链滴

读书笔记——《Java 8 实战》系列之 Lambda 表达式 (一)

作者：Jesmin

原文链接：https://ld246.com/article/1507950434190

来源网站：链滴

在上一篇博客中，我们一起学习了**行为参数化**这个有趣的概念。同时我们也知道了在Java 8之前，匿类可以用来减少那些只使用一次的实体类的啰嗦代码。

而Java 8中，Lambda表达式的出现能够让我们以一种更加简洁的方式去表示一个行为或传递代码。

简单来说，Lambda表达式是一种匿名机制，它是一种没有声明名称的方法，和匿名类一样，它也可作为参数被传递给方法。

还记得上一篇博客中我们给出的Lambda表达式的例子么？

不使用匿名类时：

```
//声明一个实现了StudentPredicate接口的实体类
class StudentHeightPredicate implements StudentPredicate{

    @Override
    public boolean test(Student s) {
        if(s.getHeight()>=180)
            return true;
        return false;
    }

}

   public static void main(String[] args) {

        //返回身高超过180cm的学生
       List<Student> filteredStudents = studentFilter(students, new StudentHeightPredicate());

    }
```

使用匿名类的话：

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

        //返回身高超过180cm的学生
     List<Student> filteredStudents = studentFilter(students, new StudentPredicate() {
            @Override
            public boolean test(Student s) {
                if(s.getHeight()>=180)
                    return true;
                return false;
            }
        });

    }
```

使用Lambda表达式的话：

```
public static void main(String[] args){
    List<Student> filteredStudents2 = studentFilter(students,
            (Student s) -> {
                    if(s.getHeight()>=180)
                        return true;
                    return false;
                   }
    );
}
```

与上一段使用了匿名类机制的代码相比，使用Lambda表达式更像是将test方法的主体传入到了studenFilter()方法当中。

那么接下来我们就一起来学习一下Lambda表达式的写法规则。

Lambda表达式包含以下三个部分：

- 参数列表——在本例中就是test方法中的参数Student s,被圆括号包围着
- 箭头——箭头 ->用来将参数列表与Lambda表达式的主题分隔开
- Lambda主体——也就是使用匿名类代码中test方法中的实现代码

在大家都了解了Lambda表达式的语法规则后，我们给出一些Lambda表达式的例子，大家可以从中习到如何简化使用Lambda表达式

例子1：

```
(String s) -> s.length()
```

***tips: 参数为String 类型的 s ，方法的主体返回了一个int类型的值s.length()，当方法主体只返回个值时，主体无需被花括号包围，同时 return 可以被省略**

例子2：

```
(Student s) -> s.getHeight()>=180
```

***tips 参数为Student 类型的 s, 方法的主体返回了一个Boolean类型的判断 s.getHeight()>=180，仔细观察这个表达式其实就是我们上面给studentFilter例子的简化版，**

大家可以学习一下这种简便的写法

例子3：

```
(int x, int y) -> {
    System.out.println("Result:");
    System.out.println(x+y);
 }
```

***tips 参数为2个int类型的值，方法的主体是两句打印输出，同时方法的主体中没有返回值，需要注**

的是，当使用花括号包围起方法主体时不能忘记每句语句结尾的分号

例子4：

```
() -> 42
```

***tips 没有参数，直接返回一个int类型的值42**

例子5：

```
s -> s.getHeight()>=180
```

***tips 例子2的简化版，当Java的编译器嗯能够根据上下文去判断参数 s 的类型时，可以省略参数类，同时当参数只有一个的时候，可以省略包围参数列表的圆括号**

那么看了这么多Lambda表达式的例子，也该考验一下大家是否真的完全掌握了Lambda表达式的写，判断下列的Lambda表达式中哪些是合法的哪些是非法的。

测验：

1. () -> { }
2. String s -> "hello world"
3. (s,s2) -> {return "hello world";}
4. s -> { "hello world" }
5. (String s) -> { return "hello world"}

解析

1. 第一个表达式合法，参数为空，返回值为空
2. 第二个表达式非法，在参数数量只有一个但是没有省略参数类型时，圆括号不能省略
3. 第三个表达式合法，Java编译器推断出s,s2的参数类型，则可省略，在方法主体内可以显式地使用return关键字
4. 第四个表达式非法，在方法主体花括号内返回值必须由return 关键字显式返回，若没有花括号则可省略return关键字
5. 第五个表达式非法，在方法主体花括号内return语句之后缺少分号

怎么样，大家有没有被这些Lambda表达式测验题难住呢?

下表给出了一些Lambda表达式的例子和使用案例

<table><tr><td>使用案例</td><td>Lambda示例</td></tr><tr><td>布尔表达式</td><td>List<String> list) -> list.isEmpty()</td></tr><tr><td>创建对象</td><td>( ) -> new String()</td></tr><tr><td>消费一个对象</td><td>(String s) -> { System.out.println(s);}</td></tr><tr><td>从某个对象中抽取属性</td><td>(Student s) -> s.getHeight()</td></tr><tr><td>组合两个值</td><td>(int a, int b) -> a*b </td></tr><tr><td>比较两个对象</td><td>(Student s1, Student s2) -> s1.getHeight().compareTo(s2.getHeight())</td></tr></table>

学习了这么多Lambda表达式相关的知识，同学们肯定会问，我们究竟在**哪里**能使用到它呢？

Lambda表达式的一个重要使用场景就是在**函数式接口**上，有些同学可能对这个**函数式接口**没有概念别担心，接下来我们首先了解一下这个**函数式接口**究竟是什么

还记得上一篇博客中，我们写的Predicate<T>这个接口么？它就是一个典型的**函数式接口**，因为在口中我们仅定义了一个抽象方法：

```
interface Predicate<T>{
    //这里使用泛型来传入对象
    boolean test(T t);
}
```

简单来说，**函数式接口就是只定义了一个抽象方法的接口。**回想一下，在平日里我们经常接触到函数接口有哪些？

```
public interface Comparator<T> {
    int compareTo(T o1, T o2);
}

public interface Runnable{
    void run();
}

public interface Callable<V>{
    V call();
}
```

为了检查你的理解程度，下面的测验能够帮助你测试是否掌握了函数式接口的概念：

下列哪些接口是函数式接口?

```
public interface Adder{
    int add(int a, int b);
}

public interface SmartAdder extends Adder{
    int add(double a, double b);
}
public interface Nothing{

}
```

解析:

只有Adder接口是函数式接口,SmartAdder从Adder接口中继承了一个add方法，因此不是函数式接，Nothing接口也不是函数式接口，因为它并没有声明方法。

在Java 8中，Lambda表达式允许你直接以内联的形式为函数式接口的抽象方法提供实现，并把整个达式作为函数式接口的实例。尽管我们匿名类也可以完成同样的事，但是却比较笨重，因为我们不得先提供一个具体实现，再直接将它内联实例化。有的同学看到这里可能会有点晕，没关系，下面我就

大家提供一些例子来理解这段话。

```
//使用匿名类声明一个Runnable接口的实例
Runnable r1 = new Runnable(){
    public void run(){
        System.out.println("hello world 1");
    }
};


//使用Lambda表达式声明一个Runnable接口的实例
Runnable r2 = () -> System.out.println("hello world 2");

//打印hello world 1
new Thread(r1).start();

//打印hello world 2
new Thread(r2).start();

//直接将Lambda表达式当作参数，打印 hello world 3
new Thread(() -> System.out.println("hello world 3")).start();
```

关注新的Java API的同学会发现，函数式接口在声明时会伴随着@FunctionalInterface的标注。当们使用了这个标注，但是接口中确声明了超过一个的抽象方法时，编译器就会报错。当然这个标注并是必须的，我么可以定义只有一个抽象方法的函数式接口但是却不加这个标注。

接下来，我给大家准备了一个常见的编程模式，让大家在实战中增强对**行为参数化**与**Lambda表达式**运用。

不知道同学们有没有注意到，在编程中我们经常会遇到以下的一种情形：

```
//1.完成一些固定的前期工作
doSomeStartingWork();

//2.完成真正的我们需要做的事，这些事往往是不固定的，可变的
doActualWork();

//3.完成一些固定的结尾工作
doSomeEndingWork();
```

给大家举个简单的例子吧：

```
public static void main(String[] args) throws IOException {
    // TODO Auto-generated method stub
    //完成一些固定的前期工作，初始化一个BufferReader与需要打开的资源
    BufferedReader br = new BufferedReader(new FileReader("data.txt"));

    //完成我们真正需要做的工作，这里仅仅是打印了一行内容，这些工作是可能发生改变的，比如
成打印两行内容
```

```
        System.out.println(br.readLine());

        //完成一些固定的结尾工作，关闭资源
        br.close();
    }
```

在上面的例子中，这个打开资源与关闭资源是固定不变的，可变的仅仅是我们需要对资源做怎样的处。再比如，我们经常需要测试一段代码的执行时间，我们会怎么做呢？记录开始时间，记录结束时间两者相减，可变的仅仅是我们需要执行的代码。像这种开始与结尾总是固定，只有中间需要处理的重代码可变的模式，我们通常称之为环绕执行(Execute Around)模式。

看到这种模式的特点之后，大家有没有觉得很眼熟。这不正式行为参数化最擅长干的事情么，**将重要代码变成参数传递给方法，以抽象应对改变。**

那接下来，博主就一步步带着大家完成一个测试代码执行时间的例子。

- 首先我们定义一个函数式接口

```
    //在这个接口中，我们只声明了一个没有任何返回值，不需要任何参数的抽象方法，这个方法的作就是为了执行我们需要知道执行时间的代码
    @FunctionalInterface
    interface Executor{
        void execute();
    }
```

- 接下来，我们定义我们的主方法

```
    public static long getExecutionTime(Executor e) {
        //记录起始时间
        long startTime = System.currentTimeMillis();
        //等待被执行代码执行
        e.execute();
        //记录结束时间
        long endTime = System.currentTimeMillis();
        //返回总共消耗的时间，并以秒为单位
        return (endTime - startTime)/ 1000;
    }
```

- 然后我们就可以获得我们需要执行代码的执行时间了

```
    public static void main(String[] args)  {
        //在getExecutionTime()方法中，使用Lambda表达式作为它的参数，在这个例子中我们什么也做，只是让主线程Sleep了5秒
        long executionTime = getExecutionTime(() -> {try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }});

        System.out.println("代码执行时间为: " +executionTime +"s");
    }
```

学了这么多关于Lambda表达式的知识，大家也需要消化一下了，其它关于Lambda表达式的知识将下一篇博客中与大家分享。

原文链接：读书笔记——《Java 8 实战》系列之 Lambda 表达式 (一)