Vue.js作者尤雨溪：Vue 2.0——渐进式前端解决方案

原创 2016-10-26 尤雨溪 InfoQ

> 框架是什么？为什么要有框架？在众多的框架之中，Vue独具魅力之处在哪里呢？其背后的核心思想是什么？Vue究竟火到什么程度？最近发布的Vue2.0又做了哪些改进呢？Vue和Weex又是怎样的一种合作？

本文根据尤雨溪在2016 QCon 全球软件开发大会（上海）上的讲演整理而成。

Tips：今天文章头图来自于两位迷妹的坚持，有颜有才，这样的程序员请再来一打！

作者简介

尤雨溪，Vue Technology LLC 创始人， Vue.js作者，设计师，开发者，开源爱好者，前端框架 Vue.js 的作者。

曾就职于 Google Creative Lab，参与多个实验项目的界面原型研发，后加入 Meteor，参与 Meteor 框架本身的维护和 Meteor Galaxy 平台的交互设计与前端开发。现全职投入 Vue.js 的开发与维护，立志将 Vue.js 打造成与 Angular/React 平起平坐的世界顶级框架。

为什么要有框架

1、框架的存在是为了帮助我们应对复杂度

前端框架特别多，那么为什么要有框架呢？我个人的看法是，框架的存在是为了帮助我们应对复杂度。当我们需要解决一些前端上工程问题的时候，这些问题会有不同的复杂度。

如果你用太简陋的工具应对非常复杂的需求，就会极大地影响你的生产力。所以，框架本身是帮我们把一些重复的并且已经受过验证的模式，抽象到一个已经帮你设计好的API封装当中，帮助我们去应对这些复杂的问题。

2、框架自身也有复杂度

但是，框架本身也会带来复杂度。相信大家在调研各种框架或学习各种框架时，会遇到学习曲线问题——有些框架会让人一时不知如何上手。这里就抽象出一个问题，就是要做的应用的复杂度与所使用的框架的复杂度的对比。 进一步说，是所要解决的问题的内在复杂度，与所使用的工具的复杂度进行对比。

3、工具复杂度是为了处理内在复杂度所做的投资

工具的复杂度是可以理解为是我们为了处理问题内在复杂度所做的投资。为什么叫投资？那是因为如果投的太少，就起不到规模的效应，不会有合理的回报。这就像创业公司拿风投，投多少是很重要的问题。如果要解决的问题本身是非常复杂的，那么你用一个过于简陋的工具应付它，就会遇到工具太弱而使得生产力受影响的问题。

反之，是如果所要解决的问题并不复杂，但你却用了很复杂的框架，那么就相当于杀鸡用牛刀，会遇到工具复杂度所带来的副作用，不仅会失去工具本身所带来优势，还会增加各种问题，例如培训成本、上手成本，以及实际开发效率等。

4、Pick the right tool for the job

“Pick the right tool for the job”——在国外，跟开发者讨论一些框架选型问题时，大家都会说这句话——一切都要看场景。因为，前端开发原生开发或者桌面开发模式相比，有自己的独特之处，它跟其实并不那么固定。在Web上面，应用可以有非常多的形态，不同形态的Web应用可能有完全不同程度的复杂度。这也是为什么我要谈工具复杂度和所要做的应用复杂度的问题。

5、怎么看前端框架的复杂度

目前的前端开发已经越来越工程化，而我们需要解决的实际问题也是不同的。如下图所示，我们可能在任何情况下都需要声明式的渲染功能，并希望尽可能避免手动操作，或者说是可变的命令式操作，希望尽可能地让DOM的更新操作是自动的，状态变化的时候它就应该自动更新到正确的状态；

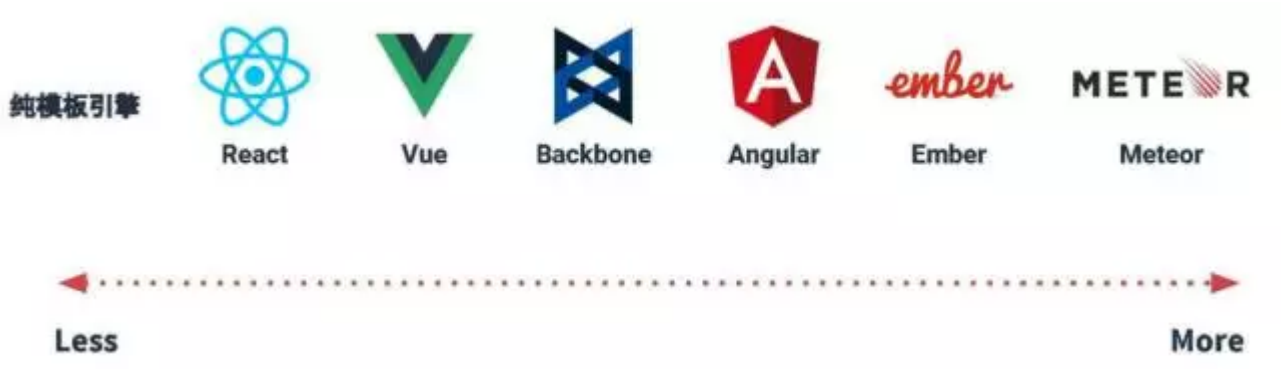
我们需要组件系统，将一个大型的界面切分成一个一个更小的可控单元；客户端路由——这是针对单页应用而言，不做就不需要，如果需要做单页应用，那么就需要有一个URL对应到一个应用的状态，就需要有路由解决方案；

大规模的状态管理——当应用简单的时候，可能一个很基础的状态和界面映射可以解决问题，但是当应用变得很大，涉及多人协作的时候，就会涉及多个组件之间的共享、多个组件需要去改动同一份状态，以及如何使得这样大规模应用依然能够高效运行，这就涉及大规模状态管理的问题，当然也涉及到可维护性，还有构建工具。

现在，如果放眼前端的未来，当HTTP2普及后，可能会带来构建工具的一次革命。但就目前而言，尤其是在中国的网络环境下，打包和工程构建依然是非常重要且不可避免的一个环节。

主流框架分析

我们看一下现有的一些主流框架从少到多所解决的问题。这个多少并不是来评价框架的好坏，而是从设计的角度出发看它涵盖多少内容。



- 纯模板引擎：最少的就是纯模板引擎，只管状态到界面的映射。
- React和Vue：其实这两者都是非常专注的只做状态到界面映射，以及组件。
- Backbone：它会给你多一些架构上指导，比如它会让你分层。
- Angular：它做的事情就更多，它有自己的路由，这些都会包含在里面。
- Ember：相比Angular，Ember做得就更加彻底，Ember信奉的是约定优于配置，它会将一切都帮你设计好打包好，你就开箱用就可以了。
- Meteor：Meteor只是一个极端，它是从前到后全都包含，从前端到数据层到数据库，全都帮你打包好。

通过简单的分析，我们可以感受到，做得少的框架不一定就不如做得多的框架，这体现出一种取舍。也就是说，做得少的框架可以给你更多的灵活性，但你需要做更多的选择；做得多的框架有更强的侵入性，学习成本更高，灵活性更低。一旦选择了一个侵入性强的框架，那么一些小的部分你就没有机会去切换成其他你更想用的方案。

所以，React和Vue有一个共同特点，它们都有各自的配套工具，核心虽然只解决一个很小的问题，但它们有生态圈及配套的可选工具，当你把他们一个一个加进来的时候，就可以组合成非常强大的栈，就可以涵盖其他的这些更完整的框架所涵盖的问题。

这样的一个配置方案，使得在你构建技术栈的时候有可弹性伸缩的工具复杂度：当所要解决的问题内在复杂度很低的时候，可以只用核心的这些很简单的功能；当需要做一个更复杂的应用时，再增添相应的工具。

例如做一个单页应用的时候才需要用路由；做一个相当庞大的应用，涉及到多组件状态共享以及多个开发者共同协作时，才可能需要大规模状态管理方案。

一个纯粹的复杂的单页应用，和只是在后端渲染的静态页面上嵌入交互内容所需要选择的工程栈其实是有相当大区别的。这就是为什么我觉得，核心+生态的栈会是一个在整体选型更为灵活的栈。

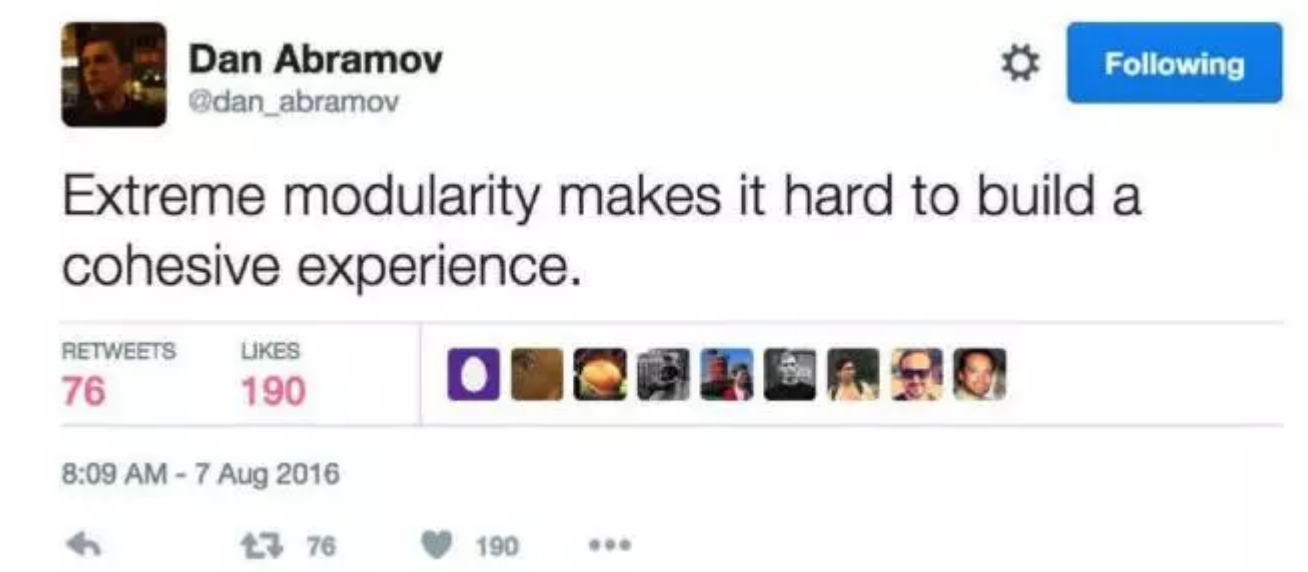
React和Vue都选择这个模式。Facebook团队只是专注做React本身，但React社区非常活跃，贡献了大量的第三方解决方案。不可否认，React社区是当前最活跃的社区，很多优秀的想法和思路，包括状态管理方案最早都是从React社区萌发出来。但是社区的这种活跃也带来一定程度的副作用，那就是时代变化太快，三天出一个新版本，同一个问题曾经存在几十种不同的解决方案。

这就使得我们在去搭建自己的栈时，需要花很多的时间去鉴别所选择的部件。同时，由于整个生态圈的这些库并不是由一个统一的团队去规划和设计的，所以很难考虑到不同的库之间的协作，这就会导致磨合问题。

同时，这也使得很多开发者抱怨有一个“JavaScript Fatigue”，说JS生态圈东西太多了，为了跟上潮流，需要不停学习最新的东西，觉得很累。当然，有很多人觉得这是生态圈繁荣的表现，但它确实使得大家面临了选择困难症的问题。

Dan Abramov 是之前React社区非常活跃的开发者，已经加入了React团队。有一天他在推特上说，极度的模块化使得他非常难去构建一个统一的体验。这句话指的就是，React生态圈整个栈的每一个部分来自不同开发者，想全部整合到一起的时就有很多零碎的问题。

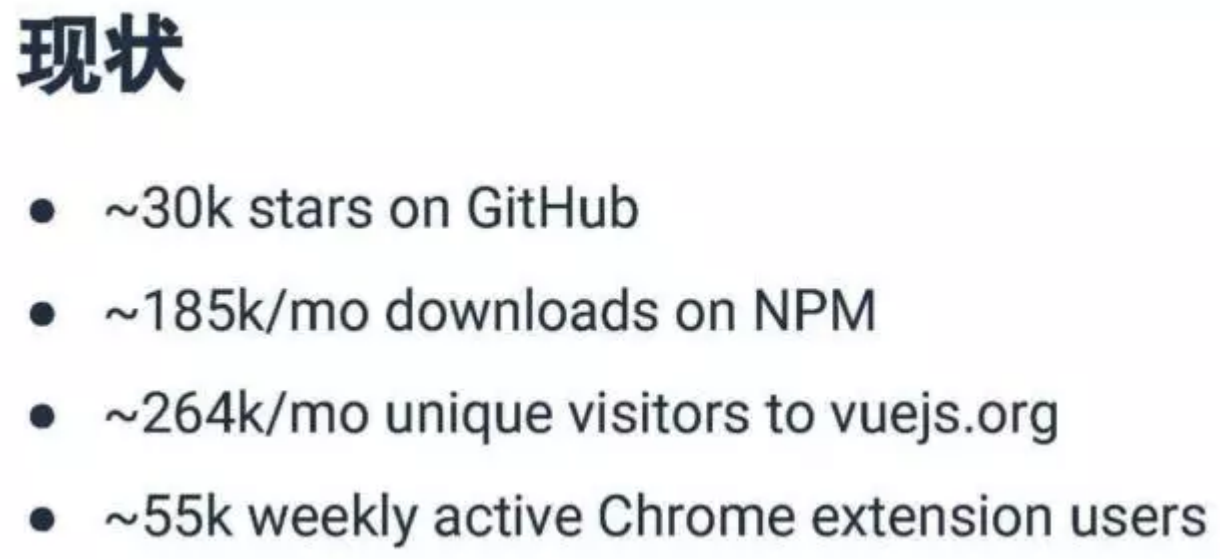
这是他刚开始做React现在的官方的CLI的时候发出的感慨，他在试图整合社区的各种东西放到一个架子里面，于是遇到了很多这样的问题。我这里完全没有否认React生态圈繁荣的意思，我只是觉得可以有另外一种选择，那就是我们可以做一个渐进式框架，这就是Vue选择的方向。



渐进式框架Vue.js

1、Vue.js现状

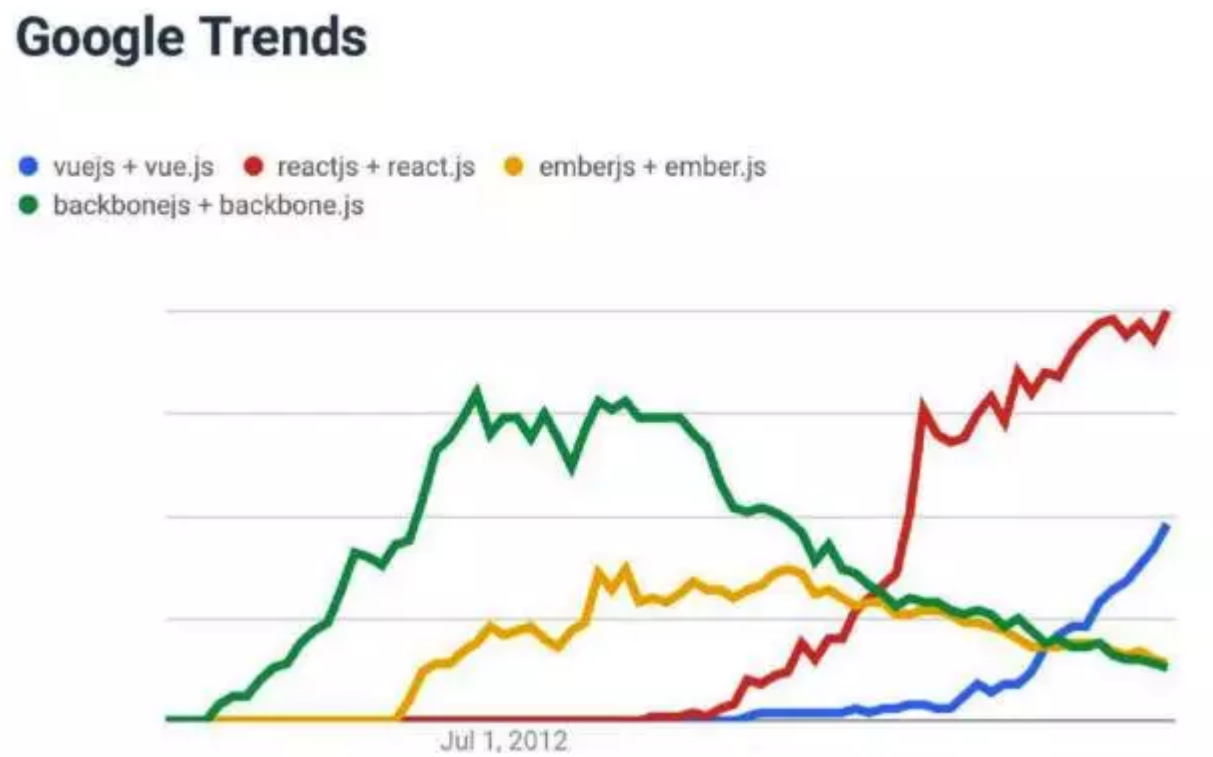
以下数据可以体现出Vue.js的现状。



- 前一段时间突破了三万星，总下载量过百万。
- 官网上每个月的用户量为26万，这个应该是不包含中国区数据。官方开发者插件的周活跃用户数在5万5左右。这个数据是我觉得最有说服力的数据。安装并且使用开发者插件的Vue用户，应该会在实际生产中真正频繁使用Vue。

Google搜索趋势的相关数据如下图所示。图中，绿色的是Backbone的数据，黄色是Ember，红色是React，蓝色是Vue。可以看出React和Vue近两年发展势头都比较迅猛。

可以看出，Vue的曲线开始的是很早，2013年已经开始，但是有很长一段时间的增长是比较低的。因为在那一段时间我还在谷歌工作，Vue基本上是作为个人项目在运营。在过去一两年中，Vue获得了非常大的突破性发展。这个图里没有Angular，因为Angular的量还是非常大的，如果放进去就破表了。



这些数据并不能绝对地代表框架当前的热度，但有一定的参考价值。可以看到React的势头很足。而由Vue的曲线还可以看出它的增长速度还在不停上扬。

2、Vue的定位

我在做Vue的过程中也在不停地思考它的定位，现在，我觉得它与其他框架的区别就是渐进式的想法，也就是“Progressive”——这个词在英文中定义是渐进，一步一步，不是说你必须一竿子把所有的东西都用上。

3、Vue的设计

接下来我们回到之前看的图：

Vue从设计角度来讲，虽然能够涵盖这张图上所有的东西，但是你并不需要一上手就把所有东西全用上，因为没有必要。无论从学习角度，还是实际情况，这都是可选的。声明式渲染和组建系统是Vue的核心库所包含内容，而客户端路由、状态管理、构建工具都有专门解决方案。这些解决方案相互独立，你可以在核心的基础上任意选用其他的部件，不一定要全部整合在一起。

4、Vue的实现

接下来深入讲一讲这些具体的概念以及Vue在这些概念上具体是做怎样的实现。

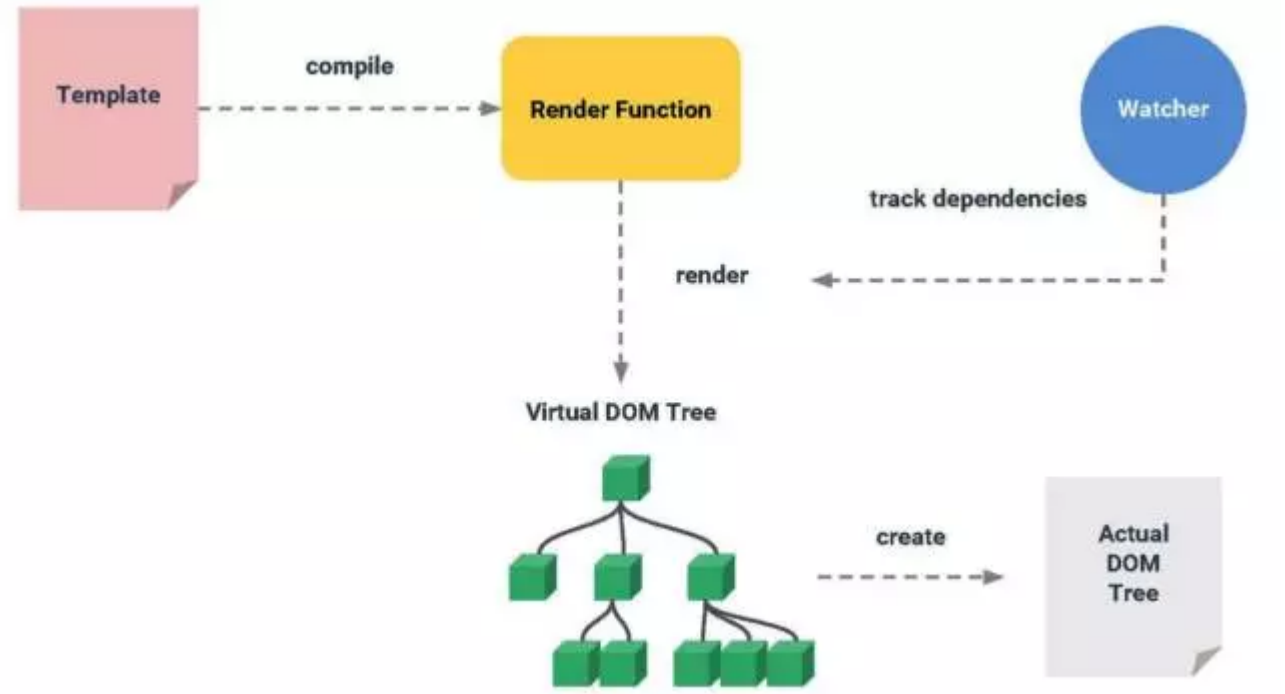
（1）声明式渲染

现在基本所有的框架都已经认同这个看法——DOM应尽可能是一个函数式到状态的映射。状态即是唯一的真相，而DOM状态只是数据状态的一个映射。所有的逻辑尽可能在状态的层面去进行，当状态改变的时候，View应该是在框架帮助下自动更新到合理的状态，而不是说当你观测到数据变化之后手动选择一个元素，再命令式地去改动它的属性。

下图是Vue的一个模板示例，如果没有用过Vue的话，可以大概感觉到这是一个怎样的概念。

```
<div id="app">
  <ul>
    <li
      v-for="todo in todos"
      v-on:click="todo.done = !todo.done">
      {{ todo.title }} ({{ todo.done ? 'done' : 'nah' }})
    </li>
  </ul>
</div>
```

其实，在模板语法上，Vue跟Angular是比较相似。在Vue1.0里面，模板实现跟Angular类似，如下图所示，把模板直接做成在浏览器里面parse成DOM树，然后去遍历这个树，提取其中的各种绑定。



在Vue2.0中，渲染层的实现做了根本性改动，那就是引入了虚拟DOM。

从架构来讲，Vue2.0 依然是写一样的模板（Vue2.0于前段时间发布，具体报道参见

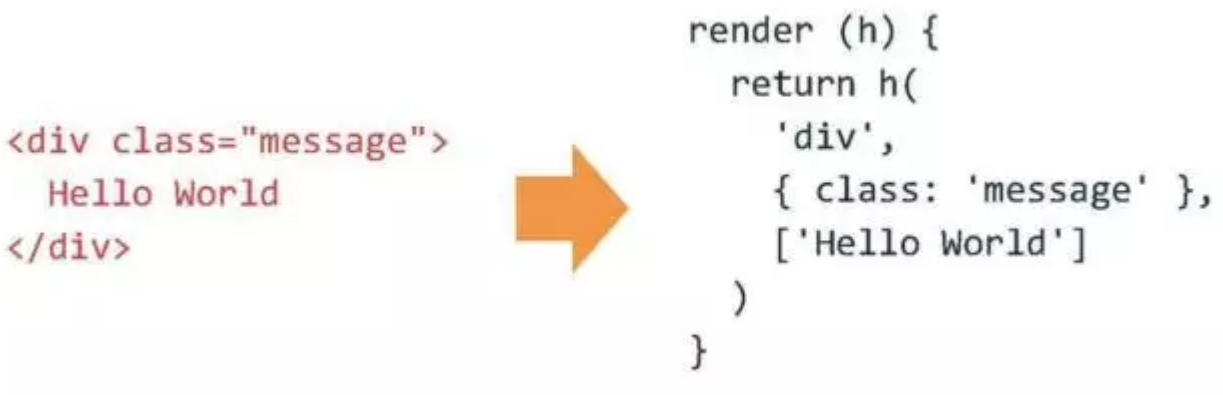
http://t.cn/RVC0foZ）。在最左边，Vue2.0跟1.0的模板语法绝大部分是兼容的。Vue的编译器在编译模板之后，会把这些模板编译成一个渲染函数。而函数被调用的时候就会渲染并且返回一个虚拟DOM的树。这个树非常轻量，它的职责就是描述当前界面所应处的状态。当我们有了这个虚拟的树之后，再交给一个patch函数，负责把这些虚拟DOM真正施加到真实的DOM上。

在这个过程中，Vue有自身的响应式系统来侦测在渲染过程中所依赖到的数据来源。在渲染过程中，侦测到的数据来源之后，之后就可以精确感知数据源的变动。到时候就可以根据需要重新进行渲染。当重新进行渲染之后，会生成一个新的树，将新树与旧树进行对比，就可以最终得出应施加到真实DOM上的改动。最后再通过patch函数施加改动。

这样做的主要原因是，在浏览器当中，JavaScript的运算在现代的引擎中非常快，但DOM本身是非常缓慢的东西。当你调用原生DOM API的时候，浏览器需要在JavaScript引擎的语境下去接触原生的DOM的实现，这个过程有相当的性能损耗。所以，本质的考量是，要把耗费时间的操作尽量放在纯粹的计算中去做，保证最后计算出来的需要实际接触真实DOM的操作是最少的。

下面看渲染函数。用过React的开发者可能知道，React是没有模板的，直接就是一个渲染函数，它中间返回的就是一个虚拟DOM树。JSX实际就是一套用于让我们更简单地去描述树状结构的语法糖。

如下图所示，在Vue2.0当中，可以看到就是说当比如左侧的模板，经过Vue的编译之后就会变成右侧的东西。



这个函数类似于创建一个虚拟元素的函数，我们可以给它一个名字，给它描述应该有的属性特性和可能其他的数据。然后后面这个最后这个参数是个数组，包含了该虚拟元素的子元素。总的来说2.0的编译器做的就是这个活。

同时，在Vue2.0里，用户可以选择直接跳过模板这一层去手写渲染函数，同时也有可选JSX支持。从开发者的偏好以及开发者的效益的角度来考量，模板和JSX是各有利弊的东西。模板更贴近我们的HTML，可以让我们更直观地思考语义结构，更好地结合CSS的书写。

JSX和直接渲染函数，因为是真正的JavaScript，拥有这个语言本身的所有的能力，可以进行复杂的逻辑判断，进行选择性的返回最终要返回的DOM结构，能够实现一些在模板的语法限制下，很难做到的一些事情。

所以在Vue2.0里，两个都是可以选择的。在绝大部分情况下使用模板，但是在需要复杂逻辑的情况下，使用渲染函数。在Vue2.0的路由和内部的一些实践上，都大量地应用渲染函数做复杂的抽象组件，比如过渡动画组件以及路由里面的link组件，都是用渲染函数实现的，同时还保留了它本身的依赖追踪系统。

如下图所示，Vue的依赖追踪通过ES5的 Object.defineProperty 方法实现。比如，我们给它一个原生对象，Vue会遍历这个数据对象的属性，然后进行属性转换。每一个属性会被转换为一个 getter 和一个 setter。同时每个组件会有一个对应的 watcher 对象，这个对象的职责就是在当前组件被渲染的时候，记录数据上面的哪些属性被用到了。

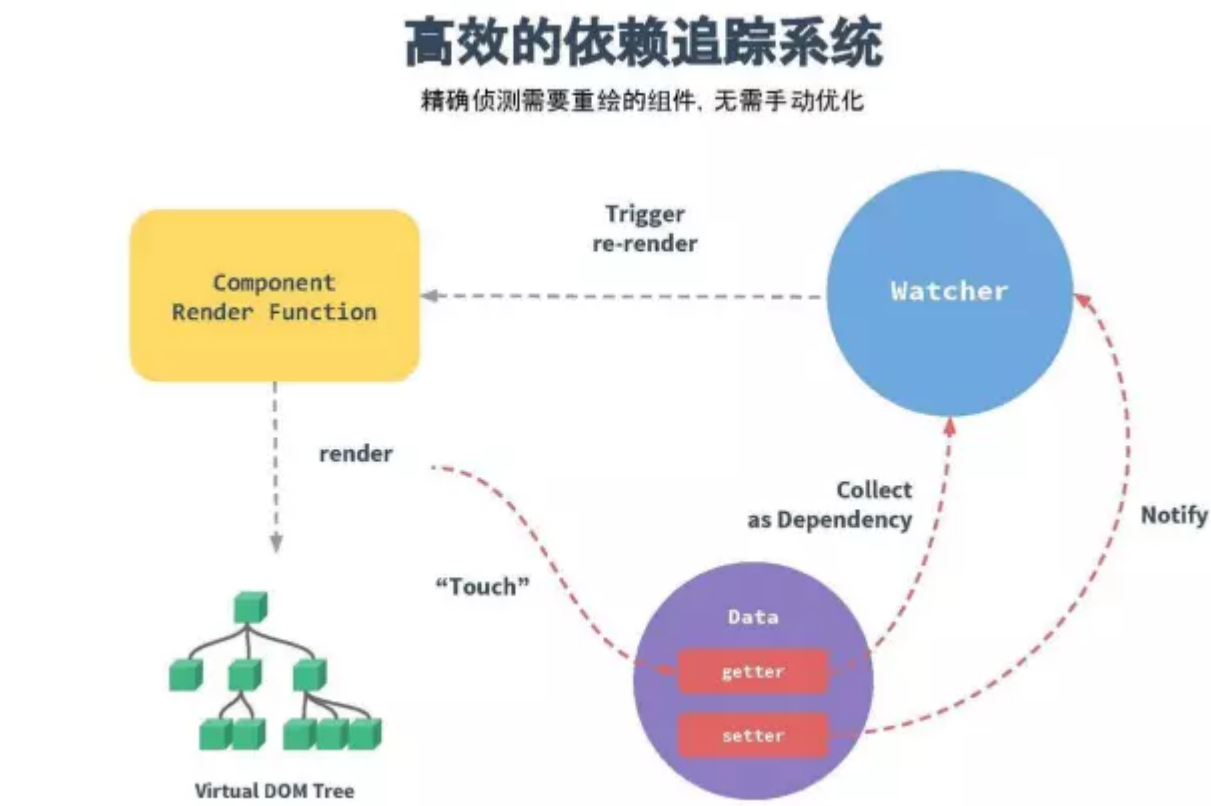
例如，在渲染函数里面用到A.B的时候，这个就会触发对应的 getter。整个渲染流程具体要点如下：

- 当某个数据属性被用到时，触发 getter，这个属性就会被作为依赖被 watcher 记录下来。
- 整个函数被渲染完的时候，每一个被用到的数据属性都会被记录。
- 相应的数据变动时，例如给它一个新的值，就会触发 setter，通知数据对象对应数据有变化。
- 此时会通知对应的组件，其数据依赖有所改动，需要重新渲染。

- 对应的组件再次调动渲染函数，生成 Virtual DOM，实现 DOM 更新。

这样一个流程跟主流的一些框架，例如React是有较大区别的。在React中，当组件复杂的时候需要用 shouldComponentUpdate 做优化。但是，它也有自己的各种坑，比如要确保该组件下面的组件不依赖外部的状态。虽说这在大部分情况下是够用的，但遇到极大复杂度的应用，遇到性能瓶颈的时候，这个流程优化起来也是相当复杂的一个话题。

如下图所示，在Vue里面由于依赖追踪系统的存在，当任意数据变动的时，Vue的每一个组件都精确地知道自己是否需要重绘，所以并不需要手动优化。用Vue渲染这些组件的时候，数据变了，对应的组件基本上去除了手动优化的必要性。



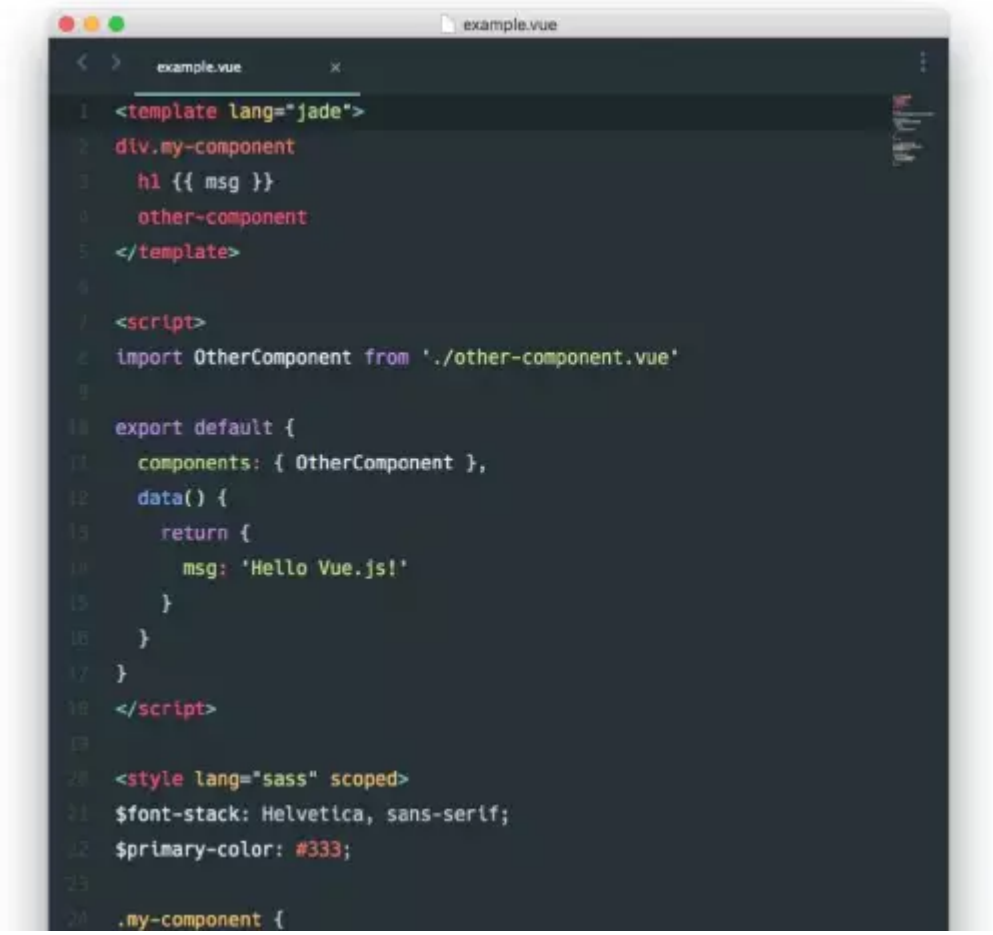
（2）组件系统

相信基本上所有的现代框架都已经走向了组件化道路，Web Components 从规范层面做这个实践。主流框架都有各有不同的封装，但核心思想都是一样，把UI结构映射到恰当的组件树。

在Vue中，父子组件之间的通信是通过 props 传递。从父向子单向传递；而如果子组件想要在父组件作用里面产生副作用，就需要去派发事件。这样就形成一个基本的父子通信模式，在涉及大规模状态管理的时候会有额外的方案，这个后面会提到。

Vue的组件引入构建工具之后有一个单文件组件概念，如下图所示，就是这个Vue文件。在同一个Vue文件里，可以同时写 template、script 和 style，三个东西放在一个里面。

同时，Vue的单文件组件和 Web Components 有一个本质不同，它是基于构建工具实现。这样的好处是有了一个构建的机会，可以对这些单文件组件做更多的分析处，在每一个语言块里可以单独使用不同的处理器，这点后面还会讲到。

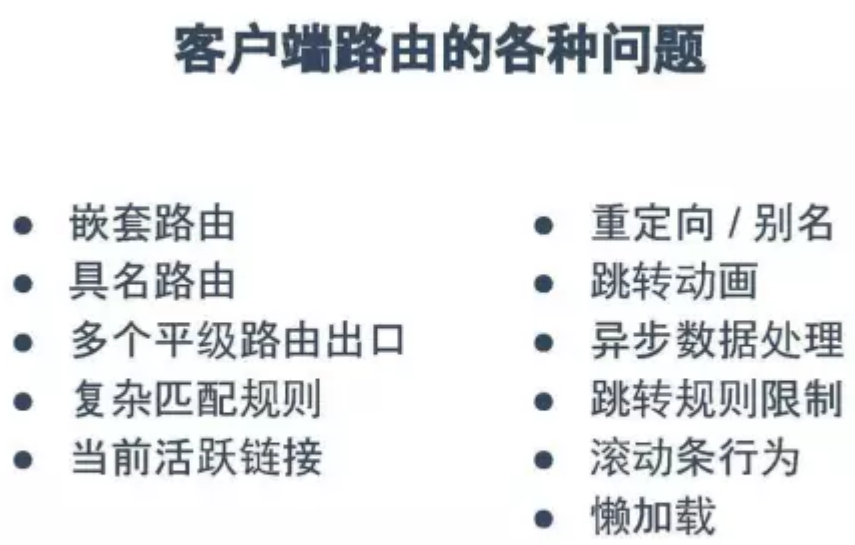


（3）客户端路由

在做一个界面复杂度非常的高应用时，它会有很多的状态，这样的应用显然不可能在每做一次操作后都刷新一个页面作为用户反馈。这就要这个应用有多个复杂的状态，同时这些状态还要对应到URL。

有一个重要的功能叫做 deep-linking，也就是当用户浏览到一个URL，然后把它传给另外的人或者复制重新打开，应用需要直接渲染出这个URL对应的状态。这就意味着应用的URL和组件树的状态之间有一个映射关系，客户端路由的职责就是让这个映射关系声明式地对应起来。

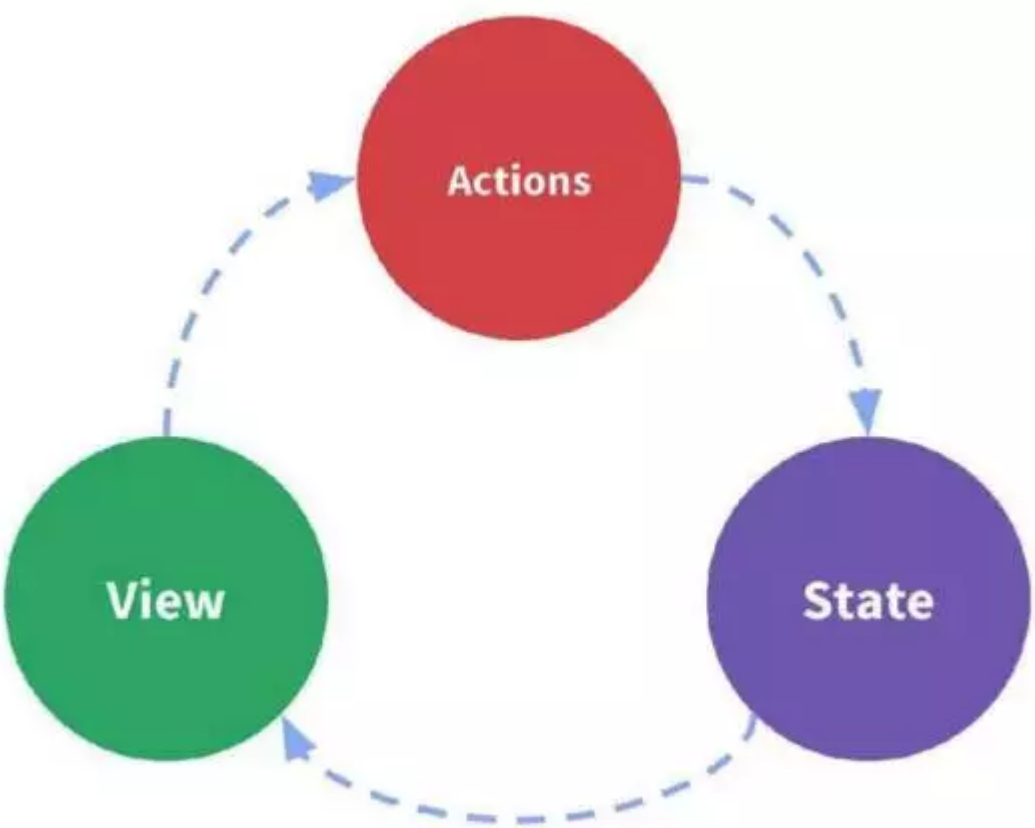
若要自己实现一个这样的路由，看上去倒是很简单，用hash去模拟一下，就可以自己很快地做出很简单的路由。但事实上，客户端路由涉及很多更复杂的问题，如下图所示。



可能同一层的路由有多个不同的出口，还有着复杂的URL匹配规则，等等。这些问题如果都由自己去一一实现，那么复杂度是非常高的。而Vue基本都有对应的解决方案（router.vuejs.org）。配合Webpack还可以实现基于路由的懒加载，一条路径所对应的组件在打包的时候，会分离成另外一块，只有当该路由被访问的时候，才会被加载出来。这有相应的解决方案，同时也有实例。

（4）状态管理

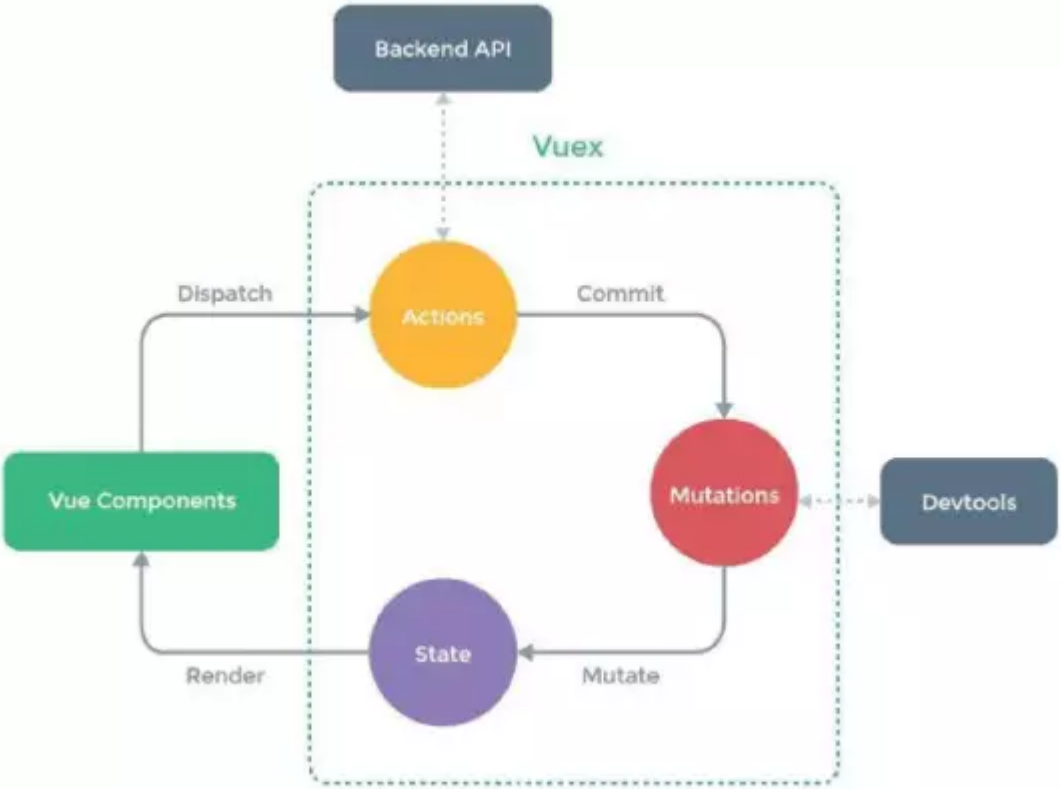
说到状态管理，本质上就是把整个应用抽象为下图中的循环。脸书最早提出 Flux 这个概念的时，也是一个很松散的概念，而且官方的实现本身做得很难用。所以，社区就做了各种各样的探索。图中的这三个东西是一个单向数据流，State 驱动 View 的渲染，而用户对 View 进行操作产生 Action，会使State产生变化，从而导致 View 重新渲染。



一个单独的Vue的组件，其实就已经是这样的结构。但是当多个这样的组件来配套的时候，就会遇到一个问题。每个组件都有它自己的状态，但整个应用的状态，跟组件之间并不一定存在一一对应的关系。这个状态可能是一个全局状态。那么状态到底放在哪里？大部分解决方案是把这个状态从组件树中提取出来，放在一个全局的 Store 里面。Vuex 也是这样做的，但是它是针对 Vue 做了特化。

我们看到最左边就是Vue的组件，这些组件在大部分情况下，就不再有私有的状态，而是从全局的 Store 里面获取状态。Actions 和 Mutations 比较难用一两句话说清楚，大致就是当应用状态进行改变的时候，需要通过 Mutations 去显式地触发，而 Actions 则是负责异步和其他副作用。

由于 Mutations 会被记录下来，我们可以把这些记录发到工具里面去做分析，甚至进行回滚。当发现bug的时候，这使得我们可以更好地理解大型应用中的状态变化。更多的细节，还请看官方文档（vuex.vuejs.org）。



（5）构建工具

构建工具方面，Vue有一个官方的，全局安装的 vue-cli。这里有一个笔误。全局安装之后，我们就可以用 vue 命令创建一个新的项目，Vue 的 CLI 跟其他 CLI 不同之处在于，有多个可选模板，有简单的也有复杂的。极简的配置，更快的安装，可以更快的上手。

它也有一个更完整的模板，包括单元测试在内的各种内容都涵盖，但是，它的复杂度也更高，这又涉及到根据用例来选择恰当复杂度的问题。所有的模板在创建之后，构建脚本都是通过 npm 脚本来执行，在国内安装 npm 依赖的时候有点卡，可以用 yarn 或者推荐用淘宝的 npm 镜象源，可以很大地提升安装速度。

之前提到了单文件组件，如下图所示，支持任意的处理器，开箱即用的热重载，所以组件都支持热重载 (hot-reload)。当你做了修改，不会刷新页面，只是对组件本身进行立刻重载，不会影响整个应用当前的状态。CSS也支持热重载。

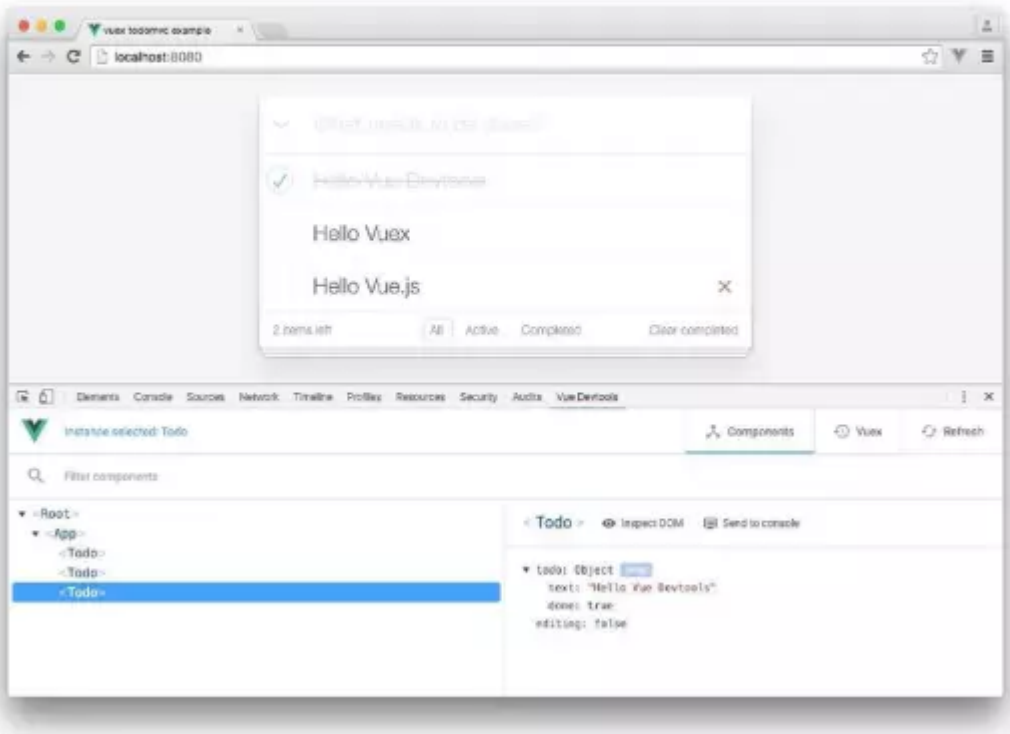
我们看下左下角，在使用这个预处理器的同时，我们只需要添加一个 scoped 特性，Vue 会通过对模板和CSS代码的解析改写，来模拟CSS的效果。同时单文件组件也支持懒加载，一个懒加载的组件和它的依赖会被打包成一个额外的包，只有被用到的时候才加载，这对首屏的加载速度也是很有帮助的。

Webpack 支持下的单文件组件

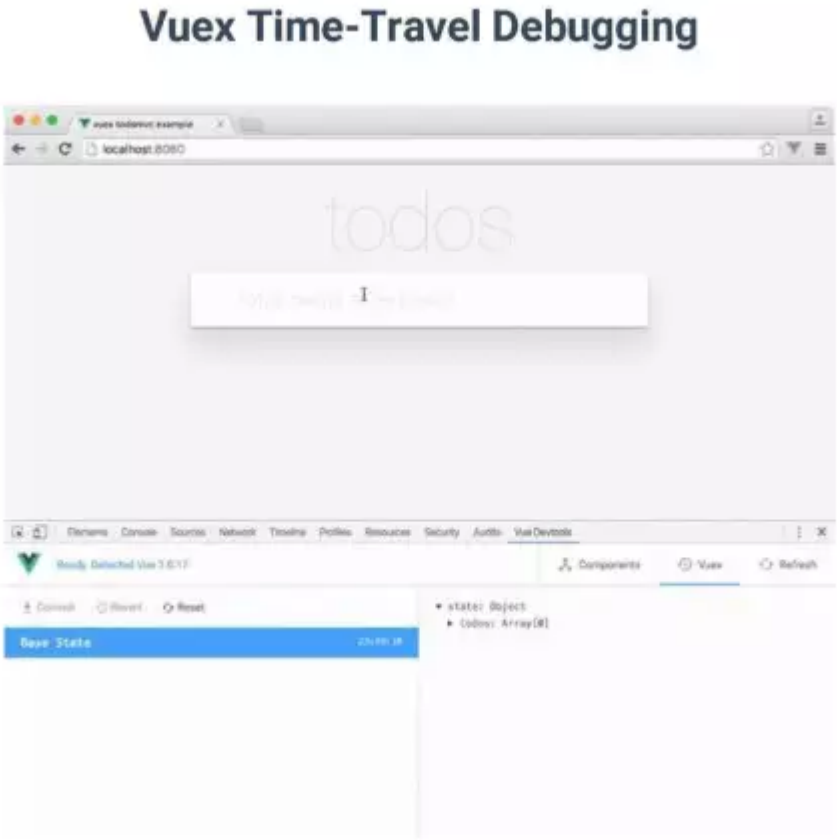
- 集中组件相关的模板，逻辑和样式
- 编译为 JavaScript 模块
- 各语言块支持任意编译 / 预处理器
  : Babel, TypeScript, LESS, Sass, PostCSS, Pug…
- 开箱即用的热重载，保存即更新
- Scoped CSS 支持
- 配合 Webpack 和 Vue 的异步组件特性
  轻松实现懒加载

如下图所示，这个开发者工具本身也是用Vue写的。

Chrome 开发者工具



使用它的话可以看到我们当前应用的组件树结构。



点击组件，就可以观察这个组件当前的状态。也可以把这个组件发送到控制台里。同时这个开发者工具还有一个 Vuex 面板，如果你用了 Vuex，那么每次操作都会被记录下来，记录下来的状态之间可以进行跳转。

除此之外，还支持把当前应用的状态快照发送给另外一个人，这个人可以在他的控制台里导入你发送的状态，就可以立刻跳转到你之前所在的状态。这对于重现一些 bug，或要描述当前状态都很有帮助。

Vue2.0

Vue2.0在不久之前刚刚发布（具体报道参见http://t.cn/RVC0foZ），之前一些技术细节在前文中已有所涉及。Vue2.0相对于1.0的改进有以下几点。

1、更轻

对Vue1.0大小压缩，Vue2.0它有一个只包含运行时的版本，所有的模板在编译的时候已经完成了。基于这个版本，下图中Vue、vue-router和vuex三个（都是 2.0 版本）加一起，跟Vue1.0的核心库大小一样。

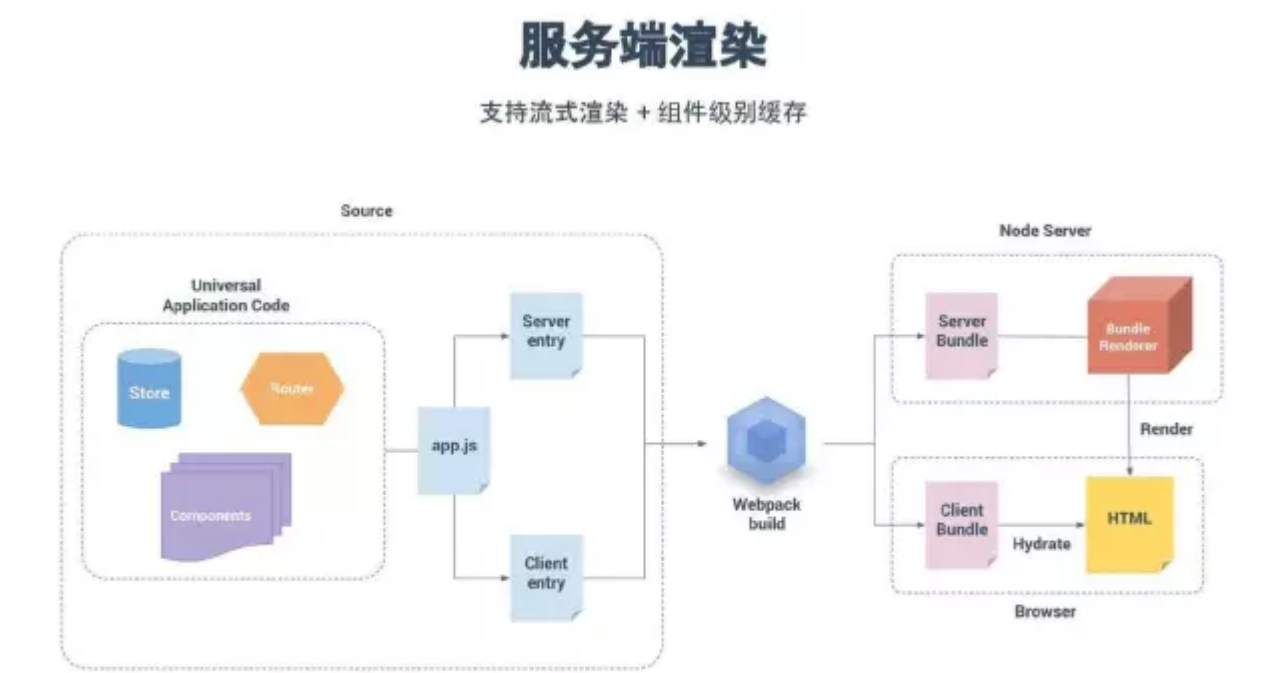
2、更快

Vue2.0可以说是当前最快的框架之一。这个是基于第三方独立测试的结果。有兴趣的话，可以移步链接（http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html）进行查看。

这个测试是一个比较综合的测试，它对于各种操作，以及在大列表里面更新移除等，都有相当完整的覆盖。可以看出，Vue2.0，不仅仅是在Vue1.0的基础上有很大提升，相比其他框架，也有相当明显的性能优势。

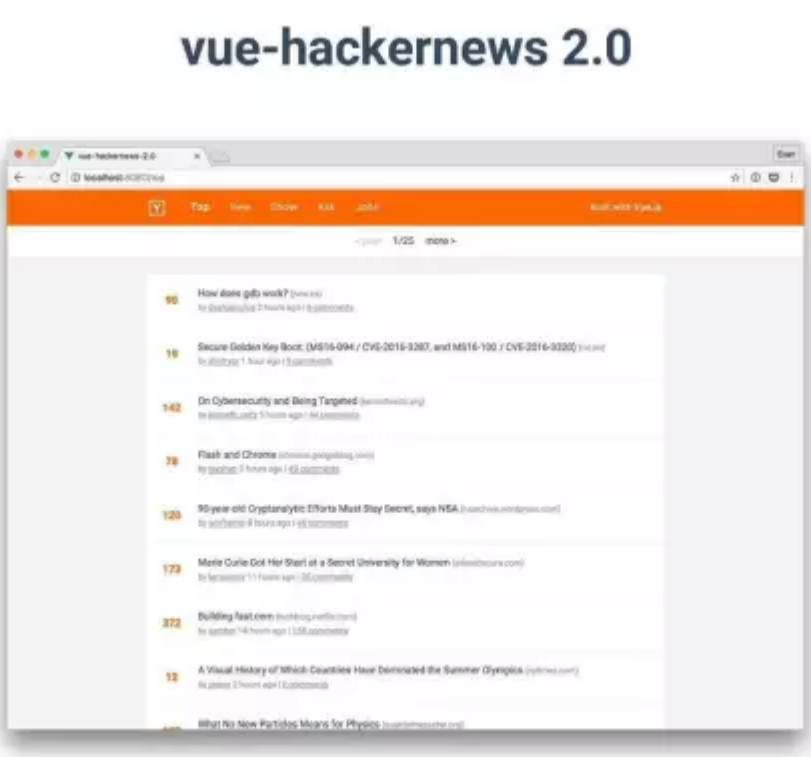
3、Vue2.0 架构

下图是Vue2.0的架构图，这里不深入讲整个架构的实现。



Vue2.0同时支持服务端，服务端渲染支持流式渲染。因为HTTP请求也是流式，Vue 的服务端渲染结果可以直接 pipe 到返回的请求里面。这样一来，就可以更早地在浏览器中呈现给用户内容，通过合理的缓存策略，可以有效地提升服务端渲染的性能。

下图是Vue2.0以及服务端渲染的相关内容，基本上把所有的功能都整合在了一起，有兴趣的同学可以在这里搜索2.0，它可以作为参考应用。



https://github.com/vuejs/vue-hackernews-2.0

除了服务端渲染还有原生渲染，这里的原生渲染是指阿里巴巴的项目Weex。在架构层面，通过编译一个 Weex 源文件（类似于 Vue 单文件组建的格式）然后运行。界面节点的操作都是抽象的，这些抽象操作会派发到不同的目标引擎做实际的渲染，同时支持 iOS, Android 和 Web。

Vue和Weex现在有一个合作，Vue 2.0 将会正式成为 Weex 的 JavaScript 运行时。这样的合作可以使得符合功能交集的Vue组件可以跨平台使用。

今日荐文

点击下方图片即可阅读

微信PaxosStore内存云揭秘：十亿Paxos/分钟的挑战

喜欢我们的会点赞，爱我们的会分享！