УДК 004.043

Кривуляк В. В., Android-розробник, team leader

РЕАКТИВНЕ ПРОГРАМУВАННЯ В ANDROID

Компанія PowerCode

Чому сучасний розробник повинен знати про реактивне програмування? Якщо у структурі програми зустрічається хоча б один асинхронний виклик, то імперативний стиль програмування дає збій й підвищує складність програми, й зводить нанівець переваги. Значна частина світу Android за природою асинхронна, починаючи від методів життєвого циклу Activity, Service, й закінчуючи роботою з сенсорами, файловою системою, мережними запитами. Припустимо, необхідно синхронізувати зміни профіля користувача на сервері, й лишень після успішного виконання оновити дані на екрані. Одразу постає задача відправлення й отримання відповіді, обробка помилок, зокрема підключення до мережі. Одне з рішень оптимістичне й полягає у миттєвому внесені локальних змін не чекаючи виконання запиту, але тоді необхідно відкочувати локальні зміни. Інше рішення це застосувати `Runnable`, що означатиме *асинхронний* виклик, й за успішного виконання оновити дані представлення локально. Фактично це є *реактивна поведінка* – зробити щось, як реакцію на подію, наприклад, виконання мережного запиту. Одначе, для можливості повідомити про результат, необхідно передавати інтерфейс `Listener`:

```
UserManager um = new UserManager();

um.setName("Jane Doe", new UserManager.Listener() {
    @Override public void success() {
        System.out.println(um.getUser());
    }

    @Override public void failure(IOException e) {
        // TODO show the error...
    }
});
```

Такий підхід швидко ускладнює код, якщо необхідно зробити декілька асинхронних викликів у певній послідовності при цьому враховувати життєвий циклу активності чи фрагмента, якими розробник не керує й на додачу пам’ятати у якому потоці будуть викликані методи `success, failure`.

У Android користувачів можна розглядати як асинхронні джерела подій, що взаємодіють з UI. Замість того, щоб координувати асинхронні елементи архітектури, можна поєднати їх напряму. Зокрема підписати UI на зміни у ViewModel, яка підписана на зміни у Gateway. Одним з таких «поєднувачів» може бути RxJava версії 2. [1]

RxJava надає розробникові:

- набір класів для представлення джерел даних;
- набір класів для прослуховування джерел;
- набір методів й класів для комбінування даних (оператори).

У реактивному програмуванні під Android типові задачі, як-то мережні запитий до БД, зчитування даних сенсорів, геолокації, жести користувача слід розглядати, як джерела, що постачають потоки даних. Джерело може працювати синхронно чи асинхронно, повертати один, декілька чи жодного елемента. Запис у БД чи у файл операції, зазвичай, не повертає елементів, а закінчується успіхом чи відмовою. У RxJava це моделюється за допомогою подій `onComplete()`, `onError()`. Ланцюжок подій може бути потенційно нескінченним, як в UI. Видно, що це ніщо інше як шаблон проектування `Observer`.

У RxJava 2 джерела представлені двома основними типами `Flowable` й `Observable`. Різниця у тому, що перший підтримує Backpressure, що дозволяє інформувати джерело

про необхідність уповільнення генерації елементів, якщо слухач не встигає їх обробляти [2].

```
interface Disposable {
    void dispose();
}interface Subscription {
    void cancel();
    void request(long r);
}
```

Відзначимо, що `Disposable` та `Subscription` різні, оскільки є результатами роботи компаній, що напрацювали стандартний набір інтерфейсів для бібліотек RxJava, які увійшли до специфікації RxJava 2. Спеціалізовані джерела: `Single`, може видати один елемент або кинути виняток, `Completable` – завершується успішно без повернення значення, або генерує виняток й `Maybe` – реактивний аналог `Optional` у Java. Ці реактивні потоки не підтримують `backpressure`, для призначений `Flowable`. Для зручного створення існує набір фабричних методів `create()`, `just()`, `fromCallable()`, `fromArray()`, та ін. Переробимо код на початку у реактивний стиль [1]:

```
interface UserManager {
    Single<Resource<String, Exception>> setName(String name);
    …
}

//ViewModel
void setUserName(String name) {
    mComposideDisposable.add(

userManager.setName(name).observeOn(AndroidSchedulers.mainThread())
            .subscribe(res -> {//handle success}, throwable -> {//
error…})
    );
}
```

Оператори дозволяють вирішувати задачі:

- модифікацію даних й їх комбінування;
- маніпулювати потоками виконання й повернення результату;
- обробляти події, що генеруються.

У наведеному фрагменті використано оператор `observeOn` щоб результат повернувся у головний потік з якого можна оновлювати компоненти UI. Бібліотека RxJava має величезний набір операторів призначених для вирішення певного кола задач [3].

Висновки:

- RxJava – це потужний інструмент, що дозволяє поєднувати асинхронні частини коду, оброблювати результат, не перейматися за поточний стан оточення, разом з тим вимагає реактивного мислення.
- Опанування правильного використання RxJava доволі складне й тривале.

Література

1. Jake Wharton, [Web – resource], Exploring RxJava 2 for Android – access https://youtu.be/htIXKI5gOQU GOTO 2016 open access.
2. David Karnok, [Web – resource] RxJava backpressure – access. https://github.com/ReactiveX/RxJava/wiki/Backpressure open access, June 27, 2016.
3. T. Nield [e-book], Learning RxJava, Packt Publishing, 2017, 400 c.