

# 面试：HashMap 夺命二十一问！鸡哥都扛不住~

点击关注  芋道源码 今天

点击上方“芋道源码”，选择“设为星标”

管她前浪，还是后浪？

能浪的浪，才是好浪！

每天 8:55 更新文章，每天掉亿点头发...

## 源码精品专栏

- [原创 | Java 2020 超神之路，很肝~](#)
- [中文详细注释的开源项目](#)
- [RPC 框架 Dubbo 源码解析](#)
- [网络应用框架 Netty 源码解析](#)
- [消息中间件 RocketMQ 源码解析](#)
- [数据库中间件 Sharding-JDBC 和 MyCAT 源码解析](#)
- [作业调度中间件 Elastic-Job 源码解析](#)
- [分布式事务中间件 TCC-Transaction 源码解析](#)
- [Eureka 和 Hystrix 源码解析](#)
- [Java 并发源码](#)

来源：cnblogs.com/Young111/p/11519952.html

## 1：HashMap 的数据结构？

A：哈希表结构（链表散列：数组+链表）实现，结合数组和链表的优点。当链表长度超过 8 时，链表转换为红黑树。

```
transient Node<K,V>[] table;
```

## 2：HashMap 的工作原理？

HashMap 底层是 hash 数组和单向链表实现，数组中的每个元素都是链表，由 Node 内部类（实现 Map.Entry 接口）实现，HashMap 通过 put & get 方法存储和获取。

存储对象时，将 K/V 键值传给 put() 方法：

①、调用 hash(K) 方法计算 K 的 hash 值，然后结合数组长度，计算得数组下标；

②、调整数组大小（当容器中的元素个数大于  $\text{capacity} * \text{loadfactor}$  时，容器会进行扩容resize为  $2n$ ）；

③、i.如果 K 的 hash 值在 HashMap 中不存在，则执行插入，若存在，则发生碰撞；

ii.如果 K 的 hash 值在 HashMap 中存在，且它们两者 equals 返回 true，则更新键值对；

iii. 如果 K 的 hash 值在 HashMap 中存在，且它们两者 equals 返回 false，则插入链表的尾部（尾插法）或者红黑树中（树的添加方式）。

（JDK 1.7 之前使用头插法、JDK 1.8 使用尾插法）（注意：当碰撞导致链表大于  $\text{TREEIFY\_THRESHOLD} = 8$  时，就把链表转换成红黑树）

获取对象时，将 K 传给 get() 方法：①、调用 hash(K) 方法（计算 K 的 hash 值）从而获取该键值所在链表的数组下标；②、顺序遍历链表，equals()方法查找相同 Node 链表中 K 值对应的 V 值。

hashCode 是定位的，存储位置；equals是定性的，比较两者是否相等。

### 3.当两个对象的 hashCode 相同会发生什么？

因为 hashCode 相同，不一定就是相等的（equals方法比较），所以两个对象所在数组的下标相同，"碰撞"就此发生。又因为 HashMap 使用链表存储对象，这个 Node 会存储到链表中。为什么要重写 hashCode 和 equals 方法？推荐看下。

### 4.你知道 hash 的实现吗？为什么要这样实现？

JDK 1.8 中，是通过 hashCode() 的高 16 位异或低 16 位实现的： $(h = k.\text{hashCode}()) \wedge (h \gg \gg 16)$ ，主要是从速度，功效和质量来考虑的，减少系统的开销，也不会造成因为高位没有参与下标的计算，从而引起的碰撞。

### 5.为什么要用异或运算符？

保证了对象的 hashCode 的 32 位值只要有一位发生改变，整个 hash() 返回值就会改变。尽可能的减少碰撞。

### 6.HashMap 的 table 的容量如何确定？loadFactor 是什么？该容量如何变化？这种变化会带来什么问题？

①、table 数组大小是由 capacity 这个参数确定的，默认是16，也可以构造时传入，最大限制是  $1 < \leq 30$ ；

②、loadFactor 是装载因子，主要目的是用来确认table 数组是否需要动态扩展，默认值是 0.75，比如table 数组大小为 16，装载因子为 0.75 时，threshold 就是12，当 table 的实际大小超过 12 时，table就需要动态扩容；

③、扩容时，调用 resize() 方法，将 table 长度变为原来的两倍（注意是 table 长度，而不是 threshold）

④、如果数据很大的情况下，扩展时将会带来性能的损失，在性能要求很高的地方，这种损失很可能很致命。

## 7.HashMap中put方法的过程？

答：“调用哈希函数获取Key对应的hash值，再计算其数组下标；

如果没有出现哈希冲突，则直接放入数组；如果出现哈希冲突，则以链表的方式放在链表后面；

如果链表长度超过阈值( TREEIFY THRESHOLD==8)，就把链表转成红黑树，链表长度低于6，就把红黑树转回链表；

如果结点的key已经存在，则替换其value即可；

如果集合中的键值对大于12，调用resize方法进行数组扩容。”

## 8.数组扩容的过程？

创建一个新的数组，其容量为旧数组的两倍，并重新计算旧数组中结点的存储位置。结点在新数组中的位置只有两种，原下标位置或原下标+旧数组的大小。

## 9.拉链法导致的链表过深问题为什么不用二叉查找树代替，而选择红黑树？为什么不一直使用红黑树？

之所以选择红黑树是为了解决二叉查找树的缺陷，二叉查找树在特殊情况下会变成一条线性结构（这就跟原来使用链表结构一样了，造成很深的问题），遍历查找会非常慢。推荐：面试问红黑树，我脸都绿了。

而红黑树在插入新数据后可能需要通过左旋，右旋、变色这些操作来保持平衡，引入红黑树就是为了查找数据快，解决链表查询深度的问题，我们知道红黑树属于平衡二叉树，但是为了保持“平衡”是需要付出代价的，但是该代价所损耗的资源要比遍历线性链表要少，所以当长度大于8的时候，会使用红黑树，如果链表长度很短的话，根本不需要引入红黑树，引入反而会慢。

## 10.说说你对红黑树的见解？

- 每个节点非红即黑
- 根节点总是黑色的
- 如果节点是红色的，则它的子节点必须是黑色的（反之不一定）
- 每个叶子节点都是黑色的空节点（NIL节点）
- 从根节点到叶节点或空子节点的每条路径，必须包含相同数目的黑色节点（即相同的黑色高度）

## 11.jdk8中对HashMap做了哪些改变？

在java 1.8中，如果链表的长度超过了8，那么链表将转换为红黑树。（桶的数量必须大于64，小于64的时候只会扩容）

发生hash碰撞时，java 1.7 会在链表的头部插入，而java 1.8会在链表的尾部插入

在java 1.8中，Entry被Node替代(换了一个马甲)。

## 12.HashMap，LinkedHashMap，TreeMap 有什么区别？

HashMap 参考其他问题；

LinkedHashMap 保存了记录的插入顺序，在用 Iterator 遍历时，先取到的记录肯定是先插入的；遍历比 HashMap 慢；

TreeMap 实现 SortMap 接口，能够把它保存的记录根据键排序（默认按键值升序排序，也可以指定排序的比较器）

## 13.HashMap & TreeMap & LinkedHashMap 使用场景？

一般情况下，使用最多的是 HashMap。

HashMap：在 Map 中插入、删除和定位元素时；

TreeMap：在需要按自然顺序或自定义顺序遍历键的情况下；

LinkedHashMap：在需要输出的顺序和输入的顺序相同的情况下。

## 14.HashMap 和 Hashtable 有什么区别？

- ①、HashMap 是线程不安全的，Hashtable 是线程安全的；
- ②、由于线程安全，所以 Hashtable 的效率比不上 HashMap；
- ③、HashMap 最多只允许一条记录的键为null，允许多条记录的值为null，而 Hashtable 不允许；
- ④、HashMap 默认初始化数组的大小为16，Hashtable 为 11，前者扩容时，扩大两倍，后者扩大两倍+1；
- ⑤、HashMap 需要重新计算 hash 值，而 Hashtable 直接使用对象的 hashCode

## 15.Java 中的另一个线程安全的与 HashMap 极其类似的类是什么？同样是线程安全，它与 Hashtable 在线程同步上有什么不同？

ConcurrentHashMap 类（是 Java 并发包 `java.util.concurrent` 中提供的一个线程安全且高效的 HashMap 实现）。

Hashtable 是使用 `synchronize` 关键字加锁的原理（就是对对象加锁）；

而针对 ConcurrentHashMap，在 JDK 1.7 中采用 分段锁的方式；JDK 1.8 中直接采用了 CAS（无锁算法）+ `synchronized`。

## 16.HashMap & ConcurrentHashMap 的区别？

除了加锁，原理上无太大区别。另外，HashMap 的键值对允许有null，但是 ConcurrentHashMap 都不允许。

## 17.为什么 ConcurrentHashMap 比 Hashtable 效率要高？

HashTable 使用一把锁（锁住整个链表结构）处理并发问题，多个线程竞争一把锁，容易阻塞；

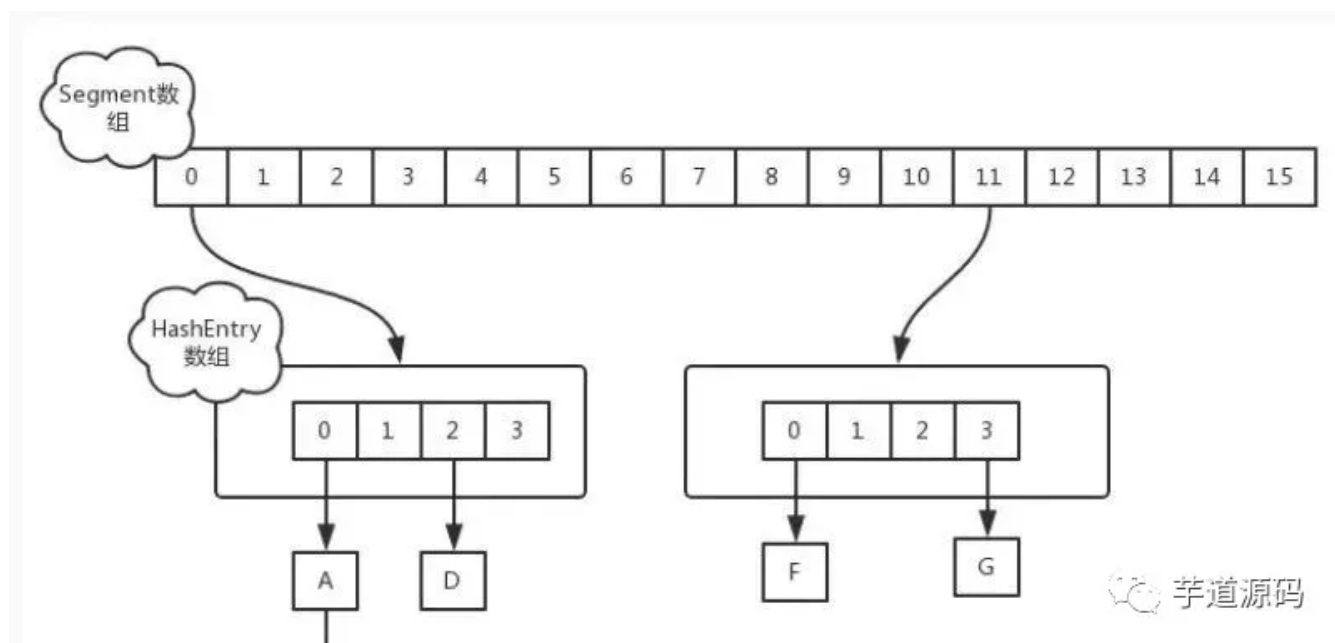
## ConcurrentHashMap

- JDK 1.7 中使用分段锁（ReentrantLock + Segment + HashEntry），相当于把一个 HashMap 分成多个段，每段分配一把锁，这样支持多线程访问。锁粒度：基于 Segment，包含多个 HashEntry。
- JDK 1.8 中使用 CAS + synchronized + Node + 红黑树。锁粒度：Node（首结点）（实现 Map.Entry）。锁粒度降低了。

## 18.针对 ConcurrentHashMap 锁机制具体分析（JDK 1.7 VS JDK 1.8）

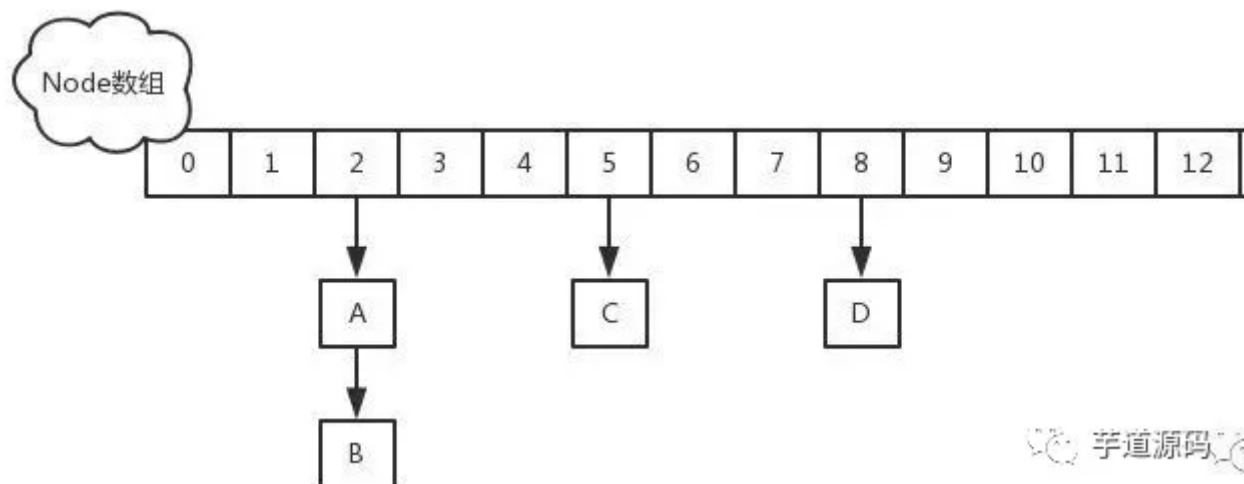
JDK 1.7 中，采用分段锁的机制，实现并发的更新操作，底层采用数组+链表的存储结构，包括两个核心静态内部类 Segment 和 HashEntry。

- ①、Segment 继承 ReentrantLock（重入锁）用来充当锁的角色，每个 Segment 对象守护每个散列映射表的若干个桶；
- ②、HashEntry 用来封装映射表的键-值对；
- ③、每个桶是由若干个 HashEntry 对象链接起来的链表



JDK 1.8 中，采用Node + CAS + Synchronized来保证并发安全。取消类 Segment，直接用 table 数组存储键值对；当 HashEntry 对象组成的链表长度超过 TREEIFY\_THRESHOLD 时，链表转换

为红黑树，提升性能。底层变更为数组 + 链表 + 红黑树。



## 19.ConcurrentHashMap 在 JDK 1.8 中，为什么要使用内置锁 synchronized 来代替重入锁 ReentrantLock?

- ①、粒度降低了；
- ②、JVM 开发团队没有放弃 synchronized，而且基于 JVM 的 synchronized 优化空间更大，更加自然。
- ③、在大量的数据操作下，对于 JVM 的内存压力，基于 API 的 ReentrantLock 会开销更多的内存。

## 20.ConcurrentHashMap 简单介绍?

### ①、重要的常量：

`private transient volatile int sizeCtl;`

当为负数时，-1 表示正在初始化，-N 表示 N - 1 个线程正在进行扩容；

当为 0 时，表示 table 还没有初始化；

当为其他正数时，表示初始化或者下一次进行扩容的大小。

### ②、数据结构：

Node 是存储结构的基本单元，继承 HashMap 中的 Entry，用于存储数据；

TreeNode 继承 Node，但是数据结构换成了二叉树结构，是红黑树的存储结构，用于红黑树中存储数据；

TreeBin 是封装 TreeNode 的容器，提供转换红黑树的一些条件和锁的控制。

### ③、存储对象时 (put() 方法)：

如果没有初始化，就调用 initTable() 方法来进行初始化；

如果没有 hash 冲突就直接 CAS 无锁插入；

如果需要扩容，就先进行扩容；

如果存在 hash 冲突，就加锁来保证线程安全，两种情况：一种是链表形式就直接遍历到尾端插入，一种是红黑树就按照红黑树结构插入；

如果该链表的数量大于阈值 8，就要先转换成红黑树的结构，break 再一次进入循环

如果添加成功就调用 addCount() 方法统计 size，并且检查是否需要扩容。

### ④、扩容方法 transfer()：默认容量为 16，扩容时，容量变为原来的两倍。

helpTransfer()：调用多个工作线程一起帮助进行扩容，这样的效率就会更高。

### ⑤、获取对象时 (get()方法)：

计算 hash 值，定位到该 table 索引位置，如果是首结点符合就返回；

如果遇到扩容时，会调用标记正在扩容结点 ForwardingNode.find()方法，查找该结点，匹配就返回；

以上都不符合的话，就往下遍历结点，匹配就返回，否则最后就返回 null。

## 21.ConcurrentHashMap 的并发度是什么？

程序运行时能够同时更新 ConcurrentHashMap 且不产生锁竞争的最大线程数。默认为 16，且可以在构造函数中设置。

当用户设置并发度时，ConcurrentHashMap 会使用大于等于该值的最小2幂指数作为实际并发度（假如用户设置并发度为17，实际并发度则为32）

---

欢迎加入我的知识星球，一起探讨架构，交流源码。加入方式，[长按下方二维码噢](#)：