对比其它框架

Angular

选择 Vue 而不选择 Angular，有下面几个原因，当然不是对每个人都适合：

- 在 API 与设计两方面上 Vue.js 都比 Angular 简单得多，因此你可以快速地掌握它的全部特性并投入开发。
- Vue.js 是一个更加灵活开放的解决方案。它允许你以希望的方式组织应用程序，而不是任何时候都必须遵循 Angular 制定的规则。它仅仅是一个视图层，所以你可以将它嵌入一个现有页面而不一定要做成一个庞大的单页应用。在配合其他库方面它给了你更大的的空间，但相应，你也需要做更多的架构决策。例如，Vue.js 核心默认不包含路由和 Ajax 功能，并且通常假定你在应用中使用了一个模块构建系统。这可能是最重要的区别。
- Angular 使用双向绑定，Vue 也支持双向绑定，不过默认为单向绑定，数据从父组件单向传给子组件。在大型应用中使用单向绑定让数据流易于理解。
- 在 Vue.js 中指令和组件分得更清晰。指令只封装 DOM 操作，而组件代表一个自给自足的独立单元 —— 有自己的视图和数据逻辑。在 Angular 中两者有不少相混的地方。
- Vue.js 有更好的性能，并且非常非常容易优化，因为它不使用脏检查。Angular，当 watcher 越来越多时会变得越来越慢，因为作用域内的每一次变化，所有 watcher 都要重新计算。并且，如果一些 watcher 触发另一个更新，脏检查循环（digest cycle）可能要

运行多次。 Angular 用户常常要使用深奥的技术，以解决脏检查循环的问题。有时没有简单的办法来优化有大量 watcher 的作用域。Vue.js 则根本没有这个问题，因为它使用基于依赖追踪的观察系统并且异步列队更新，所有的数据变化都是独立地触发，除非它们之间有明确的依赖关系。唯一需要做的优化是在 `v-for` 上使用 `track-by` 。

有意思的是，Angular 2 和 Vue 用相似的设计解决了一些 Angular 1 中存在的问题。

React

React.js 和 Vue.js 确实有一些相似 —— 它们都提供数据驱动、可组合搭建的视图组件。当然它们也有许多不同。

首先，内部实现本质上不同。React 的渲染建立在 Virtual DOM 上——一种在内存中描述 DOM 树状态的数据结构。当状态发生变化时，React 重新渲染 Virtual DOM，比较计算之后给真实 DOM 打补丁。

Virtual DOM 提供了一个函数式的方法描述视图，这真的很棒。因为它不使用数据观察机制，每次更新都会重新渲染整个应用，因此从定义上保证了视图与数据的同步。它也开辟了 JavaScript 同构应用的可能性。

Vue.js 不使用 Virtual DOM 而是使用真实 DOM 作为模板，数据绑定到真实节点。Vue.js 的应用环境必须提供 DOM。但是，相对于常见的误解——Virtual DOM 让 React 比其它的都快，Vue.js 实际上性能比 React 好，而且几乎不用手工优化。而 React，为了最优化的渲染需要处处实现 `shouldComponentUpdate` 和使用不可变数据结构。

在 API 方面，React（或 JSX）的一个问题是，渲染函数常常包含大量的逻辑，最终看着更像是程序片断（实际上就是）而不是界面的视觉呈现。对于部分开发者来说，他们可能觉得这是个优点，但对那些像我一样兼顾设计和开发的人来说，模板能让我们更好地在视觉上思考设计和 CSS。JSX 和 JavaScript 逻辑的混合干扰了我将代码映射到设计的思维过程。相反，Vue.js 通过在模板中加入一个轻量级的 DSL (指令系统)，换来一个依旧直观的模板，且能将逻辑封装进指令和过滤器中。

React 的另一个问题是：由于 DOM 更新完全交给 Virtual DOM 管理，当想要自己控制 DOM 时就有点棘手了（虽然理论上可以做到，但是这样做就本质上违背了 React 的设计思想）。如果应用需要特别的自定义 DOM 操作，特别是复杂时间控制的动画，这个限制就很讨厌。在这方面，Vue.js 更灵活，有许多用 Vue.js 制作的 **FWA/Awwwards 获奖站点**。

再多说几句：

- React 团队雄心勃勃，计划让 React 成为通用平台的 UI 开发工具，而 Vue 专注于为 Web 提供实用的解决方案。
- React，由于它的函数式特质，可以很好地使用函数式编程模式。但是对于初级开发者和初学者这也导致较大的学习难度。Vue 更易学习并能快速投入开发。
- 对于大型应用，React 社区已经创造了大量的状态管理方案，例如 Flux/Redux。Vue 本身不解决这个问题（React 内核也是），但是可以轻松地修改状态管理模式，实现一个类似的架构。Vue 有自己的状态管理方案 **Vuex**，而且 Vue 也可以与 **Redux** 一起用。
- React 的开发趋势是将所有东西都放在 JavaScript 中，包括 CSS。已经有许多 CSS-in-JS 方案，但是所有的方案多多少少都有它的问题。而且更重要的是，这么做脱离了标准的 CSS 开发经验，并且很难和 CSS 社区的已有工作配合。Vue 的 **单文件组件** 在把 CSS 封装到组件模块的同时仍然允许你使用你喜欢的预处理器。

Ember

Ember 是一个全能框架。它提供大量的约定，一旦你熟悉了它们，开发会很高效。不过，这也意味着学习曲线较高，而且不灵活。在框架和库（加上一系列松散耦合的工具）之间权衡选择。后者更自由，但是也要求你做更多的架构决定。

也就是说，最好比较 Vue.js 内核和 Ember 的模板与数据模型层：

- Vue 在普通 JavaScript 对象上建立响应，提供自动化的计算属性。在 Ember 中需要将所有东西放在 Ember 对象内，并且手工为计算属性声明依赖。
- Vue 的模板语法可以用全功能的 JavaScript 表达式，而 Handlebars 的语法和帮助函数语法相比之下非常受限。
- 在性能上，Vue 甩开 Ember 几条街，即使是 Ember 2.0 最新的 Glimmer 引擎。Vue 自动批量更新，在性能比较关键时 Ember 要手工管理循环。

Polymer

Polymer 是另一个由 Google 支持的项目，实际上也是 Vue.js 的灵感来源之一。Vue.js 的组件可以类比为 Polymer 中的自定义元素，它们提供类似的开发体验。最大的不同在于，Polymer 依赖最新的 Web 组件特性，在不支持的浏览器中，需要加载笨重的 polyfill，性能也会受到影响。相对的，Vue.js 无需任何依赖，最低兼容到IE9。

另外，在 Polymer 1.0 中，为了性能开发团队严格限制了它的数据绑定系统。例如，Polymer 模板支持的表达式仅有逻辑逆运算和简单的方法调用。它的计算属性实现得也不是很灵活。

最后，当发布到生产环境时，Polymer 元素需要用专用工具 vulcanizer 打包。相比之下，单文件 Vue 组件能与 Webpack 无缝整合，因而你可以轻松在组件中使用 ES6 及任意 CSS 预处理器。

Riot

Riot 2.0 提供类似的基于组件的开发模式（Riot 称之为“标签”），API 小而美。我认为 Riot 与 Vue 在设计思路上有许多相同点。不过，尽管比 Riot 重一点，Vue 提供了一些显著优处：

- 真实的条件渲染，Riot 渲染所有的分支，然后简单地显示/隐藏它们。

- 一个强大得多的路由器，Riot 的路由 API 过于简陋。
- 更成熟的工具链支持，见 Webpack + vue-loader。
- 过渡效果系统，Riot 没有。
- 更佳的性能。Riot 实际上使用脏检查而不是 Virtual DOM，因而遭受跟 Angular 一样的性能问题。

← **构建大型应用**

加入 Vue 社区！ →

发现错误？想参与编辑？ **在 Github 上编辑此页！**