Java 集合框架与数据结构

Author:刘家宏

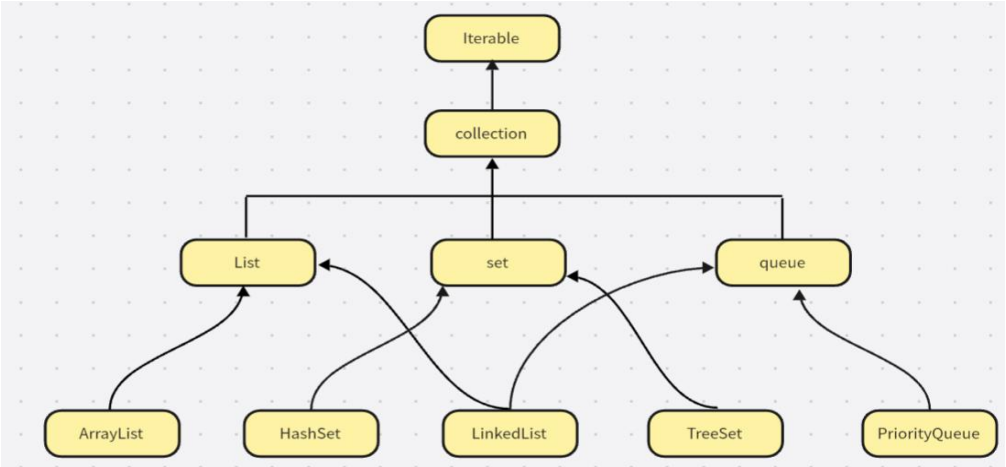
前言:本篇总结旨在为具备 Java 编程和数据结构只是的读者提供深入的洞察;我们将探讨 Java 集合框架中数据结构的实际运用,并分析各类集合的源码,揭示这些结构如何增强程序的性能和可维护性;通过分析各种集合类型,如:列表,集合,映射等,以及它们背后的数据结构,如:动态数组,链表,红黑树等,读者将能够更好地理解如何在实际编程中选择合适的数据结构,以及如何有效地利用 Java 集合框架来解决各种问题;

1. 集合框架

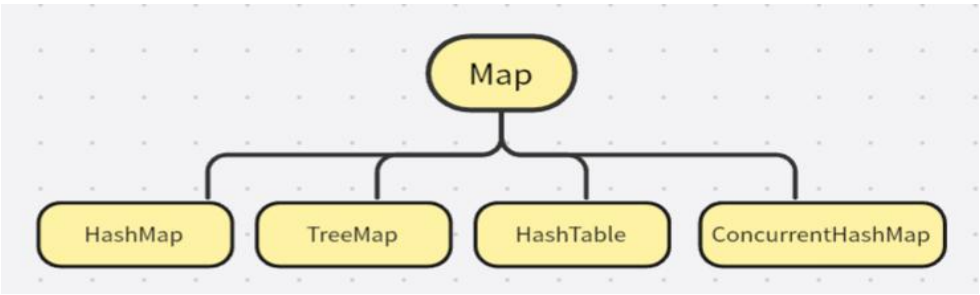
1.1. 集合框架介绍

(1)**集合**:集合也被称为容器,也是一个对象,此对象可以管理其它的对象元素:例如数组就是一个集合;
(2)**集合框架**:在 java 中有很多集合类,接口,算法类,几乎包含了所有的数据结构,作为一个整体,被称为集合框架;
(3)**集合框架中的两个顶层接口:**所有集合类都是这两个接口的实现类型:

- Collection:里面存储元素;
- Map:里面存储键值对(key,value);



Collection 接口结构图



map 接口结构图

1.2.Collection 接口

(1) 是集合框架的顶层接口之一,没有直接的实现类;
(2) Collection 有三个子接口,分别是:List,Set,Queue;
(3) **三个子接口的特点如下**:
List:有序(有索引,可以根据索引进行存取),可重复;
Set:无序,不可重复;
Queue:先进先出;
(4) Collection 接口应该有什么方法:
boolean add(E):增加一个元素
boolean addAll(Collection):增加多个元素
void clear():清空
boolean isEmpty():是否为空
int size():返回元素的个数
boolean remove(E):删除一个元素
boolean removeAll(Collection):删除多个元素
boolean retainAll(Collestion):保留多个元素
boolean contains(E):是否包含某个元素
boolean containsAll(Collection):是否包含某些元素
Object [] toArray():将集合中的元素转成数组返回

1.3Map 接口

(1) 是集合框架中用于存储键值对的接口,是集合框架的顶层接口之一,没有直接的实现类;
(2) Map 接口没有直接的实现类，但有许多实现类，如：HashMap、LinkedHashMap、TreeMap、Hashtable 等。
(3) **Map 接口的特点如下：**
存储键值对（key-value pairs），每个键映射到一个值。
键（Key）是唯一的，不允许重复。
值（Value）可以重复，且一个键可以映射到相同的值。
(4) **Map 接口应该有什么方法：**
V put(K key, V value)：将指定的值与此映射中的指定键关联。
V get(Object key)：返回指定键所映射的值。
V remove(Object key)：如果存在一个键的映射关系，则将其从映射中移除。

void putAll(Map<? extends K, ? extends V> m)：将指定映射的所有映射关系复制到此映射中。
void clear()：移除映射中的所有键值对。
boolean isEmpty()：如果映射不包含键值对，则返回 true。
int size()：返回映射中的键值对数。
boolean containsKey(Object key)：如果映射包含指定键的映射，则返回 true。
boolean containsValue(Object value)：如果映射包含指定值的映射关系，则返回 true。
Set<K> keySet()：返回映射中包含的键的 Set 视图。
Collection<V> values()：返回映射中包含的值的 Collection 视图。
Set<Map.Entry<K, V>> entrySet()：返回映射中包含的键值对的 Set 视图。
Object clone()：返回映射的浅拷贝。

(5)**遍历**

Set keyset():得到所有 key 的集合;
Collection values():获得所有 value 的集合;
entrySet():获得所有的元素的集合(Entry(key,value));

代码示例:

```
public class demo {
public static void main(String[]args) {
Map<Integer,String> map=new HashMap();
map.put(1,"张辽");
map.put(2,"乐进");
map.put(3,"钟会");
map.put(4,"曹真");
//1.Set.keySet()
Set<Integer> set=map.keySet();
for(Integer a:set) {
map.get(a);
System.out.println(map.get(a));
}
System.out.println("_____________");
//2.Collection.values()
Collection <String>coll=map.values();
for(String c:coll) {
System.out.println(c);
}
System.out.println("_____________");
//3.entrySet()
Set<Entry<Integer,String>>s=map.entrySet();
for(Entry<Integer,String> c:s) {
System.out.println(c);
}
}
}
```

运行结果:

```
张辽
乐进
钟会
曹真
_____________
张辽
乐进
钟会
曹真
_____________
1=张辽
2=乐进
3=钟会
4=曹真
```

1.4 算法类

(1)集合框架中除了提供的接口和实现类之外,还提供用于处理集合的算法类,算法类主要有:
Collections:用于处理集合的算法类;
Arrays:用于处理数组的算法类;
(2)**Collection 常用方法:**
static void sort(List):对 List 中的元素按自然顺序进行排序;
static void reverse(List):翻转集合中的元素;
static void shuffle(List):洗牌:(打乱顺序);
static E max(Collection):获取最大的元素;
static E min(Collection):获取最小的元素;
static void addAll(Collection coll,T...t):向 coll 中增加多个元素;
static int binarySearch(List):二分查找;
static void copy(List desc,List src):复制;
(3)**Arrays 常用方法:**
static List asList(T...t):创建一个 List,将添加多个元素;
static T binarySearch():二分查找
static T[] copyRange(int[],start,end);
static void sort();
static String toString();

1.5Comparable 和 Comparator

Comparable 接口:
(1)Comparable 是自然排序的接口,实现此接口的类,就拥有了比较大小的能力;
(2)**此方法中的方法只有一个:**
public int compararеTo(T t){
}
(3)**返回值说明:**
正数:当前对象比 t 大;
0:当前对象与 t 相等;
负数:当前对象比 t 小;
Comparator 接口:
(1) 自定义外部比较器;
(2)**适用场景:**

- 为同一个类定义多种不同的排序规则
- 为现有的类(例如 String,Date 等)增加一种排序规则

(3)**此接口中只包含一个方法**:

Int compare(T+1,T+2)

如果 t1>t2,则返回正数;
如果 t1=t2,返回 0;
如果 t1<t2,返回负数;

Comparable 与 Comparator 的区别:

- Comparable 为可排序的,也被称为**自然排序**或**内部比较器**,实现该接口的类的对象拥有可　排序功能;
- Comparator 为比较器,也被称为**外部比较器**,实现该接口可以定义一个针对某个类的排序方式;
- Comparator 与 Comparable 同时存在的情况下,前者的优先级高;

1.6List 接口

(1) List 代表一个元素有序,且可重复的集合,集合中的每个元素都有其对应的顺序索引;
List 允许使用可重复元素,可以通过索引来访问指定位置的集合元素;
List 默认按元素的添加顺序设置元素的索引;
List 集合里添加了一些根据索引来操作集合元素的方法;

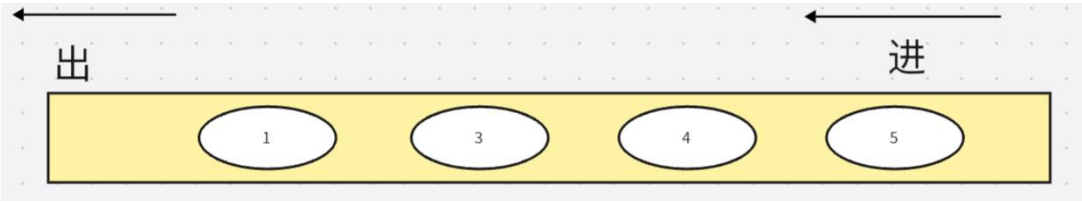
(2) List 常用方法:
void add(int index,E element):在列表的指定位置插入指定元素;
Boolean addAll(int index,Collection):在列表的指定位置插入批量元素;
E get(int index):获取列表中指定位置的元素;
Int indexOf(Object o):返回此列表中第一次出现指定元素的索引;若没有则返回-1;
E remove(int index):删除列表中指定位置的元素;
E set(int index, E element):用指定元素替换列表中指定位置的元素;
List<E>subList(int fromIndex,int toIndex):返回[fromIndex,toIndex)之间的部分视图;

1.7Queue 接口(队列)

(1) Queue 是一个队列,它的特点是先进先出(FIFO);
(2) 除了基本的集合操作之外,队列还提供额外的插入,提取和检查操作;这些方法都以两种形式存在:
一个方法在操作失败时抛出异常,
另一个方法返回一个特殊值(null 或 false,取决于操作)

	抛出异常	返回特殊值
插入	add (e)	offer(e)
删除	remove()	poll()
检查	element()	peek()

(3)图解:



先进先出

1.8set 接口

(1) **特点**:

- **不包含重复元素**:Set 接口保证集合中不会包含重复的对象;
- **无序**:Set 接口不保证元素的顺序,每次迭代可能会得到不同的顺序;
- **唯一性**:Set 接口通常用于确保存储的唯一性

(2) **三个主要的实现类**:

HashSet:基于哈希表实现的 Set 接口,它不保证集合的迭代顺序;特别是它不保证该顺序就不变;

LinkedHashList:也是基于哈希表和链表实现的,维护了元素的插入顺序,即按照将元素插入到集合中的顺序遍历元素;

TreeSet:基于红黑树(一种自平衡的二叉查找树)实现的 Set 接口,可以按照自然顺序或自定义顺序对元素进行排序;

2. ArrayList 集合(动态数组)

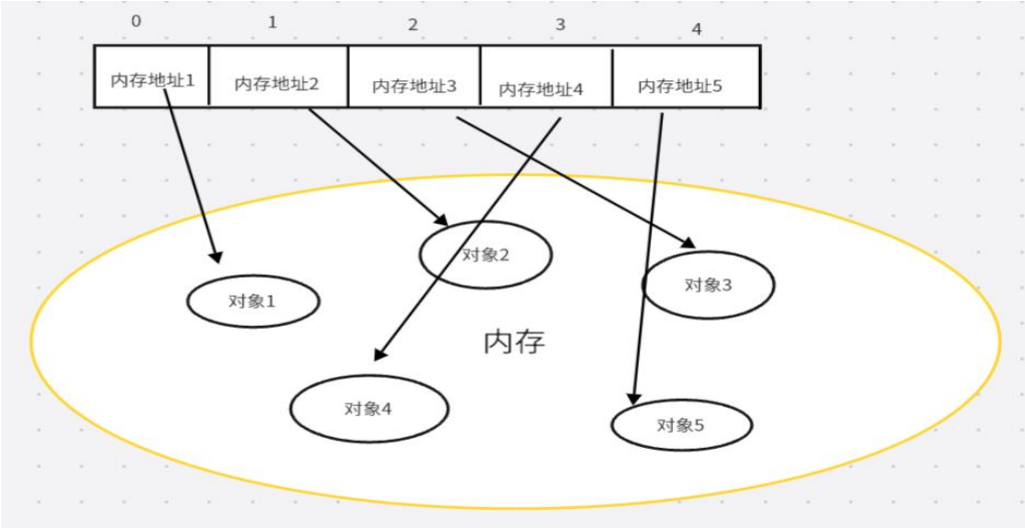
2.1ArraysList 集合

(1) **数据结构**:ArrayList 基于**动态数组**实现;它的核心是一个可变大小的数组,当数组容量不足以容纳更多元素时,ArrayList 会自动进行扩容;

(2) **特点**:

- **连续内存空间**:元素在内存中存储式连续的,这使得随机访问(通过索引访问元素)非常快;
- **动态扩容**:当元素超出当前容量时,ArrayList 会创建一个新的数组,通常时原来数组的 1.5 倍,然后将旧数组中的元素复制到新数组中;
- **随机访问性能高**:由于元素在内存中是连续存储的,所以 get(int index)操作的时间复杂度是 O(1);
- **插入和删除性能低**:在数组中间或开始位置插入或删除元素时,需要移动插入点之后的所有元素,这使得这些操作的时间复杂度时 O(n);

(3) **对象的存储方式**:ArrayList 并不直接存储对象本身,而是存储了指向这些对象**内存地址**的引用,而不是对象的副本;



2.2ArrayList 源码分析

(1)内部属性

```
/**
* The array buffer into which the elements of the ArrayList are stored.
* The capacity of the ArrayList is the length of this array buffer. Any
* empty ArrayList with elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA
* will be expanded to DEFAULT_CAPACITY when the first element is added.
*/
transient Object[] elementData; // non-private to simplify nested class access
```

定义一个 object 数组,用来存储数组的元素,此数组可以扩容;

(2)构造方法

```
/**
* Constructs an empty list with an initial capacity of ten.
*/
public ArrayList() {
this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
}

/**
* Constructs an empty list with the specified initial capacity.
*
* @param initialCapacity the initial capacity of the list
* @throws IllegalArgumentException if the specified initial capacity
* is negative
*/
public ArrayList(int initialCapacity) {
if (initialCapacity > 0) {
this.elementData = new Object[initialCapacity];
```

```
} else if (initialCapacity == 0) {
this.elementData = EMPTY_ELEMENTDATA;
} else {
throw new IllegalArgumentException("Illegal Capacity: "+
initialCapacity);
}
}
```

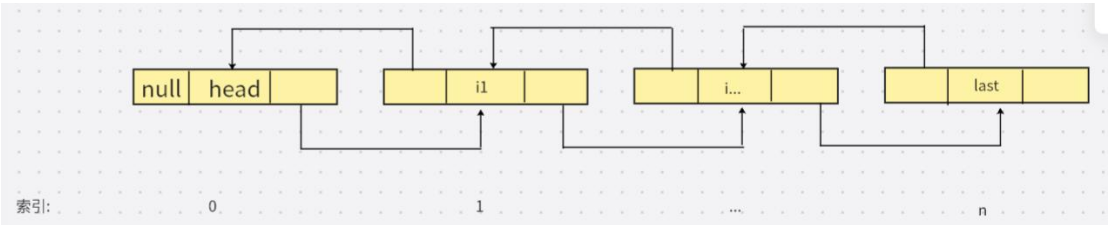
2.3 数组总结

数组的特点:元素之间的地址是连续的;
get(index):时间复杂度是 O(1);
set(index,E):O(1);
add(index,E):时间复杂度是 O(n);(在执行插入数据的过程中,指定位置后的元素将先后移动)
remove(index):O(n);

3. LinkedList 集合(双向队列,栈)

(1) **数据结构**:LinkedList 基于**双向链表**实现;每个元素由一个节点表示,节点包含数据和指向前一个节点与一个节点的引用;

节点结构



(2) **特点**:

- **非连续内存空间**:元素在内存中不是连续存储的,每个元素(节点)包含数据和指点前后元素的指针;
- **不需要扩容**:由于时基于链表,所以不需要像数组那样预留空间,可以动态地添加和删除元素;
- **随机访问性能低**:由于元素不是连续存储的,get(int index)操作需要**从头或尾开始遍历**链表,直到找到指定索引地元素,时间复杂度为 O(n);

- **插入和删除性能高**:在链表的任何位置插入或删除元素只需要改变节点之间的指针,这使得这些操作的时间复杂度是 O(1),前提是已经有了要插入或删除位置的引用;
- Linked 是非线程安全的;
- LinkedList 元素允许为 null,允许重复元素;
- **实现了栈和队列的操作方法:**因此也可以作为栈,队列和双端队列来使用;(从后加从前取为**队列**,从后加从后取为**栈**)

(3) **LinkedList 的构造函数**:

public LickedList():生成空的链表;

public LinkedList(Collection col):复制构造函数

4. Vector 和 Stack(栈)

(过期,目前不再使用)

4.1Vector

(1)ArrayList 和 Vector 是 List 接口的两个典型实现;

(2)**区别:**

Vector 是一个过期的集合类,通常建议使用 ArrayList

ArrayList 是线程不安全的,而 Vector 是线程安全的

即使为保证 List 集合线程安全,也**不推荐**使用 Vector

4.2Stack(栈)

(1)Stack 类表示后进先出(LIFO)的对象**堆栈**,它通过五个操作对类 Vector 进行扩展,允许将向量视为堆栈

(2)**常用方法:**

E push(E item):向栈顶压入一个元素;

E pop( );从栈顶获取一个元素,并将栈顶的元素移除;

E peel( );从栈顶获取一个元素,而不会移除栈顶的元素;

boolean empty:凭空栈中的元素;

Int search(Object o):查找某个元素在栈中的位置;

(3)**栈的用途**:

悔棋;

撤销;

方法的调用;(方法栈)方法的每次调用就是将该方法压到栈中,待调用完成后从栈中弹出;

5. Deque 接口(双端队列)

(1)为 Queue 的一个子接口,Deque 是一个具有**队列**和**栈**行为的数据结构,它允许从双端插入和移除

(1) **特点**:

两端操作:Deque 支持在两端进行 add,offer.poll,remove 和 peek 操作;

容量可变:与 Stack 不同,Deque 可以有一个容量限制,但不是固定大小的,如果尝试向已满的 Deque 添加元素,会抛出 IllegalStateExeption;

线程安全:Java 提供了线程安全的 Deque 实现,如 ArrayDeque 和 LinkedList;

(2) **构造方法**:

ArrayDeque():创建一个空的 ArrayDeque;

ArrayDeque(int num):创建一个具有初始容量的 ArrayDeque;

LinkedList():创建一个空的 LinkedList,可以作为 Deque 使用;

(3) **方法如下**:

addFirst(E e)：在队列的前端插入一个元素;

addLast(E e)：在队列的后端插入一个元素;

offerFirst(E e)：在队列的前端插入一个元素，如果 Deque 已满，则返回 false;

offerLast(E e)：在队列的后端插入一个元素，如果 Deque 已满，则返回 false;

pollFirst()：移除并返回队列前端的元素，如果队列为空，则返回 null;

pollLast()：移除并返回队列后端的元素，如果队列为空，则返回 null;

peekFirst()：返回队列前端的元素但不移除它，如果队列为空，则返回 null;

peekLast()：返回队列后端的元素但不移除它，如果队列为空，则返回 null;

removeFirst()：移除双端队列的第一个元素（前端）。如果双端队列为空，则抛出 NoSuchElementException;

6. PriorityQueue 集合(优先队列)

(1)PriorityQueue 是 Queue 接口的子接口(但是**不是先进先出**)

(2)**特点:**

- **元素排序**:PriorutyQueue 是具有**优先级别**的队列,优先级队列的元素按照它们的**自然顺序**排序,或者由队列构造时**提供的 Comparator** 进行排序,这取决于使用的是那个构造函数;
- **不允许 null 元素**:PriorityQueue 不允许插入 null 元素;
- **时间复杂度**:插入(add 或 offer)和删除(poll 或 remove)操作通常具有 O(log(n))的时间复杂度;

(3)**构造方法:**

PriorityQueue():创建一个空的 PriorityQueue,默认按自然顺序排序;

PriorityQueue(Collection):创建一个具有指定集合元素的 PriorityQueue,默认按自然顺序排序;

PriorityQueue(int initialCapacity):创建一个具有指定初始容量的空 PriorityQueue,默认按自然顺序排序;

PriorityQueue(Comparator comparator):创建一个空的 PriorityQueue,元素将根据提供的

Comparator 进行排序;
(4) **方法**:
add(E e):添加元素,如果队列已满,则抛出 IllcgelStateException;
offer(E e):添加元素,如果队列已满,则返回 false;
poll();移除并返回队列中优先级最高的元素(最小的元素),如果队列为空,则返回 null;
peek():返回队列中优先级最高的元素而不移除它,如果队列为空,则返回 null;
size():返回队列中的元素数量;

7. HashSet 集合

(1) HashSet 是 Set 接口的典型实现,大多数时候使用 Set 集合时都使用这个实现类;
(2) **特点**:

1.**不允许重复元素**：HashSet 会自动检查添加的元素是否已经存在，如果已存在，则不会添加重复的元素;

2.**无序性**：HashSet 中的元素**不保证**任何特定的顺序。每次遍历 HashSet 时，元素的顺序可能不同。

3.**基于哈希表**：HashSet 通常使用哈希表来存储元素，这使得它可以在平均情况下提供接近常数时间的性能，尤其是在添加、删除和查找元素时。

4.**快速查找**：由于基于哈希表，HashSet 在查找元素是否存在时非常高效，通常具有 O(1) 的时间复杂度。

5.**迭代器**：HashSet 提供迭代器，可以遍历集合中的所有元素。

6.**不支持索引访问**：与数组或列表不同，HashSet 不支持通过索引来访问元素。

7.**动态扩容**：当 HashSet 中的元素数量达到一定阈值时，哈希表会进行扩容，以保持操作的效率。

8.**线程不安全**：大多数语言中的 HashSet 实现不是线程安全的。如果需要在多线程环境中使用，需要额外的同步措施。

9.**元素可以为 null;**

10.当向 HashSet 集合中存入一个元素时,HashSet 会调用该对象的 **hashCode** 方法来得到该对象的 **hashCode** 值,然后根据 **hashCode** 值决定该对象在 HashSet 中存储位置;

(3)**Hash 是如何判断元素是否重复的?**

当调用 add(Obejct)方法的时候

1.首先会调用 Object 的 hashCode 方法判断 hashCode 是否已经存在,如不存在则直接插入元素;

2.如果已存在则调用 Object 对象的 equals 方法判断是否返回 true,如果为 true 则说 明元素已经存在,如为 false 则插入元素;

8. LinkedHashSet 集合

(1) LinkedHashSet 是 HashSet 的子类,对 HashSet 进行扩展,内部也是一个 hashSet
(2) 特点:

- LinkedHashSet 集合根据元素的 hashCode 值来决定元素的存储位置,但他同时使用链表维护元素的次序,这使得元素看起来是以插入顺序保存的;
- LinkedHashSet 插入性能略低于 HashSet,但在迭代访问 Set 里的全部元素时有很好的性能;
- LinkedHashSet 不允许集合元素重复;
- LinkedHashSet 按**存入的顺序**进行排列,遍历的时候也按照这个顺序;

9.TreeSet 集合(红黑树)

9.1SortedSet 接口

(1) 是 Set 接口的子接口,具备排序的能力;
(2) 方法
- comparator()：返回用于排序的 Comparator，如果使用自然排序则返回 null。
- first()：返回集合中的第一个（最小）元素。
- last()：返回集合中的最后一个（最大）元素。
- headSet(E toElement)：返回一个视图，包含小于 toElement 的所有元素。
- subSet(E fromElement, E toElement)：返回一个视图，在 fromElement 和 toElement 之间的所有元素。
- tailSet(E fromElement)：返回一个视图，包含大于或等于 fromElement 的所有元素

9.2TreeSet 集合

(1)**底层数据结构:**TreeSet 的底层实现是红黑树，这是一种自平衡的二叉搜索树。红黑树通过在每个节点上增加一个颜色属性（红色或黑色）来保证树的平衡，确保插入、删除和查找操作的时间复杂度为 O(log n)。
(2)**特点**:
- **元素唯一性**：TreeSet 不允许重复元素，因为它是基于 Set 接口实现的。
- **排序**：元素会按照自然顺序或者构造时提供的 Comparator 进行排序。
- **性能**：由于红黑树的特性，TreeSet 在插入、删除和查找操作上都能保持较好的性能。

(3)**构造方法:**
TreeSet 提供了多种构造方法：
- 无参数构造方法：创建一个空的 TreeSet，默认按照元素的自然顺序排序。
- 带 Comparator 的构造方法：创建一个空的 TreeSet，并接受一个 Comparator 对象来指定元素的排序规则。
- 带 Collection 的构造方法：创建一个包含指定集合中所有元素的 TreeSet，默认按照元素的自然顺序排序。
- 带 SortedSet 的构造方法：创建一个包含指定 SortedSet 中所有元素的 TreeSet，并使用相同的排序规则。

(4)**主要方法**
除了继承自 Set 接口的方法外，TreeSet 还提供了一些特定的方法：
first()：返回集合中的第一个（最小）元素。

last()：返回集合中的最后一个（最大）元素。
pollFirst()：移除并返回集合中的第一个（最小）元素。
pollLast()：移除并返回集合中的最后一个（最大）元素。
subSet(E fromElement, E toElement)：返回此集合中的一部分视图，包括 fromElement 和 toElement 之间的元素。
headSet(E toElement)：返回此集合中的一部分视图，包括小于 toElement 的所有元素。
tailSet(E fromElement)：返回此集合中的一部分视图，包括大于或等于 fromElement 的所有元素。

10.HashMap 集合(数组,链表,红黑树)

10.1HashMap 集合

(1) 在 java 中应用最广泛的哈希表的实现,它基于哈希表的基本原理,提供了快速的查找,插入和删除操作;
(2) **特点**:

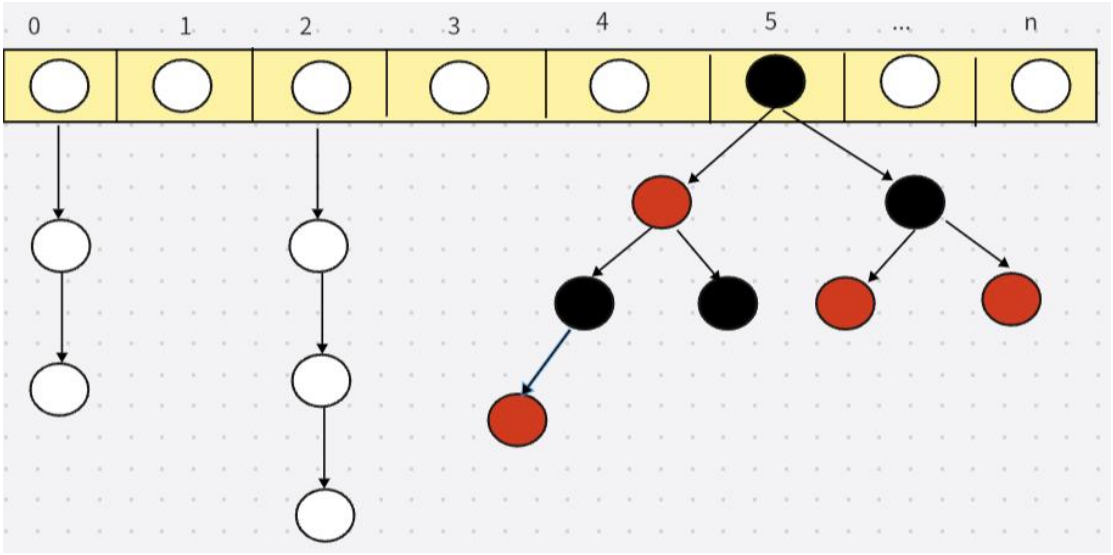
- **非同步**:HashMap 不是线程安全的,即在多线程环境下,如果没有采取额外的同步措施,可能会导致不可预知的行为;
- **允许空键和空值**:HsahMap 允许键或值为 null;
- **不保证顺序**:HashMap 不保证元素的顺序;

(3)**哈希表**

- **数组作为存储单元**:数组提供连续的内存空间,可以通过索引快速访问元素;
- **哈希函数**:HashMap 使用键对象的 hashCode() 方法计算哈希值，然后通过哈希值找到数组中的位置;
- **处理冲突的方法**:
  1.开放寻址法:当冲突发生时,寻找下一个空的数组位置来存储数据,常见的探测方法有线性探测,二次探测和双重散列;
  2.链地址法:在每个数组索引位置维护一个链表,所有映射到该索引的键值对都存储在这个表中;（在 Java 8 及以后版本中，当链表长度超过一定阈值时，**链表**会转换为**红黑树**）
- **动态扩容**:随着哈希表中的数据的增加,冲突的可能性也会增加,会影响性能;为了保持操作的效率,哈希表可能会动态地增加其大小(扩容),并重新计算现有元素的位置;
- **时间复杂度**:一般为 O(1),若所有元素都映射到同一位置则退化为 O(n),其中 n 为元素的数量;
- **负载因子**:负载因子是哈希表中已使用的槽位与总槽位的比例;负载因子越高，冲突的可能性越大;

(4)**HashMap 的内部结构**

- Key-value 在内部被封装成一个内部类 Node;
- HashMap 中定义了一个 Node 类型的数组用来保存元素
- Node 还是一个单向链表:当链表中的元素超过 8 时,则转换成红黑树;


0
1
2
3
4
5
...
n