## Hash冲突

[**https://www.cnblogs.com/wuchaodzxx/p/7396599.html**](https://www.cnblogs.com/wuchaodzxx/p/7396599.html)

## 为什么HashMap是线程不安全的，实际会如何体现？

第一，如果多个线程同时使用put方法添加元素

假设正好存在两个put的key发生了碰撞(hash值一样)，那么根据HashMap的实现，这两个key会添加到数组的同一个位置，这样最终就会发生其中一个线程的put的数据被覆盖。

第二，如果多个线程同时检测到元素个数超过数组大小\*loadFactor

这样会发生多个线程同时对hash数组进行扩容，都在重新计算元素位置以及复制数据，但是最终只有一个线程扩容后的数组会赋给table，也就是说其他线程的都会丢失，并且各自线程put的数据也丢失。且会引起死循环的错误。

## Hash冲突解决

1. [开放定址法](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_0)
   1. [线性探测再散列](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_0_1)
   2. [二次探测再散列](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_0_2)
   3. [伪随机探测再散列](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_0_3)
2. [再哈希法](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_1)
3. [链地址法](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_2)
4. [建立公共溢出区](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_3)
5. [优缺点](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_4)
   1. [开放散列（open hashing）/ 拉链法（针对桶链结构）](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_4_1)
   2. [封闭散列（closed hashing）/ 开放定址法](https://www.cnblogs.com/wuchaodzxx/p/7396599.html#H1_4_2)

# 开放定址法

这种方法也称再散列法，其基本思想是：当关键字key的哈希地址p=H（key）出现冲突时，以p为基础，产生另一个哈希地址p1，如果p1仍然冲突，再以p为基础，产生另一个哈希地址p2，…，直到找出一个不冲突的哈希地址pi ，将相应元素存入其中。这种方法有一个通用的再散列函数形式：

Hi=（H（key）+di）% m   i=1，2，…，n

其中H（key）为哈希函数，m 为表长，di称为增量序列。增量序列的取值方式不同，相应的再散列方式也不同。主要有以下三种：

## 线性探测再散列

dii=1，2，3，…，m-1

这种方法的特点是：冲突发生时，顺序查看表中下一单元，直到找出一个空单元或查遍全表。

## 二次探测再散列

di=12，-12，22，-22，…，k2，-k2    ( k<=m/2 )

这种方法的特点是：冲突发生时，在表的左右进行跳跃式探测，比较灵活。

## 伪随机探测再散列

di=伪随机数序列。

具体实现时，应建立一个伪随机数发生器，（如i=(i+p) % m），并给定一个随机数做起点。

例如，已知哈希表长度m=11，哈希函数为：H（key）= key  %  11，则H（47）=3，H（26）=4，H（60）=5，假设下一个关键字为69，则H（69）=3，与47冲突。

如果用线性探测再散列处理冲突，下一个哈希地址为H1=（3 + 1）% 11 = 4，仍然冲突，再找下一个哈希地址为H2=（3 + 2）% 11 = 5，还是冲突，继续找下一个哈希地址为H3=（3 + 3）% 11 = 6，此时不再冲突，将69填入5号单元。

如果用二次探测再散列处理冲突，下一个哈希地址为H1=（3 + 12）% 11 = 4，仍然冲突，再找下一个哈希地址为H2=（3 - 12）% 11 = 2，此时不再冲突，将69填入2号单元。

如果用伪随机探测再散列处理冲突，且伪随机数序列为：2，5，9，……..，则下一个哈希地址为H1=（3 + 2）% 11 = 5，仍然冲突，再找下一个哈希地址为H2=（3 + 5）% 11 = 8，此时不再冲突，将69填入8号单元。

# 再哈希法

这种方法是同时构造多个不同的哈希函数：

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。这种方法不易产生聚集，但增加了计算时间。

# 链地址法

这种方法的基本思想是将所有哈希地址为i的元素构成一个称为同义词链的单链表，并将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。

# 建立公共溢出区

这种方法的基本思想是：将哈希表分为基本表和溢出表两部分，凡是和基本表发生冲突的元素，一律填入溢出表。

# 优缺点

## 开放散列（open hashing）/ 拉链法（针对桶链结构）

1）优点： ①对于记录总数频繁可变的情况，处理的比较好（也就是避免了动态调整的开销） ②由于记录存储在结点中，而结点是动态分配，不会造成内存的浪费，所以尤其适合那种记录本身尺寸（size）很大的情况，因为此时指针的开销可以忽略不计了 ③删除记录时，比较方便，直接通过指针操作即可

2）缺点： ①存储的记录是随机分布在内存中的，这样在查询记录时，相比结构紧凑的数据类型（比如数组），哈希表的跳转访问会带来额外的时间开销 ②如果所有的 key-value 对是可以提前预知，并之后不会发生变化时（即不允许插入和删除），可以人为创建一个不会产生冲突的完美哈希函数（perfect hash function），此时封闭散列的性能将远高于开放散列 ③由于使用指针，记录不容易进行序列化（serialize）操作

## 封闭散列（closed hashing）/ 开放定址法

1）优点： ①记录更容易进行序列化（serialize）操作 ②如果记录总数可以预知，可以创建完美哈希函数，此时处理数据的效率是非常高的

2）缺点： ①存储记录的数目不能超过桶数组的长度，如果超过就需要扩容，而扩容会导致某次操作的时间成本飙升，这在实时或者交互式应用中可能会是一个严重的缺陷 ②使用探测序列，有可能其计算的时间成本过高，导致哈希表的处理性能降低 ③由于记录是存放在桶数组中的，而桶数组必然存在空槽，所以当记录本身尺寸（size）很大并且记录总数规模很大时，空槽占用的空间会导致明显的内存浪费 ④删除记录时，比较麻烦。比如需要删除记录a，记录b是在a之后插入桶数组的，但是和记录a有冲突，是通过探测序列再次跳转找到的地址，所以如果直接删除a，a的位置变为空槽，而空槽是查询记录失败的终止条件，这样会导致记录b在a的位置重新插入数据前不可见，所以不能直接删除a，而是设置删除标记。这就需要额外的空间和操作。