雙列集合框架：Map

1. 常用實現類結構

*|----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）*

[面試題]

*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）*

1. 存儲結構的理解：

*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）  
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()*

*一个键值对：key-value构成了一个Entry对象。  
Map中的entry:无序的、不可重复的，使用Set存储所有的entry*

圖示：



1. 常用方法

*添加：put(Object key,Object value)  
删除：remove(Object key)  
修改：put(Object key,Object value):key相同會覆蓋掉上一個value值  
查询：get(Object key)  
长度：size()  
遍历：keySet() / values() / entrySet()*

1. 內存結構說明：（難點）
   1. HashMap在jdk7中實現原理：

*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。*

* 1. HashMap在jdk8中相較於jdk7在底層實現方面的不同：

*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。*

* 1. HashMap底層典型屬性的說明：

*DEFAULT\_INITIAL\_CAPACITY : HashMap的默认容量，16*

*DEFAULT\_LOAD\_FACTOR：HashMap的默认加载因子：0.75:值大小決定了HashMap的數據密度  
threshold：扩容的临界值，=容量\*填充因子：16 \* 0.75 => 12  
TREEIFY\_THRESHOLD：Bucket中链表长度大于该默认值，转化为红黑树:8  
MIN\_TREEIFY\_CAPACITY：桶中的Node被树化时最小的hash表容量:64*

* 1. LinkedHashMap的底層實現原理(了解)

LinkedHashMap底層使用的結構與HashMap相同，因為LinkedHashMap繼承于HashMap。

區別就在於：LinkedHashMap內部提供了Entry，替換HashMap中的Node。



1. TreeMap的使用

*//向TreeMap中添加key-value，要求key必须是由同一个类创建的对象  
//因为要按照key进行排序：自然排序 、定制排序*

1. 使用Properties讀取配置文件

*//Properties:常用来处理配置文件。key和value都是String类型*public static void main(String[] args) {  
 FileInputStream fis = null;  
 try {  
 Properties pros = new Properties();  
  
 fis = new FileInputStream("jdbc.properties");  
 pros.load(fis);*//加载流对应的文件* String name = pros.getProperty("name");  
 String password = pros.getProperty("password");  
  
 System.*out*.println("name = " + name + ", password = " + password);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 if(fis != null){  
 try {  
 fis.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
  
}