ToiletPaper #125

Best Practice RxJava2 Testing

Autor: Alexander Pöhlmann / Software Engineer / Standort Leipzig

✗ Problem

Wie teste ich richtig mit RxJava? Diese Frage stellt sich vielen Entwicklern, denn es ist gar nicht so einfach, einen asynchronen Ablauf zu testen. Oft ist der andere Thread langsamer oder schneller als erwartet. Dies führt dazu, dass beim Schreiben von Tests mit den üblichen JUnit-Funktionalitäten die Tests oft zu lange angehalten werden oder ab und zu fehlschlagen.
Ein weiteres Problem ist, dass manche Tests bei normaler Durchführung häufig sehr lange dauern. Meist werden aus diesem Grund Tests nicht oder unvollständig geschrieben.

✓ Lösung

Um dies zu vermeiden, gibt es von RxJava einige Möglichkeiten, diese Abläufe optimal und effizient zu testen. Hierbei müsst ihr beachten, dass sich das Naming von RxJava1 auf RxJava2 (vielleicht auch RxJava3) verändert hat. Aber die grundlegende Logik ist die gleiche.
Zum einen gibt es den ***TestObserver*** und den ***TestSubscriber***, welche sich auf die Events von der jeweiligen ***ObservableSource*** registrieren. Diese besitzen verschiedene **Assert**-Methoden, um die empfangenden Werte zu prüfen. Aber auch die **Await**-Methoden sind sehr wichtig. Mit diesen können die Tests angehalten werden und erst mit Erfüllen der Bedingungen oder des Timeout wird der Test fortgesetzt.
Für viele Abläufe genügen die **Await**-Methoden, aber manchmal müssen Zeiträume über mehrere hundert Millisekunden oder mehr überwunden werden. Hier ist es sehr impraktikabel die ganze Laufzeit abzuwarten, was sich ab einer bestimmten Zeitdauer gravierend auf die Laufzeit der Tests auswirken würde.
Hierfür gibt es in RxJava2 den ***Testscheduler***. Mit diesem ist es möglich, die RxJava-Operationen vorzuspulen. Dabei ist zu beachten, dass der ***Testscheduler*** nicht von alleine arbeitet und nur beim „Vorspulen“ die Abläufe fortgesetzt werden.

➔ Tools und Begrifflichkeiten

- TestSubscriber
  - Ermöglicht die Untersuchung der Events mit den verschiedenen **assert***-Methoden.
  - Wird durch die Methode ***Flowable***::test erzeugt oder klassisch über ***Flowable***::subscribe registriert.
- TestObserver
  - Ermöglicht die Untersuchung der Events mit den verschiedenen **assert***-Methoden.
  - Wird durch die Methode ***Observable/Single/Completable***::test erzeugt oder klassisch über ***Observable/Single/Completable***::subscribe registriert.
- TestScheduler
  - ***TestScheduler*** ist in der Lage die Zeit zu manipulieren, in dem dieser bestimmte Zeitintervalle vorspulen kann.
  - Sehr praktisch bei Prozessen, die nur an bestimmten Zeitpunkten angestoßen werden.
- await*-Methoden
  - Test-Thread wird angehalten, bis die jeweilige Bedingung erfüllt wurde.
  - Es gibt verschiedene Varianten von **await***-Methoden:
    - **await**() – Hält den Thread für eine bestimmte Zeitdauer an (Ist das Gleiche wie Thread::sleep).
    - **awaitCount**() – Wartet bis die Anzahl der Events erreicht wird.
    - **awaitDone**()/**awaitTerminalEvent**() – Wartet bis die Source abgeschlossen wurde (auch bei Fehlern).
- assert*-Methoden
  - Verschiedene Methoden zum Überprüfen der empfangenen Werte.
  - Verhält sich ähnlich wie mit den Prüfmethoden von der org.junit.Assert Klasse.
  - Beispiele: **assertComplete**(), **assertEmpty**(), **assertError**(), **assertNever**(), **assertNoErrors**(), **assertNoValues**().

+ Weiterführende Aspekte

- https://www.baeldung.com/rxjava-testing
- https://medium.com/@vanniktech/testing-rxjava-code-made-easy-4cc32450fc9a