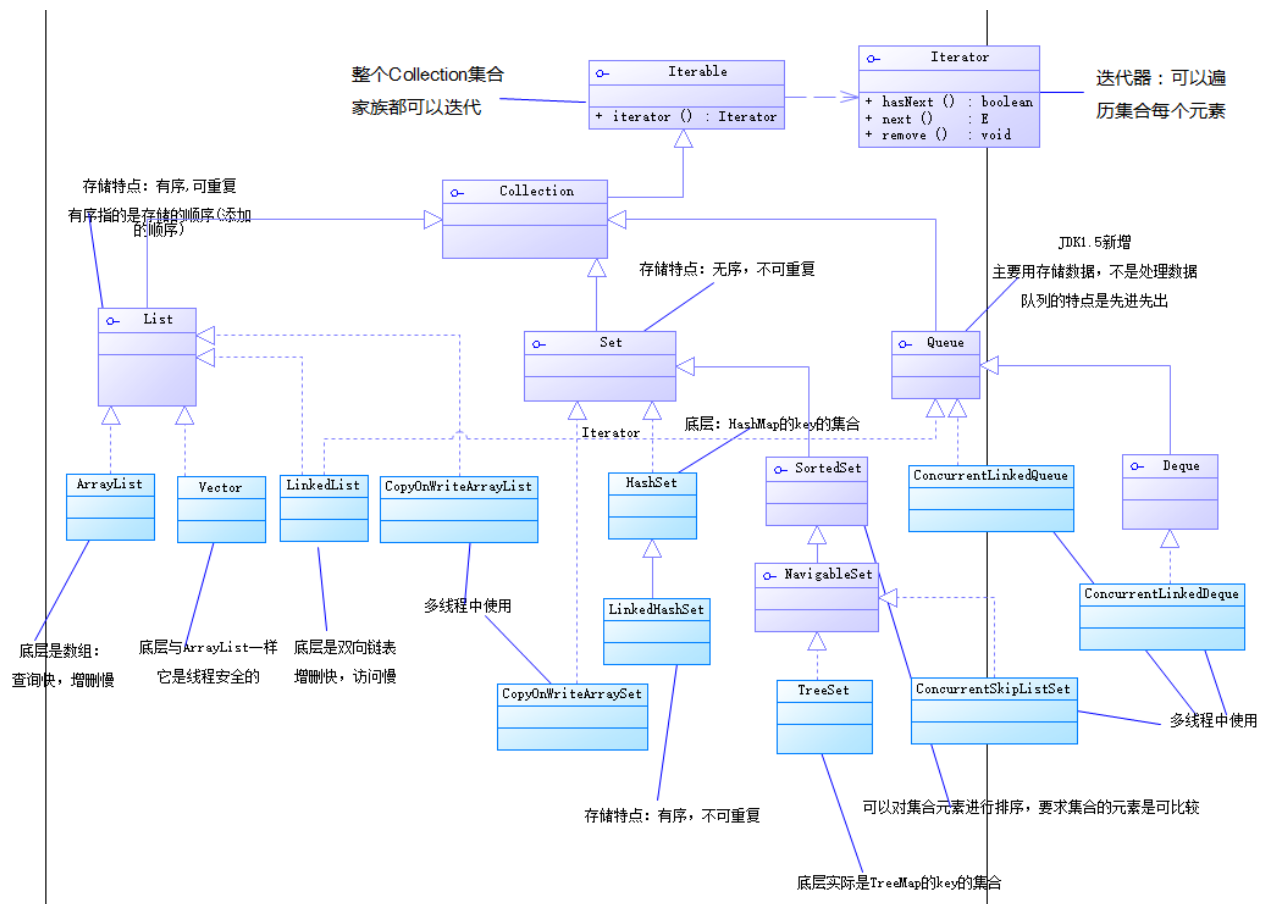


学习目标

- * 能够理解Set接口概述
 - * 无序，不可重复
- * 能够通过阅读HashSet源码理解它底层实现就是HashMap的key集合
 - * HashMap的key集合
 - * 重写hashCode，equals‘
 - * ...
- * 能够通过阅读LinkedHashSet源码理解它底层实现就是LinkedHashMap的key集合
- * 能够通过阅读TreeSet源码理解它底层实现就是TreeMap的key集合
- * 能够掌握Collections常见的方法
 - * addAll,shuffle,reverse,sort,binarySearch
 - * ...(可变参数)
 - * sort：（ unicode编码 ）
- * 能够掌握并发集合类之CopyOnWriteArrayList （ ArrayList ）
 - * ArrayList
- * 能够掌握并发集合类之ConcurrentHashMap （ HashMap ）
 - * HashMap
- * 能够掌握并发集合类之ConcurrentSkipListMap （ TreeMap ）
- * 能够掌握并发集合类之 ConcurrentSkipListSet （ TreeSet ）
- * 能够掌握并发集合类之CopyOnWriteArraySet （ HashSet ）
- * 能够掌握并发集合类之ConcurrentLinkedDeque （ LinkedList ）

-
- * 能够理解Set接口概述
 - * 特点：无序，不可重复



* 能够通过阅读HashSet源码理解它底层实现就是HashMap的key集合

```

1 * HashSet 核心代码
2 public class HashSet<E>
3     extends AbstractSet<E>
4     implements Set<E>, Cloneable, java.io.Serializable
5 {
6     // ...
7     private transient HashMap<E, Object> map;
8     private static final Object PRESENT = new Object();
9     // 构造器
10    public HashSet() {
11        map = new HashMap<>();
12    }
13    public HashSet(Collection<? extends E> c) {
14        map = new HashMap<>(Math.max((int) (c.size()/.75f) + 1, 16));
15        addAll(c);
16    }
17    public HashSet(int initialCapacity, float loadFactor) {
18        map = new HashMap<>(initialCapacity, loadFactor);
19    }

```

```

20     public HashSet(int initialCapacity) {
21         map = new HashMap<>(initialCapacity);
22     }
23     HashSet(int initialCapacity, float loadFactor, boolean dummy) {
24         map = new LinkedHashMap<>(initialCapacity, loadFactor);
25     }
26     public boolean add(E e) {
27         return map.put(e, PRESENT)!=null;
28     }
29     public Iterator<E> iterator() {
30         return map.keySet().iterator();
31     }
32     public boolean remove(Object o) {
33         return map.remove(o)==PRESENT;
34     }
35     public boolean contains(Object o) {
36         return map.containsKey(o);
37     }
38     public void clear() {
39         map.clear();
40     }
41     public int size() {
42         return map.size();
43     }
44     public boolean isEmpty() {
45         return map.isEmpty();
46     }
47     // ...
48 }
49
50 * 总结
51 * HashSet底层是HashMap
52 * 调用add()添加元素，实际上是把该元素作为键添加到了HashMap中
53 * HashSet实际上就是HashMap的键的集合
54 * 因为Map中的键不允许重复,所以Set集合中的元素不重复
55 * HashSet添加元素：建议元素重写equals, hashCode
56
57 * 案例：
58 创建一个ArrayList的集合对象，该集合对象存储一批图书，该图书存在着重复元素
59 定义一个方法清除重复元素。返回一个没有重复元素的集合对象。

```

```

60 public static void main(String[] args) {
61     List<Book> books=new ArrayList<Book>();
62     books.add(new Book(1, "Java", 28.9));
63     books.add(new Book(2, "html", 30.9));
64     books.add(new Book(1, "Java", 28.9));
65     books.add(new Book(1, "Java", 28.9));
66     books.add(new Book(1, "Java", 28.9));
67     Set<Book> books2 = getBooks(books);
68     for(Book book:books2) {
69         System.out.println(book);
70     }
71 }
72 // 定义一个方法清除重复元素。返回一个没有重复元素的集合对象
73 public static Set<Book> getBooks(List<Book> books){
74     HashSet<Book> set=new HashSet<Book>(books);
75     return set;
76 }
77 结果:
78 Book [id=1, name=Java, price=28.9]
79 Book [id=2, name=html, price=30.9]

```

* 能够通过阅读LinkedHashSet源码理解它底层实现就是LinkedHashMap的key集合

```

1 public class LinkedHashSet<E>
2     extends HashSet<E>
3     implements Set<E>, Cloneable, java.io.Serializable {
4     public LinkedHashSet(int initialCapacity, float loadFactor) {
5         super(initialCapacity, loadFactor, true);
6     }
7     public LinkedHashSet(int initialCapacity) {
8         super(initialCapacity, .75f, true);
9     }
10    public LinkedHashSet() {
11        super(16, .75f, true);
12    }
13    //...
14 }
15 * 总结:
16 * LinkedHashSet 继承 HashSet, 调用HashSet三个构造器的方法

```

```

17     * HashSet(int initialCapacity, float loadFactor, boolean dummy) {
18         map = new LinkedHashMap<>(initialCapacity, loadFactor);
19     }
20     * LinkedHashSet底层是LinkedHashMap
21     * LinkedHashSet 是有序的
22
23     * 例子演示
24     * HashSet: 是无序
25     public static void main(String[] args) {
26         Set<String> set=new HashSet<String>();
27         set.add("刘备");
28         set.add("关羽");
29         set.add("张飞");
30         for(String value:set) {
31             System.out.printf("%s ",value);
32         }
33     }
34     结果:
35     关羽 张飞 刘备
36     * LinkedHashSet 是有序的
37     public static void main(String[] args) {
38         Set<String> set=new LinkedHashSet<String>();
39         set.add("刘备");
40         set.add("关羽");
41         set.add("张飞");
42         for(String value:set) {
43             System.out.printf("%s ",value);
44         }
45     }
46     * 结果:
47     刘备 关羽 张飞

```

* 能够通过阅读TreeSet源码理解它底层实现就是TreeMap的key集合

```

1  * TreeSet 核心源码
2  public class TreeSet<E> extends AbstractSet<E>
3      implements NavigableSet<E>, Cloneable, java.io.Serializable
4  {
5      private transient NavigableMap<E, Object> m;

```

```
6     private static final Object PRESENT = new Object();
7     TreeSet(NavigableMap<E, Object> m) {
8         this.m = m;
9     }
10    public TreeSet() {
11        this(new TreeMap<E, Object>());
12    }
13    public TreeSet(Comparator<? super E> comparator) {
14        this(new TreeMap<>(comparator));
15    }
16    public TreeSet(Collection<? extends E> c) {
17        this();
18        addAll(c);
19    }
20    public TreeSet(SortedSet<E> s) {
21        this(s.comparator());
22        addAll(s);
23    }
24    public Iterator<E> iterator() {
25        return m.navigableKeySet().iterator();
26    }
27    public int size() {
28        return m.size();
29    }
30    public boolean isEmpty() {
31        return m.isEmpty();
32    }
33    public boolean contains(Object o) {
34        return m.containsKey(o);
35    }
36    public boolean add(E e) {
37        return m.put(e, PRESENT) != null;
38    }
39    public boolean remove(Object o) {
40        return m.remove(o) == PRESENT;
41    }
42    public void clear() {
43        m.clear();
44    }
45    //..
```

```

46 }
47
48 * TreeSet底层使用TreeMap的Key
49 * 使用TreeMap前提
50 * 要么Key的类实现了Comparable接口
51 * 假如Key的类没有实现Comparable接口，就需要使用Comparator比较器
52
53 * 例子
54 * 存储一批员工对象 name 、 salary， 请根据薪水排序
55 public static void main(String[] args) {
56     Set<Emp> emps=new TreeSet<Emp>(new Comparator<Emp>() {
57         @Override
58         public int compare(Emp o1, Emp o2) {
59             return (int) (o1.getSalary()-o2.getSalary());
60         }
61     });
62     emps.add(new Emp(1001,"刘备",8500));
63     emps.add(new Emp(1002,"关羽",18500));
64     emps.add(new Emp(1003,"张飞",3500));
65
66     for(Emp emp:emps) {
67         System.out.println(emp);
68     }
69 }
70 结果:
71 Emp [empId=1003, name=张飞, salary=3500.0]
72 Emp [empId=1001, name=刘备, salary=8500.0]
73 Emp [empId=1002, name=关羽, salary=18500.0]

```

* 能够掌握Collections常见的方法

* addAll,shuffle,reverse,sort,binarySearch

```

1 * 可变参数
2 public class Test1 {
3
4     public static void main(String[] args) {
5         // printArgs(1, 2, 3, 4, 5);
6         int[] arrs= {1,2,3,4,5};

```

```
7         printArgs1(arrs);
8         printArgs1(arrs);
9     }
10
11     public static void printArgs(int a,int b,int c,int d, int e) {
12         System.out.println(a);
13         System.out.println(b);
14         System.out.println(c);
15         System.out.println(d);
16         System.out.println(e);
17     }
18
19     public static void printArgs(int[] arrs) {
20         for (int i = 0; i < arrs.length; i++) {
21             System.out.println(arrs[i]);
22         }
23     }
24
25     // 可变参数: jdk1.5
26     public static void printArgs1(int... arrs) {
27         for (int i = 0; i < arrs.length; i++) {
28             System.out.println(arrs[i]);
29         }
30     }
31
32 }
33
34 *
35 package com.lg.test3;
36
37 import java.util.ArrayList;
38 import java.util.Collections;
39 import java.util.Comparator;
40 import java.util.List;
41
42 public class Test2 {
43     public static void main(String[] args) {
44         List<String> list=new ArrayList<String>();
45         // A B C D E
46         list.add("刘备");// \u5218\u5907
```



```

47     List<String> subList=new ArrayList<String>();
48     subList.toArray();
49     subList.add("赵云");// \u8d75\u4e91
50     subList.add("黄忠");// \u9ec4\u5fe0
51     list.addAll(subList);
52     //可变参数...      \u5173\u7fd4
53     Collections.addAll(list,"关羽","张飞");
54 // Collections.addAll(list,subList.toArray(new String[0]));
55     System.out.println(list);
56     // [关羽, 刘备, 张飞, 赵云, 黄忠]
57     // \u5173\u7fd4 \u5218\u5907 \u5f20\u98de \u8d75\u4e91 \u9ec4\u5fe0
58     Collections.shuffle(list);
59     System.out.println(list);
60     int index = Collections.binarySearch(list, "刘备");
61     System.out.println(index);
62
63     Collections.reverse(list);
64     System.out.println(list);
65
66     Collections.sort(list,new Comparator<String>() {
67         @Override
68         public int compare(String o1, String o2) {
69             return o1.compareTo(o2);
70         }
71     });
72     System.out.println(list);// Unicode 编码
73
74     index = Collections.binarySearch(list, "刘备");
75     System.out.println(index);
76
77     Collections.reverse(list);
78     System.out.println(list);
79
80     index = Collections.binarySearch(list, "刘备");
81     System.out.println(index);
82 }
83 }
84
85 * String compareTo重新认识
86 import java.util.ArrayList;

```

```

87 import java.util.Collections;
88 import java.util.List;
89
90 public class Test3 {
91     public static void main(String[] args) {
92         List<String> list=new ArrayList<String>();
93         list.add("李白");// \u674e\u767d
94         list.add("杜甫");// \u675c\u752b
95         list.add("白居易");// \u767d\u5c45\u6613
96         list.add("孟浩然");// \u5b5f\u6d69\u7136
97         list.add("屈原");// \u5c48\u539f
98         // 孟浩然: \u5b5f\u6d69\u7136 23391
99         //          5b5f 6d69 7136
100        // 屈原: \u5c48\u539f 23624
101        // 李白: \u674e\u767d 26446
102        // 杜甫: \u675c\u752b 26460
103        // 白居易: \u767d\u5c45\u6613 30333
104        // 假如是
105        Collections.sort(list);
106        System.out.println(list);
107
108
109        String str="中国";//\u4e2d\u56fd
110        String name="白居易";
111        char[] chs = name.toCharArray();
112        for (int i = 0; i < chs.length; i++) {
113            // System.out.println(chs[i]);
114            System.out.println(Integer.valueOf(chs[i]));
115        }
116        // Set<String> set=new TreeSet<String>();
117        // set.add("李白");// L
118        // set.add("杜甫");// D
119        // set.add("白居易");// B
120        // set.add("孟浩然");// M
121        // set.add("屈原");// Q
122        // System.out.println(set);
123        二分查找();
124    }
125
126    public static void 二分查找() {

```

```
127
128     }
129 }
130
```

* 能够掌握并发集合类之CopyOnWriteArrayList (ArrayList)

```
1 * 多个线程同时操作并且遍历集合
2 * ArrayList
3 // 多个线程同时操作并且遍历集合
4 public static List<String> list=new ArrayList<String>();
5 public static void main(String[] args) {
6     new Thread(new Runnable() {
7
8         @Override
9         public void run() {
10             for (int i = 0; i < 10; i++) {
11                 list.add(String.valueOf(i));
12             }
13             for (String v:list) {
14                 System.out.printf("%s ",v);
15             }
16             System.out.println();
17         }
18     }).start();
19
20     new Thread(new Runnable() {
21
22         @Override
23         public void run() {
24             for (int i = 10; i < 20; i++) {
25                 list.add(String.valueOf(i));
26             }
27             for (String v:list) {
28                 System.out.printf("%s ",v);
29             }
30             System.out.println();
31         }
32     }).start();
```

```

33     }
34 结果:
35 出现了异常: java.util.ConcurrentModificationException
36
37 * CopyOnWriteArrayList
38 * public static List<String> list=new CopyOnWriteArrayList<String>();
39 某次的结果:
40     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
41

```

* 能够掌握并发集合类之ConcurrentHashMap (HashMap)

```

1 * 多个线程同时操作并且遍历集合
2 * HashMap
3 public static Map<String,String> map=new HashMap<String,String>();
4 public static void main(String[] args) {
5     new Thread(new Runnable() {
6
7         @Override
8         public void run() {
9             for (int i = 0; i < 10; i++) {
10                 map.put(String.valueOf(i), String.valueOf(i));
11             }
12             Set<String> keys = map.keySet();
13             for(String key:keys) {
14                 System.out.printf("%s ",key);
15             }
16             System.out.println();
17         }
18     }).start();
19
20     new Thread(new Runnable() {
21
22         @Override
23         public void run() {
24             for (int i = 10; i < 20; i++) {
25                 map.put(String.valueOf(i), String.valueOf(i));
26             }
27             Set<String> keys = map.keySet();

```

```

28         for(String key:keys) {
29             System.out.printf("%s ",key);
30         }
31         System.out.println();
32     }
33     }).start();
34 }
35 结果:
36 出现了异常: java.util.ConcurrentModificationException
37
38 * ConcurrentHashMap
39 public static Map<String,String> map=new ConcurrentHashMap<String,String>();
40 * 某次运行结果:
41 0 11 12 13 14 15 16 17 18 1 19 0 1 2 3 4 5 6 7 8 9 10 13
42 2 14 3 15 4 16 5 17 6 18 7 19 8 9 10 （结果不确定）
43

```

* 能够掌握并发集合类之ConcurrentSkipListMap (TreeMap)

```

1 * TreeMap (数要大点才能测试异常)
2 测试:
3 public static Map<String,String> map=new TreeMap<String,String>();
4     public static void main(String[] args) {
5         new Thread(new Runnable() {
6
7             @Override
8             public void run() {
9                 for (int i = 0; i < 100; i++) {
10                     map.put(String.valueOf(i), String.valueOf(i));
11                 }
12                 Set<String> keys = map.keySet();
13                 for(String key:keys) {
14                     System.out.printf("%s ",key);
15                 }
16                 System.out.println();
17             }
18         }).start();
19
20     new Thread(new Runnable() {

```

```

21
22         @Override
23         public void run() {
24             for (int i = 10; i < 2000; i++) {
25                 map.put(String.valueOf(i), String.valueOf(i));
26             }
27             Set<String> keys = map.keySet();
28             for(String key:keys) {
29                 System.out.printf("%s ",key);
30             }
31             System.out.println();
32         }
33     }).start();
34 }

```

36 结果:

37 出现异常: java.util.ConcurrentModificationException

```

38
39 * ConcurrentSkipListMap
40 public static Map<String,String> map=
41 new ConcurrentSkipListMap<String,String>();
42 * 使用ConcurrentSkipListMap替换

```

* 能够掌握并发集合类之 ConcurrentSkipListSet (TreeSet)

* 能够掌握并发集合类之CopyOnWriteArraySet (HashSet)

```

1  * 测试HashSet
2  public static Set<String> set=new HashSet<String>();
3      public static void main(String[] args) {
4          new Thread(new Runnable() {
5
6              @Override
7              public void run() {
8                  for (int i = 0; i < 10; i++) {
9                      set.add(String.valueOf(i));
10                 }
11                 for (String v:set) {
12                     System.out.printf("%s ",v);

```

```

13         }
14         System.out.println();
15     }
16 }).start();
17
18     new Thread(new Runnable() {
19
20         @Override
21         public void run() {
22             for (int i = 10; i < 20; i++) {
23                 set.add(String.valueOf(i));
24             }
25             for (String v:set) {
26                 System.out.printf("%s ",v);
27             }
28             System.out.println();
29         }
30     }).start();
31 }
32
33 结果:
34     出现异常: java.util.ConcurrentModificationException、
35 * CopyOnWriteArraySet
36 * public static Set<String> set=new CopyOnWriteArraySet<String>();

```

* 能够掌握并发集合类之ConcurrentLinkedDeque (LinkedList)

```

1  * LinkedList做双向队列来使用
2  public static Deque<String> deque=new LinkedList<String>();
3      public static void main(String[] args) {
4          new Thread(new Runnable() {
5
6              @Override
7              public void run() {
8                  for (int i = 0; i < 10; i++) {
9                      deque.add(String.valueOf(i));
10                 }
11                 for (String v:deque) {
12                     System.out.printf("%s ",v);

```

```
13         }
14         System.out.println();
15     }
16 }).start();
17
18     new Thread(new Runnable() {
19
20         @Override
21         public void run() {
22             for (int i = 10; i < 20; i++) {
23                 deque.add(String.valueOf(i));
24             }
25             for (String v:deque) {
26                 System.out.printf("%s ",v);
27             }
28             System.out.println();
29         }
30     }).start();
31 }
32 * 出现异常: java.util.ConcurrentModificationException
33
34 * ConcurrentLinkedDeque
35     * public static Deque<String> deque=new ConcurrentLinkedDeque<String>();
36
37 * ConcurrentLinkedQueue 这个是单向
```