1. 集合

1.1. 什么是集合

存储对象的容器，面向对象语言对事物的体现都是以对象的形式，所以为了方便对多个对象的操作，存储对象，集合是存储对象最常用的一种方式。
集合的出现就是为了持有对象。集合中可以存储任意类型的对象，而且长度可变。在程序中有可能无法预先知道需要多少个对象，那么用数组来装对象的话，长度不好定义，而集合解决了这样的问题。

1.2. 集合和数组的区别

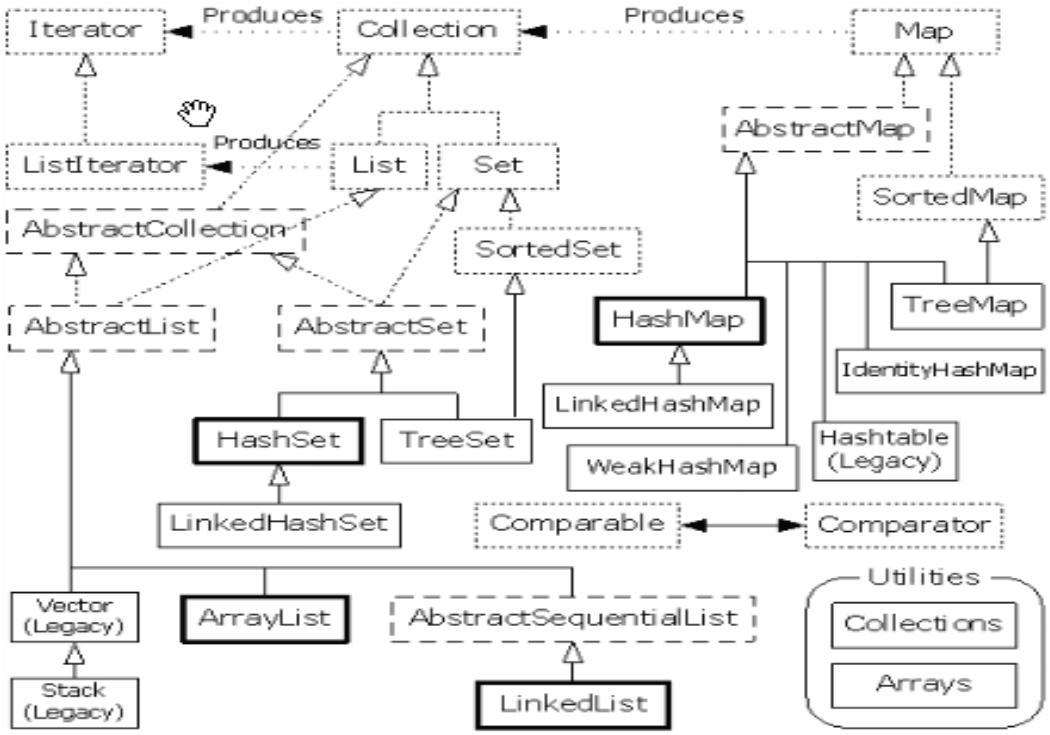
数组和集合类都是容器
数组长度是固定的，集合长度是可变的。数组中可以存储基本数据类型，集合只能存储对象
数组中存储数据类型是单一的，集合中可以存储任意类型的对象。
集合类的特点
用于存储对象，长度是可变的，可以存储不同类型的对象。

1.2.1. 数组的缺点

存储类型单一的数据容器，操作复杂(数组一旦声明好不可变)CRUD

1.3. 集合的分类

集合做什么
1：将对象添加到集合
2：从集合中删除对象
3：从集合中查找一个对象
4：从集合中修改一个对象就是增删改查
注意：集合和数组中存放的都是对象的引用而非对象本身
Java 工程师对不同的容器进行了定义,虽然容器不同,但是还是有一些共性可以抽取最后抽取了一个顶层接口,那么就形成了一个集合框架。如何学习呢?当然是从顶层学起,顶层里边具有最共性,最基本的行为。具体的使用,就要选择具体的容器了。为什么？ 因为不断向上抽取的东西有可能是不能创建对象的.抽象的可能性很大，并且子类对象的方法更多一些. 所以是看顶层,创建底层。那么集合的顶层是什么呢 叫做 Collection
集合框架体系




```
---|Collection: 单列集合
          ---|List: 有存储顺序，可重复
              ---|ArrayList:   数组实现，查找快，增删慢
                               由于是数组实现，在增和删的时候会牵扯到数组
                               增容，以及拷贝元素．所以慢。数组是可以直接
                               按索引查找，所以查找时较快
              ---|LinkedList:  链表实现，增删快，查找慢
                               由于链表实现，增加时只要让前一个元素记住自
                               己就可以，删除时让前一个元素记住后一个元
                               素，后一个元素记住前一个元素．这样的增删
效
                               率较高但查询时需要一个一个的遍历，所以效率
                               较低
              ---|Vector:    和ArrayList原理相同， 但线程安全，效率略低
                               和ArrayList实现方式相同， 但考虑了线程安全问
                               题，所以效率略低
          ---|Set: 无存储顺序，不可重复
              ---|HashSet
              ---|TreeSet
              ---|LinkedHashSet
---| Map: 键值对
       ---|HashMap
       ---|TreeMap
       ---|HashTable
       ---|LinkedHashMap
```

为什么出现这么多集合容器，因为每一个容器对数据的存储方式不同，这种存储方式称之为数据结构（data structure）
注意 集合和数组中存放的都是对象的引用。

1.4. 什么时候该使用什么样的集合

Collection	我们需要保存若干个对象的时候使用集合。
List	如果我们需要保留存储顺序，并且保留重复元素，使用 List. 如果查询较多，那么使用 ArrayList 如果存取较多，那么使用 LinkedList 如果需要线程安全，那么使用 Vector
Set	如果我们不需要保留存储顺序，并且需要去掉重复元素，使用 Set. 如果我们需要将元素排序，那么使用 TreeSet 如果我们不需要排序，使用 HashSet， HashSet 比 TreeSet 效率高. 如果我们需要保留存储顺序，又要过滤重复元素，那么使用 LinkedHashSet

2. 集合类(Collection)

Collection 接口有两个子接口：
List(链表|线性表)
Set(集)
特点：
Collection 中描述的是集合共有的功能(CRUD)
List 可存放重复元素，元素存取是有序的
Set 不可以存放重复元素，元素存取是无序的

```
java.util.Collection
        ---| Collection       描述所有接口的共性
            ----| List 接口    可以有重复元素的集合
            ----| Set  接口    不可以有重复元素的集合
```

2：学习集合对象
学习 Collection 中的共性方法，多个容器在不断向上抽取就出现了该体系。发现 Collection 接口中具有所有容器都具备的共性方法。查阅 API 时，就可以直接看该接口中的方法。并创建其子类对象对集合进行基本应用。当要使用集合对象中特有的方法，在查

看子类具体内容。
查看 api 文档 Collection 在在 java.util 中（注意是大写 Collection）
注意在现阶段遇到的 E T 之类的类型,需要暂时理解为 object 因为涉及到了泛型.
3：创建集合对象,使用 Collection 中的 List 的具体实现类 ArrayList

```
1：Collection coll=new Arraylist();
```

2.1. Collection 接口的共性方法

增加：
1：add() 将指定对象存储到容器中
add 方法的参数类型是 Object 便于接收任意对象
2：addAll() 将指定集合中的元素添加到调用该方法和集合中
删除：
3：remove() 将指定的对象从集合中删除
4：removeAll() 将指定集合中的元素删除
修改
5：clear() 清空集合中的所有元素
判断
6：isEmpty() 判断集合是否为空
7：contains() 判断集合何中是否包含指定对象

8：containsAll() 判断集合中是否包含指定集合
使用 equals()判断两个对象是否相等
获取： 9：int size() 返回集合容器的大小

转成数组 10： toArray() 集合转换数组

2.1.1. 增加：

```
public static void main(String[] args) {
        Collection list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        System.out.println(list);
        // [计算机网络, 现代操作系统, java编程思想]

        // 增加2 将list容器元素添加到list2容器中
        Collection list2 = new ArrayList();
        list2.add("java核心技术");
        list2.addAll(list);
        list2.add("java语言程序设计");
```

```
        System.out.println(list2);
        // [java核心技术, 计算机网络, 现代操作系统, java编程思想, java语言程序
设计]
    }
```

2.1.2. 删除：

```
// 删除1 remove
        boolean remove = list2.remove("java核心技术");
        System.out.println(remove); // true
        System.out.println(list2); //
        //删除2 removeAll() 将list中的元素删除
        boolean removeAll = list2.removeAll(list);
        System.out.println(removeAll);//true
        System.out.println(list2);//[java 语言程序设计]
```

2.1.3. 修改：

```
public static void main(String[] args) {
        Collection list = new ArrayList();
        // 增加: add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
        // 修改 clear() 清空集合中的所有元素
        list.clear();
        System.out.println(list); //[]
    }
```

2.1.4. 判断：

```
public static void main(String[] args) {
        Collection list = new ArrayList();
        // 增加: add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
```

```
        list.add("java语言程序设计");
        System.out.println(list);

        boolean empty = list.isEmpty();
        System.out.println(empty);// false
        boolean contains = list.contains("java编程思想");
        System.out.println(contains);// true
        Collection list2 = new ArrayList();
        list2.add("水许传");
        boolean containsAll = list.containsAll(list2);
        System.out.println(containsAll);// false

    }
```

2.1.5. 获取：

```
public static void main(String[] args) {
        Collection list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
        // 获取  集合容器的大小
        int size = list.size();
        System.out.println(size);
    }
```

2.1.6. 练习：集合中添加自定义对象

该案例要求完成使用集合:

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

        // 创建集合对象
        Collection coll = new ArrayList();

        // 创建Person对象
        Person p1 = new Person("jack", 25);
        Person p2 = new Person("rose", 22);
```

```
        Person p3 = new Person("lucy", 20);
        Person p4 = new Person("jack", 25);

        // 集合中添加一些Perosn

        // 删除指定Person

        // 删除所有Person

        // 判断容器中是否还有Person

        // 判断容器中是否包含指定Person

        // 获取容器中Person的个数

        // 将容器变为数组,遍历除所有Person

    }
```

分析:

1：Person 类

1：姓名和年龄

2：重写 hashCode 和 equals 方法

1：如果不重写，调用 Object 类的 equals 方法，判断内存地址，为 false

1：如果是 Person 类对象，并且姓名和年龄相同就返回 true

2：如果不重写，调用父类 hashCode 方法

1：如果 equals 方法相同，那么 hashCode 也要相同，需要重写 hashCode 方法

3：重写 toString 方法

1：不重写，直接调用 Object 类的 toString 方法，打印该对象的内存地址

Person 类

```
class Person {
    private String name;
    private int age;

    public Person() {

    }

    public Person(String name, int age) {

        this.name = name;
        this.age = age;
    }
```

```
    @Override
    public int hashCode() {
        return this.name.hashCode() + age;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Person)) {
            return false;
        }
        Person p = (Person) obj;
        return this.name.equals(p.name) && this.age == p.age;
    }

    @Override
    public String toString() {
        return "Person :name=" + name + ", age=" + age;
    }

}
```

```
public static void main(String[] args) {
        Person p1 = new Person("张三", 19);
        Person p2 = new Person("李四", 20);
        Person p3 = new Person("王五", 18);
        Collection list = new ArrayList();
        list.add(p1);
        list.add(p2);
        list.add(p3);
        // isEmpty() 判断集合是否为空
        boolean empty = list.isEmpty();
        System.out.println(empty);
        // 返回集合容器的大小
        int size = list.size();
        System.out.println(size);
         // contains()判断集合何中是否包含指定对象
        boolean contains = list.contains(p1);
        System.out.println(contains);

        // remove(); 将指定的对象从集合中删除
        list.remove(p1);
```

```
        // clear() 清空集合中的所有元素
        list.clear();
        System.out.println(list);

    }
```

```
//使用集合存储自定义对象2
class Book {
    private String name;
    private double price;

    public Book() {

    }

    public Book(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

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

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public int hashCode() {
        return (int) (this.name.hashCode() + price);
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Book)) {
            return false;
        }
        Book book = (Book) obj;
        return this.name.equals(book.name) && this.price == book.price;
    }

    @Override
    public String toString() {
        return "book:@ name:" + this.name + ", price:" + this.price;
    }
}
public class Demo1 {
    public static void main(String[] args) {
        Collection col = new ArrayList();
        col.add(new Book("think in java", 100));
         col.add(new Book("core java", 200));
         System.out.println(col);
    }
}
```

2.2. List

```
---| Iterable     接口
        Iterator iterator()
    ----| Collection     接口
        ------| List     接口 元素可以重复，允许在指定位置插入元素，并通过索引来访问元素
```

2.2.1. List集合特有方法

```
1：增加
        void add(int index, E element) 指定位置添加元素
        boolean addAll(int index, Collection c) 指定位置添加集合
2：删除
         E remove(int index) 删除指定位置元素

3：修改
```

```
        E set(int index, E element)     返回的是需要替换的集合中的元素
4：查找：
        E get(int index)                注意： IndexOutOfBoundsException
        int indexOf(Object o)           // 找不到返回-1
        lastIndexOf(Object o)
5：求子集合
         List<E> subList(int fromIndex, int toIndex) // 不包含 toIndex
```

2.2.1.1. 增加

```
public static void main(String[] args) {
        List list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
        // add,在0角标位置添加一本书
        list.add(0, "舒克和贝塔");
        System.out.println(list);
        // 在list2集合的1角标位置添加list集合元素
        List list2 = new ArrayList();
        list2.add("史记");
        list2.add("资治通鉴");
        list2.add("全球通史");
        boolean addAll = list2.addAll(1, list);
        System.out.println(addAll); //true
System.out.println(list2);
    }
```

2.2.1.2. 删除

```
public static void main(String[] args) {
        List list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
```

```
        System.out.println(list);
        // 删除0角标元素
        Object remove = list.remove(0);
        System.out.println(remove);
    }
```

2.2.1.3. 修改：

```
public static void main(String[] args) {
        List list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
        // 修改2角标位置的书，返回的原来2角标位置的书
        Object set = list.set(2, "边城");
        System.out.println(set); //java编程思想
        System.out.println(list);

    }
```

查找

```
    List list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
        // 查找： E get(int index)  注意角标越界
        Object set = list.get(list.size() - 1);
        System.out.println(set); // java语言程序设计
        System.out.println(list);
       list.get(list.size()); //IndexOutOfBoundsException
    }
```

2.2.1.4. 查找

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

```
        List list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        list.add("java编程思想");
        System.out.println(list);
        // 查找： E get(int index) 注意角标越界
        Object set = list.get(list.size() - 1);
        System.out.println(set); // java语言程序设计
        System.out.println(list);
        // list.get(list.size()); //IndexOutOfBoundsException

        // indexOf(Object o) 返回第一次出现的指定元素的角标
        int indexOf = list.indexOf("java编程思想");
        System.out.println(indexOf); // 2
        // 没有找到，返回-1
        int indexOf2 = list.indexOf("三国志");
        System.out.println(indexOf2); // -1

        // lastIndexOf 返回最后出现的指定元素的角标
        int lastIndexOf = list.lastIndexOf("java编程思想");
        System.out.println(lastIndexOf); // 5
    }
```

2.2.2. ArrayList

```
--| Iterable
            ----| Collection
               ------| List
                ---------| ArrayList  底层采用数组实现，默认 10。每次增长
                           60%，( (oldCapacity * 3)/2 + 1)  查询快，增
                            删慢。
               ---------| LinkedList
```

ArrayList:实现原理：
数组实现，查找快，增删慢
数组为什么是查询快?因为数组的内存空间地址是连续的.

ArrayList 底层维护了一个 Object[] 用于存储对象，默认数组的长度是 10。可以通过 new ArrayList(20)显式的指定用于存储对象的数组的长度。

当默认的或者指定的容量不够存储对象的时候，容量自动增长为原来的容量的 1.5 倍。

由于 ArrayList 是数组实现，在增和删的时候会牵扯到数组增容，以及拷贝元素．所以慢。数组是可以直接按索引查找，所以查找时较快
可以考虑,假设向数组的 0 角标未知添加元素,那么原来的角标位置的元素需要整体往后移,并且数组可能还要增容,一旦增容,就需要要将老数组的内容拷贝到新数组中.所以数组的增删的效率是很低的.

练习：去除 ArrayList 集合中重复元素
1：存入字符串元素
2：存入自定义对象元素（如 Perosn 对象）
原理：
循环遍历该集合，每取出一个放置在新的集合中，放置之前先判断新的集合是否以包含了新的元素。

思路:
存入人的对象.
1先定义person 类
2将该类的实例存入集合
3 将对象元素进行操作． 注意:自定义对象要进行复写toString 和 equals 方法.
为什么? 因为object 是person 的父类,object 中的toString 返回的是哈希值,object 类中equals
方法比较的是对象的地址值.
思路
1存入字符串对象
2存入自定义对象 如person
2创建容器,用于存储非重复元素
3对原容器进行遍历,在遍历过程中进行判断遍历到的元素是否在容器中存在.(contains)
4如果存在,就不存入,否则存入.
5 返回新容器

```
public class Demo6 {
    public static void main(String[] args) {
        ArrayList arr = new ArrayList();
        Person p1 = new Person("jack", 20);
        Person p2 = new Person("rose", 18);
        Person p3 = new Person("rose", 18);
        arr.add(p1);
        arr.add(p2);
        arr.add(p3);
        System.out.println(arr);
        ArrayList arr2 = new ArrayList();
        for (int i = 0; i < arr.size(); i++) {
            Object obj = arr.get(i);
            Person p = (Person) obj;
            if (!(arr2.contains(p))) {
                arr2.add(p);
```

```
            }
        }
        System.out.println(arr2);
    }
}

class Person {
    private String name;
    private int age;

    public Person() {

    }

    public Person(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 this.name.hashCode() + age * 37;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Person)) {
```

```
            return false;
        }
        Person p = (Person) obj;

        return this.name.equals(p.name) && this.age == p.age;
    }

    @Override
    public String toString() {
        return "Person@name:" + this.name + " age:" + this.age;
    }

}
```

在实际的开发中 ArrayList 是使用频率最高的一个集合。

2.2.3. LinkedList

```
--| Iterable
            ----| Collection
               ------| List
                ---------| ArrayList   底层采用数组实现，默认 10。每次增长
                           60%，((oldCapacity * 3)/2 + 1)  查询快，增
                            删慢。
               ---------| LinkedList   底层采用链表实现，增删快，查询慢。
```

LinkedList:链表实现，增删快，查找慢

由于 LinkedList:在内存中的地址不连续,需要让上一个元素记住下一个元素.所以每个元素中保存的有下一个元素的位置.虽然也有角标,但是查找的时候,需要从头往下找,显然是没有数组查找快的.但是,链表在插入新元素的时候,只需要让前一个元素记住新元素,让新元素记住下一个元素就可以了.所以插入很快.

由于链表实现，增加时只要让前一个元素记住自己就可以，删除时让前一个元素记住后一个元素，后一个元素记住前一个元素. 这样的增删效率较高。

但查询时需要一个一个的遍历，所以效率较低。

特有方法

```
1：方法介绍
            addFirst(E e)
            addLast(E e)
            getFirst()
            getLast()
```

```
            removeFirst()
            removeLast()
            如果集合中没有元素，获取或者删除元
            素抛：NoSuchElementException
2：数据结构
                1：栈 （1.6）
                    先进后出
                    push()
                    pop()
                2：队列（双端队列 1.5）
                    先进先出
                    offer()
                    poll()
3：返回逆序的迭代器对象
                    descendingIterator()    返回逆序的迭代器对象
```

基本方法

```
import java.util.Iterator;
import java.util.LinkedList;

public class Demo3 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.add("西游记");
        list.add("三国演义");
        list.add("石头记");
        list.add("水浒传");
        list.add("全球通史");
        list.addFirst("史记");
        list.addLast("呐喊");
        // list.addFirst(null);
        // list.addLast(null);
        System.out.println(list);
        // 获取指定位置处的元素。
        String str = (String) list.get(0);
        // 返回此列表的第一个元素。
        String str2 = (String) list.getFirst();
        System.out.println(str.equals(str2));

        // 获取指定位置处的元素。
        String str3 = (String) list.get(list.size() - 1);
        // 返回此列表的最后一个元素。
        String str4 = (String) list.getLast();
        System.out.println(str3.equals(str4));
```

```
        // 获取但不移除此列表的头（第一个元素）。
        Object element = list.element();
        System.out.println(element);

        int size = list.size();
        System.out.println(size);
}
```

迭代

```
mport java.util.Iterator;
import java.util.LinkedList;

public class Demo3 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.add("西游记");
        list.add("三国演义");
        list.add("石头记");
        list.add("水浒传");
        list.add("全球通史");
        Iterator it = list.iterator();
        while (it.hasNext()) {
            String next = (String) it.next();
            System.out.println(next);
        }
    }
}
```

逆序迭代

```
import java.util.Iterator;
import java.util.LinkedList;

public class Demo6 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.add("aa");
        list.add("bb");
        list.add("cc");
        Iterator dit = list.descendingIterator();
        while (dit.hasNext()) {
            System.out.println(dit.next());
        }
```

```
    }
}
```

注意：可以使用该集合去模拟出队列(先进先出) 或者堆栈(后进先出) 数据结构。
堆栈(后进先出)

```
//堆栈(后进先出) 数据结构
public class Demo3 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        // 压栈，先进后出
        list.push("西游记");
        list.push("三国演义");
        list.push("石头记");
        list.push("水浒传");
        System.out.println(list);
        // 弹栈
        String str1 = (String) list.pop();
        System.out.println(str1);
        String str2 = (String) list.pop();
        System.out.println(str2);
        String str3 = (String) list.pop();
        System.out.println(str3);
        String str4 = (String) list.pop();
        System.out.println(str4);
        System.out.println(list.size());// 0
        System.out.println(list); //[]
    }
}
```

队列，先进先出

```
import java.util.LinkedList;

public class Demo3 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        // 队列，先进先出
        list.offer("西游记");
        list.offer("三国演义");
        list.offer("石头记");
        list.offer("水浒传");
        System.out.println(list);
        // 出队列
```

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

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

        System.out.println(list.peek()); // 获取队列的头元素，但是不删除
        System.out.println(list.peekFirst()); // 获取队列的头元素，但是不删除
        System.out.println(list.peekLast()); // 获取队列的最后一个元素但是不删除

    }
}
```

ArrayList 和 LinkedList 的存储查找的优缺点：

1、ArrayList 是采用动态数组来存储元素的，它允许直接用下标号来直接查找对应的元素。但是，但是插入元素要涉及数组元素移动及内存的操作。总结：查找速度快，插入操作慢。

2、LinkedList 是采用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快

问题：有一批数据要存储，要求存储这批数据不能出现重复数据，ArrayList、LinkedList 都没法满足需求。解决办法：使用 set 集合。

2.2.4. Vector

Vector: 描述的是一个线程安全的 ArrayList。

ArrayList： 单线程效率高

Vector ： 多线程安全的，所以效率低

特有的方法：

void addElement(E obj) 在集合末尾添加元素

E elementAt(int index) 返回指定角标的元素

Enumeration elements() 返回集合中的所有元素，封装到 Enumeration 对象中

Enumeration 接口：

boolean hasMoreElements()

测试此枚举是否包含更多的元素。 E nextElement() 如果此枚举对象至少还有一个可提供的元素，则返回此枚举的下一个元素。

```
public static void main(String[] args)
    {
        Vector v = new Vector();
        v.addElement("aaa");
        v.addElement("bbb");
        v.addElement("ccc");
        System.out.println( v );
        System.out.println( v.elementAt(2) );   // ccc
        // 遍历Vector遍历
        Enumeration ens = v.elements();
        while ( ens.hasMoreElements() )
        {
            System.out.println( ens.nextElement() );
        }
    }
```

2.3. 迭代器

为了方便的处理集合中的元素,Java 中出现了一个对象,该对象提供了一些方法专门处理集合中的元素.例如删除和获取集合中的元素.该对象就叫做迭代器(Iterator).

对 Collection 进行迭代的类，称其为迭代器。还是面向对象的思想，专业对象做专业的事情，迭代器就是专门取出集合元素的对象。但是该对象比较特殊，不能直接创建对象（通过 new），该对象是以内部类的形式存在于每个集合类的内部。

如何获取迭代器？Collection 接口中定义了获取集合类迭代器的方法（iterator（）），所以所有的 Collection 体系集合都可以获取自身的迭代器。

正是由于每一个容器都有取出元素的功能。这些功能定义都一样，只不过实现的具体方式不同（因为每一个容器的数据结构不一样）所以对共性的取出功能进行了抽取，从而出现了 Iterator 接口。而每一个容器都在其内部对该接口进行了内部类的实现。也就是将取出方式的细节进行封装。

2.3.1. Iterable

Jdk1.5 之后添加的新接口，Collection 的父接口．实现了 Iterable 的类就是可迭代的.并且支持增强 for 循环。该接口只有一个方法即获取迭代器的方法 iterator（）可以获取每个容器自身的迭代器 Iterator。（Collection）集合容器都需要获取迭代器（Iterator）于是在 5.0 后又进行了抽取将获取容器迭代器的 iterator（）方法放入

到了 Iterable 接口中。Collection 接口进程了 Iterable，所以 Collection 体系都具备获取自身迭代器的方法，只不过每个子类集合都进行了重写（因为数据结构不同）

2.3.2. Iterator

Iterator iterator() 返回该集合的迭代器对象

该类主要用于遍历集合对象，该类描述了遍历集合的常见方法

1: java.lang. Itreable
---| Itreable 接口 实现该接口可以使用增强 for 循环
---| Collection 描述所有集合共性的接口
---| List 接口 可以有重复元素的集合
---| Set 接口 不可以有重复元素的集合

public interface Iterable<T>
Itreable 该接口仅有一个方法，用于返回集合迭代器对象。
1: Iterator<T> iterator() 返回集合的迭代器对象

Iterator 接口定义的方法

Itreator 该接口是集合的迭代器接口类，定义了常见的迭代方法

1: boolean hasNext()
判断集合中是否有元素，如果有元素可以迭代，就返回 true。

2: E next()
返回迭代的下一个元素，注意： 如果没有下一个元素时，调用 next 元素会抛出 NoSuchElementException

3: void remove()
从迭代器指向的集合中移除迭代器返回的最后一个元素(可选操作)。

思考：为什么 next 方法的返回类型是 Object 的呢？ 为了可以接收任意类型的对象,那么返回的时候,不知道是什么类型的就定义为 object

2.3.3. 迭代器遍历

1: while 循环

```
public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
```

```
        list.add("java语言程序设计");
        System.out.println(list);
        Iterator it = list.iterator();
        while (it.hasNext()) {
            String next = (String) it.next();
            System.out.println(next);
        }
    }
```

2：for 循环

```
import java.util.ArrayList;
import java.util.Iterator;

public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);

        for (Iterator it = list.iterator(); it.hasNext();) {
            //迭代器的next方法返回值类型是Object，所以要记得类型转换。
            String next = (String) it.next();
            System.out.println(next);
        }
    }
}
```

需要取出所有元素时，可以通过循环，java 建议使用 for 循环。因为可以对内存进行一下优化。

3：使用迭代器清空集合

```
public class Demo1 {
    public static void main(String[] args) {
        Collection coll = new ArrayList();
        coll.add("aaa");
        coll.add("bbb");
        coll.add("ccc");
        coll.add("ddd");
        System.out.println(coll);
        Iterator it = coll.iterator();
```

```
        while (it.hasNext()) {
            it.next();
            it.remove();
        }
        System.out.println(coll);
    }
}
```

细节一：

如果迭代器的指针已经指向了集合的末尾，那么如果再调用 next() 会返回 NoSuchElementException 异常

```
import java.util.ArrayList;
import java.util.Iterator;

public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);

        Iterator it = list.iterator();
        while (it.hasNext()) {
            String next = (String) it.next();
            System.out.println(next);
        }
        // 迭代器的指针已经指向了集合的末尾
        // String next = (String) it.next();
        // java.util.NoSuchElementException
    }
}
```

细节二：

如果调用 remove 之前没有调用 next 是不合法的，会抛出 IllegalStateException

```
import java.util.ArrayList;
import java.util.Iterator;

public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
```

```
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);

        Iterator it = list.iterator();
        while (it.hasNext()) {
            // 调用remove之前没有调用next是不合法的
            // it.remove();
            // java.lang.IllegalStateException
            String next = (String) it.next();
            System.out.println(next);
        }

    }
}
```

4：迭代器原理

查看 ArrayList 源码

```
private class Itr implements Iterator<E> {

        int cursor = 0;

        int lastRet = -1;

        int expectedModCount = modCount;

        public boolean hasNext() {
            return cursor != size();
        }

        public E next() {
            checkForComodification();
            try {
                E next = get(cursor);
                lastRet = cursor++;
                return next;
            } catch (IndexOutOfBoundsException e) {
                checkForComodification();
                throw new NoSuchElementException();
            }
```

```
        }

        public void remove() {
            if (lastRet == -1)
                throw new IllegalStateException();
            checkForComodification();

            try {
                AbstractList.this.remove(lastRet);
                if (lastRet < cursor)
                    cursor--;
                lastRet = -1;
                expectedModCount = modCount;
            } catch (IndexOutOfBoundsException e) {
                throw new ConcurrentModificationException();
            }
        }


    }
```

5：注意在对集合进行迭代过程中，不允许出现迭代器以外的对元素的操作，因为这样会产生安全隐患，java 会抛出异常并发修改异常（ConcurrentModificationException），普通迭代器只支持在迭代过程中的删除动作。

注意：　　ConcurrentModificationException：当一个集合在循环中即使用引用变量操作集合又使用迭代器操作集合对象，　会抛出该异常。

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
public class Demo1 {
    public static void main(String[] args) {
        Collection coll = new ArrayList();
        coll.add("aaa");
        coll.add("bbb");
        coll.add("ccc");
        coll.add("ddd");
        System.out.println(coll);
        Iterator it = coll.iterator();
        while (it.hasNext()) {
            it.next();
            it.remove();
            coll.add("abc"); // 出现了迭代器以外的对元素的操作
        }
        System.out.println(coll);
```

```
    }
}
```

如果是 List 集合，想要在迭代中操作元素可以使用 List 集合的特有迭代器 ListIterator，该迭代器支持在迭代过程中，添加元素和修改元素。

2.3.4. List 特有的迭代器 ListIterator

```
public interface ListIterator extends Iterator
```

```
ListIterator<E> listIterator()
```

```
---| Iterator
       hasNext()
       next()
       remove()
      ------| ListIterator  Iterator 子接口 List 专属的迭代器
                add(E e)     将指定的元素插入列表（可选操作）。该元素直接插入到 next 返回的下一个元素的前面（如果有）
                void set(E o)   用指定元素替换 next 或 previous 返回的最后一个元素
                hasPrevious()    逆向遍历列表，列表迭代器有多个元素，则返回 true。
                previous()       返回列表中的前一个元素。
```

Iterator 在迭代时，只能对元素进行获取(next())和删除(remove())的操作。
对于 Iterator 的子接口 ListIterator 在迭代 list 集合时，还可以对元素进行添加(add(obj))，修改 set(obj)的操作。

```
import java.util.ArrayList;
import java.util.ListIterator;

public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
         // 获取List专属的迭代器
        ListIterator lit = list.listIterator();

        while (lit.hasNext()) {
```

```
            String next = (String) lit.next();
            System.out.println(next);
        }

    }
}
```

倒序遍历

```
import java.util.ArrayList;
import java.util.ListIterator;

public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);
        // 获取List专属的迭代器
        ListIterator lit = list.listIterator();
        while (lit.hasNext()) {
            String next = (String) lit.next();
            System.out.println(next);
        }
        System.out.println("***************");
        while (lit.hasPrevious()) {
            String next = (String) lit.previous();
            System.out.println(next);
        }

    }
}
```

Set 方法：用指定元素替换 next 或 previous 返回的最后一个元素

```
import java.util.ArrayList;
import java.util.ListIterator;

public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
```

```
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);

        ListIterator lit = list.listIterator();
        lit.next(); // 计算机网络
        lit.next(); // 现代操作系统
        System.out.println(lit.next()); // java编程思想
        //用指定元素替换 next 或 previous 返回的最后一个元素
        lit.set("平凡的世界");// 将java编程思想替换为平凡的世界
        System.out.println(list);

    }
}
```

add 方法将指定的元素插入列表，该元素直接插入到 next 返回的元素的后

```
public class Demo2 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        // 增加：add() 将指定对象存储到容器中
        list.add("计算机网络");
        list.add("现代操作系统");
        list.add("java编程思想");
        list.add("java核心技术");
        list.add("java语言程序设计");
        System.out.println(list);

        ListIterator lit = list.listIterator();
        lit.next(); // 计算机网络
        lit.next(); // 现代操作系统
        System.out.println(lit.next()); // java编程思想
        // 将指定的元素插入列表，该元素直接插入到 next 返回的元素的后
        lit.add("平凡的世界");// 在java编程思想后添加平凡的世界
        System.out.println(list);

    }
}
```

2.4. Set

Set:注重独一无二的性质,该体系集合可以知道某物是否已近存在于集合中,不会存储重复的元素
用于存储无序(存入和取出的顺序不一定相同)元素，值不能重复。
对象的相等性

引用到堆上同一个对象的两个引用是相等的。如果对两个引用调用 hashCode 方法，会得到相同的结果，如果对象所属的类没有覆盖 Object 的 hashCode 方法的话，hashCode 会返回每个对象特有的序号（java 是依据对象的内存地址计算出的此序号），所以两个不同的对象的 hashCode 值是不可能相等的。

如果想要让两个不同的 Person 对象视为相等的，就必须覆盖 Object 继下来的 hashCode 方法和 equals 方法，因为 Object hashCode 方法返回的是该对象的内存地址，所以必须重写 hashCode 方法，才能保证两个不同的对象具有相同的 hashCode，同时也需要两个不同对象比较 equals 方法会返回 true
该集合中没有特有的方法，直接继承自 Collection。

```
---| Itreable      接口 实现该接口可以使用增强 for 循环
              ---| Collection      描述所有集合共性的接口
                  ---| List 接口      可以有重复元素的集合
                         ---| ArrayList
                         ---|  LinkedList
                  ---| Set 接口       不可以有重复元素的集合
```

案例：set 集合添加元素并使用迭代器迭代元素。

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class Demo4 {
    public static void main(String[] args) {
        //Set 集合存和取的顺序不一致。
        Set hs = new HashSet();
        hs.add("世界军事");
        hs.add("兵器知识");
        hs.add("舰船知识");
        hs.add("汉和防务");
        System.out.println(hs);
        // [舰船知识, 世界军事, 兵器知识, 汉和防务]
        Iterator it = hs.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

2.4.1. HashSet

```
---| Itreable        接口 实现该接口可以使用增强 for 循环
              ---| Collection       描述所有集合共性的接口
                  ---| List 接口     可以有重复元素的集合
                         ---| ArrayList
                         ---|  LinkedList
                  ---| Set 接口      不可以有重复元素的集合
                         ---| HashSet  线程不安全，存取速度快。底层是以
                                        哈希表实现的。
```

HashSet
哈希表边存放的是哈希值。HashSet 存储元素的顺序并不是按照存入时的顺序（和 List 显然不同） 是按照哈希值来存的所以取数据也是按照哈希值取得。
HashSet 不存入重复元素的规则.使用 hashcode 和 equals

由于 Set 集合是不能存入重复元素的集合。那么 HashSet 也是具备这一特性的。HashSet 如何检查重复？HashSet 会通过元素的 hashcode（）和 equals 方法进行判断元素师否重复。

当你试图把对象加入 HashSet 时，HashSet 会使用对象的 hashCode 来判断对象加入的位置。同时也会与其他已经加入的对象的 hashCode 进行比较，如果没有相等的 hashCode，HashSet 就会假设对象没有重复出现。

简单一句话，如果对象的 hashCode 值是不同的，那么 HashSet 会认为对象是不可能相等的。

因此我们自定义类的时候需要重写 hashCode，来确保对象具有相同的 hashCode 值。如果元素(对象)的 hashCode 值相同,是不是就无法存入 HashSet 中了？ 当然不是,会继续使用 equals 进行比较.如果 equals 为 true 那么 HashSet 认为新加入的对象重复了,所以加入失败。如果 equals 为 false 那么 HashSet 认为新加入的对象没有重复.新元素可以存入.

总结：

元素的哈希值是通过元素的 hashcode 方法 来获取的， HashSet 首先判断两个元素的哈希值，如果哈希值一样，接着会比较 equals 方法 如果 equls 结果为 true ，HashSet 就视为同一个元素。如果 equals 为 false 就不是同一个元素。

哈希值相同 equals 为 false 的元素是怎么存储呢,就是在同样的哈希值下顺延（可以认为哈希值相同的元素放在一个哈希桶中）。也就是哈希一样的存一列。

hashtable

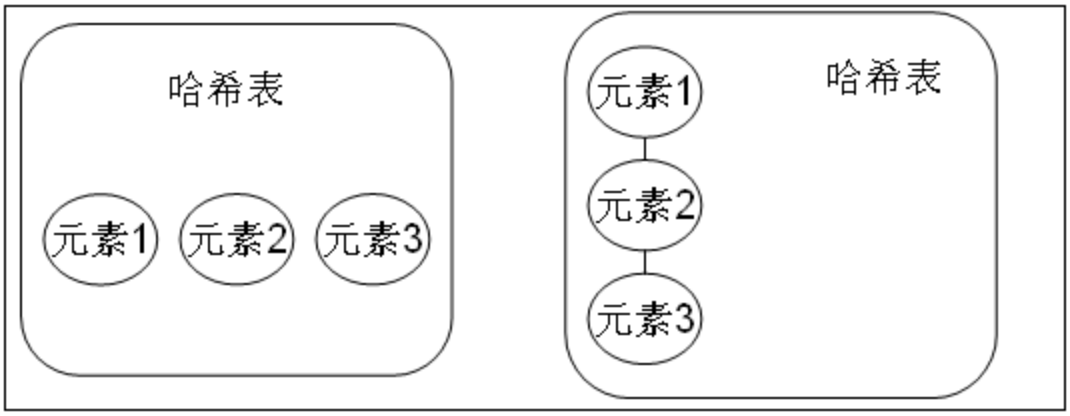


图 1：hashCode 值不相同的情况

图 2：hashCode 值相同，但 equals 不相同的情况。

HashSet：通过 hashCode 值来确定元素在内存中的位置。一个 hashCode 位置上可以存放多个元素。

当 hashcode() 值相同 equals() 返回为 true 时,hashset 集合认为这两个元素是相同的元素.只存储一个（重复元素无法放入）。调用原理:先判断 hashcode 方法的值,如果相同才会去判断 equals 如果不相同,是不会调用 equals 方法的。

HashSet 到底是如何判断两个元素重复。

通过 hashCode 方法和 equals 方法来保证元素的唯一性，add()返回的是 boolean 类型

判断两个元素是否相同，先要判断元素的 hashCode 值是否一致，只有在该值一致的情况下，才会判断 equals 方法，如果存储在 HashSet 中的两个对象 hashCode 方法的值相同 equals 方法返回的结果是 true，那么 HashSet 认为这两个元素是相同元素，只存储一个（重复元素无法存入）。

注意：HashSet 集合在判断元素是否相同先判断 hashCode 方法，如果相同才会判断 equals。如果不相同，是不会调用 equals 方法的。

HashSet 和 ArrayList 集合都有判断元素是否相同的方法，

boolean contains(Object o)

HashSet 使用 hashCode 和 equals 方法，ArrayList 使用了 equals 方法

练习：使用 HashSet 存储字符串，并尝试添加重复字符串

回顾 String 类的 equals()、hashCode()两个方法。

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class Demo4 {
    public static void main(String[] args) {
        // Set 集合存和取的顺序不一致。
        Set hs = new HashSet();
        hs.add("世界军事");
```

```
        hs.add("兵器知识");
        hs.add("舰船知识");
        hs.add("汉和防务");

        // 返回此 set 中的元素的数量
        System.out.println(hs.size()); // 4

        // 如果此 set 尚未包含指定元素，则返回 true
        boolean add = hs.add("世界军事"); // false
        System.out.println(add);

        // 返回此 set 中的元素的数量
        System.out.println(hs.size());// 4
        Iterator it = hs.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

使用 HashSet 存储自定义对象，并尝试添加重复对象（对象的重复的判定）

```
package cn.itcast.gz.map;

import java.util.HashSet;
import java.util.Iterator;

public class Demo4 {
    public static void main(String[] args) {
        HashSet hs = new HashSet();
        hs.add(new Person("jack", 20));
        hs.add(new Person("rose", 20));
        hs.add(new Person("hmm", 20));
        hs.add(new Person("lilei", 20));
        hs.add(new Person("jack", 20));

        Iterator it = hs.iterator();
        while (it.hasNext()) {
            Object next = it.next();
            System.out.println(next);
        }
    }
}
```

```
class Person {
    private String name;
    private int age;

    Person() {

    }

    public Person(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() {
        System.out.println("hashCode:" + this.name);
        return this.name.hashCode() + age * 37;
    }

    @Override
    public boolean equals(Object obj) {
        System.out.println(this + "---equals---" + obj);
        if (obj instanceof Person) {
            Person p = (Person) obj;
            return this.name.equals(p.name) && this.age == p.age;
        } else {
            return false;
```

```
        }
    }

    @Override
    public String toString() {

        return "Person@name:" + this.name + " age:" + this.age;
    }

}
```

问题：现在有一批数据，要求不能重复存储元素，而且要排序。ArrayList 、LinkedList 不能去除重复数据。HashSet 可以去除重复，但是是无序。

2.4.2. TreeSet

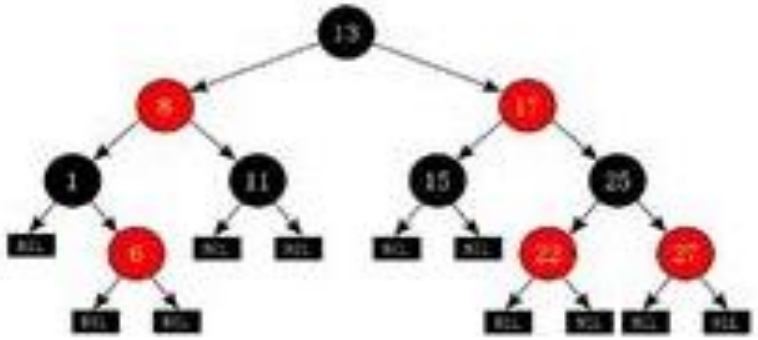
案例：使用 TreeSet 集合存储字符串元素，并遍历

```
import java.util.TreeSet;

public class Demo5 {
    public static void main(String[] args) {
        TreeSet ts = new TreeSet();
        ts.add("ccc");
        ts.add("aaa");
        ts.add("ddd");
        ts.add("bbb");

        System.out.println(ts); // [aaa, bbb, ccc, ddd]

    }
}
```

```
---| Itreable      接口 实现该接口可以使用增强 for 循环
            ---| Collection      描述所有集合共性的接口
                ---| List 接口     有序，可以重复，有角标的集合
                        ---| ArrayList
                        ---|  LinkedList
                ---| Set 接口      无序，不可以重复的集合
                        ---| HashSet  线程不安全，存取速度快。底层是以
                                    hash 表实现的。
```

---| TreeSet	红-黑树的数据结构，默认对元素进行自然排序（String）。如果在比较的时候两个对象返回值为 0，那么元素重复。

红-黑树
红黑树是一种特定类型的二叉树



红黑树算法的规则：左小右大。
既然 TreeSet 可以自然排序,那么 TreeSet 必定是有排序规则的。
1:让存入的元素自定义比较规则。

2:给 TreeSet 指定排序规则。
方式一：元素自身具备比较性

元素自身具备比较性，需要元素实现 Comparable 接口，重写 compareTo 方法，也就是让元素自身具备比较性，这种方式叫做元素的自然排序也叫做默认排序。

方式二：容器具备比较性

当元素自身不具备比较性，或者自身具备的比较性不是所需要的。那么此时可以让容器自身具备。需要定义一个类实现接口 Comparator，重写 compare 方法，并将该接口的子类实例对象作为参数传递给 TreeMap 集合的构造方法。

注意：当 Comparable 比较方式和 Comparator 比较方式同时存在时，以 Comparator 的比较方式为主；

注意：在重写 compareTo 或者 compare 方法时，必须要明确比较的主要条件相等时要比较次要条件。（假设姓名和年龄一直的人为相同的人，如果想要对人按照年龄的大小来排序，如果年龄相同的人，需要如何处理？不能直接 return 0，因为可能姓名不同（年龄相同姓名不同的人是不同的人）。此时就需要进行次要条件判断（需要判断姓名），只有姓名和年龄同时相等的才可以返回 0.）

通过 return 0 来判断唯一性。

问题:为什么使用 TreeSet 存入字符串,字符串默认输出是按升序排列的?因为字符串实现

了一个接口,叫做 Comparable 接口.字符串重写了该接口的 compareTo 方法,所以 String 对象具备了比较性.那么同样道理,我的自定义元素(例如 Person 类,Book 类)想要存入 TreeSet 集合,就需要实现该接口,也就是要让自定义对象具备比较性.
存入 TreeSet 集合中的元素要具备比较性.
比较性要实现 Comparable 接口，重写该接口的 compareTo 方法
TreeSet 属于 Set 集合，该集合的元素是不能重复的，TreeSet 如何保证元素的唯一性通过 compareTo 或者 compare 方法中的来保证元素的唯一性。
添加的元素必须要实现 Comparable 接口。当 compareTo()函数返回值为 0 时，说明两个对象相等，此时该对象不会添加进来。
比较器接口

```
----| Comparable
          compareTo(Object  o)      元素自身具备比较性
----| Comparator
          compare( Object o1, Object o2 )   给容器传入比较器
```

TreeSet集合**排序的两种方式：**
一，让元素自身具备比较性。
也就是元素需要实现Comparable接口，覆盖compareTo 方法。
这种方式也作为元素的自然排序，也可称为默认排序。
年龄按照搜要条件，年龄相同再比姓名。

```
import java.util.TreeSet;

public class Demo4 {
    public static void main(String[] args) {
        TreeSet ts = new TreeSet();
        ts.add(new Person("aa", 20, "男"));
        ts.add(new Person("bb", 18, "女"));
        ts.add(new Person("cc", 17, "男"));
        ts.add(new Person("dd", 17, "女"));
        ts.add(new Person("dd", 15, "女"));
        ts.add(new Person("dd", 15, "女"));


        System.out.println(ts);
        System.out.println(ts.size()); // 5

    }
}

class Person implements Comparable {
    private String name;
    private int age;
    private String gender;
```

```
    public Person() {

    }

    public Person(String name, int age, String gender) {

        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    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;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    @Override
    public int hashCode() {
        return name.hashCode() + age * 37;
    }

    public boolean equals(Object obj) {
        System.err.println(this + "equals :" + obj);
        if (!(obj instanceof Person)) {
            return false;
```

```
        }
        Person p = (Person) obj;
        return this.name.equals(p.name) && this.age == p.age;

    }

    public String toString() {
        return "Person [name=" + name + ", age=" + age + ", gender=" + gender
                + "]";
    }

    @Override
    public int compareTo(Object obj) {

        Person p = (Person) obj;
        System.out.println(this+" compareTo:"+p);
        if (this.age > p.age) {
            return 1;
        }
        if (this.age < p.age) {
            return -1;
        }
        return this.name.compareTo(p.name);
    }

}
```

二，让容器自身具备比较性，自定义比较器。
需求：当元素自身不具备比较性，或者元素自身具备的比较性不是所需的。
那么这时只能让容器自身具备。
定义一个类实现Comparator 接口，覆盖compare方法。
并将该接口的子类对象作为参数传递给TreeSet集合的构造函数。
当Comparable比较方式，及Comparator比较方式同时存在，以Comparator比较方式为主。

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

public class Demo5 {
    public static void main(String[] args) {
        TreeSet ts = new TreeSet(new MyComparator());
        ts.add(new Book("think in java", 100));
        ts.add(new Book("java 核心技术", 75));
        ts.add(new Book("现代操作系统", 50));
```

```
        ts.add(new Book("java就业教程", 35));
        ts.add(new Book("think in java", 100));
        ts.add(new Book("ccc in java", 100));

        System.out.println(ts);
    }
}

class MyComparator implements Comparator {

    public int compare(Object o1, Object o2) {
        Book b1 = (Book) o1;
        Book b2 = (Book) o2;
        System.out.println(b1+" comparator "+b2);
        if (b1.getPrice() > b2.getPrice()) {
            return 1;
        }
        if (b1.getPrice() < b2.getPrice()) {
            return -1;
        }
        return b1.getName().compareTo(b2.getName());
    }

}

class Book {
    private String name;
    private double price;

    public Book() {

    }

    public String getName() {
        return name;
    }

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

    public double getPrice() {
        return price;
    }
```

```
    public void setPrice(double price) {
        this.price = price;
    }

    public Book(String name, double price) {

        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Book [name=" + name + ", price=" + price + "]";
    }

}
```

TreeSet 练习
将字符串中的数值进行排序。
例如 String str="8 10 15 5 2 7"; 2,5,7,8,10,15

使用 TreeSet 完成。
思路:1，将字符串切割。
2，可以将这些对象存入 TreeSet 集合。
因为 TreeSet 自身具备排序功能。

```
public class Demo5 {
    public static void main(String[] args) {
        String str = "8 10 15 5 2 7";
        String[] strs = str.split(" ");
        TreeSet ts = new TreeSet();
        for (int x = 0; x < strs.length; x++) {
            int y = Integer.parseInt(strs[x]);
            ts.add(y);
        }
        System.out.println(ts);
    }
}
```

2.4.3. LinkedHashSet

会保存插入的顺序。

看到 array，就要想到角标。
看到 link，就要想到 first，last。
看到 hash，就要想到 hashCode,equals.
看到 tree，就要想到两个接口。Comparable，Comparator。
练习:去除数组中的重复元素.

3. Map

如果程序中存储了几百万个学生，而且经常需要使用学号来搜索某个学生，那么这个需求有效的数据结构就是 Map。Map 是一种依照键（key）存储元素的容器，键（key）很像下标，在 List 中下标是整数。在 Map 中键（key）可以使任意类型的对象。Map 中不能有重复的键（Key），每个键（key）都有一个对应的值（value）。一个键（key）和它对应的值构成 map 集合中的一个元素。
Map 中的元素是两个对象，一个对象作为键，一个对象作为值。键不可以重复，但是值可以重复。
看顶层共性方法找子类特有对象.
Map 与 Collection 在集合框架中属并列存在
Map 存储的是键值对
Map 存储元素使用 put 方法，Collection 使用 add 方法
Map 集合没有直接取出元素的方法，而是先转成 Set 集合，在通过迭代获取元素
Map 集合中键要保证唯一性
也就是 Collection 是单列集合， Map 是双列集合。
总结:
Map 一次存一对元素， Collection 一次存一个。Map 的键不能重复，保证唯一。
Map 一次存入一对元素,是以键值对的形式存在.键与值存在映射关系.一定要保证键的唯一性.
查看 api 文档:

interface Map<K,V>
K - 此映射所维护的键的类型
V - 映射值的类型
概念
将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。
特点
Key 和 Value 是 1 对 1 的关系，如：门牌号 ：家 老公:老婆
双列集合

Map 学习体系:

---| Map 接口 将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。
---| HashMap 采用哈希表实现，所以无序
---| TreeMap 可以对健进行排序

---|Hashtable:
底层是哈希表数据结构，线程是同步的，不可以存入 null 键，null 值。
效率较低，被 HashMap 替代。
---|HashMap:
底层是哈希表数据结构，线程是不同步的，可以存入 null 键，null 值。
要保证键的唯一性，需要覆盖 hashCode 方法，和 equals 方法。
---| LinkedHashMap:
该子类基于哈希表又融入了链表。可以 Map 集合进行增删提高效率。
---|TreeMap:
底层是二叉树数据结构。可以对 map 集合中的键进行排序。需要使用 Comparable 或者 Comparator 进行比较排序。return 0，来判断键的唯一性。

常见方法

1、添加:
1、V put(K key, V value) （可以相同的 key 值，但是添加的 value 值会覆盖前面的，返回值是前一个，如果没有就返回 null）
2、putAll(Map<? extends K,? extends V> m) 从指定映射中将所有映射关系复制到此映射中（可选操作）。
2、删除
1、remove() 删除关联对象，指定 key 对象
2、clear() 清空集合对象
3、获取
1: value get(key); 可以用于判断键是否存在的情况。当指定的键不存在的时候，返回的是 null。

3、判断:
1、boolean isEmpty() 长度为 0 返回 true 否则 false
2、boolean containsKey(Object key) 判断集合中是否包含指定的 key
3、boolean containsValue(Object value) 判断集合中是否包含指定的 value
4、长度:
Int size ()

添加:
该案例使用了 HashMap，建立了学生姓名和年龄之间的映射关系。并试图添加重复的键。

```
import java.util.HashMap;
import java.util.Map;
```

```
public class Demo1 {
    public static void main(String[] args) {
        // 定义一个Map的容器对象
        Map<String, Integer > map1 = new HashMap<String, Integer >();
        map1.put("jack", 20);
        map1.put("rose", 18);
        map1.put("lucy", 17);
        map1.put("java", 25);
        System.out.println(map1);
            // 添加重复的键值（值不同）,会返回集合中原有（重复键）的值,
         System.out.println(map1.put("jack", 30)); //20

        Map<String, Integer> map2 = new HashMap<String, Integer>();
        map2.put("张三丰", 100);
        map2.put("虚竹", 20);
        System.out.println("map2:" + map2);
         // 从指定映射中将所有映射关系复制到此映射中。
        map1.putAll(map2);
        System.out.println("map1:" + map1);
         //
    }
}
```

删除:

```
// 删除：
        // remove() 删除关联对象，指定key对象
        // clear() 清空集合对象

        Map<String, Integer> map1 = new HashMap<String, Integer>();
        map1.put("jack", 20);
        map1.put("rose", 18);
        map1.put("lucy", 17);
        map1.put("java", 25);
        System.out.println(map1);
       // 指定key，返回删除的键值对映射的值。
        System.out.println("value:" + map1.remove("java"));
        map1.clear();
        System.out.println("map1:" + map1);
```

获取:

```
// 获取：
```

```
        // V get(Object key) 通过指定的key对象获取value对象
        // int size() 获取容器的大小
        Map<String, Integer> map1 = new HashMap<String, Integer>();
        map1.put("jack", 20);
        map1.put("rose", 18);
        map1.put("lucy", 17);
        map1.put("java", 25);
        System.out.println(map1);
        // V get(Object key) 通过指定的key对象获取value对象
        // int size() 获取容器的大小
        System.out.println("value:" + map1.get("jack"));
        System.out.println("map.size:" + map1.size());
```

判断:

```
// 判断:
        // boolean isEmpty() 长度为0返回true否则false
        // boolean containsKey(Object key) 判断集合中是否包含指定的key
        // boolean containsValue(Object value)

        Map<String, Integer> map1 = new HashMap<String, Integer>();
        map1.put("jack", 20);
        map1.put("rose", 18);
        map1.put("lucy", 17);
        map1.put("java", 25);
        System.out.println(map1);
        System.out.println("isEmpty:" + map1.isEmpty());
        System.out.println("containskey:" + map1.containsKey("jack"));
        System.out.println("containsvalues:"                          +
 map1.containsValue(100));
```

遍历 Map 的方式:

1、将 map 集合中所有的键取出存入 set 集合。

Set<K> keySet() 返回所有的 key 对象的 Set 集合

再通过 get 方法获取键对应的值。

2、 values() ，获取所有的值.

Collection<V> values()不能获取到 key 对象

3、 Map.Entry 对象 推荐使用 重点

Set<Map.Entry<k,v>> entrySet()

将 map 集合中的键值映射关系打包成一个对象

Map.Entry 对象通过 Map.Entry 对象的 getKey，getValue 获取其键和值。

第一种方式:使用 keySet

将 Map 转成 Set 集合（keySet()），通过 Set 的迭代器取出 Set 集合中的每一个元素（Iterator）就是 Map 集合中的所有的键，再通过 get 方法获取键对应的值。

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class Demo2 {
    public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<Integer, String>();
        map.put(1, "aaaa");
        map.put(2, "bbbb");
        map.put(3, "cccc");
        System.out.println(map);

        //
        // 获取方法:
        // 第一种方式: 使用keySet
        // 需要分别获取key和value，没有面向对象的思想
        // Set<K> keySet() 返回所有的key对象的Set集合

        Set<Integer> ks = map.keySet();
        Iterator<Integer> it = ks.iterator();
        while (it.hasNext()) {
            Integer key = it.next();
            String value = map.get(key);
            System.out.println("key=" + key + " value=" + value);
        }
    }
}
```

第二种方式: 通过 values 获取所有值,不能获取到 key 对象

```
public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<Integer, String>();
        map.put(1, "aaaa");
        map.put(2, "bbbb");
        map.put(3, "cccc");
                System.out.println(map);
         // 第二种方式:
        // 通过values 获取所有值,不能获取到key对象
        // Collection<V> values()
```

```
        Collection<String> vs = map.values();
        Iterator<String> it = vs.iterator();
        while (it.hasNext()) {
            String value = it.next();
            System.out.println(" value=" + value);
        }
}
```

第三种方式：Map.Entry

public static interface Map.Entry<K,V>

通过 Map 中的 entrySet()方法获取存放 Map.Entry<K,V>对象的 Set 集合。

Set<Map.Entry<K,V>> entrySet()

面向对象的思想将 map 集合中的键和值映射关系打包为一个对象，就是 Map.Entry ,将该对象存入 Set 集合,Map.Entry 是一个对象,那么该对象具备的 getKey,getValue 获得键和值。

```
public static void main(String[] args) {
        Map<Integer, String> map = new HashMap<Integer, String>();
        map.put(1, "aaaa");
        map.put(2, "bbbb");
        map.put(3, "cccc");
        System.out.println(map);
        // 第三种方式： Map.Entry对象 推荐使用 重点
        // Set<Map.Entry<K,V>> entrySet()


        // 返回的Map.Entry对象的Set集合 Map.Entry包含了key和value对象
        Set<Map.Entry<Integer, String>> es = map.entrySet();

        Iterator<Map.Entry<Integer, String>> it = es.iterator();

        while (it.hasNext()) {

            // 返回的是封装了key和value对象的Map.Entry对象
            Map.Entry<Integer, String> en = it.next();

            // 获取Map.Entry对象中封装的key和value对象
            Integer key = en.getKey();
            String value = en.getValue();

            System.out.println("key=" + key + " value=" + value);
        }
    }
```

3.1. HashMap

底层是哈希表数据结构，线程是不同步的，可以存入 null 键，null 值。要保证键的唯一性，需要覆盖 hashCode 方法，和 equals 方法。
案例：自定义对象作为 Map 的键。

```
package cn.itcast.gz.map;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.Set;

public class Demo3 {
    public static void main(String[] args) {
        HashMap<Person, String> hm = new HashMap<Person, String>();
        hm.put(new Person("jack", 20), "1001");
        hm.put(new Person("rose", 18), "1002");
        hm.put(new Person("lucy", 19), "1003");
        hm.put(new Person("hmm", 17), "1004");
        hm.put(new Person("ll", 25), "1005");
        System.out.println(hm);
        System.out.println(hm.put(new Person("rose", 18), "1006"));

        Set<Entry<Person, String>> entrySet = hm.entrySet();
        Iterator<Entry<Person, String>> it = entrySet.iterator();
        while (it.hasNext()) {
            Entry<Person, String> next = it.next();
            Person key = next.getKey();
            String value = next.getValue();
            System.out.println(key + " = " + value);
        }
    }
}

class Person {
    private String name;
    private int age;

    Person() {

    }
```

```
    public Person(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 this.name.hashCode() + age * 37;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Person) {
            Person p = (Person) obj;
            return this.name.equals(p.name) && this.age == p.age;
        } else {
            return false;
        }
    }

    @Override
    public String toString() {

        return "Person@name:" + this.name + " age:" + this.age;
    }
```

```
}
}
```

3.2. TreeMap

TreeMap 的排序，TreeMap 可以对集合中的键进行排序。如何实现键的排序？

方式一：元素自身具备比较性

和 TreeSet 一样原理，需要让存储在键位置的对象实现 Comparable 接口，重写 compareTo 方法，也就是让元素自身具备比较性，这种方式叫做元素的自然排序也叫做默认排序。

方式二：容器具备比较性

当元素自身不具备比较性，或者自身具备的比较性不是所需要的。那么此时可以让容器自身具备。需要定义一个类实现接口 Comparator，重写 compare 方法，并将该接口的子类实例对象作为参数传递给 TreeMap 集合的构造方法。

注意：当 Comparable 比较方式和 Comparator 比较方式同时存在时，以 Comparator 的比较方式为主；

注意：在重写 compareTo 或者 compare 方法时，必须要明确比较的主要条件相等时要比较次要条件。（假设姓名和年龄一直的人为相同的人，如果想要对人按照年龄的大小来排序，如果年龄相同的人，需要如何处理？不能直接 return 0，以为可能姓名不同（年龄相同姓名不同的人是不同的人）。此时就需要进行次要条件判断（需要判断姓名），只有姓名和年龄同时相等的才可以返回 0.）

通过 return 0 来判断唯一性。

```
import java.util.TreeMap;

public class Demo4 {
    public static void main(String[] args) {
        TreeMap<String, Integer> tree = new TreeMap<String, Integer>();
        tree.put("张三", 19);
        tree.put("李四", 20);
        tree.put("王五", 21);
        tree.put("赵六", 22);
        tree.put("周七", 23);
        tree.put("张三", 24);
        System.out.println(tree);
        System.out.println("张三".compareTo("李四"));//-2094
    }
}
```

自定义元素排序

```
package cn.itcast.gz.map;
```

```
import java.util.Comparator;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.Set;
import java.util.TreeMap;

public class Demo3 {
    public static void main(String[] args) {
        TreeMap<Person, String> hm = new TreeMap<Person, String>(
                new MyComparator());
        hm.put(new Person("jack", 20), "1001");
        hm.put(new Person("rose", 18), "1002");
        hm.put(new Person("lucy", 19), "1003");
        hm.put(new Person("hmm", 17), "1004");
        hm.put(new Person("ll", 25), "1005");
        System.out.println(hm);
        System.out.println(hm.put(new Person("rose", 18), "1006"));

        Set<Entry<Person, String>> entrySet = hm.entrySet();
        Iterator<Entry<Person, String>> it = entrySet.iterator();
        while (it.hasNext()) {
            Entry<Person, String> next = it.next();
            Person key = next.getKey();
            String value = next.getValue();
            System.out.println(key + " = " + value);
        }
    }
}

class MyComparator implements Comparator<Person> {

    @Override
    public int compare(Person p1, Person p2) {
        if (p1.getAge() > p2.getAge()) {
            return -1;
        } else if (p1.getAge() < p2.getAge()) {
            return 1;
        }
        return p1.getName().compareTo(p2.getName());
    }

}

class Person implements Comparable<Person> {
```

```
    private String name;
    private int age;

    Person() {

    }

    public Person(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 this.name.hashCode() + age * 37;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Person) {
            Person p = (Person) obj;
            return this.name.equals(p.name) && this.age == p.age;
        } else {
            return false;
        }
    }
```

```
    @Override
    public String toString() {

        return "Person@name:" + this.name + " age:" + this.age;
    }

    @Override
    public int compareTo(Person p) {

        if (this.age > p.age) {
            return 1;
        } else if (this.age < p.age) {
            return -1;
        }
        return this.name.compareTo(p.name);
    }

}
```

注意：Set 的元素不可重复，Map 的键不可重复，如果存入重复元素如何处理
Set 元素重复元素不能存入 add 方法返回 false
Map 的重复健将覆盖旧键，将旧值返回。

4. Collections 与 Arrays

集合框架中的工具类：特点：该工具类中的方法都是静态的。

Collections：常见方法：
1，对 list 进行二分查找：
前提该集合一定要有序。

```
int binarySearch(list,key);
//必须根据元素自然顺序对列表进行升级排序
//要求 list 集合中的元素都是 Comparable 的子类。
 int binarySearch(list,key,Comparator);
```

2，对 list 集合进行排序。

```
sort(list);
//对 list 进行排序,其实使用的事 list 容器中的对象的 compareTo 方法
sort(list,comaprator);
//按照指定比较器进行排序
```

3，对集合取最大值或者最小值。

```
      max(Collection)
      max(Collection,comparator)
      min(Collection)
      min(Collection,comparator)
4，对 list 集合进行反转。
      reverse(list);
5，对比较方式进行强行逆转。
      Comparator reverseOrder();
      Comparator reverseOrder(Comparator);
6，对 list 集合中的元素进行位置的置换。
      swap(list,x,y);
7，对 list 集合进行元素的替换。如果被替换的元素不存在，那么原集合不变。
      replaceAll(list,old,new);
8，可以将不同步的集合变成同步的集合。
      Set synchronizedSet(Set<T> s)
      Map synchronizedMap(Map<K,V> m)
      List synchronizedList(List<T> list)
9．如果想要将集合变数组：
      可以使用Collection 中的toArray 方法。注意：是Collection不是Collections工具类
      传入指定的类型数组即可，该数组的长度最好为集合的 size。
```

Arrays:用于对数组操作的工具类

```
1，二分查找,数组需要有序
    binarySearch(int[])
    binarySearch(double[])

2，数组排序
    sort(int[])
    sort(char[])……
2，将数组变成字符串。
    toString(int[])
3，复制数组。
    copyOf();
4，复制部分数组。
     copyOfRange():
5，比较两个数组是否相同。
     equals(int[],int[]);
6，将数组变成集合。
     List asList(T[]);
     这样可以通过集合的操作来操作数组中元素，
     但是不可以使用增删方法，add，remove。因为数组长度是固定的，会出现
     UnsupportOperationExcetion。
     可以使用的方法：contains，indexOf。。。
     如果数组中存入的基本数据类型，那么 asList 会将数组实体作为集合中的元素。
```

如果数组中的存入的引用数据类型，那么 asList 会将数组中的元素作为集合中的元素。

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Arrays;
import java.util.List;
class Demo1
{
    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(4);
        list.add(3);
        list.add(1);
        list.add(2);
        list.add(3);
        // 排序
        Collections.sort(list);
        // 折半查找的前提是排序好的元素
        System.out.println( Collections.binarySearch( list , 8 ) );  //
找不到返回-插入点-1
        // 反序集合输出
        Collections.reverse( list );
        System.out.println( list );
        // 求最值
        System.out.println( Collections.max( list ) );   // 4
        // fill()  使用指定的元素替换指定集合中的所有元素
        // Collections.fill( list, 5 );
        System.out.println( list );

        // 将数组转换为集合
        Integer is[] = new  Integer[]{6,7,8};
        List<Integer> list2 =  Arrays.asList(is);
        list.addAll( list2 );
        System.out.println( list );

        // 将List转换为数组
        Object [] ins =  list.toArray();
        System.out.println( Arrays.toString( ins ) );


    }
}
```

集合的练习

问题： 定义一个 Person 数组，将 Person 数组中的重复对象剔除？

思路：

1. 描述一个 Person 类
2. 将数组转换为 Arrays.asList() List
3. Set addAll(list)
4. hashCode()且 equals()

```
import java.util.Arrays;
import java.util.Set;
import java.util.List;
import java.util.HashSet;

// 1. 描述 Person 类
class Person {
    public String name;
    public int age;

    public Person() {
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String toString() {

        return getClass().getName() + " : name=" + this.name + " age="
                + this.age;


    }

    // 4. 重写 hashCode 和 equals()
    public int hashCode() {

        return this.age;
    }

    public boolean equals(Object o) {
        Person p = null;
        if (o instanceof Person)
            p = (Person) o;
        return this.name.equals(p.name) && (this.age == p.age);
    }
```

```
}

class Demo2 {
    public static void main(String[] args) {
        Person[] ps = new Person[] { new Person("jack", 34),
                new Person("lucy", 20), new Person("lili", 10),
                new Person("jack", 34) };
        // 遍历数组
        System.out.println(Arrays.toString(ps));
        // 2. 将自定义对象数组转换为List集合
        List<Person> list = Arrays.asList(ps);
        // 3. 将List转换为Set
        Set<Person> set = new HashSet<Person>();
        set.addAll(list);
        System.out.println(set);

    }
}
```

5. 泛型(Generic)

当集合中存储的对象类型不同时，那么会导致程序在运行的时候的转型异常

```
import java.util.ArrayList;
import java.util.Iterator;

public class Demo5 {
    public static void main(String[] args) {
        ArrayList arr = new ArrayList();
        arr.add(new Tiger("华南虎"));
        arr.add(new Tiger("东北虎"));
        arr.add(new Sheep("喜羊羊"));
        System.out.println(arr);
        Iterator it = arr.iterator();
        while (it.hasNext()) {
            Object next = it.next();
            Tiger t = (Tiger) next;
            t.eat();
        }

    }
}
```

```
class Tiger {
    String name;

    public Tiger() {

    }

    public Tiger(String name) {
        this.name = name;
    }

    @Override
    public String toString() {

        return "Tiger@name:" + this.name;
    }

    public void eat() {
        System.out.println(this.name + "吃羊");
    }
}

class Sheep {
    String name;

    public Sheep() {

    }

    public Sheep(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Sheep@name:" + this.name;
    }

    public void eat() {
        System.out.println(this.name + "吃青草");
    }
}
```

原因 :发现虽然集合可以存储任意对象,但是如果需要使用对象的特有方法,那么就需要类型

转换,如果集合中存入的对象不同,可能引发类型转换异常.

```
[Tiger@name:华南虎, Tiger@name:东北虎, Sheep@name:喜羊羊]
华南虎吃羊
东北虎吃羊
Exception in thread "main" java.lang.ClassCastException:
cn.itcast.gz.map.Sheep cannot be cast to cn.itcast.gz.map.Tiger
    at cn.itcast.gz.map.Demo5.main(Demo5.java:17)
```

出现问题:

存入的是特定的对象,取出的时候是 Object 对象,需要强制类型转换,可能诱发类型转换异常.

无法控制存入的是什么类型的对象,取出对象的时候进行强转时可能诱发异常.而且在编译时期无法发现问题.

虽然可以再类型转换的时候通过 if 语句进行类型检查(instanceof),但是效率较低.(例如吃饭的时候,还需要判断米饭里有没有沙子,吃饭效率低).可以通过给容器加限定的形式规定容器只能存储一种类型的对象.

就像给容器贴标签说明该容器中只能存储什么样类型的对象。

所以在 jdk5.0 后出现了泛型

泛型应用:

格式

1. 集合类<类类型> 变量名 = new 集合类<类类型>();

```
public class Demo5 {
    public static void main(String[] args) {
        // 使用泛型后,规定该集合只能放羊,老虎就进不来了.
        ArrayList<Sheep> arr = new ArrayList<Sheep>();
        arr.add(new Sheep("美羊羊"));
        arr.add(new Sheep("懒洋洋"));
        arr.add(new Sheep("喜羊羊"));
        // 编译失败
        // arr.add(new Tiger("东北虎"));
        System.out.println(arr);
        Iterator<Sheep> it = arr.iterator();
        while (it.hasNext()) {
            // 使用泛型后,不需要强制类型转换了
            Sheep next = it.next();
            next.eat();
        }

    }
}
```

1. 将运行时的异常提前至编译时发生。
2. 获取元素的时候无需强转类型，就避免了类型转换的异常问题

格式　通过<> 来指定容器中元素的类型.
什么时候使用泛型:当类中操作的引用数据类型不确定的时候,就可以使用泛型类.
JDK5.0 之前的 Comparable

```
package java.lang;
public interface Comparable {

    public int compareTo(Object o);
}
```

JDK5.0 之后的 Comparable

```
package java.lang;
public interface Comparable<T> {

    public int compareTo(T o);
}
```

这里的<T>表示泛型类型,随后可以传入具体的类型来替换它.

细节一
声明好泛型类型之后,集合中只能存放特定类型元素

```
public class Demo6 {
    public static void main(String[] args) {
        //创建一个存储字符串的list
        ArrayList<String> arr=new ArrayList<String>();
        arr.add("gz");
        arr.add("itcast");
        //存储非字符串编译报错.
        arr.add(1);
    }
}
```

细节二:
泛型类型必须是引用类型

```
public class Demo6 {
    public static void main(String[] args) {
        // 泛型类型必须是引用类型,也就是说集合不能存储基本数据类型
        // ArrayList<int> arr2=new ArrayList<int>();

        // 使用基本数据类型的包装类
        ArrayList<Integer> arr2 = new ArrayList<Integer>();


    }
}
```

细节三: 使用泛型后取出元素不需要类型转换.

```
public class Demo6 {
    public static void main(String[] args) {

        ArrayList<String> arr = new ArrayList<String>();
        arr.add("gzitcast");
        arr.add("cditcast");
        arr.add("bjitcast");
        //使用泛型后取出元素不需要类型转换.
        String str=arr.get(0);
        System.out.println();
    }
}
```

5.1. 泛型方法

需求：写一个函数，调用者传递什么类型的变量，该函数就返回什么类型的变量?
实现一:
由于无法确定具体传递什么类型的数据.那么方法的形参就定义为 Object 类型.返回值也就是 Object 类型.但是使用该函数时需要强制类型转换.

```
private Object getDate(Object obj) {
        return obj;
}
```

当不进行强制类型转换能否写出该功能.?

目前所学的知识无法解决该问题
就需要使用泛型类解决
使用的泛型的自定义来解决以上问题。
泛型： 就是将类型当作变量处理。规范泛型的定义一般是一个大写的任意字母。

```
1. 函数上的泛型定义

        当函数中使用了一个不明确的数据类型，那么在函数上就可以进行泛型的定义。

        public <泛型的声明> 返回值类型  函数名( 泛型 变量名 ){
```

```
        }
```

```
public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5 };

        new Demo6().getData(5);

    }

    public <T> T getData(T data) {
        return data;
    }
```

细节：
使用泛型方法前需要进行泛型声明，使用一对尖括号 <泛型>，声明的位置在 static 后返回值类型前。
当一个类中有多个函数声明了泛型，那么该泛型的声明可以声明在类上。

5.2. 泛型类

格式

```
2. 类上的泛型声明

        修饰符 class 类名<泛型>{

        }
```

```
import java.util.Arrays;

public class Demo6<T> {
    public static void main(String[] args) {
        // 使用泛型类，创建对象的时候需要指定具体的类型
        new Demo6<Integer>().getData(5);
    }

    public T getData(T data) {
        return data;
    }
```

```
    // 反序任意类型数组
    public void reverse(T[] arr) {
        int start = 0;
        int end = arr.length - 1;
        for (int i = 0; i < arr.length; i++) {
            if (start < end) {
                T temp = arr[start];
                arr[start] = arr[end];
                arr[end] = temp;
            }
        }

}
```

在泛型类中定义一个静态方法

```
public class Demo6<T> {
    public static void main(String[] args) {
        System.out.println(getData2(100));
    }

    public T getData(T data) {
        return data;
    }

    //静态方法
    public static T getData2(T data) {
        return data;
    }

}
```

```
E:\0905\day13>javac Demo6.java
Demo6.java:26: 无法从静态上下文中引用非静态 类 T
        public static T getData2(T data) {
                        ^
Demo6.java:26: 无法从静态上下文中引用非静态 类 T
        public static T getData2(T data) {
                        ^
2 错误
```

注意：静态方法不可以使用类中定义的泛型

因为类中的泛型需要在对象初始化时指定具体的类型，而静态优先于对象存在。那么类中的静态方法就需要单独进行泛型声明，声明泛型一定要写在 static 后，返回值类型之前

泛型类细节：

1、创建对象的时候要指定泛型的具体类型

2、创建对象时可以不指定泛型的具体类型(和创建集合对象一眼)。默认是 Object，例如我们使用集合存储元素的时候没有使用泛型就是那么参数的类型就是 Object
3、类上面声明的泛型只能应用于非静态成员函数，如果静态函数需要使用泛型，那么需要在函数上独立声明。
4、如果建立对象后指定了泛型的具体类型，那么该对象操作方法时，这些方法只能操作一种数据类型。
5、所以既可以在类上的泛型声明，也可以在同时在该类的方法中声明泛型。

泛型练习：
定义泛型成员

```
public class Demo7 {
    public static void main(String[] args) {
        Father<String> f = new Father<String>("jack");
        System.out.println(f.getT());
        Father<Integer> f2 = new Father<Integer>(20);
        System.out.println(f2.getT());
    }

}

class Father<T> {
    private T t;

    public Father() {

    }

    public Father(T t) {
        super();
        this.t = t;
    }

    public T getT() {
        return t;
    }

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

}
```

如果 Father 类有子类，子类该如何实现

```
public class Demo7 {
```

```
    public static void main(String[] args) {
        Father<String> f = new Father<String>("jack");
        System.out.println(f.getT());
        Father<Integer> f2 = new Father<Integer>(20);
        System.out.println(f2.getT());
    }

}

class Father<T> {
    private T t;

    public Father() {

    }

    public Father(T t) {
        super();
        this.t = t;
    }

    public T getT() {
        return t;
    }

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

}
//子类指定了具体的类型
class Son extends Father<String>{

}
//子类也需要使用泛型
class Son3<T> extends Father<T>{

}
//错误写法，父类上定义有泛型需要进行处理
class Son2 extends Father<T>{

}
```

5.3. 泛型接口

```
public class Demo8 {
    public static void main(String[] args) {
        MyInter<String> my = new MyInter<String>();
        my.print("泛型");

        MyInter2 my2 = new MyInter2();
        my.print("只能传字符串");
    }
}

interface Inter<T> {
    void print(T t);
}

// 实现不知为何类型时可以这样定义
class MyInter<T> implements Inter<T> {
    public void print(T t) {
        System.out.println("myprint:" + t);
    }
}
//使用接口时明确具体类型。
class MyInter2 implements Inter<String> {

    @Override
    public void print(String t) {
        System.out.println("myprint:" + t);

    }

}
```

5.4. 泛型通配符

需求：
定义一个方法，接收一个集合对象(该集合有泛型)，并打印出集合中的所有元素。
例如集合对象如下格式:

```
Collection<Person> coll = new ArrayList<Person>();
        coll.add(new Person("jack", 20));
```

```
        coll.add(new Person("rose", 18));
        Collection<Object> coll2 = new ArrayList<Object>();
        coll2.add(new Object());
        coll2.add(new Object());
        coll2.add(new Object());

        Collection<String> coll3 = new ArrayList<String>();
        coll3.add("abc");
        coll3.add("ddd");
        coll3.add("eee");
```

分析,集合对象中的元素的类型是变化的,方法的形参的那么泛型类型就只能定义为 Object 类型.

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.Iterator;

public class Demo9 {
    public static void main(String[] args) {
        ArrayList<Object> arr = new ArrayList<Object>();
        arr.add(new Object());
        arr.add("String");
        print(arr);

        //将集合的泛型设置类String类型，是Object子类
        HashSet<String> hs = new HashSet<String>();
        hs.add("hello");
        hs.add("jack");
        //由于print方法接收的集合进行了元素限定，只接受限定为Object类型的集合,编译不通过
        //print(hs);
    }

    public static void print(Collection<Object> coll) {
        Iterator<Object> it = coll.iterator();
        while (it.hasNext()) {
            Object next = it.next();
            System.out.println(next);
        }
    }
}
```

但是,由于 print 方法接收的集合进行了元素限定，只接受限定为 Object 类型的集合,编译不

通过该问题如何解决?
可以把方法的形参的泛型去掉,那么方法中就把集合中的元素当做 Object 类型处理.
也可以使用使用泛型通配符

```
public class Demo9 {
    public static void main(String[] args) {
        ArrayList<Object> arr = new ArrayList<Object>();
        arr.add(new Object());
        arr.add("String");
        print(arr);

        // 将集合的泛型设置类String类型，是Object子类
        HashSet<String> hs = new HashSet<String>();
        hs.add("hello");
        hs.add("jack");
        // 使用泛型通配符，编译通过。
        print(hs);
    }

    public static void print(Collection<?> coll) {
        Iterator<?> it = coll.iterator();
        while (it.hasNext()) {

            Object next = it.next();
            System.out.println(next);
        }
    }
}
```

上述就使用了泛型通配符

```
通配符：?
public void show(List<?> list)
{
}
```

可以对类型进行限定范围。
? extends E: 接收E类型或者E的子类型。
? super E: 接收 E 类型或者 E 的父类型。

限定泛型通配符的边界
限定通配符的上边界:
extends
接收 **Number 类型或者 Number 的子类型**

```
正确：Vector<? extends Number> x = new Vector<Integer>();
错误：Vector<? extends Number> x = new Vector<String>();
```

限定通配符的下边界

super

接收 Integer 或者 Integer 的父类型

```
正确：Vector<? super Integer> x = new Vector<Number>();
错误：Vector<? super Integer> x = new Vector<Byte>();
```

总结：

JDK5 中的泛型允许程序员在编写集合代码时，就限制集合的处理类型，从而把原来程序运行时可能发生问题，转变为编译时的问题，以此提高程序的可读性和稳定

注意：泛型是提供给 javac 编译器使用的，它用于限定集合的输入类型，让编译器在源代码级别上，即挡住向集合中插入非法数据。但编译器编译完带有泛形的 java 程序后，生成的 class 文件中将不再带有泛形信息，以此使程序运行效率不受到影响，这个过程称之为“擦除”。

泛型的基本术语，以 ArrayList<E>为例：<>念着 typeof

ArrayList<E>中的 E 称为类型参数变量

ArrayList<Integer>中的 Integer 称为实际类型参数

整个称为 ArrayList<E>泛型类型

整个 ArrayList<Integer>称为参数化的类型 ParameterizedType

最后：

关于数据结构可以查看如下网站：

http://www.cs.armstrong.edu/liang/animation/index.html 数据结构