先取り Java SE 8

Author : Acroquest Technology 勝本 秀之

第2回 ラムダ式が力を発揮する「ストリームAPI」

今回は、前回紹介したラムダ式の威力を最大限に発揮できるJava SE 8（以下、Java8）の新機能「ストリームAPI（Application Programming Interface）」について紹介します。ラムダ式は、実装しなければならないメソッドが1つしかないインタフェース（これを「関数型インタフェース」と呼びます）を楽に実装するための記法です。ラムダ式の記法を利用すれば、これから学ぶストリームAPIを使ったコードも簡単に記述できます。ストリームAPIについて深く学ぶことで、ラムダ式の効果がよくわかるはずです。

ハードウエアの進化によって登場した

ストリームAPIが生まれた背景には、近年のハードウエアの進化があります。現在のCPUは複数のCPUコアを1パッケージにまとめた「マルチコアCPU」が主流です。サーバーでは、それを複数搭載することも珍しくありません。たくさんのCPUを活用できれば、それだけ処理を早く終えられます。

Javaでマルチ CPUを活用するには、複数のスレッドを使ったマルチスレッドプログラミングによる並列処理の実装が必要でした。しかし、これまでのJavaではマルチスレッドプログラミングの記述が複雑なだけでなく、データの更新タイミングに留意する必要があるなど、多くの注意点があります。複雑なコーディングが必要になるのです。せっかくCPUが複数あるのに、簡単なプログラミングではそれを生かせなかったのです。

Java8では、並列処理を楽に実装できる言語機能を追加しました。それが、今回紹介する「ストリームAPI」です。ストリームAPIを使用することで、プログラマは今までよりも簡単にマルチスレッド処理を記述できます。また、ストリームAPIで書かれたプログラムは、これまでのfor文を使用するような記述方法に比べて、データを操作するプログラムをより直感的に書けます。

ロジックの記述に専念できる

それでは、簡単な例を見ていきましょう。**リスト1**のList1クラスと**リスト2**のPersonクラスがそれです。冒頭のimport文に注意してください。ストリームAPIを利用するために「java.util.stream.Collectors」を取り込んでいます。

リスト1の（1）で、List＜Person＞型のlistオブジェクトを作ります。このlistはStreamインタフェースを実装しています。（2）ではストリームのfilterやmapメソッドを使って30歳以上の人のスコアを取得し、最後にcollectメソッドで平均値を得ています*1。

図1はリスト1とリスト2をコンパイル、実行した様子です。OpenJDK 1.8.0のアーリーアクセス版（ビルド113）で動作を確認しています。アーリーアクセス版のJavaをインストールしたパソコンで、javacコマンドを使ってコンパイルし、javaコマンドでコンパイルしたコードを実行しています。

比較のために、Java7までの記法を確認しておきましょう。Java7以前で同様の処理を記述する場合は、**リスト3**のようにif文や拡張for文を使ってループ処理を書いていました。リスト1より変数の数や行数が多いものとなっています。Java8のストリームAPIを使えば、よりシンプルに記述できるのです。

リスト1の（2）では「.」（ドット）を使って次々とストリームAPIに含まれるメソッドを呼び出し、ラムダ式で条件を渡しています。データセット（ここではList型のlistオブジェクト）を対象に、何をどうしたいのかを記述すればよいのです。30歳以上にフィルタリングし、スコアを抜き出し、平均値を取得する、という記述になっています。

一方のJava7以前の記法で書いたコードでは、処理の内容を追っていくのも大変です。あちらこちらに処理が移るため途中で混乱してしまうこともあるかもしれません。ドットでつなげて処理を記述できるストリームAPIの方が楽に処理を追うことができると思います。

プログラマはストリームAPIを使うことで、ロジックの記述に専念できます。米サンフランシスコで開催されたイベント「JavaOne 2013」のBrian Goetz氏の発表でも、ロジックをストレートに記述できる点がストリームAPIのメリットであると説明していました。

ここからはストリームの記述方法を説明し、後半では並列化と並列化時の性能について取り上げます。

Streamインタフェースを使う

ストリームの処理を記述するには、java.util.stream.Streamインタフェースを理解する必要があります。このインタフェースには、リスト1に登場したfilter、map、collectといったメソッドが定義されています。http://download.java.net/jdk8/docs/api/java/util/stream/Stream.htmlを見てみるとよいでしょう。**表1**の用語を知っておくと、文書が読みやすいと思います。

リスト1●ストリームAPIの使用例（List1.java）

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class List1 {
  public static void main(String... args) {
    List<Person> list = Arrays.asList(
      new Person("Ken", 100, 34),
      new Person("Shin", 60, 28),
      new Person("Takuya", 80, 32));

    double average = list.stream()
      .filter(p -> p.getAge() >= 30)
      .map(p -> p.getScore())
      .collect(Collectors.averagingInt(s -> s.intValue()));
    System.out.println(average); // 90.0
  }
}
```

（1）（2）（3）

図1●リスト1と2をコンパイル、実行した様子

リスト2●Personクラス（Person.java）

```
public class Person {
  private int age;
  private String name;
  private int score;

  public Person(String name, int score) {
    this.name = name;
    this.score = score;
  }

  public Person(String name,
    int score, int age) {
    this.name = name;
    this.score = score;
    this.age = age;
  }

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getScore() {
    return score;
  }

  public void setScore(int score) {
    this.score = score;
  }

  public String toString(){
    return this.name + "_" + this.age
      + "_" + this.score;
  }
}
```

表1の用語を使ってリスト1の処理を説明すると、ソース（listオブジェクト）からStreamインタフェースを実装したオブジェクトを取り出し、Streamインタフェースに定義されたインターミディエイトオペレータメソッド（filter、map）を利用して処理します。最後にターミナルオペレータメソッド（collect）を使って結果を取り出します。結果を取り出す際に、コレク

*1 Collectors.averagingIntメソッドは、対象要素がない場合はゼロを返します。