# 一、Set接口的框架

|----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。（自然排序和自定义排序）

1. 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。

2. 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。

二、Set：存储无序的、不可重复的数据

以HashSet为例说明：

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。

三、Set的实现类

1、HashSet类

1.添加元素的过程

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的哈希值，

此哈希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2

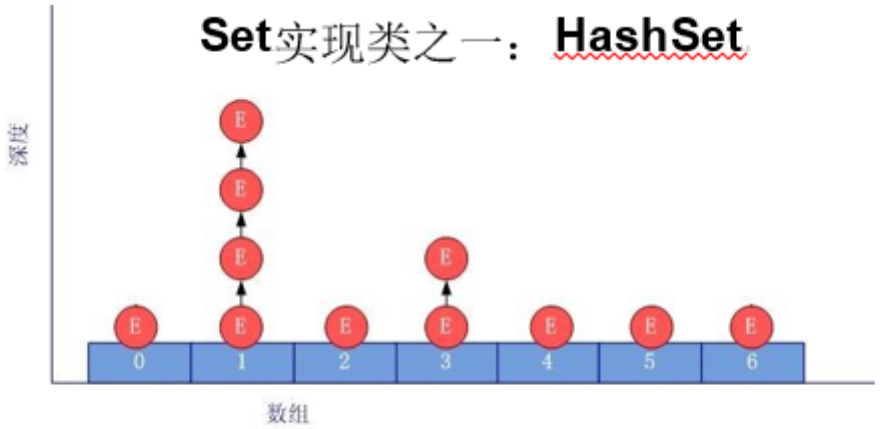
对于添加成功的情况2和情况3而言：元素a 与已经存在指定索引位置上数据以链表的方式存储。

jdk 7 :元素a放到数组中，指向原来的元素。

jdk 8 :原来的元素在数组中，指向元素a

总结：七上八下

HashSet底层：数组+链表的结构。




底层也是数组，初始容量为16，当如果使用率超过0.75，（16*0.75=12）就会扩大容量为原来的2倍。（16扩容为32，依次为64,128....等）

```
@Test
public void test1(){
Set set = new HashSet();
set.add(456);
set.add(123);
set.add(123);
set.add("AA");
set.add("CC");
set.add(new User("Tom",12));
 set.add(new User("Tom",12));
 set.add(129);

 Iterator iterator = set.iterator();
 while(iterator.hasNext()){
 System.out.println(iterator.next());
 }
 }
```

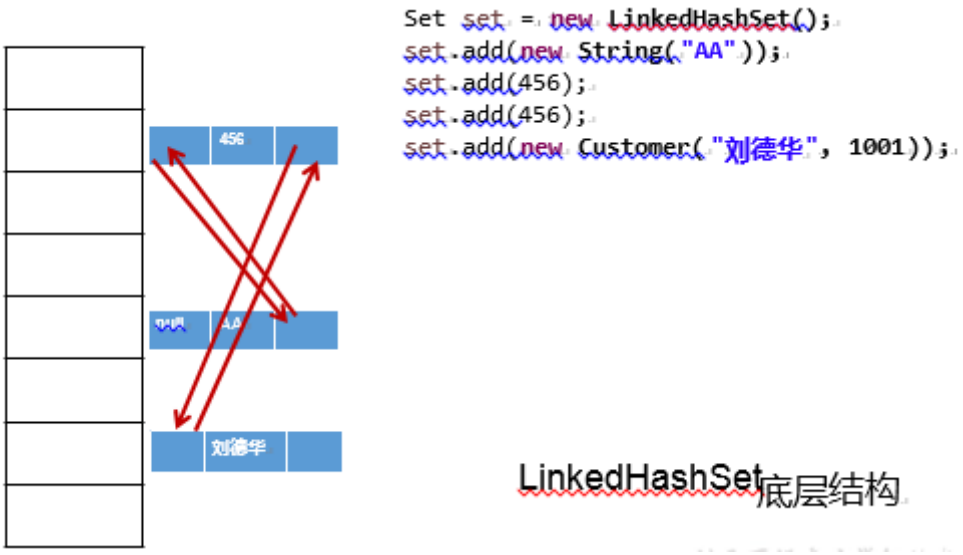
2、LinkedHashSet类

1.LinkedHashSet作为HashSet的子类

2.LinkedHashSet根据元素的hashcode的值决定元素的存储位置，但它同时采用双向链表的维护元素的次序，使得元素看起来是以插入顺序保存的。

3.LinkedHashSet的插入性能略低于HashSet，但是在迭代访问Set里的全部元素时有很好的性能

4.不允许集合元素重复。




```java
@Test
public void test2(){
Set set = new LinkedHashSet();
set.add(456);
set.add(123);
set.add(123);
set.add("AA");
set.add("CC");
set.add(new User("Tom",12));
 set.add(new User("Tom",12));
 set.add(129);

 Iterator iterator = set.iterator();
 while(iterator.hasNext()){
 System.out.println(iterator.next());
 }
 }
```

3、TreeSet类

1、TreeSet是SortedSet接口的实现类，TreeSet可以确保集合元素处于排序状态

2、TreeSet底层采用红黑树结构存储数据

3、两种排序方法：自然排序和定制排序。默认情况下，TreeSet采用自然排序

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）

Comparable 的典型实现：

BigDecimal、BigInteger 以及所有的数值型对应的包装类：按它们对应的数值大小进行比较

Character：按字符的 unicode值来进行比较

Boolean：true 对应的包装类实例大于false 对应的包装类实例

String：按字符串中字符的unicode 值进行比较

Date、Time：后边的时间、日期比前面的时间、日期大

3.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().

4.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().

```
public class User implements Comparable{
 private String name;
 private int age;

 public User() {
 }

 public User(String name, int age) {
```

```
9   this.name = name;
10   this.age = age;
11   }
12
13   public String getName() {
14   return name;
15   }
16
17   public void setName(String name) {
18   this.name = name;
19   }
20
21   public int getAge() {
22   return age;
23   }
24
25   public void setAge(int age) {
26   this.age = age;
27   }
28
29   @Override
30   public String toString() {
31   return "User{" +
32   "name='" + name + '\'' +
33   ", age=" + age +
34   '}';
35   }
36
37   @Override
38   public boolean equals(Object o) {
39   System.out.println("User equals()....");
40   if (this == o) return true;
41   if (o == null || getClass() != o.getClass()) return false;
42
43   User user = (User) o;
```

```
44
45  if (age != user.age) return false;
46  return name != null ? name.equals(user.name) : user.name == nu
 ll;
47  }
48
49  @Override
50  public int hashCode() { //return name.hashCode() + age;
51  int result = name != null ? name.hashCode() : 0;
52  result = 31 * result + age;
53  return result;
54  }
55
56  //按照姓名从大到小排列,年龄从小到大排列
57  @Override
58  public int compareTo(Object o) {
59  if(o instanceof User){
60  User user = (User)o;
61 // return -this.name.compareTo(user.name);
62  int compare = -this.name.compareTo(user.name);
63  if(compare != 0){
64  return compare;
65  }else{
66  return Integer.compare(this.age,user.age);
67  }
68  }else{
69  throw new RuntimeException("输入的类型不匹配");
70  }
71
72  }
73 }
74
75 import org.junit.Test;
76
77 import java.util.Comparator;
```

```java
import java.util.Iterator;
import java.util.TreeSet;


public class TreeSetTest {

    /*
    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）


    3.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().
    4.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().
    */
    @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：
//  set.add(34);
//  set.add(-34);
//  set.add(43);
//  set.add(11);
//  set.add(8);

     //举例二：
     set.add(new User("Tom",12));
```

```
111  set.add(new User("Jerry",32));
112  set.add(new User("Jim",2));
113  set.add(new User("Mike",65));
114  set.add(new User("Jack",33));
115  set.add(new User("Jack",56));
116
117
118  Iterator iterator = set.iterator();
119  while(iterator.hasNext()){
120  System.out.println(iterator.next());
121  }
122
123  }
124
125  @Test
126  public void test2(){
127  Comparator com = new Comparator() {
128  //按照年龄从小到大排列
129  @Override
130  public int compare(Object o1, Object o2) {
131  if(o1 instanceof User && o2 instanceof User){
132  User u1 = (User)o1;
133  User u2 = (User)o2;
134  return Integer.compare(u1.getAge(),u2.getAge());
135  }else{
136  throw new RuntimeException("输入的数据类型不匹配");
137  }
138  }
139  };
140
141  TreeSet set = new TreeSet(com);
142  set.add(new User("Tom",12));
143  set.add(new User("Jerry",32));
144  set.add(new User("Jim",2));
145  set.add(new User("Mike",65));
```

```
146   set.add(new User("Mary",33));
147   set.add(new User("Jack",33));
148   set.add(new User("Jack",56));
149
150
151   Iterator iterator = set.iterator();
152   while(iterator.hasNext()){
153   System.out.println(iterator.next());
154   }
155   }
156
157 }
```

四、面试题：

```
1
```