第07天 集合

今日内容介绍

* HashSet集合
* HashMap集合

# HashSet集合

## Set接口的特点

Set体系的集合:

A:存入集合的顺序和取出集合的顺序不一致

B:没有索引

C:存入集合的元素没有重复

## HashSet使用&唯一性原理

### HashSet的使用

#### 案例代码一:

**public** **class** HashSetDemo2 {

**public** **static** **void** main(String[] args) {

//创建集合对象

HashSet<Student> hs = **new** HashSet<Student>();

//创建元素对象

Student s = **new** Student("zhangsan",18);

Student s2 = **new** Student("lisi",19);

Student s3 = **new** Student("lisi",19);

//添加元素对象

hs.add(s);

hs.add(s2);

hs.add(s3);

//遍历集合对象

**for** (Student student : hs) {

System.*out*.println(student);

}

}

}

### HashSet唯一性原理

规则:新添加到HashSet集合的元素都会与集合中已有的元素一一比较

首先比较哈希值(每个元素都会调用hashCode()产生一个哈希值)

如果新添加的元素与集合中已有的元素的哈希值都不同,新添加的元素**存入集合**

如果新添加的元素与集合中已有的某个元素哈希值相同,此时还需要调用equals(Object obj)比较

如果equals(Object obj)方法返回true,说明新添加的元素与集合中已有的某个元素的属性值相同,那么新添加的元素**不存入集合**

如果equals(Object obj)方法返回false, 说明新添加的元素与集合中已有的元素的属性值都不同, 那么新添加的元素**存入集合**

#### 案例代码二:

**package** com.itheima\_01;

**import** java.util.HashSet;

/\*

\* 使用HashSet存储自定义对象并遍历

\* 通过查看源码发现：

\* HashSet的add()方法，首先会使用当前集合中的每一个元素和新添加的元素进行hash值比较，

\* 如果hash值不一样，则直接添加新的元素

\* 如果hash值一样，比较地址值或者使用equals方法进行比较

\* 比较结果一样，则认为是重复不添加

\* 所有的比较结果都不一样则添加

\*/

**public** **class** HashSetDemo2 {

**public** **static** **void** main(String[] args) {

//创建集合对象

HashSet<Student> hs = **new** HashSet<Student>();

//创建元素对象

Student s = **new** Student("zhangsan",18);

Student s2 = **new** Student("lisi",19);

Student s3 = **new** Student("lisi",19);

//添加元素对象

hs.add(s);

hs.add(s2);

hs.add(s3);

//遍历集合对象

**for** (Student student : hs) {

System.*out*.println(student);

}

}

}

**class** Student {

String name;

**int** age;

**public** Student(String name,**int** age) {

**this**.name = name;

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Student [name=" + name + ", age=" + age + "]";

}

@Override

**public** **boolean** equals(Object obj) {

//System.out.println("-------------------");

Student s = (Student)obj;//向下转型，可以获取子类特有成员

//比较年龄是否相等，如果不等则返回false

**if**(**this**.age != s.age) {

**return** **false**;

}

//比较姓名是否相等，如果不等则返回false

**if**(!**this**.name.equals(s.name)) {

**return** **false**;

}

//默认返回true，说明两个学生是相等的

**return** **true**;

}

@Override

**public** **int** hashCode() {

**return** 1;

}

}

#### hashCode方法优化:

如果让hashCode()方法返回一个固定值,那么每个新添加的元素都要调用equals(Object obj)方法比较,那么效率较低

只需要让不同属性的值的元素产生不同的哈希值,那么就可以不再调用equals方法比较提高效率

#### 案例代码三:

package com.itheima\_02;

public class Person {

String name;

int age;

public Person(String name,int age) {

this.name = name;

this.age = age;

}

@Override

public String toString() {

return "Person [name=" + name + ", age=" + age + "]";

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + age;

result = prime \* result + ((name == null) ? 0 : name.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Person other = (Person) obj;

if (age != other.age)

return false;

if (name == null) {

if (other.name != null)

return false;

} else if (!name.equals(other.name))

return false;

return true;

}

/\*

@Override

public int hashCode() {

\* 我们发现当hashCode方法永远返回整数1时，所有对象的hash值都是一样的，

\* 有一些对象他的成员变量完全不同，但是他们还需要进行hash和equals方法的比较，

\* 如果我们可以让成员变量不同的对象，他们的hash值也不同，这就可以减少一部分equals方法的比较

\* 从而可以提高我们程序的效率

\*

\* 可以尝试着让hashCode方法的返回值和对象的成员变量有关

\* 可以让hashCode方法返回所有成员变量之和，

\* 让基本数据类型直接想加，然后引用数据类型获取hashCode方法返回值后再相加（boolean不可以参与运算）

\*

//return age;

return age + name.hashCode();

}

@Override

public boolean equals(Object obj) {

System.out.println("-------------");

//提高效率

if(this == obj) {

return true;

}

//提高健壮性

if(this.getClass() != obj.getClass()) {

return false;

}

//向下转型

Person p = (Person)obj;

if(!this.name.equals(p.name)) {

return false;

}

if(this.age != p.age) {

return false;

}

return true;

}\*/

}

**package** com.itheima\_02;

**import** java.util.HashSet;

**public** **class** HashSetDemo3 {

**public** **static** **void** main(String[] args) {

//创建集合对象

HashSet<Person> hs = **new** HashSet<Person>();

//创建元素对象

Person p = **new** Person("zhangsan",18);

Person p2 = **new** Person("lisi",18);

Person p3 = **new** Person("lisi",18);

//添加元素对象

hs.add(p);

hs.add(p2);

hs.add(p3);

//遍历集合对象

**for** (Person person : hs) {

System.*out*.println(person);

}

}

}

## Collections中的方法

### 案例代码四:

package com.itheima\_03;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

/\*

\* Collections：

\* 面试题：Collection和Collections有什么区别？

\* Collection是集合体系的最顶层，包含了集合体系的共性

\* Collections是一个工具类，方法都是用于操作Collection

\*

\*/

public class CollectionsDemo {

public static void main(String[] args) {

//static void swap(List list, int i, int j) :将指定列表中的两个索引进行位置互换

List<Integer> list = new ArrayList<Integer>();

list.add(1);

list.add(4);

Collections.swap(list, 0, 1);

System.out.println(list);

}

private static void method6() {

//static void sort(List<T> list) :按照列表中元素的自然顺序进行排序

List<Integer> list = new ArrayList<Integer>();

list.add(1);

list.add(4);

list.add(3);

list.add(2);

Collections.sort(list);

System.out.println(list);

}

private static void method5() {

//static void shuffle(List list):傻否，随机置换

List<Integer> list = new ArrayList<Integer>();

list.add(1);

list.add(2);

list.add(3);

list.add(4);

Collections.shuffle(list);

System.out.println(list);

}

private static void method4() {

//static void reverse(List list) :反转

List<Integer> list = new ArrayList<Integer>();

list.add(1);

list.add(2);

list.add(3);

list.add(4);

Collections.reverse(list);

System.out.println(list);

}

private static void method3() {

//static void fill(List list, Object obj) :使用指定的对象填充指定列表的所有元素

List<String> list = new ArrayList<String>();

list.add("hello");

list.add("world");

list.add("java");

System.out.println(list);

Collections.fill(list, "android");

System.out.println(list);

}

private static void method2() {

//static void copy(List dest, List src) :是把源列表中的数据覆盖到目标列表

//注意：目标列表的长度至少等于源列表的长度

//创建源列表

List<String> src = new ArrayList<String>();

src.add("hello");

src.add("world");

src.add("java");

//创建目标列表

List<String> dest = new ArrayList<String>();

dest.add("java");

dest.add("java");

dest.add("java");

dest.add("java");

Collections.copy(dest, src);

System.out.println(dest);

}

private static void method() {

//static int binarySearch(List list, Object key) 使用二分查找法查找指定元素在指定列表的索引位置

List<Integer> list = new ArrayList<Integer>();

list.add(1);

list.add(2);

list.add(3);

list.add(4);

int index = Collections.binarySearch(list, 4);

System.out.println(index);

}

}

## 斗地主案例

具体规则：

1. 组装54张扑克牌

2. 将54张牌顺序打乱

3. 三个玩家参与游戏，三人交替摸牌，每人17张牌，最后三张留作底牌。

4. 查看三人各自手中的牌、底牌

### 案例代码五:

package com.itheima\_03;

import java.util.ArrayList;

import java.util.Collections;

/\*

\* 模拟斗地主发牌

买牌

洗牌

发牌

\*/

public class CollectionsTest {

public static void main(String[] args) {

//买牌

String[] arr = {"黑桃","红桃","方片","梅花"};

String[] arr2 = {"A","2","3","4","5","6","7","8","9","10","J","Q","K"};

ArrayList<String> box = new ArrayList<String>();

//添加每张牌

for (int i = 0; i < arr.length; i++) {

//获取每一个花色

for (int j = 0; j < arr2.length; j++) {

//获取每一个数

box.add(arr[i] + arr2[j]);

}

}

box.add("大王");

box.add("小王");

//System.out.println(box.size());

//洗牌

Collections.shuffle(box);

//System.out.println(box);

//发牌

ArrayList<String> 林志玲 = new ArrayList<String>();

ArrayList<String> 林心如 = new ArrayList<String>();

ArrayList<String> 舒淇 = new ArrayList<String>();

//留三张底牌给地主

for (int i = 0; i < box.size() - 3; i++) {

/\*

\* i = 0;i % 3 = 0;

\* i = 1;i % 3 = 1;

\* i = 2;i % 3 = 2;

\* i = 3;i % 3 = 0;

\* i = 4;i % 4 = 1;

\* i = 5;i % 5 = 2;

\*/

if(i % 3 == 0) {

林志玲.add(box.get(i));

}

else if(i % 3 == 1) {

林心如.add(box.get(i));

}

else if(i % 3 == 2) {

舒淇.add(box.get(i));

}

}

System.out.println("林志玲：" + 林志玲);

System.out.println("林心如：" + 林心如);

System.out.println("舒淇：" + 舒淇);

System.out.println("底牌：");

/\* System.out.println(box.get(box.size() - 1));

System.out.println(box.get(box.size() - 2));

System.out.println(box.get(box.size() - 3));\*/

for (int i = box.size() - 3; i < box.size(); i++) {

System.out.println(box.get(i));

}

}

}

# HashMap集合

## Map接口概述

我们通过查看Map接口描述，发现Map接口下的集合与Collection接口下的集合，它们存储数据的形式不同，如下图。

A:Collection中的集合，元素是孤立存在的（理解为单身），向集合中存储元素采用一个个元素的方式存储

B:Map中的集合，元素是成对存在的(理解为夫妻)。每个元素由键与值两部分组成，通过键可以找对所对应的值。

C:Collection中的集合称为单列集合，Map中的集合称为双列集合。

需要注意的是，Map中的集合不能包含重复的键，值可以重复；每个键只能对应一个值。

## Map常用功能

A:映射功能：

V put(K key, V value) :以键=值的方式存入Map集合

B:获取功能：

V get(Object key):根据键获取值

int size():返回Map中键值对的个数

C:判断功能：

boolean containsKey(Object key):判断Map集合中是否包含键为key的键值对

boolean containsValue(Object value):判断Map集合中是否包含值为value键值对

boolean isEmpty():判断Map集合中是否没有任何键值对

D:删除功能：

void clear():清空Map集合中所有的键值对

V remove(Object key):根据键值删除Map中键值对

E:遍历功能：

Set<Map.Entry<K,V>> entrySet():将每个键值对封装到一个个Entry对象中,再把所有Entry的对象封装到Set集合中返回

Set<K> keySet() :将Map中所有的键装到Set集合中返回

Collection<V> values():返回集合中所有的value的值的集合

### 案例代码四:

**package** com.itheima\_01;

**import** java.util.HashMap;

**import** java.util.Map;

/\*

\* Map的常用功能：

\* 映射功能：

\* V put(K key, V value)

\* 获取功能：

\* V get(Object key)

\* int size()

\* 判断功能：

\* boolean containsKey(Object key)

boolean containsValue(Object value)

boolean isEmpty()

\* 删除功能：

\* void clear()

\* V remove(Object key)

\*

\* 遍历功能：

\* Set<Map.Entry<K,V>> entrySet()

\*

\*

\* Set<K> keySet()

\* Collection<V> values()

\*/

**public** **class** MapDemo2 {

**public** **static** **void** main(String[] args) {

//创建Map对象

Map<String,String> map = **new** HashMap<String,String>();

//V put(K key, V value) ：就是将key映射到value，如果key存在，则覆盖value，并将原来的value返回

System.*out*.println(map.put("ITCAST001", "张三"));

System.*out*.println(map.put("ITCAST002", "李四"));

System.*out*.println(map.put("ITCAST001", "王五"));

//void clear() : 清空所有的对应关系

//map.clear();

//V remove(Object key) :根据指定的key删除对应关系，并返回key所对应的值，如果没有删除成功则返回null

//System.out.println(map.remove("ITCAST005"));

//boolean containsKey(Object key) : 判断指定key是否存在

//System.out.println(map.containsKey("ITCAST003"));

//boolean containsValue(Object value)：判断指定的value是否存在

//System.out.println(map.containsValue("王五"));

//boolean isEmpty() : 判断是否有对应关系

//System.out.println(map.isEmpty());

//int size() : 返回对应关系的个数

//System.out.println(map.size());

//V get(Object key) : 根据指定的key返回对应的value

System.*out*.println(map.get("ITCAST002"));

System.*out*.println(map);

}

}

### 案例代码五:

**package** com.itheima\_01;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

/\*

\* Set<K> keySet()

\* Collection<V> values()

\*/

**public** **class** MapDemo3 {

**public** **static** **void** main(String[] args) {

//创建Map对象

Map<String,String> map = **new** HashMap<String,String>();

//添加映射关系

map.put("ITCAST001", "张三");

map.put("ITCAST002", "李四");

map.put("ITCAST005", "李四");

//Set<K> keySet() : 以Set的形式获返回所有的key

Set<String> keys = map.keySet();

**for** (String key : keys) {

System.*out*.println(key);

}

System.*out*.println("-----------");

//Collection<V> values() :

Collection<String> values = map.values();

**for** (String value : values) {

System.*out*.println(value);

}

}

}

## Map的两种遍历方式

### 利用keySet()方法遍历

**package** com.itheima\_01;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

/\*

\* Map的第一种遍历方式：

\* 首先召集所有的丈夫

\* 遍历所有的丈夫

\* 获取每一个丈夫

\* 让每一个丈夫去找他自己的媳妇

\*/

**public** **class** MapDemo4 {

**public** **static** **void** main(String[] args) {

//创建Map对象

Map<String,String> map = **new** HashMap<String,String>();

//添加映射关系

map.put("谢婷疯", "张箔纸");

map.put("陈关西", "钟欣桶");

map.put("李亚碰", "王飞");

//遍历Map对象

//首先召集所有的丈夫

Set<String> keys = map.keySet();

//遍历所有的丈夫

**for** (String key : keys) {

//让每个丈夫去找他自己的媳妇就可以了

String value = map.get(key);

System.*out*.println("丈夫：" + key + "---" + "媳妇：" + value);

}

}

}

### 利用entrySet()方法遍历

**package** com.itheima\_01;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

/\*

\* Map的第二种遍历方式：

\* 通过结婚证对象来获取丈夫和媳妇

\*

\* class 结婚证<K,V> {

\* K 丈夫;

\* V 媳妇;

\*

\* public 结婚证(K 丈夫，V 媳妇) {

\* this.丈夫 = 丈夫;

\* this.媳妇 = 媳妇;

\* }

\*

\*

\* public K get丈夫() {

\* return 丈夫;

\* }

\*

\* public V get媳妇() {

\* return 媳妇;

\* }

\* }

\*

\*

\* class Entry<K,V> {

\* K key;

\* V value;

\*

\* public Entry(K key，V value) {

\* this.key = key;

\* this.value = value;

\* }

\*

\*

\* public K getKey() {

\* return key;

\* }

\*

\* public V getValue() {

\* return value;

\* }

\* }

\*

\* Set<Map.Entry<K,V>> entrySet()

\*

\*/

**public** **class** MapDemo5 {

**public** **static** **void** main(String[] args) {

//创建Map对象

Map<String,String> map = **new** HashMap<String,String>();

//添加映射关系

map.put("尹志平", "小龙女");

map.put("令狐冲", "东方菇凉");

map.put("玄慈", "叶二娘");

//获取所有的结婚证对象

Set<Map.Entry<String,String>> entrys = map.entrySet();

//遍历包含了结婚证对象的集合

**for** (Map.Entry<String, String> entry : entrys) {

//获取每个单独的结婚证对象

//通过结婚证对象获取丈夫和媳妇

String key = entry.getKey();

String value = entry.getValue();

System.*out*.println("丈夫：" + key + "---" + "媳妇:" + value);

}

}

}

### 练习:

A:以key为学号字符串,value为学生姓名存入HashMap集合,分别以两种方式遍历集合

#### 案例代码一:

**package** com.itheima\_02;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Set;

/\*

\*

\* 使用HashMap存储数据并遍历（字符串作为key）

\*

\*/

**public** **class** HashMapDemo {

**public** **static** **void** main(String[] args) {

//创建Map对象

HashMap<String,String> hm = **new** HashMap<String,String>();

//添加映射关系

hm.put("ITCAST001", "张三");

hm.put("ITCAST002", "李四");

hm.put("ITCAST003", "王五");

hm.put("ITCAST003", "赵六");

//遍历Map对象

//方式1 获取所有的key，通过key来获取value

Set<String> keys = hm.keySet();

**for** (String key : keys) {

String value = hm.get(key);

System.*out*.println(key + "=" + value);

}

System.*out*.println("------------------");

//方式2：获取所有的结婚证对象，然后通过结婚证对象获取丈夫和媳妇

Set<Map.Entry<String, String>> entrys = hm.entrySet();

**for** (Map.Entry<String, String> entry : entrys) {

String key = entry.getKey();

String value = entry.getValue();

System.*out*.println(key + "=" + value);

}

}

}

B: 定义一个学生类,学生类中有name和age两个属性,创建三个学生对象,分别对name和age赋值,然后以key为学生对象,value为学生的学号的方式存入HashMap集合,利用两种方式遍历这个Map

#### 案例代码二:

**package** com.itheima\_02;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Set;

/\*

\*

\* 使用HashMap存储数据并遍历（自定义对象作为key）

\*

\*/

**public** **class** HashMapDemo2 {

**public** **static** **void** main(String[] args) {

//创建Map对象

HashMap<Student,String> hm = **new** HashMap<Student,String>();

//创建key对象

Student s = **new** Student("zhangsan",18);

Student s2 = **new** Student("lisi",20);

Student s3 = **new** Student("lisi",20);

//添加映射关系

hm.put(s, "ITCAST001");

hm.put(s2, "ITCAST002");

hm.put(s3, "ITCAST002");

//遍历Map对象

//方式1： 获取所有的key，通过key来获取value

Set<Student> keys = hm.keySet();

**for** (Student key : keys) {

String value = hm.get(key);

System.*out*.println(key + "=" + value);

}

System.*out*.println("-----");

//方式2：获取所有结婚证对象，通过结婚证对象获取丈夫和媳妇

Set<Map.Entry<Student, String>> entrys = hm.entrySet();

**for** (Entry<Student, String> entry : entrys) {

Student key = entry.getKey();

String value = entry.getValue();

System.*out*.println(key + "=" + value);

}

}

}