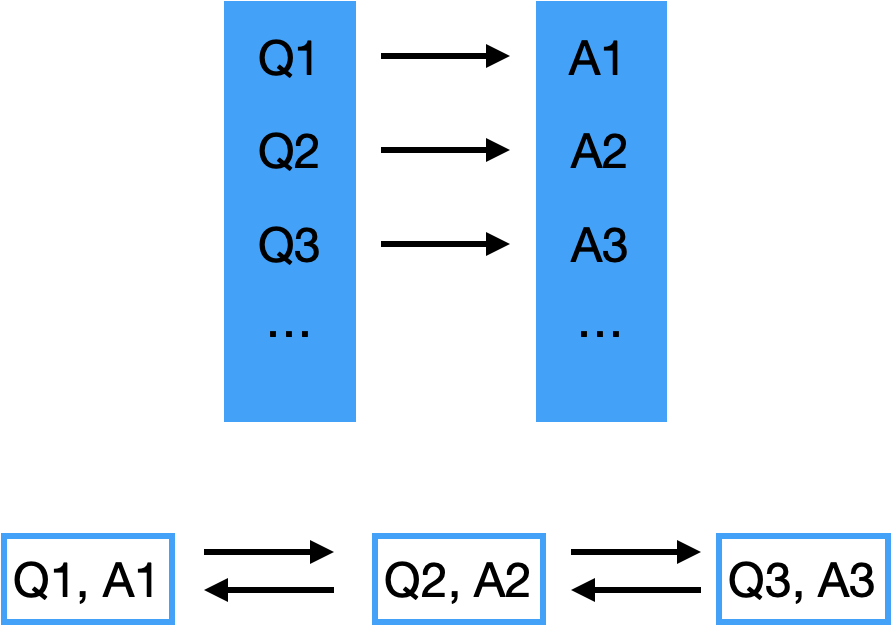
### **定义清楚数据结构的内容**

选好了数据结构之后，还需要定义清楚每个数据结构具体存储的是是什么，这两个数据结构是如何联系的，这才是核心问题。

我们先想个场景，在搜索引擎里，你输入问题 Questions，谷歌给你返回答案 Answer。

那我们就先假设这两个数据结构存的都是 <Q, A>，然后来看这些操作，如果都很顺利，那没问题，如果有问题，我们再调整。

那现在我们的 HashMap 和 LinkedList 长这样：



然后我们回头来看这四种操作：

操作 1，没问题，直接从 HashMap 里读取 Answer 即可，O(1)；

操作 2，新加入一组 Q&A，两个数据结构都得加，那先要判断一下当前的缓存里有没有这个 Q，那我们用 HashMap 判断，

* 如果没有这个 Q，加进来，都没问题；
* 如果已经有这个 Q，HashMap 这里要更新一下 Answer，然后我们还要把 LinkedList 的那个 node 移动到最后或者最前，因为它变成了最新被使用的了嘛。

可是，怎么找 LinkedList 的这个 node 呢？一个个 traverse 去找并不是我们想要的，因为要 O(n) 的时间嘛，我们想用 O(1) 的时间操作。

那也就是说这样记录是不行的，还需要记录 LinkedList 中每个 ListNode 的位置，这就是本题关键所在。

那自然是在 HashMap 里记录 ListNode 的位置这个信息了，也就是存一下每个 ListNode 的 reference。

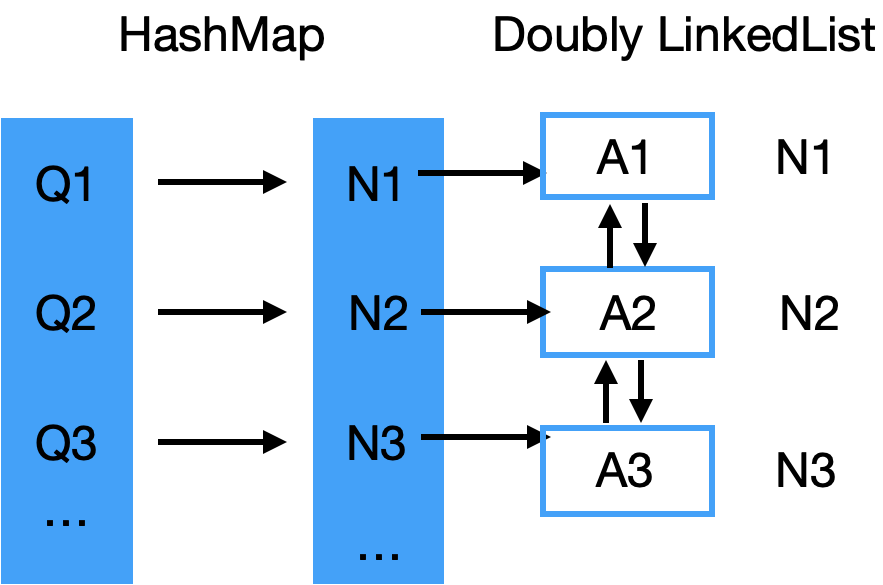
想想其实也是，HashMap 里没有必要记录 Answer，Answer 只需要在 LinkedList 里记录就可以了。

之后我们更新、移动每个 node 时，它的 reference 也不需要变，所以 HashMap 也不用改动，动的只是 previous, next pointer.

那再一想，其实 LinkedList 里也没必要记录 Question，反正 HashMap 里有。

这两个数据结构是相互配合来用的，不需要记录一样的信息。

更新后的数据结构如下：



这样，我们才分析出来用什么数据结构，每个数据结构里存的是什么，物理意义是什么。

那其实，Java 中的 LinkedHashMap 已经做了很好的实现。但是，即便面试时可以使用它，也是这么一步步推导出来的，而不是一看到题目就知道用它，那一看就是背答案啊。

有同学问我，如果面试官问我这题做没做过，该怎么回答？

答：实话实说。

**真诚**在面试、工作中都是很重要的，所以实话实说就好了。但如果面试官没问，就不必说。。。

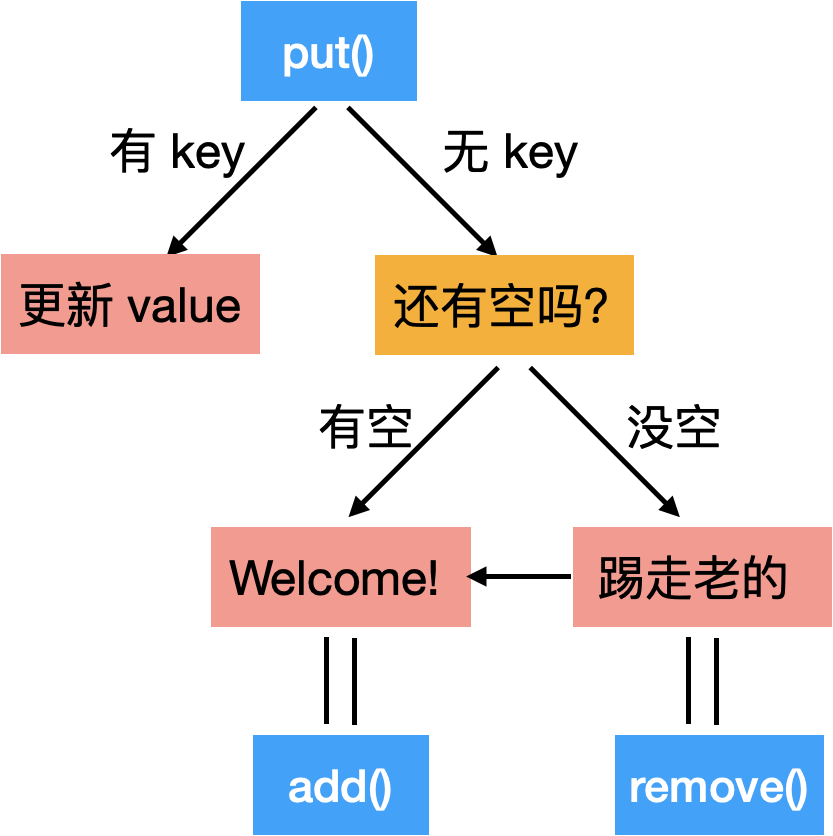
其实面试官是不 care 你做没做过这道题的，因为大家都刷题，基本都做过，问这个问题没有意义。只要你能把问题分析清楚，讲清楚逻辑，做过了又怎样？很多做过了题的人是讲不清楚的。。。

### **总结**

那我们再总结一下那四点操作：

第一个操作，也就是 get() API，没啥好说的；

二三四，是 put() API，有点小麻烦：



画图的时候边讲边写，每一步都从 high level 到 detail 再到代码，把代码模块化。

* 比如“Welcome”是要把这个新的信息加入到 HashMap 和 LinkedList 里，那我会用一个单独的 add() method 来写这块内容，那在下面的代码里我取名为 appendHead()，更精准；
* “踢走老的”这里我也是用一个单独的 remove() method 来写的。

当年我把这图画出来，面试官就没让我写代码了，直接下一题了...

那如果面试官还让你写，就写呗。。。