1.Collection集合

1.1数组和集合的区别【理解】

- 相同点

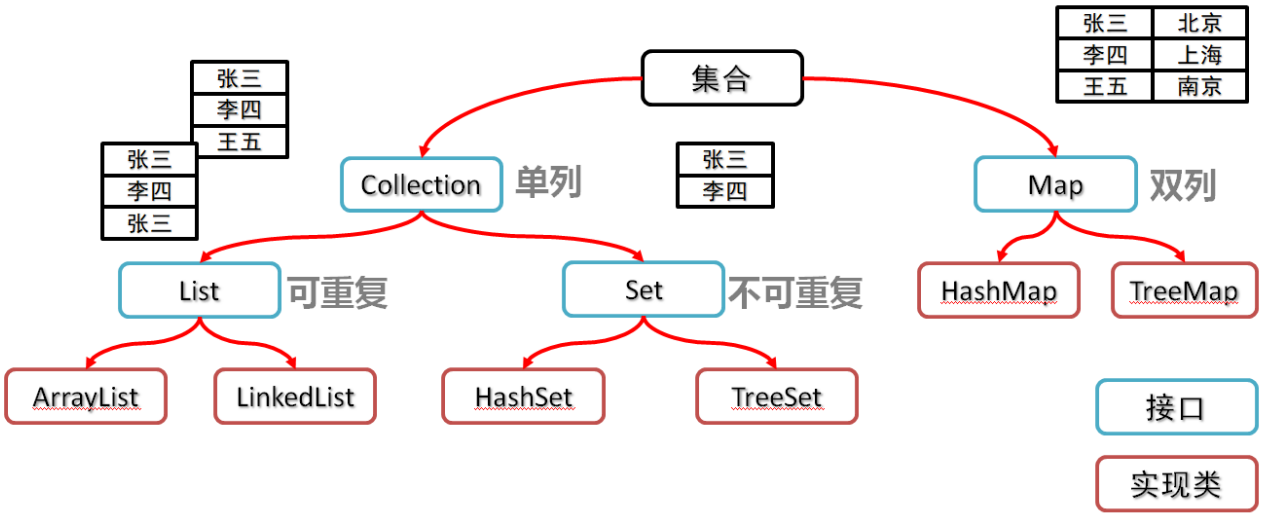
  都是容器,可以存储多个数据

- 不同点

  - 数组的长度是不可变的,集合的长度是可变的
  - 数组可以存基本数据类型和引用数据类型

    集合只能存引用数据类型,如果要存基本数据类型,需要存对应的包装类

1.2集合类体系结构【理解】



1.3Collection 集合概述和使用【应用】

- Collection集合概述
  - 是单例集合的顶层接口,它表示一组对象,这些对象也称为Collection的元素
  - JDK 不提供此接口的任何直接实现.它提供更具体的子接口(如Set和List)实现
- 创建Collection集合的对象
  - 多态的方式
  - 具体的实现类ArrayList
- Collection集合常用方法

方法名	说明
boolean add(E e)	添加元素
boolean remove(Object o)	从集合中移除指定的元素

方法名	说明
boolean removeIf(Object o)	根据条件进行移除
void clear()	清空集合中的元素
boolean contains(Object o)	判断集合中是否存在指定的元素
boolean isEmpty()	判断集合是否为空
int size()	集合的长度，也就是集合中元素的个数

1.4Collection集合的遍历

1.4.1 迭代器遍历

- 迭代器介绍
  - 迭代器,集合的专用遍历方式
  - Iterator iterator(): 返回此集合中元素的迭代器,通过集合对象的iterator()方法得到
- Iterator中的常用方法

  boolean hasNext(): 判断当前位置是否有元素可以被取出
  E next(): 获取当前位置的元素,将迭代器对象移向下一个索引位置
- Collection集合的遍历

```
public class IteratorDemo1 {
    public static void main(String[] args) {
        //创建集合对象
        Collection<String> c = new ArrayList<>();

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

        //Iterator<E> iterator(): 返回此集合中元素的迭代器，通过集合的
iterator()方法得到
        Iterator<String> it = c.iterator();

        //用while循环改进元素的判断和获取
        while (it.hasNext()) {
            String s = it.next();
            System.out.println(s);
        }
    }
}
```

- 迭代器中删除的方法

  void remove(): 删除迭代器对象当前指向的元素

```
public class IteratorDemo2 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("a");
        list.add("b");
        list.add("b");
        list.add("c");
        list.add("d");

        Iterator<String> it = list.iterator();
        while(it.hasNext()){
            String s = it.next();
            if("b".equals(s)){
                //指向谁,那么此时就删除谁.
                it.remove();
            }
        }
        System.out.println(list);
    }
}
```

1.4.2 增强for

- 介绍
  - 它是JDK5之后出现的,其内部原理是一个Iterator迭代器
  - 实现Iterable接口的类才可以使用迭代器和增强for
  - 简化数组和Collection集合的遍历
- 格式

  for(集合/数组中元素的数据类型 变量名 : 集合/数组名) {

  // 已经将当前遍历到的元素封装到变量中了,直接使用变量即可

  }
- 代码

```
public class MyCollectonDemo1 {
    public static void main(String[] args) {
        ArrayList<String> list =  new ArrayList<>();
        list.add("a");
        list.add("b");
        list.add("c");
```

```
            list.add("d");
            list.add("e");
            list.add("f");

            //1,数据类型一定是集合或者数组中元素的类型
            //2,str仅仅是一个变量名而已,在循环的过程中,依次表示集合或者数组中的每一
个元素
            //3,list就是要遍历的集合或者数组
            for(String str : list){
                System.out.println(str);
            }
        }
    }
```

- 细节点注意:

1.报错NoSuchElementException

2.迭代器遍历完毕，指针不会复位

3.循环中只能用一次next方法

4.迭代器遍历时，不能用集合的方法进行增加或者删除

```
public class A04_CollectionDemo4 {
    public static void main(String[] args) {
      /*
        迭代器的细节注意点:
            1.报错NoSuchElementException
            2.迭代器遍历完毕，指针不会复位
            3.循环中只能用一次next方法
            4.迭代器遍历时，不能用集合的方法进行增加或者删除
                暂时当做一个结论先行记忆，在今天我们会讲解源码详细的再来分析。
                如果我实在要删除：那么可以用迭代器提供的remove方法进行删除。
                如果我要添加，暂时没有办法。(只是暂时)
       */

        //1.创建集合并添加元素
        Collection<String> coll = new ArrayList<>();
        coll.add("aaa");
        coll.add("bbb");
        coll.add("ccc");
        coll.add("ddd");

        //2.获取迭代器对象
        //迭代器就好比是一个箭头，默认指向集合的0索引处
        Iterator<String> it = coll.iterator();
        //3.利用循环不断的去获取集合中的每一个元素
        while(it.hasNext()){
            //4.next方法的两件事情：获取元素并移动指针
            String str = it.next();
```

```
            System.out.println(str);
        }

        //当上面循环结束之后，迭代器的指针已经指向了最后没有元素的位置
        //System.out.println(it.next());//NoSuchElementException

        //迭代器遍历完毕，指针不会复位
        System.out.println(it.hasNext());

        //如果我们要继续第二次遍历集合，只能再次获取一个新的迭代器对象
        Iterator<String> it2 = coll.iterator();
        while(it2.hasNext()){
            String str = it2.next();
            System.out.println(str);
        }
    }
}
```

1.4.3 lambda表达式

利用forEach方法，再结合lambda表达式的方式进行遍历

```
public class A07_CollectionDemo7 {
    public static void main(String[] args) {
       /*
        lambda表达式遍历：
                default void forEach(Consumer<? super T> action):
        */

        //1.创建集合并添加元素
        Collection<String> coll = new ArrayList<>();
        coll.add("zhangsan");
        coll.add("lisi");
        coll.add("wangwu");
        //2.利用匿名内部类的形式
        //底层原理：
        //其实也会自己遍历集合，依次得到每一个元素
        //把得到的每一个元素，传递给下面的accept方法
        //s依次表示集合中的每一个数据
       /* coll.forEach(new Consumer<String>() {
            @Override
            public void accept(String s) {
                System.out.println(s);
            }
        });*/

        //lambda表达式
        coll.forEach(s -> System.out.println(s));
    }
}
```

2.List集合

2.1List集合的概述和特点【记忆】

- List集合的概述
  - 有序集合,这里的有序指的是存取顺序
  - 用户可以精确控制列表中每个元素的插入位置,用户可以通过整数索引访问元素,并搜索列表中的元素
  - 与Set集合不同,列表通常允许重复的元素
- List集合的特点
  - 存取有序
  - 可以重复
  - 有索引

2.2List集合的特有方法【应用】

- 方法介绍

方法名	描述
void add(int index,E element)	在此集合中的指定位置插入指定的元素
E remove(int index)	删除指定索引处的元素，返回被删除的元素
E set(int index,E element)	修改指定索引处的元素，返回被修改的元素
E get(int index)	返回指定索引处的元素

- 示例代码

```
public class MyListDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
        //method1(list);
        //method2(list);
        //method3(list);
        //method4(list);
    }

    private static void method4(List<String> list) {
        //        E get(int index)      返回指定索引处的元素
        String s = list.get(0);
        System.out.println(s);
    }

    private static void method3(List<String> list) {
```

```
        //        E set(int index,E element)    修改指定索引处的元素，返回被修改的元素
        //被替换的那个元素,在集合中就不存在了.
        String result = list.set(0, "qqq");
        System.out.println(result);
        System.out.println(list);
    }

    private static void method2(List<String> list) {
        //        E remove(int index)       删除指定索引处的元素，返回被删除的元素
        //在List集合中有两个删除的方法
        //第一个 删除指定的元素,返回值表示当前元素是否删除成功
        //第二个 删除指定索引的元素,返回值表示实际删除的元素
        String s = list.remove(0);
        System.out.println(s);
        System.out.println(list);
    }

    private static void method1(List<String> list) {
        //        void add(int index,E element) 在此集合中的指定位置插入指定的元素
        //原来位置上的元素往后挪一个索引.
        list.add(0,"qqq");
        System.out.println(list);
    }
}
```

2.3List集合的五种遍历方式【应用】

1. 迭代器
2. 列表迭代器
3. 增强for
4. Lambda表达式
5. 普通for循环

代码示例：

```
//创建集合并添加元素
List<String> list = new ArrayList<>();
list.add("aaa");
list.add("bbb");
list.add("ccc");

//1.迭代器
/*Iterator<String> it = list.iterator();
     while(it.hasNext()){
        String str = it.next();
        System.out.println(str);
}*/
```

```
//2.增强for
//下面的变量s，其实就是一个第三方的变量而已。
//在循环的过程中，依次表示集合中的每一个元素
/* for (String s : list) {
       System.out.println(s);
   }*/

//3.Lambda表达式
//forEach方法的底层其实就是一个循环遍历，依次得到集合中的每一个元素
//并把每一个元素传递给下面的accept方法
//accept方法的形参s，依次表示集合中的每一个元素
//list.forEach(s->System.out.println(s) );


//4.普通for循环
//size方法跟get方法还有循环结合的方式，利用索引获取到集合中的每一个元素
/*for (int i = 0; i < list.size(); i++) {
            //i:依次表示集合中的每一个索引
            String s = list.get(i);
            System.out.println(s);
        }*/

// 5.列表迭代器
//获取一个列表迭代器的对象，里面的指针默认也是指向0索引的

//额外添加了一个方法：在遍历的过程中，可以添加元素
ListIterator<String> it = list.listIterator();
while(it.hasNext()){
    String str = it.next();
    if("bbb".equals(str)){
        //qqq
        it.add("qqq");
    }
}
System.out.println(list);
```

2.4 细节点注意：

List系列集合中的两个删除的方法

```
1.直接删除元素
2.通过索引进行删除
```

代码示例:

```java
//List系列集合中的两个删除的方法
//1.直接删除元素
//2.通过索引进行删除

//1.创建集合并添加元素
List<Integer> list = new ArrayList<>();

list.add(1);
list.add(2);
list.add(3);


//2.删除元素
//请问：此时删除的是1这个元素，还是1索引上的元素?
//为什么?
//因为在调用方法的时候，如果方法出现了重载现象
//优先调用，实参跟形参类型一致的那个方法。

//list.remove(1);


//手动装箱，手动把基本数据类型的1，变成Integer类型
Integer i = Integer.valueOf(1);

list.remove(i);

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

3.List集合的实现类

3.1List集合子类的特点【记忆】

- ArrayList集合

  底层是数组结构实现，查询快、增删慢

- LinkedList集合

  底层是链表结构实现，查询慢、增删快

3.2LinkedList集合的特有功能【应用】

- 特有方法

方法名	说明
public void addFirst(E e)	在该列表开头插入指定的元素
public void addLast(E e)	将指定的元素追加到此列表的末尾

方法名	说明
public E getFirst()	返回此列表中的第一个元素
public E getLast()	返回此列表中的最后一个元素
public E removeFirst()	从此列表中删除并返回第一个元素
public E removeLast()	从此列表中删除并返回最后一个元素

- 示例代码

```
public class MyLinkedListDemo4 {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
//        public void addFirst(E e) 在该列表开头插入指定的元素
        //method1(list);

//        public void addLast(E e)  将指定的元素追加到此列表的末尾
        //method2(list);

//        public E getFirst()       返回此列表中的第一个元素
//        public E getLast()        返回此列表中的最后一个元素
        //method3(list);

//        public E removeFirst()        从此列表中删除并返回第一个元素
//        public E removeLast()     从此列表中删除并返回最后一个元素
        //method4(list);

    }

    private static void method4(LinkedList<String> list) {
        String first = list.removeFirst();
        System.out.println(first);

        String last = list.removeLast();
        System.out.println(last);

        System.out.println(list);
    }

    private static void method3(LinkedList<String> list) {
        String first = list.getFirst();
        String last = list.getLast();
        System.out.println(first);
        System.out.println(last);
    }

    private static void method2(LinkedList<String> list) {
```

```
        list.addLast("www");
        System.out.println(list);
    }

    private static void method1(LinkedList<String> list) {
        list.addFirst("qqq");
        System.out.println(list);
    }
}
```

4. 源码分析

4.1 ArrayList源码分析：

核心步骤：

1. 创建ArrayList对象的时候，他在底层先创建了一个长度为0的数组。

   数组名字：elementDate，定义变量size。

   size这个变量有两层含义：
   ①：元素的个数，也就是集合的长度
   ②：下一个元素的存入位置

2. 添加元素，添加完毕后，size++

扩容时机一：

3. 当存满时候，会创建一个新的数组，新数组的长度，是原来的1.5倍，也就是长度为15.再把所有的元素，全拷贝到新数组中。如果继续添加数据，这个长度为15的数组也满了，那么下次还会继续扩容，还是1.5倍。

扩容时机二：

4. 一次性添加多个数据，扩容1.5倍不够，怎么办呀？

   如果一次添加多个元素，1.5倍放不下，那么新创建数组的长度以实际为准。

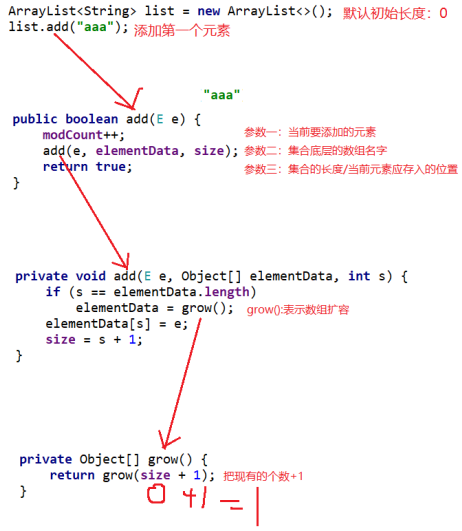
举个例子：
在一开始，如果默认的长度为10的数组已经装满了，在装满的情况下，我一次性要添加100个数据很显然，10扩容1.5倍，变成15，还是不够，

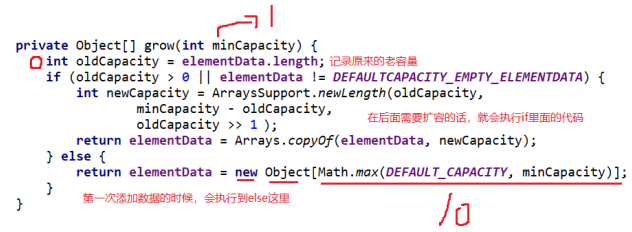
怎么办？

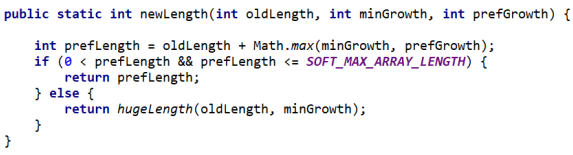
此时新数组的长度，就以实际情况为准，就是110

具体分析过程可以参见视频讲解。

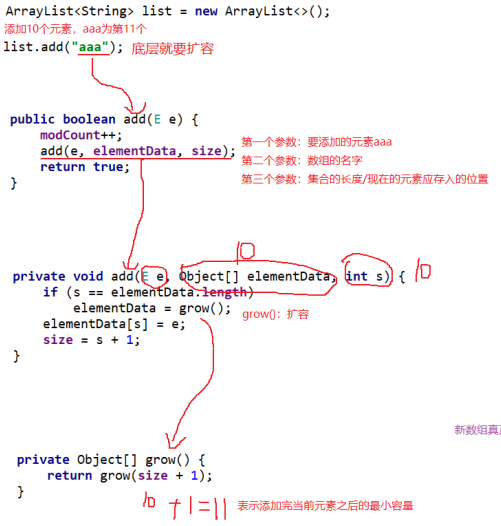
添加一个元素时的扩容：

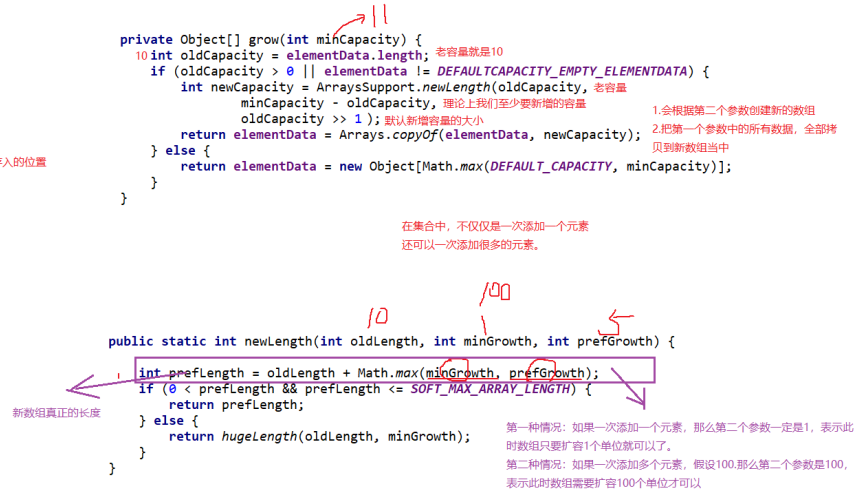






添加多个元素时的扩容：





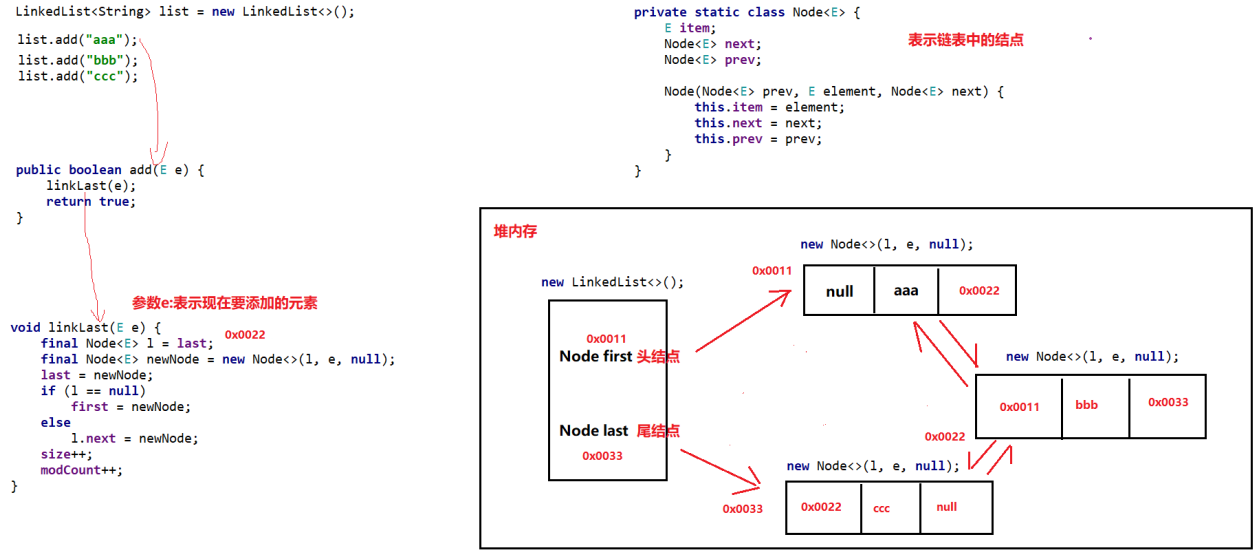
4.2 LinkedList源码分析：

底层是双向链表结构

核心步骤如下：

1. 刚开始创建的时候，底层创建了两个变量：一个记录头结点first，一个记录尾结点last，默认为null
2. 添加第一个元素时，底层创建一个结点对象，first和last都记录这个结点的地址值
3. 添加第二个元素时，底层创建一个结点对象，第一个结点会记录第二个结点的地址值，last会记录新结点的地址值

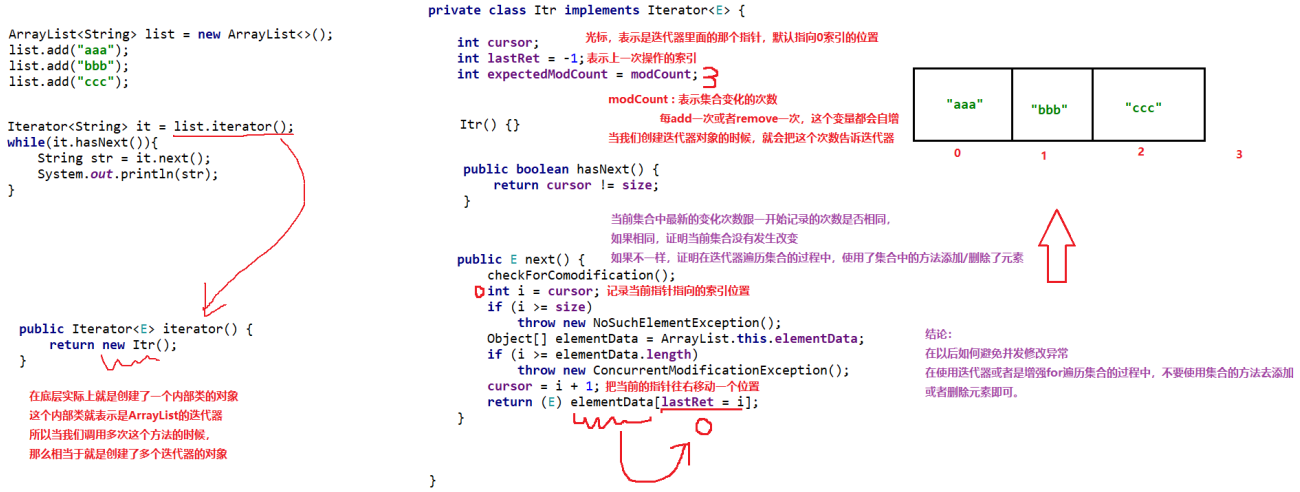
具体分析过程可以参见视频讲解。



4.3 迭代器源码分析：

迭代器遍历相关的三个方法：

- Iterator iterator()： 获取一个迭代器对象
- boolean hasNext()： 判断当前指向的位置是否有元素
- E next()： 获取当前指向的元素并移动指针



5.Set集合

5.1Set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

5.2Set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);
        }
    }
}
```

6.TreeSet集合

6.1TreeSet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

6.2TreeSet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);
        }
    }
}
```

7.HashSet集合

7.1HashSet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

7.2HashSet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);
        }
    }
}
```

- 总结

HashSet集合存储自定义类型元素,要想实现元素的唯一,要求必须重写hashCode方法和equals方法