Java进阶1 第8天

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【应用】List体系**
2. 【应用】能够独立使用LinkedList
3. 【应用】能够阐述数组和链表结构的实现原理
4. 【理解】能够理解不同数据结构的特点
5. **【应用】set体系**

a.【应用】能够使用HashSet去重(自定义对象)

b.【应用】能够用文字描述HashSet去重的过程

c.【应用】能够独立写出HashSet与增强for循环练习

1. **【应用】单列集合总结**
2. 【理解】能够阐述整个单列集合体系
3. 【理解】能够理解list和set集合中不同类的使用场景

# LinkedList的使用&数据结构

## LinkedList的基本使用

LinkedList与ArrayList不同，LinkedList是方便添加或删除的List。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，

所以该具体子类的特点在于提供了大量首尾操作。

public void **addFirst**(E e) 添加首个元素

public void **addLast**(E e) 添加最后元素

public E **getFirst**() 获取首个元素

public E **getLast**() 获取最后元素

### 案例代码一:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.util.Iterator;  **import** java.util.LinkedList;  /\*\*  \* **@ClassName**: LinkedListDemo  \* **@Description**: LinkedList的测试类  \* **@date** 2017年11月17日 下午1:41:16  \* Company www.igeekhome.com  \*  \* LinkedList是方便添加删除的List。  \* 提供了大量首尾操作  \*  \* public void addFirst(E e) 添加首个元素  \* public void addLast(E e) 添加最后元素  \* public E getFirst() 获取首个元素  \* public E getLast() 获取最后元素  \*/  **public** **class** LinkedListDemo {  **public** **static** **void** main(String[] args) {  //创建集合对象  LinkedList<String> list = **new** LinkedList<String>();    //向集合中添加元素  list.add("Jack");  list.add("Rose");  list.add("Trump");    //获取元素  String name = list.get(1);  System.***out***.println(name);    //返回集合的迭代器  Iterator<String> itr = list.iterator();  **while**(itr.hasNext()) {  String thisName = itr.next();  System.***out***.println(thisName);  }  System.***out***.println("=======================");    //使用增强for循环  **for** (String thisName : list) {  System.***out***.println(thisName);  }    System.***out***.println("======================");    //做首尾的添加与删除动作  System.***out***.println(list);    list.addFirst("LiLei");  list.addLast("HanMeiMei");    System.***out***.println(list);    System.***out***.println(list.getFirst());  System.***out***.println(list.getLast());  }  } |

## 数据结构

每种集合的特点不同(如更适合增删还是更适合查找)，是因为底层的数据结构不同。

数据结构指的数据存储和组织方式， 这里介绍最常用的数据结构：

### 栈结构

容器先进后出规则

#### 案例代码二:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.util.LinkedList;  /\*\*  \* **@ClassName**: StackDemo  \* **@Description**: stack栈结构  \* **@date** 2017年11月17日 下午1:44:14  \* Company www.igeekhome.com  \*  \* stack栈结构:先进后出  \*  \* LinkedList是支持栈结构的.  \* 对应的方法  \* push压栈  \* pop弹栈  \* peek查询出即将要弹出的是哪个元素,检查有没有要弹出的元素  \*  \* java提供了一个专门用于栈结构的类,Stack  \*/  **public** **class** StackDemo {  **public** **static** **void** main(String[] args) {  //创建栈结构的集合  LinkedList<String> stack = **new** LinkedList<String>();    stack.push("Jack");  stack.push("Rose");  stack.push("Trump");    System.***out***.println(stack);    //获取一个元素,此时,由于是栈结构,所以获取的是最后压栈的元素  String popName = stack.pop();  System.***out***.println(popName);    //弹栈动作,集合中减少元素  System.***out***.println(stack);    String peekName = stack.peek();  System.***out***.println(peekName);    //peek查看元素不会减少元素  System.***out***.println(stack);  }  } |

数组结构：一块连续的存储区域

链表结构：每个元素指向下一个元素

### 队列结构

容器先进先出的规则

#### 案例代码三:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.util.LinkedList;  /\*\*  \* **@ClassName**: QueueDemo  \* **@Description**: queue队列结构  \* **@date** 2017年11月17日 下午1:50:03  \* Company www.igeekhome.com  \*  \* queue队列结构:先进先出  \*  \* LinkedList是支持队列结构的  \*  \* 对应的方法  \* offer加入队列  \* poll离开队列  \* peek查询出即将要离开队列的是哪个元素,检查有没有要离开队列的元素  \*/  **public** **class** QueueDemo {  **public** **static** **void** main(String[] args) {  //创建队列结构的集合  LinkedList<String> queue = **new** LinkedList<String>();    queue.offer("Jack");  queue.offer("Rose");  queue.offer("Trump");    System.***out***.println(queue);    //获取一个元素,此时,由于是队列结构,所以获取的是最早入队的元素  String pollName = queue.poll();  System.***out***.println(pollName);    //出队动作,集合中减少元素  System.***out***.println(queue);    String peekName = queue.peek();  System.***out***.println(peekName);    //peek查看元素不会减少元素  System.***out***.println(queue);  }  } |

## ArrayList数组实现的原理

数组实现的特点：查询快，增删慢

原因：

查询快：由于数组的索引支持，那么可以通过索引直接计算出元素的地址值，因此就可以直接通过元素的地址值获取到指定的元素

增删慢：由于在添加元素的时候，实际上底层会先创建一个新数组(新数组的长度为原数组的长度+1)，那么在添加新元素的时候，先需要对数组中原有的数据进行拷贝，其次在末尾进行添加新的元素

因此，这样操作的效率的极低的(删除元素 刚好和添加的操作相反)

## LinkedList链表实现的原理

链表结构：查询慢，增删快

查询慢：由于不能直接找到元素的地址，需要上一个元素推导出下一个元素的地址，

这种查询速度较慢

增删快：在添加的时候，只需要更改元素所记录的地址值即可

# HashSet

## HashSet的特性和基本使用

HashSet是Set接口的子类，不包含重复元素相同元素，且无序。

HashSet下还有子类LinkedHashSet，可预测迭代顺序的Set集合。

### 案例代码四:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.util.HashSet;  **import** java.util.Iterator;  /\*\*  \* **@ClassName**: HashSetDemo  \* **@Description**: HashSet的测试类  \* **@date** 2017年11月17日 下午2:04:48  \* Company www.igeekhome.com  \*  \* HashSet，不包含重复元素相同元素，且无序，不提供索引，所以不能通过索引获取元素，只能通过迭代器访问数据。  \*/  **public** **class** HashSetDemo {  **public** **static** **void** main(String[] args) {  //创建集合对象  HashSet<String> set = **new** HashSet<String>();  //向集合中添加元素  set.add("Jack");  set.add("Rose");  set.add("Trump");  set.add("Obama");  set.add("Obama");    System.***out***.println(set);    //获取元素,只能用迭代器  Iterator<String> itr = set.iterator();    **while**(itr.hasNext()) {  String thisName = itr.next();  System.***out***.println(thisName);  }    System.***out***.println("================");  **for** (String thisName : set) {  System.***out***.println(thisName);  }  }  } |

## HashSet与增强for循环练习

### 案例代码五:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.util.HashSet;  /\*\*  \* **@ClassName**: HashSetTest  \* **@Description**: HashSet的练习类  \* **@date** 2017年11月17日 下午2:06:59  \* Company www.igeekhome.com  \*  \* 求一个字符串"aiodjl;hriWFUADJSVUEHiowfjnivowe"中一共有几个不重复的字母,区分大小写,如a,A算两个字符  \*  \* 分析：  \* 将字符串拆分成一个个单个字符  \* 判断字符是否为字母  \* 如果是字母,放到一个HashSet集合中  \* 查看集合长度  \*  \*/  **public** **class** HashSetTest {  **public** **static** **void** main(String[] args) {  String s = "aiodjl;hriWFUADJSVUEHiowfjnivowe";    //定义不包含重复元素的HashSet集合  HashSet<Character> set = **new** HashSet<Character>();    //将字符串拆分成一个个单个字符  **char**[] charArray = s.toCharArray();    //使用增强for循环遍历数组  **for**(**char** c : charArray) {    //判断字符是否为字母  **if**(('a'<=c&&c<='z')||('A'<=c&&c<='Z')) {    //如果是字母,放到一个HashSet集合中  set.add(c);  }  }    //查看集合长度  System.***out***.println("在该字符串中,一共存在不重复的字母"+set.size()+"个");  }  } |

## ArrayList中contains方法如何判断是否有重复元素

自定义类型Person类是Object类的子类，所以Person具备equals方法contains方法会调用参数的equals方法，依次与集合当中已经存在的Person对象比较。

当Person类没有重写equals方法时，直接使用Object类基础过来的equals方法，而该方法比较的是对象地址值。

当Person类重写equals方法后，就可以讲比较规则由比较地址值改为比较属性值。

### 案例代码六:

|  |
| --- |
| **package** com.igeek\_03;  **import** java.util.ArrayList;  /\*\*  \* **@ClassName**: ArrayListContainsDemo  \* **@Description**: ArrayList中contains方法  \* **@date** 2017年11月17日 下午2:14:07  \* Company www.igeekhome.com  \*  \*/  **public** **class** ArrayListContainsDemo {  **public** **static** **void** main(String[] args) {  //创建ArrayList对象  ArrayList<String> list = **new** ArrayList<String>();    //在list中加入数据  list.add("Jack");  list.add("Rose");  list.add("Trump");    //判断集合中是否存在Jack  System.***out***.println(list.contains("Jack"));  System.***out***.println("========================");  //ArrayList当中存储自定义数据类型Person  ArrayList<Person> list2 = **new** ArrayList<Person>();    Person p = **new** Person("Jack",18);  Person p2 = **new** Person("Rose",16);  Person p3 = **new** Person("Trump",62);    list2.add(p);  list2.add(p2);  list2.add(p3);    System.***out***.println(list2.contains(**new** Person("Jack", 18)));  System.***out***.println(list2.contains(p));  }  } |

## 自定义类重写equals方法

自定义类需要满足咱们认为的效果，也就是不同的对象但是属性值相同就认为是同一个对象

在集合里只存其一的效果，就必须重写equals方法，且更改该方法里的比较规则

### 案例代码七:

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月17日 下午2:16:25  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {    /\*\*  \* **@Fields** name :姓名  \*/  **private** String name;    /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;  /\*\*  \* **@Title**: Person  \* **@param** name  \* **@param** age  \*/  **public** Person(String name, **int** age) {  **this**.name=name;  **this**.age=age;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }  /\*  \* 如果没有重写该equals方法,则比较地址值.  \* 我们希望比较两个对象时,比较属性内容,所以重写该方法.  \*/  /\*\*  \* **@Title**: equals  \* **@Description**: Person对象比较是否相等的方法  \* **@param** obj  \* **@return**  \* **@see** java.lang.Object#equals(java.lang.Object)  \*/  @Override  **public** **boolean** equals(Object obj) {  System.***out***.println("equals方法被调用了");    //this:contains方法的参数对象  //obj:集合当中原有元素  //将Object类型的obj强转为子类类型  Person otherP = (Person)obj;    //比较姓名  **if**(!**this**.name.equals(otherP.name)) {  **return** **false**;  }  //比较年龄  **if**(**this**.age != otherP.age){  **return** **false**;  }  //如果所有属性值均相同,就返回true  **return** **true**;  }  } |

|  |
| --- |
| **package** com.igeek\_04;  **import** java.util.ArrayList;  /\*\*  \* **@ClassName**: ArrayListContainsDemo  \* **@Description**: ArrayList中contains方法  \* **@date** 2017年11月17日 下午2:14:07  \* Company www.igeekhome.com  \*  \*/  **public** **class** ArrayListContainsDemo {  **public** **static** **void** main(String[] args) {  //创建ArrayList对象  ArrayList<String> list = **new** ArrayList<String>();    //在list中加入数据  list.add("Jack");  list.add("Rose");  list.add("Trump");    //判断集合中是否存在Jack  System.***out***.println(list.contains("Jack"));  System.***out***.println("========================");  //ArrayList当中存储自定义数据类型Person  ArrayList<Person> list2 = **new** ArrayList<Person>();    Person p = **new** Person("Jack",18);  Person p2 = **new** Person("Rose",16);  Person p3 = **new** Person("Trump",62);    list2.add(p);  list2.add(p2);  list2.add(p3);    System.***out***.println(list2.contains(**new** Person("Jack", 18)));  System.***out***.println(list2.contains(p));  }  } |

## HashSet判断元素唯一性规则

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：

把对象加入到HashSet时，它会先使用对象的hashCode值来判断对象加入的位置；

如果此位置上没有其他对象存在，则判断元素不同，可存入新对象。

如果此位置上有其他对象存在，先判断新元素与集合内已经有的旧元素的HashCode值：

如果不同，判断元素不同，可存入新对象。

如果相同，再判断equals比较结果，返回true则相同，则不能存入新对象；返回false则仍然不同，可存入新对象。

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，使用的地址值，如果想通过内容比较元素是否相同，需要重写该类的hashCode与equals方法。

hashCode方法重写规则：

将该对象的各个属性值hashCode相加即是整个对象的HashCode值。

如果是基本类型，类似int，则直接返回int值就是该属性的hash值；

如果是引用类型，类似String，就调用该成员变量的hashCode方法返回该成 员变量hash值。

这样可以根据对象的内容返回hashCode值，从而可以根据hashCode判断 元素是否唯一。

但是由于在一些”碰巧的”情况下，可能出现内容不同但hashCode相同的情 况，为了避免这些情况，我们加入一些干扰系数。

可是加入干扰系数后，仍会出现一些”碰巧”的情况，所以我们还要进行 equals的二次判断。

## 重写hashCode和equals方法

### 案例代码八:

|  |
| --- |
| **package** com.igeek\_06;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月17日 下午2:16:25  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {    /\*\*  \* **@Fields** name :姓名  \*/  **private** String name;    /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;  /\*\*  \* **@Title**: Person  \* **@param** name  \* **@param** age  \*/  **public** Person(String name, **int** age) {  **this**.name=name;  **this**.age=age;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }  /\*  \* 如果没有重写该equals方法,则继承父类方法,比较地址值.  \* 我们希望比较两个对象时,比较属性内容,所以重写该方法.  \*/  /\*\*  \* **@Title**: equals  \* **@Description**: Person对象比较是否相等的方法  \* **@param** obj  \* **@return**  \* **@see** java.lang.Object#equals(java.lang.Object)  \*/  @Override  **public** **boolean** equals(Object obj) {  System.***out***.println("equals方法被调用了");    //this:contains方法的参数对象  //obj:集合当中原有元素  //将Object类型的obj强转为子类类型  Person otherP = (Person)obj;    //比较姓名  **if**(!**this**.name.equals(otherP.name)) {  **return** **false**;  }  //比较年龄  **if**(**this**.age != otherP.age){  **return** **false**;  }  //如果所有属性值均相同,就返回true  **return** **true**;  }  //没有重写hashCode方法时,hashCode方法继承父类方法,返回的是地址值  //重写hashCode方法,通过属性值返回一个整数数字  //即对象的hashCode值,就是各个属性的hashCode值之和  //引用数据类型属性,调用hashCode方法回去  //基本数值类型属性,直接就是数值本身  /\*\*  \* **@Title**: hashCode  \* **@Description**: Person对象的hashCode值算法方法  \* **@return**  \* **@see** java.lang.Object#hashCode()  \*/  @Override  **public** **int** hashCode() {    //张三 18 >> 56\*系数 + 18 = 74  //李四 56 >> 18\*系数 + 56 = 74  //理想上,不同属性值应该返回不同的hashCode值,可以在每次结果后乘以一个固定系数,避免该情况发生    //定义变量,记录要返回的hashCode值  **int** thisHashCode = 0;    //获取name属性的hashCode值  thisHashCode += name.hashCode()\*17;  //获取age数只给你的hashCode值  thisHashCode += age;    **return** thisHashCode;  }    } |

## Eclipse生成hashCode和equals方法

eclipse可以使用快捷键自动生成对应的hashCode方法与equals方法

### 案例代码九:

|  |
| --- |
| **package** com.igeek\_07;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月17日 下午2:16:25  \* Company www.igeekhome.com  \*  \*/  **public** **class** Person {    /\*\*  \* **@Fields** name :姓名  \*/  **private** String name;    /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;  /\*\*  \* **@Title**: Person  \* **@param** name  \* **@param** age  \*/  **public** Person(String name, **int** age) {  **this**.name=name;  **this**.age=age;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }  /\*\*  \* **@Title**: hashCode  \* **@Description**: Person对象的hashCode值算法方法  \* **@return**  \* **@see** java.lang.Object#hashCode()  \*/  @Override  **public** **int** hashCode() {  //定义系数  **final** **int** prime = 31;  //定义返回的hashCode值  **int** result = 1;  //每次将结果\*系数,再加下一个成员变量,可以大大降低不同成员变量返回相同hashCode值的情况  result = prime \* result + age;  result = prime \* result + ((name == **null**) ? 0 : name.hashCode());  **return** result;  }  /\*\*  \* **@Title**: equals  \* **@Description**: Person对象比较是否相等的方法  \* **@param** obj  \* **@return**  \* **@see** java.lang.Object#equals(java.lang.Object)  \*/  @Override  **public** **boolean** equals(Object obj) {  //自身与自身比较直接返回true  **if** (**this** == obj)  **return** **true**;    //参数是null直接返回false  **if** (obj == **null**)  **return** **false**;    //如果本对象的类型与参数对象的类型不一样,则直接返回false  **if** (getClass() != obj.getClass())  **return** **false**;    //如果类型相同,就转成(强制类型转换)相同的类型继续比较属性值  Person other = (Person) obj;  //比较年龄  **if** (age != other.age)  **return** **false**;    //比较姓名  **if** (name == **null**) {  **if** (other.name != **null**)  **return** **false**;  } **else** **if** (!name.equals(other.name))  **return** **false**;    //如果一切不相同的情况都不符合,则说明两个对象的属性值相同,返回true  **return** **true**;  }  } |

# Java单列集合框架之总结&体系图

## Java集合框架之总结&体系图

### 单列集合体系图

LinkedHashSet

HashSet

LinkedList

ArrayList

Set

List

Collection

Collection：所有单列集合的直接或间接接口，其指定了所有集合应该具备的基本功能。

List：元素可重复，有序，带索引

Set：元素不能重复，没有索引

ArrayList(重要) ：底层是数组结构。ArrayList的出现替代了Vector，增删慢，查找快

LinkedList(重要)：底层是链表结构。同时对元素的增删操作效率很高

HashSet(重要) ：底层是哈希表结构。在不重复的基础上无序

LinkedHashSet：底层是哈希表结构结合链表结构。在不重复的基础上可预测迭代顺序

### 集合的常见使用

在设计集合的使用时，经常定义对应的接口类型。在使用时，通过多态的方式给接口变量赋值，用于提高程序扩展性。

如：

定义方法public void method(List list) {}，在调用方法时依据需求，传入ArrayList或者LinkedList的对象。

重点和总结

1、LinkedList的使用及其内部实现原理

2、ArrayList的内部实现原理

3、HashSet的使用及其内部实现原理

4、重要的hashCode方法和equals方法的作用和使用

5、Java中单列集合框架的体系图