anakinf

Java8 lambda表达式10个示例

1. 实现Runnable线程案例

使用() -> {} 替代匿名类:

```
//Before Java 8:
new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("Before Java8 ");
    }
}).start();

//Java 8 way:
new Thread( () -> System.out.println("In Java8!") ).start();
```

```
Output:
too much code, for too little to do
Lambda expression rocks !!
```

你可以使用 下面语法实现Lambda:

(params) -> expression
(params) -> statement
(params) -> { statements }

如果你的方法并不改变任何方法参数，比如只是输出，那么可以简写如下：

() -> System.out.println("Hello Lambda Expressions");

如果你的方法接受两个方法参数，如下：

(int even, int odd) -> even + odd

2.实现事件处理

如果你曾经做过Swing 编程，你将永远不会忘记编写事件侦听器代码。使用lambda表达式如下所示写出更好的事件侦听器的代码。

```
// Before Java 8:
JButton show =  new JButton("Show");
show.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
          System.out.println("without lambda expression is boring");
       }
    });


// Java 8 way:
show.addActionListener((e) -> {
```

```
    System.out.println("Action !! Lambda expressions Rocks");
});
```

在java 8中你可以使用Lambda表达式替代丑陋的匿名类。

3.使用Lambda表达式遍历List集合

```
//Prior Java 8 :
List features = Arrays.asList("Lambdas", "Default Method",
"Stream API", "Date and Time API");
for (String feature : features) {
   System.out.println(feature);
}

//In Java 8:
List features = Arrays.asList("Lambdas", "Default Method", "Stream API",
 "Date and Time API");
features.forEach(n -> System.out.println(n));

// Even better use Method reference feature of Java 8
// method reference is denoted by :: (double colon) operator
// looks similar to score resolution operator of C++
features.forEach(System.out::println);

Output:
Lambdas
Default Method
Stream API
Date and Time API
```

方法引用是使用两个冒号::这个操作符号。

4.使用Lambda表达式和函数接口

MONO
MSSQL(2)
Mysql
Object C(5)
Oracle
PHP
Spring Boot(7)
Spring MVC(5)
Winform
服务器搭建(2)
设计模式
物联网(1)

随笔档案

2018年7月 (1)

为了支持函数编程，Java 8加入了一个新的包java.util.function，其中有一个接口java.util.function.Predicate是支持Lambda函数编程：

```
public static void main(args[]){
  List languages = Arrays.asList("Java", "Scala", "C++", "Haskell", "Lisp");

  System.out.println("Languages which starts with J :");
  filter(languages, (str)->str.startsWith("J"));

  System.out.println("Languages which ends with a ");
  filter(languages, (str)->str.endsWith("a"));

  System.out.println("Print all languages :");
  filter(languages, (str)->true);

   System.out.println("Print no language : ");
   filter(languages, (str)->false);

   System.out.println("Print language whose length greater than 4:");
   filter(languages, (str)->str.length() > 4);
}

 public static void filter(List names, Predicate condition) {
    for(String name: names)  {
       if(condition.test(name)) {
          System.out.println(name + " ");
       }
    }
  }
}

Output:
Languages which starts with J :
```

2018年4月 (1)
2018年3月 (3)
2017年11月 (13)
2017年10月 (5)
2012年6月 (1)
2012年3月 (1)
2012年2月 (7)
2012年1月 (7)
2011年12月 (15)
2011年11月 (1)
2011年10月 (6)
2011年9月 (2)

最新评论

```
Java
Languages which ends with a
Java
Scala
Print all languages :
Java
Scala
C++
Haskell
Lisp
Print no language :
Print language whose length greater than 4:
Scala
Haskell
```

```
//Even better
 public static void filter(List names, Predicate condition) {
    names.stream().filter((name) -> (condition.test(name)))
        .forEach((name) -> {System.out.println(name + " ");
    });
 }
```

你能看到来自Stream API 的filter方法能够接受 Predicate参数, 能够允许测试多个条件。

5.复杂的结合Predicate 使用

java.util.function.Predicate提供and(), or() 和 xor()可以进行逻辑操作，比如为了得到一串字符串中以"J"开头的4个长度:

```
// We can even combine Predicate using and(), or() And xor() logical functions
 // for example to find names, which starts with J and four letters long, you
 // can pass combination of two Predicate
 Predicate<String> startsWithJ = (n) -> n.startsWith("J");
 Predicate<String> fourLetterLong = (n) -> n.length() == 4;
```

1. Re:spring boot jar 进程自动停止，自动终止，不能后台持续运行

您好，能问一下是什么原因导致这个问题的吗

--大雄丶

2. Re:C# 读取和编辑 MP3 ID3 属性的详细操作和说明 (ID3v2)

只能英文，调试了半天，还以为什么坏了。。
还有，mp3没有id3时添加图片失败。

--望云风

3. Re:Android权限Uri.parse的详细资料

发短信有多个号码，这个URI.parse(),方法怎么用，我看系统自带的有群发功能，该如何调用系统的群发功能呢？是调用系统自带的信息发送，我现在只是把联系人传给他该如何做。我的联系人在数据库中不再系统.....

--古来征战几人回

4. Re:Android 程序中像素(px)跟 单位dp(dip)之间的转换

```
names.stream()
      .filter(startsWithJ.and(fourLetterLong))
      .forEach((n) -> System.out.print("\nName, which starts with
            'J' and four letter long is : " + n));
```

其中startsWithJ.***and***(fourLetterLong)是使用了AND逻辑操作。

6.使用Lambda实现Map 和 Reduce

最流行的函数编程概念是map，它允许你改变你的对象，在这个案例中，我们将costBeforeTeax集合中每个元素改变了增加一定的数值，我们将Lambda表达式 x -> x*x传送map()方法，这将应用到stream中所有元素。然后我们使用 forEach() 打印出这个集合的元素.

```
// applying 12% VAT on each purchase
// Without lambda expressions:
List costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
for (Integer cost : costBeforeTax) {
      double price = cost + .12*cost;
      System.out.println(price);
}

// With Lambda expression:
List costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
costBeforeTax.stream().map((cost) -> cost + .12*cost)
                      .forEach(System.out::println);

Output
112.0
224.0
336.0
448.0
560.0
```

在开发中应该会用到的，Android根据分辨率进行单位转换-(dp,sp转像素px)

---_-!

5. Re:反编译Silverlight项目

很多是在文章结尾才说说明，我在前面就就申明的了，有什么不好呢。

--anakinf

阅读排行榜

1. Android权限Uri.parse的详细资料(5446)

2. Java8 lambda表达式10个示例(2893)

3. spring boot jar 进程自动停止，自动终止，不能后台持续运行(2756)

4. iOS---弹出提示对话框(2130)

5. Android源码下载方法详解(1992)

评论排行榜

```
112.0
224.0
336.0
448.0
560.0
```

reduce() 是将集合中所有值结合进一个，Reduce类似SQL语句中的sum(), avg() 或count(),

```
// Applying 12% VAT on each purchase// Old way:List costBeforeTax =Arrays.asList(100, 200, 300, 400, 500);
double total =0;
for (Integer cost :costBeforeTax) {
 double price = cost + .12*cost;
 total = total + price;

}
System.out.println("Total : " + total);

// New way:List costBeforeTax =Arrays.asList(100, 200, 300, 400, 500);
double bill = costBeforeTax.stream().map((cost) -> cost + .12*cost)
                                    .reduce((sum, cost) -> sum + cost)
                                    .get();
System.out.println("Total : " + bill);

OutputTotal :1680.0Total:1680.0
```

7.通过filtering 创建一个字符串String的集合

Filtering是对大型Collection操作的一个通用操作，Stream提供filter()方法，接受一个Predicate对象，意味着你能传送lambda表达式作为一个过滤逻辑进入这个方法：

```
// Create a List with String more than 2 characters
List<String> filtered = strList.stream().filter(x -> x.length()> 2)
                                        .collect(Collectors.toList());
System.out.printf("Original List : %s, filtered list : %s %n",
```

1. 反编译Silverlight项目(4)

2. C#编程中的66个好习惯，你有多少个(3)

3. Android权限Uri.parse的详细资料(2)

4. Android 权限(1)

5. spring boot jar 进程自动停止，自动终止，不能后台持续运行(1)

推荐排行榜

1. C#编程中的66个好习惯，你有多少个(3)

2. NeatUpload 大文件上传(1)

3. Android应用开发——系统自带样式Android:theme(1)

4. Android源码下载方法详解(1)

5. Eclipse下对Android模拟器截屏(1)

```
                strList, filtered);

Output :
Original List : [abc, , bcd, , defg, jk], filtered list : [abc, bcd, defg]
```

8.对集合中每个元素应用函数

我们经常需要对集合中元素运用一定的功能，如表中的每个元素乘以或除以一个值等等.

```
// Convert String to Uppercase and join them using coma
List<String> G7 = Arrays.asList("USA", "Japan", "France", "Germany",
                                "Italy", "U.K.","Canada");
String G7Countries = G7.stream().map(x -> x.toUpperCase())
                                .collect(Collectors.joining(", "));
System.out.println(G7Countries);

Output :
USA, JAPAN, FRANCE, GERMANY, ITALY, U.K., CANADA
```

上面是将字符串转换为大写，然后使用逗号串起来。

9.通过复制不同的值创建一个子列表

使用Stream的distinct()方法过滤集合中重复元素。

```
// Create List of square of all distinct numbers
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);

Output :
```

```
Original List : [9, 10, 3, 4, 7, 3, 4],  Square Without
                                    duplicates : [81, 100, 9, 16, 49]
```

10.计算List中的元素的最大值，最小值，总和及平均值

```
//Get count, min, max, sum, and average for numbers
List<Integer> primes = Arrays.asList(2, 3, 5, 7, 11, 13, 17, 19, 23, 29);
IntSummaryStatistics stats = primes.stream().mapToInt((x) -> x)
                                    .summaryStatistics();
System.out.println("Highest prime number in List : " + stats.getMax());
System.out.println("Lowest prime number in List : " + stats.getMin());
System.out.println("Sum of all prime numbers : " + stats.getSum());
System.out.println("Average of all prime numbers : " + stats.getAverage());

Output :
Highest prime number in List : 29
Lowest prime number in List : 2
Sum of all prime numbers : 129
Average of all prime numbers : 12.9
```

Java 8 lambda表达式示例

我个人对Java 8发布非常激动，尤其是lambda表达式和流API。越来越多的了解它们，我能写出更干净的代码。虽然一开始并不是这样。第一次看到用lambda表达式写出来的Java代码时，我对这种神秘的语法感到非常失望，认为它们把Java搞得不可读，但我错了。花了一天时间做了一些lambda表达式和流API示例的练习后，我开心的看到了更清晰的Java代码。这有点像学习泛型，第一次见的时候我很讨厌它。我甚至继续使用老版Java 1.4来处理集合，直到有一天，朋友跟我介绍了使用泛型的好处（才意识到它的好处）。所以基本立场就是，不要畏惧lambda表达式以及方法引用的神秘语法，做几次练习，从集合类中提取、过滤数据之后，你就会喜欢上它。下面让我们开启学习Java 8 lambda表达式的学习之旅吧，首先从简单例子开始。

例1、用lambda表达式实现Runnable

我开始使用Java 8时，首先做的就是使用lambda表达式替换匿名类，而实现Runnable接口是匿名类的最好示例。看一下Java 8之前的runnable实现方法，需要4行代码，而使用lambda表达式只需要一行代码。我们在这里做了什么呢？那就是用() -> {}代码块替代了整个匿名类。

```
// 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();
```

输出：

```
too much code, for too little to do
Lambda expression rocks !!
```

这个例子向我们展示了Java 8 lambda表达式的语法。你可以使用lambda写出如下代码：

```
(params) -> expression
(params) -> statement
(params) -> { statements }
```

例如，如果你的方法不对参数进行修改、重写，只是在控制台打印点东西的话，那么可以这样写：

```
() -> System.out.println("Hello Lambda Expressions");
```

如果你的方法接收两个参数，那么可以写成如下这样：

```
(int even, int odd) -> even + odd
```

顺便提一句，通常都会把lambda表达式内部变量的名字起得短一些。这样能使代码更简短，放在同一行。所以，在上述代码中，变量名选用a、b或者x、y会比even、odd要好。

例2、使用Java 8 lambda表达式进行事件处理

如果你用过Swing API编程，你就会记得怎样写事件监听代码。这又是一个旧版本简单匿名类的经典用例，但现在可以不这样了。你可以用lambda表达式写出更好的事件监听代码，如下所示：

```
// Java 8之前：
JButton show =  new JButton("Show");
show.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
    System.out.println("Event handling without lambda expression is boring");
    }
});
```

```
// Java 8方式：
show.addActionListener((e) -> {
    System.out.println("Light, Camera, Action !! Lambda expressions Rocks");
```

```
});
```

Java开发者经常使用匿名类的另一个地方是为 Collections.sort() 定制 Comparator。在Java 8中，你可以用更可读的lambda表达式换掉丑陋的匿名类。我把这个留做练习，应该不难，可以按照我在使用lambda表达式实现 Runnable 和 ActionListener 的过程中的套路来做。

例3、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元素，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序的，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集合现在知道怎样做迭代，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或不使用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，它可以迭代所有对象，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);
```

输出：

```
Lambdas
Default Method
Stream API
Date and Time API
```

列表循环的最后一个例子展示了如何在Java 8中使用方法引用（method reference）。你可以看到C++里面的双冒号、范围解析操作符现在在Java 8中用来表示方法引用。

例4、使用lambda表达式和函数式接口Predicate

除了在语言层面支持函数式编程风格，Java 8也添加了一个包，叫做 java.util.function。它包含了很多类，用来支持Java的函数式编程。其中一个便是Predicate，使用 java.util.function.Predicate 函数式接口以及lambda表达式，可以向API方法添加逻辑，用更少的代码支持更多的动态行为。下面是Java 8 Predicate 的例子，展示了过滤集合数据的多种常用方法。Predicate接口非常适用于做过滤。

```
public static void main(args[]){
    List languages = Arrays.asList("Java", "Scala", "C++", "Haskell", "Lisp");

    System.out.println("Languages which starts with J :");
    filter(languages, (str)->str.startsWith("J"));

    System.out.println("Languages which ends with a ");
    filter(languages, (str)->str.endsWith("a"));

    System.out.println("Print all languages :");
    filter(languages, (str)->true);

    System.out.println("Print no language : ");
    filter(languages, (str)->false);

    System.out.println("Print language whose length greater than 4:");
    filter(languages, (str)->str.length() > 4);
}

public static void filter(List names, Predicate condition) {
    for(String name: names)  {
        if(condition.test(name)) {
            System.out.println(name + " ");
        }
    }
}
```

输出：

```
Languages which starts with J :
Java
Languages which ends with a
Java
Scala
Print all languages :
Java
Scala
C++
Haskell
Lisp
Print no language :
Print language whose length greater than 4:
Scala
Haskell
```

```
// 更好的办法
public static void filter(List names, Predicate condition) {
    names.stream().filter((name) -> (condition.test(name))).forEach((name) -> {
        System.out.println(name + " ");
    });
}
```

可以看到，Stream API的过滤方法也接受一个Predicate，这意味着可以将我们定制的 filter() 方法替换成写在里面的内联代码，这就是lambda表达式的魔力。另外，Predicate接口也允许进行多重条件的测试，下个例子将要讲到。

例5、如何在lambda表达式中加入Predicate

上个例子说到，java.util.function.Predicate 允许将两个或更多的 Predicate 合成一个。它提供类似于逻辑操作符AND和OR的方法，名字叫做and()、or()和xor()，用于将传入 filter() 方法的条件合并起来。例如，要得到所有以J开始，长度为四个字母的语言，可以定义两个独立的 Predicate 示例分别表示每一个条件，然后用 Predicate.and() 方法将它们合并起来，如下所示：

```
// 甚至可以用and()、or()和xor()逻辑函数来合并Predicate，
// 例如要找到所有以J开始，长度为四个字母的名字，你可以合并两个Predicate并传入
Predicate<String> startsWithJ = (n) -> n.startsWith("J");
Predicate<String> fourLetterLong = (n) -> n.length() == 4;
names.stream()
    .filter(startsWithJ.and(fourLetterLong))
    .forEach((n) -> System.out.print("nName, which starts with 'J' and four letter long is : " + n));
```

类似地，也可以使用 or() 和 xor() 方法。本例着重介绍了如下要点：可按需要将 Predicate 作为单独条件然后将其合并起来使用。简而言之，你可以以传统Java命令方式使用 Predicate 接口，也可以充分利用lambda表达式达到事半功倍的效果。

例6、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表达式传到 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。使用流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果合并起来。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);
```

输出：

```
112.0
224.0
336.0
448.0
560.0
112.0
224.0
336.0
448.0
560.0
```

例6.2、Java 8中使用lambda表达式的Map和Reduce示例

在上个例子中，可以看到map将集合类（例如列表）元素进行转换的。还有一个 reduce() 函数可以将所有值合并成一个。Map和Reduce操作是函数式编程的核心操作，因为其功能，reduce 又被称为折叠操作。另外，reduce 并不是一个新的操作，你有可能已经在使用它。SQL中类似 sum()、avg() 或者 count() 的聚集函数，实际上就是 reduce 操作，因为它们接收多个值并返回一个值。流API定义的reduceh() 函数可以接受lambda表达式，并对所有值进行合并。IntStream这样的类有类似 average()、count()、sum() 的内建方法来做 reduce 操作，也有mapToLong()、mapToDouble() 方法来做转换。这并不会限制你，你可以用内建方法，也可以自己定义。在这个Java 8的Map Reduce示例里，我们首先对所有价格应用 12% 的VAT，然后用 reduce() 方法计算总和。

```
// 为每个订单加上12%的税
// 老方法：
List costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
double total = 0;
for (Integer cost : costBeforeTax) {
    double price = cost + .12*cost;
    total = total + price;
}
System.out.println("Total : " + total);

// 新方法：
List costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
double bill = costBeforeTax.stream().map((cost) -> cost + .12*cost).reduce((sum, cost) -> sum + cost).get();
System.out.println("Total : " + bill);
```

输出：

```
Total : 1680.0
Total : 1680.0
```

例7、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过滤大规模数据集合是惊人的简单。流提供了一个 filter() 方法，接受一个 Predicate 对象，即可以传入一个lambda表达式作为过滤逻辑。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);
```

输出：

```
Original List : [abc, , bcd, , defg, jk], filtered list : [abc, bcd, defg]
```

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8、对列表的每个元素应用函数

我们通常需要对列表的每个元素使用某个函数，例如逐一乘以某个数、除以某个数或者做其它操作。这些操作都很适合用 map() 方法，可以将转换逻辑以lambda表达式的形式放在 map() 方法里，就可以对集合的各个元素进行转换了，如下所示。

```
// 将字符串换成大写并用逗号链接起来
List<String> G7 = Arrays.asList("USA", "Japan", "France", "Germany", "Italy", "U.K.","Canada");
String G7Countries = G7.stream().map(x -> x.toUpperCase()).collect(Collectors.joining(", "));
System.out.println(G7Countries);
```

输出：

```
USA, JAPAN, FRANCE, GERMANY, ITALY, U.K., CANADA
```

例9、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]
```

例10、计算集合元素的最大值、最小值、总和以及平均值

IntStream、LongStream 和 DoubleStream 等流的类中，有个非常有用的方法叫做 summaryStatistics() 。可以返回 IntSummaryStatistics、LongSummaryStatistics 或者 DoubleSummaryStatistic s，描述流中元素的各种摘要数据。在本例中，我们用这个方法来计算列表的最大值和最小值。它也有 getSum() 和 getAverage() 方法来获得列表的所有元素的总和及平均值。

```
//获取数字的个数、最小值、最大值、总和以及平均值
List<Integer> primes = Arrays.asList(2, 3, 5, 7, 11, 13, 17, 19, 23, 29);
IntSummaryStatistics stats = primes.stream().mapToInt((x) -> x).summaryStatistics();
System.out.println("Highest prime number in List : " + stats.getMax());
System.out.println("Lowest prime number in List : " + stats.getMin());
System.out.println("Sum of all prime numbers : " + stats.getSum());
System.out.println("Average of all prime numbers : " + stats.getAverage());
```

输出：

```
Highest prime number in List : 29
Lowest prime number in List : 2
Sum of all prime numbers : 129
Average of all prime numbers : 12.9
```

Lambda表达式 vs 匿名类

既然lambda表达式即将正式取代Java代码中的匿名内部类，那么有必要对二者做一个比较分析。一个关键的不同点就是关键字 this。匿名类的 this 关键字指向匿名类，而lambda表达式的 this 关键字指向包围lambda表达式的类。另一个不同点是二者的编译方式。Java编译器将lambda表达式编译成类的私有方法。使用了Java 7的 invokedynamic 字节码指令来动态绑定这个方法。

Java 8 Lambda表达式要点

10个Java lambda表达式、流API示例

到目前为止我们看到了Java 8的10个lambda表达式，这对于新手来说是个合适的任务量，你可能需要亲自运行示例程序以便掌握。试着修改要求创建自己的例子，达到快速学习的目的。我还想建议大家使用Netbeans IDE来练习lambda表达式，它对Java 8支持良好。当把代码转换成函数式的时候，Netbeans会及时给你提示。只需跟着Netbeans的提示，就能很容易地把匿名类转换成lambda表达式。此外，如果你喜欢阅读，那么记得看一下Java 8的lambdas，实用函数式编程这本书（Java 8 Lambdas, pragmatic functional programming），作者是Richard Warburton，或者也可以看看Manning的Java 8实战（Java 8 in Action），这本书虽然还没出版，但我猜线上有第一章的免费pdf。不过，在你开始忙其它事情之前，先回顾一下Java 8的lambda表达式、默认方法和函数式接口的重点知识。

1）lambda表达式仅能放入如下代码：预定义使用了 @Functional 注释的函数式接口，自带一个抽象函数的方法，或者SAM（Single Abstract Method 单个抽象方法）类型。这些称为lambda表达式的目标类型，可以用作返回类型，或lambda目标代码的参数。例如，若一个方法接收Runnable、Comparable或者 Callable 接口，都有单个抽象方法，可以传入lambda表达式。类似的，如果一个方法接受声明于 java.util.function 包内的接口，例如 Predicate、Function、Consumer 或 Supplier，那么可以向其传lambda表达式。

2）lambda表达式内可以使用方法引用，仅当该方法不修改lambda表达式提供的参数。本例中的lambda表达式可以换为方法引用，因为这仅是一个参数相同的简单方法调用。

```
1 list.forEach(n -> System.out.println(n));
2 list.forEach(System.out::println);  // 使用方法引用
```

然而，若对参数有任何修改，则不能使用方法引用，而需键入完整地lambda表达式，如下所示：

```
1 list.forEach((String s) -> System.out.println("*" + s + "*"));
```

事实上，可以省略这里的lambda参数的类型声明，编译器可以从列表的类属性推测出来。

3）lambda内部可以使用静态、非静态和局部变量，这称为lambda内的变量捕获。

4）Lambda表达式在Java中又称为闭包或匿名函数，所以如果有同事把它叫闭包的时候，不用惊讶。

5）Lambda方法在编译器内部被翻译成私有方法，并派发 invokedynamic 字节码指令来进行调用。可以使用JDK中的 javap 工具来反编译class文件。使用 javap -p 或 javap -c -v 命令来看一看lambda表达式生成的字节码。大致应该长这样：

```
private static java.lang.Object lambda$0(java.lang.String);
```

6）lambda表达式有个限制，那就是只能引用 final 或 final 局部变量，这就是说不能在lambda内部修改定义在域外的变量。

```
List<Integer> primes = Arrays.asList(new Integer[]{2, 3,5,7});
int factor = 2;
primes.forEach(element -> { factor++; });
```

```
Compile time error : "local variables referenced from a lambda expression must be final or effectively final"
```

另外，只是访问它而不作修改是可以的，如下所示：

```
List<Integer> primes = Arrays.asList(new Integer[]{2, 3,5,7});
int factor = 2;
primes.forEach(element -> { System.out.println(factor*element); });
```

输出：

```
4
6
10
14
```

因此，它看起来更像不可变闭包，类似于Python。

以上就是Java 8的lambda表达式的全部10个例子。此次修改将成为Java史上最大的一次，将深远影响未来Java开发者使用集合框架的方式。我想规模最相似的一次修改就是Java 5的发布了，它带来了很多优点，提升了代码质量，例如：泛型、枚举、自动装箱（Autoboxing）、静态导入、并发API和变量参数。上述特性使得Java代码更加清晰，我想lambda表达式也将进一步改进它。我在期待着开发并行第三方库，这可以使高性能应用变得更容易写。

更多阅读：http://javarevisited.blogspot.com/2014/02/10-example-of-lambda-expressions-in-java8.html#ixzz3gCMp6Vhc

原文链接： javarevisited 翻译： ImportNew.com - lemeilleur
译文链接： http://www.importnew.com/16436.html
[**转载请保留原文出处、译者和译文链接。**]

分类： JAVA

anakinf
关注 - 5
粉丝 - 9

0 0

+加关注

« 上一篇：我和阿里云RDS的故事

» 下一篇：Java多线程学习（吐血超详细总结）

posted @ 2017-10-27 13:37 anakinf 阅读(2894) 评论(0) 编辑 收藏