1.Set集合

1.1Set集合概述和特点【应用】

- Set集合的特点
  - 元素存取无序
  - 没有索引、只能通过迭代器或增强for循环遍历
  - 不能存储重复元素
- Set集合的基本使用

```
public class SetDemo {
    public static void main(String[] args) {
        //创建集合对象
        Set<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);
        }
    }
}
```

1.2哈希值【理解】

- 哈希值简介

  是JDK根据对象的地址或者字符串或者数字算出来的int类型的数值

- 如何获取哈希值

  Object类中的public int hashCode()：返回对象的哈希码值

- 哈希值的特点
  - 同一个对象多次调用hashCode()方法返回的哈希值是相同的
  - 默认情况下，不同对象的哈希值是不同的。而重写hashCode()方法，可以实现让不同对象的哈希值相同
- 获取哈希值的代码
  - 学生类

```
public class 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 int hashCode() {
        return 0;
    }
}
```

- 测试类

```
public class HashDemo {
    public static void main(String[] args) {
        //创建学生对象
        Student s1 = new Student("林青霞",30);

        //同一个对象多次调用hashCode()方法返回的哈希值是相同的
        System.out.println(s1.hashCode()); //1060830840
        System.out.println(s1.hashCode()); //1060830840
        System.out.println("--------");

        Student s2 = new Student("林青霞",30);

        //默认情况下，不同对象的哈希值是不相同的
        //通过方法重写，可以实现不同对象的哈希值是相同的
        System.out.println(s2.hashCode()); //2137211482
        System.out.println("--------");

        System.out.println("hello".hashCode()); //99162322
        System.out.println("world".hashCode()); //113318802
        System.out.println("java".hashCode()); //3254818

```

```
        System.out.println("world".hashCode()); //113318802
        System.out.println("--------");

        System.out.println("重地".hashCode()); //1179395
        System.out.println("通话".hashCode()); //1179395
    }
}
```

1.3HashSet集合概述和特点【应用】

- HashSet集合的特点
  - 底层数据结构是哈希表
  - 对集合的迭代顺序不作任何保证，也就是说不保证存储和取出的元素顺序一致
  - 没有带索引的方法，所以不能使用普通for循环遍历
  - 由于是Set集合，所以是不包含重复元素的集合
- HashSet集合的基本使用

```
public class HashSetDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        HashSet<String> hs = new HashSet<String>();

        //添加元素
        hs.add("hello");
        hs.add("world");
        hs.add("java");

        hs.add("world");

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

1.4HashSet集合保证元素唯一性源码分析【理解】

- HashSet集合保证元素唯一性的原理

  1.根据对象的哈希值计算存储位置

  如果当前位置没有元素则直接存入

  如果当前位置有元素存在，则进入第二步

  2.当前元素的元素和已经存在的元素比较哈希值

  如果哈希值不同，则将当前元素进行存储

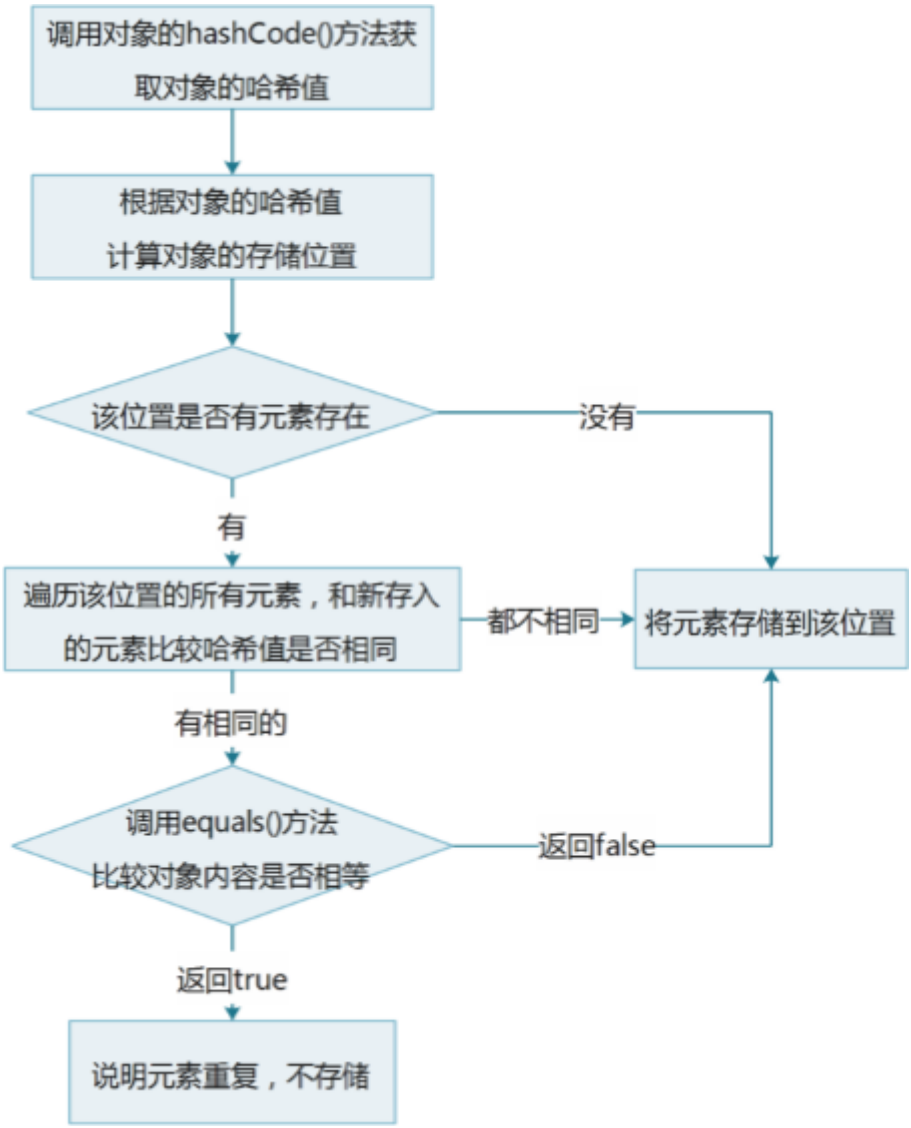
  如果哈希值相同，则进入第三步

  3.通过equals()方法比较两个元素的内容

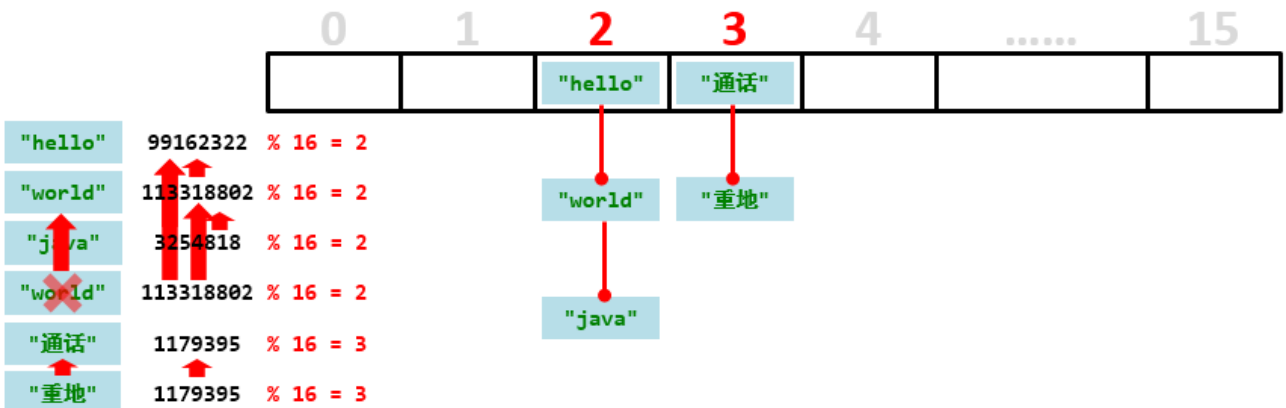
  如果内容不相同，则将当前元素进行存储

如果内容相同，则不存储当前元素

- HashSet集合保证元素唯一性的图解



1.5常见数据结构之哈希表【理解】



1.6HashSet集合存储学生对象并遍历【应用】

- 案例需求

- 创建一个存储学生对象的集合，存储多个学生对象，使用程序实现在控制台遍历该集合
- 要求：学生对象的成员变量值相同，我们就认为是同一个对象

- 代码实现
  - 学生类

```java
public class 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 boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Student student = (Student) o;

        if (age != student.age) return false;
        return name != null ? name.equals(student.name) : student.name ==
null;
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

- 测试类

```java
public class HashSetDemo02 {
    public static void main(String[] args) {
        //创建HashSet集合对象
        HashSet<Student> hs = new HashSet<Student>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);

        Student s4 = new Student("王祖贤", 33);

        //把学生添加到集合
        hs.add(s1);
        hs.add(s2);
        hs.add(s3);
        hs.add(s4);

        //遍历集合(增强for)
        for (Student s : hs) {
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

1.7LinkedHashSet集合概述和特点【应用】

- LinkedHashSet集合特点
  - 哈希表和链表实现的Set接口，具有可预测的迭代次序
  - 由链表保证元素有序，也就是说元素的存储和取出顺序是一致的
  - 由哈希表保证元素唯一，也就是说没有重复的元素
- LinkedHashSet集合基本使用

```java
public class LinkedHashSetDemo {
    public static void main(String[] args) {
        //创建集合对象
        LinkedHashSet<String> linkedHashSet = new LinkedHashSet<String>();

        //添加元素
        linkedHashSet.add("hello");
        linkedHashSet.add("world");
        linkedHashSet.add("java");

        linkedHashSet.add("world");

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

```
    }
}
```

2.Set集合排序

2.1TreeSet集合概述和特点【应用】

- TreeSet集合概述
  - 元素有序，可以按照一定的规则进行排序，具体排序方式取决于构造方法
    - 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
  - 没有带索引的方法，所以不能使用普通for循环遍历
  - 由于是Set集合，所以不包含重复元素的集合
- TreeSet集合基本使用

```
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);
        }
    }
}
```

2.2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
  - 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
  - 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
  - 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

```java
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 int compareTo(Student s) {
//        return 0;
//        return 1;
//        return -1;
        //按照年龄从小到大排序
       int num = this.age - s.age;
//        int num = s.age - this.age;
        //年龄相同时，按照姓名的字母顺序排序
       int num2 = num==0?this.name.compareTo(s.name):num;
        return num2;
    }
}
```

- 测试类

```java
public class TreeSetDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Student> ts = new TreeSet<Student>();

        //创建学生对象
        Student s1 = new Student("xishi", 29);
        Student s2 = new Student("wangzhaojun", 28);
```

```
        Student s3 = new Student("diaochan", 30);
        Student s4 = new Student("yangyuhuan", 33);

        Student s5 = new Student("linqingxia",33);
        Student s6 = new Student("linqingxia",33);

        //把学生添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);
        ts.add(s4);
        ts.add(s5);
        ts.add(s6);

        //遍历集合
        for (Student s : ts) {
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

2.3比较器排序Comparator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
  - 学生类

```
public class 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;
    }
}
```

- 测试类

```
public class TreeSetDemo {
    public static void main(String[] args) {
        //创建集合对象
        TreeSet<Student> ts = new TreeSet<Student>(new Comparator<Student>() {
            @Override
            public int compare(Student s1, Student s2) {
                //this.age - s.age
                //s1,s2
                int num = s1.getAge() - s2.getAge();
                int num2 = num == 0 ? s1.getName().compareTo(s2.getName()) : num;
                return num2;
            }
        });

        //创建学生对象
        Student s1 = new Student("xishi", 29);
        Student s2 = new Student("wangzhaojun", 28);
        Student s3 = new Student("diaochan", 30);
        Student s4 = new Student("yangyuhuan", 33);

        Student s5 = new Student("linqingxia",33);
        Student s6 = new Student("linqingxia",33);

        //把学生添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);
        ts.add(s4);
        ts.add(s5);
        ts.add(s6);

        //遍历集合
        for (Student s : ts) {
            System.out.println(s.getName() + "," + s.getAge());
        }
    }
}
```

2.4成绩排序案例【应用】

- 案例需求
  - 用TreeSet集合存储多个学生信息(姓名，语文成绩，数学成绩)，并遍历该集合
  - 要求：按照总分从高到低出现
- 代码实现
  - 学生类

```
public class Student {
    private String name;
    private int chinese;
    private int math;

    public Student() {
    }

    public Student(String name, int chinese, int math) {
        this.name = name;
        this.chinese = chinese;
        this.math = math;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getChinese() {
        return chinese;
    }

    public void setChinese(int chinese) {
        this.chinese = chinese;
    }

    public int getMath() {
        return math;
    }

    public void setMath(int math) {
        this.math = math;
    }

    public int getSum() {
        return this.chinese + this.math;
    }
}
```

- 测试类

```
public class TreeSetDemo {
    public static void main(String[] args) {
        //创建TreeSet集合对象，通过比较器排序进行排序
        TreeSet<Student> ts = new TreeSet<Student>(new Comparator<Student>() {
            @Override
            public int compare(Student s1, Student s2) {
//                int num = (s2.getChinese()+s2.getMath())-(s1.getChinese()+s1.getMath());
                //主要条件
                int num = s2.getSum() - s1.getSum();
                //次要条件
                int num2 = num == 0 ? s1.getChinese() - s2.getChinese() : num;
                int num3 = num2 == 0 ? s1.getName().compareTo(s2.getName()) : num2;
                return num3;
            }
        });

        //创建学生对象
        Student s1 = new Student("林青霞", 98, 100);
        Student s2 = new Student("张曼玉", 95, 95);
        Student s3 = new Student("王祖贤", 100, 93);
        Student s4 = new Student("柳岩", 100, 97);
        Student s5 = new Student("风清扬", 98, 98);

        Student s6 = new Student("左冷禅", 97, 99);
//        Student s7 = new Student("左冷禅", 97, 99);
        Student s7 = new Student("赵云", 97, 99);

        //把学生对象添加到集合
        ts.add(s1);
        ts.add(s2);
        ts.add(s3);
        ts.add(s4);
        ts.add(s5);
        ts.add(s6);
        ts.add(s7);

        //遍历集合
        for (Student s : ts) {
            System.out.println(s.getName() + "," + s.getChinese() + "," + s.getMath() + "," + s.getSum());
        }
    }
}
```

2.5不重复的随机数案例【应用】

- 案例需求
  - 编写一个程序，获取10个1-20之间的随机数，要求随机数不能重复，并在控制台输出
- 代码实现

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

        //创建随机数对象
        Random r = new Random();

        //判断集合的长度是不是小于10
        while (set.size()<10) {
            //产生一个随机数，添加到集合
            int number = r.nextInt(20) + 1;
            set.add(number);
        }

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

3.泛型

3.1泛型概述和好处【理解】

- 泛型概述

  是JDK5中引入的特性，它提供了编译时类型安全检测机制，该机制允许在编译时检测到非法的类型

  它的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。一提到参数，最熟悉的就是定义方法时有形参，然后调用此方法时传递实参。那么参数化类型怎么理解呢？顾名思义，就是将类型由原来的具体的类型参数化，然后在使用/调用时传入具体的类型。这种参数类型可以用在类、方法和接口中，分别被称为泛型类、泛型方法、泛型接口

- 泛型定义格式
  - <类型>：指定一种类型的格式。这里的类型可以看成是形参
  - <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。这里的类型可以看成是形参
  - 将来具体调用时候给定的类型可以看成是实参，并且实参的类型只能是引用数据类型
- 泛型的好处
  - 把运行时期的问题提前到了编译期间
  - 避免了强制类型转换

3.2泛型类【应用】

- 定义格式

```
修饰符 class 类名<类型> {  }
```

- 示例代码
  - 泛型类

```
public class Generic<T> {
    private T t;

    public T getT() {
        return t;
    }

    public void setT(T t) {
        this.t = t;
    }
}
```

  - 测试类

```
public class GenericDemo {
    public static void main(String[] args) {
        Generic<String> g1 = new Generic<String>();
        g1.setT("林青霞");
        System.out.println(g1.getT());

        Generic<Integer> g2 = new Generic<Integer>();
        g2.setT(30);
        System.out.println(g2.getT());

        Generic<Boolean> g3 = new Generic<Boolean>();
        g3.setT(true);
        System.out.println(g3.getT());
    }
}
```

3.3泛型方法【应用】

- 定义格式

```
修饰符 <类型> 返回值类型 方法名(类型 变量名) {  }
```

- 示例代码
  - 带有泛型方法的类

```
public class Generic {
    public <T> void show(T t) {
        System.out.println(t);
    }
}
```

- 测试类

```
public class GenericDemo {
    public static void main(String[] args) {
        Generic g = new Generic();
        g.show("林青霞");
        g.show(30);
        g.show(true);
        g.show(12.34);
    }
}
```

3.4泛型接口【应用】

- 定义格式

```
修饰符 interface 接口名<类型> {  }
```

- 示例代码
  - 泛型接口

```
public interface Generic<T> {
    void show(T t);
}
```

  - 泛型接口实现类

```
public class GenericImpl<T> implements Generic<T> {
    @Override
    public void show(T t) {
        System.out.println(t);
    }
}
```

  - 测试类

```
public class GenericDemo {
    public static void main(String[] args) {
        Generic<String> g1 = new GenericImpl<String>();
        g1.show("林青霞");

        Generic<Integer> g2 = new GenericImpl<Integer>();
        g2.show(30);
    }
}
```

3.5类型通配符【应用】

- 类型通配符的作用

为了表示各种泛型List的父类，可以使用类型通配符

- 类型通配符的分类
  - 类型通配符：<?>
    - List<?>：表示元素类型未知的List，它的元素可以匹配任何的类型
    - 这种带通配符的List仅表示它是各种泛型List的父类，并不能把元素添加到其中
  - 类型通配符上限：<? extends 类型>
    - List<? extends Number>：它表示的类型是Number或者其子类型
  - 类型通配符下限：<? super 类型>
    - List<? super Number>：它表示的类型是Number或者其父类型
- 类型通配符的基本使用

```
public class GenericDemo {
    public static void main(String[] args) {
        //类型通配符：<?>
        List<?> list1 = new ArrayList<Object>();
        List<?> list2 = new ArrayList<Number>();
        List<?> list3 = new ArrayList<Integer>();
        System.out.println("--------");

        //类型通配符上限：<? extends 类型>
//        List<? extends Number> list4 = new ArrayList<Object>();
        List<? extends Number> list5 = new ArrayList<Number>();
        List<? extends Number> list6 = new ArrayList<Integer>();
        System.out.println("--------");

        //类型通配符下限：<? super 类型>
        List<? super Number> list7 = new ArrayList<Object>();
        List<? super Number> list8 = new ArrayList<Number>();
//        List<? super Number> list9 = new ArrayList<Integer>();

    }
}
```

4.可变参数

4.1可变参数【应用】

- 可变参数介绍

  可变参数又称参数个数可变，用作方法的形参出现，那么方法参数个数就是可变的了

- 可变参数定义格式

```
修饰符 返回值类型 方法名(数据类型... 变量名) {  }
```

- 可变参数的注意事项
  - 这里的变量其实是一个数组
  - 如果一个方法有多个参数，包含可变参数，可变参数要放在最后
- 可变参数的基本使用

```java
public class ArgsDemo01 {
    public static void main(String[] args) {
        System.out.println(sum(10, 20));
        System.out.println(sum(10, 20, 30));
        System.out.println(sum(10, 20, 30, 40));

        System.out.println(sum(10,20,30,40,50));
        System.out.println(sum(10,20,30,40,50,60));
        System.out.println(sum(10,20,30,40,50,60,70));
        System.out.println(sum(10,20,30,40,50,60,70,80,90,100));
    }

//    public static int sum(int b,int... a) {
//        return 0;
//    }

    public static int sum(int... a) {
        int sum = 0;
        for(int i : a) {
            sum += i;
        }
        return sum;
    }
}
```

4.2可变参数的使用【应用】

- Arrays工具类中有一个静态方法：
  - public static List asList(T... a)：返回由指定数组支持的固定大小的列表
  - 返回的集合不能做增删操作，可以做修改操作
- List接口中有一个静态方法：
  - public static List of(E... elements)：返回包含任意数量元素的不可变列表
  - 返回的集合不能做增删改操作
- Set接口中有一个静态方法：
  - public static Set of(E... elements) ：返回一个包含任意数量元素的不可变集合
  - 在给元素的时候，不能给重复的元素
  - 返回的集合不能做增删操作，没有修改的方法
- 示例代码

```java
public class ArgsDemo02 {
    public static void main(String[] args) {
        //public static <T> List<T> asList(T... a)：返回由指定数组支持的固定大小的列表
//        List<String> list = Arrays.asList("hello", "world", "java");
//
////        list.add("javaee"); //UnsupportedOperationException
////        list.remove("world"); //UnsupportedOperationException
//        list.set(1,"javaee");
//
//        System.out.println(list);
```

```
11
12          //public static <E> List<E> of(E... elements)：返回包含任意数量元素的不可变列表
13  //          List<String> list = List.of("hello", "world", "java", "world");
14  //
15  ////          list.add("javaee");//UnsupportedOperationException
16  ////          list.remove("java");//UnsupportedOperationException
17  ////          list.set(1,"javaee");//UnsupportedOperationException
18  //
19  //          System.out.println(list);
20
21          //public static <E> Set<E> of(E... elements) ：返回一个包含任意数量元素的不可变集合
22  //          Set<String> set = Set.of("hello", "world", "java","world");//IllegalArgumentException
23          //Set<String> set = Set.of("hello", "world", "java");
24
25  //          set.add("javaee");//UnsupportedOperationException
26  //          set.remove("world");//UnsupportedOperationException
27
28          //System.out.println(set);
29      }
30  }
```