java新特性

简介

本文基于JDK8和JDK9的新特性说明。

官网说明：

jdk8更新指南： https://www.oracle.com/java/technologies/javase/8-whats-new.html

jdk9更新指南:https://docs.oracle.com/javase/9/language/toc.htm#JSLAN-GUID-B06D7006-D9F4-42F8-AD21-BF861747EDCF

stream流

流的定义：从支持数据处理操作的源生成的元素序列

Stream API的特点

- 声明性——更简洁，更易读；
- 可复合——更灵活；
- 可并行——性能更好。

使用Collection接口需要用户去做迭代（比如用for-each），这称为外部迭代。相反，Stream库使用内部迭代

线性操作：filter、map、limit、collect

中间操作：诸如filter或sorted等中间操作会返回另一个流。可以连接起来的流操作称为中间操作。

终端操作：关闭流的操作。终端操作会从流的流水线生成结果，其结果是任何不是流的值，比如List、Integer，甚至void。

流操作的结合：

一个数据源（如集合）来执行一个查询；

一个中间操作链，形成一条流的流水线；

一个终端操作，执行流水线，并能生成结果。

数据源：Arrays.stream(),

中间操作：filter，map,limit,sorted,distinct

终端操作：forEach,count,collect

使用流

筛选

filter：Stream接口支持filter方法（你现在应该很熟悉了）。该操作会接受一个谓词（一个返回boolean的函数）作为参数，并返回一个包括所有符合谓词的元素的流

distinct：流还支持一个叫作distinct的方法，它会返回一个元素各异（根据流所生成元素的hashCode和equals方法实现）的流

切片

takeWhile：

采用filter的缺点是，你需要遍历整个流中的数据，对其中的每一个元素执行谓词操作。

takeWhile操作就是为此而生的！它可以帮助你利用谓词对流进行分片（即便你要处理的流是无限流也毫无困难）。更妙的是，它会在遭遇第一个不符合要求的元素时停止处理。

dropWhile：

dropWhile操作是对takeWhile操作的补充。它会从头开始，丢弃所有谓词结果为false的元素。一旦遭遇谓词计算的结果为true，它就停止处理，并返回所有剩余的元素，即便要处理的对象是一个由无限数量元素构成的流，它也能工作得很好

截断

limit：流支持limit(n)方法，该方法会返回另一个不超过给定长度的流。所需的长度作为参数传递给limit。如果流是有序的，则最多会返回前n个元素

跳过

skip：流还支持skip(n)方法，返回一个扔掉了前n个元素的流

映射

map：流支持map方法，它会接受一个函数作为参数。这个函数会被应用到每个元素上，并将其映射成一个新的元素（使用映射一词，是因为它和转换类似，但其中的细微差别在于它是“创建一个新版本”而不是去“修改”）

flatMap：使用flatMap方法的效果是，各个数组并不是分别映射成一个流，而是映射成流的内容。

匹配

allMatch、anyMatch、noneMatch、findFirst和findAny

anyMatch方法可以回答“流中是否有一个元素能匹配给定的谓词”

allMatch方法的工作原理和anyMatch类似，但它会看看流中的元素是否都能匹配给定的谓词

短路：

anyMatch、allMatch和noneMatch这三个操作都用到了所谓的短路，这就是大家熟悉的Java中&&和||运算符短路在流中的版本。

不管表达式有多长，你只需找到一个表达式为false，就可以推断整个表达式将返回false，所以用不着计算整个表达式。这就是短路。

查找

findAny：查找元素findAny方法将返回当前流中的任意元素。

findFirst：查找第一个元素。

何时使用findFirst和findAny

你可能会想，为什么会同时有findFirst和findAny呢？答案是并行。找到第一个元素在并行上限制更多。如果你不关心返回的元素是哪个，请使用findAny，因为它在使用并行流时限制较少。

归约

reduce方法将流归约成一个值。用函数式编程语言的术语来说，这称为折叠（fold）

求和

```
reduce接受两个参数：

一个初始值，这里是0；

一个BinaryOperator<T>来将两个元素结合起来产生一个新值，这里用的是lambda (a, b)-> a + b

无初始值reduce还有一个重载的变体，它不接受初始值，但是会返回一个Optional对象：
```

最大值

最小值

归约方法的优势与并行化相比于前面写的逐步迭代求和，使用reduce的好处在于，这里的迭代被内部迭代抽象掉了，这让内部实现得以选择并行执行reduce操作

流操作：无状态和有状态

无状态：它们没有内部状态（假设用户提供的Lambda或方法引用没有内部可变状态）。

有状态：从流中排序和删除重复项时都需要知道先前的历史。

filter 中间.
distinct 中间(有状态-无界)
takeWhile 中间
dropWhile 中间
skip .中间(有状态-无界)
limit 中间(有状态无界)
map中间
flatMap 中间.
sorted 中间(有状态-无界)
anyMatch 终端
noneMatch 终端.
allMatch 终端.
findAny 终端
findFirst 终端
forEach 终端
collect终端
reduce 终端(有状态有界)
count 终端.

数值流

使用reduce方法计算流中元素的总和

原始类型流特化

IntStream、DoubleStream和LongStream，分别将流中的元素特化为int、long和double，从而避免了暗含的装箱成本

映射到数值流

转换回对象流

默认值OptionalInt：如何区分没有元素的流和最大值真的是0的流呢

数值范围：range和rangeClosed

构建流

静态方法Stream.of

由可空对象创建流

由数组创建流

由文件生成流

Stream.iterate和Stream.generate。这两个操作可以创建所谓的无限流：

收集器

收集器用作高级归约

Collectors实用类提供了很多静态工厂方法，可以方便地创建常见收集器的实例

最直接和最常用的收集器是toList静态方法，它会把流中所有的元素收集到一个List中

三大功能

- 将流元素归约和汇总为一个值；
- 元素分组；
- 元素分区。

利用counting工厂方法==返回==的收集器

counting收集器在和其他收集器==联合使用==的时候特别有用

Collectors.maxBy和Collectors.minBy，来计算流中的==最大值或最小值==

汇总

它可接受一个把对象映射为求和所需int的函数，并返回一个收集器；该收集器在传递给普通的collect方法后即执行我们需要的汇总操作

广义的归约汇

总事实上，我们已经讨论的所有收集器，都是一个可以用reducing工厂方法定义的归约过程的特殊情况而已。

Collectors.reducing工厂方法是所有这些特殊情况的一般化

> 收集与归约
> `Stream`接口的`collect`和`reduce`方法有何不同，因为两种方法通常会获得相同的结果
> 一个语义问题和一个实际问题。语义问题在于，`reduce`方法旨在把两个值结合起来生成一个新值，它是一个不可变的归约。与此相反，`collect`方法的设计就是要改变容器，从而累积要输出的结果。这意味着，上面的代码片段是在滥用`reduce`方法，因为它在原地改变了作为累加器的`List`、
>
> 归约过程不能并行工作，因为由多个线程并发修改同一个数据结构可能会破坏`List`本身。在这种情况下，如果你想要线程安全，就需要每次分配一个新的`List`，而对象分配又会影响性能。这就是`collect`方法特别适合表达可变容器上的归约的原因，更关键的是它适合并行操作

分组

Collectors.groupingBy工厂方法返回的收集器做到分组

给groupingBy方法传递了一个Function（以方法引用的形式），它提取了流中每一道Dish的Dish.Type。我们把这个Function叫作分类函数，因为它用来把流中的元素分成不同的组

空的列表分组

还可以更进一步地使用Collector对过滤的元素继续进行分组。通过这种方式，结果映射中依旧保存了FISH类型的条目，即便它映射的是一个空的列表

操作分组元素的另一种常见做法是使用一个映射函数对它们进行转换，这种方式也很有效。为了达成这个目标，Collectors类通过mapping方法提供了另一个Collector函数，它接受一个映射函数和另一个Collector函数作为参数。

多级分组

要实现多级分组，可以使用一个由双参数版本的Collectors.groupingBy工厂方法创建的收集器，它除了普通的分类函数之外，还可以接受collector类型的第二个参数。

按子组收集数据

分区

由一个谓词（返回一个布尔值的函数）作为分类函数，它称分区函数

分区的优势分区：在于保留了分区函数返回true或false的两套流元素列表

partitioningBy收集器也可以结合其他收集器使用。尤其是它可以与第二个partitioningBy收集器一起使用来实现多级分区。

收集器接口

为Collector接口提供自己的实现，从而自由地创建自定义归约操作

建立新的结果容器

supplier方法

supplier方法必须返回一个结果为空的Supplier，也就是一个无参数函数，在调用时它会创建一个空的累加器实例，供数据收集过程使用

将元素添加到结果容器：accumulator方法

对结果容器应用最终转换：finisher方法

合并两个结果容器：combiner方法

characteristics方法

ToListCollector

比较收集器的性能

开发自定义收集器并不是白费工夫，原因有二：第一，你学会了如何在需要的时候实现自己的收集器；第二，你获得了大约32%的性能提升。

并行数据处理与性能

并行流

并行流就是一个把内容拆分成多个数据块，用不同线程分别处理每个数据块的流。这样一来，你就可以自动地把工作负荷分配到多核处理器的所有核，让它们都忙起来

并行和顺序转换

对并行流调用sequential方法就可以把它变成顺序流

测试性能

JMH是一个以声明方式帮助大家创建简单、可靠微基准测试的工具集，它支持Java，也支持可以运行在Java虚拟机（Java virtual machine, JVM）上的其他语言

正确使用并行流

错用并行流而产生错误的首要原因，就是使用的算法改变了某些共享状态

高效使用并行流

并行流并不总是比顺序流快

自动装箱和拆箱操作会大大降低性能

有些操作本身在并行流上的性能就比顺序流差。特别是limit和findFirst等依赖于元素顺序的操作

还要考虑流的操作流水线的总计算成本

对于较小的数据量，选择并行流几乎从来都不是一个好的决定

要考虑流背后的数据结构是否易于分解

流自身的特点以及流水线中的中间操作修改流的方式，都可能会改变分解过程的性能

还要考虑终端操作中合并步骤的代价是大是小

#分支/合并框架

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整体结果。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。

使用RecursiveTask

要把任务提交到这个池，必须创建RecursiveTask的一个子类，其中R是并行化任务（以及所有子任务）产生的结果类型，或者如果任务不返回结果，则是RecursiveAction类型

ForkJoinSumCalculator

工作窃取

每个线程都为分配给它的任务保存一个双向链式队列，每完成一个任务，就会从队列头上取出下一个任务开始执行。基于前面所述的原因，某个线程可能早早完成了分配给它的所有任务，也就是它的队列已经空了，而其他的线程还很忙。这时，这个线程并没有闲下来，而是随机选了一个别的线程，从队列的尾巴上“偷走”一个任务。这个过程一直继续下去，直到所有的任务都执行完毕，所有的队列都清空。

Spliterator

Spliterator也用于遍历数据源中的元素，但它是为了并行执行而设计的

Spliterator接口声明的最后一个抽象方法是characteristics，它将返回一个int，代表Spliterator本身特性集的编码。使用Spliterator的客户可以用这些特性来更好地控制和优化它的使用

WordCounterSpliterator来处理并行流

#java9-Collection API的增强功能

Java 9引入了一些新的方法，可以很简便地创建由少量对象构成的Collection

Java 9新增的工厂方法可以简化小规模List、Set或者Map的创建

List.of

Set.of

Map.of

新方法

Java 8在List和Set的接口中新引入了以下方法

removeIf移除集合中匹配指定谓词的元素。实现了List和Set的所有类都提供了该方法（事实上，这个方法继承自Collection接口）。❑ replaceAll用于List接口中，它使用一个函数（UnaryOperator）替换元素。❑ sort也用于List接口中，对列表自身的元素进行排序。

排序

排序有两种新的工具可以帮助你对Map中的键或值排序，它们是：❑ Entry.comparingByValue❑ Entry.comparingByKey

getOrDefault方法

计算模式

❑ computeIfAbsent——如果指定的键没有对应的值（没有该键或者该键对应的值是空），那么使用该键计算新的值，并将其添加到Map中；❑ computeIfPresent——如果指定的键在Map中存在，就计算该键的新值，并将其添加到Map中；❑ compute——使用指定的键计算新的值，并将其存储到Map中。

删除模式

替换模式

merge方法

改进的ConcurrentHashMap

归约和搜索

计数

Set视图

重构、测试和调试

使用Lambda重构面向对象的设计模式

测试可见Lambda函数的行为

测试使用Lambda的方法的行为

将复杂的Lambda表达式分为不同的方法

调试

查看栈跟踪

使用日志调试

流操作方法peek大显身手的时候。peek的设计初衷就是在流的每个元素恢复运行之前，插入执行一个动作。但是它不像forEach那样恢复整个流的运行，而是在一个元素上完成操作之后，只会将操作顺承到流水线中的下一个操作

java8迭代部分

用Optional取代null

了解一下Optional里面几种可以迫使你显式地检查值是否存在或处理值不

- isPresent()将在Optional包含值的时候返回true，否则返回false。
- ifPresent(Consumer block)会在值存在的时候执行给定的代码块。第3章介绍过Consumer函数式接口，它让你传递一个接受T类型参数，并返回void的Lambda表达式。
- T get()会在值存在时返回值，否则抛出一个NoSuchElement异常。
- T orElse(T other)会在值存在时返回值，否则返回一个默认值。

应用Optional的几种模式

创建Optional对象

使用map从Optional对象中提取和转换值

使用flatMap链接Optional对象

操纵由Optional对象构成的Stream

默认行为及解引用Optional对象

用Optional封装可能为null的值

新的日期和时间API

LocalDate、LocalTime、LocalDateTime、Instant、Duration以及Period

LocalDateTime

是LocalDate和LocalTime的合体。它同时表示了日期和时间，但不带有时区信息，你可以直接创建，也可以通过合并日期和时间对象创建

定义Duration或Period

操纵、解析和格式化日期

使用TemporalAdjuster

打印输出及解析日期-时间对象I

处理不同的时区和历法

默认方法

我们已经了解了向已发布的API添加方法，对现存代码实现会造成多大的损害。默认方法是Java8中引入的一个新特性，希望能借此以兼容的方式改进API。

Java 8中的抽象类和抽象接口那么抽象类和抽象接口之间的区别是什么呢？它们不是都能包含抽象方法和方法体的实现吗？首先，一个类只能继承一个抽象类，但是一个类可以实现多个接口。其次，一个抽象类可以通过实例变量（字段）保存一个通用状态，而接口是不能有实例变量的。

默认方法的使用模式

可选方法

行为的多继承

解决冲突的规则

菱形继承问题

Java 8中的接口可以通过默认方法和静态方法提供方法的代码实现。❑ 默认方法的开头以关键字default修饰，方法体与常规的类方法相同。❑ 向发布的接口添加抽象方法不是源码兼容的。❑ 默认方法的出现能帮助库的设计者以后向兼容的方式演进API。❑ 默认方法可以用于创建可选方法和行为的多继承。❑ 我们有办法解决由于一个类从多个接口中继承了拥有相同函数签名的方法而导致的冲突。❑ 类或者父类中声明的方法的优先级高于任何默认方法。如果前一条无法解决冲突，那就选择同函数签名的方法中实现得最具体的那个接口的方法。❑ 两个默认方法都同样具体时，你需要在类中覆盖该方法，显式地选择使用哪个接口中提供的默认方法。