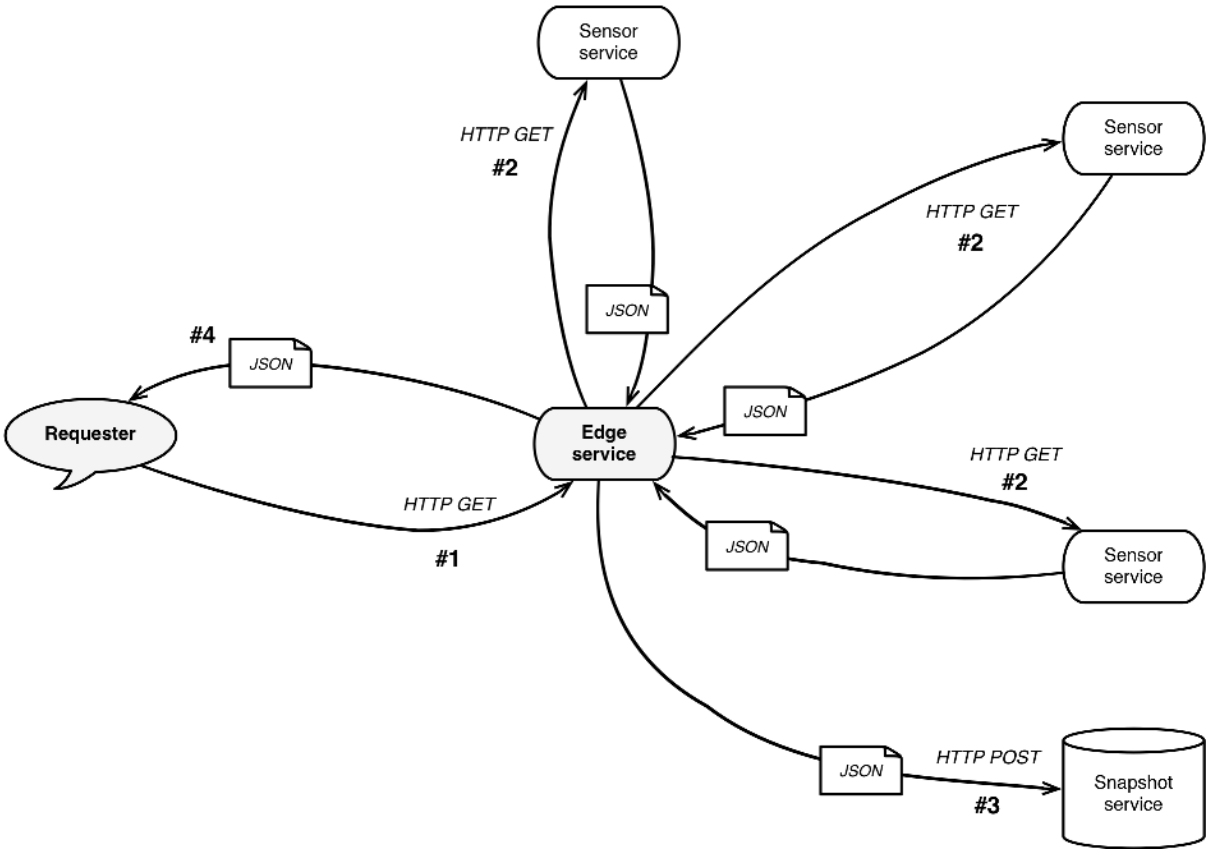
5. Beyond callbacks

Callbacks and their limitations, with a gateway/edge service example

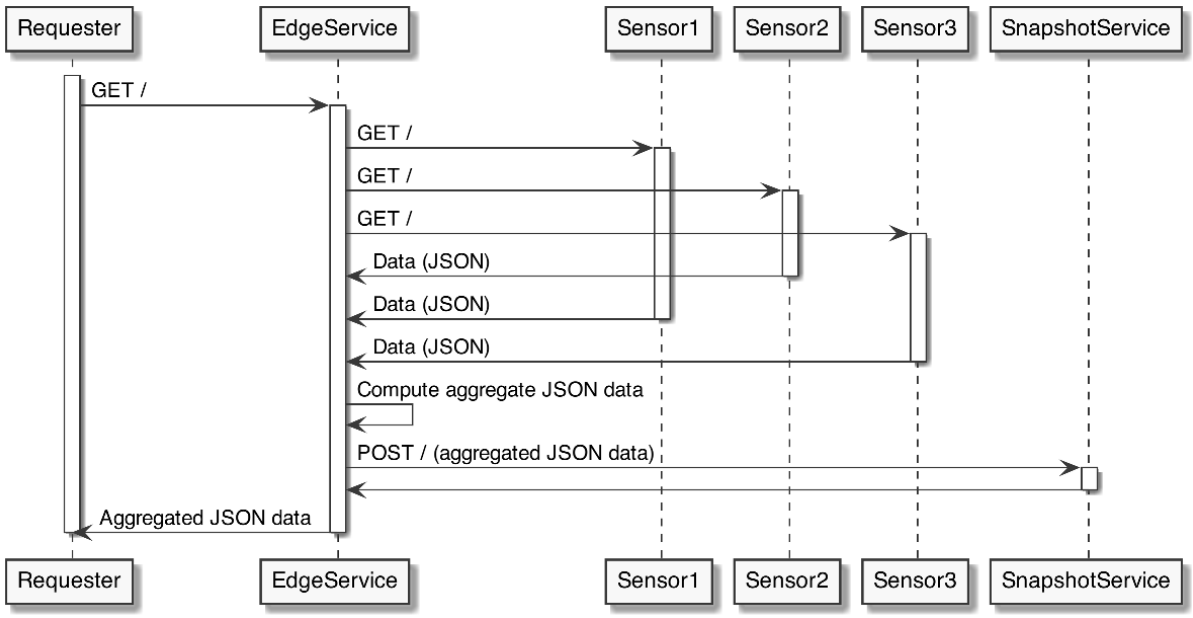
While callbacks are a simple form of asynchronous event notification, they can easily render asynchronous code complicated.

E.g : Composing asynchronous operations: the edge service example

Edge service scenario



Interactions between the edge, sensor, and snapshot services



This example allows us to reason about parallel and sequential operations:

- Parallel asynchronous operations: fetching heat sensor data
- Sequential asynchronous operations: aggregating heat sensor data, sending it to the snapshot service, and then returning it to the requester

---

The “callback hell”

Callback hell is when nested callbacks are being used to chain asynchronous operations, resulting in code that is harder to understand, due to the deep nesting. Error handling is especially more difficult with nested callbacks.

```
function iIds(startAt, showSessionRoot, iNewNmVal, endActionsVal, iStringVal, seqProp, htmlEncodeRegEx) {
    if (SbUtil.dateDisplayType === 'relative') {
        iRange();
    } else {
        iSelActionType();
    }
    iStringVal = notifyWindowTab;
    startAt = addSessionConfigs.sbRange();
    showSessionRoot = addSessionConfigs.elHiddenVal();
    var headerDataPrevious = function(tabArray, iNm) {
        iPredicateVal.SBDB.deferCurrentSessionNotifyVal(function(evalOutMatchedTabUrlsVal) {
            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                iPredicateVal.SBDB.normalizeTabList(function(appMsg) {
                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                        iPredicateVal.SBDB.detailTxt(function(evalOrientationVal) {
                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                iPredicateVal.SBDB.neutralizeWindowFocus(function(iTokenAddedCallback) {
                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                        iPredicateVal.SBDB.evalSessionConfig2(function(sessionNm) {
                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                iPredicateVal.SBDB.iWindow2TabIdx(function(iURLsStringVal) {
                                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                        iPredicateVal.SBDB.idx7Val(undefined, iStringVal, function(getWindowIndex) {
                                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                addTabList(getWindowIndex.rows, iStringVal, showSessionRoot && showSessionRoot.length > 0 ? showS
                                                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                        evalSAllowLogging(tabArray, iStringVal, showSessionRoot && showSessionRoot.length > 0 ?
                                                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                                BrowserAPI.getAllWindowsAndTabs(function(iSession1Val) {
                                                                                    if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                                        SbUtil.currentSessionSrc(iSession1Val, undefined, function(initCurrentSe
                                                                                            if (!htmlEncodeRegEx || htmlEncodeRegEx == iContextTo) {
                                                                                                addSessionConfigs.render(matchText(iSession1Val, iStringVal, eva
                                                                                                    id: -13,
                                                                                                    unfilteredWindowCount: initCurrentSessionCache,
                                                                                                    filteredWindowCount: iCtrl,
                                                                                                    unfilteredTabCount: parseTabConfig,
                                                                                                    filteredTabCount: evalRegisterValue5Val
                                                                                                }] : [], cacheSessionWindow, evalRateActionQualifier, undefined,
                                                                                                    if (seqProp) {
                                                                                                        seqProp();
                                                                                                    }
                                                                                                });
                                                                                            }
                                                                                        });
                                                                                    }
                                                                                });
                                                                            }
                                                                        });
                                                                    }
                                                                }, showSessionRoot && showSessionRoot.length > 0 ? showSessionRoot : startAt ? [startAt] : []);
                                                            }
                                                        });
                                                    }
                                                });
                                            }
                                        });
                                    }
                                });
                            }
                        });
                    }
                });
            }
        });
    };
```

Futures and promises Api

a **promise** is used to write an eventual value, and a **future** is used to read it when it is available.

- Future-based APIs in Vert.x 4
- Interoperability with CompletionStage APIs

Reactive extensions

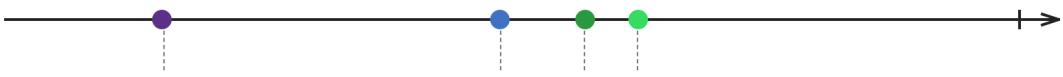
📌 Reactive extensions are an elaborated form of the observable/listener design pattern

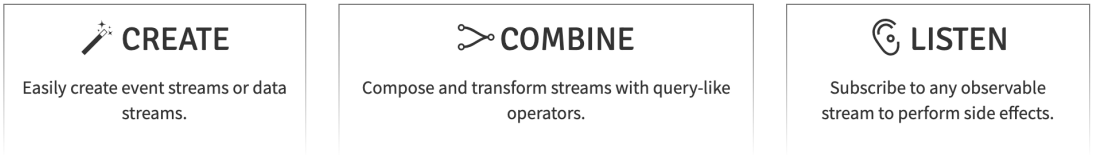
📌 They were first popularized by Erik Meijer in the Microsoft .Net ecosystem.

📌 Modern applications are increasingly composed of asynchronous event streams, not just on the server, but also in web, desktop, and mobile clients.

The Observer pattern done right

ReactiveX is a combination of the best ideas from
the Observer pattern, the Iterator pattern, and functional programming





Reactive extensions are about three things:

- Observing event or data streams (e.g., an incoming HTTP request can be observed)
- Composing operators to transform streams (e.g., merge multiple HTTP request streams as one)
- Subscribing to streams and reacting to events and errors

The ReactiveX initiative offers a common API and implementations in many languages, both for backend and frontend projects (reactivex.io/).

📌 The RxJS project offers reactive extensions for JavaScript applications in the browser,

📌 RxJava offers a general-purpose reactive extensions implementation for the Java ecosystem.

📌 Vert.x offers bindings for RxJava versions 1 and 2. Using version 2 is recommended because it supports back-pressure, while version 1 does not.

RxJava in a nutshell

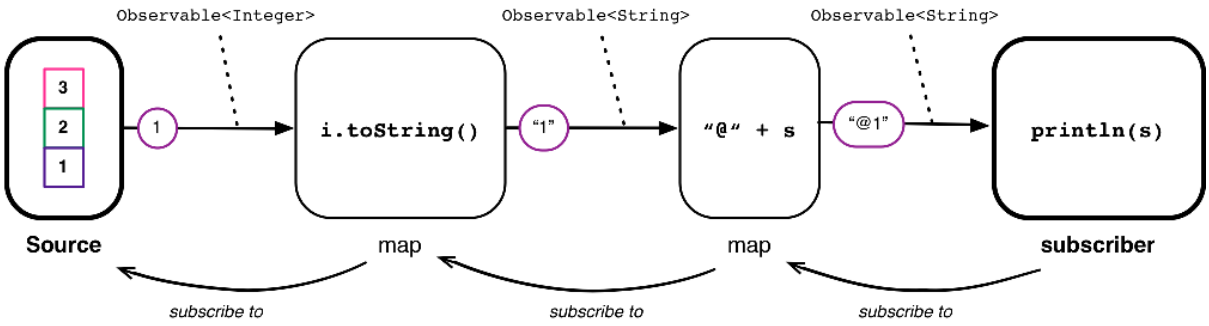
Observable types in RxJava

Aa Type	Description	Example
Observable<T>	A stream of events of type T. Does not support back-pressure	Timer events, observable source where we cannot apply back-pressure like GUI events
Flowable<T>	A stream of events of type T where back-pressure can be applied	Network data, filesystem inputs
Single<T>	A source that emits exactly one event of type T	Fetching an entry from a data store by key
Maybe<T>	A source that may emit one event of type T, or none	Fetching an entry from a data store by key, but the key may not exist
Completable	A source that notifies of some action having completed, but no value is being given	Deleting files

---

Basic examples

RxJava pipeline of listing




```
Observable.just(1, 2, 3)
  .map(Object::toString)
  .map(s -> "@" + s)
  .subscribe(System.out::println);
```

RxMarbles: Interactive diagrams of Rx Observables

Learn, build, and test Rx functions on Observables

https://rxmarbles.com/

Summary :

- Callbacks have expressiveness limitations when it comes to composing asynchronous operations, and they can render code harder to comprehend without proper care.
- Parallel and sequential asynchronous operations can be composed with other asynchronous programming models: futures and promises, reactive extensions
- Reactive extensions have a rich set of composable operators, and they are especially well suited for event streams.
- Futures and promises are great for simple chaining of asynchronous operations.
