链滴

Java8 lambda 表达式语法

作者：liononon

原文链接：https://ld246.com/article/1474811300746

来源网站：链滴

<h3>Java8初体验（一）lambda表达式语法</h3>

感谢同事【天锦】的投稿。投稿请联系 tengfei@ifeve.com
本文主要记录自己学习Java8的历程，方便大家一起探讨和自己的备忘。因为本人也是刚刚开始习Java8，所以文中肯定有错误和理解偏差的地方，希望大家帮忙指出，我会持续修改和优化。本文该系列的第一篇，主要介绍Java8对屌丝码农最有吸引力的一个特性—lambda表达式。
<h1>java8的安装</h1>
工欲善其器必先利其器，首先安装JDK8。过程省略，大家应该都可以自己搞定。但是有一点这强调一下（Windows系统）：目前我们工作的版本一般是java 6或者java 7，所以很多人安装java8本都是学习为主。这样就在自己的机器上会存在多版本的JDK。而且大家一般是希望在命令行中执行java命令是基于老版本的jdk。但是在安装完jdk8并且没有设置path的情况下，你如果在命令行中输入：java -version，屏幕上会显示是jdk 8。这是因为jdk8安装的时候，会默认在C:/Windows/System32增加java.exe，这个调用的优先级比path设置要高。所以即使path里指定是老版本的jdk，但是执行java命令显示的依然是新版本的jdk。这里我们要做的就是删除C:/Windows/System32中的java.exe文（不要手抖！）。

<h1>Lambda初体验</h1>
下面进入本文的正题–lambda表达式。首先我们看一下什么是lambda表达式。以下是维百科上对于”Lambda expression”的解释：

<blockquote>
 a function (or a subroutine) defined, and possibly called, without being bound to n identifier。
</blockquote>

简单点说就是：一个不用被绑定到一个标识符上，并且可能被调用的函数。这个解释还不够通俗lambda表达式可以这样定义（不精确，自己的理解）：一段带有输入参数的可执行语句块。这样就较好理解了吧？一例胜千言。有读者反馈：不理解Stream的含义，所以这里先提供一个没用stream的ambda表达式的例子。

<pre class="brush: java">List<String> names = ...;
Collections.sort(names, (o1, o2) -> o1.compareTo(o2));</pre>

<pre class="brush: java">List<String> names = ...;
Collections.sort(names, new Comparator<String>() {
  @Override
  public int compare(String o1, String o2) {
    return o1.compareTo(o2);
  }
});</pre>

上面两段代码分别是：使用lambda表达式来排序和使用匿名内部类来排序。这个例子可以很明的看出lambda表达式简化代码的效果。接下来展示lambda表达式和其好基友Stream的配合。
<pre class="brush: java">List<String> names = new ArrayList<>();
names.add("TaoBao");
names.add("ZhiFuBao");
List<String> lowercaseNames = names.stream().map((String name) -> {return name.tLowerCase();}).collect(Collectors.toList());</pre>
这段代码就是对一个字符串的列表，把其中包含的每个字符串都转换成全小写的字符串（熟悉Grovy和Scala的同学肯定会感觉很亲切）。注意代码第四行的map方法调用，这里map方法就是接受一个lambda表达式（其实是一个java.util.function.Function的实例，后面会介绍）。

为什么需要Lambda表达式呢？在尝试回答这个问题之前，我们先看看在Java8之前，如果我们做上面代码的操作应该怎么办。
先看看普通青年的代码：

<pre class="brush: java">List<String> names = new ArrayList<>();
names.add("TaoBao");
names.add("ZhiFuBao");
List<String> lowercaseNames = new ArrayList<>();
for (String name : names) {
  lowercaseNames.add(name.toLowerCase());
}</pre>

接下来看看文艺青年的代码（借助Guava）：

<pre class="brush: java">List<String> names = new ArrayList<>();
names.add("TaoBao");
names.add("ZhiFuBao");
List<String> lowercaseNames = FluentIterable.from(names).transform(new Function<String, String>() {
  @Override
  public String apply(String name) {
    return name.toLowerCase();
  }
}).toList();</pre>

在此，我们不再讨论普通青年和文艺青年的代码风格孰优孰劣（有兴趣的可以去google搜索“命令式编程vs声明式编程”）。本人更加喜欢声明式的编程风格，所以偏好文艺青年的写。但是在文艺青年代码初看起来看起来干扰信息有点多，Function匿名类的构造语法稍稍有点冗长。以Java8的lambda表达式给我们提供了创建SAM（Single Abstract Method）接口更加简单的语法。
<h1>
 Lambda语法详解</h1>
我们在此抽象一下lambda表达式的一般语法：

<pre class="brush: java">(Type1 param1, Type2 param2, ..., TypeN paramN) -> {
  statment1;
  statment2;
  //.............
  return statmentM;
}</pre>

从lambda表达式的一般语法可以看出来，还是挺符合上面给出的非精确版本的定义–“一段带有输入参数的可执行语句块”。
上面的lambda表达式语法可以认为是最全的版本，写起来还是稍稍有些繁琐。别着急，下面陆介绍一下lambda表达式的各种简化版：
1. 参数类型省略–绝大多数情况，编译器都可以从上下文环境中推断出lambda表达式的数类型。这样lambda表达式就变成了：

<pre class="brush: java">(param1,param2, ..., paramN) -> {
  statment1;
  statment2;
  //.............
  return statmentM;
}</pre>

所以我们最开始的例子就变成了（省略了List的创建）：

<pre class="brush: java">List<String> lowercaseNames = names.stream().map((name) -gt; {return name.toLowerCase();}).collect(Collectors.toList());</pre>

2. 当lambda表达式的参数个数只有一个，可以省略小括号。lambda表达式简写为：

<pre class="brush: java">param1 -> {
  statment1;
  statment2;
  //.............
  return statmentM;
}</pre>

所以最开始的例子再次简化为：

<pre class="brush: java">List<String> lowercaseNames = names.stream().map(name -&t; {return name.toLowerCase();}).collect(Collectors.toList());</pre>

3. 当lambda表达式只包含一条语句时，可以省略大括号、return和语句结尾的分号。lambda表式简化为：

<pre class="brush: java">param1 -> statment</pre>

所以最开始的例子再次简化为：

<pre class="brush: java">List<String> lowercaseNames = names.stream().map(name -&t; name.toLowerCase()).collect(Collectors.toList());</pre>

4. 使用Method Reference(具体语法后面介绍)

<pre class="brush: java">List<String> lowercaseNames = names.stream().map(String::towerCase).collect(Collectors.toList());</pre>

<h1>Lambda表达式眼中的外部世界</h1>
我们前面所有的介绍，感觉上lambda表达式像一个闭关锁国的家伙，可以访问给它传递的参数也能自己内部定义变量。但是却从来没看到其访问它外部的变量。是不是lambda表达式不能访问其部变量？我们可以这样想：lambda表达式其实是快速创建SAM接口的语法糖，原先的SAM接口都可访问接口外部变量，lambda表达式肯定也是可以（不但可以，在java8中还做了一个小小的升级，后会介绍）。

<pre class="brush: java">String[] array = {"a", "b", "c"};
for(Integer i : Lists.newArrayList(1,2,3)){
  Stream.of(array).map(item -> Strings.padEnd(item, i, '@')).forEach(System.out::println);
}</pre>

上面的这个例子中，map中的lambda表达式访问外部变量Integer i。并且可以访问外部变量是lmbda表达式的一个重要特性，这样我们可以看出来lambda表达式的三个重要组成部分：

输入参数
可执行语句
存放外部变量的空间

不过lambda表达式访问外部变量有一个非常重要的限制：变量不可变（只是引用不可变，而不真正的不可变）。

<pre class="brush: java">String[] array = {"a", "b", "c"};
for(int i = 1; i<4; i++){
  Stream.of(array).map(item -> Strings.padEnd(item, i, '@')).forEach(System.out::println);
}</pre>

上面的代码，会报编译错误。因为变量i被lambda表达式引用，所以编译器会隐式的把其当成fina来处理（ps：大家可以想象问什么上一个例子不报错，而这个报错。）细心的读者肯定会发现不对啊以前java的匿名内部类在访问外部变量的时候，外部变量必须用final修饰。Bingo，在java8对这个限做了优化（前面说的小小优化），可以不用显示使用final修饰，但是编译器隐式当成final来处理。</>
<h2>lambda眼中的this</h2>
在lambda中，this不是指向lambda表达式产生的那个SAM对象，而是声明它的外部对象。

<h1>方法引用（Method reference）和构造器引用（construct reference）</e></h1>
<h2>方法引用</h2>
前面介绍lambda表达式简化的时候，已经看过方法引用的身影了。方法引用可以在某些条件成的情况下，更加简化lambda表达式的声明。方法引用语法格式有以下三种：

objectName::instanceMethod
ClassName::staticMethod
ClassName::instanceMethod

前两种方式类似，等同于把lambda表达式的参数直接当成instanceMethod|staticMethod的参来调用。比如System.out::println等同于x->System.out.println(x)；Math::max等同于(x, y)-&gt Math.max(x,y)。
最后一种方式，等同于把lambda表达式的第一个参数当成instanceMethod的目标对象，其他余参数当成该方法的参数。比如String::toLowerCase等同于x->x.toLowerCase()。
<h2>构造器引用</h2>
构造器引用语法如下：ClassName::new，把lambda表达式的参数当成ClassName构造器的参数。例如BigDecimal::new等同于x->new BigDecimal(x)。
<h2>吐槽一下方法引用</h2>
表面上看起来方法引用和构造器引用进一步简化了lambda表达式的书写，但是个人觉得这方面有Scala的下划线语法更加通用。比较才能看出，翠花，上代码！

<pre class="brush: java">List<String> names = new ArrayList<>();
names.add("TaoBao");
names.add("ZhiFuBao");
names.stream().map(name -> name.charAt(0)).collect(Collectors.toList());</pre>

上面的这段代码就是给定一个String类型的List，获取每个String的首字母，并将其组合成新的Lis。这段代码就没办法使用方法引用来简化。接下来，我们简单对比一下Scala的下划线语法（不必太结Scala的语法，这里只是做个对比）：

<pre class="brush: java">List[String] names = ...
names.map(_.charAt(0))</pre>

在Scala中基本不用写lambda表达式的参数声明。