1.Map集合

1.1Map集合概述和特点【理解】

- Map集合概述

```
interface Map<K,V>  K：键的类型；V：值的类型
```

- Map集合的特点
  - 双列集合,一个键对应一个值
  - 键不可以重复,值可以重复
- Map集合的基本使用

```
public class MapDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String,String> map = new HashMap<String,String>();

        //V put(K key, V value) 将指定的值与该映射中的指定键相关联
        map.put("itheima001","林青霞");
        map.put("itheima002","张曼玉");
        map.put("itheima003","王祖贤");
        map.put("itheima003","柳岩");

        //输出集合对象
        System.out.println(map);
    }
}
```

1.2Map集合的基本功能【应用】

- 方法介绍

方法名	说明
V　put(K key,V　value)	添加元素
V　remove(Object key)	根据键删除键值对元素
void　clear()	移除所有的键值对元素
boolean containsKey(Object key)	判断集合是否包含指定的键
boolean containsValue(Object value)	判断集合是否包含指定的值
boolean isEmpty()	判断集合是否为空
int size()	集合的长度，也就是集合中键值对的个数

- 示例代码

```
public class MapDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String,String> map = new HashMap<String,String>();

        //V put(K key,V value)：添加元素
        map.put("张无忌","赵敏");
        map.put("郭靖","黄蓉");
        map.put("杨过","小龙女");

        //V remove(Object key)：根据键删除键值对元素
//        System.out.println(map.remove("郭靖"));
//        System.out.println(map.remove("郭襄"));

        //void clear()：移除所有的键值对元素
//        map.clear();

        //boolean containsKey(Object key)：判断集合是否包含指定的键
//        System.out.println(map.containsKey("郭靖"));
//        System.out.println(map.containsKey("郭襄"));

        //boolean isEmpty()：判断集合是否为空
//        System.out.println(map.isEmpty());

        //int size()：集合的长度，也就是集合中键值对的个数
        System.out.println(map.size());

        //输出集合对象
        System.out.println(map);
    }
}
```

1.3Map集合的获取功能【应用】

- 方法介绍

方法名	说明
V get(Object key)	根据键获取值
Set keySet()	获取所有键的集合
Collection values()	获取所有值的集合
Set> entrySet()	获取所有键值对对象的集合

- 示例代码

```
public class MapDemo03 {

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

```
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("张无忌", "赵敏");
        map.put("郭靖", "黄蓉");
        map.put("杨过", "小龙女");

        //V get(Object key):根据键获取值
//        System.out.println(map.get("张无忌"));
//        System.out.println(map.get("张三丰"));

        //Set<K> keySet():获取所有键的集合
//        Set<String> keySet = map.keySet();
//        for(String key : keySet) {
//            System.out.println(key);
//        }

        //Collection<V> values():获取所有值的集合
        Collection<String> values = map.values();
        for(String value : values) {
            System.out.println(value);
        }
    }
}
```

1.4Map集合的遍历(方式1)【应用】

- 遍历思路
  - 我们刚才存储的元素都是成对出现的，所以我们把Map看成是一个夫妻对的集合
    - 把所有的丈夫给集中起来
    - 遍历丈夫的集合，获取到每一个丈夫
    - 根据丈夫去找对应的妻子
- 步骤分析
  - 获取所有键的集合。用keySet()方法实现
  - 遍历键的集合，获取到每一个键。用增强for实现
  - 根据键去找值。用get(Object key)方法实现
- 代码实现

```
public class MapDemo01 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("张无忌", "赵敏");
        map.put("郭靖", "黄蓉");
        map.put("杨过", "小龙女");

        //获取所有键的集合。用keySet()方法实现
```

```
        Set<String> keySet = map.keySet();
        //遍历键的集合，获取到每一个键。用增强for实现
        for (String key : keySet) {
            //根据键去找值。用get(Object key)方法实现
            String value = map.get(key);
            System.out.println(key + "," + value);
        }
    }
}
```

1.5Map集合的遍历(方式2)【应用】

- 遍历思路
  - 我们刚才存储的元素都是成对出现的，所以我们把Map看成是一个夫妻对的集合
    - 获取所有结婚证的集合
    - 遍历结婚证的集合，得到每一个结婚证
    - 根据结婚证获取丈夫和妻子
- 步骤分析
  - 获取所有键值对对象的集合
    - Set> entrySet()：获取所有键值对对象的集合
  - 遍历键值对对象的集合，得到每一个键值对对象
    - 用增强for实现，得到每一个Map.Entry
  - 根据键值对对象获取键和值
    - 用getKey()得到键
    - 用getValue()得到值
- 代码实现

```
public class MapDemo02 {
    public static void main(String[] args) {
        //创建集合对象
        Map<String, String> map = new HashMap<String, String>();

        //添加元素
        map.put("张无忌", "赵敏");
        map.put("郭靖", "黄蓉");
        map.put("杨过", "小龙女");

        //获取所有键值对对象的集合
        Set<Map.Entry<String, String>> entrySet = map.entrySet();
        //遍历键值对对象的集合，得到每一个键值对对象
        for (Map.Entry<String, String> me : entrySet) {
            //根据键值对对象获取键和值
            String key = me.getKey();
            String value = me.getValue();
            System.out.println(key + "," + value);
        }
    }
}
```

2.HashMap集合

2.1HashMap集合概述和特点【理解】

- HashMap底层是哈希表结构的
- 依赖hashCode方法和equals方法保证键的唯一
- 如果键要存储的是自定义对象，需要重写hashCode和equals方法

2.2HashMap集合应用案例【应用】

- 案例需求
  - 创建一个HashMap集合，键是学生对象(Student)，值是居住地 (String)。存储多个元素，并遍历。
  - 要求保证键的唯一性：如果学生对象的成员变量值相同，我们就认为是同一个对象
- 代码实现

  学生类

```
public class Student {
    private String name;
    private int age;

    public Student() {
    }

    public Student(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 boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
        Student student = (Student) o;

        if (age != student.age) return false;
        return name != null ? name.equals(student.name) : student.name == null;
    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }
}
```

测试类

```
public class HashMapDemo {
    public static void main(String[] args) {
        //创建HashMap集合对象
        HashMap<Student, String> hm = new HashMap<Student, String>();

        //创建学生对象
        Student s1 = new Student("林青霞", 30);
        Student s2 = new Student("张曼玉", 35);
        Student s3 = new Student("王祖贤", 33);
        Student s4 = new Student("王祖贤", 33);

        //把学生添加到集合
        hm.put(s1, "西安");
        hm.put(s2, "武汉");
        hm.put(s3, "郑州");
        hm.put(s4, "北京");

        //遍历集合
        Set<Student> keySet = hm.keySet();
        for (Student key : keySet) {
            String value = hm.get(key);
            System.out.println(key.getName() + "," + key.getAge() + "," + value);
        }
    }
}
```

3.TreeMap集合

3.1TreeMap集合概述和特点【理解】

- TreeMap底层是红黑树结构
- 依赖自然排序或者比较器排序,对键进行排序
- 如果键存储的是自定义对象,需要实现Comparable接口或者在创建TreeMap对象时候给出比较器排序规则

3.2TreeMap集合应用案例一【应用】

- 案例需求
  - 创建一个TreeMap集合,键是学生对象(Student),值是籍贯(String),学生属性姓名和年龄,按照年龄进行排序并遍历
  - 要求按照学生的年龄进行排序,如果年龄相同则按照姓名进行排序
- 代码实现

  学生类

```
public class Student implements Comparable<Student>{
    private String name;
    private int age;

    public Student() {
    }

    public Student(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 String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Student o) {
        //按照年龄进行排序
        int result = o.getAge() - this.getAge();
        //次要条件，按照姓名排序。
        result = result == 0 ? o.getName().compareTo(this.getName()) : result;
        return result;
                                                compareTo按字典顺序比较两个字符串。
    }
```

```
}
```

测试类

```
public class Test1 {
    public static void main(String[] args) {
        // 创建TreeMap集合对象
        TreeMap<Student,String> tm = new TreeMap<>();

         // 创建学生对象
        Student s1 = new Student("xiaohei",23);
        Student s2 = new Student("dapang",22);
        Student s3 = new Student("xiaomei",22);

         // 将学生对象添加到TreeMap集合中
        tm.put(s1,"江苏");
        tm.put(s2,"北京");
        tm.put(s3,"天津");

         // 遍历TreeMap集合,打印每个学生的信息
        tm.forEach(
                (Student key, String value)->{
                    System.out.println(key + "---" + value);
                }
        );
    }
}
```

3.3TreeMap集合应用案例二【应用】

- 案例需求
  - 给定一个字符串,要求统计字符串中每个字符出现的次数。
  - 举例: 给定字符串是“aababcabcdabcde”,在控制台输出: “a(5)b(4)c(3)d(2)e(1)”
- 代码实现

```
public class Test2 {
    public static void main(String[] args) {
        // 给定字符串
        String s = "aababcabcdabcde";
         // 创建TreeMap集合对象,键是Character,值是Integer
        TreeMap<Character,Integer> tm = new TreeMap<>();

        //遍历字符串，得到每一个字符
        for (int i = 0; i < s.length(); i++) {
            //c依次表示字符串中的每一个字符
            char c = s.charAt(i);
            // 判断当前遍历到的字符是否在集合中出现过
            if(!tm.containsKey(c)){
                //表示当前字符是第一次出现。
                tm.put(c,1);

            }else{
```

```
                //存在，表示当前字符已经出现过了
                //先获取这个字符已经出现的次数
                Integer count = tm.get(c);
                //自增，表示这个字符又出现了依次
                count++;
                //将自增后的结果再次添加到集合中。
                tm.put(c,count);
            }
        }
        //  a (5) b (4) c (3) d (2) e (1)
        //System.out.println(tm);
        tm.forEach(
                (Character key,Integer value)->{
                    System.out.print(key + " (" + value + ") ");
                }
        );
    }
}
```

4.可变参数

4.1可变参数【应用】

- 可变参数介绍
  - 可变参数又称参数个数可变，用作方法的形参出现，那么方法参数个数就是可变的了
  - 方法的参数类型已经确定,个数不确定,我们可以使用可变参数
- 可变参数定义格式

```
修饰符 返回值类型 方法名(数据类型… 变量名) {  }
```

- 可变参数的注意事项
  - 这里的变量其实是一个数组
  - 如果一个方法有多个参数，包含可变参数，可变参数要放在最后
- 可变参数的基本使用

```
public class ArgsDemo01 {
    public static void main(String[] args) {
        System.out.println(sum(10, 20));
        System.out.println(sum(10, 20, 30));
        System.out.println(sum(10, 20, 30, 40));

        System.out.println(sum(10,20,30,40,50));
        System.out.println(sum(10,20,30,40,50,60));
        System.out.println(sum(10,20,30,40,50,60,70));
        System.out.println(sum(10,20,30,40,50,60,70,80,90,100));
    }

//    public static int sum(int b,int... a) {

//        return 0;
```

```
//    }

    public static int sum(int... a) {
        int sum = 0;
        for(int i : a) {
            sum += i;
        }
        return sum;
    }
}
```

4.2创建不可变集合【理解】

- 方法介绍
  - 在List、Set、Map接口中,都存在of方法,可以创建一个不可变的集合
    - 这个集合不能添加,不能删除,不能修改
    - 但是可以结合集合的带参构造,实现集合的批量添加
  - 在Map接口中,还有一个ofEntries方法可以提高代码的阅读性
    - 首先会把键值对封装成一个Entry对象,再把这个Entry对象添加到集合当中
- 示例代码

```
public class MyVariableParameter4 {
    public static void main(String[] args) {
        // 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

        //method1();
        //method2();
        //method3();
        //method4();

    }

    private static void method4() {
        Map<String, String> map = Map.ofEntries(
                Map.entry("zhangsan", "江苏"),
                Map.entry("lisi", "北京"));
        System.out.println(map);
    }

    private static void method3() {
        Map<String, String> map = Map.of("zhangsan", "江苏", "lisi", "北京", "wangwu", "天
津");
        System.out.println(map);
    }

    private static void method2() {
        //传递的参数当中，不能存在重复的元素。

        Set<String> set = Set.of("a", "b", "c", "d","a");
```

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

    private static void method1() {
        List<String> list = List.of("a", "b", "c", "d");
        System.out.println(list);
        //list.add("Q");
        //list.remove("a");
        //list.set(0,"A");
        //System.out.println(list);

//        ArrayList<String> list2 = new ArrayList<>();
//        list2.add("aaa");
//        list2.add("aaa");
//        list2.add("aaa");
//        list2.add("aaa");

        //集合的批量添加。
        //首先是通过调用List.of方法来创建一个不可变的集合，of方法的形参就是一个可变参数。
        //再创建一个ArrayList集合，并把这个不可变的集合中所有的数据，都添加到ArrayList中。
        ArrayList<String> list3 = new ArrayList<>(List.of("a", "b", "c", "d"));
        System.out.println(list3);
    }
}
```

5.Stream流

5.1体验Stream流【理解】

- 案例需求

  按照下面的要求完成集合的创建和遍历

  - 创建一个集合，存储多个字符串元素
  - 把集合中所有以"张"开头的元素存储到一个新的集合
  - 把"张"开头的集合中的长度为3的元素存储到一个新的集合
  - 遍历上一步得到的集合

- 原始方式示例代码

```
public class StreamDemo {
    public static void main(String[] args) {
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");
        list.add("张敏");
        list.add("张无忌");


        //把集合中所有以"张"开头的元素存储到一个新的集合
```

```
        ArrayList<String> zhangList = new ArrayList<String>();

        for(String s : list) {
            if(s.startsWith("张")) {
                zhangList.add(s);
            }
        }

//        System.out.println(zhangList);

        //把"张"开头的集合中的长度为3的元素存储到一个新的集合
        ArrayList<String> threeList = new ArrayList<String>();

        for(String s : zhangList) {
            if(s.length() == 3) {
                threeList.add(s);
            }
        }

//        System.out.println(threeList);

        //遍历上一步得到的集合
        for(String s : threeList) {
            System.out.println(s);
        }
        System.out.println("--------");

        //Stream流来改进
//        list.stream().filter(s -> s.startsWith("张")).filter(s -> s.length() ==
3).forEach(s -> System.out.println(s));
        list.stream().filter(s -> s.startsWith("张")).filter(s -> s.length() ==
3).forEach(System.out::println);
    }
}
```

- 使用Stream流示例代码

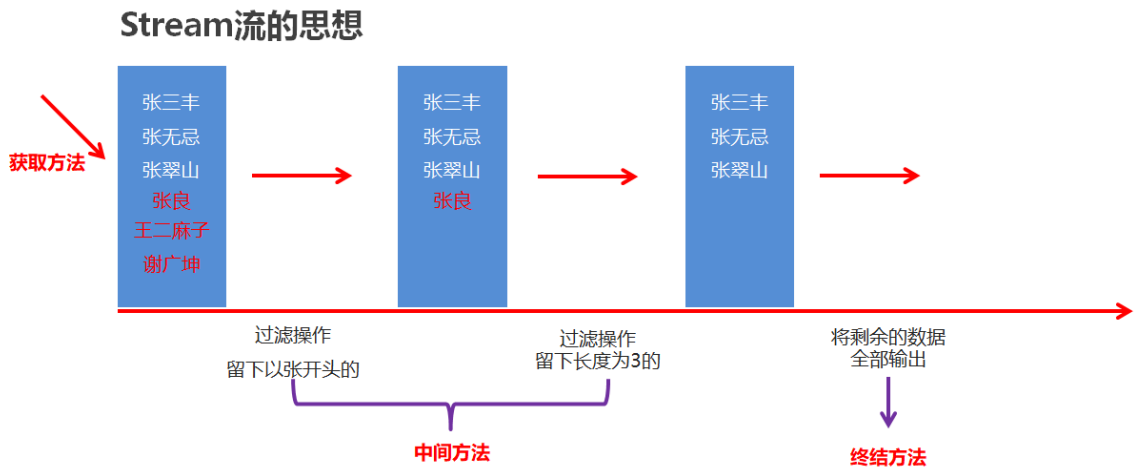
```
public class StreamDemo {
    public static void main(String[] args) {
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");
        list.add("张敏");
        list.add("张无忌");

        //Stream流来改进
        list.stream().filter(s -> s.startsWith("张")).filter(s -> s.length() ==
3).forEach(System.out::println);
    }
```

```
}
```

- Stream流的好处
  - 直接阅读代码的字面意思即可完美展示无关逻辑方式的语义：获取流、过滤姓张、过滤长度为3、逐一打印
  - Stream流把真正的函数式编程风格引入到Java中
  - 代码简洁

5.2Stream流的常见生成方式【应用】

- Stream流的思想



- Stream流的三类方法
  - 获取Stream流
    - 创建一条流水线,并把数据放到流水线上准备进行操作
  - 中间方法
    - 流水线上的操作
    - 一次操作完毕之后,还可以继续进行其他操作
  - 终结方法
    - 一个Stream流只能有一个终结方法
    - 是流水线上的最后一个操作
- 生成Stream流的方式
  - Collection体系集合

    使用默认方法stream()生成流， default Stream stream()
  - Map体系集合

    把Map转成Set集合，间接的生成流
  - 数组

    通过Arrays中的静态方法stream生成流
  - 同种数据类型的多个数据

    通过Stream接口的静态方法of(T... values)生成流

- 代码演示

```
public class StreamDemo {
    public static void main(String[] args) {
        //Collection体系的集合可以使用默认方法stream()生成流
        List<String> list = new ArrayList<String>();
        Stream<String> listStream = list.stream();

        Set<String> set = new HashSet<String>();
        Stream<String> setStream = set.stream();

        //Map体系的集合间接的生成流
        Map<String,Integer> map = new HashMap<String, Integer>();
        Stream<String> keyStream = map.keySet().stream();
        Stream<Integer> valueStream = map.values().stream();
        Stream<Map.Entry<String, Integer>> entryStream = map.entrySet().stream();

        //数组可以通过Arrays中的静态方法stream生成流
        String[] strArray = {"hello","world","java"};
        Stream<String> strArrayStream = Arrays.stream(strArray);

        //同种数据类型的多个数据可以通过Stream接口的静态方法of(T... values)生成流
        Stream<String> strArrayStream2 = Stream.of("hello", "world", "java");
        Stream<Integer> intStream = Stream.of(10, 20, 30);
    }
}
```

5.3Stream流中间操作方法【应用】

- 概念

  中间操作的意思是,执行完此方法之后,Stream流依然可以继续执行其他操作

- 常见方法

方法名	说明
Stream filter(Predicate predicate)	用于对流中的数据进行过滤
Stream limit(long maxSize)	返回此流中的元素组成的流，截取前指定参数个数的数据
Stream skip(long n)	跳过指定参数个数的数据，返回由该流的剩余元素组成的流
static Stream concat(Stream a, Stream b)	合并a和b两个流为一个流
Stream distinct()	返回由该流的不同元素（根据Object.equals(Object) ）组成的流

- filter代码演示

```
public class StreamDemo01 {

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

```
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");
        list.add("张敏");
        list.add("张无忌");

        //需求1：把list集合中以张开头的元素在控制台输出
        list.stream().filter(s -> s.startsWith("张")).forEach(System.out::println);
        System.out.println("--------");

        //需求2：把list集合中长度为3的元素在控制台输出
        list.stream().filter(s -> s.length() == 3).forEach(System.out::println);
        System.out.println("--------");

        //需求3：把list集合中以张开头的，长度为3的元素在控制台输出
        list.stream().filter(s -> s.startsWith("张")).filter(s -> s.length() ==
3).forEach(System.out::println);
    }
}
```

- limit&skip代码演示

```
public class StreamDemo02 {
    public static void main(String[] args) {
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");
        list.add("张敏");
        list.add("张无忌");

        //需求1：取前3个数据在控制台输出
        list.stream().limit(3).forEach(System.out::println);
        System.out.println("--------");

        //需求2：跳过3个元素，把剩下的元素在控制台输出
        list.stream().skip(3).forEach(System.out::println);
        System.out.println("--------");

        //需求3：跳过2个元素，把剩下的元素中前2个在控制台输出
        list.stream().skip(2).limit(2).forEach(System.out::println);
    }
}
```

- concat&distinct代码演示

```
public class StreamDemo03 {
    public static void main(String[] args) {
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");
        list.add("张敏");
        list.add("张无忌");

        //需求1：取前4个数据组成一个流
        Stream<String> s1 = list.stream().limit(4);

        //需求2：跳过2个数据组成一个流
        Stream<String> s2 = list.stream().skip(2);

        //需求3：合并需求1和需求2得到的流，并把结果在控制台输出
//        Stream.concat(s1,s2).forEach(System.out::println);

        //需求4：合并需求1和需求2得到的流，并把结果在控制台输出，要求字符串元素不能重复
        Stream.concat(s1,s2).distinct().forEach(System.out::println);
    }
}
```

5.4Stream流终结操作方法【应用】

- 概念

  终结操作的意思是,执行完此方法之后,Stream流将不能再执行其他操作

- 常见方法

方法名	说明
void forEach(Consumer action)	对此流的每个元素执行操作
long count()	返回此流中的元素数

- 代码演示

```
public class StreamDemo {
    public static void main(String[] args) {
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");
        list.add("张敏");
        list.add("张无忌");
```

```
        //需求1：把集合中的元素在控制台输出
//        list.stream().forEach(System.out::println);

        //需求2：统计集合中有几个以张开头的元素，并把统计结果在控制台输出
        long count = list.stream().filter(s -> s.startsWith("张")).count();
        System.out.println(count);
    }
}
```

5.5Stream流的收集操作【应用】

- 概念

  对数据使用Stream流的方式操作完毕后,可以把流中的数据收集到集合中

- 常用方法

方法名	说明
R collect(Collector collector)	把结果收集到集合中

- 工具类Collectors提供了具体的收集方式

方法名	说明
public static Collector toList()	把元素收集到List集合中
public static Collector toSet()	把元素收集到Set集合中
public static Collector toMap(Function keyMapper,Function valueMapper)	把元素收集到Map集合中

- 代码演示

```
public class CollectDemo {
    public static void main(String[] args) {
        //创建List集合对象
        List<String> list = new ArrayList<String>();
        list.add("林青霞");
        list.add("张曼玉");
        list.add("王祖贤");
        list.add("柳岩");

        /*
        //需求1：得到名字为3个字的流
        Stream<String> listStream = list.stream().filter(s -> s.length() == 3);

        //需求2：把使用Stream流操作完毕的数据收集到List集合中并遍历
        List<String> names = listStream.collect(Collectors.toList());
        for(String name : names) {

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

```
        }
        */

        //创建Set集合对象
        Set<Integer> set = new HashSet<Integer>();
        set.add(10);
        set.add(20);
        set.add(30);
        set.add(33);
        set.add(35);

        /*
        //需求3：得到年龄大于25的流
        Stream<Integer> setStream = set.stream().filter(age -> age > 25);

        //需求4：把使用Stream流操作完毕的数据收集到Set集合中并遍历
        Set<Integer> ages = setStream.collect(Collectors.toSet());
        for(Integer age : ages) {
            System.out.println(age);
        }
        */
        //定义一个字符串数组，每一个字符串数据由姓名数据和年龄数据组合而成
        String[] strArray = {"林青霞,30", "张曼玉,35", "王祖贤,33", "柳岩,25"};

        //需求5：得到字符串中年龄数据大于28的流
        Stream<String> arrayStream = Stream.of(strArray).filter(s ->
Integer.parseInt(s.split(",")[1]) > 28);

        //需求6：把使用Stream流操作完毕的数据收集到Map集合中并遍历，字符串中的姓名作键，年龄作值
        Map<String, Integer> map = arrayStream.collect(Collectors.toMap(s -> s.split(",")
[0], s -> Integer.parseInt(s.split(",")[1])));

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

5.6Stream流综合练习【应用】

- 案例需求

  现在有两个ArrayList集合，分别存储6名男演员名称和6名女演员名称，要求完成如下的操作

  - 男演员只要名字为3个字的前三人
  - 女演员只要姓林的，并且不要第一个
  - 把过滤后的男演员姓名和女演员姓名合并到一起
  - 把上一步操作后的元素作为构造方法的参数创建演员对象,遍历数据

  演员类Actor已经提供，里面有一个成员变量，一个带参构造方法，以及成员变量对应的get/set方法

- 代码实现

演员类

```
public class Actor {
    private String name;

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

    public String getName() {
        return name;
    }

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

测试类

```
public class StreamTest {
    public static void main(String[] args) {
        //创建集合
        ArrayList<String> manList = new ArrayList<String>();
        manList.add("周润发");
        manList.add("成龙");
        manList.add("刘德华");
        manList.add("吴京");
        manList.add("周星驰");
        manList.add("李连杰");

        ArrayList<String> womanList = new ArrayList<String>();
        womanList.add("林心如");
        womanList.add("张曼玉");
        womanList.add("林青霞");
        womanList.add("柳岩");
        womanList.add("林志玲");
        womanList.add("王祖贤");

        /*
        //男演员只要名字为3个字的前三人
        Stream<String> manStream = manList.stream().filter(s -> s.length() == 3).limit(3);

        //女演员只要姓林的，并且不要第一个
        Stream<String> womanStream = womanList.stream().filter(s ->
s.startsWith("林")).skip(1);

        //把过滤后的男演员姓名和女演员姓名合并到一起
        Stream<String> stream = Stream.concat(manStream, womanStream);

        //把上一步操作后的元素作为构造方法的参数创建演员对象,遍历数据

//        stream.map(Actor::new).forEach(System.out::println);
```

```
        stream.map(Actor::new).forEach(p -> System.out.println(p.getName()));
        */

        Stream.concat(manList.stream().filter(s -> s.length() == 3).limit(3),
                womanList.stream().filter(s ->
s.startsWith("林")).skip(1)).map(Actor::new).
                forEach(p -> System.out.println(p.getName()));
    }
}
```