API基础第四天：

回顾:

1. 什么是集合：
2. Collection接口：所有集合的顶级接口，下面有两个子接口：
    - List接口：线性表，可重复集合，并且有序
    - Set接口：不可重复集合，大部分实现类都是无序的
3. Collection的常用方法：
    - add()：添加元素
    - size()：获取元素个数
    - isEmpty()：判断是否为空集
    - clear()：清空集合
    - contains()：判断是否包含
    - remove()：删除元素
    - addAll()：将一个集合中是的所有数据添加到另一个集合中
    - containsAll()：判断一个集合中是否包含另一个集合中的所有元素
    - retainAll()：取交集(交集的留着)
    - removeAll()：删交集(交集的删掉)
4. 集合的遍历：
    - Collection接口提供统一遍历集合的方式：迭代器模式。通过iterator()可以获取Iterator接口
    - 迭代器遍历过程：问(hasNext())、取(next())、删(remove())，但删除并非是必要操作
5. 增强for循环：
    - 可以使用相同的方式遍历集合和数组，底层采用的是迭代器模式，所以遍历过程中不能增删元素
    - 语法：

    ```
    for(元素类型 元素 : 数组或集合){
        循环体
    }
    ```

6. 泛型：参数化类型，允许我们在使用类中，传入某个类型来规定它属性的类型、方法参数的类型、返回值的类型，用起来方便
7. 集合和数组转换：
    - 集合转数组：toArray()
    - 数组转集合：Arrays.asList()

精华笔记：

1. List接口：
    - 继承自Collection接口，List集合是可重复集合，并且有序，还提供了一套可以通过下标来操作元素的方法
    - 常见的实现类：

        - ArrayList：内部使用数组实现，查询性能更好(直接下标找到物理地址)、增删性能不好
        - LinkedList：内部使用链表实现，查询性能不好，首尾增删性能好

        > 注意：在对集合操作的增删性能没有特别苛刻的要求时，通常选择ArrayList

2. List集常用方法：
    - get()：根据下标获取元素
    - set()：将指定元素设置到指定位置，并返回被替换的元素(用时才接收)
    - 重载remove()：删除指定位置元素，并返回被删除的元素(用时才接收)
    - 重载add()：将指定元素添加到指定位置，理解为插入操作
    - subList()：获取集合的子集(含头不含尾)
3. 集合排序：
    - Collections为集合的工具类，里面定义了很多静态方法用于操作集合
    - Collections.sort(List list)方法：可以对list集合进行自然排序(从小到大)，但是其要求List集合中的元素必须是可比较的，判断是否可以比较的标准是元素是否实现了Comparable接口，若没有实现该接口则直接发生编译错误，但是在实际开发中，我们一般是不会不让我们自己写的类去实现Comparable接口的，因为这对我们的程序有侵入性。
    - 侵入性：当我们调用某个API功能时，其要求我们为其修改其它额外的代码，这个现象叫做侵入性。侵入性越强越不利于程序的后期维护，应尽量避免。
    - 建议使用重载的Collections.sort(List list, Comparator o);可以通过Comparator来自定义比较规则
4. Lambda表达式：
    - JDK1.8之后推出了一个新特性：Lambda表达式
    - 规则：
        - 不是任何匿名内部类都可以转换为Lambda表达式
        - Lambda表达式对接口的要求：只能是函数式接口
        - 函数式接口:接口中要求实现类必须重写的方法只有一个
    - 语法：

        ```
        (参数列表)->{
            方法体
        }
        ```

5. Set接口：
    - 不可重复集合，并且大部分实现类都是无序的
    - 小面试题：如何去重?

笔记：

1. List接口：
    - 继承自Collection接口，List集合是可重复集合，并且有序，还提供了一套可以通过下标来操作元素的方法
    - 常见的实现类：
        - ArrayList：内部使用数组实现，查询性能更好(直接下标找到物理地址)、增删性能不好
        - LinkedList：内部使用链表实现，查询性能不好，首尾增删性能好

> 注意：在对集合操作的增删性能没有特别苛刻的要求时，通常选择ArrayList

2. List集常用方法：
    - get()：根据下标获取元素
    - set()：将指定元素设置到指定位置，并返回被替换的元素(用时才接收)
    - 重载remove()：删除指定位置元素，并返回被删除的元素(用时才接收)
    - 重载add()：将指定元素添加到指定位置，理解为插入操作

```
package collection;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
/**
 * List集合的演示
 */
public class ListDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("one");
        list.add("two");
        list.add("three");
        list.add("four");
        list.add("five");
        list.add("one");
        System.out.println("list:"+list); //[one, two, three, four,
five, one]

        //E get(int index):获取指定下标所对应的元素
        String e = list.get(2); //获取第3个元素
        System.out.println(e); //three

        for(int i=0;i<list.size();i++){
            System.out.println(list.get(i));
        }
        for(String s : list){
            System.out.println(s);
        }
        Iterator<String> it = list.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }

        System.out.println("-----------------------");

        //E set(int index,E e):将给定元素设置到指定位置，返回被替换的元素
        //list.set(2,"six"); //将list中下标为2的元素设置为six---常规用法
        String old = list.set(2,"six"); //将list中下标为2的元素设置为six，同
时将原数据返回给old
        System.out.println(old); //three
        System.out.println("list:"+list); //[one, two, six, four, five,
one]

        //E remove(int index):删除指定位置元素，并返回被删除的元素
```

```
        //list.remove(2); //删除下标为2的元素---常规用法
        String s = list.remove(2); ///删除下标为2的元素，并将被删除元素返回给s
        System.out.println(s); //six
        System.out.println("list:"+list); //[one, two, four, five, one]

        //void add(int index,E e):将给定元素e添加到index所指定的位置，相当于插入操作
        list.add(3,"seven"); //在list下标为3的位置插入seven
        System.out.println("list:"+list); //[one, two, four, seven, five, one]
    }
}
```

- subList()：获取集合的子集(含头不含尾)

```
package collection;
import java.util.ArrayList;
import java.util.List;
/**
 * subList()获取子集的演示
 */
public class SubListDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        for(int i=0;i<10;i++){
            list.add(i*10);
        }
        System.out.println("list:"+list); //[0, 10, 20, 30, 40, 50 ,60, 70, 80, 90]

        //List subList(int start,int end):含头不含尾
        List<Integer> subList = list.subList(3,8);  //获取下标3到7的子集
        System.out.println("subList:"+subList); //[30, 40, 50 ,60, 70]

        //将子集每个元素都扩大10倍
        for(int i=0;i<subList.size();i++){
            subList.set(i,subList.get(i)*10);
        }
        System.out.println("subList:"+subList); //[300, 400, 500 ,600, 700]
        //对子集的操作就是对原集合对应的元素操作
        System.out.println("list:"+list); //[0, 10, 20, 300, 400, 500 ,600, 700, 80, 90]

        list.set(3,1000); //将原集合下标为3的元素修改为1000
        System.out.println("list:"+list); //[0, 10, 20, 1000, 400, 500 ,600, 700, 80, 90]
        //原集合数据改变后，子集也跟着变了
        System.out.println("subList:"+subList); //[1000, 400, 500 ,600, 700]

        //对子集增删后，原集合跟着改
        subList.remove(0);
        System.out.println("subList:"+subList);
        System.out.println("list:"+list);
```

```
        //对原集增删后，子集不能再进行操作了----因为载体变了
        list.remove(0);
        System.out.println("list:"+list);
        //System.out.println("subList:"+subList); //运行时发生不支持修改异常
    }
}
```

3. 集合排序：

- Collections为集合的工具类，里面定义了很多静态方法用于操作集合
- Collections.sort(List list)方法：可以对list集合进行自然排序(从小到大)，但是其要求List集合中的元素必须是可比较的，判断是否可以比较的标准是元素是否实现了Comparable接口，若没有实现该接口则直接发生编译错误，但是在实际开发中，我们一般是不会不让我们自己写的类去实现Comparable接口的，因为这对我们的程序有侵入性。
- 侵入性：当我们调用某个API功能时，其要求我们为其修改其它额外的代码，这个现象叫做侵入性。侵入性越强越不利于程序的后期维护，应尽量避免。
- 建议使用重载的Collections.sort(List list, Comparator o);可以通过Comparator来自定义比较规则

```
package collection;
import java.util.Random;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
/**
 * 对Integer元素数据排序
 */
public class SortInteger {
    public static void main(String[] args) {
        Random rand = new Random();
        List<Integer> list = new ArrayList<>();
        for(int i=0;i<10;i++){
            list.add(rand.nextInt(100));
        }
        System.out.println("list原始数据:"+list);

        Collections.sort(list); //自然排序(从小到大)
        System.out.println("list排序后数据:"+list);

        Collections.reverse(list); //反转list集合(数据已经变了--大到小)
        System.out.println("list反转后数据:"+list);
        System.out.println("第1个元素为:"+list.get(0)); //大
    }
}

package collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
```

```java
/**
 * 对String元素排序
 */
public class SortString {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("王克晶");
        list.add("传奇sdfsdf");
        list.add("国斌老师");
        System.out.println("list原始数据: "+list); //[王克晶, 传奇
djflkjflw, 国斌老师]

        //自定义排序规则:
        Collections.sort(list, new Comparator<String>() {
            /*
              compare()方法用于定义o1和o2比较大小的规则，它的返回值表达大小关系
              结论:
                1)前面的-后面的------------升序
                2)后面的-前面的------------降序
             */
            public int compare(String o1, String o2) {
                //return o1.length()-o2.length(); //升序
                return o2.length()-o1.length(); //降序
            }
        });

        //Lambda简化版
        Collections.sort(list, (o1,o2)->o1.length()-o2.length());

        System.out.println("list排序后数据:"+list);

        /*
        List<String> list = new ArrayList<>();
        list.add("jack");
        list.add("rose");
        list.add("tom");
        list.add("jerry");
        list.add("black");
        list.add("Kobe");
        System.out.println("list原始数据:  "+list); //[jack, rose, tom,
jerry, black, Kobe]

        //对英文字符串排序时，会按首字母的ASCII码来排
        //若首字母相同，则比较第2个字母的ASCII码，以此类推
        Collections.sort(list);
        System.out.println("list排序后数据:"+list);
         */

    }
}

public class SortPoint {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
        list.add(new Point(5,8));
```

```
        list.add(new Point(15,60));
        list.add(new Point(57,89));
        list.add(new Point(1,4));
        list.add(new Point(10,8));
        list.add(new Point(22,35));
        System.out.println("list原始数据:"+list);

        //jdk1.8时，List集合自身提供了sort方法进行排序，方法中需要传入比较器
        list.sort(new Comparator<Point>() {
            public int compare(Point o1, Point o2) {
                return o1.getX()-o2.getX();
            }
        });
        System.out.println("list排序后数据:"+list);

        list.sort((o1,o2)->{
            int len1 = o1.getX()*o1.getX()+o1.getY()*o1.getY();
            int len2 = o2.getX()*o2.getX()+o2.getY()*o2.getY();
            return len1-len2; //升序
        });

        /*
        //自定义排序规则:
        Collections.sort(list, new Comparator<Point>() {
            public int compare(Point o1, Point o2) {
                int len1 = o1.getX()*o1.getX()+o1.getY()*o1.getY();
                int len2 = o2.getX()*o2.getX()+o2.getY()*o2.getY();
                //return len1-len2; //升序
                return len2-len1; //降序
                //return o1.getX()-o2.getX(); //按x坐标升序
                //return o2.getY()-o1.getY(); //按y坐标降序
            }
        });
         */
        Collections.sort(list,(o1,o2)->{
            int len1 = o1.getX()*o1.getX()+o1.getY()*o1.getY();
            int len2 = o2.getX()*o2.getX()+o2.getY()*o2.getY();
            return len1-len2; //升序
        });

        System.out.println("list排序后数据:"+list);

    }
}
```

4. Lambda表达式：
    - JDK1.8之后推出了一个新特性：Lambda表达式
    - 规则：
        - 不是任何匿名内部类都可以转换为Lambda表达式
        - Lambda表达式对接口的要求：只能是函数式接口
        - 函数式接口:接口中要求实现类必须重写的方法只有一个
    - 语法：

```
(参数列表)->{
   方法体
}
```

```
public class LambdaDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();

        //匿名内部类写法
        Collections.sort(list, new Comparator<String>() {
            public int compare(String o1, String o2) {
                return o1.length()-o2.length();
            }
        });

        //Lambda表达式写法
        Collections.sort(list, (String o1, String o2) -> {
                return o1.length()-o2.length();
        });

        //Lambda表达式中的参数类型可以不写
        Collections.sort(list, (o1, o2) -> {
            return o1.length()-o2.length();
        });

        //Lambda表达式方法体中只有一句代码，方法体的{}可以不写，如果这句话中有
return，也一并不写
        Collections.sort(list, (o1, o2) -> o1.length()-o2.length());

        //Lambda表达式的方法参数只有1个,那么()可以忽略不写---本案例不适用

    }
}
```

5. Set接口：
    - 不可重复集合，并且大部分实现类都是无序的

```
package collection;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * Set集合的演示
 */
public class SetDemo {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        set.add("one");
        set.add("two");
        set.add("three");
```

```
        set.add("four");
        set.add("five");
        set.add("two"); //无法被正确添加进去，因为Set集是不可重复集合
        System.out.println(set); //并且大部分实现类都是无序的
    }
}
```

- 小面试题：如何去重?

```
package collection;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * Set集合的演示
 */
public class SetDemo {
    public static void main(String[] args) {
        //小面试题:如何去重?
        List<String> list = new ArrayList<>();
        list.add("one");
        list.add("two");
        list.add("three");
        list.add("four");
        list.add("five");
        list.add("two");
        System.out.println("list:"+list); //[one, two, three, four,
five, two]

        Set<String> set = new HashSet<>();
        set.addAll(list); //将list集合元素添加到set集合中，重复的添加不进去
        System.out.println("set:"+set); //[one, two, three, four, five]
    }
}
```

补充:

1. 匿名内部类中不能修改外面局部变量的值，因为在此处变量会默认为final的
2. 方法重写时，要求：派生类方法的访问权限必须大于或等于超类方法中
3. 接口补充:

```
package collection;
/**
 * 接口补充
 */
public class InterfaceMore {
}
```

```
//接口中可以包含:
interface Inter1{
    public abstract void show(); //抽象方法-----用得最多

    public static final int NUM = 5; //常量----用得少

    public default void test(){ //默认方法----用得少，jdk1.8加入的
    }

    public static void say(){ //静态方法------用得少，jdk1.8加入的
    }
}

//接口中的成员访问权限只能是public(默认也是public)
interface Inter{
    public abstract void show();
    void test(); //接口中成员访问权限默认是public的
}
abstract class Aoo{
    abstract void say(); //类中成员访问权限默认是默认权限
}
//方法重写时，要求：派生类方法的访问权限必须大于或等于超类方法中
class Boo extends Aoo implements Inter{
    //重写方法时，访问权限必须大于或等于超类方法的
    public void show(){}
    public void test(){}
    void say(){}
}
```

4. 明日单词:

```
1)file:文件
2)write:写
3)read:读
4)exists:存在
5)dictionary:目录
6)filter:过滤
```