Table of contents

Introduction

TBD.

API signature changes

as() and to() operators

In 2.x, the `to()` operator used the generic `Function` to allow assembly-time conversion of flows into arbitrary types. The drawback of this approach was that each base reactive type had the same `Function` interface in their method signature, thus it was impossible to implement multiple converters for different reactive types within the same class. To work around this issue, the `as` operator and `XConverter` interfaces have been introduced in 2.x, which interfaces are distinct and can be implemented on the same class. Changing the signature of `to` in 2.x was not possible due to the pledged binary compatibility of the library.

From 3.x, the `as()` methods have been removed and the `to()` methods now each work with their respective `XConverer` interfaces:

- `Flowable.to(Function<Flowable<T>, R>)` is now `Flowable.to(FlowableConverter<T, R>)`
- `Observable.to(Function<Observable<T>, R>)` is now `Observable.to(ObservableConverter<T, R>)`
- `Maybe.to(Function<Flowable<T>, R>)` is now `Maybe.to(MaybeConverter<T, R>)`
- `Single.to(Function<Flowable<T>, R>)` is now `Maybe.to(SingleConverter<T, R>)`
- `Completable.to(Function<Completable, R>)` is now `Completable.to(CompletableConverter<R>)`
- `ParallelFlowable.to(Function<ParallelFlowable<T>, R)` is now `ParallelFlowable.to(ParallelFlowableConverter<T, R>)`

If one was using these methods with a lambda expression, only a recompilation is needed:

```
// before
source.to(flowable -> flowable.blockingFirst());

// after
source.to(flowable -> flowable.blockingFirst());
```

If one was implementing a Function interface (typically anonymously), the interface type, type arguments and the `throws` clause have to be adjusted

```
// before
source.to(new Function<Flowable<Integer>, Integer>() {
    @Override
    public Integer apply(Flowable<Integer> t) throws Exception {
        return t.blockingFirst();
    }
});

// after
```

```
source.to(new FlowableConverter<Integer, Integer>() {
    @Override
    public Integer apply(Flowable<Integer> t) {
        return t.blockingFirst();
    }
});
```

TBD.

- some operators returning a more appropriate Single or Maybe
- functional interfaces throws widening to Throwable
- standard methods removed
- standard methods signature changes

Standardized operators

(former experimental and beta operators from 2.x)

TBD.

Operator behavior changes

TBD.

- connectable sources lifecycle-fixes

Test support changes

TBD.

- methods removed from the test consumers