1. 存儲的數據特點：無序的、不可重複的元素
2. 使用場景:可以用來過濾List中重複的元素: hashset.addAll(list)

具體的：

*1.无序性：不等于随机性。存储的数据在底层数组中并非按照数组索引的顺序添加，而是根据数据的哈希值决定的。  
2. 不可重复性：保证添加的元素按照equals()判断时，不能返回true.即：相同的元素只能添加一个。*

1. 元素添加過程：（以HashSet為例）(了解就好)

*我们向HashSet中添加元素a,首先调用元素a所在类的hashCode()方法，计算元素a的hash值，此hash值接着通过某种算法计算出在HashSet底层数组中的存放位置（即为：索引位置），判断数组此位置上是否已经有元素：  
 如果此位置上没有其他元素，则元素a添加成功。 --->情况1  
 如果此位置上有其他元素b(或以链表形式存在的多个元素），则比较元素a与元素b的hash值：  
 如果hash值不相同，则元素a添加成功。--->情况2  
 如果hash值相同，进而需要调用元素a所在类的equals()方法：  
 equals()返回true,元素a添加失败  
 equals()返回false,则元素a添加成功。--->情况3　  
对于添加成功的情况2和情况3而言：元素a 与已经存在指定索引位置上数据以链表的方式存储。  
jdk 7 :元素a放到数组中，指向原来的元素。新元素會取代舊元素位置,再與舊元素鏈結,順序為3>2>1的垂直链表結構  
jdk 8 :原来的元素在数组中，指向元素a。新元素會在舊元素的下面,再與舊元素鏈結,順序為1>2>3的垂直链表結構  
总结：七上八下  
HashSet底层：数组+链表的结构。(前提: jdk7)*

(每次加進新元素前，都要先算它的Hash值經過某種算法找出其對應位置，看此對應位置是否已有元素，沒有則添加成功，有其他元素的話先比較其Hash值，不同則添加成功，如果相同再比較equals()，不同則添加成功，相同則添加失敗。)

1. 常用方法

*Set接口中没有额外定义新的方法，使用的都是Collection中声明过的方法。*

1. 常用實現類

*|----Collection接口：单列集合，用来存储一个一个的对象  
 |----Set接口：存储无序的、不可重复的数据 -->高中讲的“集合”  
 |----HashSet：作为Set接口的主要实现类；线程不安全的；可以存储null值  
 |----LinkedHashSet：作为HashSet的子类；遍历其内部数据时，可以按照添加的顺*

*序在添加数据的同时，每个数据还维护了两个引用，记录此数据前一个数据和后一个*

*数据。遍历对于频繁的遍历操作，LinkedHashSet效率高于HashSet.*

*|----TreeSet：可以按照添加对象的指定属性，进行排序。*

1. 存儲對象所在類的要求：

*HashSet / LinkedHashSet：*

*要求：向Set(主要指：HashSet、LinkedHashSet)中添加的数据，其所在的类一定要重写hashCode()和equals()  
要求：重写的hashCode()和equals()尽可能保持一致性：相等的对象必须具有相等的散列码重写两个方法的小技巧：对象中用作 equals() 方法比较的 Field，都应该用来计算 hashCode 值。*

*範例:*

@Test

**public** **void** test3(){

HashSet set = **new** HashSet();

Person p1 = **new** Person(1001,"AA");

Person p2 = **new** Person(1002,"BB");

set.add(p1);

set.add(p2);

System.out.println(set);//[Person{id=1002, name='BB'}, Person{id=1001,name='AA'}]

p1.name = "CC";

set.remove(p1);

/\*

\* 對p1屬性進行修改其對應的位置不變,而set.remove(p1)要刪除的是"Person{id=1001,

name='CC'}"的hash值的對應位置,原p1的hash值是"Person{id=1001, name='AA'}"的,所

以是不同位置,所以沒刪除到東西

\*/

System.out.println(set);//[Person{id=1002, name='BB'}, Person{id=1001,name='CC'}]

set.add(**new** Person(1001,"CC"));

/\*

\* 添加的是"new Person(1001,"CC")",跟原來的"new Person(1001,"AA")"是不同hash值,所以

對應位置不同,所以添加成功

\*/

System.out.println(set);//[Person{id=1002, name='BB'}, Person{id=1001,name='CC'},

Person{id=1001, name='CC'}]

set.add(**new** Person(1001,"AA"));

/\*

\* 跟原來p1的位置相同,可以p1對屬性進行修改,所以兩個equals判斷是不同的,所以添加成功

\*/

System.out.println(set);//[Person{id=1002, name='BB'}, Person{id=1001,name='CC'},

Person{id=1001, name='CC'}, Person{id=1001,name='AA'}]

}

*TreeSet：*

1. *自然排序中，比较两个对象是否相同的标准为：compareTo()返回0.不再是equals().*
2. *定制排序中，比较两个对象是否相同的标准为：compare()返回0.不再是equals().*

*(比較其方法重寫中的屬性是否相同,如果相同會認為是同一個對象就不能添加,不同則依重寫中的排序要求進行排序。)*

1. TreeSet的使用
   1. 使用說明：

*1.向TreeSet中添加的数据，要求是相同类的对象。  
2.两种排序方式：自然排序（实现Comparable接口） 和 定制排序（Comparator）*

* 1. 常用的排序方式：

// 方式一：自然排序

@Test

public void test1(){  
 TreeSet set = new TreeSet();  
 *//失败：不能添加不同类的对象  
// set.add(123); // set.add(456);  
// set.add("AA"); // set.add(new User("Tom",12));  
//举例一：輸出結果為-34 8 11 34 43  
// set.add(34);  
// set.add(-34);  
// set.add(43);  
// set.add(11);  
// set.add(8);  
//举例二：* set.add(new User("Tom",12));  
 set.add(new User("Jerry",32));  
 set.add(new User("Jim",2));  
 set.add(new User("Mike",65));  
 set.add(new User("Jack",33));  
 set.add(new User("Jack",56));  
 Iterator iterator = set.iterator();  
 while(iterator.hasNext()){  
 System.*out*.println(iterator.next());  
 }  
 }  
 @Test  
 */\*  
 小結:  
 如需使用定制排序則在TreeSet的構造器中添加comparator去重寫compare(Object o1, Object o2),  
 來比較其當中的屬性是否相同,如果相同會認為是同一個對象就不能添加,不同則依重寫中的排序要求進行排*

*序。  
 TreeSet set = new TreeSet(com);  
 \*/*

//方式二:定制排序public void test2(){  
 Comparator com = new Comparator() {  
 *//按照年龄从小到大排列* @Override  
 public int compare(Object o1, Object o2) {  
 if(o1 instanceof User && o2 instanceof User){  
 User u1 = (User)o1;  
 User u2 = (User)o2;  
 return Integer.*compare*(u1.getAge(),u2.getAge());  
 }else{  
 throw new RuntimeException("输入的数据类型不匹配");  
 }  
 }  
 };  
 TreeSet set = new TreeSet(com);  
 set.add(new User("Tom",12));  
 set.add(new User("Jerry",32));  
 set.add(new User("Jim",2));  
 set.add(new User("Mike",65));  
 set.add(new User("Mary",33));  
 set.add(new User("Jack",33));  
 set.add(new User("Jack",56));  
 Iterator iterator = set.iterator();  
 while(iterator.hasNext()){  
 System.*out*.println(iterator.next());  
 }  
 }