Svelte 3: Rethinking reactivity

Rich Harris

After several months of being just days away, we are over the moon to announce the stable release of Svelte 3. This is a huge release representing hundreds of hours of work by many people in the Svelte community, including invaluable feedback from beta testers who have helped shape the design every step of the way.

We think you're going to love it.

What is Svelte?

Svelte is a component framework — like React or Vue — but with an important difference. Traditional frameworks allow you to write *declarative* state-driven code, but there's a penalty: the browser must do extra work to convert those declarative structures into DOM operations, using techniques like virtual DOM diffing that eat into your frame budget and tax the garbage collector.

Instead, Svelte runs at *build time*, converting your components into highly efficient *imperative* code that surgically updates the DOM. As a result, you're able to write ambitious applications with excellent performance characteristics.

The first version of Svelte was all about testing a hypothesis — that a purpose-built compiler could generate rock-solid code that delivered a great user experience. The second was a small upgrade that tidied things up a bit.

Version 3 is a significant overhaul. Our focus for the last five or six months has been on delivering an outstanding *developer* experience. It's now possible to write components with significantly less boilerplate than you'll find elsewhere. Try the brand new tutorial and see what we mean — if you're familiar with other frameworks we think you'll be pleasantly surprised.

To make that possible we first needed to rethink the concept at the heart of modern UI frameworks: reactivity.

```
<iframe style="position: absolute; width: 100%; height: 100%; left: 0; top: 0; margin: 0;" s
```

'Rethinking Reactivity' from You Gotta Love Frontend Code Camp

Moving reactivity into the language

In old Svelte, you would tell the computer that some state had changed by calling the `this.set` method:

```
const { count } = this.get();
this.set({
    count: count + 1
});
```

That would cause the component to *react*. Speaking of which, `this.set` is almost identical to the `this.setState` method used in classical (pre-hooks) React:

```
const { count } = this.state;
this.setState({
    count: count + 1
});
```

There are some important technical differences (as I explain in the video above, React is not reactive) but conceptually it's the same thing.

In fact, Svelte 3 is basically Sunil's fault.

That all changed with the advent of hooks, which handle state in a very different fashion. Many frameworks started experimenting with their own implementations of hooks, but we quickly concluded it wasn't a direction we wanted to go in. Hooks have some intriguing properties, but they also involve some unnatural code and create unnecessary work for the garbage collector. For a framework that's used in embedded devices as well as animation-heavy interactives, that's no good.

So we took a step back and asked ourselves what kind of API would work for us... and realised that the best API is no API at all. We can just *use the language*. Updating some `count` value — and all the things that depend on it — should be as simple as this:

```
count += 1;
```

Since we're a compiler, we can do that by instrumenting assignments behind the scenes:

```
count += 1; $$invalidate('count', count);
```

Importantly, we can do all this without the overhead and complexity of using proxies or accessors. It's just a variable.

New look

Your components aren't the only thing that's getting a facelift. Svelte itself has a completely new look and feel, thanks to the amazing design work of Achim Vedam who created our new logo and website, which has moved from svelte.technology to svelte.dev.

We've also changed our tagline, from 'The magical disappearing UI framework' to 'Cybernetically enhanced web apps'. Svelte has many aspects — outstanding performance, small bundles, accessibility, built-in style encapsulation, declarative transitions, ease of use, the fact that it's a compiler, etc — that focusing on any one of them feels like an injustice to the others. 'Cybernetically enhanced' is designed to instead evoke Svelte's overarching philosophy that our tools should work as intelligent extensions of ourselves — hopefully with a retro, William Gibson-esque twist.

Upgrading from version 2

If you're an existing Svelte 2 user, I'm afraid there is going to be some manual upgrading involved. In the coming days we'll release a migration guide and an updated version of svelte-upgrade which will do the best it can to automate the process, but this *is* a significant change and not everything can be handled automatically.

We don't take this lightly: hopefully once you've experienced Svelte 3 you'll understand why we felt it was necessary to break with the past.

Still to come

As grueling as this release has been, we're nowhere near finished. We have a ton of ideas for generating smarter, more compact code, and a long feature wish-list. Sapper, our Next.js-style app framework, is still in the middle of being updated to use Svelte 3. The Svelte Native community project, which allows you to write Android and iOS apps in Svelte, is making solid progress but deserves more complete support from core. We don't yet have the bounty of editor extensions, syntax highlighters, component kits, devtools and so on that other frameworks have, and we should fix that. We *really* want to add first-class TypeScript support.

But in the meantime we think Svelte 3 is the best way to build web apps yet. Take an hour to go through the tutorial and we hope to convince you of the same. Either way, we'd love to see you in our Discord chatroom and on GitHub — everyone is welcome, especially you.