# Java 8介绍

Lambdas和Streams的快速入门和引导

英文文档请见：https://www.oreilly.com/programming/free/files/introducing-java-8.pdf

# 第一章：为何你要关心java8

Java有了新的变革，在2014年3月发布了java8的新版本。Java8的新特性将改变你每天的基础编程。无须担心，本篇简介将带你快速浏览新特性的本质以便你快速的进行上手。

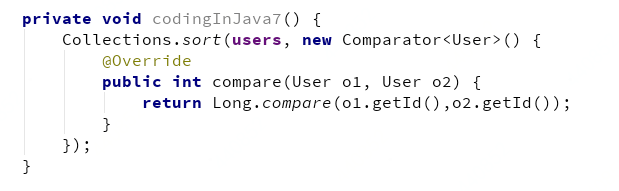
本章概述了java8中主要新增的特性，后序两章将详细介绍Lambda表达式和Streams

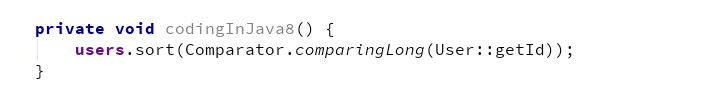
Java8作出如此改变的动机有以下两点：

* 更好代码可读性
* 更简单的去支持多核（多线程编程）

## 代码可读性：

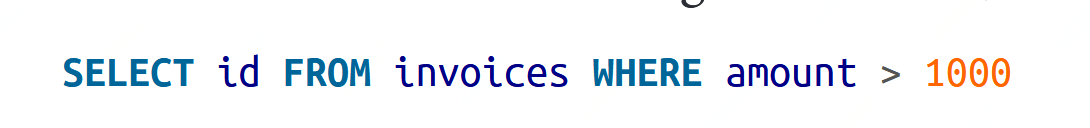
Java编码本身是十分冗长的，导致了降低了代码的可读性。换句话说，java需要大量的代码去描述一个简单的概念。这有一个例子：说的是你需要从一个User的List集合中依据id进行排序。在java8之前你可能会这样去写：

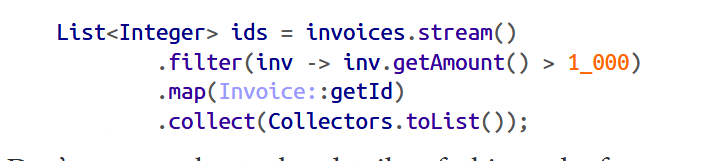
这种编码方式下，你需要注意很多小的细节在如何进行比较上。换句话说，它很难去直观的展示这部分代码的业务逻辑。你需要创建一个Comparator对象去定义如何对User对象进行比较，因此你需要提供一个compare方法的实现。当你阅读这段代码的时候你需要花大量的时间在搞清楚这个compare实现上，而不是这段代码所实现的业务逻辑。

在Java8中你可以这样去写这段代码来解决刚才所描述的业务逻辑：

现在业务逻辑变得十分清晰（不用担心语法问题，新的语法将会在后面章节进行介绍）。Java8的语法特征和API的更新使得你写的代码更加简洁和易读，而这恰恰正是为何要学习java8。

此外Java8还新增了一个API接口Streams API。Stream可以让你写出可读性更高的对于数据处理的代码。Streams API内置了一些简单的对数据处理的方法。例如：在企业经营的场景下，你想搞一个各个部门的发票总计日结算报表。好消息是在Streams API下你无需担心如何实现这个查询本身。

在Stream API中跟使用SQL进行查询十分相似。事实上在SQL中，你可以十分清晰的进行一个查询而不用关系其底层是如何实现的。例如说，你想查询所有的金额大于1000元的发票ID。

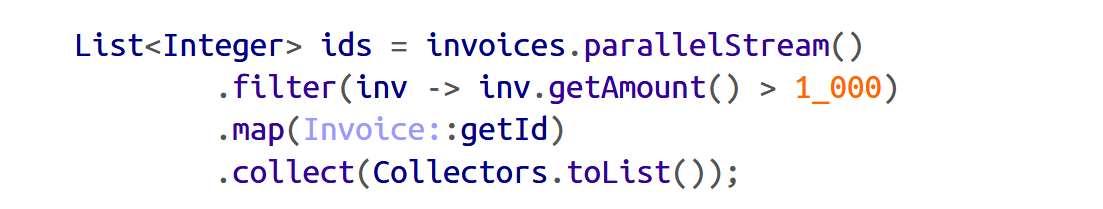
这种便于查询的书写风格通常我们通常称之为声明式编程。这是一个你如何应该使用Stream API 解决上述问题的编码示例：（invoices 是一个Invoice 对象的List集合）

暂时不用担心如何编码实现这个功能，在第三章中将会深入的介绍Stream API。现在想一下，Stream作为一种新的概念以一种清晰易读的方式表达了数据查询的处理。

## 多核

Java8中第二大改变是更好的支持了多核处理器。在过去，你的电脑只会使用一个核（cpu）。如果想让程序跑的更快通常意味着增和核心的处理能力。不幸的是，处理器的时钟速度很难变得更快了。而如今，大多数电脑和移动设备都使用多核进行并行处理。

应用程序应当利用不多个核（cpu）去提升性能。Java应用具有代表性的解决方案就是使用Thread(线程)。不幸的是，使用线程去进行编码变得越来越复杂，并且通常只有少部分专家才会。

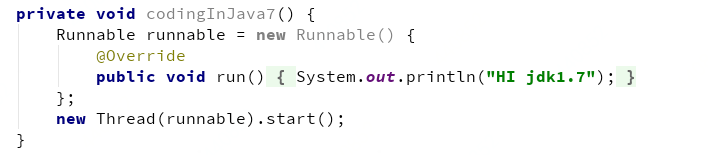
Streams API在java8中是的多线程处理数据更加的容易。例如在解决刚才的问题时，编码只需将stream()方法替换为parallelStream()方法。

在第三章我将会详细的讨论和举出更合适去使用parallel streams的例子

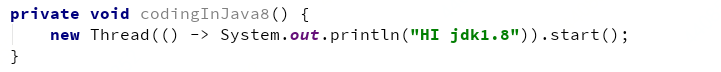
## Java8新特性一览表

这一小段提供了java8中新特性的一个概览，并且提供了代码示例以便更好的理解如何应用这些特性。接下来的两个章节将会专注于最重要两个特性：lamdba表达式和Streams。

### Lambda表达式

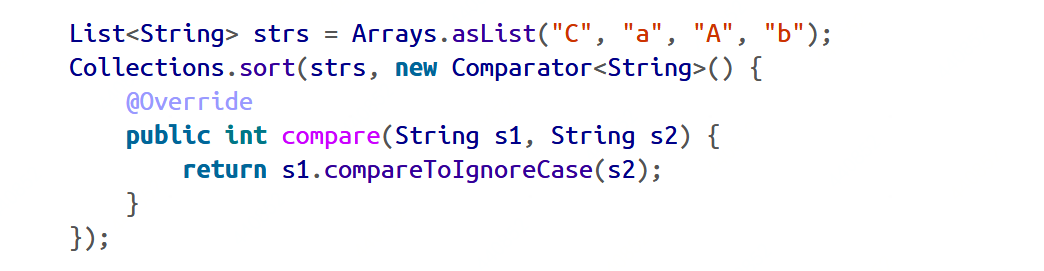
Lambda表达式使你的代码变的简洁，例如你需要一个线程去执行一项任务，你会像下面这段代码一样创建一个Runnable对象,然后使用一个线程去调用它。

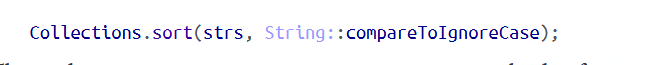
使用lambda表达式则可以让你之前写的代码变得更加易于理解：



### 方法引用（Method Reference）

方法引用是一个与lambda表达式相关的新特性。它可以让你选择一个类中已经存在的方法进行传递使用。例如，你要忽略大小写的去对一个字母的List进行排序。Java8之前你这样去实现：

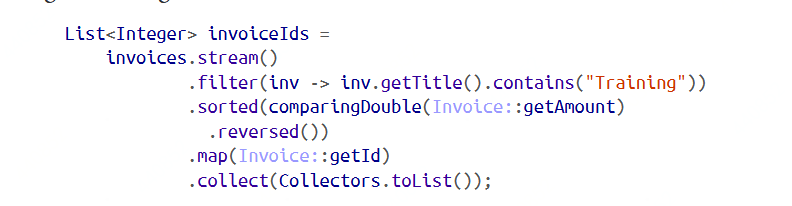
这段代码极其的冗长。毕竟你需要compareToIgnoreCase这个方法。使用方法引用可以明确的指出是运用String类中的compareToIgnoreCase去执行比较方法。

String:: compareToIgnoreCase 这段代码就是使用了方法引用，它使用了双冒号::这种语法。

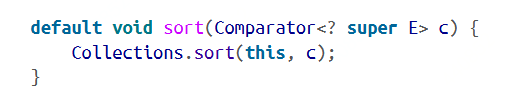
### Stream

最近，每个java应用都被创建用来处理集合。处理和组装数据成为了许多编码中的最基础的部分。然而使用集合进行处理使得代码变得十分冗长并且难以并行执行。下面这段代码展示了对集合进行处理的代码是何等的冗余。业务逻辑是：按照金额大小排序收集与培训（training）有关的发票编号(id)：

Java8介绍了一种新的概念Stream，它可以让你更加清晰的以声明的方式去处理数据。在Java8中你可以使用Stream重构刚才的代码。

此外，你还可以明确的指定并发的去执行，只需要在集合对象上使用parallelStream()方法来替换掉stream()就可以了。（后面章节将会详细的讲解）

### 增强型接口

Interface在java8中可以有两种方式进行方法实现的声明。第一，Java8中可以使用default methods在接口中进行方法实现的编码。这种特性被以一种向前兼容的机制被引入Java API中。比如说，你可以看到在Java8中List支持的sort方法就是使用default methods去定义的。

deault methods也可以通过继承机制成为子类的行为显示。事实上，在Java8之前，一个类可能实现了多个接口。现在你可以继承不同Interface的deault methods。在Java8中已经明确的避免了在c++中的公共继承的问题（比如说多继承问题 又称钻石问题）

### 新的日期类型API

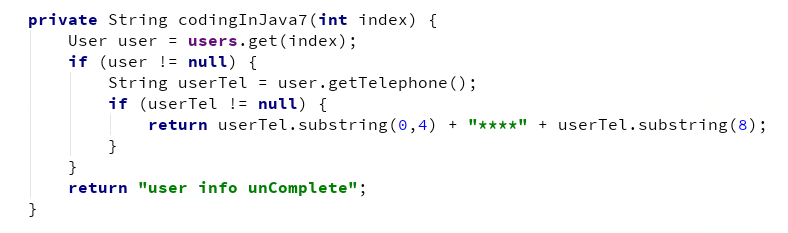
这个没啥意思了，不太想翻译了。大家自己看英文原文吧，大概就是说有了有了一个叫LocatedDateTime的类，可以更友好的去操作时间

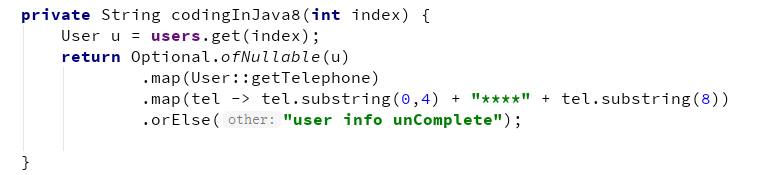
### ComletableFuture

这个有点复杂，翻译不动。

### Optional

Java8中介绍了一个叫做Optional的类。受到函数式编程的启发，引进了一种在你的对象可能是null的情形下的更好的模板。可以把它想象成一个包含一个单一值或没有值的对象。Optional已经在集合框架（例如guava）中被提供，但现在也是Java API中的一部分了。Optional的另一个好处就是它可以帮助你避免空指针异常。事实上Optional的方法检查了对象的存在与否。看以下的代码：

如果users.get(5) 返回的是null，那么这段代码将抛出空指针异常。如果getTelePhone()返回的是null，那这段代码也会返回一个空指针异常。换句话说，在链式调用中如果任何方法返回了null，那么就会抛出一个空指针异常。当然，你也可以这么去写代码去强制的检查是否为null

在这个代码中，需要连续判断很多次是否为null，并且容易就被忽略导致异常。除此之外这段代码变得十分冗长并且可读性较低。使用Optional，你可以让编码变得更加简洁和易读。如下：

在任何一个节点如果获取到的值是null那么将会返回“user info unComplete”。