April 18th, 16:00 pm

@haffani95

ASYNCHRONOUS THINGS

WHY ASYNC SEEMS HARD?

IN WHAT WAY OBSERVABLE & ITERATOR PATTERNS ARE SYMMETRICAL?

WHAT LINKS ARRAYS AND EVENTS?

WHAT IS THE ANATOMY OF AN OBSERVABLE?

EPICS OR EFFECTS, WHAT RX OPERATORS TO USE?

HAMZA AFFANI

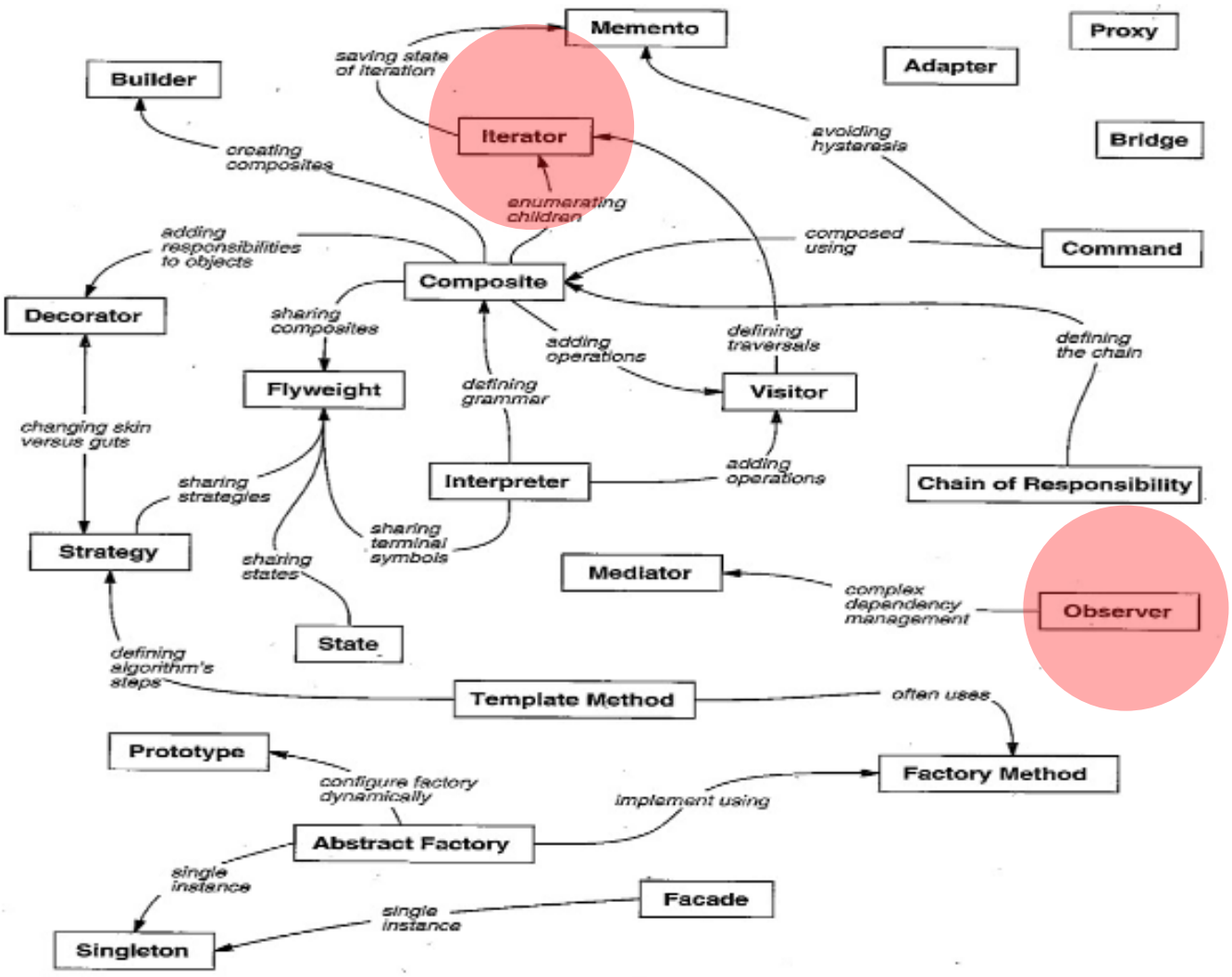
Why ASYNC SEEMS HARD?

- Race Conditions
- Memory Leaks
- Complex State Machines
- Uncaught Async Errors
- Callbacks hell

Design patterns relationships

April 18th, 16:00 pm
@haffani95

What's relation is missing?



Design Pattern Relationships

1994

April 18th, 16:00 pm
@haffani95

PROGRESSIVELY SEND INFORMATION

To the consumer

Iterator

Observer

PATTERNS ARE SYMMETRICAL

Consumer

Producer

HAS CONTROL

decides when to receive/push data

Actually, design patterns authors failed to see the pretty connection between those two patterns, as a result we thought about async differently than we should for this whole time and we gave similar things different semantics, let's list some of the subsequent consequences, and solutions adopted to rectify this misconception.

ISSUE #1

NO STANDARDIZATION

So many push APIs appeared with different implementations.

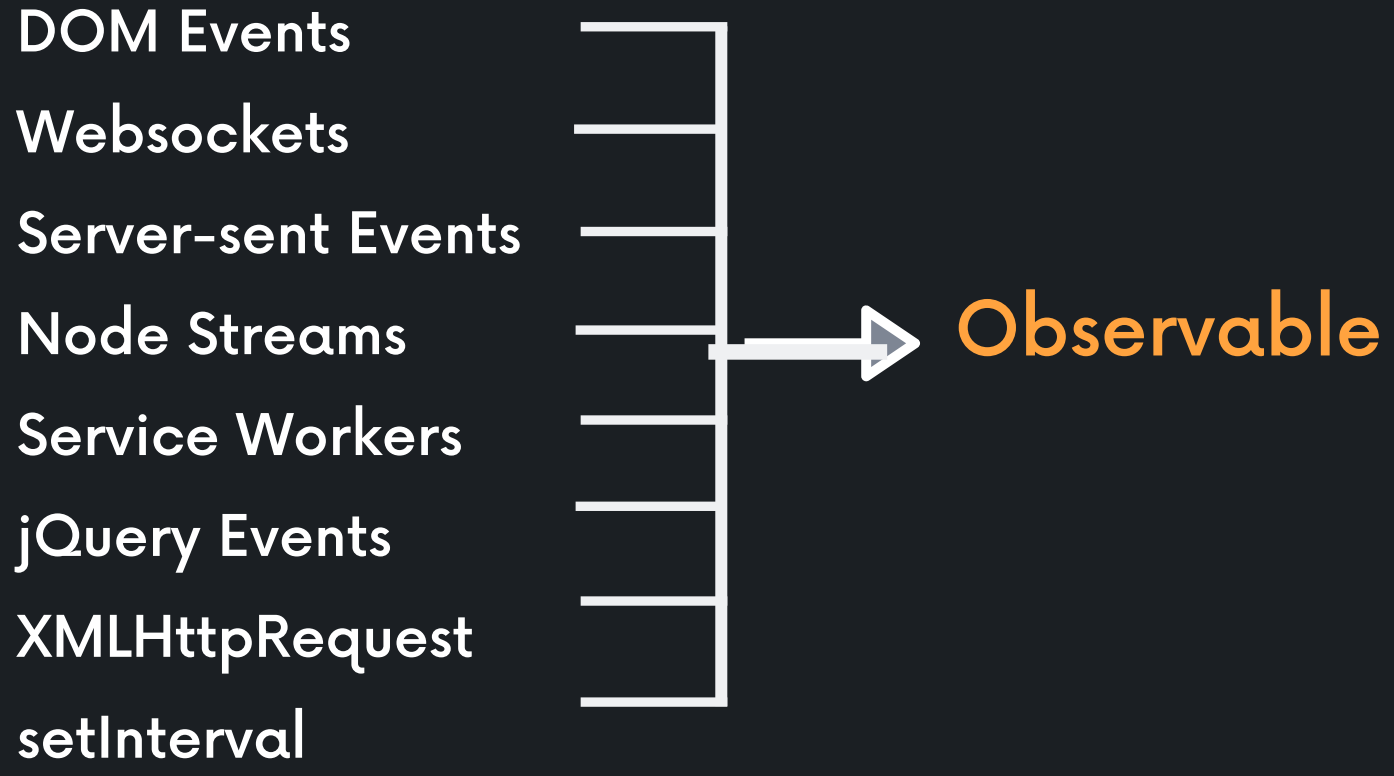
- Because the opportunity to have standardized semantics from the get go was squandered, there is no well-defined way to indicate completion or error.
- Today on the web, we have a proliferation of slightly different stream APIs.

- DOM Events
- Websockets
- Server-sent Events
- Node Streams
- Service Workers
- jQuery Events
- XMLHttpRequest
- setInterval

Solution #1

NO STANDARDIZATION

Adapt Push APIs to Observable



ISSUE #2

DIFFERENT SEMANTICS (1/2)

Events and Arrays are actually both collections, but we don't program & handle them similarly! because they're implementing different interfaces, Arrays use an iterator pattern. Events use an observer pattern, which is faulty.

Here is some @Jafar's examples showing how the source code can be similar in handling a top rated movies collection as well as a mouse drag events collection.

Top rated movies collection

```
var getTopRatedFilms = user =>
   user.videoLists.
      map(videoList =>
         videoList.videos.
            filter(video => video.rating === 5.0)).
      concatAll();

getTopRatedFilms(user).
   forEach(film => console.log(film));
```

```
[{id: 1, title: "Stranger Things"},
 {id: 2, title: "Hangover"},… ]
```

Mouse drags collection

```
var getElementDrags  = elmt =>
   elmt.mouseDowns.
      map(mouseDown =>
         document.mouseMoves.
            filter takeUntil(document.mouseUps)).
      concatAll();

getElementDrags(image).
   forEach(pos => image.position = pos);
```

```
[{x: 23, y: 44}, {x:27, y:55},
 {x:27, y:55}]
```

ISSUE #2

DIFFERENT SEMANTICS (2/2)

Limits

- In iterator, there is no way for the consumer to know if an error has occurred.
- By using events interface based on observer pattern, there is no way for the consumer to know when it's done, you should manually unhook the listener => another line to code.

Observer pattern

```
> document.addEventListener(
    "mousemove",
    function next(e) {
     console.log(e);
    });

> { clientX: 450, clientY: 558 }
> { clientX: 455, clientY: 562 }
```

```
> document.removeEventListener("mousemoves",
handler);
```

Iterator pattern

```
> var iterator = [1,2].iterator();
> console.log(iterator.next());
> { value: 1, done: false }
> console.log(iterator.next());
> { value: 2, done: false }
> console.log(iterator.next());
> { done: true }
>
```

No way to rise an error occured

Solution #2

DIFFERENT SEMANTICS

- Adapt your event to an Observable using fromEvent ✅
- Handle async the right way, don't unsubscribe from events, complete them when you're done or when another event fires ✅
- On the UI, Observable can model events, async server requests or animation.
- Now, what's an Observable?

Observable === Collection + Time

ANATOMY OF AN OBSERVABLE

Observable is an object with forEach function that accept an Observer.

Observer is nothing but an object with 3 functions: onNext, onError onCompleted to be invoked.

This example show how u can adapt whatever to an observable by just adding a pretty forEach function.

```
// "subscribe"
var subscription =
    mouseMoves.forEach({
        onNext: event => console.log(event),
        // error
        onError: error => console.error(error),
        // completed
        onCompleted: () => console.log("done")
    });

// "unsubscribe"
subscription.dispose();
```

Observer

RXJS OPERATORS

Whether you're building complex applications using Angular or React, it's very probable then that you will be handling async behaviors, actions, managing redux stores and redux stuffs, using RX reactive libraries such as redux-observable to handle Epics or effects for Angular.

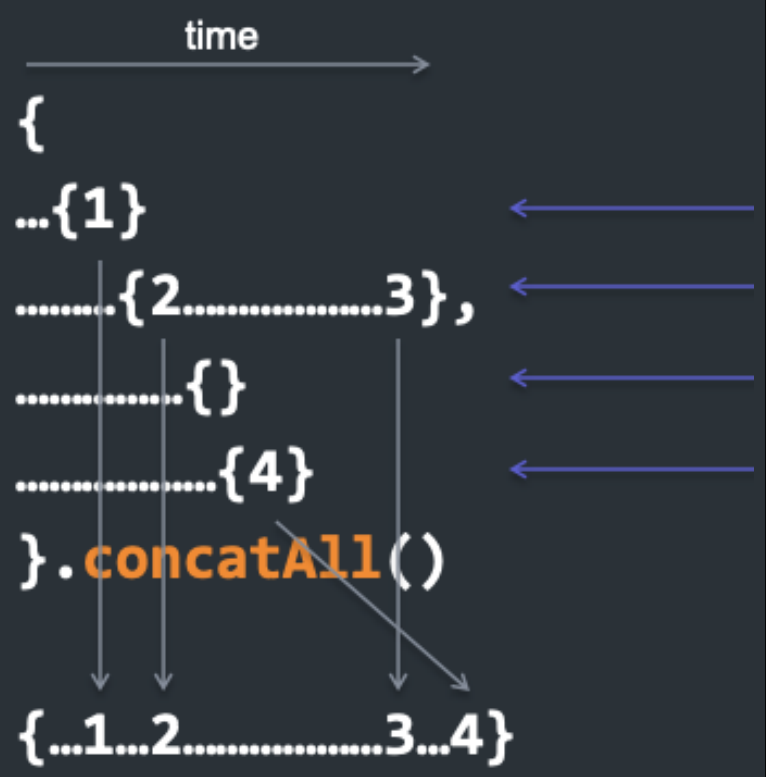
Here are the most relevant RX operators illustrations you'll need for Epics/Effects handling.

April 18th, 16:00 pm

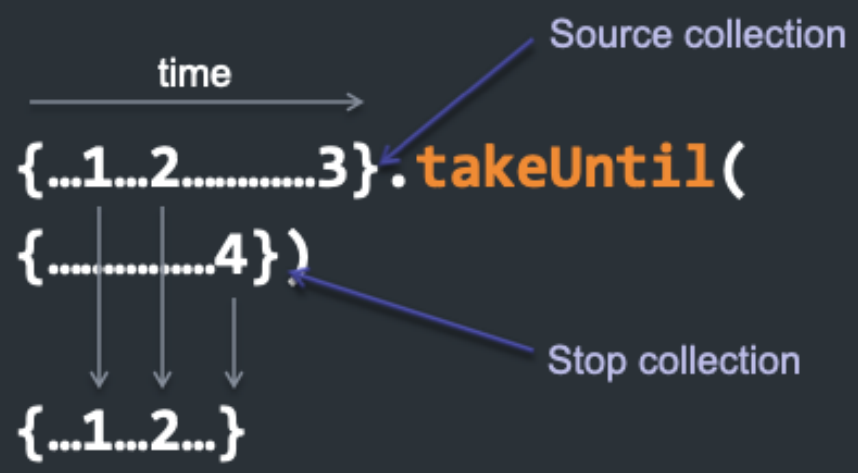
@haffani95

RXJS OPERATORS

concatAll



takeUntil

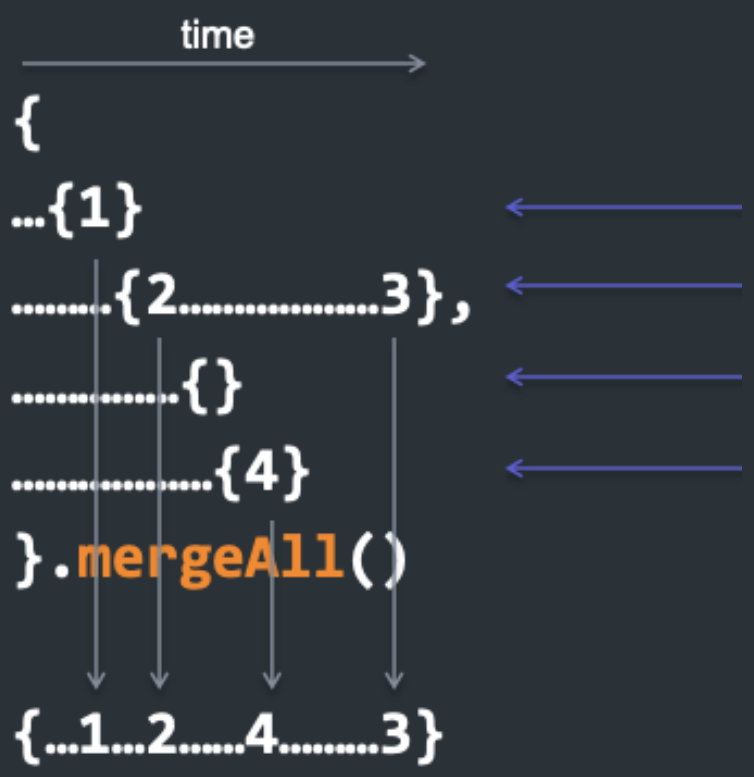


April 18th, 16:00 pm

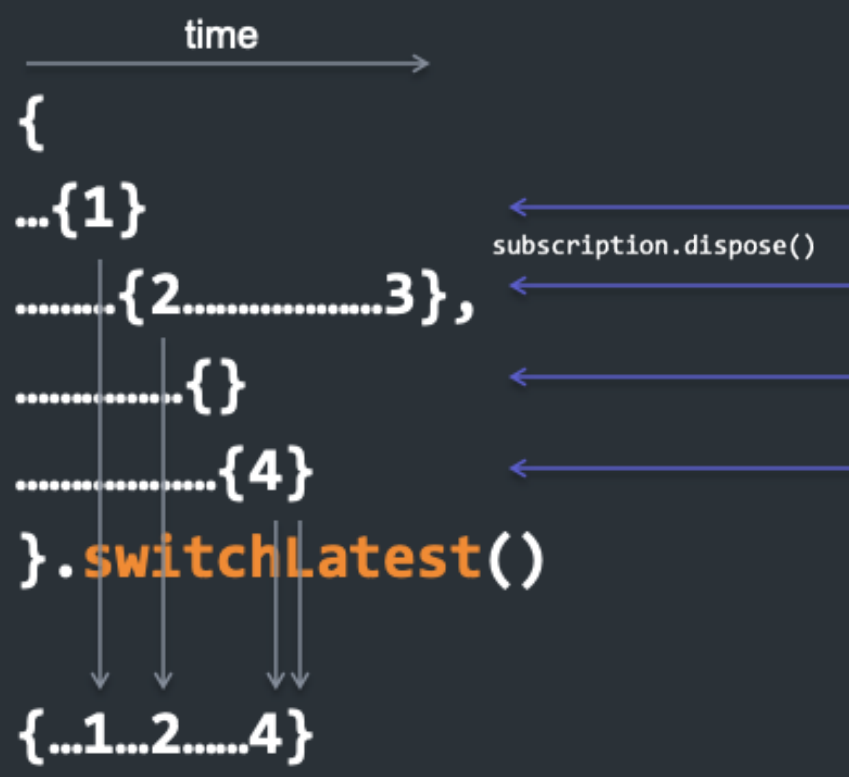
@haffani95

RXJS OPERATORS

mergeAll



switchLatest

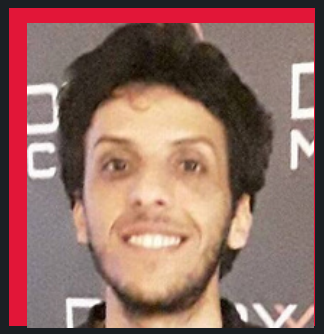


SOURCES

Based on some Jafar Hussain's (software developer at Facebook and previous Netflix's Cross-UI Team Technical Lead) publications, presentations, conferences on youtube about reactive programming.

Slides: https://www.slideshare.net/InfoQ/asynchronous-programming-at-netflix/
Netflix Javascript talks: https://www.youtube.com/watch?v=FAZJsxcykPs
Netflix blog: https://netflixtechblog.com/

Feel free to comment, I would now be interested to hear from you with your thoughts or questions 😄



HAMZA AFFANI 2020