

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

```
1 public class SetDemo {
2     public static void main(String[] args) {
3         //创建集合对象
4         Set<String> set = new HashSet<String>();
5
6         //添加元素
7         set.add("hello");
8         set.add("world");
9         set.add("java");
10        //不包含重复元素的集合
11        set.add("world");
12
13        //遍历
14        for(String s : set) {
15            System.out.println(s);
16        }
17    }
18 }
```

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

```
1 public class Student {
2     private String name;
3     private int age;
4 }
```

```

5     public Student() {
6     }
7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28
29    @Override
30    public int hashCode() {
31        return 0;
32    }
33 }

```

o 测试类

```

1  public class HashDemo {
2      public static void main(String[] args) {
3          //创建学生对象
4          Student s1 = new Student("林青霞",30);
5
6          //同一个对象多次调用hashCode()方法返回的哈希值是相同的
7          System.out.println(s1.hashCode()); //1060830840
8          System.out.println(s1.hashCode()); //1060830840
9          System.out.println("-----");
10
11         Student s2 = new Student("林青霞",30);
12
13         //默认情况下，不同对象的哈希值是不相同的
14         //通过方法重写，可以实现不同对象的哈希值是相同的
15         System.out.println(s2.hashCode()); //2137211482
16         System.out.println("-----");
17
18         System.out.println("hello".hashCode()); //99162322
19         System.out.println("world".hashCode()); //113318802
20         System.out.println("java".hashCode()); //3254818
21     }

```

```

22     System.out.println("world".hashCode()); //113318802
23     System.out.println("-----");
24
25     System.out.println("重地".hashCode()); //1179395
26     System.out.println("通话".hashCode()); //1179395
27 }
28 }

```

1.3 HashSet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

```

1  public class HashSetDemo01 {
2      public static void main(String[] args) {
3          //创建集合对象
4          HashSet<String> hs = new HashSet<String>();
5
6          //添加元素
7          hs.add("hello");
8          hs.add("world");
9          hs.add("java");
10
11         hs.add("world");
12
13         //遍历
14         for(String s : hs) {
15             System.out.println(s);
16         }
17     }
18 }

```

1.4 HashSet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

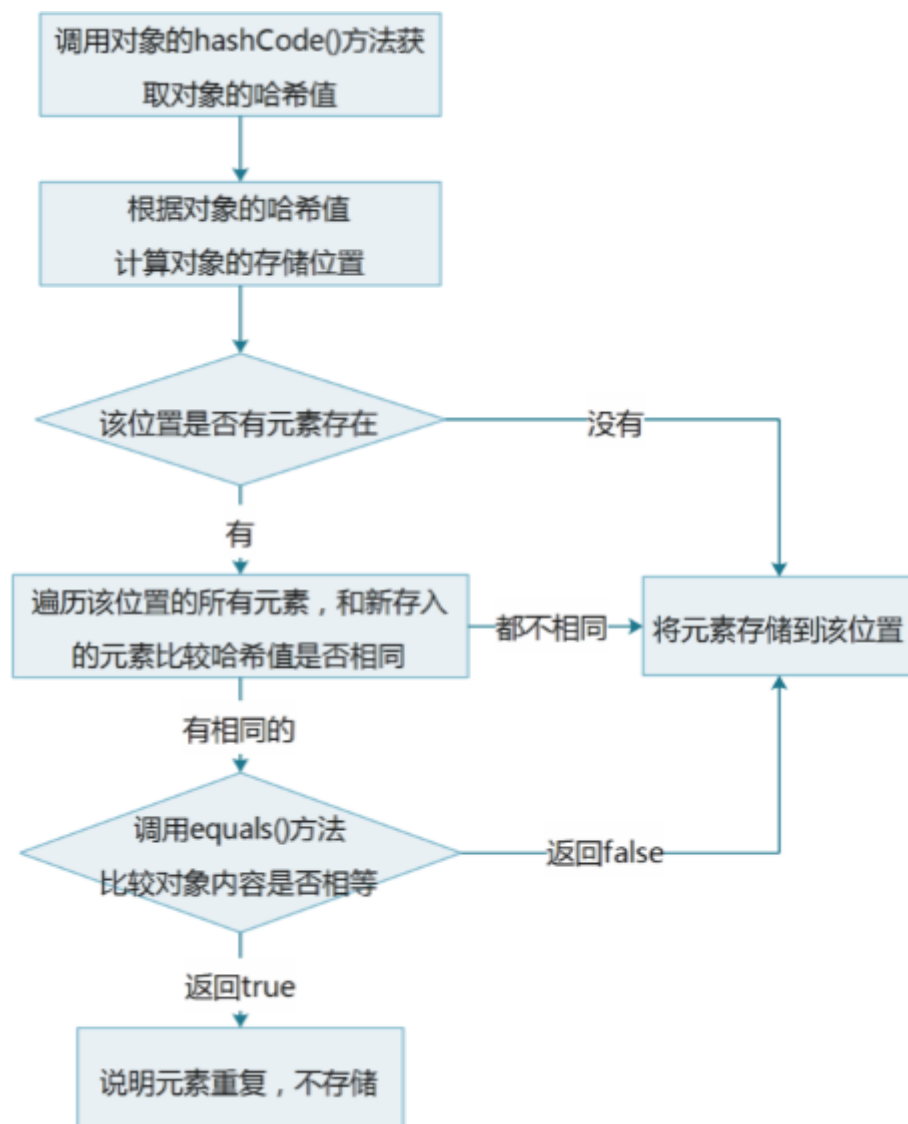
如果哈希值不同，则将当前元素进行存储

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

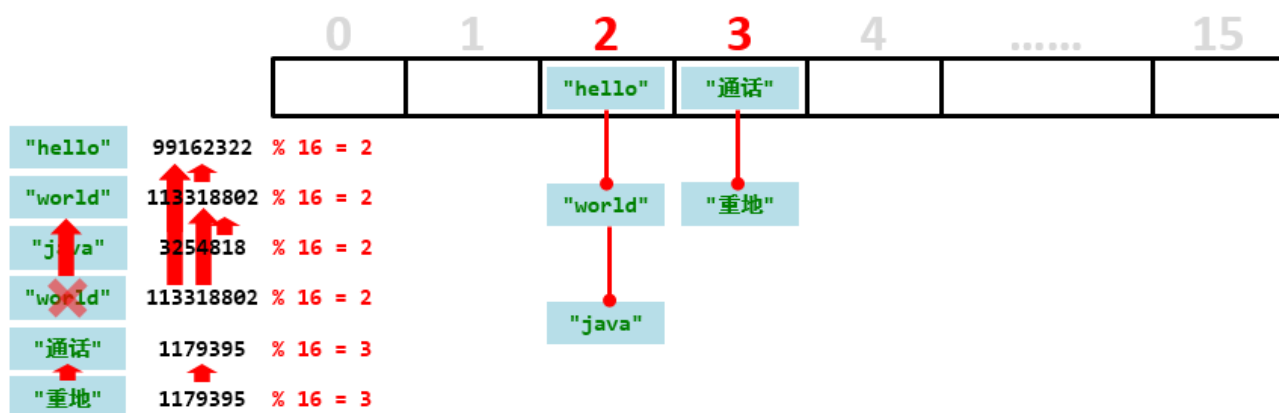
如果内容不相同，则将当前元素进行存储

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

```
1 public class Student {
2     private String name;
3     private int age;
4
5     public Student() {
6     }
7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28
29    @Override
30    public boolean equals(Object o) {
31        if (this == o) return true;
32        if (o == null || getClass() != o.getClass()) return false;
33
34        Student student = (Student) o;
35
36        if (age != student.age) return false;
37        return name != null ? name.equals(student.name) : student.name ==
null;
38    }
39
40    @Override
41    public int hashCode() {
42        int result = name != null ? name.hashCode() : 0;
43        result = 31 * result + age;
44        return result;
45    }
46 }
```

- 测试类

```
1 public class HashSetDemo02 {
2     public static void main(String[] args) {
3         //创建HashSet集合对象
4         HashSet<Student> hs = new HashSet<Student>();
5
6         //创建学生对象
7         Student s1 = new Student("林青霞", 30);
8         Student s2 = new Student("张曼玉", 35);
9         Student s3 = new Student("王祖贤", 33);
10
11         Student s4 = new Student("王祖贤", 33);
12
13         //把学生添加到集合
14         hs.add(s1);
15         hs.add(s2);
16         hs.add(s3);
17         hs.add(s4);
18
19         //遍历集合(增强for)
20         for (Student s : hs) {
21             System.out.println(s.getName() + "," + s.getAge());
22         }
23     }
24 }
```

1.7 LinkedHashSet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

```
1 public class LinkedHashSetDemo {
2     public static void main(String[] args) {
3         //创建集合对象
4         LinkedHashSet<String> linkedHashSet = new LinkedHashSet<String>();
5
6         //添加元素
7         linkedHashSet.add("hello");
8         linkedHashSet.add("world");
9         linkedHashSet.add("java");
10
11         linkedHashSet.add("world");
12
13         //遍历集合
14         for(String s : linkedHashSet) {
15             System.out.println(s);
16         }
17     }
18 }
```

```
17     }
18 }
```

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

```
1  public class TreeSetDemo01 {
2      public static void main(String[] args) {
3          //创建集合对象
4          TreeSet<Integer> ts = new TreeSet<Integer>();
5
6          //添加元素
7          ts.add(10);
8          ts.add(40);
9          ts.add(30);
10         ts.add(50);
11         ts.add(20);
12
13         ts.add(30);
14
15         //遍历集合
16         for(Integer i : ts) {
17             System.out.println(i);
18         }
19     }
20 }
```

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

```

1 public class Student implements Comparable<Student> {
2     private String name;
3     private int age;
4
5     public Student() {
6     }
7
8     public Student(String name, int age) {
9         this.name = name;
10        this.age = age;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28
29    @Override
30    public int compareTo(Student s) {
31        // return 0;
32        // return 1;
33        // return -1;
34        //按照年龄从小到大排序
35        int num = this.age - s.age;
36        // int num = s.age - this.age;
37        //年龄相同时，按照姓名的字母顺序排序
38        int num2 = num==0?this.name.compareTo(s.name):num;
39        return num2;
40    }
41 }

```

◦ 测试类

```

1 public class TreeSetDemo02 {
2     public static void main(String[] args) {
3         //创建集合对象
4         TreeSet<Student> ts = new TreeSet<Student>();
5
6         //创建学生对象
7         Student s1 = new Student("xishi", 29);
8         Student s2 = new Student("wangzhaojun", 28);

```



```

9      Student s3 = new Student("diaochan", 30);
10     Student s4 = new Student("yangyuhuan", 33);
11
12     Student s5 = new Student("linqingxia", 33);
13     Student s6 = new Student("linqingxia", 33);
14
15     //把学生添加到集合
16     ts.add(s1);
17     ts.add(s2);
18     ts.add(s3);
19     ts.add(s4);
20     ts.add(s5);
21     ts.add(s6);
22
23     //遍历集合
24     for (Student s : ts) {
25         System.out.println(s.getName() + "," + s.getAge());
26     }
27 }
28 }

```

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

```

1  public class Student {
2      private String name;
3      private int age;
4
5      public Student() {
6      }
7
8      public Student(String name, int age) {
9          this.name = name;
10         this.age = age;
11     }
12
13     public String getName() {
14         return name;
15     }
16
17     public void setName(String name) {
18         this.name = name;

```

```

19     }
20
21     public int getAge() {
22         return age;
23     }
24
25     public void setAge(int age) {
26         this.age = age;
27     }
28 }

```

○ 测试类

```

1  public class TreeSetDemo {
2      public static void main(String[] args) {
3          //创建集合对象
4          TreeSet<Student> ts = new TreeSet<Student>(new Comparator<Student>
5              () {
6                  @Override
7                  public int compare(Student s1, Student s2) {
8                      //this.age - s.age
9                      //s1,s2
10                     int num = s1.getAge() - s2.getAge();
11                     int num2 = num == 0 ? s1.getName().compareTo(s2.getName())
12                     : num;
13                     return num2;
14                 }
15             });
16
17         //创建学生对象
18         Student s1 = new Student("xishi", 29);
19         Student s2 = new Student("wangzhaojun", 28);
20         Student s3 = new Student("diaochan", 30);
21         Student s4 = new Student("yangyuhuan", 33);
22
23         Student s5 = new Student("linqingxia", 33);
24         Student s6 = new Student("linqingxia", 33);
25
26         //把学生添加到集合
27         ts.add(s1);
28         ts.add(s2);
29         ts.add(s3);
30         ts.add(s4);
31         ts.add(s5);
32         ts.add(s6);
33
34         //遍历集合
35         for (Student s : ts) {
36             System.out.println(s.getName() + "," + s.getAge());
37         }
38     }
39 }

```

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

```
1 public class Student {
2     private String name;
3     private int chinese;
4     private int math;
5
6     public Student() {
7     }
8
9     public Student(String name, int chinese, int math) {
10         this.name = name;
11         this.chinese = chinese;
12         this.math = math;
13     }
14
15     public String getName() {
16         return name;
17     }
18
19     public void setName(String name) {
20         this.name = name;
21     }
22
23     public int getChinese() {
24         return chinese;
25     }
26
27     public void setChinese(int chinese) {
28         this.chinese = chinese;
29     }
30
31     public int getMath() {
32         return math;
33     }
34
35     public void setMath(int math) {
36         this.math = math;
37     }
38
39     public int getSum() {
40         return this.chinese + this.math;
41     }
42 }
```

- 测试类

```
1 public class TreeSetDemo {
2     public static void main(String[] args) {
3         //创建TreeSet集合对象，通过比较器排序进行排序
4         TreeSet<Student> ts = new TreeSet<Student>(new Comparator<Student>
5     () {
6         @Override
7         public int compare(Student s1, Student s2) {
8             //          int num = (s2.getChinese()+s2.getMath())-
9             (s1.getChinese()+s1.getMath());
10            //主要条件
11            int num = s2.getSum() - s1.getSum();
12            //次要条件
13            int num2 = num == 0 ? s1.getChinese() - s2.getChinese() :
14            num;
15            int num3 = num2 == 0 ? s1.getName().compareTo(s2.getName())
16            : num2;
17            return num3;
18        }
19    });
20
21    //创建学生对象
22    Student s1 = new Student("林青霞", 98, 100);
23    Student s2 = new Student("张曼玉", 95, 95);
24    Student s3 = new Student("王祖贤", 100, 93);
25    Student s4 = new Student("柳岩", 100, 97);
26    Student s5 = new Student("风清扬", 98, 98);
27
28    Student s6 = new Student("左冷禅", 97, 99);
29    //      Student s7 = new Student("左冷禅", 97, 99);
30    Student s7 = new Student("赵云", 97, 99);
31
32    //把学生对象添加到集合
33    ts.add(s1);
34    ts.add(s2);
35    ts.add(s3);
36    ts.add(s4);
37    ts.add(s5);
38    ts.add(s6);
39    ts.add(s7);
40
41    //遍历集合
42    for (Student s : ts) {
43        System.out.println(s.getName() + "," + s.getChinese() + "," +
44        s.getMath() + "," + s.getSum());
45    }
46 }
```

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

```
1 public class SetDemo {
2     public static void main(String[] args) {
3         //创建Set集合对象
4         // Set<Integer> set = new HashSet<Integer>();
5         Set<Integer> set = new TreeSet<Integer>();
6
7         //创建随机数对象
8         Random r = new Random();
9
10        //判断集合的长度是不是小于10
11        while (set.size() < 10) {
12            //产生一个随机数，添加到集合
13            int number = r.nextInt(20) + 1;
14            set.add(number);
15        }
16
17        //遍历集合
18        for(Integer i : set) {
19            System.out.println(i);
20        }
21    }
22 }
```

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

```
1 | 修饰符 class 类名<类型> { }
```

- 示例代码
 - 泛型类

```
1 | public class Generic<T> {  
2 |     private T t;  
3 |  
4 |     public T getT() {  
5 |         return t;  
6 |     }  
7 |  
8 |     public void setT(T t) {  
9 |         this.t = t;  
10 |    }  
11 | }
```

- 测试类

```
1 | public class GenericDemo {  
2 |     public static void main(String[] args) {  
3 |         Generic<String> g1 = new Generic<String>();  
4 |         g1.setT("林青霞");  
5 |         System.out.println(g1.getT());  
6 |  
7 |         Generic<Integer> g2 = new Generic<Integer>();  
8 |         g2.setT(30);  
9 |         System.out.println(g2.getT());  
10 |  
11 |         Generic<Boolean> g3 = new Generic<Boolean>();  
12 |         g3.setT(true);  
13 |         System.out.println(g3.getT());  
14 |     }  
15 | }
```

3.3泛型方法【应用】

- 定义格式

```
1 | 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

```
1 | public class Generic {  
2 |     public <T> void show(T t) {  
3 |         System.out.println(t);  
4 |     }  
5 | }
```

- 测试类

```
1 public class GenericDemo {
2     public static void main(String[] args) {
3         Generic g = new Generic();
4         g.show("林青霞");
5         g.show(30);
6         g.show(true);
7         g.show(12.34);
8     }
9 }
```

3.4泛型接口【应用】

- 定义格式

```
1 修饰符 interface 接口名<类型> { }
```

- 示例代码

- 泛型接口

```
1 public interface Generic<T> {
2     void show(T t);
3 }
```

- 泛型接口实现类

```
1 public class GenericImpl<T> implements Generic<T> {
2     @Override
3     public void show(T t) {
4         System.out.println(t);
5     }
6 }
```

- 测试类

```
1 public class GenericDemo {
2     public static void main(String[] args) {
3         Generic<String> g1 = new GenericImpl<String>();
4         g1.show("林青霞");
5
6         Generic<Integer> g2 = new GenericImpl<Integer>();
7         g2.show(30);
8     }
9 }
```

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

```
1 public class GenericDemo {
2     public static void main(String[] args) {
3         //类型通配符：<?>
4         List<?> list1 = new ArrayList<Object>();
5         List<?> list2 = new ArrayList<Number>();
6         List<?> list3 = new ArrayList<Integer>();
7         System.out.println("-----");
8
9         //类型通配符上限：<? extends 类型>
10        // List<? extends Number> list4 = new ArrayList<Object>();
11        List<? extends Number> list5 = new ArrayList<Number>();
12        List<? extends Number> list6 = new ArrayList<Integer>();
13        System.out.println("-----");
14
15        //类型通配符下限：<? super 类型>
16        List<? super Number> list7 = new ArrayList<Object>();
17        List<? super Number> list8 = new ArrayList<Number>();
18        // List<? super Number> list9 = new ArrayList<Integer>();
19
20    }
21 }
```

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

```
1 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


```

1 public class ArgsDemo01 {
2     public static void main(String[] args) {
3         System.out.println(sum(10, 20));
4         System.out.println(sum(10, 20, 30));
5         System.out.println(sum(10, 20, 30, 40));
6
7         System.out.println(sum(10,20,30,40,50));
8         System.out.println(sum(10,20,30,40,50,60));
9         System.out.println(sum(10,20,30,40,50,60,70));
10        System.out.println(sum(10,20,30,40,50,60,70,80,90,100));
11    }
12
13    // public static int sum(int b,int... a) {
14    //     return 0;
15    // }
16
17    public static int sum(int... a) {
18        int sum = 0;
19        for(int i : a) {
20            sum += i;
21        }
22        return sum;
23    }
24 }

```

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

```

1 public class ArgsDemo02 {
2     public static void main(String[] args) {
3         //public static <T> List<T> asList(T... a)：返回由指定数组支持的固定大小的列
4         表
5         // List<String> list = Arrays.asList("hello", "world", "java");
6         //
7         //// list.add("javaee"); //UnsupportedOperationException
8         //// list.remove("world"); //UnsupportedOperationException
9         // list.set(1,"javaee");
10        //
11        // System.out.println(list);
12    }
13 }

```

```
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 }
```