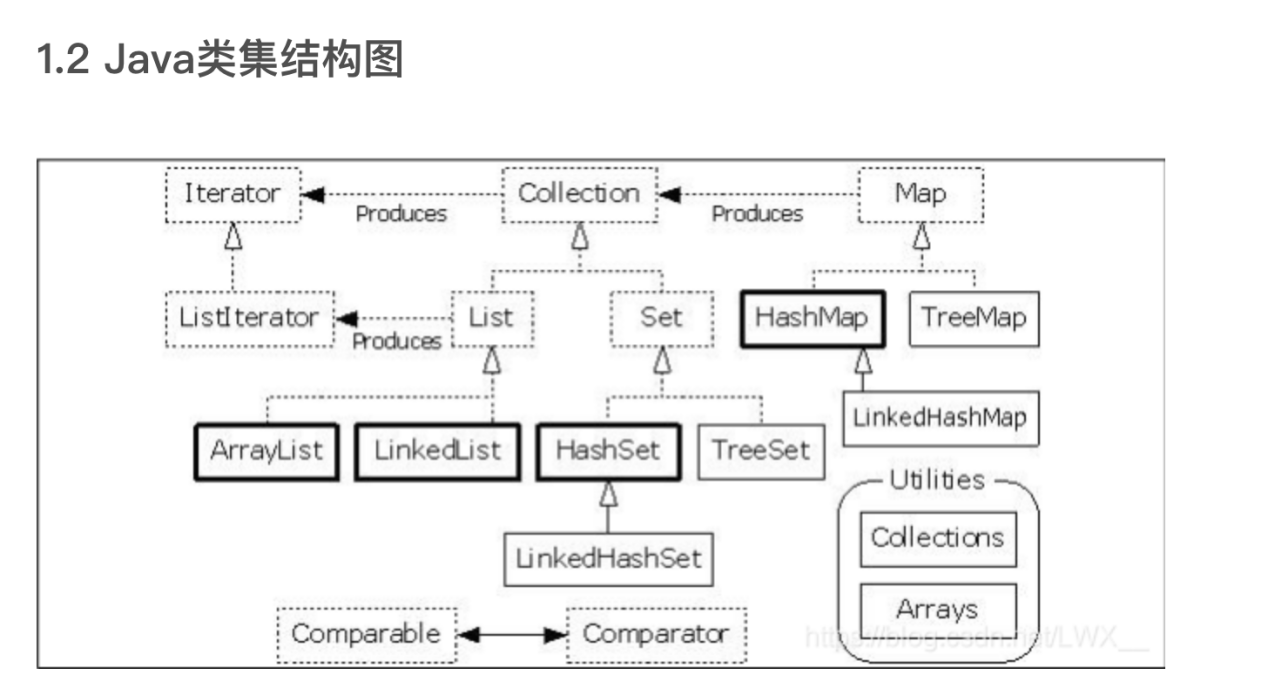
## Java集合



## HashMap

<https://blog.csdn.net/qq_38350925/article/details/105888581>

<https://blog.csdn.net/chen213wb/article/details/84647179>

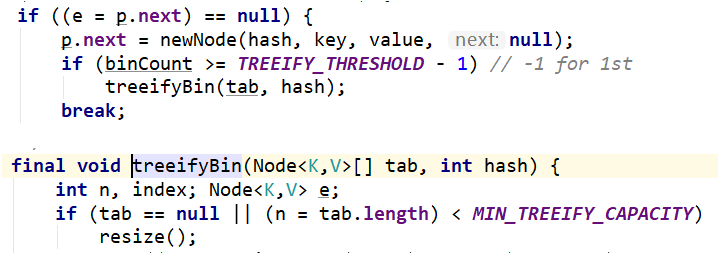
字典通过哈希表来存储数据。哈希表(哈希算法)是字典的一种表现形式。

数组初始容量16，负载因子0.75，链表超过8考虑转化为红黑树，小于6转回链表

1. 进行树化的条件，同时满足

(1)首先判断链表长度是否大于8

(2)再判断桶（数组）是否大于64，小于64的时候只会扩容，不会进化树。



1. HashMap扩容问题

（1）当负载因子大于初始0.75的时候，使得HashMap达到一定饱和度时，Key映射位置发生冲突的几率会逐渐提高。这时候，HashMap需要扩展它的长度，也就是进行Resize。1.扩容：创建一个新的Entry空数组，长度是原数组的2倍。2.ReHash：遍历原Entry数组，把所有的Entry重新Hash到新数组。

1. hashcode()与hash()

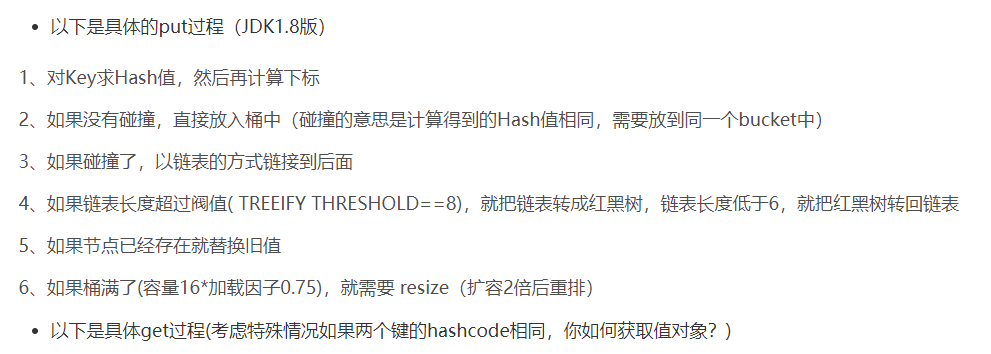


考虑到int类型值的范围[-2^31 , 2^31 -1]，虽然只要hash表映射比较松散的话，碰撞几率很小，但是映射空间太大，内存放不下，所以先做对数组的长度取模运算，得到的余数才能用来访问数组下标。

static int indexFor(int h, int length) { return h & (length-1);}

Jdk1.8中没有indexFor函数，直接使用table[index = (n – 1) & hash]（与运算交换左右，结果不变）。其中table数组为HashMap解决哈希冲突的数组+链表法中的数组。

1. Put过程



如果key经过hash后的值相同，即产生冲突后，进行key的值比较，即进行equals()方法，如果所有equals都返回false,则通过头插法插入链表头，若有一个true则表示key值相同，进行key的value的更新。

如果没有冲突，则1.8以前采用头插法插入链表头。1.8以后则采用尾插了。

****使用头插会改变链表的上的顺序，但是如果使用尾插，在扩容时会保持链表元素原本的顺序，就不会出现链表成环的问题了****

get过程

也是一样，先算出hash值，快速定位到位置，若没有值则返回null,若有值或者链表，则进行key的equals比较，

1. 为什么是16，为什么是8

16的原因是2的正整数次方，在跟16-1做与运算的时候，可以得到奇数和偶数，方便分散的存到桶里面。8或者4也可以，只是太小了碰撞概率会增大。

8的原因是：若一开始就变化为树节点，树节点是占的内存很大，不适合。当超过8个的时候，红黑树执行效率更高，以空间换时间。其实经过hash分散后，链表长度超过8的概率已经非常小了，千万分之6.

## ArrayList，vector

ArrayList（ArrayList继承了List接口）同样也是一个容器，但是其大小不固定，ArrayList的底层数据结构就是一个数组，数组元素的类型为Object类型，对ArrayList的所有操作底层都是基于数组的，初始容量10，当线性连续空间不足以存放元素时，又重新申请一片更大的空间(1.5倍)，将原有的内容移过去。

对ArrayList进行添加元素的操作的时候是分两个步骤进行的，即第一步先在object[size]的位置上存放需要添加的元素；第二步将size的值增加1。由于这个过程在多线程的环境下是不能保证具有原子性的，因此ArrayList在多线程的环境下是线程不安全的。

优点：

大小是动态扩充与收缩的。在声明ArrayList对象时不需要指定它的长度。ArrayList继承了List接口，可以很方便的进行数据的添加、插入和移除.

缺点：

当向集合插入不同类型的数据后（ArrayList将数据当作object存储），在进行数据处理时容易出现类型不匹配的错误，使用时需要进行类型转换处理，存在装箱与拆箱操作，造成性能大量损耗的现象。

Vector(线程安全)与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一个线程能够写Vector，避免多线程同时写而引起的不一致性，但实现同步需要很高的花费，因此，访问它比访问ArrayList慢。

## LinkedList

1. 一个 Node 类，这是 LinkedList 类中的一个内部类，其中每一个元素就代表一个 Node 类对象，LinkedList 集合就是由许多个 Node 对象类似于手拉着手构成。(双向链表)
2. 相对于 ArrayList 集合，LinkedList 集合多实现了一个Deque 接口，这是一个双向队列接口，双向队列就是两端都可以进行增加和删除操作。Deque<TreeNode> queue = new LinkedList<>();
3. 和 ArrayList 集合一样，LinkedList 集合也实现了Cloneable接口和Serializable接口，分别用来支持克隆以及支持序列化。List 接口也不用多说，定义了一套 List 集合类型的方法规范。

### 为什么LinkedList可以当作队列和栈来使用

因为它包含以下方法，恰好是栈和队列需要用到的方法



## ArrayList 和LinkedList的区别

1. ArrayList是基于索引的数据接口，它的底层是数组。它可以以O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。
2. 相对于ArrayList，LinkedList的插入，添加，删除操作速度更快，（双向链表）因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。
3. LinkedList比ArrayList更占内存，因为LinkedList为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

LinkedList以空间换时间

### HashSet和HaspMap，HashTable，LinkedHashMap

原因一方面是因为Hashtable是线程安全的，效率比较低。另一方面可能是因为Hashtable没有遵循驼峰命名法吧。



HashSet实现了Set接口，它不允许集合中有重复的值

### set和map的区别：

线程安全的set：



安全的set底层采用的是ReentrantLock lock = this.lock;来实现线程安全。

Set: 包括hashset treeset linkedhashset

set就是只存储key，

Map包括hashmap treemap linkedhashmap  
map是存储的（key,value）

两者依次对比，例如其中

Treemap：

是默认按照key升序排列，可以按照自定义排序compareTo() ,底层是红黑树排序形式。

Linkedhashmap:

LinkedHashMap通过维护一个运行于所有条目的双向链表,保证了集合元素迭代的顺序,这个顺序可以是插入顺序或者访问顺序.访问顺序就是访问过后会将访问节点直接挪到链表尾。

Treeset:

与 HashSet 完全类似的是，TreeSet 里绝大部分方法都是直接调用 TreeMap 的方法来实现的

treeset和linkedhashset的区别跟treemap和linkedhashmap类似。

### Queue

