Observables

Updated 27 Mar 2024

Info

Definition

- asynchronous
- emitting a sequence
- produce events over a period of time
- events contain values

Lifecycle

- starts emitting when subscribed to
- terminated when completed event or error event
- manually terminates by canceling a subscription using `dispose()`

Traits

observables with a narrower set of behaviors; helps to clearly convey intent to readers

- **Singles**
  - emit either a success(value) or error(error) event
  - useful for one-time processes (eg dowloading data, uploading from disk, ...)
  - subscribe to any observable and use `.asSingle()` to convert it
- **Completable**
  - emit a completed or error(error) event
  - convert an observable sequence to a completable by using the `ignoreElements()` operator
  - when you only care that an operation completed successfully or failed, such as a file write
  - create a completable sequence by using `Completable.create { ... }`
- **Maybe**
  - either emit a success(value), completed or error(error)
  - create a Maybe directly by using Maybe.create({ ... }) or by converting any observable sequence via `.asMaybe()`

Subjects

act as both observables and observers

- PublishSubject: starts empty and only emits new elements to subscribers
  - notify of new events from the point at which subscribers subscribed
  - if it receives a stop event (eg completed or error)
    - emits a stop event and stops emitting
    - re-emits a stop event to new subscriber
- BehaviorSubject: starts with an initial value and replays it or the latest element to new subscribers
  - similar as PublishSubjects
  - replay the latest next events to new subscribers
  - need a default value at creation time
- ReplaySubject: initialized with a buffer size and will maintain a buffer of elements up to that size and replay it to new subscribers
  - will temporarily cache, or buffer, the latest elements they emit, up to a specified size
  - will then replay that buffer to new subscribers
- AsyncSubject: emits only the last next event in the sequence, and only when the subject receives a completed event

RxSwift also provides a concept called Relays

PublishRelay, BehaviorRelay

- wrap their respective subjects, but only `accept`
- garantee to never terminate
- add values onto it by using `accept` (error or completed values are not allowed)
- you can ask it for its current value at any time
- bridges the imperative and reactive worlds