

illionx



Global Knowledge.®

Angular Module – RxJS and operators

Peter Kassenaar –
info@kassenaar.com

WORLDWIDE LOCATIONS

BELGIUM CANADA COLOMBIA DENMARK EGYPT FRANCE IRELAND JAPAN KOREA MALAYSIA MEXICO NETHERLANDS NORWAY QATAR
SAUDI ARABIA SINGAPORE SPAIN SWEDEN UNITED ARAB EMIRATES UNITED KINGDOM UNITED STATES OF AMERICA

The Problem with RxJS Operators...

- There are so many of them...

The screenshot shows a website titled "Learn RxJS". The sidebar on the left contains a navigation menu with the following items:

- LEARN RXJS
- Introduction
- Operators (highlighted with a red box)
- Combination
 - combineAll
 - combineLatest
 - concat
 - concatAll
 - forkJoin
 - merge
 - mergeAll
 - race
 - startWith
 - withLatestFrom
 - zip
- Conditional

The main content area features the title "Learn RxJS" and a subtitle "Clear examples, explanations, and resources for RxJS.". Below this is a section titled "Introduction" which contains text about RxJS being one of the hottest libraries in web development and its applications across various frameworks and languages. It also mentions the challenges of learning reactive programming. A "But..." section follows, discussing the difficulty of learning RxJS due to its complexity and the shift from imperative to declarative style.

[Introduction](#)

Operators

Combination

combineAll

combineLatest

concat

concatAll

forkJoin

merge

mergeAll

race

startWith

withLatestFrom

zip

Conditional

Learn RxJS

Clear examples, explanations, and resources for RxJS.

Introduction

RxJS is one of the hottest libraries in web development today. Offering a powerful, functional API for dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to spread your knowledge across nearly any language, having a solid grasp on reactive programming and RxJS can offer seems like a no-brainer.

But...

Learning RxJS and reactive programming is hard. There's the multitude of concepts, large API surface area, and fundamental shift in mindset from an imperative to declarative style. This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the official documentation and existing learning material while offering a new, fresh perspective to clear any hurdles and to help you learn RxJS.

<https://www.learnrxjs.io/>

Start With These

- map
- filter
- scan
- mergeMap
- switchMap
- combineLatest
- concat
- do



10:14 / 39:03



RxJS 5 Thinking Reactively | Ben Lesh



AngularConnect

<https://www.youtube.com/watch?v=3LKMwkuK0ZE>

*"Don't try to "Rx everything". Start with
the operators you know, and go
imperatively from there. There's
nothing wrong with that."*

- Ben Lesh

Example code

PeterKassenaar / **ng-rxjs-operators**

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Examples of using different RxJS operators in an Angular CLI-project

Add topics Edit

3 commits 1 branch 0 releases 1 contributor

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

File	Description	Time
.gitignore	Updated .gitignore. Added yarn support	Latest commit 9ac806d 22 hours ago
src	Added first version of code.	22 hours ago
.angular-cli.json	Added first version of code.	22 hours ago
.editorconfig	Added first version of code.	22 hours ago
.gitignore	Updated .gitignore. Added yarn support	22 hours ago

<https://github.com/PeterKassenaar/ng-rxjs-operators>

This presentation - 2 sections

1. Basic Streams

- Create a stream, or multiple streams from scratch, based on a UI-element
- Subscribe to that stream(s) and do something

2. Operators

Demo the purpose and inner workings of some often used operators



Basic streams

Creating observables from scratch

Turn something into an observable

- Basic creation operators
 - `create()` – create an observable manually
 - `from()` – turn an array, promise or iterable into an observable
 - `fromEvent()` – turn an event into an observable
 - `of()` – turn a variable into an observable, emit it's value('s) and complete
- <https://www.learnrxjs.io/learn-rxjs/operators/creation>

The screenshot shows the 'Learn RxJS' website interface. On the left, there is a sidebar with a navigation menu. The 'Operators' section is expanded, and 'Creation' is selected, highlighted with a blue background. Under 'Creation', there are three items: 'ajax', 'create', and 'defer'. The main content area has a title 'Creation' and a paragraph explaining that these operators allow creating an observable from nearly anything. Below the paragraph is a 'Contents' section with a list of operators: 'ajax ★', 'create', 'defer', 'empty', and 'from ★'.

Learn RxJS

Introduction

LEARN RXJS

Operators

Combination

Conditional

Creation

ajax

create

defer

Creation

These operators allow the creation of an observable from nearly anything. From generic to specific use-cases you are free, and encouraged, to turn [everything into a stream](#).

Contents

- [ajax ★](#)
- [create](#)
- [defer](#)
- [empty](#)
- [from ★](#)

Creating an observable from a textbox

```
<input class="form-control-lg" type="text" #text1  
      id="text1" placeholder="Text as a stream">  
<p>{{ textStream1 }}</p>
```

```
textStream1 = 'Type some text above...';  
@ViewChild('text1', {static: true}) text1; // two parameters for @ViewChild()  
  
// Suggestion: break the ngOnInit() up in smaller functions  
ngOnInit(): void {  
  this.onTextStream1();  
}  
  
onTextStream1() {  
  fromEvent(this.text1.nativeElement, 'keyup')  
    .subscribe((event: any) => this.textStream1 = event.target.value);  
}
```

Result

Basic stream: we subscribe to chars coming from the textbox:

Text as a stream

Type some text above...

Basic stream: we subscribe to chars coming from the textbox:

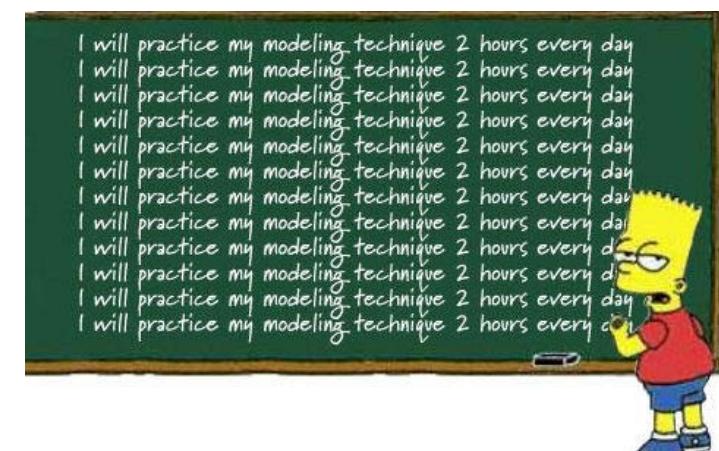
This is a stream...

This is a stream...



Workshop

- Create a textbox.
 - Add items that are typed in, to a Todo-list
 - Use an observable to subscribe to keyup-events.
- Capture key events, test if the key pressed is the Enter-key
 - If it's Enter, Add the current value of the textbox to a static array todoItems[]
- Example `.../basic-stream/basic-stream.component.ts`
 - Start from scratch or expand this component



Using the pipe()

- We can transform the stream by adding operators to the pipeline
- Inside the pipeline we calculate new values and bind them to the UI, using generic attribute binding [...]

```
<button #btnRight class="btn btn-secondary">Right</button>
Mario is moved by observable streams from the button click.
<div>
  
</div>
```

Inside our class

```
// mario
position: any;
@ViewChild('btnRight', {static: true}) btnRight;
```

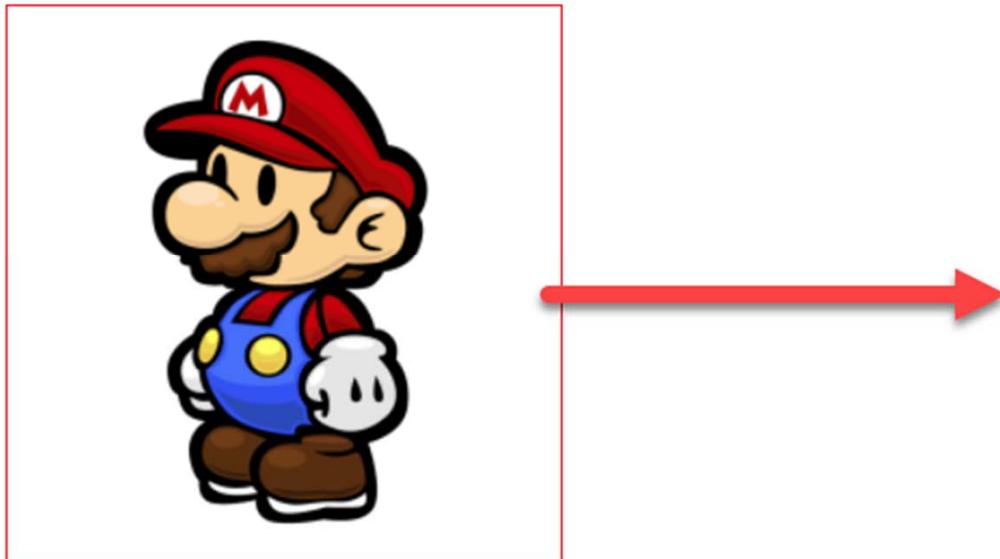
```
fromEvent(this.btnRight.nativeElement, 'click')
.pipe(
  map(event => 10), // 1. map the event to a useful value, in this case 10px
  startWith({x: 100, y: 100}), // 2. start with an object of { 100, 100}.
  scan((acc: any, current: number) => { // 3, use the scan operator as reducer function.
    return {
      x: acc.x + current,
      y: acc.y
    };
  })
)
.subscribe(result => {
  this.position = result;
});
```

The result of the previous operator is the input of the next operator in the pipeline

Result

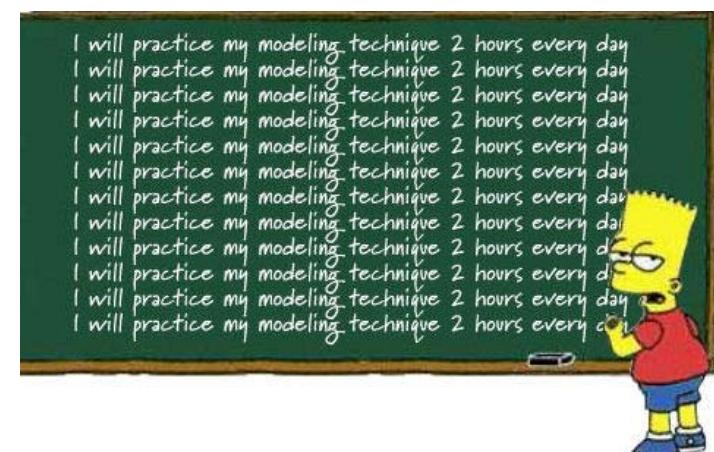
Right

Mario is moved by observable streams from the button click.



Workshop

- Create button that moves Mario to the left.
- Example `./basic-stream/basic-stream.component.ts`
 - Start from scratch or expand this component





Combining streams

There are **a lot** of options to combine multiple streams into one stream

Options for combining streams

- `combineLatest()` - When any observable emits a value, emit the last emitted value from each
- `concat()` - Subscribe to observables in order as previous completes
- `forkJoin()` - When all observables complete, emit the last emitted value from each
- `merge()` - Turn multiple observables into a single observable
- `startWith()` - Emit given value first
- <https://www.learnrxjs.io/learn-rxjs/operators/combination>

The screenshot shows the Learn RxJS website's navigation bar on the left and a detailed page on the right.

Navigation Bar:

- Learn RxJS logo
- Search icon
- Introduction
- LEARN RXJS
- Operators (dropdown menu)
- Combination (selected item in dropdown)
- combineAll
- combineLatest
- concat
- concatAll
- endWith
- forkJoin

Page Content:

Combination

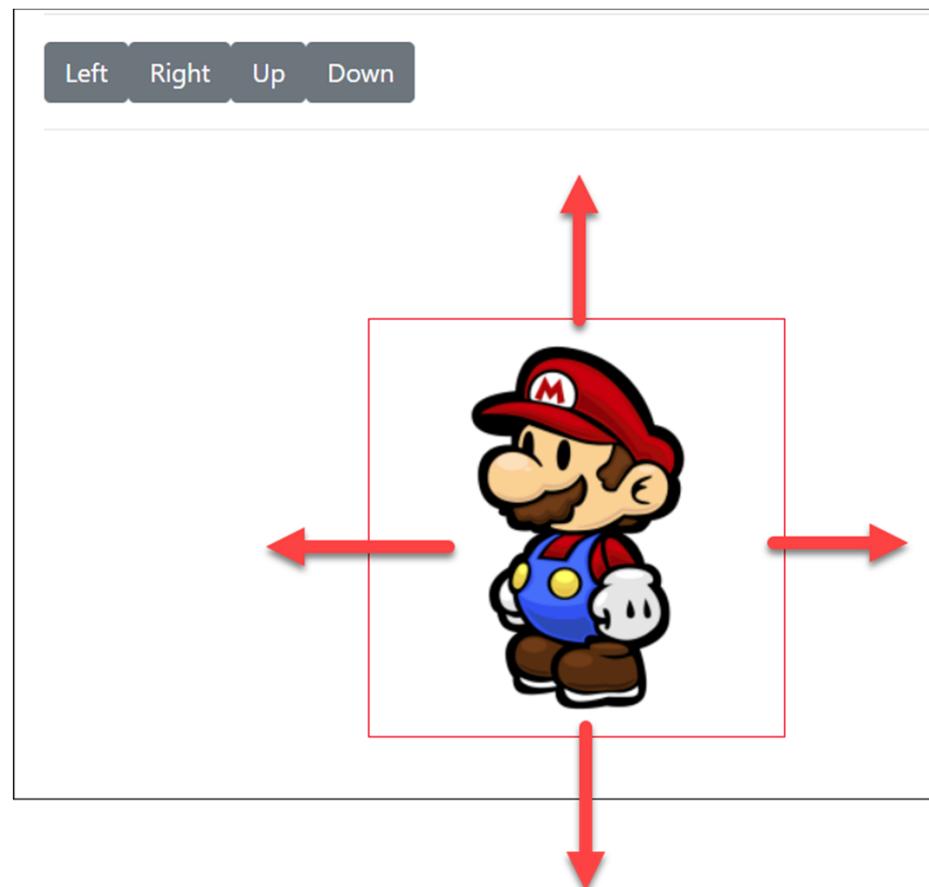
The combination operators allow the joining of information from multiple observables. Order, time, and structure of emitted values is the primary variation among these operators.

Contents

- [combineAll](#)
- [combineLatest ★](#)
- [concat ★](#)
- [concatAll](#)
- [endWith](#)
- [forkJoin](#)

Use case: move Mario

- Create four buttons to move Mario in different directions
 - all combined in one stream and one subscription



Template

```
<button #btnLeft> Left </button>
<button #btnRight> Right </button>
<button #btnUp> Up </button>
<button #btnDown> Down </button>
<hr>

<img id="mario" #mario
      [style.left] = "position.x + 'px'"
      [style.top] = "position.y + 'px'"
      src = "../assets/img/mario-1.png" alt = "Mario">
```

```

position: any;
@ViewChild('btnLeft', {static: true}) btnLeft;
@ViewChild('btnRight', {static: true}) btnRight;
@ViewChild('btnUp', {static: true}) btnUp;
@ViewChild('btnDown', {static: true}) btnDown;

ngOnInit(): void {
  // Create multiple streams
  const right$ = fromEvent(this.btnRight.nativeElement, 'click')
    .pipe(
      map(event => {
        return {direction: 'horizontal', value: 10};
      }) // 10 px to the right
    );
  const left$ = fromEvent(this.btnLeft.nativeElement, 'click')
    .pipe(
      map(event => {
        return {direction: 'horizontal', value: -10};
      }) // -10 px to the left
    );
  ...
  // combine our streams
  merge(right$, left$, up$, down$)
    .pipe(
      startWith({x: 200, y: 100}),
      scan((acc: any, current: any) => {
        return {
          x: acc.x + (current.direction === 'horizontal' ? current.value : 0),
          y: acc.y + (current.direction === 'vertical' ? current.value : 0)
        };
      })
    ).subscribe(result => {
      this.position = result;
    });
}

```

Create 4 different streams

Merge the streams

One subscriber



Sequencing streams

Use streams to start other streams so you can use the combined behavior

Use case: drag & drop

- We want to move Mario around with the mouse: **drag-n-drop**
- We compose different streams for that:
 - `down$`, when the mouse is pressed
 - `move$`, when the mouse is moved, return the current mouse position
 - `up$`, when the mouse is released
- Again, we have only one (1) subscription
 - **Subscribe** to the `down$`. This is the initiator
 - After the initial event, **switch** to the `move$`: using the `switchMap()` operator
 - But only **until** the `up$` event occurs!
 - Then complete the observable and release Mario
- This translates to the following code:

```

// correction factor for image and current page. Your mileage may vary!
const OFFSET_X = 180;
const OFFSET_Y = 280;

// 1. With Drag and drop, first you capture the mousedown event
const down$ = fromEvent(document, 'mousedown');

// 2. What to do when the mouse moves
const move$ = fromEvent(document, 'mousemove')
  .pipe(
    map((event: any) => {
      return {x: event.pageX - OFFSET_X, y: event.pageY - OFFSET_Y}; // OFFSET as correction factor
    })
  );

// 3. Capture the mouseup event
const up$ = fromEvent(document, 'mouseup');

// 4. extend the down$ stream and subscribe
down$
  .pipe(
    // IF the down$-event happens, we are no longer interested in it. Instead,
    // we switch the focus to the move$ event.
    // BUT: we are only interested in the move until the mouse is released again.
    // So we add *another* pipe and use the takeUntil() operator.
    switchMap(event => move$.pipe(
      takeUntil(up$)
    )),
    startWith(this.position)
  )
  .subscribe(result => this.position = result);
}

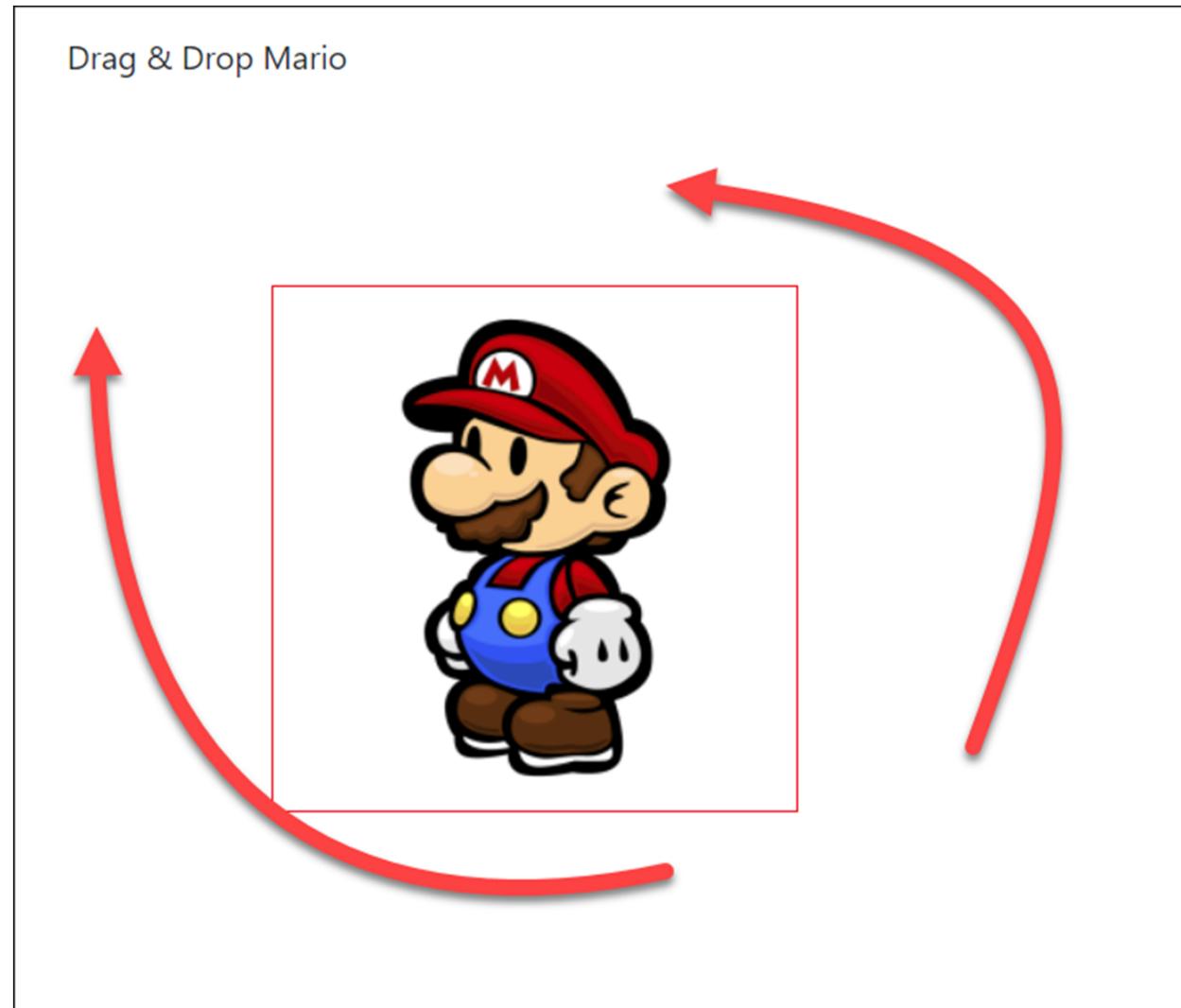
```

Compose streams

Switch execution to another observable

Subscribe and update position

Result



`../streams/drag-drop-stream.component.html|ts`

An autocomplete/typeahead demo

An autocomplete/typeahead demo

bel



Belarus
Minsk



Belgium
Brussels



Belize
Belmopan



Palau
Ngerulmud

Template

As per usual – little HTML

```
<h4>An autocomplete/typeahead demo</h4>
<!--Get the keyword, the class is subscribed to this inputbox-->
<input type="text" placeholder="Country name..." 
       #typeahead class="form-control-lg">
<hr/>
<!--Results-->
<ul class="list-group">
  <li class="list-group-item" *ngFor="let country of countries$ | async ">
    
    <p>
      <strong>{{ country.name }}</strong><br>
      {{ country.capital }}
    </p>
  </li>
</ul>
```

This time: using the
async pipe

Code

```
interface ICountry {  
  name: string;  
  capital: string;  
  flag: string;  
}  
  
const errorCountry: ICountry = {  
  name: 'Error',  
  capital: 'Not found',  
  flag: ''  
};
```

Creating interface and simple error message

```
@ViewChild('typeahead', {static: true}) typeahead;  
countries$: Observable<ICountry[]>;  
  
constructor(private http: HttpClient) {  
}
```

Inside the component class

```
ngOnInit(): void {  
  this.countries$ = fromEvent(this.typeahead.nativeElement, 'keyup')  
    .pipe(  
      filter((e: any) => e.target.value.length >= 2),  
      debounceTime(400),  
      map((e: any) => e.target.value),  
      distinctUntilChanged(),  
      switchMap(keyword => this.getCountries(keyword))  
    );  
}
```

Main method. See example code for comments

Fetching the countries

```
getCountries(keyword): Observable<ICountry[]> {
  // 7. Create the actual http-call.
  return this.http
    .get<ICountry[]>(`https://restcountries.eu/rest/v2/name/${keyword}?fields=name;capital;flag`)
    .pipe(
      // 8. catch http-errors and return a 'not found' country
      catchError(err => {
        console.log(err);
        return of([errorCountry]);
      })
    );
}
```

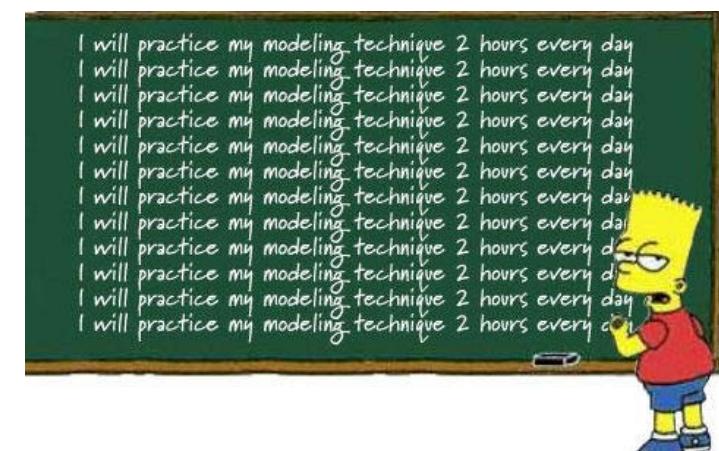
Documentation:

- filter: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/filter>
- debounceTime: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/debounceTime>
- distinctUntilChanged: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/distinctUntilChanged>
- switchMap: <https://www.learnrxjs.io/learn-rxjs/operators/transformation/switchmap>
- catchError: https://www.learnrxjs.io/learn-rxjs/operators/error_handling/catch

Workshop

- Search movies in the Open Movie Database.
 - [https://www.omdbapi.com/?apikey=f1f56c8e&s=\[keyword\]](https://www.omdbapi.com/?apikey=f1f56c8e&s=[keyword])
 - Create an AutoComplete inputfield for movie titles.
 - For instance, if you type 'ava...' it should show movies like Avatar, and more
- Example [.../basic-stream/typeahead.component.ts](#)
 - Start from scratch or expand this component

The screenshot shows the OMDb API homepage. At the top, there are navigation links: OMDb API, Usage, Parameters, Examples, Change Log, and API Key. Below these, the title "OMDb API" and subtitle "The Open Movie Database" are displayed. A sub-section titled "Poster API" features a thumbnail image of the movie "Blade Runner 2049". Text next to the image states: "The Poster API is only currently over 280,000 with resolutions up to". At the bottom left, a blue box contains "Attention Users" with three bullet points: "04/08/19 - Added support for eight digit IMDB IDs.", "01/20/19 - Suppressed adult content from search results.", and "01/20/19 - Added Swagger files ([YAML](#) [JSON](#)) to expose current API abilities and upcoming REST functions."





More operators

Getting familiar with some of the more used operators

map() and mapTo()

```
const source = from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

// .map() - apply a function every emitted output.
source.pipe(
  map((val: number) => val = val * 10)
)
  .subscribe(result => this.mapData.push(result));

// .mapTo() - map the emission to a constant value
source.pipe(
  mapTo('Hello World')
)

.subscribe(result => this.mapToData.push(result));
```

filter()

```
const source      = Observable.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
const sourceCities = Observable.from([
  'Haarlem',
  'Breda',
  'Amsterdam',
  'Groningen',
  'Hengelo',
  ...
]);
// filter() - only emit values that pass the provided condition.
// In this case: only even numbers are passed through.
source.pipe(
  filter(val => val % 2 === 0)
)
.subscribe(result => this.filterData.push(result));

// Emit only the cities that starts with an 'H'.
sourceCities.pipe(filter(city => city.startsWith('H')))
.subscribe(result => this.cityData.push(result));
```

scan()

```
const source      = Observable.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
```

```
// .scan() acts as the classic .reduce() function on array's. It takes an
// accumulator and the current value. The accumulator is persisted over time.
source
  .pipe(
    startWith(0),
    scan((acc, curr) => acc + curr)
  )
  .subscribe(result => this.scanData.push(result));
```

concat()

- Concat subscribes to observables *in order*.
- “Only when the first one completes, let me know.”
- Then move on to the next one.
- Use concat() when the order of your observables matters.
- Example code: simulated delay.
 - The second observable (which is finished first) only emits when the first observable completes.

app/shared/services/data.service.ts

```
// constants that are used as pointers to some json-data
const BOOKS: string    = 'assets/data/books.json';
const AUTHORS: string = 'assets/data/authors.json';

getConcatData(): Observable<any> {
  // First call. Simulate delay of 1 second
  const authors = this.http.get(AUTHORS)
    .pipe(
      delay(2000)
    );
  // Second call. Simulate delay of 2 seconds
  const books = this.http.get(BOOKS)
    .pipe(
      delay(1000)
    );
  // return the concatenated observable. It will always deliver
  // FIRST the results of the first call. No matter how long the delay.
  return concat(authors, books);
}
```

[https://rxjs-dev.firebaseio.com/api/operators\(concat](https://rxjs-dev.firebaseio.com/api/operators(concat)

merge()

- Merge combines multiple observables into one single observable.
- It emits *as soon as* it gets a result.
- Use `merge()` when order of observables is not important.

```
getMergeData(): Observable<any> {
  // First call. Simulate delay of 3 seconds, so this observable will emit last.
  const authors = this.http.get(AUTHORS)
    .pipe(
      delay(3000)
    );

  // Second call. Simulate delay of 1 seconds, so this observable will emit first.
  const books = this.http.get(BOOKS)
    .pipe(
      delay(1000)
    );

  // return the merged observable. BOOKS will be delivered first
  return merge(authors, books);
}
```

<https://rxjs-dev.firebaseio.com/api/index/function/merge>

mergeMap()

- Merges the value of an inner observable into an outer observable.
- Need only the *last* value emitted? Use `.switchMap()`.
- Use this for example to retrieve results in a second observable, based on the output of a first observable

.mergeMap() result

See [/app/shared/services/data.service.ts](#) for example code.

We are looking for books by authorName here. But we only have an authorId, not a name.

```
[  
  {  
    "title": "Web Development Library - TypeScript",  
    "author": "Peter Kassenaar",  
    "bookID": 1  
  },  
  {  
    "title": "Web Development Library - Angular",  
    "author": "Peter Kassenaar",  
    "bookID": 2  
  }  
]
```



```

getMergeMapData(authorID: number = 0): Observable<any> {
  // first http-call, outer observable.
  return this.http.get(AUTHORS)
    .pipe(
      tap(response => console.log(response)),
      map((authors: any[]) => {
        console.log(authors);
        // find the correct author, using the array .find() method
        return authors.find((author: any) => author.id === authorID);
      }),
      mergeMap((author: any) => {
        if (author) {
          // second http-call, inner observable
          return this.http.get(BOOKS)
            .pipe(map((books: any[]) => {
              // filter books, bases on authorname we found earlier.
              return books.filter((book: any) => book.author === author.name);
            }));
        } else {
          // nothing found. Return empty array
          return of([]);
        }
      })
    );
}

```

.forkJoin()

- Make multiple (http) requests and return one combined response *once all observables are completed.*
- .forkJoin() returns an array with the last emitted value from each observable.
- Compose results as per your needs

```

getForkJoinData(): Observable<any> {
  return forkJoin(
    // first call
    this.http.get(AUTHORS)
      .pipe(
        delay(3000)
      ),
    // second call
    this.http.get(BOOKS)
  )
  .pipe(
    map((data: any[]) => {
      // data is now an array with 2 objects, b/c we did 2 http-calls.
      // First result, from the http-call to AUTHORS
      const author: any = data[0][0]; // Get just first author from file.
      // Second result, from the http-call to BOOKS
      const books: any[] = data[1];
      // Compose result, in this case adding the books to the extracted author.
      author.books = books.filter(book => book.author === author.name);
      return author;
    })
  );
}

```

Other interesting / often used Operators

- `from()`, `of()` - create observables from almost everything
- `fromEvent()` - create observable from a DOM-event.
- `debounce()`, `debounceTime()` - Discard emitted values that take less than the specified time between output
- `tap()` - utility operator - transform side-effects, such as logging
- ...and much more... See <http://www.learnrxjs.io>
- See also <https://rxjs-dev.firebaseio.com/>

ARTICLES SPEAKING TRAINING WORKSHOPS VIDEOS



My name is [Cory Rylan](#). [Google Developer Expert](#) and Front End Developer at VMware Clarity. [Angular Boot Camp](#) instructor.

[!\[\]\(3adacafff8284323a4ad5aa5cb457aa9_img.jpg\) Follow @coryrlan](#)

[!\[\]\(82000065e84e59d5fab9cab7a5089787_img.jpg\)](#) [!\[\]\(98d8d568c09298a209226e363e9c5237_img.jpg\)](#) [!\[\]\(39bdced44e6bfdb067462eca49c9c9d2_img.jpg\)](#) [!\[\]\(569de1ebad75dd50ce8a91d04f91b107_img.jpg\)](#)

Angular Multiple HTTP Requests with RxJS



Cory Rylan
Nov 15, 2016
Updated Jun 18, 2019 - 5 min read

[!\[\]\(4aaa9e147ab7695772f2f7e206121a1a_img.jpg\)](#) [!\[\]\(5aab9da522939c346db6dde90bb91e33_img.jpg\)](#)

This article has been updated to the latest version of [Angular 9](#) and tested with Angular 8. The content is likely be applicable for older Angular 2 or other previous versions.

A typical pattern we run into with single page apps is to gather up data from multiple API endpoints and then display the gathered data to the user. Fetching numerous asynchronous requests and managing them can be tricky but with the Angular's Http service and a little help from the included RxJS library, it can be accomplished in just a few of lines of code. There are multiple ways to handle multiple requests; they can be sequential or in parallel. In this post, we will cover both.



Angular Form Essentials
Learn the essentials to get started creating amazing forms with Angular!

[GET E-BOOK NOW!](#)

<https://coryrlan.com/blog/angular-multiple-http-requests-with-rxjs>

More information

The screenshot shows a Medium article page. At the top right are 'Sign in / Sign up' and social sharing icons for LinkedIn and Twitter. Below that is the author's profile picture and name, 'Netanel Basal', with a 'Follow' button, and the publish date, 'Jan 24 · 3 min read'. The main title of the article is 'RxJS—Six Operators That you Must Know'. The article content is a block of RxJS code:

```
class TakeSubscriber<T> extends Subscriber<T> {  
  private count: number = 0;  
  
  constructor(destination: Subscriber<T>, private total: number) {  
    super(destination);  
  }  
  
  protected _next(value: T): void {  
    const total = this.total;  
    const count = ++this.count;  
    if (count <= total) {  
      this.destination.next(value);  
    }  
  }  
}  
// Example usage:  
const take5 = new TakeSubscriber(5);  
take5.subscribe(x => console.log(x));  
take5.next('a'); // Logs 'a'  
take5.next('b'); // Logs 'b'  
take5.next('c'); // Logs 'c'  
take5.next('d'); // Logs 'd'  
take5.next('e'); // Logs 'e'  
take5.next('f'); // No output, because count (6) is greater than total (5)
```

At the bottom left is the author's profile picture and name, 'NetanelBasal'. To the right is a call-to-action button 'GET UPDATES'.

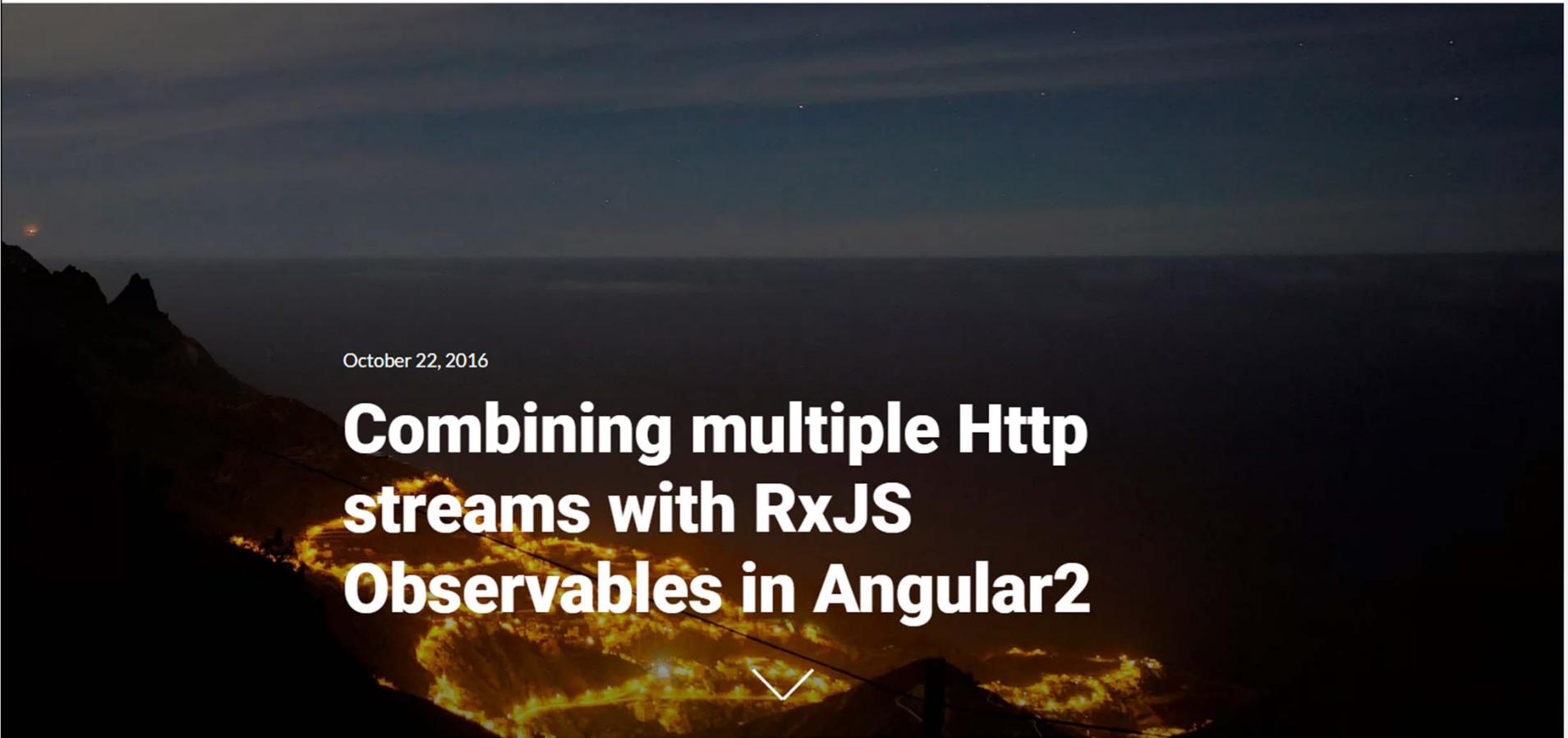
<https://netbasal.com/rxjs-six-operators-that-you-must-know-5ed3b6e238a0#.ocz4xfwl>



Daniele Ghidoli

About me

...



October 22, 2016

Combining multiple Http streams with RxJS Observables in Angular2

<http://blog.danieleghidoli.it/2016/10/22/http-rxjs-observables-angular/>



RxJS

Reactive Extensions Library for JavaScript

[GET STARTED](#)

[API DOCS](#)

REACTIVE EXTENSIONS LIBRARY FOR JAVASCRIPT

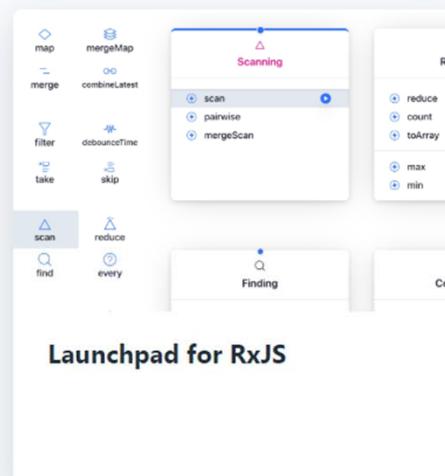
RxJS is a library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code. This project is a rewrite of Reactive-Extensions/RxJS with better performance, better modularity, better debuggable call stacks, while staying mostly backwards compatible, with some breaking changes that reduce the API surface

<https://rxjs-dev.firebaseio.com/>

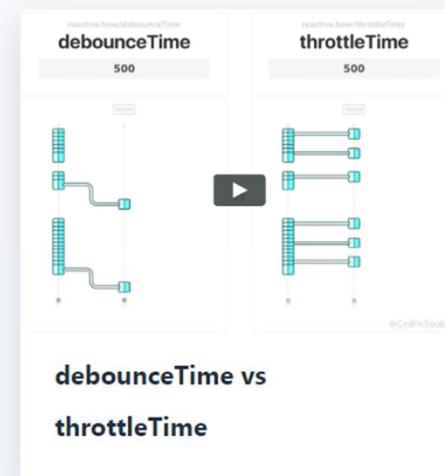
reactive.how 2.0-alpha.4   Launchpad for RxJS →

ESSENTIALS REDUCING TAKING SKIPPING COMBINING TIME CONCATENATING
map reduce take skip (soon) combineLatest delay startWith (soon)
filter max takeWhile skipWhile zip debounceTime endWith (soon)
merge count first takeLast throttleTime concat (soon)
scan last

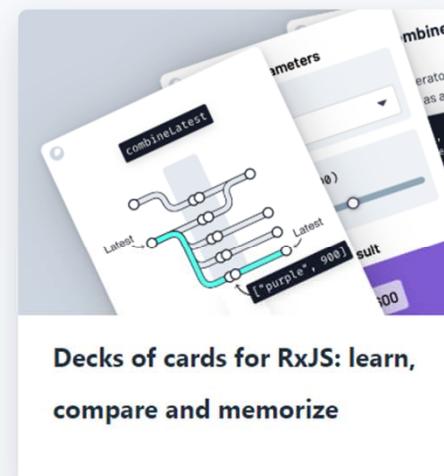
Learn RxJS operators and Reactive Programming principles



Launchpad for RxJS



debounceTime vs throttleTime



Decks of cards for RxJS: learn, compare and memorize



scan reduce



map filter



zip combineLatest

<https://reactive.how/>

Workshop

- Create a call to the Open Movie Database API, using a keyword to search for 10 movies
 - <http://www.omdbapi.com/?apikey=f1f56c8e&s=<keyword>>
- Create an additional call to get details for every movie returned. Use the `imdbID` property for that
 - <http://www.omdbapi.com/?apikey=f1f56c8e&i=<imdbID>>
- Display results in the UI, first a list of movies, then details per movie as soon as they come available.
- What is your best solution? Multiple ways of doing this!

