哈希表理论基础

哈希表

首先什么是哈希表，哈希表（英文名字为Hash table，国内也有一些算法书籍翻译为散列表，大家看到这两个名称知道都是指hash table就可以了）。

> 哈希表是根据关键码的值而直接进行访问的数据结构。

这么官方的解释可能有点懵，其实直白来讲其实数组就是一张哈希表。

哈希表中关键码就是数组的索引下标，然后通过下标直接访问数组中的元素，如下图所示：

那么哈希表能解决什么问题呢，**一般哈希表都是用来快速判断一个元素是否出现集合里。**

例如要查询一个名字是否在这所学校里。

要枚举的话时间复杂度是O(n)，但如果使用哈希表的话， 只需要O(1)就可以做到。

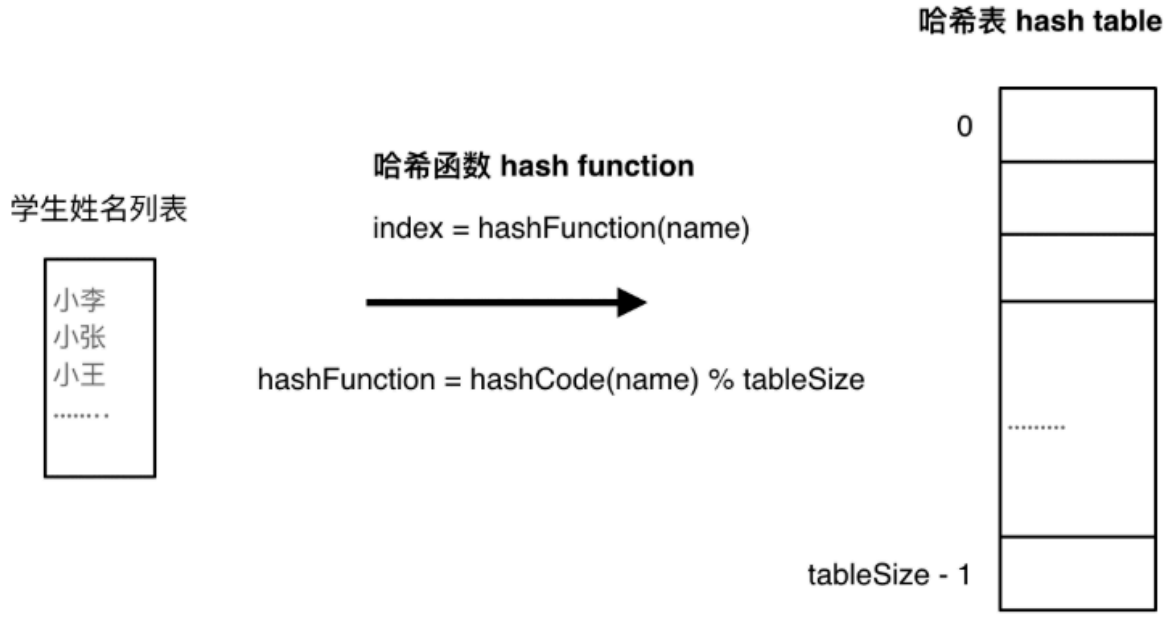
我们只需要初始化把这所学校里学生的名字都存在哈希表里，在查询的时候通过索引直接就可以知道这位同学在不在这所学校里了。

将学生姓名映射到哈希表上就涉及到了**hash function ， 也就是哈希函数。**

哈希函数

哈希函数，把学生的姓名直接映射为哈希表上的索引，然后就可以通过查询索引下标快速知道这位同学是否在这所学校里了。

哈希函数如下图所示，通过hashCode把名字转化为数值，一般hashcode是通过特定编码方式，可以将其他数据格式转化为不同的数值，这样就把学生名字映射为哈希表上的索引数字了。



如果hashCode得到的数值大于 哈希表的大小了，也就是大于tableSize了，怎么办呢？

此时为了保证映射出来的索引数值都落在哈希表上，我们会在再次对数值做一个取模的操作，这样我们就保证了学生姓名一定可以映射到哈希表上了。

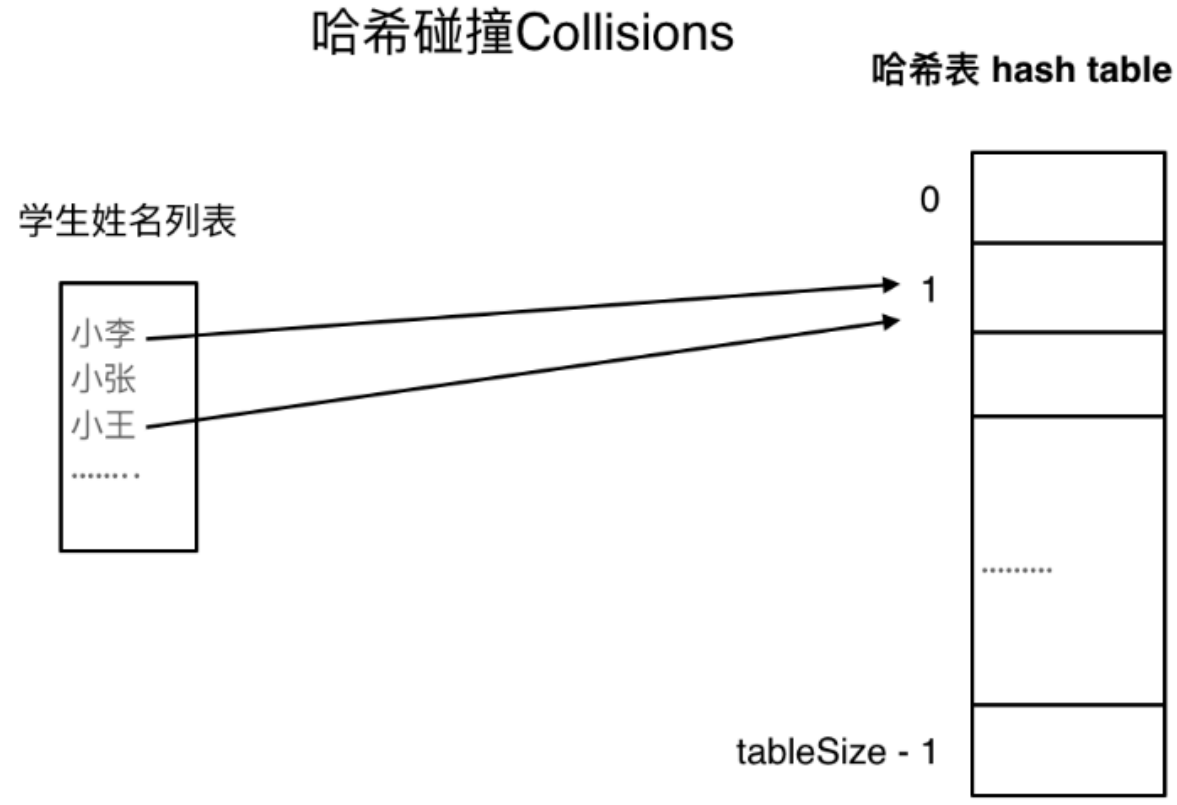
此时问题又来了，哈希表我们刚刚说过，就是一个数组。

如果学生的数量大于哈希表的大小怎么办，此时就算哈希函数计算的再均匀，也避免不了会有几位学生的名字同时映射到哈希表 同一个索引下标的位置。

接下来**哈希碰撞**登场

哈希碰撞

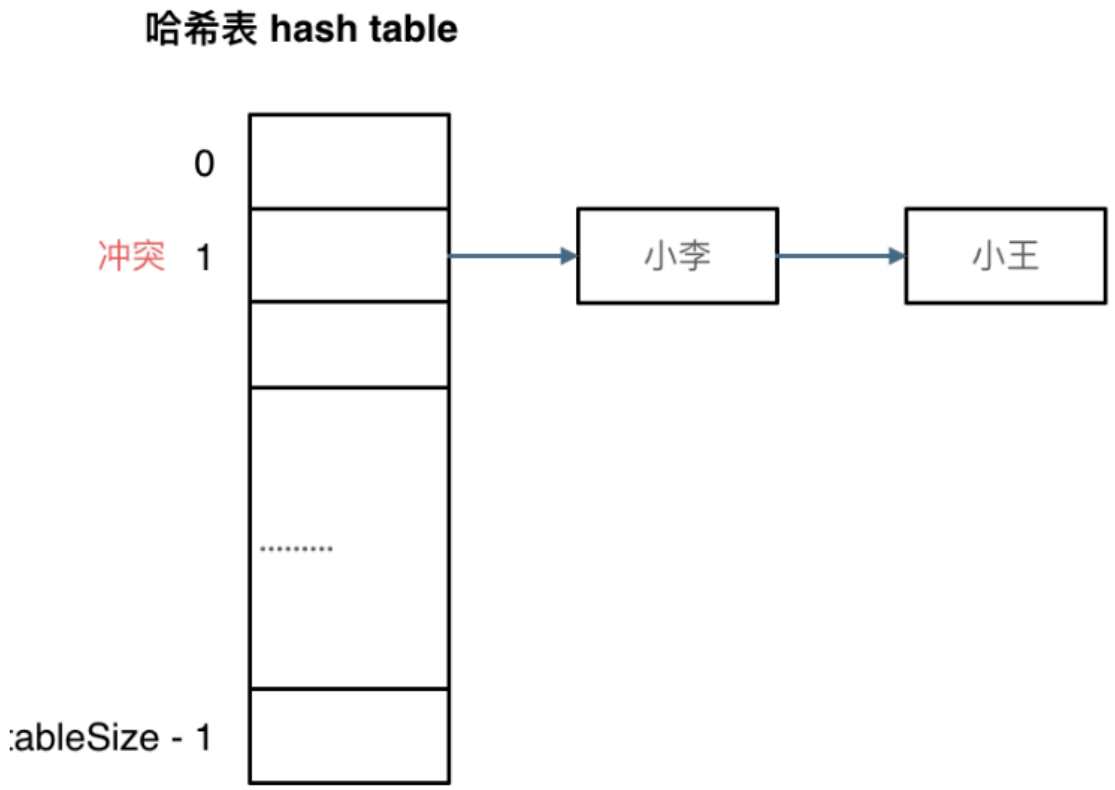
如图所示，小李和小王都映射到了索引下标 1 的位置，**这一现象叫做哈希碰撞。**



般哈希碰撞有两种解决方法， 拉链法和线性探测法。

拉链法

刚刚小李和小王在索引1的位置发生了冲突，发生冲突的元素都被存储在链表中。 这样我们就可以通过索引找到小李和小王了



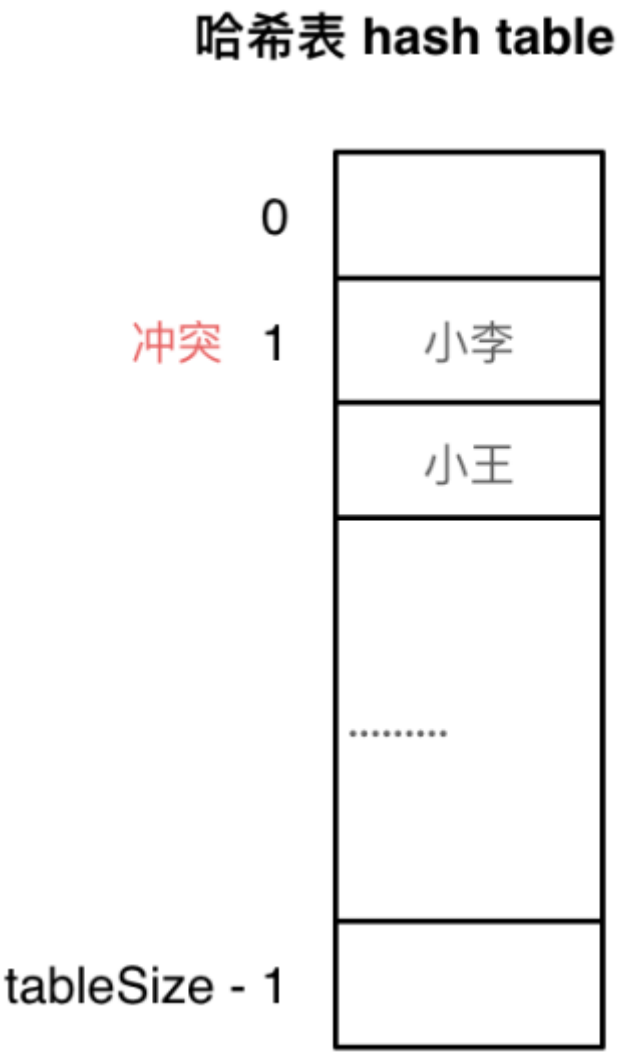
（数据规模是dataSize， 哈希表的大小为tableSize）

其实拉链法就是要选择适当的哈希表的大小，这样既不会因为数组空值而浪费大量内存，也不会因为链表太长而在查找上浪费太多时间。

线性探测法

使用线性探测法，一定要保证tableSize大于dataSize。 我们需要依靠哈希表中的空位来解决碰撞问题。

例如冲突的位置，放了小李，那么就向下找一个空位放置小王的信息。所以要求tableSize一定要大于dataSize ，要不然哈希表上就没有空置的位置来存放冲突的数据了。如图所示：



其实关于哈希碰撞还有非常多的细节，感兴趣的同学可以再好好研究一下，这里我就不再赘述了。

常见的三种哈希结构

当我们想使用哈希法来解决问题的时候，我们一般会选择如下三种数据结构。

- 数组
- set （集合）
- map(映射)

这里数组就没啥可说的了，我们来看一下set。

1.哈希集合（Set）

Python 中的集合是基于 **哈希表** 实现的，提供了高效的元素查找、插入和删除操作。常见的集合包括：`set` 和 `frozenset`。

(1)Set 不能修改元素的操作:

`set` 是一个 **无序** 的集合，它 **不允许重复元素**，并且 **元素必须是不可变的（hashable）**，例如元组可以作为集合的元素，而列表不行。集合本身是 **可变的**，这意味着你可以 **添加** 或 **删除** 元素，但 **不能修改** 已存在的元素。

(2)为什么集合不能修改元素?

- **集合是无序的**：集合中的元素是没有顺序的，因此不能像列表那样通过索引直接修改一个元素。

- **元素必须是不可变类型**：集合要求元素是 **不可变类型**，也就是 `hashable`，例如整数、字符串、元组等。这是因为哈希表底层需要使用元素的哈希值来存储和查找元素。如果允许修改元素，哈希值可能会发生变化，从而破坏哈希表的结构。

(3)集合的操作

你可以对集合进行以下操作：

- **添加元素**：你可以使用 `.add()` 方法向集合中添加元素。
- **删除元素**：你可以使用 `.remove()` 或 `.discard()` 方法删除集合中的元素。

但是，你不能直接修改已经存在的元素。

`set`

- **底层实现**：基于 **哈希表。**
- **是否有序**：无序，元素的存储顺序不固定。
- **元素是否可以重复**：**否**，集合中的元素是唯一的，重复的元素会被自动去除。
- **能否更改元素**：**否**，集合中的元素必须是不可变的（例如，元组可以作为集合的元素，但列表不能）。
- **查询效率**：**O(1)**，由于哈希表的特性，查找一个元素的时间复杂度是常数级别。
- **增删效率**：**O(1)**，添加和删除操作也通常是常数时间复杂度，哈希冲突的影响较小。

`frozenset`

- **底层实现**：与 `set` 类似，`frozenset` 也是基于哈希表实现的，但是它是 **不可变的。**
- **是否有序**：无序。
- **元素是否可以重复**：**否。**
- **能否更改元素**：**否**，`frozenset` 是不可变的，因此不支持对其进行修改。

`frozenset` 主要用于那些需要集合行为，但又不能更改集合内容的场景，例如，作为字典的键。

2. 哈希映射（Dictionary）

Python 的字典（`dict`）是最常用的哈希映射数据结构，底层也是使用 **哈希表** 来实现的。它允许我们将一个 **键（key）** 映射到一个 **值（value）**。

`dict`

- **底层实现**：基于 **哈希表**，支持快速查找、插入和删除。
- **是否有序**：从 Python 3.7 版本开始，字典保持 **插入顺序。**这意味着，当你以特定的顺序插入元素时，元素的顺序会被保留（但它不保证按键的大小顺序）。
- **键是否可以重复**：**否**，字典中的键必须是唯一的。如果重复插入键，后插入的键值对会覆盖前一个键值对。
- **值是否可以重复**：**是**，字典中的值可以重复。
- **能否更改元素**：**是**，你可以修改字典中的值，也可以添加或删除键值对。
- **查询效率**：**O(1)**，查找一个键对应的值在哈希表中是常数时间复杂度。
- **增删效率**：**O(1)**，插入和删除操作的时间复杂度通常也是常数级别。

3. 哈希映射（OrderedDict）

`OrderedDict` 是 Python 标准库 `collections` 模块中的一个类，它的行为和普通字典类似，但它的一个特别之处在于 **保持插入顺序**，即使在删除键值对后，插入顺序依然会被保留。

`OrderedDict`

- **底层实现**：与 `dict` 类似，也是基于哈希表，但会额外维护一个顺序链表来保持插入顺序。
- **是否有序**：有序，始终按照插入的顺序保存键值对。
- **键是否可以重复**：**否**，字典中的键必须是唯一的。
- **值是否可以重复**：**是**，字典中的值可以重复。
- **能否更改元素**：**是**，和普通字典一样，你可以修改值、插入或删除键值对。
- **查询效率**：**O(1)**，和普通字典一样，查找操作是常数时间复杂度。
- **增删效率**：**O(1)**，插入和删除的效率也是常数时间复杂度。

应用场景：
`OrderedDict` 适用于那些需要保持插入顺序的场景。例如，在一些场合下，你可能需要在字典中按插入顺序进行迭代，或者需要删除并重新插入某个键值对时保留顺序。

比较：Set vs Dict

特性	Set	Dict
底层实现	哈希表	哈希表
有序性	无序	有序（Python 3.7+）
元素是否可以重复	否	否（键不可重复）
能否更改元素	否	是（可以修改值）
查询效率	**O(1)**	**O(1)**
增删效率	**O(1)**	**O(1)**
适用场景	集合去重、集合运算	存储键值对、快速查找

总结:

1. **集合（Set）**：是一个不允许重复元素的无序集合，底层使用哈希表。主要用于去重和快速查找。
2. **字典（Dict）**：是一个键值对数据结构，其中键唯一，值可以重复。底层也是使用哈希表，支持快速查找、插入和删除操作。Python 3.7+ 保证字典保持插入顺序。
3. `OrderedDict`：与普通字典相似，但它额外保证了插入顺序。适用于需要按插入顺序处理键值对的场景。

总结

总结一下，**当我们遇到了要快速判断一个元素是否出现集合里的时候，就要考虑哈希法。**

但是哈希法也是**牺牲了空间换取了时间**，因为我们要使用额外的数组，set或者是map来存放数据，才能实现快速的查找。

如果在做面试题目的时候遇到需要判断一个元素是否出现过的场景也应该第一时间想到哈希法！