# **第一章 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**>

# ****第二章：VueJS 常用系统指令（重点）****

## 2.1 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

## 2.2 案例 todolist

### 2.2.1 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**>

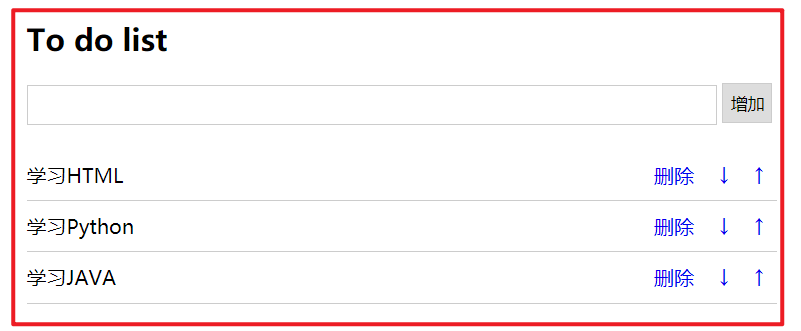
</**li**>

</**ul**>

</**div**>

</**body**>

</**html**>



# ****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？

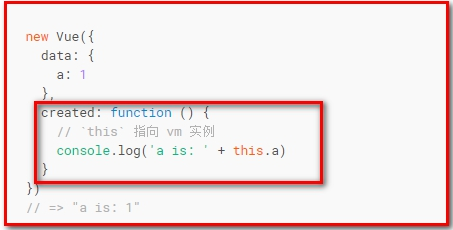
每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 、并在数据变化时更新 DOM 等。

什么叫钩子函数？

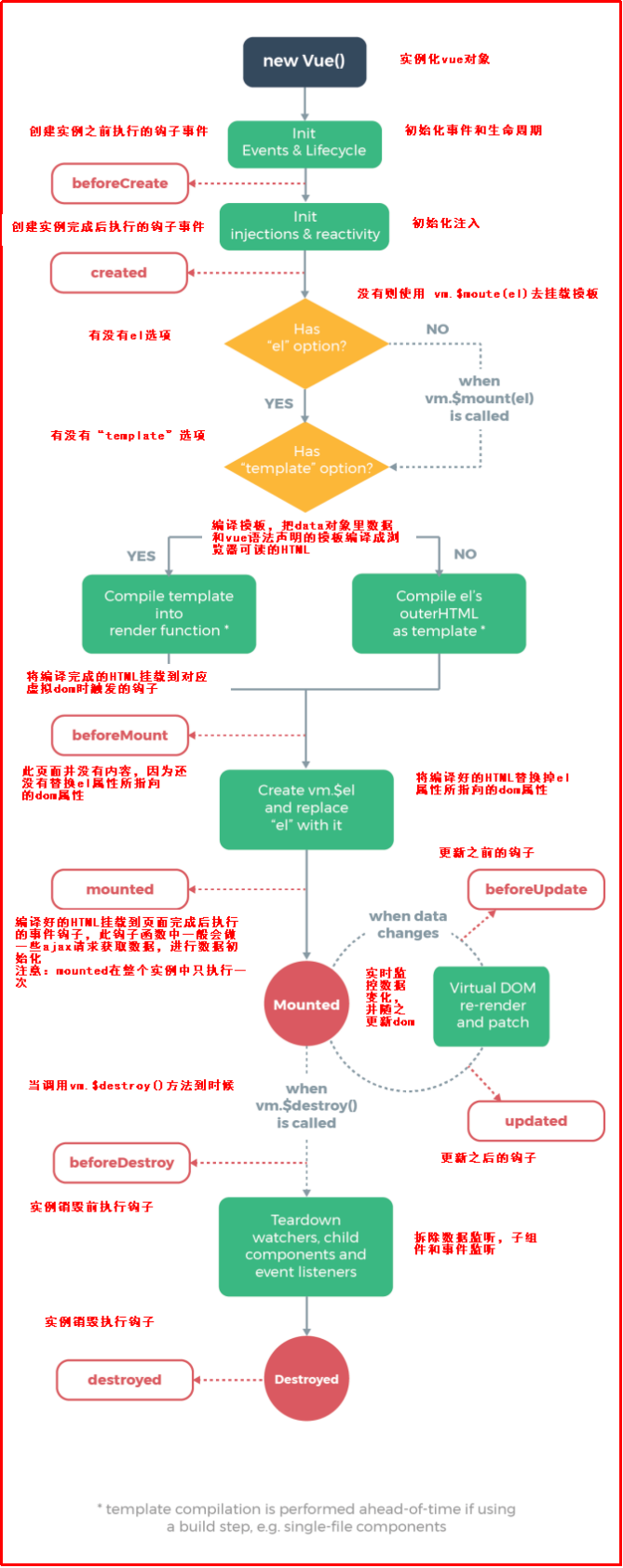
在生命周期执行的这个过程中会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。



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(){

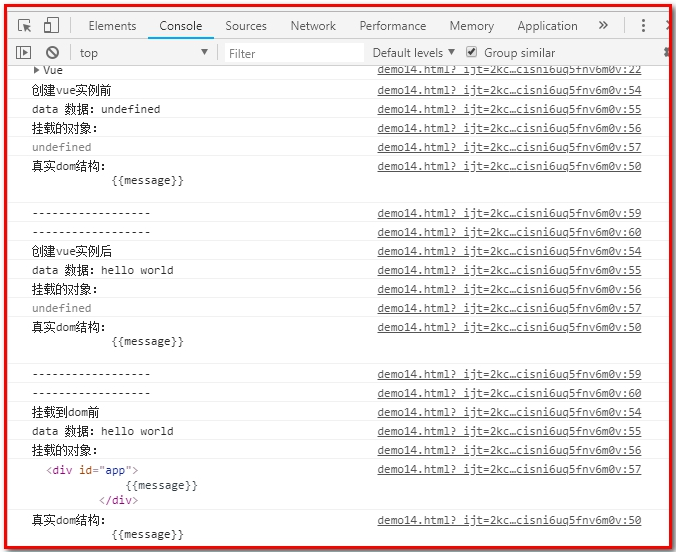
vue.$destroy(); // 销毁了监听，不会执行数据变化的监听，即不会将message的值改成good...

//},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，destroyed ：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

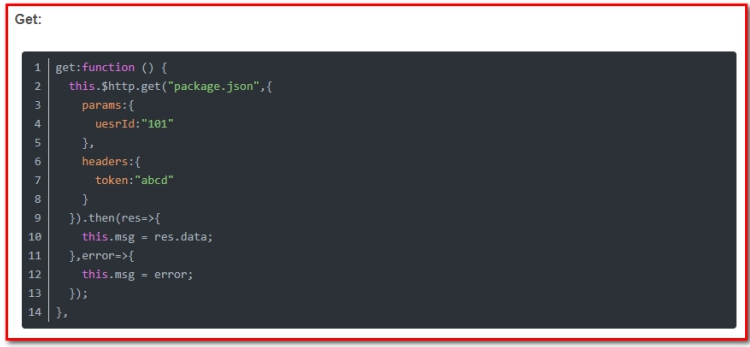
（3）编写代码，调用json数据

### 【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>

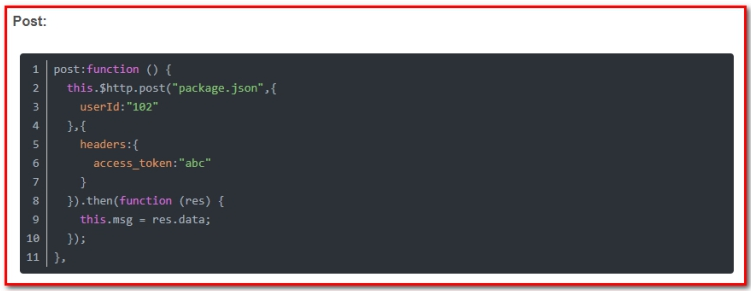


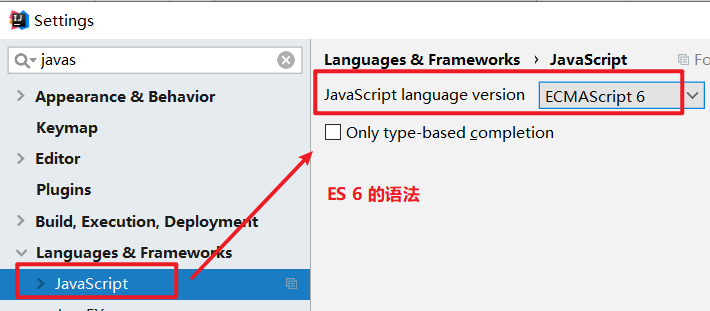
ES6的语法：

​ 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();

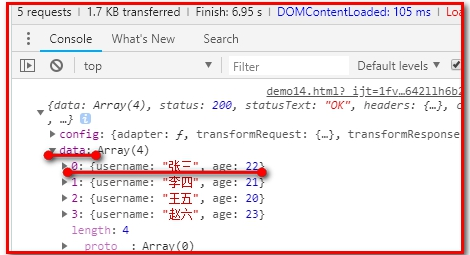
}

});

</**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