**Java基础**

# HashMap

## 为什么浮在因子设置为0.75

<https://juejin.im/post/5e532459e51d4526cd1de801>

HashMap采用的结构为数组+链表+红黑树(1.8),利用hash算法将数据分配到各个桶中，如果两个值计算出的hash不一样，那么这个两个值一定不相同，如果一样，这两个值有可能相同，也有可能不相同，这就发生了hash碰撞，发生了hash碰撞，hashmap将第二个值加第一个值的链表中，如果发生碰撞的概率大，那么hashmap就会退化成一个链表，导致查询的效率低下。如果减小发生碰撞的概率，有两个方法:1.扩容 2.设计一个良好的hash算法

什么时候扩容，在hashmap中是的达到了临界值的时候就会扩容，临界值=负载因子\*容量，如果负载因子过小，那么会导致频繁的扩容，耗费内存空间，过大，就会导致hash碰撞的概率增加，会导致查询的速度降低，0.75不大也不小，而且\*2的幂是整数，所以选择0.75是合适的。

# ConcurrentHashMap源码

# 三种for循环区别

1. 如果不满足条件判断，那么**for循环和while循环就不会执行**，do while循环会执行至少一次。
2. for循环的变量再小括号中定义，只有循环内部才可以使用。（初始化语句不可以输出）。
3. While循环和do-while循环初始化语句本来就在外面，所以出来循环之后还可以继续使用。（初始化语句可以输出）

# equals和hashcode的作用

1.若重写了equals(Object obj)方法，则有必要重写hashCode()方法。

2.若两个对象equals(Object obj)返回true，则hashCode（）有必要也返回相同的int数。

3.若两个对象equals(Object obj)返回false，则hashCode（）不一定返回不同的int数。

4.若两个对象hashCode（）返回相同int数，则equals（Object obj）不一定返回true。

5.若两个对象hashCode（）返回不同int数，则equals（Object obj）一定返回false。

6.同一对象在执行期间若已经存储在集合中，则不能修改影响hashCode值的相关信息，否则会导致内存泄露问题。

一般来说涉及到对象之间的比较大小就需要重写equals方法，

# string和stringbuilder、stringbuffer的区别

**String是只读字符串，所引用的字符串不能被改变，一经定义，无法再增删改。**

**String 定义的字符串保存在常量池里面，进行+操作时不能直接在原有基础上拼接。**

**每次+操作 ： 隐式在堆上new了一个跟原字符串相同的StringBuilder对象，再调用append方法 拼接+后面的字符。**

String和Stringbuilder在单线程环境下使用；

StringBuffer在多线程环境下使用，可以保证线程同步；

Stringbuilder 和StringBuffer 实现方法类似，均表示可变字符序列，不过StringBuffer 用synchronized关键字修饰（保证线程同步）

# 容器

## ArrayLIst、LinkedLIst的区别

1. ArrayList的实现是基于数组来实现的，LinkedList的基于双向链表来实现。这两个数据结构的逻辑关系是不一样，当然物理存储的方式也会是不一样。

2. 对于随机访问，ArrayList优于LinkedList。

3. 对于插入和删除操作，LinkedList优于ArrayList

1. LinkedList比ArrayList更占内存，因为LinkedList的节点除了存储数据，还存储了两个引用，一个指向前一个元素，一个指向后一个元素。

## Hashmap

### ****谈一下HashMap的特性？****

1.HashMap存储键值对实现快速存取，允许为null。key值不可重复，若key值重复则覆盖。

2.非同步，线程不安全。

1. 底层是hash表，不保证有序(比如插入的顺序)

### ****谈一下HashMap的底层原理是什么？****

基于hashing的原理，jdk8后采用数组+链表+红黑树的数据结构。我们通过put和get存储和获取对象。当我们给put()方法传递键和值时，先对键做一个hashCode()的计算来得到它在bucket数组中的位置来存储Entry对象。当获取对象时，通过get获取到bucket的位置，再通过键对象的equals()方法找到正确的键值对，然后在返回值对象。

### ****谈一下hashMap中put是如何实现的？****

**1.计算关于key的hashcode值（与Key.hashCode的高16位做异或运算）**

**2.如果散列表为空时，调用resize()初始化散列表**

**3.如果没有发生碰撞，直接添加元素到散列表中去**

**4.如果发生了碰撞(hashCode值相同)，进行三种判断**

**4.1:若key地址相同或者equals后内容相同，则替换旧值**

**4.2:如果是红黑树结构，就调用树的插入方法**

**4.3：链表结构，循环遍历直到链表中某个节点为空，尾插法进行插入，插入之后判断链表个数是否到达变成红黑树的阙值8；也可以遍历到有节点与插入元素的哈希值和内容相同，进行覆盖。**

**5.如果桶满了大于阀值，则resize进行扩容**

### ****谈一下hashMap中什么时候需要进行扩容，扩容resize()又是如何实现的？****

1.初始化数组table

2.当数组table的size达到阙值时即++size > load factor \* capacity 时，也是在putVal函数中

实现过程：(细讲)

1.通过判断旧数组的容量是否大于0来判断数组是否初始化过

否：进行初始化

判断是否调用无参构造器，

是:使用默认的大小和阙值

否:使用构造函数中初始化的容量，当然这个容量是经过tableSizefor计算后的2的次幂数

是，进行扩容，扩容成两倍(小于最大值的情况下)，之后在进行将元素重新进行与运算复制到新的散列表中

概括的讲：扩容需要重新分配一个新数组，新数组是老数组的2倍长，然后遍历整个老结构，把所有的元素挨个重新hash分配到新结构中去。

### 谈一下hashMap中get是如何实现的？

对key的hashCode进行hashing，与运算计算下标获取bucket位置，如果在桶的首位上就可以找到就直接返回，否则在树中找或者链表中遍历找，如果有hash冲突，则利用equals方法去遍历链表查找节点。

### 谈一下HashMap中hash函数是怎么实现的？还有哪些hash函数的实现方式？

对key的hashCode做hash操作，与高16位做异或运算

还有平方取中法，除留余数法，伪随机数法

### ****.为什么不直接将key作为哈希值而是与高16位做异或运算？****

因为数组位置的确定用的是与运算，仅仅最后四位有效，设计者将key的哈希值与高16为做异或运算使得在做&运算确定数组的插入位置时，此时的低位实际是高位与低位的结合，增加了随机性，减少了哈希碰撞的次数。

HashMap默认初始化长度为16，并且每次自动扩展或者是手动初始化容量时，必须是2的幂。

### .为什么是16？为什么必须是2的幂？如果输入值不是2的幂比如10会怎么样？

<https://blog.csdn.net/sidihuo/article/details/78489820>

<https://blog.csdn.net/eaphyy/article/details/84386313>

1.为了数据的均匀分布，减少哈希碰撞。因为确定数组位置是用的位运算，若数据不是2的次幂则会增加哈希碰撞的次数和浪费数组空间。(PS:其实若不考虑效率，求余也可以就不用位运算了也不用长度必需为2的幂次)

1. 输入数据若不是2的幂，HashMap通过一通位移运算和或运算得到的肯定是2的幂次数，并且是离那个数最近的数字

### ****.谈一下当两个对象的hashCode相等时会怎么样？****

会产生哈希碰撞，若key值相同则替换旧值，不然链接到链表后面，链表长度超过阙值8就转为红黑树存储

### .如果两个键的hashcode相同，你如何获取值对象？

HashCode相同，通过equals比较内容获取值对象

### ****."如果HashMap的大小超过了负载因子(load factor)定义的容量，怎么办？****

超过阙值会进行扩容操作，概括的讲就是扩容后的数组大小是原数组的2倍，将原来的元素重新hashing放入到新的散列表中去。

### .HashMap和HashTable的区别

相同点：都是存储key-value键值对的

不同点：

HashMap允许Key-value为null，hashTable不允许；

hashMap没有考虑同步，是线程不安全的。hashTable是线程安全的，给api套上了一层synchronized修饰;

HashMap继承于AbstractMap类，hashTable继承与Dictionary类。

迭代器(Iterator)。HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException。

容量的初始值和增加方式都不一样：HashMap默认的容量大小是16；增加容量时，每次将容量变为"原始容量x2"。Hashtable默认的容量大小是11；增加容量时，每次将容量变为"原始容量x2 + 1"；

添加key-value时的hash值算法不同：HashMap添加元素时，是使用自定义的哈希算法。Hashtable没有自定义哈希算法，而直接采用的key的hashCode()。

### ****请解释一下HashMap的参数loadFactor，它的作用是什么？****

loadFactor表示HashMap的拥挤程度，影响hash操作到同一个数组位置的概率。默认loadFactor等于0.75，当HashMap里面容纳的元素已经达到HashMap数组长度的75%时，表示HashMap太挤了，需要扩容，在HashMap的构造器中可以定制loadFactor。

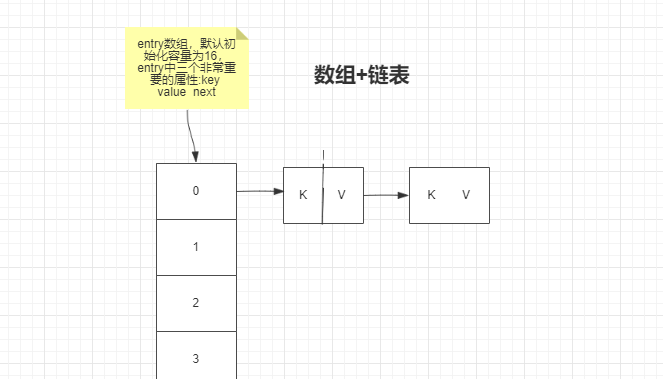
### ****传统hashMap的缺点(为什么引入红黑树？)：****

JDK 1.8 以前 HashMap 的实现是 数组+链表，即使哈希函数取得再好，也很难达到元素百分百均匀分布。当 HashMap 中有大量的元素都存放到同一个桶中时，这个桶下有一条长长的链表，这个时候 HashMap 就相当于一个单链表，假如单链表有 n 个元素，遍历的时间复杂度就是 O(n)，完全失去了它的优势。针对这种情况，JDK 1.8 中引入了 红黑树（查找时间复杂度为 O(logn)）来优化这个问题。

### ****平时在使用HashMap时一般使用什么类型的元素作为Key？****

选择Integer，String这种不可变的类型，像对String的一切操作都是新建一个String对象，对新的对象进行拼接分割等，这些类已经很规范的覆写了hashCode()以及equals()方法。作为不可变类天生是线程安全的

### 1.7版本和1.8版本HashMap的区别



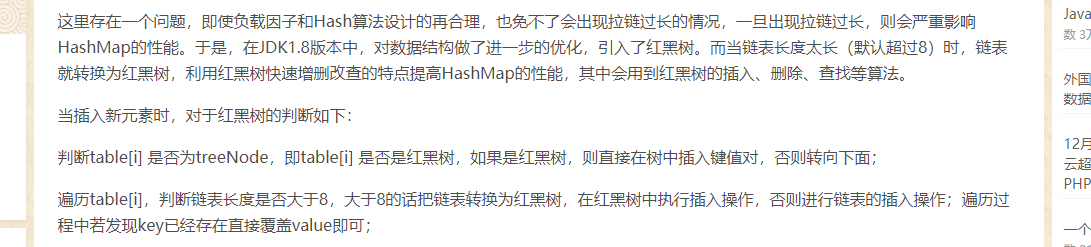
回答的时候:

1. 数组和链表的优缺点

数组: 数组存储区间是连续的，占用内存严重，故空间复杂的很大。但数组的二分查找时间复杂度小，为O(1)；数组的特点是：寻址容易，插入和删除困难；

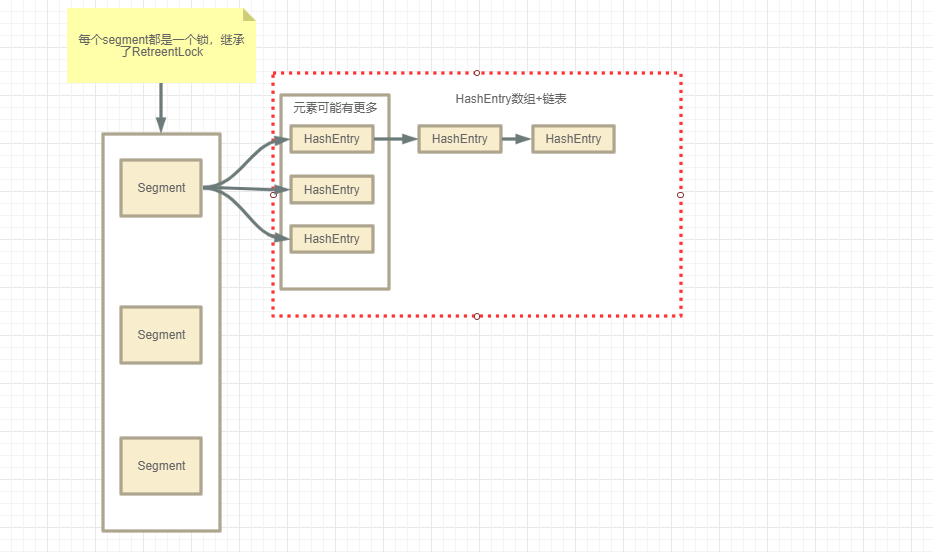
链表: 链表存储区间离散，占用内存比较宽松，故空间复杂度很小，但时间复杂度很大，达O（N）。链表的特点是：寻址困难，插入和删除容易。

1. 先说出结构是什么(数组(存储的元素类型为entry:key、value、next)+链表)
2. 如何put元素
3. Put元素的时候，hash冲突的怎么办?(解决hash冲突的方法:开放地址法，链地址法，再hash法，建立一个公共溢出区。)
4. 如何get元素
5. Java8性能改善:



### 1.7版本和1.8版本ConcurrentHashMap的区别

#### 1.7



1. 结构(segment数组+HashEntry数组+链表)

Segment继承RetreentLock，每个Segment对象是一把锁，Segment数组的意义是把一个大的table划分成为一个个小的table来加锁，即利用了分段锁的技术来提高并发性。

1. Put操作:

Put操作时，先同过hash计算出Segment的位置，如果Segment没有初始化，即通过CAS进行操作，然后进行第二次hash操作，找到对应的hashentry的位置，这里会利用继承过来的锁的特性，在将数据插入指定的HashEntry的位置时，会通过tryLock方法尝试获取锁，如果获取成功，就直接插入，没有成功会通过自旋的方式继续调用tryLock方法获取锁，超过指定次数就挂起，等待被唤醒。

1. get操作

通过hash计算出Segment的位置，然后再hash定位到指定的HashEntry，遍历该HashEntry下的链表，成功就返回，不成功就返回null

1. Size

方案一:使用不加锁的方式取尝试多次计算ConcurrentHashMap的size，最多三次，比较前后两次计算的结果，结果一致就认为当前没有元素加入，计算的结果是准确的，计算出的值即为当前的size

方案二：如果第一种方案不符合，就会对每个Segment加上锁，然后计算ConcurrentHashMap的size返回。

#### 1.8

# 多线程

## ThreadLocal

