HashMap 与 HashTable 的区别

备注：让面试官引入 多线程、锁、ConcurrentHashMap

1.HashMap 线程不安全 HashTable 线程是安全的采用 synchronized
2.HashMap 允许存放 key 为 null HashTable 不允许存放 key 为 null
3.在多线程的情况下，推荐使用 ConcurrentHashMap 线程安全 且效率非常高
面试官下一个问题 就会问题说“ConcurrentHashMap 原理”

重写了 equals 方法 为什么 也需要重写 HashCode 方法？

== 与 equals 区别
== 比较两个对象内存地址
Equals 方法属于 Object 父类中，默认的情况下两个对象内存地址是否相等，只是我们重写 Object 父类中 Equals 方法来实现 比较 对象属性值是否相等

两个对象值如果相等的话， HashCode 相等
两个对象值如果不相等的话， HashCode 不一定

两个对象 hashCode 值 如果相等， 值是否相等 不一定
两个对象值如果是相等的话，hashcode 是相等。

```
st06.java × Object.java × MayiktEntity.java × Has
     *
     * @param   obj   the reference object wi
     * @return  {@code true} if this object i
     *          argument; {@code false} other
     * @see     #hashCode()
     * @see     java.util.HashMap
     */
    public boolean equals(Object obj) {
        return (this == obj);
    }
```

什么是 Hash 冲突

Key 值不同 但是 hashcode 值相等

谈谈 HashMap 的 Hash 冲突如何解决的

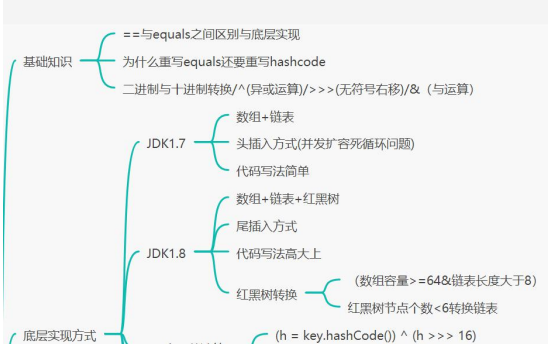
Jdk1.7 数组+链表 链表缺陷：如果链表过长的情况下

查询的时间复杂度就是为 o(n) 需要从头查询尾部

效率非常低

JDK1.8 数组+链表+红黑树

HashMap 底层是如何实现的



在 HashMap1.7 版本中底层是基于数组+链表实现的，如果发生 Hash 冲突概率问题，会存放到同一个链表中，链表如果过长 会从头查询到尾部 效率非常低。所以在 HashMap1.8 版本 （数组容量>=64&链表长度大于 8） 就会将该链表转化红黑树。

```
/**
 * @author 余胜军
 * @ClassName MayiktHashMap
 * @qq 644064779
 * @addres www.mayikt.com
 * 微信:yushengjun644
 */
public class MayiktHashMap<K, V> {
    private Entry[] objects = new Entry[10000];

    class Entry<K, V> {
        K k;
        V v;
        Entry<K, V> next;

        public Entry(K k, V v) {
            this.k = k;
            this.v = v;
        }
    }

    public void put(K k, V v) {
        int index = k.hashCode() % objects.length;
        Entry<K, V> oldEntry = objects[index];
        if (oldEntry == null) {
            objects[index] = new Entry<K, V>(k, v);
        } else {
            oldEntry.next = new Entry<K, V>(k, v);
        }
    }

    public V get(K k) {
        int index = k.hashCode() % objects.length;
        for (Entry<K, V> entry = objects[index]; entry != null; entry = entry.next) {
            if (entry.k.equals(k) || entry.k == k) {
                return entry.v;
            }
        }
```

```
        return null;
    }


    public static void main(String[] args) {
        MayiktHashMap mayiktHashMap = new MayiktHashMap();
        mayiktHashMap.put("a", "a");
        mayiktHashMap.put(97, 97);
        System.out.println(mayiktHashMap.get("a"));
        System.out.println(mayiktHashMap.get(97));
    }


}
```

HashMap 根据 Key 查询时间复杂度？

1.Key 没有产生 hash 冲突 时间复杂度是为 o(1); 只需要查询一次
2.Key 产生 hash 冲突 采用链表存放则为 O(N) 从头查询到尾部
3.key 产生 hash 冲突采用红黑树存放则为 O（LogN）

HashMap 底层是有序存放的吗？

是无序的，因为 Hash 算法是散列计算的 没有顺序，如果需要顺序
可以使用 LinkedHashMap 集合采用双向链表存放。

Put(1,1) --index=6
Put(2,2)---index=0
Put(3,3)---index=7
2,1,3
遍历是根据数组 index=0

HashMap7 扩容产生死循环问题有了解过吗？

其实这个 JDK 官方不承认这个 bug，因为 HashMap 本身是线程不安全的，不推荐在

多线程的情况下使用，是早期阿里一名员工 发生在多线程 的情况下使用 HashMap1.7 扩容会发生死循环问题，因为 HashMap1.7 采用头插入法 后来在在 HashMap1.8 改为尾插法 。
如果是在多线程的情况下 推荐使用 ConcurrentHashMap

HashMap Key 为 null 存放在 什么位置

存放在数组 index 为 0 的位置。

ConcurrentHashMap 底层是如何实现？

1.传统方式 使用 HashTable 保证线程问题，是采用 synchronized 锁将整个 HashTable 中的数组锁住，
在多个线程中只允许一个线程访问 Put 或者 Get，效率非常低，但是能够保证线程安全问题。
2.多线程的情况下 JDK 官方推荐使用 ConcurrentHashMap
ConcurrentHashMap 1.7 采用分段锁设计 底层实现原理：数组+Segments 分段锁 +HashEntry 链表实现
大致原理就是将一个大的 HashMap 分成 n 多个不同的小的 HashTable
不同的 key 计算 index 如果没有发生冲突 则存放到不同的小的 HashTable 中 ，从而可以实现多线程，同时做 put 操作，但是如果多个线程同时 put 操作 key 发生了 index 冲突落到同一个小的 HashTable 中还是会发生竞争锁。
3.ConcurrentHashMap 1.7 采用 Lock 锁+CAS 乐观锁+UNSAFE 类 里面有实现 类似于 synchronized
锁的升级过程。
4.ConcurrentHashMap 1.8 版本 put 操作 取消 segment 分段设计 直接使用 Node 数组来保存数据
index 没有发生冲突使用 cas 锁 index 如果发生冲突则 使用 synchronized