一、Map的实现类结构

|----Map:双列数据，存储key-value对的数据 ---类似于高中的函数：y = f(x)

|----HashMap:作为Map的主要实现类；线程不安全的，效率高；存储null的key和value

|----LinkedHashMap:保证在遍历map元素时，可以按照添加的顺序实现遍历。

原因：在原有的HashMap底层结构基础上，添加了一对指针，指向前一个和后一个元素。

对于频繁的遍历操作，此类执行效率高于HashMap。

|----TreeMap:保证按照添加的key-value对进行排序，实现排序遍历。此时考虑key的自然排序或定制排序

底层使用红黑树

|----Hashtable:作为古老的实现类；线程安全的，效率低；不能存储null的key和value

|----Properties:常用来处理配置文件。key和value都是String类型

HashMap的底层：数组+链表 （jdk7及之前）

数组+链表+红黑树 （jdk 8）

二、面试题：

1. HashMap的底层实现原理?
2. HashMap 和 Hashtable的异同?
3. CurrentHashMap 与 Hashtable的异同？（暂时不讲）

三、Map结构的理解

1、Map与Collection并列存在。用于保存具有映射关系的数据:key-value

2、Map 中的 key 和 value 都可以是任何引用类型的数据

3、Map中的key:无序的、不可重复的，使用Set存储所有的key ---> key所在的类要重写equals()和hashCode() （以HashMap为例）

4、Map中的value:无序的、可重复的，使用Collection存储所有的value --->value所在的类要重写equals()

5、一个键值对：key-value构成了一个Entry对象。

6、Map中的entry:无序的、不可重复的，使用Set存储所有的entry

四、Map的实现类

1、HashMap类

1、注意点

1、HashMap是 Map 接口使用频率最高的实现类

2、允许使用null键和null值，与HashSet一样，不保证映射的顺序。

3、所有的key构成的集合是Set:无序的、不可重复的。所以，key所在的类要重写：equals()和hashCode()

4、所有的value构成的集合是Collection:无序的、可以重复的。所以，value所在的类要重写：equals()

5、HashMap 判断两个 **key** 相等的标准是：两个 key 通过 equals() 方法返回 true， hashCode 值也相等。

6、HashMap 判断两个 **value**相等的标准是：两个 value 通过 equals() 方法返回 true。

2、HashMap的底层实现原理？以jdk7为例说明：

HashMap map = new HashMap():

在实例化以后，底层创建了长度是16的一维数组Entry[] table。

...可能已经执行过多次put...

map.put(key1,value1):

首先，调用key1所在类的hashCode()计算key1哈希值，此哈希值经过某种算法计算以后，得到在Entry数组中的存放位置。

如果此位置上的数据为空，此时的key1-value1添加成功。 ----情况1

如果此位置上的数据不为空，(意味着此位置上存在一个或多个数据(以链表形式存在)),比较key1和已经存在的一个或多个数据的哈希值：

如果key1的哈希值与已经存在的数据的哈希值都不相同，此时key1-value1添加成功。----情况2

如果key1的哈希值和已经存在的某一个数据(key2-value2)的哈希值相同，继续比较：调用key1所在类的equals(key2)方法，比较：

如果equals()返回false:此时key1-value1添加成功。----情况3

如果equals()返回true:使用value1替换value2。

补充：关于情况2和情况3：此时key1-value1和原来的数据以链表的方式存储。

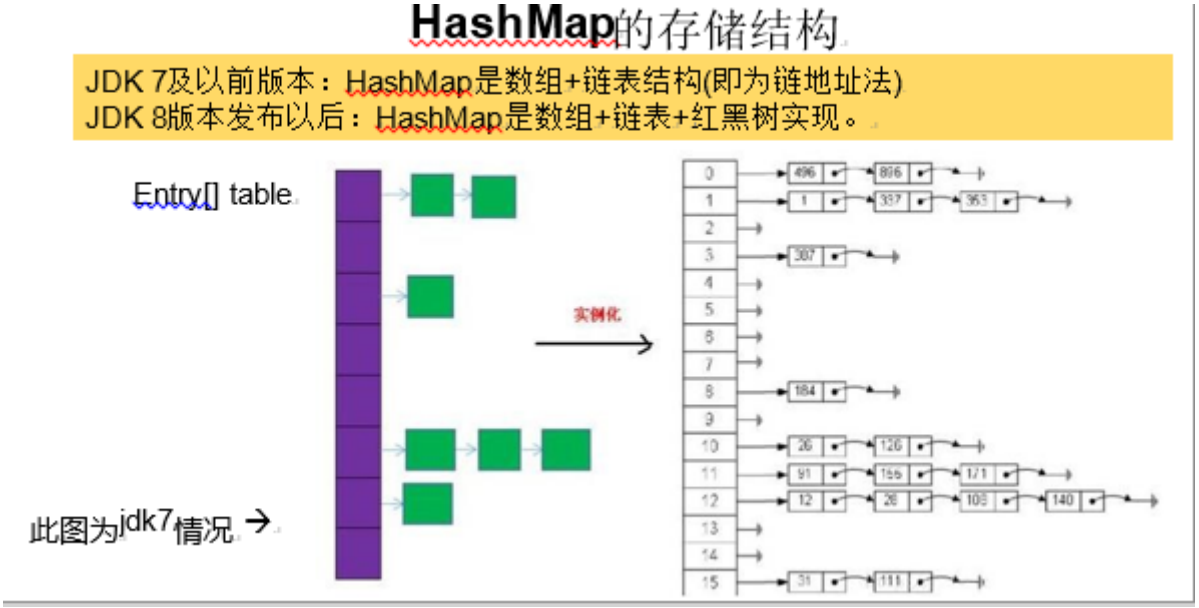
在不断的添加过程中，会涉及到扩容问题，当超出临界值(且要存放的位置非空)时，扩容。默认的扩容方式：扩容为原来容量的2倍，并将原有的数据复制过来。(然后重新计算)

jdk8 相较于jdk7在底层实现方面的不同：

1. new HashMap():底层没有创建一个长度为16的数组
2. jdk 8底层的数组是：Node[],而非Entry[]
3. 首次调用put()方法时，底层创建长度为16的数组
4. jdk7底层结构只有：数组+链表。jdk8中底层结构：数组+链表+红黑树。

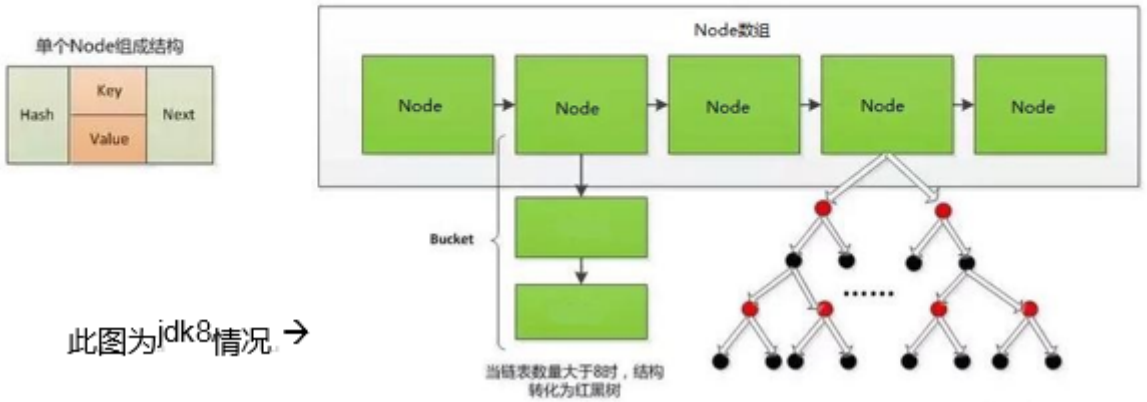
4.1 形成链表时，七上八下（jdk7:新的元素指向旧的元素。jdk8：旧的元素指向新的元素）

4.2 当数组的某一个索引位置上的元素以链表形式存在的数据个数 > 8 且当前数组的长度 > 64时，此时此索引位置上的所数据改为使用红黑树存储。



HashMap的存储结构

JDK 7及以前版本：HashMap是数组+链表结构(即为链地址法)
JDK 8版本发布以后：HashMap是数组+链表+红黑树实现。



3、面试题

1、谈谈你对HashMap中put/get方法的认识？如果了解再谈谈HashMap的扩容机制？默认大小是多少？什么是负载因子（或填充比）？什么是吞吐临界值（或阈值、threshold）？

2、负载因子的大小，对HashMap有什么影响？

1.负载因子的大小决定了HashMap的数据密度。

2.负载因子越大密度越大，发生碰撞的几率越高，数组中的链表越容易长，造成查询或插入时的比较次数增多，性能会下降。

3.负载因子越小，就越容易触发扩容，数据密度也越小，意味着发生碰撞的几率越小，数组中的链表也就越短，查询和插入时比较的次数也越小，性能会更高。但是会浪费一定的内容空间。而且经常扩容也会影响性能，建议初始化预设大一点的空间。

4.按照其他语言的参考及研究经验，会考虑将负载因子设置为0.7~0.75，此时平均检索长度接近于常数。

2、LinkedHashMap

1、注意点：

1.LinkedHashMap是HashMap的子类

2.在HashMap的基础上，使用了一对双向链表来记录添加元素的顺序

3.与LinkedHashSet类似，LinkedHashMap可以维护Map的迭代顺序，迭代顺序与Key-Value对的插入顺序一致

2、LinkedHashMap的底层实现原理（了解）

```
源码中：
static class Entry<K,V> extends HashMap.Node<K,V> {
Entry<K,V> before, after;//能够记录添加的元素的先后顺序
Entry(int hash, K key, V value, Node<K,V> next) {
super(hash, key, value, next);
}
}
```

五、Map中定义的常见方法

1. 添加、删除、修改操作

a. Object put(Object key,Object value)：将指定key-value添加到(或修改)当前map对象中

b. void putAll(Map m):将m中的所有key-value对存放到当前map中

c. Object remove(Object key)：移除指定key的key-value对，并返回value

d. void clear()：清空当前map中的所有数据

```
@Test
public void test3(){
Map map = new HashMap();
//添加
map.put("AA",123);
map.put(45,123);
map.put("BB",56);
//修改
map.put("AA",87);

 System.out.println(map);

 Map map1 = new HashMap();
 map1.put("CC",123);
 map1.put("DD",123);

```

```
17  map.putAll(map1);
18
19  System.out.println(map);
20
21  //remove(Object key)
22  Object value = map.remove("CC");
23  System.out.println(value);
24  System.out.println(map);
25
26  //clear()
27  map.clear();//与map = null操作不同
28  System.out.println(map.size());
29  System.out.println(map);
30  }
```

2、元素查询的操作

a. Object get(Object key)：获取指定key对应的value

b. boolean containsKey(Object key)：是否包含指定的key

c. boolean containsValue(Object value)：是否包含指定的value

d. int size()：返回map中key-value对的个数

e. boolean isEmpty()：判断当前map是否为空

f. boolean equals(Object obj)：判断当前map和参数对象obj是否相等

```
1  @Test
2  public void test4(){
3  Map map = new HashMap();
4  map.put("AA",123);
5  map.put(45,123);
6  map.put("BB",56);
7  // Object get(Object key)
8  System.out.println(map.get(45));
```

```
//containsKey(Object key)
 boolean isExist = map.containsKey("BB");
 System.out.println(isExist);

 isExist = map.containsValue(123);
 System.out.println(isExist);

 map.clear();

 System.out.println(map.isEmpty());

 }
```

3、元视图操作的方法

a. Set keySet()：返回所有key构成的Set集合

b. Collection values()：返回所有value构成的Collection集合

c. Set entrySet()：返回所有key-value对构成的Set集合

```
@Test
public void test5(){
Map map = new HashMap();
map.put("AA",123);
map.put(45,1234);
map.put("BB",56);

//遍历所有的key集：keySet()
Set set = map.keySet();
 Iterator iterator = set.iterator();
 while(iterator.hasNext()){
 System.out.println(iterator.next());
 }
 System.out.println();

 //遍历所有的value集：values()
 Collection values = map.values();
 for(Object obj : values){
```

```
19  System.out.println(obj);
20  }
21  System.out.println();
22  //遍历所有的key-value
23  //方式一：entrySet()
24  Set entrySet = map.entrySet();
25  Iterator iterator1 = entrySet.iterator();
26  while (iterator1.hasNext()){
27  Object obj = iterator1.next();
28  //entrySet集合中的元素都是entry
29  Map.Entry entry = (Map.Entry) obj;
30  System.out.println(entry.getKey() + "---->" +
entry.getValue());
31
32  }
33  System.out.println();
34  //方式二：
35  Set keySet = map.keySet();
36  Iterator iterator2 = keySet.iterator();
37  while(iterator2.hasNext()){
38  Object key = iterator2.next();
39  Object value = map.get(key);
40  System.out.println(key + "=====" + value);
41  }
42  }
```

4、总结（首先会）

a. 添加：put(Object key,Object value)

b. 删除：remove(Object key)

c. 修改：put(Object key,Object value)

d. 查询：get(Object key)

e. 长度：size()

f. 遍历：keySet() / values() / entrySet()