Java 核心技术

第十章 Java数据结构

第四节 集合Set

华东师范大学 陈良育

集合(1)

- 集合 Set
  - 确定性：对任意对象都能判定其是否属于某一个集合
  - 互异性：集合内每个元素都是无差异的，注意是内容差异
  - 无序性：集合内的顺序无关

集合(2)

- Java中的集合接口Set
  - HashSet （基于散列函数的集合，无序，不支持同步）
  - TreeSet (基于树结构的集合，可排序的，不支持同步)
  - LinkedHashSet(基于散列函数和双向链表的集合，可排序的，不支持同步)

集合(3)

- HashSet
  - 基于HashMap实现的，可以容纳null元素，不支持同步
    - Set s = Collections.synchronizedSet(new HashSet(...));
  - add 添加一个元素
  - clear 清除整个HashSet
  - contains 判定是否包含一个元素
  - remove 删除一个元素　　size 大小
  - retainAll 计算两个集合交集
  - 查看HashSetTest.java了解其基本用法

集合(4)

- LinkedHashSet
  - 继承HashSet，也是基于HashMap实现的，可以容纳null元素
  - 不支持同步
    - Set s = Collections.synchronizedSet(new LinkedHashSet(...));
  - 方法和HashSet基本一致
    - add, clear, contains, remove, size
  - 通过一个双向链表维护插入顺序
  - 查看LinkedHashSetTest.java了解其基本用法

集合(5)

- TreeSet
  - 基于TreeMap实现的，不可以容纳null元素，不支持同步
    - SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
  - add 添加一个元素
  - clear 清除整个TreeSet
  - contains 判定是否包含一个元素
  - remove 删除一个元素　size 大小
  - 根据compareTo方法或指定Comparator排序
  - 查看TreeSetTest.java了解其基本用法

集合(6)

- HashSet, LinkedHashSet, TreeSet的元素都只能是对象
- HashSet和LinkedHashSet判定元素重复的原则
  - 判定两个元素的hashCode返回值是否相同，若不同，返回false
  - 若两者hashCode相同，判定equals方法，若不同，返回false；否则返回true。
  - hashCode和equals方法是所有类都有的，因为Object类有
- TreeSet判定元素重复的原则
  - 需要元素继承自Comparable接口
  - 比较两个元素的compareTo方法

集合(7)

- 总结
  - HashSet、LinkedHashSet、TreeSet
  - 注意不同Set判定元素重复的原则

代码(1) HashSetTest.java

```
public class HashSetTest {
    public static void main(String[] args) {
        HashSet<Integer> hs = new HashSet<Integer>();
        hs.add(null);
        hs.add(1000);
        hs.add(20);
        hs.add(3);
        hs.add(40000);
        hs.add(5000000);
        hs.add(3);                          //3 重复
        hs.add(null);                       //null重复
        System.out.println(hs.size());  //6
        if(!hs.contains(6))
        {
            hs.add(6);
        }
        System.out.println(hs.size());  //7
        hs.remove(4);
        System.out.println(hs.size());  //6
        //hs.clear();
        //System.out.println(hs.size());  //0

        System.out.println("============for循环遍历==============");
        for(Integer item : hs)
        {
            System.out.println(item);
        }
```

代码(2) HashSetTest.java


```
        System.out.println("==============测试集合交集==============");

        HashSet<String> set1 = new HashSet<String>();
        HashSet<String> set2 = new HashSet<String>();

        set1.add("a");
        set1.add("b");
        set1.add("c");

        set2.add("c");
        set2.add("d");
        set2.add("e");

        //交集
        set1.retainAll(set2);
        System.out.println("交集是 "+set1);

        System.out.println("==============测试多种遍历方法速度==============");

        HashSet<Integer> hs2 = new HashSet<Integer>();
        for(int i=0;i<100000;i++)    {
            hs2.add(i);
        }
        traverseByIterator(hs2);
        traverseByFor(hs2);
    }
```

代码(3) HashSetTest.java


```
    public static void traverseByIterator(HashSet<Integer> hs)
    {
        long startTime = System.nanoTime();
        System.out.println("=============迭代器遍历==============");
        Iterator<Integer> iter1 = hs.iterator();
        while(iter1.hasNext()){
            iter1.next();
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println(duration + "纳秒");
    }
    public static void traverseByFor(HashSet<Integer> hs)
    {
        long startTime = System.nanoTime();
        System.out.println("=============for循环遍历==============");
        for(Integer item : hs)
        {
            ;
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println(duration + "纳秒");
    }
}
```

代码(4) LinkedHashSet.java


```
public class LinkedHashSetTest {
    public static void main(String[] args) {
        LinkedHashSet<Integer> lhs = new LinkedHashSet<Integer>();
        lhs.add(null);
        lhs.add(1000);
        lhs.add(20);
        lhs.add(3);
        lhs.add(40000);
        lhs.add(5000000);
        lhs.add(3);                          //3 重复
        lhs.add(null);                       //null 重复
        System.out.println(lhs.size());  //6
        if(!lhs.contains(6))
        {
            lhs.add(6);
        }
        System.out.println(lhs.size());  //7
        lhs.remove(4);
        System.out.println(lhs.size());  //6
        //lhs.clear();
        //System.out.println(lhs.size());  //0
```

代码(5) LinkedHashSet.java


```
        System.out.println("============for循环遍历==============");
        for(Integer item : lhs)
        {
            System.out.println(item);
        }

        LinkedHashSet<Integer> lhs2 = new LinkedHashSet<Integer>();
        for(int i=0;i<100000;i++)
        {
            lhs2.add(i);
        }
        traverseByIterator(lhs2);
        traverseByFor(lhs2);

}
```

代码(6) LinkedHashSet.java


```
    public static void traverseByIterator(LinkedHashSet<Integer> hs)
    {
        long startTime = System.nanoTime();
        System.out.println("==============迭代器遍历==============");
        Iterator<Integer> iter1 = hs.iterator();
        while(iter1.hasNext()){
            iter1.next();
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println(duration + "纳秒");
    }
    public static void traverseByFor(LinkedHashSet<Integer> hs)
    {
        long startTime = System.nanoTime();
        System.out.println("==============for循环遍历==============");
        for(Integer item : hs)
        {
            ;
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println(duration + "纳秒");
    }
}
```

代码(7) TreeSetTest.java


```
public class TreeSetTest {
    public static void main(String[] args) {
        TreeSet<Integer> ts = new TreeSet<Integer>();
        // ts.add(null);   错误，不支持null
        ts.add(1000);
        ts.add(20);
        ts.add(3);
        ts.add(40000);
        ts.add(5000000);
        ts.add(3);                                  //3 重复
        System.out.println(ts.size());   //5
        if(!ts.contains(6))
        {
            ts.add(6);
        }
        System.out.println(ts.size());   //6
        ts.remove(4);
        System.out.println(ts.size());   //5
        //lhs.clear();
        //System.out.println(lhs.size());   //0
```

代码(8) TreeSetTest.java


```
        System.out.println("============for循环遍历==============");
        for(Integer item : ts)
        {
            System.out.println(item);
        }

        TreeSet<Integer> ts2 = new TreeSet<Integer>();
        for(int i=0;i<100000;i++)
        {
            ts2.add(i);
        }
        traverseByIterator(ts2);
        traverseByFor(ts2);

}
```

代码(9) TreeSetTest.java


```
    public static void traverseByIterator(TreeSet<Integer> hs)
    {
        long startTime = System.nanoTime();
        System.out.println("==============迭代器遍历==============");
        Iterator<Integer> iter1 = hs.iterator();
        while(iter1.hasNext()){
            iter1.next();
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println(duration + "纳秒");
    }
    public static void traverseByFor(TreeSet<Integer> hs)
    {
        long startTime = System.nanoTime();
        System.out.println("==============for循环遍历==============");
        for(Integer item : hs)
        {
            ;
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println(duration + "纳秒");
    }

}
```

代码(10) Cat.java && Tiger.java


```java
class Cat
{
    private int size;

    public Cat(int size)
    {
        this.size = size;
    }
}
```

```java
public class Tiger implements Comparable{
    private int size;

    public Tiger(int s) {
        size = s;
    }

    public int getSize() {
        return size;
    }

    public int compareTo(Object o) {
        System.out.println("Tiger compareTo()~~~~~~~~~~");
        return size - ((Tiger) o).getSize();
    }
}
```

代码(11) Dog.java

```
class Dog {
    private int size;

    public Dog(int s) {
        size = s;
    }
    public int getSize() {
        return size;
    }

    public boolean equals(Object obj2)    {
        System.out.println("Dog equals()~~~~~~~~~~~~~~");
        if(0==size - ((Dog) obj2).getSize()) {
            return true;
        } else {
            return false;
        }
    }

    public int hashCode() {
        System.out.println("Dog hashCode()~~~~~~~~~~~~~");
        return size;
    }

    public String toString() {
        System.out.print("Dog toString()~~~~~~~~~~~~~");
        return size + "";
    }
}
```

代码(12) HashSetJudgeRuleTest.java


```java
public class HashSetJudgeRuleTest {

    public static void main(String[] args) {
        HashSet<Cat> hs = new HashSet<Cat>();
        hs.add(new Cat(1));
        hs.add(new Cat(2));
        hs.add(new Cat(3));
        hs.add(new Cat(3));
        System.out.println(hs.size()); //4
    }
}
```

代码(13) ObjectHashSetTest.java


```
public class ObjectHashSetTest {

    public static void main(String[] args) {
        System.out.println("==========Cat HashSet ==============");
        HashSet<Cat> hs = new HashSet<Cat>();
        hs.add(new Cat(2));
        hs.add(new Cat(1));
        hs.add(new Cat(3));
        hs.add(new Cat(5));
        hs.add(new Cat(4));
        hs.add(new Cat(4));
        System.out.println(hs.size());  //6

        System.out.println("============================");
        LinkedHashSet<Cat> lhs= new LinkedHashSet<Cat>();
        lhs.add(new Cat(2));
        lhs.add(new Cat(1));
        lhs.add(new Cat(3));
        lhs.add(new Cat(5));
        lhs.add(new Cat(4));
        lhs.add(new Cat(4));
        System.out.println(lhs.size());  //6
```

代码(14) ObjectHashSetTest.java


```
System.out.println("==========Dog HashSet ==============");
HashSet<Dog> hs2 = new HashSet<Dog>();
hs2.add(new Dog(2));
hs2.add(new Dog(1));
hs2.add(new Dog(3));
hs2.add(new Dog(5));
hs2.add(new Dog(4));
hs2.add(new Dog(4));
System.out.println(hs2.size());   //5

System.out.println("============================");
LinkedHashSet<Dog> lhs2= new LinkedHashSet<Dog>();
lhs2.add(new Dog(2));
lhs2.add(new Dog(1));
lhs2.add(new Dog(3));
lhs2.add(new Dog(5));
lhs2.add(new Dog(4));
lhs2.add(new Dog(4));
System.out.println(lhs2.size());   //5
```

代码(15) ObjectHashSetTest.java


```
        System.out.println("==========Tiger HashSet ==============");
        HashSet<Tiger> hs3 = new HashSet<Tiger>();
        hs3.add(new Tiger(2));
        hs3.add(new Tiger(1));
        hs3.add(new Tiger(3));
        hs3.add(new Tiger(5));
        hs3.add(new Tiger(4));
        hs3.add(new Tiger(4));
        System.out.println(hs3.size());  //6

        System.out.println("==========================");
        LinkedHashSet<Tiger> lhs3= new LinkedHashSet<Tiger>();
        lhs3.add(new Tiger(2));
        lhs3.add(new Tiger(1));
        lhs3.add(new Tiger(3));
        lhs3.add(new Tiger(5));
        lhs3.add(new Tiger(4));
        lhs3.add(new Tiger(4));
        System.out.println(lhs3.size());  //6
    }
}
```

代码(16) ObjectTreeSetTest.java


```
public class ObjectTreeSetTest {

    public static void main(String[] args) {

        //添加到TreeSet的，需要实现Comparable接口，即实现compareTo方法

        System.out.println("==========Tiger TreeSet ==============");


        TreeSet<Tiger> ts3 = new TreeSet<Tiger>();
        ts3.add(new Tiger(2));
        ts3.add(new Tiger(1));
        ts3.add(new Tiger(3));
        ts3.add(new Tiger(5));
        ts3.add(new Tiger(4));
        ts3.add(new Tiger(4));
        System.out.println(ts3.size());  //5
    }

}
```

谢 谢！