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

## VueJS介绍

Vue.js是一个构建数据驱动的 web 界面的渐进式框架。Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。它不仅易于上手，还便于与第三方库或既有项目整合。

官网:https://cn.vuejs.org/





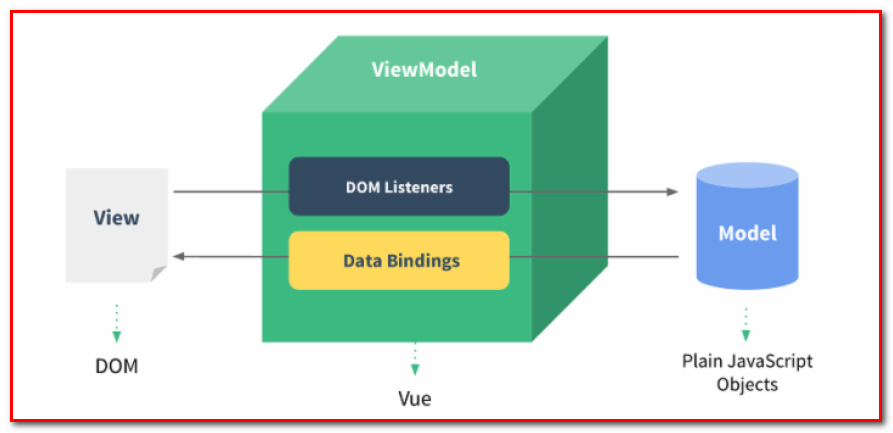
|  |
| --- |
| <!-- 开发环境版本，包含了有帮助的命令行警告 --> <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script> |

## MVVM模式

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）

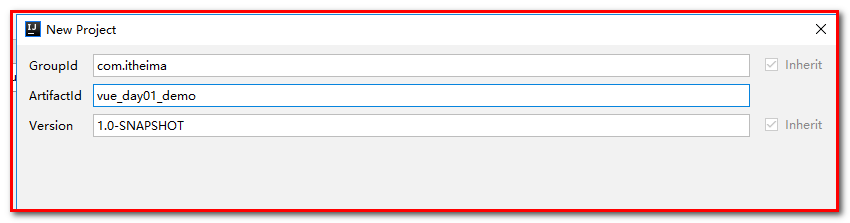
Vue.js 是一个提供了 MVVM 风格的双向数据绑定的 Javascript 库，专注于View 层。它的核心是 MVVM 中的 VM，也就是 ViewModel。 ViewModel负责连接 View 和 Model，保证视图和数据的一致性，这种轻量级的架构让前端开发更加高效、便捷



## VueJS 快速入门

### 创建工程：

web工程

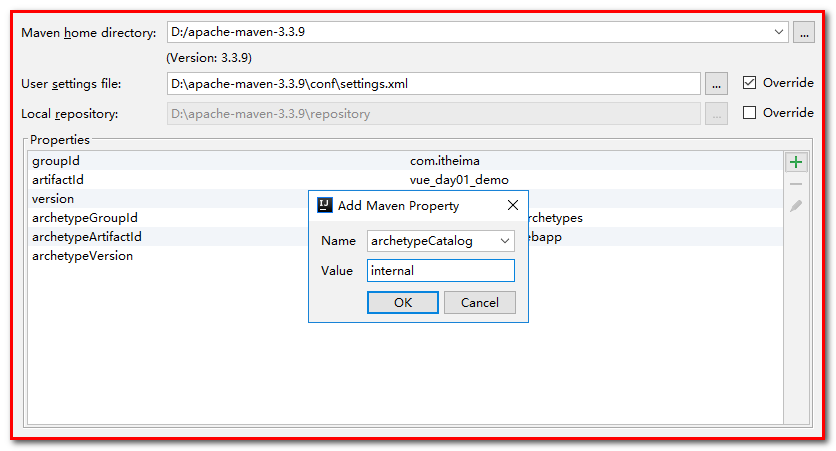


输入：

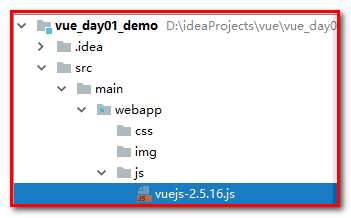
Name：archetypeCatalog

Value：internal

防止web项目加载过慢。



### 导入js



### 入门程序

demo1.html

【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**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
 <**body**>  
 *<!--定义id=app的目的是：app定义的div标签中可以使用vue-->* <**div id="app"**>  
 {{100+100+message}}<**br**>  
 {{message>10?"ok":"不ok"}}<**br**>  
 {{message1}}  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**:**"#app"**,*// 表示当前vue对象接管了div区域*

**data**:{  
 **message**:10, *//model* **message1**:**"helloworld"** *//model* }  
 });  
 </**script**>  
</**html**>

data ：用于定义属性，实例中有2个属性分别为：message和message1。数据类型定义数字、字符串、json。

methods： 用于定义的函数，可以通过 return 来返回函数值。

### 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。

{{ number + 1 }}

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

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 } }}

# 第二章：VueJS 常用系统指令

## 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

### v-on:click（等同于@click）

demo2.html

【需求】：点击按钮事件，改变message的值

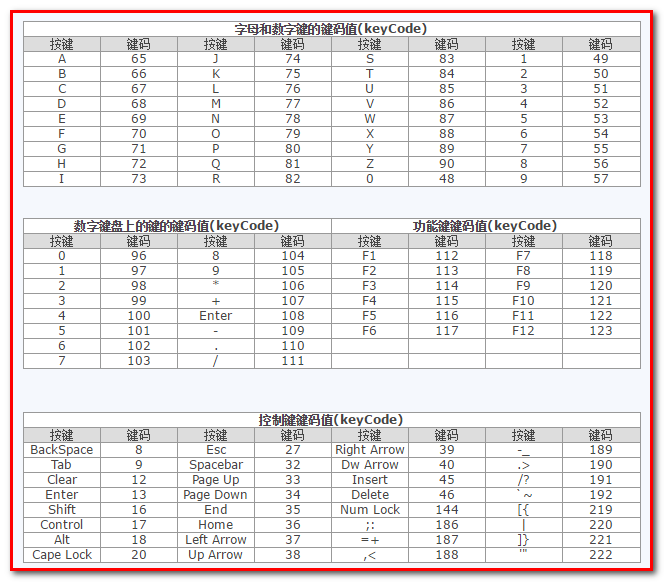
<!DOCTYPE **html**>  
<**html**>  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-on:click单击事件</**title**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
 <**body**>  
 <**div id="app"**>  
 {{message}}   
 *<!--<button v-on:click="fun('good')">改变</button>-->* <**button @click="fun('good')"**>改变</**button**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **var *vm*** = **new** *Vue*({  
 **el**:**"#app"**,  
 **data**:{  
 **message**:**"hello world"***//model* },  
 **methods**:{  
 fun:**function**(msg){  
 *// this代表的是vue对象，或者使用vm* **this**.**message**=msg;  
 }  
 }  
 });  
 </**script**>  
</**html**>

### v-on:keydown

表示键盘按下事件。

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

Keycode对照表



demo3.html

【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)"**>   
   
 </**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**;  
 **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。

### v-on:mouseover

鼠标移入区域事件

demo4.html

【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】：在div中添加<textarea>，鼠标移动到<textarea>，再弹出一个窗口

【需求3】：阻止上一层事件的执行

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-on:mouseover</**title**>  
 <**style**>  
 **#div** {  
 **background-color**: **red**;  
 }  
 </**style**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 <**div @mouseover="fun1" id="div" style="width**: 70%;**height**: 80%**"**>

12345  
 <**textarea @mouseover="fun2($event)"**>abc</**textarea**>  
 </**div**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**: **"#app"**,  
 **methods**:{  
 fun1:**function**(){  
 alert(**"div区域......"**);  
 },  
 fun2:**function**(e){  
 alert(**"textarea区域......"**);  
 *// 阻止冒泡（如果2个方法重叠，阻止上一层fun1方法的执行 ）* e.stopPropagation();  
 }  
 }  
 });  
 </**script**>  
  
</**html**>

### 事件修饰符（了解）

Vue.js 为 v-on 提供了事件修饰符来处理 DOM 事件细节，如：event.preventDefault() 或event.stopPropagation()。

Vue.js通过由点(.)表示的指令后缀来调用修饰符。

.stop // 停止触发，阻止冒泡修饰符（阻止上一层事件）

.prevent // 阻止事件发生，阻止事件默认行为

.capture // 捕获

.self //只点自己身上才运行

.once // 只执行一次

demo5.html

【需求】：在表单中，点击“提交”按钮，阻止执行（.prevent）；在超链接中，点击url，阻止上一层事件执行（.stop）

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-on:事件修饰符</**title**>  
  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 <**form v-on:submit.prevent action="http://www.itcast.cn"**>  
 <**input type="submit" value="提交"**>  
 </**form**>  
  
 <**div @click="fun1"**>  
 <**a @click.stop href="http://www.itcast.cn"**>itcast</**a**>  
 </**div**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**: **"#app"**,  
 **methods**: {  
 fun1:**function**(){  
 alert(**"hello"**);  
 }

*// fun2:function (e) {  
// e.preventDefault();  
// },  
// fun3:function (e) {  
// e.stopPropagation();  
// }*

}  
 });  
 </**script**>  
  
</**html**>

### 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

demo6.html

【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fun1事件-->* <**input type="text" @keydown.enter="fun1"**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**: **"#app"**,  
 **methods**: {  
 fun1:**function**(){  
 alert(**"输入的回车键！"**);  
 }

*// fun1:function (e) {  
// var keyCode = e.keyCode;  
// //alert(keyCode);  
// if(keyCode == 13){  
// alert("输入的回车键！")  
// }  
// }*  
 }  
 });  
 </**script**>  
  
</**html**>

## 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

demo7.html

【需求】：使用message属性赋值“**<h1>hello world</h1>**”，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**>

## v-bind

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

demo8.html

【需求】：使用vue定义属性**ys，**对页面中的字体标签<font>赋值颜色（color）。

使用vue定义属性**info**，对页面中的超链接<a>传递参数（href）。

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-bind</**title**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 <**font size="5" v-bind:color="ys1"**>传智播客</**font**>  
 <**font size="5" :color="ys2"**>黑马程序员</**font**>  
 <**hr**>  
 <**a v-bind={href:"http://www.itcast.cn/"+info}**>itcast</**a**>  
 </**div**>  
 </**body**>  
 <**script**>  
 **new** *Vue*({  
 **el**:**'#app'**, *//表示当前vue对象接管了div区域* **data**:{  
 **ys1**:**"red"**,  
 **ys2**:**"green"**,  
 **info**:**"subject/javaeezly/index.shtml"** }  
 });  
 </**script**>  
  
</**html**>

v-bind简写方式

<!-- 完整语法 -->

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

<!-- 缩写 -->

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

## v-model

用于读取数据。

demo9.html

【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，改变json数据，验证同时输入框的内容也发生改变。

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-model</**title**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 用户名:<**input type="text" v-model="user.username"**><**br**>

密码:<**input type="text" v-model="user.password"**><**br**>  
 <**input type="button" @click="fun" value="按钮"**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**: **"#app"**,  
 **data**: {  
 **user**: {  
 **username**: **"tom"**,  
 **password**: **"123"** }  
 },  
 **methods**: {  
 fun: **function**() {  
 alert(**this**.**user**.**username**+**" "**+**this**.**user**.**password**);  
 **this**.**user**.**username** = **"fox"**;  
 **this**.**user**.**password** = **"456"**;  
 }  
 }  
 });  
 </**script**>  
  
</**html**>

## v-for

用于操作array/集合，遍历

demo10.html

【需求1】：使用vue赋值list集合，分别存放数组和集合，并把数据遍历到页面上的<li>标签中。

<!DOCTYPE **html**>  
<**html**>  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-for遍历数组</**title**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
 <**body**>  
 <**div id="app"**>  
 <**ul**>  
 <**li v-for="(value,index) in arr"**>{{value}}----{{index}}</**li**>  
 </**ul**>  
 <**hr**>  
 <**ul**>  
 <**li v-for="(value,index) in list"**>

{{value.username}}----{{value.age}}----{{index}}

</**li**>  
 </**ul**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**:**"#app"**,  
 **data**:{  
 **arr**:[**'aaa'**,**'bbb'**,**'ccc'**], *//model* **list**:[{**username**:**"张三"**,**age**:18},{**username**:**"李四"**,**age**:22}]  
 }  
 });  
 </**script**>  
</**html**>

demo11.html

【需求2】：使用vue赋值json数据，并把数据遍历到页面上的<table>里的<tr>标签中。

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-for遍历对象</**title**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 <**table border="1"**>  
 <**tr**>  
 <**td**>序号</**td**>  
 <**td**>编号</**td**>  
 <**td**>名称</**td**>  
 <**td**>价格</**td**>  
 </**tr**>  
  
 <**tr v-for="(p,index) in products"**>  
 <**td**>{{index+1}}</**td**>  
 <**td**>{{p.id}}</**td**>  
 <**td**>{{p.name}}</**td**>  
 <**td**>{{p.price}}</**td**>  
 </**tr**>  
 </**table**>  
 </**div**>  
 </**body**>  
 <**script**>  
 *//view model* **new** *Vue*({  
 **el**: **"#app"**,  
 **data**: {  
 **products**: [{  
 **id**: **'itcast001'**,  
 **name**: **"电视机"**,  
 **price**: 1000  
 }, {  
 **id**: **"itcast002"**,  
 **name**: **"洗衣机"**,  
 **price**: 2000  
 }] *//model* }  
 });  
 </**script**>  
  
</**html**>

## v-if与v-show

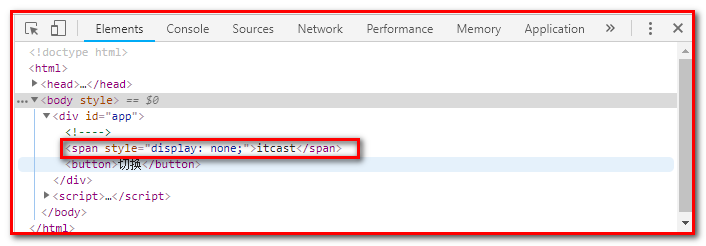
v-if是根据表达式的值来决定是否渲染元素

v-show是根据表达式的值来切换元素的display css属性

demo12.html

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

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="utf-8"** />  
 <**title**>v-if和v-show</**title**>  
 <**script src="js/vuejs-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 <**span v-if="flag"**>传智播客</**span**>  
 <**span v-show="flag"**>itcast</**span**>  
 <**button @click="toggle"**>切换</**button**>  
 </**div**>  
 </**body**>  
 <**script**>  
 **new** *Vue*({  
 **el**:**'#app'**, *//表示当前vue对象接管了div区域* **data**:{  
 **flag**:**false** },  
 **methods**:{  
 toggle:**function**(){  
 **this**.**flag**=!**this**.**flag**;  
 }  
 }  
 });  
 </**script**>  
  
</**html**>



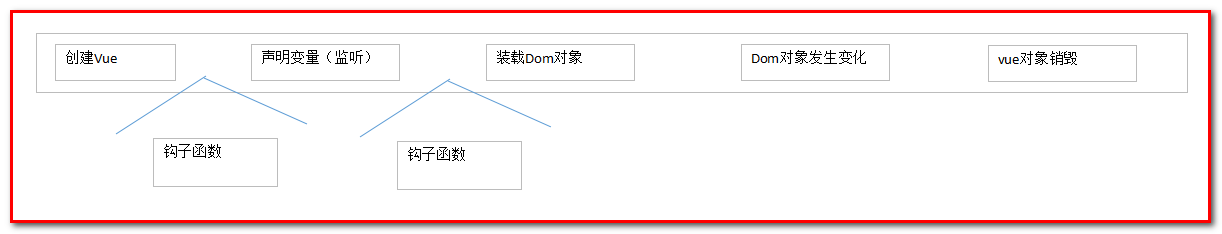
# 第三章：VueJS生命周期

什么叫生命周期？

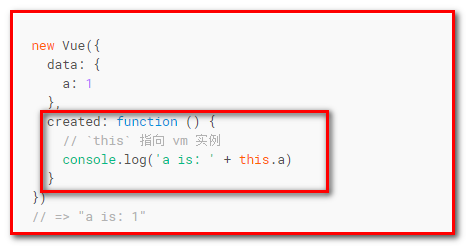
每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。

什么叫钩子函数？

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。



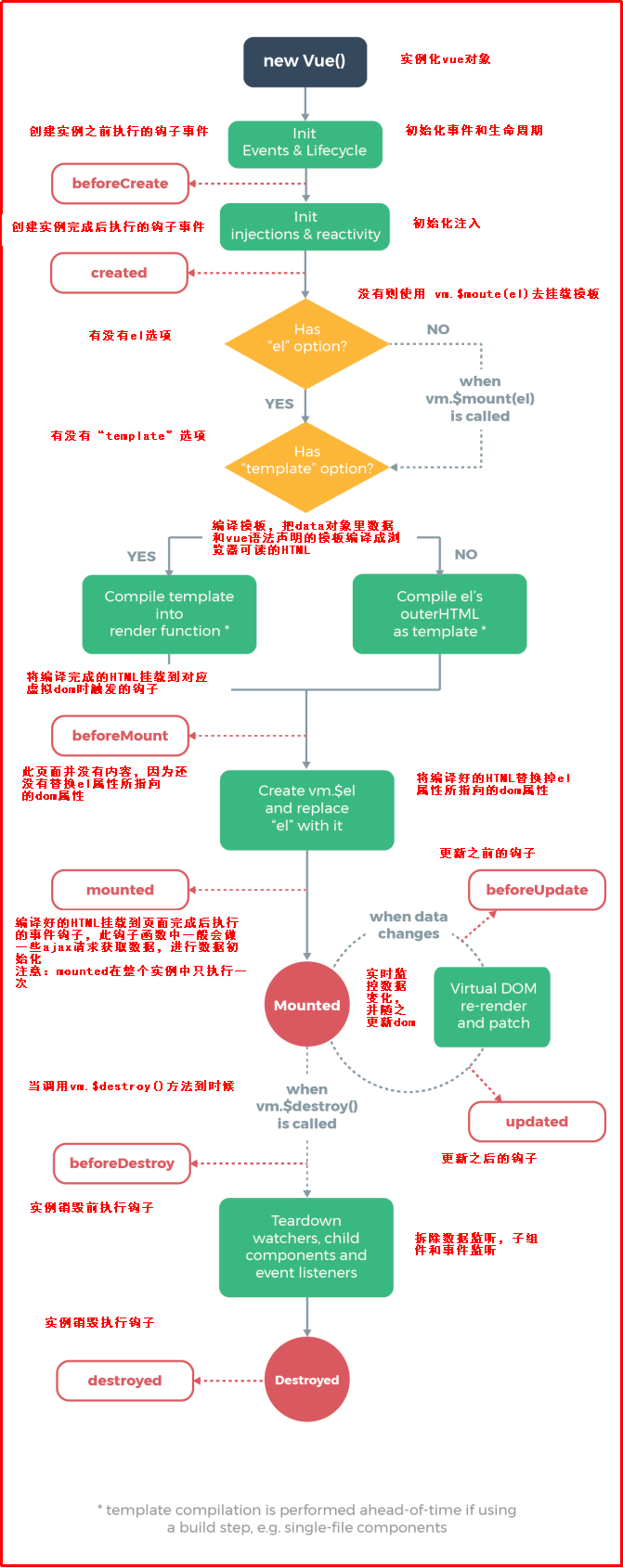
比如 [created](https://cn.vuejs.org/v2/api/" \l "created) 钩子可以用来在一个Vue实例被创建之后执行代码，例如ajax可以在created钩子函数下运行，对页面数据初始化：



生命周期



添加注释：



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

测试代码

demo13.html

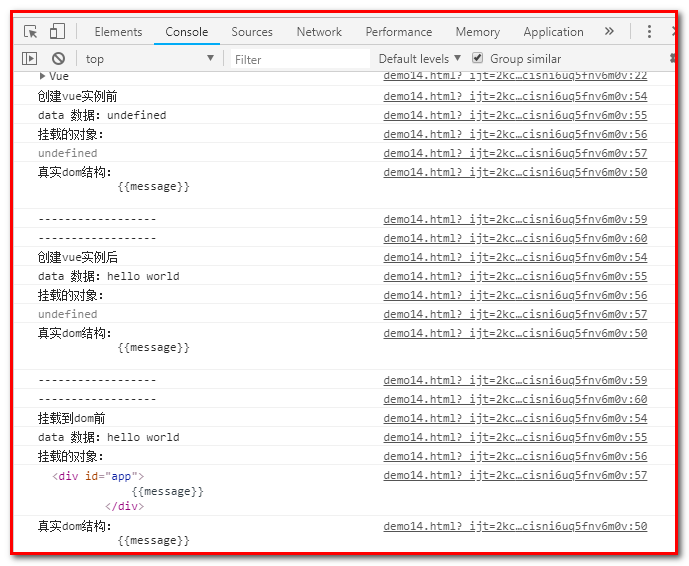
【需求】：演示vue对象的创建、赋值、显示、改值、销毁的全过程，即Vue的生命周期，同时使用钩子函数添加自己的业务逻辑

<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
   
 <**script src="js/vuej**<**meta charset="utf-8"** />  
 <**title**>vuejs生命周期</**title**>**s-2.5.16.js"**></**script**>  
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 {{message}}  
 </**div**>  
 </**body**>  
 <**script**>  
 **var *vue*** = **new** *Vue*({  
 **el**: **"#app"**,

*//template: "<fort color='red'>{{message +'！ 我爱 黑马程序员'}}</fort>", //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dom对象：'**)  
 **console**.log(obj.**$el**)  
 **console**.log(**'真实dom结构：'** + **document**.getElementById(**'app'**).**innerHTML**);  
 **console**.log(**'------------------'**)  
 **console**.log(**'------------------'**)  
 }

*//vue.$mount("#app");*  
 ***vue***.**message** = **"good..."**;  
 ***vue***.$destroy();  
 </**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

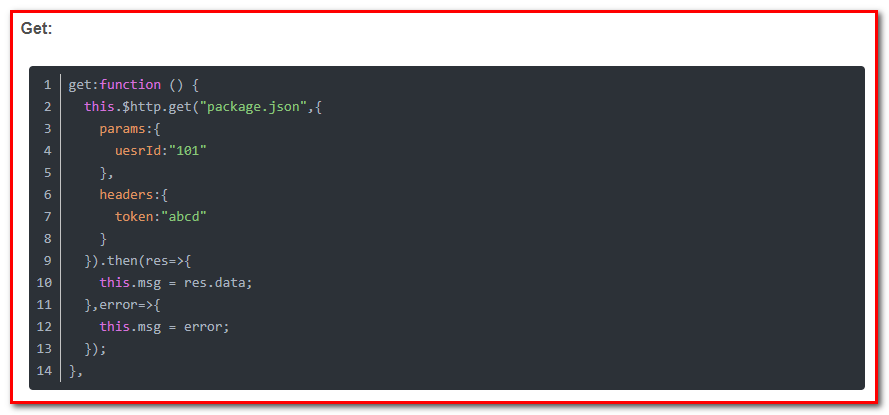
与view的绑定，即数据驱动

# 第四章：VueJS ajax

## 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。

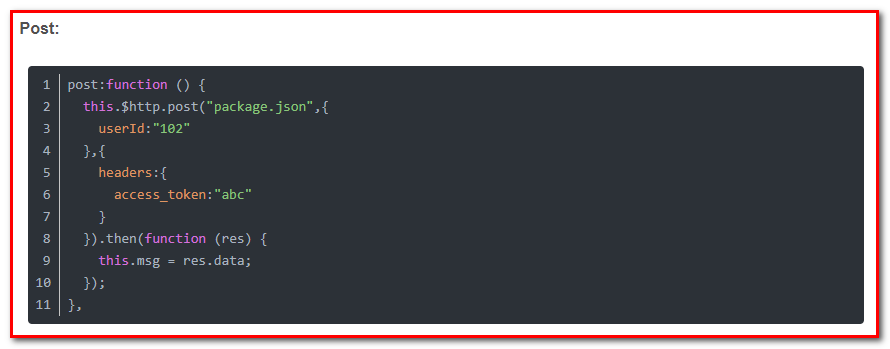
vue-resource的github: <https://github.com/pagekit/vue-resource>



ES6的语法：

then(res=>{}) 等同于

then(function(res){}}



## axios

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

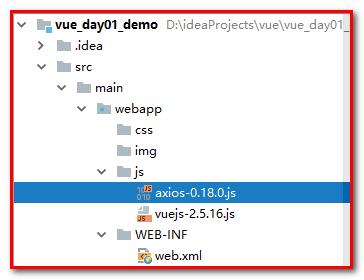
axios的github:https://github.com/axios/axios

## 引入axios

可以用script引入

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

引入axios-0.18.0.js



## 案例

### 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(){

});

### post请求

axios.post('/user',{

firstName:'Fred',

lastName:'Flintstone'

}).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])

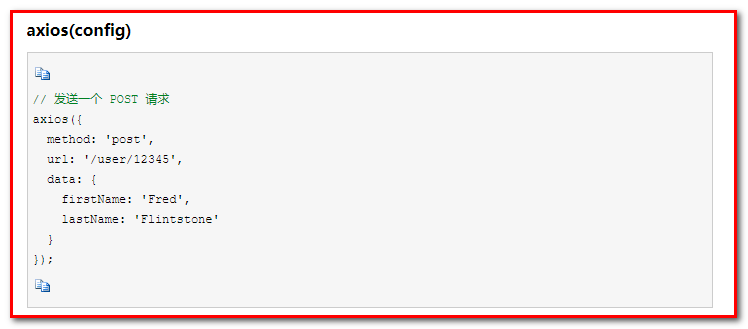
axios.head(url[, config])

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

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

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

对比：在没使用简化的方案的时候：以下代码



## 代码：

demo14.html

【需求】：创建data/user.json文件

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**>  
   
 </**head**>  
  
 <**body**>  
 <**div id="app"**>  
 {{message}}  
 </**div**>  
 </**body**>

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

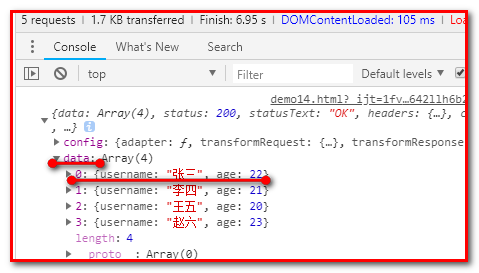
<**script src="js/axios-0.18.0.js"**></**script**>  
 <**script**>  
 **var *vm*** = **new** *Vue*({  
 **el**: **"#app"**,  
 **data**: {  
 **message**: **'helloworld'** },  
 **methods**: {  
 init: **function**(){

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**>

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}  
]



# 第五章：综合案例

## 案例需求

完成用户的查询与修改操作

前端（浏览器端）：vue

后端（服务器端）：spring boot+spring mvc+spring data jpa

Springboot：不需要在写spring的配置文件（xml），简化sprng开发。

Spring data jpa：用来操作数据库，很多方法都定义好了，不需要在sql语句，对象和数据库表的映射。

## 数据库设计与表结构

CREATE DATABASE itcastvue;

USE itcastvue;

CREATE TABLE USER(

id INT PRIMARY KEY AUTO\_INCREMENT,

age INT,

username VARCHAR(20),

password VARCHAR(50),

email VARCHAR(50),

sex VARCHAR(20)

)

导入数据：

INSERT INTO USER (username,PASSWORD,age,sex,email) VALUES ('张三','123',22,'男','zhangsan@163.com'),('李四','123',20,'女','lisi@163.com')

User类

public class User {

private Integer id;

private String username;

private String password;

private String sex;

private Integer age;

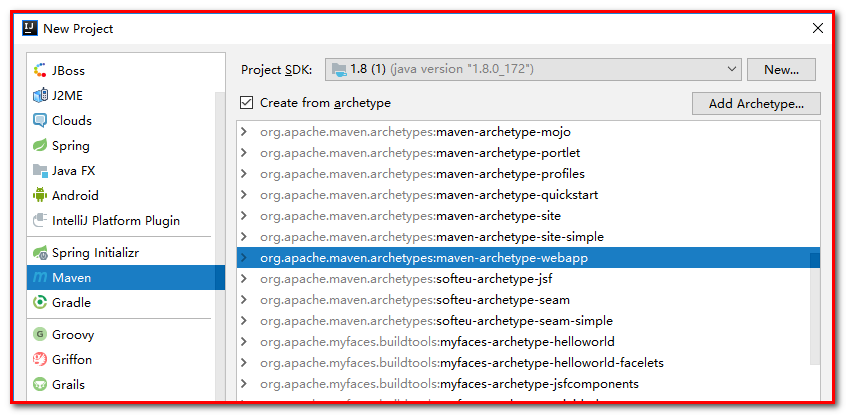
private String email;

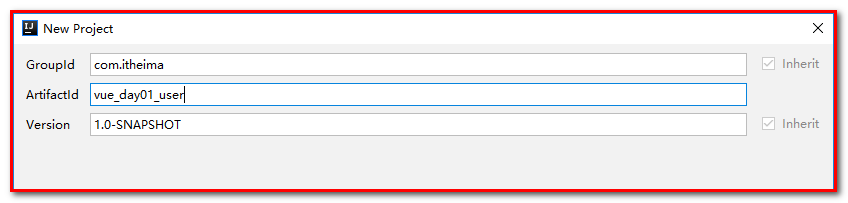
省略getter/setter

｝

## 服务器端

### 创建项目





### 导入jar包

pom.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>1.4.0.RELEASE</**version**>  
 </**parent**>  
  
 <**groupId**>com.itheima</**groupId**>  
 <**artifactId**>vue\_day01\_user</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**packaging**>war</**packaging**>  
  
 <**name**>vue\_day01\_user Maven Webapp</**name**>  
 *<!--* ***FIXME change it to the project's website*** *-->* <**url**>http://www.example.com</**url**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**maven.compiler.source**>1.8</**maven.compiler.source**>  
 <**maven.compiler.target**>1.8</**maven.compiler.target**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

### 配置文件（application.properties）

application.properties文件

**spring.datasource.driverClassName**=**com.mysql.jdbc.Driver  
spring.datasource.url**=**jdbc:mysql://127.0.0.1:3306/itcastvue  
spring.datasource.username**=**root  
spring.datasource.password**=**root  
  
  
spring.jpa.show-sql**=**true  
spring.jpa.database**=**MySQL  
spring.jpa.generate-ddl**=**true  
spring.jpa.hibernate.ddl-auto**=**update**

### 创建springboot引导类

**package** com.itheima;  
  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
**public class** MyApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(MyApplication.**class**, args);  
 }  
  
}

运行起步类（main方法），会自动启动内置的tomcat服务器，方便开发。

### 创建实体

创建包com.itheima.domain，创建类User.java

@Entity  
**public class** User **implements** Serializable {  
 @Id  
 @GeneratedValue(strategy=GenerationType.***IDENTITY***)  
 **private** Integer **id**;  
 **private** String **username**;  
 **private** String **password**;  
 **private** String **sex**;  
 **private int age**;  
 **private** String **email**;

}

@Entity：表示当前类是实体类

@Id：表示该字段是主键id

@GeneratedValue：表示该主键是自增长

### 创建Dao

创建包com.itheima.dao，创建接口UserDao.java

**package** com.itheima.dao;  
  
**import** com.itheima.pojo.User;  
**import** org.springframework.data.jpa.repository.JpaRepository;  
  
**public interface** UserDao **extends** JpaRepository<User, Integer> {  
  
}

### 创建Service

创建包com.itheima.service，创建接口UserService.java

**package** com.itheima.service;  
  
**import** com.itheima.pojo.User;  
  
**import** java.util.List;  
  
**public interface** UserService {  
  
 **public** List<User> findAll();  
   
 **public** User findOne(**int** id);  
   
 **public void** update(User user);  
}

创建接口的实现类，UserServiceImpl.java

**package** com.itheima.service.impl;  
  
**import** com.itheima.dao.UserDao;  
**import** com.itheima.pojo.User;  
**import** com.itheima.service.UserService;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.List;  
  
@Service

@Transactional  
**public class** UserServiceImpl **implements** UserService {  
  
 @Autowired  
 **private** UserDao **userDao**;  
   
 **public** List<User> findAll() {  
 **return userDao**.findAll();  
 }  
   
 **public** User findOne(**int** id) {  
 **return userDao**.findOne(id);  
 }  
   
 **public void** update(User user) {  
 **userDao**.save(user);  
 }  
  
}

注意：如果是高版本的spring data jpa，findOne方法的调用略有变化。

@Override  
 **public** User findOne(Integer id) {  
*// User user = new User();  
// user.setId(id);  
// Example<User> example = Example.of(user);*

*// User u = userDao.findOne(example).get();* **return userDao**.findById(id).get();  
 }

使用Example用来封装查询条件

### 创建Controller

创建包com.itheima.controller，创建类UserController.java

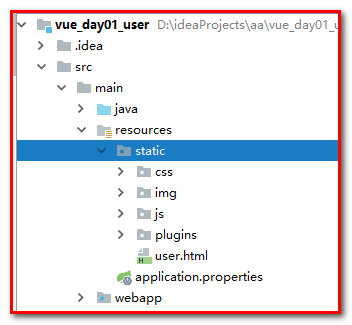
**package** com.itheima.controller;  
  
**import** com.itheima.pojo.User;  
**import** com.itheima.service.UserService;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.web.bind.annotation.PathVariable;  
**import** org.springframework.web.bind.annotation.RequestBody;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** java.util.List;  
  
@RestController  
@RequestMapping(**"/user"**)  
**public class** UserController {  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @RequestMapping(**"/findAll"**)  
 **public** List<User> findAll(){  
 **return userService**.findAll();  
 }

// 使用restful接收id属性值  
 @RequestMapping(**"/findOne/{id}"**)  
 **public** User findOne(@PathVariable(**"id"**) Integer id){  
 **return userService**.findOne(id);  
 }  
  
 @RequestMapping(**"/update"**)  
 **public void** update(@RequestBody User user){  
 **userService**.update(user);  
 }  
   
}

// 使用传统方式接收id属性值  
 @RequestMapping(**"/findOne"**)  
 **public** User findOne(Integer id){  
 **return userService**.findOne(id);  
 }

## 浏览器端

拷贝页面到resources下的static中



### user.html

我们把所有的vue的内容放置到user.js中

<**script src="./js/vuejs-2.5.16.js"**></**script**>  
<**script src="./js/axios-0.18.0.js"**></**script**>  
<**script src="./js/user.js"**></**script**>

### 查询所有

user.js

**var *vue*** = **new** *Vue*({  
 **el**:**'#app'**,  
 **data**:{  
 **userList**:[]  
 },  
 **methods**:{  
 findAll:**function**(){  
 **var** \_this=**this**; *//this代表是vue对象* **axios**.get(**"./user/findAll"**).then(**function**(response){  
*// this变成了window对象（可以使用\_this或者vue）* ***vue***.**userList** = response.**data**;  
 })  
 }},  
 created:**function**(){  
 **this**.findAll();  
 }  
   
})

user.html

<**tr v-for="u in userList"**>  
 <**td**><**input name="ids" type="checkbox"**></**td**>  
 <**td**>{{u.id}}</**td**>  
 <**td**>{{u.username}}  
 </**td**>  
 <**td**>{{u.password}}</**td**>  
 <**td**>{{u.sex}}</**td**>  
 <**td**>{{u.age}}</**td**>  
 <**td class="text-center"**>{{u.email}}</**td**>  
 <**td class="text-center"**>  
 <**button type="button" class="btn bg-olive btn-xs"**>详情</**button**>  
 <**button type="button" class="btn bg-olive btn-xs" data-toggle="modal" data-target="#myModal"** >编辑</**button**>  
 </**td**>  
</**tr**>

### 主键查询

user.html

<**td class="text-center"**>  
 <**button type="button" class="btn bg-olive btn-xs"**>详情</**button**>  
 <**button type="button" class="btn bg-olive btn-xs" @click="findOne(u.id)" data-toggle="modal" data-target="#myModal"** >编辑</**button**>  
</**td**>

user.js

**var *vue*** = **new** *Vue*({  
 **el**:**'#app'**,  
 **data**:{  
 **userList**:[],  
 **user**:{}  
 },  
 **methods**:{  
 findAll:**function**(){  
 **var** \_this=**this**; *//this代表是vue对象* **axios**.get(**"./user/findAll"**).then(**function**(response){  
*// this变成了window（可以使用\_this或者vue）* ***vue***.**userList** = response.**data**;  
 })  
 },  
 findOne:**function**(id){  
 **var** \_this=**this**; *//this代表是vue对象* **axios**.get(**"./user/findOne/"**+id).then(**function**(response){  
*// this变成了window，使用\_this* \_this.**user** = response.**data**;  
 })  
 }},  
 created:**function**(){  
 **this**.findAll();  
 }  
})

对应的Controller

// 使用restful接收id属性值  
 @RequestMapping(**"/findOne/{id}"**)  
 **public** User findOne(@PathVariable(**"id"**) Integer id){  
 **return userService**.findOne(id);  
 }

方案二：或者findOne方法可以使用正常传递参数：

*// get请求传递参数，方案一：***axios**.get(**"./user/findOne"**,{**params**:{**id**:id}}).then(**function** (resposne) {  
 alert(***JSON***.stringify(resposne.**data**));  
 \_this.**user** = resposne.**data**;  
}).**catch**(**function** (error) {  
  
})

对应的Controller

*// id查询*@RequestMapping(value = **"/findOne"**)  
**public** User findOne(Integer id){

User user = **userService**.findById(id);  
 **return** user;  
}

user.html

数据绑定回显。

<**div id="myModal" class="modal modal-primary" role="dialog"**>  
 <**div class="modal-dialog modal-lg"**>  
 <**div class="modal-content"**>  
 <**div class="modal-header"**>  
 <**button type="button" class="close" data-dismiss="modal" aria-label="Close"**>  
 <**span aria-hidden="true"**>**&times;**</**span**></**button**>  
 <**h4 class="modal-title"**>用户信息</**h4**>  
 </**div**>  
 <**div class="modal-body"**>  
  
 <**div class="box-body"**>  
 <**div class="form-horizontal"**>  
  
  
 <**div class="form-group"**>  
 <**label class="col-sm-2 control-label"**>用户名:</**label**>  
 <**div class="col-sm-5"**>  
 <**input type="text" class="form-control" v-model="user.username"**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label class="col-sm-2 control-label"**>密码:</**label**>  
 <**div class="col-sm-5"**>  
 <**input type="text" class="form-control" v-model="user.password"**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label class="col-sm-2 control-label"**>性别:</**label**>  
 <**div class="col-sm-5"**>  
 <**input type="text" class="form-control" v-model="user.sex"**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label class="col-sm-2 control-label"**>年龄:</**label**>  
 <**div class="col-sm-5"**>  
 <**input type="text" class="form-control" v-model="user.age"**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label class="col-sm-2 control-label"**>邮箱:</**label**>  
 <**div class="col-sm-5"**>  
 <**input type="text" class="form-control" v-model="user.email"**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 </**div**>  
 <**div class="modal-footer"**>  
 <**button type="button" class="btn btn-outline" data-dismiss="modal"**>关闭</**button**>  
 <**button type="button" class="btn btn-outline" data-dismiss="modal"**>修改</**button**>  
 </**div**>  
 </**div**>  
 *<!-- /.modal-content -->* </**div**>  
  
 *<!-- /.modal-dialog -->*</**div**>  
*<!-- /.modal -->*

### 更新

user.html

<**div class="modal-footer"**>  
 <**button type="button" class="btn btn-outline" data-dismiss="modal"**>关闭</**button**>  
 <**button type="button" class="btn btn-outline" @click="update()" data-dismiss="modal"**>修改</**button**>  
</**div**>

user.js

**var *vue*** = **new** *Vue*({  
 **el**:**'#app'**,  
 **data**:{  
 **userList**:[],  
 **user**:{}  
 },  
 **methods**:{  
 findAll:**function**(){  
 **var** \_this=**this**; *//this代表是vue对象* **axios**.get(**"./user/findAll"**).then(**function**(response){  
*// this变成了window（可以使用\_this或者vue）* ***vue***.**userList** = response.**data**;  
 })  
 },  
 findOne:**function**(id){  
 **var** \_this=**this**; *//this代表是vue对象* **axios**.get(**"./user/findOne/"**+id).then(**function**(response){  
*// this变成了window* \_this.**user** = response.**data**;  
 })  
 },  
 update:**function**(){  
 **var** \_this=**this**; *//this代表是vue对象* **axios**.post(**"./user/update"**,**this**.**user**).then(**function**(response){  
 \_this.findAll();  
 }).**catch**(**function**(err){  
 alert(**"修改失败："+err**);  
 })  
 }  
   
 },  
 created:**function**(){  
 **this**.findAll();  
 }  
   
   
   
})

访问：http://localhost:8080/user.html

