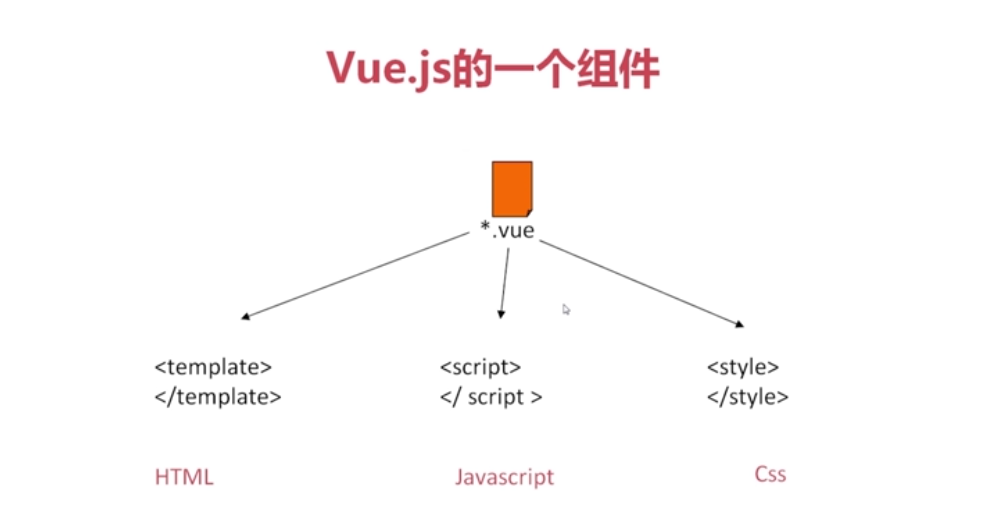
VUE js

1. 前端模块化
2. ES6初步了解
3. 轻量级的MVVM框架

Vue js是相当于进行了一个整合，放到同一个VUE文件中。

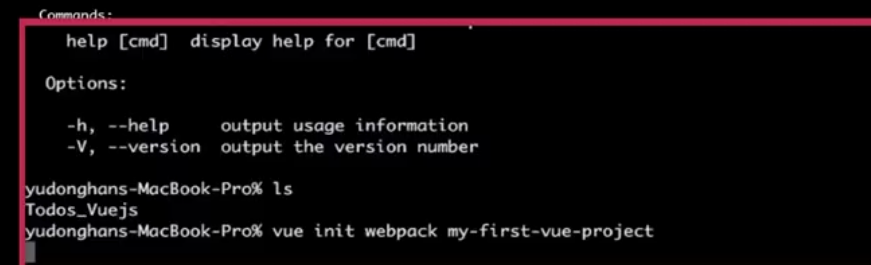


快速开发大型单页应用。

Npm是node的包管理工具。

Npm建议使用淘宝镜像，要不然可能会慢。

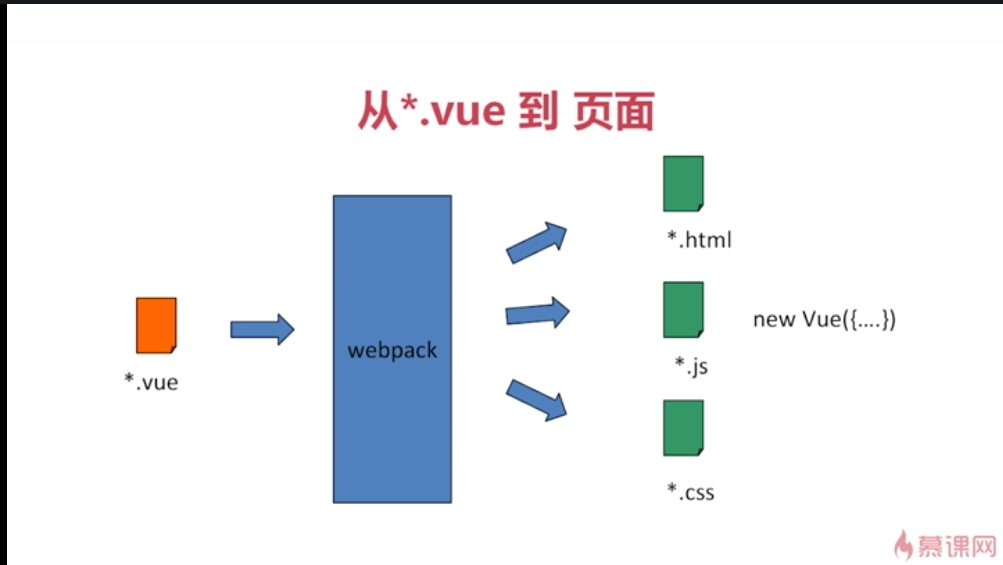
box-sizing:border-box; 是CSS3新增属性，了解这个属性，我们先从块级元素的盒子大小说起，通常一个块级元素实际所占宽高度=外边距（margin）+ 边界宽度(border-width) + 内边距（padding）+ 高度（height） /  宽度（width）  
如果设置了border-box，实际所占宽高度 = 设置的高度（height）/ 设置的宽度（width）+ 外边距（margin）



在当前目录下生成 webpack打包的模板---

下载依赖，任何一个项目都不是独立存在。安装所有依赖。用到了babel—对ES6的转换，以及CSS-loader等等来解析CSS之类。。。。。

Vue 有热更新功能。一旦文件改变 页面立即改变，不用刷新。



Vue脚手架工具。

SPA—single page application

学习VUE

刚开始学习vue时，不建议从vue-cli开始，先外部引入来学习vue基本语法。

<script src="https://unpkg.com/vue"></script>

|  |
| --- |
| <div id="app">  {{ message }}  </div> |

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

现在数据和 DOM 已经被关联在一起，所有的数据和 DOM 都是**响应式**的。我们如何对这一切清晰领会？只需打开你的浏览器的 JavaScript 控制台（现在，就在当前页面打开），然后设置 app.message 的值，你将看到上面的示例所渲染的 DOM 元素会相应地更新。

组件系统是VUE的另一个重要概念，因为它是一种抽象，可以让我们使用小型、自成体系和通常可复用的组件，把这些组合来构建大型应用程序。

每个 Vue.js 应用都是通过构造函数 Vue 创建一个 Vue 的根实例来启动的：

var vm = new Vue({

// 选项

})

每个 Vue 实例都会代理其 data 对象里所有的属性：

var data = { a: 1 }

var vm = new Vue({

data: data

})

vm.a === data.a // -> true

// 设置属性也会影响到原始数据

vm.a = 2

data.a // -> 2

// ... 反之亦然

data.a = 3

vm.a // -> 3

除了 data 属性， Vue 实例暴露了一些有用的实例属性与方法。这些属性与方法都有前缀 $，以便与代理的 data 属性区分。例如：

var data = { a: 1 }

var vm = new Vue({

el: '#example',

data: data

})

vm.$data === data // -> true

vm.$el === document.getElementById('example') // -> true

// $watch 是一个实例方法

vm.$watch('a', function (newVal, oldVal) {

// 这个回调将在 `vm.a` 改变后调用

})

//v-if="seen===true"这里是三个等号

v-show 是否显示该元素。

{{5+5}}直接进行计算 结果显示10

<li v-for="n in 10">

{{ n }}

</li>

用来迭代整数

计算属性

例如 {{ message.split('').reverse().join('') }}

<p>计算后反转字符串: {{ reversedMessage }}</p>

可以写在vue中的compute属性中

computed: { // 计算属性的 getter

reversedMessage: function () { // `this` 指向 vm 实例 return this.message.split('').reverse().join('') }

}

我们可以使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。

可以说使用 computed 性能会更好，但是如果你不希望缓存，你可以使用 methods 属性。

<p>使用方法后反转字符串: {{ reversedMessage2() }}</p>

<http://www.runoob.com/try/try.php?filename=vue2-str-reverse4>

<http://www.runoob.com/try/try.php?filename=vue2-str-reverse3>

对比以下这两个函数 回头自己实现一下

<http://www.runoob.com/vue2/vue-custom-directive.html>

**继续学习vue组件**

**结合文档**

**https://vuefe.cn/v2/guide/#声明式渲染**

**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：

|  |
| --- |
| // 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：

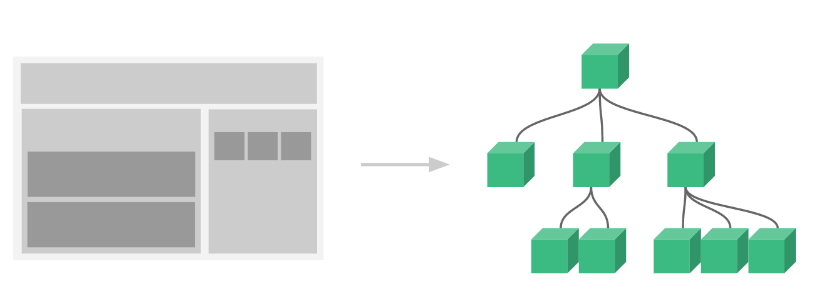
|  |
| --- |
| <ol>  <!-- 创建一个 todo-item 组件的实例 -->  <todo-item></todo-item>  </ol> |

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**](https://cn.vuejs.org/v2/guide/components.html#Props)：

组件（Component）是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



注册一个全局组件语法格式如下：

Vue.component(tagName, options)

tagName 为组件名，options 为配置选项。注册后，我们可以使用以下方式来调用组件：

<tagName></tagName>

**局部组件：组件只在局部实例中去使用。**

props 是父组件用来传递数据的一个自定义属性。

父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "props"：

注意: prop 是单向绑定的：当父组件的属性变化时，将传导给子组件，但是不会反过来。

父组件是使用 props 传递数据给子组件，但如果子组件要把数据传递回去，就需要使用自定义事件！

我们可以使用 v-on 绑定自定义事件, 每个 Vue 实例都实现了事件接口(Events interface)，即：

* 使用 $on(eventName) 监听事件
* 使用 $emit(eventName) 触发事件

另外，父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件。.HTML