链滴

后端 | Java 常用容器知识速成

作者：z875479694h

原文链接：https://ld246.com/article/1626110912311

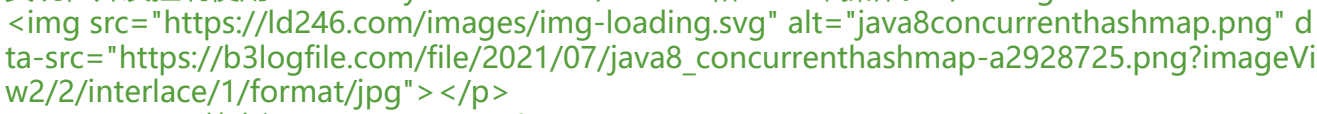
来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20181226.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="一-继承关系">一.继承关系</h2>
<h2 id="1-1-基于Collection">1.1 基于 Collection</h2>
Iterable 接口： 迭代器（负责迭代元素，用于遍历元素）
Collection 接口： 集合（常用的方法有添加，全部添加，删除，清空集合，是
存在，集合个数等）
List 接口： 列表 （list 提供比数组更丰富的 API，有序，可重复，）
Queue 接口： 队列（遵循先进先出原则，第一个进队列的元素，必定第一个
队列）
Set 接口： Set 集合（不允许出现重复元素，并且无序的集合）
<img src="https://ld246.com/images/img-loading.svg" alt="容器继承关系图" data-src="ht
ps://b3logfile.com/siyuan/1608522634335/assets/java-collection-hierarchy-20210322212932
njsrel6.png?imageView2/2/interlace/1/format/jpg">
<h2 id="1-2-不属于Collection">1.2 不属于 Collection</h2>
Map 接口： 散列表（使用 K-V（键值对）存储的一种数据结构，Key 是
序的、不可重复的，value 是无序的、可重复的）不属于 Collection 接口
<img src="https://ld246.com/images/img-loading.svg" alt="Java Map Hierarchy" data-sr
="https://b3logfile.com/file/2021/07/solo-fetchupload-5675258265756846954-830450f2.png
imageView2/2/interlace/1/format/jpg">
<h2 id="二-List">二.List</h2>
<h2 id="2-1-ArrayList">2.1 ArrayList</h2>
底层实现： Objcet[]（动态数组)
扩容方式： 当原本 ArrayList 的底层的数组容量不足以存放新插入的元素
一组元素时，会触发扩容机制，调用的本地 C 方法。
线程安全性： 不安全
补充：如果要安全的调用 ArrayList，使用 Collections.synchronizedList()方法。其他 API
与原先无异
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">List<</
pan>Map<String,<span class="highligh
-n">Object>> <span class="highlight-n"
data=Collections</s
an>.synchronizedList</spa
>(new <span class=
highlight-n">ArrayList<<span class="highligh
-n">Map<String<
span>,Object<span c
ass="highlight-o">>>());
</code></pre>
使用场景：
适合于进行大量的随机访问的情况下使用（说人话就是可以直接根据下标取值），时间复杂度 O(1)</
>
插入与删除： 默认追加到尾部，时间复杂度 O(1)。追加或删除指定位置
素，时间复杂度 O(n-i)
补充：clear()方法可以清空当前 List 且不改动已分配的空间
<h2 id="2-2-LinkedList">2.2 LinkedList</h2>
底层实现： 双向链表
扩容方式： 正常链表的新增节点
线程安全性： 不安全
补充：如果要安全的调用，同上。其他 API 与原先无异
<pre><code class="language-java highlight-chroma"><span c
ass="highlight-cl">List<</

pan>Map<String,Object>> <span class="highlight-n" data=Collections</s an>.synchronizedList</spa >(new LinkedList<Map<String /span>,Object>>());
</code></pre>
使用场景：
适合频繁的插入或者删除操作，较少的随机访问（说人话就是可以直接根据下标取值），获取元素的间复杂度 O(n)。
插入与删除： 默认追加到尾部，时间复杂度<strong约等于 O(1)（为啥不等于一，因为还要创建节点然后追加，而 ArrayList 直接分配好空间接赋值就行）。追加或删除指定位置元素，时间复杂度近似 O(n)。
内存空间占用：
ArrayList 的空间浪费体现在 list 列表的结尾会预留一定的容量空间</trong>。
LinkedList 的空间花费主要体现在每一个元素都（存放直接后继和直前驱以及数据） 需要消耗比 ArrayList 更多的空间。
<h2 id="2-3-CopyOnWriteArrayList-不常用-">2.3 CopyOnWriteArrayList（不常用）</h2>
底层实现： Objcet[]（动态数组)
线程安全性： 安全
补充：如何保证安全性，
读：未加锁。
写：添加或修改时元素时 使用 lock()进行加锁并复制 一份 副本，然后添加或改，然后使用新副本。
使用场景：
适合于进行大量的随机访问的情况下使用（说人话就是可以直接根据下标取值）以及读多写少的情况不需要强一致性（不能保证即时一致性），时间复杂度 O(1)。
内存空间占用：
每一次写都需要复制一份副本。严重浪费内存。
<h2 id="2-4-Vector-历史遗留-古早时代的类-">2.4 Vector（历史遗留，古早时代的类）</h2>
底层实现： Objcet[]（动态数组)
线程安全性： 安全
补充：如何保证安全性，方法全部加了 <code>synchronized</code> 关键字（恶臭的史残留。进了用这玩意的公司赶紧跑路吧，代码一定是屎山）
<h2 id="三-Set">三.Set</h2>
<h2 id="3-1-HashSet">3.1 HashSet</h2>
底层实现： HashMap(使用了 Key 部分)
线程安全性： 不安全
使用场景： 无序，数据去重
允许插入 Null 值
<h2 id="3-2-TreeSet">3.2 TreeSet</h2>
底层实现： TreeMap，红黑树（自平衡二叉查找树）
线程安全性： 不安全
使用场景： 有序（自然排序，key 的开头第一个字符的顺序 a-z，或者数的大小 0-9）或自定义排序，数据去重
允许插入 Null 值
注意：要安全调用非线程安全的 Set。调用 <code>Collections.synchronizedSet()</ode>
<h2 id="四-Map">四.Map</h2>

<h2 id="4-1-HashMap">4.1 HashMap</h2>
底层实现： 哈希表（又称散列表，使用杂凑算法），当链表（放相同 Hash 值的 Value）长度大于阈值（默认为 8）（将链表转换成红黑树前会判断，如果当前数的长度小于 64（存 Key），那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为黑树，以减少搜索时间。
线程安全性： 不安全
扩容机制：
源码规定了默认扩容加载因子为 0.75
面试官：为什么是 0.75？
如果是 0.5，就是说哈希表填到一半就开始扩容了，这样会导致扩容频繁，并且空间利用率较低。 如果是 1，就是说哈希表完全填满才开始扩容，这样虽然空间利用提高了，但是哈希冲突机会大了。
负载因子 0.75 就是冲突的机会 与空间利用率权衡的最后体现，也是实验的经验值。
<pre><code class="highlight-chroma">static final float DEFAULT_LOAD_FACTOR = 0.75f;
</code></pre>
使用场景： 需要根据键的 hashCode 值进行存储数据。无序，获取元素的间复杂度限制为 O(1)的时候。
Key 与 Value 可为 null
内存空间占用：
① 默认的初始化大小为 16。之后每次扩充，容量变为原来的 2 倍。
面试官为什么是 16？
如果太小，4 或者 8，扩容比较频繁；如果太大，32 或者 64 甚至太大，又占用内存空间。
位运算更快，不需十进制和二进制相互转换。
<pre><code class="highlight-chroma">static final int DEFAULT_INITIAL_CAPACITY = 1 << 4; // aka 16
</code></pre>
② 创建时如果给定了容量初始值 <code>HashMap</code> 会将其扩充为 2 的幂次方大小，就是说 <code>HashMap</code> 总是使用 2 的幂作为哈希表的大小。
简单的杂凑算法演示： 设对 7 求膜，
新增 key：6，value："随便"
6%7=6
<table>
<thead>
<tr>
<th>数组</th>
<th>数组 0</th>
<th>数组 1</th>
<th>数组 2</th>
<th>数组 3</th>
<th>数组 4</th>
<th>数组 5</th>
<th>数组 6</th>
<th>数组 7</th>
<th>数组 8</th>
<th>数组 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>null</td>

<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>6</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>Value</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>"随便"</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
新增 key：7 ， value："第二次"
7%7=0
<table>
<thead>
<tr>
<th>数组</th>
<th>数组 0</th>
<th>数组 1</th>
<th>数组 2</th>
<th>数组 3</th>
<th>数组 4</th>
<th>数组 5</th>
<th>数组 6</th>
<th>数组 7</th>
<th>数组 8</th>
<th>数组 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>7</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>6</td>
<td>null</td>

<td>null</td>
<td>null</td>
</tr>
<tr>
<td>Value</td>
<td>"第二次"</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>"随便"</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
解决 Hash 冲突： 拉链法（以链表的形式存放相同 Hash 值的 Value），文的红黑树也可。但是拉链法较简单。
<h2 id="4-2-TreeMap">4.2 TreeMap</h2>
底层实现： 红黑树
线程安全性： 不安全
使用场景： 有序（自然排序，key 的开头第一个字符的顺序 a-z，或者数的大小 0-9）或自定义排序。
Key 与 Value 可为 null
注意：要安全调用非线程安全的 Map。调用 <code>Collections.synchronizedMap(</code>
<h2 id="4-3-ConcurrentHashMap-建议使用-">4.3 ConcurrentHashMap（建议使用）</h2>
底层实现： 数组 + 链表/红黑二叉树(JDK1.8)在链表长度超过一定阈值（8时将链表（寻址时间复杂度为 O(N)）转换为红黑树（寻址时间复杂度为 O(log(N)））
线程安全性： 安全
保证线程安全的方法：用 <code>Node</code> 数组 + 链表 + 红黑树的数据结构实现，并发控制使用 <code>synchronized</code> 和 CAS 来操作。


效率问题：
<code>synchronized</code> 只锁定当前链表或红黑二叉树的首节点，这样只要 hash 不冲突，不会产生并发，效率又提升 N 倍。
<h2 id="4-4-Hashtabel-古早的线程安全类-">4.4 Hashtabel（古早的线程安全类）</h2>
如何保证安全性，方法全部加了 <code>synchronized</code> 关键字（恶臭的历残留。进了用这玩意的公司赶紧跑路吧，代码一定是屎山）
<h2 id="资料参考">资料参考</h2>
JavaGuide：<a href="https://ld246.com/forward?goto=https%3A%2F%2Fsnailclimb.gitee
io%2Fjavaguide" target="_blank" rel="nofollow ugc">https://snailclimb.gitee.io/javaguide</
>
CSDN：<a href="https://ld246.com/forward?goto=https%3A%2F%2Fblog.csdn.net" target="
blank" rel="nofollow ugc">https://blog.csdn.net