# 带着问题去学习：

**1.为什么散列表和链表经常放在一起使用?**

**2.散列表和链表如何组合起来使用?**

## 一、为什么散列表和链表经常放在一起使用?

1. 散列表的优点：支持高效的数据插入、删除和查找操作

2. 散列表的缺点：不支持快速顺序遍历散列表中的数据

3. 如何按照顺序快速遍历散列表的数据？只能将数据转移到数组，然后排序，最后再遍历数据。

4. 我们知道散列表是动态的数据结构，需要频繁的插入和删除数据，那么每次顺序遍历之前都需要先排序，这势必会造成效率非常低下。

5. 如何解决上面的问题呢？就是将散列表和链表（或跳表）结合起来使用。

## 二、散列表和链表如何组合起来使用?

### 1.LRU（Least Recently Used）缓存淘汰算法

#### 1.1.LRU缓存淘汰算法主要操作有哪些？主要包含3个操作:

①往缓存中添加一个数据；

②从缓存中删除一个数据；

③在缓存中查找一个数据；

④总结：上面3个都涉及到查找。

#### 1.2.如何用链表实现LRU缓存淘汰算法?

①需要维护一个按照访问时间从大到小的有序排列的链表结构。

②缓冲空间有限，当空间不足需要淘汰一个数据时直接删除链表头部的节点。

③当要缓存某个数据时，先在链表中查找这个数据。若未找到，则直接将数据放到链表的尾部。若找到，就把它移动到链表尾部。

④前面说了，LRU缓存的3个主要操作都涉及到查找，若单纯由链表实现，查找的时间复杂度很高为O(n)。若将链表和散列表结合使用，查找的时间复杂度会降低到O(1)。

#### 1.3.如何使用散列表和链表实现LRU缓存淘汰算法?

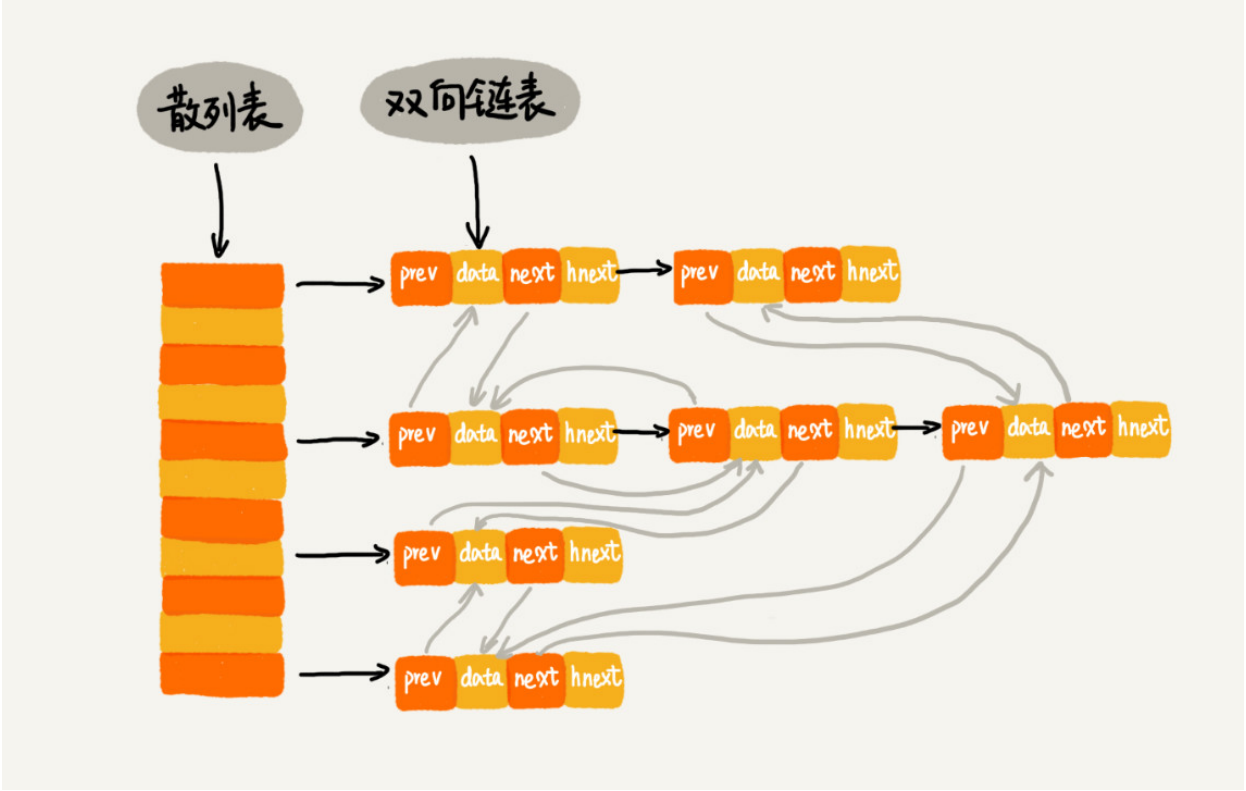
①使用双向链表存储数据，链表中每个节点存储数据（data）、前驱指针（prev）、后继指针（next）和hnext指针（解决散列冲突的链表指针）。

②散列表通过链表法解决散列冲突，所以每个节点都会在两条链中。一条链是双向链表，另一条链是散列表中的拉链。前驱和后继指针是为了将节点串在双向链表中，hnext指针是为了将节点串在散列表的拉链中。

③LRU缓存淘汰算法的3个主要操作如何做到时间复杂度为O(1)呢？

首先，我们明确一点就是链表本身插入和删除一个节点的时间复杂度为O(1)，因为只需更改几个指针指向即可。

接着，来分析查找操作的时间复杂度。当要查找一个数据时，通过散列表可实现在O(1)时间复杂度找到该数据，再加上前面说的插入或删除的时间复杂度是O(1)，所以我们总操作的时间复杂度就是O(1)。




## 2.Redis有序集合

### 2.1.什么是有序集合?

①在有序集合中，每个成员对象有2个重要的属性，即key（键值）和score（分值）。

②不仅会通过score来查找数据，还会通过key来查找数据。

### 2.2.有序集合的操作有哪些?

举个例子，比如用户积分排行榜有这样一个功能：可以通过用户ID来查找积分信息，也可以通过积分区间来查找用户ID。这里用户ID就是key，积分就是score。所以，有序集合的操作如下：

①添加一个对象；

②根据键值删除一个对象；

③根据键值查找一个成员对象；

④根据分值区间查找数据，比如查找积分在[100.356]之间的成员对象；

⑤按照分值从小到大排序成员变量。

这时可以按照分值将成员对象组织成跳表结构，按照键值构建一个散列表。那么上面的所有操作都非常高效。

## 3.Java LinkedHashMap

和LRU缓存淘汰策略实现一模一样。支持按照插入顺序遍历数据，也支持按照访问顺序遍历数据。

## 三、课后思考

**1.上面所讲的几个散列表和链表组合的例子里，我们都是使用双向链表。如果把双向链表改成单链表，还能否正常工作？为什么呢？**

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.假设猎聘网有10万名猎头，每个猎头可以通过做任务（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10万个猎头的ID和积分信息，让它能够支持这样几个操作：**

1）根据猎头ID查收查找、删除、更新这个猎头的积分信息；

2）查找积分在某个区间的猎头ID列表；

3）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。

极客时间文档：https://time.geekbang.org/column/article/64858