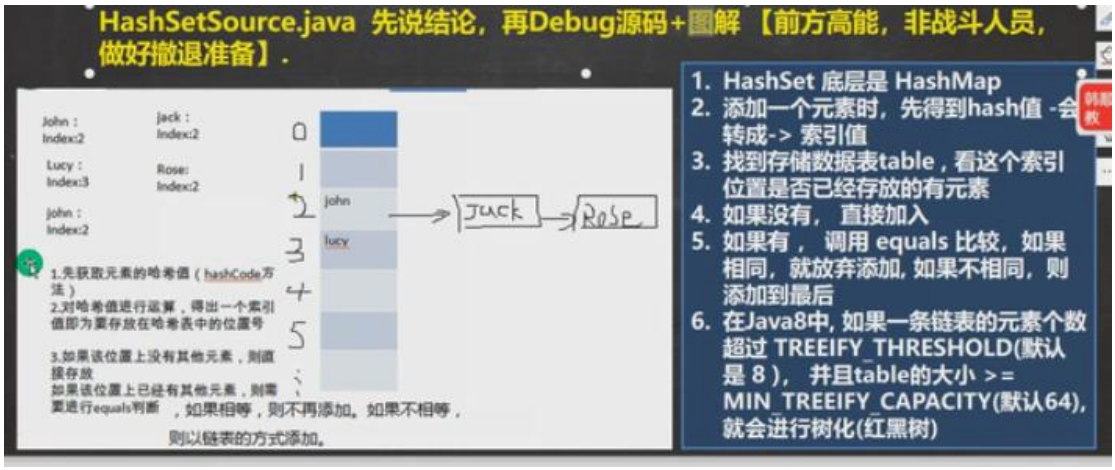
//Set 接口实现类的对象，不能存放重复的元素
//可以添加一个 null
//Set 接口对象存放数据是无序的（即添加的顺序和取出的顺序不一致）
//但取出的顺序是固定的（虽然不是添加的顺序）
//set 接口对象不能通过索引来获取

HashSet 的说明：
1）HashSet 实现了 Set 接口
2）HashSet 实际上是 HashMap

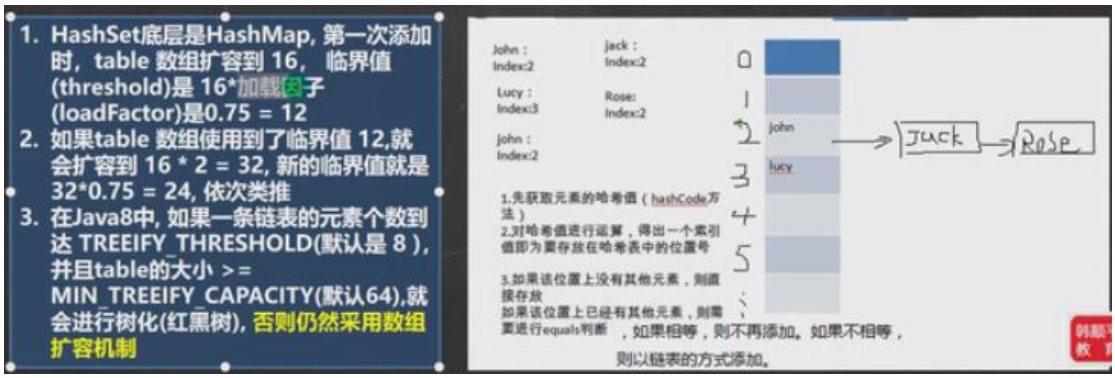
```
public HashSet(){
        map = new HashMap<>();
}
```

3）可以存放 null 值，但是只能有一个 null
1）HashSet 不保证元素是有序的，取决于 hash 后，再确定索引的结果
2）不能有重复元素/对象

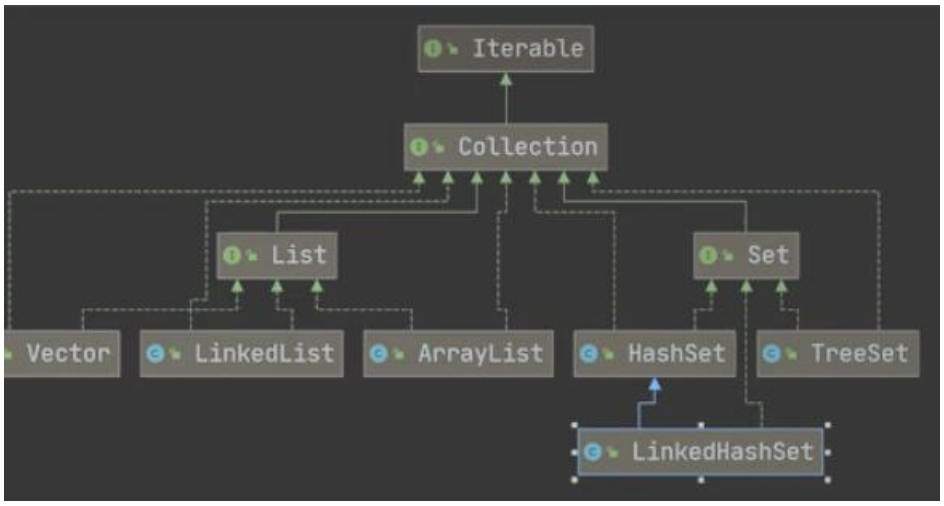
HashSet 的底层机制：HashSet 的底层是 HashMap，HashMap 的底层是数组+链表+红黑树



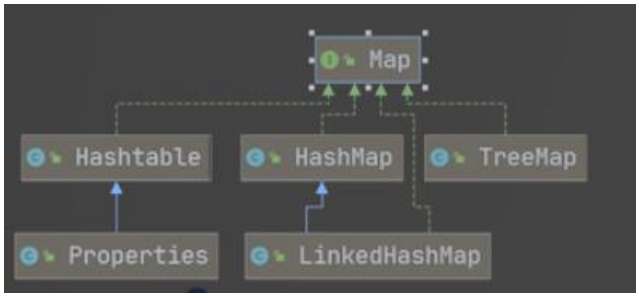
HashSet 的扩容和转成红黑树机制：



LinkedHashSet 的解释：

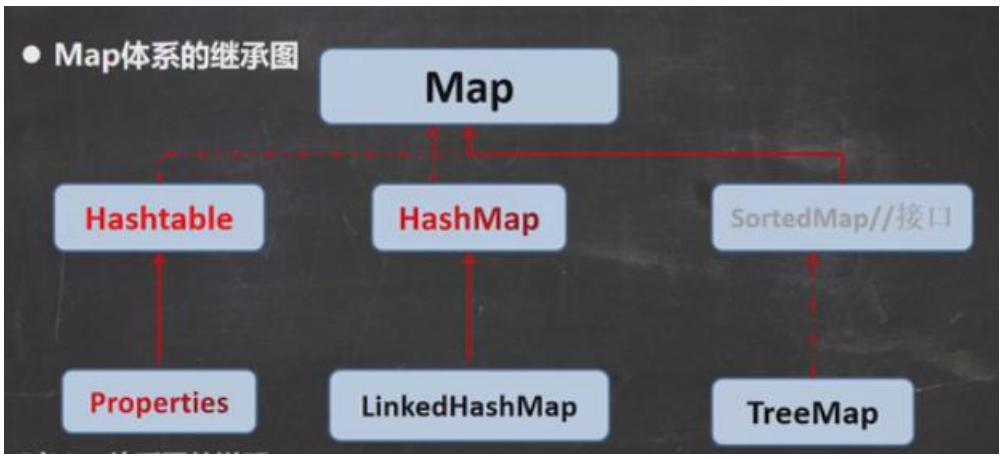


1）LinkedHashSet 是 HashSet 的子类
2）LinkedHashSet 底层是一个 LinkedHashMap，底层维护了一个数组+双向链表
3）LinkedHashSet 根据元素的 hashCode 值来决定元素的存储位置，同时使用链表维护元素的次序，使得元素看起来是以插入顺序保存的
4）LinkedHashSet 不允许添重复元素

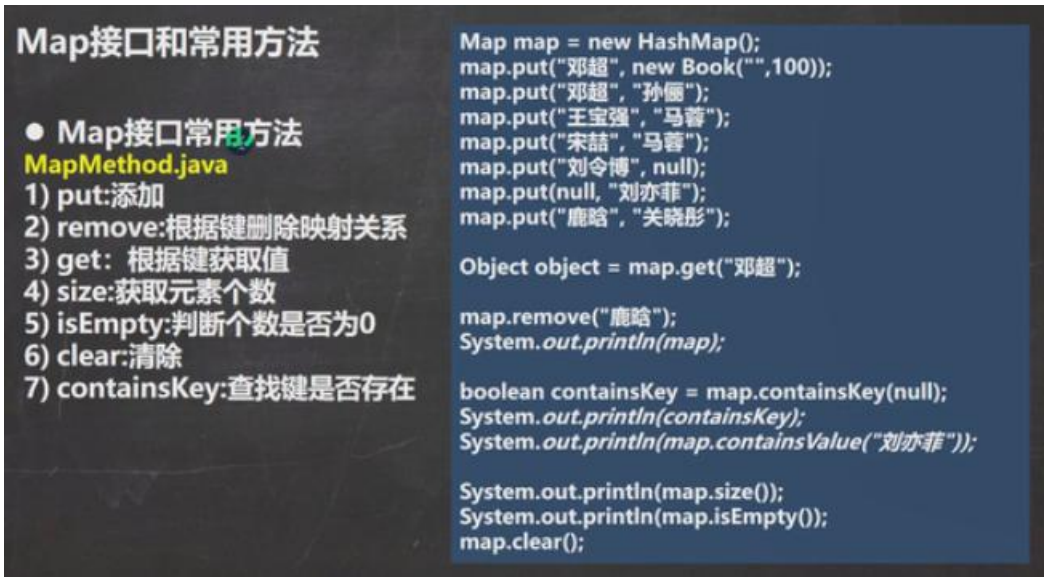


Map 接口（JDK8）实现类的特点：
1）Map 与 Collection 并列存在，用于保存具有映射关系的数据：Key-Value
2）Map 中的 key 和 value 可以是任何引用类型的数据，会封装到 HashMap$Node 对象中
3）Map 中的 key 不允许重复，原因和 HashSet 一样
4）Map 中的 value 可以重复
5）Map 的 key 可以为 null，value 也可以为 null，注意 key 为 null 只能有一个，value 为 null 可以多个
6）常用 String 类作为 Map 的 key
7）key 和 value 之间存在单向一对一关系，即通过指定的 key 总能找到对应的 value
8）Map 存放的数据的 key-value 是一堆 k-v 放在一个 Node 中的，有因为 Node 实现了 Entry 接口

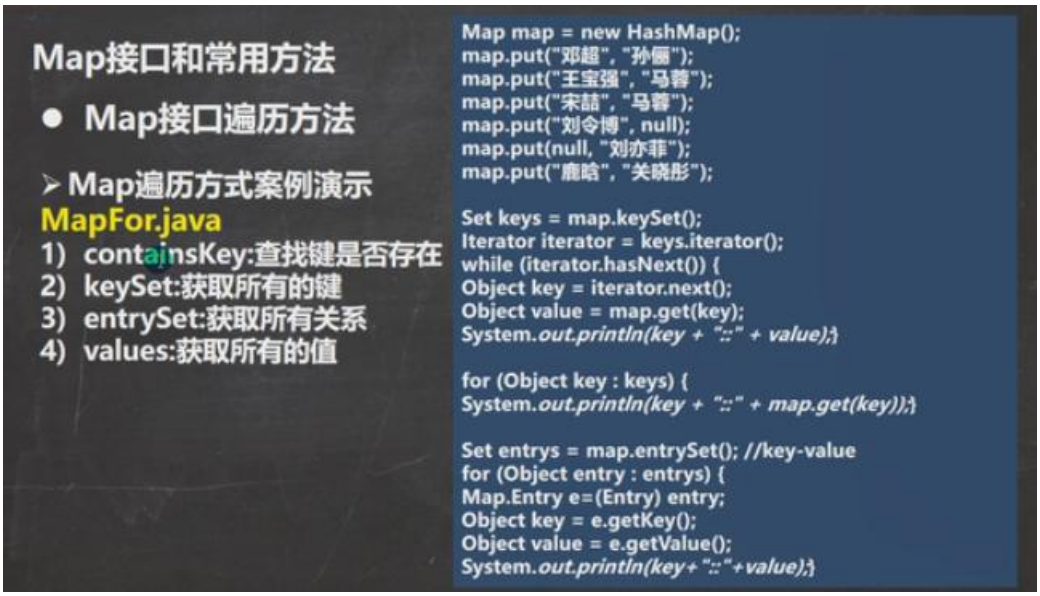
Map 体系的继承：



Map 接口的常用方法：



Map 接口的遍历方法：



```java
//第一组: 先取出 所有的Key , 通过Key 取出对应的Value
Set keyset = map.keySet();
//(1) 增强for
System.out.println("-----第一种方式-------");
for (Object key : keyset) {
    System.out.println(key + "-" + map.get(key));
}
//(2) 迭代器
System.out.println("----第二种方式--------");
Iterator iterator = keyset.iterator();
while (iterator.hasNext()) {
    Object key =  iterator.next();
    System.out.println(key + "-" + map.get(key));
}
```

d.java | MapFor.java | Map_java | MapSource_java | HashSetSource.ja

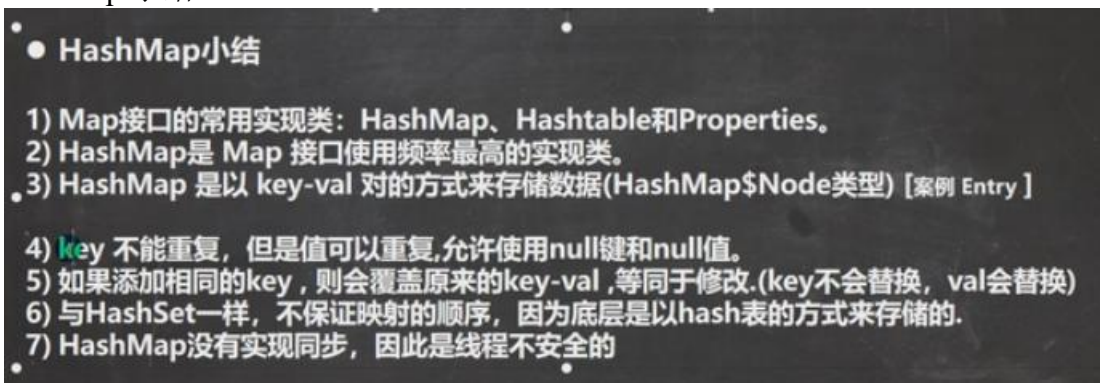
```java
//第二组: 把所有的values取出
Collection values = map.values();
//这里可以使用所有的Collections使用的遍历方法
//(1) 增强for
System.out.println("---取出所有的value 增强for----");
for (Object value : values) {
    System.out.println(value);
}
//(2) 迭代器
System.out.println("---取出所有的value 迭代器----");
Iterator iterator2 = values.iterator();
while (iterator2.hasNext()) {
    Object value =  iterator2.next();
    System.out.println(value);

}
```

```java
//第三组: 通过EntrySet 来获取 k-v
Set entrySet = map.entrySet();// EntrySet<Map.Entry<K,V>>
//(1) 增强for
System.out.println("----使用EntrySet 的 for增强(第3种)----");
for (Object entry : entrySet) {
    //将entry 转成 Map.Entry
    Map.Entry m = (Map.Entry) entry;
    System.out.println(m.getKey() + "-" + m.getValue());
}
```

```java
//(2) 迭代器
System.out.println("----使用EntrySet 的 迭代器(第4种)----");
Iterator iterator3 = entrySet.iterator();
while (iterator3.hasNext()) {
    Object entry =  iterator3.next();
    //System.out.println(next.getClass());//HashMap$Node -实现-> Map.Entry (getKey,getVal
    //向下转型 Map.Entry
    Map.Entry m = (Map.Entry) entry;
    System.out.println(m.getKey() + "-" + m.getValue());
}
```

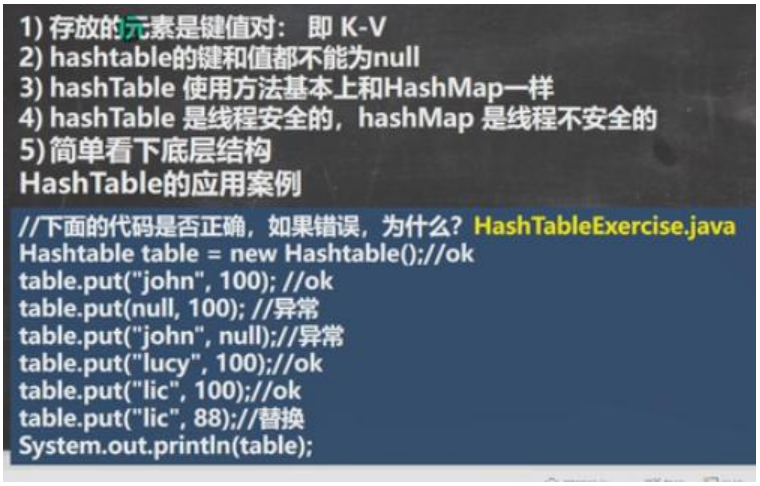
HashMap 小结：



HashMap 底层机制：

关于树化（转成红黑树）
模拟 HashMap 出发扩容，树化情况

Map 接口的实现类——HashTable



HashTable 的扩容机制：

```
//简单说明一下Hashtable的底层
//1. 底层有数组 Hashtable$Entry[] 初始化大小为 11
//2. 临界值 threshold 8 = 11 * 0.75
//3. 扩容：按照自己的扩容机制来进行即可.
//4. 执行 方法 addEntry(hash, key, value, index); 添加K-V 封装到Entry
//5. 当 if (count >= threshold) 满足时，就进行扩容
//5. 按照 int newCapacity = (oldCapacity << 1) + 1; 的大小扩容.
```

Hashtable 和 HashMap 的对比：

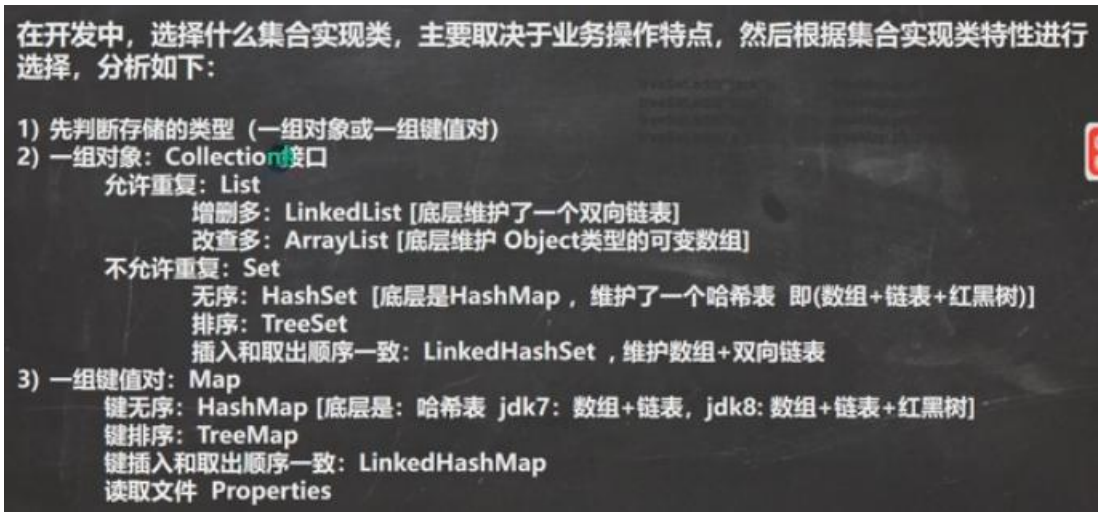
	版本	线程安全（同步）	效率	允许null键null值
HashMap	1.2	不安全	高	可以
Hashtable	1.0	安全	较低	不可以

Map 接口实现类---Properties

1. Properties 类继承自 Hashtable 类并且实现了 Map 接口，也使用一种键值对的形式来保存数据；
2. 其使用特点和 Hashtable 类似
3. Properties 还可以用于从 xxx.properties 文件中，加载数据到 Properties 类对象
4. 说明：工作后 xxx.properties 文件通常作为配置文件

开发中如何选择集合实现类：

1. 先判断存储的类型（一组对象[单列]或一组键值对[双列]）
2. 一组对象的情况下：Collection 接口
   - 允许重复：List
     - 增删情况多：LinkedList（底层维护了一个双向链表）
     - 改查情况多：ArrayList（底层维护 Object 类型的可变数组）
   - 不允许重复：Set
     - 无序：HashSet（底层是 HashMap，维护了一个哈希表 即（数组+链表+红黑树））
     - 排序：TreeSet
     - 插入和取出顺序一致：LinkedHashSet，维护数组+双向链表
3. 一组键值对的情况下：Map
   - 键无序：HashMap（底层是：哈希表 JDK7：数组+链表，JDK8：数组+链表+红黑树）
   - 键排序：TreeMap
   - 键插入和取出顺序一致：LinkedHashMap
   - 读取文件 Properties



TreeSet 类：

```
TreeSet treeSet = new TreeSet(new Comparator() {
    @Override
    public int compare(Object o1, Object o2) {
        //return ((String) o2).length() - ((String) o1).length();
        return ((String) o2).compareTo((String) o1);
    }
});
//使用传统无参构造器，创建TreeSet时，仍然是无序的
//添加的元素是按照字符串的大小来排序的
//使用TreeSet提供的一个构造器可以传入一个比较器(匿名内部类)，并制定排序规则

//添加数据
treeSet.add("jack");
treeSet.add("tom");
treeSet.add("sp");
treeSet.add("a");
//在按照长度比较时，加不进去（规则规定了长度相等则加不进去），
//前面有a ，所以后面的b 也是长度为1的就加不进去了
treeSet.add("b");
```

```
    public int compare(Object o1, Object o2) {
        //按照传入的 k(String) 的大小进行排序
        //按照K(String) 的长度大小排序
        //return ((String) o2).compareTo((String) o1);
        return ((String) o2).length() - ((String) o1).length();
    }
});
treeMap.put("jack", "杰克");
treeMap.put("tom", "汤姆");
treeMap.put("kristina", "克瑞斯提诺");
treeMap.put("smith", "斯密斯");
treeMap.put("hsp", "韩顺平");//加入不了

System.out.println("treemap=" + treeMap);
```

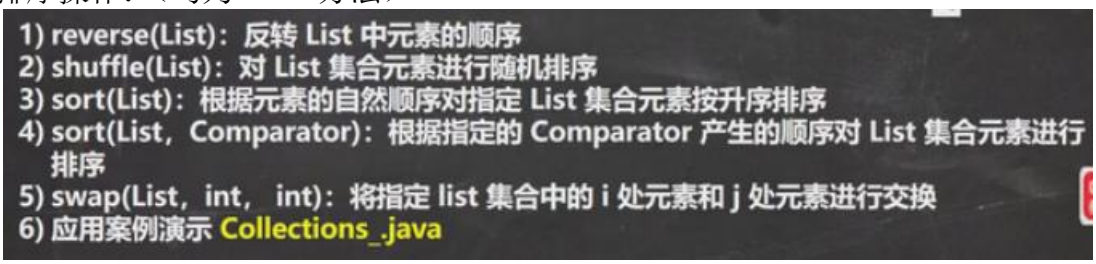
根据比较的要求和规范决定是否能加入一些对象，比如：比较的规则是按照字符串的长度，所以，如果有两个或多个字符串长度的相等的，那底层源码不会让后面的那个字符串加入。

Collections 工具类

1）Collections 是一个操作 Set、List 和 Map 等集合的工具类

2）Collections 中提供了一系列静态的方法对集合元素进行排序、查询和修改等操作

排序操作：（均为 static 方法）



试分析HashSet和TreeSet分别如何实现去重的
(1) HashSet的去重机制：hashCode() + equals() ,底层先通过存入对象,进行运算得到一个hash值，通过hash值得到对应的索引，如果发现table索引所在的位置，没有数据，就直接存放，如果有数据，就进行equals比较[遍历比较]，如果比较后，不相同,就加入，否则就不加入.
(2) TreeSet的去重机制: 如果你传入了一个Comparator匿名对象，就使用实现的compare去重，如果方法返回0,就认为是相同的元素/数据，就不添加，如果你没有传入一个Comparator匿名对象，则以你添加的对象实现的Compareable接口的compareTo去重.

	底层结构	版本	线程安全（同步） 效率	扩容倍数
ArrayList	可变数组	jdk1.2	不安全，效率高	如果使用有参构造器1.5倍，如果是无参构造器 1. 第一次扩容10 2. 从第二次开始按照1.5被
Vector	可变数组 Object[]	jdk1.0	安全，效率不高	如果是无参,默认10,满后，按照2倍扩容 如果是指定大小创建Vector,则每次按照2倍扩容.

集合工具类的使用特性和使用场景：
Collection（集合）：
1）可以动态保存多个对象，使用比较方便；
2）提供了一系列的操作对象的方法：add、remove、set、get 等。

collection 接口实现类的特点：
1）collection 实现子类可以存放多个元素，每个元素可以是 Object；
2）有些 collection 的实现类可以存放重复元素，有些不可以；
3）有些 collection 的实现类是有序的（list），有些是无序的（Set）；
4）collection 接口没有直接的实现子类，是通过它的子接口 Set 和 List 来实现的。

collection 接口遍历元素的方式：
1. 使用 Iterator（迭代器）
1）Iterator 对象称为迭代器，主要用于遍历 collection 集合中的元素；
2）所有实现了 collection 接口的集合类都有一个 iterator()方法，用以返回一个实现了 iterator 接口的对象，即可以返回一个迭代器；
3）Iterator 仅用于遍历集合，其本身不存放对象。

Iterator 的执行原理：
```
Iterator iterator = coll.iterator();//得到一个集合的迭代器
//hasNext;判断是否还有下一个元素
While(iterator.hasNext()){
//next()作用：1.下移 2.将下移后的集合位置上的元素返回
System.out.println(iterator.next());
}
```
2. 增强 for 循环

```
for(元素类型 元素名：集合名或数组名){
        访问元素
}
```

List 接口（是 Collection 接口的子接口）
1）List 集合类中元素有序（即添加顺序和取出顺序一致）且可重复；
2）List 集合中的每个元素都有其对应的顺序索引，即支持索引；
3）List 容器宏的元素都对应一个整数型的序号记载其在容器中的位置，可以根据序号存取容器中的元素；
4）常用的 List 接口实现类有：ArrayList、LinkedList 和 Vector

List 接口的常用方法：
1）add(int index, Object ele);//在 index 位置插入 ele 元素；
2）Boolean addAll(int index, Collection eles);//从 index 位置开始将 eles 中的所有元素添加进来；
3）Object get(int index);//获取指定 Index 位置的元素；
4）Int indexOf(Object obj);//返回 obj 在集合中首次出现的位置；
5）Int lastIndexOf(Object obj);//返回 obj 在当前集合中末次出现的位置
6）Object remove(int index);//溢出指定 index 位置的元素，并返回此元素
7）Object set(int index, Object ele);//设置指定 index 位置的元素为 ele，相当于替换；
8）List subList(int fromIndex, int toIndex);//返回从 fromIndex 到 toIndex 位置的子集合

List 的三种遍历方式
1）使用 Iterator 迭代器循环遍历；
2）使用增强 for 循环遍历；
3）使用普通 for 循环遍历。

ArrayList 结构：
1）permits all elements，including null，ArrayList 可以加入 null 并且多个加入；
2）ArrayList 是由数组来实现数据存储的；
3）ArrayList 基本等同于 Vector，除了 ArrayList 是线程不安全（执行效率高）；在多线程情况下，ArrayList 可能不太适合；
4）ArrayList 中维护了一个 Object 类型的数组 elementData；
5）当创建 ArrayList 对象时，如果使用的是无参构造器，则初始 elementData 容量为 0，第 1 次添加则扩容 elementData 为 10，如需要再次扩建，则扩容 elementData 为 1.5 倍；
6）如果使用的是指定大小的构造器，则初始 elementData 容量为指定大小，如果需要扩容，则直接扩容 elementData 为 1.5 倍。

Vector 结构：
1）Vector 底层也是一个对象数组，protected Object[] elementData；

2）Vector 是线程同步的，即线程安全，Vector 类的操作方法带有 synchronized；
3）在开发中，需要线程同步安全时，考虑使用 Vector。

LinkedList 结构：
1）LinkedList 底层实现了双向链表和双端队列的特点；
2）可以添加任意元素（元素可以重复），包括 null；
3）线程不安全，没有实现同步。
4）LinkedList 底层维护了一个双向链表；
5）LinkedList 中维护了两个属性 first 和 last 分别指向首节点和尾节点；
6）每个结点（Node 对象），里面又维护了 prev、next、item 三个属性，其中通过 prev 指向前一个，通过 next 指向后一个节点，最终实现双向链表；
7）LinkedList 的元素的添加和删除不是通过数组完成的，相对来说效率较高；

LinkedList 和 ArrayList 的比较：
1）如果改查比较多，选择 ArrayList；
2）如果增删比较多，选择 LinkedList；
3）一般来说，在程序中，绝大部分都是查询，因此大部分情况下会选择 ArrayList。

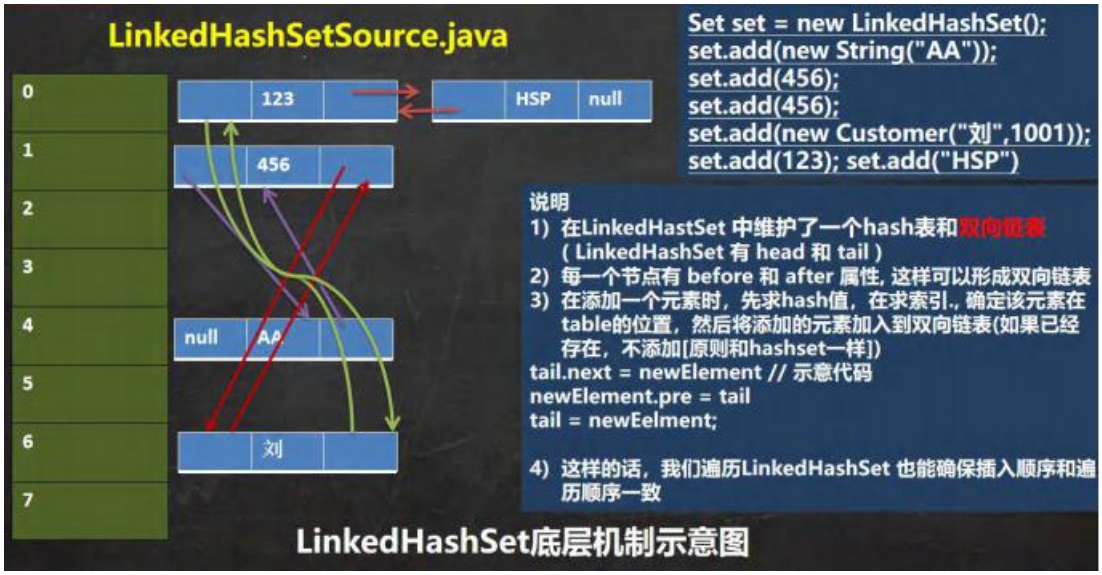
Set 接口
1）无序（添加和取出的顺序不一致），没有索引；
2）不允许重复元素，所以最多包含一个 null；
3）和 List 接口一样，Set 接口也是 Collection 的子接口，因此常用方法和 Collection 接口一样；
4）以 Set 接口的实现类 HashSet；
5）Set 接口的实现类的对象(Set 接口对象)，不能存放重复的元素，可以添加一个 null；
6）Set 接口对象存放数据是无序（即添加的顺序和取出的顺序不一致）。

HashSet 实现类
1）HashSet 实现了 Set 接口；
2）HashSet 实际上是 HashMap；
3）利用存放 null 值，但是只能有一个 null；
4）HashSet 不保证元素是有序的，取决于 hash 后，再确定索引的结果；
5）不能有重复元素/对象。

HashSet 添加元素原理：
1）HashSet 底层是 HashMap；
2）添加一个元素时，先得到 hash 值-转成-索引值；
3）找到存储数据表 table，看这个索引位置是否已经存放元素；
4）如果没有则直接加入；
5）如果有，调用 equals 比较，如果相同就放弃添加，如果不相同则添加到最后；

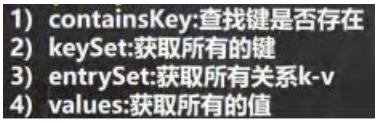
LinkedHashSet 接口
1）LinkedHashSet 是 HashSet 的子类；

2）LinkedHashSet 底层是一个 LinkedHashMap，其底层维护了一个数组+双向链表；
3）LinkedHashSet 根据元素的 hashCode 值来决定元素的存储位置，同时使用链表来维护元素的次序，使得元素看起来是以插入顺序保存的；
4）LinkedHashSet 不允许添加重复元素。



Map 接口实现类：
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 可以是任何引用类型的数据，会封装到 HashMap$Node 对象中；
3）Map 中的 key 不允许重复；
4）Map 中的 value 可以重复；
5）Map 中的 Key 可以为 null、value 也可以为 null，注意 key 为 null 时只能有一个，value 为 null 可以有多个；
6）常用 String 类作为 Map 的 key；
7）key 和 value 之间存在单向一对一关系，即通过指定的 key 总能找到对应的 value；
8）Map 存放数据的 key-value 中，一对 key-value 是放在一个 HashMap$Node 中的。

Map 接口的常用方法：
1）put()：根据 k-v 添加元素；
2）remove()：根据键删除映射关系；
3）get()：根据键获取值；
4）size()：获取元素个数；
5）isEmpty()：判断个数是否为 0；
6）clear()：清除 k-v；
7）containsKey()：查找键是否存在。



HashMap 扩容机制：
1）HashMap 底层维护了 Node 类型的数组 table，默认为 null；
2）当创建对象时，将加载因子（loadfactor）初始化为 0.75；
3）当添加 key-value 时，通过 key 的哈希值得到在 table 的索引。然后判断该索引处是否有元素，如果没有元素直接添加或该索引处有元素，则继续判断该元素的 key 和准备加入的 key 是否相等，如果相等则直接替换 value；如果不相等则需要判断是树结构还是链表结构，做出相应的处理；如果添加时发现容量不够则需要扩容；
4）第 1 次添加则需要扩容 table 容量为 16，临界值（threshold）为 12（16*0.75）；
5）以后再扩容则需要扩容 table 容量为原来是 2 倍（32），临界值为原来的 2 倍即 24，以此类推。

HashTable 结构：
1）存放的元素是键值对：即 k-v；
2）HashTable 的键和值都不能为 null，否则会抛出 NullPointerException；
3）HashTable 使用方法基本上和 HashMap 一样；
4）HashTable 是线程安全的（synchronized），HashMap 是线程不安全的。

Properties 实现类：
1）Properties 类继承自 Hashtable 类并且实现了 Map 接口，也是使用一种键值对的形式来保存数据；
2）其使用特点和 HashTable 类似；
3）Properties 还可以用于从 xxx.properties 文件中加载数据到 Properties 类对象，并进行读取和修改。

Collection 工具类：
排序操作：
1）reverse(List)：反转 List 中元素的顺序；
2）shuffle(List)：对 List 集合元素进行随机排序；
3）sort(List)：根据元素的自然顺序指定 List 集合元素按照升序排列；
4）sort(List, Comparator)：根据指定的 Comparator 产生的顺序对 List 集合元素进行排序；
5）swap(List, int, int)：将指定 List 集合中的 i 处元素和 j 处元素进行交换。
6）Object max(Collection)：根据元素的自然顺序，返回给定集合中的最大元素
7）Object max(Collection, Comparator)：根据 Comparator 指定的顺序，返回给定集合中的最大元素；
8）Object min(Collection)；
9）Object min(Collection, Comparator)；
10）Int frequency(Collection, Object)：返回指定集合中指定元素的出现次数；
11）Void copy(List dest, List src)：将 src 中的内容复制到 dest 中；
12）Boolean replaceAll(List list, Object oldVal, Object newVal)：使用新值替换 List 对象的所有旧值。