

5 Beyond callbacks

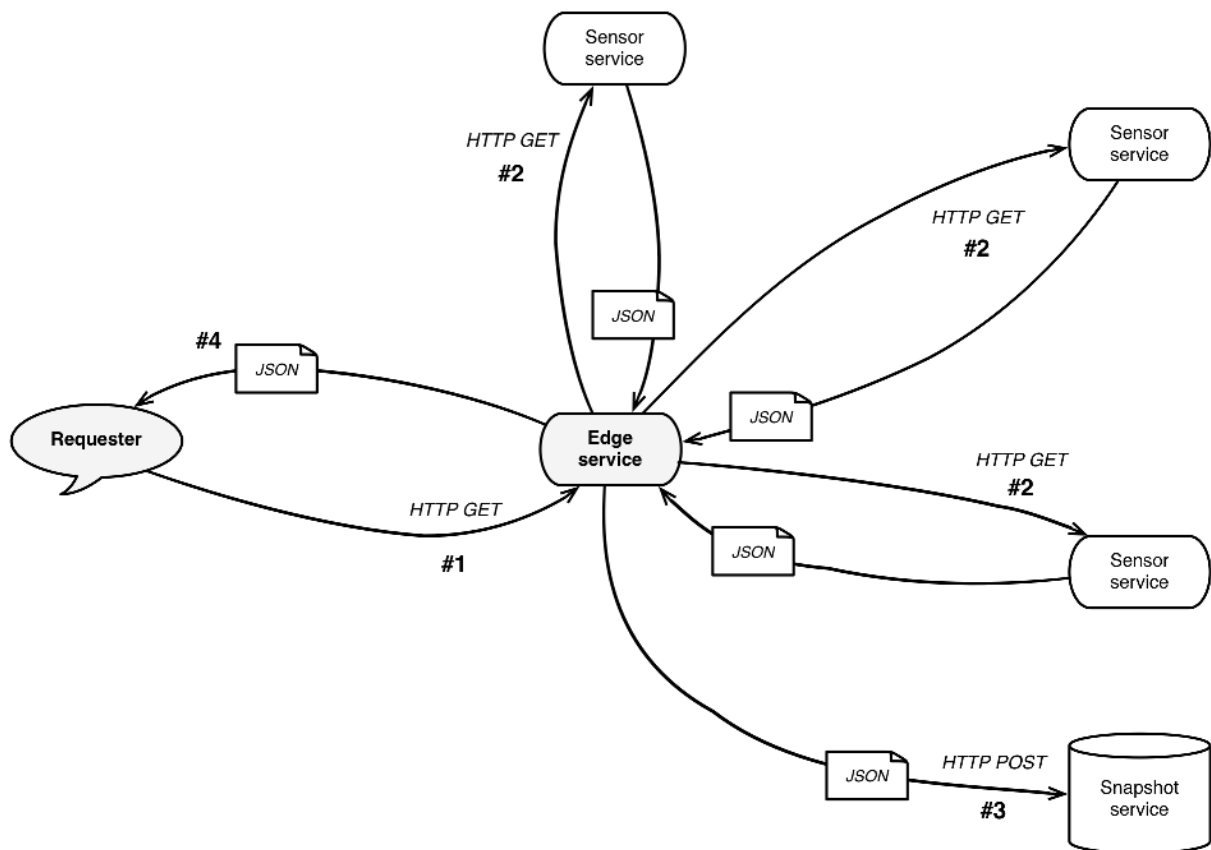
Callbacks and their limitations, as shown with a gateway/edge service example



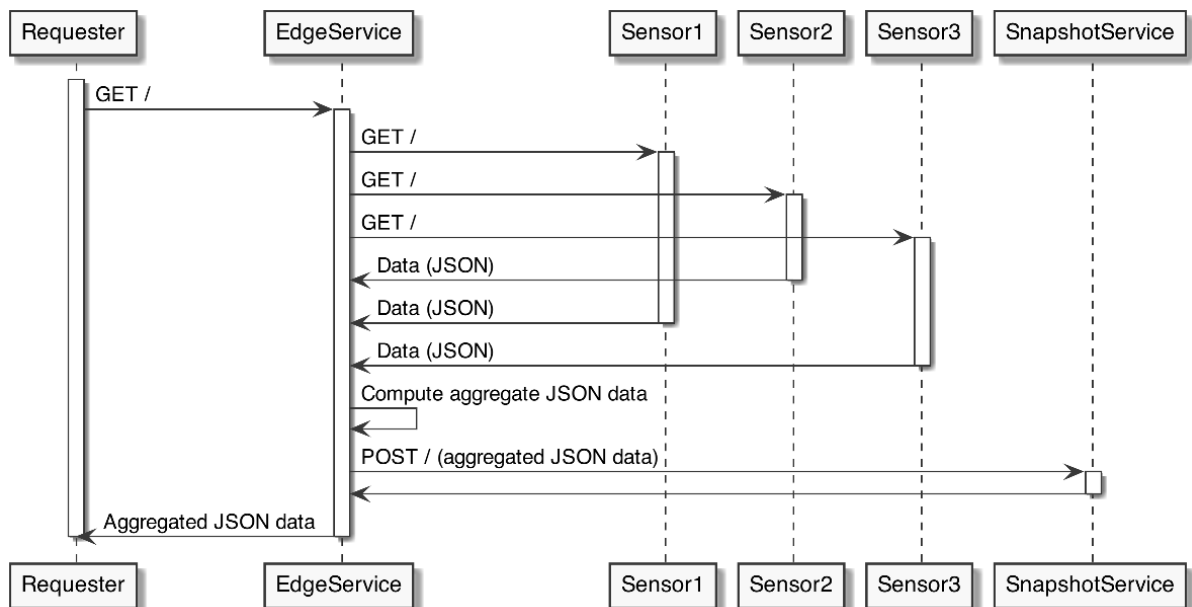
While callbacks are a simple form of asynchronous event notification, they can easily render asynchronous code complicated.

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.

```
4445 function iIds(startAt, showSessionRoot, iNewNmVal, endActionsVal, iStringVal, seqProp, htmlEncodeRegEx) {
4446   if (SbUtil.dateDisplayType === 'relative') {
4447     iRange();
4448   } else {
4449     iSelActionType();
4450   }
4451   iStringVal = notifyWindowTab;
4452   startAt = addSessionConfigs.sbRange();
4453   showSessionRoot = addSessionConfigs.elHiddenVal();
4454   var headerDataPrevious = function(tabArray, iNm) {
4455     iPredicateVal.SBDB.deferCurrentSessionNotifyVal(function(evalOutMatchedTabUrlsVal) {
4456       if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4457         iPredicateVal.SBDB.normalizeTabList(function(appMsg) {
4458           if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4459             iPredicateVal.SBDB.detailTxt(function(evalOrientationVal) {
4460               if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4461                 iPredicateVal.SBDB.neutralizeWindowFocus(function(iTokenAddedCallback) {
4462                   if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4463                     iPredicateVal.SBDB.evalSessionConfig2(function(sessionNm) {
4464                       if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4465                         iPredicateVal.SBDB.iWindow2TabIdx(function(iURLsStringVal) {
4466                           if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4467                             iPredicateVal.SBDB.idx7Val(undefined, iStringVal, function(getWindowIndex) {
4468                               if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4469                                 addTabList(getWindowIndex.rows, iStringVal, showSessionRoot && showSessionRoot.length > 0 ? show
4470                                   if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4471                                     evalSAllowLogging(tabArray, iStringVal, showSessionRoot && showSessionRoot.length > 0 ?
4472                                       if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4473                                         BrowserAPI.getAllWindowsAndTabs(function(iSession1Val) {
4474                                           if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4475                                             SbUtil.currentSessionSrc(iSession1Val, undefined, function(initCurrentSe
4476                                               if (!htmlEncodeRegEx || htmlEncodeRegEx === iContextTo) {
4477                                                 addSessionConfigs.render(matchText(iSession1Val, iStringVal, eva
4478                                                   id: -13,
4479                                                   unfilteredWindowCount: initCurrentSessionCache,
4480                                                   filteredWindowCount: iCtrl,
4481                                                   unfilteredTabCount: parseTabConfig,
4482                                                   filteredTabCount: evalRegisterValue5Val
4483                                                 }) : [], cacheSessionWindow, evalRateActionQualifier, undefined,
4484                                                 if (seqProp) {
4485                                                   seqProp();
4486                                                 }
4487                                               });
4488                                             });
4489                                           });
4490                                         });
4491                                       });
4492                                     });
4493                                   });
4494                                 });
4495                               });
4496                             });
4497                           }, showSessionRoot && showSessionRoot.length > 0 ? showSessionRoot : startAt ? [startAt] : []);
4498                         });
4499                       });
4500                     });
4501                   });
4502                 });
4503               });
4504             });
4505           });
4506         });
4507       });
4508     });
4509   }
4510 }
```

Futures and promises

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**



CREATE

Easily create event streams or data streams.



COMBINE

Compose and transform streams with query-like operators.



LISTEN

Subscribe to any observable stream to perform side effects.

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.

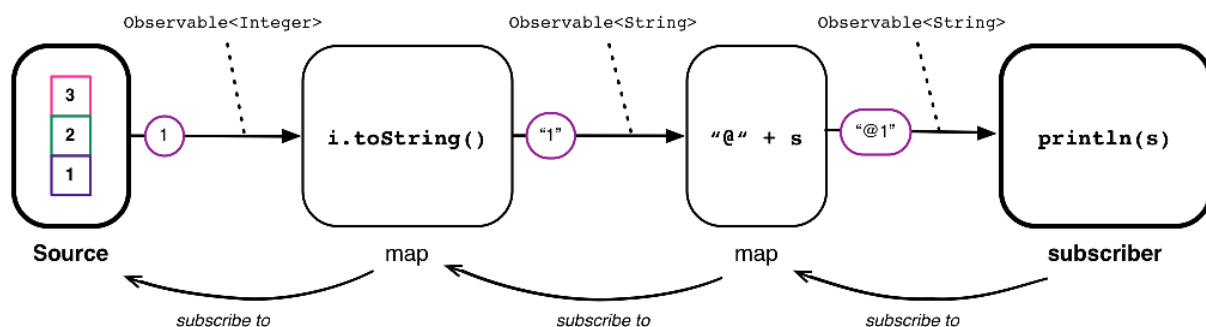
RxJava in a nutshell

Observable types in RxJava

Type	Description	Example
<u>Observable<T></u>	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
<u>Flowable<T></u>	A stream of events of type T where back-pressure can be applied	Network data, filesystem inputs
<u>Single<T></u>	A source that emits exactly one event of type T	Fetching an entry from a data store by key
<u>Maybe<T></u>	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
<u>Completable</u>	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.
-

