## 哈希表

### Hash

一般翻译做散列、杂凑，或音译为哈希，是把任意长度的[输入](https://baike.baidu.com/item/%E8%BE%93%E5%85%A5/5481954" \t "https://baike.baidu.com/item/Hash/_blank)（又叫做预映射pre-image）通过散列算法变换成固定长度的[输出](https://baike.baidu.com/item/%E8%BE%93%E5%87%BA/11056752" \t "https://baike.baidu.com/item/Hash/_blank)，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来确定唯一的输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的[消息摘要](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E6%91%98%E8%A6%81/4547744" \t "https://baike.baidu.com/item/Hash/_blank)的函数。

### [哈](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E8%A1%A8/10027933" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)希表概念

Hash table，也叫[散列表](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E8%A1%A8/10027933" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)，是根据关键码值(Key value)而直接进行访问的[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/1450" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做[散列函数](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0/2366288" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)，存放记录的[数组](https://baike.baidu.com/item/%E6%95%B0%E7%BB%84/3794097" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)叫做[散列表](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E8%A1%A8/10027933" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)。给定表M，存在函数f(key)，对任意给定的关键字值key，代入函数后若能得到包含该关键字的记录在表中的地址，则称表M为哈希(Hash）表，函数f(key)为哈希(Hash) 函数。

### 哈希表优点

在实现编程中，常常面临着两个问题：存储和查询。存储和查询的效率往往决定了整个程序的效率。而我们常见存储数据的数据结构比如线性表，树等。数据在结构中的位置都是不明确的，当我们在这些数据结构中要查询一个数据，都避免不了的执行查询算法，去遍历数据结构，拿关键字和结构中的数据一一比较，从而得到想要的数据。整个查询的效率就决定了程序的执行效率，查询效率又依赖查找过程中所进行的比较次数。所以我们就希望能不能不通过比较就能获得我们想要的结果呢?答案是有的，不经过任何比较，一次存储便能取得所查记录。但这就必须在记录的存储位置和它的关键字之间建立一个确定的对应关系f。使得每个关键字和结构中一个唯一的存储位置相对应。这个关系就是我们所说的哈希函数f（x）。在这个思想上建立起来的表就称为哈希表。

### 哈希表理解

关键字(Key):哈希表是通过一个信息来查找另一个信息，将这两个信息在哈希表从形成映射关系，而关键字则是我们要提供的信息

值(Value):值是我们想要获取到的信息，如map.put(key,value)

哈希函数:f(x)哈希函数是用来构成哈希表的工具，也是哈希表的核心思想，是关键字和对应数据的存储位置的一个映射关系，通过把关键字代入哈希函数中进行计算，可以得到关键字所对应的数据在哈希表中的存储位置即哈希地址。

哈希地址:=f(key)哈希地址记录的是我们所需要的数据在哈希表中的存储位置，哈希地址只是表示查找表中的存储位置，不是实际的物理存储位置

关键字、值、哈希函数、哈希地址、哈希表之间的关系

哈希表是通过哈希函数来构建的，我们把哈希函数想象成数学中的函数f()。而函数中的X就是关键字Key，既f(x)。然后将关键字带入一个公式中，如f（x） = x \* 2 + 3 。然后经过运算就可以求出一个值，这个值表示要查询的数据(Value)在哈希表中的存储位置，也就是哈希地址。而记录这整个Key-Value信息的表就是哈希表。

我们现在把一个抽象的哈希表具体表现成下面的形式，可以说下面就是一个哈希表。我们把它想象成一本电话簿，该电话簿用拼音首字母来区分数据，数据记录了具体的人名和其电话号码。换成哈希表的概念，就是说ABCD是哈希地址，用于记录值所存储的位置。人名为查询关键字(Key)。电话号码是我们需要获得的数据(Value)。

为了简化流程且易懂，我们这里假设了哈希地址就是关键字的拼音首字母大写，那么哈希函数就是f（艾力）= A，f(包三) = B等…。我们通过将关键字(包三)代入计算公式(哈希函数)中，得到包三的电话号码所在的位置，既哈希地址(B)。那我们就能通过位置直接获得我们想要的数据，而不需要遍历比较。

| 哈希地址 | Key and Value |
| --- | --- |
| A | 艾力 13912345678 |
| B | 包三 15823457890 |
| C | 成五 15823457890 |

### 哈希冲突

哈希冲突就是key1!=key2.但key1和key2所对应的数据的存储位置都一致，既哈希地址一致。这就是哈希冲突。我们看来下面的哈希表：

| 哈希地址 | Key and Value |
| --- | --- |
| A | 艾力 13912345678 ；爱丽丝 15823787890 |
| B | 包三 15823457890 |

如上表我们可以看到，艾力和爱丽丝两个人的拼音首字母大写都是A,这就意味着这两个人的电话号码都存储在哈希表(电话簿)的A区，这就存在冲突了。所以在构建哈希表中，最为重要的核心就是哈希函数的设计。哈希冲突只能尽可能减少，是不能完全避免的

处理冲突

1. [开放寻址法](https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E5%AF%BB%E5%9D%80%E6%B3%95" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)：Hi=(H(key) + di) MOD m,i=1,2，…，k(k<=m-1），其中H(key）为[散列函数](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)，m为[散列表](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E8%A1%A8" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)长，di为增量序列，可有下列三种取法：

1.1. di=1,2,3，…，m-1，称线性探测再散列；

1.2. di=1^2,-1^2,2^2,-2^2，⑶^2，…，±（k)^2,(k<=m/2）称二次探测再散列；

1.3. di=[伪随机数](https://baike.baidu.com/item/%E4%BC%AA%E9%9A%8F%E6%9C%BA%E6%95%B0" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)序列，称伪随机探测再散列。

2. 再[散列法](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E6%B3%95" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)：Hi=RHi(key),i=1,2，…，k RHi均是不同的[散列函数](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)，即在同义词产生地址冲突时计算另一个散列函数地址，直到冲突不再发生，这种方法不易产生“聚集”，但增加了计算时间。

3. 链地址法（拉链法）,java中的HashMap就是采用这种方法

4. 建立一个公共溢出区

### 哈希函数构造方法

如何判断一个哈希函数的优劣：

* 能否将关键字均匀影射到哈希空间上
* 有无好的解决冲突的方法
* 计算哈希函数是否简单高效

常见构建哈希函数的六个方法

* 直接定值法：即H(key)=key或H(key) = a·key + b
* 数字分析法：找出数字的规律，尽可能利用这些数据来构造冲突几率较低的散列地址
* 平方取中法：当无法确定关键字中哪几位分布较均匀时，可以先求出关键字的平方值，然后按需要取平方值的中间几位作为哈希地址
* 折叠法：将关键字分割成位数相同的几部分，最后一部分位数可以不同，然后取这几部分的叠加和（去除进位）作为散列地址
* 除留余数法：选择一[随机函数](https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/_blank)，取关键字的随机值作为散列地址，通常用于关键字长度不同的场合。
* 随机数法：f (key) = random (key)

java中Object的hashCode就是一个哈希函数，方法是本地方法，具体用的什么哈希函数综合了上面几种算法，String、Integer等则重写了hashCode，可以查看源码得知。