**前言**

网络上很多教程都是在讲解JDK8有哪些新特性，如何使用这些新特性，而忽略了Java为什么要引入这些新特性。旧的方式不能满足需求吗，旧的特性有什么问题吗。我们今天从编程范式或者说编程语言学的角度谈谈JDK8的新特性。

在计算机刚开始出现的蛮荒时代的近三十年中，人们通过对汇编语言、B语言等一系列初级语言的总结，最终C语言这个面向过程的语言横空出世。同时，人们在使用面向过程语言的过程中发现，面向过程语言的可维护性很差，当时的程序设计领域正面临着一种危机：在软硬件环境逐渐复杂的情况下，软件如何得到良好的维护？面向对象程序设计在某种程度上通过强调可重复性解决了这一问题。人类的世界，相对于抽象的过程，人类更擅长将真实世界的对象映射到抽象的对象，即：“建模”。

思维方式的转变带了的极大的便利，人们交流起来更加方便，软件工程学得到了极大的发展，设计模式，建模语言（UML），OOD这些基于面向对象的工程方法学出现，软件开发不再跟硬件和过程强绑定，不再晦涩难懂门槛高企，更多的人加入到软件开发当中来，普通公司和个人也可以快速组织足够的人手开发出满足自己期望的软件。

然而，人们在使用面向对象的过程中，发现虽然面向对象具有极大的便利，但却不是尽善尽美，Java语言的发展便是如此，Java开始的设计是一个纯粹的面向对象语言，后面却因为这种面向对象的非尽善尽美作出了各种妥协和调整。下面我们将循着这个脉络讲一下这些调整对Java语言的影响。

**什么是编程范式**

托马斯库恩提出“科学革命的结构”一书中第一次提出范式的概念和理论，罗伯特弗洛伊德*1979*年图灵奖的颁奖演说中首次在计算机领域使用了编程范式一词。范即模范、典范之意，范型即模式、方法，是一类典型的编程风格，是指从事软件工程的一类典型的风格。

我们举个例子，便于理解什么是编程范式。在写文章时候，我们既可以用白话文，也可以用文言文，甚至在表达一些特殊事物或者心情的时候还可以用诗词歌赋，而我们所选的表达方式就是相当于“编程范式”。这些语言或者说表达方式都是由甲骨文、金文（相当于汇编或者B语言）发展而来。然而不同的表达方式却有不同的效果或者说应用场景。

在程序世界中，编程范式有很多，如最知名的“面向过程编程”和“面向对象编程”，Linux中shell的“面向管道编程”，前端开发常用的“事件驱动编程”思想，还有时下最火的“函数式编程”。当然，当前目前很多语言，包括Java在内都已不再是单一编程范式，而是混合编程模式，即一种编程语言杂糅了多种编程范式的思想，Java语言的发展过程正体现了这些编程思想去芜存菁的过程。了解这个过程使我们能更深刻的理解Java这门编程语言，同时为我们从更高层面学习其他语言提供借鉴。

Java语言被设计之初的想法是比较纯粹的面向对象编程，后面加入了各种编程范式，如：“泛型”、“反射”、“注解”以及“函数式编程”。而面向对象是整个Java的基础，我们来重新审视一下究竟面向对象便利在哪里。“函数式编程”是JDK8引入的新特性，如果要了解JDK8为什么要引入这个编程范式，那我们就得先去了解什么是“函数式”编程。下面我们会先介绍这两个基本的概念。

**面向对象编程**

很多人在开始学Java编程的时候都会接触到一个例子：开车去某地。例子说：面向过程编程就是你要先挂挡，再踩离合，再加油门，然后开始开车，然后经过某个过程，到达某地；面向对象就是你叫一辆车去，然后说去某地，然后就把你送到了；然后就说面向对象不需要知道细节等等等等。这个例子可以说误导了很多人，导致很多Java初学者反而越学越迷惑。

其实，“面向对象”的核心在于对象两个字，面向对象具有三大特性：封装、继承和多态。然后这里还默认省略了一个特性，就是：对象。我们真的不需要了解细节吗，实际不是，如果我们不知道怎么开车，无论如何都无法完成开车去某地这件事的。“面向对象”开发第一件事就是抽象。开车去某地这件事最基本的抽象我们可以抽象出两个对象：车和人，然后将开车的所有操作封装成对对象的操作，即对象的中操作方法。同样，如果有对人的操作那也是将操作封装到“人”这个对象当中。核心由操作过程变成了对象，这才是“面向对象”的核心。而就这种简单的思想转变带来了开发模式的巨大转变。

基于这种思想，诞生了面向对象程序设计（OOD），诞生了统一建模语言（UML）、诞生了基于对象的需求用例分析，总结出了面向对象设计模式，这些思想构建了几乎整个现代软件开发的基石。可以说自从有了面向对象，软件设计开发的面貌就焕然一新了。

**函数式编程**

“函数式编程”又称函数程序设计、泛函编程，它将电脑运算视为函数运算，并且避免使用程序状态以及易变对象。λ演算为该语言最重要的基础。相比“面向过程编程”，“函数式编程”更加强调程序执行的结果而非执行的过程。在“函数式编程”中，函数是第一类“对象”，即一个函数可以作为其它函数的参数，也可以从函数中返回，同时可以被修改或者被分配给一个变量。

举例说明一下，假设我们有四个函数add（加）、sub（减）、mul（乘）、div（除），有个算式(a + b - c\*d)/e。如果是面向过程，可能是这样的：

1. f1 = c\*d
2. f2 = a + b
3. f3 = f2 - f1
4. f4 = f3/e

而函数式编程是这样：

1. f5 = div(sub(add(a + b),mul(c, d)),e)

可以看出，在“函数式编程”，函数是“函数式编程”最重要的概念，一般情况下会要求我们将所有操作过程封装成函数，然后函数可以作为普通的入参和出参使用，这是面向函数最重要的思想。

当然，在与面向对象结合的过程中，我们看到的是对象，无法再直观看到具体的函数或者方法。那么这个过程就可能变成：

1. Object o2 = o1.mul(c, d).reverse().add(a).add(b).div(e)

像不像JDK8中的流式编程，下面章节我们将做详细的介绍。

**Java编程范式的演变**

对Java语言来说，JDK自诞生之后最重要的版本有两个，一个是JDK5.0，一个是JDK8。为什么说重要，主要是这两个版本带来了编程方式和思想的极大变化，其他版本并非不重要，而是因为它们更多是JDK功能的增加和JVM性能的改善，对于语言思想或者说编程范式来说，并没有太大改变。首先我们来看一下JDK5.0。

JDK5.0新增加的支持的功能有：

* 泛型（Generics）
* 元数据（Metadata）
* 自动封装与解封装
* 枚举
* 可变参数
* 增强For Each
* 静态导入

JDK5.0除了极大简化了Java开发，还带来了两个最重要的功能，泛型和元数据。这两者对Java编程习惯的改变非常之大，甚至很多设计模式都因此而修改，同时诞生了许多新的设计模式和思想。泛型的出现使得参数类型化，使得类或接口更加通用。比如JDK中的集合类，通过泛型实现了对各种类型的兼容，成为极其通用的类库。很多框架设计都用上了泛型的思想。另一个极易被忽略的就是元数据，即注解。注解的出现配合上反射，为Java开发提供了新的开发思路，通过Spring我们可以看见注解带来的影响，不熟悉Spring的人甚至可能认为Spring的配置文件开发方式和Spring的注解开发是两种框架，而这种改变包括Servlet，就是从注解出现以后才得以实现的。

JDK8的主要新特性有：

* Lambda表达式
* 流式编程
* 嵌入JavaScript代码
* Joda-Time日期时间处理库
* 静态链接 JNI 程序库
* 执行JavaFX应用程序
* 移除永久代

其中，Lambda表达式和流式编程都是函数式编程的体现，而Lambda表达式是流式编程的基础。Java社区中曾经就是否要加入 lambda 表达式举行过辩论。最终Sun公司宣布 lambda 表达式将会包含在Java中，并请提交给社区（JSR 335）。支持lambda表达式使得流中元素的函数式操作成为可能，因此也使得实现由 MapReduce 启发的函数式集合操作成为可能。JDK8相当于在原来面向对象基础上又增加了一种类型变量：函数。接口传参出参都可以使用函数变量。下一章节我们将详细介绍一下，JDK8新特性，Lambda表达式和流编程的具体使用。

**JDK8的新特性**

函数式编程究竟给Java语言带来了哪些变化，我们通过一些例子来看一下。首先，我们来看一下Lambda表达式，比如多线程编程，我们现在创建一个线程去遍历一个Map，如果完全按照面向对象的操作，那么应该是：

1. **class** SetThread **implements** Runnable {
2. **private** ConcurrentHashMap<String, String> chp;
3. **public** SetThread(ConcurrentHashMap<String, String> chp) {
4. **this**.chp = chp;
5. }
6. **public** **void** run() {
7. **for** (Entry<String, String> entry : chp.entrySet()) {
8. System.out.println(entry.getKey() + ":" + entry.getValue());
9. }
10. }
11. }
13. **public** **class** App {
14. **public** **static** **void** main(String[] args) **throws** InterruptedException {
15. ConcurrentHashMap<String, String> concurrentHashMap = **new** ConcurrentHashMap<String, String>();
16. SetThread setThread = **new** SetThread(concurrentHashMap);
17. Thread t = **new** Thread(setThread);
18. t.start();
19. Thread.sleep(10000);
20. }
21. }

即使我们使用了匿名类的写法也仍然糅杂：

1. **public** **class** App {
2. **public** **static** **void** main(String[] args) **throws** InterruptedException {
3. **final** ConcurrentHashMap<String, String> concurrentHashMap = **new** ConcurrentHashMap<String, String>();
4. **new** Thread(**new** Runnable() {
5. **public** **void** run() {
6. // TODO Auto-generated method stub
7. **for** (Entry<String, String> entry : concurrentHashMap.entrySet()) {
8. System.out.println(entry.getKey() + ":" + entry.getValue());
9. }
10. }
11. }).start();
12. Thread.sleep(10000);
13. }
14. }

而当我们使用了Lambda表达式就会变成：

1. **public** **class** App {
2. **public** **static** **void** main(String[] args) **throws** InterruptedException {
3. **final** ConcurrentHashMap<String, String> concurrentHashMap = **new** ConcurrentHashMap<String, String>();
4. **new** Thread(() -> {
5. concurrentHashMap.forEach((k, v) -> System.out.println(k + ":" + v));
6. }).start();
7. Thread.sleep(10000);
8. }
9. }

从中我们可以看到起作用的代码由解决二十行缩减到只剩下三行，而且非常清晰简洁易于理解，有没有一种感觉，我们写了一幅长篇文章来描写秦岭巍峨高耸，甚至连“卧槽”都用上了，结果李白来了一句：“连峰去天不盈尺”，完美。这也就是为什么Lambda表达式或者说函数式编程严重违反面向对象编程理念，但JCP最终仍然决定将其加入到Java的原因。

基于Lambda表达式，JDK实现了流式编程，这是函数式编程思想的重要体现。流式编程有两个好处，第一，编程效率高，简洁性好，程序可读性好；第二，基于函数式编程，即基于Java线程栈的操作，较少存在并发问题，很方便地写出高性能的并发程序。

流式编程主要作用于数据流，相关实现类位于java.util.stream包中，Collection集合接口默认添加了方法default stream 用来获取流，所以其所有实现类均可获取流。Stream按照操作结果可分为两种，中间操作和结束操作：

|  |  |
| --- | --- |
| 中间操作 | contact(), distinct(), filter(), flatMap(), limit(), map(), parallelStream(), peek(), skip(), sorted(), parallel(), sequential(), unsorted() |
| 结束操作 | allMatch(), anyMatch(), collect(), count(), findAny(), findFirst(), forEach(), forEachOrdered(), max(),min(), noneMatch(), reduce(), toArray() |

中间操返回结果为Stream<T>泛型类型，可以继续进行流式操作，结束操作即返回一个值，从而结束流无法继续迭代。如：

1. IntStream.range(1, upTo).parallel().filter(**this**::isPrime).count();

另外，Stream封装了对流的Fork/Join线程池处理，使用过Fork/Join的同学明白，其写起来来有多麻烦。在JDK8中，我们无需再关心Fork/Join的过程，parallelStream()会自动帮我们做分片处理，一行即可解决原来几十行才能解决的问题。

1. Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9).parallelStream().forEach((i) -> System.out.println(i));

总体来说，JDK8这次加入函数式编程之后，虽然一定程度“牺牲”些面向对象的思想，但代码的可读性得到了极大的加强。更重要的是，原来在编程时候只需要考虑“对象”即可，而现在应该调整为“对象”，“函数”，“流”等并存的局面。

**Java语言的未来**

JDK在8以后最重要的一个议题就是模块化，JCP在osgi和Jigsaw之间也犹豫了一段时间，目前选定使用Jigsaw作为JDK的模块化方法。JDK 9、10、11一直在使用Jigsaw围绕着JDK做模块化调整的工作，针对JDK模块化以及模块化后对Java开发的影响，我们后续文章会陆续谈到。