Hashmap

## 重要参数

HashMap 的实例有两个参数影响其性能：初始容量 和加载因子。容量 是哈希表中桶的数量，初始容量只是哈希表在创建时的容量。加载因子 是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 rehash 操作（即重建内部[数据结构](http://baike.baidu.com/view/9900.htm" \t "_blank)），从而哈希表将具有大约两倍的桶数。

　　通常，默认加载因子 (.75) 在时间和空间成本上寻求一种折衷。加载因子过高虽然减少了空间开销，但同时也增加了查询成本（在大多数 HashMap 类的操作中，包括 get 和 put 操作，都反映了这一点）。在设置初始容量时应该考虑到映射中所需的条目数及其加载因子，以便最大限度地减少 rehash 操作次数。如果初始容量大于最大条目数除以加载因子，则不会发生 rehash 操作。

如果很多映射关系要存储在 HashMap 实例中，则相对于按需执行自动的 rehash 操作以增大表的容量来说，使用足够大的初始容量创建它将使得映射关系能更有效地存储。

**HashMap的数据结构**

 HashMap主要是用数组来存储数据的，我们都知道它会对key进行哈希运算，哈系运算会有重复的哈希值，对于哈希值的冲突，HashMap采用链表来解决的。在HashMap里有这样的一句属性声明：  
transient Entry[] table;  
Entry就是HashMap存储数据所用的类，它拥有的属性如下  
final K key;  
V value;  
final int hash;  
Entry<K,V> next;

看到next了吗？next就是为了哈希冲突而存在的。比如通过哈希运算，一个新元素应该在数组的第10个位置，但是第10个位置已经有Entry，那么好吧，将新加的元素也放到第10个位置，将第10个位置的原有Entry赋值给当前新加的 Entry的next属性。数组存储的是链表，链表是为了解决哈希冲突的，这一点要注意。

**几个关键的属性**  
存储数据的数组  
transient Entry[] table; 这个上面已经讲到了  
默认容量  
static final int DEFAULT\_INITIAL\_CAPACITY = 16;  
最大容量  
static final int MAXIMUM\_CAPACITY = 1 << 30;  
默认加载因子，加载因子是一个比例，当HashMap的数据大小>=容量\*加载因子时，HashMap会将容量扩容  
static final float DEFAULT\_LOAD\_FACTOR = 0.75f;  
当实际数据大小超过threshold时，HashMap会将容量扩容，threshold＝容量\*加载因子  
int threshold;  
加载因子  
final float loadFactor;

**正确的使用HashMap**

1：不要在并发场景中使用HashMap  
           HashMap是线程不安全的，如果被多个线程共享的操作，将会引发不可预知的问题，据sun的说法，在扩容时，会引起链表的闭环，在get元素时，就会无限循环，后果是cpu100%。  
看看get方法的红色部分  
public V get(Object key) {  
    if (key == null)  
        return getForNullKey();  
        int hash = hash(key.hashCode());  
        for (Entry<K,V> e = table[indexFor(hash, table.length)];  
             e != null;  
             e = e.next) {  
            Object k;  
            if (e.hash == hash && ((k = e.key) == key || key.equals(k)))  
                return e.value;  
        }  
        return null;  
    }  
  
2：如果数据大小是固定的，那么最好给HashMap设定一个合理的容量值  
        根据上面的分析，HashMap的初始默认容量是16，默认加载因子是0.75，也就是说，如果采用HashMap的默认构造函数，当增加数据时，数据实际容量超过10\*0.75=12时，HashMap就扩容，扩容带来一系列的运算，新建一个是原来容量2倍的数组，对原有元素全部重新哈希，如果你的数据有几千几万个，而用默认的HashMap构造函数，那结果是非常悲剧的，因为HashMap不断扩容，不断哈希，在使用HashMap的场景里，不会是多个线程共享一个HashMap,除非对HashMap包装并同步，由此产生的内存开销和cpu开销在某些情况下可能是致命的。