Java 8 のイディオム: 関数型プログラミングへの移行の近道

Java プログラムに宣言型の考え方で関数型手法を導入する

Venkat Subramaniam
Founder
Agile Developer, Inc.

2017年 7月 06日
(初版 2017年 6月 01日)

Java プログラムに関数型手法を容易に導入する最初のステップは、命令型ではなく、宣言型の考え方でプログラミングする方法を学ぶことです。

このシリーズの他の記事を見る

Java は最初のリリースからこれまで常に、命令型プログラミングとオブジェクト指向プログラミングを支持してきたので、Java 開発者はこれらのスタイルに十分慣れているものですが、Java 8 では一連の新しい強力な関数型の機能と構文を使えるようになっています。登場してからすでに数十年が経っている関数型プログラミングは、概してオブジェクト指向プログラミングよりも簡潔で表現力に優れていて、エラーが発生しにくく、並列化するのも簡単です。これらの点は、関数型の機能を導入して Java プログラムを作成する十分な理由になりますが、関数型スタイルでプログラミングするには、コードの設計方法をいくつかの点で変えなければなりません。

このシリーズについて
Java 始まって以来の最も大きな更新となっている Java 8 には、どこから手を付けてよいのか戸惑ってしまうほど新しい機能が溢れています。このシリーズでは、教育専門家でもある著者の Venkat Subramaniam が簡潔にまとめた解説を通じて、イディオムを考慮した Java 8 手法を紹介し、当たり前のように思ってきた Java の慣例を見直してプログラムに新しい手法と構文を徐々に取り込んでいくよう読者を誘います。

関数型スタイルのプログラミングに移行するには、命令型ではなく、宣言型の考え方でコードに取り組むことが役立ちます。この新シリーズ「Java 8 のイディオム」の第 1 回では、命令型、宣言型、関数型プログラミング・スタイルの間で異なる点と一致する点を明らかにした上で、Java プログラムに宣言型の考え方で関数型の手法を徐々に取り込んでいく方法を説明します。

命令型スタイル

命令型スタイルでプログラミングの訓練を積んだ開発者は、プログラムに処理する内容とその方法を指示するのに慣れています。以下の単純な例を見てください。

リスト 1. 命令型スタイルでの findNemo

```
import java.util.*;
```

```
public class FindNemo {
  public static void main(String[] args) {
    List<String> names =
      Arrays.asList("Dory", "Gill", "Bruce", "Nemo", "Darla", "Marlin", "Jacques");

    findNemo(names);
  }

  public static void findNemo(List<String> names) {
    boolean found = false;
    for(String name : names) {
      if(name.equals("Nemo")) {
        found = true;
        break;
      }
    }

    if(found)
      System.out.println("Found Nemo");
    else
      System.out.println("Sorry, Nemo not found");
  }
}
```

上記の `findNemo()` メソッドは、まずガーベッジ変数とも呼ばれる可変のフラグ変数を初期化します。開発者は多くの場合、これらの変数は一時的な存在にすべきであるという開発者の一般的態度を反映して、フラグ変数に `f`、`t`、`temp` といった使い捨ての名前を付けます。上記の例では、フラグ変数に `found` という名前を付けています。

次に、このプログラムは指定された `names` リストに含まれる要素を一度に 1 つずつループ処理し、その名前が探している値 (この例では `Nemo`) と一致するかどうかをチェックします。値が一致するとフラグを `true` に設定し、制御フローにループを「抜ける」よう指示します。

このプログラムは、多くの Java 開発者にとって最も馴染み深い命令型スタイルのプログラムであるため、各要素を繰り返し処理し、値を比較し、フラグを設定し、ループを抜けるというプログラムのすべてのステップを定義しています。このように、命令型スタイルではすべてを完全に制御することができます。この特性は利点となることもありますが、開発者がすべてのコードを作成しなければなりません。大抵は、作業が少ないほうが生産性は上がります。

宣言型スタイル

宣言型プログラミングでもプログラムに処理内容を指示することに変わりはありませんが、実装の詳細については基礎となる関数ライブラリーに委ねます。リスト 1 の `findNemo` を、宣言型スタイルを使用して作成し直すとどうなるか見てみましょう。

リスト 2. 宣言型スタイルを使用した findNemo

```
  public static void findNemo(List<String> names) {
    if(names.contains("Nemo"))
      System.out.println("Found Nemo");
    else
      System.out.println("Sorry, Nemo not found");
  }
```

まず注目すべき点は、このバージョンではガーベッジ変数を一切使っていないことです。また、コレクションのループ処理に無駄な努力を費やすこともしていません。その処理は、組み込み

`contains()` メソッドに任せています。プログラムに処理内容 (探している値がコレクションに含まれているかどうかをチェックする) を指示するという点は命令型と同じですが、処理の詳細については、この基礎となるメソッドに任せているのです。

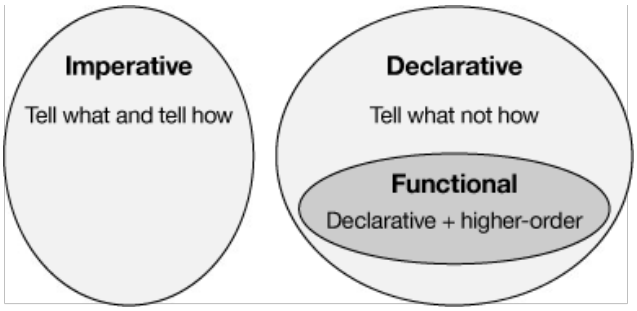
命令型バージョンの例では、開発者が反復処理を制御し、プログラムが指示されたとおりの処理を行いました。宣言型バージョンでは、処理が行われる限り、開発者はどのように処理されるのかに関与しません。`contains()` はさまざまな方法を使用して実装されることが考えられますが、期待する結果が得られるのであれば、開発者はそれで満足です。なにせ、少ない作業で同じ成果を上げられるのですから。

宣言型スタイルで考えるよう習慣づけると、Java の関数型プログラミングへの移行が大幅に楽になります。それは、関数型スタイルは宣言型スタイルに基づいているためです。宣言型で考えることが、命令型プログラミングから関数型プログラミングへの段階的移行になります。

関数型スタイル

関数型スタイルのプログラミングは常に宣言型スタイルですが、宣言型スタイルを使用するだけでは関数型プログラミングを実践していることにはなりません。このため、関数型プログラミングでは宣言型メソッドを高階関数と組み合わせて使用します。図 1 に、命令型、宣言型、および関数型プログラミング・スタイルの相互関係を示します。

図 1. 命令型、宣言型、関数型スタイルの相互関係



Java での高階関数

Java では、オブジェクトをメソッドに渡し、そのメソッド内でオブジェクトを作成し、そのオブジェクトをメソッドから返すというパターンに従いますが、このパターンを関数に適用することもできます。つまり、関数をメソッドに渡し、そのメソッド内で関数を作成し、作成した関数をメソッドから返すことができます。

このコンテキストでは、メソッドはクラスの一部 (静的またはインスタンス) になりますが、関数はメソッドにローカルなものである場合があります。その場合、関数にクラスやインスタンスを意図的に関連付けることはできません。関数を受け取ったり、関数を作成または返したりできるメソッドまたは関数は、高階関数と見なされます。

関数型プログラミングの例

新しいプログラミング・スタイルを導入するには、プログラムに関する考え方を改めなければなりません。単純な例を使用して練習を重ねることで、徐々に複雑なプログラムに取り組めるようになるという過程なのです。

リスト 3. 命令型スタイルでの Map の例

```
import java.util.*;

public class UseMap {
  public static void main(String[] args) {
    Map<String, Integer> pageVisits = new HashMap<>();

    String page = "https://agiledeveloper.com";

    incrementPageVisit(pageVisits, page);
    incrementPageVisit(pageVisits, page);

    System.out.println(pageVisits.get(page));
  }

  public static void incrementPageVisit(Map<String, Integer> pageVisits, String page) {
    if(!pageVisits.containsKey(page)) {
       pageVisits.put(page, 0);
    }

    pageVisits.put(page, pageVisits.get(page) + 1);
  }
}
```

リスト 3 では、`main()` 関数が Web サイトのページ閲覧数を保持する `HashMap` を作成します。それと同時に、`incrementPageVisit()` メソッドは特定のページが閲覧されるたびにカウントを増やしていきます。ここでは、このメソッドに焦点を当てます。

上記の `incrementPageVisit()` メソッドは命令型スタイルを使用して作成されています。このメソッドの役目は、特定のページのカウントを増分して `Map` に保管することです。このメソッドは特定のページのカウントがすでに存在するかどうかを知らないため、最初にカウントの有無をチェックします。存在しない場合は、その特定のページのカウントとして「0」を挿入します。以降はそのカウントを取得し、増分し、新しい値を `Map` に保管します。

宣言型で考えるには、このメソッドの設計を「方法」から「目標」に移行させなければなりません。`incrementPageVisit()` メソッドを呼び出すときは、特定のページのカウントを 1 に初期化するか、現行の値を 1 カウント増分する必要があります。それが、「目標」です。

宣言型を使用してプログラミングしていることから、次のステップとなるのは、JDK ライブラリーを調べて、`Map` インターフェース内でこの目標を達成できるメソッドを見つけることです。つまり、特定のタスクを完了する方法を把握している組み込みメソッドを見つけなければなりません。

ライブラリーを調べると、`merge()` メソッドがこの目標にぴったりであることがわかります。この新しい宣言型手法を用いてリスト 3 の `incrementPageVisit()` メソッドを変更しているのが、リスト 4 です。ただしこの新しいコードでは、より賢いメソッドを選んで宣言型スタイルの導入を進めているだけではありません。`merge()` は高階関数であるため、実際に関数型スタイルの好例となるコードを作成していることになります。

リスト 4. 関数型スタイルの Map の例

```
public static void incrementPageVisit(Map<String, Integer> pageVisits, String page) {
    pageVisits.merge(page, 1, (oldValue, value) -> oldValue + value);
}
```

リスト 4 では、`merge()` に最初の引数として `page` が渡されます。この最初の引数が、更新対象の値を持つキーです。該当するキーが `Map` 内にまだ存在していない場合は、2 番目の引数が初期値としての役割を果たします (この例では「1」)。3 番目の引数はラムダ式です。このラムダ式は、該当するキーのマップ内での現行値と、merge に 2 番目の引数として渡された値を格納する変数をパラメーターとして受け取ります。ラムダ式はこれらのパラメーターを合計した値を返し、つまりはカウントを増分します (編集者による注記: このコードの誤りを訂正してくれたIstván Kovács に感謝します)。

リスト 4 に示されている `incrementPageVisit()` メソッドのわずか 1 行のコードと、リスト 3 の複数行にわたるコードを見比べてください。リスト 4 のプログラムは関数型スタイルの一例ですが、宣言型の考え方で問題に取り組んだことが、プログラミング・スタイルの移行での大きな飛躍につながったのです。

まとめ

Java プログラムに関数型の手法と構文を取り入れると大きなメリットがもたらされます。具体的には、コードが簡潔になり、表現力が豊かになり、可変の部分が少なくなり、並列化が容易になり、一般にオブジェクト指向のコードより理解しやすくなります。課題となるのは、命令型スタイル (大多数の開発者にとって馴染み深いスタイル) のプログラミングの考え方を宣言型スタイルに変えることです。

関数型プログラミングは簡単でも単純でもありませんが、プログラムでの処理方法ではなく、何を目標としているかに重点を置く習慣をつけると、関数型プログラミングへの移行を大きく前進させることができます。また、基礎となる関数ライブラリーに処理方法を任せることで、そのうち徐々に、そして直感的に、関数型プログラミングのビルディング・ブロックである高階関数を把握できるようになります。

関連トピック： Java 8 言語での変更内容 Java 8 での並行処理の基礎 関数型の考え方: 関数型の観点で考える、第 1 回 - 関数型プログラマーのような考え方を習得する Javaによる関数型プログラミング —Java 8ラムダ式とStream (オライリージャパン、2014 年)