SRA GROUP
GSLetterNeo Vol.101
2016年12月

lambda 式の遅延実行を利用したログ出力

佐々木 和繁

◆ Java の lambda 式

Java の最新バージョンである Java SE 8 では、たくさんの新機能が追加されました。その中でも特に注目されることが多いのが「lambda 式」です。

lambda 式は関数型プログラミングっぽく記述できることがフィーチャーされがちですが、Java での lambda 式の特徴はそれだけではありません。その特徴の一つに遅延実行があります。

例えば、

```
method1(method2());
```

というコードでは、method2 が先に実行され、その return 値が method1 の引数に渡され、method1 が実行されます。つまり、メソッドが実行される順番は method2 → method1 の順番になります。

これが lambda 式の場合、

```
method3(() -> method4());
```

とすると、まず method3 が実行され、その method3 の処理の中で、引数で渡された lambda 式が実行されます。通常のメソッドと異なり、lambda 式は遅延して実行することが可能になります。

◆ Java のログ出力の問題

次に Java のログ出力についてですが、Java でログ出力を行う時に使用するもので有名なものには、以下のようなものがあります。

Java 標準のロギング機能(java.util.logging)
Log4j
Log4j2
Logback
Apache Commons Logging
SLF4J

多くの場合、これらの中から取捨選択して利用することになります。最初の 4 つは実際にログを出力するロギングライブラリ、後の 2 つはアダプタです。

今でも実際の開発現場では Log4j が使われているのを目にすることがありますが、Log4j は既に EOL (End Of Life) となっているので、新規にロギングライブラリを採用する場合は避けるべきでしょう。

本記事では Apache Commons Logging + Log4j2 の組み合わせで話を進めます。

Java のログ出力でよくあるのが、

```
if (LOG.isDebugEnabled()) {
    LOG.debug("user name is " + userName);
}
```

のように、ログレベルを検査する if 文で囲んでログを出力するコードです。これは、例えばログレベルが ERROR の場合、ログには出力されないのに文字列を生成するコストがかかってしまうのを避けるためです。

上の例はコストのかかるような文字列ではありませんが、例えばサイズの大きな List オブジェクトや、そもそもログに出力する内容自体をコストのかかる計算で求める場合などには役に立つイディオムだと思います。

しかしこの if 文のせいで JUnit のカバレッジ(C1)が下がってしまうし、かといってそのためだけにログレベルを動的に変更して無駄に 2 回テストを実行するのもイヤだし、ポリモーフィズムを駆使してせっかく if 文のないコードを書いているのにこの if 文のせいで汚された感じだし、第一このためだけに if 文を書くのは骨が折れます。

◆遅延実行による問題の回避

ところが、lambda 式の遅延実行によってこの問題を回避することができます。

```
LOG.debug(() -> "user name is " + userName);
```

もしこのように書けたら、そして debug メソッドの内部でログレベルを判断し、ログを出力する時だけ lambda 式を実行し文字列を生成するのであれば、無駄な文字列生成を回避することができます。

実際、Java8 の java.util.logging.Logger クラスには、Supplier<String>を引数にとるメソッドが定義されています。例えば info メソッドは文字列を引数にとる

```
public void info(String msg)
```

以外に

```
public void info(Supplier<String> msgSupplier)
```

というものが追加されています。同様に、Log4j2 にも引数に lambda 式を指定できるメソッドが存在します。これらは、実際にログ出力しない場合に無駄な文字列生成を回避するために追加されたメソッドと言えます。

それではログアダプタ、例えば Apache Commons Logging を使っている場合はどうでしょうか？

残念ながら org.apache.commons.logging.Log インターフェイスには lambda 式を引数に取ることができるメソッドはありません。

◆未対応ライブラリの解決策

Apache Commons Logging は実際のアプリケーションを開発する場合にもよく使われます。あるいは Java の開発時には Apache Commons のライブラリを使うことが多いですが、それらのライブラリは内部で Apache Commons Logging を使ってログ出力をしていることがほとんどです。

せっかく Java8 の lambda 式を使えば無駄な文字列生成コストを回避してログ出力を行えるのに、使用するライブラリが対応していなかったらどうしようもありません。

そこで今回、Apache Commons Logging と互換性のある Log4j2 用のブリッジを作りました。

https://github.com/kazsharp/lambda-logging-jcl

Maven のセントラルリポジトリにも登録しましたので、pom.xml に以下のように定義すれば使うことができます。

```
<dependency>
  <groupId>jp.gr.java_conf.kazsharp</groupId>
    <artifactId>lambda-logging-jcl</artifactId>
    <version>0.0.1</version>
</dependency>
```

使い方は Apache Commons Logging とほぼ同じですが、オブジェクト生成時に

```
private static LambdaLog4jLog LOG =
(LambdaLog4jLog)LogFactory.getLog(Hoge.class);
```

というように LambdaLog4jLog 型にキャストする必要があります。キャストしない場合は今まで通りの Apache Commons Logging として使うことができます。

つまり、既に Apache Commons Logging を使っているプロジェクトや、参照ライブラリが Apache Commons Logging を使用している場合でも、何の問題もなく使うことができます。

※今回の記事は、私が書いた「lambda 式の遅延実行を利用したログ出力」（http://qiita.com/kazsharp/items/b08ec29038f18403e62e） を元に、加筆・訂正したものです。

GSLetterNeo Vol.101
2016 年 12 月 20 日発行
発行者●株式会社 SRA 先端技術研究所
編集者●土屋正人

バックナンバを公開しています●http://www.sra.co.jp/gsletter
ご感想・お問い合わせはこちらへお願いします●gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋２－３２－８