

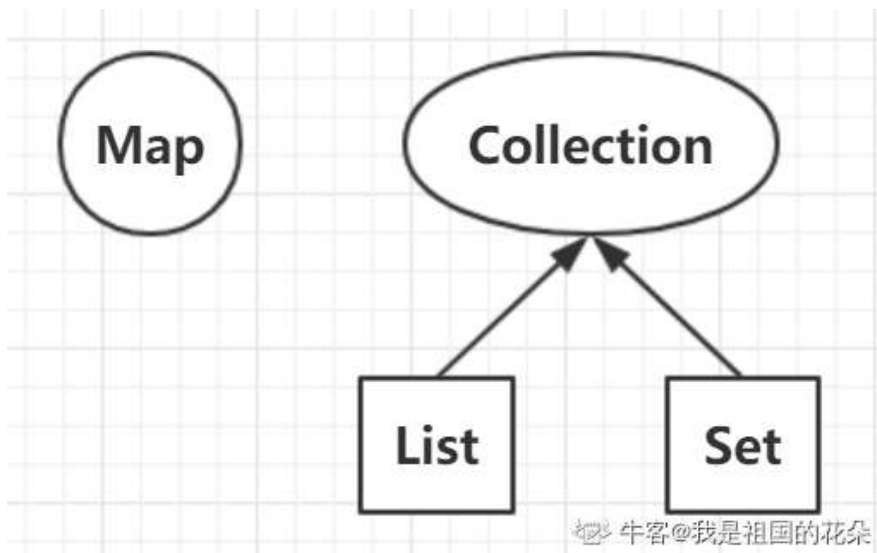
大家好，很高兴我们可以继续学习交流Java高频面试题。本小节是Java基础篇章的第四小节，主要介绍Java中的**常用集合知识点**，涉及到的内容包括**Java中的三大集合的引出，以及HashMap，Hashtable和ConcurrentHashMap。**

本小节内容几乎是Java面试中必考的点，或者说你是必须要熟练掌握的知识点。在实际的开发的工作中，我们经常借助集合完成数据的排序，查找等操作。熟练掌握Java中的常用集合，对于实际开发工作效率的提升也很有帮助。

我们先来介绍下Java中集合知识的整体情况吧。

## 三大集合接口的引出：

Java中的集合，从上层接口上看分为了两类，**Map和Collection**。也就是说，我们平时接触到的常用的集合，包括HashMap，ArrayList和HashSet等都直接或者间接的实现了这两个接口之一。而Collection接口的子接口又包括了Set和List接口。这样我们**常见的Map，Set和List三大集合接口**就出来了。接口类图如下所示：



这个时候，比较“机灵”的面试官就会发问了。

**面试官：Map，List和Set都是Collection的子接口吗？**

**答：**Map是和Collection并列的集合上层接口，没有继承关系；List和Set是Collection的子接口。

在本小节的附图中，我们给出了本节所涉及到的集合的类图结构，列出来是为了大家学习的时候方便查阅，接下来我们**结合面试题**来进行各个知识点的解析吧。

### (1) 说说Java中常见的集合吧。

**答：**Java中的常见集合可以概括如下。

- **Map接口和Collection接口是所有集合框架的父接口**
- Collection接口的子接口包括：Set接口和List接口
- Map接口的实现类主要有：**HashMap**、TreeMap、Hashtable、LinkedHashMap、**ConcurrentHashMap**以及Properties等
- Set接口的实现类主要有：HashSet、TreeSet、LinkedHashSet等

- List接口的实现类主要有：**ArrayList**、**LinkedList**、Stack以及Vector等

## (2) HashMap和Hashtable的区别有哪些？

17

**答：**HashMap和Hashtable之间的区别可以总结如下。

- HashMap没有考虑同步，是线程不安全的；Hashtable使用了synchronized关键字，是线程安全的；
- HashMap允许null作为Key；Hashtable不允许null作为Key，Hashtable的value也不可以为null

**解析：**

这个算是面试官针对HashMap的一个开胃小菜，重点是根据候选人的回答进行下一步的考察。既然候选人说出了线程安全和不安全的区别，面试官会接着考察线程安全的具体含义，如下所示：

### HashMap是线程不安全的是吧？你可以举一个例子吗？

**答：**（注意，以下是候选人常见的错误理解！！！，因为上边的答案是大家背出来的）

有一个快速失败fast-fail机制，当对HashMap遍历的时候，调用了remove方法使其迭代器发生改变的时候会抛出一个异常**ConcurrentModificationException**。Hashtable因为在方法上做了synchronized处理，所以不会抛出异常。（自信的语气^\_^感觉面试官很low）。

**我们这里先给出正确答案：**

- HashMap线程不安全主要是考虑到了**多线程环境下进行扩容**可能会出现**HashMap死循环**
- Hashtable线程安全是由于其内部实现在put和remove等方法上使用synchronized进行了同步，所以对**单个方法的使用是线程安全的**。但是对多个方法进行**复合操作时，线程安全性无法保证**。比如一个线程在进行get操作，一个线程在进行remove操作，往往会导致下标越界等异常。

既然说到了这里，那么我们来看看大家一直想说的Java集合快速失败（fast-fail）机制是怎么回事儿吧~

### Java集合中的快速失败（fast-fail）机制：

**答：**快速失败是Java集合的一种**错误检测机制**，当多个线程对集合进行结构上的改变的操作时，**有可能会产生fail-fast**。

**例如：**

假设存在两个线程（线程1、线程2），线程1通过Iterator在遍历集合A中的元素，在某个时候线程2**修改了集合A的结构**（是结构上面的修改，而不是简单的修改集合元素的内容），那么这个时候程序就**可能会抛出ConcurrentModificationException异常**，从而产生fast-fail快速失败。

### 那么快速失败机制底层是怎么实现的呢？

迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个 modCount 变量。集合在被遍历期间如果内容发生变化，就会改变modCount的值。当迭代器使用hashNext()/next()遍历下一个元素之前，都会检测modCount变量是否为expectedModCount值，是的话就返回遍历；否则抛出异常，终止遍历。JDK源码中的判断大概是这样的：

```

final Node<K,V> nextNode() {
    Node<K,V>[] t;
    Node<K,V> e = next;
    if (modCount != expectedModCount)
        throw new ConcurrentModificationException();
    if (e == null)
        throw new NoSuchElementException();
    if ((next = (current = e).next) == null && (t = table) != null) {
        do {} while (index < t.length && (next = t[index++]) == null);
    }
    return e;
}

```

17

牛客@我是祖国的花朵

我们再来接着看异常ConcurrentModificationException，JDK中是怎么介绍该异常的：

```

/**
 * This exception may be thrown by methods that have detected concurrent
 * modification of an object when such modification is not permissible.
 *
 * <p>
 * For example, it is not generally permissible for one thread to modify a Collection
 * while another thread is iterating over it. In general, the results of the
 * iteration are undefined under these circumstances. Some Iterator
 * implementations (including those of all the general purpose collection implementations
 * provided by the JRE) may choose to throw this exception if this behavior is
 * detected. Iterators that do this are known as <i>fail-fast</i> iterators,
 * as they fail quickly and cleanly, rather than risking arbitrary,
 * non-deterministic behavior at an undetermined time in the future.
 *
 * <p>
 * Note that this exception does not always indicate that an object has
 * been concurrently modified by a <i>different</i> thread. If a single
 * thread issues a sequence of method invocations that violates the
 * contract of an object, the object may throw this exception. For
 * example, if a thread modifies a collection directly while it is
 *
 * contract of an object, the object may throw this exception. For
 * example, if a thread modifies a collection directly while it is
 * iterating over the collection with a fail-fast iterator, the iterator
 * will throw this exception.
 *
 * <p>Note that fail-fast behavior cannot be guaranteed as it is, generally
 * speaking, impossible to make any hard guarantees in the presence of
 * unsynchronized concurrent modification. Fail-fast operations
 * throw {@code ConcurrentModificationException} on a best-effort basis.
 * Therefore, it would be wrong to write a program that depended on this
 * exception for its correctness: <i>{@code ConcurrentModificationException}</i>
 * should be used only to detect bugs. </i>
 *
 */

```

牛客@我是祖国的花朵

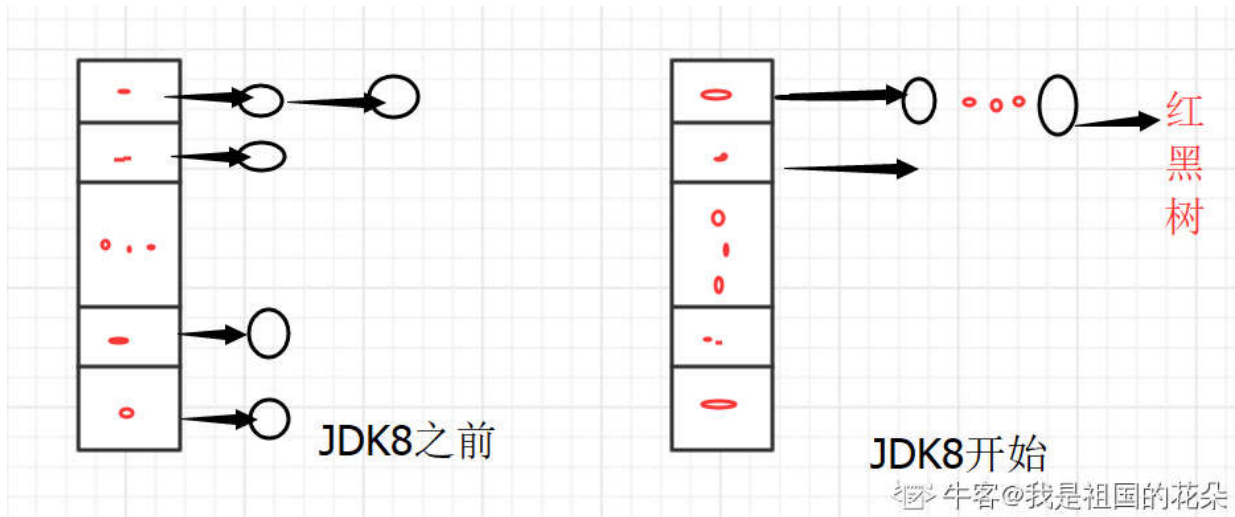
我来解释下JDK中的英文，大概意思就是说当检测到一个并发的修改，就可能会抛出该异常，一些迭代器的实现会抛出该异常，以便可以快速失败。但是你不可以为了便捷而依赖该异常，而应该仅仅作为一个程序的侦测。

前面常见的错误答案，**错误的认为**快速机制就是HashMap线程不安全的表现。并且坚定的认为Hashtable和Vector等线程安全的集合不会存在并发修改时候的快速失败，**这是大错特错**。概念和原理理解的不清晰导致掉入了面试官的陷阱里了，大家可以打开JDK源码，会发现Hashtable也会在迭代的时候抛出该异常，可能发生快速失败。

17

### (3) HashMap底层实现结构有了解吗？

答：HashMap底层实现数据结构为**数组+链表**的形式，JDK8及其以后的版本中使用了**数组+链表+红黑树**实现，解决了链表太长导致的查询速度变慢的问题。大概结构如下图所示：



#### 面试官追问：HashMap的初始容量，加载因子，扩容增量是多少？

答：HashMap的初始容量16，加载因子为0.75，扩容增量是原容量的1倍。如果HashMap的容量为16，一次扩容后容量为32。HashMap扩容是指元素个数（包括数组和链表+红黑树中）超过了 $16 \times 0.75 = 12$ 之后开始扩容。

解析：

这个题目，好多同学表现的不够出色，出现许多记忆不准确的情况。这说明，大家对为什么初始容量是16，扩容后为什么是32的原理不太清晰。那么我们接着看下一个知识点吧，也许会有启发（联想记忆）

#### HashMap的长度为什么是2的幂次方？

答：

- 我们将一个键值对插入HashMap中，通过将Key的hash值与length-1进行&运算，实现了当前Key的定位，2的幂次方可以减少冲突（碰撞）的次数，提高HashMap查询效率
- 如果length为2的幂次方，则length-1 转化为二进制必定是1111.....的形式，在与h的二进制与操作效率会非常的快，而且空间不浪费
- 如果length不是2的幂次方，比如length为15，则length-1为14，对应的二进制为1110，在与h与操作，最后一位都为0，而0001, 0011, 0101, 1001, 1011, 0111, 1101这几个位置永远都不能存放元素了，空间浪费相当大，更糟的是这种情况中，数组可以使用的位置比数组长度小了很多，这意味着进一步增加了碰撞的几率，减慢了查询的效率！这样就会造成空间的浪费。

接下来，我们来做一个简单的总结：

## 总结：

也就是说**2的N次幂有助于减少碰撞的几率**，空间利用率比较大。这样你就明白为什么第一次扩容会从16 -> 32了吧？总不会再说 $32+1=33$ 或者其余答案了吧？至于加载因子，如果设置太小不利于空间利用，设置太大则会导致碰撞增多，降低了查询效率，所以设置了0.75。

17

上边介绍了HashMap在存储空间不足的时候会进行扩容操作。那么，我们接着来看**HashMap中的存储和扩容**等相关知识点吧。

## HashMap的存储和获取原理：

当调用put()方法传递键和值来存储时，先对键调用hashCode()方法，返回的hashCode用于**找到bucket位置来储存Entry对象**，也就是找到了该元素应该被存储的**桶中（数组）**。当两个键的hashCode值相同时，bucket位置发生了冲突，也就是**发生了Hash冲突**，这个时候，会在每一个bucket后边接上一个链表（JDK8及以后的版本中还会加上红黑树）来解决，将新存储的键值对放在表头（也就是bucket中）。

当调用get方法**获取存储的值**时，首先根据键的hashCode找到对应的bucket，然后根据equals方法来在链表和红黑树中找到对应的值。

## HashMap的扩容步骤：

HashMap里面默认的负载因子大小为0.75，也就是说，当Map中的元素个数（**包括数组，链表和红黑树中**）超过了 $16*0.75=12$ 之后开始扩容。将会创建原来HashMap大小的两倍的bucket数组，来重新调整map的大小，并将原来的对象放入新的bucket数组中。这个过程叫作**rehashing**，因为它调用hash方法找到新的bucket位置。

但是，需要注意的是在**多线程环境**下，HashMap扩容可能会导致**死循环**。

前面我们介绍了在**HashMap存储的时候，会发生Hash冲突**，那么我们一起来看Hash冲突的解决办法吧。

## 解决Hash冲突的方法有哪些？

- 拉链法（HashMap使用的方法）
- 线性探测再散列法
- 二次探测再散列法
- 伪随机探测再散列法

## 哪些类适合作为HashMap的键？

String和Integer这样的包装类很适合做为HashMap的键，因为他们是final类型的类，而且**重写了equals和hashCode方法**，避免了键值对改写，有效提高HashMap性能。

为了计算hashCode()，就要防止键值改变，如果键值在放入时和获取时返回不同的hashCode的话，那么就不能从HashMap中找到你想要的对象。

## 扩展知识点：

在高级的算法中，还有一个一致性Hash算法，有能力和精力的同学可以去研究下“**一致性Hash算法**”，有所了解一致性Hash算法对于面试是一个很好的加分点。



## (4) ConcurrentHashMap和Hashtable的区别?

答: **ConcurrentHashMap结合了HashMap和Hashtable二者的优势**。HashMap没有考虑同步, Hashtable考虑了同步的问题。但是Hashtable在每次同步执行时都要锁住整个结构。


17

ConcurrentHashMap锁的方式是稍微细粒度的, ConcurrentHashMap将hash表分为16个桶(默认值), 诸如get, put, remove等常用操作只锁上当前需要用到的桶。

### ConcurrentHashMap的具体实现方式(分段锁):


- 该类包含两个**静态内部类MapEntry和Segment**, 前者用来封装映射表的键值对, 后者用来充当锁的角色。

```
static final class MapEntry<K,V> implements Map.Entry<K,V> {  
    final K key; // non-null  
    V val; // non-null  
    final ConcurrentHashMap<K,V> map;  
    MapEntry(K key, V val, ConcurrentHashMap<K,V> map) {  
        this.key = key;  
        this.val = val;  
        this.map = map;  
    }  
}
```

 牛客@我是祖国的花朵

- Segment**是一种**可重入的锁ReentrantLock**, 每个Segment守护一个HashEntry数组里的元素, 当对HashEntry数组的数据进行修改时, 必须首先获得对应的Segment锁。

```
static class Segment<K,V> extends ReentrantLock implements Serializable {  
    private static final long serialVersionUID = 2249069246763182397L;  
    final float loadFactor;  
    Segment(float lf) { this.loadFactor = lf; }  
}
```

 牛客@我是祖国的花朵

### 解析:

ConcurrentHashMap与Hashtable以及HashMap的比较是一个**绝对高频的考察点**, 我们必须熟练掌握

ConcurrentHashMap分段锁的实现方式。在实际的开发中, 我们在**单线程环境下可以使用HashMap, 多线程环境下可以使用ConcurrentHashMap**, 至于Hashtable已经不被推荐使用了(也就是说Hashtable只存在于面试题目中了)。

## 上节问题解析:

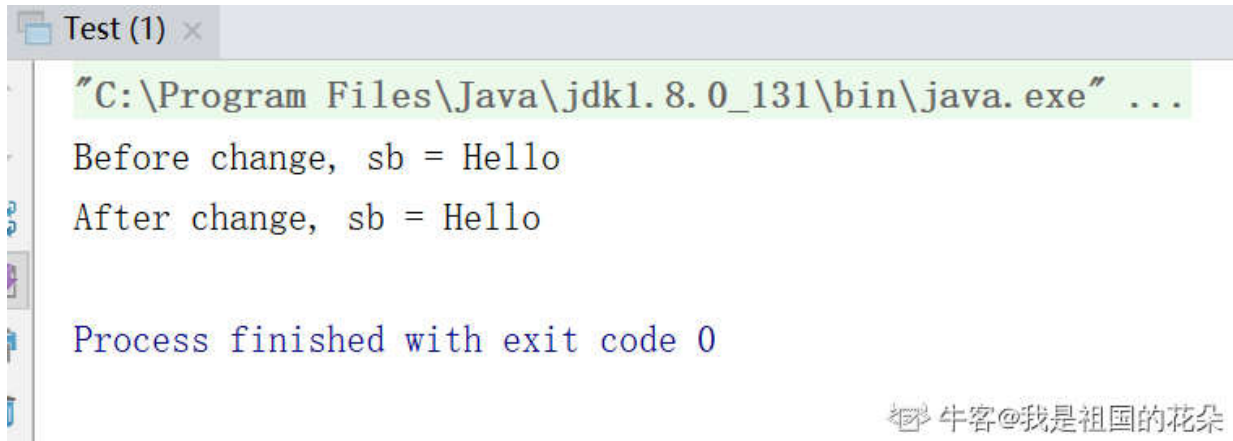
在上一节中, 我们留下了一个题目, 以下代码的输出结果是什么?

```
1 public class Test {  
2     public static void main(String[] args) {
```

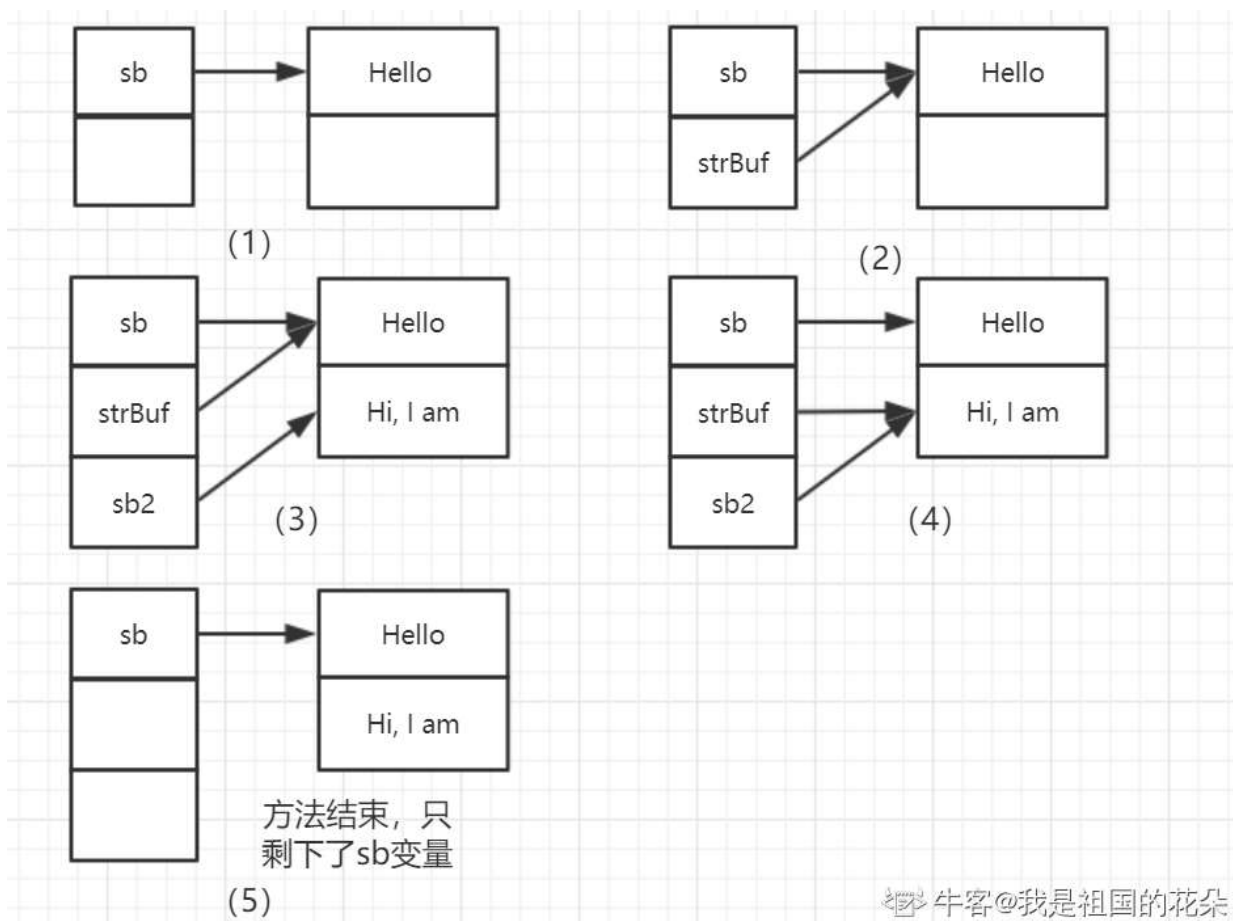
```
3   StringBuffer sb = new StringBuffer("Hello ");
4   System.out.println("Before change, sb = " + sb);
5   changeData(sb);
6   System.out.println("After change, sb = " + sb);
7   }
8   public static void changeData(StringBuffer strBuf) {
9       StringBuffer sb2 = new StringBuffer("Hi, I am ");
10      strBuf = sb2;
11      sb2.append("World!");
12  }
13 }
```

17

首先，我们给出输出答案：



聪明的大家都答对了吗？内存分析图如下所示：



总结：

本小节中，我们交流学习了Java基础中的三大集合，重点阐述了HashMap相关的知识点。这里郑重提示，**本小节所涉及到的内容几乎是面试中的必现考察点**。有能力的同学，最好是**打开JDK的源码，好好研究HashMap以及ConcurentHashMap的实现方式**。当然如果你遇到问题，可以在评论区留言，我们可以一起探讨学习，一起进步。

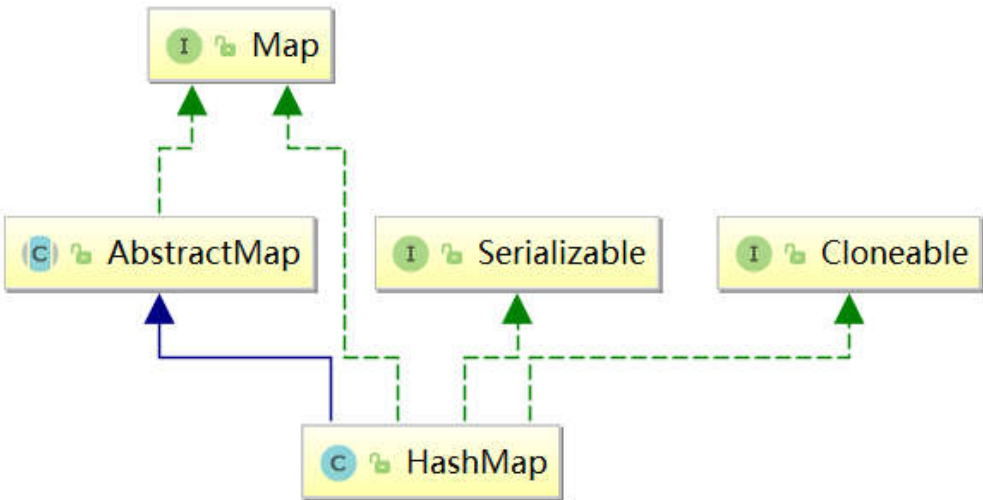
限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

附图：

集合的类图：

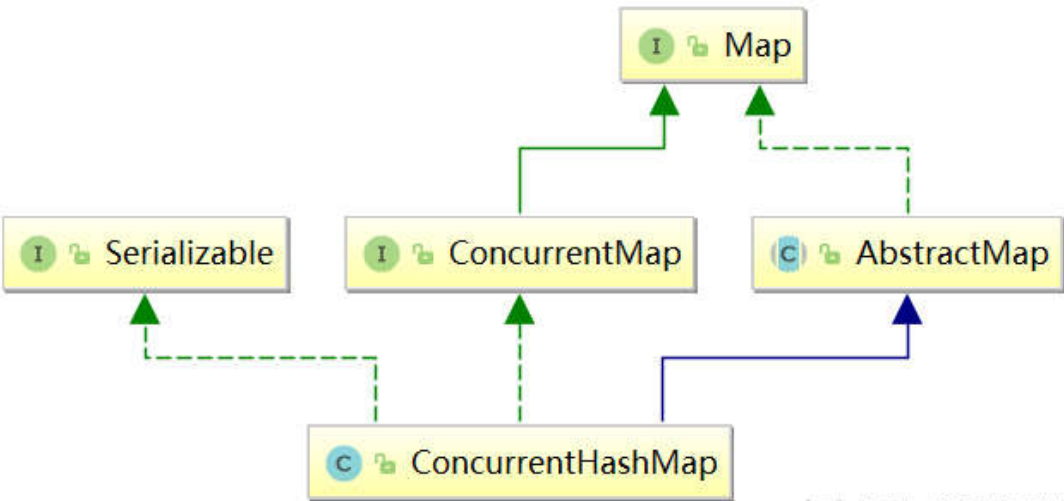
我们来看下本节涉及到的集合的类图结构：（C表示这是一个类，I表示这是一个接口）

- HashMap的类图结构：



牛客@我是祖国的花朵

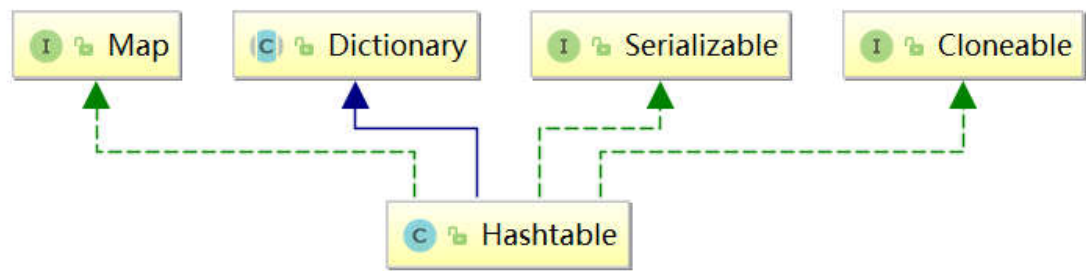
- ConcurrentHashMap的类图结构：



牛客@我是祖国的花朵

- Hashtable的类图结构：





牛客@我是祖国的花朵

讨论

评论



一只没有感情的鸽子

1#

打卡o(\*￣▽￣\*)o

发表于 2019-11-25 17:10:47

赞(1) 回复(0)



菜鸡真的不配有offer

2#

卡打

发表于 2019-11-28 19:15:23

赞(0) 回复(0)



Albattoss1997

3#

ConcurentHashMap是重中之重吧 不管是1.7的分段锁还是1.8的CAS 希望能讲一讲

发表于 2019-12-02 21:54:53

赞(5) 回复(1)

我是祖国的花朵 N 作者： 这块确实是没有准备，有那块想了解的，说出来我们可以一起研究下！  
大概就是JDK1.8的实现降低锁的粒度，JDK1.7版本锁的粒度是基于Segment的，包含多个  
HashEntry，而JDK1.8锁的粒度就是HashEntry（首节点），使用synchronized来进行同步。

2020-02-24 21:43:22

赞(0) 回复(0)

请输入你的观点

回复



刘畅201904211606816

4#

打卡

发表于 2019-12-29 09:44:32

赞(0) 回复(0)



Wadigger

5#

打卡

发表于 2020-01-02 19:59:05

赞(0) 回复(0)



牧水s N

6#

打卡

发表于 2020-01-11 22:15:43

赞(0) 回复(0)



牧水s N

7#

hashmap应该是在容量小于八的时候使用链表，超过8时会改变结构变成红黑树。可以看一下put的流程图有这个操作

17

发表于 2020-01-11 22:17:15

赞(0) 回复(3)

我是祖国的花朵 N (作者) : 是的，链表长度超过8的时候，会将链表转换为红黑树的

2020-02-19 20:32:23

赞(0) 回复(0)

sicnu\_Cxi : 在低于6的时候才会从红黑树转换成链表吧，防止复杂度抖动

2020-02-22 21:23:01

赞(0) 回复(0)

牧水s N : concurrenthashmap是低于六的时候退化为链表，hashmap是高于8转换红黑树

2020-02-23 16:06:05

赞(1) 回复(0)

回复



星如月勿忘初心

8#

打卡

发表于 2020-02-09 15:17:59

赞(1) 回复(0)



亦成风

9#

打卡

发表于 2020-02-22 21:47:02

赞(0) 回复(0)



Gaido

10#

老师，想问一下，HashMap扩容的元素个数是指（包括数组和链表+红黑树中）。还是指链表中的个数(在别的地方看到的)。

发表于 2020-02-24 21:07:00

赞(0) 回复(2)

我是祖国的花朵 N (作者) : 好问题，先来看下HashMap中的这几个概念吧。transient int size: 记录了Map中K-V对的个数; loadFactor: 装载因子，用来衡量HashMap满的程度。loadFactor的默认值为0.75f (static final float DEFAULT\_LOAD\_FACTOR = 0.75f); int threshold: 临界值，当实际K-V个数超过threshold时，HashMap会将容量扩容，threshold = 容量\*加载因子; 容量 (capacity) : 如果不指定，默认容量是16(static final int DEFAULT\_INITIAL\_CAPACITY = 1 << 4); 在扩容的时候，我们比较的是if (++size > threshold) resize(); 总结：当前HashMap中实际拥有的K-V个数超过threshold即进行扩容操作 有能力的同学可以稍微看看源码哦，看不懂就在网上找点源码解析文章互相印证！

2020-02-24 21:39:13

赞(2) 回复(0)

Gaido 回复 我是祖国的花朵 N (作者) : 好的，谢谢老师

2020-02-24 21:58:28

赞(0) 回复(0)

回复

17



牛客248955670号

11#

打卡

发表于 2020-02-25 01:19:41

赞(0) 回复(0)




0XCAFEBABY

12#

老师，想问一下HashMap扩容死循环的问题是因为头插法嘛，JDK1.8改用了尾插法就不会出现死循环的问题了，那么HashMap这个线程安全问题应该怎么答呢？还有面试的时候会让我们详细的说出具体的死循环过程嘛？

发表于 2020-02-25 11:59:27

赞(1) 回复(1)

**我是祖国的花朵**  **作者**： JDK1.8中扩容时候使用了尾插法来解决了之前版本显而易见的死循环问题。但是HashMap的线程不安全问题还体现在数据丢失上，也就是多个线程在进行put操作的时候，存在数据的覆盖问题，所以依然是线程不安全的。另外，有网友在实际使用中确实发现了TreeNode父节点的相互引用还是会造成死循环的。总结起来就是：（1）死循环；（2）数据丢失

2020-02-26 20:15:49

赞(0) 回复(0)

回复



老宋啊啊啊

13#

第二天，打卡，看源码跳来跳去的有点懵...

发表于 2020-03-01 21:30:50

赞(0) 回复(0)



牛客374217958号

14#

hashtable复合操作，多线程这块是不是描述的有什么问题呢？获取的不是对象锁么？


发表于 2020-03-07 21:24:07

赞(1) 回复(2)

**中都**： 我也觉得啊，对于同一个对象来说，多个线程操作它都要获取同一对象锁，不会出现并发问题啊，，，，，

2020-04-20 13:47:54

赞(0) 回复(0)

**我是祖国的花朵**  **作者**： 抱歉，回复有点慢了。这块不安全是指，比如get和put操作。你先get出来value，然后对value++之后，put操作放回去。但是，在两个操作之间，可能该key已经被别的线程操作过了，所以可能你的put操作会与预期效果不符合。这就是线程不安全了，所以说hashtable无法保证复合操作的线程安全。

2020-05-31 11:28:55

赞(0) 回复(0)

[回复](#)

17



zzz8965

15#

学习了，谢谢楼主分享

发表于 2020-03-11 01:29:12

[赞\(0\)](#) [回复\(0\)](#)

她说i

16#

卡

发表于 2020-03-26 14:54:21

[赞\(0\)](#) [回复\(0\)](#)

sky爱吃青菜

17#

## HashMap和Hashtable的区别有哪些？

- 1.是否线程安全
- 2.是否可以存null键，null值
- 3.初始容量，以及扩容方式
  - HashTable的默认初始容量是11，扩容是 $2n+1$
  - HashMap的默认初始容量是16，扩容是 $2*n$

发表于 2020-06-20 11:49:46

[赞\(0\)](#) [回复\(0\)](#)