链滴

java8-lambda

作者：chenxc

原文链接：https://ld246.com/article/1540352399193

来源网站：链滴

基本语法：

```
([形参列表，不带数据类型]) ->{
//执行语句
[return ...;]
}
```

变量作用域

lambda 表达式只能引用标记了 final 的外层局部变量，这就是说不能在 lambda 内部修改定义在域的局部变量，否则会编译错误。

过滤案例

```
public static void main(String[] args) {
        List<Book> bookList = prepareData();

        // 要被找出的书的ID
        ArrayList<String> ids = new ArrayList<String>();
        ids.add("3");
        ids.add("6");
        ids.add("8");
        ids.add("9");

        // 存放过滤结果的列表
        List<Book> result = null;

        // 使用lambda表达式过滤出结果并放到result列表里
        result = bookList.stream()
                .filter((Book b) -> ids.contains(b.getId()))
                .collect(Collectors.toList());

        // 打印结果列表
        if (result != null && !result.isEmpty()) {
            result.forEach((Book b) -> System.out.println(b.getId() + "  " + b.getName()));
        }
    }
```

案例

例1、用lambda表达式实现Runnable

```
// Java 8之前：
new Thread(new Runnable() {
    @Override
    public void run() {
    System.out.println("Before Java8, too much code for too little to do");
    }
}).start();

//Java 8方式：
new Thread( () -> System.out.println("In Java8, Lambda expression rocks !!") ).start();
```

例2、使用lambda表达式对列表进行迭代

如果你使过几年Java，你就知道针对集合类，最常见的操作就是进行迭代，并将业务逻辑应用于各个素，例如处理订单、交易和事件的列表。由于Java是命令式语言，Java 8之前的所有循环代码都是顺的，即可以对其元素进行并行化处理。如果你想做并行过滤，就需要自己写代码，这并不是那么容易通过引入lambda表达式和默认方法，将做什么和怎么做的问题分开了，这意味着Java集合现在知道样做迭代，并可以在API层面对集合元素进行并行处理。下面的例子里，我将介绍如何在使用lambda不使用lambda表达式的情况下迭代列表。你可以看到列表现在有了一个 forEach() 方法，它可以迭所有对象，并将你的lambda代码应用在其中。

```
// Java 8之前：
List features = Arrays.asList("Lambdas", "Default Method", "Stream API", "Date and Time API")

for (String feature : features) {
    System.out.println(feature);
}

// Java 8之后：
List features = Arrays.asList("Lambdas", "Default Method", "Stream API", "Date and Time API")

features.forEach(n -> System.out.println(n));

// 使用Java 8的方法引用更方便，方法引用由::双冒号操作符标示，

// 看起来像C++的作用域解析运算符

features.forEach(System.out::println);
```

例3、Java 8中使用lambda表达式的Map和Reduce示例

本例介绍最广为人知的函数式编程概念map。它允许你将对象进行转换。

例如在本例中，我们将 costBeforeTax 列表的每个元素转换成为税后的值。我们将 x -> x*x lambda 达式传到 map() 方法，后者将其应用到流中的每一个元素。然后用 forEach() 将列表元素打印出来。用流API的收集器类，可以得到所有含税的开销。有 toList() 这样的方法将 map 或任何其他操作的结合并起来。由于收集器在流上做终端操作，因此之后便不能重用流了。你甚至可以用流API的 reduce() 方法将所有数字合成一个，下一个例子将会讲到。

```
// 不使用lambda表达式为每个订单加上12%的税
List costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
for (Integer cost : costBeforeTax) {
    double price = cost + .12*cost;
    System.out.println(price);
}
// 使用lambda表达式
List costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
costBeforeTax.stream().map((cost) -> cost + .12*cost).forEach(System.out::println);
```

例4、通过过滤创建一个String列表

过滤是Java开发者在大规模集合上的一个常用操作，而现在使用lambda表达式和流API过滤大规模数集合是惊人的简单。流提供了一个 filter() 方法，接受一个 Predicate 对象，即可以传入一个lambda 达式作为过滤逻辑。下面的例子是用lambda表达式过滤Java集合，将帮助理解。

```
// 创建一个字符串列表，每个字符串长度大于2
List<String> filtered = strList.stream().filter(x -> x.length()> 2).collect(Collectors.toList());
System.out.printf("Original List : %s, filtered list : %s %n", strList, filtered);
```

另外，关于 filter() 方法有个常见误解。在现实生活中，做过滤的时候，通常会丢弃部分，但使用filter(方法则是获得一个新的列表，且其每个元素符合过滤原则。

例5、复制不同的值，创建一个子列表

本例展示了如何利用流的 distinct() 方法来对集合进行去重。

```
// 用所有不同的数字创建一个正方形列表
List<Integer> numbers = Arrays.asList(9, 10, 3, 4, 7, 3, 4);
List<Integer> distinct = numbers.stream().map( i -> i*i).distinct().collect(Collectors.toList());
System.out.printf("Original List : %s,  Square Without duplicates : %s %n", numbers, distinct);
输出：
Original List : [9, 10, 3, 4, 7, 3, 4],  Square Without duplicates : [81, 100, 9, 16, 49]
```