45 | 哈希表与哈希算法：哈希表适合用在什么样的情景?

2023-05-26 王健伟 来自北京

《快速上手C++数据结构与算法》

你好，我是王健伟。

这节课我们来看一个新的数据结构——哈希表。

哈希表也叫散列表或 Hash 表，是由数组演化来的一种扩展，用于存放数据。哈希表跟数组一样，支持查询、插入、删除等操作，但哈希表的表现尤其突出，因为它大大降低了查找数据所消耗的时间。

它是怎么做到这一点的呢？这节课，我就先带你一起了解哈希表。

基本概念、适用范围及范例

哈希表不适合范围查找，不适合排序、找最大值最小值的情形，也不适合用某一个条件一次性找出一堆记录的情形，比如用性别为“男”搜索一个班级的学生。哈希表适合用诸如学生身份

证号或学号来快速找到某一位学生的情形。也就是说，**哈希表非常适合对于查找性能要求高，查找的数据之间彼此没有关系的场合。**

以一个日常生活中的例子来说明哈希表。假设有一个小卖店，店里有 500 种不同的商品，每种商品的价格也各不相同。我们如何通过商品名称获取商品价格信息呢？传统的方法是创建一个商品数组来记录商品名称和对应的价格信息。当顾客询问某种商品的价格时，可以根据商品名称从数组的开始位置向下搜索，当搜索到该名称的商品时，也就获取到了该商品的价格。

上述方法查询商品的价格需要花费大量时间，但是人们总是希望能够根据商品的名称立刻获取到商品的价格。其实这并不难，利用一个被称为“哈希”的函数（哈希函数）就可以做到。

哈希函数当然是由程序员自己实现，不过这里我们暂时不讨论哈希函数如何实现，而是先看一看哈希函数能够实现什么功能：**当把不同的商品名作为参数传递给哈希函数时，哈希函数会返回一个整数。**

一般来讲，人们完全无法看出传递给哈希函数的参数和哈希函数返回的整数之间有什么对应的规律。但是有两点是可以确定的。

1. **传递给哈希函数不同的参数（不同的商品名称），哈希函数肯定会返回不同的整数。**例如用“苹果”做参数传递给哈希函数，哈希函数返回的整数是 11，而用“鸭梨”做参数传递给哈希函数，哈希函数返回的是整数是 23，绝不可能也返回整数 11。
2. **如果传递给哈希函数的参数相同，那么返回的一定是同一个整数。**比如无论何时，传递给函数的参数是“苹果”，那么哈希函数返回的整数一定是 11。

回到小卖店的例子，有了哈希函数，小卖店中根据商品名称获取商品价格信息就会非常容易。我们来尝试把这件事翻译成技术语言描述一下，就是下面这样。

首先，因为店里有 500 种不同商品，所以可以定义一个长度为 500 的价格数组，用于存放 500 种不同商品的价格信息。这个数组下标能够取值的范围是 0~499。

其次，给定一个商品名称比如“苹果”，使用哈希函数返回一个数字（当然要保证哈希函数返回的数字在 0~499 之间），比如返回的是 11，于是，在下标为 11 的价格数组位置，保

存“苹果”的价格比如 8.9。再给定一个商品比如“猕猴桃”，哈希函数返回的数字是 209，于是，在下标为 209 的价格数组位置，保存“猕猴桃”的价格比如 15.8。

如此反复，就可以把 500 种商品的价格都保存到价格数组中。

最后，当需要查询某个商品的价格时，以“商品名”为参数调用哈希函数返回一个数字，然后以这个数字为下标到价格数组中就可以立即取得商品价格。

“哈希”这个词是英文单词 Hash 的音译，Hash 翻译成中文的意思是“把……弄乱”的意思。所以哈希函数也被称为**散列函数**或者**杂凑函数**。它的功能已经显而易见——根据输入数据 / 原始数据（包括但是不限于字符串，比如也可以是数字等），生成一个整数。借助哈希函数，也就实现了数据（关键字）到存储位置的一对一映射关系。这样就可以把数据的各种属性信息放入一个数组或一张表中，比如数据库中的一张学生表，这样的表称为哈希表。上面例子中的“价格数组”显然就是一个哈希表。

好了，最后我们用相对书面化的词汇描述一下各种概念：传递给哈希函数的参数（商品名称）称为关键字 / 键（Key），在这里用来标识一件商品。把键转化为数组下标的函数就是哈希函数 / 散列函数。执行哈希函数得到的数值称为哈希值 / 散列值 /Hash 值。

参考图 1 所示：

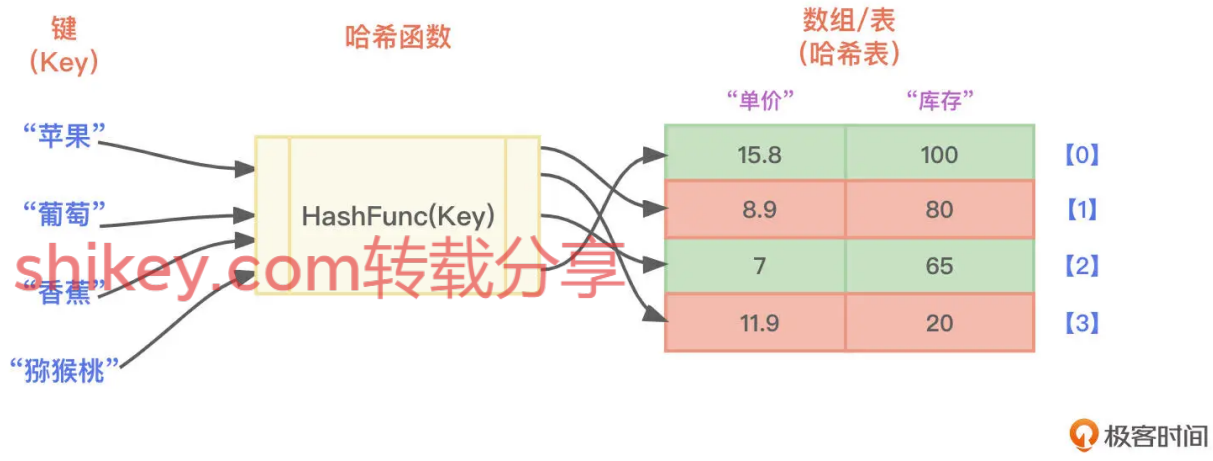

图1 键、哈希函数、哈希表 示意图

从图 1 中可以看到，哈希表中可以保存针对商品的多种信息，比如商品价格、商品库存数量等，把商品名称（键）存进去也是可以的。上述范例可以看作是根据给定的关键字计算出关键字在哈希表中的地址（数组下标），换句话说，上述的范例的哈希表建立了关键字和存储地址之间的一种直接映射关系。

哈希函数、哈希算法简介

不难看出，哈希函数在哈希表的建立中起着至关重要的作用。哈希函数是一个函数，其形参（输入）是一个键，其输出是一个哈希值。

那**哈希函数**怎么设计、实现呢？一般应该遵循几点要求：

1. 哈希函数的计算结果（哈希值）是一个非负整数，因为要对应数组下标，而数组下标是从 0 开始的。
2. 如果传递给哈希函数的参数相同，那么返回的哈希值一定是同一个整数。
3. 如果传递给哈希函数不同的参数，那么返回的哈希值是不同的整数。

上述三项要求中，第三项要求虽然看似合理，但很难做到，常用的哈希算法比如 CRC、MD5、SHA 等，也无法做到不同的参数返回的一定是不同的整数。

哈希函数中会用到**哈希算法**。将任意长度的二进制值映射为较短的固定长度的二进制值（哈希值），这个映射规则就是哈希算法。设计一个好的哈希算法并不是一件容易的事，要综合考虑的因素比较多，比如关键字长度、散列表大小等，一般来说：

1. 哈希算法应该得到单向哈希值。这意味着通过哈希值无法反推出原始数据（键）。
2. 一点微小的对原始数据的修改（哪怕只有一位数据），也会导致得到的哈希值大不相同。
3. 生成的哈希值要尽可能随机且均匀地分布。如果出现传递给哈希函数不同的参数，但返回的哈希值是相同的整数，就叫做**哈希冲突**。比如上述如果定义的是长度为 100 的价格数组却要保存 500 种不同商品的价格信息，这种要保存的商品数量比存储容量大的情形肯定会产生哈希冲突。

4. 算法不能太复杂，执行效率要高，即便原始数据量庞大，也要在尽可能短的时间内计算出哈希值。

哈希算法各式各样，这意味着哈希函数的设计方法有很多，比如直接定址法、除留余数法、数字分析法、平方取中法、折叠法、减去法、基数转换法、随机乘数法、字符串哈希法（将字符串映射为哈希值：前面举的商店的例子就是把商品名映射为一个哈希值）、旋转法、伪随机数法等。

这里我简单说一下其中的一些设计方法，对于没有提到的，可以通过搜索引擎了解。

直接定址法

直接取关键字的某个线性函数值为哈希值，哈希函数为 HashFunc(Key)=a*Key+b。其中 a 和 b 是常数。这样的哈希函数简单、均匀，也不会产生哈希冲突。因为只要 Key 值不同，得到的哈希值一定不同。但需要事先知道关键字的分布情况，所以并不常用。

除留余数法

这是最常用的构造哈希函数的方法。假定哈希表长为 m，取一个不大于 m 但最接近或等于 m 的数 p。这里的 p 一般可以为一个素数，素数就是除了 1 和它本身，不能被其他数整除的数。然后，利用公式“HashFunc(Key)=key%p”把关键字转换成哈希值。注意这里的 % 指的是取余（取模）的意思。另外，此方法不但可以对关键字直接取余，也可以在采用后续介绍的“折叠法”“平方取中法”后再取余。

这个方法的关键是要选好 p，比如上述提出的 p 一般为一个素数，让每个关键字通过哈希函数计算后能够比较等概率地分布在哈希表中，进而尽量减少哈希冲突。

试想一种极端的情况，比如一组数字：12、24、36、48、60，如果选择 p=12，则哈希值全部会为 0，这简直太糟糕了，哈希冲突严重。但如果选择 p=11，则就大大减少了哈希冲突的可能性。

数字分析法

设关键字是 r 进制（比如十进制），而该 r 进制数在各个位上出现的频率不一定相同。可能在某些位上分布均匀一些，因为每个数字出现的机会相对均等；在某些位上分布不均匀，因为只有某几个数字经常出现。这个时候，就应该选取数字分布均匀的若干个位作为哈希值。这种方法适合于关键字集合是已知的情形。

比如某公司员工号码薄如下：

```
张三　139XXXX5628
李四　139XXXX3791
王五　138XXXX8473
赵六　138XXXX1956
```

可以看到，手机前 7 位号码可能会比较相似（分布不均匀），而最后 4 位数字比较随机，分布相对要均匀一些。所以选择后 4 位号码作为哈希值比较好，哈希冲突的机率会比较小。

平方取中法

取关键字的平方值的中间几位作为哈希值。具体取中间几位依实际情况而定。这个方法得到的哈希值与关键字的每一位都有关系，哈希值分布会比较均匀。

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短一些），取得这几部分的叠加和作为哈希值。当关键字位数很多且关键字中每一位的数字分布比较均匀时可以采用此法。

比如，如果关键字是 1234567890，可以将关键字设置为 3 位一组一共 4 组即 123、456、789、0，然后将这 4 组叠加求和即 123+456+789+0=1368，取后面 3 位即 368 作为哈希值。

在实际开发中，可以根据不同情况采用不同的哈希函数，比如可以考虑如下因素：计算时间、关键字长度、哈希表尺寸、关键字分布情况、记录查找的频率等。

C++ 标准库中有一些关联容器比如 hash_set、hash_map、hash_multiset、hash_multimap 等用到了哈希表，当然，这些关联容器有些过时了，目前已经推荐改用如下这些依旧用到了哈希表的无序容器 unordered_set、unordered_map、unordered_multimap。有兴趣可以通过搜索引擎对这些容器进行详细了解。

小结

本节我带你学习了哈希表的概念，提出了哈希表的适用范围。要记得，哈希表对于查找性能要求高，查找的数据之间彼此没有关系的场合非常合适。

那什么是哈希函数呢？哈希函数根据输入的数据生成一个整数。借助哈希函数，可以实现数据到存储位置的一对一映射关系，以便把数据的各种属性信息记录到一张表中，这张表就是哈希表。

接下来，我详细介绍了哈希函数的设计和实现所要遵循的要求。哈希函数的设计会用到哈希算法这种将任意长度的二进制值映射为较短固定长度的二进制值的映射规则。

哈希函数的设计方法有很多，我们重点谈到了直接定址法、除留余数法、数字分析法、平方取中法、折叠法。

当传递给哈希函数不同的参数，但返回的哈希值相同时，就产生了哈希冲突。哈希冲突当然不是好事，但却无法完全避免。下节课，我们来聊聊解决哈希冲突的方法。

思考题

在这节课的最后，我也给你留了两道复习思考题。

1. 什么是哈希表，如何实现哈希表？
2. 哈希算法的概念是什么？常见的哈希算法有哪些，分别有什么优缺点？

欢迎你在留言区和我互动。如果觉得有所收获，也可以把课程分享给更多的朋友一起学习。我们下节课见！

精选留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。