# Vue.js

## 介绍

#### Vue.js 是什么

Vue (读音 /vjuː/，类似于 **view**) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与[现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)以及各种[支持类库](https://github.com/vuejs/awesome-vue#libraries--plugins)结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。<信息来源：vue 官网>

#### 作者

尤雨溪毕业于上海复旦附中，在美国完成大学学业，本科毕业于上 Colgate University，后在 Parsons 设计学院获得 Design & Technology 艺术硕士学位，任职于纽约 Google Creative Lab。<信息来源：百度百科>

#### 作用

Vue.js 框架的作用主要注重动态的构建用户界面，前端工程化和模块化开发。

#### Vue 与其他框架的对比

| **框架** | **设计模式** | **数据绑定** | **灵活度** | **文件模式** | **复杂性** | **学习曲线** | **生态** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Vue | MVVM | 双向 | 灵活 | 单文件 | 小 | 缓 | 完善 |
| React | MVC | 单向 | 较灵活 | all in js | 大 | 陡 | 丰富 |
| Angular | MVC | 双向 | 固定 | 多文件 | 较大 | 较陡（Typescript） | 独立 |

**更多对比细节：vue 官网：**[**https://cn.vuejs.org/v2/guide/comparison.html**](https://cn.vuejs.org/v2/guide/comparison.html)

Vue 是一个推陈出新的前端框架，吸收了很多前端框架的优点。例如，Vue 借鉴了 React 的组件化和虚拟 DOM ，借鉴了 Angular 的模块化和数据绑定。因此，我们以 Vue.js 作为入手，以后深入学习其他框架，你会发现他们的共通之处，更好地高效地学习。

选择 Vue.js 的更多原因是，就框架的 API 而言，对比之下，Vue 更加轻便简洁。Vue 自身拥有开箱即用的生态开发包（Vuex,Vue-Router）等，复杂性低、学习成本低，是一门比较好入门的前端框架。

如果你刚开始学习前端开发，将框架作为你的第一步可能不是最好的选择，推荐进入实验楼前端基础知识课程。假入你已掌握了关于 HTML、CSS 和 JavaScript 的中级知识，那让我们进入 Vue 的学习吧!

## 安装

#### <script> 标签引入

标签引入模式，和平常引入JQuery一样，直接下载并用 <script> 标签引入

下载地址：<https://vuejs.org/js/vue.js>

<script src="js/vue.min.js"></script>

#### CDN引入

不用下载到本地，即引即用，学习推荐使用这种模式：

推荐 2 个较稳定的 cdn,以下任选其一

<script src="https://cdn.bootcss.com/vue/2.6.6/vue.min.js"></script>

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

由于非会员用户无法访问外网，所以后续的代码中，统一使用引用链接为https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js。

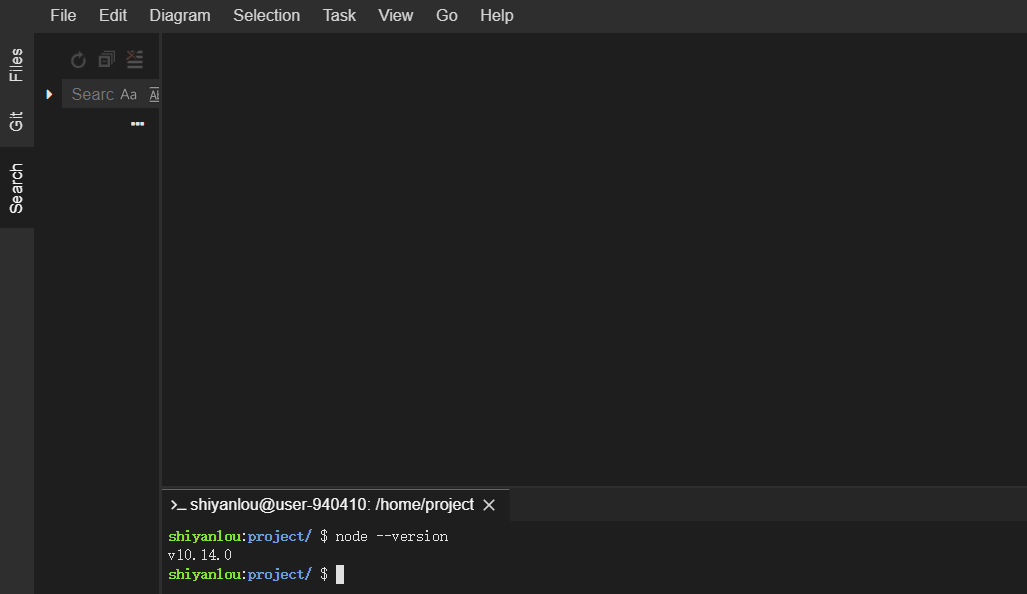
#### 脚手架工具

由于安装需要使用到外网，非会员用户不能访问外网。所以实验楼提供的在线环境已经安装好了所需工具，可以直接使用。步骤 1、2、3 可以直接略过。

1、vue-cli是基于npm的，所以应该先安装node环境，通过node官网：<http://nodejs.cn/> 下载系统对应的node安装程序。



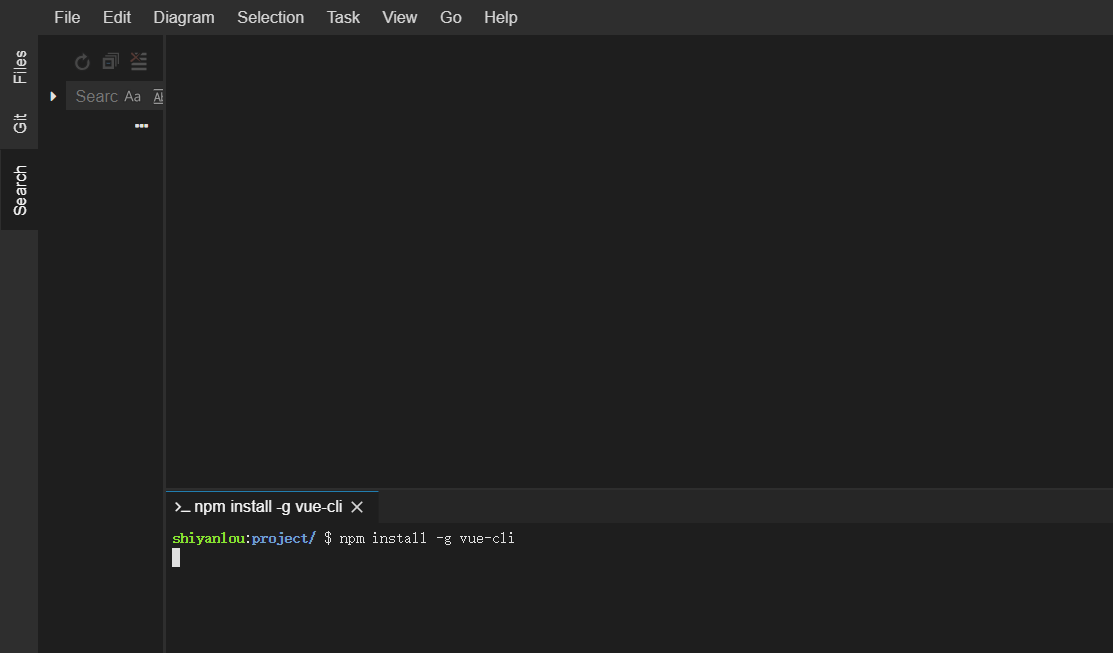
**注意：** NPM 是随同 Node.js 一起安装的包管理工具。因此当我们安装好 Node.js 的时候，也安装好了 NPM。由于实验楼的环境下已经安装好了 Node.js，大家可以直接使用，实验楼环境中跳过安装 Node 步骤，直接进行第二步。



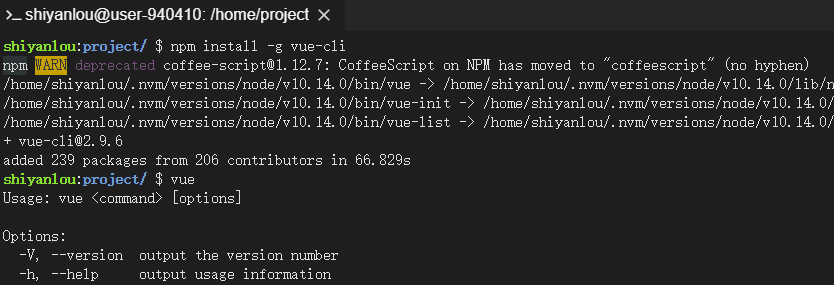
2、node安装完毕使用，npm包管理工具全局安装vue-cli

npm install -g vue-cli



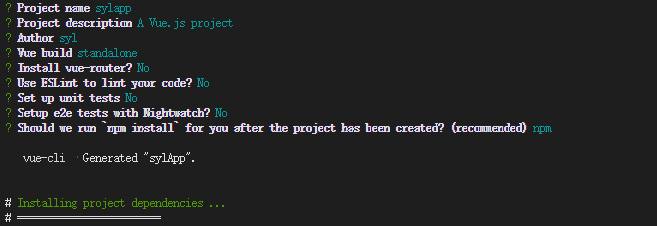


3、命令行输入vue，出现 Usage 表示安装成功。



4、使用指令生成一个 vue 应用

vue init webpack sylApp # sylApp 这里是项目名AppName



参考上图在命令行中输入对应的内容。

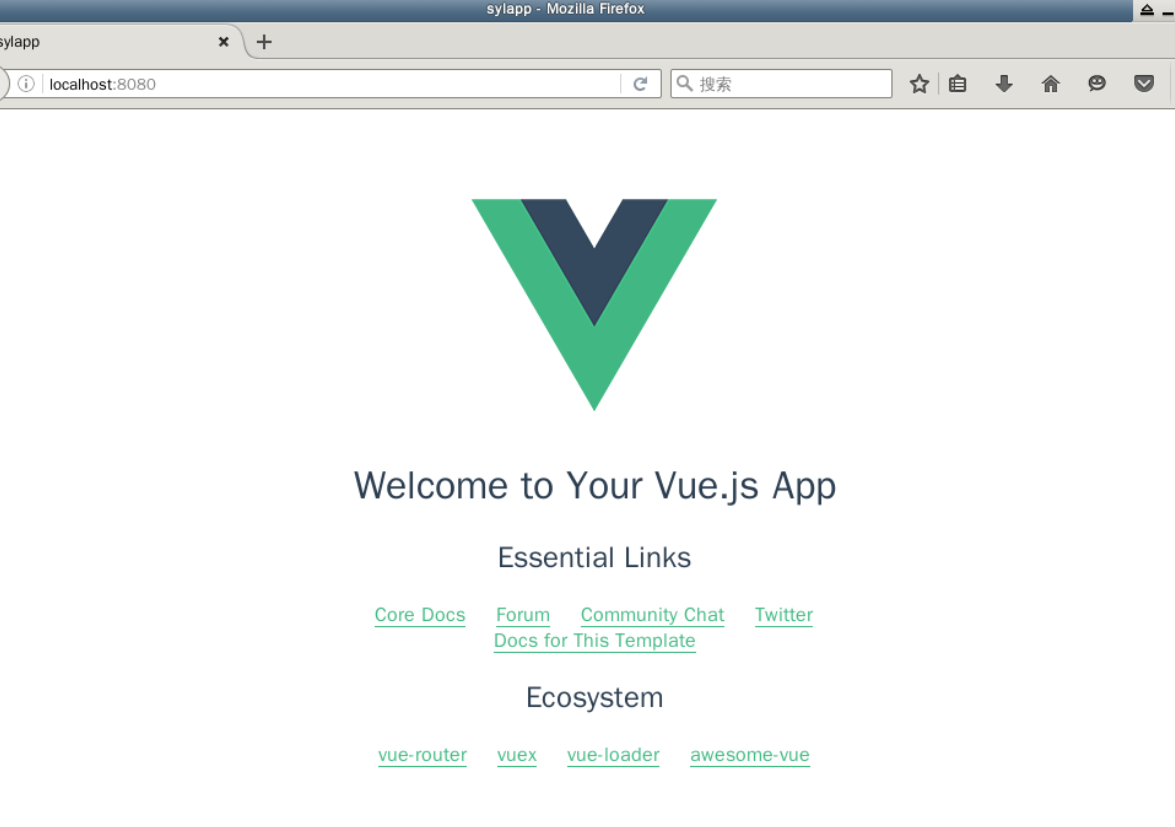
5、命令行进入刚刚创建的 app 的文件夹内，输入npm run start将 vue 应用运行起来

cd sylApp

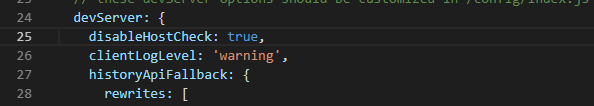
npm run start



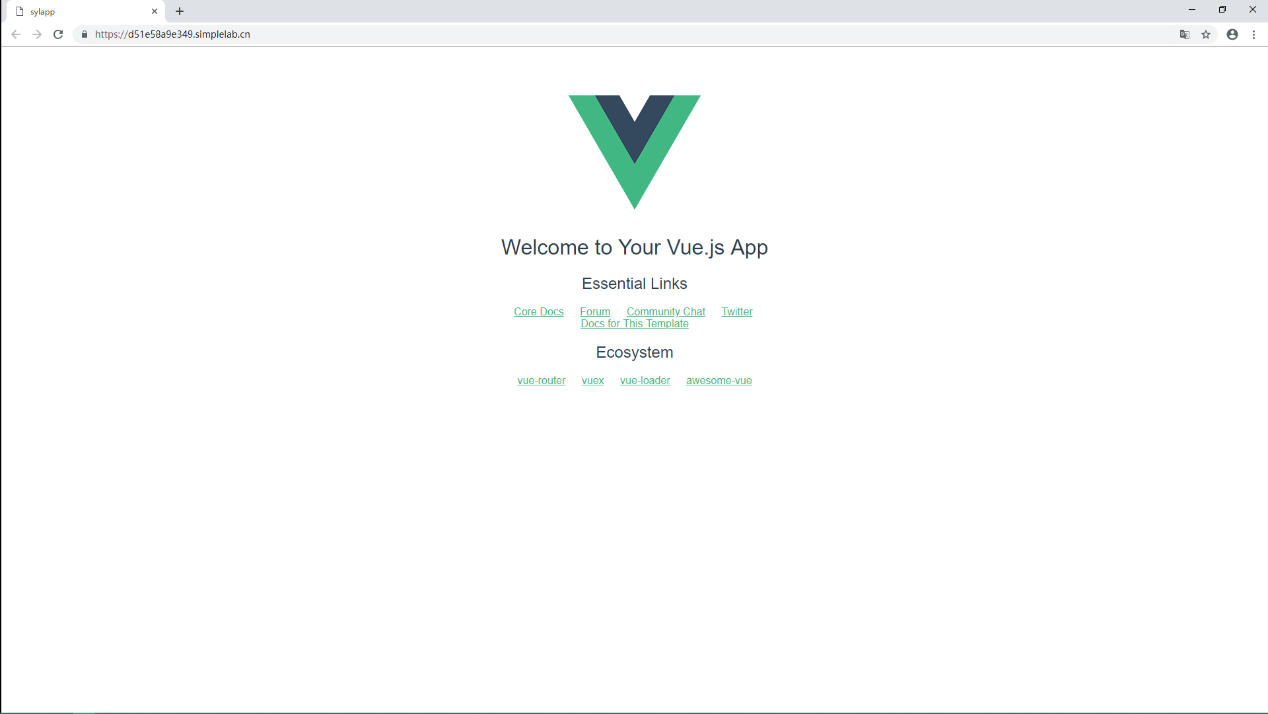
6、成功执行命令行后，我们打开浏览器输入[http://localhost:8080，即可看到一个](http://localhost:8080%EF%BC%8C%E5%8D%B3%E5%8F%AF%E7%9C%8B%E5%88%B0%E4%B8%80%E4%B8%AA/) vue 应用。



这是正常的操作流程，由于实验楼环境特殊，使用该方法需要额外的配置，我们需要修改一些文件：打开 sylApp/build/webpack.dev.conf.js,修改 devServer选项下的 host 配置代码为 host: HOST || '0.0.0.0', 在 devServer 中添加 disableHostCheck:true,,修改成下图，所示



接着我们重复第五步的步骤，构建成功后，我们打开工具中的 Web 服务：



#### vue 开发者工具安装

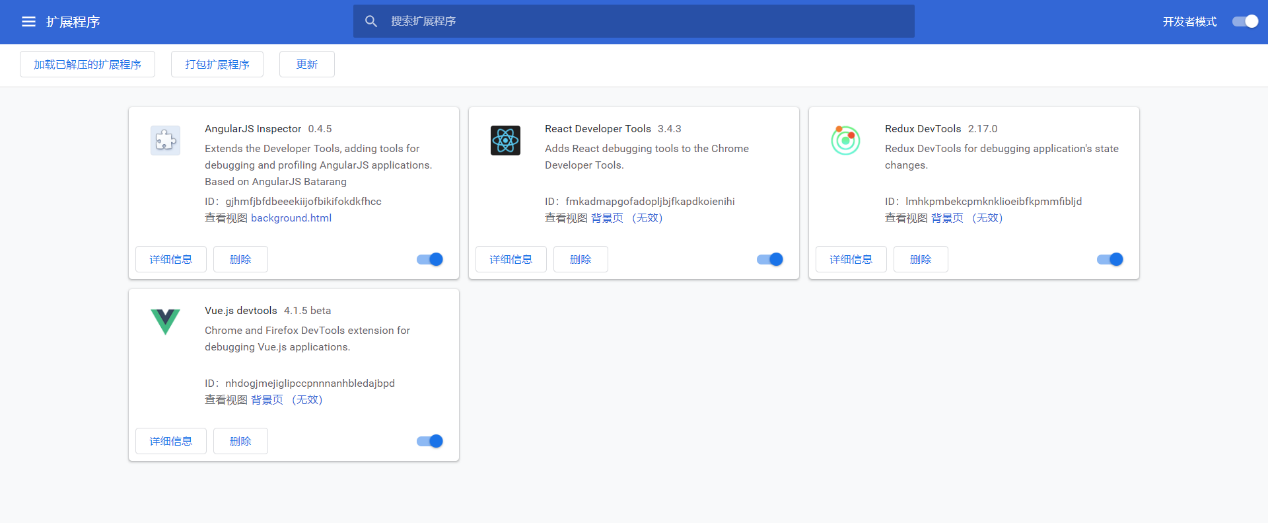
**注意：** 由于实验楼环境使用的是 Preview 或 Mini Browser 是一个轻量级的浏览器，不支持插件安装，如需安装，请在自己的电脑上对应安装。

在使用 Vue 时，我们推荐在你的浏览器上安装 [Vue Devtools](https://github.com/vuejs/vue-devtools#vue-devtools)。它允许你在一个更友好的界面中审查和调试 Vue 应用。

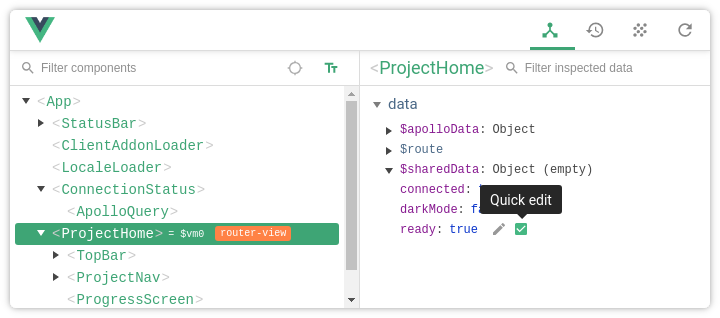
1、下载对应浏览器的 Vue Devtools

* [Get the Chrome Extension](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd) / ([beta channel](https://chrome.google.com/webstore/detail/vuejs-devtools/ljjemllljcmogpfapbkkighbhhppjdbg))
* [Get the Firefox Addon](https://addons.mozilla.org/en-US/firefox/addon/vue-js-devtools/) / ([beta channel](https://github.com/vuejs/vue-devtools/releases))
* [Get standalone Electron app (works with any environment!)](https://github.com/vuejs/vue-devtools/blob/master/shells/electron/README.md)

2、打开浏览器，打开设置>开发工具>扩展程序，将下载好的 Vue Devtools 拖到界面中，即可完成安装(谷歌浏览器为例)。



3、你就可以在浏览器中轻松调试你的 vue 应用



#### 更多安装方式

其他安装方式请参考：[官网教程](https://cn.vuejs.org/v2/guide/installation.html)

## 创建第一个实例

每个 Vue 应用都是通过用 Vue 函数创建一个新的 **Vue 实例** 开始的：

var app = new Vue({

// 选项

});

由于实验楼使用是 WebIDE 的在线环境，所以有不熟悉对同学请阅读下：[实验楼 WebIDE 使用指南](https://www.lanqiao.cn/library/shiyanlou-docs/feature/webide) 的内容。

Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统，我们新建一个.html 后缀的文件，输入以下代码，运行（右击文件 > open with > Preview 或 Mini Browser），你就会看到 {{msg}} 被渲染成 hello

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">{{msg}}</div>

<script>

var app = new Vue({

el: '#app', //dom挂载点

data: {

//数据项

msg: 'hello syl',

},

});

</script>

</body>

</html>

运行效果：



**说明：** el 为实例挂载点，上面表示挂载在 id 为 app 的 dom 元素中。data 选项为数据选项，存放绑定数据。除了这两个之外还有实例选项，methods（实例方法）、computed（计算属性） 等，后面我们会学习到。

## 数据双向绑定

在 Vue 中数据双向绑定随处可见，最常见的是表单数据中的双向绑定，例如：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 数据双向绑定 -->

<div id="app">

<input type="text" v-model="msg" />

<p>{{msg}}</p>

</div>

<script>

var app = new Vue({

el: '#app', //el: 挂载点

data: {

//data:数据选项

msg: 'hello',

},

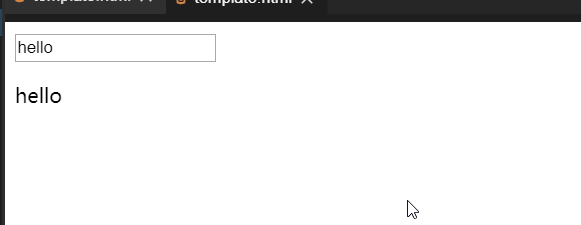
});

</script>

</body>

</html>

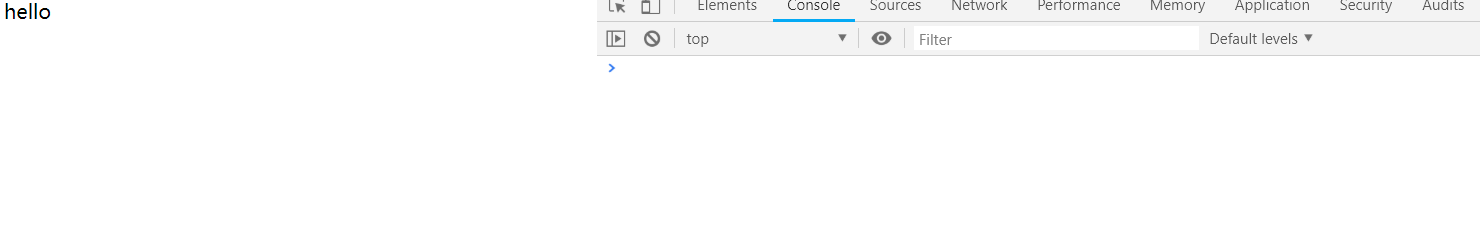
在我们对文本框输入值时，实例 data 中的 msg 值也随之变化。运行效果：



## 感受响应式

上面我们了解到 Vue 是一个 MVVM 架构的框架，成功创建了一个 Vue 应用！看起来这跟渲染一个字符串模板非常类似，但是 Vue 在背后做了大量工作。现在数据和 DOM 已经被建立了关联，所有东西都是 **响应式的**。我们可以看看他是不是响应式的，打开你的浏览器的 JavaScript 控制台 (就在这个页面打开)，并修改 app.msg 的值，你将看到上例相应地更新，更改数据也触发视图的相应更新。

注：由于实验楼环境特殊，上述操作需要在自己电脑上的浏览器中完成。



## 模板语法

### 双大括号表达式

在前端的发展历程中，为了提高开发效率，诞生了很多模板引擎，方便渲染元素或者绑定数据，很多引擎模板都采用 {{双大括号表达式}} 的语法进行插值。同样 Vue.js 也借鉴了 Angular.js 的双花括号的方式，进行向页面输出数据和调用对象方法。让我们感受一下双大括号表达式，在 IED 中新建一个.html 后缀文件，引入 Vue.js，输入以下代码，运行（open with Preview 或 Mini Browser）

代码：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 数据双向绑定 -->

<div id="app">

<input type="text" v-model="msg" />

<p>{{msg}}</p>

</div>

<script>

var app = new Vue({

el: '#app', //el: 挂载点

data: {

//data:数据选项

msg: 'hello',

},

});

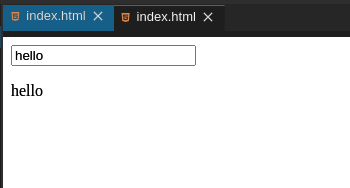
</script>

</body>

</html>

双大括号中的 {{msg}}，绑定至底层 Vue 实例的数据，在浏览器中就被渲染成实例 data 选项中 msg 的值。

运行结果：



### 插值

#### 文本

在 Vue.js 中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：

<div id="app">msg：{{msg}}</div>

双大括号中的值将会被替代为对应 data 对象上 msg 属性的值。无论何时，绑定的数据对象上 msg 属性发生了改变，插值处的内容都会更新。

但是通过使用 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**。但是你需要注意一下，该元素节点下面其他数据的绑定，数据改变，内容也不会更新，所以，注意代码块的划分。

<p v-once>msg：{{msg}}</p>

#### 原始 HTML

上面的双大括号表达式会将数据解释为普通文本，即使你的数据为 HTML 元素，也不会渲染成对应的标签元素，只能渲染成普通文本，而非 HTML 代码，例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 数据绑定 -->

<div id="app">

<p>{{msg}}</p>

</div>

<script>

var app = new Vue({

el: '#app', //el: 挂载点

data: {

//data:数据选项

msg: '<h1>hello syl</h1>',

},

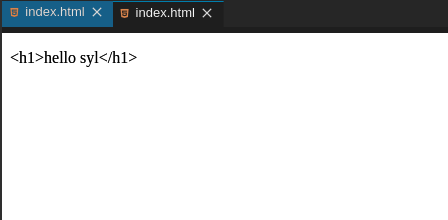
});

</script>

</body>

</html>

运行结果：



上面明明我们写的是 HTML 标签，为什么没渲染出来，那就是因为双大括号表达式会将数据解释为普通文本。有的同学会问，有的需求就是要把标签渲染出来，那么我们就需要认识另外一个指令 v-html，使用它我们就能将它正确渲染，试一试，代码：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- v-html 渲染html元素-->

<div id="app" v-html="msg"></div>

<script>

var app = new Vue({

el: '#app', //el: 挂载点

data: {

//data:数据选项

msg: '<h1>hello syl</h1>',

},

});

</script>

</body>

</html>

运行结果：



#### 特性

双大括号语法不能作用在 HTML 特性（标签属性）上，需要对标签属性操作，应该使用 v-bind 指令：

<div v-bind:class="syl-vue-course"></div>

HTML 标签属性为布尔特性时，它们的存在表示为 true，v-bind 工作起来略有不同，在这个例子中：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 布尔特性绑定-->

<div id="app">

<input type="checkbox" v-bind:checked="isChecked" />

</div>

<script>

var app = new Vue({

el: '#app',

data: {

isChecked: false, // isChecked是否选中boolean

},

});

</script>

</body>

</html>

运行结果：



**注意：** 如果 isChecked 的值是 null、undefined 或 false，则 checked 特性甚至不会被包含在渲染出来的 <input> 元素中，我们将 data 中的 isChecked 值改为 null

var app = new Vue({ el:'#app', data:{ isChecked:null // isChecked是否选中 } })

#### avascript 表达式

上面，我们只进行了绑定简单的属性键值。但实际上，对于所有的数据绑定，Vue.js 都提供了完全的 JavaScript 表达式支持，感受强大的模板语法力量吧！例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- javascript表达式-->

<div id="app">

<!-- 运算符 -->

<p>num + 24 = {{num + 24}}</p>

<!-- 三元表达式 -->

<p>Are you ok? {{ok ? 'I am ok !':'no'}}</p>

<!-- 对象方法直接调用 -->

<p>名字倒过来写：{{name.split('').reverse().join('')}}</p>

<!-- 属性值运算操作 -->

<p v-bind:class="'col'+colNum">syl</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

num: 20,

ok: true,

name: '实验楼',

colNum: '12',

},

});

</script>

</body>

</html>

运行结果：



### 指令

指令 (Directives) 是带有 v- 前缀的特殊特性。

#### 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 特性，在这里 href 是参数，告知 v-bind 指令将该元素的 href 特性与表达式 url 的值绑定，例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 指令 参数-->

<div id="app">

<a v-bind:href="url">实验楼</a>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

url: 'https://www.lanqiao.cn',

},

});

</script>

</body>

</html>

另外一个例子，v-on 指令，用于监听 DOM 事件，例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 指令 参数-->

<div id="app">

<p>我叫：{{name}}</p>

<!-- handleClick 使我们在实例 methods 中写的方法 -->

<button v-on:click="handleClick">点我</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

name: '实验楼',

},

methods: {

//实例方法对象

handleClick: function () {

this.name = this.name.split('').reverse().join('');

},

},

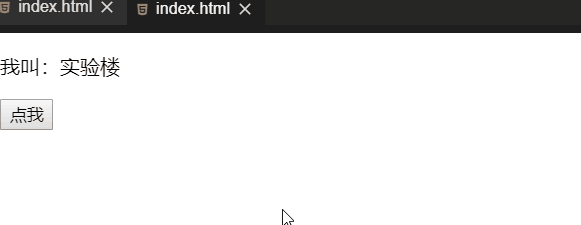
});

</script>

</body>

</html>

运行结果：



#### 动态参数

上面属性或者事件我们都是写死的，其实在 Vue 它也可以是动态的。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 指令 动态参数-->

<div id="app">

<p>我叫：{{name}}</p>

<button v-on:[event]="handleClick">点我</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

name: '实验楼',

event: 'click',

},

methods: {

handleClick: function () {

this.name = this.name.split('').reverse().join('');

},

},

});

</script>

</body>

</html>

event 此时的值为 click ，那我们点击按钮时就会触发事件回调，运行结果和上面一样。

#### 修饰符

修饰符是以半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定，大致分为三类，后面课程我们会一一接触到：

* 事件修饰符
* 按键修饰符
* 系统修饰符

例如，事件修饰符中的.prevent修饰符和原生 event.preventDefault() 效果一样，可以阻止事件默认行为，在表单中点击提交按钮，就会发生页面跳转，但是使用了 .prevent 就不会发生跳转，例如：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue</title>

<!-- 通过cdn方式引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<!-- 指令 修饰符-->

<div id="app">

<form action="/" v-on:submit.prevent="submit">

<button type="submit">提交</button>

</form>

</div>

<script>

var app = new Vue({

el: '#app',

data: {},

methods: {

submit: function () {

console.log('成功提交！');

},

},

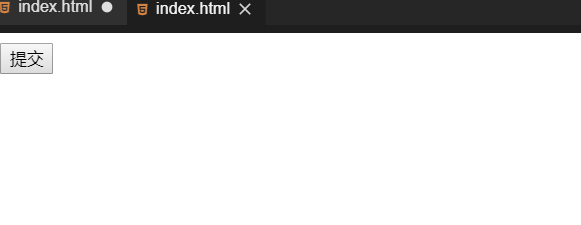
});

</script>

</body>

</html>

没有加修饰符，发生默认跳转，运行效果：



使用了 .prevent ，阻止了默认跳转，运行结果：



#### 指令缩写

##### v-bind

上面例子中我们使用了 v-bind 绑定属性

<a v-bind:href="url">实验楼</a>

我们可以简写为：

<a :href="url">实验楼</a>

同样的使用 v-bind 绑定的其他属性也可以简写：

v-bind:class="className" 简写为 :class="className" v-bind:value="myValue" 简写为

:value

##### v-on

上面 v-bind 指令提供简写，同样 v-on 指令也提供简写，但是与 v-bind 有一些差异，v-on: 使用 @ 简写。

<!-- 完整语法 -->

<button v-on:click="handleClick">点我</button>

<!-- 缩写 -->

<button @click="handleClick">点我</button>

## 计算属性和侦听属性与过滤器

在开发中经常会涉及到一种需求，一个数据需要通过其他数据计算而来。例如：购物车，平常开发数据与数据关联计算是非常麻烦的，而 Vue 中的计算属性可以轻松帮你解决，可以像绑定普通属性一样在模板中绑定计算属性，可以直接使用{{}}向页面输出。

### 计算属性的基本使用

在实例的 computed 选项中定义你的计算属性，直接使用{{}}向页面输出。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p>我名字正着写：{{name}}</p>

<!-- reverseName 计算属性 可以像绑定普通属性一样在模板中绑定计算属性-->

<p>计算出我名字倒着写：{{reverseName}}</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

name: '实验楼',

},

computed: {

//reverseName 是一个计算属性

reverseName: function () {

return this.name.split('').reverse().join('');

},

},

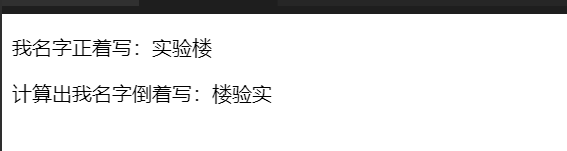
});

</script>

</body>

</html>

运行结果：



这里我们声明了一个计算属性 reverseName ，你可以像绑定普通属性一样在模板中绑定计算属性。当你的计算属性的依赖数据发生改变时，你的相关计算属性也会重新计算。上面通过计算属性轻松地实现出展示名字倒着写，你也快看看，你名字倒过来是什么！

**重点：** 在 Vue 中计算属性是 **惰性的**，只有当依赖数据发生改变时，才会触发计算，否则，它的值是上一次触发计算的缓存值，下面例子很清楚说明，它是惰性的。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">{{now}}</div>

<script>

var app = new Vue({

el: '#app',

data: {},

computed: {

now: function () {

return Date.now();

},

},

});

</script>

</body>

</html>

在你的理解中界面会一直时间更新,其实我们定义的 now 并没有和实例中数据建立响应式依赖，只是依赖 Date 对象获取系统时间，它只会计算一次，然后将值缓存，要使得他改变只有 **刷新** 才能触发，运行结果：



### 计算属性的 setter 和 getter

计算属性的 setter 和 getter 是它的高级使用，上面的例子只使用到它的 getter 属性，通过其他数据计算而得，其实我们也可以，直接赋值，通过计算属性来改变依赖数据的值。

下面例子我们通过触发计算属性 setter，改变原本关联数据。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p>firstName:{{firstName}}</p>

<p>lastName:{{lastName}}</p>

<p>全名是:{{fullName}}</p>

<button v-on:click="changeName">改姓</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

firstName: '王',

lastName: '花花',

},

methods: {

//changeName 定义一个方法改变 计算属性 fullName 的值

changeName: function () {

//修改计算属性 fullName 等于李花花

this.fullName = '李花花';

//上面一句等于触发了 fullName 属性的 setter

},

},

computed: {

fullName: {

//getter

get: function () {

return this.firstName + this.lastName;

},

//setter 直接改变计算属性 fullName的值就可以触发setter this.fullName='XX'

set: function (newName) {

var name = newName;

this.firstName = name.slice(0, 1); //取新值的第一个字符

this.lastName = name.slice(1); //从新值的第二个字符开始取值

},

},

},

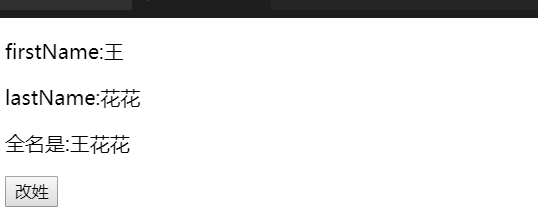
});

</script>

</body>

</html>

运行结果：



### 侦听属性

在开发我们需要监听数据的变化，Vue 提供了一种更通用的方式来观察和响应 Vue 实例上的数据变动，侦听属性。在实例 watch 选项中确定监听项。

下面这个例子，我们通过按钮点击改变 msg 的值，并且监听新旧值改变，并输出新旧值。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p>{{msg}}</p>

<!-- v-on:click 简写为 @click -->

<button @click="handleClick('hello syl')">改变msg</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

msg: 'hello',

},

methods: {

//改变 msg的值

handleClick: function (val) {

this.msg = val;

},

},

// watch 监听属性

watch: {

//监听新旧值 监听属性有两个参数，第一个新值，第二个旧值

msg: function (newVal, oldVal) {

alert('新值' + newVal + '----' + '旧值' + oldVal);

},

},

});

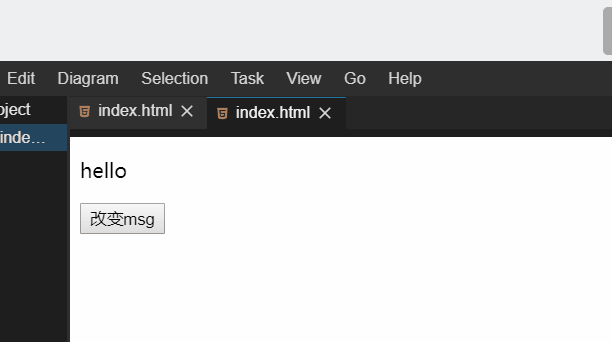
</script>

</body>

</html>



运行效果：



#### 计算属性与侦听属性对比

计算属性和侦听属性两者在很多场景都是共同，都可以实现同样的需求。

我们使用侦听属性，改写计算属性例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p>firstName:{{firstName}}</p>

<p>lastName:{{lastName}}</p>

<p>全名是:{{fullName}}</p>

<button v-on:click="changeName">改姓</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

firstName: '王',

lastName: '花花',

fullName: '王花花',

},

methods: {

//changeName 定义一个方法改变 fullName 的值

changeName: function () {

this.fullName = '李花花';

},

},

watch: {

//fullName 侦听属性，监听fullName 值的改变

fullName: function (val) {

var name = val;

this.firstName = name.slice(0, 1);

this.lastName = name.slice(1);

},

},

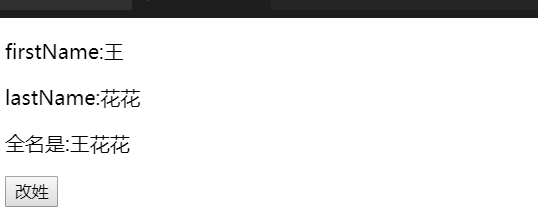
});

</script>

</body>

</html>

效果是一样的，运行效果:



**注意：** 通常来说能用计算属性代替就用计算属性代替，多大情况计算属性代码更精简，可读性更高。

### 过滤器

计算属性和侦听属性，在数据处理环节中很有作用，但是用它来处理数据过滤不是最优的，在 Vue 中我们有一个专门处理数据过滤的东西：过滤器。过滤器可以用在两个地方：**双花括号插值和 v-bind 表达式** (后者从 2.1.0+ 开始支持)，用法示例：

<p>{{msg2|getString}}</p>

<p v-bind:class="msg2|getString"></p>

#### 过滤器使用方法

在双花括号插值和 v-bind 表达式中把需要过滤的数据用 **|** 与过滤器分割 **（data|fliter）**

例子：

使用 filters 过滤器实现大写转换和自动去除字符串中的数字

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- toUpperCase getString 为自定义的过滤器-->

<p>小写转换大写：过滤前：{{msg}} 过滤后： {{msg|toUpperCase}}</p>

<p>去除数字：过滤前：{{msg2}} 过滤后： {{msg2|getString}}</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

msg: 'hello',

msg2: '1s2y3l',

},

// filters 过滤器选项

filters: {

//toUpperCase 定义一个字符串转大写的过滤器

toUpperCase: function (val) {

return val.toUpperCase();

},

//getString 定义一个获取去除数字的过滤器

getString: function (val) {

let newVal = '';

val.split('').map(function (item) {

if (9 >= item && item >= 0) {

return;

} else {

return (newVal += item);

}

});

return newVal;

},

},

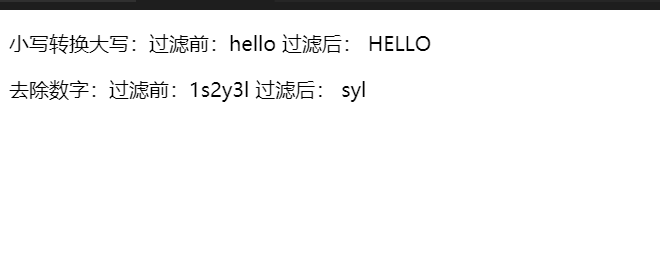
});

</script>

</body>

</html>

运行结果：



#### 过滤器应用场景

应用比较多的商品价格过滤、表单数据过滤等。

我们从后台获取的数据一般是这样

{ courseName:'xxx', price:199, coupon:8 }



我们请求到的数据并没有￥，在开发中直接去操作数据源是不推荐的，此时我们的过滤器就派上用场，定义一个拼接￥的过滤器，只是在视图层面实现了效果。



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- joint 为自定义的过滤器-->

<p>不要￥899，只要{{price|joint}}</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

//后台价格数据

price: 199,

},

// filters 过滤器选项

filters: {

//joint 定义￥拼接过滤器

joint: function (price) {

return '￥' + price;

},

},

});

</script>

</body>

</html>

运行结果：



### 综合小练习

综合计算属性和过滤器，实现一个简易的购物车：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

\* {

padding: 0;

margin: 0;

}

.price {

font-size: 22px;

color: brown;

}

</style>

</head>

<body>

<div id="app">

<!-- joint 为自定义的过滤器-->

<p>单价<span class="price">{{price|joint}}</span></p>

数量：<input type="number" v-model="goodsNum" />

<p>总价：<span class="price">{{allPrice|joint}}</span></p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

goodsNum: 0,

price: 199,

},

computed: {

allPrice: function () {

return this.goodsNum \* this.price;

},

},

// filters 过滤器选项

filters: {

//joint 定义￥拼接过滤器

joint: function (price) {

return '￥' + price;

},

},

});

</script>

</body>

</html>

效果：



## class 与 style 绑定

### 元素的 class 绑定

#### 对象语法

给 v-bind:class 一个对象，以动态地切换 class，语法表示 active 这个 class 存在与否将取决于数据属性 isActive 的 Boolean 值，大致语法 {className:Boolean}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

.active {

color: pink;

font-size: 22px;

}

</style>

</head>

<body>

<div id="app">

<!-- 数组语法绑定 class 当 isActive 为 true 时，active 就成 span 标签的 class -->

<span v-bind:class="{'active':isActive}">syl</span>

<!-- isActive 为true 渲染结果 <span class="active">syl</span> -->

</div>

<script>

var app = new Vue({

el: '#app',

data: {

isActive: true,

},

});

</script>

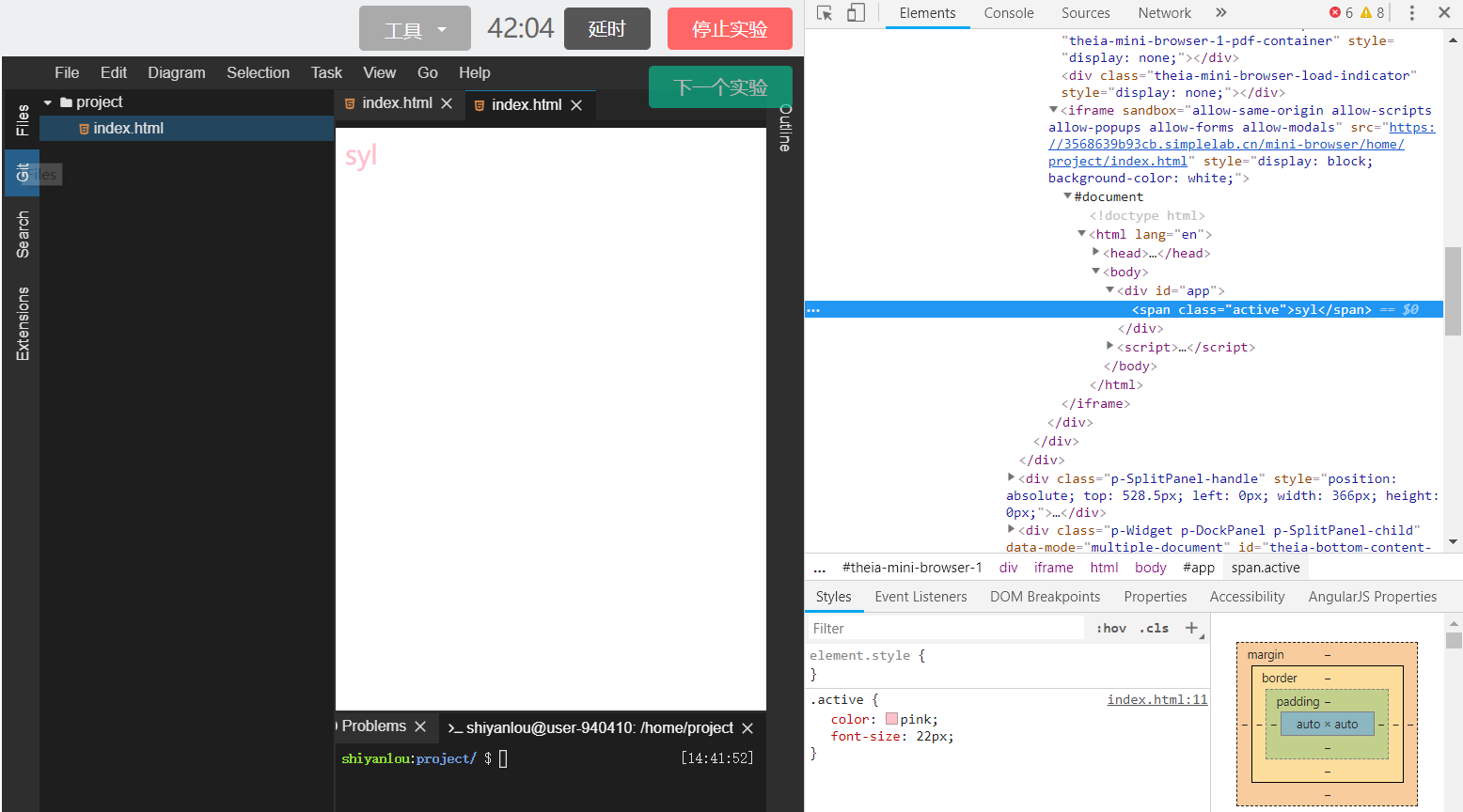
</body>

</html>

运行结果：



最终标签 class 渲染结果：



你可以在对象中传入更多属性来动态切换多个 class，v-bind:class 指令也可让普通的 class 并存。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

.active {

color: pink;

font-size: 22px;

}

.red-bg {

background: red;

}

</style>

</head>

<body>

<div id="app">

<!-- 数组语法绑定 class 当isActive为true时，active就成成span标签的class -->

<span class="static" v-bind:class="{'active':isActive,'red-bg':isRed}"

>syl</span

>

<!-- isActive 为true 渲染后 <span class="static active red-bg">syl</span> -->

</div>

<script>

var app = new Vue({

el: '#app',

data: {

isActive: true,

isRed: true,

},

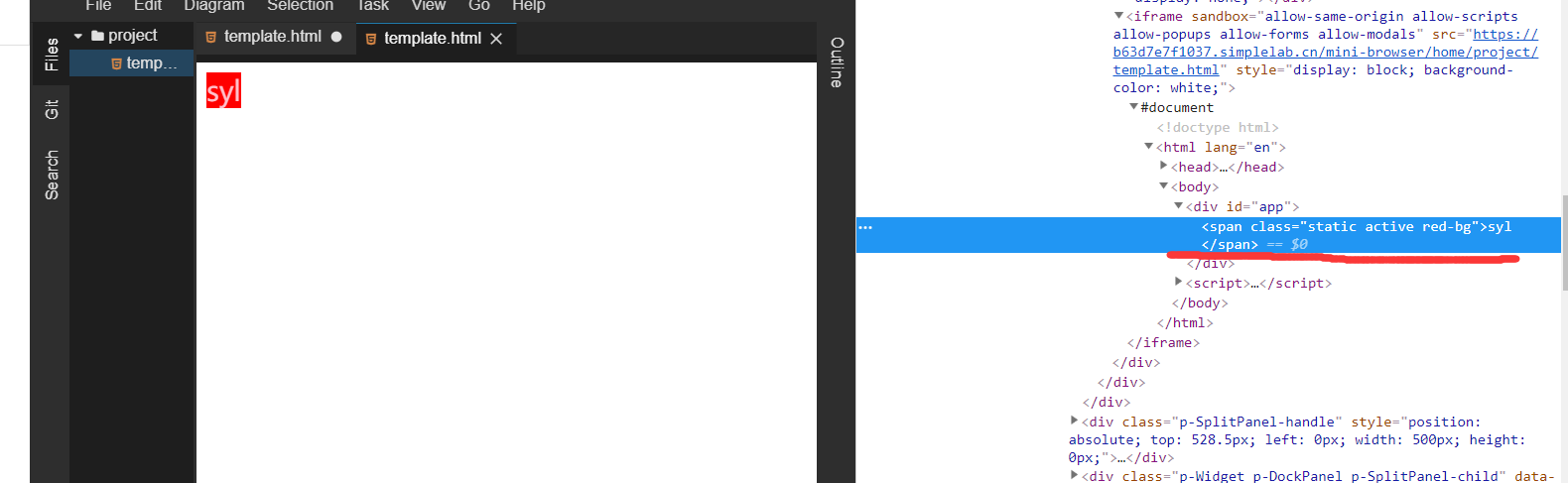
});

</script>

</body>

</html>

运行结果:



#### 数组语法

我们可以把一个数组传给 v-bind:class，以应用一个 class 列表：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

.active {

color: pink;

font-size: 22px;

}

.red-bg {

background: red;

}

</style>

</head>

<body>

<div id="app">

<!-- 数组语法绑定 class -->

<span v-bind:class="[activeClass,bgColorClass]">syl</span>

<!-- 渲染后 <span class="active red-bg">syl</span> -->

</div>

<script>

var app = new Vue({

el: '#app',

data: {

activeClass: 'active',

bgColorClass: 'red-bg',

},

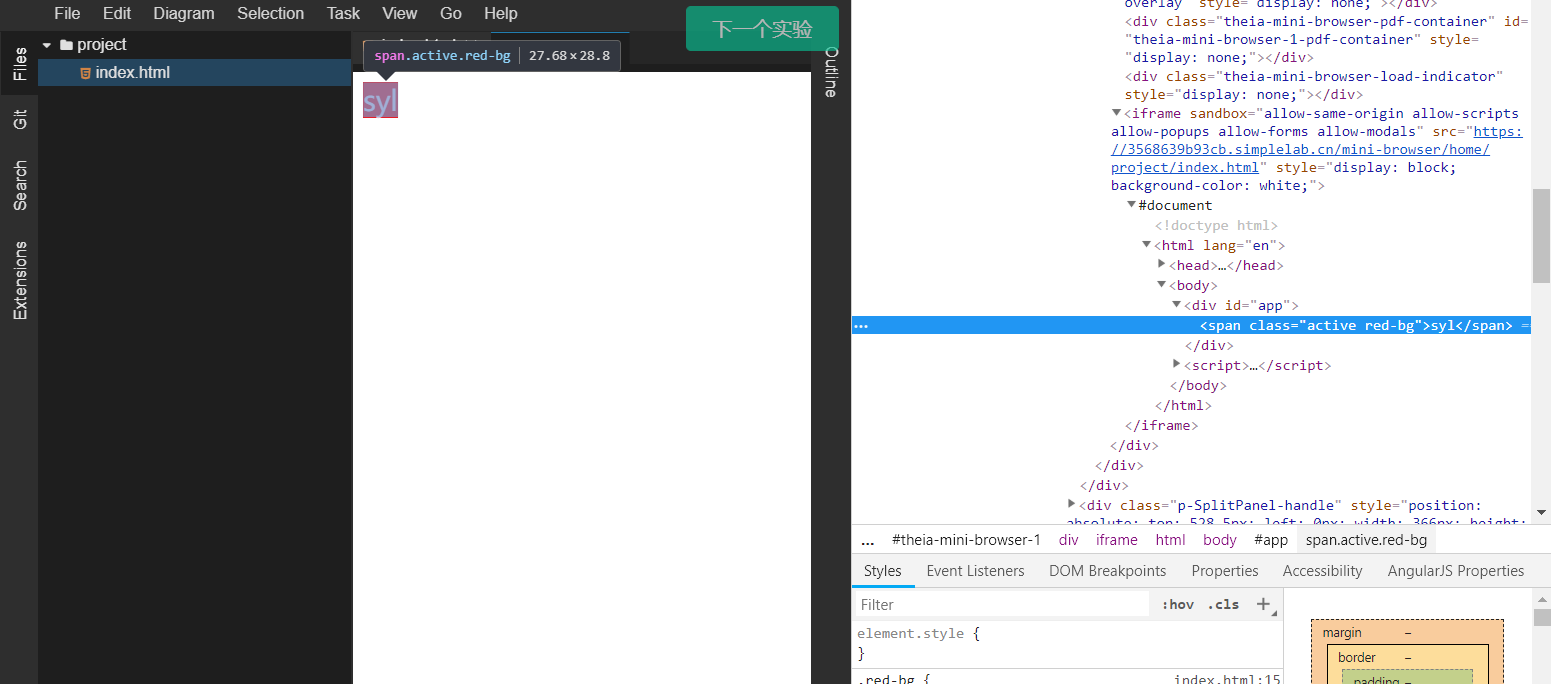
});

</script>

</body>

</html>

运行结果：



### 元素 style 绑定

在前端开发中，内联样式经常被使用到，Vue 中内联样式绑定语法灵活。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。

小建议：如果，你曾使用过 React 或者将要学习它，建议采用与 React 同样的 CSS 属性代码风格驼峰式 (camelCase)，本教程中也使用风格。

#### 对象语法

使用对象方式绑定 style

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p v-bind:style="{fontSize:size,backgroundColor:bgColor}">你好，实验楼</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

size: '26px',

bgColor: 'pink',

},

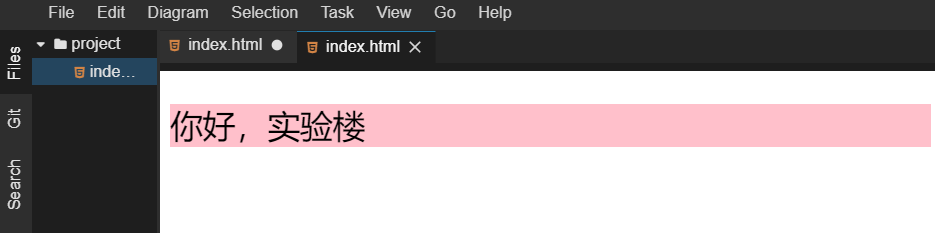
});

</script>

</body>

</html>

运行结果：



直接绑定到一个样式对象通常更好，这会让模板更清晰，和上面一个例子是同样的效果：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p v-bind:style="styleObject">你好，实验楼</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

styleObject: {

fontSize: '26px',

backgroundColor: 'pink',

},

},

});

</script>

</body>

</html>

#### 数组语法

v-bind:style 的数组语法可以将多个样式对象应用到同一个元素上

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p v-bind:style="[styleObject1,styleObject2]">你好，实验楼</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

//样式一

styleObject1: {

fontSize: '26px',

backgroundColor: 'pink',

},

//样式二

styleObject2: {

marginTop: '200px',

textAlign: 'center',

},

},

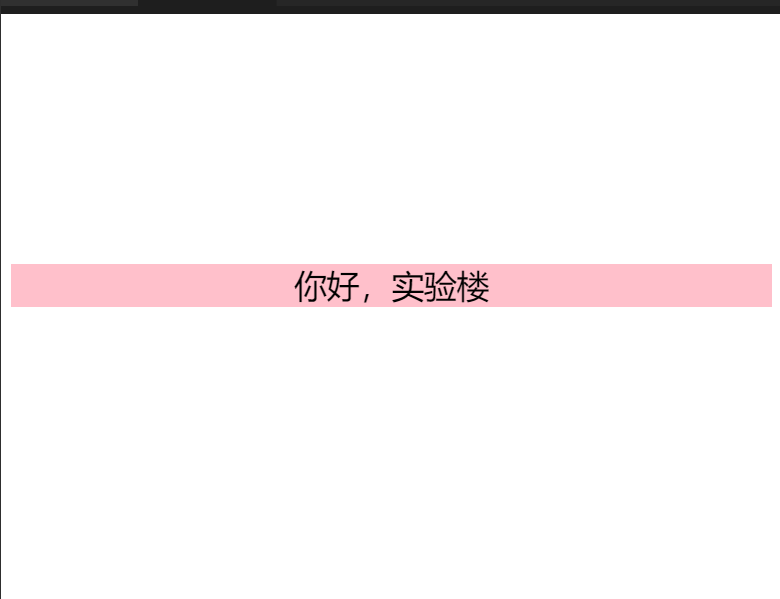
});

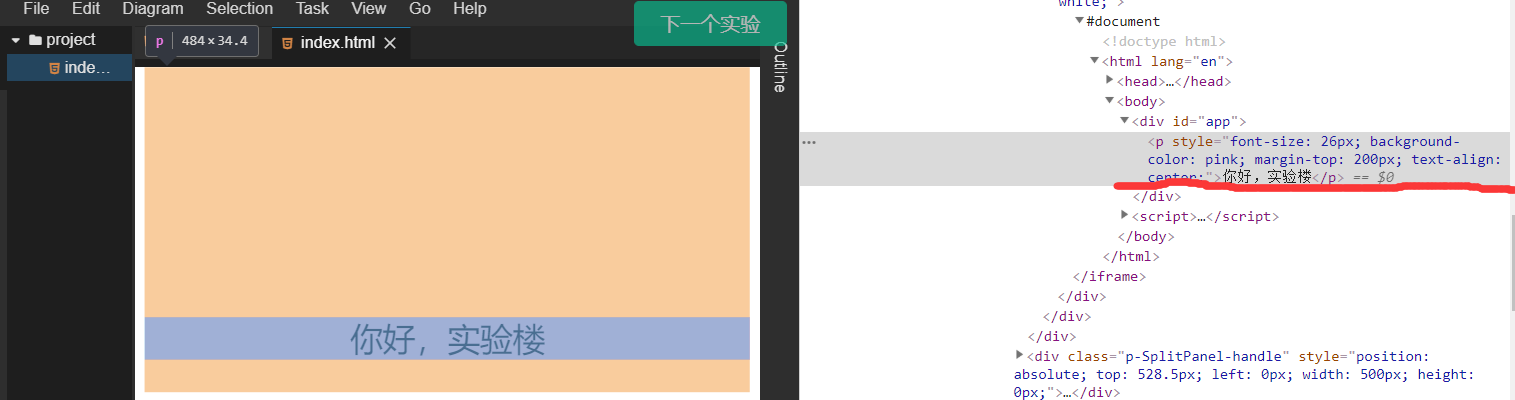
</script>

</body>

</html>

运行结果：





### 属性的自动前缀

在开发中，由于浏览器内核不同，使用某些 CSS 属性需要带各浏览器的前缀，然而如果你在 Vue 中使用了 v-bind:style你完全不用去考虑，因为 Vue 在编译过程中，会自动给需要前缀的属性加前缀。

**引申知识：**通过[CANIUSE](https://www.caniuse.com/)查看询 CSS 属性兼容性，自动前缀包 Autoprefixer、postcss。

### 综合练习

动态改变 style 和 切换 class，案例：变大变色

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

.active {

color: red;

}

</style>

</head>

<body>

<div id="app">

<p

v-bind:style="{fontSize:fontSize+'px'}"

v-bind:class="{active:isActive}"

>

你好，实验楼

</p>

<button @click="handleClick">变大变色</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

fontSize: 20,

isActive: true,

},

methods: {

handleClick: function () {

this.fontSize += 2;

this.isActive = !this.isActive; //取反

},

},

});

</script>

</body>

</html>

结果：



## 条件与循环渲染

前面提到 Vue.js 的作用是动态的构建用户界面，灵活的控制渲染是很有必要的，快速构建也是很有必要的，条件渲染和循环渲染，在视图构建中起着重要作用。

### v-if

v-if 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回除了 false，0，""，null，undefined 和 NaN 外的值的时候被渲染。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p v-if="display">你好，实验楼</p>

<p>SYL</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

display: true,

},

});

</script>

</body>

</html>

display:true 运行结果：



display:fasle ,运行结果:



### v-else

使用 v-else 指令来表示 v-if 的“else 块”，当 v-if 返回除了 false，0，""，null，undefined 和 NaN 外的值的时候被渲染，否则，就渲染 v-else 块元素。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p v-if="display">你好，实验楼</p>

<p v-else>SYL</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

display: true,

},

});

</script>

</body>

</html>

运行结果：



**注意：**v-else 元素必须紧跟在带 v-if 或者 v-else-if 的元素的后面，否则它将不会被识别。

### v-show

另一个用于根据条件展示元素的选项是 v-show 指令。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。而 v-if 是从虚拟 DOM 的层面操作。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p v-show="show">你好，实验楼</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

show: true,

},

});

</script>

</body>

</html>

运行结果：



### v-if 和 v-show 比较

一般来说，v-if 会牵涉到虚拟 DOM diff 算法，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。

### 循环渲染

循环渲染经常会使用到，在开发中经常会遇到 DOM 结构一样的块代码，那么我们就可以使用循环渲染来一步到位。

#### v-for 将数组数据渲染成元素

将菜单数组循环成一个完成的菜单：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

\* {

padding: 0;

margin: 0;

}

ul {

width: 100%;

height: 40px;

list-style: none;

display: flex;

flex-direction: row;

align-items: center;

justify-content: center;

background: yellowgreen;

}

ul li {

width: 20%;

height: 100%;

color: white;

line-height: 40px;

text-align: center;

text-transform: uppercase; //大写转换

}

</style>

</head>

<body>

<div id="app">

<ul class="nav">

<li v-for="navItem in nav">

{{navItem}}

</li>

</ul>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

nav: ['home', 'shop', 'contact', 'about', 'name', 'mroe', 'histroy'],

},

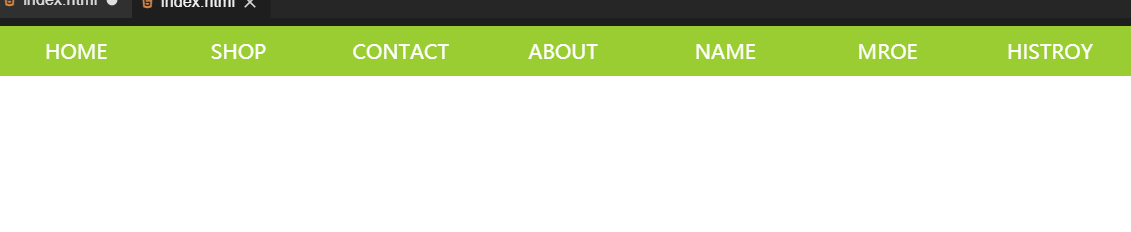
});

</script>

</body>

</html>

运行结果：



#### v-for 将对象数据渲染成元素

在开发中我们获取到的数据经常是对象形式

对象数据循环出数据：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<div v-for="val in userInfo">

<p>{{val}}</p>

</div>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

userInfo: {

name: 'whh',

age: 1,

sex: 'woman',

},

},

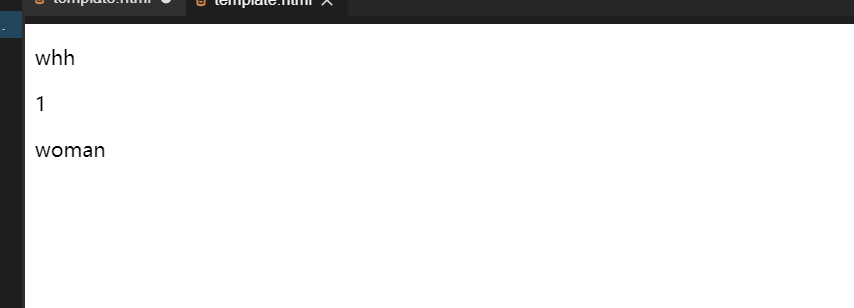
});

</script>

</body>

</html>

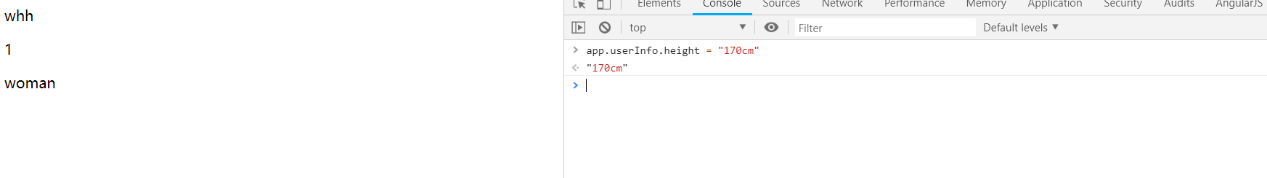
运行结果：



#### 注意

还是由于 JavaScript 的限制，Vue 不能检测对象属性的添加或删除，直接进行 app.userInfo.height='180cm' 这样操作是不会构成响应式，不会触发视图更新。必须使用 Vue.set(object, key, value) 方法向嵌套对象添加响应式属性

直接进行 app.userInfo.height='170cm' ，视图未能更新：



使用 Vue.set(object, key, value) 方法：

Vue.set(app.userInfo, 'height', '170cm');



所以，使用对象方法时，要想触发视图更新，常常需要使用 Vue.set()去操作。

#### key 属性

为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key 属性。理想的 key 值是每项都有的唯一 id。这个特殊的属性相当于 Vue 1.x 的 track-by ，但它的工作方式类似于一个属性，所以你需要用 v-bind 来绑定动态值 (在这里使用简写)：

<div v-for="item in items" :key="item.id">

<!-- 内容 -->

</div>

**注意：**key 并不与 v-for 特别关联，建议尽可能在使用 v-for 时提供 key，除非遍历输出的 DOM 内容非常简单，或者是刻意依赖默认行为以获取性能上的提升。

#### 数组更新检测

一些操作数组的方法，编译会检测，从而会促使视图更新。

##### 变异方法

* push()
* pop()
* shift()
* unshift()
* splice()
* sort()
* reverse()

上面这些数组操作方法，会直接改变原始数组称为变异方法，会促使视图自动更新。

##### 替换数组

学了 JavaScript 标准对象库，都知道有些数组方法是不直接改变原数组的，这里称他们为非变异方法，例如：filter()、slice()、concat()，他们都是返回一个新数组，那么，在 Vue 中使用到这些方法，怎么样才能促使视图更新呢？我们就必须使用数组替换法，将非变异方法返回的新数组直接赋值给的旧数组

this.nav = this.nav.slice(1, 4);

##### 注意

由于 JavaScript 的限制，Vue 不能检测以下变动的数组：

1. 当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue
2. 当你修改数组的长度时，例如：vm.items.length = newLength

例子：

var app = new Vue({

data: {

items: ['a', 'b', 'c'],

},

});

app.items[1] = 'x'; // 不是响应性的

app.items.length = 2; // 不是响应性的

上去直接这样改值操作是没有问题的，但是不是响应式的，并不能触发视图更新，需要用其他方法代替。

例如这样的操作 app.items[indexOfItem] = newValue ，可以用以下两种代替。

// Vue.set

Vue.set(vm.items, indexOfItem, newValue);

// Array.prototype.splice

vm.items.splice(indexOfItem, 1, newValue);

#### 显示过滤

显示一个数组的过滤或排序副本，而不实际改变或重置原始数据。在这种情况下，可以创建返回过滤或排序数组的计算属性。

例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<span v-for="number in oddNumber">{{number}}</span>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

numberArray: [1, 2, 3, 4, 5, 6, 7, 8],

},

computed: {

//计算 numberArray 中为奇数的 oddNumber 奇数数组

oddNumber: function () {

return this.numberArray.filter(function (number) {

return number % 2 === 1;

});

},

},

});

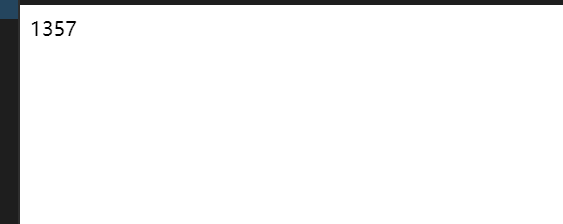
</script>

</body>

</html>



运行结果：



如果你不想用计算属性，你也可以直接使用 **方法** 去操作，例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- v-for 内直接调用方法 -->

<span v-for="number in getOddNumber()">{{number}}</span>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

numberArray: [1, 2, 3, 4, 5, 6, 7, 8],

},

methods: {

//定一个一个获取数组内奇数的方法 filter 数组对象的过滤方法

getOddNumber: function () {

return this.numberArray.filter(function (number) {

return number % 2 === 1;

});

},

},

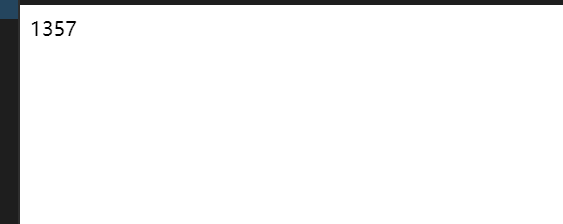
});

</script>

</body>

</html>

运行结果：



#### v-for 循环一段值

在开发中我们会遇到要需要一段值的情况，不管是字符拼接，还是分页，我们都要用到，在 Vue 中我们可以用 v-for 轻松实现。

循环出 10 页分页：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- :style 绑定样式 -->

<span v-for="number in 10" :style="styleObject">{{number}}</span>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

//指定分页样式

styleObject: {

fontSize: '14px',

color: '#fff',

background: 'green',

padding: '5px 10px',

border: '1px solid #fff',

},

},

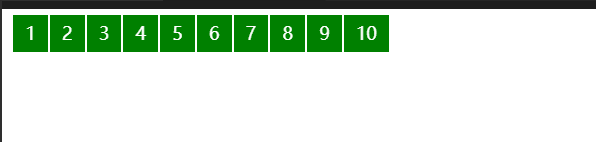
});

</script>

</body>

</html>

运行结果：



#### v-for 与 v-if 搭配使用

循环中嵌套控制，开发中很常见，在 Vue 中我们也可以这样使用。

例子，数组有元素就循环渲染，没有就渲染提示信息：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<div v-if="numbers.length">

<div v-for="item in numbers">

{{item}}

</div>

</div>

<div style="color:red" v-else>请添加数组元素</div>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

numbers: [1, 2, 3, 4],

},

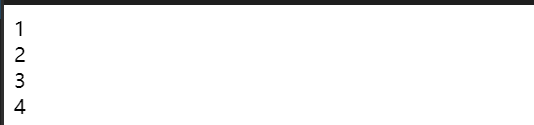
});

</script>

</body>

</html>

数组内有元素：



数组内没有元素 numbers:[]：



#### 综合小练习，多级炫酷菜单渲染

该练习综合了数组渲染、对象渲染，v-if 、v-show、 key 、事件方法等知识

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<style>

\* {

padding: 0;

margin: 0;

}

a {

text-decoration: none;

color: #fff;

}

ul {

list-style: none;

}

nav,

ul {

width: 100%;

display: flex; /\* 开启弹性盒模型 布局方式\*/

flex-direction: row;

justify-content: center;

background: yellowgreen;

}

nav > ul > li {

width: 20%;

height: 100%;

text-align: center;

line-height: 50px;

}

nav > ul > li:hover {

box-shadow: 1px 0px 10px #fff;

}

nav > ul > li > ul {

display: flex;

flex-direction: column;

}

nav > ul > li > ul > li {

box-shadow: 1px 0px 10px #fff;

}

nav > ul > li > a {

text-transform: uppercase;

}

</style>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<nav>

<ul>

<!-- 循环渲染一级菜单 -->

<!-- 鼠标移入触发currentIndex(index)函数，更正current是当前菜单的index, 鼠标移出重置current为空 事件回调方法在methods中实现-->

<li

v-for="(nav,index) in navbar"

:key="index"

@mouseover="currentIndex(index)"

@mouseout="changeIndex"

>

<!-- nav.name 一级菜单名字 -->

<a href="javascript:;">{{nav.name}}</a>

<!-- 如果nav.child存在，说明有子菜单，再次循环渲染子菜单 -->

<!-- 子菜单v-show 如果当前菜单的 index 等于 鼠标移入那个菜单的下标我们就展示出子菜单-->

<ul v-if="nav.child" v-show="current===index">

<li v-for="item in nav.child">

<a href="javascript:;">{{item}}</a>

</li>

</ul>

</li>

</ul>

</nav>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

//navbar 模拟后台获取到的菜单列表

navbar: [

{

name: 'home',

child: ['homeItem', 'homeItem'],

},

{

name: 'contact',

child: ['contactItem', 'contactItem'],

},

{

name: 'about',

},

],

//current 当前鼠标在那个菜单上 ，初始时没有值

current: null,

},

methods: {

//更正 当前鼠标移入的是哪个菜单的 index

currentIndex: function (index) {

this.current = index;

},

//鼠标移出 重置current 值

changeIndex: function () {

this.current = null;

},

},

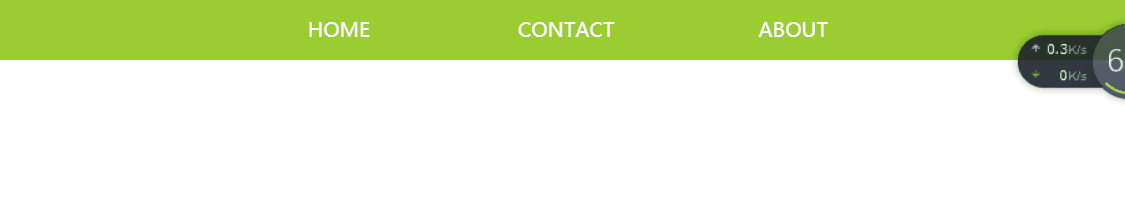
});

</script>

</body>

</html>

虽然上面项目用 v-show 去实现子菜单，显得很累赘（css hover 轻松解决），重在传递的是一种思想，运行结果：



## 事件处理

### 事件监听

简单示例，计数器：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

button {

width: 150px;

height: 40px;

border-radius: 10px;

background: green;

outline: none;

color: #fff;

}

</style>

</head>

<body>

<div id="app">

<!-- v-on:click 绑定点击监听 点一次counter累加一次-->

<button v-on:click="counter+=1">点击</button>

<p>你点了{{counter}}次</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

counter: 0,

},

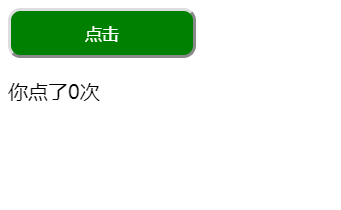
});

</script>

</body>

</html>

试一试你的手速，运行效果：



### 事件监听处理方法

许多事件处理逻辑会更为复杂，所以直接把 JavaScript 代码写在 v-on 指令中是不可行的。因此 v-on 还可以接收一个需要调用的方法名称。该方法写在 methods 对象中。

#### methods 对象中实现方法

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

button {

width: 150px;

height: 40px;

border-radius: 10px;

background: green;

outline: none;

color: #fff;

}

</style>

</head>

<body>

<div id="app">

<!-- 绑定点击监听 -->

<button v-on:click="say">点击</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

counter: 0,

},

methods: {

//声明事件点击监听 say方法

say: function (event) {

//监听事件回调处理 event.type 触发事件类型 说明：`${}`为es6模板字符串，拼接字符串的

alert(`小楼提醒：你触发了${event.type}事件`);

},

},

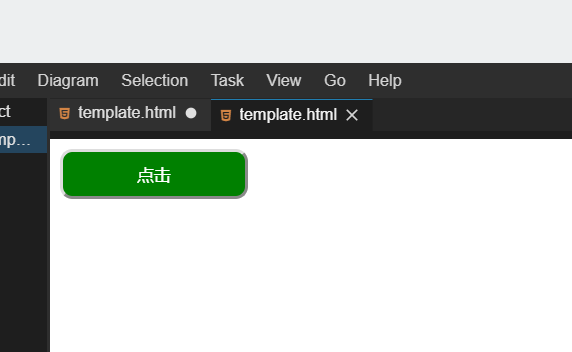
});

</script>

</body>

</html>

运行结果：



#### 内联处理器的方法

不同元素触发同一事件或不同事件，调用同用一个方法。

例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

button {

width: 150px;

height: 40px;

border-radius: 10px;

background: green;

outline: none;

color: #fff;

}

</style>

</head>

<body>

<div id="app">

<!-- 绑定点击监听 共用say 方法-->

<button v-on:click="say('实验楼')">实验楼</button>

<button v-on:click="say('小楼')">小楼</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {},

methods: {

//声明事件点击监听 say方法

say: function (name) {

alert(`我是${name}`);

},

},

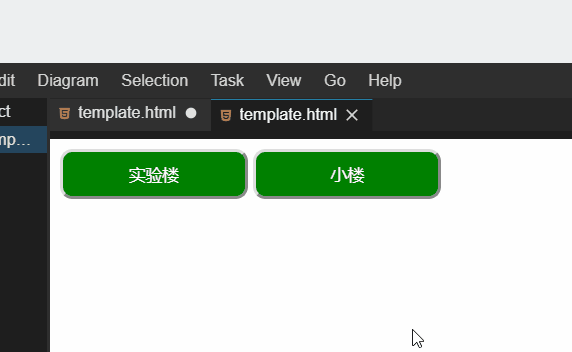
});

</script>

</body>

</html>

运行结果：



#### 事件修饰符

在事件处理程序中调用 event.preventDefault() 或 event.stopPropagation() 是非常常见的需求，阻止事件冒泡或捕获或者事件默认行为。尽管我们可以在方法中轻松实现这点，但更好的方式是：方法只有纯粹的数据逻辑，而不是去处理 DOM 事件细节。

为了解决这个问题，Vue.js 为 v-on 提供了 **事件修饰符**。之前提过，修饰符是由点开头的指令后缀来表示的。

* .stop(阻止单击事件继续传播)
* .prevent（阻止事件默认行为）
* .capture（添加事件监听器时使用事件捕获模式）
* .self（只当在 event.target 是当前元素自身时触发处理函数 ）
* .once（点击事件将只会触发一次）
* .passive（滚动事件的默认行为 (即滚动行为) 将会立即触发 ）

例子：

.stop 修饰符的应用

未添加 .stop修饰符，事件会触发冒泡行为，点击子元素也会触发父元素的相同事件

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

/\* 居中 \*/

.super,

.child {

position: absolute;

top: 0;

left: 0;

bottom: 0;

right: 0;

margin: auto;

}

.super {

width: 300px;

height: 300px;

background: pink;

}

.super .child {

width: 100px;

height: 100px;

background: green;

}

</style>

</head>

<body>

<div id="app">

<div class="super" v-on:click="handleClick('super')">

父

<div class="child" v-on:click="handleClick('child')">

子

</div>

</div>

</div>

<script>

var app = new Vue({

el: '#app',

data: {},

methods: {

//声明事件点击监听 handleClick

handleClick: function (name) {

alert(`我是${name}`);

},

},

});

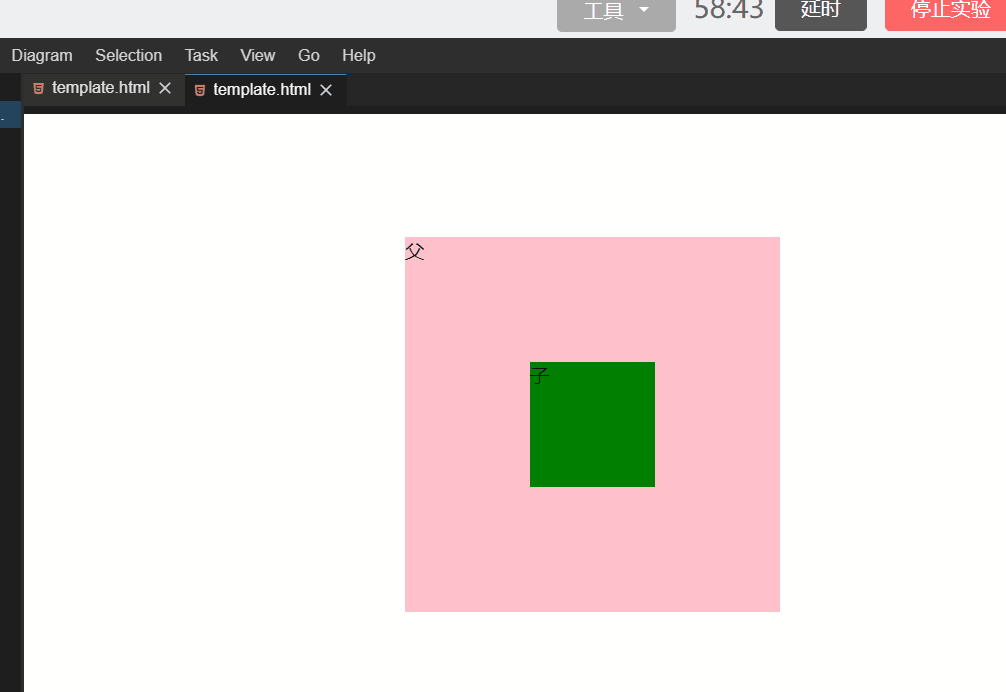
</script>

</body>

</html>



运行效果：



添加修饰符：

<div id="app">

<div class="super" v-on:click.stop="handleClick('super')">

父

<div class="child" v-on:click.stop="handleClick('child')">

子

</div>

</div>

</div>

成功阻止事件冒泡，运行结果：



#### 按键修饰符

在监听键盘事件时，我们经常需要检查详细的按键。Vue 允许为 v-on 在监听键盘事件时添加按键修饰符：

.enter

.tab

.delete (捕获“删除”和“退格”键)

.esc

.space

.up

.down

.left

.right

##### .enter 例子

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<input

type="text"

v-on:keyup.enter="alert('你按了enter,确定输入完毕？')"

/>

</div>

<script>

var app = new Vue({

el: '#app',

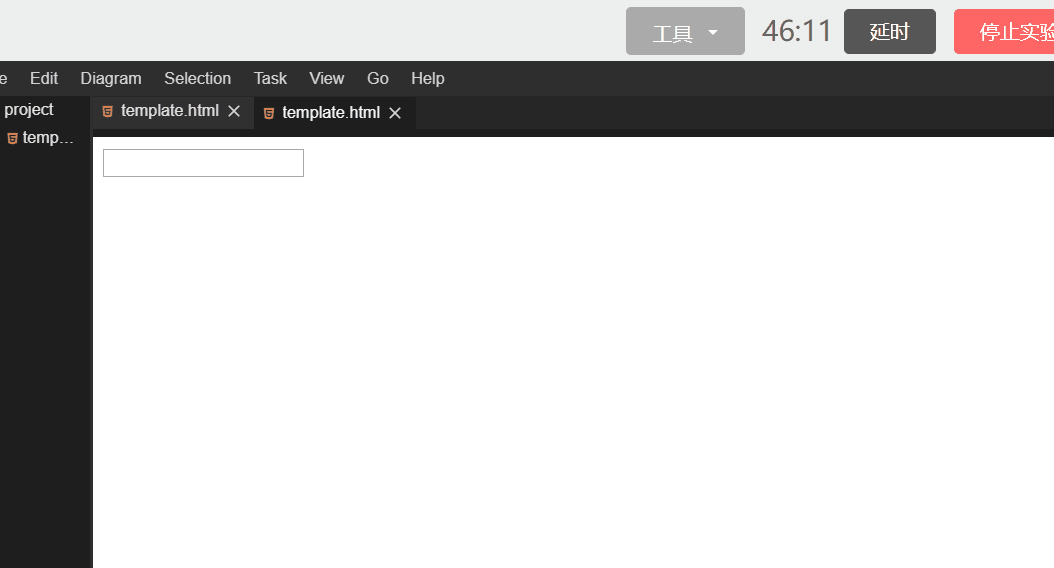
});

</script>

</body>

</html>

运行结果：



##### 键码 keycode

使用 keyCode 特征，了解更多[键码](https://jingyan.baidu.com/article/fec7a1e5d6700a1190b4e725.html)：

| **按键** | **键码** |
| --- | --- |
| Enter | 13 |
| Shift | 16 |
| Alt | 18 |
| Spacebar | 32 |
| Page Up | 33 |
| Page Down | 34 |

示例：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 文本框内按了键码为13的键（enter） 提示 是否输入完毕 -->

<input type="text" v-on:keyup.13="alert('你按了enter,确定输入完毕？')" />

</div>

<script>

var app = new Vue({

el: '#app',

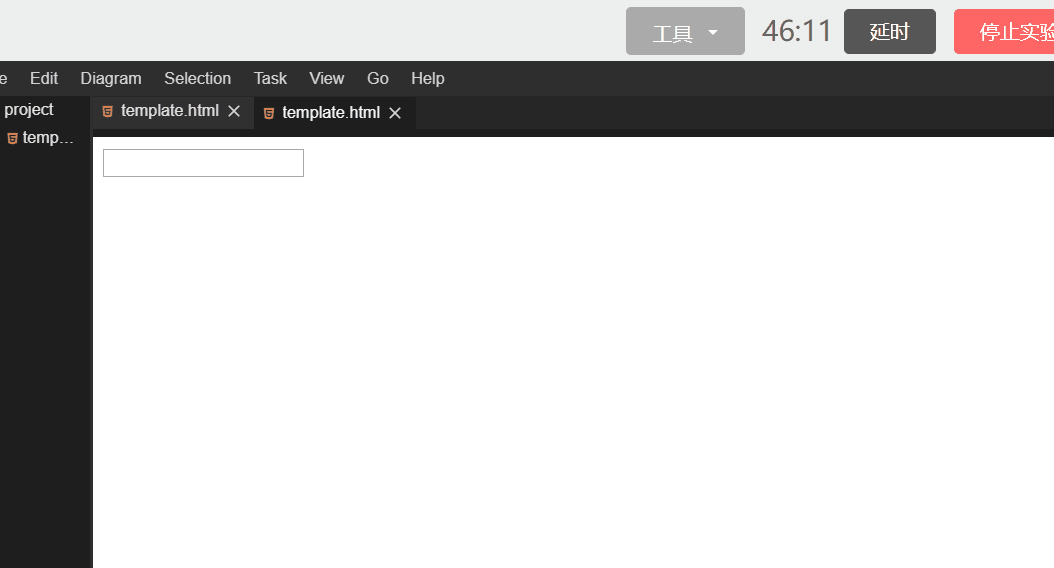
});

</script>

</body>

</html>

运行效果：



#### 系统修饰符

系统特定的组合功能键，如 ctrl+c 、ctrl+v 。

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

* .ctrl
* .alt
* .shift
* .meta

**说明：**在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)。在 Sun 操作系统键盘上，meta 对应实心宝石键 (◆)。在其他特定键盘上，尤其在 MIT 和 Lisp 机器的键盘、以及其后继产品，比如 Knight 键盘、space-cadet 键盘，meta 被标记为“META”。在 Symbolics 键盘上，meta 被标记为“META”或者“Meta”。

例子，同时鼠标左击和按 ctrl 弹出提示：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 同时鼠标左击和按ctrl 弹出提示 -->

<div @click.ctrl="alert('你同时按了鼠标点击和ctrl')">Do something</div>

</div>

<script>

var app = new Vue({

el: '#app',

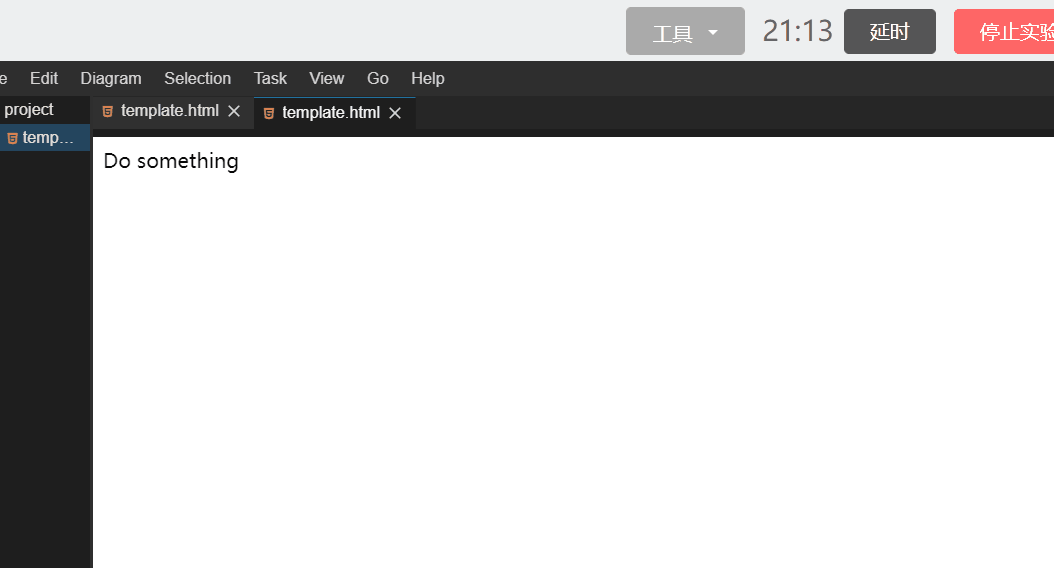
});

</script>

</body>

</html>

运行结果：



#### .exact

.exact 精确按键修饰符，允许你控制由精确的系统修饰符组合触发的事件。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 即使 Alt 或 Shift 被一同按下时也会触发 -->

<button

@click.ctrl="alert('你不单单只按了鼠标左键和 Ctrl键，同时按其他键我也可以触发')"

>

A

</button>

<!-- 有且只有 ctrl键+鼠标左键 被按下的时候才触发 -->

<button @click.ctrl.exact="alert('你只按ctrl键+鼠标左键，才能触发我')">

A

</button>

<!-- 没有任何系统修饰符被按下的时候才触发 -->

<button @click.exact="alert('没有按任何系统修饰符')">A</button>

</div>

<script>

var app = new Vue({

el: '#app',

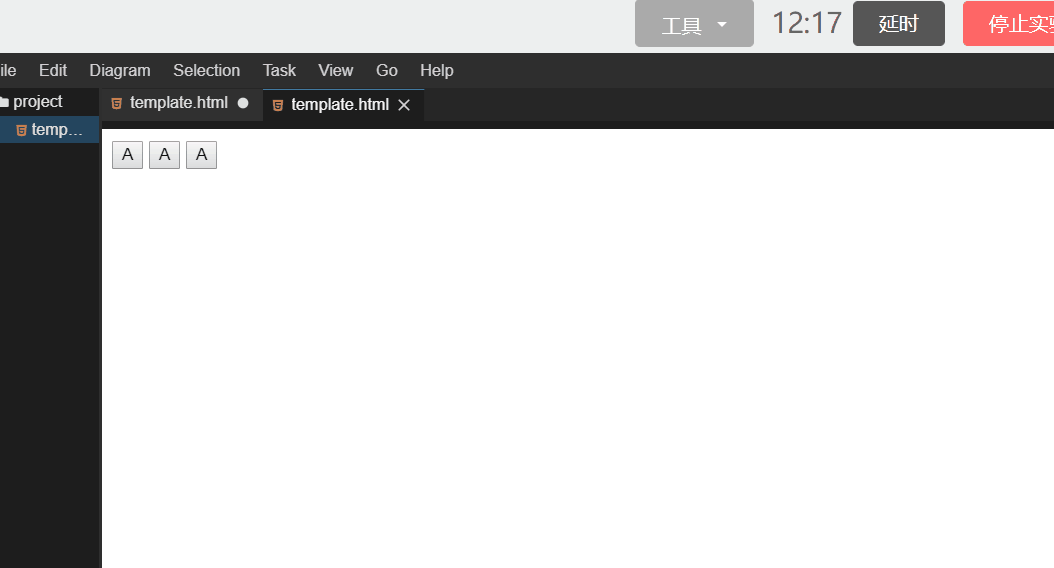
});

</script>

</body>

</html>

运行结果：



#### 鼠标按钮修饰符

这些修饰符会限制处理函数仅响应特定的鼠标按钮。

* .left
* .right
* .middle

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<button @click.left="alert('你按了鼠标左击键')">按钮</button>

<button @click.middle="alert('你按了鼠标滚轮')">按钮</button>

<button @click.right="alert('你按了鼠标右击键')">按钮</button>

</div>

<script>

var app = new Vue({

el: '#app',

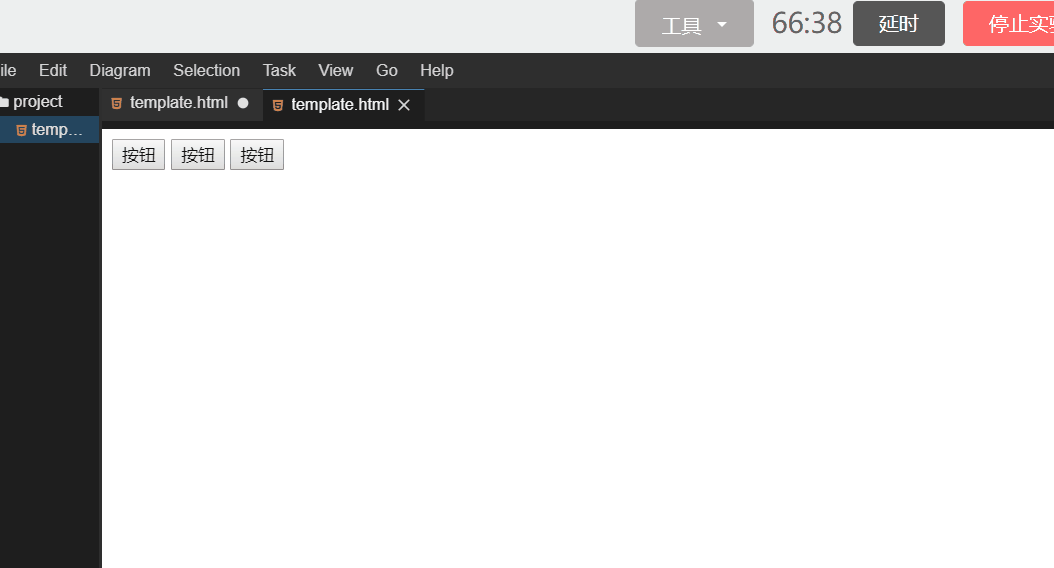
});

</script>

</body>

</html>

运行结果：



#### 综合小练习

小游戏（500px），一直鼠标左键，将小球向右移 500px，拼手速的哦：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

\* {

padding: 0;

margin: 0;

}

#app {

position: relative;

width: 500px;

height: 100px;

background: black;

}

.box {

position: absolute;

left: 0;

top: 0;

width: 100px;

height: 100px;

border-radius: 50%;

background: red;

transition: all 0.3s ease-in;

}

</style>

</head>

<body>

<div id="app" @click.left="changePosition">

<div class="box" :style="{left:left+'px'}"></div>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

left: 0,

},

methods: {

changePosition: function () {

this.left += 25;

if (this.left >= 500) {

alert('走了500px了');

this.left = 0;

}

},

},

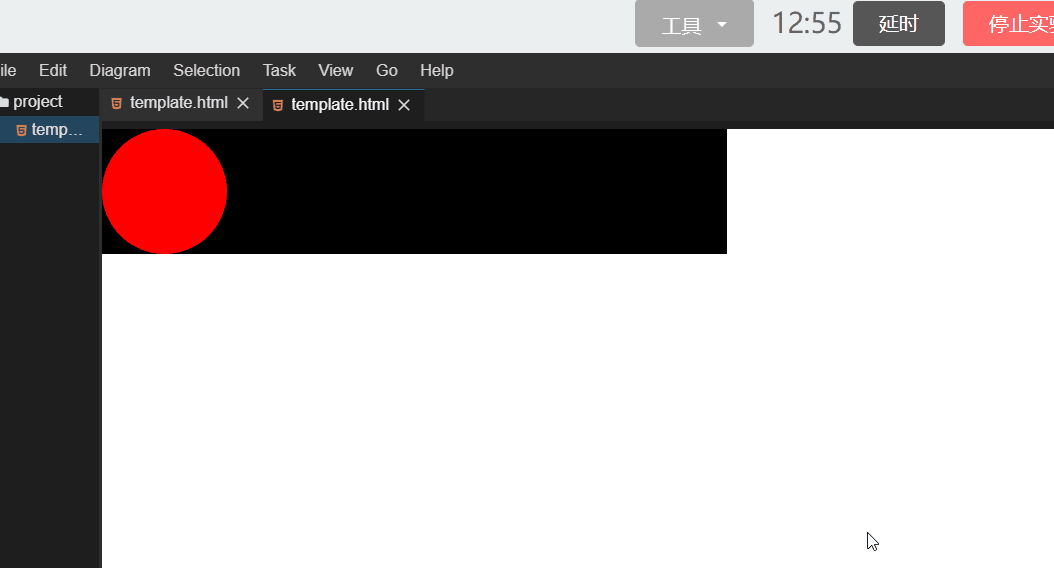
});

</script>

</body>

</html>

运行效果：



## 表单处理

### 基本用法

#### 常用表单元素

| **元素** |  |
| --- | --- |
| input[type=\*] | 文本输入框 |
| textarea | 多行文本 |
| radio | 单选按钮 |
| checkbox | 复选框 |
| select | 选择框 |

#### 注意

**注意一**：v-model  会忽略所有表单元素的  value、checked、selected  特性的初始值而总是将 Vue 实例的数据作为数据来源。直接给元素 value 赋值不会生效的，你应该通过 JavaScript 在组件的  data  选项中声明初始值。

**注意二**：v-model 在内部使用不同的属性为不同的输入元素并抛出不同的事件，具体体现我们在表单 **修饰符小节**，给大家说明：

* text 和 textarea 元素使用 value 属性和 input 事件(内部监听 input 事件)；
* checkbox 和 radio 使用 checked 属性和 change 事件(内部监听 change 事件)；
* select 字段将 value 作为 prop 并将 change 作为事件(内部监听 change 事件)。

**说明：** change 和 input 区别就是，input 实时更新数据，change 不是实时更新。

**input 监听**



**change 监听**



带着注意事项，进入 Vue 表单处理学习。

### 文本

直接为文本绑定v-model 指令，在 data 数据项中声明绑定数据项，可以轻松完成双向数据绑定。让我们来看看神奇的v-model 指令。

例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<input v-model="msg" placeholder="请输入..." />

<p>输入的是: {{ msg }}</p>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

msg: '',

},

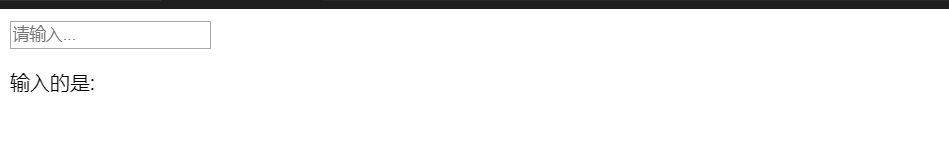
});

</script>

</body>

</html>

运行结果：



### 多行文本

多行文本和文本用法相同，使用v-model 指令，在实例 data 中声明绑定的 message 数据项，即可完成多行文本数据双向绑定

例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<span>Multiline message is:</span>

<p style="white-space: pre-line;">{{ message }}</p>

<br />

<textarea v-model="message" placeholder="add multiple lines"></textarea>

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

message: '',

};

},

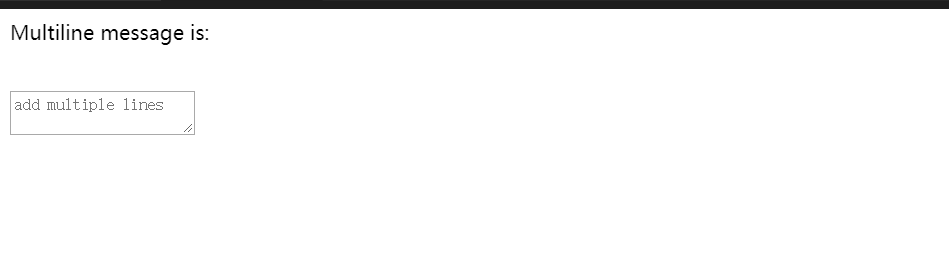
});

</script>

</body>

</html>

运行效果：



### 单选按钮

将单选按钮绑定到同一个 picked，即可完成数据绑定,当第一个单选被选中 picked 的值为第一个单选按钮的 value，同样当第二个单选被选中 picked 的值为第二个单选按钮的 value。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 将单选按钮绑定到同一个picked -->

<input type="radio" id="one" value="One" v-model="picked" />

<label for="one">One</label>

<br />

<input type="radio" id="two" value="Two" v-model="picked" />

<label for="two">Two</label>

<br />

<span>Picked: {{ picked }}</span>

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

picked: '',

};

},

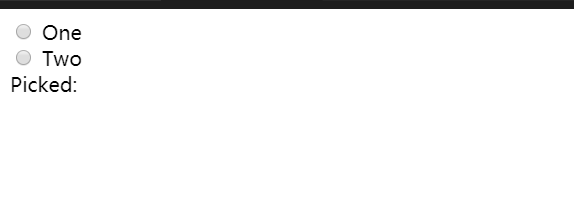
});

</script>

</body>

</html>

运行效果：



### 复选框

复选框绑定的是一个布尔值（true or false）,同样在复选框元素上使用v-model 指令，在实例 data 中声明 checked,即可完成复选框数据的双向绑定

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<input type="checkbox" id="checkbox" v-model="checked" />

<label for="checkbox">{{ checked }}</label>

</div>

<script>

//绑定布尔值

var vue = new Vue({

el: '#app',

data() {

return {

checked: false,

};

},

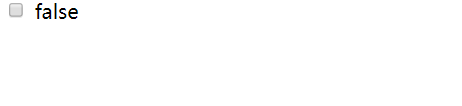
});

</script>

</body>

</html>

运行结果：



### 多个复选框

通常开发中，会用到多个复选框，那么我们就需要把复选框，绑定到同一个数组，方便数据收集。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<input type="checkbox" id="syl1" value="syl1" v-model="checkedNames" />

<label for="syl1">syl1</label>

<input type="checkbox" id="syl2" value="syl2" v-model="checkedNames" />

<label for="syl2">syl2</label>

<input type="checkbox" id="syl3" value="syl3" v-model="checkedNames" />

<label for="syl3">syl3</label>

<br />

<span>Checked names: {{ checkedNames }}</span>

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

checkedNames: [],

};

},

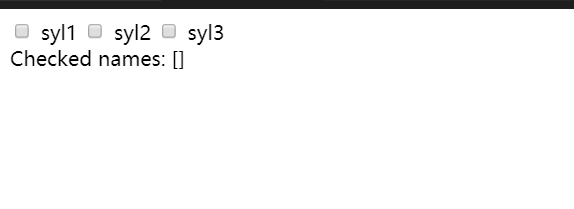
});

</script>

</body>

</html>

运行结果：



### 选择框

在 select 元素上使用v-model 指令，可以绑定当前选中的 option。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- select标签是绑定 数据项 selected -->

<select v-model="selected">

<option disabled value="">请选择</option>

<option>A</option>

<option>B</option>

<option>C</option>

</select>

<span>Selected: {{ selected }}</span>

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

selected: '',

};

},

});

</script>

</body>

</html>

运行结果：



### 值绑定

#### 文本和多行文本值绑定

直接在标签上指定 value 是不能成功的，会被 Vue 忽略

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 直接给value不会生效 -->

<input v-model="msg" value="hello" />

</div>

<script>

var app = new Vue({

el: '#app',

data: {

msg: '',

},

});

</script>

</body>

</html>

运行结果：



正确模式，通过 JavaScript 在组件的 data 选项中声明初始值：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>syl-vue-test</title>

<!-- 引入 vue.js -->

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 直接给value不会生效 -->

<input v-model="msg" />

<br />

<br />

<br />

<textarea v-model="msg"></textarea>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

msg: 'hello',

},

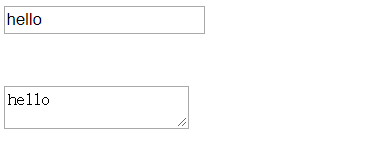
});

</script>

</body>

</html>

运行结果：



#### 单选按钮

checkbox 和 radio 使用 checked 属性，所以直接给元素 value 值，当选中时 data 中声明的绑定项的值就为元素 value。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 当选中时，`picked` 为字符串 "a" -->

<input type="radio" v-model="picked" value="a" />

{{picked}}

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

picked: '',

};

},

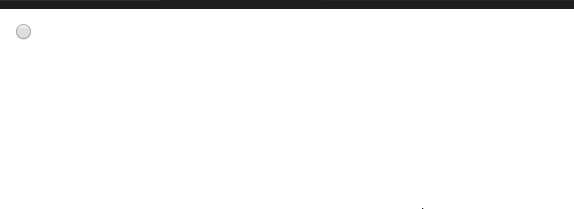
});

</script>

</body>

</html>

运行结果：



#### 复选框

复选框可以是布尔值，toggle 为 true 或 false，当选中时，toggle 为 true,未选中时，toggle 为 false

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 当复选框选中时，toggle的值为true，未选中为false -->

<input type="checkbox" v-model="toggle" />

{{toggle}}

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

toggle: false,

};

},

});

</script>

</body>

</html>



运行结果：



在日常开发中，复选框的值很多情况是特定的值，那么我们可以这样做，在标签中声明 true-value="yes" 和 false-value="no"这两个属性，当选中时就是 true-value 属性指定的值，当未选中时就是 false-value 属性值。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<input

type="checkbox"

v-model="toggle"

true-value="yes"

false-value="no"

/>

<p>toggle:{{toggle}}</p>

</div>

<script>

//通过true-value="yes" false-value="no"属性控制，选中时toggle值为yes,未选中时为no

var vue = new Vue({

el: '#app',

data() {

return {

toggle: '',

};

},

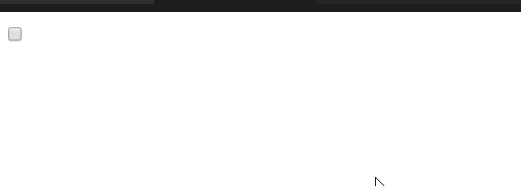
});

</script>

</body>

</html>

运行结果：



如果你只想要选中有值，你可以这样做，true-value 属性指定值，false-value 属性设为空值

<input type="checkbox" v-model="toggle" true-value="name" false-value="" />

#### 选择框

选择框的值绑定，直接指定每个option的value，可以是固定的，也可以是使用 v-bind:value 动态绑定的:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 当选中第一个选项时，`selected` 为字符串 "abc" -->

<select v-model="selected">

<!-- 固定赋值value -->

<option value="abc">ABC</option>

<!-- 使用 v-bind 绑定值 -->

<option v-bind:value="optionValue">DEF</option>

</select>

<p>{{selected}}</p>

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

selected: '',

//第二个option 的值

optionValue: 'efg',

};

},

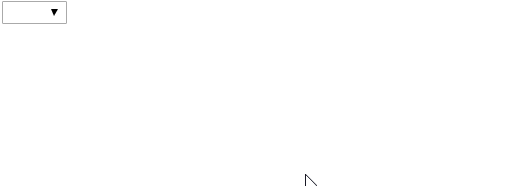
});

</script>

</body>

</html>

运行效果：



### 修饰符

#### .lazy

开始介绍表单处理时，我们说了几点注意，不同的元素，使用的值不同，抛出的事件也不同。可能开发中，我们不需要数据实时更新，那么，我们怎么将 input 事件与 change 事件替换，可以使用 **.lazy** 修饰符，可以将抛出事件由 input 改为 change ,使表单元素惰性更新，不实时更新。

例子:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!--使用 .lazy 修饰符将文本框 抛出的事件改为 change 事件，不再实时更新，只有文本框失去焦点才更新数据 惰性更新 -->

<input v-model.lazy="msg" />

<p>{{msg}}</p>

</div>

<script>

var vue = new Vue({

el: '#app',

data: {

msg: 'hello',

},

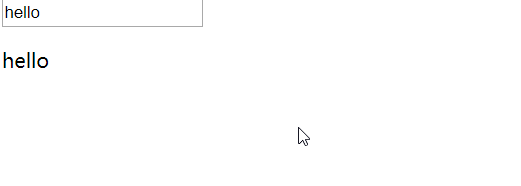
});

</script>

</body>

</html>

使用 .lazy 运行效果:



#### .number

如果想自动将用户的输入值转为数值类型，可以给  v-model  添加  number  修饰符：

这通常很有用，因为即使在  type="number"  时，HTML 输入元素的值返回字符串（默认），需要自己进行类型转换。如果这个值无法被  parseFloat()  解析，则会返回原始的值。 给  v-model  添加  number  修饰符,用户即使输入的是非数值类型，也会进行转换，无法转换时，会返回原始的。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<p>没有使用 .number 修饰符</p>

<input v-model="number1" type="number" />

<!-- 使用typeof对值类型检测 -->

<p>{{typeof(number1)}}</p>

<p>使用 .number 修饰符</p>

<input v-model.number="number2" type="number" />

<!-- 使用typeof对值类型检测 -->

<p>{{typeof(number2)}}</p>

</div>

<script>

var vue = new Vue({

el: '#app',

data: {

number1: '',

number2: '',

},

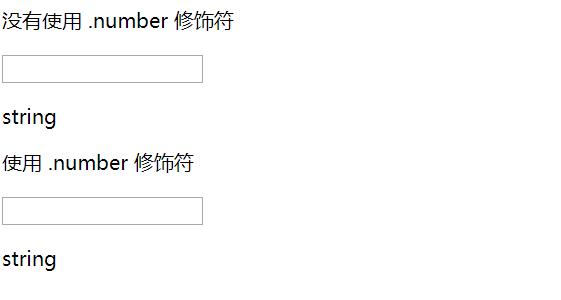
});

</script>

</body>

</html>

运行结果：



#### .trim

表单元素值首尾空格，自动过滤。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<input v-model.trim="msg" type="text" />

<p>首尾空格被过滤了：{{msg}}</p>

</div>

<script>

var vue = new Vue({

el: '#app',

data: {

msg: '',

},

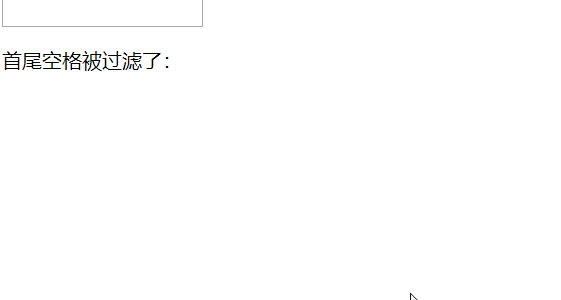
});

</script>

</body>

</html>

运行结果：



### 综合小练习

表单数据收集与绑定

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

\* {

padding: 0;

margin: 0;

}

html,

body {

width: 100%;

height: 100%;

overflow: hidden;

}

#app {

position: absolute;

top: 0;

left: 0;

bottom: 0;

right: 0;

margin: auto;

width: 400px;

height: 400px;

}

</style>

</head>

<body>

<div id="app">

<form class="app-form">

<span>name:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</span>

<input type="text" v-model="username" />

<br />

<span>password:</span><input type="password" v-model="password" />

<br />

<span

>sex:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</span

>

<input type="radio" id="man" value="man" v-model="sex" />

<label for="man">man</label>

<input type="radio" id="woman" value="woman" v-model="sex" />

<label for="woman">women</label>

<br />

<span>hobby:</span>

<input type="checkbox" id="game" value="game" v-model="hobby" />

<label for="game">game</label>

<input

type="checkbox"

id="basketball"

value="basketball"

v-model="hobby"

/>

<label for="basketball">basketball</label>

<input type="checkbox" id="study" value="study" v-model="hobby" />

<label for="study">study</label>

<br />

<br />

<p>名字：{{username}}</p>

<p>密码：{{password}}</p>

<p>性别:{{sex}}</p>

<p>爱好：{{hobby}}</p>

</form>

</div>

<script>

var vue = new Vue({

el: '#app',

data() {

return {

username: '',

password: '',

sex: 'man', //性别单选默认勾选男

hobby: [],

};

},

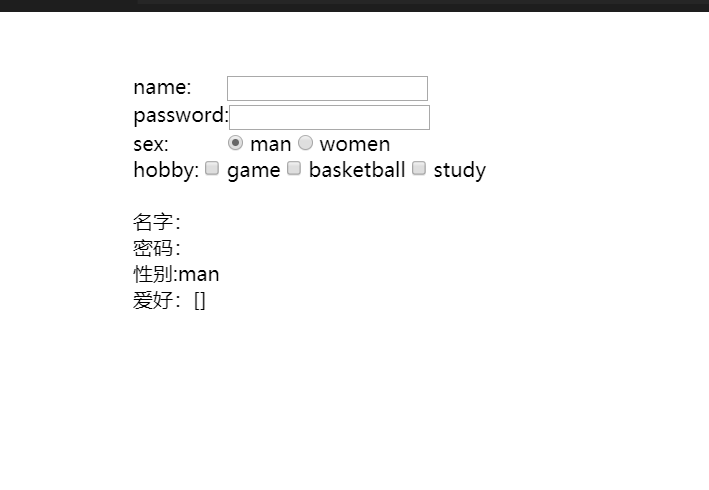
});

</script>

</body>

</html>

运行效果：



## 组件

### 组件注册

组件分为全局组件和局部组件。

#### 全局注册

Vue.component()方法注册全局组件

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<syl></syl>

<syl></syl>

<syl></syl>

</div>

<script>

//Vue.component(组件名字,template:{元素标签})

Vue.component('syl', {

template: '<h1>实验楼全局组件</h1>',

});

var app = new Vue({

el: '#app',

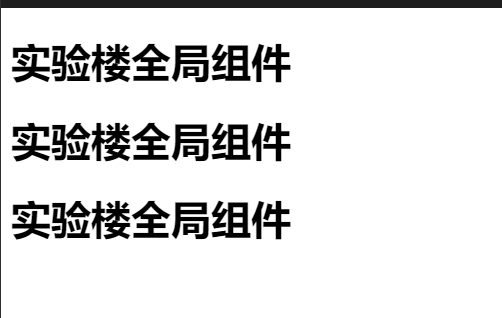
});

</script>

</body>

</html>

运行结果:



#### 局部组件

在父级 components 对象中声明，局部组件只有它的父级才能调用

var header = new Vue({

el: '#header',

//子组件必须声明后使用，不然不能起效

components: {

'syl-header': childComponent,

},

});

局部组件使用：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="header">

<syl-header></syl-header>

</div>

<div id="mid">

<syl-mid></syl-mid>

</div>

<script>

//头部组件

var childComponent = {

template: '<h2>我是实验楼局部组件header，只有我们父级才能调用</h2>',

};

//中间部分组件

var childComponent2 = {

template: '<h2>我是实验楼局部组件mid，只有我们父级才能调用</h2>',

};

//header vm

var header = new Vue({

el: '#header',

//子组件必须声明后使用，不然不能起效

components: {

'syl-header': childComponent,

},

});

var mid = new Vue({

el: '#mid',

//子组件必须声明后使用，不然不能起效

components: {

'syl-mid': childComponent2,

},

});

</script>

</body>

</html>

运行效果：



#### 组件复用

组件的优点就在于能够复用，一次代码编写，整个项目受用。

**注意：** 复用组件内的 data 必须是一个函数，如果是一个对象（引用类型），组件与组件间会相互影响，组件数据不能独立管理。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<button-counter></button-counter>

<button-counter></button-counter>

<button-counter></button-counter>

</div>

<script>

//注册一个全局可复用组件

Vue.component('button-counter', {

//data必须是一个函数不然会影响其他组件

data() {

return {

counter: 0,

};

},

template: '<button @click="counter++">{{counter}}</button>',

});

var app = new Vue({

el: '#app',

});

</script>

</body>

</html>

运行效果：



data 使用对象，运行结果：



### 组件间通信

#### 父子组件之 props

props 是一个单向的数据流，只允许父组件向子组件传值，值类型可以是一个数值、字符、布尔值、数值、对象,子组件需要显式地用 props 选项声明 "prop"。

**注意**：HTML 中的特性名是大小写不敏感的，所以浏览器会把所有大写字符解释为小写字符。这意味着当你使用 DOM 中的模板时，camelCase (驼峰命名法) 的 prop 需要使用其等价的 kebab-case (短横线分隔命名) 命名，如下面例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<title-component post-title="syl1"></title-component>

<title-component post-title="syl2"></title-component>

<title-component post-title="syl3"></title-component>

</div>

<script>

//注册一个 title组件,通过传入不同的title值,渲染不同的东西

//组件上 传递的 props 属性名为 kebab-case（短横线分隔命名)的要转换为驼峰命名

Vue.component('title-component', {

props: ['postTitle'], //post-title 转换为驼峰命名

template: '<p>{{postTitle}}</p>',

});

var app = new Vue({

el: '#app',

});

</script>

</body>

</html>

运行效果：



#### props 类型检测

到这里，我们只看到了以字符串数组形式列出的 prop：

props:['postTitle']

但是，通常你希望每个 prop 都有指定的值类型。这时，你可以以对象形式列出 prop，这些属性的名称和值分别是 prop 各自的名称和类型：

props:{

title:String,

id:Number,

content:String

}



例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<child-component

id="1"

title="hello syl"

content="you are welcom"

></child-component>

</div>

<script>

//注册一个子组件

Vue.component('child-component', {

//props 对象形式，传递属性值 进行类型检测，在脚手架环境中很有用

props: {

id: Number,

title: String,

content: String,

},

//使用`` es6模板字符串书写格式更优美

template: `<div><p>id:{{id}}</p><p>title:{{title}}</p><p>content:{{content}}</p></div>`,

});

var app = new Vue({

el: '#app',

});

</script>

</body>

</html>

#### 子父组件通信之 emit

上面提到 props 实现父向子组件传递数据是单向流的，那么，如何实现子组件向父组件通信呢？这里要使用自定义事件 emit 方法，通过自定义事件来由下到上的数据流动。

大致语法

this.$emit('自定义事件名'，参数)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<child-component v-on:send-msg="getMsg"></child-component>

</div>

<script>

//定义一个子组件，template绑定click 事件

//当click事件触发就使用 emit 自定义一个事件send-msg，传入参数 “我是子组件请求与你通信”

//$emit('send-msg','我是子组件请求与你通信')

//子组件标签上绑定自定义事件 send-msg，并绑定上父级的方法 getMsg，即可完成了子父组件通信

//<child-component v-on:send-msg="getMsg"></child-component>

Vue.component('child-component', {

template: `

<button v-on:click="$emit('send-msg','我是子组件请求与你通信')">

Click me

</button>

`,

});

var app = new Vue({

el: '#app',

methods: {

getMsg: function (msg) {

//弹出子组件传递的信息

alert(msg);

},

},

});

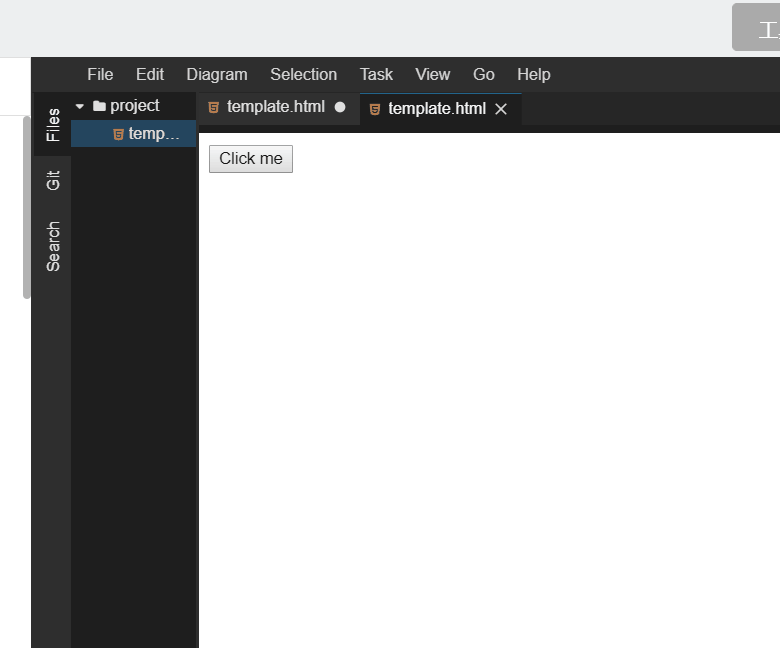
</script>

</body>

</html>



运行结果：



子组件向父组件数据传递套路：

第一步：子组件绑定事件。

第二步：子组件绑定事件触发，使用 $emit 创建自定义事件并传入需要传值给父组件的数据。

第三步：在子组件标签上 用 v-on 绑定自定义事件，在父组件中声明自定义事件处理的方法。

第四步：父组件方法，接受自定义事件传的参数，就完成了整个由下到上的数据流。

#### 动态组件

上面例子我们传值都是直接传的固定值，其实动态传值我们也支持，生成动态组件，使用 v-bind 动态绑定 props 值。

例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- 使用v-bind简写模式 动态绑定 props 值 -->

<child-component

:name="name"

:age="age"

:height="height"

></child-component>

<child-component

:name="name+'2'"

:age="age+1"

:height="height"

></child-component>

</div>

<script>

//定义一个子组件

Vue.component('child-component', {

//使用属性类型检测

props: {

name: String,

age: Number,

height: String,

},

template: `

<ul>

<li>{{name}}</li>

<li>{{age}}</li>

<li>{{height}}</li>

</ul>

`,

});

var app = new Vue({

el: '#app',

data() {

return {

name: 'syl',

age: 20,

height: '180cm',

};

},

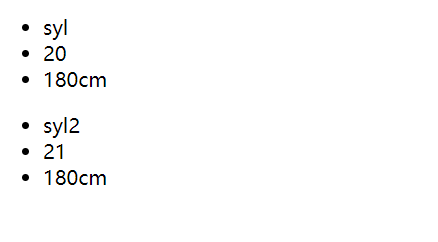
});

</script>

</body>

</html>

运行结果：



#### 生命周期函数

接触了组件后，应该需要了解整个实例的生命周期是怎么样的，有哪些钩子函数，在哪个阶段我们能操作什么，实例在不同的阶段都会抛出不同的钩子函数，方便开发者能够精确的控制整个流程。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<button @click="handleClick">{{name}}</button>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

name: 'syl',

};

},

methods: {

handleClick: function () {

this.name = 'syl syl';

},

},

beforeCreate() {

alert(

'在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用'

);

},

created() {

alert(

'在实例创建完成后被立即调用,挂载阶段还没开始，$el 属性目前不可见'

);

},

beforeMount() {

alert('在挂载开始之前被调用：相关的 render 函数首次被调用');

},

mounted() {

alert('el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子');

},

beforeUpdate() {

alert('数据更新时调用');

},

updated() {

alert('组件 DOM 已经更新');

},

beforeDestroy() {},

destroyed() {},

});

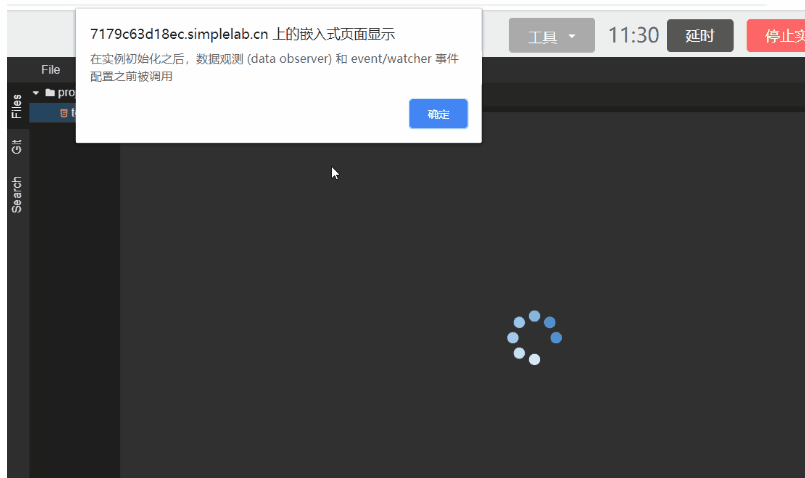
</script>

</body>

</html>



注意弹窗与数据变化，运行结果：



这么多钩子函数我们经常主要用到有：

1.created 钩子函数内我们可以进行异步数据请求

created() {

fetch('url')

.then(function(response) {

console.log(response)

})

}



2.mounted 我们可以直接操作元素 DOM 了 ,但是并不推荐这样做，不利于性能提升。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

</head>

<body>

<div id="app">

<div id="box" style="width:40px;background: tomato;">点击</div>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {};

},

//div#box 开始并没有绑定事件，挂载后我们直接操作 原生 dom

mounted() {

var box = document.querySelector('#box');

box.addEventListener('click', function () {

alert('我们挂载后 原生点击事件');

});

},

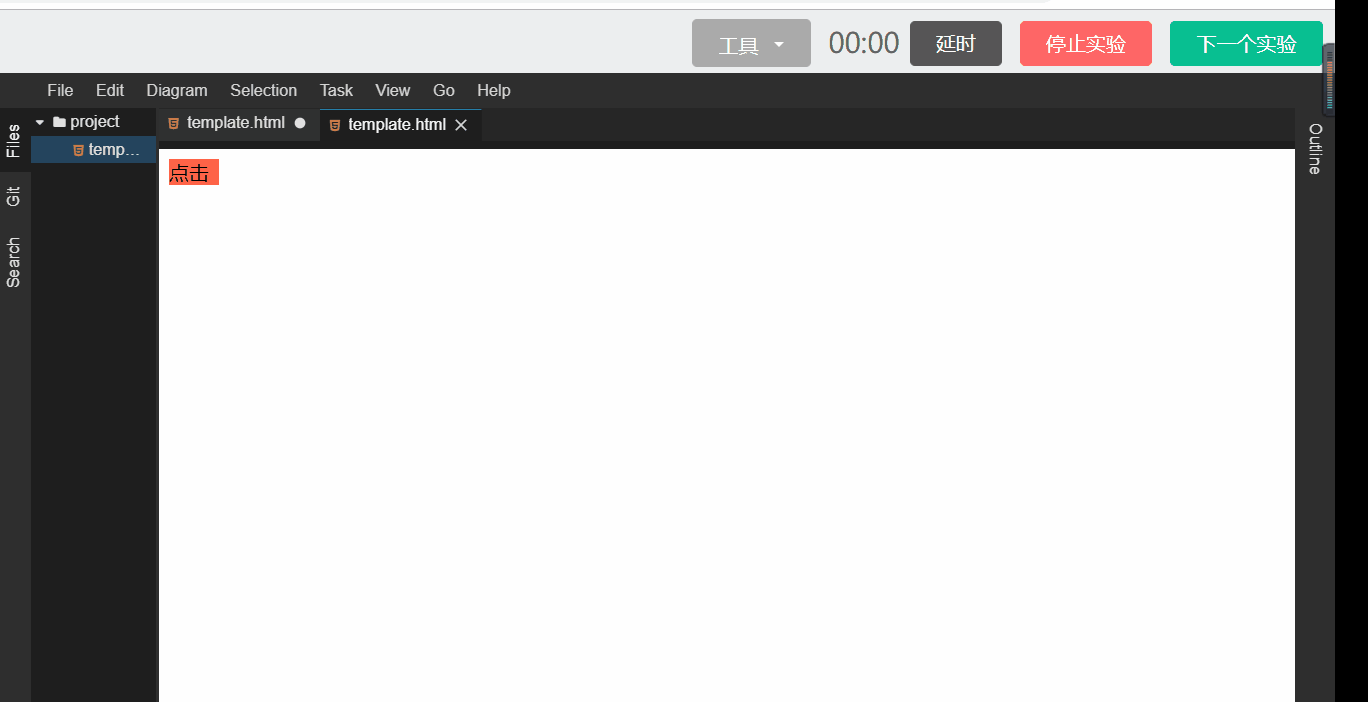
});

</script>

</body>

</html>

运行效果：



## 过渡与动画

### 过渡的 class

基本示例：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

/\* 定义进入和离开过渡生效时的状态 \*/

.fade-enter-active,

.fade-leave-active {

transition: opacity 0.5s;

}

/\* 定义过渡的开始和结束状态 \*/

.fade-enter,

.fade-leave-to {

opacity: 0;

}

</style>

</head>

<body>

<div id="app">

<!-- 点击显示隐藏 切换 按钮 改变实例 show数据的值 ！=== 取反 -->

<button v-on:click="show = !show">

Toggle

</button>

<!-- transition ’见名知意‘ 这是一个 Vue 内置封装的 动画组件 他可以指定被他包囊 元素的交互效果 -->

<!-- name 就是动画过渡名字 fade 这个是内置动画过渡名-->

<transition name="fade">

<!-- v-if 如果show 为true 就渲染 -->

<p v-if="show">hello syl</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: true,

};

},

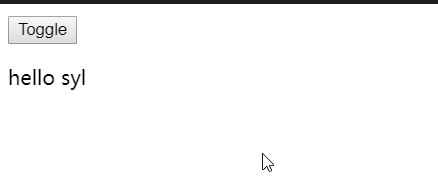
});

</script>

</body>

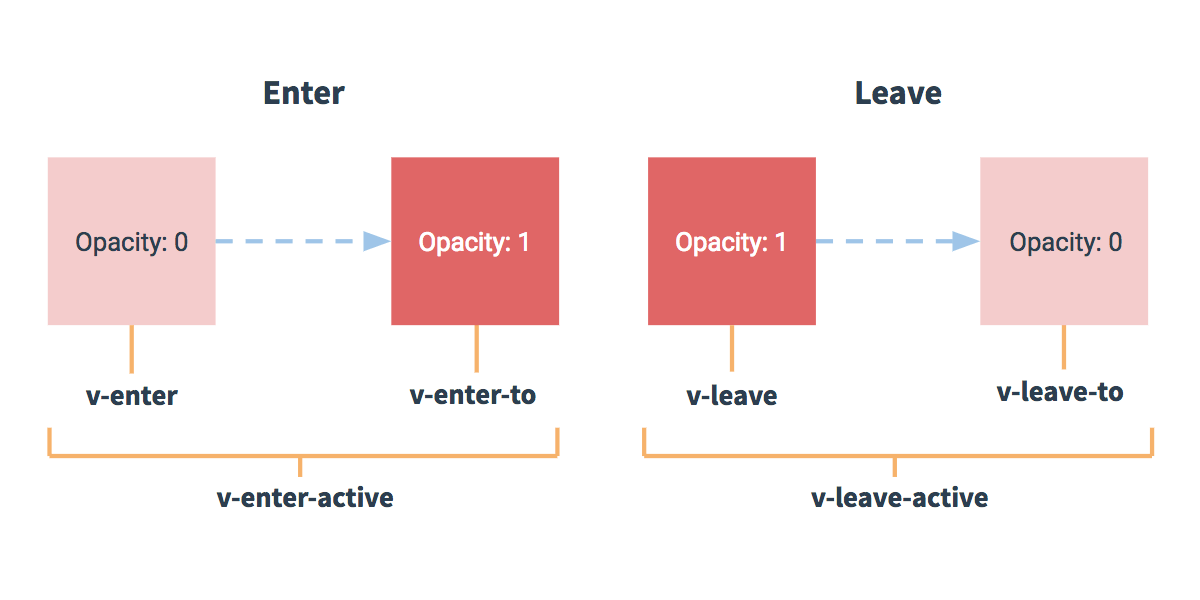
</html>

运行效果：



上面看了一个基础示例，我们是可以通过特定的 css 来控制动画或者过渡的全过程，我们来学习一下过渡的类名

1. v-enter：定义进入**过渡的开始状态**。在元素被插入之前生效，在元素被插入之后的下一帧移除。
2. v-enter-active：定义进入**过渡生效时的状态**。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡/动画完成之后移除。这个类可以被用来定义进入**过渡的过程时间，延迟和曲线函数**。
3. v-enter-to: 定义进入**过渡的结束状态**。在元素被插入之后下一帧生效 (与此同时 v-enter 被移除)，在过渡/动画完成之后移除。
4. v-leave: 定义**离开过渡的开始状态**。在离开过渡被触发时立刻生效，下一帧被移除。
5. v-leave-active：定义**离开过渡生效时的状态**。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在过渡/动画完成之后移除。这个类可以被用来定义**离开过渡的过程时间，延迟和曲线函数。**
6. v-leave-to: 定义**离开过渡的结束状态**。在离开过渡被触发之后下一帧生效 (与此同时 v-leave 被删除)，在过渡/动画完成之后移除。



一般套路，可以根据自己需求自由组合定制：

.v-enter,

.v-leave-to {

/\* 定义元素默认状态 例如：opacity:0\*/

}

.v-enter-to,

.v-leave {

/\* 定义元素激活时状态 例如：opacity:1\*/

}

.v-enter-active,

.v-leave-active {

/\* 定义过渡或动画，在过渡中的状态 \*/

/\*

最常用的就是在这里面指定过渡动画 时间/延迟/曲线函数

transition:opacity:1s 1s ease-in-out

\*/

}

### CSS 过渡

上面其实就是一个 CSS 过渡实例，根据上面介绍的过渡类名和一般套路，自己来写一个过渡。

打造 **快进缓出** 过渡效果：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

.my-transition-enter,

.my-transition-leave-to {

/\* 定义元素默认状态 例如：opacity:0 \*/

opacity: 0;

}

.my-transition-enter-to,

.my-transition-leave {

/\* 定义元素激活时状态 例如：opacity:1\*/

opacity: 1;

}

.my-transition-enter-active {

/\* 默认到激活状态过渡，0.5s 快速进入 \*/

transition: opacity 0.5s;

}

.my-transition-leave-active {

/\* 激活到默认状态过渡 2s 缓出过渡打造 \*/

transition: opacity 2s;

}

</style>

</head>

<body>

<div id="app">

<!-- 点击显示隐藏 切换 按钮 改变实例 show数据的值 '取反' -->

<button v-on:click="show = !show">

快进缓出过渡

</button>

<!-- transition ’见名知意‘ 这是一个 Vue 内置封装的 动画组件 他可以指定被他包囊 元素的交互效果 -->

<!--自定义一个 my-transition 过渡-->

<transition name="my-transition">

<!-- v-if 如果show 为true 就渲染 -->

<p v-if="show">hello syl</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: false,

};

},

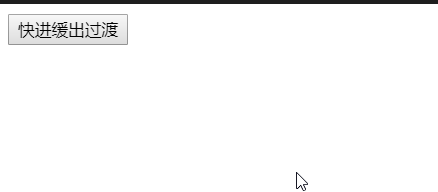
});

</script>

</body>

</html>

运行效果：



### CSS 动画

开发中不仅仅只需要过渡效果，往往是两者的结合，CSS 动画用法同 CSS 过渡一样。如果对 CSS3 动画不了解的可以进入 CSS 基础课程学习一下。

放大缩小动画：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

#app {

width: 200px;

height: 200px;

overflow: hidden;

}

.bounce-enter-active {

animation: bounce-in 0.5s;

}

.bounce-leave-active {

animation: bounce-in 0.5s reverse;

}

@keyframes bounce-in {

0% {

transform: scale(0);

}

50% {

transform: scale(1.5);

}

100% {

transform: scale(1);

}

}

</style>

</head>

<body>

<div id="app">

<button @click="show = !show">css动画</button>

<transition name="bounce">

<p v-if="show">hello syl hello syl hello syl</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: true,

};

},

});

</script>

</body>

</html>

运行效果：



### 自定义过渡类名

我们可以通过以下特性来自定义过渡类名：

* enter-class
* enter-active-class
* enter-to-class
* leave-class
* leave-active-class
* leave-to-class

这对于 Vue 的过渡系统和其他第三方 **CSS 动画库**，如 [Animate.css](https://daneden.github.io/animate.css/) 结合使用十分有用。

例子，Vue 结合 [Animate.css](https://daneden.github.io/animate.css/) ：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<!-- 引入animate.css -->

<link

href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1"

rel="stylesheet"

type="text/css"

/>

</head>

<body>

<div id="app">

<button @click="show = !show">

Toggle render

</button>

<!-- 自定义过渡的类名 animate.css动画库的类名 -->

<transition

name="custom-classes-transition"

enter-active-class="animated tada"

leave-active-class="animated bounceOutRight"

>

<p v-if="show">hello</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: true,

};

},

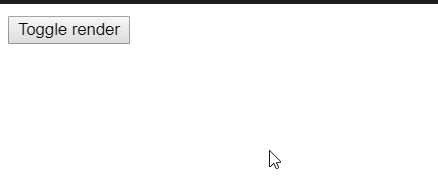
});

</script>

</body>

</html>

运行结果：



### 同时使用过渡与动画

Vue 为了知道过渡的完成，必须设置相应的事件监听器。它可以是 transitionend 或 animationend ，这取决于给元素应用的 CSS 规则。如果你使用其中任何一种，Vue 能自动识别类型并设置监听。

但是，在一些场景中，你需要给同一个元素同时设置两种过渡动效，比如 animation很快的被触发并完成了，而 transition 效果还没结束。在这种情况下，你就需要使用 type 特性并设置 animation 或 transition 来明确声明你需要 Vue 监听的类型。

显性过渡时间

我们可以编排一系列过渡效果，其中一些嵌套的内部元素相比于过渡效果的根元素有延迟的或更长的过渡效果。

<transition> 组件上的 duration 属性定制一个显性的过渡持续时间 (以毫秒计)，形成先后顺序的动画队列。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

#app {

width: 200px;

height: 200px;

overflow: hidden;

}

.bounce-enter-active {

animation: bounce-in 0.5s;

}

.bounce-leave-active {

animation: bounce-in 0.5s reverse;

}

@keyframes bounce-in {

0% {

transform: scale(0);

}

50% {

transform: scale(1.5);

}

100% {

transform: scale(1);

}

}

</style>

</head>

<body>

<div id="app">

<button @click="show = !show">动画队列</button>

<transition name="bounce" :duration="2000">

<p v-if="show">hello syl hello syl hello syl</p>

</transition>

<transition name="bounce" :duration="1500">

<p v-if="show">hello syl hello syl hello syl</p>

</transition>

<transition name="bounce" :duration="1000">

<p v-if="show">hello syl hello syl hello syl</p>

</transition>

<transition name="bounce" :duration="500">

<p v-if="show">hello syl hello syl hello syl</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: true,

};

},

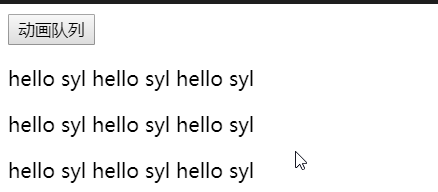
});

</script>

</body>

</html>

运行效果：



### JavaScript 动画钩子

使用动画钩子你更能掌控动画的全过程，更好的与 **JavaScript 动画库** 结合。

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

v-on:enter-cancelled="enterCancelled"

v-on:before-leave="beforeLeave"

v-on:leave="leave"

v-on:after-leave="afterLeave"

v-on:leave-cancelled="leaveCancelled"

>

</transition>



前面四个进入动画钩子，后面四个离开动画钩子

// ...

methods: {

// --------

// 进入中

// --------

beforeEnter: function (el) {

// ...

},

// 当与 CSS 结合使用时

// 回调函数 done 是可选的

enter: function (el, done) {

// ...

done()

},

afterEnter: function (el) {

// ...

},

enterCancelled: function (el) {

// ...

},

// --------

// 离开时

// --------

beforeLeave: function (el) {

// ...

},

// 当与 CSS 结合使用时

// 回调函数 done 是可选的

leave: function (el, done) {

// ...

done()

},

afterLeave: function (el) {

// ...

},

// leaveCancelled 只用于 v-show 中

leaveCancelled: function (el) {

// ...

}

}



示例使用 velocity.js 动画库和动画钩子函数，打造精细动画：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<!-- velocity.js 引入 -->

<script src="https://labfile.oss.aliyuncs.com/courses/1379/velocity.min.js"></script>

</head>

<body>

<div id="app">

<button @click="show = !show">

Toggle

</button>

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:leave="leave"

v-bind:css="false"

>

<p v-if="show">

Demo

</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: false,

};

},

methods: {

beforeEnter: function (el) {

el.style.opacity = 0;

el.style.transformOrigin = 'left';

},

enter: function (el, done) {

Velocity(

el,

{

opacity: 1,

fontSize: '1.4em',

},

{

duration: 300,

}

);

Velocity(

el,

{

fontSize: '1em',

},

{

complete: done,

}

);

},

leave: function (el, done) {

Velocity(

el,

{

translateX: '15px',

rotateZ: '50deg',

},

{

duration: 600,

}

);

Velocity(

el,

{

rotateZ: '100deg',

},

{

loop: 2,

}

);

Velocity(

el,

{

rotateZ: '45deg',

translateY: '30px',

translateX: '30px',

opacity: 0,

},

{

complete: done,

}

);

},

},

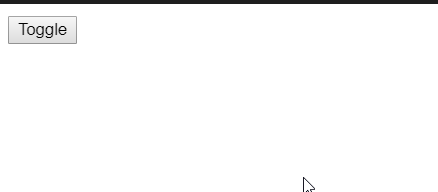
});

</script>

</body>

</html>

运行效果：



### 初始化过渡

我们想要页面初次渲染就有一个过渡效果，可以通过 appear 特性设置节点在初始渲染的过渡。 使用 F5 刷新查看效果：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

#app {

width: 200px;

height: 200px;

overflow: hidden;

}

/\* 自定义初始化css \*/

.bounce-enter-active,

.bounce-appear-active-class {

animation: bounce-in 0.5s;

}

@keyframes bounce-in {

0% {

transform: scale(0);

}

50% {

transform: scale(1.5);

}

100% {

transform: scale(1);

}

}

</style>

</head>

<body>

<div id="app">

<!-- 自定义初始化渲染过渡和动画 -->

<transition

appear

appear-class="bounce-appear-class"

appear-to-class="bounce-appear-to-class"

appear-active-class="bounce-appear-active-class"

>

<p>可以通过 appear 特性设置节点在初始渲染的过渡</p>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

show: true,

};

},

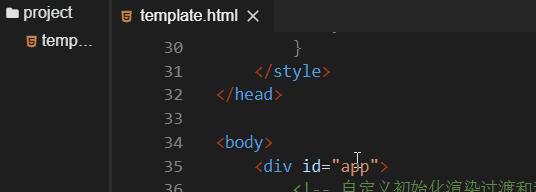
});

</script>

</body>

</html>

运行结果：



**注意：** 和动画钩子一样，该特性也提供钩子函数。

<transition

appear

v-on:before-appear="customBeforeAppearHook"

v-on:appear="customAppearHook"

v-on:after-appear="customAfterAppearHook"

v-on:appear-cancelled="customAppearCancelledHook"

>

</transition>

### 多个元素过渡

多个元素过渡，可以使用 v-if/v-else，确定那个元素渲染，但是注意为相同标签的元素绑定 key 值，提高性能。

丝滑开关切换例子：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<title>vue</title>

<script src="https://labfile.oss.aliyuncs.com/courses/1262/vue.min.js"></script>

<style>

#app {

position: relative;

height: 18px;

}

button {

position: absolute;

}

.fade-enter-active,

.fade-leave-active {

transition: all 1s;

}

.fade-enter,

.fade-leave-active {

opacity: 0;

}

.fade-enter {

transform: translateX(31px);

}

.fade-leave-active {

transform: translateX(-31px);

}

</style>

</head>

<body>

<div id="app">

<transition name="fade">

<button v-if="docState === 'saved'" key="saved" @click="handleClick">

Edit

</button>

<button v-if="docState === 'edited'" key="edited" @click="handleClick">

Save

</button>

</transition>

</div>

<script>

var app = new Vue({

el: '#app',

data() {

return {

docState: 'saved',

};

},

//点击切换docState值

methods: {

handleClick: function () {

if (this.docState == 'saved') {

this.docState = 'edited';

} else {

this.docState = 'saved';

}

},

},

});

</script>

</body>

</html>

运行效果：



## Vue 进阶引导

### 工程化与组件化开发引导

Vue 的应用场景很多，很大部分用它来开发 SPA (single page web application)，构建单页应用，与 React 和 Angular 一样都有自己的脚手架开发工具，Vue-cli 是 Vue 官方脚手架工具，使用 webpack 进行模块打包，实现工程化与组件化开发。如果你对前端模块化没有一点了解，可以去先学习一下 ES6 模块规范，在项目中会大量用到模块的引入或导出，如下图。

import Vue from 'vue';

import App from './App';

import router from './router';

new Vue({

el: '#app',

router,

components: { App },

template: '<App/>',

});



vue 脚手架官网与 webpack 官网：

* [vue-cli](https://cli.vuejs.org/zh/guide/cli-service.html)
* [webpack](https://www.webpackjs.com/)

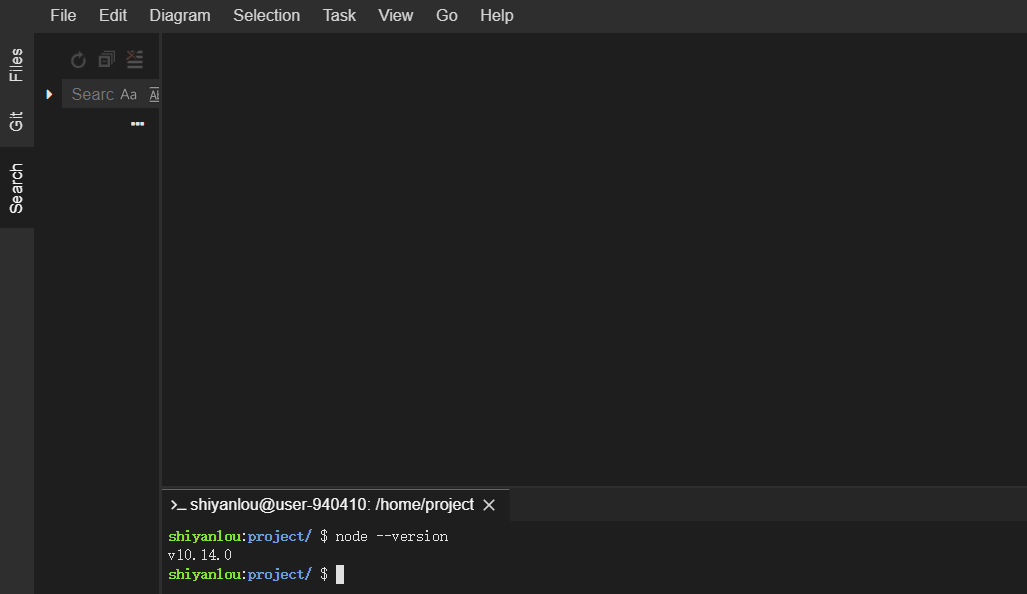
回顾一下第一小节 Vue 安装部分的脚手架工具，怎么通过 vue-cli 初始化一个 Vue 工程。

在线环境已经提前安装好了脚手架工具，下面的安装步骤 1、2、3 可以直接略过。

1、vue-cli是基于npm的，所以应该先安装node环境，通过node官网（[http://nodejs.cn/）下载系统对应的`node`安装程序。](http://nodejs.cn/%EF%BC%89%E4%B8%8B%E8%BD%BD%E7%B3%BB%E7%BB%9F%E5%AF%B9%E5%BA%94%E7%9A%84%60node%60%E5%AE%89%E8%A3%85%E7%A8%8B%E5%BA%8F%E3%80%82)



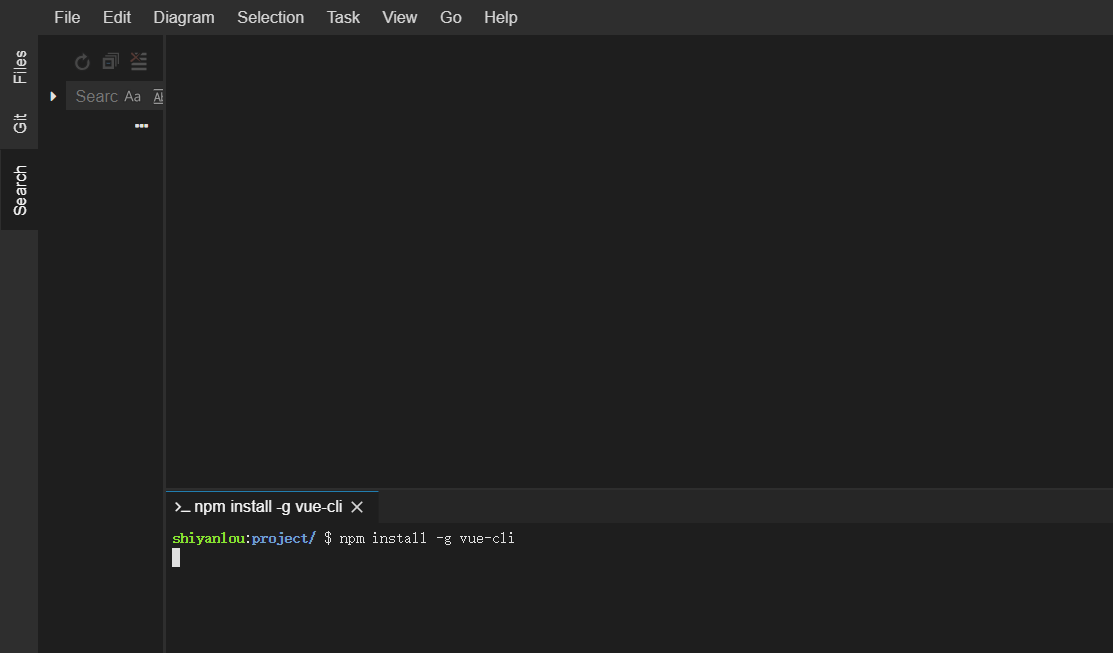
**注意：** NPM 是随同 Node.js 一起安装的包管理工具。因此当我们安装好 Node.js 的时候，也安装好了 NPM。由于实验楼的环境下已经安装好了 Node.js，大家可以直接使用，实验楼环境中跳过安装 Node 步骤，直接进行第二步。



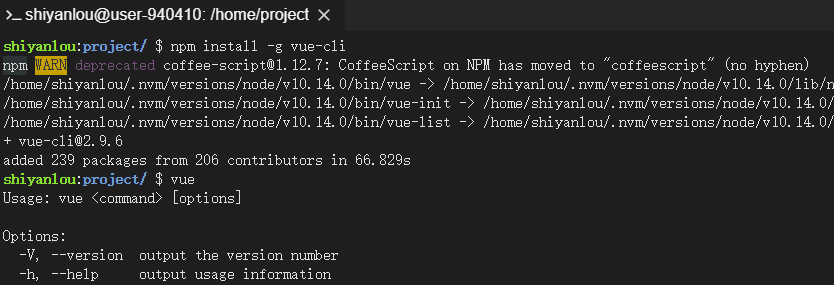
2、node安装完毕使用，npm包管理工具全局安装vue-cli

npm install -g vue-cli





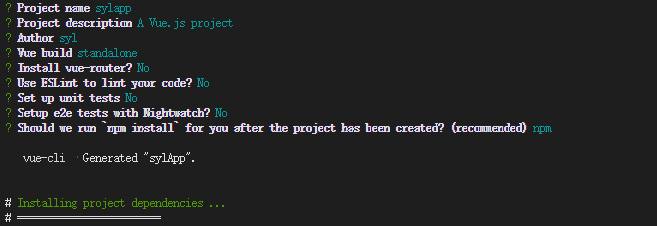
3、命令行输入vue，出现 Usage 表示安装成功。



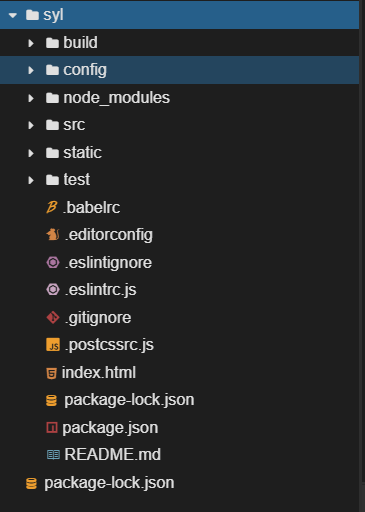
4、使用指令生成一个 vue 应用

vue init webpack appName





5、得到一个 Vue 项目文件



​ 主要包括 Vue webpack 配置文件（config）、相关包文件(node\_modules)、项目组件目录（src），想要了解项目文件目录的具体涵义的可以去 vue 脚手架官网了解。

​ 重点介绍一下 src 目录下的 App.vue 单文件组件，他就是 Vue 组件化开发的重点，他可以将任何页面内容独立出来成为组件。

<template>

<!-- 组件中的 template html 表达式 -->

<div class="header">

<p>{{msg}}</p>

</div>

</template>

<script>

// JavaScript 组件选项

export default {

name: 'Header',

data() {

return {

msg: 'hello syl',

};

},

methods: {

name() {},

},

computed: {

name() {

return this.data;

},

},

};

</script>

<style>

/\* 组件css \*/

.header {

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>

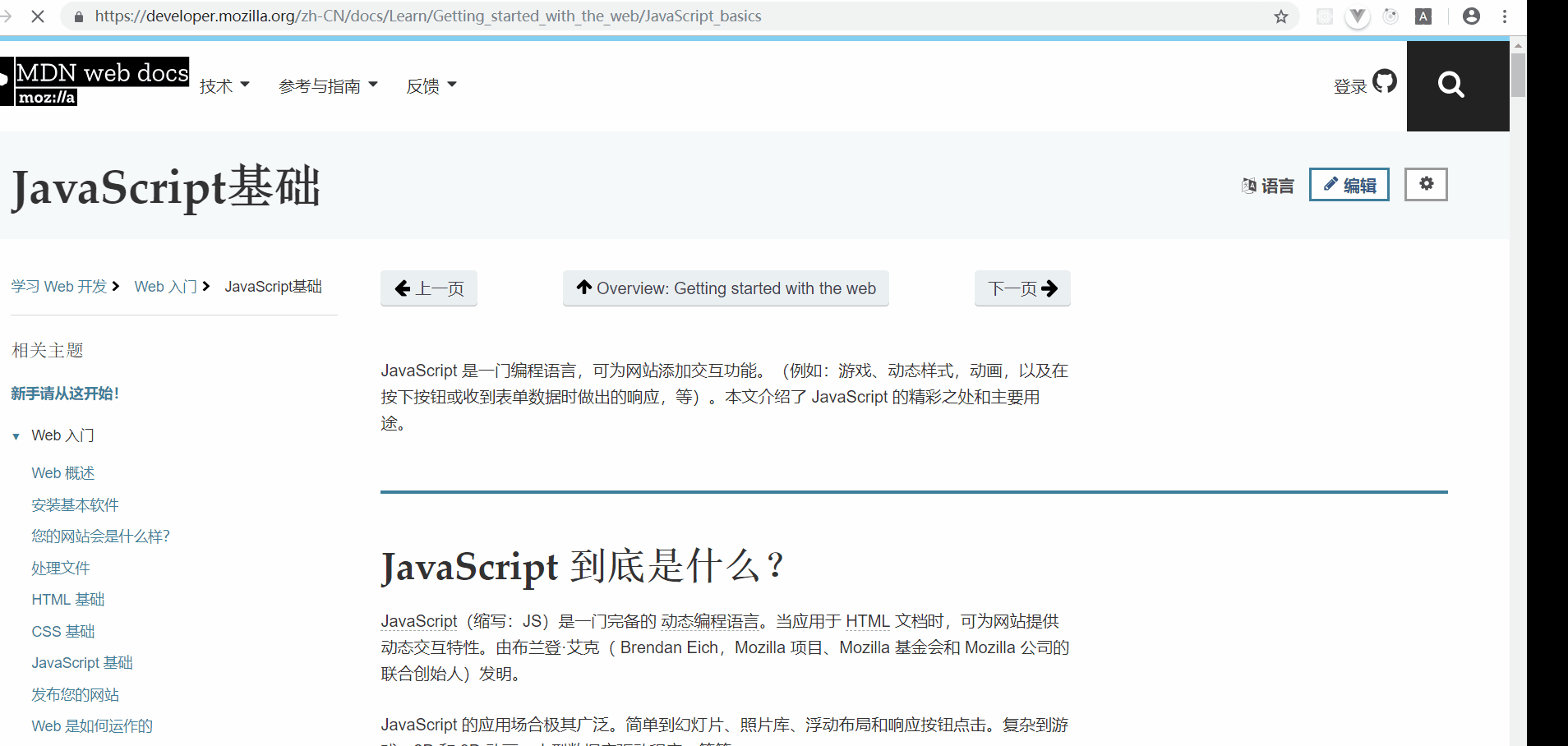
### 前端路由管理(vue-router）

在这里简单说一说，前端路由与后台路由的区别。通常我们接触到是后台路由，例如：[https://developer.mozilla.org/en-US/docs/Web/Web\_Components，后台路由通常的业务是](https://developer.mozilla.org/en-US/docs/Web/Web_Components%EF%BC%8C%E5%90%8E%E5%8F%B0%E8%B7%AF%E7%94%B1%E9%80%9A%E5%B8%B8%E7%9A%84%E4%B8%9A%E5%8A%A1%E6%98%AF) api 逻辑处理和指定渲染页面。

​ 而在大前端时代，前后端分离趋势逐渐加强，促使服务器渲染向浏览器渲染转变，后台路由更多的专注于 API 逻辑处理，更少的处理页面渲染。由此，前端路由逐渐火热，使用前端路由来管理页面切换，地址栏的变动，处理变化 DOM，用于构建单页应用，使得用户体验提升，如图：

后台路由加服务器渲染：

**注意：** 观察两者在路由切换时的区别



前端路由加浏览器渲染：



明显的前端路由加浏览器渲染用户体验更高，后台路由加服务器渲染明显出现了较长的白屏时间和较多的页面刷新次数。

重要的是前端路由都是基于两种模式 hash 模式 和 H5 history 模式，存在兼容性问题，在 Vue 你不用担心，Vue 官方对前端路由基础模式进行了精封装，[Vue Router](https://router.vuejs.org/zh/) 是 [Vue.js](http://cn.vuejs.org/) 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建 **单页面应用** 变得易如反掌。包含的功能有：

* 嵌套的路由/视图表
* 模块化的、基于组件的路由配置
* 路由参数、查询、通配符
* 基于 Vue.js 过渡系统的视图过渡效果
* 细粒度的导航控制
* 带有自动激活的 CSS class 的链接
* HTML5 历史模式或 hash 模式，在 IE9 中自动降级
* 自定义的滚动条行为

使用 Vue 开发 SPA ，Vue-Router 是必不可少的。