链滴

HashMap 深入解析（二）

作者：zsr251

原文链接：https://ld246.com/article/1512407349581

来源网站：链滴

共两篇，本文是第二篇，包含后六节。ps：你看到的是我写的第二遍！～坑die的有道云，写完之后然给我清空了，无力吐槽

目录

装填因子默认值及作用

```
/**
 * The load factor used when none specified in constructor.
 */
static final float DEFAULT_LOAD_FACTOR = 0.75f;
```

装填因子默认是0.75，当put值时

```
// The next size value at which to resize (capacity * load factor).
threshold = (int) Math.min(capacity * loadFactor, MAXIMUM_CAPACITY + 1);
// 如果大于下一个重置大小的值 则把数组扩大一倍
if ((size >= threshold) && (null != table[bucketIndex])) {
    resize(2 * table.length);
    hash = (null != key) ? hash(key) : 0;
    bucketIndex = indexFor(hash, table.length);
}
```

意思是说，当数组中的元素的个数>=总长度*装填因子时，数组长度扩容一倍

HashMap默认长度及原因

```
/**
 * The default initial capacity - MUST be a power of two.
 */
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4;
```

hashmap的默认长度是16，而且长度必须是2的倍数

```
private void inflateTable(int toSize) {
    // Find a power of 2 >= toSize
    int capacity = roundUpToPowerOf2(toSize);
```

```
    threshold = (int) Math.min(capacity * loadFactor, MAXIMUM_CAPACITY + 1);
    table = new Entry[capacity];
    initHashSeedAsNeeded(capacity);
}
// 把不是2的倍数的数转换成 > 原始值的2的倍数
 private static int roundUpToPowerOf2(int number) {
    // assert number >= 0 : "number must be non-negative";
    return number >= MAXIMUM_CAPACITY
            ? MAXIMUM_CAPACITY
            : (number > 1) ? Integer.highestOneBit((number - 1) << 1) : 1;
}
```

之所以必须是2的倍数，是因为HashMap进行Entry数组定位的时候，不是使用的取模操作，而是进的位操作默认取hash值的后四位。

例如：hash值为12345 Entry数组长度是16 该hash在数组中的位置计算不是 12345 mod 16 = 9 而是 12345 & 15 = 9，虽然结果是一样的，但是计算效率会更高，速度更快。

HashMap的线程安全问题

HashMap在并发编程中可能导致程序死循环

在多线程环境下，使用HashMap进行put操作时，如果多个线程同时进行扩容操作，有可能会使链表成闭环，造成获取Entry时死循环，导致CPU利用率接近100%。主要是因为HashMap在插入Entry的候是插在链表头的而不是链表尾，具体原因参考：https://www.cnblogs.com/andy-zhou/p/5402944.html

Java8中HashMap的优化

- JDK1.8中HashMap是数组+链表+红黑树实现的，链表长度是大于8的话把链表转换为红黑树
- JDK1.8种优化了扩容机制
- JDK1.8中优化了高位运算的算法
- 等等 虽然有点贱，但是的确还有很多～

HashMap和HashSet的关系

```
// 构造方法中就是初始化一个默认的HashMap
public HashSet() {
    map = new HashMap<>();
}
private static final Object PRESENT = new Object();
// 增加元素 就是在HashMap中增加一个Key为增加的元素，值为Object实例的键值对
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
// 判断是否存在也是判断元素是否在HashMap中是否存在
public boolean contains(Object o) {
    return map.containsKey(o);
}
```

综上，**HashSet是通过HashMap的Key实现的**

线程安全的HashMap：CurrentHashMap简介

HashMap存在线程安全问题，HashTable虽然是线程安全的，但是效率太低。所以在多线程的环境我们一般使用CurrentHashMap。

CurrentHashMap使用了锁分段技术，首先把数据分成一段一段地存储，然后给每一段都加上一把，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。

有机会的话我会写一篇文章仔细分析，埋个坑，敬请期待：）