Plans for the Next Iteration of Vue.js

This article is from Evan You on medium

Last week at Vue.js London I gave a brief sneak peek of what's coming in the next major version of Vue. This post provides an in-depth overview of the plan.

Why a new majorversion?

Vue 2.0 was released exactly two years ago (how time flies!). During this period, the core has remained backwards compatible with five minor releases. We've accumulated a number of ideas that would bring improvements, but they were held off because they would result in breaking changes. At the same time, the JavaScript ecosystem and the language itself has been evolving rapidly. There are greatly improved tools that could enhance our workflow, and many new language features that could unlock simpler, more complete, and more efficient solutions to the problems Vue is trying to solve. What's more exciting is that we are seeing ES2015 support becoming a baseline for all major evergreen browsers. Vue 3.0 aims to leverage these new language features to make Vue core smaller, faster, and more powerful.

Vue 3.0 is currently in prototyping phase, and we have already implemented a runtime close to feature-parity with 2.x.**Many of the items listed below are either already implemented, or confirmed to be feasible. Ones that are not yet implemented or still in exploration phase are marked with a *.**

The Details

High-Level APIChanges

TL;DR: Everything except render function API and scoped-slots syntax will either remain the same or can be made 2.x compatible via a compatibility build.

Since it's a new major, there is going to be some breaking changes. However, we take backwards compatibility seriously, so we want to start communicating these changes as soon as possible. Here's the currently planned public API changes:

- Template syntax will remain 99% the same. There may be small tweaks in scoped slots syntax, but other than that we have no plans to change anything else for templates.

- 3.0 will support class-based components natively, with the aim to provide an API that is pleasant to use in native ES2015 without requiring any transpilation or stage-x features. Most current options will have a reasonable mapping in the class-based API. Stage-x features such as class fields and decorators can still be used optionally to enhance the authoring experience. In addition, the API is designed with TypeScript type inference in mind. The 3.x codebase will itself be written in TypeScript, and providing improved TypeScript support. (That said, usage of TypeScript in an application is still entirely optional.)
- The 2.x object-based component format will still be supported by internally transforming the object to a corresponding class.
- Mixins will still be supported.*
- Top level APIs will likely receive an overhaul to avoid globally mutating the Vue runtime when installing plugins. Instead, plugins will be applied and scoped to a component tree. This will make it easier to test components that rely on specific plugins, and also make it possible to mount multiple Vue applications on the same page with different plugins, but using the same Vue runtime.*
- Functional components can finally be plain functions —however, async components will now need to be explicitly created via a helper function.
- The part that will receive the most changes is the Virtual DOM format used in render functions. We are currently collecting feedback from major library authors and will be sharing more details as we are more confident of the changes, but as long as you don’t heavily rely on hand-written (non-JSX) render functions in your app, upgrading should be a reasonably straightforward process.

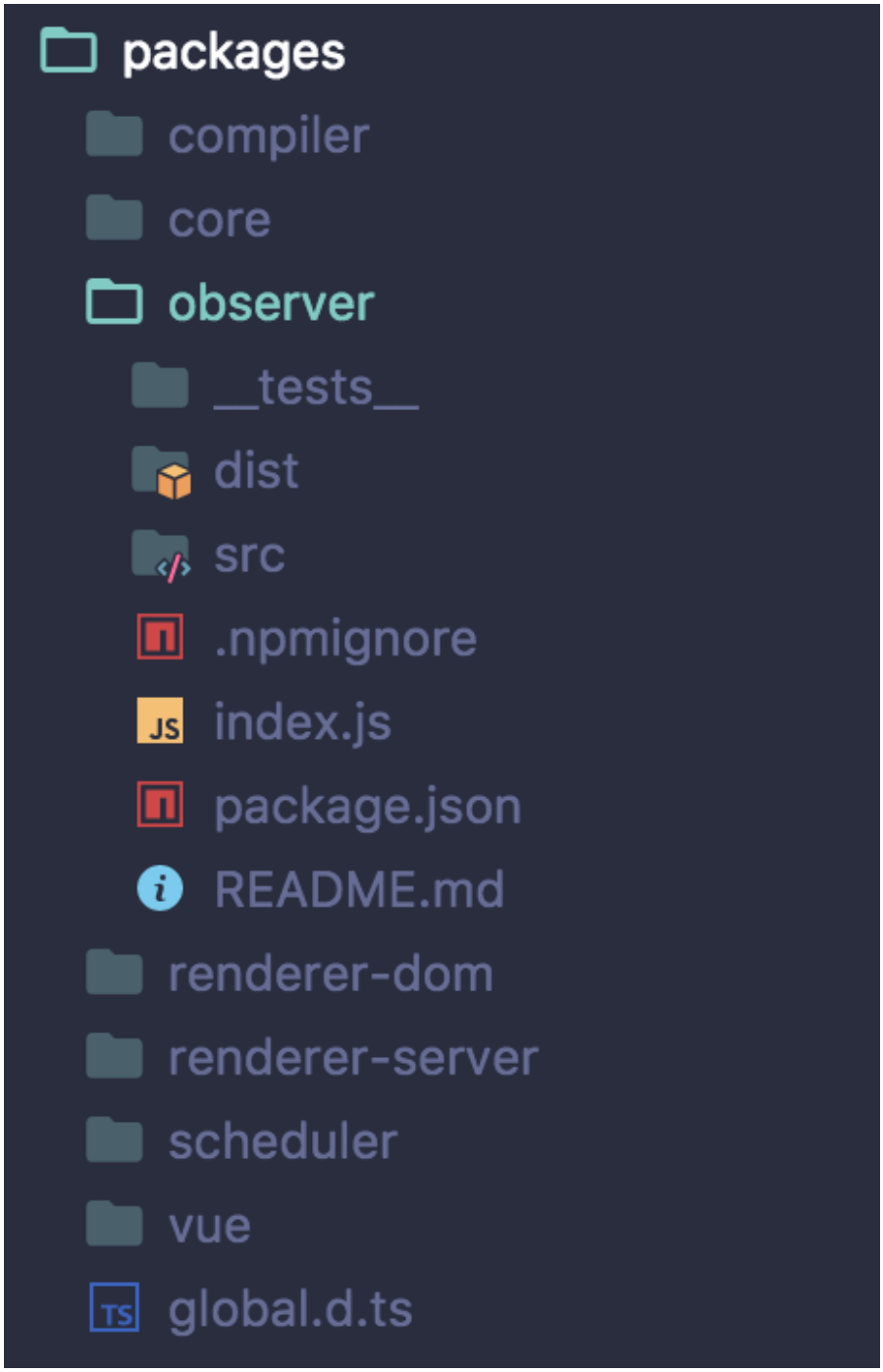
Source Code Architecture

TL;DR: better decoupled internal modules, TypeScript, and a codebase that is easier to contribute to.

We are re-writing 3.0 from the ground up for a cleaner and more maintainable architecture, in particular trying to make it easier to contribute to. We are breaking some internal functionalities into individual packages in order to isolate the scope of complexity. For example, the observer module will become its own package, with its own public API and tests. Note this does not affect framework-level API—you will not have to manually import individual bits from multiple packages in order to use Vue. Instead, the final Vue package is assembled using these internal packages.

The codebase is also now written in TypeScript. Although this will make proficiency in TypeScript a pre-requisite for contributing to the new codebase, we believe the type information and IDE support will actually make it easier for a new contributor to make meaningful contributions.

Decoupling the observer and scheduler into separate packages also allows us to easily experiment with alternative implementations of these parts. For example, we can implement an IE11 compatible observer implementation with the same API, or an alternative scheduler that leverages `requestIdleCallback` to yield to the browser during long updates.*



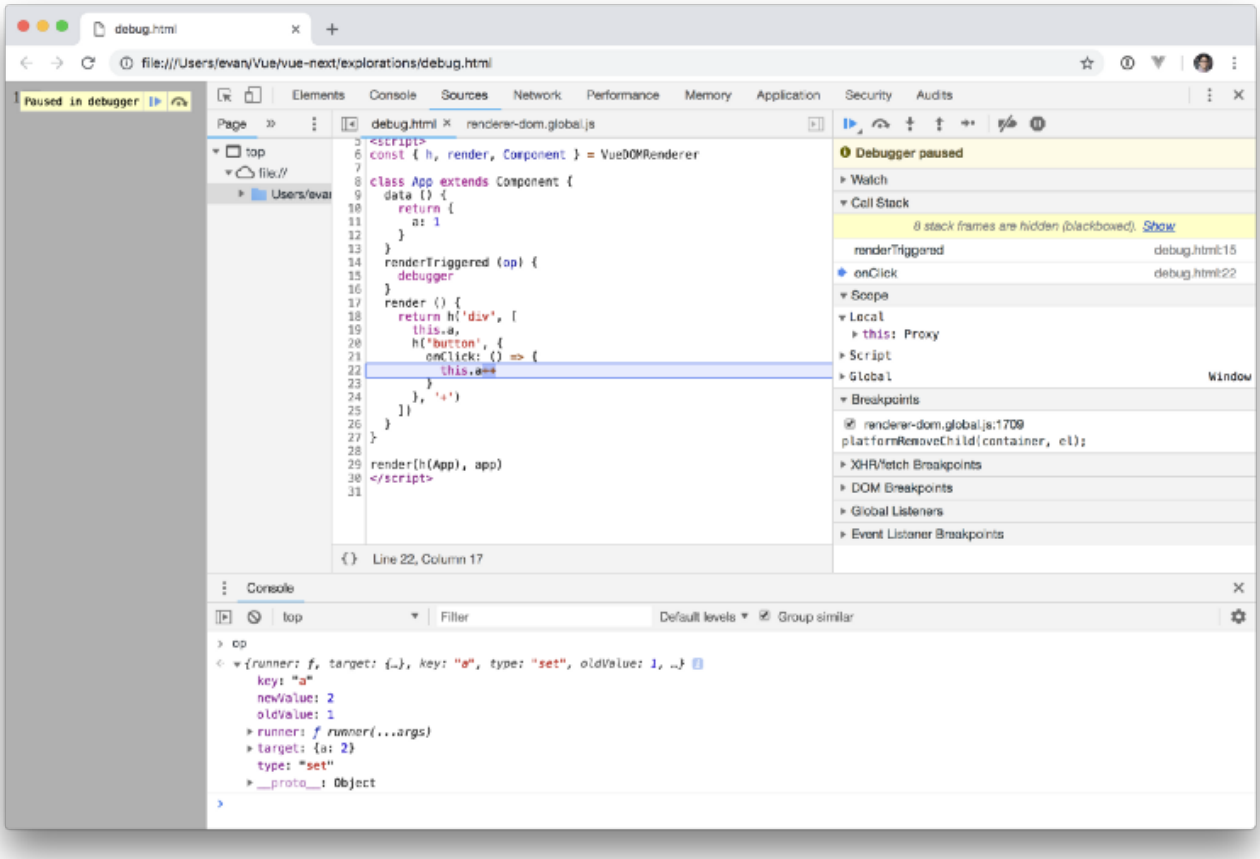
Observation Mechanism

TL;DR: more complete, precise, efficient and debuggable reactivity tracking & API for creating observables.

3.0 will ship with a Proxy-based observer implementation that provides reactivity tracking with full language coverage. This eliminates a number of limitations of Vue 2's current implementation based on `Object.defineProperty`:

The new observer also features the following:

Easily understand why a component is re-rendering



Other Runtime Improvements

TL;DR: smaller, faster, tree-shakable features, fragments & portals, custom renderer API.

Compiler Improvements*

TL;DR: tree-shaking friendly output, more AOT optimizations, parser with better error info and source map support.

IE11 Support*

TL;DR: it will be supported, but in a separate build with the same reactivity limitations of Vue 2.x.

The new codebase currently targets evergreen browsers only and assumes baseline native ES2015 support. But alas, we know a lot of our users still need to support IE11 for the foreseeable future. Most of the ES2015 features used can be transpiled / polyfilled for IE11, with the exception for Proxies. Our plan is to implement an alternative observer with the same API, but using the good old ES5 `Object.defineProperty` API. A separate build of Vue 3.x will be distributed using this observer implementation. However, this build will be subject to the same change detection caveats of Vue 2.x and thus not fully compatible with the "modern" build of 3.x. We are aware that this imposes some inconvenience for library authors as they will need to be aware of compatibility for two different builds, but we will make sure to provide clear guidelines on this when we reach that stage.

How Do We GetThere

First of all, although we are announcing it today, we do not have a definitive timeline yet. What we do know at the moment is the steps we will be taking to get there:

1. Internal Feedback for the Runtime Prototype

This is the phase we are in right now. Currently, we already have a working runtime prototype that includes the new observer, Virtual DOM and component implementation. We have invited a group of authors of influential community projects to provide feedback for the internal changes, and would like to make sure they are comfortable with the changes before moving forward. We want to ensure that important libraries in the ecosystem will be ready at the same time when we release 3.0, so that users relying on those projects can upgrade easily.

2. Public Feedback viaRFCs

Once we gain a certain level of confidence in the new design, for each breaking change we will be opening a dedicated RFC issue which includes:

We will anticipate public feedback from the wider community to help us consolidate these ideas.

3. Introduce Compatible Features in 2.x &2.x-next

We are not forgetting about 2.x! In fact, we plan to use 2.x to progressively accustom users to the new changes. We will be gradually introducing confirmed API changes into 2.x via opt-in adaptors, and 2.x-next will allow users to try out the new Proxy-based observer.

The last minor release in 2.x will become LTS and continue to receive bug and security fixes for 18 months when 3.0 is released.

4. AlphaPhase

Next, we will finish up the compiler and server-side rendering parts of 3.0 and start making alpha releases. These will mostly be for stability testing purposes in small greenfield apps.

5. BetaPhase

During beta phase, our main goal is updating support libraries and tools like Vue Router, Vuex, Vue CLI, Vue DevTools and make sure they work smoothly with the new core. We will also be working with major library authors from the community to help them get ready for 3.0.

6. RCPhase

Once we consider the API and codebase stable, we will enter RC phase with API freeze. During this phase we will also work on a “compat build”: a build of 3.0 that includes compatibility layers for 2.x API. This build will also ship with a flag you can turn on to emit deprecation warnings for 2.x API usage in your app. The compat build can be used as a guide to upgrade your app to 3.0.

7. IE11build

The last task before the final release will be the IE11 compatibility build as mentioned above.

8. FinalRelease

In all honesty, we don’t know when this will happen yet, but likely in 2019. Again, we care more about shipping something that is solid and stable rather than hitting specific dates. There is a lot of work to be done, but we are excited for what’s coming next!