

set

1. HashSet

HashSet 集合，没有索引，存储和读取是无序的，存储的值不能重复，当存储的值是重复之时，返回false。

HashSet 可以用迭代器迭代，自然也可以用增强for循环

2. HashSet存储自定义对象，和不能存储重复值得原理

利用HashSet 的add方法存储值得时候，需要去除重复，如果在存储的时候逐个进行比较，效率比较低，这是运用哈希算法提高效率，降低equals()方法被调用的次数。

当调用add方法的时候，先调用对象的HashCode方法，得到一个hash值，然后和集合中的哈希值比较

如果不同：就直接存入集合

如果相同：就调用equals方法，相同哈希值的对象逐个比较。

因此：

当自定义对象去除重复时，

必须重写HashCode 和 equals方法，

hashCode：属性相同返回值相同，属性不同 尽量不同 不能排除巧合存在

equals：属性相同，true；不同 false

hashCode的原理：源码如下

```
* 为什么是31?
* 1, 31是一个质数, 质数是能被1和自己本身整除的数
* 2, 31这个数既不大也不小
* 3, 31这个数好算, 2的五次方-1, 2向左移动5位
*/
@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;
}
```

equals源码：

```
public boolean equals(Object obj) {
    if (this == obj) //调用的对象和传入的对象是同一个对象
        return true; //直接返回true
    if (obj == null) //传入的对象为null
        return false; //返回false
    if (getClass() != obj.getClass()) //判断两个对象对应的字节码文件是否是同一个字节码
        return false; //如果不是直接返回false
    Person other = (Person) obj; //向下转型
    if (age != other.age) //调用对象的年龄不等于传入对象的年龄
        return false; //返回false
    if (name == null) { //调用对象的姓名为null
        if (other.name != null) //传入对象的姓名不为null
            return false; //返回false
    } else if (!name.equals(other.name)) //调用对象的姓名不等于传入对象的姓名
        return false; //返回false
    return true;
}
```

3.hashset子类LinkedList()

```
/**
 * @param args
 * LinkedHashSet
 * 底层是链表实现的, 是set集合中唯一一个能保证怎么存就怎么取的集合对象
 * 因为是HashSet的子类, 所以也是保证元素唯一的, 与HashSet的原理一样
 */
public static void main(String[] args) {
    LinkedHashSet<String> lhs = new LinkedHashSet<>();
    lhs.add("a");
    lhs.add("a");
    lhs.add("a");
    lhs.add("a");
    lhs.add("b");
}
```

4.treeSet

treeSet 用于排序，同样也可以保证元素唯一。

存储自定义对象来发现 排序的原因：

自定义必须实现comparable接口重写 compareTo方法：

```

/
 * @param args
 * TreeSet集合是用来对元素进行排序的, 同样他也可以保证元素的唯一
 * 当compareTo方法返回0的时候集合中只有一个元素
 * 当compareTo方法返回正数的时候集合会怎么存就怎么取
 * 当compareTo方法返回负数的时候集合会倒序存储
 */
public static void main(String[] args) {

```

存储自定义对象Person () ;

有姓名 和年龄 属性, 要求按姓名长度排序: 那么compareTo方法的重写如下:

首先主要条件: 姓名的长度: 如果长度小于则返回负数, 大于则整数, 等于就返回0, 返回0这个值将不会存储, 但是姓名的值可能不同, 所以不但要比较长度, 还需要比较姓名的值, 同样的如果姓名值相同, 那么还需要比较年龄的值。

```

public int compareTo(Person o) {
    int length = this.name.length() - o.name.length(); //比较长度为主要条件
    int num = length == 0 ? this.name.compareTo(o.name) : length; //比较内容为次要条件
    return num == 0 ? this.age - o.age : num; //比较年龄为次要条件
}

```

treeSet 存储原理:

向treeSet中存储数据时, 会调用compareTo方法, 进行比较,

如果compareTo返回值是0

The screenshot shows a Java IDE with the following code in the editor:

```

TreeSet<Person> ts = new TreeSet<>();
ts.add(new Person("张三", 23));
ts.add(new Person("李四", 13));
ts.add(new Person("王五", 43));
ts.add(new Person("赵六", 33));

System.out.println(ts);

```

To the right of the code is a binary tree diagram illustrating the storage of these objects:

- Root node: (new Person("张三", 23))
- Left child of root: (new Person("李四", 13))
- Right child of root: (new Person("王五", 43))
- Left child of (new Person("王五", 43)): (new Person("赵六", 33))

Below the diagram, there is a list of rules for TreeSet storage:

- 二叉树: 两个叉
- 小的存储在左边(负数), 大的存储在右边(正数), 相等就不存(0)
- compareTo方法, 在TreeSet集合如何存储元素取决于compareTo方法的返回值

Then, there is a list of rules for compareTo:

1. 返回0, 集合中只有一个元素
2. 返回-1 集合会将存储的元素倒序
3. 返回1 集合会怎么存就怎么取

Finally, there is a code snippet for the compareTo method:

```

@Override
public int compareTo(Person o) {
    return this.age - o.age;
}

```

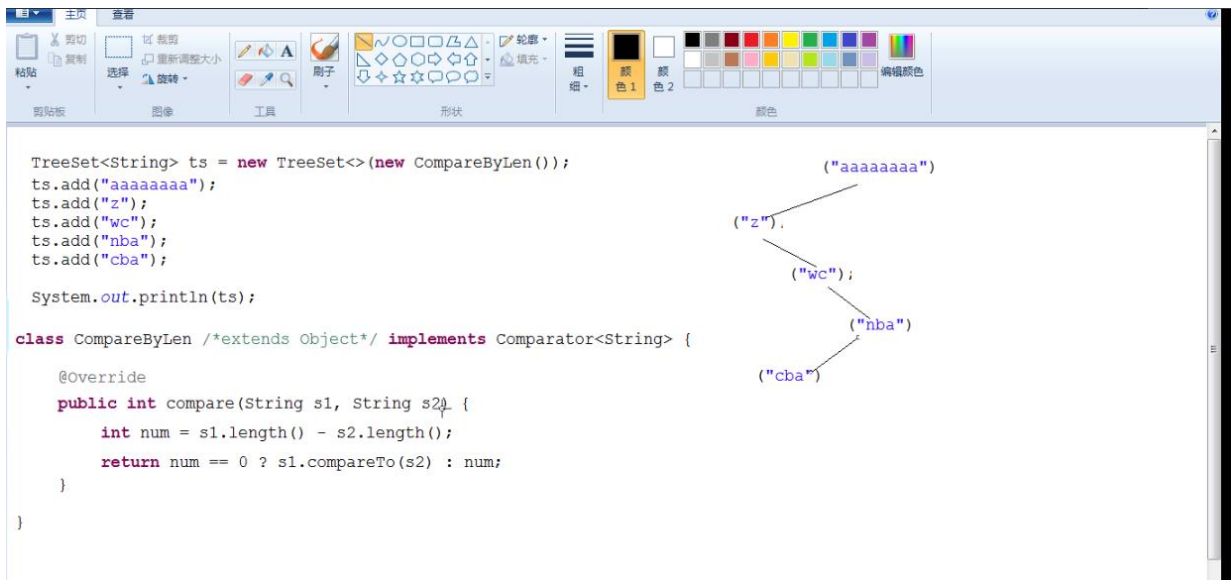
treeSet比较器排序: 需要传入构造函数

```

6 ###17.15 集合框架(TreeSet原理)
7 * 1. 特点
8 * TreeSet是用来排序的, 可以指定一个顺序, 对象存入之后会按照指定的顺序排列
9 * 2. 使用方式
10 * a. 自然顺序(Comparable)
11 * TreeSet类的add()方法中会把存入的对象提升为Comparable类型
12 * 调用对象的compareTo()方法和集合中的对象比较
13 * 根据compareTo()方法返回的结果进行存储
14 * b. 比较器顺序(Comparator)
15 * 创建TreeSet的时候可以制定一个Comparator
16 * 如果传入了Comparator的子类对象, 那么TreeSet就会按照比较器中的顺序排序
17 * add()方法内部会自动调用Comparator接口中compare()方法排序
18 * c. 两种方式的区别
19 * TreeSet构造函数什么都不传, 默认按照类中Comparable的顺序(没有就报错ClassCastException)
20 * TreeSet如果传入Comparator, 就优先按照Comparator
21

```

比较器: String实现了comparable接口



treeSet 练习一：

有arraylist，把其中的值进行排序，并且不能去掉重复值。

```

* 1, 创建TreeSet集合对象, 因为String本身就具备比较功能, 但是重复不会保留, 所以我们用比较器
* 2, 将list集中所有的元素添加到TreeSet集合中, 对其排序, 保留重复
* 3, 清空list集合
* 4, 将TreeSet集合中排好序的元素添加到list中
*/
public static void sort(List<String> list) {
    //1, 创建TreeSet集合对象, 因为String本身就具备比较功能, 但是重复不会保留, 所以我们用比较器
    TreeSet<String> ts = new TreeSet<>(new Comparator<String>() {

        @Override
        public int compare(String s1, String s2) {
            int num = s1.compareTo(s2);
            return num == 0 ? 1 : num;
        }
    });
    //2, 将list集中所有的元素添加到TreeSet集合中, 对其排序, 保留重复
    ts.addAll(list);
    //3, 清空list集合
    list.clear();
}

```

package treeset;

```

import java.util.ArrayList;
import java.util.Comparator;
import java.util.TreeSet;

```

public class treeset01 {

```

    public static void main(String[] args) {
        ArrayList<String> list=new ArrayList<>();
        list.add("a");
        list.add("b");
        list.add("c");
        list.add("a");
        list.add("b");
        list.add("c");
        list.add("a");
    }

```

```

        sort(list);
        System.out.println(list);
    }

```

```

    /*
    * 1.创建一个treeset 集合对象, 因为string本身重写了comparable的compareTo方法, 因此会去除重复
    * 2.把list中的值写入treeset并排序, 但是需要传入自定义排序器
    * 3.清空list
    * 4.treeset回填list
    */

```

```

    private static void sort(ArrayList<String> list) {
        TreeSet<String> tree=new TreeSet<>(new Comparator<String>() {

```

```

@Override
public int compare(String o1, String o2) {
    //对姓名进行比较
    int number=o1.compareTo(o2);
    //如果相同 不能返回0 , 否则去重了
    return number==0 ? 1 :number;
}

});
tree.addAll(list);

//清空 list
list.clear();
//回填
list.addAll(tree);
}

```

treeSet练习二：

从键盘输入字符串，对其进行排序，并且不去除重复

```

/**
 * 从键盘接收一个字符串，程序对其中所有字符进行排序，例如键盘输入：helloitcast程序打印：acehilllostt
 * 分析：
 * 1, 键盘录入字符串, Scanner
 * 2, 将字符串转换为字符数组
 * 3, 定义TreeSet集合, 传入比较器对字符排序并保留重复
 * 4, 遍历字符数组, 将每一个字符存储在TreeSet集合中
 * 5, 遍历TreeSet集合, 打印每一个字符
 */

```

```

package treeset;

import java.util.Comparator;
import java.util.Scanner;
import java.util.TreeSet;

import javax.swing.plaf.synth.SynthStyle;

public class treeset02 {
    public static void main(String[] args) {
        //输入字符串
        Scanner sca=new Scanner(System.in);
        //将字符串转换为字符存储在字符数组里
        String str =sca.nextLine();
        char[] arr=str.toCharArray();
        //定义自定义比较器进行字符排序
        TreeSet<Character> tree=new TreeSet<>(new Comparator<Character>() {

            @Override
            public int compare(Character o1, Character o2) {
                int number =o1.compareTo(o2);
                return number==0 ? 1:number;
            }
        });

        //遍历赋值给tree
        for (Character character :arr) {
            tree.add(character);
        }
        //遍历打印
        for (Character character : tree) {
            System.out.print(character);
        }

    }
}
练习三：

```

键盘输入五个学生信息（姓名，语文成绩，数学成绩，英语成绩）按照总分进行排序

4. 案例演示

```
* 需求：键盘录入5个学生信息（姓名，语文成绩，数学成绩，英语成绩），按照总分从高到低输出到控制台。  
*  
* 分析：  
* 1, 定义一个学生类  
*     成员变量：姓名，语文成绩，数学成绩，英语成绩，总成绩  
*     成员方法：空参，有参构造，有参构造的参数分别是姓名，语文成绩，数学成绩，英语成绩  
*             toString方法，在遍历集合中的Student对象打印对象引用的时候会显示属性值  
* 2, 键盘录入需要Scanner，创建键盘录入对象  
* 3, 创建TreeSet集合对象，在TreeSet的构造函数中传入比较器，按照总分比较  
* 4, 录入五个学生，所以以集合中的学生个数为判断条件，如果size是小于5就进行存储  
* 5, 将录入的字符串切割，用逗号切割，会返回一个字符串数组，将字符串数组中从二个元素转换成int数，  
* 6, 将转换后的结果封装成Student对象，将Student添加到TreeSet集合中  
* 7, 遍历TreeSet集合打印每一个Student对象  
*/
```

```
package treeset;
```

```
import java.util.Comparator;
```

```
import java.util.Scanner;
```

```
import java.util.TreeSet;
```

```
public class treeset03 {
```

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

```
        //键盘录入 格式姓名，语文，数学，英语
```

```
        System.out.println("输入信息");
```

```
        Scanner sac=new Scanner(System.in);
```

```
        //新建treeset
```

```
        TreeSet<student> tree=new TreeSet<> (new Comparator<student>() {
```

```
            @Override
```

```
            public int compare(student o1, student o2) {
```

```
                //降序排列
```

```
                int num=o2.getSum()-(o1.getSum());
```

```
                return num==0 ? 1: num;
```

```
            }
```

```
        });
```

```
        //循环遍历
```

```
        while (tree.size()<5) {
```

```
            String line=sac.nextLine();
```

```
            String[] arr=line.split(",");
```

```
            int chin=Integer.parseInt(arr[1]);
```

```
            int ma=Integer.parseInt(arr[2]);
```

```
            int eng=Integer.parseInt(arr[3]);
```

```
            //转换后的结果复制给tree
```

```
            tree.add(new student(arr[0],chin,ma,eng));
```

```
        }
```

```
        System.out.println("排序后的信息");
```

```
        for (student student : tree) {
```

```
            System.out.println(student);
```

```
        }
```

```
    }
```

```
}
```