20 | 散列表（下）：为什么散列表和链表经常会一起使用?

2018-11-05 王争

20 | 散列表（下）：为什么散列表和链表经常会一起使用?
朗读人：修阳 11′49″ | 5.42M

我们已经学习了 20 节内容，你有没有发现，有两种数据结构，散列表和链表，经常会被放在一起使用。你还记得，前面的章节中都有哪些地方讲到散列表和链表的组合使用吗？我带你一起回忆一下。

在链表那一节，我讲到如何用链表来实现 LRU 缓存淘汰算法，但是链表实现的 LRU 缓存淘汰算法的时间复杂度是 O(n)，当时我也提到了，通过散列表可以将这个时间复杂度降低到 O(1)。

在跳表那一节，我提到 Redis 的有序集合是使用跳表来实现的，跳表可以看作一种改进版的链表。当时我们也提到，Redis 有序集合不仅使用了跳表，还用到了散列表。

除此之外，如果你熟悉 Java 编程语言，你会发现 LinkedHashMap 这样一个常用的容器，也用到了散列表和链表两种数据结构。

今天，我们就来看看，在这几个问题中，散列表和链表都是如何组合起来使用的，以及为什么散列表和链表会经常放到一块使用。

LRU 缓存淘汰算法

在链表那一节中，我提到，借助散列表，我们可以把 LRU 缓存淘汰算法的时间复杂度降低为 O(1)。现在，我们就来看看它是如何做到的。

首先，我们来回顾一下当时我们是如何通过链表实现 LRU 缓存淘汰算法的。

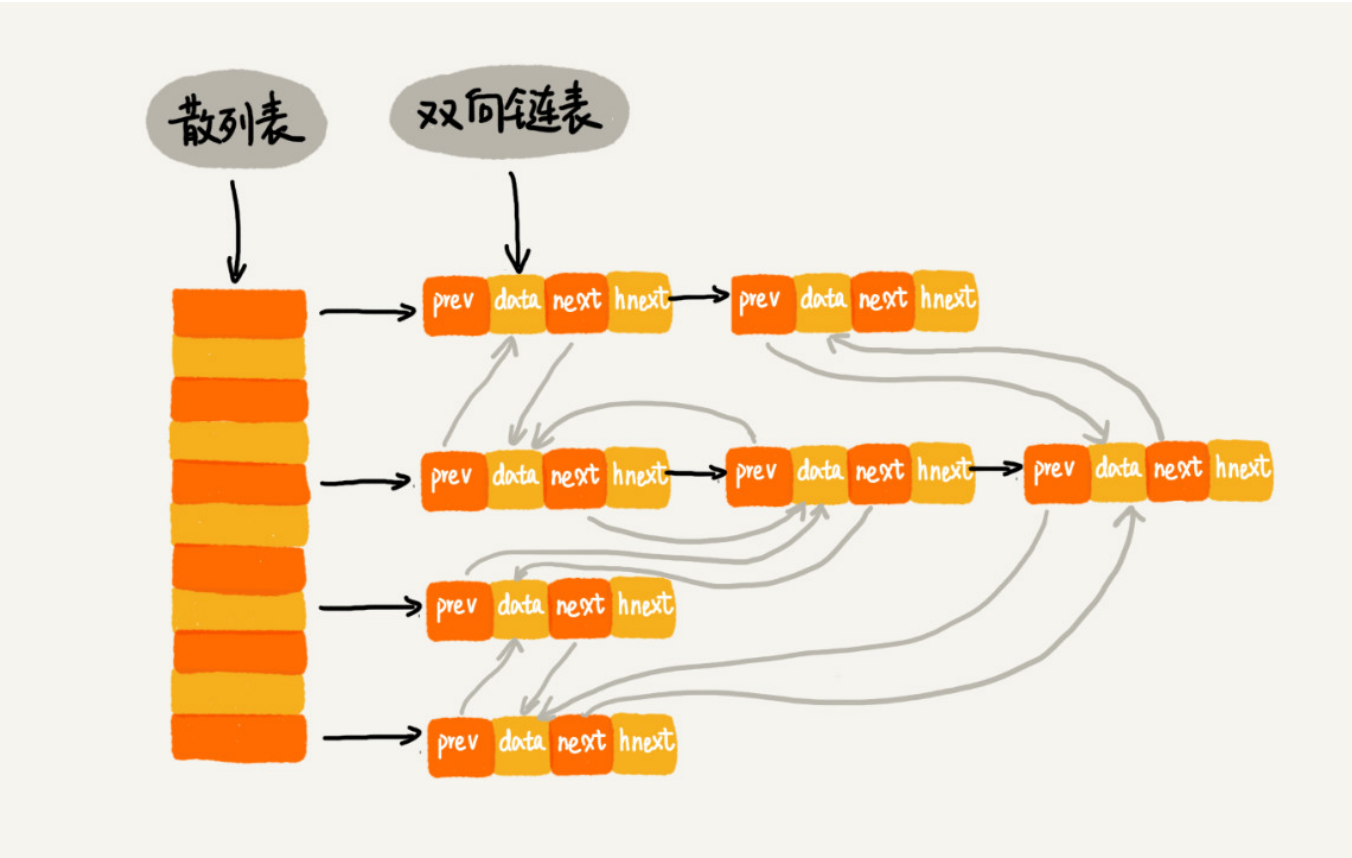
我们需要维护一个按照访问时间从大到小有序排列的链表结构。因为缓存大小有限，当缓存空间不够，需要淘汰一个数据的时候，我们就直接将链表头部的结点删除。

当要缓存某个数据的时候，先在链表中查找这个数据。如果没有找到，则直接将数据放到链表的尾部；如果找到了，我们就把它移动到链表的尾部。因为查找数据需要遍历链表，所以单纯用链表实现的 LRU 缓存淘汰算法的时间复杂很高，是 O(n)。

实际上，我总结一下，一个缓存（cache）系统主要包含下面这几个操作：

- 往缓存中添加一个数据；
- 从缓存中删除一个数据；
- 在缓存中查找一个数据。

这三个操作都要涉及“查找”操作，如果单纯地采用链表的话，时间复杂度只能是 O(n)。如果我们将散列表和链表两种数据结构组合使用，可以将这三个操作的时间复杂度都降低到 O(1)。具体的结构就是下面这个样子：



我们使用双向链表存储数据，链表中的每个结点处理存储数据（data）、前驱指针（prev）、后继指针（next）之外，还新增了一个特殊的字段 hnext。这个 hnext 有什么作用呢?

因为我们的散列表是通过链表法解决散列冲突的，所以每个结点会在两条链中。一个链是刚刚我们提到的**双向链表**，另一个链是散列表中的**拉链**。**前驱和后继指针是为了将结点串在双向链表中，hnext 指针是为了将结点串在散列表的拉链中**。

了解了这个散列表和双向链表的组合存储结构之后，我们再来看，前面讲到的缓存的三个操作，是如何做到时间复杂度是 O(1) 的?

首先，我们来看**如何查找一个数据**。我们前面讲过，散列表中查找数据的时间复杂度接近 O(1)，所以通过散列表，我们可以很快地在缓存中找到一个数据。当找到数据之后，我们还需要将它移动到双向链表的尾部。

其次，我们来看**如何删除一个数据**。我们需要找到数据所在的结点，然后将结点删除。借助散列表，我们可以在 O(1) 时间复杂度里找到要删除的结点。因为我们的链表是双向链表，双向链表可以通过前驱指针 O(1) 时间复杂度获取前驱结点，所以在双向链表中，删除结点只需要 O(1) 的时间复杂度。

最后，我们来看**如何添加一个数据**。添加数据到缓存稍微有点麻烦，我们需要先看这个数据是否已经在缓存中。如果已经在其中，需要将其移动到双向链表的尾部；如果不在其中，还要看缓存有没有满。如果满了，则将双向链表头部的结点删除，然后再将数据放到链表的尾部；如果没有满，就直接将数据放到链表的尾部。

这整个过程涉及的查找操作都可以通过散列表来完成。其他的操作，比如删除头结点、链表尾部插入数据等，都可以在 O(1) 的时间复杂度内完成。所以，这三个操作的时间复杂度都是 O(1)。至此，我们就通过散列表和双向链表的组合使用，实现了一个高效的、支持 LRU 缓存淘汰算法的缓存系统原型。

Redis 有序集合

在跳表那一节，讲到有序集合的操作时，我稍微做了些简化。实际上，在有序集合中，每个成员对象有两个重要的属性，**key**（键值）和**score**（分值）。我们不仅会通过 score 来查找数据，还会通过 key 来查找数据。

举个例子，比如用户积分排行榜有这样一个功能：我们可以通过用户的 ID 来查找积分信息，也可以通过积分区间来查找用户 ID 或者姓名信息。这里包含 ID、姓名和积分的用户信息，就是成员对象，用户 ID 就是 key，积分就是 score。

所以，如果我们细化一下 Redis 有序集合的操作，那就是下面这样：

- 添加一个成员对象；

- 按照键值来删除一个成员对象；
- 按照键值来查找一个成员对象；
- 按照分值区间查找数据，比如查找积分在 [100, 356] 之间的成员对象；
- 按照分值从小到大排序成员变量；

如果我们仅仅按照分值将成员对象组织成跳表的结构，那按照键值来删除、查询成员对象就会很慢，解决方法与 LRU 缓存淘汰算法的解决方法类似。我们可以再按照键值构建一个散列表，这样按照 key 来删除、查找一个成员对象的时间复杂度就变成了 O(1)。同时，借助跳表结构，其他操作也非常高效。

实际上，Redis 有序集合的操作还有另外一类，也就是查找成员对象的排名（Rank）或者根据排名区间查找成员对象。这个功能单纯用刚刚讲的这种组合结构就无法高效实现了。这块内容我后面的章节再讲。

Java LinkedHashMap

前面我们讲了两个散列表和链表结合的例子，现在我们再来看另外一个，Java 中的 LinkedHashMap 这种容器。

如果你熟悉 Java，那你几乎天天会用到这个容器。我们之前讲过，HashMap 底层是通过散列表这种数据结构实现的。而 LinkedHashMap 前面比 HashMap 多了一个“Linked”，这里的“Linked”是不是说，LinkedHashMap 是一个通过链表法解决散列冲突的散列表呢？

实际上，LinkedHashMap 并没有这么简单，其中的“Linked”也并不仅仅代表它是通过链表法解决散列冲突的。关于这一点，在我是初学者的时候，也误解了很久。

我们先来看一段代码。你觉得这段代码会以什么样的顺序打印 3，1，5，2 这几个 key 呢？原因又是什么呢？

复制代码

```
HashMap<Integer, Integer> m = new LinkedHashMap<>();
m.put(3, 11);
m.put(1, 12);
m.put(5, 23);
m.put(2, 22);

for (Map.Entry e : m.entrySet()) {
  System.out.println(e.getKey());
}
```

我先告诉你答案，上面的代码会按照数据插入的顺序依次来打印，也就是说，打印的顺序就是 3，1，5，2。你有没有觉得奇怪？散列表中数据是经过散列函数打乱之后无规律存储的，这里是

如何实现按照数据的插入顺序来遍历打印的呢?

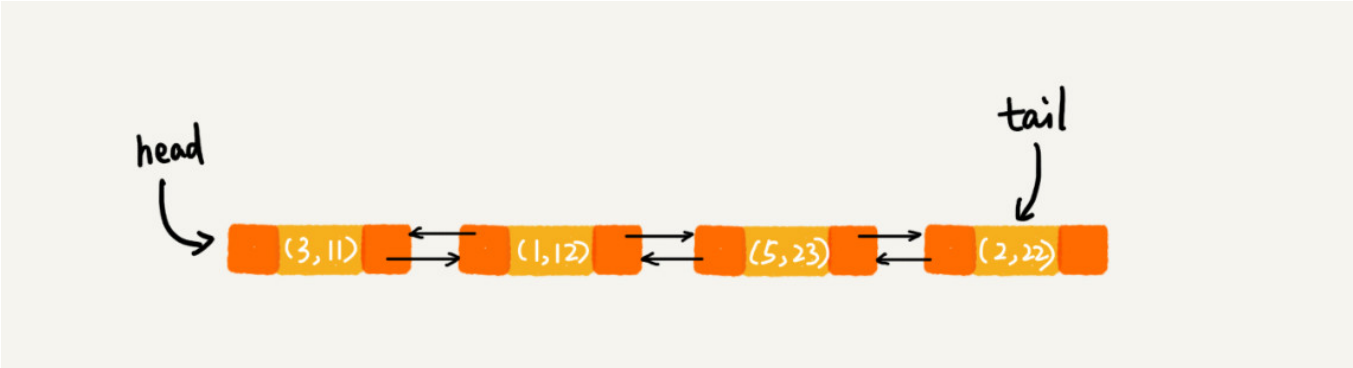
你可能已经猜到了，LinkedHashMap 也是通过散列表和链表组合在一起实现的。实际上，它不仅支持按照插入顺序遍历数据，还支持按照访问顺序来遍历数据。你可以看下面这段代码：

复制代码

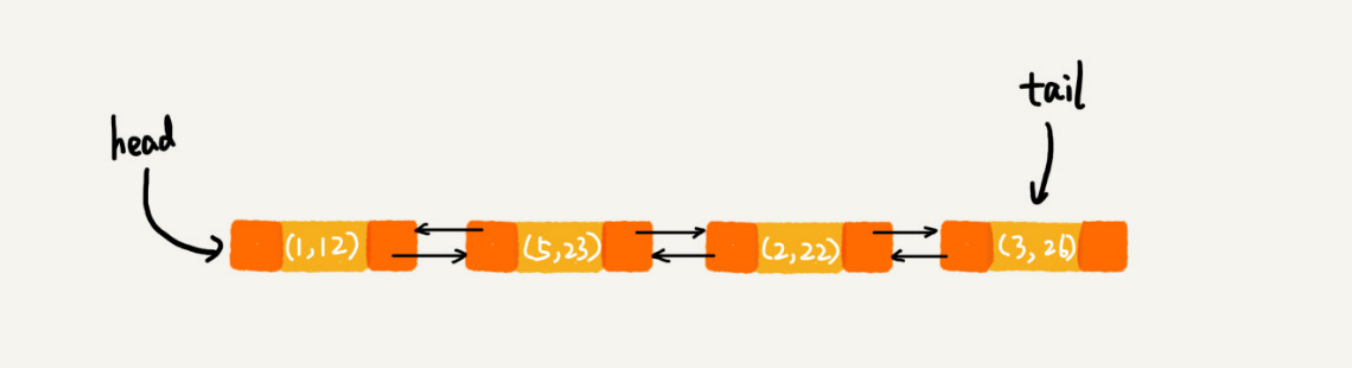
```
1  // 10 是初始大小，0.75 是装载因子，true 是表示按照访问时间排序
2  HashMap<Integer, Integer> m = new LinkedHashMap<>(10, 0.75f, true);
3  m.put(3, 11);
4  m.put(1, 12);
5  m.put(5, 23);
6  m.put(2, 22);
7
8  m.put(3, 26);
9  m.get(5);
10
11 for (Map.Entry e : m.entrySet()) {
12   System.out.println(e.getKey());
13 }
```

这段代码打印的结果是 1，2，3，5。我来具体分析一下，为什么这段代码会按照这样顺序来打印。

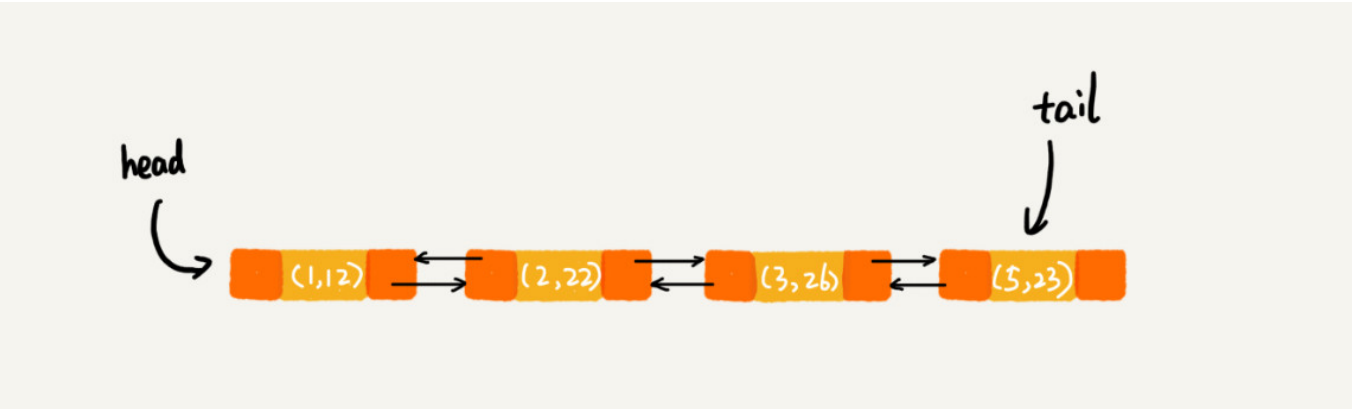
每次调用 put() 函数，往 LinkedHashMap 中添加数据的时候，都会将数据添加到链表的尾部，所以，在前四个操作完成之后，链表中的数据是下面这样：



在第 8 行代码中，再次将键值为 3 的数据放入到 LinkedHashMap 的时候，会先查找这个键值是否已经有了，然后，再将已经存在的 (3,11) 删除，并且将新的 (3,26) 放到链表的尾部。所以，这个时候链表中的数据就是下面这样：



当第 9 行代码访问到 key 为 5 的数据的时候，我们将被访问到的数据移动到链表的尾部。所以，第 9 行代码之后，链表中的数据是下面这样：



所以，最后打印出来的数据是 1，2，3，5。从上面的分析，你有没有发现，按照访问时间排序的 LinkedHashMap 本身就是一个支持 LRU 缓存淘汰策略的缓存系统？实际上，它们两个的实现原理也是一模一样的。我也就不再啰嗦了。

我现在来总结一下，实际上，**LinkedHashMap 是通过双向链表和散列表这两种数据结构组合实现的。LinkedHashMap 中的“Linked”实际上是指的是双向链表，并非指用链表法解决散列冲突。**

解答开篇 & 内容小结

弄懂刚刚我讲的这三个例子，开篇的问题也就不言而喻了。我这里总结一下，为什么散列表和链表经常一块使用？

散列表这种数据结构虽然支持非常高效的数据插入、删除、查找操作，但是散列表中的数据都是通过散列函数打乱之后无规律存储的。也就说，它无法支持按照某种顺序快速地遍历数据。如果希望按照顺序遍历散列表中的数据，那我们需要将散列表中的数据拷贝到数组中，然后排序，再遍历。

因为散列表是动态数据结构，不停地有数据的插入、删除，所以每当我们希望按顺序遍历散列表中的数据的时候，都需要先排序，那效率势必会很低。为了解决这个问题，我们将散列表和链表（或者跳表）结合在一起使用。

课后思考

1. 今天讲的几个散列表和链表结合使用的例子里，我们用的都是双向链表。如果把双向链表改成单链表，还能否正常工作呢？为什么呢？

2. 假设猎聘网有 10 万名猎头，每个猎头都可以通过做任务（比如发布职位）来积累积分，然后通过积分来下载简历。假设你是猎聘网的一名工程师，如何在内存中存储这 10 万个猎头 ID 和积分信息，让它能够支持这样几个操作：

- 根据猎头的 ID 快速查找、删除、更新这个猎头的积分信息；
- 查找积分在某个区间的猎头 ID 列表；
- 查找按照积分从小到大排名在第 x 位到第 y 位之间的猎头 ID 列表。

欢迎留言和我分享，我会第一时间给你反馈。

写留言

精选留言

Keep-Moving 4

LRU查找数据，查找到之后，不是应该把数据放到链表的头部吗？为什么这里说是尾部？

2018-11-05

作者回复

两种方式都可以的

2018-11-05

等风来 2

1.改成单链表,删除/插入的时候需要O(n)去找前驱节点;
2.如文中第一个列子,按ID顺序存储双向链表;在双向链表按积分hash和按ID跳表;

2018-11-05

Smallfly 2

1.

在删除一个元素时，虽然能 O(1) 的找到目标结点，但是要删除该结点需要拿到前一个结点的指针，遍历到前一个结点复杂度会变为 O(N) ，所以用双链表实现比较合适。

（但其实硬要操作的话，单链表也是可以实现 O(1) 时间复杂度删除结点的）。

iOS 的同学可能知道，YYMemoryCache 就是结合散列表和双向链表来实现的。

2.

以积分排序构建一个跳表，再以猎头 ID 构建一个散列表。

1）ID 在散列表中所以可以 O(1) 查找到这个猎头；
2）积分以跳表存储，跳表支持区间查询；
3）这点根据目前学习的知识暂时无法实现，老师文中也提到了。

2018-11-05

Smallfly 2

通过这 20 节课学习下来，个人感觉其实就两种数据结构，链表和数组。

数组占据随机访问的优势，却有需要连续内存的缺点。

链表具有可不连续存储的优势，但访问查找是线性的。

散列表和链表、跳表的混合使用，是为了结合数组和链表的优势，规避它们的不足。

我们可以得出数据结构和算法的重要性排行榜：连续空间 > 时间 > 碎片空间。

PS：跟专业的书籍相比，老师讲的真的是通俗易懂不废话，篇篇是干货。如果这个课程学不下去，学其它的会更加困难。暂时不懂的话反复阅读复习，外加查阅，一定可以的！

2018-11-05

国富 1

1.双联表改成单链表，依然可以工作。可以用一个变量存储遍历到的节点的前驱指针。
2.可以把猎聘网的猎头的信息存储在 散列表和链表（跳表）组合使用的容器中，其中按照猎

头id建立散列表，按照猎头的积分建立一个跳表。这样，无论是按照id查用户，还是按照积分进行排序和区间查找都会很高效。

2018-11-05

灰飞灰猪不会灰飞.烟灭 1

red is中key是按照什么方式（算法）hash的?

一般key是个字符串，是不是按照什么方式方式进行hash散列存储?

2018-11-05

莫问流年 1

怎么判断缓存已满，是要维护一个计数变量吗

2018-11-05

峰 0

思考题第一题，让散列表的指向不再是原节点，而是其前驱节点，就可以在相同的时间复杂度内支持原先的操作。

第二题，主要是排序区间怎么支持，没想出来。。。。

2018-11-05

komo0104 0

有一个问题。

LRU算法中，查找的一个数据是O(1)，然后将他移动到尾部为什么还是O(1)呢。

除非也维护了一个指向尾部的指针？不然找到尾部的复杂度是O(n)呀。

2018-11-05

longer 0

1、如果改成单链表:对插入和删除有影响,因为单链表的插入和删除时间复杂度为O(n)，无法做到快速插入和删除；但是对查找没有影响，因为查找还是走的散列表，时间复杂度为O(1)。

2、(猎头Id， 积分)构建散列表，以猎头id做散列，分数做链表进行排序。

2018-11-05

小动物很困 0

对于第一个问题:

单向链表可以实现,但是对于数据找到后的对于链表的操作时间复杂度高.

第二个问题:

可以使用hash表+链表实现,认为id基本上唯一,不存在hash冲突

hash表 key=id value=指向 {id,分数}对象链表节点的引用

但是有一个问题:

是获取范围数据的时候对链表排序,还是在插入的时候使对象链表按照分数有序化.

欢迎大佬指正

2018-11-05

雪无痕 0

java LinkedHashMap讲解中的第二个链表图和第三个链表图中的（3，11）节点写错了，应该是（3，26）

2018-11-05

刘远通 0

散列表 按照key(一般是文字) 映射成数值 方便查找
score 链表 一般是 按顺序遍历 可以使用红黑树方式分层加速

2018-11-05

猫头鹰爱拿铁 0

1.可以通过单链表和散列表实现，但是删除和添加的时间复杂度就变成了O(n)，因为需要遍历一次链表将前驱节点找到，再进行删除。
2.猎头问题：每个猎头对象由node构成（pre，next，hnext，data）将id作为键值建立类似hashmap的结构来存放猎头的对象，同时再将每个节点使用双向链表按照积分大小（快排排序）链接起来。根据id查找、删除、添加时间复杂度为O(1)，查找排名的时间复杂度为O(n),如果想提高查找排名的时间复杂度，可以再和跳表结合一块，根据积分建立索引，查找排名的时间复杂度将提升为O(logn)

2018-11-05

NeverMore 0

对于第一题，我觉得是可以的，每次查找时同时增加一个变量为前驱节点，多了一个O(1)的空间复杂度而已
第二题，类似于举的例子，通过散列表和链表或者调表等，都可以实现

2018-11-05

『LHCY』 0

1.不能，如果是单向链表，通过hash找到这个节点时，并不知道这个节点的上一个节点，在删除和移节点时还要从头遍历链表。
2.一个存储id和得分的哈希表，一个存储按照得分排序的猎头对象的堆。

2018-11-05

拉欧 0

1.双向链表的插入和删除是O(1),单向链表因为要查额外查询前驱节点，所以是O(N),所以不合适
2.维护两个数据结构，第一个是key为id,value为score的hashmap，第二个是key为score，value为ID的hashmap。

2018-11-05

城 0

1.如果将双向链表改成单向链表，则无法正常工作，因为单向链表没有指向前驱节点的指针，在插入时，或许和双向链表一样，但是删除时，就要遍历单链表了。
2.考虑到有根据积分范围查询符合条件的猎头集合，因此选择跳表，要支持快速查找、删除、更新，数据结构再加上散列表。最后根据排名选择x-y排名的猎头，我简单的使用遍历。

2018-11-05

卡罗

0

课后思考1，能用单向链表时间，不过每次删除操作，就需要重新遍历所有单向链表，时间复杂度会提高。

2018-11-05