第 19 天 集合

第1章 List 接口

我们掌握了 Collection 接口的使用后，再来看看 Collection 接口中的子类，他们都具备那些特性呢？

接下来，我们一起学习 Collection 中的常用几个子类（List 集合、Set 集合）。

1.1 List 接口介绍

查阅 API，看 List 的介绍。有序的 collection（也称为序列）。此接口的用户可以对列表中每个元素的插入位置进行精确地控制。用户可以根据元素的整数索引（在列表中的位置）访问元素，并搜索列表中的元素。与 set 不同，列表通常允许重复的元素。

看完 API，我们总结一下：

List 接口：

- 它是一个元素存取有序的集合。例如，存元素的顺序是 11、22、33。那么集合中，元素的存储就是按照 11、22、33 的顺序完成的）。
- 它是一个带有索引的集合，通过索引就可以精确的操作集合中的元素（与数组的索引是一个道理）。
- 集合中可以有重复的元素，通过元素的 equals 方法，来比较是否为重复的元素。

List 接口的常用子类有：

- ArrayList 集合
- LinkedList 集合

1.2 List 接口中常用的方法

boolean	**add**(E e) 向列表的尾部添加指定的元素（可选操作）。
void	**add**(int index, E element) 在列表的指定位置插入指定元素（可选操作）。
E	**get**(int index) 返回列表中指定位置的元素。
E	**remove**(int index) 移除列表中指定位置的元素（可选操作）。
boolean	**remove**(Object o) 从此列表中移除第一次出现的指定元素（如果存在）（可选操作）。
E	**set**(int index, E element) 用指定元素替换列表中指定位置的元素（可选操作）。

- 增加元素方法
  - add(Object e)：向集合末尾处，添加指定的元素
  - add(int index, Object e)：向集合指定索引处，添加指定的元素，原有元素依次后移
- 删除元素删除
  - remove(Object e)：将指定元素对象，从集合中删除，返回值为被删除的元素
  - remove(int index)：将指定索引处的元素，从集合中删除，返回值为被删除的元素
- 替换元素方法
  - set(int index, Object e)：将指定索引处的元素，替换成指定的元素，返回值为替换前的元素
- 查询元素方法
  - get(int index)：获取指定索引处的元素，并返回该元素

方法演示：

```
List<String> list = new ArrayList<String>();
//1,添加元素。
list.add("小红");
list.add("小梅");
list.add("小强");
//2,插入元素。插入元素前的集合["小红","小梅","小强"]
list.add(1, "老王"); //插入元素后的集合["小红","老王","小梅","小强"]
//3,删除元素。
list.remove(2);// 删除元素后的集合["小红","老王","小强"]
//4,修改元素。
list.set(1, "隔壁老王");// 修改元素后的集合["小红","隔壁老王","小强"]

Iterator<String> it = list.iterator();
while (it.hasNext()) {
    String str = it.next();
    System.out.println(str);
}
```

由于 List 集合拥有索引，因此 List 集合迭代方式除了使用迭代器之外，还可以使用索引进行迭代。

```
for (int i = 0; i < list.size(); i++) {
    String str = list.get(i);
    System.out.println(str);
}
```

1.2.1 Iterator 的并发修改异常

在 list 集合迭代元素中，对元素进行判断，一旦条件满足就添加一个新元素。代码如下

```
public class IteratorDemo {
//在 list 集合迭代元素中，对元素进行判断，一旦条件满足就添加一个新元素
```

```
    public static void main(String[] args) {
        //创建 List 集合
        List<String> list = new ArrayList<String>();
        //给集合中添加元素
        list.add("abc1");
        list.add("abc2");
        list.add("abc3");
        list.add("abc4");
        //迭代集合，当有元素为"abc2"时，集合加入新元素"itcast"
        Iterator<String> it = list.iterator();
        while(it.hasNext()){
            String str = it.next();
            //判断取出的元素是否是"abc2"，是就添加一个新元素
            if("abc2".equals(str)){
                list.add("itcast");// 该操作会导致程序出错
            }
        }
        //打印容器中的元素
        System.out.println(list);
    }
}
```

运行上述代码发生了错误 java.util.ConcurrentModificationException 这是什么原因呢？

Comment [L1]: 并发修改异常

在迭代过程中，使用了集合的方法对元素进行操作。导致迭代器并不知道集合中的变化，容易引发数据的不确定性。

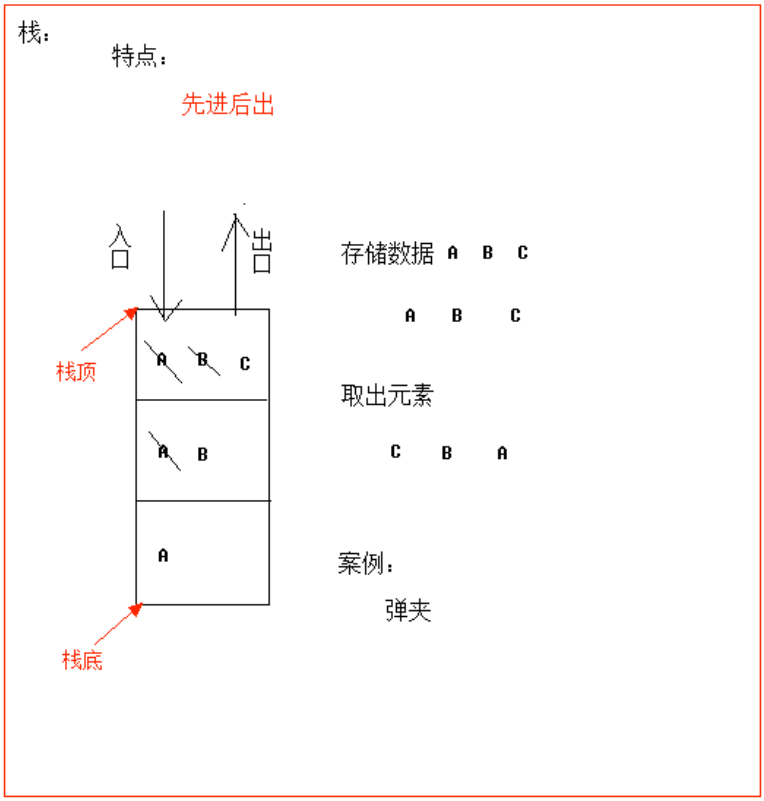
并发修改异常解决办法：在迭代时，不要使用集合的方法操作元素。

那么想要在迭代时对元素操作咋办？通过 ListIterator 迭代器操作元素是可以的，ListIterator 的出现，解决了使用 Iterator 迭代过程中可能会发生的错误情况。

1.3 List 集合存储数据的结构

List 接口下有很多个集合，它们存储元素所采用的结构方式是不同的，这样就导致了这些集合有它们各自的特点，供给我们在不同的环境下进行使用。数据存储的常用结构有：堆栈、队列、数组、链表。我们分别来了解一下：

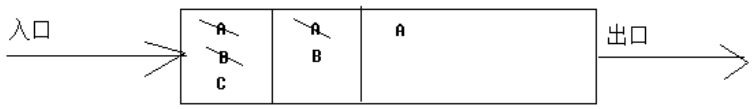
- 堆栈，采用该结构的集合，对元素的存取有如下的特点：
  - 先进后出（即，存进去的元素，要在后它后面的元素依次取出后，才能取出该元素）。例如，子弹压进弹夹，先压进去的子弹在下面，后压进去的子弹在上面，当开枪时，先弹出上面的子弹，然后才能弹出下面的子弹。
  - 栈的入口、出口的都是栈的顶端位置
  - 压栈：就是存元素。即，把元素存储到栈的顶端位置，栈中已有元素依次向栈底方向移动一个位置。
  - 弹栈：就是取元素。即，把栈的顶端位置元素取出，栈中已有元素依次向栈顶方向移动一个位置。



- 队列，采用该结构的集合，对元素的存取有如下的特点：
  - 先进先出（即，存进去的元素，要在后它前面的元素依次取出后，才能取出该元素）。例如，安检。排成一列，每个人依次检查，只有前面的人全部检查完毕后，才能排到当前的人进行检查。
  - 队列的入口、出口各占一侧。例如，下图中的左侧为入口，右侧为出口。

队列

特点： 先进先出



存储数据 A B C

A B C

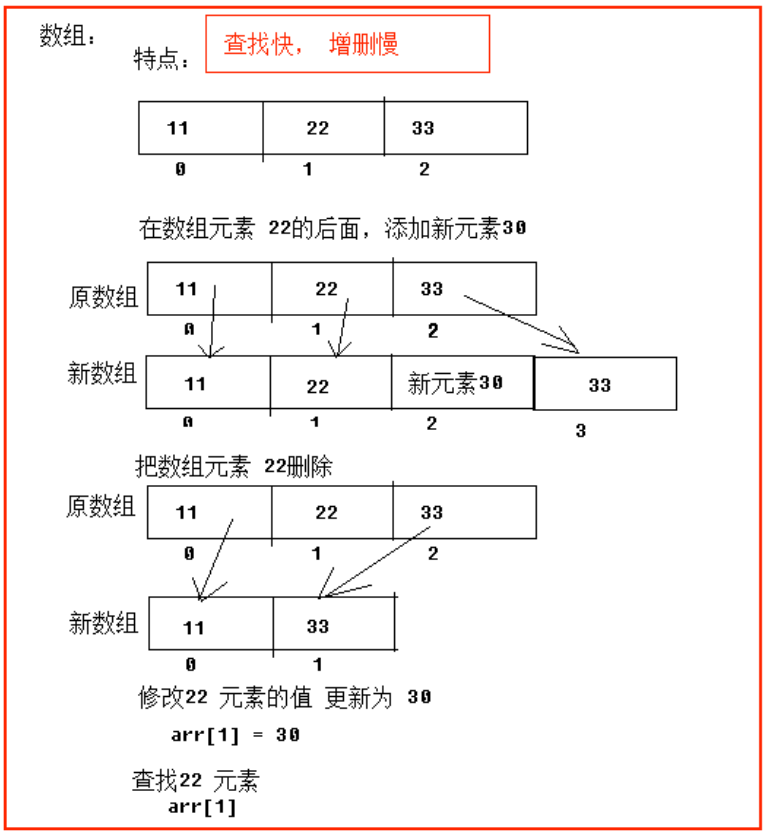
取出元素

A B C

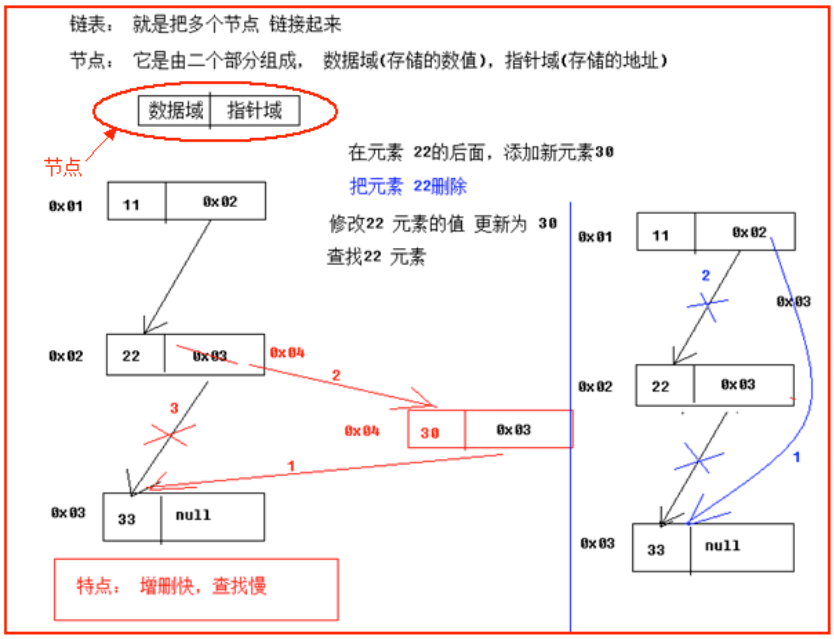
案例：

安检

- 数组，采用该结构的集合，对元素的存取有如下的特点：
  - 查找元素快：通过索引，可以快速访问指定位置的元素
  - 增删元素慢：
    - **指定索引位置增加元素**：需要创建一个新数组，将指定新元素存储在指定索引位置，再把原数组元素根据索引，复制到新数组对应索引的位置。如下图
    - **指定索引位置删除元素**：需要创建一个新数组，把原数组元素根据索引，复制到新数组对应索引的位置，原数组中指定索引位置元素不复制到新数组中。如下图



- 链表，采用该结构的集合，对元素的存取有如下的特点：
  - 多个节点之间，通过地址进行连接。例如，多个人手拉手，每个人使用自己的右手拉住下个人的左手，依次类推，这样多个人就连在一起了。
  - 查找元素慢：想查找某个元素，需要通过连接的节点，依次向后查找指定元素
  - 增删元素快：
    - 增加元素：操作如左图，只需要修改连接下个元素的地址即可。
    - 删除元素：操作如右图，只需要修改连接下个元素的地址即可。



1.4ArrayList 集合

ArrayList 集合数据存储的结构是数组结构。元素增删慢，查找快，由于日常开发中使用最多的功能为查询数据、遍历数据，所以 ArrayList 是最常用的集合。

许多程序员开发时非常随意地使用 ArrayList 完成任何需求，并不严谨，这种用法是不提倡的。

1.5LinkedList 集合

LinkedList 集合数据存储的结构是链表结构。方便元素添加、删除的集合。实际开发中对一个集合元素的添加与删除经常涉及到首尾操作，而 LinkedList 提供了大量首尾操作的方法。如下图

void	**addFirst**(E e) 将指定元素插入此列表的开头。
void	**addLast**(E e) 将指定元素添加到此列表的结尾。
E	**getFirst**() 返回此列表的第一个元素。
E	**getLast**() 返回此列表的最后一个元素。
E	**removeFirst**() 移除并返回此列表的第一个元素。
E	**removeLast**() 移除并返回此列表的最后一个元素。
E	**pop**() 从此列表所表示的堆栈处弹出一个元素。
void	**push**(E e) 将元素推入此列表所表示的堆栈。
boolean	**isEmpty**() 如果列表不包含元素，则返回 true。

LinkedList 是 List 的子类，List 中的方法 LinkedList 都是可以使用，这里就不做详细介绍，我们只需要了解 LinkedList 的特有方法即可。在开发时，LinkedList 集合也可以作为堆栈，队列的结构使用。

方法演示：

```
        LinkedList<String> link = new LinkedList<String>();
        //添加元素
        link.addFirst("abc1");
        link.addFirst("abc2");
        link.addFirst("abc3");
        //获取元素
        System.out.println(link.getFirst());
        System.out.println(link.getLast());
        //删除元素
        System.out.println(link.removeFirst());
        System.out.println(link.removeLast());

        while(!link.isEmpty()){ //判断集合是否为空
            System.out.println(link.pop()); //弹出集合中的栈顶元素
    }
```

1.6Vector 集合

Vector 集合数据存储的结构是数组结构，为 JDK 中最早提供的集合。Vector 中提供了一个独特

的取出方式，就是枚举 Enumeration，它其实就是早期的迭代器。此接口 Enumeration 的功能与 Iterator 接口的功能是类似的。Vector 集合已被 ArrayList 替代。枚举 Enumeration 已被迭代器 Iterator 替代。

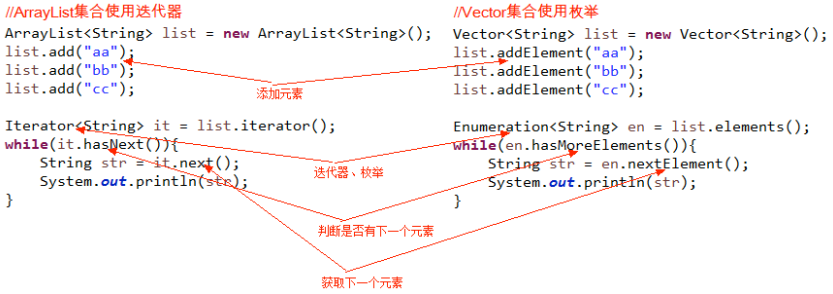
- Vector 常见的方法：

void	addElement(E obj) 将指定的组件添加到此向量的末尾，将其大小增加 1。
E	elementAt(int index) 返回指定索引处的组件。
Enumeration<E>	elements() 返回此向量的组件的枚举。

- Enumeration 枚举常见的方法：

boolean	hasMoreElements() 测试此枚举是否包含更多的元素。
E	nextElement() 如果此枚举对象至少还有一个可提供的元素，则返回此枚举的下一个元素。

- Vector 集合对 ArrayList 集合使用的对比



第2章 Set 接口

学习 Collection 接口时，记得 Collection 中可以存放重复元素，也可以不存放重复元素，那么我们知道 List 中是可以存放重复元素的。那么不重复元素给哪里存放呢？那就是 Set 接口，它里面的集合，所存储的元素就是不重复的。

2.1 Set 接口介绍

查阅 Set 集合的 API 介绍，通过元素的 equals 方法，来判断是否为重复元素，

2.2 HashSet 集合介绍

查阅 HashSet 集合的 API 介绍：此类实现 Set 接口，由哈希表支持（实际上是一个 HashMap 集

合）。HashSet 集合不能保证的迭代顺序与元素存储顺序相同。

HashSet 集合，采用哈希表结构存储数据，保证元素唯一性的方式依赖于：hashCode()与 equals()方法。

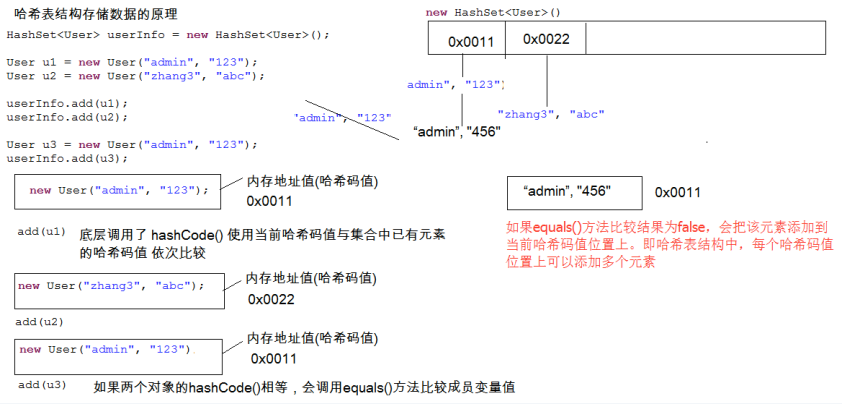
2.3HashSet 集合存储数据的结构（哈希表）

什么是哈希表呢？

哈希表底层使用的也是数组机制，数组中也存放对象，而这些对象往数组中存放时的位置比较特殊，当需要把这些对象给数组中存放时，那么会根据这些对象的特有数据结合相应的算法，计算出这个对象在数组中的位置，然后把这个对象存放在数组中。而这样的数组就称为哈希数组，即就是哈希表。

当向哈希表中存放元素时，需要根据元素的特有数据结合相应的算法，这个算法其实就是 Object 类中的 hashCode 方法。由于任何对象都是 Object 类的子类，所以任何对象有拥有这个方法。即就是在给哈希表中存放对象时，会调用对象的 hashCode 方法，算出对象在表中的存放位置，这里需要注意，如果两个对象 hashCode 方法算出结果一样，这样现象称为哈希冲突，这时会调用对象的 equals 方法，比较这两个对象是不是同一个对象，如果 equals 方法返回的是 true，那么就不会把第二个对象存放在哈希表中，如果返回的是 false，就会把这个值存放在哈希表中。

总结：保证 HashSet 集合元素的唯一，其实就是根据对象的 hashCode 和 equals 方法来决定的。如果我们往集合中存放自定义的对象，那么保证其唯一，就必须复写 hashCode 和 equals 方法建立属于当前对象的比较方式。



2.4HashSet 存储 JavaAPI 中的类型元素

给 HashSet 中存储 JavaAPI 中提供的类型元素时，不需要重写元素的 hashCode 和 equals 方法，因为这两个方法，在 JavaAPI 的每个类中已经重写完毕，如 String 类、Integer 类等。

- 创建 HashSet 集合，存储 String 对象。

```
public class HashSetDemo {
    public static void main(String[] args) {
        //创建 HashSet 对象
```

```
        HashSet<String> hs = new HashSet<String>();
        //给集合中添加自定义对象
        hs.add("zhangsan");
        hs.add("lisi");
        hs.add("wangwu");
        hs.add("zhangsan");
        //取出集合中的每个元素
        Iterator<String> it = hs.iterator();
        while(it.hasNext()){
            String s = it.next();
            System.out.println(s);
        }
    }
}
```

输出结果如下，说明集合中不能存储重复元素：

```
wangwu
lisi
zhangsan
```

2.5HashSet 存储自定义类型元素

给 HashSet 中存放自定义类型元素时，需要重写对象中的 hashCode 和 equals 方法，建立自己的比较方式，才能保证 HashSet 集合中的对象唯一

- 创建自定义对象 Student

```
public class Student {
    private String name;
    private int age;
    public Student(String name, int age) {
        super();
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

```
    @Override
    public String toString() {
        return "Student [name=" + name + ", age=" + age + "]";
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if(!(obj instanceof Student)){
            System.out.println("类型错误");
            return false;
        }
        Student other = (Student) obj;
        return this.age ==  other.age && this.name.equals(other.name);
    }
}
```

- 创建 HashSet 集合，存储 Student 对象。

```
public class HashSetDemo {
    public static void main(String[] args) {
        //创建 HashSet 对象
        HashSet hs = new HashSet();
        //给集合中添加自定义对象
        hs.add(new Student("zhangsan",21));
        hs.add(new Student("lisi",22));
        hs.add(new Student("wangwu",23));
        hs.add(new Student("zhangsan",21));
        //取出集合中的每个元素
        Iterator it = hs.iterator();
        while(it.hasNext()){
            Student s = (Student)it.next();
            System.out.println(s);
        }
    }
}
```

输出结果如下，说明集合中不能存储重复元素：

Student [name=lisi, age=22]
Student [name=zhangsan, age=21]
Student [name=wangwu, age=23]

2.6 LinkedHashSet 介绍

我们知道 HashSet 保证元素唯一，可是元素存放进去是没有顺序的，那么我们要保证有序，怎么办呢？

在 HashSet 下面有一个子类 LinkedHashSet，它是链表和哈希表组合的一个数据存储结构。

演示代码如下：

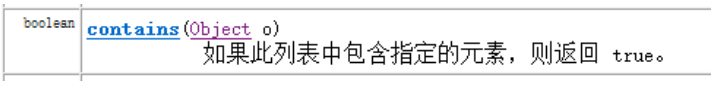
```
public class LinkedHashSetDemo {
    public static void main(String[] args) {
        Set<String> set = new LinkedHashSet<String>();
        set.add("bbb");
        set.add("aaa");
        set.add("abc");
        set.add("bbc");
        Iterator it = set.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

输出结果如下，LinkedHashSet 集合保证元素的存入和取出的顺序：

bbb
aaa
abc
bbc

第3章 判断集合元素唯一的原理

3.1 ArrayList 的 contains 方法判断元素是否重复原理



ArrayList 的 contains 方法会使用调用方法时，传入的元素的 equals 方法依次与集合中的旧元素所比较，从而根据返回的布尔值判断是否有重复元素。此时，当 ArrayList 存放自定义类型时，由于自定义类型在未重写 equals 方法前，判断是否重复的依据是地址值，所以如果想根据内容判断是否为重复元素，需要重写元素的 equals 方法。

3.2 HashSet 的 add/contains 等方法判断元素是否重复原理

boolean	**add**(E e) 如果此 set 中尚未包含指定元素，则添加指定元素。
boolean	**contains**(Object o) 如果此 set 包含指定元素，则返回 true。

Set 集合不能存放重复元素，其添加方法在添加时会判断是否有重复元素，有重复不添加，没重复则添加。

HashSet 集合由于是无序的，其判断唯一的依据是元素类型的 hashCode 与 equals 方法的返回结果。规则如下：

先判断新元素与集合内已经有的旧元素的 HashCode 值

- 如果不同，说明是不同元素，添加到集合。
- 如果相同，再判断 equals 比较结果。返回 true 则相同元素；返回 false 则不同元素，添加到集合。

所以，使用 HashSet 存储自定义类型，如果没有重写该类的 hashCode 与 equals 方法，则判断重复时，使用的是地址值，如果想通过内容比较元素是否相同，需要重写该元素类的 hashcode 与 equals 方法。

第4章 总结

4.1 知识点总结

- List 与 Set 集合的区别？
  - List:
    - 它是一个有序的集合(元素存与取的顺序相同)
    - 它可以存储重复的元素
  - Set:
    - 它是一个无序的集合(元素存与取的顺序可能不同)
    - 它不能存储重复的元素
- List 集合中的特有方法
  - void add(int index, Object element) 将指定的元素，添加到该集合中的指定位置上
  - Object get(int index)返回集合中指定位置的元素。
  - Object remove(int index) 移除列表中指定位置的元素, 返回的是被移除的元素
  - Object set(int index, Object element)用指定元素替换集合中指定位置的元素,返回值的更新前的元素
- ArrayList:
  底层数据结构是数组，查询快，增删慢
- LinkedList:
  底层数据结构是链表，查询慢，增删快

- HashSet:
  元素唯一，不能重复
  底层结构是 哈希表结构
  元素的存与取的顺序不能保证一致
  如何保证元素的唯一的？
  重写 hashCode() 与 equals()方法
- LinkedHashSet:
  元素唯一不能重复
  底层结构是 哈希表结构 + 链表结构
  元素的存与取的顺序一致