哈希算法

哈希

前言：什么是哈希？哈希冲突如何解决？python中的字典和java的hashmap有什么区别？通过本文，我们一起学习哈希算法

什么是哈希

哈希算法：根据设定的哈希函数H（key）和处理冲突方法将一组关键字映象到一个有限的地址区间上的算法。也称为散列算法、杂凑算法。

哈希表：数据经过哈希算法之后得到的集合。这样关键字和数据在集合中的位置存在一定的关系，可以根据这种关系快速查询。

什么是哈希冲突

由于哈希算法被计算的数据是无限的，而计算后的结果范围有限，因此总会存在不同的数据经过计算后得到的值相同，这就是哈希冲突。

哈希冲突解决的方法

哈希冲突解决方法分为以下3个大类：

开放定址法

开放定址法一般有如下3个方案，哈希函数：Hi=（H（key）+di）% m　i=1，2，…，n，

1.线性探测

di=i

发生hash冲突时，顺序查找下一个位置，直到找到一个空位置（固定步长1探测）

2.平方探测

di = ± i2(+12, −12, +22, −22……)

在发生hash冲突时，在表的左右位置进行按一定步长跳跃式探测（固定步长n探测）

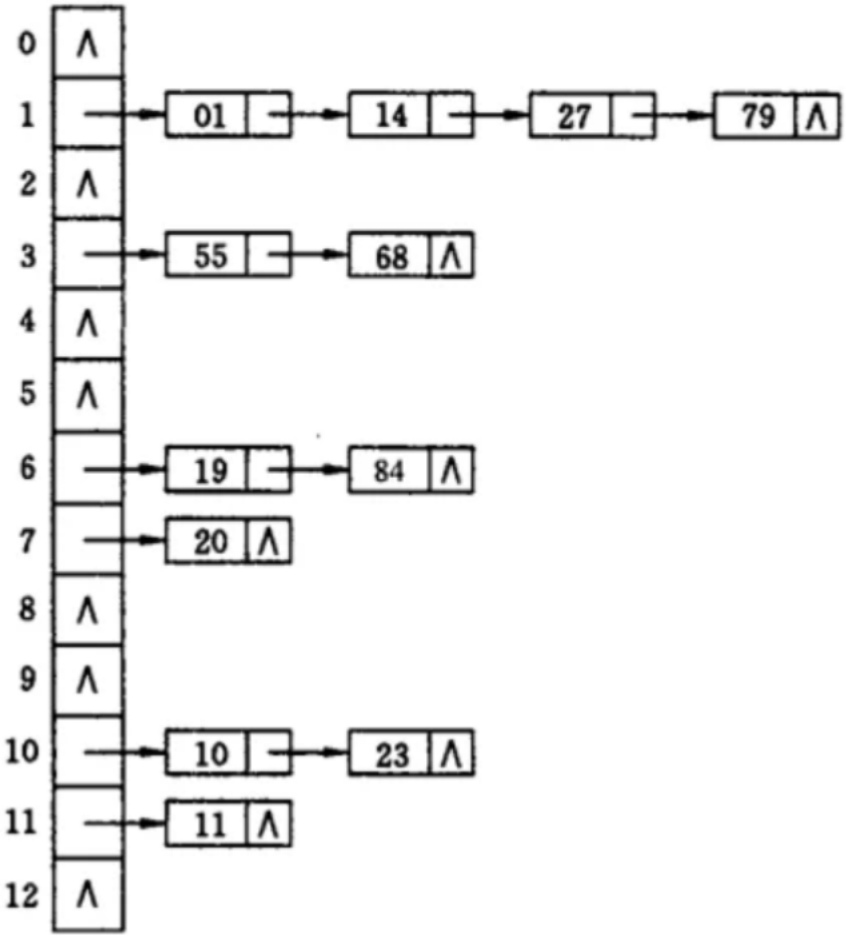
3.伪随机探测

di=random

在发生hash冲突时，根据公式生成一个随机数，作为此次探测空位置的步长（随机步长n探测）

链地址法(拉链法)

在出现冲突的地方存储一个链表，所有的同义词记录都存在其中。形象点说就行像是在出现冲突的地方直接把后续的值摞上去。



再哈希

Hi=RH1（key） i=1，2，…，k

当哈希地址Hi=RH1（key）发生冲突时，再计算Hi=RH2（key）……，直到冲突不再产生。这种方式是同时构造多个哈希函数，当产生冲突时，计算另一个哈希函数的值。这种方法不易产生聚集，但增加了计算时间。主要用于加密使用。

开放地址法与链地址法的比较

所谓比较，算是对前面的一个总结：

No	链地址法	开放地址法
1	易于实现	需要更多的计算
2	使用链地址法，哈希表永远不会填充满，不用担心溢出。	哈希表可能被填满，需要通过拷贝来动态扩容。
3	对于哈希函数和装载因子不敏感	需要额外关注如何规避聚集以及装载因子的选择。
4	适合不知道插入和删除的关键字的数量和频率的情况	适合插入和删除的关键字已知的情况。
5	由于使用链表来存储关键字，缓存性能差。	所有关键字均存储在同一个哈希表中，所以可以提供更好的缓存性能。
6	空间浪费（哈希表中的有些链一直未被使用）	哈希表中的所有槽位都会被充分利用。
7	指向下一个结点的指针要消耗额外的空间。	不存储指针。

python的字典

代码示例

Python | 复制代码

```python
hashmap = {}
hashmap['a'] = 1
hashmap['c'] = 2
hashmap['b'] = 3
print(hashmap)
#输出 {'a': 1, 'c': 2, 'b': 3}
```

python的字典是保序字典，输出的key顺序和添加顺序一致。python字典是采用的开放定址法解决的哈希冲突，根据网上资料显示，使用的是伪随机探测。

java的hashmap详解

强力推荐：美团技术团队hashmap

代码示例

```java
HashMap<String, Integer> hashMap = new HashMap<>();
hashMap.put("a", 1);
hashMap.put("c", 3);
hashMap.put("b", 2);
System.out.println(hashMap);
//输出 {a=1, b=2, c=3}
```

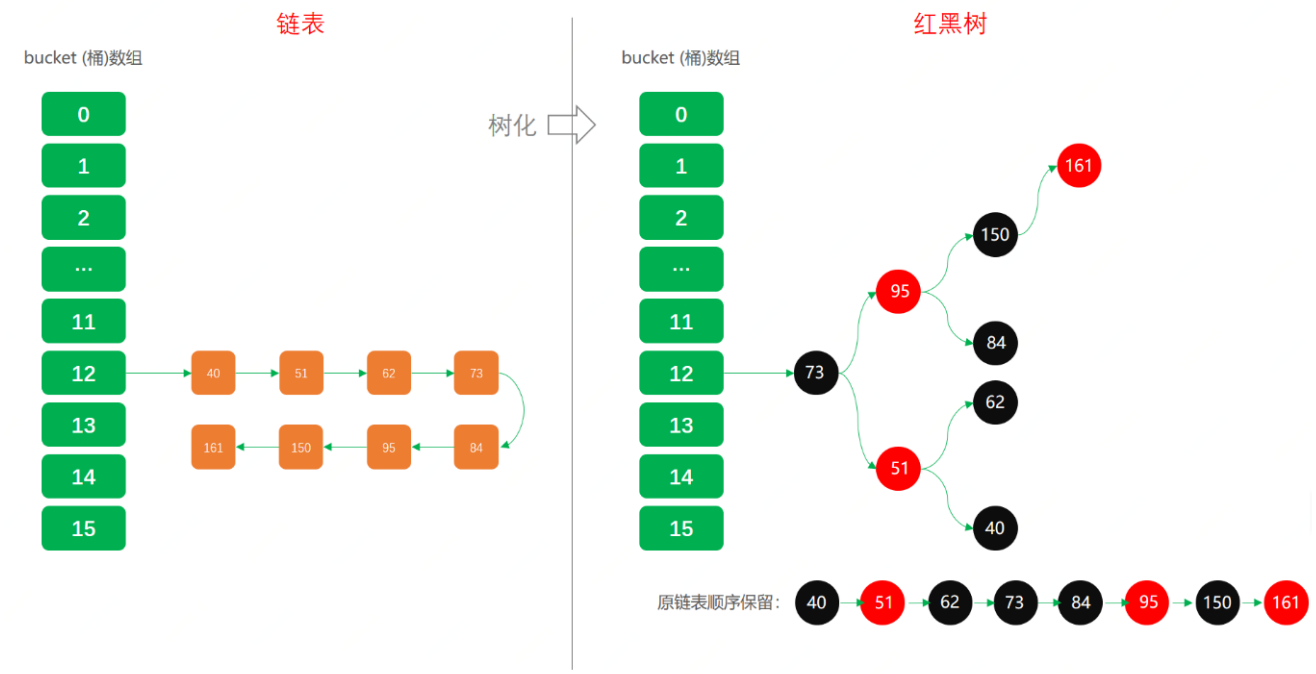
java的hashmap并不是保序的。

底层实现

实现：数组+链表

优化：数组+红黑树

链表转红黑树，如下图；



1.null值可以做为hashmap的键

2.哈希冲突解决：拉链法

3.扰动函数：增加随机性，让数据更加均衡散列，减少哈希碰撞

初始化容量：

Java | 复制代码

```java
static final int tableSizeFor(int cap) {
        int n = cap - 1;
        n |= n >>> 1;
        n |= n >>> 2;
        n |= n >>> 4;
        n |= n >>> 8;
        n |= n >>> 16;
        return (n < 0) ? 1 : (n >= MAXIMUM_CAPACITY) ? MAXIMUM_CAPACITY : n + 1;
    }
```

这段代码的作用是，计算初始化容量，看着复杂，实际上很简单：比如我们设置cap为17，那么n就是17-1=16，二进制为，10000，后续算法的作用就是把剩下的0置为1，那么得到的返回值为11111->31，然后得到31+1=32。

4.负载因子：默认值为0.75f，负载因子决定了数据量达到多少以后就需要进行扩容；负载因子越小，哈希碰撞越小，开辟空间越大；负载因子越大，哈希碰撞越大，开辟空间小。计算公式为：临界值(threshold)=负载因子(loadFacotr)*容量(cap)

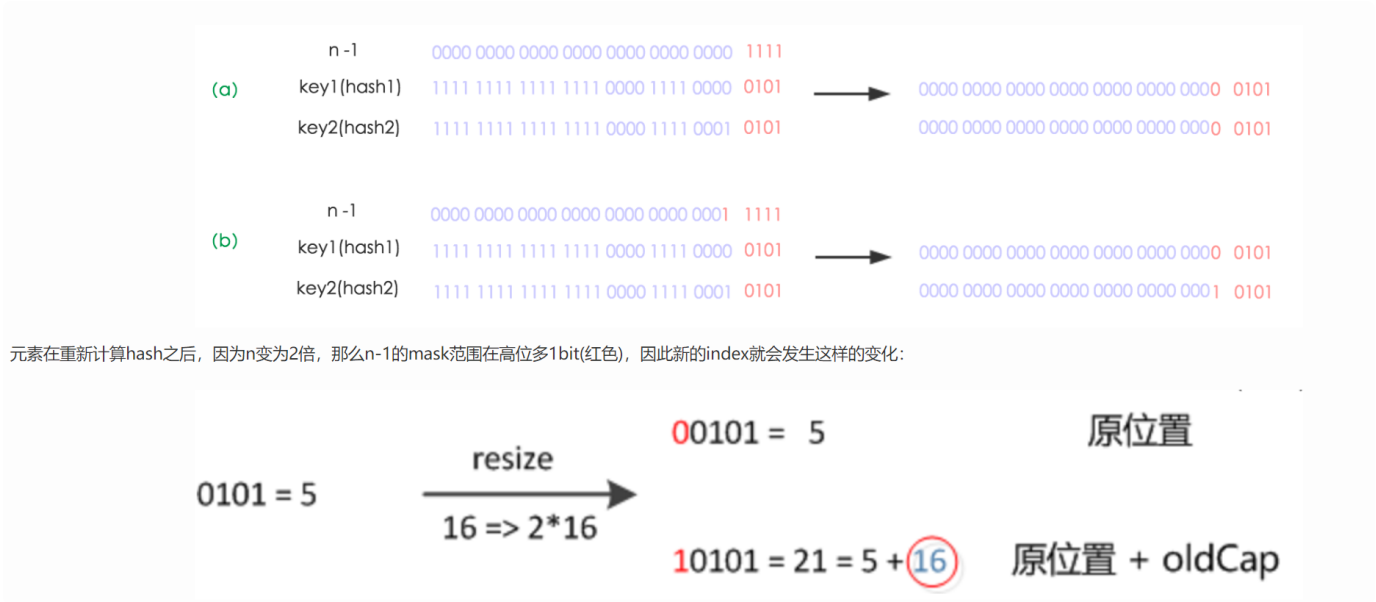
5.扩容：默认初始容量为16，每次扩容为2的倍数。JDK8进行了扩容优化，方法是：如果原来的哈希值新增的bit为0，索引就不变，如果为1，则索引为：原索引+oldCap，即以前的：索引+以前的容量。

确定哈希桶数组索引位置

Java | 复制代码

```java
static final int hash(Object key) {   //jdk1.8 & jdk1.7
     int h;
     // h = key.hashCode() 为第一步 取hashCode值
     // h ^ (h >>> 16)  为第二步 高位参与运算
     return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

重点：扩容前，key1和key2的返回值都是0101;扩容后，key1返回值00101,key2返回值10101



根据上面的方法，key1的索引不变，key2的索引为5+16=21

6.链表转红黑树：当单个链表的节点个数超过8，就转换成红黑树进行存储；如果红黑树节点小于8，退化成链表

7.红黑树的优点：查找、插入、删除的时间复杂度都为O(logn)，对于随机插入的数据源，平衡性好

8.插入：链表采用的尾插法；

9.hashmap非线性安全，涉及到多线程时，使用concurrentHashmap

参考

https://cloud.tencent.com/developer/article/1672781

https://jishuin.proginn.com/p/763bfbd2ce15

https://tech.meituan.com/2016/06/24/java-hashmap.html

https://bugstack.cn/md/algorithm/data-structures/2022-08-27-hash-table.html