

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题 最近阅读

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

10 Map源码会问哪些面试题

更新时间：2019-09-10 10:34:08



“
人的一生可能燃烧也可能腐朽，我不能腐朽，我愿意燃烧起来！
——奥斯特洛夫斯基

果断更，请联系QQ/微信6426006

Map 在面试中，占据了很大一部分的面试题目，其中以 HashMap 为主，这些面试题目有的可以说得清楚，有的很难说清楚，如果是面对面面试的话，建议画一画。

1 Map 整体数据结构类问题

1.1 说一说 HashMap 底层数据结构

答：HashMap 底层是数组 + 链表 + 红黑树的数据结构，数组的主要作用是方便快速查找，时间复杂度是 $O(1)$ ，默认大小是 16，数组的下标索引是通过 key 的 hashCode 计算出来的，数组元素叫做 Node，当多个 key 的 hashCode 一致，但 key 值不同时，单个 Node 就会转化成链表，链表的查询复杂度是 $O(n)$ ，当链表的长度大于等于 8 并且数组的大小超过 64 时，链表就会转化成红黑树，红黑树的查询复杂度是 $O(\log(n))$ ，简单来说，最坏的查询次数相当于红黑树的最大深度。

1.2 HashMap、TreeMap、LinkedHashMap 三者有啥相同点，有啥不同点？

答：相同点：

- 1. 三者在特定的情况下，都会使用红黑树；
- 2. 底层的 hash 算法相同；
- 3. 在迭代的过程中，如果 Map 的数据结构被改动，都会报 `ConcurrentModificationException` 的错误。

目录

1. HashMap 数据结构以数组为主，查询非常快， TreeMap 数据结构以红黑树例为主，利用了红黑树左小右大的特点，可以实现 key 的排序， LinkedHashMap 在 HashMap 的基础上增加了链表的结构，实现了插入顺序访问和最少访问删除两种策略；
2. 由于三种 Map 底层数据结构的差别，导致了三者的使用场景的不同， TreeMap 适合需要根据 key 进行排序的场景， LinkedHashMap 适合按照插入顺序访问，或需要删除最少访问元素的场景， 剩余场景我们使用 HashMap 即可， 我们工作中大部分场景基本都在使用 HashMap；
3. 由于三种 map 的底层数据结构的不同，导致上层包装的 api 略有差别。

1.3 说一下 Map 的 hash 算法

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
key 在数组中的位置公式： tab[(n - 1) & hash]
```

如上代码是 HashMap 的hash 算法。

这其实是一个数学问题， 源码中就是通过以上代码来计算 hash 的， 首先计算出 key 的 hashCode， 因为 key 是 Object， 所以会根据 key 的不同类型进行 hashCode 的计算， 接着计算 $h \wedge (h \ggg 16)$ ， 这么做的好处是使大多数场景下， 算出来的 hash 值比较分散。

一般来说， hash 值算出来之后， 要计算当前 key 在数组中的索引下标位置时， 可以采用取模的方式， 就是索引下标位置 = hash 值 % 数组大小， 这样做的好处， 就是可以保证计算出来的索引下标值可以均匀的分布在数组的各个索引位置上， 但取模操作对于处理器的计算是比较慢的， 数学上有个公式， 当 b 是 2 的幂次方时， $a \% b = a \& (b-1)$ ， 所以此处索引位置的计算公式我们可以更换为： $(n-1) \& hash$ 。

此问题可以延伸出三个小问题：

1：为什么不用 key % 数组大小， 而是需要用 key 的 hash 值 % 数组大小。

答：如果 key 是数字， 直接用 key % 数组大小是完全没有问题的， 但我们的 key 还有可能是字符串， 是复杂对象， 这时候用 字符串或复杂对象 % 数组大小是不行的， 所以需要先计算出 key 的 hash 值。

2：计算 hash 值时， 为什么需要右移 16 位？

答：hash 算法是 $h \wedge (h \ggg 16)$ ， 为了使计算出的 hash 值更分散， 所以选择先将 h 无符号右移 16 位， 然后再于 h 异或时， 就能达到 h 的高 16 位和低 16 位都能参与计算， 减少了碰撞的可能性。

3：为什么把取模操作换成了 & 操作？

答：key.hashCode() 算出来的 hash 值还不是数组的索引下标， 为了随机的计算出索引的下标位置， 我们还会用 hash 值和数组大小进行取模， 这样子计算出来的索引下标比较均匀分布。

取模操作处理器计算比较慢， 处理器对 & 操作就比较擅长， 换成了 & 操作， 是有数学上证明的支撑， 为了提高了处理器处理的速度。

合：因为只有大小是 2 的幂次方时，才能使 $hash \& (n-1) == (hash \% n)$ (数组大小) $== (hash \% n)$ 公式成立。

1.4 为解决 hash 冲突，大概有哪些办法。

- 答：1：好的 hash 算法，细问的话复述一下上题的 hash 算法;
- 2：自动扩容，当数组大小快满的时候，采取自动扩容，可以减少 hash 冲突;
- 3：hash 冲突发生时，采用链表来解决;
- 4：hash 冲突严重时，链表会自动转化成红黑树，提高遍历速度。

网上列举的一些其它办法，如开放定址法，尽量不要说，因为这些方法资料很少，实战用过的人更少，如果你没有深入研究的话，面试官让你深入描述一下很难说清楚，反而留下不好的印象，说 HashMap 现有的措施就足够了。

2 HashMap 源码细节类问题

2.1 HashMap 是如何扩容的？

- 答：扩容的时机：
1. put 时，发现数组为空，进行初始化扩容，默认扩容大小为 16;
 2. put 成功后，发现现有数组大小大于扩容的门限值时，进行扩容，扩容为老数组大小的 2 倍;
- 扩容的门限是 threshold，每次扩容时 threshold 都会被重新计算，门限值等于数组的大小 * 影响因子（0.75）。

新数组初始化之后，需要将老数组的值拷贝到新数组上，链表和红黑树都有自己拷贝的方法。

2.2 hash 冲突时怎么办？

- 答：hash 冲突指的是 key 值的 hashCode 计算相同，但 key 值不同的情况。
- 如果桶中元素原本只有一个或已经是链表了，新增元素直接追加到链表尾部；
- 如果桶中元素已经是链表，并且链表个数大于等于 8 时，此时有两种情况：
1. 如果此时数组大小小于 64，数组再次扩容，链表不会转化成红黑树;
 2. 如果数组大小大于 64 时，链表就会转化成红黑树。

这里不仅仅判断链表个数大于等于 8，还判断了数组大小，数组容量小于 64 没有立即转化的原因，猜测主要是因为红黑树占用的空间比链表大很多，转化也比较耗时，所以数组容量小的情况下冲突严重，我们可以先尝试扩容，看看能否通过扩容来解决冲突的问题。

2.3 为什么链表个数大于等于 8 时，链表要转化成红黑树了？

答：当链表个数太多了，遍历可能比较耗时，转化成红黑树，可以使遍历的时间复杂度降低，但转化成红黑树，有空间和转化耗时的成本，我们通过泊松分布公式计算，正常情况下，链表个数出现 8 的概念不到千万分之一，所以说正常情况下，链表都不会转化成红黑树，这样设计的目

目录

延伸问题：红黑树什么时候转变成链表。

答：当节点的个数小于等于 6 时，红黑树会自动转化成链表，主要还是考虑红黑树的空间成本问题，当节点个数小于等于 6 时，遍历链表也很快，所以红黑树会重新变成链表。

2.4 HashMap 在 put 时，如果数组中已经有了这个 key，我不想把 value 覆盖怎么办？取值时，如果得到的 value 是空时，想返回默认值怎么办？

答：如果数组有了 key，但不想覆盖 value，可以选择 putIfAbsent 方法，这个方法有个内置变量 onlyIfAbsent，内置是 true，就不会覆盖，我们平时使用的 put 方法，内置 onlyIfAbsent 为 false，是允许覆盖的。

取值时，如果为空，想返回默认值，可以使用 getOrDefault 方法，方法第一参数为 key，第二个参数为你想返回的默认值，如 map.getOrDefault(“2”，“0”)，当 map 中没有 key 为 2 的值时，会默认返回 0，而不是空。

2.5 通过以下代码进行删除，是否可行？

```
HashMap<String,String > map = Maps.newHashMap();
map.put("1","1");
map.put("2","2");
map.forEach((s, s2) -> map.remove("1"));
```

答：不行，会报错误 ConcurrentModificationException，原因如下图：

```
public void forEach(BiConsumer<? super K, ? super V> action) {
    Node<K,V>[] tab;
    if (action == null)
        throw new NullPointerException();
    if (size > 0 && (tab = table) != null) {
        int mc = modCount;
        for (int i = 0; i < tab.length; ++i) {
            for (Node<K,V> e = tab[i]; e != null; e = e.next)
                action.accept(e.key, e.value);
        }
        if (modCount != mc)
            throw new ConcurrentModificationException();
    }
}
```

建议使用迭代器的方式进行删除，原理同 ArrayList 迭代器原理，我们在《List 源码会问那些面试题》中有说到。

2.6 描述一下 HashMap get、put 的过程

答：我们在源码解析中有说，可以详细描述下源码的实现路径，说不清楚的话，可以画一画。

3 其它 Map 面试题

3.1 DTO 作为 Map 的 key 时，有无需要注意的点？

目录

看是什么类型的 Map，如果是 HashMap 的话，一定需要覆写 equals 和 hashCode 方法，因为在 get 和 put 的时候，需要通过 equals 方法进行相等的判断；如果是 TreeMap 的话，DTO 需要实现 Comparable 接口，因为 TreeMap 会使用 Comparable 接口进行判断 key 的大小；如果是 LinkedHashMap 的话，和 HashMap 一样的。

3.2 LinkedHashMap 中的 LRU 是什么意思，是如何实现的。

答：LRU，英文全称：Least recently used，中文叫做最近最少访问，在 LinkedHashMap 中，也叫做最少访问删除策略，我们可以通过 removeEldestEntry 方法设定一定的策略，使最少被访问的元素，在适当的时机被删除，原理是在 put 方法执行的最后，LinkedHashMap 会去检查这种策略，如果满足策略，就删除头节点。

保证头节点就是最少访问的元素的原理是：LinkedHashMap 在 get 的时候，都会把当前访问的节点，移动到链表的尾部，慢慢的，就会使头部的节点都是最少被访问的元素。

3.3 为什么推荐 TreeMap 的元素最好都实现 Comparable 接口？但 key 是 String 的时候，我们却没有额外的工作呢？

答：因为 TreeMap 的底层就是通过排序来比较两个 key 的大小的，所以推荐 key 实现 Comparable 接口，是为了往你希望的排序顺序上发展，而 String 本身已经实现了 Comparable 接口，所以使用 String 时，我们不需要额外的工作，不仅仅是 String，其他包装类型也都实现了 Comparable 接口，如 Long、Double、Short 等等。

果断更，请总结联系QQ/微信6426006

Map 的面试题主要是 HashMap 为主，会问很多源码方面的东西，TreeMap 和 LinkedHashMap 主要以功能和场景为主，作为加分项。

Map 的面试题型很多，但只要弄懂原理，题目再多变化，回答起来都会比较简单。

← 09 TreeMap 和 LinkedHashMap 核心源码解析

11 HashSet、TreeSet 源码解析 →

精选留言 8

欢迎在这里发表留言，作者筛选后可公开显示

qq_起风了_90

老师，能写写jdk1.7hashmap的死循环吗？

👍 0 回复

2019-12-04

Sicimike

老师，1.2说HashMap、TreeMap、LinkedHashMap底层hash算法相同，但是TreeMap没有用到hash算法吧，因为它没有bucket，无需根据hash(key)去找到索引。

<div>← 慕课专栏</div> <div>面试官系统精讲Java源码及大厂真题 / 10 Map源码会问哪些面试题</div>		
目录	文贺 回复 Sicimike	
	你说的很有道理，周一找编辑订正下。	
	回复	2019-11-17 11:10:51
	qq_踏ㄋ寻梅_03702812	
	老师。扩容为老数组大小的 2 倍。你确定吗？？不是裸照成2的n次方吗	
	👍 0 回复	2019-09-14
	文贺 回复 qq_踏ㄋ寻梅_03702812	
	没有问题哦，你看一仔细看下咱们 HashMap 源码章节，源码中扩容中代码是这么写的：newThr = oldThr << 1，新数组大小为老数组大小的 2 倍。	
	回复	2019-09-15 09:28:10
	qq_踏ㄋ寻梅_03702812 回复 qq_踏ㄋ寻梅_03702812	
	哦。不好意思，我傻了。	
	回复	2019-09-17 00:35:28
	shuangyueliao	
	hashmap小于64是链表，大于等于是转化为红黑树，老师漏了等于	
	👍 0 回复	2019-09-12
	文贺 回复 shuangyueliao	
	这个本文 2.2 面试题和 HashMap 源码解析章节多次提到 64 哦，其余地方在说扩容的时候，可能并不会总是带上 64，你说的应该是 2.3 问题，问的是为什么在大于等于 8 时扩容是吧，这时候根据语境，问的是为什么，而不是什么时候，所以我们就无需说 64 哦。	
	回复	2019-09-15 09:26:15
	一个被女人上过的男人	
	老师，更新一集concurrenthashmap吧，谢谢	
	👍 0 回复	2019-09-12
	文贺 回复 一个被女人上过的男人	
	第三章会说的哈，并发集合类。	
	回复	2019-09-12 12:28:40
	一个被女人上过的男人	
	底层实现这个1.7和1.8有区别的啊，我想问哈老师，这个问题面试时要不要给面试官说哈对比呢？😁	
	👍 0 回复	2019-09-12
	文贺 回复 一个被女人上过的男人	
	面试官问啥答啥哈，不问就不用说哈，HashMap 1.7 比较简单，问的人现在也比较少了	
	回复	2019-09-12 12:31:07

目录	章节怎么跟之前简介内容说得得不一样0.0.
	<div><div>👍 0</div><div>回复</div><div>2019-09-11</div></div>
	<div><div>文贺</div><div>回复</div><div>Spring_inthis</div><div>一样的吧，哪里不一样？</div><div>回复</div><div>2019-09-11 17:48:47</div></div>
	<div><div>慕仰0328976</div><div>其实想要hash的更加彻底还有很多神奇的算法，比如redis的murmurHash或者DJB HASH算法，但是因为性能差不多最后还是用了效率最高的^（抑或）来解决问题</div><div><div>👍 0</div><div>回复</div><div>2019-09-10</div></div></div>
	<div><div>文贺</div><div>回复</div><div>慕仰0328976</div><div>有道理的，学了这么多算法，工作中可能一个都用不到。。</div><div>回复</div><div>2019-09-10 13:59:14</div></div>

千学不如一看，千看不如一练

果断更， 请联系QQ/微信64260060