Java8 Stream流式编程，极大解放你的生产力！

原创 海向 [Java知音](javascript:void(0);) 昨天

**java8自带常用的函数式接口**

* **Predicate<T> boolean test(T t)** 传入一个参数返回boolean值
* **Consumer<T> void accept(T t)** 传入一个参数，无返回值
* **Function<T,R> R apply(T t)** 传入一个参数，返回另一个类型

**准备数据**

//计算机俱乐部  
**private** **static** List<Student> computerClub = Arrays.asList(  
        **new** Student("2015134001", "小明", 15, "1501"),  
        **new** Student("2015134003", "小王", 14, "1503"),  
        **new** Student("2015134006", "小张", 15, "1501"),  
        **new** Student("2015134008", "小梁", 17, "1505")  
);  
//篮球俱乐部  
**private** **static** List<Student> basketballClub = Arrays.asList(  
        **new** Student("2015134012", "小c", 13, "1503"),  
        **new** Student("2015134013", "小s", 14, "1503"),  
        **new** Student("2015134015", "小d", 15, "1504"),  
        **new** Student("2015134018", "小y", 16, "1505")  
);  
//乒乓球俱乐部  
**private** **static** List<Student> pingpongClub = Arrays.asList(  
        **new** Student("2015134022", "小u", 16, "1502"),  
        **new** Student("2015134021", "小i", 14, "1502"),  
        **new** Student("2015134026", "小m", 17, "1504"),  
        **new** Student("2015134027", "小n", 16, "1504")  
);  
  
**private** **static** List<List<Student>> allClubStu = **new** ArrayList<>();  
allClubStu.add(computerClub);  
allClubStu.add(basketballClub);  
allClubStu.add(pingpongClub);

**常用的stream三种创建方式**

* 集合 **Collection.stream()**
* 静态方法 **Stream.of**
* 数组 **Arrays.stream**

//1.集合  
Stream<Student> stream = basketballClub.stream();  
//2.静态方法  
Stream<String> stream2 = Stream.of("a", "b", "c");  
//3.数组  
String[] arr = {"a","b","c"};  
Stream<String> stream3 = Arrays.stream(arr);

**Stream的终止操作**

* **foreach(Consumer c)** 遍历操作
* **collect(Collector)** 将流转化为其他形式
* **max(Comparator)** 返回流中最大值
* **min(Comparator)** 返回流中最小值
* **count** 返回流中元素综述

**Collectors 具体方法**

* **toList List<T>** 把流中元素收集到List
* **toSet Set<T>** 把流中元素收集到Set
* **toCollection Coolection<T>** 把流中元素收集到Collection中
* **groupingBy Map<K,List<T>>** 根据K属性对流进行分组
* **partitioningBy Map<boolean, List<T>>** 根据boolean值进行分组

//此处只是演示 此类需求直接用List构造器即可  
List<Student> collect = computerClub.stream().collect(Collectors.toList());  
Set<Student> collect1 = pingpongClub.stream().collect(Collectors.toSet());  
//注意key必须是唯一的 如果不是唯一的会报错而不是像普通map那样覆盖  
Map<String, String> collect2 = pingpongClub.stream()  
        .collect(Collectors.toMap(Student::getIdNum, Student::getName));  
//分组 类似于数据库中的group by  
Map<String, List<Student>> collect3 = pingpongClub.stream()  
        .collect(Collectors.groupingBy(Student::getClassNum));  
//字符串拼接 第一个参数是分隔符 第二个参数是前缀 第三个参数是后缀  
String collect4 = pingpongClub.stream().map(Student::getName).collect(Collectors.joining(",", "【", "】"));  
  //【小u,小i,小m,小n】  
//三个俱乐部符合年龄要求的按照班级分组  
Map<String, List<Student>> collect5 = Stream.of(basketballClub, pingpongClub, computerClub)  
        .flatMap(e -> e.stream().filter(s -> s.getAge() < 17))  
        .collect(Collectors.groupingBy(Student::getClassNum));  
//按照是否年龄>16进行分组 key为true和false  
ConcurrentMap<Boolean, List<Student>> collect6 = Stream.of(basketballClub, pingpongClub, computerClub)  
        .flatMap(Collection::stream)  
        .collect(Collectors.groupingByConcurrent(s -> s.getAge() > 16));

**Stream的中间操作**

1. **filter(Predicate)** 筛选流中某些元素

//筛选1501班的学生  
computerClub.stream().filter(e -> e.getClassNum().equals("1501")).forEach(System.out::println);  
//筛选年龄大于15的学生  
List<Student> collect = computerClub.stream().filter(e -> e.getAge() > 15).collect(Collectors.toList());

1. **map(Function f)** 接收流中元素，并且将其映射成为新元素，例如从student对象中取name属性

//篮球俱乐部所有成员名 + 暂时住上商标^\_^,并且获取所有队员名  
List<String> collect1 = basketballClub.stream()  
        .map(e -> e.getName() + "^\_^")  
        .collect(Collectors.toList());  
collect1.forEach(System.out::println);  
//小c^\_^^\_^  
//小s^\_^^\_^  
//小d^\_^^\_^  
//小y^\_^^\_^

1. **flatMap(Function f)** 将所有流中的元素并到一起连接成一个流

//获取年龄大于15的所有俱乐部成员  
List<Student> collect2 = Stream.of(basketballClub, computerClub, pingpongClub)  
        .flatMap(e -> e.stream().filter(s -> s.getAge() > 15))  
        .collect(Collectors.toList());  
collect2.forEach(System.out::println);  
  
//用双层list获取所有年龄大于15的俱乐部成员  
List<Student> collect3 = allClubStu.stream()  
        .flatMap(e -> e.stream().filter(s -> s.getAge() > 15))  
        .collect(Collectors.toList());  
collect3.forEach(System.out::println);

1. **peek(Consumer c)** 获取流中元素，操作流中元素，与foreach不同的是不会截断流，可继续操作流

//篮球俱乐部所有成员名 + 赞助商商标^\_^,并且获取所有队员详细内容  
List<Student> collect = basketballClub.stream()  
        .peek(e -> e.setName(e.getName() + "^\_^"))  
        .collect(Collectors.toList());  
collect.forEach(System.out::println);  
//Student{idNum='2015134012', name='小c^\_^', age=13, classNum='1503'}  
//Student{idNum='2015134013', name='小s^\_^', age=14, classNum='1503'}  
//Student{idNum='2015134015', name='小d^\_^', age=15, classNum='1504'}  
//Student{idNum='2015134018', name='小y^\_^', age=16, classNum='1505'}

1. **distinct()** 通过流所生成元素的equals和hashCode去重
2. **limit(long val)** 截断流，取流中前val个元素
3. **sorted(Comparator)** 产生一个新流，按照比较器规则排序
4. **sorted()** 产生一个新流，按照自然顺序排序

List<String> list = Arrays.asList("b","b","c","a");  
list.forEach(System.out::print); //bbca  
List<String> collect = list.stream().distinct().sorted().collect(Collectors.toList());  
collect.forEach(System.out::print);//abc  
//获取list中排序后的top2 即截断取前两个  
List<String> collect1 = list.stream().distinct().sorted().limit(2).collect(Collectors.toList());  
collect1.forEach(System.out::print);//ab

**匹配**

1. **booelan allMatch(Predicate)** 都符合
2. **boolean anyMatch(Predicate)** 任一元素符合
3. **boolean noneMatch(Predicate)** 都不符合

**boolean** b = basketballClub.stream().allMatch(e -> e.getAge() < 20);  
**boolean** b1 = basketballClub.stream().anyMatch(e -> e.getAge() < 20);  
**boolean** b2 = basketballClub.stream().noneMatch(e -> e.getAge() < 20);

**寻找元素**

1. **findFirst**——返回第一个元素
2. **findAny**——返回当前流中的任意元素

Optional<Student> first = basketballClub.stream().findFirst();  
**if** (first.isPresent()) {  
    Student student = first.get();  
    System.out.println(student);  
}  
  
Optional<Student> any = basketballClub.stream().findAny();  
**if** (any.isPresent()) {  
    Student student2 = any.get();  
    System.out.println(student2);  
}  
Optional<Student> any1 = basketballClub.stream().parallel().findAny();  
System.out.println(any1);

**计数和极值**

1. **count**——返回流中元素的总个数
2. **max**——返回流中最大值
3. **min**——返回流中最小值

**long** count = basketballClub.stream().count();  
Optional<Student> max = basketballClub.stream().max(Comparator.comparing(Student::getAge));  
**if** (max.isPresent()) {  
    Student student = max.get();  
}  
Optional<Student> min = basketballClub.stream().min(Comparator.comparingInt(Student::getAge));  
**if** (min.isPresent()) {  
    Student student = min.get();  
}

**END**