JDK8 新特性

前面部分的内容请观看视频:

https://www.wkcto.com/course/112

https://www.wkcto.com/course/113

3 集合增强

3.1 Stream 流

3.1.1 流概述

流是 JDK8 新增的成员,允许以声明性方式处理数据集合,可以把 Stream 流看作是遍历数据集合的一个高级迭代器

使用流的好处:

代码以声明性方式书写:说明想要完成什么,而不是说明如何完成一个操作

可以把几个基础操作连接起来,来表达复杂的数据处理的流水线,同时保持代码清晰可读

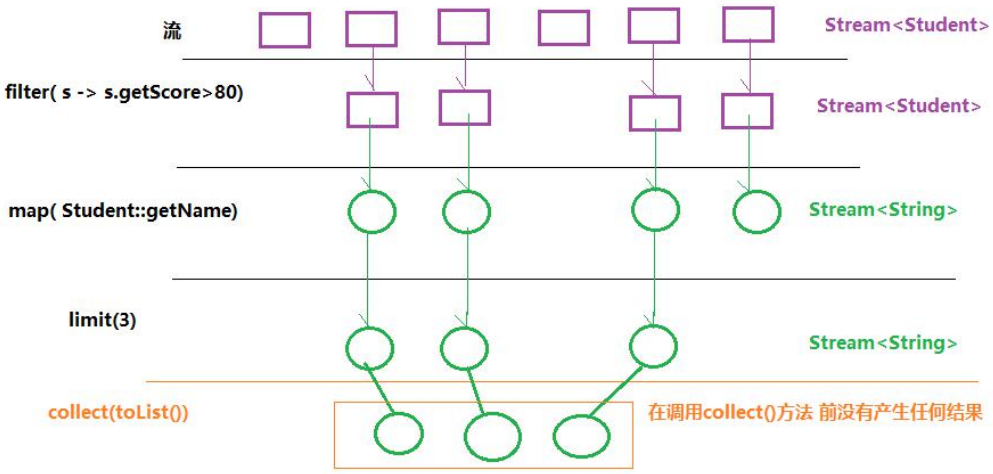
流是什么?

从支持数据处理操作的源生成元素序列.数据源可以是集合,数组或 IO 资源

从操作角度来看,流与集合是不同的. 流不存储数据值; 流的目的是处理数据,它是关于算法与计算的.

如果把集合作为流的数据源,创建流时不会导致数据流动; 如果流的终止操作需要值时,流会从集合中获取值; 流只使用一次

流中心思想是延迟计算,流直到需要时才计算值



流使用时一般包括三件事:

1) 一个数据源(如集合)来执行一个查询;
2) 一个中间操作链,形成一条流的流水线
3) 一个终端操作,执行流水线,生成结果

3.1.2 如何获得流

```
package com.wkcto.stream;

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

```
import java.util.stream.Stream;

/**
 * 流的基本操作
 * Author : 动力节点老崔
 */
public class Test01 {
    public static void main(String[] args) {
        //1 如何获得流,可以通过 Collection 集合,数据, 根据字面量获得流

        //1.1 通过 Collection 获得流
        List<String> list = new ArrayList<>();
        Collections.addAll(list,"wkcto","hello", "jj", "dd", "mm", "bjpowernode");
        Stream<String> stream1 = list.stream();
//          System.out.println(stream1);
        stream1.forEach(System.out::println);

        //1.2 根据数据获得流
        String[]data = {"zhangsan","lisi","wangwu"};
        Stream<String> stream2 = Arrays.stream(data);
        stream2.forEach(s -> System.out.print(s + "    "));
        System.out.println();

        //1.3 直接通过值获得流
        Stream<String> stream3 = Stream.of("1", "2", "3", "4");
        stream3.forEach(s -> System.out.print(s + "    "));
        System.out.println();

        //1.4 无限流
        Stream.iterate(100, x->x+3).limit(10).forEach(s -> System.out.print(s + "    "));
        System.out.println();

    }
}
```

3.1.3 流的使用

```
package com.wkcto.stream;

import javax.sound.midi.Soundbank;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

/**
 * 流的基本操作

 * Author : 动力节点老崔

 */
public class Test02 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        Collections.addAll(list, "wkcto", "good", "hello", "jj", "xx", "jj", "jj","bjpowernode","xx",
"good");
        //1)通过 List 集合获得 Stream 流
        Stream<String> stream = list.stream();

        //2 流的中间操作

        //2.1 流的筛选与切片

        //distinct()去掉重复的数据
        stream.distinct().forEach(System.out::println);
        System.out.println("----------------------");

        //filter()过滤
//                                stream.filter(x->x.length()    >    4).forEach(System.out::println);
//java.lang.IllegalStateException: 流已关闭，流只能使用一次
        list.stream().filter(x->x.length() > 4).forEach(System.out::println);
        System.out.println("----------------------");
```

```
//sort()排序
list.stream().sorted(String::compareTo).forEach(System.out::println);
System.out.println("----------------------");

//limt()截断操作
list.stream().limit(5).forEach(System.out::println);
System.out.println("----------------------");

//skip()跳过
list.stream().skip(3).forEach(System.out::println);
System.out.println("----------------------");

//2.2 map 映射
//map 映射转换元素, map 方法接受一个函数，把这个函数应用于每个元素上,并将它映射为一个新的元素
list.stream()
        .map(String::toUpperCase)     //为每个元素应用 toUpperCase()把小写转换为大写
        .forEach(System.out::println);
System.out.println("----------------------");

list.stream()
        .map(String::length)
        .map(len -> len + "    ")
        .forEach(System.out::print);
System.out.println();
System.out.println("----------------------");

//转换为数值流
List<Integer> integerList = Arrays.asList(54,1,78,90,345);
IntStream intStream = integerList.stream().mapToInt(x -> x);
intStream.forEach(x -> System.out.print( x + " "));
```

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

//2.3 匹配与查找操作

//allMatch()判断流中所有的元素是否都匹配给定的谓词
System.out.println( list.stream().allMatch(s -> s.length() > 3 ) );        //false
//anyMatch()判断流中是否有某个元素可以匹配指定的谓词
System.out.println( list.stream().anyMatch(s -> s.equals("wkcto")));       //true
//noneMathc()判断流中的元素是否都没有匹配指定谓词的
System.out.println( list.stream().noneMatch(s -> s.equals("jj")));          //false

//查找
System.out.println(list.stream().filter(s -> s.length() > 10).findAny().get() );
System.out.println( list.stream().filter(s -> s.length() > 20).findFirst().orElse("不存在"));

//3 Stream 流的终端操作

//3.1 forEach 遍历
list.stream().forEach(System.out::println);

//3.2 cout 统计
System.out.println( list.stream().filter(x->x.length() > 2).count());

//3.3 reduce 归纳合并
Optional<String> reduce = list.stream().reduce((s1, s2) -> s1 + "--" + s2);
System.out.println(reduce.get());
reduce.ifPresent(System.out::println);

//数值操作
List<Integer> list2 = Arrays.asList(6,21,87,34,1,78,54);
//求和,指定初始值
```

```
System.out.println(list2.stream().reduce(0, Integer::sum) );

//求和,没有初始值

System.out.println(list2.stream().reduce((x,y)->x+y).orElse(0) );

//最值

System.out.println(list2.stream().reduce(Math::max).get() );
System.out.println(list2.stream().reduce(Math::min).get() );


//3.4 映射到数值流

System.out.println(list2.stream().mapToInt(x->x).sum() );            //求和

System.out.println(list2.stream().mapToInt(x->x).max().getAsInt() );      //最大值

System.out.println(list2.stream().mapToInt(x->x).min().orElse(0) );       //最小值

System.out.println(list2.stream().mapToDouble(x->x).average().getAsDouble());      //平
均值

System.out.println(list2.stream().max(Integer::compareTo).get());
list2.stream().min(Integer::compareTo).ifPresent(System.out::println);


//3.5 collect 归约

//把 stream 流转换为集合

Set<String> stringSet = list.stream().collect(Collectors.toSet());
System.out.println( stringSet );


//把 Stream 流转换为数组

Object[] objects = list.stream().toArray();
System.out.println( Arrays.toString(objects));
String[] toArray = list.stream().toArray(String[]::new);
System.out.println(Arrays.toString(toArray));


//Stream 流转换为字符串

String collect = list.stream().collect(Collectors.joining(","));
System.out.println(collect);
```

```
    }
}
```

3.1.4 Stream 在开发中的应用

```
package com.wkcto.stream;


import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

import java.util.function.BinaryOperator;

import java.util.stream.Collectors;


/**

 * Author : 动力节点老崔

 */

public class Test03 {

    public static void main(String[] args) {

        //把 Car 小汽车的数据保存到 List 集合中

        List<Car> carshop = new ArrayList<>();

        carshop.add(new Car("Benz", false, 68, CarType.SUV));
```

```
        carshop.add(new Car("Audi", true, 28, CarType.HATCHBACK));
        carshop.add(new Car("BMW", false, 88, CarType.HATCHBACK));
        carshop.add(new Car("Geeley", true, 18, CarType.HATCHBACK));
        carshop.add(new Car("Xiali", true, 8, CarType.THREECOMPARTMENT));
        carshop.add(new Car("Haval", false, 18, CarType.SUV));
        carshop.add(new Car("Jeep", true, 38, CarType.SUV));
        carshop.add(new Car("Honda", false, 28, CarType.THREECOMPARTMENT));
        carshop.add(new Car("Chery", true, 18, CarType.THREECOMPARTMENT));
        carshop.add(new Car("Benz", false, 58, CarType.THREECOMPARTMENT));


        //1 根据价格降序排序后,显示汽车品牌
        carshop.stream()
                .sorted((c1,c2) -> c2.getPrice()-c1.getPrice())
                .map(Car::getBrand)
                .distinct()
                .forEach(System.out::println);
```

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


//2 找出已卖的车，按价格升序排序
carshop.stream()
        .filter(car -> car.isSold() )
        .sorted(Comparator.comparing(Car::getPrice))
        .forEach(System.out::println);
System.out.println("--------------------");


//3 查看有哪些车型
carshop.stream()
        .map(car -> car.getType())
        .distinct()
        .forEach(System.out::println);
System.out.println("--------------------");


//4SUV 型号的车有哪些
List<Car> collect = carshop.stream()
        .filter(car -> car.getType() == CarType.SUV)
        .collect(Collectors.toList());
System.out.println( collect );
```

```
        // 40 万以下的车的品牌，排序
        carshop.stream()
                .filter(car -> car.getPrice() < 40)
                .map(Car::getBrand)
                .distinct()
                .sorted()
                .forEach(System.out::println);
        System.out.println("--------------------");

        //6)统计没有卖出去的车的数量
        System.out.println( carshop.stream().filter(car -> !car.isSold()).count() );

        //7)判断是否有 Geeley 品牌的汽车
        System.out.println( carshop.stream().anyMatch(car -> "Geeley".equals(car.getBrand())));

        //8)最贵的车的价格

System.out.println(carshop.stream().map(Car::getPrice).reduce(Integer::
```

```
max).get() );


        //9)显示已卖出去最贵的车
        System.out.println( carshop.stream().filter(Car::isSold)
//                                .reduce(BinaryOperator.maxBy((car1,
car2)->(car1.getPrice()- car2.getPrice())))
                    .collect(Collectors.maxBy(Comparator.comparing(Car::
getPrice)))
                     .get()
        );
    }
}
```

```
package com.wkcto.stream;


import java.util.Objects;

/**
 * Author : 动力节点老崔
 */
public class Car {
```

```
    private String brand;      //品牌
    private    boolean sold;    //是否已卖
    private int price;
    private CarType type;      //车型

    public Car(String brand, boolean sold, int price, CarType type) {
        this.brand = brand;
        this.sold = sold;
        this.price = price;
        this.type = type;
    }

    public Car() {
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Car car = (Car) o;
        return sold == car.sold &&
```

```
                price == car.price &&
                Objects.equals(brand, car.brand) &&
                type == car.type;
    }

    @Override
    public int hashCode() {
        return Objects.hash(brand, sold, price, type);
    }

    @Override
    public String toString() {
        return "Car{" +
                "brand='" + brand + '\'' +
                ", sold=" + sold +
                ", price=" + price +
                ", type=" + type +
                '}';
    }

    public String getBrand() {
```

```
        return brand;
    }

    public Car setBrand(String brand) {
        this.brand = brand;
        return this;
    }

    public boolean isSold() {
        return sold;
    }

    public Car setSold(boolean sold) {
        this.sold = sold;
        return this;
    }

    public int getPrice() {
        return price;
    }
```

```
    public Car setPrice(int price) {
        this.price = price;
        return this;
    }

    public CarType getType() {
        return type;
    }

    public Car setType(CarType type) {
        this.type = type;
        return this;
    }
}
```

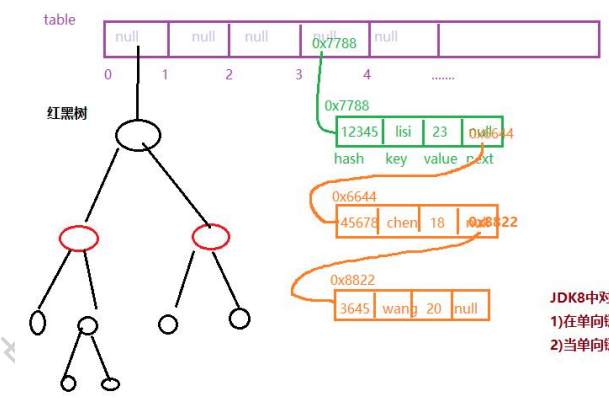
```
package com.wkcto.stream;

/**
 * Author : 动力节点老崔
 */
public enum CarType {
```

```
    HATCHBACK, THREECOMPARTMENT,SUV

}
```


3.2 对 HashMap 性能的提升



1)HashMap底层是哈希表,哈希表是一个数组,数组的每个元素是一个单向链表

2)hashmap.put(key, value)添加键值对时,
2.1 根据key键的哈希码,经常hash函数得到hash值
2.2 再使用hash值计算出对应数组的下标i
2.3 访问table[i]数组元素
如果table[i]元素为null, 就会生成一个结点存储到table[i]位置

如果table[i]元素不为null, 遍历table[i]单向链表的每个结点,是否存在某个结点的key与当前的键equals()相等,如果相等就使用新的value值替换原来的值, 如果所有结点的key与当前的键都不匹配, , 会创建一个新的结点插入到链表的尾部

JDK8中对HashMap性能的提升
1)在单向链表中,把新添加的结点添加链表的尾部
2)当单向链表中结点的数量>=8 个时, 系统会把单向链表的结构转换为红黑树

4 新增了 java.time 包

java.util.Date 日期类不是线程安全的

在 JDK8 中引入了线程安全的时间类,

```
package com.wkcto.time;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.time.temporal.ChronoUnit;

/**
 * java.time 包中线程安全的日期类
 * Author : 动力节点老崔
 */
public class Test {
    public static void main(String[] args) {
        //1)LocalDate 日期类

        LocalDate date = LocalDate.now();     //当前日期
        System.out.println( date );         //2019-04-24


        //2)当前时间
        LocalTime time = LocalTime.now();
        System.out.println( time );         //19:50:35.344


        //3)日期与时间
        LocalDateTime now = LocalDateTime.now();
        System.out.println(now);              //2019-04-24T19:51:03.726
```

```
//4)返回当前日期的各个属性值
System.out.println( now.getYear());
System.out.println( now.getMonthValue());
System.out.println( now.getDayOfMonth());
System.out.println( now.getHour());
System.out.println( now.getMinute());
System.out.println( now.getSecond());
System.out.println( now.getNano());        //在毫秒数后面添加 6 个 0


//5)自定义时间
LocalDateTime another = LocalDateTime.of(2100, 10, 12, 8, 58, 28);
System.out.println( another );


//6)使用 plus 增加时间, minus 可以减少时间
another = another.plusYears(1);
System.out.println(another);

another = another.plusMonths(1);
System.out.println(another);

another = another.plusDays(1);
System.out.println(another);

another = another.plusHours(1);
System.out.println(another);

another = another.plusMinutes(1);
System.out.println(another);

another = another.plusSeconds(1);
System.out.println(another);

another = another.plus(10, ChronoUnit.YEARS);
System.out.println(another);


//7)设置时间
another = another.withYear(2088);
```

```
        another = another.withMonth(7);
        System.out.println( another);

        //8)判断星期几
        System.out.println( another.getDayOfWeek());

        //9)把日期转换为指定格式的字符串
        DateTimeFormatter formatter = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG);
        //LocalDateTime 类中有一个 format( DateTimeFormatter) 实例方法可以把日期转换为指定格式字符串
        System.out.println( now.format(formatter) );          //2019 年 4 月 24 日 下午 08 时 03 分 28 秒

        formatter = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT);
        System.out.println( now.format(formatter));       //19-4-24 下午 8:03

        //自定义日期格式: y 年, M 月 d 日 H 小时 m 分钟 s 秒 S 毫秒
        formatter = DateTimeFormatter.ofPattern("yyyy 年 MM 月 dd 日 HH:mm:ss SSS");
        System.out.println( now.format(formatter));   //2019 年 04 月 24 日 20:05:16 338

        //10)把字符串转换为日期
        String text = "2089 年 8 月 12 日 8:28:58";
        formatter = DateTimeFormatter.ofPattern("yyyy 年 M 月 dd 日 H:mm:ss");
        // LocalDateTime.parse(text, formatter)静态方法,可以把 text 文件以 formatter 格式转换为日期对象
        another = LocalDateTime.parse(text, formatter);
```

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

        }
}
```