1.泛型

1.1泛型概述

- 泛型的介绍

  泛型是JDK5中引入的特性，它提供了编译时类型安全检测机制

- 泛型的好处

  1. 把运行时期的问题提前到了编译期间
  2. 避免了强制类型转换

- 泛型的定义格式

  - <类型>: 指定一种类型的格式.尖括号里面可以任意书写,一般只写一个字母.例如:
  - <类型1,类型2...>: 指定多种类型的格式,多种类型之间用逗号隔开.例如: <E,T> <K,V>

2.Set集合

2.1Set集合概述和特点【应用】

- 不可以存储重复元素
- 没有索引,不能使用普通for循环遍历

2.2Set集合的使用【应用】

存储字符串并遍历

```
public class MySet1 {
    public static void main(String[] args) {
        //创建集合对象
        Set<String> set = new TreeSet<>();
        //添加元素
        set.add("ccc");
        set.add("aaa");
        set.add("aaa");
        set.add("bbb");

//        for (int i = 0; i < set.size(); i++) {
//            //Set集合是没有索引的，所以不能使用通过索引获取元素的方法
//        }

        //遍历集合
        Iterator<String> it = set.iterator();
        while (it.hasNext()){
            String s = it.next();
            System.out.println(s);
        }
        System.out.println("----------------------------------");
```

```
        for (String s : set) {
            System.out.println(s);
        }
    }
}
```

3.TreeSet集合

3.1TreeSet集合概述和特点【应用】

- 不可以存储重复元素
- 没有索引
- 可以将元素按照规则进行排序
    - TreeSet()：根据其元素的自然排序进行排序
    - TreeSet(Comparator comparator)： 根据指定的比较器进行排序

3.2TreeSet集合基本使用【应用】

存储Integer类型的整数并遍历

```
public class TreeSetDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Integer> ts = new TreeSet<Integer>();

        //添加元素
        ts.add(10);
        ts.add(40);
        ts.add(30);
        ts.add(50);
        ts.add(20);

        ts.add(30);

        //遍历集合
        for(Integer i : ts) {
            System.out.println(i);
        }
    }
}
```

3.3自然排序Comparable的使用【应用】

- 案例需求
    - 存储学生对象并遍历，创建TreeSet集合使用无参构造方法
    - 要求：按照年龄从小到大排序，年龄相同时，按照姓名的字母顺序排序

- 实现步骤

    1. 使用空参构造创建TreeSet集合
        - 用TreeSet集合存储自定义对象，无参构造方法使用的是自然排序对元素进行排序的
    2. 自定义的Student类实现Comparable接口
        - 自然排序，就是让元素所属的类实现Comparable接口，重写compareTo(T o)方法
    3. 重写接口中的compareTo方法
        - 重写方法时，一定要注意排序规则必须按照要求的主要条件和次要条件来写

- 代码实现

    学生类

```
public class Student implements Comparable<Student>{
    private String name;
    private int age;

    public Student() {
    }

    public Student(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;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Student o) {
```

```
        //按照对象的年龄进行排序
        //主要判断条件：按照年龄从小到大排序
        int result = this.age - o.age;
        //次要判断条件：年龄相同时，按照姓名的字母顺序排序
        result = result == 0 ? this.name.compareTo(o.getName()) : result;
        return result;
    }
}
```

测试类

```
public class MyTreeSet2 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Student> ts = new TreeSet<>();
        //创建学生对象
        Student s1 = new Student("zhangsan",28);
        Student s2 = new Student("lisi",27);
        Student s3 = new Student("wangwu",29);
        Student s4 = new Student("zhaoliu",28);
        Student s5 = new Student("qianqi",30);
        //把学生添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);
        ts.add(s4);
        ts.add(s5);
        //遍历集合
        for (Student student : ts) {
            System.out.println(student);
        }
    }
}
```

3.4比较器排序Comparator的使用【应用】

- 案例需求
  - 存储老师对象并遍历，创建TreeSet集合使用带参构造方法
  - 要求：按照年龄从小到大排序，年龄相同时，按照姓名的字母顺序排序
- 实现步骤
  - 用TreeSet集合存储自定义对象，带参构造方法使用的是比较器排序对元素进行排序的
  - 比较器排序，就是让集合构造方法接收Comparator的实现类对象，重写compare(T o1,T o2)方法
  - 重写方法时，一定要注意排序规则必须按照要求的主要条件和次要条件来写
- 代码实现

老师类

```
public class Teacher {
    private String name;
    private int age;

    public Teacher() {
    }

    public Teacher(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;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Teacher{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

测试类

```
public class MyTreeSet4 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Teacher> ts = new TreeSet<>(new Comparator<Teacher>() {
            @Override
            public int compare(Teacher o1, Teacher o2) {
                //o1表示现在要存入的那个元素
                //o2表示已经存入到集合中的元素
```

```
                //主要条件
                int result = o1.getAge() - o2.getAge();
                //次要条件
                result = result == 0 ?
o1.getName().compareTo(o2.getName()) : result;
                return result;
            }
        });
        //创建老师对象
        Teacher t1 = new Teacher("zhangsan",23);
        Teacher t2 = new Teacher("lisi",22);
        Teacher t3 = new Teacher("wangwu",24);
        Teacher t4 = new Teacher("zhaoliu",24);
        //把老师添加到集合
        ts.add(t1);
        ts.add(t2);
        ts.add(t3);
        ts.add(t4);
        //遍历集合
        for (Teacher teacher : ts) {
            System.out.println(teacher);
        }
    }
}
```

3.5两种比较方式总结【理解】

- 两种比较方式小结
  - 自然排序: 自定义类实现Comparable接口,重写compareTo方法,根据返回值进行排序
  - 比较器排序: 创建TreeSet对象的时候传递Comparator的实现类对象,重写compare方法,根据返回值进行排序
  - 在使用的时候,默认使用自然排序,当自然排序不满足现在的需求时,必须使用比较器排序
- 两种方式中关于返回值的规则
  - 如果返回值为负数，表示当前存入的元素是较小值，存左边
  - 如果返回值为0，表示当前存入的元素跟集合中元素重复了，不存
  - 如果返回值为正数，表示当前存入的元素是较大值，存右边

4.HashSet集合

4.1HashSet集合概述和特点【应用】

- 底层数据结构是哈希表
- 存取无序
- 不可以存储重复元素
- 没有索引,不能使用普通for循环遍历

4.2HashSet集合的基本应用【应用】

存储字符串并遍历

```
public class HashSetDemo {
    public static void main(String[] args) {
        //创建集合对象
        HashSet<String> set = new HashSet<String>();

        //添加元素
        set.add("hello");
        set.add("world");
        set.add("java");
        //不包含重复元素的集合
        set.add("world");

        //遍历
        for(String s : set) {
            System.out.println(s);
        }
    }
}
```