

为什么要学习 Vue——前端框架角度

什么是框架

框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法；另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义。

可以说，一个框架是一个可复用的设计构件，它规定了应用的体系结构，阐明了整个设计、协作构件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间协作的方法，它为构件复用提供了上下文（Context）关系。因此构件库的大规模重用也需要框架。

构件领域框架方法在很大程度上借鉴了硬件技术发展的成就，它是构件技术、软件体系结构研究和应用软件开发三者发展结合的产物。在很多情况下，框架通常以构件库的形式出现，但构件库只是框架的一个重要部分。框架的关键还在于框架内对象间的交互模式和控制流模式。

框架比构件可定制性强。在某种程度上，将构件和框架看成两个不同但彼此协作的技术或许更好。框架为构件提供重用的环境，为构件处理错误、交换数据及激活操作提供了标准的方法。

应用框架的概念也很简单。它并不是包含构件应用程序的小片程序，而是实现了某应用领域通用完备功能（除去特殊应用的部分）的底层服务。使用这种框架的编程人员可以在一个通用功能已经实现的基础上开始具体的系统开发。框架提供了所有应用期望的默认行为的类集合。具体的应用通过重写子类（该子类属于框架的默认行为）或组装对象来支持应用专用的行为。

应用框架强调的是软件的设计重用性和系统的可扩充性，以缩短大型应用软件系统的开发周期，提高开发质量。与传统的基于类库的面向对象重用技术比较，应用框架更侧重于面向专业领域的软件重用。应用框架具有领域相关性，构件根据框架进行复合而生成可运行的系统。框架的粒度越大，其中包含的领域知识就更加完整。

框架，即 **framework**。其实就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。框架的概念最早起源于 Smalltalk 环境，其中最著名的框架是 Smalltalk 80 的用户界面框架

MVC(Model-View-Controller)。随着用户界面框架 Interviews 【Linton 89】和 ET++ 【Weinand 89】的开发和发布, 框架研究越来越受到研究人员的重视。虽然框架研究最初起源于用户界面领域, 但它还被成功地应用到其他领域中, 如操作系统、火警系统等。Taligent 公司于 1992 年成立后, 框架研究受到了广泛的重视。该公司计划基于框架来开发一个完整的面向对象操作系统。另外, 该公司还发布了一套支持快速应用开发的工具集 CommonPoint, 其中包括了上百个面向对象框架 【Andert 94,Cotter 95】。框架还没有统一的定义, 其中 Ralph Johnson 所给出的定义基本上为大多数研究人员所接受:

一个框架是一个可复用设计, 它是由一组抽象类及其实例间协作关系来表达的。

这个定义是从框架内涵的角度来定义框架的, 当然也可以从框架用途的角度来给出框架的定义:

一个框架是在一个给定的问题领域内, 一个应用程序的一部分设计与实现。从以上两个定义可以看出, 框架是对特定应用领域中的应用系统的部分设计和实现的整体结构。框架将应用系统划分为类和对象, 定义类和对象的责任, 类和对象如何互相协作, 以及对象之间的控制线程。这些共有的设计因素由框架预先定义, 应用开发人员只须关注于特定的应用系统特有部分。框架刻画了其应用领域所共有的设计决策, 所以说框架着重于设计复用, 尽管框架中可能包含用某种程序设计语言实现的具体类。

一个基于框架开发的应用系统包含一个或多个框架, 与框架相关的构件类, 以及与应用系统相关的功能扩展。与应用系统相关的扩展包括与应用系统相关的类和对象。应用系统可能仅仅复用了面向对象框架的一部分, 或者说, 它可能需要对框架进行一些适应性修改, 以满足系统需求。

面向对象的框架作为一种可复用的软件, 在基于框架的软件开发过程中会涉及到框架的开发和利用两个方面的工作。框架的开发阶段在于产生领域中可复用的设计。该阶段的主要结果是框架以及与框架相关的构件类。该阶段的一个重要活动是框架的演变和维护。像所有软件一样, 框架也易于变化。产生变化的原因很多, 如应用出错, 业务领域变化, 等等。

不论是哪一种技术, 最终都是为业务发展而服务的。从业务的角度来讲。首先, 框架的是为了企业的业务发展和战略规划而服务的, 他服从于企业的愿景; 其次, 框架最重要的目标是提高企业的竞争能力, 包括降低成本、提高质量、改善客户满意程度, 控制进度等方面。最后, 框架实现这一目标的方式是进行有效的知识积累。软件开发是一种知识活动, 因此知识的聚集和积累是至关重要的。框架能够采用一种结构化的方式对某个特定的业务领域进行描述, 也就是将这个领域相关的技术以代码、文档、模型等方式固化下来。

以上是框架的广义概念。接下来说前端框架。

前端框架也是框架，是框架更具体的分类，是提供一套解决方案，你得按我的规定来安排代码结构，它是随着前端功能的增强而产生的，对于往应用方向发展（也就是越来越像客户端）的 **web** 产品就很必要做前端架构这件事，它开始以模型为中心，**DOM** 操作只是附加，通过关注点分离鼓励改进应用程序。

未来的发展趋势是前后端只靠 **json** 数据进行通信，后端只处理和发送一段 **json** 到前端，计算和模板渲染都在前端进行，后台程序不再做模板的任何处理。使用 **MV*** 框架能有效实现前后端的解耦，简化开发流程，便于维护管理，可以把精力更多放到业务逻辑，提升开发效率。

所以考虑是否需要引入前端框架，可以根据产品类型做个基本判断：对于页面型产品，处理交互更多，**jquery** 也够用；但如果是应用软件类产品，需要关注处理复杂模型，很有必要引入 **MV*** 框架。如今的互联网公司的产品基本都是 **web app**，越来越像传统应用软件开发靠拢，使用个框架就还是很有必要的。

前端框架的特点

1. 声明式 & 数据驱动渲染

更深一步思考，**React** 提供的 **JSX** 和 **Vue** 提供的模板，它们的目的是什么？目的是为了实现在声明式渲染的功能。不论是 **JSX**，或者是 **Vue** 中的模板，本质上都是描述了『状态』与『视图』之间的映射关系。

所以声明式渲染是框架的特性。

声明了映射关系之后，可以得到一个公式：

UI = render(state)

状态与视图之间的映射关系，等同于 **render** 函数。熟悉 **React** 的同学对这个公式应该并不陌生。**JSX** 与 **Vue** 的模板在这一点上是相同的。在框架的内部，不论是 **JSX** 还是 **Vue** 的模板，最终会编译成 **render** 函数。

上面这个公式，输入的是 **state**，输出的是 **DOM**。所以输入变了输出就变了。

这个特性就是我们常说的数据驱动视图。

这里会引出一个问题，框架必须要知道 **Web** 应用在运行时”状态“是否发

生了变化，然后才能使用前面提到的公式重新输出一个新的 UI。所以如何知道 Web 应用的状态在运行时是否发生了变化这个问题是所有框架必须去解决的。

解决方案有很多种。不同框架，或者同一个框架的不同版本对这个问题的解决方案都不同，但相同的是都可以解决问题。关于这个问题如何解决，我在曾在我的文章、分享的 PPT 以及目前还未上市的书中都有详细的介绍。这个问题不是本文所讨论的重点，感兴趣的同学可以点击[这里](#)了解更多信息。

不同的解决方案，导致的直接结果就是它所提供的给用户的上层语法或 API 完全不一样。

2. 组件化

现代主流框架都具备的一个特性是“组件”，它们都会以“组件”作为一个基本的抽象单元。

可能不同的框架，它所提供的操控组件的方式不一样，但概念上是相似的。之前听过一次尤雨溪的知乎 Live，他将实际应用中的组件分为四种类型并依次介绍了四种组件之间的区别：

展示型组件

展示型组件是最直接也是最常用的组件，就是用数据渲染视图，“数据进，DOM 出”。

接入型组件

接入型组件通常会跟接入数据的 service 层打交道。包含一些和服务层或数据源打交道的逻辑，然后接入型组件会将数据往下传，传给比较简单的展示型组件。在 React 中这种类型的组件被称为“容器组件（container component）”。

交互型组件

交互型组件典型的例子是对表单组件的封装和增强。大部分组件库，像 ElementUI 都是以交互型组件为主。这一类组件会有比较复杂的交互逻辑，但是它是一个非常通用的逻辑，所以它强调复用。

功能型组件

功能型组件是比较抽象的组件。用 Vue 举例，路由的<router-view>和 Vue 自带的<transition>都属于功能型组件。它本身不渲染任何内容，它是一个逻辑型的组件。它通常作为一个扩展或一种抽象机制存在。

不同框架操控组件的方式可能不一样，但使用组件的“心法”永远是一样的。这就是关注特性带来的好处，你可以切换到任意一个框架，使用组件或封装组件时，总是上面列出的几种类型。

掌握了“心法”的程序员在切换框架时，他的状态通常是这样的：我现在想写一个交互型组件，这个框架都提供了哪些 API？去翻翻文档看一下。然后就可以写出一个很优雅的组件出来，哪怕刚使用这个框架才不到一天。如果没有掌握“心法”，用了一个框架写出的代码很糟糕，那么换了一个框架也不会写出更好的代码，甚至更糟糕。。

3. 路由

在前端框架中都是前端路由，可以保证性能和用户体验的层面来比较的话，后端路由每次访问一个新页面的时候都要向服务器发送请求，然后服务器再响应请求，这个过程肯定会有延迟。而前端路由在访问一个新页面的时候仅仅是变换了一下路径而已，没有了网络延迟，对于用户体验来说会有相当大的提升。

在某些场合中，用 ajax 请求，可以让页面无刷新，页面变了但 Url 没有变化，用户就不能复制到想要的地址，用前端路由做单页面网页就很好的解决了这个问题

4. 良好周边以及社区

一个框架的成长必然离不开别人的辅助升级。不断吸收新的想法使得框架走的会越来越好。

5. 其他特点

前面详细介绍了几个特性给大家感受下为什么要重视特性。框架的特性太多，而且不同的框架都会有不同的特性，不能每一个都详细介绍，下面列出一些大家比较熟悉的通用特性：

状态和数据流管理













CLI 工具

同构/服务端渲染

CSS 管理方案

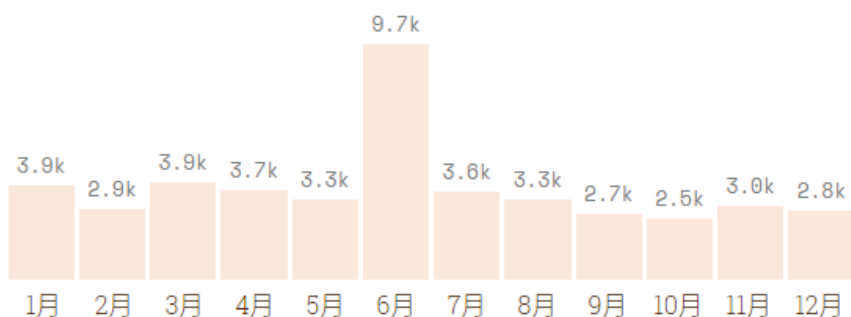
前端框架对比

框架的使用情况

1		Vue.js 👉 Vue.js is a progressive, incrementally-adoptable JavaScript ...	+45.3k ★
2		React A declarative, efficient, and flexible JavaScript library for buildi...	+34.2k ★
3		Angular One framework. Mobile & desktop.	+12.4k ★
4		Hyperapp 1 kB JavaScript micro-framework for building declarative web a...	+7.6k ★
5		Omi Next generation web framework using web components with om...	+5.1k ★
6		dva 👉 React and redux based, lightweight and elm-style framework...	+4.8k ★
7		Preact 👉 Fast 3kB React alternative with the same modern API. Comp...	+4.2k ★
8		Nerv A blazing fast React alternative, compatible with IE8 and React 16.	+3.6k ★
9		Svelte The magical disappearing UI framework	+3.0k ★
10		Stencil A Web Component compiler for building fast, reusable UI compo...	+2.8k ★
11		Mithril A Javascript Framework for Building Brilliant Applications	+2.1k ★
12		Ember Ember.js - A JavaScript framework for creating ambitious web a...	+1.8k ★

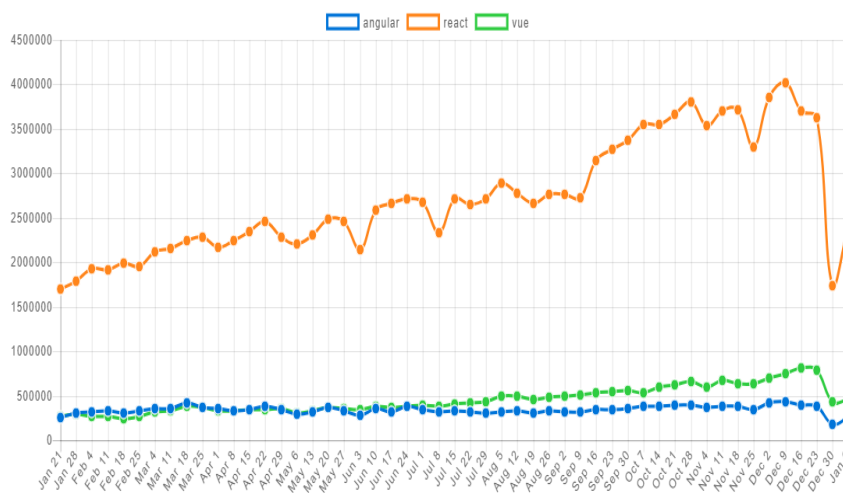
这个是 2018 年 github 中各大前端框架的 Star 数量，Vue 是最受欢迎的明星项目。

在 2018 年, Vue.js 增加了 45.3k 个 star, 在 GitHub 前端框架 分类中排名第 1。 - Tweet



下面看一下三大框架的下载量

Downloads in past 1 Year



Vue 在过去的一年中稳超 Angular，称为下载量第二的框架。这个是 npm 包下载统计。以上的数据是全球互联网公司的数据（毕竟我们的 Vue 有一部分，路转粉到 cnpm 那里去了）

GitHub Stats

	stars 🌟	forks 🍴	issues 🚩	updated ⚙	created 🗓
angular.js	59330	28960	481	Jan 12, 2019	Jan 6, 2010
react	119475	21637	540	Jan 11, 2019	May 25, 2013
vue	124655	17818	208	Jan 11, 2019	Jul 29, 2013

前端框架三巨头 React、Angular 和 Vue，虽然都很受欢迎，且保持着上

升趋势，但 Vue 爆发力最强，但在使用率上，仍低于 React。
在中国大厂中 Vue 的使用普及率是高于 React。

三大框架差别

对比 React

React 和 Vue 有许多相似之处，它们都有：

使用 Virtual DOM

提供了响应式 (Reactive) 和组件化 (Composable) 的视图组件。
将注意力集中保持在核心库，而将其他功能如路由和全局状态管理交给相关的库。

由于有着众多的相似处，我们会用更多的时间在这一块进行比较。这里我们不只保证技术内容的准确性，同时也兼顾了平衡的考量。我们需要承认 React 比 Vue 更好的地方，比如更丰富的生态系统。

React 社区为我们准确进行平衡的考量提供了非常积极的帮助，特别感谢来自 React 团队的 Dan Abramov。他非常慷慨的花费时间来贡献专业知识来帮助我们完善这篇文档。

运行时性能

React 和 Vue 都是非常快的，所以速度并不是在它们之中做选择的决定性因素。对于具体的数据表现，可以移步这个第三方 benchmark，它专注于渲染/更新非常简单的组件树的真实性能。

优化

在 React 应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。

如要避免不必要的子组件的重渲染，你需要在所有可能的地方使用 PureComponent，或是手动实现 shouldComponentUpdate 方法。同时你可能会需要使用不可变的数据结构来使得你的组件更容易被优化。

然而，使用 PureComponent 和 shouldComponentUpdate 时，需要保证该组件的整个子树的渲染输出都是由该组件的 props 所决定的。如果不符合这个情况，那么此类优化就会导致难以察觉的渲染结果不一致。这使得

React 中的组件优化伴随着相当的心智负担。

在 Vue 应用中，组件的依赖是在渲染过程中自动追踪的，所以系统能精确知晓哪个组件确实需要被重渲染。你可以理解为每一个组件都已经自动获得了 `shouldComponentUpdate`，并且没有上述的子树问题限制。

Vue 的这个特点使得开发者不再需要考虑此类优化，从而能够更好地专注于应用本身。

HTML & CSS

在 React 中，一切都是 JavaScript。不仅仅是 HTML 可以用 JSX 来表达，现在的潮流也越来越多地将 CSS 也纳入到 JavaScript 中来处理。这类方案有其优点，但也存在一些不是每个开发者都能接受的取舍。

Vue 的整体思想是拥抱经典的 Web 技术，并在其上进行扩展。我们下面会详细分析一下。

JSX vs Templates

在 React 中，所有的组件的渲染功能都依靠 JSX。JSX 是使用 XML 语法编写 JavaScript 的一种语法糖。

JSX 说是手写的渲染函数有下面这些优势：

你可以使用完整的编程语言 JavaScript 功能来构建你的视图页面。比如你可以使用临时变量、JS 自带的流程控制、以及直接引用当前 JS 作用域中的值等等。

开发工具对 JSX 的支持相比于现有可用的其他 Vue 模板还是比较先进的（比如，linting、类型检查、编辑器的自动完成）。

事实上 Vue 也提供了渲染函数，甚至支持 JSX。然而，我们默认推荐的还是模板。任何合乎规范的 HTML 都是合法的 Vue 模板，这也带来了一些特有的优势：

对于很多习惯了 HTML 的开发者来说，模板比起 JSX 读写起来更自然。这里当然有主观偏好的成分，但如果这种区别会导致开发效率的提升，那么它就有客观的价值存在。

基于 HTML 的模板使得将已有的应用逐步迁移到 Vue 更为容易。

这也使得设计师和新人开发者更容易理解和参与到项目中。

你甚至可以使用其他模板预处理器，比如 Pug 来书写 Vue 的模板。

有些开发者认为模板意味着需要学习额外的 DSL (Domain-Specific Language 领域特定语言) 才能进行开发——我们认为这种区别是比较肤浅的。首先，JSX 并不是免费的——它是基于 JS 之上的一套额外语法，因此也有它自己的学习成本。同时，正如同熟悉 JS 的人学习 JSX 会很容易一样，熟悉 HTML 的人学习 Vue 的模板语法也是很容易的。最后，DSL 的存在使得我们可以让开发者用更少的代码做更多的事，比如 v-on 的各种修饰符，在 JSX 中实现对应的功能会需要多得多的代码。

更抽象一点来看，我们可以把组件区分为两类：一类是偏视图表现的 (presentational)，一类则是偏逻辑的 (logical)。我们推荐在前者中使用模板，在后者中使用 JSX 或渲染函数。这两类组件的比例会根据应用类型的不同有所变化，但整体来说我们发现表现类的组件远远多于逻辑类组件。

组件作用域内的 CSS

除非你把组件分布在多个文件上 (例如 CSS Modules)，CSS 作用域在 React 中是通过 CSS-in-JS 的方案实现的 (比如 styled-components、glamorous 和 emotion)。这引入了一个新的面向组件的样式范例，它和普通的 CSS 撰写过程是有区别的。另外，虽然在构建时将 CSS 提取到一个单独的样式表是支持的，但 bundle 里通常还是需要有一个运行时程序来让这些样式生效。当你能够利用 JavaScript 灵活处理样式的同时，也需要权衡 bundle 的尺寸和运行时的开销。

如果你是一个 CSS-in-JS 的爱好者，许多主流的 CSS-in-JS 库也都支持 Vue (比如 styled-components-vue 和 vue-emotion)。这里 React 和 Vue 主要的区别是，Vue 设置样式的默认方法是单文件组件里类似 style 的标签。

单文件组件让你可以在同一个文件里完全控制 CSS，将其作为组件代码的一部分。

```
<style scoped>
  @media (min-width: 250px) {
    .list-container:hover {
      background: orange;
    }
  }
</style>
```

这个可选 `scoped` 属性会自动添加一个唯一的属性 (比如 `data-v-21e5b78`) 为组件内 CSS 指定作用域, 编译的时候 `.list-container:hover` 会被编译成类似 `.list-container[data-v-21e5b78]:hover`。

最后, `Vue` 的单文件组件里的样式设置是非常灵活的。通过 `vue-loader`, 你可以使用任意预处理器、后处理器, 甚至深度集成 `CSS Modules`——全部都在 `<style>` 标签内。

规模

向上扩展

`Vue` 和 `React` 都提供了强大的路由来应对大型应用。`React` 社区在状态管理方面非常有创新精神 (比如 `Flux`、`Redux`), 而这些状态管理模式甚至 `Redux` 本身也可以非常容易的集成在 `Vue` 应用中。实际上, `Vue` 更进一步地采用了这种模式 (`Vuex`), 更加深入集成 `Vue` 的状态管理解决方案 `Vuex` 相信能为你带来更好的开发体验。

两者另一个重要差异是, `Vue` 的路由库和状态管理库都是由官方维护支持且与核心库同步更新的。`React` 则是选择把这些问题交给社区维护, 因此创建了一个更分散的生态系统。但相对的, `React` 的生态系统相比 `Vue` 更加繁荣。

最后, `Vue` 提供了 `Vue-cli` 脚手架, 能让你非常容易地构建项目, 包含了 `Webpack`, `Browserify`, 甚至 `no build system`。`React` 在这方面也提供了 `create-react-app`, 但是现在还存在一些局限性:

它不允许在项目生成时进行任何配置, 而 `Vue` 支持 `Yeoman-like` 定制。它只提供一个构建单页面应用的单一模板, 而 `Vue` 提供了各种用途的模板。

它不能用用户自建的模板构建项目, 而自建模板对企业环境下预先建立协议是特别有用的。

而要注意的是这些限制是故意设计的，这有它的优势。例如，如果你的项目需求非常简单，你就不需要自定义生成过程。你能把它作为一个依赖来更新。如果阅读更多关于不同的设计理念。

向下扩展

React 学习曲线陡峭，在你开始学 **React** 前，你需要知道 **JSX** 和 **ES2015**，因为许多示例用的是这些语法。你需要学习构建系统，虽然你在技术上可以用 **Babel** 来实时编译代码，但是这并不推荐用于生产环境。

就像 **Vue** 向上扩展好比 **React** 一样，**Vue** 向下扩展后就类似于 **jQuery**。你只要把如下标签放到页面就可以运行：

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

然后你就可以编写 **Vue** 代码并应用到生产中，你只要用 **min** 版 **Vue** 文件替换掉就不用担心其他的性能问题。

由于起步阶段不需学 **JSX**，**ES2015** 以及构建系统，所以开发者只需不到一天的时间阅读指南就可以建立简单的应用程序。

原生渲染

React Native 能使你用相同的组件模型编写有本地渲染能力的 **APP** (**iOS** 和 **Android**)。能同时跨多平台开发，对开发者是非常棒的。相应地，**Vue** 和 **Weex** 会进行官方合作，**Weex** 是阿里巴巴发起的跨平台用户界面开发框架，同时也正在 **Apache** 基金会进行项目孵化，**Weex** 允许你使用 **Vue** 语法开发不仅仅可以运行在浏览器端，还能被用于开发 **iOS** 和 **Android** 上的原生应用的组件。

在现在，**Weex** 还在积极发展，成熟度也不能和 **React Native** 相抗衡。但是，**Weex** 的发展是由世界上最大的电子商务企业的需求在驱动，**Vue** 团队也会和 **Weex** 团队积极合作确保为开发者带来良好的开发体验。

Angular 事实上必须用 **TypeScript** 来开发，因为它的文档和学习资源几乎全部是面向 **TS** 的。**TS** 有很多好处——静态类型检查在大规模的应用中非常有用，同时对于 **Java** 和 **C#** 背景的开发者的也是非常提升开发效率的。

然而，并不是所有人都想用 **TS**——在中小型规模的项目中，引入 **TS** 可能

并不会带来太多明显的优势。在这些情况下，用 **Vue** 会是更好的选择，因为在不用 **TS** 的情况下使用 **Angular** 会很有挑战性。

最后，虽然 **Vue** 和 **TS** 的整合可能不如 **Angular** 那么深入，我们也提供了官方的 类型声明 和 组件装饰器，并且知道有大量用户在生产环境中使用 **Vue + TS** 的组合。我们也和微软的 **TS / VSCode** 团队进行着积极的合作，目标是 为 **Vue + TS** 用户提供更好的类型检查和 **IDE** 开发体验。

运行时性能

这两个框架都很快，有非常类似的 **benchmark** 数据。你可以浏览具体的数据做更细粒度的对比，不过速度应该不是决定性的因素。

体积

在体积方面，最近的 **Angular** 版本中在使用了 **AOT** 和 **tree-shaking** 技术后使得最终的代码体积减小了许多。但即使如此，一个包含了 **Vuex + Vue Router** 的 **Vue** 项目 (**gzip** 之后 **30kB**) 相比使用了这些优化的 **angular-cli** 生成的默认项目尺寸 (~**130kB**) 还是要小得多。

灵活性

Vue 相比于 **Angular** 更加灵活，**Vue** 官方提供了构建工具来协助你构建项目，但它并不限制你去如何组织你的应用代码。有人可能喜欢有严格的代码组织规范，但也有开发者喜欢更灵活自由的方式。

学习曲线

要学习 **Vue**，你只需要有良好的 **HTML** 和 **JavaScript** 基础。有了这些基本的技能，你就可以非常快速地通过阅读 指南 投入开发。

Angular 的学习曲线是非常陡峭的——作为一个框架，它的 **API** 面积比起 **Vue** 要大得多，你也因此需要理解更多的概念才能开始有效率地工作。当然，**Angular** 本身的复杂度是因为它的设计目标就是只针对大型的复杂应用；但不可否认的是，这也使得它对于经验不甚丰富的开发者相当的不友好。