

WARMTEBOUW.

Angular Advanced Using Signals



Peter Kassenaar
info@kassenaar.com



Angular Signals

A new kind of reactivity with signal based primitives,

AKA "The future of Change Detection"

What are signals?

- Change Detection Changed!
 - New: reactive mechanism called *Signals*
- When a signal changes, everything that depends on that signal is also updated.

"A Signal is kind of like a BehaviorSubject, but you don't need to subscribe to it to get the value out – so you can more easily react to signal changes"

*“A **signal** is a **wrapper around a value** that notifies interested consumers when that value changes. Signals can contain any value, from primitives to complex data structures.”*

*You read a signal's value by calling its **getter** function, which allows Angular to track where the signal is used.*

*Signals may be
either **writable** or **read-only**.*

What do simple signals look like?

```
export class SignalsComponent {  
  // signal variables  
  count = signal(0);  
  double = computed(() => this.count() * 2);  
  doubleDouble = computed(() => this.double() * 2);  
  
  // methods  
  updateCount(value: number) {  
    this.count.set(this.count() + value);  
  }  
  
  resetCount() {  
    this.count.set(0);  
  }  
}
```

repo: ngx-new-features

```
<!--Value of signals -->
<h2>Count: {{ count() }}</h2>
<h2>Double: {{ double() }}</h2>
<h2>doubleDouble: {{ doubleDouble() }}</h2>
<!--Methods -->
<button (click)="updateCount(5)">Add 5 to Count</button>
<button (click)="resetCount()">Reset to 0</button>
```

Component using signals

The value 0 is a signal here. See source code for details.

Count: 15

Double: 30

doubleDouble: 60

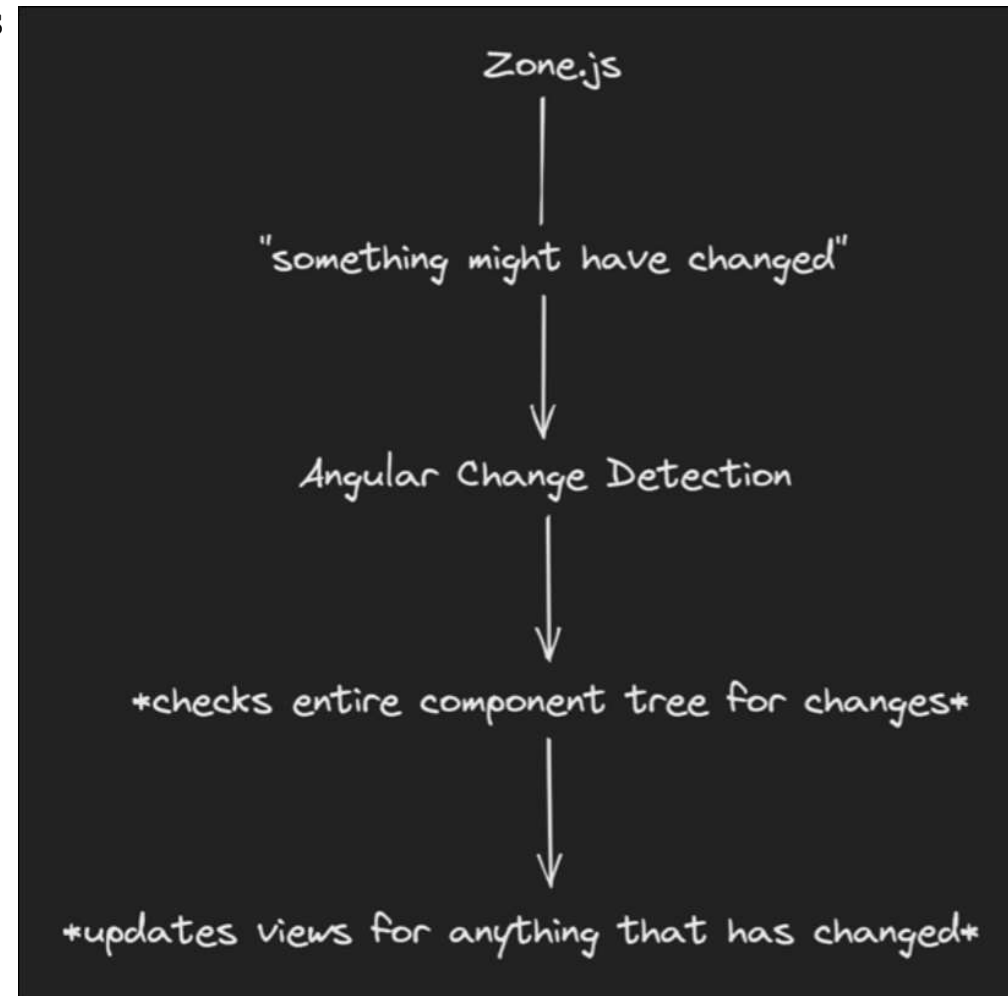
Add 5 to Count Reset to 0

Using signals

- Note the `()` – **notation** in the template.
 - `{{ signalName() }}`
 - We want the *value* of the signal. Not the variable itself
 - Hence we call the *getter function* in the template!
- Note the `computed(() => {...})` function.
 - The variable is automatically updated when the signal the variable depends on, changes.
- Signals are for *synchronous* operations.
 - Not suitable (yet?) for asynchronous statements like `setTimeout()`, http-requests and more.
 - Use RxJS for that.

On Change Detection

- Classic/Current situation: `zone.js`
 - `Zone.js` detects changes
 - `setTimeout()`
 - DOM events
 - `http-requests`
 - ...
1. Tells Angular to run CD
 2. Angular updates model + DOM
 3. Plus all nested components!
- Quite efficiently done by Angular, but still an extra layer of updates.



CD when using signal

```
export class AppComponent {  
  count = signal(0);  
  double = computed(() => this.count() * 2);  
  
  changeCount() {  
    this.count.set(5);  
  }  
}
```

I have been updated!

Angular Change Detection

Then I shall update the view for you!

Also, I see that "double" depends on you so I will recalculate and update that as well!

Source: <https://www.youtube.com/watch?v=4FkFmn0LmLI>

'Signals' in the RxJS world

- Previously: use `BehaviorSubject()` to mimic the behavior of a signal
- We have to **subscribe** to get the data out.
 - `.subscribe()` in the TypeScript file
 - `| async` in the DOM

// OLD: using RxJS BehaviorSubject:

```
count$ = new BehaviorSubject(0);  
double$ = this.count$.pipe(  
  map(count => count * 2)  
);
```

// methods on the stream

```
updateCountStream(value: number) {  
  let count = this.count$.value;  
  this.count$.next(count + value);  
}
```

```
<h2>Count$: {{ count$ | async }}</h2>  
<h2>double$ {{ double$ | async }}</h2>  
<button (click)="updateCountStream(5)">  
  Update Count stream  
</button>
```

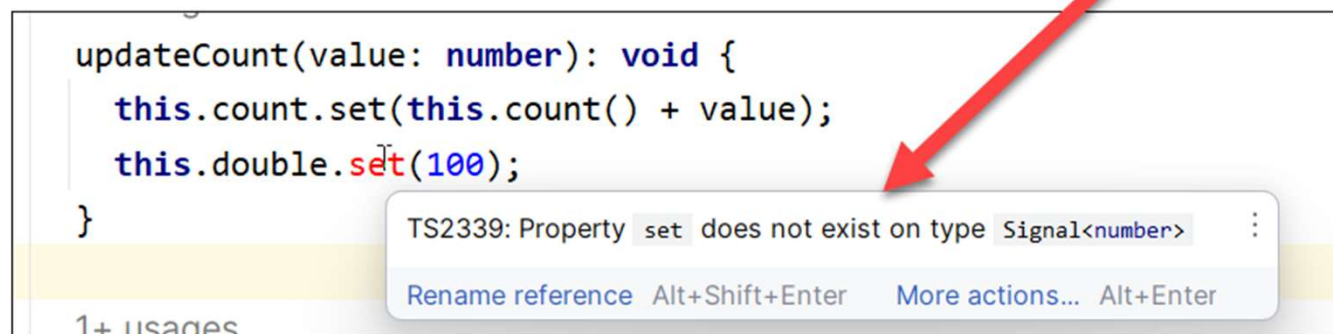
Count\$: 10

double\$ 20

Update Count stream

Updating a Signal

- **Change value** of a `WritableSignal<T>`, use `.set()`
 - For instance: `this.count.set(5);`
- Updating a value to **compute new value from the previous one**:
 - For instance: `this.count.update (value => value + 5);`
- **Don't** directly assign values to computed signals!
 - For instance: `this.double.set(5) // error`





Signal effects

Automatically run code when a signal value changes

Automatically run operation on signal change

- Effects are also a **building block** of signals:
 1. Writable Signals
 2. Computed Signals
 3. Effects

*“An **effect** is an operation that runs whenever one or more signal values change”*

Simple effect

```
constructor() {  
  effect(() => {  
    console.log('[Constructor] The count is ' + this.count);  
  });  
}
```

Component using signals

The value 0 is a signal here. See source code for details.

Count: 0

Double: 0

doubleDouble: 0

Add 5 to Count Reset to 0

Angular is running in development mode.

[Constructor] The count is [Signal: 0]

Attempting initialization Fri Mar 29 2024 14:34:26 GMT+0100 (Midden-Europese standaardtijd)

[Constructor] The count is [Signal: 5]

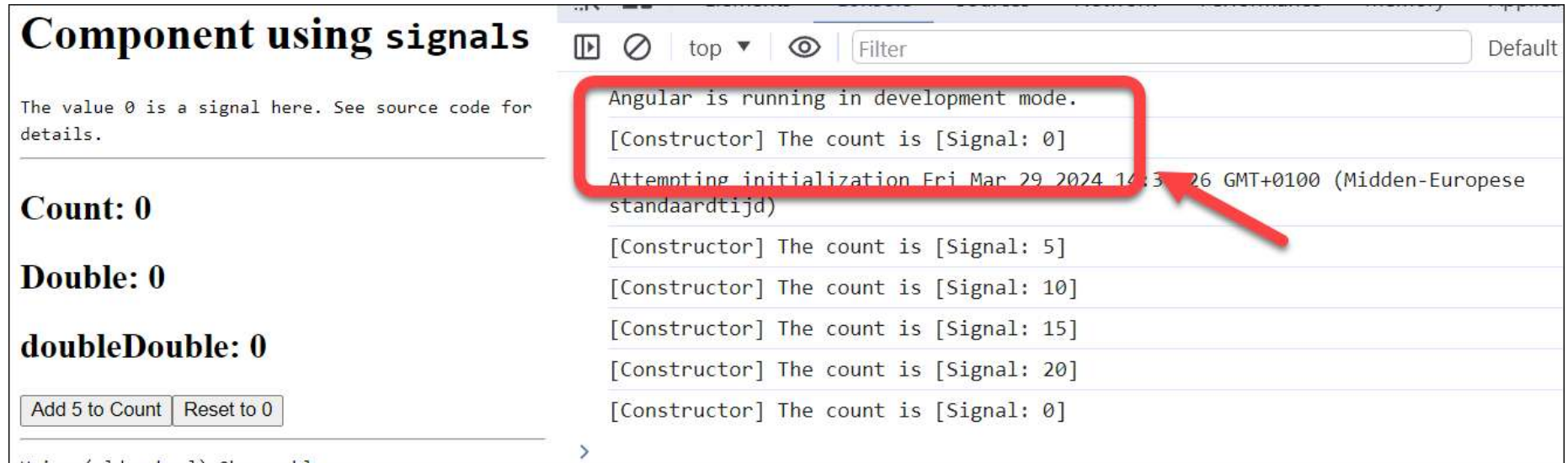
[Constructor] The count is [Signal: 10]

[Constructor] The count is [Signal: 15]

[Constructor] The count is [Signal: 20]

[Constructor] The count is [Signal: 0]

Signals run *at least once*



The screenshot shows an Angular application with a component titled "Component using signals". The component has three variables: "Count: 0", "Double: 0", and "doubleDouble: 0". Below these variables are two buttons: "Add 5 to Count" and "Reset to 0". To the right of the component is the Angular developer console, which displays a series of logs. The first log, "Angular is running in development mode.", is highlighted with a red box. The second log, "[Constructor] The count is [Signal: 0]", is also highlighted with a red box. A red arrow points to the timestamp of the second log, "Fri Mar 29 2024 14:37:26 GMT+0100 (Midden-Europese standaardtijd)".

Component using signals

The value 0 is a signal here. See source code for details.

Count: 0

Double: 0

doubleDouble: 0

Angular is running in development mode.

[Constructor] The count is [Signal: 0]

Attempting initialization Fri Mar 29 2024 14:37:26 GMT+0100 (Midden-Europese standaardtijd)

[Constructor] The count is [Signal: 5]

[Constructor] The count is [Signal: 10]

[Constructor] The count is [Signal: 15]

[Constructor] The count is [Signal: 20]

[Constructor] The count is [Signal: 0]

Whenever the signal value changes,
the effect runs again

Injection Context

- Effects can only be created inside *injection contexts*
 - Use for instance the `constructor()` of a component:

```
constructor() {  
  effect(() => {  
    console.log('[Constructor] The count is ' + this.count);  
  });  
}
```


Alternative – pass Injector

```
constructor(private injector: Injector) {  
}  
  
// 7. Effects - need an *injection context*. So the example  
// below will ONLY WORK if {injector: this.injector} is passed as parameter  
addEffect(){  
  effect(() => {  
    console.log('[addEffect] The count is ' + this.count);  
  }, { injector: this.injector});  
}
```

The value 0 is a signal here. See source code for details.

Count: 15

Double: 30

doubleDouble: 60

Add 5 to Count Reset to 0 Add an effect

Using (old school) Observables:

Angular is running in development mode.

[Constructor] The count is [Signal: 0]

Attempting initialization Fri Mar 29 2024 14:53:32 GMT+0100 (Midden-Eu
standaardtijd)

[Constructor] The count is [Signal: 5]

[addEffect] The count is [Signal: 5]

[Constructor] The count is [Signal: 10]

[addEffect] The count is [Signal: 10]

[Constructor] The count is [Signal: 15]

[addEffect] The count is [Signal: 15]

OR, using the `inject()` function

```
injector = inject(Injector);

addEffect(){
  effect(() => {
    console.log('[addEffect] The count is ' + this.count);
  }, { injector: this.injector});
}
```

Don't:

Update a signal inside an effect

- When the effect runs, the signal would be updated...
- ...this causes the effect to run, which updates the signal...
- ...we have created an **infinite loop**!

```
constructor() {  
  effect(() => {  
    console.log('[INVALID] The count is ' +  
      this.count.update(value => value+ 10));  
  });  
}
```

Angular is running in development mode.

```
✖ ERROR Error: NG0600: Writing to signals is not allowed in a `computed` or an `effect` by default. Use `allowSignalWrites` in the `CreateEffectOptions` to enable this inside effects.  
    at core.mjs:32060:15  
    at throwInvalidWriteToSignalError (signals.mjs:407:5)  
    at signalUpdateFn (signals.mjs:453:9)  
    at signalFn.update (core.mjs:13786:37)  
    at EffectHandle.effectFn (signals.component.ts:57:58)  
    at EffectHandle.runEffect (core.mjs:15164:18)
```

Workshop

- Create in your app a component that automatically shows if a value is even or odd.
 - Use a `signal()` and `computed()` for that.
 - You must be able to update the signal and show things like "1 = odd", "2= even", etc.
- Optional: use an `effect()` to log the changes to the signal.
 - So: `signal()` → State
 - `computed()` → derived state
 - `effect()` → side effects

Even / Odd Checker, using signals
2 = even

Even / Odd Checker, using signals
3 = odd

