1 Branch 23 Tags Go to file Go to file Code

brandonmcconnell 0.0.25		c8e0771 · 4 months ago
dist	Make type safe	5 months ago
.eslintrc.json	v0.0.1	6 months ago
.gitignore	v0.0.1	6 months ago
.prettierignore	v0.0.1	6 months ago
.prettierrc	v0.0.1	6 months ago
LICENSE	Initial commit	6 months ago
README.md	Update README	5 months ago
index.ts	Make type safe	5 months ago
package-lock.json	0.0.25	4 months ago
package.json	0.0.25	4 months ago
tsconfig.json	Make type safe	5 months ago

About

Signals for Tailwind CSS simplifies styling based on ancestor state via style queries. Its declarative API for signaling states eliminates complex selectors, resulting in cleaner, more maintainable code.

#tailwind #tailwindcss #tailwind-css #tailwindcss-plugin

Readme
MIT license
Activity
768 stars
5 watching
10 forks
Report repository

Releases

23 tags

README MIT license

Signals for Tailwind CSS

No packages published

Languages

TypeScript 100.0%

⚠️ This plugin is experimental and relies on style queries (via container queries), which are not yet widely supported in browsers.

The good news is that Safari and Firefox, the browsers lacking support, have already begun implementing style queries in their development versions, so it's only a matter of time before they're widely available.

See the browser compatibility table on MDN or caniuse for more information.

Signals for Tailwind CSS is a plugin that utilizes style queries (via container queries) to reactively enable a custom state, which can then be consumed by any of its descendants in the DOM.

`signal` is similar to the existing `group` variant/utility in that both provide methods for styling elements based on their ancestors' state. Unlike `group` states, however, signal states can be explicitly signaled, allowing their state to be both set and consumed with a single, simple, unchained variant.

This reduces development effort and the need to compose a chain of variants, improving the developer experience with a more declarative API.

Depending on your use case, a traditional `group` may make more sense, but often, particularly when managing a parent or ancestor state with anything more complex than a simple `peer-X` or `group-X`, a `signal` may be a simpler option.

Installation

You can install the plugin via npm:

```
npm install tailwindcss-signals
```

Then, include it in your `tailwind.config.js`:

```
module.exports = {
  plugins: [
    require('tailwindcss-signals'),
  ]
}
```

Usage

The plugin introduces the `signal` variant, which can be used to apply styles based on an ancestor's signaled state.

Here's an example comparing the traditional approach with the new signals approach:

Example: Without Signals

```
<input type="checkbox" class="peer" /> 👆 check/uncheck here
<div class="hover:[&>div]:bg-green-800 peer-checked:[&>div]:bg-green-800">
  <div class="bg-red-800 p-1 text-white">or hover here</div>
</div>
```

Open this example in Tailwind Play: https://play.tailwindcss.com/E3ig9SPTsc

Example: With Signals

```
<input type="checkbox" class="peer" /> 👆 check/uncheck here
<div class="peer-checked:signal hover:signal">
  <div class="signal:bg-green-800 bg-red-800 p-1 text-white">or hover here</div>
</div>
```

Open this example in Tailwind Play: https://play.tailwindcss.com/weFkMf4U5K

Notice how, with signals, we don't have to use any arbitrary selector variants like `[&>div]` and can instead apply those styles directly to the targeted descendants. This allows us to consolidate some redundancy in the parent so that whatever condition activates the signal only needs to be specified once rather than once per style/utility.

The benefits of Signals for Tailwind CSS become more apparent as the complexity of your styles and conditions increase.

Activating a `signal` based on a descendant condition

The general purpose of this plugin is to provide a declarative approach to applying styles based on an ***ancestor's*** state.

However, thanks to the power of the `:has()` CSS pseudo-class, we can even activate a signal based on a ***descendant's*** state.

Example: Descendant condition

```
<div class="has-[:is(input:checked,div:hover)]:signal">
  <input type="checkbox" /> 👆 check/uncheck here
```

```
  <div class="bg-red-800 p-1 text-white signal:bg-green-800">or hover here</div>
</div>
```

Open this example in Tailwind Play: https://play.tailwindcss.com/YnlzSITNqF

This is most useful for situations where you want to apply styles to an entire block based on the current state of one of its descendants.

Here are a few examples of cases where such a feature might be useful:

- Activating a signal based on the presence or visibility of a specific child element
- Activating a signal on a form based on the validity of one or more of its descendant form fields
- Activating a signal when a specific descendant element is focused or hovered
- Activating a signal based on the presence of a specific class on a descendant element
- and many more!

⚠️ Some cautions:

- Watch out for circularity issues. If you set up a signal that activates based on a descendant's state, and that descendant's state is also based on the signal, you may run into issues.
- In some cases, if you want to check if **any** descendant is focused, for example, you may not need `:has()` and could use a simpler pseudo-class variant such as…
  - `focus-within:signal` instead of `has-[:focus]:signal`
  - `valid:signal` instead of `has-[:valid]:signal` (for a `form`, which checks if all form contents are valid)
- This is a bit less declarative when you use `:has()`, but for use cases where you would need it, it would likely still be simpler than the alternative.

Differentiating signals

When using multiple signals, you may run into situations where you want one signal nested in another, which could cause issues. In that case, you can distinguish signals apart by naming them using the modifier syntax built into Tailwind CSS, the same naming convention used for `group` and `peer` variants.

Example: Naming a signal

```
<input type="checkbox" class="peer/checkable origin-bottom-left" /> 👈🏻 check/uncheck here
<div class="peer/hoverable bg-slate-700 text-white">✨ hover/unhover here ✨</div>
<div class="active:signal/custom peer-checked/checkable:signal peer-hover/hoverable:signal">
  <div class="
    text-white
    bg-red-800 after:content-['_👀']
    signal/custom:!bg-purple-800 signal:bg-green-800
    signal/custom:after:!content-['_🦄'] signal:after:content-['_😱']
  ">press me</div>
</div>
```

Open this example in Tailwind Play: https://play.tailwindcss.com/MkWvEuaWtO

By giving a signal a name, you can ensure it is unique and doesn't conflict with other signals. You can name a signal by adding a slash and the name after the `signal` variant, like `signal/{name}`.

Consuming a named signal is the same as consuming a regular signal, but with the name appended to the variant: `signal/{name}`.

For more information on this modifier syntax, see Differentiating peers from the official Tailwind CS documentation.

Why use Signals for Tailwind CSS?

Signals for Tailwind CSS provides a more declarative and straightforward approach to applying styles based on an ancestor's state. Leveraging style queries (via container queries) eliminates the need for complex selector chaining and arbitrary targeting, resulting in a cleaner and more maintainable codebase.

This plugin is particularly useful for:

- Simplifying the application of styles based on ancestor states
- Improving developer experience with a more declarative API
- Reducing the need for complex selector chaining and arbitrary targeting

Why NOT use Signals for Tailwind CSS?

⚠️ **Browser support for style queries is still limited, so Signals for Tailwind CSS may not be suitable for projects that require broad compatibility.**

The good news is that Safari and Firefox, the browsers lacking support, have already begun implementing style queries in their development versions, so it's only a matter of time before they're widely available.

See the browser compatibility table on MDN or caniuse for more information.

I hope you find `tailwindcss-signals` a valuable addition to your projects. If you have any issues or suggestions, don't hesitate to open an issue or pull request.

If you liked this, you might also like my other Tailwind CSS plugins:

- tailwindcss-multi: Group utilities together by variant
- tailwindcss-mixins: Construct reusable & aliased sets of utilities inline
- tailwindcss-members: Apply styles based on child or descendant state, the inverse of groups
- tailwindcss-selector-patterns: Dynamic CSS selector patterns
- tailwindcss-js: Effortless build-time JS script injection
- tailwindcss-directional-shadows: Supercharge your shadow utilities with added directional support (includes directional `shadow-border` utilities too ✨)