Java, SQL and jOOQ. *Best Practices and Lessons Learned from Writing Awesome Java and SQL Code. Get some hands-on insight on what's behind developing jOOQ.*

January 24, 2014

- **in java, java 8**
- **Leave a comment**

Java 8 Friday Goodies: The New New I/O APIs

i
2 Votes

At Data Geekery (http://www.datageekery.com/), we love Java. And as we're really into jOOQ's fluent API and query DSL (http://www.jooq.org), we're absolutely thrilled about what Java 8 will bring to our ecosystem. We have blogged a couple of times about some nice Java 8 goodies (http://blog.jooq.org/tag/java-8/), and now we feel it's time to start a new blog series, the…

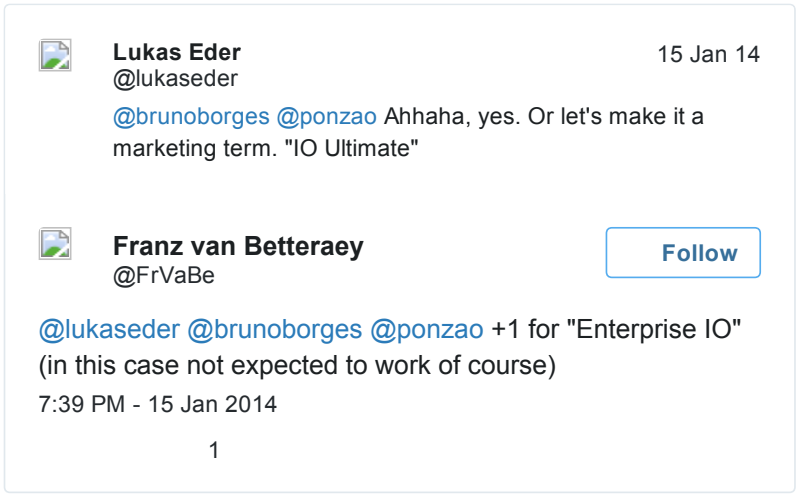
Java 8 Friday

Every Friday, we're showing you a couple of nice new tutorial-style Java 8 features, which take advantage of lambda expressions, extension methods, and other great stuff. You'll find the source code on GitHub (https://github.com/jOOQ/jOOQ-s-Java-8-Goodies). (http://twitter.com/home?status=Awesome+Weekly+%23Java8+Goodies+by+@JavaOOQ!+http://blog.jooq.org/category/java-8)

Java 8 Goodie: The New New I/O APIs

In a previous blog post from this series (http://blog.jooq.org/2014/01/10/java-8-friday-goodies-java-io-finally-rocks/), we have shown how Java 8's lambda expressions improve on the existing (yet outdated) JDK 1.2 I/O API, mainly by helping you express `java.io.FileFilter (http://docs.oracle.com/javase/7/docs/api/java/io/FileFilter.html)` instances as lambda expressions.

Many readers have rightfully pointed out that much of the `java.io` API has been superseded by Java 7's `java.nio` API, where "N" stands for "New" (http://en.wikipedia.org/wiki/New_I/O) (I know. New. Old. Old-2. Old-2-FIXME. Old-2-TODO…). But things get even better with Java 8. We're calling it the New New I/O APIs (NNIO), though jOOQ community members have suggested to call it "Enterprise IO":



Back to more constructive blogging. Let's have a short walk (pun intended, see `Files.walk()` (http://download.java.net/jdk8/docs/api/java/nio/file/Files.html#walk-java.nio.file.Path-java.nio.file.FileVisitOption...-)) around the improved Java 8 NIO features. Let's first have a look at the new methods in `java.nio.Files` (http://download.java.net/jdk8/docs/api/java/nio/file/Files.html). It's actually quite awesome that we can finally just list contents of a Path! In Java 8 we would use the newly introduced `Files.list()` (http://download.java.net/jdk8/docs/api/java/nio/file/Files.html#list-java.nio.file.Path-), which returns a lazy Stream of files:

```
1  Files.list(new File(".").toPath())
2       .forEach(System.out::println);
```

The output I get is this:

```
.\.git
.\.gitignore
.\.idea
.\java8-goodies.iml
.\LICENSE.txt
.\pom.xml
.\README.txt
.\src
.\target
```

Remember that `forEach()` is a "terminal method" (http://blog.credera.com/technology-insights/java/java-8-part-1-lamdas-streams-functional-interfaces/), i.e. a method that consumes the stream. You mustn't call any further methods on such a Stream.

We could also skip all hidden files and list only the first three "regular" files like this:

```
1  Files.list(new File(".").toPath())
2       .filter(p -> !p.getFileName()
3                      .toString().startsWith("."))
4       .limit(3)
5       .forEach(System.out::println);
```

The new output I get is this one:

```
.\java8-goodies.iml
.\LICENSE.txt
.\pom.xml
```

Now, that's already pretty awesome. Can it get better? Yes it can. You can also "walk" (http://download.java.net/jdk8/docs/api/java/nio/file/Files.html#walk-java.nio.file.Path-java.nio.file.FileVisitOption...-) a whole file hierarchy by descending into directories using the new `Files.walk()` method. Here's how:

```
1  Files.walk(new File(".").toPath())
2       .filter(p -> !p.getFileName()
```

```
3                       .toString().startsWith("."))
4       .forEach(System.out::println);
```

Unfortunately, the above will create a Stream of Paths excluding all the hidden files and directories, but their descendants are still listed. So we get:

```
Omitted:
.\.git

But listed:
.\.git\COMMIT_EDITMSG
.\.git\config
.\.git\description
[...]
```

It is easy to understand why this happens. `Files.walk()` returns a (lazy) Stream of *all* descendant files. The call to `.filter()` will remove the ones that are hidden from the Stream, but this has no influence on any recursive algorithm that might apply in the implementation of `walk()`. Frankly, this is a bit disappointing. We cannot leverage Java 7's `Files.walkFileTree()` (http://docs.oracle.com/javase/7/docs/api/java/nio/file/Files.html#walkFileTree(java.nio.file.Path, java.nio.file.FileVisitor)) method, because the receiving `FileVisitor` (http://docs.oracle.com/javase/7/docs/api/java/nio/file/FileVisitor.html) type is not a @FunctionalInterface (http://download.java.net/jdk8/docs/api/java/lang/FunctionalInterface.html)

We can, however, inefficiently work around this limitation with the following trivial logic:

```
1   Files.walk(new File(".").toPath())
2        .filter(p -> !p.toString()
3                       .contains(File.separator + "."))
4        .forEach(System.out::println);
```

This now yields the expected

```
.
.\java8-goodies.iml
.\LICENSE.txt
.\pom.xml
.\README.txt
.\src
.\src\main
.\src\main\java
.\src\main\java\org
.\src\main\java\org\jooq
[...]
```

Good news, however, is the new `Files.lines()` (http://download.java.net/jdk8/docs/api/java/nio/file/Files.html#lines-java.nio.file.Path-) method. The following example shows how we can easily read line by line from a file, trimming each line (removing indentation) and filtering out the empty ones:

```
1   Files.lines(new File("pom.xml").toPath())
2        .map(s -> s.trim())
3        .filter(s -> !s.isEmpty())
4        .forEach(System.out::println);
```

The above yields:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xs
<modelVersion>4.0.0</modelVersion>
<groupId>org.jooq</groupId>
<artifactId>java8-goodies</artifactId>
<version>1.0-SNAPSHOT</version>
[...]
```

Conclusion

Clearly, the notion of lazy evaluation will produce a big amount of confusion in the community, similar to the fact that a Stream can be consumed only once. We are placing a bet that the Java 8 Streams API (http://stackoverflow.com/questions/tagged/streams-api) will be the single biggest source of new Stack Overflow questions.

Nonetheless, the Streams API will be awesome (http://blog.jooq.org/2013/11/02/does-java-8-still-need-linq-or-is-it-better-than-linq/), and next week on the Java 8 Friday series, we'll see how we can leverage lambda expressions and Streams to *sort* things, before we'll see how Java 8 will improve our database interactions!

More on Java 8

In the mean time, have a look at Eugen Paraschiv's awesome Java 8 resources page (http://www.baeldung.com/java8)

Tags: File system, Files, IO, java, Java 8, NIO, Walk