УДК 004

Использование Mono и Flux

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье будут рассмотрены принципы работы с Mono и Flux методами. Работа будет происходить в среде разработки IntelijIdea.
Ключевые слова: Mono, Flux, Spring, Spring boot

Using Mono and Flux

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article will cover the principles of working with Mono and Flux methods. The work will take place in the IntelijIdea development environment.
Keywords: Mono, Flux, Spring, Spring boot

Project Reactor — это платформа с открытым исходным кодом от Pivotal, которая выступает в качестве основы для реактивных приложений. В настоящее время он основан на инициативе реактивных потоков, инициативе, основанной инженерами Netflix, Pivotal и Lightbend при участии других крупных разработчиков Java, таких как Oracle и Red Hat.

В рамках этой инициативы была создана спецификация реактивного потока с несколькими ключевыми факторами, такими как:

- Реактивный поток должен быть неблокирующим,
- Это должен быть поток данных,
- Он должен работать асинхронно,
- И он должен выдерживать обратное давление.

Цель данной статьи рассмотреть принципы работы с Mono и Flux.

А.А. Голикова, В.А. Строганов в своей работе описывают систему, разработанную для пользователей новостных порталов в сети Интернет [1]. Д. Ю. Балакишиев рассказывает в своей статье о написании приложения с использованием технологий Java 8, Spring Boot, Angular 2 [2]. Р.И. Ибраимов и др.. в статье описывают процесс разработки генеалогического дерева на языке Java с использованием фреймворка Spring и библиотеки gedcom4j [3]. В.И. Зарайский описывает разработку модуля автоматизации работы с конференциями в кафедральном приложении [4]. Р.И. Ибраимов и др. в своей работе описывают информацию для веб-разработчиков, целью которых является улучшение написанного ими кода путём его проверки и тестирования [5].

Первый тип издателя — это Mono. MonoAPI позволяет получать только одно значение, после чего оно будет немедленно завершено. Это означает, что Mono это противоположность возврата простого объекта или Optional.

Имеется следующий код (рис.1).

```
public Person findCurrentUser() {
    if (isAuthenticated()) return new Person("Jane", "Doe");
    else return null;
}
```

Рисунок 1 - Пример кода

В Java 8 можно было бы переписать это так (рис.2).

```
public Optional<Person> findCurrentUser() {
    if (isAuthenticated()) return Optional.of(new Person("Jane", "Doe"));
    else return Optional.empty();
}
```

Рисунок 2 – Пример на Java 8

С другой стороны, если использовать реактивные потоки, можно было бы использовать следующий код (рис.3).

```
public Mono<Person> findCurrentUser() {
    if (isAuthenticated()) return Mono.just(new Person("Jane", "Doe"));
    else return Mono.empty();
}
```

Рисунок 3 – Пример на потоках

Когда Mono используется для обработки нуля или одного результата, Flux используется для обработки от нуля до многих результатов, возможно,

даже до бесконечных результатов. Можно рассматривать это как часть коллекции, массива или потока с реактивным счетчиком (рис.4).

```
public List<Person> findAll() {
    return Arrays.asList(
        new Person("Jane", "Doe"),
        new Person("John", "Doe")
    );
}
```

Рисунок 4 – Пример кода

В Java 8 можно было бы переписать это, используя потоки (рис.5).

```
public Stream<Person> findAll() {
    return Stream.of(
        new Person("Jane", "Doe"),
        new Person("John", "Doe")
    );
}
```

Рисунок 5 – Использование Java 8

А с реактивными потоками можно переписать это как (рис.6).

```
public Flux<Person> findAll() {
    return Flux.just(
        new Person("Jane", "Doe"),
        new Person("John", "Doe")
    );
}
```

Рисунок 6 – Использование потоков

Концепции реактивных потоков частично совпадают с концепциями, которые были в функциональном программировании.

Предположим следующий код, который будет напечатан на консоли (рис.7).

```
Flux
    .just(1, 2, 3, 4)
    .reduce(Integer::sum)
    .log();
```

Рисунок 7 – Пример кода

Реактивные потоки используют модель push. Это означает, что элементы помещаются в поток со скоростью издателя, независимо от того, может ли подписчик следовать или нет.

По этой логике можно было бы подумать, что реактивный поток в предыдущем примере будет выдавать 10. Однако реактивные потоки не запускаются, пока нет подписчика.

Дополним некоторыми строками кода рисунок выше (рис.8).

```
AtomicInteger sum = new AtomicInteger(0);
Flux
    .just(1, 2, 3, 4)
    .reduce(Integer::sum)
    .subscribe(sum::set);
log.info("Sum is: {}", sum.get());
```

Рисунок 8 – Пример кода

В данном примере сложно понять, что выведет программа в консоль. Так как поток обрывается асинхронно, то сумма чисел может быть и не вычислена, а может и наоборот сделаться подсчет чисел. Ответ на этот вопрос заключается в том, что он напечатает 10 на консоли, потому что Flux.just() по умолчанию использует текущий поток, и, таким образом, результат был вычислен, когда он достигнет оператора регистрации.

Дополним еще пример некоторыми командами (рис.9).

```
AtomicInteger sum = new AtomicInteger(0);
Flux
    .just(1, 2, 3, 4)
    .subscribeOn(Schedulers.elastic())
    .reduce(Integer::sum)
    .subscribe(sum::set);
logger.info("Sum is: {}", sum.get());
```

Рисунок 9 – Пример кода

В этом случае ответом будет 0, потому что при использовании subscribeOn() подписка будет выполняться в другом рабочем потоке планировщика, что сделает его асинхронным.

Таким образом, в зависимости от характера реактивного потока он будет либо синхронным, либо асинхронным.

По умолчанию потоки в реакторе проекта холодные. Можно продемонстрировать это, используя следующий код (рис.10).

```
Flux<Integer> numbers = Flux
    .just(1, 2, 3, 4)
    .log();
numbers
    .reduce(Integer::sum)
    .subscribe(sum -> logger.info("Sum is: {}", sum));
numbers
    .reduce((a, b) -> a * b)
    .subscribe(product -> logger.info("Product is: {}", product));
```

Рисунок 10 – Пример кода

В этом примере числа от 1 до 4 передаются дважды: один раз для первого подписчика и еще раз для второго подписчика.

Однако в некоторых случаях, например при выполнении HTTP-запроса, пользователь не хочет снова начинать с источника и хочет превратить свой холодный поток в горячий. В Project Reactor можно сделать это, поделившись потоком (рис.11).

```
Flux<Integer> numbers = Flux
    .just(1, 2, 3, 4)
    .log()
    .share();
numbers
    .reduce(Integer::sum)
    .subscribe(sum -> logger.info("Sum is: {}", sum));
numbers
    .reduce((a, b) -> a * b)
    .subscribe(product -> logger.info("Product is: {}", product));
```

Рисунок 11 – Пример кода

При этом значения от 1 до 4 передаются только один раз, и один и тот же источник используется обоими подписчиками.

У Mono нет share() метода, поэтому необходимо использовать cache() вместо него. Это превращает его во что-то похожее на BehaviorSubjectRxJS, где значение кэшируется и отправляется для каждого нового подписчика.

В данной работе был произведен анализ Mono и Flux методы. Рассмотрены примеры использования.

Библиографический список

1. Голикова А.А., Строганов В.А. Система обработки и расширения данных

новостной ленты // Теоретические основы программирования. Учебное пособие 2019. 137с.
2. Балакишиев Д. Ю. Использование java 8, spring boot и angular 2+ для разработки веб-приложения «справочник услуг» // Образовательные технологии и общество. 2019. №3. С. 145-154.
3. Ибраимов Р.И., Зайчик А.Р., Минзатов Н.С. Разработка генеалогического дерева средствами фреймвока spring boot // Молодость. Интеллект. Инициатива. 2017. С. 21-22.
4. Зарайский В.И. Разработка модуля автоматизации работы с конференциями в кафедральном приложении // Информатика, управляющие системы, математическое и компьютерное моделирование 2020. №3. С. 549-462.
5. Ибраимов Р.И., Джемалетдинов А.Б., Шевченко А.А. Spring boot: создание тестов для spring mvc контроллеров // Прикладная математика и фундаментальная информатика. 2020. №2. С. 23-28.