《Java8 实战》笔记

-- Dawei Min

1 为什么要关心 Java 8

1.1 Java 怎么还在变

1.1.2 流处理

尽管流水线实际上是一个序列，但不同加工站的运行一般是并行的。

可以在一个更高的抽象层次上写 Java 8 程序了：思路变成了把这样的流变成了那样的流（就像写数据库查询语句时的那种思路），而不是一次只处理一个项目。

Java 8 可以透明地把输入的不相关部分拿到几个 CPU 内核上分别执行你的 Stream 操作流水线——这是几乎免费的并行，用不着费劲搞 Thread 了。

1.1.3 用行为参数化把代码传递给方法

Java 8 增加了把方法（你的代码）作为参数传递给另一个方法的能力，把这一概念称为行为参数化。

1.1.4 并行与共享的可变数据

没有共享的可变数据，将方法和函数即代码传递给其他方法的能力是我们平常所说的函数式编程范式的基石。

1.2 Java 中的函数

1.2.1 方法和 Lambda 作为一等公民

Java8 的方法引用**::**语法（把这个方法作为值）。例如：File::isHidden

Lambda（或匿名函数），例如：（int x）-> x+1

1.3 流

我们把 for-each 循环一个个去迭代元素称为外部迭代，有了 Stream API 数据处理全是在库内进行的，这种思想叫作内部迭代。

函数式编程中的函数的主要意思是“把函数作为一等值”，不过它也常常隐含着第二层意思，即“执行时在元素之间无互动”。

1.4 默认方法

如何改变已发布的接口而不破坏已有的实现呢？为了解决这个问题，Java 8 中加入了接口的默认方法。加入默认方法后，Java 有了某种形式的多重继承，Java 8 用一些限制来避免出现类似 C++中凑名昭著的菱形继承问题。

2 通过行为参数化传递代码

行为参数化就是可以帮你处理频繁变更的需求的一种软件开发模式。

匿名类的不足之处：占用了很多空间，显得很笨重；用起来可能会让人费解。

场景：Comparator 排序，用 Runnable 执行一个代码块，以及 GUI 事件处理。

3 Lambda 表达式

基本语法：

（parameters）-> expression　　// 表达式

（parameters）-> { statements; }　// 语句

在需要函数接口的地方可以使用 Lambda 表达式。

函数式接口就是只定义一个**抽象方法**的接口。哪怕有很多**默认方法**，只要接口只定义了一个**抽象方法**，它就任然是一个函数式接口。

@FunctionalInterface 这个注解表示该接口会设计成一个函数式接口。不是必须的，类似于@Override。

任何函数式接口都不允许抛出受检查异常（checked exception）。如果你需要 Lambda 表达式来抛出异常，有两种办法：定义自己的函数式接口，并声明受检查异常；或者把 Lambda 包在一个 try/catch 块中。

4 引入流

Stream API 可以让你写出这样的代码：

（1）声明性——更简洁，更易读

（2）可复合——更灵活

（3）可并行——性能更好

Java 8 引入流的理由：Streams 库的内部迭代可以自动选择一种适合你硬件的数据表示和并行实现。Java 8 需要一个类似于 Collection 却没有迭代器的接口，与是就有了 Stream!

和迭代器类似，流只能遍历一次。

5 使用流

5.1 筛选和切片

filter、distinct、limit(n)、skip(n)

5.2 映射

map、flatMap

5.3 查找和匹配

allMatch、anyMatch、noneMatch、findFirst、findAny

5.4 归约

reduce

5.5 构建流

由值创建流、由数组创建流、由文件创建流、由函数生成流（创建无限流）
函数生成流：iterate、generate

6 用流收集数据

Collectors 类提供工厂方法创建收集器，提供了三大功能：
（1）将元素规约和汇总为一个值
（2）元素分组
（3）元素分区

7 并行数据处理与性能

Java 7 引入了一个叫作分支/合并的框架，让让并行处理数据操作更加稳定，更不易出错。

parallel、sequential 调用会影响整个流水线。

优化性能时，你应该始终遵循三个黄金规则：测量，测量，测量。

很重要的的一点是要保证在内核中并行执行工作的时间比在内核之间传输数据的时间长。总而言之，很多情况下不可能或不方便进行并行化。

记住要避免共享可变状态，确保并行 Stream 得到正确的结果。

留意装箱。自动装箱和拆箱操作会大大降低性能。

对于较少的数据量，选择并行流几乎从来都不是一个好的决定。

分支/合并框架的目的是以递归方式将可以并行的任务拆分成更小的任务，然后将每个子任务的结果合并起来生成结果。它是 ExecutorService 接口的一个实现，它把子任务分配给线程池（称为 ForkJoinPool）中的工作线程。

分支/合并框架工程用一种称为工作窃取（work stealing）的技术来解决这个问题。每个线程池都为分配给它的任务保存一个双向链式队列，完成任务的线程可以从其他线程任务队列的尾端“偷走”一个任务。

Spliterator（可分迭代器）是 Java 8 中加入的另一个新接口，可以让你控控制分数据结构的策略。

8 重构、测试和调试

如果你使用了匿名类，尽量使用 Lambda 表达式替换它们，但是要注意二者间语义的微妙差别，比如关键字 this，以及变量隐藏。

尽量使用 Stream API 替换迭代式的集合处理。

尽量将复杂的 Lambda 表达式抽象到普通方法中。

Lambda 表达式会让栈跟踪的分析变得更为复杂。

流提供的 peek 方法在分析 Stream 流水线时，能将中间变量的值输出到日志中，是非常有用的工具。

9 默认方法

Java 8 允许在接口内声明静态方法；Java 8 引入了默认方法，通过默认方法你可以指定接口方法的默认实现。

引入默认方法的目的：它让类可以自动地继承接口的一个默认实现。

抽象类与抽象接口的区别：

（1）一个类只能继承一个抽象类，但一个类可以实现多个接口。

（2）一个抽象类可以通过实例变量（字段）保存一个通用状态，而接口是不能有示例变量的。

默认方法的使用模式：可选方法和行为的多继承。

多个接口默认方法冲突的解决机制：

（1）首先，类或父类中显示声明的方法，其优先级高于所有的默认方法。

（2）如果用第一条无法判断，方法签名又没有区别，那么选择提供最具体实现的默认方法的接口。

（3）最后，如果冲突依旧无法解决，你就只能在你的类中覆盖默认方法，显示地指定在你的类中使用哪一个接口中的方法。

10 用 Optional 取代 null

Java 8 中引入了一个新的类 java.util.Optional<T>，对存在或缺失的变量进行建模。

可以使用 Optional 中的 empty、of、ofNullable 创建 Optional 对象。

Optional 类支持多种方法，比如：map、flatMap、filter，它们在概念上与 Stream 类中对应的方法十分相似。

11 CompletableFuture 组合式异步编程

同步 API 中调用方在被调用方运行的过程中会等待，异步 API 会直接返回。

如果你进行的是计算密集型的操作，并且没有 I/O，那么推荐使用 Stream 接口，因为实现简单，同时效率也可能是最高的（如果所有的线程都是计算密集型的，那么没有必要创建比处理器核数更多的线程）。

反之，如果你并行的工作单元还涉及等待 I/O 的操作（包括网络连接等待），那么使用 CompletableFuture 灵活性更好。

CompletableFuture 类还提供了异常管理的机制，让你有机会抛出/管理异步任务执行中发生的异常。

如果异步任务之间相互独立，或者它们之间某一些的结果是另一些的输入，你可以将这些异步构造或者合并一个。

你可以为 CompletableFuture 注册一个回调函数；你可以决定在什么时候结束程序的运行，是等待由 CompletableFuture 对象构成的列表中所有对象都执行完毕，还是只要其中任何一个首先完成就中止程序的运行。