# 重新发明 Vue：经验和教训

尤雨溪 Evan You
JetBrains 码上道 May 2024

# Vue 的发展简史

- 2013 年第一次发布带有 VueJS 名字的版本
- 2014 年第一次公开宣传
- 2015 年 10 月 :1.0 版本发布
- 2016 年 10 月 :2.0 版本发布
- 2018 年 9 月 :3.0 开始开发
- 2020 年 9 月 :3.0 软发布
- 2022 年 1 月 :3.x 正式成为默认版本

# Vue 的发展简史

- 2013 年第一次发布带有 VueJS 名字的版本
- 2014 年第一次公开宣传
- 2015 年 10 月 : 1.0 版本发布
- 2016 年 10 月 : 2.0 版本发布
- 2018 年 9 月 : 3.0 开始开发
- 2020 年 9 月 : 3.0 软发布
- 2022 年 1 月 : 3.x 正式成为默认版本

Vue 3 “正式完成”
花费了三年多！

# Vue 2 面对的问题

- 代码架构
- 性能优化空间
- API 在大型项目中的可维护性
- 浏览器版本限制

# 代码架构

- 当时还在使用 Flow 而不是 TypeScript，类型检查不完全，IDE 支持不佳
- 需要手动维护 dts 类型声明文件
- 内部一些模块界限模糊

# 性能优化空间

- 几乎是纯虚拟 DOM，很多 overhead
- 编译器结构过于简单，能够进行的分析有限
- 组件实例化开销较大

# API

- Options API 虽然易用，但限制了代码的可重构性，在大型、长期的项目中，可维护性展现出问题。
- Options API 对类型推导不友好，边界情况多，类型实现维护难度高
- 逻辑抽取复用依赖 mixin，容易导致属性来源不清晰

# 浏览器版本限制

- 最低支持 IE9
- 无法使用不能被 polyfill ES2015+ 的新语言特性
- 为了避免 polyfill 的性能影响或是 babel 生成代码对打包尺寸的影响，维护时有很多额外的心智负担

# Vue 3 的目标

- 代码架构
  - 迁移到 TypeScript + 自动生成类型声明
  - 重新设计内部模块分层
  - 为以后的长期维护打好基础
- 性能
  - VDOM 算法重构
  - 结合编译器对虚拟 VDOM 进行优化
  - 优化组件实例化开销
- API
  - 引入对重构、复用、类型推导更友好的新 API
- 浏览器
  - 语言支持最低要求 ES2015+

# 开发 Vue 3 遇到的挑战

- 寻找性能优化突破点
- 探索新的 API 设计
- 周边配套设置工作量巨大

# 寻找性能优化突破点

- 彻底重构 VDOM 算法
- 花费了比较久的时间才找到并验证了编译 + 运行时结合的优化模式

# 探索新的 API 设计

- 尝试 Class API
  - 因为过于依赖还在 stage 2 的 decorators 提案而放弃
- Composition API
  - 因为风格迥异引发争议
  - <script setup> 稳定前 DX 欠佳

# 周边配套设施工作量巨大

- 文档
- 构建工具 (Vue CLI / Vite)
- Vue Router
- Vuex -> Pinia
- Devtools
- IDE 支持 (Vetur -> Volar)
- SFC type check (vue-tsc)
- Lint (eslint-plugin-vue)
- Unit testing (@vue/test-utils)

# 社区迁移的痛

- 2 到 3 的升级难度不低
- 许多用户仍然卡在 Vue 2

# 对一些错误决定的反思

## 失误 #1

## 太多微小的 Breaking Change

- 大部分的破坏性改动，单独看都不难处理
- 然而在实际应用中需要同时面对所有改动的时候，复杂度就指数上升

## 经验

- 不要一次性发布大量破坏性改动，将小的破坏性改动分批慢慢发布
- 在保证可运行的基础上，采用 deprecate -> opt-in -> remove 的策略

## 失误 #2

## 低估了对生态中依赖的影响

- 大部分的破坏性改动在讨论时都只关注了对应用层代码的影响
- 许多大型依赖用到了私有 API，在 Vue 3 中改动较多
- 库升级困难，导致依赖这些库的项目也升级困难

## 经验

- 生态中的库是框架升级的重要考量 - 破坏性改动应该优先考虑对库的影响
- 引导用户避免使用私有 API

# 失误 #3
# 发布节奏的处理

- 2020 年 9 月 Vue 3 core 发布 3.0，但配套设施还未完善
  - 临时文档比较粗糙，缺少 Composition API 作为一等公民的学习流程
  - Router / Vuex 还在 beta / rc
  - Migration Build 尚未提供
  - 浏览器调试插件尚未支持
  - <script setup> 还未稳定

## 失误 #3
## 发布节奏的处理

- 单独发布 core 是希望能鼓励 library 作者和 early adopters 开始适配升级
- 但对普通开发者而言是一个比较令人困惑的体验

## 经验

- 第一印象很重要，宁可推迟，不要发布半成品
- 在正式发布前积极向社区库的维护者寻求协作，帮助他们提前适配

# 做对了的事

正确决定 #1

# 拥抱 TypeScript

- 良好的类型支持已经是现代框架必须的功能
- Vue 本身的可维护性也大大提升，为后续的迭代打下了坚实的基础

## 正确决定 #2

# 坚持 Composition API

- 3.2 <script setup> 提升开发体验后，用户接受度大大提高
- 切实提升了可重构性和可维护性
- 逻辑复用：社区涌现出了如 VueUse 这样的优秀项目

## 正确决定 #3

# 对开发体验持续投入

- 在 Vite 上的投入虽然超出预期，但值得
- Vue 3 新文档对现有内容进行了大量的重写和结构调整
- Volar：大大提升 Vue SFC 的 TypeScript 支持
  - 通过 LSP (Language Server Protocol) 支持了包括 WebStorm, Neovim 等 IDE

# Vue 3 确实更好更强大

- 虽然花了比预想要长得多的时间，但基本达成了当初的目标
- 目前已经接近全部 vue 下载量的 60%
- 2022 年初切换为默认至今下载量翻了 8.5 倍
- 过去一年增长 +85%
- 生态逐渐成熟丰富：https://ui-libs.vercel.app/

# 未来：近期到中期

- 过渡还未完成，稳定优先
- 不再会有 2 到 3 这种程度的破坏性升级
- 短期内不会推动“彻底改变现有范式”的新 API
- 专注于不影响现有代码的改进
  - Vapor Mode
    - 新的编译策略：同样的模版，更优化的渲染性能
    - 组件级别的 opt-in，不影响现有代码
  - parser 性能优化 (3.4)
  - 响应式系统效率优化 (3.4 & 3.5)
  - SSR hydration 优化 (3.5+)

# 未来：长期

- 框架之间逐渐形成底层 primitive（原语）的共识，并合作推进标准化
  - Reactive 状态管理：Signals (Stage 1)
  - Native CSS @scope
  - DOM Parts
- Vue 会积极跟进和关注这些标准化努力的进展，并在未来利用平台原生能力简化内部实现和提升性能

# 未来：框架之外

- Vite 已经成为跨框架的基础设施，但需要更坚实的底层基础

- Rolldown: Rust bundler for Vite
  - 目前已经完成对标 esbuild 的打包功能，并且性能更优
  - WIP: 基于 OXC 的内置 TS 转译、压缩等功能
  - WIP: splitChunks, module federation 等现在 Vite/Rollup 不具备的功能

- OXC: the fastest language toolchain for JS
  - Parser, resolver, linter, transformer, formatter, minifier…

# Vite + Rolldown + OXC =
# 下一代 JS 基础工具链

# Thank you!