[ArrayList 1](#_Toc528758668)

[LinkedList 4](#_Toc528758669)

[LinkedList和ArrayList对比 5](#_Toc528758670)

[Set 6](#_Toc528758671)

[TreeSet 7](#_Toc528758672)

[LinkedHashSet 7](#_Toc528758673)

[Map接口 8](#_Toc528758674)

[散列算法介绍 8](#_Toc528758675)

[哈希函数的冲突现象 9](#_Toc528758676)

[Equals和HashCode(曾经是面试题) 10](#_Toc528758677)

[HashMap 10](#_Toc528758678)

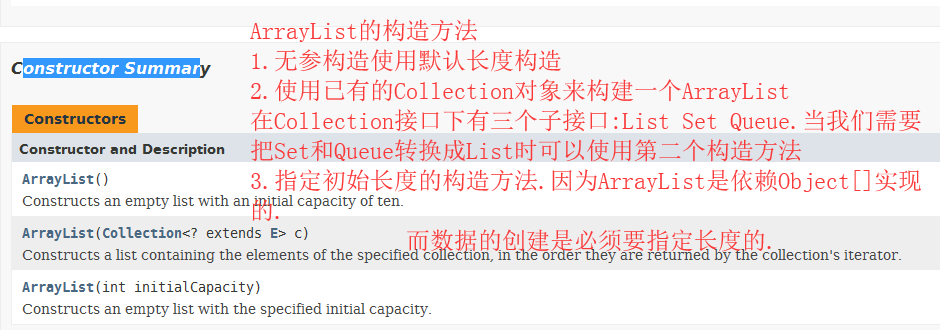
[作业讲解 10](#_Toc528758679)

[课堂练习 13](#_Toc528758680)

[预习 13](#_Toc528758681)

# ArrayList

回顾下ArrayList中我们使用的过的那些API方法.



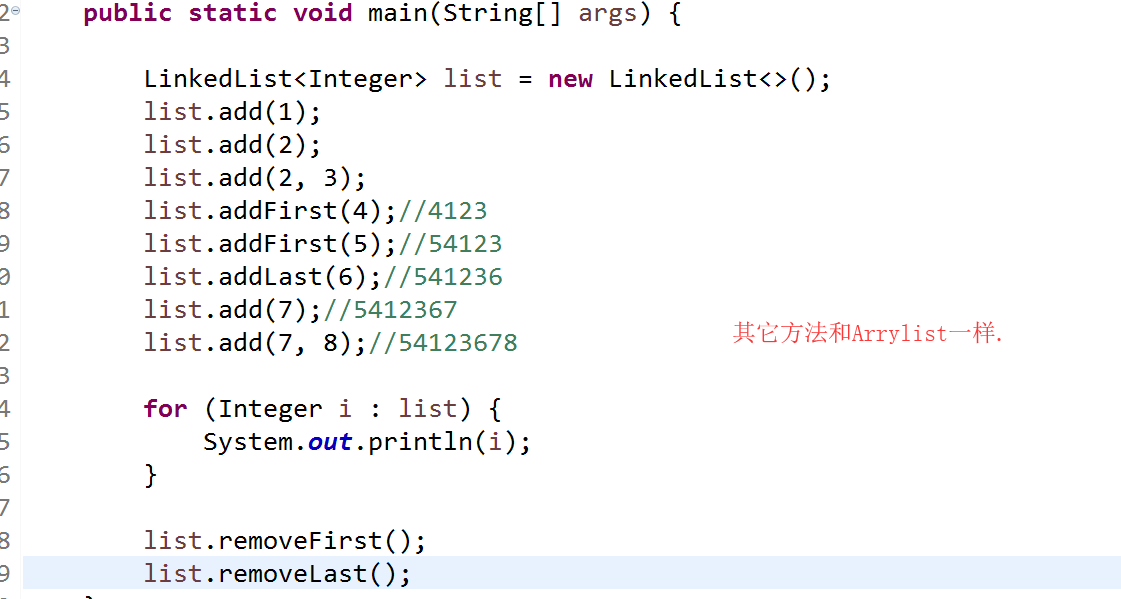
1. **package** com.hwua.demo01;
3. **import** java.util.ArrayList;
4. **import** java.util.Comparator;
5. **import** java.util.Iterator;
6. **import** java.util.List;
7. **import** java.util.ListIterator;
9. /\*\*
10. \* ArrayList中的常用API方法
11. \*
12. \* @author Administrator
13. \*
14. \*/
15. **public** **class** ListDemo {
17. **public** **static** **void** main(String[] args) {
19. // -- 1.构造方法
20. ArrayList<Integer> list1 = **new** ArrayList<>();
21. ArrayList<Integer> list2 = **new** ArrayList<>(10);
23. // -- 把List2中的元素赋值到List3中
24. ArrayList<Integer> list3 = **new** ArrayList<>(list2);
26. // -- 2.添加元素的方法 下面统一使用List1
28. // -- add(Object)
29. list1.add(1);
30. // -- add(index,Object)
31. list1.add(0, 2);
32. // -- 把List2中的所有元素添加到List1中
33. list1.addAll(list2);
34. // -- 把List3中的所有远在添加到List1中.从list1中下标为2的地方开始放元素
35. list1.addAll(2, list3);
37. // -- 3.把List中所有元素清空 实现方式就是对数组进行遍历.把所有位置都赋值null并把size置为0.
38. list1.clear();
40. // -- 4.判断集合中是否包含指定元素.返回布尔值.true包含.false不包含
41. // -- 实现方式: 求指定元素在集合中第一次出现的下标.然后和0做比较
42. list1.contains(0);
44. // -- 5.从集合中根据下标取对应的元素
45. // -- get(index). 需要注意的是取元素之前会先判断index和size的关系.如果index >= size 则会越界
46. list1.get(0);
48. // -- 6.求元素第一次出现下标
49. // -- 通过原码我们可以发现list中是可以保存null的.
50. list1.indexOf(0);
52. // -- 7.判断集合中有无元素.即集合是否是空的 返回布尔值.
53. list1.isEmpty();
55. // -- 8.获取迭代器.通过迭代器来对集合进行遍历.foreach里面就包含迭代器
56. Iterator<Integer> iterator = list1.iterator();
58. // -- 9.和上面一样也是获取迭代器
59. ListIterator<Integer> listIterator = list1.listIterator();
61. // -- 10.求元素最后一次出现的下标 如果不包含该元素返回-1
62. list1.lastIndexOf(0);
64. // -- 11.移除指定位置上的元素
65. // -- remove(index)
66. list1.remove(1);
68. // -- 12.移除指定的对象
69. // -- remove(Object). 注意如果泛型是Integer.当我们写1是会默认为是index.
70. // -- 把1保存成Integer类型 Integer i = 1; 把i放到小括号内.
71. list1.remove(1);
73. // -- 13.从list1中移除和list2中元素相同的元素.如果有重复的则全部移除.
74. list1.removeAll(list2);
76. // -- 14.替换指定位置上的元素
77. // -- set(index,object);
78. list1.set(1, 3);
80. // -- 15.获取集合中元素的个数
81. list1.size();
83. // -- 16.对集合进行排序 .需要通过匿名内部类的形式提供一个Comparator接口的对象
84. list1.sort(**null**);
86. // -- 17.从集合中截取一段内容形成子集合存在
87. List<Integer> subList = list1.subList(0, 3);
89. // -- 18.把集合转变成数组(Object数组)
90. Object[] array = list1.toArray();
92. // -- 19.把集合转变成指定类型的数组.需要给类型的数组先有一个
93. Integer[] array1 = **new** Integer[list1.size()];
94. array1 = list1.toArray(array1);
96. // -- 20.把一个数组中的所有元素装到集合中.
97. **for** (Integer i : array1) {
98. list1.add(i);
99. }
100. }
101. }

# LinkedList

ArrayList被叫做动态数组,因为它是基于数组形成的.LinkedList被叫做链表,因为它是基于链表形成的.

链表最明显的特点是:在内存中不连续.链表以节点保存数据.每个节点有两个区域:数据区和地址区.数据区存放要保存的数据.而地址区记录下一个节点和上一个节点的地址.

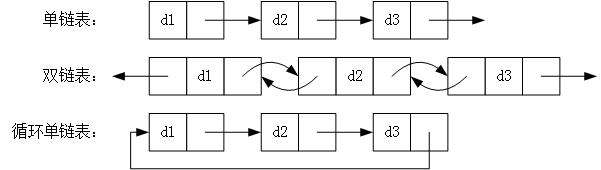
Java中的LinkedList针对头尾提供了专门的操作方法.其它的和ArrayList一样.

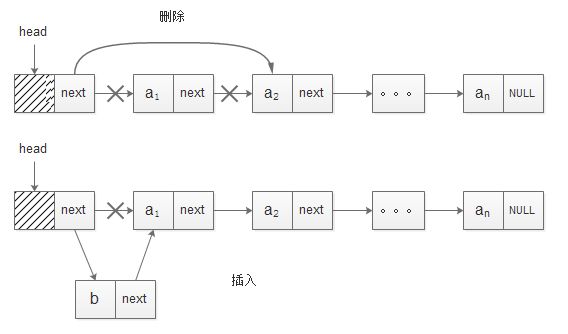


## LinkedList和ArrayList对比

ArrayList是基于数组的,所以它有个明显的特点和弊端.特点是空间连续,遍历的速度比较快.弊端就是如果要插入元素假设M个元素需要在N位置上插入元素,那M-N个元素需要整体向右平移一位.

LinkedList是基于链表的.本身在内存中空间就是不连续的.所以在遍历上比较慢,因为它会多一个寻址的过程.但是向LinkedList中插入元素的时候.比较便捷.只需要就近的两个节点.断开已有的链接,然后重新绑定.



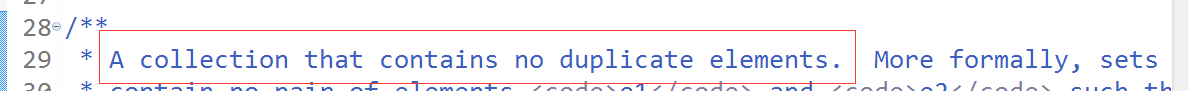


在Java中LinkedList和ArrayList都是List的实现类.所以基本功能是相同的.LinkedList比ArrayList多的功能就是在头尾的操作上.

# Set

Set是Collection接口的一个子接口.在Java中接口只可以继承接口.但是不可以实现接口.并且Java的接口是允许多继承的.

Set同List一样都是线性结构.但是Set和List有一个显著的区别.在Set接口的注释中是怎么说的

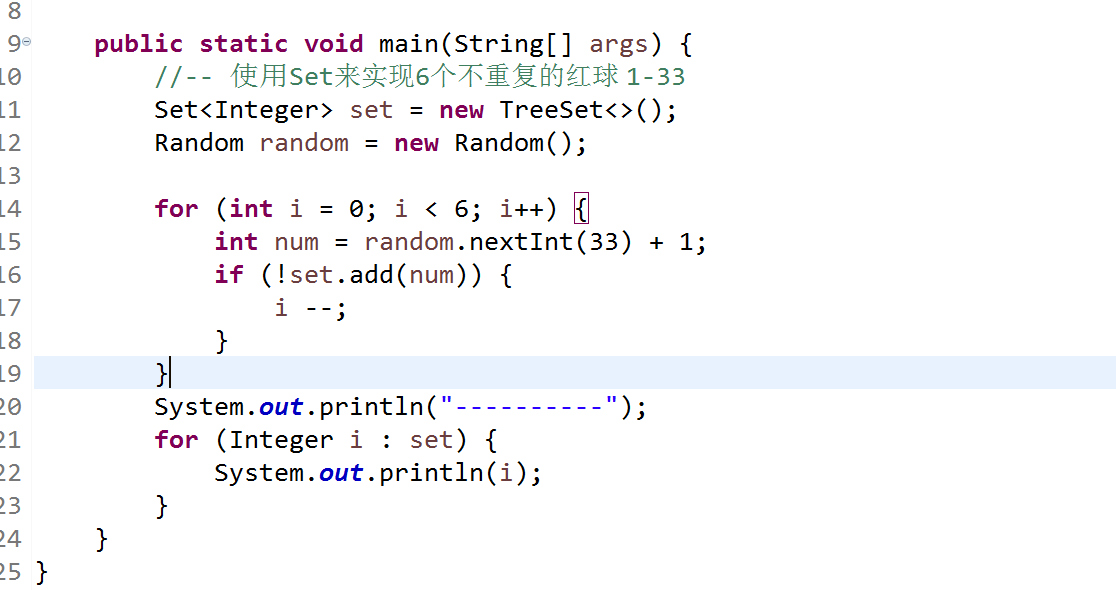


即Set中是没有重复元素的.

Set是一个接口.接口不可以直接实例化.我们可以使用的是它的两个实现类:HashSet和TreeSet

## TreeSet

TreeSet拥有Set的唯一性质.同时还拥有排序的特点.我们可以使用TreeSet来重新生成双色球的6个红球



## LinkedHashSet

是JDK1.4的时候推出的新集合同时具备了LinkedList的特点和Hash算法特点的Set集合.

使用方式和普通Set没有区别.

# Map接口

是Java集合中的一大块内容.并且Map所设计的思想不仅仅用于Java中.我们在很多语言.或者很多地方都可以遇到键值对.(key-value)

## 散列算法介绍

一般的[线性表](https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E8%A1%A8/3228081)，树中，记录在结构中的相对位置是[随机](https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA/4878685)的，即和记录的关键字之间不存在确定的关系，因此，在结构中查找记录时需进行一系列和关键字的比较。这一类查找方法建立在“比较“的基础上，查找的效率依赖于查找过程中所进行的比较次数。 理想的情况是能直接找到需要的记录，因此必须在记录的存储位置和它的关键字之间建立一个确定的对应关系f，使每个关键字和结构中一个唯一的存储位置相对应。

[散列表](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E8%A1%A8/10027933)（Hash table，也叫哈希表），是根据关键码值(Key value)而直接进行访问的[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/1450)。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做[散列函数](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0/2366288)，存放记录的[数组](https://baike.baidu.com/item/%E6%95%B0%E7%BB%84/3794097)叫做[散列表](https://baike.baidu.com/item/%E6%95%A3%E5%88%97%E8%A1%A8/10027933)。

给定表M，存在函数f(key)，对任意给定的关键字key，代入函数后若能得到包含该关键字的记录在表中的地址，则称表M为哈希(Hash）表，函数f(key)为哈希(Hash) 函数。

## 哈希函数的冲突现象

问?有没有可能不同的key,经过同一个算法计算后得到的地址是一样的.

对不同的关键字可能得到同一散列地址，即k1≠k2，而f(k1)=f(k2)，这种现象称为冲突（英语：Collision）。具有相同函数值的关键字对该散列函数来说称做同义词。综上所述，根据散列函数f(k)和处理冲突的方法将一组关键字映射到一个有限的连续的地址集（区间）上，并以关键字在地址集中的“像”作为记录在表中的存储位置，这种表便称为散列表，这一映射过程称为散列造表或散列，所得的存储位置称散列地址。

若对于关键字集合中的任一个关键字，经散列函数映象到地址集合中任何一个地址的概率是相等的，则称此类散列函数为均匀散列函数（Uniform Hash function），这就是使关键字经过散列函数得到一个“随机的地址”，从而减少冲突。

## Equals和HashCode(曾经是面试题)

在Object中有两个方法Equals和HashCode方法.这两个方法的作用是用于判定两个对象是否相等的.

两个方法没必要都执行.因为它们的方向不一样.equals用于判断对象的字段值是否一样.而hashCode用于判断,对象在Hash表中的地址是否一样.

如果我们需要比较对象是否相等.可以先判断HashCode.当HasCode一样时,再考虑判断equals是否一样.

## HashMap

是无序的.这个序指输入和输出的顺序不一致.会按照Key的HashCode.在Hash表中寻找位置位置爆出数据.

HashMap本身是无法进行排序的.所有都是利用TreeMap来进行排序.

## TreeMap

是基于Red-Black-Tree(红黑树)的.该Map是根据其键进行自然序排序的,或者提供相应的Comparator对象来指定排序方式

# 作业讲解

1. 使用ArrayList来重写双色球
2. **package** com.hwua.demo01;
4. **import** java.util.ArrayList;
5. **import** java.util.Collections;
6. **import** java.util.Random;
8. /\*\*
9. \* ' 彩票类
10. \*
11. \* @author Administrator
12. \*
13. \*/
14. **public** **class** Lottery {
15. // -- 记录期号
16. **private** **int** stage;
18. **public** **int** getStage() {
19. **return** stage;
20. }
22. **public** **void** setStage(**int** stage) {
23. **this**.stage = stage;
24. }
26. /\*\*
27. \* 提供常量用来控制红球和蓝球的个数
28. \*/
29. **private** **static** **final** **int** MAX\_RED = 6;
31. // -- 注解.目的是忽略 unused的警告
32. @SuppressWarnings("unused")
33. **private** **static** **final** **int** MAX\_BLUE = 1;
35. **private** Random random;
36. /\*\*
37. \* '提供容器保存6个取值1-33的红球
38. \*/
39. **private** ArrayList<Integer> reds;
41. // -- 提供一个变量保存每一期的蓝球
42. **private** **int** blue;
44. /\*\*
45. \* ' 提供红球的get方法
46. \*
47. \* @return
48. \*/
49. **public** ArrayList<Integer> getReds() {
50. **return** reds;
51. }
53. **public** **int** getBlue() {
54. **return** blue;
55. }
57. /\*\*
58. \* '利用构造方法完成号码的初始化操作
59. \*/
60. **public** Lottery(**int** stage) {
61. **this**.stage = stage;
62. initLottery();
63. }
65. /\*\*
66. \* '初始化集合
67. \*/
68. **public** **void** initLottery() {
69. // -- 实例化保存红球的集合
70. reds = **new** ArrayList<>();
71. random = **new** Random();
72. createRed();
73. createBlue();
74. }
76. /\*\*
77. \* '生成红球
78. \*/
79. **public** **void** createRed() {
80. // -- 生成红球并放入集合
81. **for** (**int** i = 0; i < MAX\_RED; i++) {
82. // -- 取值范围1-33
83. Integer ball = random.nextInt(33) + 1;
84. **if** (reds.contains(ball)) {
85. i--;
86. **continue**;
87. }
88. reds.add(ball);
89. }
90. // -- 生成数字是随机的导致集合中元素的顺序是乱序的.排序
91. Collections.sort(reds);
92. }
94. /\*\*
95. \* ' 生成蓝球
96. \*/
97. **public** **void** createBlue() {
98. // -- 取值 1-16
99. blue = random.nextInt(16) + 1;
100. }
102. /\*\*
103. \* ' 格式话打印!
104. \*/
105. @Override
106. **public** String toString() {
107. String line = "第" + stage + "期彩票号码:\t";
108. **for** (**int** i = 0; i < reds.size(); i++) {
109. **if** (reds.get(i) < 10) {
110. line = line + "0" + reds.get(i) + " ";
111. } **else** {
112. line = line + reds.get(i) + " ";
113. }
114. }
115. line = line + blue;
116. **return** line;
117. }
118. }

# 课堂练习

1. 使用LinkedList来实现栈和队列.
2. 使用Map保存String-Person.并分别根据键和值进行排序.值是根据Person的Age进行排序.
3. **package** com.hwua.demo02;
5. **import** java.util.ArrayList;
6. **import** java.util.Collections;
7. **import** java.util.Comparator;
8. **import** java.util.HashMap;
9. **import** java.util.Iterator;
10. **import** java.util.Map;
11. **import** java.util.Map.Entry;
12. **import** java.util.TreeMap;
14. **public** **class** Test {
16. **public** **static** **void** main(String[] args) {
17. // --
18. HashMap<String, Person> map = **new** HashMap<>();
19. Person p1 = **new** Person(1, 17, "张三");
20. Person p2 = **new** Person(2, 16, "李四");
21. Person p3 = **new** Person(3, 15, "王五");
22. Person p4 = **new** Person(4, 18, "赵六");
23. Person p5 = **new** Person(5, 19, "孙七");
24. Person p6 = **new** Person(6, 20, "周八");
26. map.put("B", p1);
27. map.put("A", p2);
28. map.put("Z", p3);
29. map.put("G", p4);
30. map.put("H", p5);
31. map.put("C", p6);
33. // -- 按照键排序.使用TreeMap.
34. TreeMap<String, Person> tm = **new** TreeMap<>(**new** Comparator<String>() {
35. @Override
36. **public** **int** compare(String o1, String o2) {
37. // TODO Auto-generated method stub
38. **return** o1.compareTo(o2);
39. }
40. });
42. // -- 放好了就排好了
43. tm.putAll(map);
45. Iterator<String> iter = tm.keySet().iterator();
46. **while** (iter.hasNext()) {
47. String string = (String) iter.next();
48. System.out.println(string);
49. }
51. // -- 利用值排序 是利用Collections.sort(List,Comparator)
52. // -- 根据Map中的所有键值对构建一个list对象.
53. ArrayList<Map.Entry<String, Person>> list = **new** ArrayList<>(map.entrySet());
55. Collections.sort(list, **new** Comparator<Entry<String, Person>>() {
57. @Override
58. **public** **int** compare(Entry<String, Person> o1, Entry<String, Person> o2) {
59. // TODO Auto-generated method stub
60. **return** o1.getValue().getAge() - o2.getValue().getAge();
61. }
62. });
64. **for** (Entry<String, Person> entry : list) {
65. System.out.println(entry);
66. }
67. }
68. }
69. 生成50个30-35的随机数.使用Map来记录重复的个数.
70. **package** com.hwua.demo03;
72. **import** java.util.HashMap;
73. **import** java.util.Map.Entry;
74. **import** java.util.Random;
75. **import** java.util.Set;
76. **import** java.util.TreeMap;
78. **public** **class** Demo01 {
80. **public** **static** **void** main(String[] args) {
81. TreeMap<Integer, Integer> map = **new** TreeMap<>();
83. Random random = **new** Random();
85. **for** (**int** i = 0; i < 50; i++) {
86. Integer num = random.nextInt(6) + 30;
87. **if** (map.containsKey(num)) {
88. map.put(num, map.get(num) + 1);
89. } **else** {
90. map.put(num, 1);
91. }
92. }
93. // -- Map 的内部接口Entry.
94. Set<Entry<Integer, Integer>> entrySet = map.entrySet();
95. **for** (Entry<Integer, Integer> entry : entrySet) {
96. System.out.println(entry);
97. }
98. }
99. }

# 预习

1. 联系人