1.函数式接口

1.1函数式接口概述【理解】

- 概念

  有且仅有一个抽象方法的接口

- 如何检测一个接口是不是函数式接口

  @FunctionalInterface

  放在接口定义的上方：如果接口是函数式接口，编译通过；如果不是，编译失败

- 注意事项

  我们自己定义函数式接口的时候，@FunctionalInterface是可选的，就算我不写这个注解，只要保证满足函数式接口定义的条件，也照样是函数式接口。但是，建议加上该注解

1.2函数式接口作为方法的参数【应用】

- 需求描述

  定义一个类(RunnableDemo)，在类中提供两个方法

  一个方法是：startThread(Runnable r) 方法参数Runnable是一个函数式接口

  一个方法是主方法，在主方法中调用startThread方法

- 代码演示

```
public class RunnableDemo {
    public static void main(String[] args) {
        //在主方法中调用startThread方法

        //匿名内部类的方式
        startThread(new Runnable() {
            @Override
            public void run() {
                System.out.println(Thread.currentThread().getName() + "线程启动了");
            }
        });

        //Lambda方式
        startThread(() -> System.out.println(Thread.currentThread().getName() + "线
程启动了"));

    }

    private static void startThread(Runnable r) {
        new Thread(r).start();
    }
}
```

1.3函数式接口作为方法的返回值【应用】

- 需求描述

  定义一个类(ComparatorDemo)，在类中提供两个方法

  一个方法是：Comparator getComparator() 方法返回值Comparator是一个函数式接口

  一个方法是主方法，在主方法中调用getComparator方法

- 代码演示

```
public class ComparatorDemo {
    public static void main(String[] args) {
        //定义集合，存储字符串元素
        ArrayList<String> array = new ArrayList<String>();

        array.add("cccc");
        array.add("aa");
        array.add("b");
        array.add("ddd");

        System.out.println("排序前: " + array);

        Collections.sort(array, getComparator());

        System.out.println("排序后: " + array);

    }

    private static Comparator<String> getComparator() {
        //匿名内部类的方式实现
//        return new Comparator<String>() {
//            @Override
//            public int compare(String s1, String s2) {
//                return s1.length()-s2.length();
//            }
//        };

        //Lambda方式实现
        return (s1, s2) -> s1.length() - s2.length();
    }
}
```

1.4常用函数式接口之Supplier【应用】

- Supplier接口

  Supplier接口也被称为生产型接口，如果我们指定了接口的泛型是什么类型，那么接口中的get方法就会生产什么类型的数据供我们使用。

- 常用方法

  只有一个无参的方法

方法名	说明
T get()	按照某种实现逻辑(由Lambda表达式实现)返回一个数据

- 代码演示

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

        String s = getString(() -> "林青霞");
        System.out.println(s);

        Integer i = getInteger(() -> 30);
        System.out.println(i);
    }

    //定义一个方法，返回一个整数数据
    private static Integer getInteger(Supplier<Integer> sup) {
        return sup.get();
    }

    //定义一个方法，返回一个字符串数据
    private static String getString(Supplier<String> sup) {
        return sup.get();
    }

}
```

1.5Supplier接口练习之获取最大值【应用】

- 案例需求

  定义一个类(SupplierTest)，在类中提供两个方法

  一个方法是：int getMax(Supplier sup) 用于返回一个int数组中的最大值

  一个方法是主方法，在主方法中调用getMax方法

- 示例代码

```
public class SupplierTest {
    public static void main(String[] args) {
        //定义一个int数组
        int[] arr = {19, 50, 28, 37, 46};

        int maxValue = getMax(()-> {
           int max = arr[0];

           for(int i=1; i<arr.length; i++) {
               if(arr[i] > max) {
                   max = arr[i];
               }
           }
```

```
            return max;
        });

        System.out.println(maxValue);

    }

    //返回一个int数组中的最大值
    private static int getMax(Supplier<Integer> sup) {
        return sup.get();
    }
}
```

1.6常用函数式接口之Consumer【应用】

- Consumer接口

  Consumer接口也被称为消费型接口，它消费的数据的数据类型由泛型指定

- 常用方法

  Consumer：包含两个方法

方法名	说明
void accept(T t)	对给定的参数执行此操作
default Consumer andThen(Consumer after)	返回一个组合的Consumer，依次执行此操作，然后执行after操作

- 代码演示

```
public class ConsumerDemo {
    public static void main(String[] args) {
        //操作一
        operatorString("林青霞", s -> System.out.println(s));
        //操作二
        operatorString("林青霞", s -> System.out.println(new
StringBuilder(s).reverse().toString()));

        System.out.println("--------");
        //传入两个操作使用andThen完成
        operatorString("林青霞", s -> System.out.println(s), s ->
System.out.println(new StringBuilder(s).reverse().toString()));
    }

    //定义一个方法，用不同的方式消费同一个字符串数据两次
    private static void operatorString(String name, Consumer<String> con1,
Consumer<String> con2) {
//        con1.accept(name);
//        con2.accept(name);
        con1.andThen(con2).accept(name);
    }
```

```
    //定义一个方法，消费一个字符串数据
    private static void operatorString(String name, Consumer<String> con) {
        con.accept(name);
    }
}
```

1.7Consumer接口练习之按要求打印信息【应用】

- 案例需求

  String[] strArray = {"林青霞,30", "张曼玉,35", "王祖贤,33"};

  字符串数组中有多条信息，请按照格式：“姓名：XX,年龄：XX"的格式将信息打印出来

  要求：

  把打印姓名的动作作为第一个Consumer接口的Lambda实例

  把打印年龄的动作作为第二个Consumer接口的Lambda实例

  将两个Consumer接口按照顺序组合到一起使用

- 示例代码

```
public class ConsumerTest {
    public static void main(String[] args) {
        String[] strArray = {"林青霞,30", "张曼玉,35", "王祖贤,33"};

        printInfo(strArray, str -> System.out.print("姓名：" + str.split(",")[0]),
                str -> System.out.println(",年龄：" +
Integer.parseInt(str.split(",")[1])));
    }

    private static void printInfo(String[] strArray, Consumer<String> con1,
Consumer<String> con2) {
        for (String str : strArray) {
            con1.andThen(con2).accept(str);
        }
    }
}
```

1.8常用函数式接口之Predicate【应用】

- Predicate接口

  Predicate接口通常用于判断参数是否满足指定的条件

- 常用方法

方法名	说明
boolean test(T t)	对给定的参数进行判断(判断逻辑由Lambda表达式实现)，返回一个布尔值
default Predicate negate()	返回一个逻辑的否定，对应逻辑非
default Predicate and(Predicate other)	返回一个组合判断，对应短路与
default Predicate or(Predicate other)	返回一个组合判断，对应短路或

- 代码演示

```
public class PredicateDemo01 {
    public static void main(String[] args) {
        boolean b1 = checkString("hello", s -> s.length() > 8);
        System.out.println(b1);

        boolean b2 = checkString("helloworld",s -> s.length() > 8);
        System.out.println(b2);

    }

    //判断给定的字符串是否满足要求
    private static boolean checkString(String s, Predicate<String> pre) {
//        return !pre.test(s);
        return pre.negate().test(s);
    }
}

public class PredicateDemo02 {
    public static void main(String[] args) {
        boolean b1 = checkString("hello", s -> s.length() > 8);
        System.out.println(b1);
        boolean b2 = checkString("helloworld", s -> s.length() > 8);
        System.out.println(b2);

        boolean b3 = checkString("hello",s -> s.length() > 8, s -> s.length() <
15);
        System.out.println(b3);

        boolean b4 = checkString("helloworld",s -> s.length() > 8, s -> s.length()
< 15);
        System.out.println(b4);
    }

    //同一个字符串给出两个不同的判断条件，最后把这两个判断的结果做逻辑与运算的结果作为最终的结果
    private static boolean checkString(String s, Predicate<String> pre1,
Predicate<String> pre2) {
        return pre1.or(pre2).test(s);
    }
```

```
    //判断给定的字符串是否满足要求
    private static boolean checkString(String s, Predicate<String> pre) {
        return pre.test(s);
    }
}
```

1.9Predicate接口练习之筛选满足条件数据【应用】

- 练习描述
  - String[] strArray = {"林青霞,30", "柳岩,34", "张曼玉,35", "貂蝉,31", "王祖贤,33"};
  - 字符串数组中有多条信息，请通过Predicate接口的拼装将符合要求的字符串筛选到集合ArrayList中，并遍历ArrayList集合
  - 同时满足如下要求：姓名长度大于2；年龄大于33
- 分析
  - 有两个判断条件,所以需要使用两个Predicate接口,对条件进行判断
  - 必须同时满足两个条件,所以可以使用and方法连接两个判断条件
- 示例代码

```
public class PredicateTest {
    public static void main(String[] args) {
        String[] strArray = {"林青霞,30", "柳岩,34", "张曼玉,35", "貂蝉,31", "王祖
贤,33"};

        ArrayList<String> array = myFilter(strArray, s -> s.split(",")[0].length()
> 2,
                s -> Integer.parseInt(s.split(",")[1]) > 33);

        for (String str : array) {
            System.out.println(str);
        }
    }

    //通过Predicate接口的拼装将符合要求的字符串筛选到集合ArrayList中
    private static ArrayList<String> myFilter(String[] strArray, Predicate<String>
pre1, Predicate<String> pre2) {
        //定义一个集合
        ArrayList<String> array = new ArrayList<String>();

        //遍历数组
        for (String str : strArray) {
            if (pre1.and(pre2).test(str)) {
                array.add(str);
            }
        }

        return array;
    }
}
```

1.10常用函数式接口之Function【应用】

- Function接口

  Function<T,R>接口通常用于对参数进行处理，转换(处理逻辑由Lambda表达式实现)，然后返回一个新的值

- 常用方法

方法名	说明
R apply(T t)	将此函数应用于给定的参数
default Function andThen(Function after)	返回一个组合函数，首先将该函数应用于输入，然后将after函数应用于结果

- 代码演示

```
public class FunctionDemo {
    public static void main(String[] args) {
        //操作一
        convert("100",s -> Integer.parseInt(s));
        //操作二
        convert(100,i -> String.valueOf(i + 566));

        //使用andThen的方式连续执行两个操作
        convert("100", s -> Integer.parseInt(s), i -> String.valueOf(i + 566));
    }

    //定义一个方法，把一个字符串转换int类型，在控制台输出
    private static void convert(String s, Function<String,Integer> fun) {
//        Integer i = fun.apply(s);
        int i = fun.apply(s);
        System.out.println(i);
    }


    //定义一个方法，把一个int类型的数据加上一个整数之后，转为字符串在控制台输出
    private static void convert(int i, Function<Integer,String> fun) {
        String s = fun.apply(i);
        System.out.println(s);
    }


    //定义一个方法，把一个字符串转换int类型，把int类型的数据加上一个整数之后，转为字符串在控制台输出
    private static void convert(String s, Function<String,Integer> fun1,
Function<Integer,String> fun2) {

        String ss = fun1.andThen(fun2).apply(s);
        System.out.println(ss);
    }

}
```

1.11Function接口练习之按照指定要求操作数据【应用】

- 练习描述
  - String s = "林青霞,30";
  - 请按照我指定的要求进行操作：

    1:将字符串截取得到数字年龄部分

    2:将上一步的年龄字符串转换成为int类型的数据

    3:将上一步的int数据加70，得到一个int结果，在控制台输出
  - 请通过Function接口来实现函数拼接
- 示例代码

```
public class FunctionTest {
    public static void main(String[] args) {
        String s = "林青霞,30";
        convert(s, ss -> ss.split(",")[1], Integer::parseInt, i -> i + 70);
    }

    private static void convert(String s, Function<String, String> fun1,
Function<String, Integer> fun2, Function<Integer, Integer> fun3) {
        int i = fun1.andThen(fun2).andThen(fun3).apply(s);
        System.out.println(i);
    }
}
```

2.Strem流

2.1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

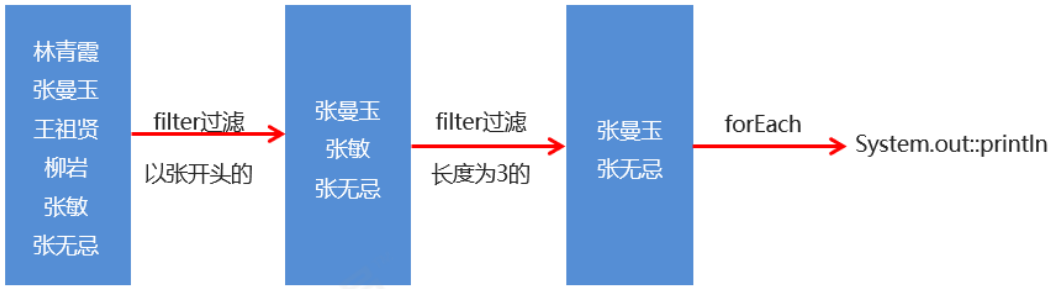
```
public class StreamDemo {
    public static void main(String[] args) {
        //创建一个集合，存储多个字符串元素
        ArrayList<String> list = new ArrayList<String>();

        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

2.2Stream流的常见生成方式【应用】

- Stream流的思想



- 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Stream接口的静态方法of(T... values)生成流
        String[] strArray = {"hello","world","java"};
        Stream<String> strArrayStream = Stream.of(strArray);
        Stream<String> strArrayStream2 = Stream.of("hello", "world", "java");
        Stream<Integer> intStream = Stream.of(10, 20, 30);
    }
}
```

2.3Stream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
Stream sorted()	返回由此流的元素组成的流，根据自然顺序排序
Stream sorted(Comparator comparator)	返回由该流的元素组成的流，根据提供的Comparator进行排序
Stream map(Function mapper)	返回由给定函数应用于此流的元素的结果组成的流
IntStream mapToInt(ToIntFunction mapper)	返回一个IntStream其中包含将给定函数应用于此流的元素的结果

- 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);
    }
}
```

- sorted代码演示

```
public class StreamDemo04 {
    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("linqingxia");
        list.add("zhangmanyu");
        list.add("wangzuxian");
        list.add("liuyan");
        list.add("zhangmin");
        list.add("zhangwuji");

        //需求1：按照字母顺序把数据在控制台输出
//        list.stream().sorted().forEach(System.out::println);

        //需求2：按照字符串长度把数据在控制台输出
        list.stream().sorted((s1,s2) -> {
            int num = s1.length()-s2.length();
            int num2 = num==0?s1.compareTo(s2):num;
            return num2;
        }).forEach(System.out::println);
    }
}
```

- map&mapToInt代码演示

```
public class StreamDemo05 {
    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("10");
```

```
        list.add("20");
        list.add("30");
        list.add("40");
        list.add("50");

        //需求：将集合中的字符串数据转换为整数之后在控制台输出
//        list.stream().map(s -> Integer.parseInt(s)).forEach(System.out::println);
//        list.stream().map(Integer::parseInt).forEach(System.out::println);
//        list.stream().mapToInt(Integer::parseInt).forEach(System.out::println);

        //int sum() 返回此流中元素的总和
        int result = list.stream().mapToInt(Integer::parseInt).sum();
        System.out.println(result);
    }
}
```

2.4Stream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);
    }
}
```

2.5Stream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

```
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;
    }
}


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()));
    }
}
```

2.6Stream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);
        }
    }
}
```