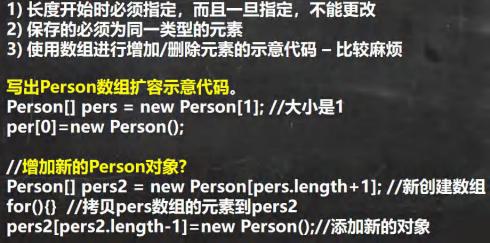
第 14 章集合

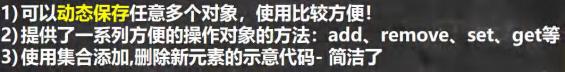
14. 1 集合的理解和好处

前面我们保存多个数据使用的是数组，那么数组有不足的地方，我们分析一下

14.1.1 数组



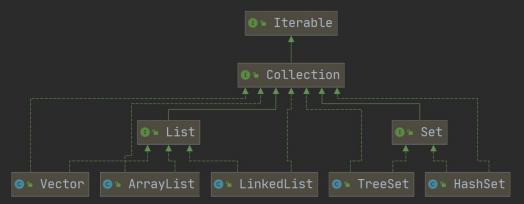
14.1.2 集合

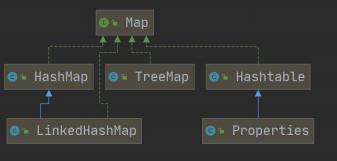


14.2 集合的框架体系

Java 的集合类很多，主要分为两大类，如图 ：[背下来]

Collection\_.java



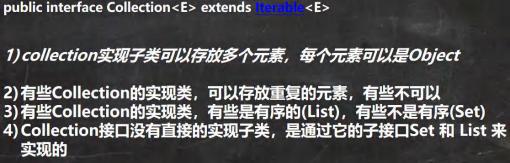


|  |
| --- |
| package com.hspedu.collection\_;  import java.util.ArrayList;  import java.util.Collection;  import java.util.HashMap;  import java.util.Map;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Collection\_ {  @SuppressWarnings({"all"})  public static void main(String[] args) { |

|  |
| --- |
| //老韩解读  //1. 集合主要是两组(单列集合 , 双列集合)  //2. Collection 接口有两个重要的子接口 List Set , 他们的实现子类都是单列集合  //3. Map 接口的实现子类 是双列集合，存放的 K-V  //4. 把老师梳理的两张图记住  //Collection  //Map  ArrayList arrayList = new ArrayList();  arrayList.add("jack");  arrayList.add("tom");  HashMap hashMap = new HashMap();  hashMap.put("NO1", "北京");  hashMap.put("NO2", "上海");  }  } |

14.3 Collection 接口和常用方法

14.3.1 Collection 接口实现类的特点



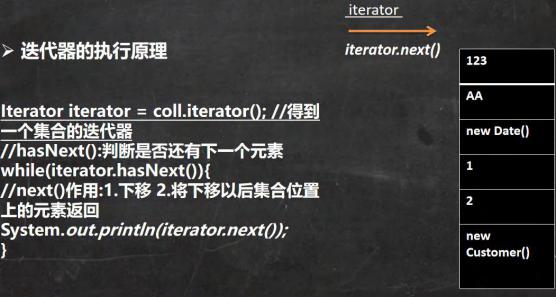
Collection 接口常用方法, 以实现子类 ArrayList 来演示. CollectionMethod.java

|  |
| --- |
| package com.hspedu.collection\_;  import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class CollectionMethod {  @SuppressWarnings({"all"})  public static void main(String[] args) {  List list = new ArrayList();  // add:添加单个元素  list.add("jack");  list.add(10);//list.add(new Integer(10))  list.add(true);  System.out.println("list=" + list);  // remove:删除指定元素  //list.remove(0);//删除第一个元素  list.remove(true);//指定删除某个元素  System.out.println("list=" + list);  // contains:查找元素是否存在  System.out.println(list.contains("jack"));//T  // size:获取元素个数  System.out.println(list.size());//2 |

|  |
| --- |
| // isEmpty:判断是否为空  System.out.println(list.isEmpty());//F  // clear:清空  list.clear();  System.out.println("list=" + list);  // addAll:添加多个元素  ArrayList list2 = new ArrayList();  list2.add("红楼梦");  list2.add("三国演义");  list.addAll(list2);  System.out.println("list=" + list);  // containsAll:查找多个元素是否都存在  System.out.println(list.containsAll(list2));//T  // removeAll：删除多个元素  list.add("聊斋");  list.removeAll(list2);  System.out.println("list=" + list);//[聊斋]  // 说明：以 ArrayList 实现类来演示.  }  } |

14.3.2 Collection 接口遍历元素方式 1-使用 Iterator(迭代器)







迭代器的使用案例

看下老师的案例演示 CollectionIterator.java

|  |
| --- |
| package com.hspedu.collection\_;  import java.util.ArrayList;  import java.util.Collection;  import java.util.Iterator;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class CollectionIterator { |

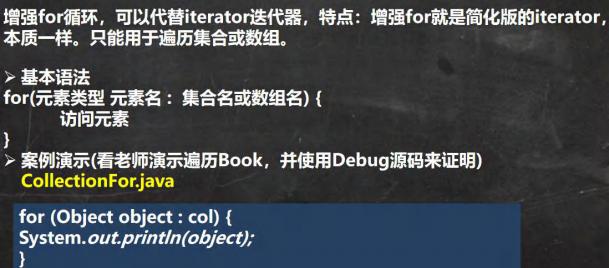
|  |
| --- |
| @SuppressWarnings({"all"})  public static void main(String[] args) {  Collection col = new ArrayList();  col.add(new Book("三国演义", "罗贯中", 10. 1));  col.add(new Book("小李飞刀", "古龙", 5. 1));  col.add(new Book("红楼梦", "曹雪芹", 34.6));  //System.out.println("col=" + col);  //现在老师希望能够遍历 col 集合  //1. 先得到 col 对应的 迭代器  Iterator iterator = col.iterator();  //2. 使用 while 循环遍历  // while (iterator.hasNext()) {//判断是否还有数据  // //返回下一个元素，类型是 Object  // Object obj = iterator.next();  // System.out.println("obj=" + obj);  // }  //老师教大家一个快捷键，快速生成 while => itit  //显示所有的快捷键的的快捷键 ctrl +j  while (iterator.hasNext()) {  Object obj = iterator.next();  System.out.println("obj=" + obj); |

|  |
| --- |
| }  //3. 当退出 while 循环后 , 这时 iterator 迭代器，指向最后的元素  // iterator.next();//NoSuchElementException  //4. 如果希望再次遍历，需要重置我们的迭代器  iterator = col.iterator();  System.out.println("===第二次遍历===");  while (iterator.hasNext()) {  Object obj = iterator.next();  System.out.println("obj=" + obj);  }  }  }  class Book {  private String name;  private String author;  private double price;  public Book(String name, String author, double price) {  this.name = name;  this.author = author;  this.price = price;  } |

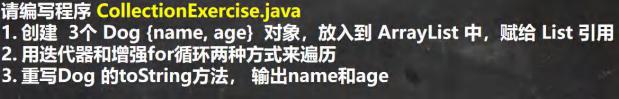
|  |
| --- |
| public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getAuthor() {  return author;  }  public void setAuthor(String author) {  this.author = author;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  @Override  public String toString() { |

|  |
| --- |
| return "Book{" +  "name='" + name + '\'' +  ", author='" + author + '\'' +  ", price=" + price +  '}';  }  } |

14.3.3 Collection 接口遍历对象方式 2-for 循环增强



14.3.4 课堂练习



|  |
| --- |
| package com.hspedu.collection\_;  import java.util.ArrayList;  import java.util.Iterator; |

|  |
| --- |
| import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class CollectionExercise {  @SuppressWarnings({"all"})  public static void main(String[] args) {  List list = new ArrayList();  list.add(new Dog("小黑", 3));  list.add(new Dog("大黄", 100));  list.add(new Dog("大壮", 8));  //先使用 for 增强  for (Object dog : list) {  System.out.println("dog=" + dog);  }  //使用迭代器  System.out.println("===使用迭代器来遍历===");  Iterator iterator = list.iterator();  while (iterator.hasNext()) {  Object dog = iterator.next();  System.out.println("dog=" + dog); |

|  |
| --- |
| }  }  }  /\*\*  \* 创建 3 个 Dog {name, age} 对象，放入到 ArrayList 中，赋给 List 引用  \* 用迭代器和增强 for循环两种方式来遍历  \* 重写 Dog 的 toString 方法， 输出 name 和 age  \*/  class Dog {  private String name;  private int age;  public Dog(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  } |



1

|  |
| --- |
| public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  @Override  public String toString() {  return "Dog{" +  "name='" + name + '\'' +  ", age=" + age +  '}';  }  } |

14.4 List 接口和常用方法

14.4.1 List 接口基本介绍



|  |
| --- |
| package com.hspedu.list\_;  import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class List\_ {  @SuppressWarnings({"all"})  public static void main(String[] args) {  //1. List 集合类中元素有序(即添加顺序和取出顺序一致) 、且可重复 [案例]  List list = new ArrayList();  list.add("jack");  list.add("tom");  list.add("mary");  list.add("hsp"); |

|  |
| --- |
| list.add("tom");  System.out.println("list=" + list);  //2. List 集合中的每个元素都有其对应的顺序索引，即支持索引  // 索引是从 0 开始的  System.out.println(list.get(3));//hsp  //3.  }  } |

14.4.2 List 接口的常用方法

ListMethod.java

|  |
| --- |
| package com.hspedu.list\_;  import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class ListMethod {  @SuppressWarnings({"all"})  public static void main(String[] args) {  List list = new ArrayList();  list.add("张三丰");  list.add("贾宝玉"); |

|  |
| --- |
| // void add(int index, Object ele):在 index 位置插入 ele 元素  //在 index= 1 的位置插入一个对象  list.add(1, "韩顺平");  System.out.println("list=" + list);  // boolean addAll(int index, Collection eles):从 index 位置开始将 eles 中的所有元素添加进来  List list2 = new ArrayList();  list2.add("jack");  list2.add("tom");  list.addAll(1, list2);  System.out.println("list=" + list);  // Object get(int index):获取指定 index 位置的元素  //说过  // int indexOf(Object obj):返回 obj 在集合中首次出现的位置  System.out.println(list.indexOf("tom"));//2  // int lastIndexOf(Object obj):返回 obj 在当前集合中末次出现的位置  list.add("韩顺平");  System.out.println("list=" + list);  System.out.println(list.lastIndexOf("韩顺平"));  // Object remove(int index):移除指定 index 位置的元素，并返回此元素  list.remove(0);  System.out.println("list=" + list);  // Object set(int index, Object ele):设置指定 index 位置的元素为 ele , 相当于是替换.  list.set(1, "玛丽");  System.out.println("list=" + list);  // List subList(int fromIndex, int toIndex):返回从 fromIndex 到 toIndex 位置的子集合  // 注意返回的子集合 fromIndex <= subList < toIndex |

|  |
| --- |
| List returnlist = list.subList(0, 2);  System.out.println("returnlist=" + returnlist);  }  } |

14.4.3 List 接口课堂练习

ListExercise.java

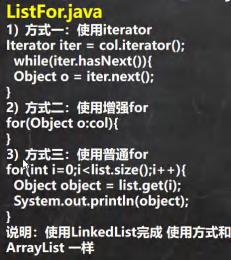


|  |
| --- |
| package com.hspedu.list\_;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class ListExercise {  @SuppressWarnings({"all"})  public static void main(String[] args) {  /\*  添加 10 个以上的元素(比如 String "hello" ) ，在 2 号位插入一个元素"韩顺平教育"，  获得第 5 个元素，删除第 6 个元素，修改第 7 个元素，在使用迭代器遍历集合， |

|  |
| --- |
| 要求:使用 List 的实现类 ArrayList 完成。  \*/  List list = new ArrayList();  for (int i = 0; i < 12; i++) {  list.add("hello" + i);  }  System.out.println("list=" + list);  //在 2 号位插入一个元素"韩顺平教育"  list.add(1, "韩顺平教育");  System.out.println("list=" + list);  //获得第 5 个元素  System.out.println("第五个元素=" + list.get(4));  //删除第 6 个元素  list.remove(5);  System.out.println("list=" + list);  //修改第 7 个元素  list.set(6, "三国演义");  System.out.println("list=" + list);  //在使用迭代器遍历集合  Iterator iterator = list.iterator();  while (iterator.hasNext()) {  Object obj = iterator.next();  System.out.println("obj=" + obj);  } |

|  |
| --- |
| }  } |

14.4.4 List 的三种遍历方式 [ArrayList, LinkedList,Vector]

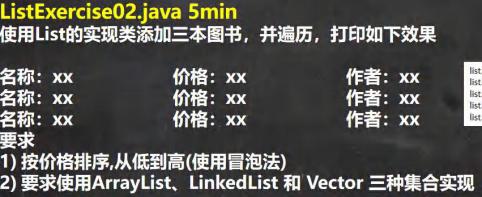


|  |
| --- |
| package com.hspedu.list\_;  import java.util.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class ListFor {  @SuppressWarnings({"all"})  public static void main(String[] args) {  //List 接口的实现子类 Vector LinkedList  //List list = new ArrayList(); |

|  |
| --- |
| //List list = new Vector();  List list = new LinkedList();  list.add("jack");  list.add("tom");  list.add("鱼香肉丝");  list.add("北京烤鸭子");  //遍历  //1. 迭代器  Iterator iterator = list.iterator();  while (iterator.hasNext()) {  Object obj = iterator.next();  System.out.println(obj);  }  System.out.println("=====增强 for=====");  //2. 增强 for  for (Object o : list) {  System.out.println("o=" + o);  }  System.out.println("=====普通 for====");  //3. 使用普通 for  for (int i = 0; i < list.size(); i++) { |

|  |
| --- |
| System.out.println("对象=" + list.get(i));  }  }  } |

14.4.5 实现类的课堂练习 2



|  |
| --- |
| package com.hspedu.list\_;  import java.util.ArrayList;  import java.util.LinkedList;  import java.util.List;  import java.util.Vector;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"}) |

|  |
| --- |
| public class ListExercise02 {  public static void main(String[] args) {  //List list = new ArrayList();  List list = new LinkedList();  //List list = new Vector();  list.add(new Book("红楼梦", "曹雪芹", 100));  list.add(new Book("西游记", "吴承恩", 10));  list.add(new Book("水浒传", "施耐庵", 19));  list.add(new Book("三国", "罗贯中", 80));  //list.add(new Book("西游记", "吴承恩", 10));  //如何对集合进行排序  //遍历  for (Object o : list) {  System.out.println(o);  }  //冒泡排序  sort(list);  System.out.println("==排序后=="); |

|  |
| --- |
| for (Object o : list) {  System.out.println(o);  }  }  //静态方法  //价格要求是从小到大  public static void sort(List list) {  int listSize = list.size();  for (int i = 0; i < listSize - 1; i++) {  for (intj = 0; j < listSize - 1 - i; j++) {  //取出对象 Book  Book book1 = (Book) list.get(j);  Book book2 = (Book) list.get(j + 1);  if (book1.getPrice() > book2.getPrice()) {//交换  list.set(j, book2);  list.set(j + 1, book1);  }  }  }  }  } |

14.5 ArrayList 底层结构和源码分析

14.5.1 ArrayList 的注意事项



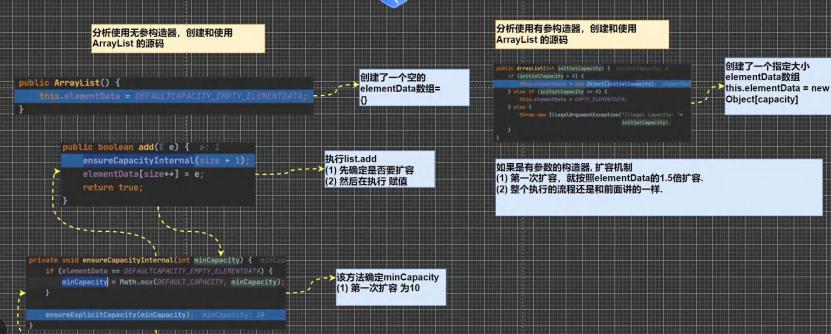
14.5.2 ArrayList 的底层操作机制源码分析(重点，难点.)



|  |
| --- |
| package com.hspedu.list\_;  import java.util.ArrayList;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class ArrayListSource { |

|  |
| --- |
| public static void main(String[] args) {  //老韩解读源码  //注意，注意，注意，Idea 默认情况下，Debug 显示的数据是简化后的，如果希望看到完整的数据  //需要做设置.  //使用无参构造器创建 ArrayList 对象  //ArrayList list = new ArrayList();  ArrayList list = new ArrayList(8);  //使用 for 给 list 集合添加 1- 10 数据  for (int i = 1; i <= 10; i++) {  list.add(i);  }  //使用 for 给 list 集合添加 11- 15 数据  for (int i = 11; i <= 15; i++) {  list.add(i);  }  list.add(100);  list.add(200);  list.add(null);  }  } |

示意图:





14.6 Vector 底层结构和源码剖析

14.6.1 Vector 的基本介绍

Vector\_.java



|  |
| --- |
| package com.hspedu.list\_;  import java.util.Vector;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class Vector\_ {  public static void main(String[] args) {  //无参构造器  //有参数的构造  Vector vector = new Vector(8);  for (int i = 0; i < 10; i++) {  vector.add(i); |

|  |
| --- |
| }  vector.add(100);  System.out.println("vector=" + vector);  //老韩解读源码  //1. new Vector() 底层  /\*  public Vector() {  this(10);  }  补充：如果是 Vector vector = new Vector(8);  走的方法:  public Vector(int initialCapacity) {  this(initialCapacity, 0);  }  2. vector.add(i)  2.1 //下面这个方法就添加数据到 vector 集合  public synchronized boolean add(E e) {  modCount++;  ensureCapacityHelper(elementCount + 1);  elementData[elementCount++] = e;  return true;  }  2.2 //确定是否需要扩容 条件 ： minCapacity - elementData.length>0  private void ensureCapacityHelper(int minCapacity) {  // overflow-conscious code  if (minCapacity - elementData.length > 0) |

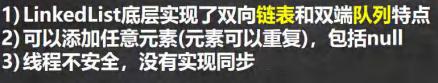
|  |
| --- |
| grow(minCapacity);  }  2.3 //如果 需要的数组大小 不够用，就扩容 , 扩容的算法  //newCapacity = oldCapacity + ((capacityIncrement > 0) ?  // capacityIncrement : oldCapacity);  //就是扩容两倍.  private void grow(int minCapacity) {  // overflow-conscious code  int oldCapacity = elementData.length;  int newCapacity = oldCapacity + ((capacityIncrement > 0) ?  capacityIncrement : oldCapacity);  if (newCapacity - minCapacity < 0)  newCapacity = minCapacity;  if (newCapacity - MAX\_ARRAY\_SIZE > 0)  newCapacity = hugeCapacity(minCapacity);  elementData = Arrays.copyOf(elementData, newCapacity);  }  \*/  }  } |

14.6.2 Vector 和 ArrayList 的比较

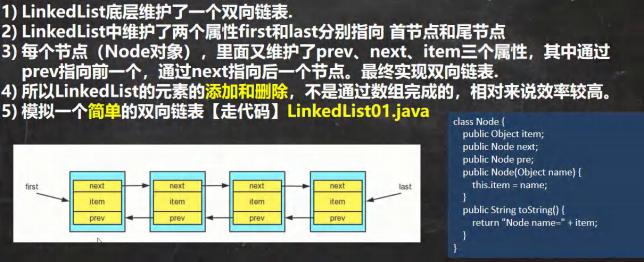


14.7 LinkedList 底层结构

14.7.1 LinkedList 的全面说明



14.7.2 LinkedList 的底层操作机制



|  |
| --- |
| package com.hspedu.list\_;  /\*\* |

|  |
| --- |
| \* @author 韩顺平  \* @version 1.0  \*/  public class LinkedList01 {  public static void main(String[] args) {  //模拟一个简单的双向链表  Node jack = new Node("jack");  Node tom = new Node("tom");  Node hsp = new Node("老韩");  //连接三个结点，形成双向链表  //jack -> tom -> hsp  jack.next = tom;  tom.next = hsp;  //hsp -> tom -> jack  hsp.pre = tom;  tom.pre = jack;  Node first = jack;//让 first 引用指向jack,就是双向链表的头结点  Node last = hsp; //让 last 引用指向 hsp,就是双向链表的尾结点  //演示，从头到尾进行遍历  System.out.println("===从头到尾进行遍历===");  while (true) { |

|  |
| --- |
| if(first == null) {  break;  }  //输出 first 信息  System.out.println(first);  first = first.next;  }  //演示，从尾到头的遍历  System.out.println("====从尾到头的遍历====");  while (true) {  if(last == null) {  break;  }  //输出 last 信息  System.out.println(last);  last = last.pre;  }  //演示链表的添加对象/数据，是多么的方便  //要求，是在 tom --------- 老韩直接，插入一个对象 smith  //1. 先创建一个 Node 结点，name 就是 smith  Node smith = new Node("smith");  //下面就把 smith 加入到双向链表了  smith.next = hsp; |

|  |
| --- |
| smith.pre = tom;  hsp.pre = smith;  tom.next = smith;  //让 first 再次指向jack  first = jack;//让 first 引用指向jack,就是双向链表的头结点  System.out.println("===从头到尾进行遍历===");  while (true) {  if(first == null) {  break;  }  //输出 first 信息  System.out.println(first);  first = first.next;  }  last = hsp; //让 last 重新指向最后一个结点  //演示，从尾到头的遍历  System.out.println("====从尾到头的遍历====");  while (true) {  if(last == null) {  break;  }  //输出 last 信息  System.out.println(last); |

|  |
| --- |
| last = last.pre;  }  }  }  //定义一个 Node 类，Node 对象 表示双向链表的一个结点  class Node {  public Object item; //真正存放数据  public Node next; //指向后一个结点  public Node pre; //指向前一个结点  public Node(Object name) {  this.item = name;  }  public String toString() {  return "Node name=" + item;  }  } |

14.7.3 LinkedList 的增删改查案例

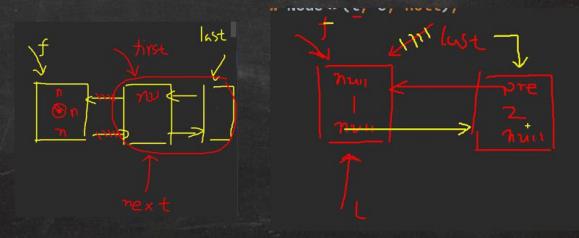
|  |
| --- |
| package com.hspedu.list\_;  import java.util.Iterator;  import java.util.LinkedList; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class LinkedListCRUD {  public static void main(String[] args) {  LinkedList linkedList = new LinkedList();  linkedList.add(1);  linkedList.add(2);  linkedList.add(3);  System.out.println("linkedList=" + linkedList);  //演示一个删除结点的  linkedList.remove(); // 这里默认删除的是第一个结点  //linkedList.remove(2);  System.out.println("linkedList=" + linkedList);  //修改某个结点对象  linkedList.set(1, 999);  System.out.println("linkedList=" + linkedList);  //得到某个结点对象  //get(1) 是得到双向链表的第二个对象 |

|  |
| --- |
| Object o = linkedList.get(1);  System.out.println(o);//999  //因为 LinkedList 是 实现了 List 接口, 遍历方式  System.out.println("===LinkeList 遍历迭代器====");  Iterator iterator = linkedList.iterator();  while (iterator.hasNext()) {  Object next = iterator.next();  System.out.println("next=" + next);  }  System.out.println("===LinkeList 遍历增强 for====");  for (Object o1 : linkedList) {  System.out.println("o1=" + o1);  }  System.out.println("===LinkeList 遍历普通 for====");  for (int i = 0; i < linkedList.size(); i++) {  System.out.println(linkedList.get(i));  }  //老韩源码阅读.  /\* 1. LinkedList linkedList = new LinkedList();  public LinkedList() {}  2. 这时 linkeList 的属性 first = null last = null |

|  |
| --- |
| 3. 执行 添加  public boolean add(E e) {  linkLast(e);  return true;  }  4.将新的结点，加入到双向链表的最后  void linkLast(E e) {  final Node<E> l = last;  final Node<E> newNode = new Node<>(l, e, null);  last = newNode;  if (l == null)  first = newNode;  else  l.next = newNode;  size++;  modCount++;  }  \*/  /\*  老韩读源码 linkedList.remove(); // 这里默认删除的是第一个结点  1. 执行 removeFirst  public E remove() {  return removeFirst();  } |

|  |
| --- |
| 2. 执行  public E removeFirst() {  final Node<E> f= first;  if (f == null)  throw new NoSuchElementException();  return unlinkFirst(f);  }  3. 执行 unlinkFirst, 将 f 指向的双向链表的第一个结点拿掉  private E unlinkFirst(Node<E> f) {  // assert f == first && f != null;  final E element = f.item;  final Node<E> next = f.next;  f.item = null;  f.next = null; // help GC  first = next;  if (next == null)  last = null;  else  next.prev = null;  size--;  modCount++;  return element;  }  \*/  }  } |



14.8 ArrayList 和 LinkedList 比较

14.8.1 ArrayList 和 LinkedList 的比较



14.9 Set 接口和常用方法

14.9.1 Set 接口基本介绍



14.9.2 Set 接口的常用方法

和 List 接口一样, Set 接口也是 Collection 的子接口，因此，常用方法和 Collection 接口一样.

14.9.3 Set 接口的遍历方式



14.9.4 Set 接口的常用方法举例

看老师案例演示 SetMethod.java

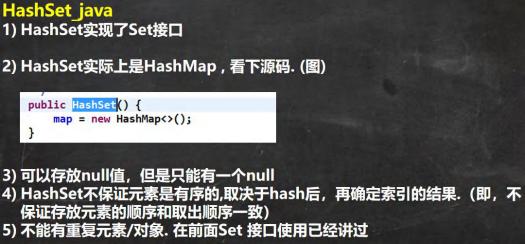
|  |
| --- |
| package com.hspedu.set\_;  import java.util.HashSet;  import java.util.Iterator;  import java.util.Set;  /\*\* |

|  |
| --- |
| \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class SetMethod {  public static void main(String[] args) {  //老韩解读  //1. 以 Set 接口的实现类 HashSet 来讲解 Set 接口的方法  //2. set 接口的实现类的对象(Set 接口对象), 不能存放重复的元素, 可以添加一个 null  //3. set 接口对象存放数据是无序(即添加的顺序和取出的顺序不一致)  //4. 注意：取出的顺序的顺序虽然不是添加的顺序，但是他的固定.  Set set = new HashSet();  set.add("john");  set.add("lucy");  set.add("john");//重复  set.add("jack");  set.add("hsp");  set.add("mary");  set.add(null);//  set.add(null);//再次添加 null  for(int i = 0; i <10;i ++) {  System.out.println("set=" + set);  }  //遍历  //方式 1： 使用迭代器 |

|  |
| --- |
| System.out.println("=====使用迭代器====");  Iterator iterator = set.iterator();  while (iterator.hasNext()) {  Object obj = iterator.next();  System.out.println("obj=" + obj);  }  set.remove(null);  //方式 2: 增强 for  System.out.println("=====增强 for====");  for (Object o : set) {  System.out.println("o=" + o);  }  //set 接口对象，不能通过索引来获取  }  } |

14. 10Set 接口实现类-HashSet

14.10. 1 HashSet 的全面说明



|  |
| --- |
| package com.hspedu.set\_;  import java.util.HashSet;  import java.util.Set;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class HashSet\_ {  public static void main(String[] args) {  //老韩解读  //1. 构造器走的源码  /\*  public HashSet() {  map = new HashMap<>();  } |

|  |
| --- |
| 2. HashSet 可以存放 null ,但是只能有一个 null, 即元素不能重复  \*/  Set hashSet = new HashSet();  hashSet.add(null);  hashSet.add(null);  System.out.println("hashSet=" + hashSet);  }  } |

14.10.2 HashSet 案例说明



|  |
| --- |
| package com.hspedu.set\_;  import java.util.HashSet;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class HashSet01 {  public static void main(String[] args) {  HashSet set = new HashSet(); |

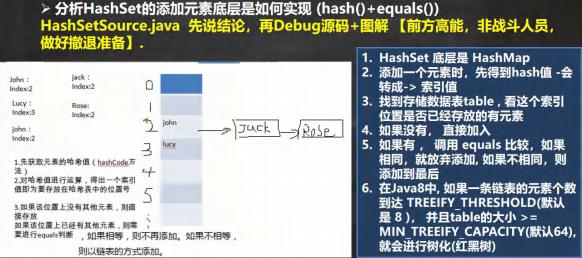
|  |
| --- |
| //说明  //1. 在执行 add 方法后，会返回一个 boolean 值  //2. 如果添加成功，返回 true, 否则返回 false  //3. 可以通过 remove 指定删除哪个对象  System.out.println(set.add("john"));//T  System.out.println(set.add("lucy"));//T  System.out.println(set.add("john"));//F  System.out.println(set.add("jack"));//T  System.out.println(set.add("Rose"));//T  set.remove("john");  System.out.println("set=" + set);//3 个  //  set = new HashSet();  System.out.println("set=" + set);//0  //4 Hashset 不能添加相同的元素/数据?  set.add("lucy");//添加成功  set.add("lucy");//加入不了  set.add(new Dog("tom"));//OK  set.add(new Dog("tom"));//Ok  System.out.println("set=" + set);  //在加深一下. 非常经典的面试题. |

|  |
| --- |
| //看源码，做分析， 先给小伙伴留一个坑，以后讲完源码，你就了然  //去看他的源码，即 add 到底发生了什么?=> 底层机制.  set.add(new String("hsp"));//ok  set.add(new String("hsp"));//加入不了.  System.out.println("set=" + set);  }  }  class Dog { //定义了 Dog 类  private String name;  public Dog(String name) {  this.name = name;  }  @Override  public String toString() {  return "Dog{" +  "name='" + name + '\'' +  '}';  }  } |

14.10.3 HashSet 底层机制说明







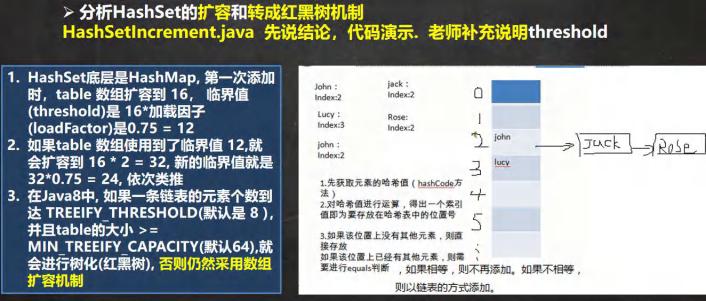
|  |
| --- |
| package com.hspedu.set\_;  import java.util.HashSet;  /\*\*  \* @author 韩顺平 |

|  |
| --- |
| \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class HashSetSource {  public static void main(String[] args) {  HashSet hashSet = new HashSet();  hashSet.add("java");//到此位置，第 1 次 add 分析完毕.  hashSet.add("php");//到此位置，第 2 次 add 分析完毕  hashSet.add("java");  System.out.println("set=" + hashSet);  /\*  老韩对 HashSet 的源码解读  1. 执行 HashSet()  public HashSet() {  map = new HashMap<>();  }  2. 执行 add()  public boolean add(E e) {//e = "java"  return map.put(e, PRESENT)==null;//(static) PRESENT = new Object();  }  3.执行 put() , 该方法会执行 hash(key) 得到 key 对应的 hash 值 算法 h = key.hashCode()) ^ (h >>> 16) public V put(K key, V value) {//key = "java" value = PRESENT 共享  return putVal(hash(key), key, value, false, true);  } |

|  |
| --- |
| 4.执行 putVal  final V putVal(int hash, K key, V value, boolean onlyIfAbsent,  boolean evict) {  Node<K,V>[] tab; Node<K,V> p; int n, i; //定义了辅助变量  //table 就是 HashMap 的一个数组，类型是 Node[]  //if 语句表示如果当前 table 是 null, 或者 大小=0  //就是第一次扩容，到 16 个空间.  if ((tab = table) == null || (n = tab.length) == 0)  n = (tab = resize()).length;  //(1)根据 key ，得到 hash 去计算该 key 应该存放到 table 表的哪个索引位置  //并把这个位置的对象，赋给 p  //(2)判断 p 是否为 null  //(2. 1) 如果 p 为 null, 表示还没有存放元素, 就创建一个 Node (key="java",value=PRESENT) //(2.2) 就放在该位置 tab[i] = newNode(hash, key, value, null)  if ((p = tab[i = (n - 1) & hash]) == null)  tab[i] = newNode(hash, key, value, null);  else {  //一个开发技巧提示： 在需要局部变量(辅助变量)时候，在创建  Node<K,V> e; K k; //  //如果当前索引位置对应的链表的第一个元素和准备添加的 key 的 hash 值一样  //并且满足 下面两个条件之一:  //(1) 准备加入的 key 和 p 指向的 Node 结点的 key 是同一个对象  //(2) p 指向的 Node 结点的 key 的 equals() 和准备加入的 key 比较后相同  //就不能加入 |

|  |
| --- |
| if (p.hash == hash &&  ((k = p.key) == key || (key != null && key.equals(k))))  e = p;  //再判断 p 是不是一颗红黑树,  //如果是一颗红黑树，就调用 putTreeVal , 来进行添加  else if (p instanceof TreeNode)  e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);  else {//如果 table 对应索引位置，已经是一个链表, 就使用 for 循环比较  //(1) 依次和该链表的每一个元素比较后，都不相同, 则加入到该链表的最后  // 注意在把元素添加到链表后，立即判断 该链表是否已经达到 8 个结点  // , 就调用 treeifyBin() 对当前这个链表进行树化(转成红黑树)  // 注意，在转成红黑树时，要进行判断, 判断条件  // if (tab == null || (n = tab.length) < MIN\_TREEIFY\_CAPACITY(64))  // resize();  // 如果上面条件成立，先 table 扩容.  // 只有上面条件不成立时，才进行转成红黑树  //(2) 依次和该链表的每一个元素比较过程中，如果有相同情况,就直接 break  for (int binCount = 0; ; ++binCount) {  if ((e = p.next) == null) {  p.next = newNode(hash, key, value, null);  if (binCount >= TREEIFY\_THRESHOLD(8) - 1) // - 1 for 1st  treeifyBin(tab, hash);  break;  }  if (e.hash == hash && |

|  |
| --- |
| ((k = e.key) == key || (key != null && key.equals(k))))  break;  p = e;  }  }  if (e != null) { // existing mapping for key  V oldValue = e.value;  if (!onlyIfAbsent || oldValue == null)  e.value = value;  afterNodeAccess(e);  return oldValue;  }  }  ++modCount;  //size 就是我们每加入一个结点 Node(k,v,h,next), size++  if (++size > threshold)  resize();//扩容  afterNodeInsertion(evict);  return null;  }  \*/  }  } |



|  |
| --- |
| package com.hspedu.set\_;  import java.util.HashSet;  import java.util.Objects;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class HashSetIncrement {  public static void main(String[] args) {  /\*  HashSet 底层是 HashMap, 第一次添加时，table 数组扩容到 16，  临界值(threshold)是 16\*加载因子(loadFactor)是 0.75 = 12  如果 table 数组使用到了临界值 12,就会扩容到 16 \* 2 = 32, |

|  |
| --- |
| 新的临界值就是 32\*0.75 = 24, 依次类推  \*/  HashSet hashSet = new HashSet();  // for(int i = 1; i <= 100; i++) {  // hashSet.add(i);//1,2,3,4,5...100  // }  /\*  在 Java8 中, 如果一条链表的元素个数到达 TREEIFY\_THRESHOLD(默认是 8 )，  并且 table 的大小 >= MIN\_TREEIFY\_CAPACITY(默认 64),就会进行树化(红黑树),  否则仍然采用数组扩容机制  \*/  // for(int i = 1; i <= 12; i++) {  // hashSet.add(new A(i));//  // }  /\*  当我们向 hashset 增加一个元素，-> Node -> 加入 table , 就算是增加了一个 size++  \*/  for(int i = 1; i <= 7; i++) {//在 table 的某一条链表上添加了 7 个 A 对象  hashSet.add(new A(i));// |

|  |
| --- |
| }  for(int i = 1; i <= 7; i++) {//在 table 的另外一条链表上添加了 7 个 B 对象  hashSet.add(new B(i));//  }  }  }  class B {  private int n;  public B(int n) {  this.n = n;  }  @Override  public int hashCode() {  return 200;  }  }  class A {  private int n; |

|  |
| --- |
| public A(int n) {  this.n = n;  }  @Override  public int hashCode() {  return 100;  }  } |

14.10.4 HashSet 课堂练习 1



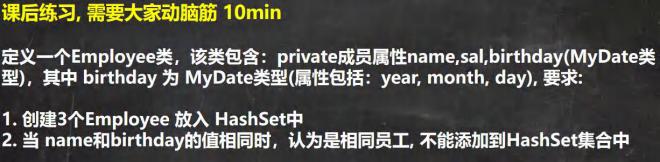
|  |
| --- |
| package com.hspedu.set\_;  import java.util.HashSet;  import java.util.LinkedHashSet;  import java.util.Objects;  import java.util.Set; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class HashSetExercise {  public static void main(String[] args) {  /\*\*  定义一个 Employee 类，该类包含：private 成员属性 name,age 要求:  创建 3 个 Employee 对象放入 HashSet 中  当 name 和 age 的值相同时，认为是相同员工, 不能添加到 HashSet 集合中  \*/  HashSet hashSet = new HashSet();  hashSet.add(new Employee("milan", 18));//ok  hashSet.add(new Employee("smith", 28));//ok  hashSet.add(new Employee("milan", 18));//加入不成功.  //回答,加入了几个? 3 个  System.out.println("hashSet=" + hashSet);  }  }  //创建 Employee |

|  |
| --- |
| class Employee {  private String name;  private int age;  public Employee(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  @Override  public String toString() {  return "Employee{" +  "name='" + name + '\'' +  ", age=" + age + |

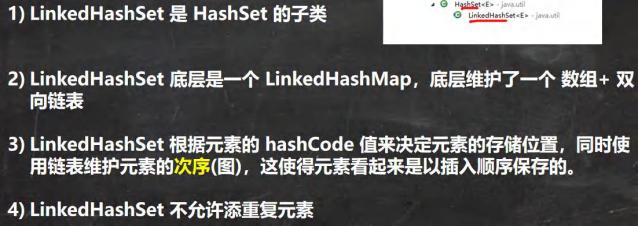
|  |
| --- |
| '}';  }  public void setAge(int age) {  this.age = age;  }  //如果 name 和 age 值相同，则返回相同的 hash 值  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Employee employee = (Employee) o;  return age == employee.age &&  Objects.equals(name, employee.name);  }  @Override  public int hashCode() {  return Objects.hash(name, age);  }  } |

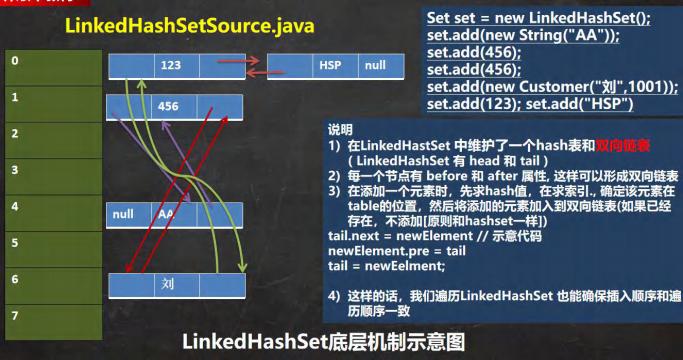
14.10.5 HashSet 课后练习 2



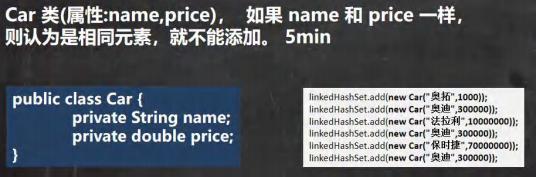
14. 11Set 接口实现类-LinkedHashSet

14.11. 1 LinkedHashSet 的全面说明





14.11.2 LinkedHashSet 课后练习题 LinkedHashSetExercise.java



|  |
| --- |
| package com.hspedu.set\_;  import java.util.LinkedHashSet;  import java.util.Objects;  /\*\* |

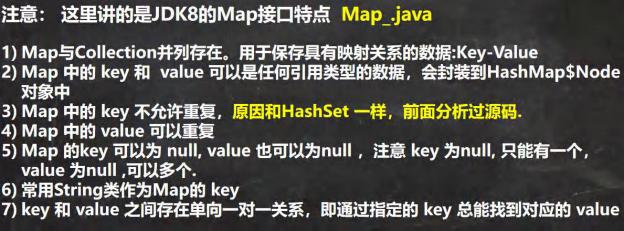
|  |
| --- |
| \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class LinkedHashSetExercise {  public static void main(String[] args) {  LinkedHashSet linkedHashSet = new LinkedHashSet();  linkedHashSet.add(new Car("奥拓", 1000));//OK  linkedHashSet.add(new Car("奥迪", 300000));//OK  linkedHashSet.add(new Car("法拉利", 10000000));//OK  linkedHashSet.add(new Car("奥迪", 300000));//加入不了  linkedHashSet.add(new Car("保时捷", 70000000));//OK  linkedHashSet.add(new Car("奥迪", 300000));//加入不了  System.out.println("linkedHashSet=" + linkedHashSet);  }  }  /\*\*  \* Car 类(属性:name,price) ， 如果 name 和 price 一样，  \* 则认为是相同元素，就不能添加。 5min  \*/  class Car { |

|  |
| --- |
| private String name;  private double price;  public Car(String name, double price) {  this.name = name;  this.price = price;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  @Override  public String toString() { |

|  |
| --- |
| return "\nCar{" +  "name='" + name + '\'' +  ", price=" + price +  '}';  }  //重写 equals 方法 和 hashCode  //当 name 和 price 相同时， 就返回相同的 hashCode 值, equals 返回 t  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Car car = (Car) o;  return Double.compare(car.price, price) == 0 &&  Objects.equals(name, car.name);  }  @Override  public int hashCode() {  return Objects.hash(name, price);  }  } |

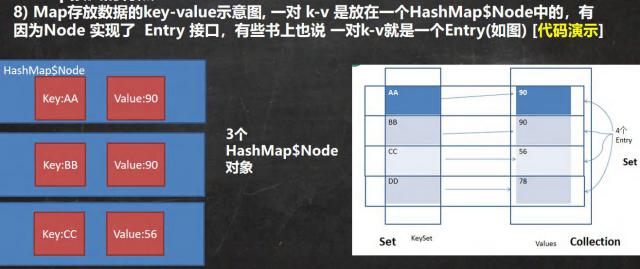
14. 12Map 接口和常用方法

14.12. 1 Map 接口实现类的特点 [很实用]



|  |
| --- |
| package com.hspedu.map\_;  import java.util.HashMap;  import java.util.Map;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class Map\_ {  public static void main(String[] args) {  //老韩解读 Map 接口实现类的特点, 使用实现类 HashMap  //1. Map 与 Collection 并列存在。用于保存具有映射关系的数据:Key-Value(双列元素)  //2. Map 中的 key 和 value 可以是任何引用类型的数据，会封装到 HashMap$Node 对象中  //3. Map 中的 key 不允许重复，原因和 HashSet 一样，前面分析过源码.  //4. Map 中的 value 可以重复  //5. Map 的 key 可以为 null, value 也可以为 null ，注意 key 为 null, |

|  |
| --- |
| // 只能有一个，value 为 null ,可以多个  //6. 常用 String 类作为 Map 的 key  //7. key 和 value 之间存在单向一对一关系，即通过指定的 key 总能找到对应的 value  Map map = new HashMap();  map.put("no1", "韩顺平");//k-v  map.put("no2", "张无忌");//k-v  map.put("no1", "张三丰");//当有相同的 k , 就等价于替换.  map.put("no3", "张三丰");//k-v  map.put(null, null); //k-v  map.put(null, "abc"); //等价替换  map.put("no4", null); //k-v  map.put("no5", null); //k-v  map.put(1, "赵敏");//k-v  map.put(new Object(), "金毛狮王");//k-v  // 通过 get 方法，传入 key ,会返回对应的 value  System.out.println(map.get("no2"));//张无忌  System.out.println("map=" + map);  }  } |



14.12.2 Map 接口常用方法

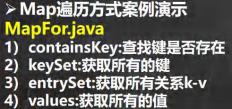
|  |
| --- |
| package com.hspedu.map\_;  import java.util.HashMap;  import java.util.Map;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class MapMethod {  public static void main(String[] args) {  //演示map 接口常用方法  Map map = new HashMap();  map.put("邓超", new Book("", 100));//OK |

|  |
| --- |
| map.put("邓超", "孙俪");//替换-> 一会分析源码  map.put("王宝强", "马蓉");//OK  map.put("宋喆", "马蓉");//OK  map.put("刘令博", null);//OK  map.put(null, "刘亦菲");//OK  map.put("鹿晗", "关晓彤");//OK  map.put("hsp", "hsp 的老婆");  System.out.println("map=" + map);  // remove:根据键删除映射关系  map.remove(null);  System.out.println("map=" + map);  // get：根据键获取值  Object val = map.get("鹿晗");  System.out.println("val=" + val);  // size:获取元素个数  System.out.println("k-v=" + map.size());  // isEmpty:判断个数是否为 0  System.out.println(map.isEmpty());//F  // clear:清除 k-v  //map.clear();  System.out.println("map=" + map);  // containsKey:查找键是否存在  System.out.println("结果=" + map.containsKey("hsp"));//T |

|  |
| --- |
| }  }  class Book {  private String name;  private int num;  public Book(String name, int num) {  this.name = name;  this.num = num;  }  } |

14.12.3 Map 接口遍历方法





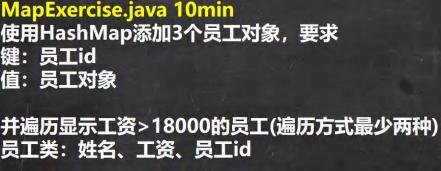
|  |
| --- |
| package com.hspedu.map\_;  import java.util.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class MapFor {  public static void main(String[] args) {  Map map = new HashMap();  map.put("邓超", "孙俪");  map.put("王宝强", "马蓉");  map.put("宋喆", "马蓉");  map.put("刘令博", null);  map.put(null, "刘亦菲");  map.put("鹿晗", "关晓彤");  //第一组: 先取出 所有的 Key , 通过 Key 取出对应的 Value  Set keyset = map.keySet(); |

|  |
| --- |
| //(1) 增强 for  System.out.println("-----第一种方式-------");  for (Object key : keyset) {  System.out.println(key + "-" + map.get(key));  }  //(2) 迭代器  System.out.println("----第二种方式--------");  Iterator iterator = keyset.iterator();  while (iterator.hasNext()) {  Object key = iterator.next();  System.out.println(key + "-" + map.get(key));  }  //第二组: 把所有的 values 取出  Collection values = map.values();  //这里可以使用所有的 Collections 使用的遍历方法  //(1) 增强 for  System.out.println("---取出所有的 value 增强 for----");  for (Object value : values) {  System.out.println(value);  }  //(2) 迭代器  System.out.println("---取出所有的 value 迭代器----");  Iterator iterator2 = values.iterator();  while (iterator2.hasNext()) {  Object value = iterator2.next(); |

|  |
| --- |
| System.out.println(value);  }  //第三组: 通过 EntrySet 来获取 k-v  Set entrySet = map.entrySet();// EntrySet<Map.Entry<K,V>>  //(1) 增强 for  System.out.println("----使用 EntrySet 的 for 增强(第 3 种)----");  for (Object entry : entrySet) {  //将 entry 转成 Map.Entry  Map.Entry m = (Map.Entry) entry;  System.out.println(m.getKey() + "-" + m.getValue());  }  //(2) 迭代器  System.out.println("----使用 EntrySet 的 迭代器(第 4 种)----");  Iterator iterator3 = entrySet.iterator();  while (iterator3.hasNext()) {  Object entry = iterator3.next();  //System.out.println(next.getClass());//HashMap$Node -实现-> Map.Entry (getKey,getValue)  //向下转型 Map.Entry  Map.Entry m = (Map.Entry) entry;  System.out.println(m.getKey() + "-" + m.getValue());  }  } |

|  |
| --- |
| } |

14.12.4 Map 接口课堂练习



|  |
| --- |
| package com.hspedu.map\_;  import java.util.HashMap;  import java.util.Iterator;  import java.util.Map;  import java.util.Set;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class MapExercise {  public static void main(String[] args) {  //完成代码  Map hashMap = new HashMap();  //添加对象 |

|  |
| --- |
| hashMap.put(1, new Emp("jack", 300000, 1));  hashMap.put(2, new Emp("tom", 21000, 2));  hashMap.put(3, new Emp("milan", 12000, 3));  //遍历 2 种方式  //并遍历显示工资>18000 的员工(遍历方式最少两种)  //1. 使用 keySet -> 增强 for  Set keySet = hashMap.keySet();  System.out.println("====第一种遍历方式====");  for (Object key : keySet) {  //先获取 value  Emp emp = (Emp) hashMap.get(key);  if(emp.getSal() >18000) {  System.out.println(emp);  }  }  //2. 使用 EntrySet -> 迭代器  // 体现比较难的知识点  // 慢慢品，越品越有味道.  Set entrySet = hashMap.entrySet();  System.out.println("======迭代器======");  Iterator iterator = entrySet.iterator();  while (iterator.hasNext()) {  Map.Entry entry = (Map.Entry)iterator.next(); |

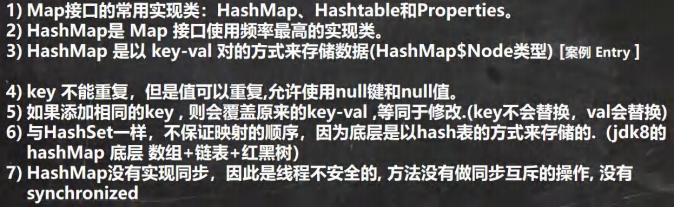
|  |
| --- |
| //通过 entry 取得 key 和 value  Emp emp = (Emp) entry.getValue();  if(emp.getSal() > 18000) {  System.out.println(emp);  }  }  }  }  /\*\*  \* 使用 HashMap 添加 3 个员工对象，要求  \* 键：员工 id  \* 值：员工对象  \*  \* 并遍历显示工资>18000 的员工(遍历方式最少两种)  \* 员工类：姓名、工资、员工 id  \*/  class Emp {  private String name;  private double sal;  private int id;  public Emp(String name, double sal, int id) {  this.name = name;  this.sal = sal;  this.id = id; |

|  |
| --- |
| }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getSal() {  return sal;  }  public void setSal(double sal) {  this.sal = sal;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  } |

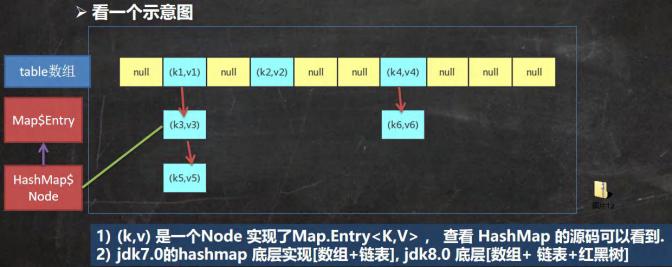
|  |
| --- |
| @Override  public String toString() {  return "Emp{" +  "name='" + name + '\'' +  ", sal=" + sal +  ", id=" + id +  '}';  }  } |

14. 13Map 接口实现类-HashMap

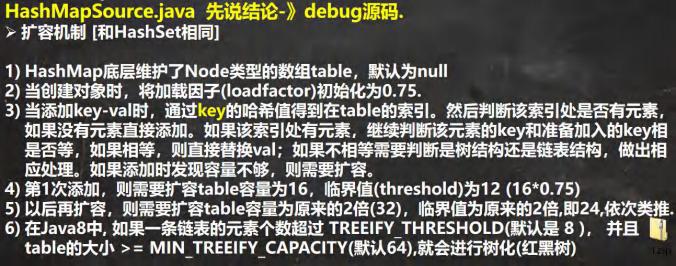
14.13. 1 HashMap 小结



14.13.2 HashMap 底层机制及源码剖析



14.13.3 HashMap 底层机制及源码剖析



|  |
| --- |
| package com.hspedu.map\_;  import java.util.HashMap;  /\*\*  \* @author 韩顺平  \* @version 1.0 |

|  |
| --- |
| \*/  @SuppressWarnings({"all"})  public class HashMapSource1 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put("java", 10);//ok  map.put("php", 10);//ok  map.put("java", 20);//替换 value  System.out.println("map=" + map);//  /\*老韩解读 HashMap 的源码+ 图解  1. 执行构造器 new HashMap()  初始化加载因子 loadfactor = 0.75  HashMap$Node[] table = null  2. 执行 put 调用 hash 方法，计算 key 的 hash 值 (h = key.hashCode()) ^ (h >>> 16)  public V put(K key, V value) {//K = "java" value = 10  return putVal(hash(key), key, value, false, true);  }  3. 执行 putVal  final V putVal(int hash, K key, V value, boolean onlyIfAbsent,  boolean evict) {  Node<K,V>[] tab; Node<K,V> p; int n, i;//辅助变量  //如果底层的 table 数组为 null, 或者 length =0 , 就扩容到 16  if ((tab = table) == null || (n = tab.length) == 0)  n = (tab = resize()).length; |

|  |
| --- |
| //取出 hash 值对应的 table 的索引位置的 Node, 如果为 null, 就直接把加入的 k-v  //, 创建成一个 Node ,加入该位置即可  if ((p = tab[i = (n - 1) & hash]) == null)  tab[i] = newNode(hash, key, value, null);  else {  Node<K,V> e; K k;//辅助变量  // 如果 table 的索引位置的 key 的 hash 相同和新的 key 的 hash 值相同，  // 并 满足(table 现有的结点的 key 和准备添加的 key 是同一个对象 || equals 返回真)  // 就认为不能加入新的 k-v  if (p.hash == hash &&  ((k = p.key) == key || (key != null && key.equals(k))))  e = p;  else if (p instanceof TreeNode)//如果当前的 table 的已有的 Node 是红黑树，就按照红黑树的方式处  理  e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key, value);  else {  //如果找到的结点，后面是链表，就循环比较  for (int binCount = 0; ; ++binCount) {//死循环  if ((e = p.next) == null) {//如果整个链表，没有和他相同,就加到该链表的最后 p.next = newNode(hash, key, value, null);  //加入后，判断当前链表的个数，是否已经到 8 个，到 8 个，后  //就调用 treeifyBin 方法进行红黑树的转换  if (binCount >= TREEIFY\_THRESHOLD - 1) // - 1 for 1st  treeifyBin(tab, hash);  break;  } |

|  |
| --- |
| if (e.hash == hash && //如果在循环比较过程中，发现有相同,就 break,就只是替换 value ((k = e.key) == key || (key != null && key.equals(k))))  break;  p = e;  }  }  if (e != null) { // existing mapping for key  V oldValue = e.value;  if (!onlyIfAbsent || oldValue == null)  e.value = value; //替换，key 对应 value  afterNodeAccess(e);  return oldValue;  }  }  ++modCount;//每增加一个 Node ,就 size++  if (++size > threshold[12-24-48])//如 size > 临界值，就扩容  resize();  afterNodeInsertion(evict);  return null;  }  5. 关于树化(转成红黑树)  //如果 table 为 null ,或者大小还没有到 64 ，暂时不树化，而是进行扩容.  //否则才会真正的树化 -> 剪枝  final void treeifyBin(Node<K,V>[] tab, int hash) {  int n, index; Node<K,V> e; |

|  |
| --- |
| if (tab == null || (n = tab.length) < MIN\_TREEIFY\_CAPACITY)  resize();  }  \*/  }  } |





|  |
| --- |
| package com.hspedu.map\_;  import java.util.HashMap;  import java.util.Objects; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class HashMapSource2 {  public static void main(String[] args) {  HashMap hashMap = new HashMap();  for(int i = 1; i <= 12; i++) {  hashMap.put(i, "hello");  }  hashMap.put("aaa", "bbb");  System.out.println("hashMap=" + hashMap);//12 个 k-v  //布置一个任务， 自己设计代码去验证，table 的扩容  //0 -> 16(12) -> 32(24) -> 64(64\*0.75=48)-> 128 (96) ->  //自己设计程序，验证- 》 增强自己阅读源码能力. 看别人代码.  }  }  class A { |

|  |
| --- |
| private int num;  public A(int num) {  this.num = num;  }  //所有的 A 对象的 hashCode 都是 100  // @Override  // public int hashCode() {  // return 100;  // }  @Override  public String toString() {  return "\nA{" +  "num=" + num +  '}';  }  } |

14. 14Map 接口实现类-Hashtable

14.14. 1 HashTable 的基本介绍

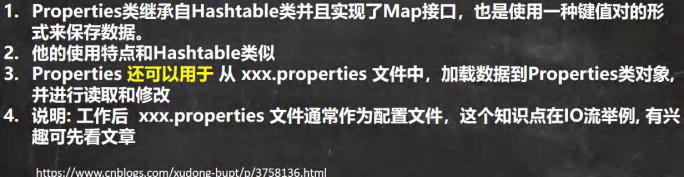


14.14.2 Hashtable 和 HashMap 对比



14. 15Map 接口实现类-Properties

14.15. 1 基本介绍

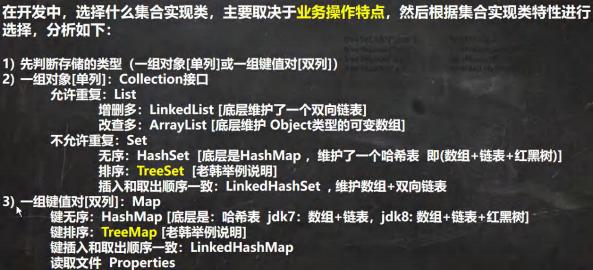


14.15.2 基本使用

|  |
| --- |
| package com.hspedu.map\_;  import java.util.Properties;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class Properties\_ {  public static void main(String[] args) {  //老韩解读  //1. Properties 继承 Hashtable  //2. 可以通过 k-v 存放数据，当然 key 和 value 不能为 null  //增加  Properties properties = new Properties();  //properties.put(null, "abc");//抛出 空指针异常  //properties.put("abc", null); //抛出 空指针异常  properties.put("john", 100);//k-v  properties.put("lucy", 100);  properties.put("lic", 100);  properties.put("lic", 88);//如果有相同的 key ， value 被替换  System.out.println("properties=" + properties); |

|  |
| --- |
| //通过 k 获取对应值  System.out.println(properties.get("lic"));//88  //删除  properties.remove("lic");  System.out.println("properties=" + properties);  //修改  properties.put("john", "约翰");  System.out.println("properties=" + properties);  }  } |

14. 16总结-开发中如何选择集合实现类(记住)



|  |
| --- |
| package com.hspedu.set\_;  import java.util.Comparator; |

|  |
| --- |
| import java.util.TreeSet;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class TreeSet\_ {  public static void main(String[] args) {  //老韩解读  //1. 当我们使用无参构造器，创建 TreeSet 时，仍然是无序的  //2. 老师希望添加的元素，按照字符串大小来排序  //3. 使用 TreeSet 提供的一个构造器，可以传入一个比较器(匿名内部类)  // 并指定排序规则  //4. 简单看看源码  //老韩解读  /\*  1. 构造器把传入的比较器对象，赋给了 TreeSet 的底层的 TreeMap 的属性 this.comparator  public TreeMap(Comparator<? super K> comparator) {  this.comparator = comparator;  }  2. 在 调用 treeSet.add("tom"), 在底层会执行到  if (cpr != null) {//cpr 就是我们的匿名内部类(对象) |

|  |
| --- |
| do {  parent = t;  //动态绑定到我们的匿名内部类(对象)compare  cmp = cpr.compare(key, t.key);  if (cmp < 0)  t = t.left;  else if (cmp > 0)  t = t.right;  else //如果相等，即返回 0,这个 Key 就没有加入  return t.setValue(value);  } while (t != null);  }  \*/  // TreeSet treeSet = new TreeSet();  TreeSet treeSet = new TreeSet(new Comparator() {  @Override  public int compare(Object o1, Object o2) {  //下面 调用 String 的 compareTo 方法进行字符串大小比较  //如果老韩要求加入的元素，按照长度大小排序  //return ((String) o2).compareTo((String) o1);  return ((String) o1).length() - ((String) o2).length();  }  });  //添加数据.  treeSet.add("jack"); |

|  |
| --- |
| treeSet.add("tom");//3  treeSet.add("sp");  treeSet.add("a");  treeSet.add("abc");//3  System.out.println("treeSet=" + treeSet);  }  } |
| package com.hspedu.map\_;  import java.util.Comparator;  import java.util.TreeMap;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"all"})  public class TreeMap\_ {  public static void main(String[] args) { |

|  |
| --- |
| //使用默认的构造器，创建 TreeMap, 是无序的(也没有排序)  /\*  老韩要求：按照传入的 k(String) 的大小进行排序  \*/  // TreeMap treeMap = new TreeMap();  TreeMap treeMap = new TreeMap(new Comparator() {  @Override  public int compare(Object o1, Object o2) {  //按照传入的 k(String) 的大小进行排序  //按照 K(String) 的长度大小排序  //return ((String) o2).compareTo((String) o1);  return ((String) o2).length() - ((String) o1).length();  }  });  treeMap.put("jack", "杰克");  treeMap.put("tom", "汤姆");  treeMap.put("kristina", "克瑞斯提诺");  treeMap.put("smith", "斯密斯");  treeMap.put("hsp", "韩顺平");//加入不了  System.out.println("treemap=" + treeMap);  /\*  老韩解读源码： |

|  |
| --- |
| 1. 构造器. 把传入的实现了 Comparator 接口的匿名内部类(对象) ，传给给 TreeMap 的 comparator  public TreeMap(Comparator<? super K> comparator) {  this.comparator = comparator;  }  2. 调用 put 方法  2. 1 第一次添加, 把 k-v 封装到 Entry 对象，放入 root  Entry<K,V> t = root;  if (t == null) {  compare(key, key); // type (and possibly null) check  root = new Entry<>(key, value, null);  size = 1;  modCount++;  return null;  }  2.2 以后添加  Comparator<? super K> cpr = comparator;  if (cpr != null) {  do { //遍历所有的 key , 给当前 key 找到适当位置  parent = t;  cmp = cpr.compare(key, t.key);//动态绑定到我们的匿名内部类的 compare  if (cmp < 0)  t = t.left;  else if (cmp > 0)  t = t.right;  else //如果遍历过程中，发现准备添加 Key 和当前已有的 Key 相等，就不添加 |

|  |
| --- |
| return t.setValue(value);  } while (t != null);  }  \*/  }  } |

14. 17Collections 工具类

14.17. 1 Collections 工具类介绍



14.17.2 排序操作：（均为 static 方法)



|  |
| --- |
| package com.hspedu.collections\_;  import java.util.\*;  /\*\*  \* @author 韩顺平  \* @version 1.0 |

|  |
| --- |
| \*/  @SuppressWarnings({"all"})  public class Collections\_ {  public static void main(String[] args) {  //创建 ArrayList 集合，用于测试.  List list = new ArrayList();  list.add("tom");  list.add("smith");  list.add("king");  list.add("milan");  list.add("tom");  // reverse(List)：反转 List 中元素的顺序  Collections.reverse(list);  System.out.println("list=" + list);  // shuffle(List)：对 List 集合元素进行随机排序  // for (int i = 0; i < 5; i++) {  // Collections.shuffle(list);  // System.out.println("list=" + list);  // }  // sort(List)：根据元素的自然顺序对指定 List 集合元素按升序排序  Collections.sort(list);  System.out.println(" 自然排序后"); |

|  |
| --- |
| System.out.println("list=" + list);  // sort(List ，Comparator)：根据指定的 Comparator 产生的顺序对 List 集合元素进行排序  //我们希望按照 字符串的长度大小排序  Collections.sort(list, new Comparator() {  @Override  public int compare(Object o1, Object o2) {  //可以加入校验代码.  return ((String) o2).length() - ((String) o1).length();  }  });  System.out.println("字符串长度大小排序=" + list);  // swap(List ，int ， int)：将指定 list 集合中的 i 处元素和 j 处元素进行交换  //比如  Collections.swap(list, 0, 1);  System.out.println("交换后的情况");  System.out.println("list=" + list);  //Object max(Collection)：根据元素的自然顺序，返回给定集合中的最大元素  System.out.println(" 自然顺序最大元素=" + Collections.max(list));  //Object max(Collection ，Comparator)：根据 Comparator 指定的顺序，返回给定集合中的最大元素  //比如，我们要返回长度最大的元素  Object maxObject = Collections.max(list, new Comparator() {  @Override  public int compare(Object o1, Object o2) {  return ((String)o1).length() - ((String)o2).length(); |

|  |
| --- |
| }  });  System.out.println("长度最大的元素=" + maxObject);  //Object min(Collection)  //Object min(Collection ，Comparator)  //上面的两个方法，参考 max 即可  //int frequency(Collection ，Object)：返回指定集合中指定元素的出现次数  System.out.println("tom 出现的次数=" + Collections.frequency(list, "tom"));  //void copy(List dest,List src)：将 src 中的内容复制到 dest 中  ArrayList dest = new ArrayList();  //为了完成一个完整拷贝，我们需要先给 dest 赋值，大小和 list.size()一样  for(int i = 0; i < list.size(); i++) {  dest.add("");  }  //拷贝  Collections.copy(dest, list);  System.out.println("dest=" + dest);  //boolean replaceAll(List list ，Object oldVal ，Object newVal)：使用新值替换 List 对象的所有旧值  //如果 list 中，有 tom 就替换成 汤姆  Collections.replaceAll(list, "tom", "汤姆"); |

|  |
| --- |
| System.out.println("list 替换后=" + list);  }  } |

14.17.3 排序操作：（均为 static 方法)

应用案例演示

14.17.4 查找、替换



14. 18本章作业

