Spring Framework 5 の新機能

Spring 5 での Java 8 の関数型構文および新しいリアクティブ・プログラミング・モデルの活用方法

Alex Theedom
Senior Java developer
Consultant

2018年 4月 19日

Spring Framework 5.0 は今後数年のうちに、開発者が Spring ベースのアプリケーションを理解して作成する方法を変えていくでしょう。Spring 5 では、そのコア構造に関数型プログラミング・モデルとリアクティブ・プログラミング・モデルを統合しています。さらに、主要な依存関係を更新し、レガシーの重荷から解放されてパフォーマンス・ブーストを達成しながらも、お馴染みの Spring Framework のルック・アンド・フィールを維持している仕組みを学んでください。

2017 年 8 月に一般公開版 (GA) としてリリースされた Spring 5 は、2013 年 12 月以来、初となる Spring Framework のメジャー・リリースです。Spring 5 には長年待ち望まれていた改善が加えられているだけでなく、リアクティブ宣言に記載されたリアクティブ・プログラミングの原則に基づく新しいプログラミング・パラダイムが採用されています。

今回のリリースは、長年にわたる Spring Framework の歴史の中で、群を抜いて画期的なものとなっています。それは、Spring 5 は Java 8 と JDK 9 との互換性を持ち、エンドポイントおよび Web アプリケーションの開発の革新的手法として、リアクティブ・ストリームを統合しているためです。

リアクティブ・プログラミングはまさにこのリリースのテーマであり、これまで多くの開発者を駆り立ててきた注目の機能です。変動する負荷に応じてシームレスにスケーリングする、回復力のある弾力的かつレスポンシブなサービスに対する要件は次第に増えてきていますが、リアクティブ・プログラミングはこの増大する要件にぴったり適合します。

この記事では Spring 5 の大まかな概要として、Java SE 8 および Java EE 7 API へのベースラインのアップグレード、Spring 5 の新たなリアクティブ・プログラミング・モデル、HTTP/2 のサポート、Spring での Kotlin を使用した完全な関数型プログラミングのサポートについて紹介します。また、テストとパフォーマンスの機能拡張についても簡単に触れ、最後に Spring のコアとコンテナーのリビジョンの概略を説明して記事を締めくくります。

Java SE 8 および Java EE 7 へのアップグレード

これまで、Spring Framework は非推奨となった Java リリースをサポートしてきましたが、Spring 5 はこのレガシーの重荷から解放されています。Spring Framework のコード・ベースは Java 8 の機能を利用するように改良されているため、最小 JDK バージョンとして Java 8 を使用する必要があります。

Spring 5 はクラス/パスだけでなく、モジュール・パスに関しても Java 9 と完全な互換性を持ち、JDK 9 のテスト・スイートを合格しています。つまり、Spring は Java 9 のリリースと同時に Java 9 で実行できるため、Java 9 のファンにとっては嬉しいニュースです。

API レベルでは、Spring 5 は Java EE テクノロジーとの互換性を持ち、Servlet 4.0、Bean Validation 2.0、そして新しく登場したばかりの JSON Binding API の要件を満たします。Java EE API の最小要件はバージョン 7 です。このバージョンで、Servlet、JPA、および Bean Validation API のマイナー・バージョン・リリースが導入されているためです。

リアクティブ・プログラミング・モデル:

Spring 5 で最も画期的な新しい特徴となっているのは、リアクティブ・プログラミング・モデルです。Spring 5 フレームワークはリアクティブな基盤の上に構築されていて、完全に非同期であり、ノンブロッキング型となっています。新しいイベント・ループ形式の実行モデルは、わずかな数のスレッドだけを使用して垂直スケーリングすることができます。

リアクティブ・コンポーネントのパイプライン内でバックプレッシャーを伝えるためのメカニズムを提供するために、Spring Framework ではリアクティブ・ストリームを導入しています。バックプレッシャーとは、複数のプロデューサーから送られてくるデータによってコンシューマーに過剰な負荷がかからないようにするための概念です。

Spring WebFlux と呼ばれる Spring 5 のリアクティブなコアでは、開発者がアノテーション・ベースのモデルと関数型 Web フレームワーク (`WebFlux.fn`) という Spring Web プログラミング用に設計された 2 つのプログラミング・モデルを使用できるようになっています。

アノテーション・ベースのモデルは、リアクティブな基盤の上に構築された、Spring WebMVC に代わる最新の手法です。一方、関数型 Web フレームワークは、`@Controller` アノテーション・ベースのプログラミング・モデルの代替手段として使用します。これらのモデルはいずれも、ノンブロッキング HTTP をリアクティブ・ストリーム API に適応させた、同じリアクティブなベース上で動作します。

アノテーションを使用したプログラミング

WebMVC プログラマーにとって、Spring 5 のアノテーション・ベースのプログラミング・モデルは非常に馴染み深いと感じられることでしょう。このプログラミング・モデルは、WebMVC の `@Controller` プログラミング・モデルを順応させたもので、同じアノテーションを採用しているためです。

リスト 1 に記載する `BookController` クラスは、HTTP リクエストに応答する 2 つのメソッドを示しています。一方のメソッドは、書籍のリストを取得するためのHTTP リクエストに応答し、もう

一方は指定の `ID` を持つ書籍を取得するための HTTP リクエストに応答します。それぞれのリソース・メソッドから返される `Mono`、`Flux` というオブジェクトに注目してください。これらのオブジェクトはリアクティブ型であり、リアクティブ・ストリーム仕様に従った `Publisher` インターフェースを実装します。Mono と Flux の役割は、いずれもデータのストリームを処理することです。`Mono` オブジェクトは 1 個の要素のストリームを処理し、Flux は N 個の要素のストリームを処理します。

リスト 1. リアクティブ・コントローラー

```
@RestController
public class BookController {

 @GetMapping("/book")
 Flux<Book> list() {
  return this.repository.findAll();
 }

 @GetMapping("/book/{id}")
 Mono<Book> findById(@PathVariable String id) {
  return this.repository.findOne(id);
 }

    // Plumbing code omitted for brevity
}
```

上記のアノテーションは、Spring Web プログラミングに使用するものです。次は、同じ問題を関数型 Web フレームワークを使って解決しましょう。

関数型プログラミング

Spring 5 の新しい関数型の手法では、リクエストをハンドラー関数に委任し、ハンドラー関数がサーバー・リクエストを受け入れてリアクティブな型を返します。この仕組みリスト 2 に示します。ここでは、`listBook` メソッドと `getBook` メソッドにリスト 1 の機能が反映されています。

リスト 2. BookHandler 関数クラス

```
public class BookHandler {

    public Mono<ServerResponse> listBooks(ServerRequest request) {
        return ServerResponse.ok()
            .contentType(APPLICATION_JSON)
            .body(repository.allPeople(), Book.class);
    }

    public Mono<ServerResponse> getBook(ServerRequest request) {
        return repository.getBook(request.pathVariable("id"))
            .then(book -> ServerResponse.ok()
            .contentType(APPLICATION_JSON)
            .body(fromObject(book)))
            .otherwiseIfEmpty(ServerResponse.notFound().build());
    }
    // Plumbing code omitted for brevity
}
```

クライアント・リクエストをハンドラーにルーティングするために使用するルーティング関数は、HTTP リクエストの述語とメディア・タイプを突き合わせて、該当するハンドラーにリクエス

トをルーティングします。リスト 3 に、呼び出しを該当するハンドラー関数に委任する、books リソースのエンドポイント URI を示します。

リスト 3. ルーティング関数

```
BookHandler handler = new BookHandler();

RouterFunction<ServerResponse> personRoute =
    route(
        GET("/books/{id}")
        .and(accept(APPLICATION_JSON)), handler::getBook)
        .andRoute(
    GET("/books")
        .and(accept(APPLICATION_JSON)), handler::listBooks);
```

以上の例を裏で支えているデータ・リポジトリーも、Spring Data のリアクティブな Couchbase、Reactive MongoDB、および Cassandra のサポートを使用して完全なリアクティブ・エクスペリエンスに対応します。

REST エンドポイントを使用したリアクティブ・スタイルのプログラミング

新しいプログラミング・モデルは、従来の Spring WebMVC モデルから脱却し、非常に便利な新機能をいくつか取り込んでいます。

そのような機能のうちの 1 つが、WebFlux モジュールです。このモジュールは、`RestTemplate` に代わる、`WebClient` というノンブロッキングかつリアクティブな手段を提供しています。リスト 4 では、`WebClient` を作成し、`books` エンドポイントを呼び出して指定の `ID 1234` を持つ書籍をリクエストしています。

リスト 4. WebClient を使用した REST エンドポイントの呼び出し

```
Mono<Book> book = WebClient.create("http://localhost:8080")
      .get()
      .url("/books/{id}", 1234)
      .accept(APPLICATION_JSON)
      .exchange(request)
      .then(response -> response.bodyToMono(Book.class));
```

HTTP/2 のサポート

> **HTTP/2 の内幕**: リッチな Web アプリケーションをさらに充実させることを目的に作成された HTTP/2 は、転送パフォーマンスを改善し、待ち時間を短縮し、アプリケーションのスループット向上を促します。この待望のアップグレードについて、このリンク先の私の記事を読んでください。

Spring Framework 5.0 には、専用の HTTP/2 機能のサポート、ならびに JDK 9 に期待されている新しい HTTP クライアントのサポートが付随しています。HTTP/2 のサーバー・プッシュ機能は、Spring 開発者がかなり以前から、Jetty エンジンの `ServerPushFilter` クラスという手段を介して利用することができました。けれども Spring 5 では HTTP/2 のパフォーマンス強化機能をそのまますぐに利用できるようになっていることを知れば、Web オプティマイザーは大喜びで飛び跳ねることでしょう。

Java EE Servlet 仕様にリリースは、2017 年の第 4 四半期になると見込まれています。それを機に、Spring 5.1 での Servlet 4.0 のサポートが有効になります。それまでは、HTTP/2 の機能は Tomcat 9.0、Jetty 9.3、Undertow 1.4 によってネイティブに提供されることになります。

Kotlin と Spring WebFlux

Kotlin は、JetBrains 社が開発した、関数型プログラミングに対応するオブジェクト指向の言語です。Kotlin の主な長所の 1 つとしては、Java との相互運用性に非常に優れていることが挙げられます。Spring のバージョン 5 では、この利点を全面的に利用するために、Kotlin 専用のサポートを導入しています。Kotlin の関数型プログラミング・スタイルは Spring WebFlux モジュールと抜群に相性が良く、しかも Kotlin の新しいルーティング DSL では簡潔でイディオムのようなコードを使用する関数型 Web フレームワークを活用しています。例えばエンドポイントのルーティングは、リスト 5 のように単純な形で表現できます。

リスト 5. Kotlin のルーティング DSL を使用したエンドポイントの定義

```
    @Bean
    fun apiRouter() = router {
        (accept(APPLICATION_JSON) and "/api").nest {
            "/book".nest {
                GET("/", bookHandler::findAll)
                GET("/{id}", bookHandler::findOne)
            }
   "/video".nest {
                GET("/", videoHandler::findAll)
                GET("/{genre}", videoHandler::findByGenre)
            }
  }
 }
```

Kotlin 1.1.4 以降を使用する場合は、Kotlin の不変クラスのサポートが追加されています。これらのクラスでは、オプション・パラメーターとそのデフォルト値、および完全なヌルセーフに設計された API を使用できるようになっています。

ラムダ式による Bean の登録

現在、従来の XML と JavaConfig を使用した方法ではなく、ラムダ式を使用して Spring Bean を登録できるようになっているため、Bean をサプライヤーとして効率的に登録できます。リスト 6 では、ラムダ式を使用して `Book` Bean を登録しています。

リスト 6. サプライヤーとしての Bean の登録

```
GenericApplicationContext context = new GenericApplicationContext();
context.registerBean(Book.class, () -> new
        Book(context.getBean(Author.class))
  );
```

最新の API に対する Spring WebMVC サポート

真新しい WebFlux モジュールには新しい、多数の画期的な機能が用意されていますが、Spring 5 は、引き続き Spring MVC を使用したいという開発者の要望にも応えられるようになっていま

す。Spring 5 のモデル・ビュー・コントローラー・フレームワークは更新されて、WebFlux や最新バージョンの Jackson 2.9 および Protobuf 3.0 と連動するようになっていて、新しい Java EE 8 JSON-Binding API のサポートも組み込まれています。

HTTP/2 機能の基礎となるサーバー実装に加え、Spring WebMVC は MVC コントローラー・メソッドの引数を介して Servlet 4.0 の `PushBuilder` もサポートします。さらに、WebMVC には Reactor 3.1 の `Flux` および `Mono` オブジェクトに加え、RxJava 1.3 および 2.1 のオブジェクトに対する完全なサポートも組み込まれているため、これらのオブジェクトは MVC コントローラー・メソッドからの戻り値として扱われるようになっています。このサポートで対象としているのは、Spring Data に新しく追加されたリアクティブ WebClient とリアクティブ・リポジトリーです。

JUnit 5 による条件付きテストと同時テスト

JUnit と Spring 5: 関数型パラダイムを全面的に採用している Spring 5 は、JUnit 5 とその新しい関数型テスト・スタイルをサポートしています。また、レガシー・コードで問題が起きないように、JUnit 4 との後方互換性も備わっています。

Spring 5 のテスト・スイートは、さまざまな形で拡張されていますが、そのうち最も注目に値するのは、JUnit 5 をサポートするようになっていることです。したがってこれからは、Java 8 に含まれている関数型プログラミングの機能を単体テストで利用できます。リスト 7 に一例を記載します。

リスト 7. Java 8 のストリームとラムダ式を全面的に活用した JUnit 5

```
@Test
void givenStreamOfInts_SumShouldBeMoreThanFive() {
    assertTrue(Stream.of(20, 40, 50)
      .stream()
      .mapToInt(i -> i)
      .sum() > 110, () -> "Total should be more than 100");
}
```

JUnit 5 への移行: JUnit 5 にアップグレードするかどうか決めかねているとしたら、Steve Perry が作成した全 2 回からなる詳細なチュートリアルを読んでください。思い切ってアップグレードする決心がつくはずです。

Spring 5 では JUnit 5 の柔軟性を活用して、Spring TestContext Framework 内に複数の拡張 API を実装しています。その一例として、開発者は条件付きテストを実行するための JUnit 5 のアノテーション `@EnabledIf` および `@DisabledIf` を使用することで、SpEL (Spring Expression Language) 式を自動的に評価し、必要に応じてテストを有効/無効にすることができます。これらのアノテーションにより、Spring 5 では以前は実現するのが難しかった高度な条件付きテスト・シナリオをサポートできるようになっています。また、Spring TextContext Framework では同時テストを実行することもできるようになりました。

Spring WebFlux へのテストの統合

Spring Test に新しく追加された `WebTestClient` により、Spring WebFlux サーバー・エンドポイントのテストを統合できるようになっています。`WebTestClient` はサーバーの起動を回避するためにモック・リクエストとモック・レスポンスを使用するため、WebFlux サーバー・インフラストラクチャーに直接バインドすることができます。

`WebTestClient` は実際のサーバーにバインドすることも、コントローラーまたは関数と連動させることもできます。リスト 8 では、`WebTestClient` がローカル・ホストにバインドされています。

リスト 8. ローカル・ホストにバインドされた `WebTestClient`

```
WebTestClient testClient = WebTestClient
  .bindToServer()
  .baseUrl("http://localhost:8080")
  .build();
```

リスト 9 では、`RouterFunction` をテストします。

リスト 9. `RouterFunction` にバインドされた `WebTestClient`

```
RouterFunction bookRouter = RouterFunctions.route(
  RequestPredicates.GET("/books"),
  request -> ServerResponse.ok().build()
);

WebTestClient
  .bindToRouterFunction(bookRouter)
  .build().get().uri("/books")
  .exchange()
  .expectStatus().isOk()
  .expectBody().isEmpty();
```

パッケージのクレンジングと廃止

Spring 5 ではいくつかの古くなった API のサポートを廃止しています。廃止の対象となったのは、後継の Hibernate 5 に取って代わられた Hibernate 3 および 4 です。また、Portlet、Velocity、JasperReports、XMLBeans、JDO、Guava のサポートも廃止されました。

このような大掃除はパッケージ・レベルでも行われています。具体的には、Spring 5 では `beans.factory.access`、`jdbc.support.nativejdbc`、`mock.staticmock` (spring-aspects モジュールの一部)、`web.view.tiles2M` をサポートしなくなりました。現在、Spring の最小要件となっているのは Tiles 3 です。

Spring のコアとコンテナーに対して行われた一般的な更新

Spring Framework 5 ではコンポーネントをスキャンして識別する方法が改善されていることから、大規模なプロジェクトのパフォーマンスが向上します。現在、スキャンはコンパイル時に行われ、コンポーネント候補が META-INF/spring.components ファイル内で指定されているインデックス・ファイルに追加されます。インデックスは、プロジェクトに対して定義されているプラットフォーム固有のアプリケーション・ビルド・タスクによって生成されます。

`@Index` アノテーションでマークされたクラスやインターフェースと併せて、javax パッケージのアノテーションでマークされたコンポーネントがインデックスに追加されます。Spring の従来のクラス・パス・スキャンは行われなくなっていますが、フォールバック手段として残されています。大規模なコード・ベースでは顕著にパフォーマンスが向上する一方、多数の Spring をホストするサーバーでは起動時間の短縮が認められるはずです。

Spring 5 では `@Nullable` のサポートも追加されていて、このアノテーションを使用して任意の注入ポイントを指定できるようになっています。これで、コンシューマーは null 値を受け入れられるようになるはずです。さらに、このアノテーションは、null になり得る引数、フィールド、戻り値をマークするためにも使用できます。主に IntelliJ IDEA のような IDE で使用するように意図されていますが、Eclipse や FindBugs でも、実行時に `NullPointerException` を返すのではなく、コンパイル時に null 値を処理するのに役立ちます。

Spring ロギングも強化されていて、独自の Commons Logging ブリッジを追加の設定なしですぐに使用できるようになっています。また、リソース抽象化によって `getFile` アクセスに対して `isFile` インジケーターを指定することにより、防御的プログラミングもサポートされるようになりました。

まとめ

Spring 5 で注目の機能となっている新しいリアクティブ・プログラミング・モデルは、シームレスにスケーリングするレスポンシブな Spring ベースのサービスを実現することへの大きなコミットメントを表しています。Spring 5 の導入により、開発者は、Java での Web およびエンタープライズ・アプリケーション開発へと導く軌道として、リアクティブ・プログラミングが定着しつつあることを実感できるはずです。

今後の Spring Framework リリースでもこのコミットメントを引き続き反映して、Spring Security、Spring Data、Spring Integration でリアクティブ・プログラミングの特性と長所を採用することが見込まれています。

全体的に見て、Spring 5 は Spring ベースの開発者にとって歓迎すべきパラダイム・シフトであり、他のフレームワークが従うべき道を見事に指し示していると言えます。

関連トピック： Spring 5 WebFlux: Performance tests Introducing Kotlin support in Spring Framework 5.0 Reactive Spring presentation by Josh Long Spring Boot の基礎 Introducing JUnit 5 (全 2 回からなるチュートリアル) HTTP/2 の内幕

著者について

Alex Theedom

May 2017