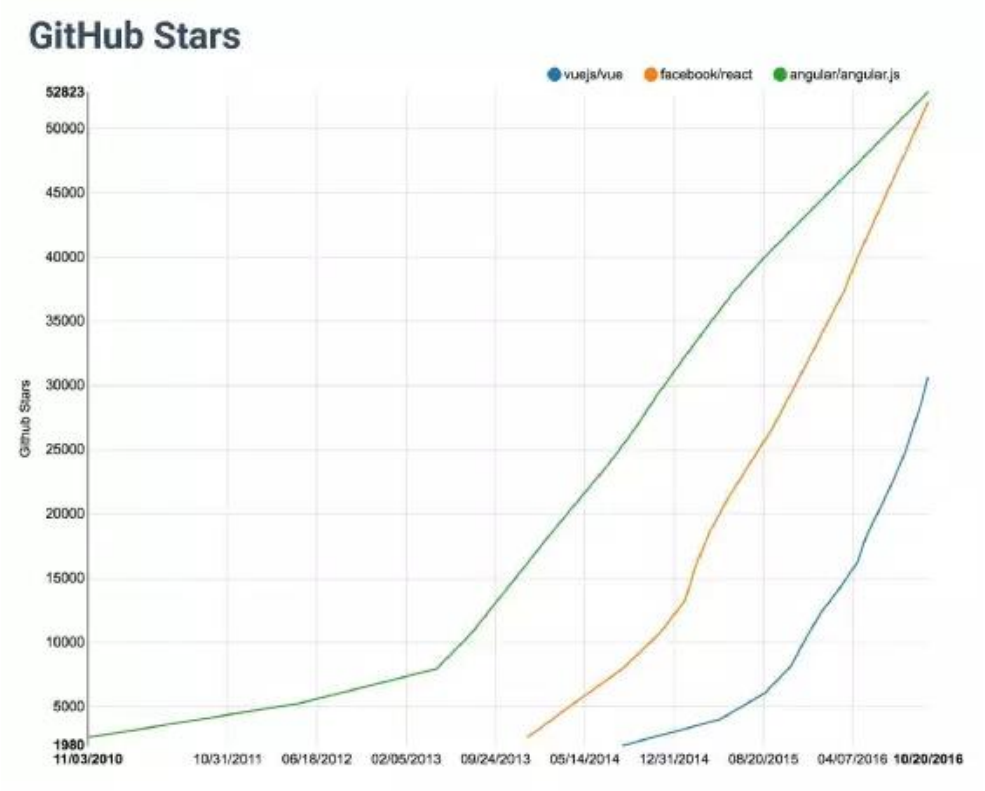
Vue.js

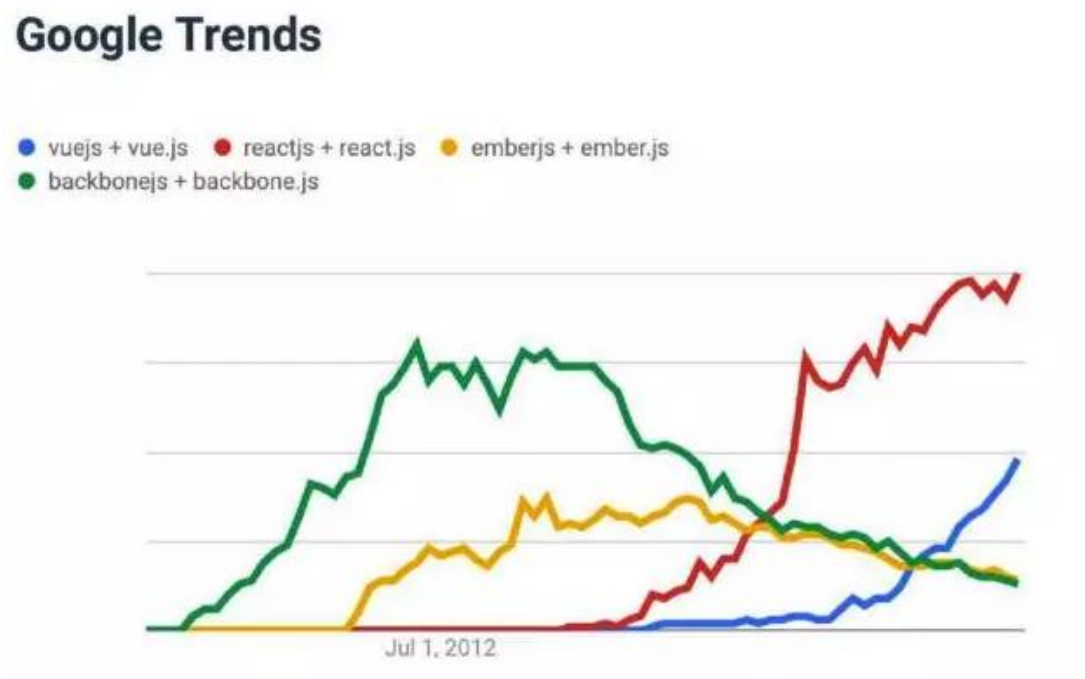
Vue.js 是一个构建数据驱动的 web 界面的框架。

Vue.js 的目标是通过尽可能简单的 API 实现**响应的数据绑定**和**组合的视图组件**。

现状



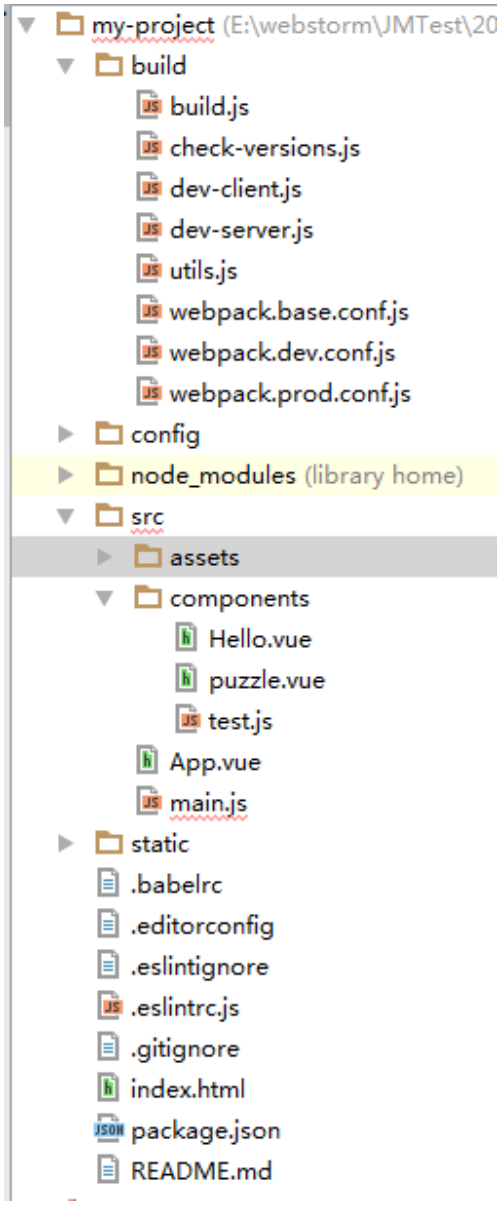
- ~30k stars on GitHub
- ~185k/mo downloads on NPM
- ~264k/mo unique visitors to vuejs.org
- ~55k weekly active Chrome extension users



安装使用

利用 vue-cli 来自动生成我们项目的前端目录及文件

```
# install vue-cli
$ npm install --global vue-cli
$ vue init webpack my-project
$ cd my-project
$ npm install
$ npm run dev
```



声明式渲染

DOM 应尽可能是一个函数式到状态的映射，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，而不需要手动选择一个元素，再命令式地去改动它的属性。

http://vuejs.org/guide/index.html

vue 2.0 引入了虚拟 DOM。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。函数被调用的时候就会渲染并且返回一个虚拟 DOM 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。

Javascript 表达式：

{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

有个限制就是，每个绑定都只能包含单个表达式，语句、控制流的都不行。

另一种定义模版的方式是在 JavaScript 标签里使用 text/x-template 类型 ,并且指定一个 id。

```
<script type="text/x-template" id="hello-world-template">
  <p>Hello hello hello</p>
</script>
```

```
Vue.component('hello-world', {
  template: '#hello-world-template'
})
```

Vue 实例：通过 new Vue()构建一个 Vue 的实例，在实例中，可以包含挂在元素（el），数据（data），模板（template），方法（methods）与生命周期钩子（created 等）等选项。钩子的 this 指向调用它的 Vue 实例。

```
new Vue({
  el: '#app',
  data: {
    msg: 1,
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  },
  methods:{
    toggle: function() {
      this.msg++;
    }
  },
  template: '<p>hello</p>',
  created: function() {
    console.log(this.msg);
  }
});
```

```
<body>
  <div id="app">
    <button @click='toggle()'>{{msg}}</button>
    <ol>
      <!--
      Now we provide each todo-item with the todo object
      it's representing, so that its content can be dynamic
      -->
      <todo-item v-for="todo in todos" v-bind:todo="todo"></todo-item>
    </ol>
  </div>

  <!-- built files will be auto injected -->
</body>
```

vue 1.0+	vue 2.0	Description
init	beforeCreate	组件实例刚被创建，组件属性计算之前，如 data 属性等
created	created	组件实例创建完成，属性已绑定，但 DOM 还未生成，`$el` 属性还不存在
beforeCompile	beforeMount	模板编译/挂载之前
compiled	mounted	模板编译/挂载之后
ready	mounted	模板编译/挂载之后（不保证组件已在 document 中）
-	beforeUpdate	组件更新之前
-	updated	组件更新之后
-	activated	for `keep-alive`，组件被激活时调用
-	deactivated	for `keep-alive`，组件被移除时调用
attached	-	不用了还说啥哪...
detached	-	那就不说了吧...
beforeDestory	beforeDestory	组件销毁前调用
destoryed	destoryed	组件销毁后调用

指令

```
<span v-text="msg"></span>
<div v-html="html"></div>  注意 XSS 攻击
<h1 v-if="ok">Yes</h1>
<h1 v-else>No</h1>
<h1 v-show="ok">Hello!</h1>
<div v-for="(item, index) in items"></div>
<button v-on:click="doThat('hello', $event)"></button>
<button @click.prevent="doThis"></button>
<input @keyup.enter="onEnter">
```

- **修饰符：**
  - `.stop` - 调用 `event.stopPropagation()` 。
  - `.prevent` - 调用 `event.preventDefault()` 。
  - `.capture` - 添加事件侦听器时使用 capture 模式。
  - `.self` - 只当事件是从侦听器绑定的元素本身触发时才触发回调。
  - `.{keyCode | keyAlias}` - 只当事件是从侦听器绑定的元素本身触发时才触发回调。
  - `.native` - 监听组件根元素的原生事件。

```
<button v-bind:disabled="someDynamicCondition">Button</button>
```

```
<img :src="imageSrc">
```

v-model 指令在表单控件元素上创建双向数据绑定

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

- **修饰符：**
  - `.lazy` - 取代 `input` 监听 `change` 事件
  - `.number` - 输入字符串转为数字
  - `.trim` - 输入首尾空格过滤

自定义指令

```
<div v-demo="{ color: 'white', text: 'hello!' }"></div>
```

```
Vue.directive('demo', function (el, binding) {
  console.log(binding.value.color) // => "white"
  console.log(binding.value.text)  // => "hello!"
})
```

http://vuejs.org/guide/custom-directive.html

计算属性

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

结果：

Original message: "Hello"

Computed reversed message: "olleH"

通过调用表达式中的 method 来达到同样的效果

```
<p>Reversed message: "{{ reverseMessage() }}"</p>
```

```
// in component
methods: {
  reverseMessage: function () {
    return this.message.split('').reverse().join('')
  }
}
```

计算属性是基于它的依赖缓存，只有在它的相关依赖发生改变时才会重新取值。这就意味着只要 message 没有发生改变，多次访问 reversedMessage 计算属性会立即返回之前的计算结果，而不必再次执行函数。

样式绑定

绑定 html calss

```
<div v-bind:class="{ active: isActive }"></div>
```

绑定内联样式

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

组件

```
// 注册
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
```

```
<template>
  <div id="app">
    <hello></hello>
  </div>
</template>

<script>
import Hello from './components/Hello';

export default {
  name: 'app',
  components: {
    Hello
  }
};
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

注意：当使用 DOM 作为模版时会受 , <ol> , <table> , <select>等元素限制

```
<table>
  <my-row>...</my-row>
</table>
```

使用特殊的 is 属性

```
<table>
  <tr is="my-row"></tr>
</table>
```

使用组件时， data 必须是函数。

```
var data = { counter: 0 }

Vue.component('simple-counter', {
  template: '<button v-on:click="counter += 1">{{ counter }}</button>',
  // data is technically a function, so Vue won't
  // complain, but we return the same object
  // reference for each component instance
  data: function () {
    return data
  }
})
```

父组件通过 props 向下传递数据给子组件，子组件通过 events 给父组件发送消息。

```
Vue.component('child', {
  // declare the props
  props: ['message'],
  // just like data, the prop can be used inside templates
  // and is also made available in the vm as this.message
  template: '<span>{{ message }}</span>'
})
```

属性检测

```
Vue.component('example', {
  props: {
    // basic type check (`null` means accept any type)
    propA: Number,
    // multiple possible types
    propB: [String, Number],
    // a required string
    propC: {
      type: String,
      required: true
    },
    // a number with default value
    propD: {
      type: Number,
      default: 100
    },
    // object/array defaults should be returned from a
    // factory function
    propE: {
      type: Object,
      default: function () {
        return { message: 'hello' }
      }
    },
    // custom validator function
    propF: {
      validator: function (value) {
        return value > 10
      }
    }
  }
})
```

prop 是单向绑定的，每次父组件更新时，子组件的所有 prop 都会更新为最新值。这意味着不应该在子组件内部改变 prop 。

在 JavaScript 中对象和数组是引用类型，指向同一个内存空间，如果 prop 是一个对象或数组，在子组件内部改变它会影响父组件的状态。

使用 $on(eventName) 监听事件

使用 $emit(eventName) 触发事件

```
<div id="counter-event-example">
  <p>{{ total }}</p>
  <button-counter v-on:increment="incrementTotal"></button-counter>
  <button-counter v-on:increment="incrementTotal"></button-counter>
</div>
```

```
Vue.component('button-counter', {
  template: '<button v-on:click="increment">{{ counter }}</button>',
  data: function () {
    return {
      counter: 0
    }
  },
  methods: {
    increment: function () {
      this.counter += 1
      this.$emit('increment')
    }
  },
})

new Vue({
  el: '#counter-event-example',
  data: {
    total: 0
  },
  methods: {
    incrementTotal: function () {
      this.total += 1
    }
  }
})
```

与 react 和 angular 的比较

React

相似之处：使用虚拟 DOM；响应式、组件化；相对完整的生态系统，包括路由、状态的管理。

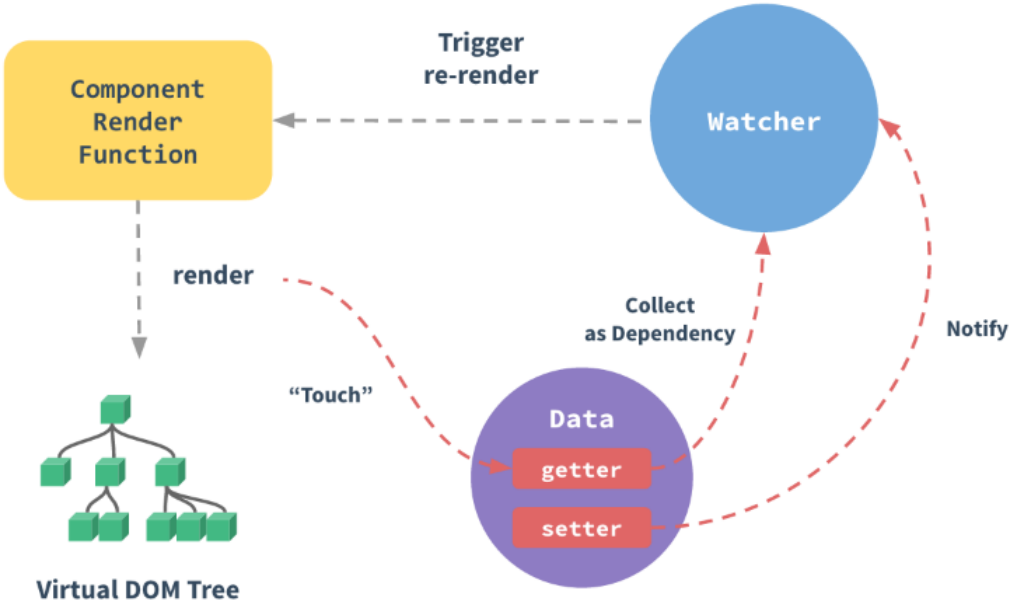
但性能上 vue 优于 react

在 React 中，我们设定渲染一个元素的额外开销是 1，那么在 Vue 中，一个元素的开销更像是 0.1。

2014 年产的 MacBook Air 在 Chrome 52 版本下运行负责渲染 10,000 个列表项 100 次的项目所产生的数据如下

	Vue	React
Fastest	23ms	63ms
Median	42ms	81ms
Average	51ms	94ms
95th Perc.	73ms	164ms
Slowest	343ms	453ms

另外更新性能更好。在 React 中，需要在处处去实现 shouldComponentUpdate，并且用不可变数据结构才能实现最优化的渲染。在 Vue 中，组件的依赖被自动追踪。
每个组件实例都有相应的 **watcher** 程序实例，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 `setter` 被调用时，会通知 `watcher` 重新计算，从而致使它关联的组件得以更新。



模板语言比 jsx 可读性更高。

CSS 组件作用域

```
<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-1）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-1]:hover。

Angular

没有控制器

灵活性和模块化；

Vue 有更好的性能，并且非常容易优化，因为它不使用脏检查。

vue-router： Vue 提供的前端路由工具，利用其我们实现页面的路由控制，局部刷新及按需加载，构建单页应用，实现前后端分离。

vuex：Vue 提供的状态管理工具，用于同一管理我们项目中各种数据的交互和重用，存储我们需要用到数据对象。

文档：http://vuejs.org/guide/index.html

例子：https://luozhihao.github.io/vue-puzzle/index.html#

https://github.com/luozhihao/vue-puzzle/blob/master/src/App.vue