21 不可变集合

- 1 不可变集合

不可变集合：不可以被修改的集合

- 1.1 应用场景
  - 如果某个数据不能被修改，把它防御性地拷贝到不可变集合中是个很好的实践。
  - 或者当集合对象被不可信的库调用时，不可变形式是安全的。

简单理解：不想让别人修改集合中的内容

- 1.2 书写格式

在List、Set、Map接口中，都存在静态的of方法，可以获取一个不可变的集合。

方法名称	说明
static <E> List<E> of(E...elements)	创建一个具有指定元素的List集合对象
static <E> Set<E> of(E...elements)	创建一个具有指定元素的Set集合对象
static <K , V> Map<K，V> of(E...elements)	创建一个具有指定元素的Map集合对象

注意：这个集合不能添加，不能删除，不能修改。

```
//一旦创建完毕之后，是无法进行修改的，在下面的代码中，只能进行查询操作
List<String> list = List.of("张三", "李四", "王五", "赵六");

System.out.println(list.get(0));
System.out.println(list.get(1));
System.out.println(list.get(2));
System.out.println(list.get(3));

Set<String> set = Set.of("张三", "李四", "王五", "赵六");

for (String s : set) {
    System.out.println(s);
}

//一旦创建完毕之后，是无法进行修改的，在下面的代码中，只能进行查询操作
Map<String, String> map = Map.of( k1: "张三", v1: "南京", k2: "李四", v2: "北京", k3: "王五", v3: "上海",
        k4: "赵六", v4: "广州", k5: "孙七", v5: "深圳", k6: "周八", v6: "杭州",
        k7: "吴九", v7: "宁波", k8: "郑十", v8: "苏州", k9: "刘一", v9: "无锡",
        k10: "陈二", v10: "嘉兴");


Set<String> keys = map.keySet();
for (String key : keys) {
    String value = map.get(key);
    System.out.println(key + "=" + value);
}
```

- 细节：
  - 当我们要获取一个不可变的set集合时，里面的参数一定要保证唯一性。
  - 当我们要获取一个不可变的map集合时，键是不能重复的。
  - map里面的of方法，参数是有上限的，最多只能传递20个参数，10个键值对
  - 如果要传递多个键值对对象，数量大于10个，在Map接口中ofEntries

```
//1.创建一个普通的Map集合
HashMap<String, String> hm = new HashMap<>();
hm.put("张三", "南京");
hm.put("李四", "北京");
hm.put("王五", "上海");
hm.put("赵六", "北京");
hm.put("孙七", "深圳");
hm.put("周八", "杭州");
hm.put("吴九", "宁波");
hm.put("郑十", "苏州");
hm.put("刘一", "无锡");
hm.put("陈二", "嘉兴");
hm.put("aaa", "111");

//2.利用上面的数据来获取一个不可变的集合

//获取到所有的键值对对象（Entry对象）
Set<Map.Entry<String, String>> entries = hm.entrySet();
//把entries变成一个数组
Map.Entry[] arr1 = new Map.Entry[0];
//toArray方法在底层会比较集合的长度跟数组的长度两者的大小
//如果集合的长度 > 数组的长度 ：数据在数组中放不下，此时会根据实际数据的个数，重新创建数组
//如果集合的长度 <= 数组的长度：数据在数组中放的下，此时不会创建新的数组，而是直接用
Map.Entry[] arr2 = entries.toArray(arr1);

//不可变的map集合
Map map = Map.ofEntries(arr2);
```

  - 简化

```
Map<Object, Object> objectObjectMap = Map.ofEntries(hm.entrySet().toArray(new Map.Entry[0]));
```

```
Map<String, String> map = Map.copyOf(hm);
```

- 1.3 总结

1. 不可变集合的特点？
   - 定义完成后不可以修改，或者添加、删除
2. 如何创建不可变集合？
   - List、Set、Map接口中，都存在of方法可以创建不可变集合
3. 三种方式的细节
   - List：直接用
   - Set：元素不能重复
   - Map：元素不能重复、键值对数量最多是10个。

     超过10个用ofEntries方法

- 2 Stream流

- 作用

  结合了Lambda表达式，简化集合、数组的操作

- 使用步骤

  ① 先得到一条Stream流（流水线），并把数据放上去

  ② 利用Stream流中的API进行各种操作

  过滤　转换　中间方法　方法调用完毕之后，还可以调用其他方法

  统计　打印　终结方法　最后一步，调用完毕之后，不能调用其他方法

- 步骤1

  ① 先得到一条Stream流（流水线），并把数据放上去

获取方式	方法名	说明
单列集合	default Stream<E> stream()	Collection中的默认方法
双列集合	无	无法直接使用stream流
数组	public static <T> Stream<T> stream(T[] array)	Arrays工具类中的静态方法
一堆零散数据	public static<T> Stream<T> of(T... values)	Stream接口中的静态方法

- 代码实现

```
//1.单列集合获取Stream流
ArrayList<String> list = new ArrayList<>();
Collections.addAll(list, ...elements: "a","b","c","d","e");

list.stream().forEach(s -> System.out.println(s));

//1.创建双列集合
HashMap<String,Integer> hm = new HashMap<>();
//2.添加数据
hm.put("aaa",111);
hm.put("bbb",222);
hm.put("ccc",333);
hm.put("ddd",444);

//3.第一种获取stream流
//hm.keySet().stream().forEach(s -> System.out.println(s));

//4.第二种获取stream流
hm.entrySet().stream().forEach(s-> System.out.println(s));
```

```
//1.创建数组
int[] arr1 = {1,2,3,4,5,6,7,8,9,10};

String[] arr2 = {"a","b","c"};


//2.获取stream流
Arrays.stream(arr1).forEach(s-> System.out.println(s));

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

Arrays.stream(arr2).forEach(s-> System.out.println(s));

 //一堆零散数据    public static<T> Stream<T> of(T... values)          Stream接口中的静态方法

 Stream.of(1,2,3,4,5).forEach(s-> System.out.println(s));
 Stream.of("a","b","c","d","e").forEach(s-> System.out.println(s));
```

- 注意: stream接口中静态方法of的细节
- 方法的形参是一个可变参数，可以传递一堆零散的数据，也可以传递数组
- 但是数组必须是引用数据类型的，如果传递基本数据类型，是会把整个数组当做一个元素，放到Stream当中。

- **Stream流的中间方法**

名称	说明
Stream<T> **filter**(Predicate<? super T> predicate)	过滤
Stream<T> **limit**(long maxSize)	获取前几个元素
Stream<T> **skip**(long n)	跳过前几个元素
Stream<T> **distinct**()	元素去重，依赖(hashCode和equals方法)
static <T> Stream<T> **concat**(Stream a, Stream b)	合并a和b两个流为一个流
Stream<R> **map**(Function<T , R> mapper)	转换流中的数据类型

注意1： 中间方法，返回新的Stream流，原来的Stream流只能使用一次，建议使用链式编程

注意2： 修改Stream流中的数据，不会影响原来集合或者数组中的数据

- **代码实现**

```
list.stream().filter(s -> s.startsWith("张")).forEach(s -> System.out.println(s));

//"张无忌", "周芷若", "赵敏", "张强", "张三丰", "张翠山", "张良", "王二麻子", "谢广坤"
list.stream().limit(3)
        .forEach(s -> System.out.println(s));

/list.stream().skip(4) .forEach(s -> System.out.println(s));

list1.stream().distinct().forEach(s -> System.out.println(s));

Stream.concat(list1.stream(),list2.stream()).forEach(s -> System.out.println(s));
```

```java
//需求：只获取里面的年龄并进行打印
//String->int

//第一个类型：流中原本的数据类型
//第二个类型：要转成之后的类型

//apply的形参s：依次表示流里面的每一个数据
//返回值：表示转换之后的数据

//当map方法执行完毕之后，流上的数据就变成了整数
//所以在下面forEach当中，s依次表示流里面的每一个数据，这个数据现在就是整数了
list.stream().map(new Function<String, Integer>() {
    @Override
    public Integer apply(String s) {
        String[] arr = s.split( regex: "-");
        String ageString = arr[1];
        int age = Integer.parseInt(ageString);
        return age;
    }
}).forEach(s-> System.out.println(s));

//"张无忌-15"
list.stream().map(s-> Integer.parseInt(s.split( regex: "-")[1])).forEach(s-> System.out.println(s));
```

- Stream流的终结方法

名称	说明
void **forEach**(Consumer action)	遍历
long **count**()	统计
toArray()	收集流中的数据，放到数组中
collect(Collector collector)	收集流中的数据，放到集合中

- 代码实现

```java
//list.stream().forEach(s -> System.out.println(s));

// long count()                                    统计
//long count = list.stream().count();
//System.out.println(count);
```

- toArray()方法空参，返回的是object类型

```java
// toArray()                                        收集流中的数据，放到数组中
Object[] arr1 = list.stream().toArray();
System.out.println(Arrays.toString(arr1));
```

- toArray()方法有形参，返回的是指定类型

```
//IntFunction的泛型：具体类型的数组
//apply的形参:流中数据的个数，要跟数组的长度保持一致
//apply的返回值：具体类型的数组
//方法体：就是创建数组


//toArray方法的参数的作用：负责创建一个指定类型的数组
//toArray方法的底层，会依次得到流里面的每一个数据，并把数据放到数组当中
//toArray方法的返回值：是一个装着流里面所有数据的数组
String[] arr = list.stream().toArray(new IntFunction<String[]>() {
    @Override
    public String[] apply(int value) {
        return new String[value];
    }
});

System.out.println(Arrays.toString(arr));

String[] arr2 = list.stream().toArray(value -> new String[value]);

System.out.println(Arrays.toString(arr2));
```

- **收集到List、Set集合中**

```
//收集List集合当中
//需求：
//我要把所有的男性收集起来
List<String> newList1 = list.stream()
        .filter(s -> "男".equals(s.split( regex: "-")[1]))
        .collect(Collectors.toList());


//收集Set集合当中
//需求：
//我要把所有的男性收集起来
Set<String> newList2 = list.stream().filter(s -> "男".equals(s.split( regex: "-")[1]))
        .collect(Collectors.toSet());
```

- 区别：数据收集到List集合中，不会去重。数据收集到Set集合中，会去重。
- **收集到Map集合中**

```
/*
*   toMap : 参数一表示键的生成规则
*           参数二表示值的生成规则
*
* 参数一:
*       Function泛型一：表示流中每一个数据的类型
*                泛型二：表示Map集合中键的数据类型
*
*        方法apply形参：依次表示流里面的每一个数据
*                方法体：生成键的代码
*                返回值：已经生成的键
*
*
* 参数二:
*        Function泛型一：表示流中每一个数据的类型
*                泛型二：表示Map集合中值的数据类型
*
*       方法apply形参：依次表示流里面的每一个数据
 *                方法体：生成值的代码
 *                返回值：已经生成的值
*
* */

Map<String, Integer> map2 = list.stream()
        .filter(s -> "男".equals(s.split( regex: "-")[1]))
        .collect(Collectors.toMap(
                s -> s.split( regex: "-")[0],
                s -> Integer.parseInt(s.split( regex: "-")[2])));
```

- 注意：如果要收集到Map集合中，键不能重复，否则会报错

- **总结**

1. Stream流的作用

   结合了Lambda表达式，简化集合、数组的操作

2. Stream的使用步骤
   - 获取Stream流对象
   - 使用中间方法处理数据
   - 使用终结方法处理数据
3. 如何获取Stream流对象
   - 单列集合：Collection中的默认方法stream
   - 双列集合：不能直接获取
   - 数组：Arrays工具类型中的静态方法stream
   - 一堆零散的数据：Stream接口中的静态方法of
4. 常见方法

   中间方法： filter，limit，skip，distinct，concat，map

   终结方法： forEach，count，collect