# Java 集合：HashMap（put方法的实现与哈希冲突） #19

New issue

Open **pzxwhc** opened this issue on Mar 18, 2016 · 9 comments

**pzxwhc** commented on Mar 18, 2016 • edited Owner

## HashMap 概念

对于 Map ，最直观就是理解就是键值对，映射，key-value 形式。一个映射不能包含重复的键，一个键只能有一个值。平常我们使用的时候，最常用的无非就是 HashMap。

HashMap 实现了 Map 接口，允许使用 **null** 值 和 **null** 键，并且不保证映射顺序。

HashMap 有两个参数影响性能：

- 初始容量：表示哈希表在其容量自动增加之前可以达到多满的一种尺度
- 加载因子：当哈希表中的条目超过了容量和加载因子的乘积的时候，就会进行重哈希操作。

如下成员变量源码：

```
static final float DEFAULT_LOAD_FACTOR = 0.75f;
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4;
transient Node<K,V>[] table;
```

可以看到，默认加载因子为 0.75， 默认容量为 1 << 4，也就是 16。加载因子过高，容易产生哈希冲突，加载因子过小，容易浪费空间，0.75是一种折中。

另外，整个 HashMap 的实现原理可以简单的理解成： 当我们 **put** 的时候，首先根据 **key** 算出一个数值 **x**，然后在 **table[x]** 中存放我们的值。 这样有一个好处是，以后的 get 等操作的时间复杂度直接就是O(1)，因为 HashMap 内部就是基于数组的一个实现。

另外，是怎么算出这个位置的，非常非常推荐看下 JDK 源码中 HashMap 的 hash 方法原理是什么?

## put 方法的实现 与 哈希冲突

下面再结合代码重点分析下 **HashMap** 的 **put** 方法的内部实现 和 哈希冲突的解决办法：

```
public V put(K key, V value) {
    return putVal(hash(key), key, value, false, true);
}
```

```
final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
                   boolean evict) {
    Node<K,V>[] tab; Node<K,V> p; int n, i;
    if ((tab = table) == null || (n = tab.length) == 0)
        n = (tab = resize()).length;
    if ((p = tab[i = (n - 1) & hash]) == null)
        tab[i] = newNode(hash, key, value, null);
    else {
        Node<K,V> e; K k;
        if (p.hash == hash &&
            ((k = p.key) == key || (key != null && key.equals(k))))
            e = p;
        else if (p instanceof TreeNode)
            e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);
        else {
            for (int binCount = 0; ; ++binCount) {
                if ((e = p.next) == null) {
                    p.next = newNode(hash, key, value, null);
                    if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st
                        treeifyBin(tab, hash);
```

**Assignees**

No one assigned

**Labels**

None yet

**Projects**

None yet

**Milestone**

No milestone

**Notifications**

Subscribe

You're not receiving notifications from this thread.

**4 participants**

```
                    break;
                }
            if (e.hash == hash &&
                    ((k = e.key) == key || (key != null && key.equals(k))))
                break;
            p = e;
        }
    }
    if (e != null) { // existing mapping for key
        V oldValue = e.value;
        if (!onlyIfAbsent || oldValue == null)
            e.value = value;
        afterNodeAccess(e);
        return oldValue;
    }
}
++modCount;
if (++size > threshold)
    resize();
afterNodeInsertion(evict);
return null;
}
```

首先我们看到 `hash(key)` 这个就是表示要根据 key 值算出一个数值，以此来决定在 table 数组的哪一个位置存放我们的数值。

（**Ps**：这个 **hash(key)** 方法 也是大有讲究的，会严重影响性能，实现得不好会让 **HashMap** 的 **O(1)** 时间复杂度降到 **O(n)**，在**JDK8**以下的版本中带来灾难性影响。它需要保证得出的数在哈希表中的均匀分布，目的就是要减少哈希冲突）

重要说明一下：

- **JDK8** 中哈希冲突过多，链表会转红黑树，时间复杂度是**O(logn)**，不会是**O(n)**
- **JDK8** 中哈希冲突过多，链表会转红黑树，时间复杂度是**O(logn)**，不会是**O(n)**
- **JDK8** 中哈希冲突过多，链表会转红黑树，时间复杂度是**O(logn)**，不会是**O(n)**

然后，我们再看到：

```
if ((p = tab[i = (n - 1) & hash]) == null)
    tab[i] = newNode(hash, key, value, null);
else {
    ......
```

这就表示，如果没有 哈希冲突，那么就可以放入数据 `tab[i] = newNode(hash, key, value, null);` 如果有哈希冲突，那么就执行 else 需要解决哈希冲突。

那么放入数据 其实就是 建立一个 Node 节点，该 Node节点有属性 key，value，分别保存我们的 key 值 和 value 值，然后再把这个 Node 节点放入到 table 数组中，并没有什么神秘的地方。

```
static class Node<K,V> implements Map.Entry<K,V> {
    final int hash;
    final K key;
    V value;
    Node<K,V> next;

    Node(int hash, K key, V value, Node<K,V> next) {
        this.hash = hash;
        this.key = key;
        this.value = value;
        this.next = next;
    }
```

上述可以看到 Node 节点中 有一个 `Node<K,V> next;` ，其实仔细思考下就应该知道这个是用来解决哈希冲突的。下面再看看是如何解决哈希冲突的：

哈希冲突：通俗的讲就是首先我们进行一次 **put** 操作，算出了我们要在 **table** 数组的 **x** 位置放入这个值。那么下次再进行一个 **put** 操作的时候，又算出了我们要在 **table** 数组的 **x** 位置放入这个值，那之前已经放入过值了，那现在怎么处理呢？

其实就是通过链表法进行解决。

首先，如果有哈希冲突，那么：

```
if (p.hash == hash &&
    ((k = p.key) == key || (key != null && key.equals(k))))
e = p;
```

需要判断 两者的 key 是否一样的，因为 HashMap 不能加入重复的键。如果一样，那么就覆盖，如果不一样，那么就先判断是不是 TreeNode 类型的：

```
else if (p instanceof TreeNode)
   e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);
```

这里表示 是不是现在已经转红黑树了（在大量哈希冲突的情况下，链表会转红黑树），一般我们小数据的情况下，是不会转的，所以这里暂时不考虑这种情况（Ps：本人也没太深入研究红黑树，所以就不说这个了）。

如果是正常情况下，会执行下面的语句来解决哈希冲突：

```
for (int binCount = 0; ; ++binCount) {
    if ((e = p.next) == null) {
        p.next = newNode(hash, key, value, null);
        if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st
            treeifyBin(tab, hash);
        break;
    }
    if (e.hash == hash &&
            ((k = e.key) == key || (key != null && key.equals(k))))
        break;
    p = e;
}
```

这里其实就是用链表法来解决。并且：

- 冲突的节点放在链表的最下面。
- 冲突的节点放在链表的最下面。
- 冲突的节点放在链表的最下面。

因为 首先有： `p = tab[i = (n - 1) & hash]` ，再 for 循环，然后有 `if ((e = p.next) == null) {` ,并且如果 当前节点的下一个节点有值的话，那么就 `p = e;` ，这就说明了放在最下面。

强烈建议自己拿笔拿纸画画。

# 总结

- 一个映射不能包含重复的键，一个键只能有一个值。允许使用 **null** 值 和 **null** 键，并且不保证映射顺序。
- **HashMap** 解决冲突的办法先是使用链表法，然后如果哈希冲突过多，那么会把链表转换成红黑树，以此来保证效率。
- 如果出现了哈希冲突，那么新加入的节点放在链表的最后面。

# 参考

推荐看一下：

1. Java HashMap工作原理及实现
2. Java 8：HashMap的性能提升



**HFOwnCity** commented on Jul 21, 2016

加载因子过高，容易产生哈希冲突这句是不是有点错误，怎么会增加哈希冲突呢，应该是增加refresh的次数

**pzxwhc** commented on Jul 21, 2016 Owner

@**HFOwnCity** 你好，我觉得这样表述应该是没有问题的。在 HashMap 类中的注释有一句 "Higher values decrease the space overhead but increase the lookup cost"，那么，为什么会增加 lookup cost，我觉得就是因为哈希冲突，导致 table 数组的每个元素有多个 Entry。另外，你说的增加refresh的次数是指resize() table 数组吗

**Se7enGo** commented on Jan 16 • edited

如果负载因子是默认的0.75，HashMap(16)的时候，占16个内存空间，实际上只用到了12个，超过12个就扩容。
如果负载因子是1的话，HashMap(16)的时候，占16个内存空间，实际上会填满16个以后才会扩容。
但是，用到的空间越多(也就是负载因子越大的情况)，hashcode重复的可能性就越大，同一个空间里面的元素数目就可能会增加，会增加查找的时间。摘自http://bbs.csdn.net/topics/350237871

**pzxwhc** changed the title from **Java 集合：HashMap（put方法的实现 与 哈希冲突）** to **Java 集合：HashMap（put方法的实现与哈希冲突）** on Jan 18

**tyCode5834** commented 25 days ago

讲道理，不应该瞎讲，但是第一句- -最直观就是理解就是键值对

**pzxwhc** commented 25 days ago Owner

**@tyCode5834** map 不就是 key-value 键值对的形式吗？ 瞎讲？

**tyCode5834** commented 25 days ago

大神。。写的很好没毛病。只是（最直观的理解就是键值对）（最直观理解就是键值对）。。感谢大神的讲解，我说不能瞎讲是我不能瞎讲，不是说你。

**pzxwhc** commented 25 days ago Owner

**@tyCode5834** 额，我的意思是我是不是说错了，因为按照我的理解应该可以这样理解的。如果你有什么不认同的，可以探讨啊，比如说是不是某些情况下 map 就和 key-value 键值对的形式有很大差异之类的。。。

**tyCode5834** commented 25 days ago

**@pzxwhc** 没有毛病，只是看你的作品，好多收获。然后看到第一句话的语句结构我笑了，没忍住下来评论一句。我倒是有个问题请教你，看了您写的这些，还需要看整个实现的源码么，还有你都是学习源码的？

**pzxwhc** commented 25 days ago Owner

**@tyCode5834** 没啊，但是我写的那些我感觉是蛮常见的，常见的得稍微看下，很多人也懂，只是没写出来。你可以看下 ThreadLocal 那个，要是不知道原理估计实际中会出问题的。

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Comment