Vue 的进化历程

尤雨溪

VueConf China 2022

库阶段
Library Phase (Pre 1.0)
2013-2015

2013.12 - 第一个以 “Vue.js” 命名的版本 (0.6.0)

2013.12 - 第一个以 “Vue.js” 命名的版本 (0.6.0)

2014.02 - 第一次在 HackerNews 上公开发布

2013.12 - 第一个以 “Vue.js” 命名的版本 (0.6.0)

2014.02 - 第一次在 HackerNews 上公开发布

2014.10 - 第一次实现 Vue SFC 单文件组件 (vueify)

2013.12 - 第一个以 “Vue.js” 命名的版本 (0.6.0)

2014.02 - 第一次在 HackerNews 上公开发布

2014.10 - 第一次实现 Vue SFC 单文件组件 (vueify)

2014.11 - 第一次完全重写 (0.11)

库阶段的设计重点

- 基于 ES5 getter/setters 和原生 JS 对象的响应式系统
- MVVM / 模版数据绑定
- 像 jQuery 一样可以直接通过 <script> 引用的简单库

库阶段的特征

- (还)不算是个框架
- 受 Backbone / Ractive 影响的 API:
  - 响应式系统和组件实例有很强的耦合，影响逻辑复用
  - 直到 0.11 才引入 Mixins
- 还在摸索完善模版语法和作用域规则

库阶段的特征

- 基于 DOM 的渲染机制
  - 没有“编译”过程
  - 模版直接实例化为 DOM 树
  - 通过遍历 DOM 树实现数据绑定
  - 类似现在 petite-vue 的实现

框架阶段

Framework Phase (1.x)

2015-2016

2015.08 - 第一版 Vue Router 发布

2015.08 - 第一版 Vue Router 发布

2015.09 - 开始开发 1.0

2015.08 - 第一版 Vue Router 发布

2015.09 - 开始开发 1.0

2015.10.26 - 1.0 “Evangelion” 发布

2015.08 - 第一版 Vue Router 发布

2015.09 - 开始开发 1.0

2015.10.26 - 1.0 “Evangelion” 发布

2015.12 - 第一版 vue-cli 发布

2015.08 - 第一版 Vue Router 发布

2015.09 - 开始开发 1.0

2015.10.26 - 1.0 “Evangelion” 发布

2015.12 - 第一版 vue-cli 发布

2016.03 - 第一版 Vuex 发布

框架阶段的设计重点

- 稳定模版语法和作用域设计
  - 确定了 v-bind, v-on 和对应简写的语法
  - 第一次引入 v-for (取代了 v-repeat)
- 将项目的涵盖范畴扩大到 SPA 框架
  - SPA 路由
  - 状态管理
  - 工具链
    - 热更新
    - Scoped CSS

通用框架阶段

Universal Framework Phase (2.x)

2016-2019

2016.03 - 第一次明确提出 “渐进式框架” 的概念

2016.03 - 第一次明确提出 “渐进式框架” 的概念

2016.04 - 开始开发 2.0

2016.03 - 第一次明确提出 “渐进式框架” 的概念

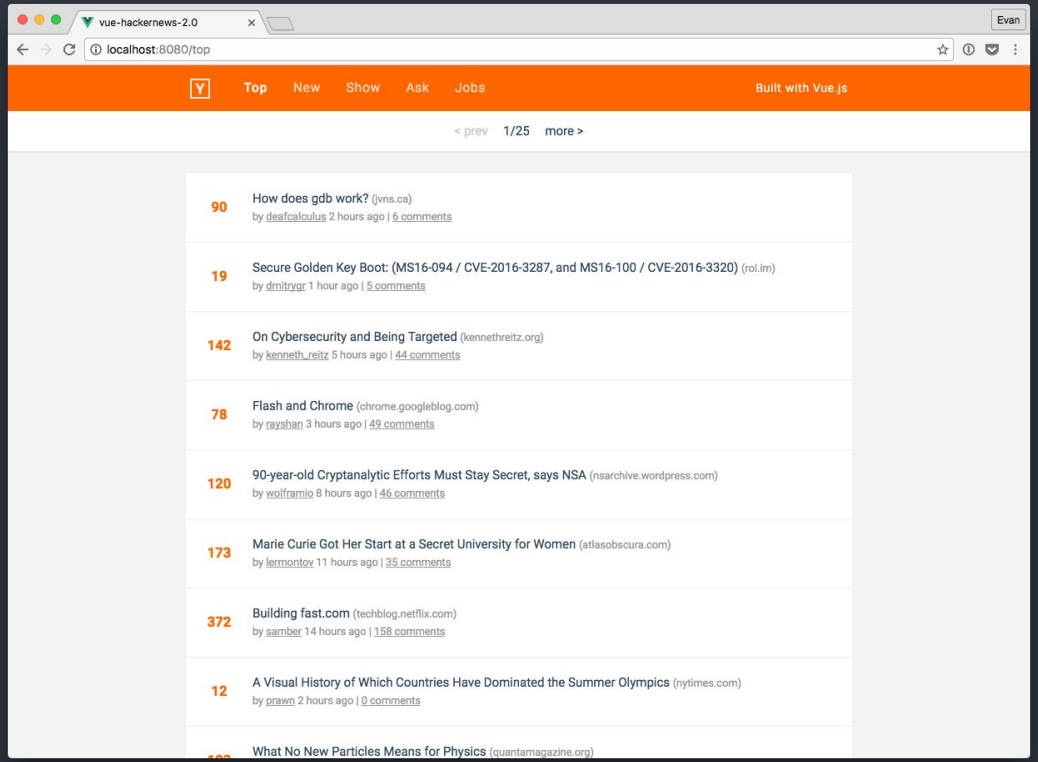
2016.04 - 开始开发 2.0

2016.10.01 - 2.0 “Ghost in the Shell” 发布

2.0 阶段的设计重点

- 第二次彻底重写，代码架构改进
- 引入了将模版编译为 Virtual DOM 渲染函数的编译流程
- 基于 Virtual DOM 的服务端渲染 (SSR)
- 基于 Virtual DOM 的跨端渲染 (Weex, NativeScript)
- 和类型系统的结合
  - 源码中使用 Flow
  - 手动 TypeScript 类型定义

vue-hackernews-2.0



vue-hackernews-2.0

- webpack + SFC + Vue Router + Vuex + SSR
- 第一个完整展示 Vue 2 SSR 架构的 demo
- Vue 2 SSR 相关功能的测试平台
- 启发了上层 SSR 框架如 Nuxt

2016.11 - 2.1 “Hunter X Hunter” - 作用域插槽

2016.11 - 2.1 “Hunter X Hunter” - 作用域插槽

2017.02 - 2.2 “Initial D” - SSR 支持基于路由的代码分割

2016.11 - 2.1 “Hunter X Hunter” - 作用域插槽

2017.02 - 2.2 “Initial D” - SSR 支持基于路由的代码分割

2017.04 - 2.3 “JoJo” - SSR 支持基于路由的资源预加载

2016.11 - 2.1 “Hunter X Hunter” - 作用域插槽

2017.02 - 2.2 “Initial D” - SSR 支持基于路由的代码分割

2017.04 - 2.3 “JoJo” - SSR 支持基于路由的资源预加载

2017.06 - 2.4 “Kill la Kill” - SSR 完整异步组件支持

\+ 部分优化的 SSR 编译输出

2018.01-08 - Vue CLI 3.0

- 针对 SPA 的高度集成工具链
- 进一步拓展框架的边界
  - “工具链也是框架的一部分”

编译/运行时混合阶段
Compiler/Runtime Hybrid Phase (3.x)

2019-now

2018.09 - 在 Vue.js London 远程宣布 Vue 3 开发计划

2018.09 - 在 Vue.js London 远程宣布 Vue 3 开发计划

2018.09 - 2019.05 - **调研阶段**

3.0 重构初期的重心

- 提高浏览器最低支持要求，使用现代 ES 语法和功能
- 全面提升性能
- 改善类型系统整合
- 改善在大型应用中的可扩展性

2018.09 - 在 Vue.js London 远程宣布 Vue 3 开发计划

2018.09 - 2019.05 - 调研阶段

2019.05 - 基于编译优化 VDOM 性能的新策略

2018.09 - 在 Vue.js London 远程宣布 Vue 3 开发计划

2018.09 - 2019.05 - 调研阶段

2019.05 - 基于编译优化 VDOM 性能的新策略

2019.08 - Composition API RFC

Composition API 的意义

- Vue 的用例越来越多进入企业、大型项目领域
- Options API 在可扩展性方面有明显上限
- 对类型系统更友好
- 提供灵活且可维护的逻辑组合/复用

2020.01 - 3.0 alpha

2020.01 - 3.0 alpha

2020.04 - 3.0 beta - 完全优化的 SSR 编译输出

2020.01 - 3.0 alpha

2020.04 - 3.0 beta - 完全优化的 SSR 编译输出

2020.04 - 2021.02 - 绕道开发了 Vite

Vite 的意义

- 大幅优化开发体验
- 将和框架没有耦合的工具链职责剥离，交给一个更大的社区去维护
- 缩减 Vue 本身的框架范畴和维护负担
  - Vue CLI -> create-vue

2020.09 - 3.0 稳定版发布

Hybrid Phase (3.0)

2020.09 - 3.0 稳定版发布

2021.06 - 3.1 Migration Build

2020.09 - 3.0 稳定版发布

2021.06 - 3.1 Migration Build

2021.08 - 3.2 <script setup>

2022.01 - Vue 3 切换为默认版本

2022.02 - 全新的 Vue 3 文档

Vue 3 目前所定义的框架范畴

- 核心（运行时 + 编译器）
- 文档
- 工具链 (create-vue)
- SPA 路由 (Vue Router)
- 状态管理 (Pinia)
- 浏览器开发工具 (vue-devtools)
- IDE 支持 (Volar)
- TypeScript 支持 (vue-tsc)
- 静态分析 (eslint-plugin-vue)
- 单元测试 (@vue/test-utils)

向编译/运行时混合模式进化

- 同一份模版，不同的编译输出
  - 浏览器: 高度优化的 Virtual DOM 渲染函数
  - SSR: buffer + 字符串拼接
  - 将来: Vapor mode (无 Virtual DOM 的渲染代码)
- 单文件组件语法糖
  - <script setup>
  - CSS v-bind()
  - Reactivity Transform

现状

- 社区仍然处于 2 -> 3 的过渡阶段
- 2022 年 6 月发布的 2.7，进一步弥补 2 和 3 之间的断层
- 基于目前的 npm 数据:
  - 超过 30% 的项目在使用 Vue 3
  - 大约 25% 在用 2.7
  - 超过一半的项目可以使用 Composition API 和 <script setup>

展望

- 接下来的阶段将以 API 稳定性为优先
- 重点会放在不影响使用方式的改进上
- 不计划引入类似 React Server Components 的需要和服务器强绑定的特性

展望

- 短期
  - Reactivity Transform / Suspense 稳定化
  - SSR 水合改进 (lazy / on-demand / server-only)
- 中到长期
  - Vapor mode (受 Solid 启发的模版编译策略)

Vapor Mode

- 完全一样的模版/组件语法
- 输出不再依赖 Virtual DOM 运行时
- 极致的性能和内存占用
- 零成本组件抽象
- 无缝嵌入在现有应用中（兼容所有基于 VDOM 的第三方库）
- 或者: 启用 Vapor-only (不兼容 VDOM)
- 明年公布更多信息

Thank you!