2017年11月 技术分享会

2017年11月3日

Agenda

- 《Vue.js（初阶篇）》

  -- Xiaojun Pan

- 《Webpack利器入门》

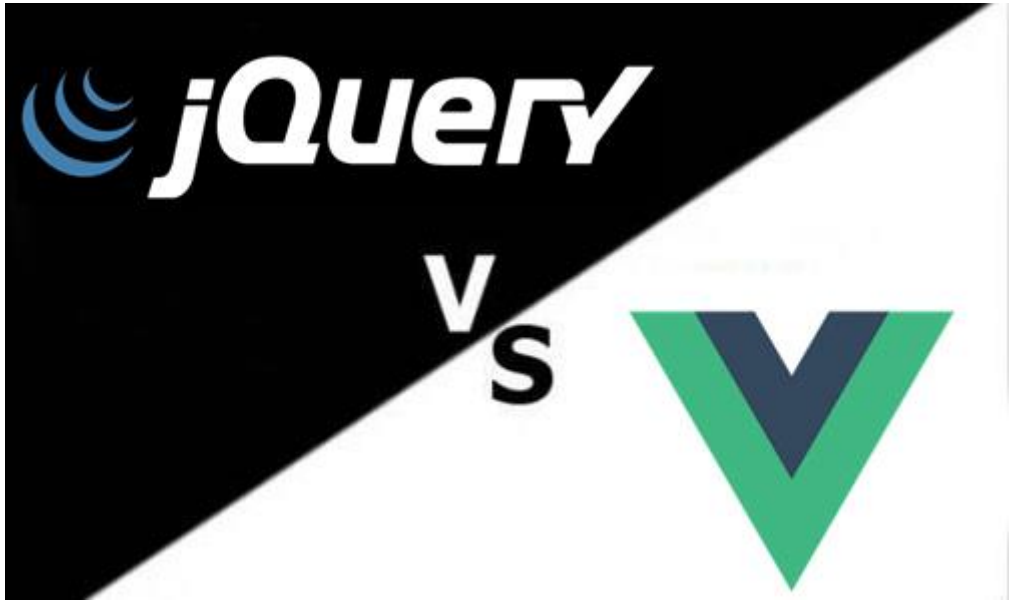
  -- Xiaojun Pan

Vue.js（初阶篇）

XIAOJUN PAN

2017年11月 技术分享会

JQuery vs Vue.js



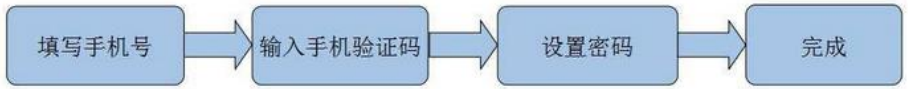
http://www.jianshu.com/p/f2f042d65bf6

JQuery vs Vue.js

场景1： 注册账号：

本场景主要体现一个页面多个步骤的逻辑处理， 类似的场景包括：购买流程、商品购买预约流程等；

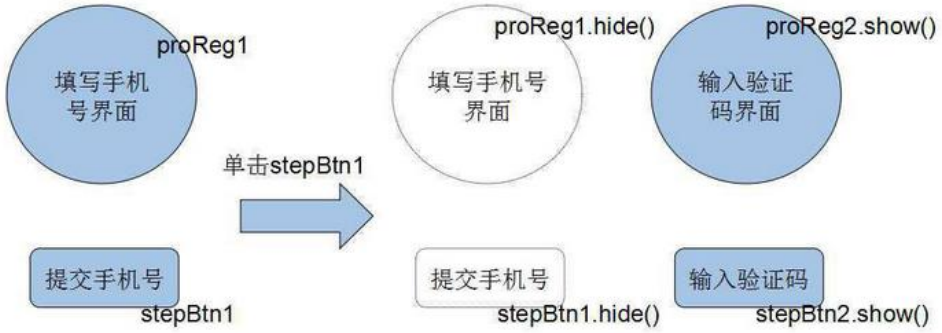
注册账号设计图如下：



1.1. JQ实现方式：

如贴代码， 那本文将被代码完全占据， 所以代码部分以链接方式附上； jq 的实现思路如下：**选择 流程dom对象， 点击按钮隐藏当前活动流程dom对象，显示下一流程dom对象。**

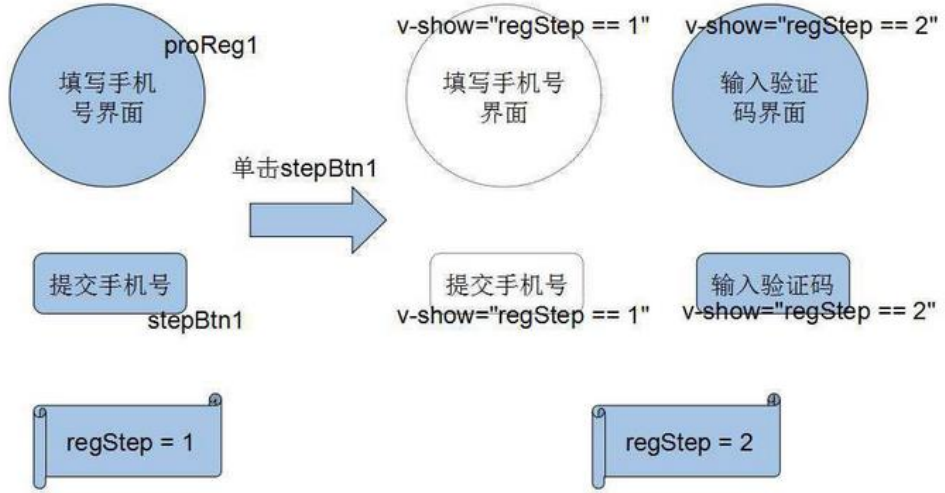
实现思路图：



1.2.VUE实现方式：

与jq不同 mvvm框架基本不操作dom节点， 双向绑定使 dom节点跟变量绑定后， 通过修改变量的值控制dom节点的各类属性。所以其实现思路为： **视图层使用一变量控制dom节点显示与否，点击按钮则改变该变量。**

实现思路图：

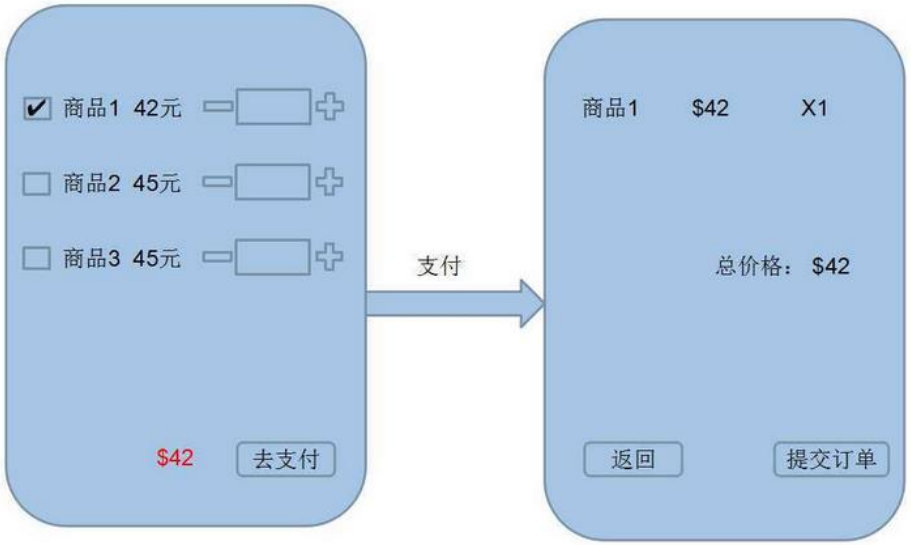


JQuery vs Vue.js

场景2： 购物列表：

本场景主要体现界面交互较多的逻辑处理， 类似的场景包括： 用户资料填写（城市、性别点选）、ERP工单申请（申请类型点选）等；

PS: 实际项目中的电商网站不会将购物车、订单结果单页面显示；因为这样不好兼容也不安全， 因而本例只是提供一种多交互的场景：
购物列表设计图如下：

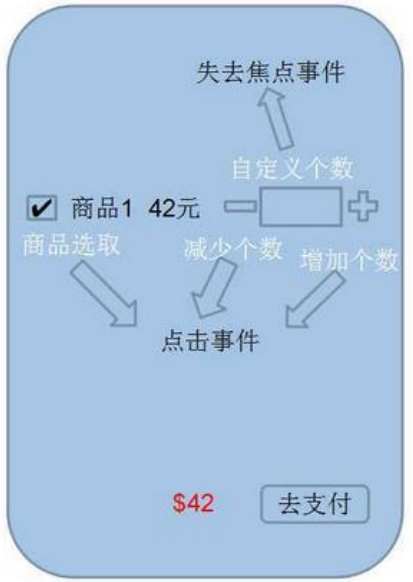


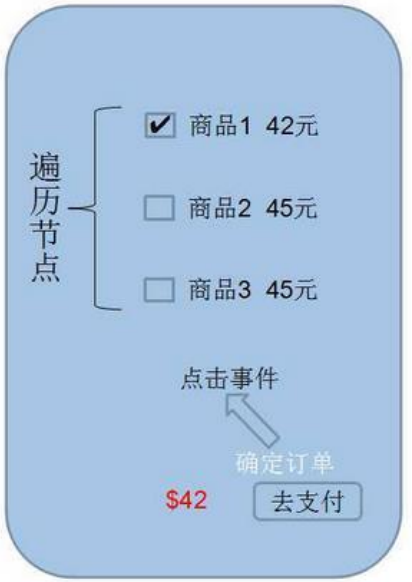
2.1.JQ实现方式：

jquery 主要考虑 勾选、增加、减少、编辑 商品时对应的逻辑， 对这四种操作赋予不同的事件（点击、失去焦点）；所有事件围绕总价格变化， 故公共事件就是修改价格显示的dom节点；

点击“去支付”按钮时， 遍历商品列表节点然后显示已勾选的商品；

实现思路图：



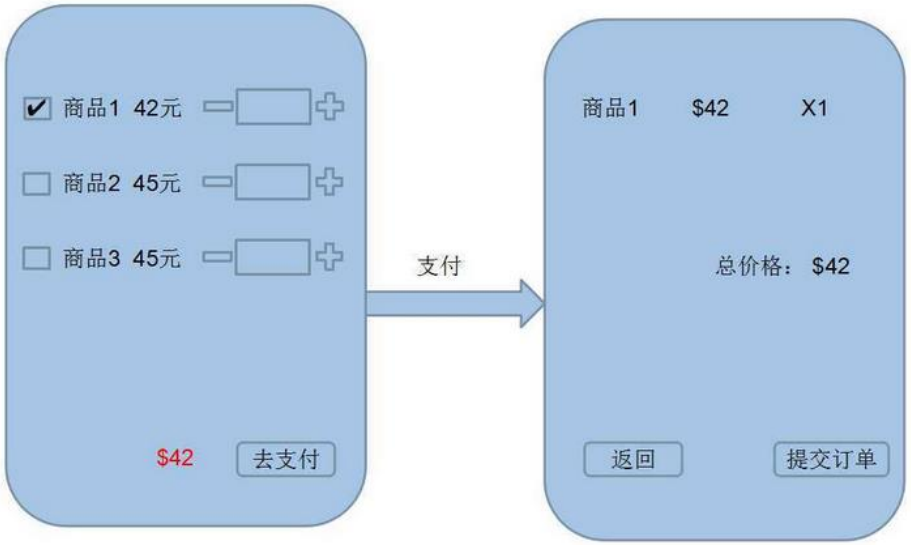


JQuery vs Vue.js

场景2： 购物列表：

本场景主要体现界面交互较多的逻辑处理， 类似的场景包括： 用户资料填写（城市、性别点选）、ERP工单申请（申请类型点选）等；

PS: 实际项目中的电商网站不会将购物车、订单结果单页面显示；因为这样不好兼容也不安全， 因而本例只是提供一种多交互的场景：
购物列表设计图如下：

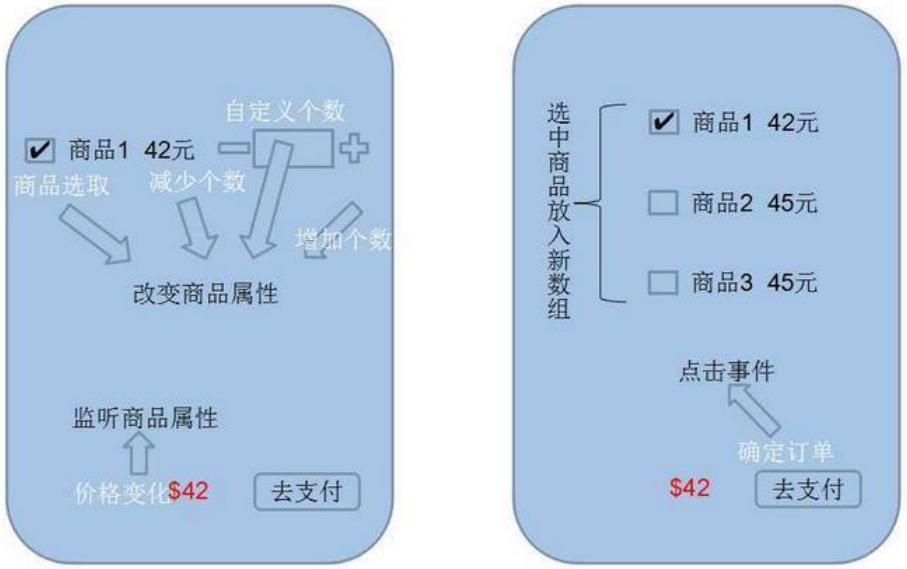


2.2.VUE实现方式：

实现与上述jq一样的功能， 在不操作dom节点的情况下vue如何实现呢？商品列表当然是使用v-repeat 遍历实现展示，主要难点在于如何交互： 勾选、增加、减少商品数量时如何改变**总价格？** 这时就用到了对象数组的**深度监听****（具体实现请参见代码）

使用一新数组**存储选中商品，点击"去支付"按钮时， 直接（v-repeat方式）显示该新数组即可。

实现思路图：

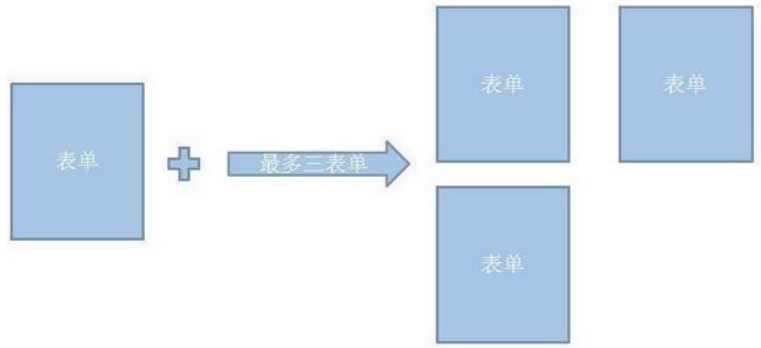


JQuery vs Vue.js

场景3：表单提交：

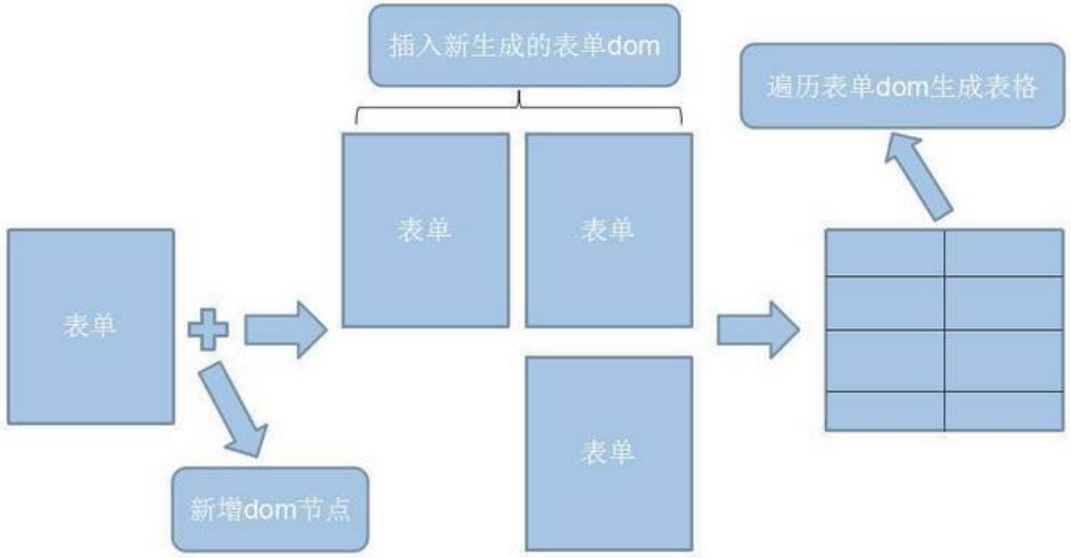
本场景主要在于mvvm框架如何动态添加新dom节点；
之前有知友提过类似问题：
angular js的点击一个li标签，触发事件添加li标签或者添加div应该怎么写？ - 前端开发

表单提交设计图：



3.1.JQ实现方式：

jq实现思路很简单： 监听"新增"按钮点击事件，然后生成dom节点，插入到表单父类节点中。点击"提交"按钮时，遍历所有表单，从表单中获取用户填写的数据即可；
（PS: 本部分代码中没细化用户交互，表单提交时判空处理、表单项手机格式、身份证格式校验等均没实现）
实现思路图：

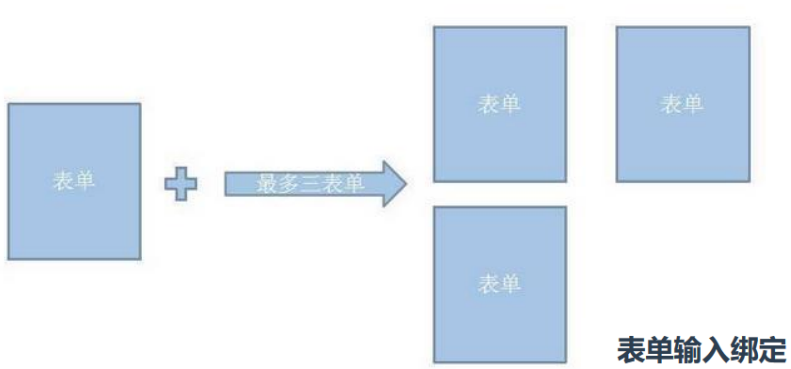


JQuery vs Vue.js

场景3：表单提交：

本场景主要在于mvvm框架如何动态添加新dom节点；
之前有知友提过类似问题：
angular js的点击一个li标签，触发事件添加li标签或者添加div应该怎么写？ - 前端开发

表单提交设计图：



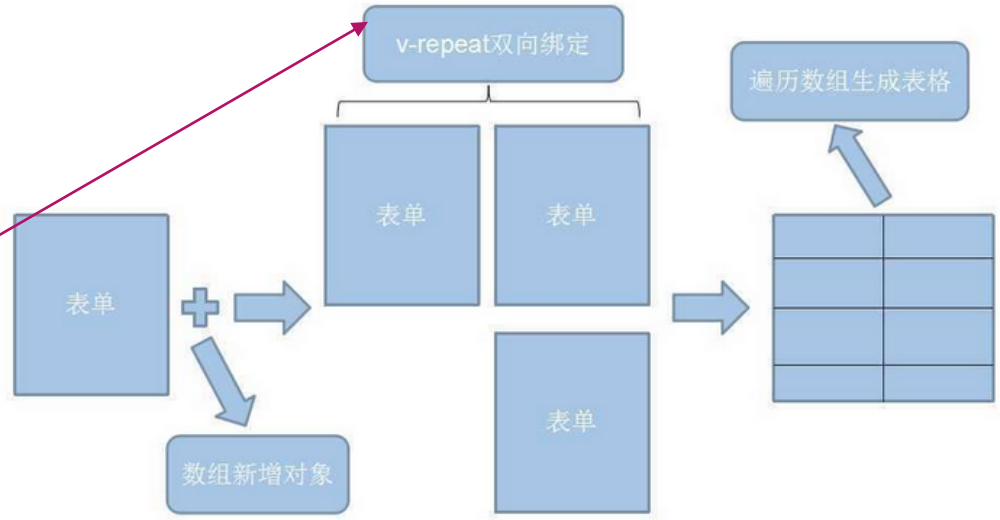
表单输入绑定

基础用法

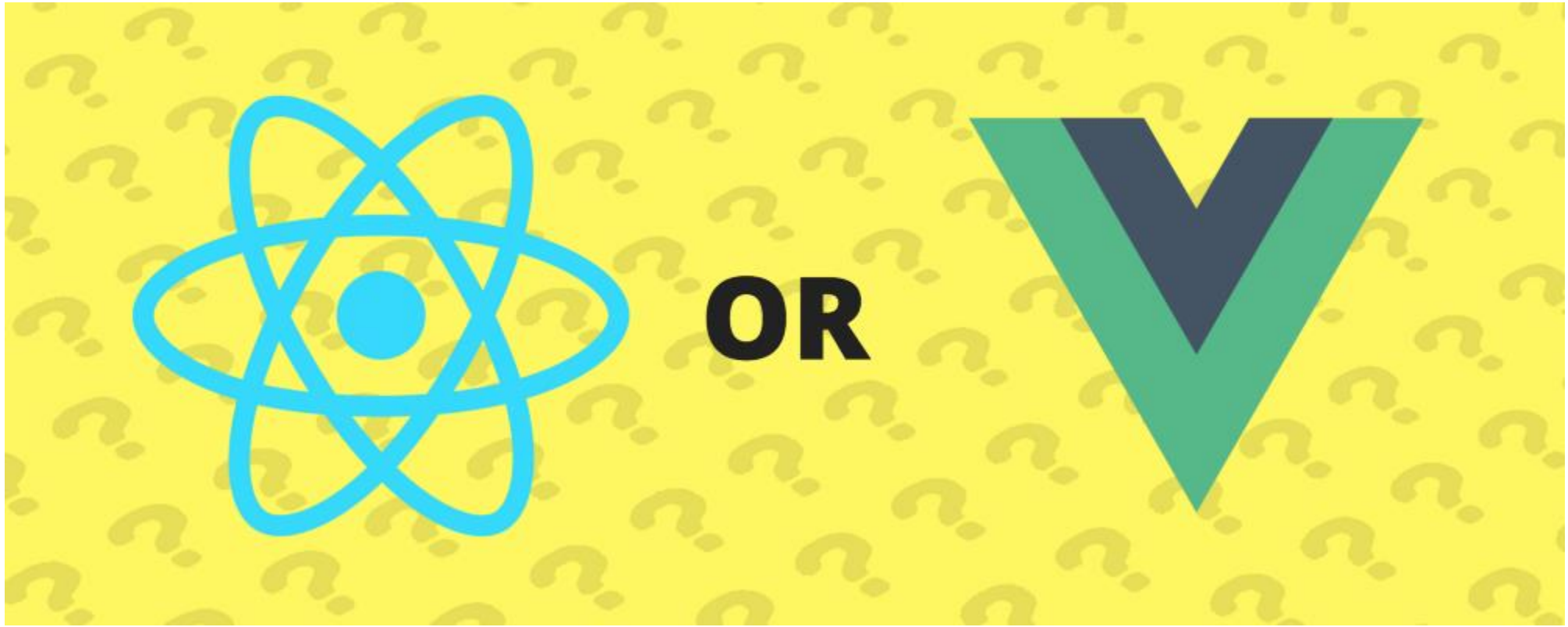
你可以用 `v-model` 指令在表单控件元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇，但 `v-model` 本质上不过是语法糖，它负责监听用户的输入事件以更新数据，并特别处理一些极端的例子。

3.2.VUE实现方式：

mvvm框架相对于jq给使用者感触最深的或许就是表单， 无论是vue还是angularjs在表单的处理上都有很多特定的官方指令；可以去官网感受下： vue表单用法；
所以说mvvm框架最适合做erp类型单页面应用， 一来不用管seo, 二来开发效率极高；
vue实现思路跟场景2差不多， v-repeat双向绑定；只要给绑定的数组添加新数据；对应的dom节点就会对应变化， 点击"提交"按钮时，直接展示该数组即可；
（表单操作时，vue对比jq优势就比较明显了，少了操作dom代码； 开发、维护效率都会明显提高）
实现思路图：



React.js vs Vue.js

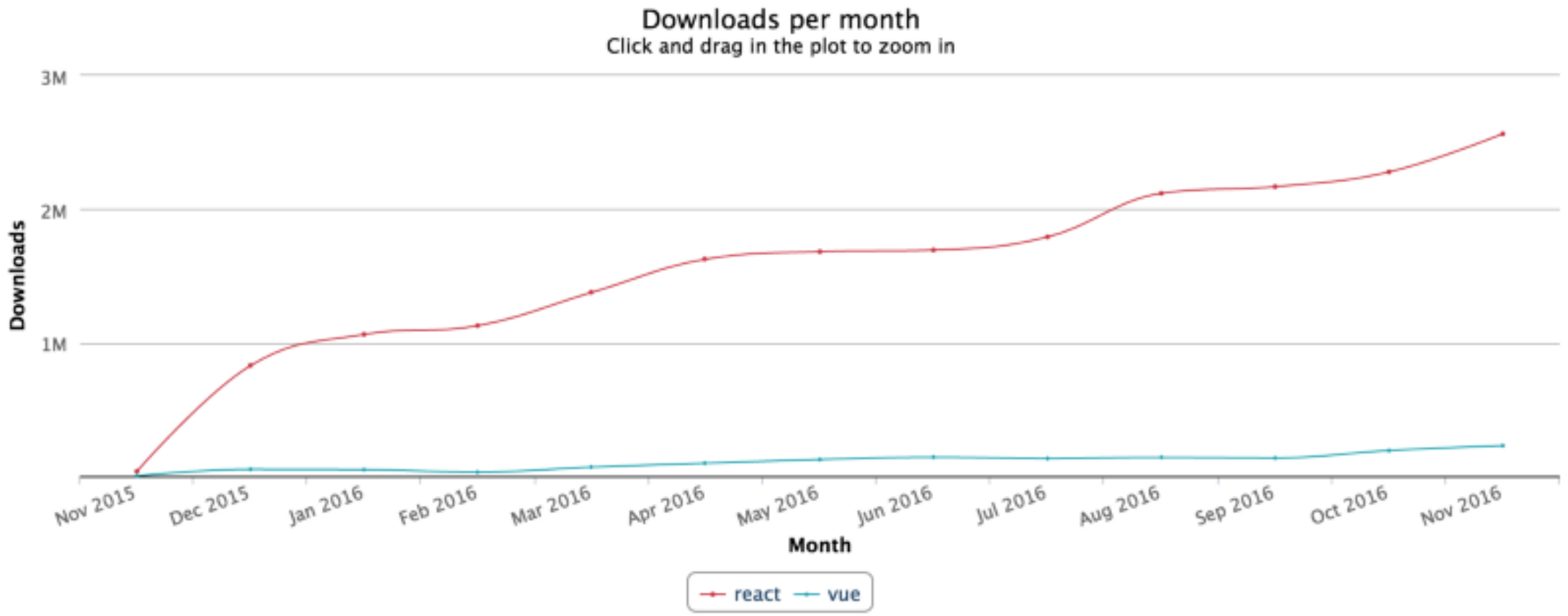


http://www.zcfy.cc/article/react-or-vue-which-javascript-ui-library-should-you-be-using-2159.html

React.js vs Vue.js

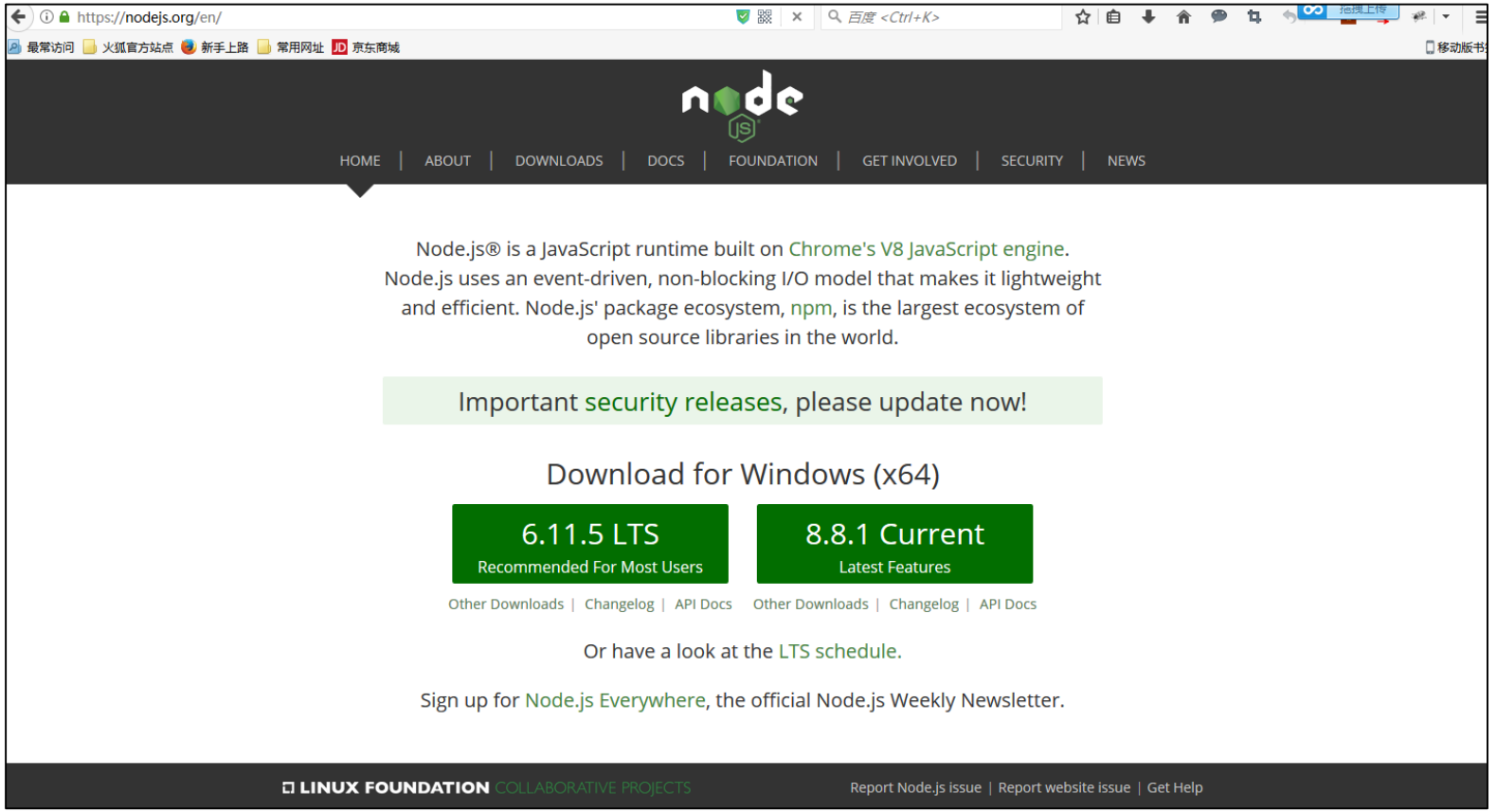
- **如果你喜欢用（或希望能够用）模板搭建应用，请使用Vue**
- **如果你喜欢简单和“能用就行”的东西，请使用Vue**
- **如果你的应用需要尽可能的小和快，请使用Vue**
- **如果你计划构建一个大型应用程序，请使用React**
- **如果你想要一个同时适用于Web端和原生App的框架，请选择React**
- **如果你想要最大的生态圈，请使用React**
- **如果你已经对其中一个用得满意了，就没有必要换了**

React.js vs Vue.js



工具篇

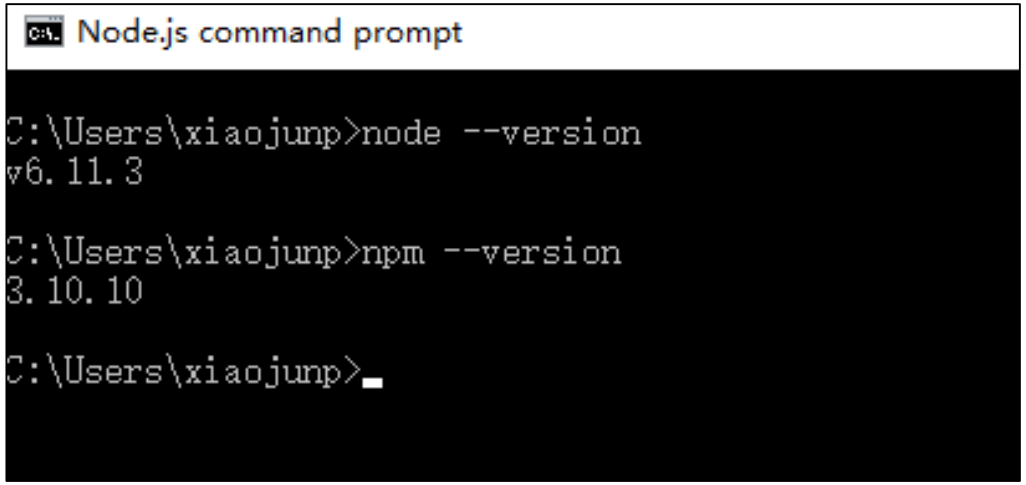
- Node.js


https://nodejs.org/en/
HOME | ABOUT | DOWNLOADS | DOCS | FOUNDATION | GET INVOLVED | SECURITY | NEWS
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.
Important security releases, please update now!
Download for Windows (x64)
6.11.5 LTS
Recommended For Most Users
8.8.1 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs
Or have a look at the LTS schedule.
Sign up for Node.js Everywhere, the official Node.js Weekly Newsletter.
LINUX FOUNDATION COLLABORATIVE PROJECTS
Report Node.js issue | Report website issue | Get Help

npm config set registry http://registry.npm.taobao.org/

工具篇

- npm



将 Npm 的源替换成淘宝的源

在墙内久了，难免会碰到撞墙的时候，所幸国内也有众多 NPM 镜像可供选择，在大多数情况下我们可以使用国内的源（比如 淘宝 NPM 镜像）去替换官方的源以加快下载包的速度。

不过呢，我们在发布自己的包的时候却需要将源修改回官方的 https://registry.npmjs.org/ 源。

修改源地址为淘宝 NPM 镜像

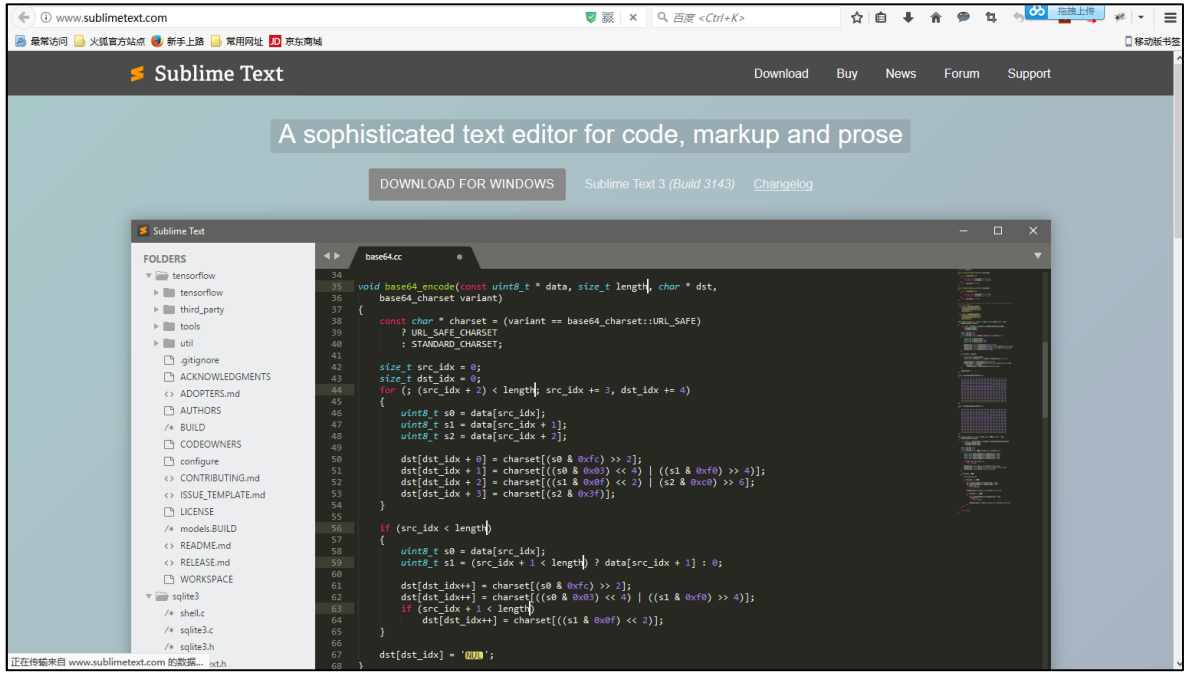
```
npm config set registry http://registry.npm.taobao.org/
```

修改源地址为官方源

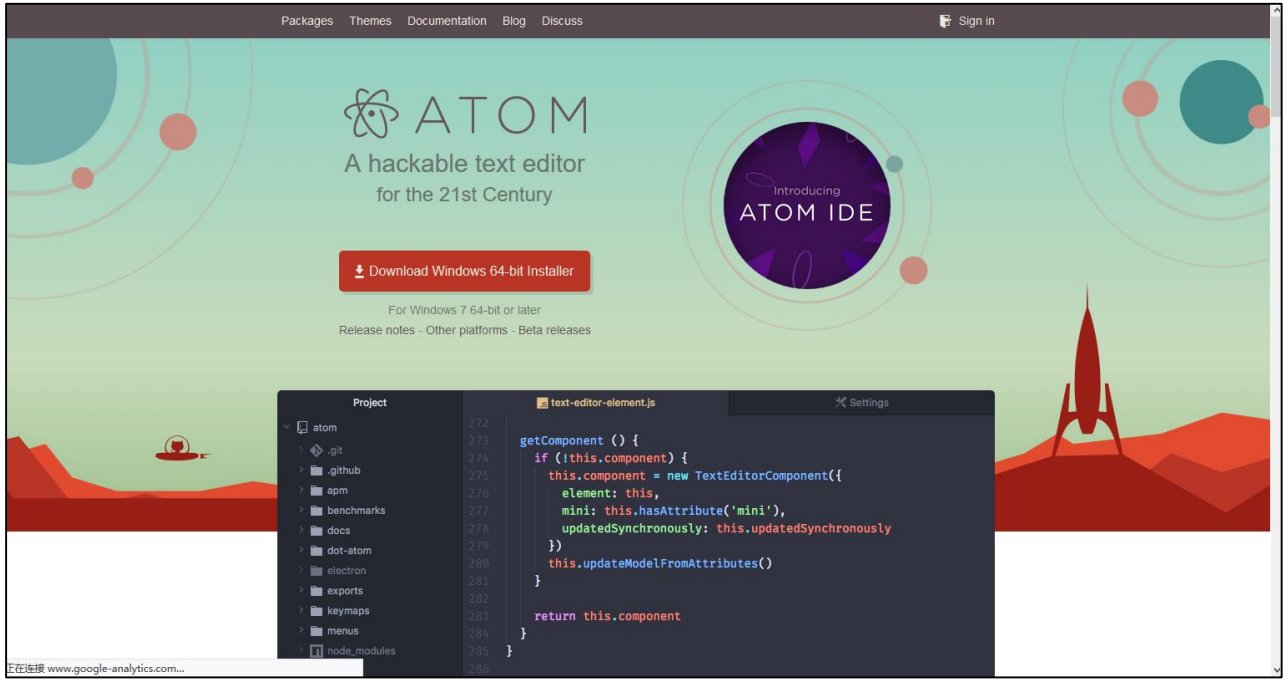
```
npm config set registry https://registry.npmjs.org/
```

工具篇

- Sumlime Text 3



- Atom (Github)



Vue.js基本语义

声明式渲染

Vue.js 的核心是一个允许采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统：

HTML

```
<div id="app">
  {{ message }}
</div>
```

JS

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

Hello Vue!

除了文本插值，我们还可以采用这样的方式绑定 DOM 元素属性：

HTML

```
<div id="app-2">
  <span v-bind:title="message">
    鼠标悬停几秒钟查看此处动态绑定的提示信息!
  </span>
</div>
```

JS

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: '页面加载于 ' + new Date().toLocaleString()
  }
})
```

鼠标悬停几秒钟查看此处动态绑定的提示信息！

Vue.js基本语义

条件与循环

控制切换一个元素的显示也相当简单：

```HTML
<div id="app-3">
  <p v-if="seen">现在你看到我了</p>
</div>
```

```JS
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

现在你看到我了

在控制台里，输入 `app4.todos.push({ text: '新项目' })` ，你会发现列表中添加了一个新项。

还有其它很多指令，每个都有特殊的功能。例如， `v-for` 指令可以绑定数组的数据来渲染一个项目列表：

```HTML
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

```JS
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: '学习 JavaScript' },
      { text: '学习 Vue' },
      { text: '整个牛项目' }
    ]
  }
})
```

1. 学习 JavaScript
2. 学习 Vue
3. 整个牛项目

Vue.js基本语义

处理用户输入

为了让用户和你的应用进行互动，我们可以用 `v-on` 指令绑定一个事件监听器，通过它调用我们 Vue 实例中定义的方法：

HTML
```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">逆转消息</button>
</div>
```

JS
```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

Hello Vue.js!

逆转消息

Vue 还提供了 `v-model` 指令，它能轻松实现表单输入和应用状态之间的双向绑定。

HTML
```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

JS
```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

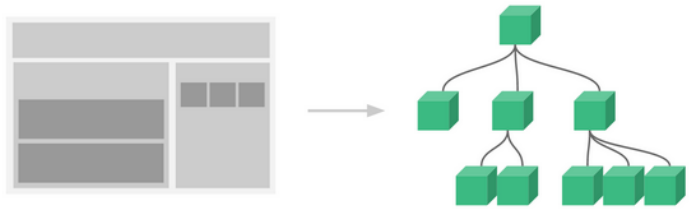
Hello Vue!

Hello Vue!

Vue.js基本语义

组件化应用构建

组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。仔细想想，几乎任意类型的应用界面都可以抽象为一个组件树：



在 Vue 里，一个组件本质上是一个拥有预定义选项的一个 Vue 实例，在 Vue 中注册组件很简单：

```JS
// 定义名为 todo-item 的新组件
Vue.component('todo-item', {
  template: '<li>这是个待办项</li>'
})
```

现在你可以用它构建另一个组件模板：

```HTML
<ol>
  <!-- 创建一个 todo-item 组件的实例 -->
  <todo-item></todo-item>
</ol>
```

但是这样会为每个待办项渲染同样的文本，这看起来并不炫酷，我们应该能将数据从父作用域传到子组件。让我们来修改一下组件的定义，使之能够接受一个**属性**：

```JS
Vue.component('todo-item', {
  // todo-item 组件现在接受一个
  // "prop"，类似于一个自定义属性
  // 这个属性名为 todo。
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

现在，我们可以使用 `v-bind` 指令将 todo 传到每一个重复的组件中：

```HTML
<div id="app-7">
  <ol>
    <!--
      现在我们为每个 todo-item 提供 todo 对象
      todo 对象是变量，即其内容可以是动态的。
      我们也需要为每个组件提供一个"key"，晚些时候我们会做个解释。
    -->
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id">
    </todo-item>
  </ol>
</div>
```

```JS
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})

var app7 = new Vue({
  el: '#app-7',
  data: {
    groceryList: [
      { id: 0, text: '蔬菜' },
      { id: 1, text: '奶酪' },
      { id: 2, text: '随便其他什么人吃的东西' }
    ]
  }
})
```

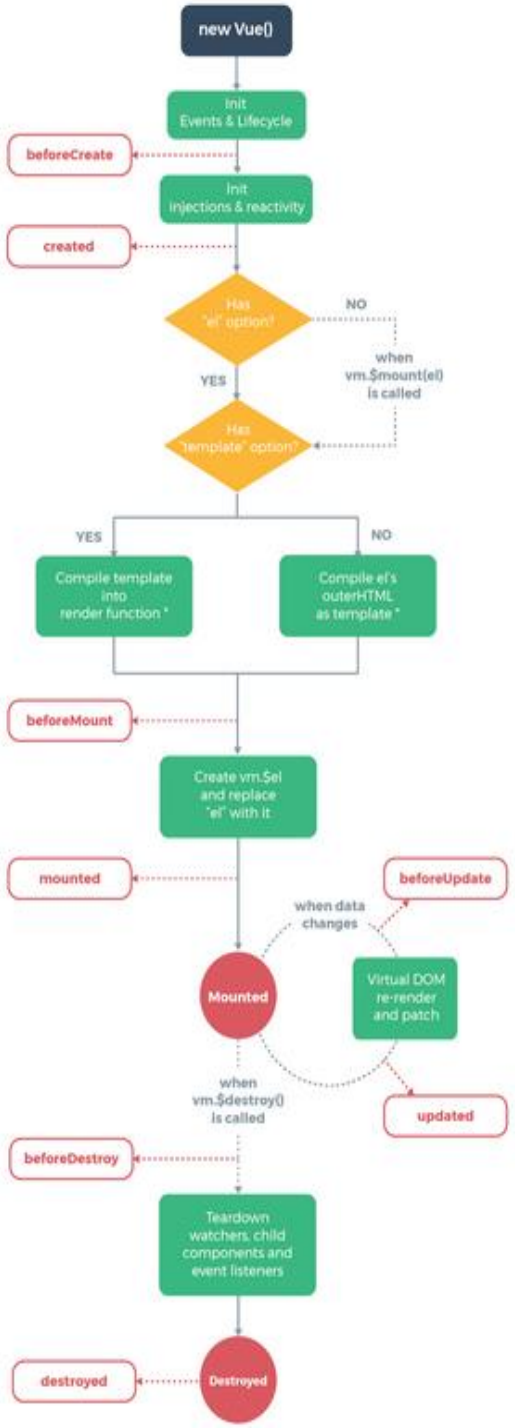
1. 蔬菜
2. 奶酪
3. 随便其他什么人吃的东西

Vue.js基本语义

生命周期图示

下图说明了实例的生命周期。你不需要立马弄明白所有的东西，不过随着你的不断学习和使用，它的参考价值会越来越高。

https://cn.vuejs.org/v2/guide/instance.html#%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F%E5%9B%BE%E7%A4%BA



Vue.js基本语义

文本

数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值：

```HTML
<span>Message: {{ msg }}</span>
```

通过使用 v-once 指令，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。但请留心这会影响到该节点上所有的数据绑定：

```HTML
<span v-once>这个将不会改变: {{ msg }}</span>
```

原始 HTML

双大括号会将数据解释为普通文本，而非 HTML 代码。为了输出真正的 HTML ，你需要使用 `v-html` 指令：

```HTML
<div v-html="rawHtml"></div>
```

修饰符

修饰符 (Modifiers) 是以半角句号 `.` 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。例如， `.prevent` 修饰符告诉 `v-on` 指令对于触发的事件调用 `event.preventDefault()` ：

```HTML
<form v-on:submit.prevent="onSubmit">...</form>
```

在接下来对 `v-on` 和 `v-for` 等功能的探索中，你会看到修饰符的其它例子。

指令

指令 (Directives) 是带有 `v-` 前缀的特殊属性。指令属性的值预期是**单个 JavaScript 表达式** (`v-for` 是例外情况，稍后我们再讨论)。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。回顾我们在介绍中看到的例子：

```HTML
<p v-if="seen">现在你看到我了</p>
```

这里， `v-if` 指令将根据表达式 `seen` 的值的真假来插入/移除 `<p>` 元素。

参数

一些指令能够接收一个“参数”，在指令名称之后以冒号表示。例如， `v-bind` 指令可以用于响应式地更新 HTML 属性：

```HTML
<a v-bind:href="url">...</a>
```

在这里 `href` 是参数，告知 `v-bind` 指令将该元素的 `href` 属性与表达式 `url` 的值绑定。

另一个例子是 `v-on` 指令，它用于监听 DOM 事件：

```HTML
<a v-on:click="doSomething">...</a>
```

在这里参数是监听的事件名。我们也会更详细地讨论事件处理。

v-bind 缩写

```
<!-- 完整语法 -->
<a v-bind:href="url">...</a>

<!-- 缩写 -->
<a :href="url">...</a>
```

v-on 缩写

```
<!-- 完整语法 -->
<a v-on:click="doSomething">...</a>

<!-- 缩写 -->
<a @click="doSomething">...</a>
```

Reference：https://cn.vuejs.org/v2/guide/syntax.html

Vue.js基本语义

基础例子

```
<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: {
    // 计算属性的 getter
    reversedMessage: function () {
      // `this` 指向 vm 实例
      return this.message.split('').reverse().join('')
    }
  }
})
```

结果：

Original message: "Hello"

Computed reversed message: "olleH"

HTML

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

JS

```
// 在组件中
methods: {
  reversedMessage: function () {
    return this.message.split('').reverse().join('')
  }
}
```

我们可以将同一函数定义为一个方法而不是一个计算属性。两种方式的最终结果确实是完全相同的。然而，不同的是**计算属性是基于它们的依赖进行缓存的**。计算属性只有在它的相关依赖发生改变时才会重新求值。这就意味着只要 `message` 还没有发生改变，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。

```
<div id="demo">{{ fullName }}</div>
```

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  },
  watch: {
    firstName: function (val) {
      this.fullName = val + ' ' + this.lastName
    },
    lastName: function (val) {
      this.fullName = this.firstName + ' ' + val
    }
  }
})
```

上面代码是命令式且重复的。将它与计算属性的版本进行比较：

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

计算属性的 setter

计算属性默认只有 getter ，不过在需要时你也可以提供一个 setter ：

```
// ...
computed: {
  fullName: {
    // getter
    get: function () {
      return this.firstName + ' ' + this.lastName
    },
    // setter
    set: function (newValue) {
      var names = newValue.split(' ')
      this.firstName = names[0]
      this.lastName = names[names.length - 1]
    }
  }
}
// ...
```

现在再运行 `vm.fullName = 'John Doe'` 时，setter 会被调用， `vm.firstName` 和 `vm.lastName` 也会相应地被更新。

Reference：https://cn.vuejs.org/v2/guide/computed.html

Vue.js基本语义

侦听器

虽然计算属性在大多数情况下更合适，但有时也需要一个自定义的侦听器。这就是为什么 Vue 通过 `watch` 选项提供了一个更通用的方法，来响应数据的变化。当需要在数据变化时执行异步或开销较大的操作时，这个方式是最有用的。

例如：

```
<div id="watch-example">
  <p>
    Ask a yes/no question:
    <input v-model="question">
  </p>
  <p>{{ answer }}</p>
</div>
```

```
<!-- 因为 AJAX 库和通用工具的生态已经相当丰富，Vue 核心代码没有重复 -->
<!-- 提供这些功能以保持精简。这也可以让你自由选择自己更熟悉的工具。 -->
<script src="https://cdn.jsdelivr.net/npm/axios@0.12.0/dist/axios.min.js"></scrip
<script src="https://cdn.jsdelivr.net/npm/lodash@4.13.1/lodash.min.js"></script>
<script>
var watchExampleVM = new Vue({
  el: '#watch-example',
  data: {
    question: '',
    answer: 'I cannot give you an answer until you ask a question!'
  },
  watch: {
    // 如果 `question` 发生改变，这个函数就会运行
    question: function (newQuestion) {
      this.answer = 'Waiting for you to stop typing...'
      this.getAnswer()
    }
  },
```

```
  methods: {
    // `_.debounce` 是一个通过 Lodash 限制操作频率的函数。
    // 在这个例子中，我们希望限制访问 yesno.wtf/api 的频率
    // AJAX 请求直到用户输入完毕才会发出。想要了解更多关于
    // `_.debounce` 函数（及其近亲 `_.throttle`）的知识，
    // 请参考：https://lodash.com/docs#debounce
    getAnswer: _.debounce(
      function () {
        if (this.question.indexOf('?') === -1) {
          this.answer = 'Questions usually contain a question mark. ;-)'
          return
        }
        this.answer = 'Thinking...'
        var vm = this
        axios.get('https://yesno.wtf/api')
          .then(function (response) {
            vm.answer = _.capitalize(response.data.answer)
          })
          .catch(function (error) {
            vm.answer = 'Error! Could not reach the API. ' + error
          })
      },
      // 这是我们为判定用户停止输入等待的毫秒数
      500
    )
  }
})
</script>
```

结果：

Ask a yes/no question: yes

Questions usually contain a question mark. ;-)

https://cn.vuejs.org/v2/guide/computed.html#%E4%BE%A6%E5%90%AC%E5%99%A8

Vue.js基本语义

Class 与 Style 绑定

操作元素的 class 列表和内联样式是数据绑定的一个常见需求。因为它们都是属性，所以我们可以用 `v-bind` 处理它们：只需要通过表达式计算出字符串结果即可。不过，字符串拼接麻烦且易错。因此，在将 `v-bind` 用于 `class` 和 `style` 时，Vue.js 做了专门的增强。表达式结果的类型除了字符串之外，还可以是对象或数组。

自动添加前缀

当 `v-bind:style` 使用需要添加浏览器引擎前缀的 CSS 属性时，如 `transform`，Vue.js 会自动侦测并添加相应的前缀。

绑定 HTML Class

对象语法

我们可以传给 `v-bind:class` 一个对象，以动态地切换 class：

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

上面的语法表示 `active` 这个 class 存在与否将取决于数据属性 `isActive` 的 truthiness。

你可以在对象中传入更多属性来动态切换多个 class。此外，`v-bind:class` 指令也可以与普通的 class 属性共存。当有如下模板:

```HTML
<div class="static"
     v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
```

和如下 data：

```JS
data: {
  isActive: true,
  hasError: false
}
```

结果渲染为：

```HTML
<div class="static active"></div>
```

绑定内联样式

对象语法

`v-bind:style` 的对象语法十分直观——看着非常像 CSS，但其实是一个 JavaScript 对象。CSS 属性名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用单引号括起来) 来命名：

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

```JS
data: {
  activeColor: 'red',
  fontSize: 30
}
```

直接绑定到一个样式对象通常更好，这会让模板更清晰：

```HTML
<div v-bind:style="styleObject"></div>
```

```JS
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

同样的，对象语法常常结合返回对象的计算属性使用。

Vue.js基本语义

条件渲染

```
<h1 v-if="ok">Yes</h1>
<h1 v-else>No</h1>
```

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

`v-if` vs `v-show`

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，`v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，`v-if` 有更高的切换开销，而 `v-show` 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 `v-show` 较好；如果在运行时条件很少改变，则使用 `v-if` 较好。

`v-show`

另一个用于根据条件展示元素的选项是 `v-show` 指令。用法大致一样：

```
<h1 v-show="ok">Hello!</h1>
```

不同的是带有 `v-show` 的元素始终会被渲染并保留在 DOM 中。`v-show` 只是简单地切换元素的 CSS 属性 `display`。

> 注意，`v-show` 不支持 `<template>` 元素，也不支持 `v-else`。

列表渲染

用 `v-for` 把一个数组对应为一组元素

我们用 `v-for` 指令根据一组数组的选项列表进行渲染。`v-for` 指令需要使用 `item in items` 形式的特殊语法，`items` 是源数据数组并且 `item` 是数组元素迭代的别名。

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

结果：

- Foo
- Bar

变异方法

Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。这些方法

- `push()`
- `pop()`
- `shift()`
- `unshift()`
- `splice()`
- `sort()`
- `reverse()`

你打开控制台，然后用前面例子的 `items` 数组调用变异方法：`example1.items.push({ message: 'Baz' })`。

Vue.js基本语义

表单输入绑定

基础用法

你可以用 `v-model` 指令在表单控件元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇，但 `v-model` 本质上不过是语法糖，它负责监听用户的输入事件以更新数据，并特别处理一些极端的例子。

> `v-model` 会忽略所有表单元素的 `value`、`checked`、`selected` 特性的初始值。因为它会选择 Vue 实例数据来作为具体的值。你应该通过 JavaScript 在组件的 `data` 选项中声明初始值。

修饰符

.lazy

在默认情况下，`v-model` 在 `input` 事件中同步输入框的值与数据 (除了 上述 IME 部分)，但你可以添加一个修饰符 `lazy`，从而转变为在 `change` 事件中同步：

```HTML
<!-- 在 "change" 而不是 "input" 事件中更新 -->
<input v-model.lazy="msg" >
```

.number

如果想自动将用户的输入值转为 Number 类型 (如果原值的转换结果为 NaN 则返回原值)，可以添加一个修饰符 `number` 给 `v-model` 来处理输入值：

```HTML
<input v-model.number="age" type="number">
```

这通常很有用，因为在 `type="number"` 时 HTML 中输入的值也总是会返回字符串类型。

.trim

如果要自动过滤用户输入的首尾空格，可以添加 `trim` 修饰符到 `v-model` 上过滤输入：

```HTML
<input v-model.trim="msg">
```

https://cn.vuejs.org/v2/guide/forms.html

Vue.js基本语义

过滤器

Vue.js 允许你自定义过滤器，可被用作一些常见的文本格式化。过滤器可以用在两个地方：**mustache 插值和 `v-bind` 表达式** (后者从 2.1.0+ 开始支持)。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示：

```HTML
<!-- in mustaches -->
{{ message | capitalize }}

<!-- in v-bind -->
<div v-bind:id="rawId | formatId"></div>
```

过滤器函数总接收表达式的值 (之前的操作链的结果) 作为第一个参数。在这个例子中，`capitalize` 过滤器函数将会收到 `message` 的值作为第一个参数。

```JS
new Vue({
  // ...
  filters: {
    capitalize: function (value) {
      if (!value) return ''
      value = value.toString()
      return value.charAt(0).toUpperCase() + value.slice(1)
    }
  }
})
```

过滤器可以串联：

```HTML
{{ message | filterA | filterB }}
```

在这个例子中，`filterA` 被定义为接收单个参数的过滤器函数，表达式 `message` 的值将作为参数传入到函数中，然后继续调用同样被定义为接收单个参数的过滤器函数 `filterB` ，将 `filterA` 的结果传递到 `filterB` 中。

过滤器是 JavaScript 函数，因此可以接收参数：

```HTML
{{ message | filterA('arg1', arg2) }}
```

这里，`filterA` 被定义为接收三个参数的过滤器函数。其中 `message` 的值作为第一个参数，普通字符串 `'arg1'` 作为第二个参数，表达式 `arg2` 取值后的值作为第三个参数。

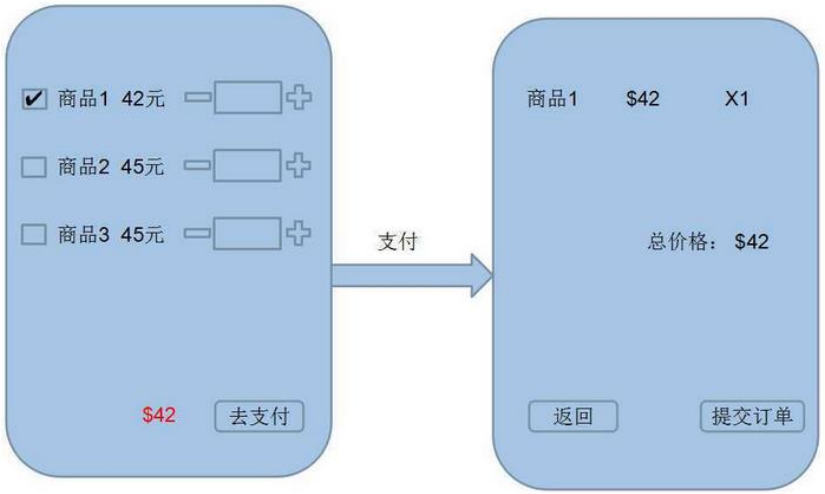
https://cn.vuejs.org/v2/guide/filters.html#main

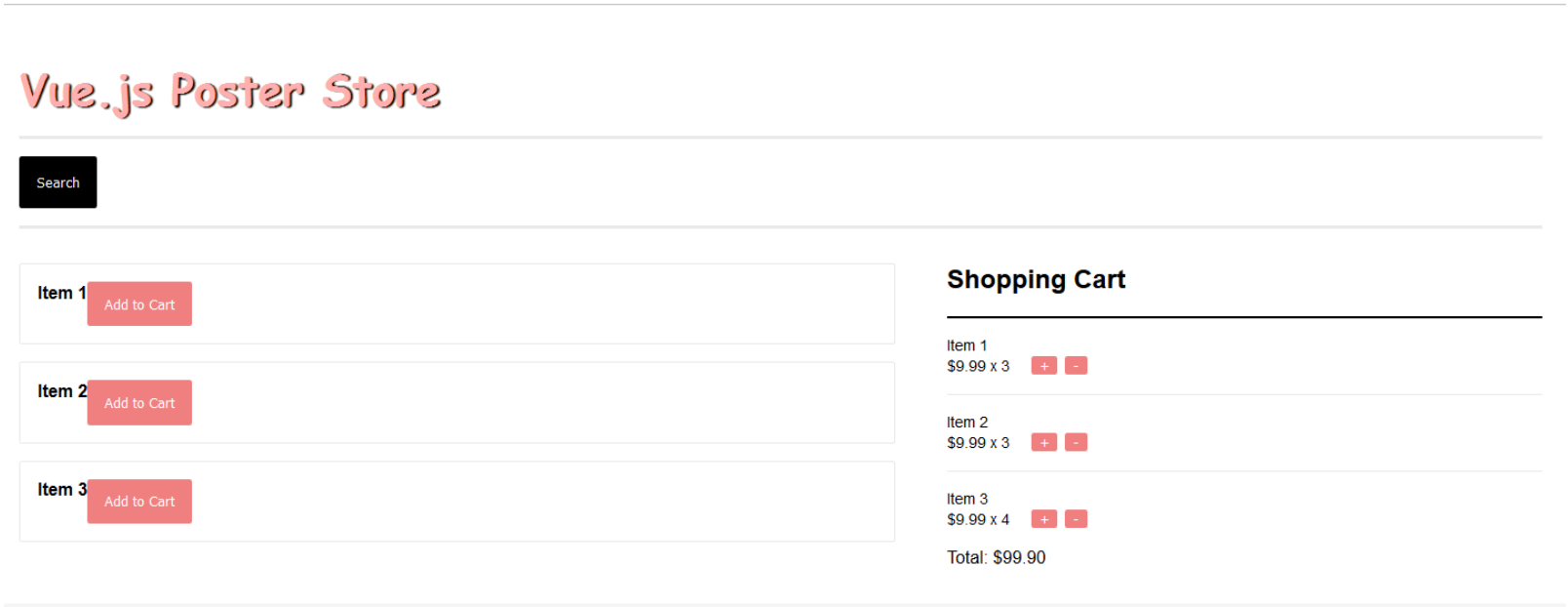
Vue Poster Store

场景2： 购物列表：

本场景主要体现界面交互较多的逻辑处理， 类似的场景包括： 用户资料填写（城市、性别点选）、ERP工单申请（申请类型点选）等；

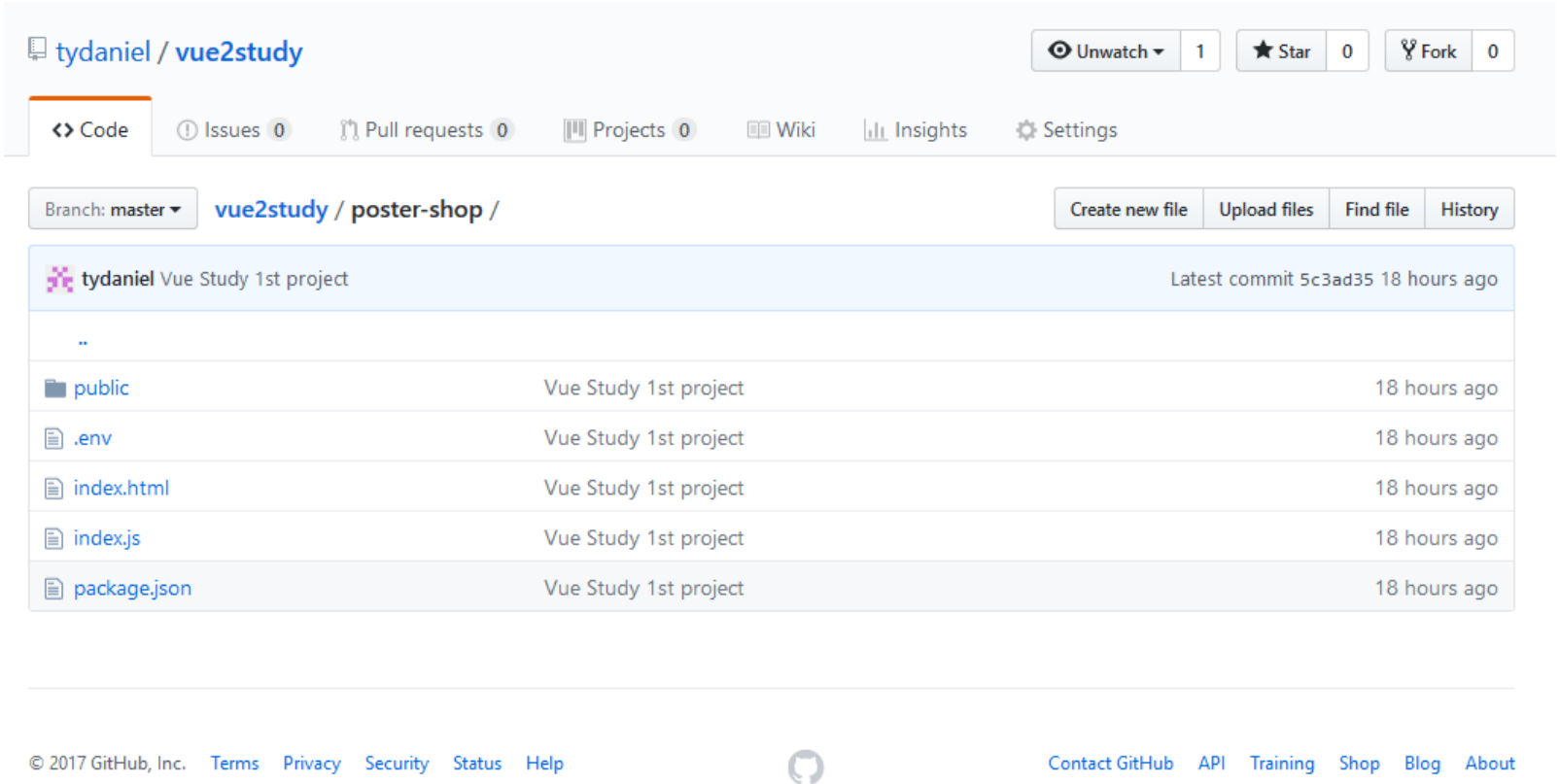
PS: 实际项目中的电商网站不会将购物车、订单结果单页面显示；因为这样不好兼容也不安全， 因而本例只是提供一种多交互的场景：
购物列表设计图如下：





Vue Poster Store

https://github.com/tydaniel/vue2study/tree/master/poster-shop



Running Steps :

1. **git clone**
2. **npm install**
3. **npm start**
4. **Access Web Browser: http://localhost:3000**

script.js

```
var PRICE = 9.99;
new Vue({
    el: '#app',
    data: {
        total: 0,
        items: [
            { id: 1, title: 'Item 1'},
            { id: 2, title: 'Item 2'},
            { id: 3, title: 'Item 3'}
        ],
        cart: []
    },
    methods: {
        // addItem: function(){
        //  console.log("addItem");
        //  this.total += 9.99;
        // }
        addItem: function(index){
        },
        inc: function(item) {
        },
        dec: function(item) {
        },
        onSubmit: function(){
            console.log('Search Function here!');
        }
    },
    filters: {
        currency: function(price) {
            return '$'.concat(price.toFixed(2));
        }
    }
});
```

index.html

```
<!doctype html>
<html lang="en">
<head>
</head>
<body>
<!-- Vue App Enter -->
<div id="app">
    <div class="header">
    </div>
    <div class="main">
        <div class="products">
            <div class="product" v-for="(item, index) in items">
                <h4 class="product-title">{{ item.title }}</h4>
                <button v-on:click="addItem(index)" class="add-to-cart btn">Add to Cart</button>
            </div>
            <!-- <button v-on:click="total += 9.99">Add to Cart</button> -->
            <!-- <button v-on:click="addItem">Add to Cart</button> -->
        </div>
        <div class="cart">
            <h2>Shopping Cart</h2>
            <ul>
                <li class="cart-item" v-for="item in cart">
                    <div class="item-title">{{ item.title }}</div>
                    <!-- <span class="item-qty">{{ item.price }} x {{ item.qty }}</span> -->
                    <span class="item-qty">{{ item.price | currency }} x {{ item.qty }}</span>
                    <button class="btn" v-on:click="inc(item)">+</button>
                    <button class="btn" v-on:click="dec(item)">-</button>
                </li>
            </ul>
            <div v-if="cart.length">
                <div>Total: {{ total | currency }}</div>
            </div>
            <div v-else class="empty-cart">
                No items in the cart.
            </div>
        </div>
    </div>
</div>
```

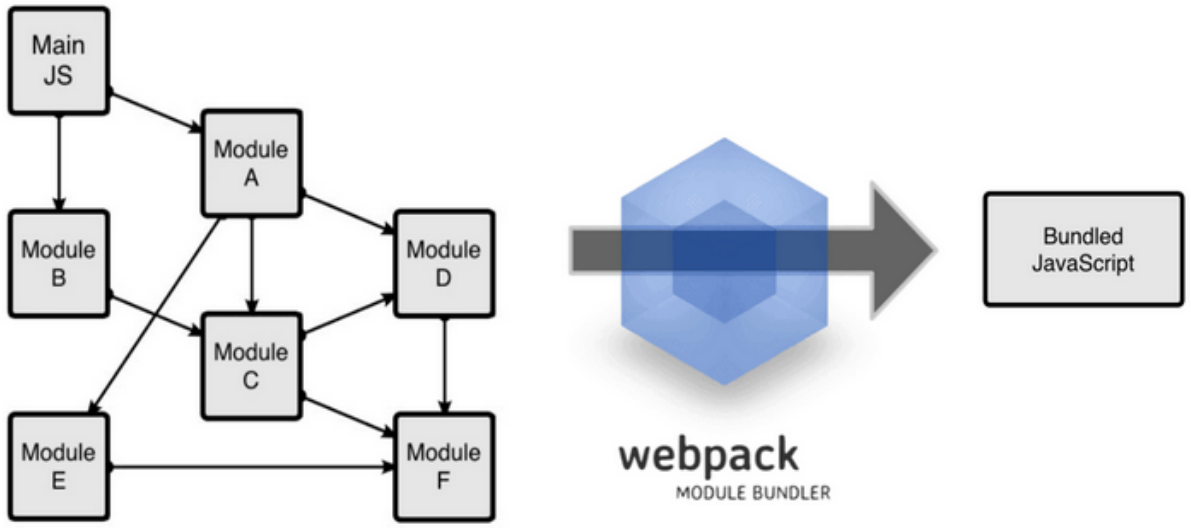
Webpack利器入门

XIAOJUN PAN

2017年11月 技术分享会

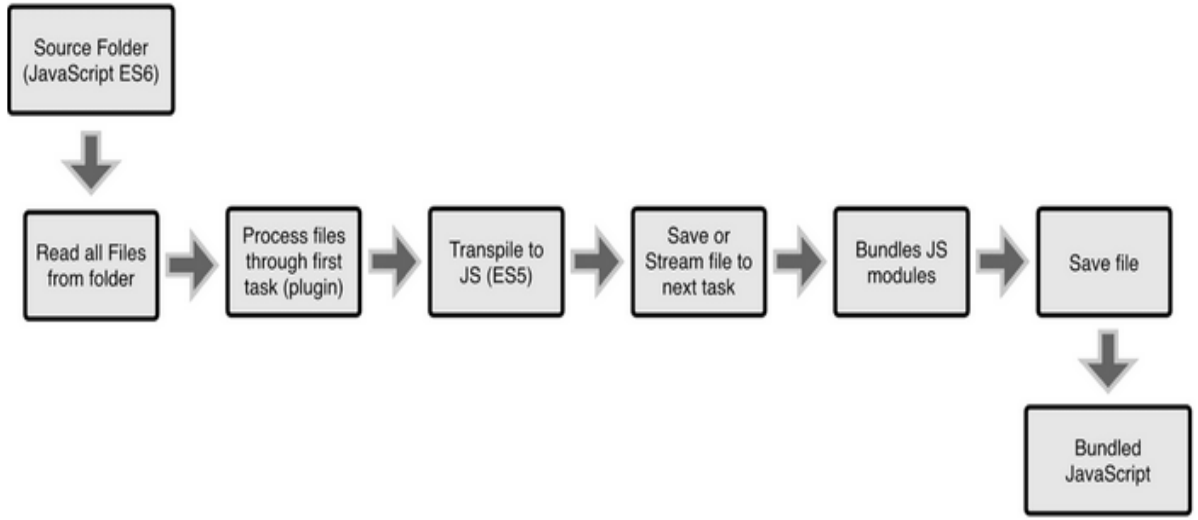
Webpack vs. Gulp

Webpack的工作方式是：把你的项目当做一个整体，通过一个给定的主文件（如：index.js），Webpack将从这个文件开始找到你的项目的所有依赖文件，使用loaders处理它们，最后打包为一个（或多个）浏览器可识别的JavaScript文件。



Webpack工作方式

Grunt和Gulp的工作方式是：在一个配置文件中，指明对某些文件进行类似编译，组合，压缩等任务的具体步骤，工具之后可以自动替你完成这些任务。



Grunt和Gulp的工作流程

Webpack Installation

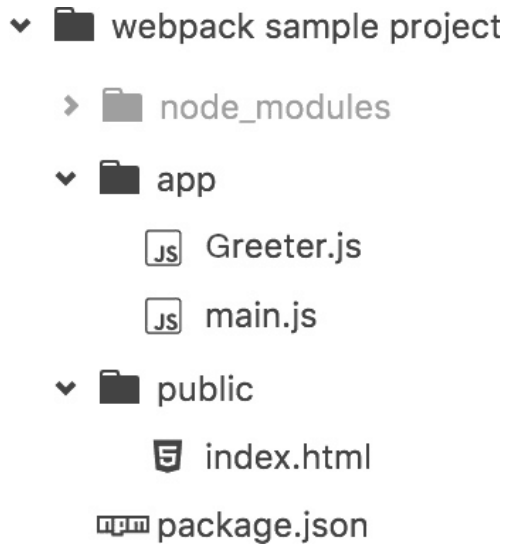
```
npm init
```

```
// 安装Webpack
npm install --save-dev webpack
```

package.json

```
{
  "name": "webtest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "webpack",
    "server": "webpack-dev-server --open",
    "build": "cross-env NODE_ENV=production webpack --config ./webpack.production.config.js --progress"
  },
  "author": "",
  "license": "MIT",
  "devDependencies": {
    "autoprefixer": "^7.1.6",
    "babel-core": "^6.26.0",
    "babel-loader": "^7.1.2",
    "babel-plugin-react-transform": "^3.0.0",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "cross-env": "^5.1.1",
    "css-loader": "^0.28.7",
    "extract-text-webpack-plugin": "^3.0.2",
    "html-webpack-plugin": "^2.30.1",
    "postcss-loader": "^2.0.8",
    "react-transform-hmr": "^1.0.4",
    "sass-loader": "^6.0.6",
    "style-loader": "^0.19.0",
    "webpack": "^3.8.1",
    "webpack-dev-server": "^2.9.3"
  },
  "dependencies": {
    "react": "^16.0.0",
    "react-dom": "^16.0.0"
  }
}
```

Webpack-starter Project Init




```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Webpack Sample Project</title>
  </head>
  <body>
    <div id='root'>
    </div>
    <script src="bundle.js"></script>
  </body>
</html>
```

```
// Greeter.js
module.exports = function() {
  var greet = document.createElement('div');
  greet.textContent = "Hi there and greetings!";
  return greet;
};
```

```
//main.js
const greeter = require('./Greeter.js');
document.querySelector("#root").appendChild(greeter());
```

```
# {extry file}出填写入口文件的路径，本文中就是上述main.js的路径，
# {destination for bundled file}处填写打包文件的存放路径
# 填写路径的时候不用添加{}
webpack {entry file} {destination for bundled file}
```

```
# webpack非全局安装的情况
node_modules/.bin/webpack app/main.js public/bundle.js
```

Webpack.config.js

package.json

```
{
  "name": "webtest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "webpack"
  },
  "author": "",
  "license": "MIT",
  "devDependencies": {...
  },
  "dependencies": {...
  }
}
```

webpack.config.js

```
module.exports = {
  entry:  __dirname + "/app/main.js",//已多次提及的唯一入口文件
  output: {
    path: __dirname + "/public",//打包后的文件存放的地方
    filename: "bundle.js"//打包后输出文件的文件名
  }
}
```

```
~/Git/test is 📦 v1.0.0 via  via 🐹 v1.7
➜ npm start

> test@1.0.0 start /Users/zhangwang/Git/test
> webpack

Hash: e617000114aae964fd73
Version: webpack 3.5.3
Time: 76ms
    Asset     Size  Chunks             Chunk Names
bundle.js  2.79 kB       0  [emitted]  main
   [0] ./app/main.js 96 bytes {0} [built]
   [1] ./app/Greeter.js 148 bytes {0} [built]
```

Webpack-dev-server

```
npm install --save-dev webpack-dev-server
```

devserver作为webpack配置选项中的一项，以下是它的一些配置选项，更多配置可参考这里

devserver的配置选项	功能描述
contentBase	默认webpack-dev-server会为根文件夹提供本地服务器，如果想为另外一个目录下的文件提供本地服务器，应该在这里设置其所在目录（本例设置到"public"目录）
port	设置默认监听端口，如果省略，默认为"8080"
inline	设置为 `true` ，当源文件改变时会自动刷新页面
historyApiFallback	在开发单页应用时非常有用，它依赖于HTML5 history API，如果设置为 `true` ，所有的跳转将指向index.html

webpack.config.js

```
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

    entry: __dirname + "/app/main.js",
    output: {
        path: __dirname + "/build",
        filename: "bundle.js"
    },
    /**使用eval打包源文件模块，在同一个文件中生成干净的完
    devtool: 'eval-source-map',//使用eval-source-map
    devServer: {
        contentBase: "./public",//本地服务器所加载的页面
        historyApiFallback: true,//不跳转
        port: "8081",//指定端口
        inline: true//实时刷新
    },
    module: {...
    },
    plugins: [...
    ]
}
```

```
D:\pxj\Web\vue2study\webpack-starter>npm run server

> webtest@1.0.0 server D:\pxj\Web\vue2study\webpack-starter
> webpack-dev-server --open

Project is running at http://localhost:8081/
webpack output is served from /
Content not from webpack is served from ./public
404s will fallback to /index.html
webpack: wait until bundle finished: /
Hash: 0158e6aebe1f9a6f046b
Version: webpack 3.8.1
Time: 5140ms
     Asset       Size  Chunks                    Chunk Names
 bundle.js    3.57 MB       0  [emitted]  [big]  main
index.html  223 bytes          [emitted]
  [14] ./node_modules/react/index.js 190 bytes {0} [built]
  [70] multi (webpack)-dev-server/client?http://localhost:8081 ./app/main.js 40 bytes {0} [built]
  [71] (webpack)-dev-server/client?http://localhost:8081 7.95 kB {0} [built]
  [72] ./node_modules/url/url.js 23.3 kB {0} [built]
  [78] ./node_modules/strip-ansi/index.js 161 bytes {0} [built]
  [80] ./node_modules/loglevel/lib/loglevel.js 7.74 kB {0} [built]
  [81] (webpack)-dev-server/client/socket.js 1.05 kB {0} [built]
  [83] (webpack)-dev-server/client/overlay.js 3.73 kB {0} [built]
  [88] (webpack)/hot nonrecursive ^\.\/log$ 170 bytes {0} [built]
  [90] (webpack)/hot/emitter.js 75 bytes {0} [built]
  [92] ./app/main.js 587 bytes {0} [built]
  [95] ./node_modules/react-dom/index.js 1.36 kB {0} [built]
  [99] ./node_modules/react-dom/cjs/react-dom.development.js 607 kB {0} [built]
 [109] ./app/Greeter.js 3.2 kB {0} [built]
 [219] ./app/main.css 1.17 kB {0} [built]
    + 205 hidden modules
Child html-webpack-plugin for "index.html":
     1 asset
       [0] ./node_modules/html-webpack-plugin/lib/loader.js!./app/index.tmpl.html 547 bytes {0} [built]
       [1] ./node_modules/lodash/lodash.js 540 kB {0} [built]
       [2] (webpack)/buildin/global.js 488 bytes {0} [built]
       [3] (webpack)/buildin/module.js 495 bytes {0} [built]
webpack: Compiled successfully.
```

生成Source Maps（使调试更容易）

生成Source Maps（使调试更容易）

开发总是离不开调试，方便的调试能极大的提高开发效率，不过有时候通过打包后的文件，你是不容易找到出错了的地方，对应的你写的代码的位置的，`Source Maps`就是来帮我们解决这个问题的。

通过简单的配置，`webpack`就可以在打包时为我们生成的`source maps`，这为我们提供了一种对应编译文件和源文件的方法，使得编译后的代码可读性更高，也更容易调试。

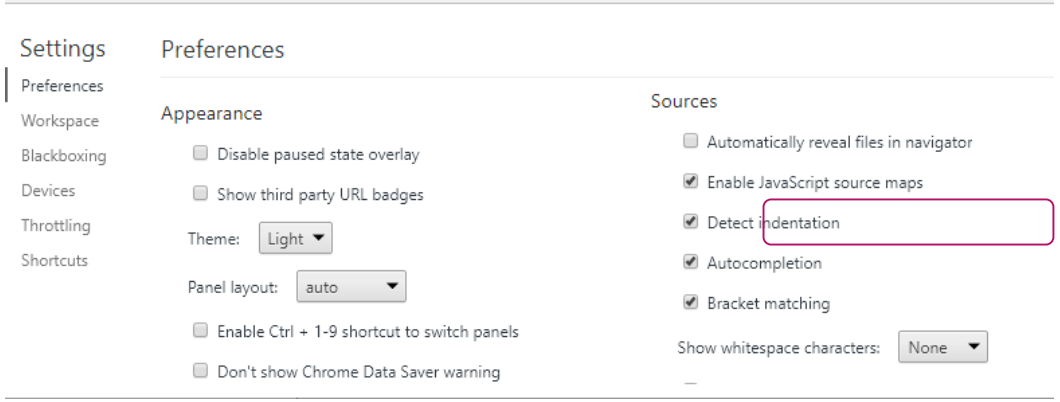
在`webpack`的配置文件中配置`source maps`，需要配置`devtool`，它有以下四种不同的配置选项，各具优缺点，描述如下：

devtool选项	配置结果
`source-map`	在一个单独的文件中产生一个完整且功能完全的文件。这个文件具有最好的`source map`，但是它会减慢打包速度；
`cheap-module-source-map`	在一个单独的文件中生成一个不带列映射的`map`，不带列映射提高了打包速度，但是也使得浏览器开发者工具只能对应到具体的行，不能对应到具体的列（符号），会对调试造成不便；
`eval-source-map`	使用`eval`打包源文件模块，在同一个文件中生成干净的完整的`source map`。这个选项可以在不影响构建速度的前提下生成完整的`sourcemap`，但是对打包后输出的JS文件的执行具有性能和安全的隐患。在开发阶段这是一个非常好的选项，在生产阶段则一定不要启用这个选项；
`cheap-module-eval-source-map`	这是在打包文件时最快的生成`source map`的方法，生成的`Source Map`会和打包后的`JavaScript`文件同行显示，没有列映射，和`eval-source-map`选项具有相似的缺点；

正如上表所述，上述选项由上到下打包速度越来越快，不过同时也具有越来越多的负面作用，较快的打包速度的后果就是对打包后的文件的的执行有一定影响。

对小到中型的项目中，`eval-source-map`是一个很好的选项，

生成Source Maps（使调试更容易）




```
Uncaught ReferenceError: Name is not defined
    at o (hello.js:4)
    at hello.js:8
```

注意，Chrome的报错信息没有列号，因此4为错误的行号。Chrome不仅可以通过Source Map还原真实的出错位置，还可以根据Source Map的sourcesContent还原出错的源代码。点击出错位置，即可跳转到源码，这样Debug将非常方便。

hello.min.js | hello.js ×

```
function sayHello()
{
    var name = "Fundebug";
    var greeting = "Hello, " + Name;
    console.log(greeting);
}

sayHello();
```

Loaders

Loaders

鼎鼎大名的Loaders登场了！

`Loaders`是`webpack`提供的最激动人心的功能之一了。通过使用不同的`loader`，`webpack`有能力调用外部的脚本或工具，实现对不同格式的文件的处理，比如说分析转换scss为css，或者把下一代的JS文件（ES6，ES7)转换为现代浏览器兼容的JS文件，对React的开发而言，合适的Loaders可以把React的中用到的JSX文件转换为JS文件。

Loaders需要单独安装并且需要在`webpack.config.js`中的`modules`关键字下进行配置，Loaders的配置包括以下几方面：

- `test`：一个用以匹配loaders所处理文件的拓展名的正则表达式（必须）
- `loader`：loader的名称（必须）
- `include/exclude`:手动添加必须处理的文件（文件夹）或屏蔽不需要处理的文件（文件夹）（可选）；
- `query`：为loaders提供额外的设置选项（可选）

```
module: {

    loaders: [
        // 使用vue-loader 加载 .vue 结尾的文件
        {
            test: /\.vue$/,
            loader: 'vue'
        },
        {
            test: /\.js$/,
            loader: 'babel?presets=es2015',
            exclude: /node_modules/
        }
    ]
},
```

Babel

Babel

Babel其实是一个编译JavaScript的平台，它的强大之处表现在可以通过编译帮你达到以下目的：

- 使用下一代的JavaScript代码（ES6，ES7...），即使这些标准目前并未被当前的浏览器完全的支持；
- 使用基于JavaScript进行了拓展的语言，比如React的JSX；

Babel的安装与配置

Babel其实是几个模块化的包，其核心功能位于称为 `babel-core` 的npm包中，webpack可以把其不同的包整合在一起使用，对于每一个你需要的功能或拓展，你都需要安装单独的包（用得最多的是解析Es6的 `babel-preset-es2015` 包和解析JSX的 `babel-preset-react` 包）。

我们先来一次性安装这些依赖包

```
// npm一次性安装多个依赖模块，模块之间用空格隔开
npm install --save-dev babel-core babel-loader babel-preset-es2015 babel-preset-react
```

```
19      module: {
20          rules: [
21              {
22                  test: /(\.jsx|\.js)$/,
23                  use: {
24                      loader: "babel-loader",
25                      options: {
26                          presets: [
27                              "es2015", "react"
28                          ]
29                      }
30                  },
31                  exclude: /node_modules/
32              },
33              {
34                  test: /\.css$/,
35                  use: [
36                      {
37                          loader: "style-loader"
38                      }, {
39                          loader: "css-loader",
40                          options: {
41                              modules: true
42                          }
43                      }, {
44                          loader: "postcss-loader"
45                      }
46                  ]
47              }
48          ]
49      },
```

.babelrc

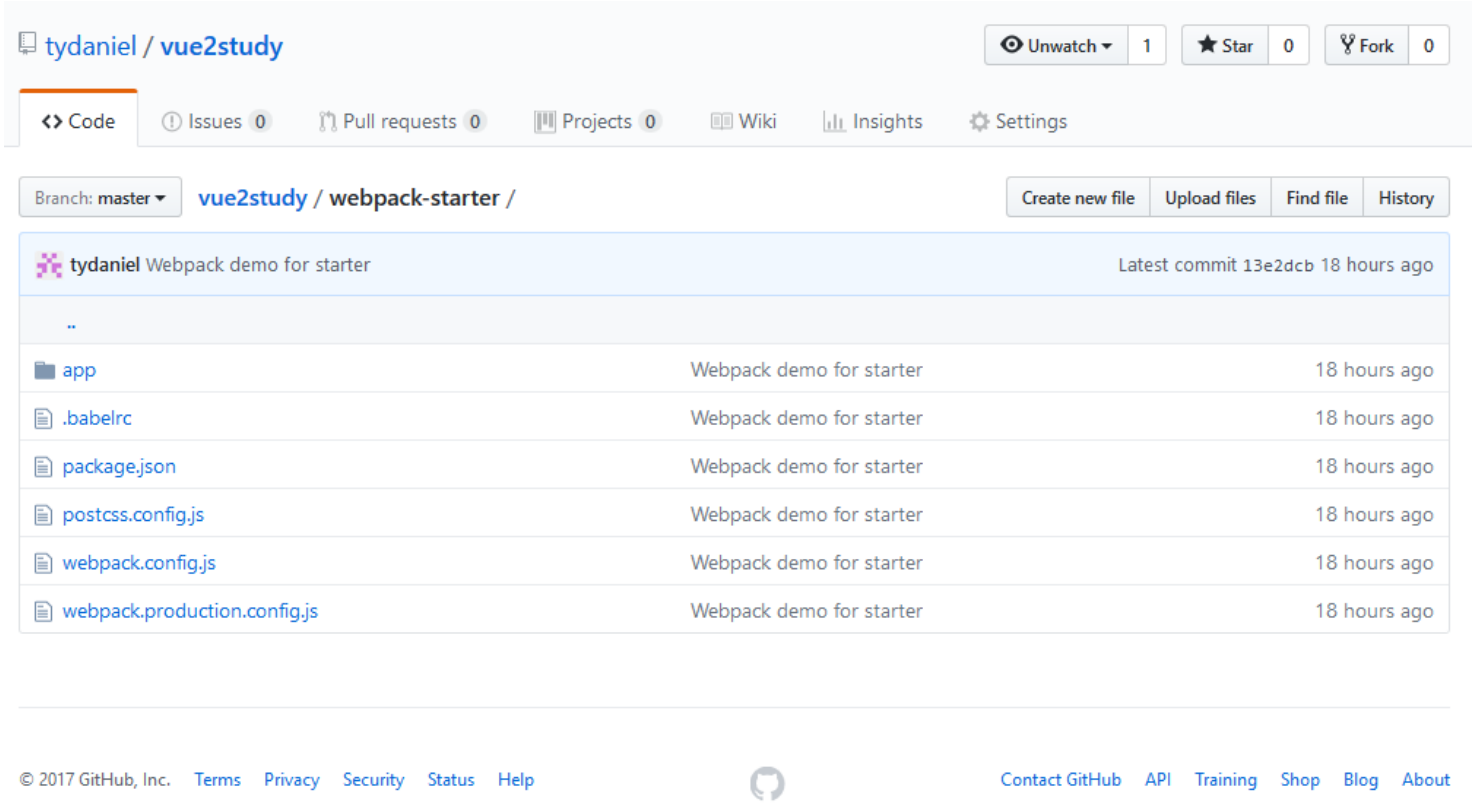
```
1   //.babelrc
2   {
3     "presets": ["react", "es2015"],
4     "env": {
5       "development": {
6         plugins: [[
7           "react-transform",
8           {"transforms": [{
9             "transform": "react-transform-hmr",
10            "imports": ["react"],
11            "locals": ["module"]
12            }]
13          }
14        ]]
15      }
16    }
17  }
```

Webpack Starter

http://www.jianshu.com/p/42e11515c10f

Webpack Starter

https://github.com/tydaniel/vue2study/tree/master/webpack-starter



Running Steps :

1. **git clone**
2. **npm install**
3. **npm start**
4. **npm run build**

Thanks

感謝いたします！

谢谢侬！