

## 深入浅出 Java 8 Lambda 表达式 (http://blog.oneapm.com/apm-tech/226.html)

OneAPM (http://blog.oneapm.com/author-16.html) 2015-11-01

59833 7

**摘要：**此篇文章主要介绍 Java (http://blog.oneapm.com/tags-Java.html)8 Lambda 表达式产生的背景和用法，以及 Lambda 表达式与匿名类的不同等。本文系 OneAPM (http://www.oneapm.com/index.html?utm\_source=Community&utm\_medium=Article&utm\_term=lambda&utm\_campaign=NovArti&from=matefinola) 工程师编译整理。

Java (http://www.oneapm.com/ai/java.html?utm\_source=Community&utm\_medium=Article&utm\_term=lambda&utm\_campaign=NovArti&from=matefinola)

是一流的面向对象语言，除了部分简单数据类型，Java 中的一切都是对象，即使数组也是一种对象，每个类创建的实例也是对象。在 Java 中定义的函数或方法不可能完全独立，也不能将方法作为参数或返回一个方法给实例。

从 Swing 开始，我们总是通过匿名类给方法传递函数功能，以下是旧版的事件监听代码：

```
someObject.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {

        //Event listener implementation goes here...

    }
});
```

在上面的例子里，为了给 Mouse 监听器添加自定义代码，我们定义了一个匿名内部类 MouseAdapter 并创建了它的对象，通过这种方式，我们将一些函数功能传给 addMouseListener 方法。

邮件

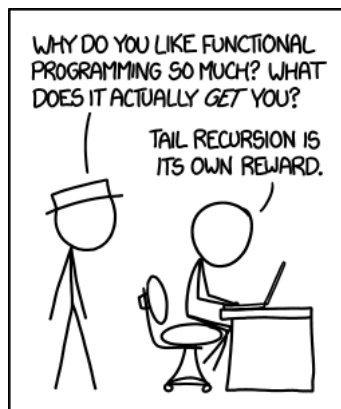
请输入邮箱

简而言之，在 Java 里将普通的方法或函数像参数一样传值并不简单，为此，Java 8 增加了一个语言级的新特性，名为 **Lambda 表达式**。

提交

## 为什么 Java 需要 Lambda 表达式？

如果忽视注解(Annotations)、泛型(Generics)等特性，自 Java 语言诞生时起，它的变化并不大。Java 一直都致力维护其对象至上的特征，在使用过 JavaScript 之类的函数式语言之后，Java 如何强调其面向对象的本质，以及源码层的数据类型如何严格变得更加清晰可感。其实，函数对 Java 而言并不重要，在 Java 的世界里，函数无法独立存在。



在函数式编程语言中，函数是一等公民，它们可以独立存在，你可以将其赋值给一个变量，或将它们当做参数传给其他函数。JavaScript 是最典型的函数式编程语言。点击此处 (http://eloquentjavascript.net/chapter6.html)以及此处 (http://www.ibm.com/developerworks/library/wa-javascript/index.html)可以清楚了解 JavaScript 这种函数式语言的好处。函数式语言提供了一种强大的功能——闭包，相比于传统的编程方法有很多优势，闭包是一个可调用的对象，它记录了一些信息，这些信息来自于创建它的作用域。Java 现在提供的最接近闭包的概念便是 Lambda 表达式，虽然闭包与 Lambda 表达式之间存在显著差别，但至少 Lambda 表达式是闭包很好的替代者。

在 Steve Yegge 辛辣又幽默的博客文章 (http://steve-yegge.blogspot.com/2006/03/execution-in-kingdom-of-nouns.html)里，描绘了 Java 世界是如何严格地以名词为中心的，如果你还没看过，赶紧去读吧，写得非常风趣幽默，而且恰如其分地解释了为什么 Java 要引进 Lambda 表达式。

Lambda 表达式为 Java 添加了缺失的函数式编程特点, 使我们能将函数当做一等公民看待。尽管不完全正确, 我们很快就会见识到 Lambda 与闭包的不同之处, 但是又无限地接近闭包。在支持一类函数的语言中, Lambda 表达式的类型将是函数。但是, 在 Java 中, Lambda 表达式是对象, 他们必须依附于一类特别的对象类型——函数式接口(functional interface)。我们会在后文详细介绍函数式接口。

Mario Fusco 的这篇思路清晰的文章 (<http://java.dzone.com/articles/why-we-need-Lambda-expressions>)介绍了为什么 Java 需要 Lambda 表达式。他解释了为什么现代编程语言必须包含闭包这类特性。

## Lambda 表达式简介

Lambda 表达式是一种匿名函数(对 Java 而言这并不完全正确, 但现在姑且这么认为), 简单地说, 它是没有声明的方法, 也即没有访问修饰符、返回值声明和名字。

你可以将其想做一种速记, 在你需要使用某个方法的地方写上它。当某个方法只使用一次, 而且定义很简短, 使用这种速记替代之尤其有效, 这样, 你就不必在类中费力写声明与方法了。



Java 中的 Lambda 表达式通常使用 (argument) -> (body) 语法书写, 例如:

```
(arg1, arg2...) -> { body }  
  
(type1 arg1, type2 arg2...) -> { body }
```

以下是一些 Lambda 表达式的例子:

```
(int a, int b) -> { return a + b; }  
  
() -> System.out.println("Hello World");  
  
(String s) -> { System.out.println(s); }  
  
() -> 42  
  
() -> { return 3.1415; }
```

### 邮件

## Lambda 表达式的结构

让我们了解一下 Lambda 表达式的结构。

- 一个 Lambda 表达式可以有零个或多个参数
- 参数的类型既可以明确声明, 也可以根据上下文来推断。例如: (int a) 与 (a) 效果相同
- 所有参数需包含在圆括号内, 参数之间用逗号相隔。例如: (a, b) 或 (int a, int b) 或 (String a, int b, float c)
- 空圆括号代表参数集为空。例如: () -> 42
- 当只有一个参数, 且其类型可推导时, 圆括号 () 可省略。例如: a -> return a\*a
- Lambda 表达式的主体可包含零条或多条语句
- 如果 Lambda 表达式的主体只有一条语句, 花括号{}可省略。匿名函数的返回类型与该主体表达式一致
- 如果 Lambda 表达式的主体包含一条以上语句, 则表达式必须包含在花括号{}中 (形成代码块)。匿名函数的返回类型与代码块的返回类型一致, 若没有返回则为空

## 什么是函数式接口

在 Java 中, Marker (标记) 类型的接口是一种没有方法或属性声明的接口, 简单地说, marker 接口是空接口。相似地, 函数式接口是只包含一个抽象方法声明的接口。

java.lang.Runnable 就是一种函数式接口, 在 Runnable 接口中只声明了一个方法 void run(), 相似地, ActionListener 接口也是一种函数式接口, 我们使用匿名内部类来实例化函数式接口的对象, 有了 Lambda 表达式, 这一方式可以得到简化。

每个 Lambda 表达式都能隐式地赋值给函数式接口, 例如, 我们可以通过 Lambda 表达式创建 Runnable 接口的引用。

```
Runnable r = () -> System.out.println("hello world");
```

当不指明函数式接口时, 编译器会自动解释这种转化:

```
new Thread(  
    () -> System.out.println("hello world")  
).start();
```

因此，在上面的代码中，编译器会自动推断：根据线程类的构造函数签名 `public Thread(Runnable r) { }`，将该 Lambda 表达式赋给 `Runnable` 接口。

以下是一些 Lambda 表达式及其函数式接口：

```
Consumer<Integer> c = (int x) -> { System.out.println(x) };  
  
BiConsumer<Integer, String> b = (Integer x, String y) -> System.out.println(x + " : " + y);  
  
Predicate<String> p = (String s) -> { s == null };
```

`@FunctionalInterface` (<http://download.java.net/jdk8/docs/api/java/lang/FunctionalInterface.html>) 是 Java 8 新加入的一种接口，用于指明该接口类型声明是根据 Java 语言规范定义的函数式接口。Java 8 还声明了一些 Lambda 表达式可以使用的函数式接口，当你注释的接口不是有效的函数式接口时，可以使用 `@FunctionalInterface` 解决编译层面的错误。

以下是一种自定义的函数式接口：`@FunctionalInterface public interface WorkerInterface {`

```
    public void doSomeWork();  
}
```

根据定义，函数式接口只能有一个抽象方法，如果你尝试添加第二个抽象方法，将抛出编译时错误。例如：

```
@FunctionalInterface  
public interface WorkerInterface {  
  
    public void doSomeWork();  
  
    public void doSomeMoreWork();  
}
```

#### 邮件

错误：

```
Unexpected @FunctionalInterface annotation  
@FunctionalInterface ^ WorkerInterface is not a functional interface multiple  
non-overriding abstract methods found in interface WorkerInterface 1 error
```

函数式接口定义好后，我们可以在 API 中使用它，同时利用 Lambda 表达式。例如：

```
//定义一个函数式接口  
@FunctionalInterface  
public interface WorkerInterface {  
  
    public void doSomeWork();  
}  
  
public class WorkerInterfaceTest {  
  
    public static void execute(WorkerInterface worker) {  
        worker.doSomeWork();  
    }  
  
    public static void main(String [] args) {  
  
        //invoke doSomeWork using Anonymous class  
        execute(new WorkerInterface() {  
            @Override  
            public void doSomeWork() {  
                System.out.println("Worker invoked using Anonymous class");  
            }  
        });  
  
        //invoke doSomeWork using Lambda express (http://blog.oneapm.com/tags-express.html)  
        execute( () -> System.out.println("Worker invoked using Lambda expression") );  
    }  
}
```

输出：

```
Worker invoked using Anonymous class  
Worker invoked using Lambda expression
```

这上面的例子里，我们创建了自定义的函数式接口并与 Lambda 表达式一起使用。execute() 方法现在可以将 Lambda 表达式作为参数。

## Lambda 表达式举例

学习 Lambda 表达式的最好方式是学习例子。

线程可以通过以下方法初始化：

```
//旧方法:
new Thread(new Runnable() {
@Override
public void run() {
    System.out.println("Hello from thread");
}
}).start();

//新方法:
new Thread(
() -> System.out.println("Hello from thread")
).start();
```

事件处理可以使用 Java 8 的 Lambda 表达式解决。下面的代码中，我们将使用新旧两种方式向一个 UI 组件添加 ActionListener：

```
//Old way:
button.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
    System.out.println("The button was clicked using old fashion code!");
}
});

//New way:
button.addActionListener( (e) -> {
    System.out.println("The button was clicked. From Lambda expressions !");
});
```

以下代码的作用是打印出给定数组中的所有元素。注意，使用 Lambda 表达式的方法不止一种。在下面的例子中，我们先用常用的箭头语法创建 Lambda 表达式，之后，使用 Java 8 全新的双冒号(::)操作符将一个常规方法转化为 Lambda 表达式：

```
//Old way:
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
for(Integer n: list) {
    System.out.println(n);
}

//New way:
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
list.forEach(n -> System.out.println(n));

//or we can use :: double colon operator in Java 8
list.forEach(System.out::println);
```

邮件

请输入邮箱

提交

在下面的例子中，我们使用断言(Predicate)函数式接口创建一个测试，并打印所有通过测试的元素，这样，你就可以使用 Lambda 表达式规定一些逻辑，并以此为基础有所作为：

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;

public class Main {

    public static void main(String [] a) {

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);

        System.out.println("Print all numbers:");
        evaluate(list, (n)->true);

        System.out.println("Print no numbers:");
        evaluate(list, (n)->false);

        System.out.println("Print even numbers:");
        evaluate(list, (n)-> n%2 == 0 );

        System.out.println("Print odd numbers:");
        evaluate(list, (n)-> n%2 == 1 );

        System.out.println("Print numbers greater than 5:");
        evaluate(list, (n)-> n > 5 );

    }

    public static void evaluate(List<Integer> list, Predicate<Integer> predicate) {
        for(Integer n: list) {
            if(predicate.test(n)) {
                System.out.println(n + " ");
            }
        }
    }
}
```

输出:

```
Print all numbers: 1 2 3 4 5 6 7
Print no numbers:
Print even numbers: 2 4 6
Print odd numbers: 1 3 5 7
Print numbers greater than 5: 6 7
```

邮件

下面的例子使用 Lambda 表达式打印数值中每个元素的平方，注意我们使用了 `.stream()` 方法将常规数组转化为流。Java 8 增加了一些超棒的流 APIs。 `java.util.stream.Stream` (<http://download.java.net/jdk8/docs/api/java/util/stream/Stream.html>) 接口包含许多有用的方法，能结合 Lambda 表达式产生神奇的效果。我们将 Lambda 表达式 `x -> x*x` 传给 `map()` 方法，该方法会作用于流中的所有元素。之后，我们使用 `forEach` 方法打印数据中的所有元素：

```
//Old way:
List<Integer> list = Arrays.asList(1,2,3,4,5,6,7);
for(Integer n : list) {
    int x = n * n;
    System.out.println(x);
}

//New way:
List<Integer> list = Arrays.asList(1,2,3,4,5,6,7);
list.stream().map((x) -> x*x).forEach(System.out::println);
```

下面的例子会计算给定数值中每个元素平方后的总和。请注意，Lambda 表达式只用一条语句就能达到此功能，这也是 MapReduce 的一个初级例子。我们使用 `map()` 给每个元素求平方，再使用 `reduce()` 将所有元素计入一个数值：

```
//Old way:
List<Integer> list = Arrays.asList(1,2,3,4,5,6,7);
int sum = 0;
for(Integer n : list) {
    int x = n * n;
    sum = sum + x;
}
System.out.println(sum);

//New way:
List<Integer> list = Arrays.asList(1,2,3,4,5,6,7);
int sum = list.stream().map(x -> x*x).reduce((x,y) -> x + y).get();
System.out.println(sum);
```

## Lambda 表达式与匿名类的区别

使用匿名类与 Lambda 表达式的一大区别在于关键词的使用。对于匿名类，关键词 `this` 解读为匿名类，而对于 Lambda 表达式，关键词 `this` 解读为写就 Lambda 的外部类。

Lambda 表达式与匿名类的另一不同在于两者的编译方法。Java 编译器编译 Lambda 表达式并将他们转化为类里面的私有函数，它使用 Java 7 中新加的 `invokedynamic` 指令动态绑定该方法，关于 Java 如何将 Lambda 表达式编译为字节码，Tal Weiss 写了一篇很好的文章 (<http://www.takipiblog.com/2014/01/16/compiling-Lambda-expressions-scala-vs-java-8/>)。

## 到此为止啦，亲们！

Mark Reinhold，甲骨文的首席架构师，将 Lambda 表达式描述 ([https://blogs.oracle.com/javaone/entry/the\\_javaone\\_2013\\_technical\\_keynote](https://blogs.oracle.com/javaone/entry/the_javaone_2013_technical_keynote))为该编程模型最大的提升——比泛型(generics)还强大。事实的确如此，Lambda 表达式赋予了 Java 程序员 (<http://blog.oneapm.com/tags-programmer.html>)相较于其他函数式编程语言缺失的特性，结合虚拟扩展方法之类的特性，Lambda 表达式能写出一些极好的代码。

希望这篇文章能让您对 Java 8 ([http://www.oneapm.com/ai/java.html?](http://www.oneapm.com/ai/java.html?utm_source=Community&utm_medium=Article&utm_term=lambda&utm_campaign=NovArti&from=matefinola)

[utm\\_source=Community&utm\\_medium=Article&utm\\_term=lambda&utm\\_campaign=NovArti&from=matefinola](http://www.oneapm.com/ai/java.html?utm_source=Community&utm_medium=Article&utm_term=lambda&utm_campaign=NovArti&from=matefinola)) 的新特性所有了解。

原文地址：<http://viralpatel.net/blogs/Lambda-expressions-java-tutorial/> (<http://viralpatel.net/blogs/Lambda-expressions-java-tutorial/>)

**OneAPM for Java** ([http://www.oneapm.com/ai/java.html?](http://www.oneapm.com/ai/java.html?utm_source=Community&utm_medium=Article&utm_term=lambda&utm_campaign=NovArti&from=matefinola)

[utm\\_source=Community&utm\\_medium=Article&utm\\_term=lambda&utm\\_campaign=NovArti&from=matefinola](http://www.oneapm.com/ai/java.html?utm_source=Community&utm_medium=Article&utm_term=lambda&utm_campaign=NovArti&from=matefinola)) 能够深入到所有 Java 应用内部完成应用性能管理 (<http://www.oneapm.com/>)和监控，包括代码级别性能问题的可见性、性能瓶颈的快速识别与追溯、真实用户体验监控 (<http://www.oneapm.com/bi/feature.html>)、服务器监控 (<http://www.oneapm.com/ci/feature.html>)和端到端的应用性能管理 (<http://www.oneapm.com/>)。想阅读更多技术文章，请访问 OneAPM 官方博客 (<http://news.oneapm.com/>)

[utm\\_source=Community&utm\\_medium=Article&utm\\_term=lambda&utm\\_campaign=NovArti&from=matefinola](http://www.oneapm.com/ai/java.html?utm_source=Community&utm_medium=Article&utm_term=lambda&utm_campaign=NovArti&from=matefinola))。想阅读更多技术文章，请访问 OneAPM 官方博客 (<http://news.oneapm.com/>)

[utm\\_source=Community&utm\\_medium=Article&utm\\_term=Spring&utm\\_campaign=OctArti&from=matefiocsp](http://www.oneapm.com/ai/java.html?utm_source=Community&utm_medium=Article&utm_term=Spring&utm_campaign=OctArti&from=matefiocsp))。

本站内容未经允许不得转载

« 上一篇 (<http://blog.oneapm.com/apm-tech/234.html>)

下一篇 » (<http://blog.oneapm.com/apm-tech/232.html>)

邮件

请输入邮箱

提交

订阅

已有 7 条评论



zill  
2017-09-13 11:47:43

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urldirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urldirect.php?url=Tm9iaXJk))  
很棒，谢谢博主的分享。



jack  
2017-11-26 15:54:59

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urldirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urldirect.php?url=Tm9iaXJk))  
Lambda简化了，但同时相应的可读性也减少了



xiao  
2018-01-19 21:14:09

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urldirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urldirect.php?url=Tm9iaXJk))  
很赞



yuki  
2018-02-27 15:35:11

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urldirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urldirect.php?url=Tm9iaXJk))  
说的很清楚，没怎么学过java的终于明白了



Road  
2018-02-28 15:10:53

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urldirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urldirect.php?url=Tm9iaXJk))  
学习了，很棒，点赞。



Road

2018-02-28 15:11:33

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urlredirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urlredirect.php?url=Tm9iaXJk))

了解了Lambda相关用法，很受教。



哈哈

2018-03-26 14:48:40

回复

([http://blog.oneapm.com/zb\\_users/plugin/Nobird\\_Seo\\_Tools/NoLinks/urlredirect.php?url=Tm9iaXJk](http://blog.oneapm.com/zb_users/plugin/Nobird_Seo_Tools/NoLinks/urlredirect.php?url=Tm9iaXJk))

最好的学习方式就是例子，生动，点赞！

发表评论:



名称  必填

邮箱

内容

邮件

提交

提交

## 热门话题

[Cloud Insight \(http://blog.oneapm.com/tags-Cloud+Insight.html\)](http://blog.oneapm.com/tags-Cloud+Insight.html)[OneRASP \(http://blog.oneapm.com/tags-OneRASP.html\)](http://blog.oneapm.com/tags-OneRASP.html)[Mobile Insight \(http://blog.oneapm.com/tags-Mobile+Insight.html\)](http://blog.oneapm.com/tags-Mobile+Insight.html)[OneAlert \(http://blog.oneapm.com/tags-OneAlert.html\)](http://blog.oneapm.com/tags-OneAlert.html)[Application Insight \(http://blog.oneapm.com/tags-Application+Insight.html\)](http://blog.oneapm.com/tags-Application+Insight.html)[DevOps \(http://blog.oneapm.com/tags-DevOps.html\)](http://blog.oneapm.com/tags-DevOps.html)[Java \(http://blog.oneapm.com/tags-Java.html\)](http://blog.oneapm.com/tags-Java.html)[RASP \(http://blog.oneapm.com/tags-RASP.html\)](http://blog.oneapm.com/tags-RASP.html)[Browser Insight \(http://blog.oneapm.com/tags-Browser+Insight.html\)](http://blog.oneapm.com/tags-Browser+Insight.html)[Docker \(http://blog.oneapm.com/tags-Docker.html\)](http://blog.oneapm.com/tags-Docker.html)[iOS \(http://blog.oneapm.com/tags-iOS.html\)](http://blog.oneapm.com/tags-iOS.html)

## 最热文章

是谁拖了网站访问速度的「后腿」？ (<http://blog.oneapm.com/apm-tech/240.html>)

多彩投牵手OneAPM：「生活不只是苟且，还有诗和远方」 (<http://blog.oneapm.com/casestudy/261.html>)

深入浅出 Java 8 Lambda 表达式 (<http://blog.oneapm.com/apm-tech/226.html>)

作为一个安静的程序员，我们应该如何去创业呢？ (<http://blog.oneapm.com/apm/669.html>)