



Adieu RxJS ! Vive les Signals
! Oh wait...

Anthony Pena



Anthony Pena

Développeur Web Fullstack @ [sf=ir]



Avant les frameworks



La Réactivité ?



"A declarative programming model for updating based on changes to state."

-- Kristen / pzuraq

A painting depicting a rural scene. In the foreground, a man with a long white beard and a brown hat sits in a wooden cart, holding a long wooden pole. Another man, also with a beard and wearing a hat, is crouching on the ground, pushing a wooden sled or plow. Two dark-colored oxen are harnessed to the sled, pulling it through a field of golden, harvested wheat. The background shows rolling hills under a blue sky with scattered clouds.

AngularJS et ses watchers

A detailed painting of a steam locomotive pulling a passenger train through a dense forest. The train is emitting a large plume of white smoke. In the background, a massive industrial complex with numerous tall smokestacks and pipes is visible, emitting smoke and steam. Beyond the factory, a modern city skyline with many skyscrapers is nestled among hills. The foreground shows the dark, textured surface of the train's engine and the tracks.

Angular 2 et Zone.js

```
@Component({
  template: `
    <p>{{ text }}</p>
  `,
})
export class PlaygroundComponent {
  text = "";

  ngOnInit() {
    setInterval(() => this.text += '!', 1_000)
  }
}
```

```
@Component({
  template: `
    <input (change)="setText($event)"/>
    <p>{{ text }}</p>
  `,
})
export class PlaygroundComponent {
  text = "";

  setText(event: Event) {
    this.text = (event.target as HTMLInputElement).value;
  }
}
```

```
@Component({
  template: `
    <input [(ngModel)]="text"/>
    <p>{{ text }}</p>
  `,
})
export class PlaygroundComponent {
  text = "";

  ngOnInit() {
    setInterval(() => this.text += '!', 1_000)
  }
}
```

```
@Component({
  selector: 'app-root',
  standalone: true,
  template: `
    <p>{{ text }}</p>
  `,
})
export class PlaygroundComponent {
  text = "waiting...";

  ngOnInit() {
    this.asyncHello().then(text => this.text = text);
  }

  private asyncHello(): Promise<string> {
    // ...
  }
}
```

```
@Component({
  template: `
    <p>{{ text | async }}</p>
  `,
})
export class PlaygroundComponent {
  text = this.asyncHello();

  private asyncHello(): Promise<string> {
    // ...
  }
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  @Input() text: string;
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  @Input() text: string;
  @Output() textChange = new EventEmitter<string>();
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  @Input() text: string;
  @Output() textChange = new EventEmitter<string>();
  @ViewChild(Alert) alerts: Alert;
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  @Input() text: string;
  @Output() textChange = new EventEmitter<string>();
  @ViewChild(Alert) alerts: Alert;
  @ViewChildren(CustomCard) cards: QueryList<CustomCard>;
}
```



C'est cool tout ça non ?



Oui mais Zone.js

The illustration depicts a cartoon character with brown hair and a beard, looking shocked with wide eyes and a mouth open. He is wearing a dark grey t-shirt with a white circular logo containing a hand icon. He is sitting at a desk with a computer monitor displaying a blue screen with a white skull and crossbones symbol. The background is a chaotic laboratory or workshop filled with various scientific equipment, bones, and organic debris. A yellow sign above the monitor reads "ZONE" with a asterisk symbol. To the right, there's a "SPC" sign and a "No Entry" sign. The overall scene conveys a sense of horror or surprise related to the Zone.js framework.



Angular 17 et les Signals

```
@Component({
  template: `<p>{{ text() }}</p> `,
})
export class PlaygroundComponent {
  text = signal('');

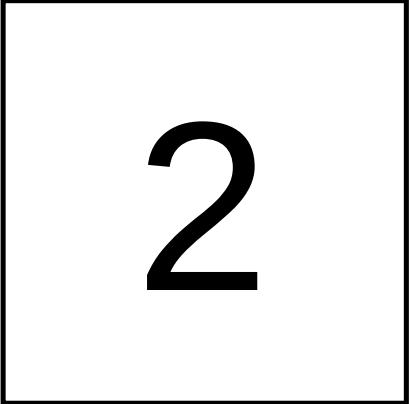
  constructor() {
    setInterval(() => {
      this.text.set(this.text() + '!');
      // or
      // this.text.update((actual) => actual + '!')
    }, 1_000);
  }
}
```

```
@Component({
  template: `<p>{{ text() }}</p> `,
})
export class PlaygroundComponent {
  text = signal('');

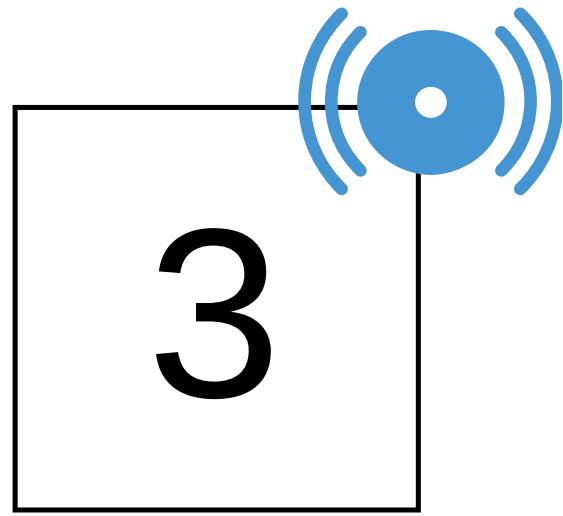
  constructor() {
    setInterval(() => {
      this.text.set(this.text() + '!');
      // or
      // this.text.update(actual => a
    }, 1_000);
  }
}
```



N = 2

N = 

N = 3



```
@Component({
  template: `
    <input (change)="setText($event)" />
    <p>{{ text() }}</p>
  `,
})
export class PlaygroundComponent {
  text = signal('');

  setText(event: Event) {
    this.text.set((event.target as HTMLInputElement).value);
  }
}
```

```
@Component({
  template: `<p>{{ text() }}</p> `,
})
export class PlaygroundComponent {
  text = signal('waiting...');

  constructor() {
    this.asyncHello().then((text) => this.text.set(text));
  }

  private asyncHello(): Promise<string> {
    // ...
  }
}
```

```
@Component({
  template: `
    <p>{{ text() }}</p>
    <p>{{ questionText() }}</p>
  `,
})
export class PlaygroundComponent {
  text = signal('');
  questionText = computed(() => this.text().replaceAll('!', '?'));

  constructor() {
    setInterval(() => {
      this.text.set(this.text() + '!');
    }, 1_000);
  }
}
```

```
@Component({
  template: `<p>{{ text() }}</p> `,
})
export class PlaygroundComponent {
  text = signal('');

  constructor() {
    setInterval(() => this.text.set(this.text() + '!'), 1_000);
    effect(() => {
      console.log('text =', this.text());
    });
  }
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  @Input() text: string;
  @Output() textChange = new EventEmitter<string>();
  @ViewChild(Alert) alerts: Alert;
  @ViewChildren(CustomCard) cards: QueryList<CustomCard>;
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text() }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  text = input('');
  @Output() textChange = new EventEmitter<string>();
  @ViewChild(Alert) alerts: Alert;
  @ViewChildren(CustomCard) cards: QueryList<CustomCard>;
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text() }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  text = input('');
  textChange = output<string>();
  @ViewChild(Alert) alerts: Alert;
  @ViewChildren(CustomCard) cards: QueryList<CustomCard>;
}
```

```
@Component({
  template: `
    <app-alert>Alert</app-alert>
    <p>{{ text() }}</p>
    <app-card />
    <app-card />
  `,
})
export class PlaygroundComponent {
  text = input('');
  textChange = output<string>();
  alerts = viewChild(Alert);
  cards = viewChildren(CustomCard);
}
```

Signals ❤





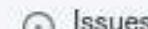
tc39 / proposal-signals



Type ⌘ to search



Code



101



11



TC
39 proposal-signals

Public

generated from [tc39/template-for-proposals](#)



Unwatch

97



Fork

54



Starred

3k

main

8 Branches

0 Tags



Go to file



Add file

Code



littledan Simpler logo (#229)

b608efa · 4 days ago

146 Commits

README

Code of conduct

MIT license

Security



JavaScript Signals standard proposal

Stage 1 ([explanation](#))

TC39 proposal champions: Daniel Ehrenberg, Yehuda Katz, Jatin Ramanathan, Shay Lewis, Kristen Hewell Garrett, Dominic Gannaway, Preston Sego, Milo M, Rob Eisenberg

Original authors: Rob Eisenberg and Daniel Ehrenberg



This document proposes a new JavaScript standard similar to the

Promises/A+ effort which preceded the Promises standardized by TC39 in ES2015. Try it for yourself, using [a polyfill](#).

@-Anthony Pena

Similarly to Promises/A+, this effort focuses on aligning the JavaScript ecosystem. If this a <https://github.com/tc39/proposal-signals>

then a standard could emerge based on that experience. Several framework authors are collaborating on this.

About

A proposal to add signals to JavaScript.

Readme

MIT license

Code of conduct

Security policy

Activity

Custom properties

3k stars

96 watching

54 forks

Report repository

Contributors 20





Zone.js c'est fini ?

Oups j'ai oublié RxJS dans tous ça 🙄 (non)



Mais au fait... C'est quoi RxJS ?

@_Anthony_Pena



RxJS

ReactiveX for JavaScript



ReactiveX

An API for asynchronous programming
with observable streams

A photograph of an elderly man with a white beard and glasses, wearing a blue and white plaid robe, sitting in a light-colored armchair. He is looking through a pair of binoculars towards a window. Outside the window, a small bird is perched on a branch. The room has a wooden floor, a large window showing a green landscape, and a wooden coffee table with a mug and a small potted plant. A sofa is visible in the background.

Parlons Observable

```
import { toObservable, toSignal } from '@angular/core/rxjs-interop';

@Component({})
export class PlaygroundComponent {
  text = signal('Hello');
  text$ = toObservable(this.text);
  textSignal = toSignal(this.text$);
}
```

Observables

Composable functions with guarantees.

Use observables for cancellation and event coordination.

- + Generally stateless
process event and clean up

- + Multiple values
a collection of events or values over time

- + Push N values

- + Convert to Signal? Yes

<https://twitter.com/BenLesh/status/175207971410039230>

Signals

A dependency graph of values and logic around notification, invalidation, computation, and memoization.

Use signals for state management.

- + Maintain state
retain things in memory

- + Single value
a value that changes over time

- + Read 1 value

- Convert to Observable? Avoidable
the event that set the signal can also notify the observable





Mais du coup les Signals...

Quelques cas concrets



HttpClient

```
@Component({
  template: `<button (click)="clicked()">Get</button> `,
})
export class PlaygroundComponent {
  http = inject(HttpClient);

  clicked() {
    this.http.get('/axolotl').subscribe();
  }
}
```

... et interceptor

```
export function authInterceptor(
  req: HttpRequest<unknown>,
  next: HttpHandlerFn
): Observable<HttpEvent<unknown>> {
  return next(
    req.clone({
      headers: req.headers.set('X-Quiz', 'Axolotl'),
    })
  );
}
```

resource & rxResource (Angular 19+)

```
@Component({
  template: `<p>{{axo.value().name}}</p>`,
})
export class PlaygroundComponent {
  axo = resource({
    loader: async () => (await fetch('/axolotl')).json(),
  });
}
```

resource & rxResource (Angular 19+)

```
@Component({
  template: `<p>{{axo.value().name}}</p>`,
})
export class PlaygroundComponent {
  http = inject(HttpClient);
  axo = rxResource({ loader: () => this.http.get('/axolotl') });
}
```

État interne des composants

```
@Component({
  template: `<p>{{ text() }}</p> `,
})
export class PlaygroundComponent {
  text = signal('');
}
```

Reactive Forms

Services



The background of the image is a dense, sun-dappled jungle. A large, rectangular stone tablet with intricate carvings and text is set into a wall, partially obscured by vines and foliage. The tablet is the central focus of the slide.

Global state management (NgRx, NgXs)



The NgXs way

Classic way

```
@Component({ ... })
export class ZooComponent {
  animals$: Observable<string[]> = this.store.select(ZooState.getAnimals);

  constructor(private store: Store) {}
}
```

Signal way

```
@Component({ ... })
export class ZooComponent {
    animals = select(ZooState.getAnimals);
}
```

The NgRx way



```
type BooksState = {
  books: Book[];
  isLoading: boolean;
  filter: { query: string; order: 'asc' | 'desc' };
};

const initialState: BooksState = {
  books: [],
  isLoading: false,
  filter: { query: '', order: 'asc' },
};

export const BooksStore = signalStore(
  useState(initialState),
  withComputed(/* ... */),
  withMethods((store, booksService = inject(BooksService)) => ({/* ... */})))
);
```

```
@Component({
  providers: [BooksStore],
})
export class BooksComponent {
  readonly store = inject(BooksStore);
}
```

The NgRx way

```
export const BooksStore = signalStore(
  withMethods((store, booksService = inject(BooksService)) => ({
    loadByQuery: rxMethod<string>(
      pipe(
        debounceTime(300),
        distinctUntilChanged(),
        tap(() => patchState(store, { isLoading: true })),
        switchMap((query) => {
          /* ... */
        })
      ),
    )));
);
```



En résumé



Les Signals c'est pour gérer les états et la réactivité dans les composants



RxJS est là pour gérer tous vos flux de données



Ben Lesh ✅

@BenLesh · Follow

A Angular folks,

Some Signals vs Observables info...

1. Use signals for state management, not observables.
2. Use observables for cancellation and other event coordination.
3. DO NOT TRY TO USE SIGNALS LIKE RXJS. It's a bad/silly idea.

They are complimentary technologies.

1/

7:05 PM · Apr 2, 2024



563



Reply



Copy link

Read 19 replies

Feedback





Anthony Pena

Développeur Web Fullstack @ **[sfΞir]**

 @_Anthony_Pena_

 @kuroidoruido

 @penaanthon

<https://k49.fr.nf>

<https://github.com/kuroidoruido/talks>





Thank you!