# Stream接口实例化

Java.util.Stream

利用Collection 的方法:

default Stream<E> stream() JDK1.8才有

范例:

|  |
| --- |
| List<String> all = new ArrayList<>();  Stream<String> stream = all.stream(); |

# 2.Stream的一些方法

|  |  |
| --- | --- |
| long count() | 返回数据个数和集合中的size()一样 |
| Stream<T> distinct() | 消除重复数据的方法,返回消除重复后的Stream |
| <R,A> R collect(Collector<? super T,A,R> collector) | 这个方法主要是将Stream转化为集合和集合中的stream()相反  对于这个函数的参数是通过Collectors这个类里的toList()方法返回的  Collectors里的方法  public static <T> Collector<T,?,List<T>> toList() |
| Stream<T> filter(Predicate<? super T> predicate) | 返回过滤后的Stream,参数类型为断言型接口 |
| <R> Stream<R> map(Function<? super T,? extends R> mapper) | 对每个数据进行处理,返回处理后的数据，参数类型为功能型接口 |
| boolean allMatch(Predicate<? super T> predicate | 全匹配 |
| boolean anyMatch(Predicate<? super T> predicate) | 匹配任意一个 |
| void forEach(Consumer<? super T> action) | 消费性接口 |
| Stream<T> skip(long n) | 设置跳过的数据行数 |
| Stream<T> limit(long maxSize) | 设置取出的数据个数 |
| Optional<T> reduce(BinaryOperator<T> accumulator)  @FunctionalInterface  public interface BinaryOperator<T>  extends BiFunction<T,T,T>  @FunctionalInterface  public interface BiFunction<T,U,R>  R apply(T t, U u) 和功能型接口类似 | 对数据进行统计  返回类型Optional是一个容器用来存储<T>类型的值,通过get()方法能取得容器里的值 |
| DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper)  注意:Stream是BaseStream的子接口,而DoubleStream也是BaseStream的子接口 | 按照Double处理 |
| IntStream mapToInt(ToIntFunction<? super T> mapper) | 按照Int处理 |
| LongStream mapToLong(ToLongFunction<? super T> mapper) | 按照Long处理 |

范例:数据过滤操作

|  |
| --- |
| List<String> all = new ArrayList<>();  all.add("Android");  all.add("Java");  all.add("Ios");  all.add("jsp");  all.add("ORACLE");  Stream<String> stream = all.stream();  //功能型接口接受一个参数,返回一个数据  stream = stream.distinct().map((x)->x.toLowerCase());  // filter 的参数为断言型接口boolean test(T t);  List<String> newAll = stream.filter((x)->x.contains("a")).collect(Collectors.toList());  newAll.forEach(System.out::println); |

范例:数据分页操作

|  |
| --- |
| List<String> all = new ArrayList<>();  all.add("Android");  all.add("Java");  all.add("Ios");  all.add("jsp");  all.add("ORACLE");  Stream<String> stream = all.stream();  List<String> newALl = stream.skip(2).limit(2).collect(Collectors.toList());  System.out.print(newALl); |

范例:匹配查询

|  |
| --- |
| List<String> all = new ArrayList<>();  all.add("Android");  all.add("Java");  all.add("Ios");  all.add("jsp");  all.add("ORACLE");  Stream<String> stream = all.stream();  //断言型接口 boolean test(T t);  if(stream.anyMatch((x)->x.contains("jsp"))){  System.out.print("数据存在");} |

在实例情况中可能要匹配多个条件在Predicate里提供了些方法:

或操作:default Predicate<T> and(Predicate<? super T> other)

与操作:default [Predicate](file:///C:\Users\lw\Downloads\jdk-8u144-docs-all\docs\api\java\util\function\Predicate.html)<[T](file:///C:\Users\lw\Downloads\jdk-8u144-docs-all\docs\api\java\util\function\Predicate.html)> or([Predicate](file:///C:\Users\lw\Downloads\jdk-8u144-docs-all\docs\api\java\util\function\Predicate.html)<? super [T](file:///C:\Users\lw\Downloads\jdk-8u144-docs-all\docs\api\java\util\function\Predicate.html)> other)

范例:多匹配

|  |
| --- |
| List<String> all = new ArrayList<>();  all.add("Android");  all.add("Java");  all.add("Ios");  all.add("jsp");  all.add("ORACLE");  Stream<String> stream = all.stream();  Predicate<String> p1 = (x)->x.contains("jsp");  Predicate<String> p2 = (x)->x.contains("Ios");  //断言型接口 boolean test(T t);  if(stream.anyMatch(p1.or(p2))){  System.out.print("数据存在");  } |

范例:mapReduce 统计操作

|  |
| --- |
| List<ShopCar> all = new ArrayList<ShopCar>();  all.add(new ShopCar("王惊雷娃娃",800.0,20));  all.add(new ShopCar("生姜",1.0,2000));  all.add(new ShopCar("鸭子",19.0,20));  all.add(new ShopCar("盆",20.0,10));  // all.stream().map((x)->x.getAmount() \* x.getPrice()).forEach(System.out::println);  double xx = all.stream().map((x)->x.getAmount() \* x.getPrice()).reduce((sum,m)->sum+m).get();  System.out.print("总金额" + xx);  class ShopCar{  private String name; //商品名称  private double price; //商品单价  private int amount; //购买个数  public ShopCar(String name, double price, int amount) {  this.name = name;  this.price = price;  this.amount = amount;  }  public String getName() {  return name;  }  public double getPrice() {  return price;  }  public int getAmount() {  return amount;  }  } |

范例:

|  |
| --- |
| public class test {  public static void main(String args[]){  List<ShopCar> all = new ArrayList<ShopCar>();  all.add(new ShopCar("王惊雷娃娃",800.0,20));  all.add(new ShopCar("生姜",1.0,2000));  all.add(new ShopCar("鸭子",19.0,20));  all.add(new ShopCar("盆",20.0,10));  DoubleSummaryStatistics dss = all.stream().mapToDouble((sc)->sc.getPrice()\*sc.getAmount()).summaryStatistics();  System.out.println("商品个数:" + dss.getCount());  System.out.println("总话费:" + dss.getSum());  System.out.println("平均话费:" + dss.getAverage());  System.out.println("最高花费:" + dss.getMax());  System.out.println("最底花费:" + dss.getMin());  }  }  class ShopCar{  private String name; //商品名称  private double price; //商品单价  private int amount; //购买个数  public ShopCar(String name, double price, int amount) {  this.name = name;  this.price = price;  this.amount = amount;  }  public String getName() {  return name;  }  public double getPrice() {  return price;  }  public int getAmount() {  return amount;  }  } |