# HBuilder安装与使用

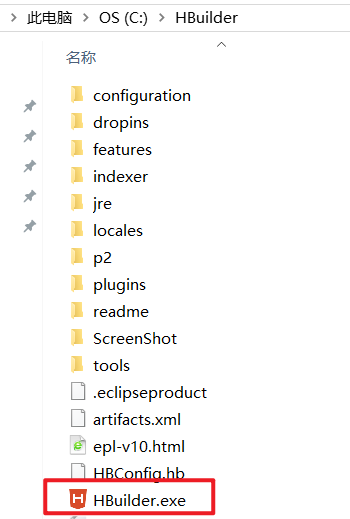
HBuilder是DCloud（数字天堂）推出的一款支持HTML5的Web开发IDE。

## 1.1 安装

1) 官网下载地址：<http://www.dcloud.io/>

2) 安装：解压zip文件到指定目录

3) 运行安装文件下的exe文件



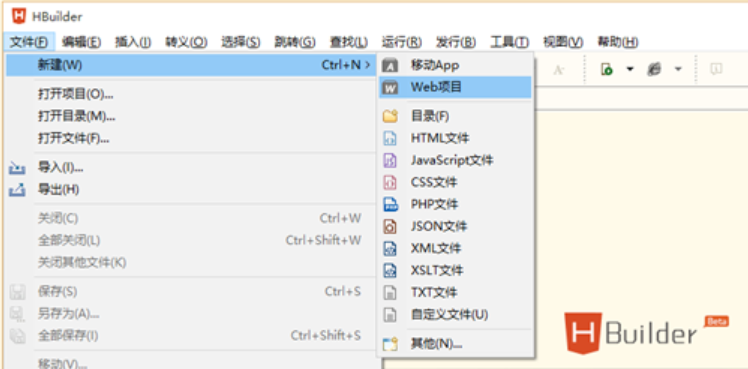
4) 第一次使用需要登录（可选），不登录的话每次都会提示该窗口



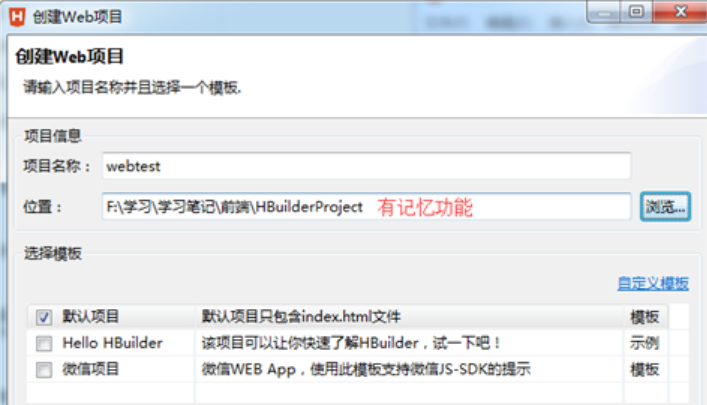
## 1.2 使用

HBuilder没有“工作空间workspace”的概念，其创建的项目，可以指定到任意位置。

HBuilder只能创建“WEB项目”和“移动APP”两种项目。

1) 依次点击文件→新建→选择Web项目(按下Ctrl+N,W可以触发快速新建(MacOS请使用Command+N,然后左键点击Web项目))  


2) 填写项目名称，选择项目位置

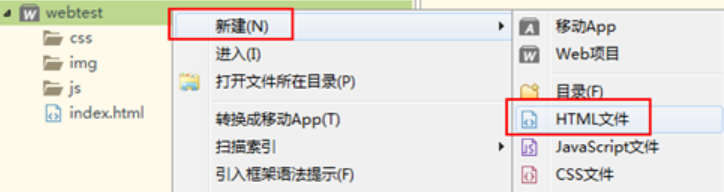


3) 创建完成，点击项目查看目录结构



### 1.2.1 创建页面

1) 在项目资源管理器中选择刚才新建的项目，依次点击文件→新建→选择HTML文件(按下Ctrl+N,W可以触发快速新建(MacOS请使用Command+N,然后左键点击HTML文件))



2) 选中文件位置，编写文件名称，选中文件使用版本



3) 创建完成，自动生成html模板





# vuejs简单入门

学习目标：

1. 了解vue（第一章）
2. 掌握vue常用系统指令（第二章）
3. 了解vue生命周期（第三章）
4. 掌握vue的ajax的使用（第四章）





# ****第一章：VueJS 概述与快速入门****

### 【目标】

​ （1）什么是VueJS？

​ （2）VueJS的应用？

### 【路径】

​ （1）VueJS介绍

​ （2）MVVM模式（视图、模型的双向绑定）

​ （3）VueJS的快速入门

### 【讲解】

## 1.1 VueJS介绍

* Vue.js 是前端的主流框架之一，和Angular.js、React.js 一起，并成为前端三大主流框架！
* Vue.js 是一套构建用户界面的框架，只关注视图层，它不仅易于上手，还便于与第三方库或既有项目整合。（Vue有配套的第三方类库，可以整合起来做大型项目的开发）
* 前端的主要工作？主要负责MVC中的V这一层；主要工作就是和界面打交道，来制作前端页面效果；

官网：<https://cn.vuejs.org/>





【小结】：

VueJS是前端渐进式框架，让Html和JavaScript无缝的整合，实现了视图和模型的双向数据绑定（MVVM）。

## 1.2 MVVM模式

【目标】

什么是MVVM模式？

【路径】

（1）什么是Model？

（2）什么是View？

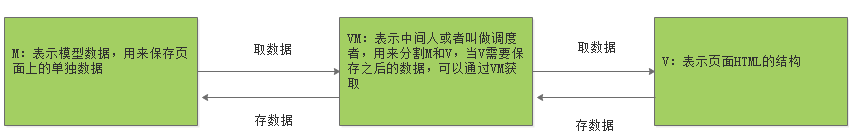
（3）什么是ViewModel？

​ MVVM是Model-View-ViewModel的简写。它本质上就是MVC 的改进版。MVVM 就是将其中的View 的状态和行为抽象化，让我们将视图 UI 和业务逻辑分开。

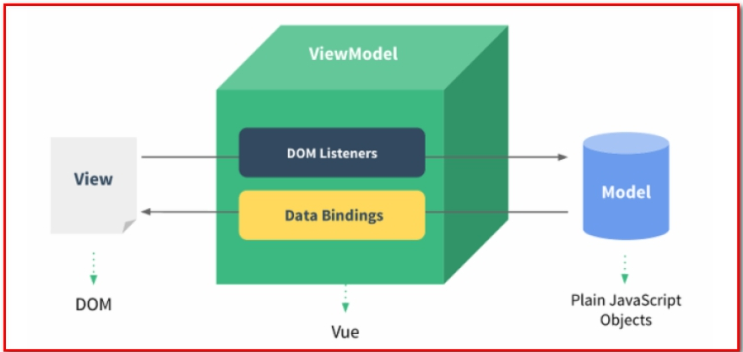
MVVM模式和MVC模式一样，主要目的是分离视图（View）和模型（Model）

MVVM是前端视图层的分层开发思想，主要是把每个页面，分成了M , V 和 VM 其中，VM是 MVVM 思想核心；

因为VM 是 M 和 V 之间的调度者



​ Vue.js 是一个提供了 MVVM 风格的**双向数据绑定**的 Javascript 库，专注于View 层。它的核心是 MVVM 中的 VM，也就是 ViewModel。 ViewModel负责连接 View 和 Model，保证视图和数据的一致性，这种轻量级的架构让前端开发更加高效、便捷。



【小结】：MVVM模式是视图和模型的双向数据绑定，通过ViewModel实现，当View发生变化，Model变化；当Model变化，View也相应的变化。

## 1.3 ****VueJS 快速入门****

【目标】：快速搭建VueJS的入门程序

【路径】：

（1）创建工程

（2）导入js

（3）编写入门程序（Vue实例）

（4）插值表达式{{message}}

### 1.3.1 ****创建工程****

### 1.3.2 导入js

### 1.3.3 入门程序

【需求】：使用vue，对message赋值，并把值显示到页面{{}}中。

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="utf-8" />

<**title**>快速入门</**title**>

<!-- 1. 导入Vue的包 -->

<**script** **src**="js/vuejs-2.5.16.js"></**script**>

</**head**>

<**body**>

<!-- 将来 new 的Vue实例，会控制这个 元素中的所有内容 -->

<!-- Vue 实例所控制的这个元素区域，就是我们的 V -->

<**div** **id**="app">

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

</**div**>

<**script**>

// 2. 创建一个Vue的实例

// 当我们导入包之后，在浏览器的内存中，就多了一个 Vue 构造函数

// 注意：我们 new 出来的这个 vm 对象，就是我们 MVVM中的 VM调度者

**var** vm = **new** Vue({

el: '#app', // 表示，当前我们 new 的这个 Vue 实例，要控制页面上的哪个区域

// 这里的 data 就是 MVVM中的 M，专门用来保存 每个页面的数据的

data: { // data 属性中，存放的是 el 中要用到的数据

msg: '欢迎学习Vue' // 通过 Vue 提供的指令，很方便的就能把数据渲染到页面上，程序员不再手动操作DOM元素了【前端的Vue之类的框架，不提倡我们去手动操作DOM元素了】

}

})

</**script**>

</**body**>

</**html**>

### 1.3.4 插值表达式（{{message}}）

​ 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，Mustache 标签将会被替代为对应数据对象上属性的值。无论何时，绑定的数据对象上属性发生了改变，插值处的内容都会更新。

​ Vue.js 都提供了完全的 JavaScript 表达式支持。

<!DOCTYPE html>

<**html** **lang**="en">

<**head**>

<**meta** **charset**="UTF-8">

<**title**>Title</**title**>

</**head**>

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

<**body**>

<**div** **id**="app">

{{message}}<**br**>

{{message1}}<**br**>

{{message1+10+100}}<**br**>

{{message1>5?"大于5的数":"小于5的数"}}

</**div**>

</**body**>

</**html**>

<**script**>

**var** vue = **new** Vue({

el:"#app", // 表示vue的程序入口，在id=app的div中可以使用vue声明的指令

data:{ // 声明模型，模型的数据可以是基本类型，json数据

message:"你好,vuejs",

message1:10

},

methods:{ // 声明函数，绑定页面的事件

}

})

// 改变模型的数据，测试视图是否发生变化

vue.message = "你好，尚硅谷"

</**script**>

​ 这些表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析。有个限制就是，每个绑定都只能包含单个表达式，所以下面的例子都不会生效。

<!-- 这是语句，不是表达式 -->

{{ var a = 1 }}

<!-- 流控制也不会生效，请使用三元表达式 -->

{{ **if** (ok) { return message } }}

### 【小结】

​ （1）VueJS介绍（前端渐进式框架）

​ （2）MVVM模式（视图、模型的双向绑定）

​ （3）VueJS的快速入门

​ 是不是很简单呢^\_^？只需要导入1个js，定义1个Vue实例，写3个属性（el，data，methods），写1个插值表达式输出结果，即可实现效果

​ 其中data：用来定义模型（Model）

​ 其中 {{}}：用来显示数据（View）

​ 其中ViewModel用来将模型的数据显示到视图上

# ****第二章：VueJS 常用系统指令（重点）****

### 【目标】

掌握vue的常用命令，在项目开发中会经常使用

### 【路径】

1：v-on：事件

2：v-text与v-html：文本

3：v-bind：属性

4：v-model：绑定表单（用于回显）

5：v-for：循环数据

6：v-if与v-show：判断

### 【讲解】

## 2.1 v-on:（等同于@）

可以用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码

### 2.1.1 v-on:click（等同于@click）

【需求】：点击按钮事件，改变message的值

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="utf-8" />

<**title**>v-on:click单击事件</**title**>

<**script** **src**="js/vuejs-2.5.16.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

{{message}}

<!--<button v-on:click="fun('good')">改变</button>-->

<**button** @**click**="fun('good')">改变</**button**>

</**div**>

</**body**>

<**script**>

//view model

**var** vm = **new** Vue({

el:"#app",

data:{

message:"hello world" //model表示模型，封装数据

},

methods:{

fun:**function**(msg){

// this代表的是vue对象，或者使用vm

**this**.message=msg;

}

}

});

</**script**>

</**html**>

### 2.1.2 跑马灯

<!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**>Document</**title**>

<!-- 1. 导入Vue包 -->

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<!-- 2. 创建一个要控制的区域 -->

<**div** **id**="app">

<**input** **type**="button" **value**="浪起来" @**click**="lang">

<**input** **type**="button" **value**="低调" @**click**="stop">

<**h4**>{{ msg }}</**h4**>

</**div**>

<**script**>

// 注意：在 VM实例中，如果想要获取 data 上的数据，或者 想要调用 methods 中的 方法，必须通过 this.数据属性名 或 this.方法名 来进行访问，这里的this，就表示 我们 new 出来的 VM 实例对象

**var** vm = **new** Vue({

el: '#app',

data: {

msg: '硅谷，别浪~~！',

intervalId: null // 在data上定义 定时器Id

},

methods: {

lang() {

// console.log(this.msg)

// 获取到头的第一个字符

**if** (**this**.intervalId != null) **return**;

//setInterval(function(){

**this**.intervalId = setInterval(() => {

**var** start = **this**.msg.substring(0, 1)

console.log(start); // 获取到 后面的所有字符

**var** end = **this**.msg.substring(1)

console.log(end);

// 重新拼接得到新的字符串，并赋值给 this.msg

**this**.msg = end + start

console.log(**this**.message);

}, 400);

// 注意： VM实例，会监听自己身上 data 中所有数据的改变，只要数据一发生变化，就会自动把 最新的数据，从data 上同步到页面中去；【好处：程序员只需要关心数据，不需要考虑如何重新渲染DOM页面】

},

stop() { // 停止定时器

clearInterval(**this**.intervalId)

// 每当清除了定时器之后，需要重新把 intervalId 置为 null

**this**.intervalId = null;

}

}

})

// 分析：

// 1. 给 【浪起来】 按钮，绑定一个点击事件 v-on @

// 2. 在按钮的事件处理函数中，写相关的业务逻辑代码：拿到 msg 字符串，然后 调用 字符串的 substring 来进行字符串的截取操作，把 第一个字符截取出来，放到最后一个位置即可；

// 3. 为了实现点击下按钮，自动截取的功能，需要把 2 步骤中的代码，放到一个定时器中去；

</**script**>

</**body**>

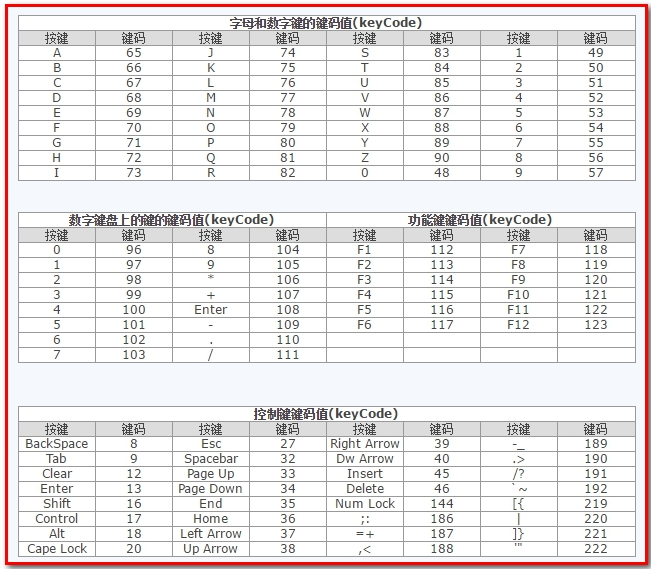
</**html**>

### 2.1.3 v-on:keydown

表示键盘按下事件。

<http://www.t086.com/article/4315>

Keycode对照表



【需求】：对文本输入框做校验，使用键盘按下事件，如果按下0-9的数字，正常显示，其他按键则阻止事件执行。

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="utf-8" />

<**title**>v-on:keydown</**title**>

<**script** **src**="js/vuejs-2.5.16.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<**input** **type**="text" **v-on:keydown**="fun($event)">

<!-- <input type="text" @keydown="fun($event)" /> -->

</**div**>

</**body**>

<**script**>

//view model

**new** Vue({

el: "#app",

data: {

message: 10 //model

},

methods: {

fun: **function**(e) {

//1.捕获keyCode 判断它是否是0-9 需要使用event对象

**var** keyCode = e.keyCode;

// 只能输入数字

**if**(!(keyCode >= 48 && keyCode <= 57)) {

//2.阻止默认行为执行

e.preventDefault();

}

}

}

});

</**script**>

</**html**>

输入框中只能输入0-9的数字，如果不是0-9的数字，不能输入。

### 2.1.4 v-on:mouseover

鼠标移入区域事件

【需求1】：给指定区域大小的div中添加样式，鼠标移到div中，弹出窗口。

【需求2】：阻止上一层事件的执行

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="UTF-8">

<**title**></**title**>

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

<**style**>

.father{

**width**: 500px;

**height**: 500px;

**background**: red;

}

.son{

**width**: 300px;

**height**: 300px;

**background**: blue;

}

</**style**>

</**head**>

<**body**>

<**div** **id**="app">

<**div** **class**="father" @**mouseover**="alert('father')">

<!--<div class="son" @mouseover.stop="alert('son')"></div>-->

<**div** **class**="son" @**mouseover**="alert('son')"></**div**>

</**div**>

</**div**>

</**body**>

<**script**>

**var** vm = **new** Vue({

el:"#app"

})

</**script**>

</**html**>

### 2.1.5 按键修饰符（了解）

Vue 允许为 v-on 在监听键盘事件时添加按键修饰符

全部的按键别名：

* .enter // 表示键盘的enter键
* .tab
* .delete (捕获 “删除” 和 “退格” 键)
* .esc
* .space
* .up
* .down
* .left
* .right
* .ctrl
* .alt
* .shift
* .meta

【需求】：在输入框中，如果输入回车键，就执行弹出窗口事件（可用于网页登录）。

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="utf-8" />

<**title**>v-on:按键修饰符</**title**>

<**script** **src**="js/vuejs-2.5.16.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<!--当按下按键enter的时候，触发fun事件-->

<**input** **type**="text" @**keydown.enter**="fun()">

<!--<input type="text" @keydown="fun($event)" />-->

</**div**>

</**body**>

<**script**>

//view model

**new** Vue({

el: "#app",

methods: {

fun:**function**(){

alert("输入的回车键！");

}

// fun:function (e) {

// var keyCode = e.keyCode;

// //alert(keyCode);

// if(keyCode == 13){

// alert("输入的回车键！")

// }

// }

}

});

</**script**>

</**html**>

## 2.2 v-text与v-html

使用{{}}可以输出文本的值。

v-text：输出文本内容，不会解析html元素

v-html：输出文本内容，会解析html元素

【需求】：使用message属性赋值，查看页面输出内容。

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="utf-8" />

<**title**>v-text与v-html</**title**>

<**script** **src**="js/vuejs-2.5.16.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<**div** **v-text**="message"></**div**>

<**div** **v-html**="message"></**div**>

</**div**>

</**body**>

<**script**>

//view model

**new** Vue({

el:"#app",

data:{

message:"<h1>hello world</h1>"//model

}

});

</**script**>

</**html**>

## 2.3 class绑定（等同于:）

插值语法不能作用在 HTML 属性上，遇到这种情况应该使用 v-bind指令

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="UTF-8">

<**title**></**title**>

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

<**style**>

.red{

**color**: red;

}

.big{

**font-size**: 30px;

}

</**style**>

</**head**>

<**body**>

<**div** **id**="app">

<**p**>尚硅谷</**p**>

<**p** **class**="red big">尚硅谷</**p**>

<**p** **:class**="{red:isRed,big:isBig}">尚硅谷</**p**>

<!-- 在数组中使用三元表达式 -->

<**p** **:class**="[ok?'red':'','big']">尚硅谷</**p**>

<!-- 直接传递一个数组，注意： 这里的 class 需要使用 v-bind 做数据绑定 -->

<**p** **:class**="[showred,'big']">尚硅谷</**p**>

<**a** **:href**="url">尚硅谷</**a**>

</**div**>

</**body**>

<**script**>

**var** vm = **new** Vue({

el:"#app",

data:{

isRed:true,

isBig:true,

ok:true,

showred:'red',

url:"http://www.atguigu.com"

}

})

</**script**>

</**html**>

v-bind简写方式

<!-- 完整语法 -->

<**a** **v-bind:href**="url"></**a**>

<!-- 缩写 -->

<**a** **:href**="url"></**a**>

## 2.4 选项卡

### 2.4.1 原生JS实现选项卡

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="UTF-8">

<**title**></**title**>

<**script** **type**="text/javascript" **src**="js/jquery-1.12.4.min.js"></**script**>

<**style** **type**="text/css">

div {

**width**: 300px;

**height**: 300px;

**background**: pink;

**display**: none;

}

.block {

**display**: block;

}

.current {

**background**: pink;

}

</**style**>

</**head**>

<**body**>

<**input** **type**="button" **value**="按钮一" **class**="current" />

<**input** **type**="button" **value**="按钮二" />

<**input** **type**="button" **value**="按钮三" />

<**div** **class**="block">000000</**div**>

<**div**>111111</**div**>

<**div**>222222</**div**>

</**body**>

<**script** **type**="text/javascript">

$('input').click(**function**() {

//siblings()同胞元素

$(**this**).addClass('current').siblings().removeClass('current')

$('div').eq($(**this**).index()).addClass('block').siblings().removeClass('block')

})

</**script**>

</**html**>



### 2.4.2 vue.js实现选项卡

<!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**>Document</**title**>

<**style**>

.tab\_con{

**width**:500px;

**height**:350px;

**margin**:50px auto 0;

}

.tab\_btns{

**height**:50px;

}

.tab\_btns input{

**width**:100px;

**height**:50px;

**background**:#ddd;

**border**:0px;

**outline**:none;

}

.tab\_btns .active{

**background**:gold;

}

.tab\_cons{

**height**:300px;

**background**:gold;

}

.tab\_cons div{

**height**:300px;

**line-height**:300px;

**text-align**:center;

**display**:none;

**font-size**:30px;

}

.tab\_cons .current{

**display**:block;

}

</**style**>

<**script** **type**="text/javascript" **src**="js/vue.min.js" ></**script**>

</**head**>

<**body**>

<**div** **class**="tab\_con">

<**div** **class**="tab\_btns">

<**input** **type**="button" **value**="按钮一" **:class**="[(num==0)?'active':'']" @**click**="num=0">

<**input** **type**="button" **value**="按钮二" **:class**="[(num==1)?'active':'']" @**click**="num=1">

<**input** **type**="button" **value**="按钮三" **:class**="[(num==2)?'active':'']" @**click**="num=2">

</**div**>

<**div** **class**="tab\_cons">

<**div** **:class**="[(num==0)?'current':'']">按钮一对应的内容</**div**>

<**div** **:class**="[(num==1)?'current':'']">按钮二对应的内容</**div**>

<**div** **:class**="[(num==2)?'current':'']">按钮三对应的内容</**div**>

</**div**>

</**div**>

</**body**>

<**script** **type**="text/javascript">

**var** vm = **new** Vue({

el:".tab\_con",

data:{

num:0

}

})

</**script**>

</**html**>



## 2.5 ****v-model****

用于读取视图中的数据。

【需求】：使用vue赋值json数据，并显示到页面的输入框中（表单回显）。

​ 点击提交按钮：

​ 测试：改变输入框的值，同时验证模型的数据发生改变。

​ 测试：改变json数据，验证同时输入框的内容也发生改变。

​ 这就是MVVM模式

<!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**>Document</**title**>

<**script** **type**="text/javascript" **src**="js/vue.min.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<**p**>

<**label**>用户名：</**label**>{{username}}<**br**>

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

<**input** **type**="button" **value**="改变值" @**click**="fnChange" />

</**p**>

<**select** **v-model**="sel">

<**option** **value**="0">大班</**option**>

<**option** **value**="1">中班</**option**>

<**option** **value**="2">小班</**option**>

</**select**>

{{sel}}

<**br** />

<**input** **type**="checkbox" **v-model**="danCheck"/>同意用户协议<**br** />

<**h1**>{{danCheck}}</**h1**>

<**input** **type**="checkbox" **value**="西瓜" **v-model**="duoCheck" />西瓜

<**input** **type**="checkbox" **value**="苹果" **v-model**="duoCheck" />苹果

<**input** **type**="checkbox" **value**="黄瓜" **v-model**="duoCheck" />黄瓜

<**h1**>{{duoCheck}}</**h1**>

<**input** **type**="radio" **value**="男" **v-model**="sex"/>男

<**input** **type**="radio" **value**="女" **v-model**="sex" />女

<**hr** />

<**textarea** **v-model**="tarea"></**textarea**>

<**h1**>{{tarea}}</**h1**>

</**div**>

</**body**>

<**script** **type**="text/javascript">

**var** vm = **new** Vue({

el: "#app",

data: {

sel:'0',

username: "",

danCheck:true,

duoCheck:[],

sex:'男',

tarea:'',

},

methods: {

fnChange: **function**() {

**this**.username += 'haha'

}

}

})

</**script**>

</**html**>

练习：简易计算器

<!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**>Document</**title**>

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<**input** **type**="text" **v-model**="n1">

<**select** **v-model**="opt">

<**option** **value**="+">+</**option**>

<**option** **value**="-">-</**option**>

<**option** **value**="\*">\*</**option**>

<**option** **value**="/">/</**option**>

</**select**>

<**input** **type**="text" **v-model**="n2">

<**input** **type**="button" **value**="=" @**click**="calc">

<**input** **type**="text" **v-model**="result">

</**div**>

<**script**>

// 创建 Vue 实例，得到 ViewModel

**var** vm = **new** Vue({

el: '#app',

data: {

n1: 0,

n2: 0,

result: 0,

opt: '+'

},

methods: {

calc() { // 计算器算数的方法

// 逻辑：

**switch** (**this**.opt) {

**case** '+':

**this**.result = parseInt(**this**.n1) + parseInt(**this**.n2)

**break**;

**case** '-':

**this**.result = parseInt(**this**.n1) - parseInt(**this**.n2)

**break**;

**case** '\*':

**this**.result = parseInt(**this**.n1) \* parseInt(**this**.n2)

**break**;

**case** '/':

**this**.result = parseInt(**this**.n1) / parseInt(**this**.n2)

**break**;

}

}

}

});

</**script**>

</**body**>

</**html**>



## 2.6 ****v-for****

### 2.6.1 普通数组

用于操作array/集合，遍历

<!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**>Document</**title**>

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<!-- <p>{{list[0]}}</p>

<p>{{list[1]}}</p>

<p>{{list[2]}}</p>

<p>{{list[3]}}</p>

<p>{{list[4]}}</p> -->

<**p** **v-for**="(item, i) in list">

索引值：{{i}} --- 每一项：{{item}}

</**p**>

</**div**>

<**script**>

// 创建 Vue 实例，得到 ViewModel

**var** vm = **new** Vue({

el: '#app',

data: {

list: [1, 2, 3, 4, 5, 6]

},

methods: {}

});

</**script**>

</**body**>

</**html**>

### 2.6.2 对象数组

<!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**>Document</**title**>

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<**p** **v-for**="(user, i) in list">Id：{{ user.id }} --- 名字：{{ user.name }} --- 索引：{{i}}</**p**>

</**div**>

<**script**>

// 创建 Vue 实例，得到 ViewModel

**var** vm = **new** Vue({

el: '#app',

data: {

list: [

{ id: 1, name: 'zs1' },

{ id: 2, name: 'zs2' },

{ id: 3, name: 'zs3' },

{ id: 4, name: 'zs4' }

]

},

methods: {}

});

</**script**>

</**body**>

</**html**>

### 2.6.3 循环对象

<!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**>Document</**title**>

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<!-- 注意：在遍历对象身上的键值对的时候， 除了 有 val key ,在第三个位置还有 一个 索引 -->

<**p** **v-for**="(val, key, i) in user">值是： {{ val }} --- 键是： {{key}} -- 索引： {{i}}</**p**>

</**div**>

<**script**>

// 创建 Vue 实例，得到 ViewModel

**var** vm = **new** Vue({

el: '#app',

data: {

user: {

id: 1,

name: '尚硅谷',

address: '深圳'

}

},

methods: {}

});

</**script**>

</**body**>

</**html**>

### 2.6.4 迭代数字

<!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**>Document</**title**>

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<!-- in 后面我们放过 普通数组，对象数组，对象， 还可以放数字 -->

<!-- 注意：如果使用 v-for 迭代数字的话，前面的 count 值从 1 开始 -->

<**p** **v-for**="count in 10">这是第 {{ count }} 次循环</**p**>

</**div**>

<**script**>

// 创建 Vue 实例，得到 ViewModel

**var** vm = **new** Vue({

el: '#app',

data: {},

methods: {}

});

</**script**>

</**body**>

</**html**>

### 2.6.5 for循环中key的使用

<!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**>Document</**title**>

<**script** **src**="./lib/vue-2.4.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

<**div**>

<**label**>Id:

<**input** **type**="text" **v-model**="id">

</**label**>

<**label**>Name:

<**input** **type**="text" **v-model**="name">

</**label**>

<**input** **type**="button" **value**="添加" @**click**="add">

</**div**>

<!-- 注意： v-for 循环的时候，key 属性只能使用 number或者string -->

<!-- 注意： key 在使用的时候，必须使用 v-bind 属性绑定的形式，指定 key 的值 -->

<**p** **v-for**="item in list" **:key**="item.id">

<**input** **type**="checkbox">{{item.id}} --- {{item.name}}

</**p**>

</**div**>

<**script**>

// 创建 Vue 实例，得到 ViewModel

**var** vm = **new** Vue({

el: '#app',

data: {

id: '',

name: '',

list: [

{ id: 1, name: '李斯' },

{ id: 2, name: '嬴政' },

{ id: 3, name: '赵高' },

{ id: 4, name: '韩非' },

{ id: 5, name: '荀子' }

]

},

methods: {

add() { // 添加方法

//unshift() 方法可向数组的开头添加一个或更多元素，并返回新的长度。

**this**.list.unshift({ id: **this**.id, name: **this**.name })

}

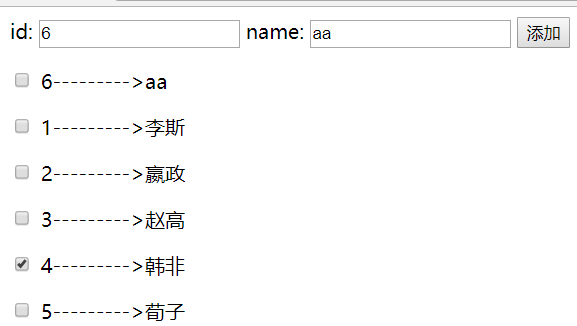
}

});

</**script**>

</**body**>

</**html**>



## 2.7 ****v-if****与v-show

v-if是根据表达式的值来决定是否渲染元素

v-show是根据表达式的值来切换元素的display css属性

【需求】：使用vue赋值flag变量（boolean类型），用来判断元素中的内容是否显示。

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="UTF-8">

<**title**></**title**>

<**script** **type**="text/javascript" **src**="js/vue.min.js" ></**script**>

</**head**>

<!-- v-if 的特点：每次都会重新删除或创建元素 -->

<!-- v-show 的特点： 每次不会重新进行DOM的删除和创建操作，只是切换了元素的 display:none 样式 -->

<**body**>

<**div** **id**="app">

<**p** **v-if**="ok">这是使用了v-if的p标签</**p**>

<**p** **v-else**>这是使用了v-else的p标签</**p**>

<**p** **v-show**="ok">这是使用v-show的p标签</**p**>

<**div** **v-if**="code=='A'">A</**div**>

<**div** **v-else-if**="code=='B'">B</**div**>

<**div** **v-else-if**="code=='C'">C</**div**>

<**div** **v-else**>NOT A/B/C</**div**>

</**div**>

</**body**>

<**script** **type**="text/javascript">

**var** vm = **new** Vue({

el:"#app",

data:{

ok:true,

code:'D'

}

})

</**script**>

</**html**>

## 2.8 回顾 数组 splice() 方法

splice() 方法向/从数组中添加/删除项目，然后返回被删除的项目。

该方法会改变原始数组。

语法

arrayObject.**splice**(**index**,howmany,item1,.....,itemX)

| **参数** | **描述** |
| --- | --- |
| index | 必需。整数，规定添加/删除项目的位置，使用负数可从数组结尾处规定位置。 |
| howmany | 必需。要删除的项目数量。如果设置为 0，则不会删除项目。 |
| item1, …, itemX | 可选。向数组添加的新项目。 |

返回值

| **类型** | **描述** |
| --- | --- |
| Array | 包含被删除项目的新数组，如果有的话。 |

说明

splice() 方法可删除从 index 处开始的零个或多个元素，并且用参数列表中声明的一个或多个值来替换那些被删除的元素。

如果从 arrayObject 中删除了元素，则返回的是含有被删除的元素的数组。

① 案例一

在本例中，我们将创建一个新数组，并向其添加一个元素：

<**script** **type**="text/javascript">

**var** arr = **new** Array(6)

arr[0] = "George"

arr[1] = "John"

arr[2] = "Thomas"

arr[3] = "James"

arr[4] = "Adrew"

arr[5] = "Martin"

document.write(arr + "<br />")

arr.splice(2,0,"William")

document.write(arr + "<br />")

</**script**>

输出：

George,John,Thomas,James,Adrew,Martin

George,John,William,Thomas,James,Adrew,Martin

② 案例二

在本例中我们将删除位于 index 2 的元素，并添加一个新元素来替代被删除的元素：

<**script** **type**="text/javascript">

**var** arr = **new** Array(6)

arr[0] = "George"

arr[1] = "John"

arr[2] = "Thomas"

arr[3] = "James"

arr[4] = "Adrew"

arr[5] = "Martin"

document.write(arr + "<br />")

arr.splice(2,1,"William")

document.write(arr)

</**script**>

输出：

George,John,Thomas,James,Adrew,Martin

George,John,William,James,Adrew,Martin

③ 案例三

在本例中我们将删除从 index 2 (“Thomas”) 开始的三个元素，并添加一个新元素 (“William”) 来替代被删除的元素：

<**script** **type**="text/javascript">

**var** arr = **new** Array(6)

arr[0] = "George"

arr[1] = "John"

arr[2] = "Thomas"

arr[3] = "James"

arr[4] = "Adrew"

arr[5] = "Martin"

document.write(arr + "<br />")

var del = arr.splice(2,3,"William")

document.write(arr + "<br />")

document.write(del)

</**script**>

输出：

George,John,Thomas,James,Adrew,Martin

George,John,William,Martin

Thomas,James,Adrew

## 2.9 案例 todolist

### 2.9.1 使用vue.js实现

<!DOCTYPE html>

<**html** **lang**="en">

<**head**>

<**meta** **charset**="UTF-8">

<**title**>todolist</**title**>

<**style** **type**="text/css">

.list\_con {

**width**: 600px;

**margin**: 50px auto 0;

}

.inputtxt {

**width**: 550px;

**height**: 30px;

**border**: 1px solid #ccc;

**padding**: 0px;

**text-indent**: 10px;//段落缩进

}

.inputbtn {

**width**: 40px;

**height**: 32px;

**padding**: 0px;

**border**: 1px solid #ccc;

}

.list {

**margin**: 0;

**padding**: 0;

**list-style**: none;

**margin-top**: 20px;

}

.list li {

**height**: 40px;

**line-height**: 40px;

**border-bottom**: 1px solid #ccc;

}

.list li span {

**float**: left;

}

.list li a {

**float**: right;

**text-decoration**: none;//文本修饰,none表示标准的文本

**margin**: 0 10px;

}

</**style**>

<**script** **type**="text/javascript" **src**="js/vue.min.js" ></**script**>

<**script** **type**="text/javascript">

window.onload = **function**(){

**var** vm = **new** Vue({

el:"#app",

data:{

content:['学习HTML','学习CSS','学习JS'],

txt:''

},

methods:{

fnAdd:**function**(){

**this**.content.push(**this**.txt)

**this**.txt = ''

document.getElementById("txt1").focus()

},

fnDel:**function**(index){

**this**.content.splice(index,1)

},

fnUp:**function**(index){

**if** (index == 0) {

alert('亲到顶了')

**return**

}

**this**.content.splice(index-1,0,**this**.content[index])

**this**.content.splice(index+1,1)

},

fnDown:**function**(index){

**if** (index == **this**.content.length-1) {

alert('亲到底了')

**return**

}

**this**.content.splice(index+2,0,**this**.content[index])

**this**.content.splice(index,1)

}

}

})

}

</**script**>

</**head**>

<**body**>

<**div** **class**="list\_con" **id**="app">

<**h2**>To do list</**h2**>

<**input** **type**="text" **name**="" **id**="txt1" **class**="inputtxt" **v-model**="txt">

<**input** **type**="button" **name**="" **value**="增加" **id**="btn1" **class**="inputbtn" @**click**="fnAdd">

<**ul** **id**="list" **class**="list">

<**li** **v-for**="(item,index) in content">

<**span**>{{item}}</**span**>

<**a** **href**="javascript:;" **class**="up" @**click**="fnUp(index)"> ↑ </**a**>

<**a** **href**="javascript:;" **class**="down" @**click**="fnDown(index)"> ↓ </**a**>

<**a** **href**="javascript:;" **class**="del" @**click**="fnDel(index)">删除</**a**>

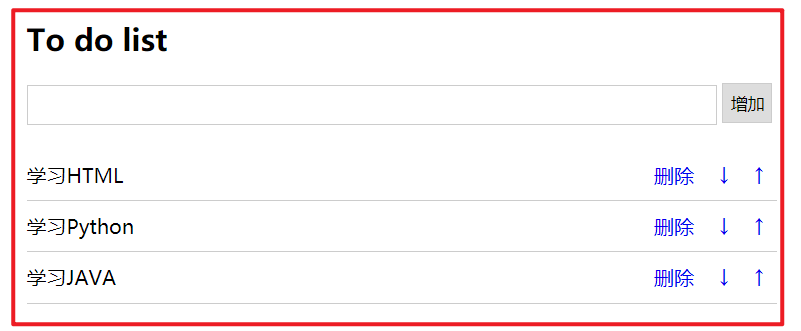
</**li**>

</**ul**>

</**div**>

</**body**>

</**html**>



## 2.10 聊天对话框【课后练习】

### 2.10.1 使用vue.js实现

<!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**>Document</**title**>

<**style** **type**="text/css">

.talk\_con {

**width**: 600px;

**height**: 500px;

**border**: 1px solid #666;

**margin**: 50px auto 0;

**background**: #f9f9f9;

}

.talk\_show {

**width**: 580px;

**height**: 420px;

**border**: 1px solid #666;

**background**: #fff;

**margin**: 10px auto 0;

**overflow**: auto;

}

.talk\_input {

**width**: 580px;

**margin**: 10px auto 0;

}

.whotalk {

**width**: 80px;

**height**: 30px;

**float**: left;

**outline**: none;

}

.talk\_word {

**width**: 420px;

**height**: 26px;

**padding**: 0px;

**float**: left;

**margin-left**: 10px;

**outline**: none;

**text-indent**: 10px;

}

.talk\_sub {

**width**: 56px;

**height**: 30px;

**float**: left;

**margin-left**: 10px;

}

.atalk {

**margin**: 10px;

}

.atalk span {

**display**: inline-block;

**background**: #0181cc;

**border-radius**: 10px;

**color**: #fff;

**padding**: 5px 10px;

}

.btalk {

**margin**: 10px;

**text-align**: right;

}

.btalk span {

**display**: inline-block;

**background**: #ef8201;

**border-radius**: 10px;

**color**: #fff;

**padding**: 5px 10px;

}

</**style**>

<**script** **type**="text/javascript" **src**="js/vue.min.js"></**script**>

<**script** **type**="text/javascript">

// 写出对应功能代码

window.onload = **function**() {

**var** vm = **new** Vue({

el: "#app",

data: {

content: [{

who: 'A',

say: '吃饭了吗'

},

{

who: 'B',

say: '吃饭了吗'

},

],

txt: '',

sel: '0'

},

methods: {

fnAdd: **function**() {

**if**(**this**.txt==''){

alert('请您输入内容');

**return;**

}

**this**.content.push({

who: **this**.sel == '0' ? 'A' : 'B',

say: **this**.txt

})

**this**.txt = ''

oTxt.focus()

setTimeout(**function**(){

document.getElementById("words").scrollTop = '9999'

},1)

}

}

});

**var** oTxt = document.getElementById("talkwords")

oTxt.focus()

}

</**script**>

</**head**>

<**body**>

<**div** **class**="talk\_con" **id**="app">

<**div** **class**="talk\_show" **id**="words">

<**div** **:class**="[item.who=='A'? 'atalk':'btalk']" **v-for**="item in content">

<**span**>{{item.who}}说：{{item.say}}</**span**>

</**div**>

<!--<div class="btalk"><span>B说：还没呢，你呢？</span></div>-->

</**div**>

<**div** **class**="talk\_input">

<**select** **class**="whotalk" **id**="who" **v-model**="sel">

<**option** **value**="0">A说：</**option**>

<**option** **value**="1">B说：</**option**>

</**select**>

<**input** **type**="text" **class**="talk\_word" **id**="talkwords" **v-model**="txt">

<**input** **type**="button" **value**="发送" **class**="talk\_sub" **id**="talksub" @**click**="fnAdd">

</**div**>

</**div**>

</**body**>

</**html**>



## 【小结】

1：v-on（@）：事件

2：v-text与v-html（{{}}）：文本

3：v-bind（:）：属性

4：v-model：绑定表单（用于回显）

5：v-for：循环数据

6：v-if与v-show：判断

# ****第三章：VueJS生命周期****

### 【目标】

1：什么叫生命周期

2：什么叫钩子函数

3：演示生命周期的执行和钩子函数的使用

### 【路径】

1：什么是生命周期

2：什么是钩子函数

* beforeCreate，==created==
* beforeMount，==mounted==
* beforeUpdate，updated
* beforeDestroy，destroyed

3：演示vue对象的创建、赋值、显示、改值、销毁的全过程，即Vue的生命周期

### 【讲解】

什么叫生命周期？

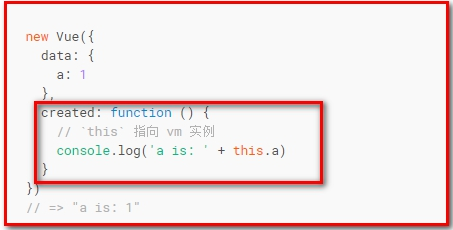
每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 、并在数据变化时更新 DOM 等。

什么叫钩子函数？

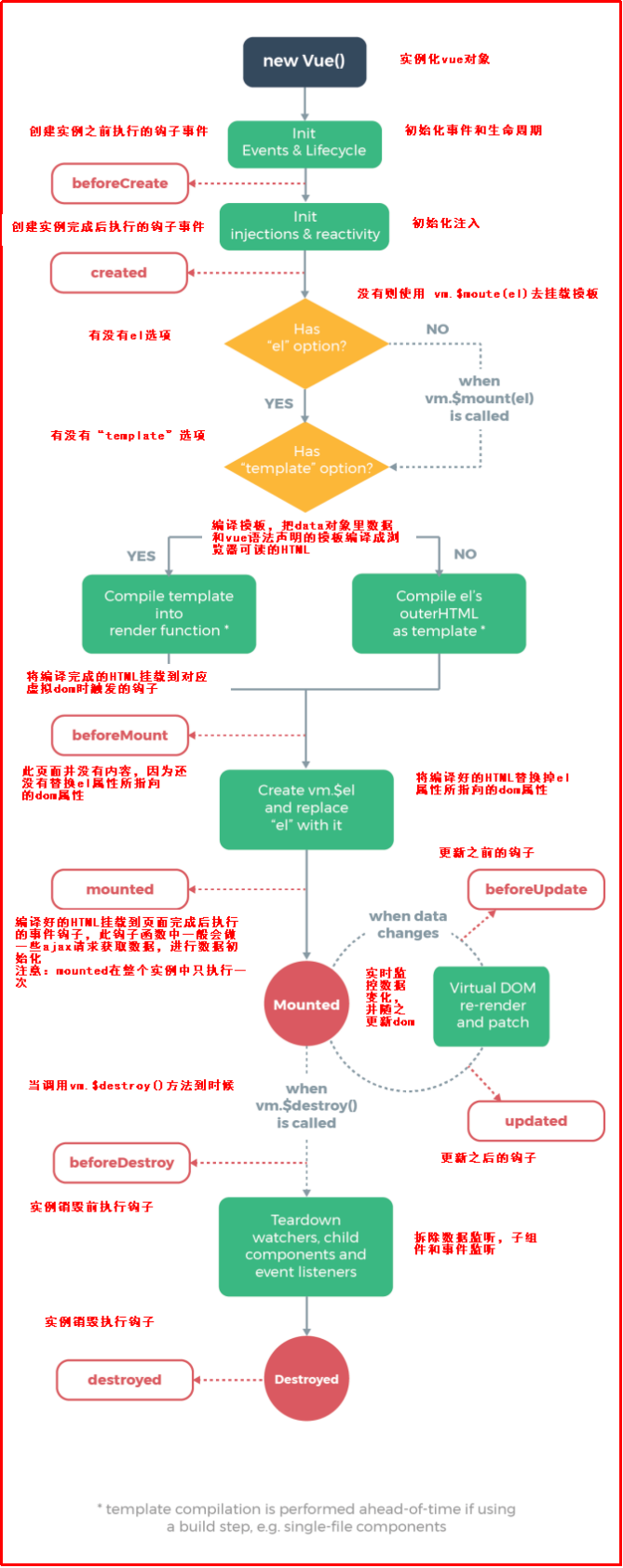
在生命周期执行的这个过程中会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。



比如 created 钩子可以用来在一个Vue实例被创建之后执行代码，例如ajax可以在created钩子函数下运行，使用ajax对页面数据初始化：



生命周期：



vue在生命周期中有这些钩子函数，

* beforeCreate，==created==
* beforeMount，==mounted==
* beforeUpdate，updated
* beforeDestroy，destroyed

Vue在实例化的过程中，会调用这些生命周期的钩子，给我们提供了执行自定义逻辑的机会。那么，在这些vue钩子中，vue实例到底执行了哪些操作，我们先看下面执行的例子

【需求】：演示vue对象的创建、赋值、显示、改值、销毁的全过程，即Vue的生命周期，同时使用钩子函数添加自己的业务逻辑

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** charset="utf-8" />

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

<**title**>vuejs生命周期</**title**>

</**head**>

<**body**>

<**div** **id**="app">

{{message}}

</**div**>

</**body**>

<**script**>

**var** vue = **new** Vue({

el: "#app",

//template:"<h1><font color='red'>你好硅谷</font></h1>", //在vue配置项中修改的

data: {

message: 'hello world'

},

beforeCreate: **function**() {

showData('创建vue实例前', **this**);

},

created: **function**() {

showData('创建vue实例后', **this**);

},

beforeMount: **function**() {

showData('挂载到dom前', **this**);

},

mounted: **function**() {

showData('挂载到dom后', **this**);

},

beforeUpdate: **function**() {

showData('数据变化更新前', **this**);

},

updated: **function**() {

showData('数据变化更新后', **this**);

},

beforeDestroy: **function**() {

showData('vue实例销毁前', **this**);

},

destroyed: **function**() {

showData('vue实例销毁后', **this**);

}

});

**function** **showData**(process, obj) {

console.log(process);

console.log('data 数据：' + obj.message)

console.log('vue挂载的虚拟dom对象：')

console.log(obj.$el)

console.log('真实dom结构：' + document.getElementById('app').innerHTML);

console.log('------------------')

console.log('------------------')

}

vue.message = "good...";

//setTimeout(function(){

vue.message = "good...";

//},3000)

//setTimeout(function(){

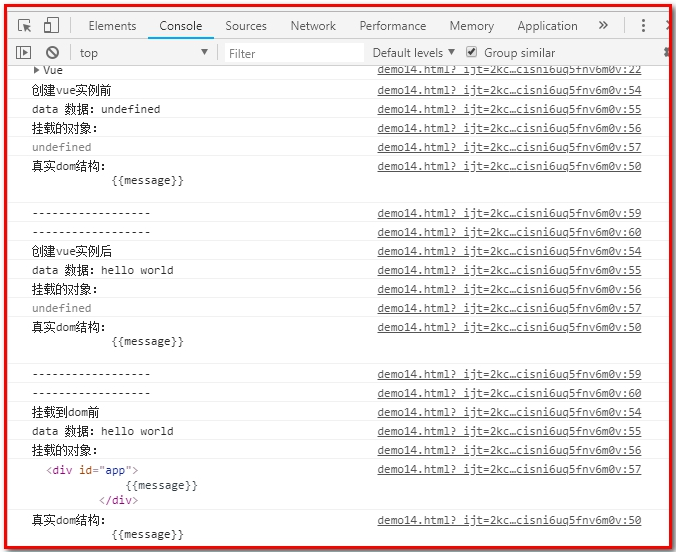
vue.$destroy(); // 销毁了监听，不会执行数据变化的监听，即不会将message的值改成good...

//},6000)

</**script**>

</**html**>

查看谷歌浏览器。







总结：

vue对象初始化过程中，会执行到beforeCreate,created,beforeMount,mounted 这几个钩子的内容

beforeCreate ：数据还没有监听，没有绑定到vue对象实例，同时也没有挂载对象

==created== ：数据已经绑定到了对象实例，但是还没有挂载对象（使用ajax可在此方法中查询数据，调用函数，页面的初始化）

beforeMount: 模板已经编译好了，根据数据和模板已经生成了对应的元素对象，将数据对象关联到了对象的el属性，el属性是一个HTMLElement对象，也就是这个阶段，vue实例通过原生的createElement等方法来创建这个html片段，准备注入到我们vue实例指明的el属性所对应的挂载点

==mounted== ：将el的内容挂载到了el，相当于我们在jquery执行了(el).html(el),生成页面上真正的dom，上面我们就会发现dom的元素和我们el的元素是一致的。在此之后，我们能够用方法来获取到el元素下的dom对象，并进行各种操作。

当我们的data发生改变时，会调用beforeUpdate和updated方法

beforeUpdate ：数据更新到dom之前，我们可以看到$el对象已经修改，但是我们页面上dom的数据还没有发生改变

updated ：dom结构会通过虚拟dom的原则，找到需要更新页面dom结构的最小路径，将改变更新到dom上面，完成更新

当调用vue.$destroy()，用来销毁vue。

beforeDestroy，destroed ：实例的销毁，vue实例还是存在的，只是解绑了事件的监听、还有model对象数据与view的绑定，即数据驱动

### 【小结】

1：什么是生命周期

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 、并在数据变化时更新 DOM 等。

2：什么是钩子函数

同时在这个过程中也会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。

* beforeCreate，==created（数据模型初始化，页面还没有加载数据）
* beforeMount，==mounted（页面已经加载数据）
* beforeUpdate，updated
* beforeDestroy，destroyed

3：演示vue对象的创建、赋值、显示、改值、销毁的全过程，即Vue的生命周期，同时也可以使用钩子函数添加自己的业务逻辑。

# ****第四章：VueJS ajax****

### 【目标】

学会使用VueJS的ajax，即axios，异步查询数据，传递json数据和响应json数据

### 【路径】

1：vue-resource（了解）

2：axios

3：页面引入axios

4：案例

（1）get请求

（2）post请求

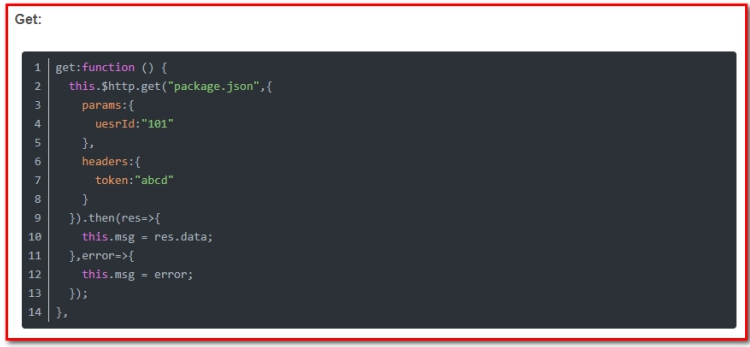
（3）编写代码，调用json数据

### 【讲解】

## 4.1 ****vue-resource****

vue-resource是Vue.js的插件提供了使用XMLHttpRequest或JSONP进行Web请求和处理响应的服务。 当vue更新到2.0之后，作者就宣告不再对vue-resource更新，而是推荐的axios，在这里大家了解一下vue-resource就可以。

vue-resource的github：<https://github.com/pagekit/vue-resource>

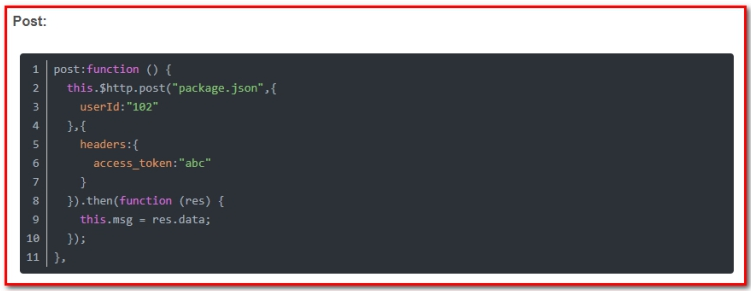


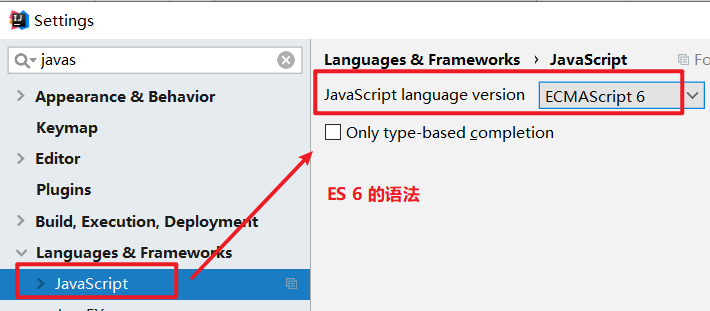
ES6的语法：

​ then(res=>{}) 等同于

ES5中的

​ then(function(res){}}





## 4.2 ****axios****

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中

axios的github：<https://github.com/axios/axios>

## 4.3 ****引入axios****

可以用script引入

<**script** **src**="https://unpkg.com/axios/dist/axios.min.js"></**script**>

## 4.4 ****案例****

### 4.4.1 ****get请求****

//通过给定的ID来发送请求

axios.get('/user?ID=12345').then(**function**(response){

console.log(response);

}).catch(**function**(err){

console.log(err);

}).finally(**function**(){

});

//以上请求也可以通过这种方式来发送

axios.get('/user',{

params:{

ID:12345

}

}).then(**function**(response){

console.log(response);

}).catch(**function**(err){

console.log(err);

}).finally(**function**(){

});

### 4.4.2 ****post请求****

axios.post('/user',{

name:'张三',

age:'22'

}).then(**function**(res){

console.log(res);

}).catch(**function**(err){

console.log(err);

}).finally(**function**(){

});

为方便起见，axios为所有支持的请求方法提供了别名

axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])

对比：没使用简化别名的代码（了解）



## 4.5 ****代码****

【需求】：创建data/user.json文件，使用axios调用json，将数据显示到页面上

第一步：创建data目录，创建user.json（后续我们从数据库查询，获取json数据）

[

{"**username**":"张三","**age**":22},

{"**username**":"李四","**age**":21},

{"**username**":"王五","**age**":20},

{"**username**":"赵六","**age**":23}

]

第二步：使用axios读取user.json文件的内容，并在页面上输出内容。

<!DOCTYPE html>

<**html**>

<**head**>

<**meta** **charset**="utf-8" />

<**title**>vuejs中axios数据调用</**title**>

<**script** **src**="js/vuejs-2.5.16.js"></**script**>

<**script** **src**="js/axios-0.18.0.js"></**script**>

</**head**>

<**body**>

<**div** **id**="app">

{{message}}

</**div**>

</**body**>

<**script**>

**var** vm = **new** Vue({

el: "#app",

data: {

message: 'helloworld'

},

methods: {

init: **function**(){

alert("传递的参数是："+**this**.message);

axios.get("./data/user.json").then(**function**(response){

// alert(response);

console.log(response);

alert(JSON.stringify(response));

alert(response.data[0].username);

})

}

},

created: **function**(){

**this**.init();

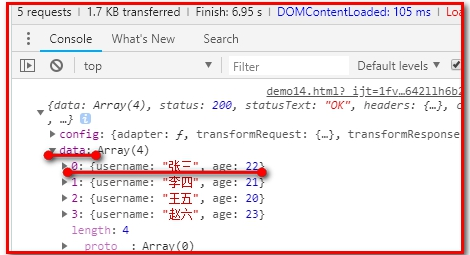
}

});

</**script**>

</**html**>

注意：response需要调用response.data才可以调用真正的json数据



### 【小结】

1：页面引入axios

2：学会get请求和post请求

（1）get请求

//通过给定的ID来发送请求

axios.get('/user?ID=12345').then(**function**(response){

console.log(response);

}).catch(**function**(err){

console.log(err);

}).finally(**function**(){

});

（2）post请求

axios.post('/user',{

name:'张三',

age:'22'

}).then(**function**(res){

console.log(res);

}).catch(**function**(err){

console.log(err);

}).finally(**function**(){

});

（3）编写案例代码，调用json数据