Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注
4 2018.07.28 10:58:06 字数 4,556 阅读 47,869

> 前言：把这段时间复习的关于集合类的东西整理出来，特别是HashMap相关的一些东西，之前都没有很注意1.7 ->> 1.8的变化问题，但后来发现这其实变化挺大的，而且很多整理的面试资料都没有更新（包括我之前整理的...）

1）说说常见的集合有哪些吧?

答：Map接口和Collection接口是所有集合框架的父接口：

1. Collection接口的子接口包括：Set接口和List接口
2. Map接口的实现类主要有：HashMap、TreeMap、Hashtable、ConcurrentHashMap以及Properties等
3. Set接口的实现类主要有：HashSet、TreeSet、LinkedHashSet等
4. List接口的实现类主要有：ArrayList、LinkedList、Stack以及Vector等

2）HashMap与HashTable的区别?

答：

1. 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；
2. HashMap允许K/V都为null；后者K/V都不允许为null；
3. HashMap继承自AbstractMap类；而Hashtable继承自Dictionary类；

3）HashMap的put方法的具体流程?

推荐阅读
五轮阿里面试题及答案
阅读 12,872
面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651
【JVM系统基础知识】Java 中的并发
阅读 755
两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195
Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637

Java集合必会14问（精选面试题整理）

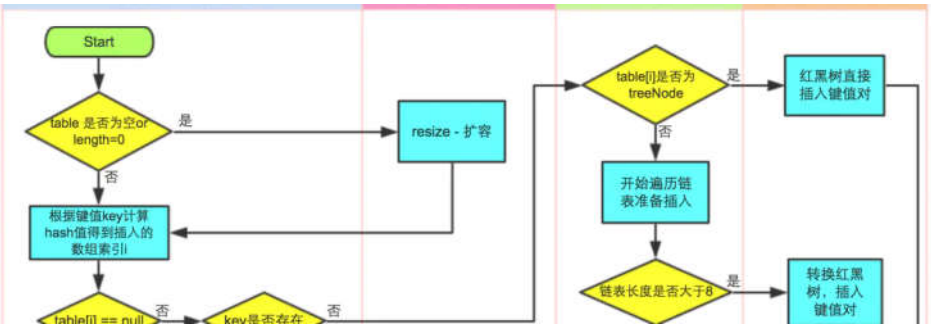
我没有三颗心脏 关注 赞赏支持

```
final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
               boolean evict) {
    HashMap.Node<K,V>[] tab; HashMap.Node<K,V> p; int n, i;
    // 1.如果table为空或者长度为0，即没有元素，那么使用resize()方法扩容
    if ((tab = table) == null || (n = tab.length) == 0)
        n = (tab = resize()).length;
    // 2.计算插入存储的数组索引i，此处计算方法同 1.7 中的indexFor()方法
    // 如果数组为空，即不存在Hash冲突，则直接插入数组
    if ((p = tab[i = (n - 1) & hash]) == null)
        tab[i] = newNode(hash, key, value, null);
    // 3.插入时，如果发生Hash冲突，则依次往下判断
    else {
        HashMap.Node<K,V> e; K k;
        // a.判断table[i]的元素的key是否与需要插入的key一样，若相同则直接用新的value覆盖掉旧的value
        // 判断原则equals() - 所以需要当key的对象重写该方法
        if (p.hash == hash &&
                ((k = p.key) == key || (key != null && key.equals(k))))
            e = p;
        // b.继续判断：需要插入的数据结构是红黑树还是链表
        // 如果是红黑树，则直接在树中插入 or 更新键值对
        else if (p instanceof HashMap.TreeNode)
            e = ((HashMap.TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);
        // 如果是链表，则在链表中插入 or 更新键值对
        else {
            // i .遍历table[i]，判断key是否已存在：采用equals对比当前遍历结点的key与需要插入数据的key
            //    如果存在相同的，则直接覆盖
            // ii.遍历完毕后任务发现上述情况，则直接在链表尾部插入数据
            //    插入完成后判断链表长度是否 > 8：若是，则把链表转换成红黑树
            for (int binCount = 0; ; ++binCount) {
                if ((e = p.next) == null) {
                    p.next = newNode(hash, key, value, null);
                    if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st
                        treeifyBin(tab, hash);
                    break;
                }
                if (e.hash == hash &&
                        ((k = e.key) == key || (key != null && key.equals(k))))
                    break;
                p = e;
            }
        }
        // 对于i 情况的后续操作：发现key已存在，直接用新value覆盖旧value&返回旧value
        if (e != null) { // existing mapping for key
            V oldValue = e.value;
            if (!onlyIfAbsent || oldValue == null)
                e.value = value;
            afterNodeAccess(e);
            return oldValue;
        }
    }
    ++modCount;
    // 插入成功后，判断实际存在的键值对数量size > 最大容量
    // 如果大于则进行扩容
    if (++size > threshold)
        resize();
    // 插入成功时会调用的方法（默认实现为空）
    afterNodeInsertion(evict);
    return null;
}
```

图片简单总结为：

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持



4）HashMap的扩容操作是怎么实现的?

答：通过分析源码我们知道了HashMap通过 `resize()` 方法进行扩容或者初始化的操作，下面是对源码进行的一些简单分析：

```
/**
 * 该函数有2中使用情况：1.初始化哈希表；2.当前数组容量过小，需要扩容
 */
final Node<K,V>[] resize() {
    Node<K,V>[] oldTab = table;// 扩容前的数组（当前数组）
    int oldCap = (oldTab == null) ? 0 : oldTab.length;// 扩容前的数组容量（数组长度）
    int oldThr = threshold;// 扩容前数组的阈值
    int newCap, newThr = 0;

    if (oldCap > 0) {
        // 针对情况2：若扩容前的数组容量超过最大值，则不再扩容
        if (oldCap >= MAXIMUM_CAPACITY) {
            threshold = Integer.MAX_VALUE;
            return oldTab;
        }
        // 针对情况2：若没有超过最大值，就扩容为原来的2倍（左移1位）
        else if ((newCap = oldCap << 1) < MAXIMUM_CAPACITY &&
                 oldCap >= DEFAULT_INITIAL_CAPACITY)
            newThr = oldThr << 1; // double threshold
    }

    // 针对情况1：初始化哈希表（采用指定或者使用默认值的方式）
    else if (oldThr > 0) // initial capacity was placed in threshold
        newCap = oldThr;
    else {               // zero initial threshold signifies using defaults
        newCap = DEFAULT_INITIAL_CAPACITY;
        newThr = (int)(DEFAULT_LOAD_FACTOR * DEFAULT_INITIAL_CAPACITY);
    }

    // 计算新的resize上限
    if (newThr == 0) {
        float ft = (float)newCap * loadFactor;
        newThr = (newCap < MAXIMUM_CAPACITY && ft < (float)MAXIMUM_CAPACITY ?
                  (int)ft : Integer.MAX_VALUE);
    }
    threshold = newThr;
    @SuppressWarnings({"rawtypes","unchecked"})
    Node<K,V>[] newTab = (Node<K,V>[])new Node[newCap];
    table = newTab;
    if (oldTab != null) {
        // 把每一个bucket都移动到新的bucket中去
        for (int j = 0; j < oldCap; ++j) {
            Node<K,V> e;
            if ((e = oldTab[j]) != null) {
                oldTab[j] = null;
                if (e.next == null)
                    newTab[e.hash & (newCap - 1)] = e;
                else if (e instanceof TreeNode)
                    ((TreeNode<K,V>)e).split(this, newTab, j, oldCap);
                else { // preserve order
                    Node<K,V> loHead = null, loTail = null;
                    Node<K,V> hiHead = null, hiTail = null;
                    Node<K,V> next;
                    do {
```

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持

```
                            loTail.next = e;
                        loTail = e;
                    }
                    else {
                        if (hiTail == null)
                            hiHead = e;
                        else
                            hiTail.next = e;
                        hiTail = e;
                    }
                } while ((e = next) != null);
                if (loTail != null) {
                    loTail.next = null;
                    newTab[j] = loHead;
                }
                if (hiTail != null) {
                    hiTail.next = null;
                    newTab[j + oldCap] = hiHead;
                }
            }
        }
    }
    return newTab;
}
```

5）HashMap是怎么解决哈希冲突的?

> 参考资料：https://juejin.im/post/5ab99afff265da23a2291dee

答：在解决这个问题之前，我们首先需要知道**什么是哈希冲突**，而在了解哈希冲突之前我们还要知道**什么是哈希**才行；

什么是哈希?

Hash，**一般翻译为“散列”，也有直接音译为“哈希”的，这就是把任意长度的输入通过散列算法，变换成固定长度的输出，该输出就是散列值（哈希值）；**这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来唯一的确定输入值。**简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。**

所有散列函数都有如下一个基本特性：**根据同一散列函数计算出的散列值如果不同，那么输入值肯定也不同。但是，根据同一散列函数计算出的散列值如果相同，输入值不一定相同。**

什么是哈希冲突?

当两个不同的输入值，根据同一散列函数计算出相同的散列值的现象，我们就把它叫做碰撞（哈希碰撞）。

HashMap的数据结构

在Java中，保存数据有两种比较简单的数据结构：数组和链表。**数组的特点是：寻址容易，插入和删除困难；链表的特点是：寻址困难，但插入和删除容易；**所以我们将数组和链表结合在一起，发挥两者各自的优势，使用一种叫做**链地址法**的方式可以解决哈希冲突：

推荐阅读

五轮阿里面试题及答案
阅读 12,872

面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651

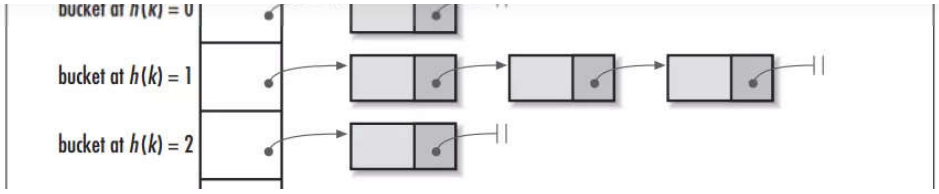
【JVM系统基础知识】Java 中的并发
阅读 755

两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195

Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持



这样我们就可以将拥有相同哈希值的对象组织成一个链表放在hash值所对应的bucket下，**但相比于**hashCode**返回的**int**类型，我们**HashMap**初始的容量大小** `DEFAULT_INITIAL_CAPACITY = 1 << 4` **（即2的四次方16）要远小于**int**类型的范围，所以我们如果只是单纯的用**hashCode**取余来获取对应的**bucket**这将会大大增加哈希碰撞的概率，并且最坏情况下还会将**HashMap**变成一个单链表，**所以我们还需要对hashCode作一定的优化

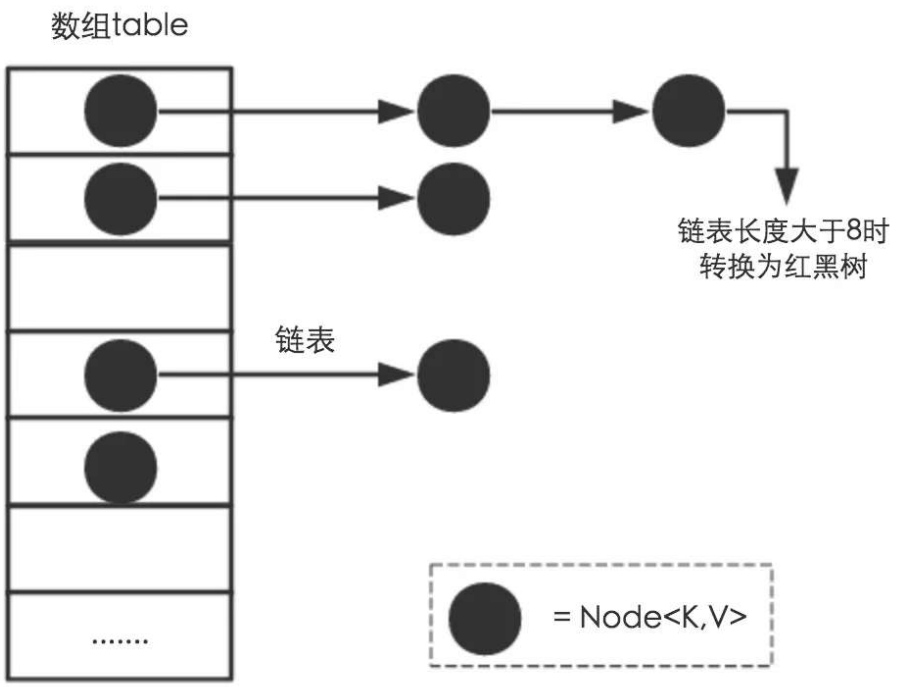
hash()函数

上面提到的问题，主要是因为如果使用hashCode取余，那么相当于**参与运算的只有**hashCode**的低位**，高位是没有起到任何作用的，所以我们的思路就是让hashCode取值出的高位也参与运算，进一步降低hash碰撞的概率，使得数据分布更平均，我们把这样的操作称为**扰动**，在JDK 1.8中的hash()函数如下：

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);// 与自己右移16位进行异或运算（高位
}
```

这比在JDK 1.7中，更为简洁，**相比在**1.7**中的**4**次位运算，**5**次异或运算（**9**次扰动），在**1.8**中，只进行了**1**次位运算和**1**次异或运算（**2**次扰动）；**

JDK1.8新增红黑树



通过上面的**链地址法（使用散列表）**和**扰动函数**我们成功让我们的数据分布更平均，哈希碰撞减少，但是当我们的HashMap中存在大量数据时，加入我们某个bucket下对应的链表有n个元素，那么遍历时间复杂度就为O(n)，为了针对这个问题，JDK1.8在HashMap中新增了红黑树的数据结

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持

简单总结一下HashMap是使用了哪些方法来有效解决哈希冲突的：

1. **使用链地址法（使用散列表）来链接拥有相同hash值的数据；**
2. **使用2次扰动函数（hash函数）来降低哈希冲突的概率，使得数据分布更平均；**
3. **引入红黑树进一步降低遍历的时间复杂度，使得遍历更快；**

6）HashMap为什么不直接使用hashCode()处理后的哈希值直接作为table的下标?

答：`hashCode()`方法返回的是int整数类型，其范围为-(2 ^ 31)~(2 ^ 31 - 1)，约有40亿个映射空间，而HashMap的容量范围是在16（初始化默认值）~2 ^ 30，HashMap通常情况下是取不到最大值的，并且设备上也难以提供这么多的存储空间，从而导致通过`hashCode()`计算出的哈希值可能不在数组大小范围内，进而无法匹配存储位置；

面试官：那怎么解决呢?

答：

1. HashMap自己实现了自己的`hash()`方法，通过两次扰动使得它自己的哈希值高低位自行进行异或运算，降低哈希碰撞概率也使得数据分布更平均；
2. 在保证数组长度为2的幂次方的时候，使用`hash()`运算之后的值与运算（&）（数组长度 - 1）来获取数组下标的方式进行存储，这样一来是比取余操作更加有效率，二来也是因为只有当数组长度为2的幂次方时，h&(length-1)才等价于h%length，三来解决了"哈希值与数组大小范围不匹配"的问题；

面试官：为什么数组长度要保证为2的幂次方呢?

答：

1. 只有当数组长度为2的幂次方时，h&(length-1)才等价于h%length，即实现了key的定位，2的幂次方也可以减少冲突次数，提高HashMap的查询效率；
2. 如果 length 为 2 的次幂 则 length-1 转化为二进制必定是 11111……的形式，在于 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 的次幂，比如 length 为 15，则 length - 1 为 14，对应的二进制为 1110，在于 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。

面试官：那为什么是两次扰动呢?

答：这样就是加大哈希值低位的随机性，使得分布更均匀，从而提高对应数组存储下标位置的随机性&均匀性，最终减少Hash冲突，两次就够了，已经达到了高位低位同时参与运算的目的；

7）HashMap在JDK1.7和JDK1.8中有哪些不同?

答：

不同	JDK 1.7	JDK 1.8

推荐阅读

五轮阿里面试题及答案
阅读 12,872

面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651

【JVM系统基础知识】Java 中的并发
阅读 755

两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195

Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持

初始化方式	单独函数：`inflateTable()`	直接集成到了扩容函数`resize()`中
hash值计算方式	扰动处理 = 9次扰动 = 4次位运算 + 5次异或运算	扰动处理 = 2次扰动 = 1次位运算 + 1次异或运算
存放数据的规则	无冲突时，存放数组；冲突时，存放链表	无冲突时，存放数组；冲突 & 链表长度 < 8：存放单链表；冲突 & 链表长度 > 8：树化并存放红黑树
插入数据方式	头插法（先讲原位置的数据移到后1位，再插入数据到该位置）	尾插法（直接插入到链表尾部/红黑树）
扩容后存储位置的计算方式	全部按照原来方法进行计算（即hashCode ->> 扰动函数 ->> (h&length-1)）	按照扩容后的规律计算（即扩容后的位置=原位置 or 原位置 + 旧容量）

8）为什么HashMap中String、Integer这样的包装类适合作为K?

答：String、Integer等包装类的特性能够保证Hash值的不可更改性和计算准确性，能够有效的减少Hash碰撞的几率

1. 都是final类型，即不可变性，保证key的不可更改性，不会存在获取hash值不同的情况
2. 内部已重写了`equals()`、`hashCode()`等方法，遵守了HashMap内部的规范（不清楚可以去上面看看putValue的过程），不容易出现Hash值计算错误的情况；

面试官：如果我想要让自己的Object作为K应该怎么办呢?

答：重写`hashCode()`和`equals()`方法

1. **重写`hashCode()`是因为需要计算存储数据的存储位置**，需要注意不要试图从散列码计算中排除掉一个对象的关键部分来提高性能，这样虽然能更快但可能会导致更多的Hash碰撞；
2. **重写`equals()`方法**，需要遵守自反性、对称性、传递性、一致性以及对于任何非null的引用值x，x.equals(null)必须返回false的这几个特性，**目的是为了保证key在哈希表中的唯一性**；

9）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。ConcurrentHashMap 锁的方式是稍微细粒度的。

面试官：ConcurrentHashMap的具体实现知道吗?

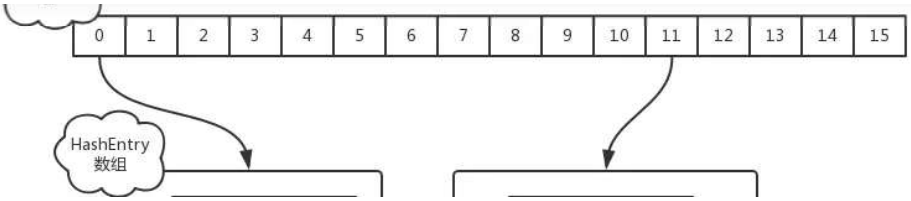
> 参考资料：http://www.importnew.com/23610.html

答：**在JDK1.7中，ConcurrentHashMap采用Segment + HashEntry的方式进行实现**，结构如下：

推荐阅读
五轮阿里面试题及答案
阅读 12,872
面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651
【JVM系统基础知识】Java 中的并发
阅读 755
两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195
Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637

Java集合必会14问（精选面试题整理）

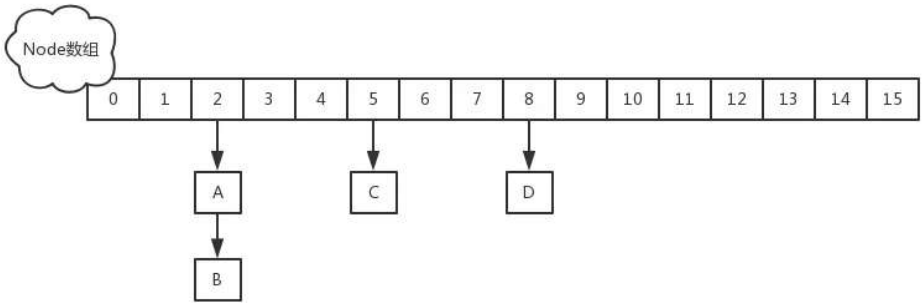
我没有三颗心脏 关注 赞赏支持



1. 该类包含两个静态内部类 HashEntry 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；

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

在JDK1.8**中，放弃了**Segment**臃肿的设计，取而代之的是采用**Node + CAS + Synchronized**来保证并发安全进行实现**，结构如下：



插入元素过程（建议去看看源码）：

1. 如果相应位置的Node还没有初始化，则调用CAS插入相应的数据；

```
else if ((f = tabAt(tab, i = (n - 1) & hash)) == null) {
    if (casTabAt(tab, i, null, new Node<K,V>(hash, key, value, null)))
        break;                   // no lock when adding to empty bin
}
```

2. 如果相应位置的Node不为空，且当前该节点不处于移动状态，则对该节点加synchronized锁，如果该节点的hash不小于0，则遍历链表更新节点或插入新节点；

```
if (fh >= 0) {
    binCount = 1;
    for (Node<K,V> e = f;; ++binCount) {
        K ek;
        if (e.hash == hash &&
            ((ek = e.key) == key ||
             (ek != null && key.equals(ek)))) {
            oldVal = e.val;
            if (!onlyIfAbsent)
                e.val = value;
            break;
        }
        Node<K,V> pred = e;
        if ((e = e.next) == null) {
            pred.next = new Node<K,V>(hash, key, value, null);
            break;
        }
    }
}
```

3. 如果该节点是TreeBin类型的节点，说明是红黑树结构，则通过putTreeVal方法往红黑树中插入

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持

4. 如果插入的是一个新节点，则执行addCount()方法尝试更新元素个数baseCount；

10）Java集合的快速失败机制 “fail-fast”?

答：

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 机制。

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会抛出 ConcurrentModificationException 异常，从而产生fail-fast机制。

原因：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。每当迭代器使用** 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。**

解决办法：

1. **在遍历过程中，所有涉及到改变**modCount**值得地方全部加上**synchronized。

2. **使用**CopyOnWriteArrayList**来替换**ArrayList

11）ArrayList 和 Vector 的区别?

答：

这两个类都实现了 List 接口（List 接口继承了 Collection 接口），他们都是有序集合，即存储在这两个集合中的元素位置都是有顺序的，相当于一种动态的数组，我们以后可以按位置索引来取出某个元素，并且其中的数据是允许重复的，这是与 HashSet 之类的集合的最大不同处，HashSet 之类的集合不可以按索引号去检索其中的元素，也不允许有重复的元素。

ArrayList 与 Vector 的区别主要包括两个方面：

1. 同步性：
   Vector 是线程安全的，也就是说它的方法之间是线程同步（加了synchronized 关键字）的，而 ArrayList 是线程不安全的，它的方法之间是线程不同步的。如果只有一个线程会访问到集合，那最好是使用 ArrayList，因为它不考虑线程安全的问题，所以效率会高一些；如果有多个线程会访问到集合，那最好是使用 Vector，因为不需要我们自己再去考虑和编写线程安全的代码。

2. 数据增长：
   ArrayList 与 Vector 都有一个初始的容量大小，当存储进它们里面的元素的个人超过了容量时，就需要增加 ArrayList 和 Vector 的存储空间，每次要增加存储空间时，不是只增加一个存储单元，而是增加多个存储单元，每次增加的存储单元的个数在内存空间利用与程序效率之间要去的一定的平衡。Vector 在数据满时（加载因子1）增长为原来的两倍（扩容增量：原容量的 2 倍），而 ArrayList 在数据量达到容量的一半时（加载因子 0.5）增长为原容量的 (0.5 倍 + 1) 个空间。

推荐阅读

五轮阿里面试题及答案
阅读 12,872

面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651

【JVM系统基础知识】Java 中的并发
阅读 755

两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195

Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持

1. LinkedList 实现了 List 和 Deque 接口，一般称为双向链表；ArrayList 实现了 List 接口，动态数组；
2. LinkedList 在插入和删除数据时效率更高，ArrayList 在查找某个 index 的数据时效率更高；
3. LinkedList 比 ArrayList 需要更多的内存；

面试官：Array 和 ArrayList 有什么区别？什么时候该应 Array 而不是 ArrayList 呢？

答：它们的区别是：

1. Array 可以包含基本类型和对象类型，ArrayList 只能包含对象类型。
2. Array 大小是固定的，ArrayList 的大小是动态变化的。
3. ArrayList 提供了更多的方法和特性，比如：addAll()，removeAll()，iterator() 等等。

对于基本类型数据，集合使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式相对比较慢。

13）HashSet是如何保证数据不可重复的？

答：HashSet的底层其实就是HashMap，只不过我们**HashSet是实现了Set接口并且把数据作为K值，而V值一直使用一个相同的虚值来保存**，我们可以看到源码：

```
public boolean add(E e) {
    return map.put(e, PRESENT)==null;// 调用HashMap的put方法,PRESENT是一个至始至终都相同的虚值
}
```

由于HashMap的K值本身就不允许重复，并且在HashMap中如果K/V相同时，会用新的V覆盖掉旧的V，然后返回旧的V，那么在HashSet中执行这一句话始终会返回一个false，导致插入失败，这样就保证了数据的不可重复性；

14）BlockingQueue是什么？

答：

Java.util.concurrent.BlockingQueue是一个队列，在进行检索或移除一个元素的时候，它会等待队列变为非空；当在添加一个元素时，它会等待队列中的可用空间。BlockingQueue接口是Java集合框架的一部分，主要用于实现生产者-消费者模式。我们不需要担心等待生产者有可用的空间，或消费者有可用的对象，因为它都在BlockingQueue的实现类中被处理了。Java提供了集中BlockingQueue的实现，比如ArrayBlockingQueue、LinkedBlockingQueue、PriorityBlockingQueue,、SynchronousQueue等。

> 欢迎转载，转载请注明出处！
> 简书ID：@我没有三颗心脏
> github：wmyskxz
> 欢迎关注公众微信号：wmyskxz_javaweb
> 分享自己的Java Web学习之路以及各种Java学习资料
> 想要交流的朋友也可以加qq群：3382693

推荐阅读

五轮阿里面试题及答案
阅读 12,872

面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651

【JVM系统基础知识】Java 中的并发
阅读 755

两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195

Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637

Java集合必会14问（精选面试题整理）

我没有三颗心脏 关注 赞赏支持

还没有人赞赏，支持一下

我没有三颗心脏 独立域名博客：https://www.wmyskxz.com 分享知识&技术&思...
总资产451 (约38.24元) 共写了44.3W字 获得3,087个赞 共2,888个粉丝
关注

被以下专题收入，发现更多相似内容

程序员 java Java Java Java之路 Java基础知识 面试相关 展开更多

推荐阅读

更多精彩内容 >

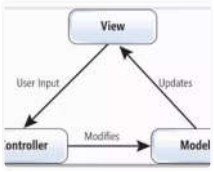
Java面试宝典Beta5.0

pdf下载地址：Java面试宝典 第一章内容介绍 20 第二章JavaSE基础 21 一、Java面向对象 21 ...

王震阳 阅读 80,425 评论 26 赞 514

【JAVA】面试宝典

Java中如何实现代理机制(JDK、CGLIB) JDK动态代理：代理类和目标类实现了共同的接口，用到Invoca...



Y了个J 阅读 1,952 评论 1 赞 13

2017年11月15号星期三晴

今天早上出门送菌结果把钥匙手机都锁家里了，好在是二楼，窗户没关找梯子从窗户爬进家了，唉！这都第二次了。下午接菌她下...

白塔实验小学一年级三班任晴菌 阅读 35 评论 0 赞 0

2018年5月30号星期三亲子日记第八十九篇

今天中午孙星堂放学回来就和我说妈妈下午穿新校服，我说好的我给你洗洗澡吧，他说正好好热的洗洗澡。等...

孙星堂妈妈 阅读 33 评论 0 赞 1

家乡的味道：莲藕糖

文|吴少如 荷塘里碧叶连天，白的粉的荷花争相怒放，格外惹人喜爱。更惹人喜爱的还有莲藕、莲角、莲子。采藕人是最辛苦的...

吴少如 阅读 557 评论 0 赞 0

推荐阅读

五轮阿里面试题及答案
阅读 12,872

面试阿里P6，竟被MySQL难倒，再次二战，进入天猫团队（研发岗）
阅读 12,651

【JVM系统基础知识】Java 中的并发
阅读 755

两年开发经验，跳槽字节跳动，三面斩获Offer，定级P6（Java研发岗，...
阅读 1,195

Java性能优化JVM、Tomcat、MySQL面试题一次彻底扫盲
阅读 637