**目录**

[目录 1](#_Toc12341)

[JavaScript 3](#_Toc20527)

[1. 入口函数 3](#_Toc30047)

[2. 复选框操作 3](#_Toc31592)

[jQuery 5](#_Toc14767)

[1. 入口函数 5](#_Toc14889)

[2. jquery对象和dom对象相互转换 5](#_Toc16765)

[3. 选择器 5](#_Toc8488)

[4. is判断 5](#_Toc12644)

[5. 查找节点 6](#_Toc617)

[6. 查找节点属性 6](#_Toc7136)

[7. 创建节点 6](#_Toc13529)

[8. 插入节点 6](#_Toc22786)

[9. 删除节点 6](#_Toc438)

[10. 复制节点 6](#_Toc24767)

[11. 替换节点 6](#_Toc11816)

[12. 包裹节点 7](#_Toc25751)

[13. 属性操作 7](#_Toc2745)

[14. 判断是否含有某个class 8](#_Toc2995)

[15. 连续单击事件 8](#_Toc32663)

[16. 事件对象 event 8](#_Toc3236)

[17. 绑定/移除事件 9](#_Toc956)

[18. 动画方法 9](#_Toc31339)

[19. 复选框操作 9](#_Toc7787)

[20. 下拉框 11](#_Toc1652)

[21. 表格 11](#_Toc19889)

[22. Cookie插件 11](#_Toc18338)

[23. jQuery load() 方法 12](#_Toc16837)

[24. jQuery $.get() 方法 14](#_Toc22187)

[25. jQuery post() get() 方法 16](#_Toc7286)

[26. jQuery getScript() getJson() 方法 16](#_Toc21070)

[27. jQuery ajax() 方法 16](#_Toc7439)

[28. jQuery 全局事件 ajaxStart ajaxStop 19](#_Toc12834)

[29. jQuery 插件中的闭包 20](#_Toc9303)

[30. jQuery 插件编写 21](#_Toc17690)

[1. jQuery.extend() 21](#_Toc6553)

[2. jQuery.fn.extend() 23](#_Toc24818)

[31. jQuery 自动隐藏或关闭元素 23](#_Toc16597)

[32. jQuery 代码优化 24](#_Toc12823)

[33. jQuery获取回车事件 25](#_Toc21380)

[34. jQuery ajax规范写法 26](#_Toc22479)

[Angular 28](#_Toc30885)

[css 28](#_Toc5806)

[1.文本域textarea 28](#_Toc24032)

[2.渲染数据闪烁问题 28](#_Toc31640)

[Filter 29](#_Toc764)

[1.实现截取字符串长度过滤器 29](#_Toc23514)

[Vue 31](#_Toc4625)

[交互请求 31](#_Toc17195)

[1. 结合jquery 31](#_Toc12810)

[2. 使用vue-resource.js插件 31](#_Toc9876)

[组件间的消息分发 33](#_Toc28279)

[1. $emit、$on 33](#_Toc11134)

[css 34](#_Toc26308)

[1.表格隔行变色 34](#_Toc3214)

[2.侧边导航样式，动态展示是否为选中状态 34](#_Toc18510)

[3.vue渲染数据出现表达式闪烁问题 36](#_Toc12780)

**JavaScript**

1. 入口函数

Window.onload = function(){

//......

}

1. 复选框操作

$('#checkAll').click(function() {

$('[name = "items"]: checkbox').each(function(){

this.checked=!this.checked;

});

}); //反选

$("#checkAll").click(function(){

$('[name="items"]:checkbox').attr('checked',this.checked);

});//全选 全不选

//点击全选按钮 选中状态-->>复选框状态同样选中

//再次点击全选按钮 未选中状态-->>复选框状态同样未选中

$('[name="items"]:checkbox').click(function(){

var flag = true; $('[name="items"]:checkbox').each(function(){

if(!this.checked){flag = false;}

});

$("#checkAll").attr('checked',flag);

});//反选 下属复选框有一个是未选中状态，flag为flase，联动

//判断复选框是否被选中

var $e = $("#elementID"); //jquery变量

var e = $e.get(0); //dom变量

$.click(function(){

if(e.checked){//原生的javascript判断

//若选择 则·····

}

})

**jQuery**

1. 入口函数

$(document).ready(function(){

//......

});

可简写为：

$(function(){

//......

});

1. jquery对象和dom对象相互转换
   1. jquery对象转换为dom对象

var $c = $(“#c”); //jquery对象

var c = $c[0]; //dom对象

var c = $c.get(0); //dom对象

* 1. dom对象转换为jquery对象

var c = document.getElementById(“c”); //dom对象

var $c = $(c); //jquery对象

1. 选择器
   1. 内容过滤选择器：

contains选择器//匹配包含指定文本的元素

栗子：$("tr:contains('程')").addClass('selected');

1. is判断

$c.is(":checked") --> true false //当前是否为选中状态

$c.is(":animated") --> true false //当前是否处于动画状态

$c..is(".className") --> true false //当前手否含有className这个类

1. 查找节点

$("#id")

1. 查找节点属性

$("#id").attr("title")

1. 创建节点

$("<li></li>")

1. 插入节点

$("#id").append("<li>文本</li>")

appendTo()、prepend()、prependTo()、after()、insertAfter()、before()、insertBefore()

1. 删除节点

remove() //删除节点以及该节点的所有后代节点

empty() //清空节点

1. 复制节点

clone()//复制

clone(true) //复制该节点，并且复制该节点的事件

1. 替换节点

$(“p”).replaceWith(“<h2>被我替换</h2>”);

$(“<h2>被我替换</h2>”).replaceAll(“p”);

1. 包裹节点

<strong>元素1</strong>

<strong>元素2</strong>

* 1. .wrap() //包裹单个元素

$(“strong”).wrap(‘<b></b>’)

<b><strong>元素1</strong></b>

<b><strong>元素2</strong></b>

* 1. .wrapAll() //包裹全部元素

$(“strong”).wrapAll(‘<b></b>’)

<b>

<strong>元素1</strong>

<strong>元素2</strong>

</b>

* 1. .wrapInner(); //包裹内在

$(“strong”).wrapInner(‘<b></b>’)

<strong><b>元素1</b></strong>

<strong><b>元素2</b></strong>

1. 属性操作
   1. 获取属性和设置属性

var $attr = $(“#id”).attr(‘title’);

$(“p”).attr(“title”,”this is title”);

* 1. 删除属性

.removeAttr(“title”);

* 1. 切换样式

<p class=”class1”onclick=”toggleClass()”>你好</p>

function toggleClass(){

$(this).toggleClass(‘class2’);

}

<p class=”class1 class2”onclick=”toggleClass()”>你好</p>

<p class=”class1”onclick=”toggleClass()”>你好</p>

//点击后，在两个状态之间来回切换

1. 判断是否含有某个class

.hasClass(‘className’);

.is(‘.className’);//不仅仅可以判断是否含有类

1. 连续单击事件

toggle(fn,fn1,fn2,fn3....);

鼠标第一次点击触发fn，第二次点击触发fn1....直至最后一个。随后的每次单击，都重复对这几个函数轮番调用。

$(“#id”).toggle(function(){

},function(){

});

1. 事件对象 event

event.which(); //获取鼠标左中右键 1 2 3

event.type(); //获取该事件的类型 click....

event.target(); //获取到触发事件的元素 event.target.href

event.metakey(); //获取ctrl/Meta键

Demo:$(document).keydown(function(event) {

if(event.metaKey) {

alert("你按下了Meta键!");

}

});

1. 绑定/移除事件

bind();//绑定事件

one();//绑定事件，只在第一次生效

unbind();//移除事件

1. 动画方法

hide()、show() //同时修改多个样式 即高度、宽度和不同明度

fadeIn()、fadeOut() //只改变不透明度

slideUp()、slideDown() //只改变高度

fadeTo() //只改变不透明度

toggle() //用来代替hide()方法和show()方法，所以会同时修改多个样式属性，即高度、宽度和不透明度

slideToggle() //用来代替slideUp()方法和slideDown()方法，只更改高度

animate() //用于自定义动画的方法，以上各种动画方法实质都调用了animate()方法。除此之外，还能定义其他的样式属性，例如left、marginLeft、scrollTop等。可以使用该方法代替其他的所有动画方法。

1. 复选框操作

$('[name="item"]:checkbox').attr('checked',true);//设置选中

$('#checkAll').click(function() {

$('[name = "items"]: checkbox').each(function(){

$(this).attr('checked', !$(this).attr('checked');

});

});//反选

$("#send").click(function(){

var str;

$("[name='items']:checkbox:checked").each(function(){

str += $(this).val();

});

});//获得选中的值

$('[name="items"]:checkbox').click(function(){

//定义一个临时变量，避免重复使用同一个选择器选择页面中的元素，提高程序效率

var $tmp = $('[name="items"]:checkbox');

//使用filter()方法，筛选出 选中的复选框

$("#checkAll").attr('checked',

$tmp.length==$tmp.filter(':checked').length);

});//反选 联动

1. 下拉框

var $option = $("#slect option:slected");//获得选中的下拉框

var val = $option.val();//获得选中的下拉框的值

var $val = $("#slect").find("option:slected").val();//使用find()获取

1. 表格

$('tbody>tr').click(function(){

//声明 选中状态 Boolean

var hasSelected = $(this).hasClass('selected');

//三元运算 有 移除该类，复选框变为未选中状态

//无 添加该类 复选框为选中状态

$(this)[hasSelected ? 'removeClass' : 'addClass']('selected')

.find(':checkbox').attr('checked',!hasSelected);

});

1. Cookie插件

**1.添加一个"会话cookie"**

$.cookie('the\_cookie', 'the\_value');

这里没有指明 cookie有效时间，所创建的cookie有效期默认到用户关闭浏览器为止，所以被称为 “会话cookie（session cookie）”。

**2.创建一个cookie并设置有效时间为 7天**

$.cookie('the\_cookie', 'the\_value', { expires: 7 });

这里指明了cookie有效时间，所创建的cookie被称为“持久 cookie （persistent cookie）”。注意单位是：天；

**3.创建一个cookie并设置 cookie的有效路径**

$.cookie('the\_cookie', 'the\_value', { expires: 7, path: '/' });

在默认情况下，只有设置 cookie的网页才能读取该 cookie。如果想让一个页面读取另一个页面设置的cookie，必须设置cookie的路径。cookie的路径用于设置能够读取 cookie的顶级目录。将这个路径设置为网站的根目录，可以让所有网页都能互相读取 cookie （一般不要这样设置，防止出现冲突）。

**4.读取cookie**

$.cookie('the\_cookie');

**5.删除cookie**

$.cookie('the\_cookie', null);   //通过传递null作为cookie的值即可

**6.可选参数**

$.cookie('the\_cookie','the\_value',{//可选参数

    expires:7,

    path:'/',

    domain:'jquery.com',

    secure:true

})

**expires**：（Number|Date）有效期；设置一个整数时，单位是天；也可以设置一个日期对象作为Cookie的过期日期；  
**path**：（String）创建该Cookie的页面路径；  
**domain**：（String）创建该Cookie的页面域名；  
**secure**：（Booblean）如果设为true，那么此Cookie的传输会要求一个安全协议，例如：HTTPS；

1. jQuery load() 方法

jQuery load() 方法是简单但强大的 AJAX 方法。

load() 方法从服务器加载数据，并把返回的数据放入被选元素中。

**语法：**

$(*selector*).load(*URL*,*data*,*callback*);

必需的 URL 参数规定您希望加载的 URL。

可选的 data 参数规定与请求一同发送的查询字符串键/值对集合。

可选的 callback 参数是 load() 方法完成后所执行的函数名称。

**详细说明**

该方法是最简单的从服务器获取数据的方法。它几乎与 $.get(url, data, success) 等价，不同的是它不是全局函数，并且它拥有隐式的回调函数。当侦测到成功的响应时（比如，当 textStatus 为 "success" 或 "notmodified" 时），.load() 将匹配元素的 HTML 内容设置为返回的数据。这意味着该方法的大多数使用会非常简单：

$("#result").load("ajax/test.html");

如果提供回调函数，则会在执行 post-processing 之后执行该函数：

$("#result").load("ajax/test.html", function() {

alert("Load was performed.");

});

上面的两个例子中，如果当前文档不包含 "result" ID，则不会执行 .load() 方法。

如果提供的数据是对象，则使用 POST 方法；否则使用 GET 方法。

**加载页面片段**

.load() 方法，与 $.get() 不同，允许我们规定要插入的远程文档的某个部分。这一点是通过 url 参数的特殊语法实现的。如果该字符串中包含一个或多个空格，紧接第一个空格的字符串则是决定所加载内容的 jQuery 选择器。

我们可以修改上面的例子，这样就可以使用所获得文档的某部分：

$("#result").load("ajax/test.html #container");

如果执行该方法，则会取回 ajax/test.html 的内容，不过然后，jQuery 会解析被返回的文档，来查找带有容器 ID 的元素。该元素，连同其内容，会被插入带有结果 ID 的元素中，所取回文档的其余部分会被丢弃。

jQuery 使用浏览器的 .innerHTML 属性来解析被取回的文档，并把它插入当前文档。在此过程中，浏览器常会从文档中过滤掉元素，比如 <html>, <title> 或 <head> 元素。结果是，由 .load() 取回的元素可能与由浏览器直接取回的文档不完全相同。

**注释：**由于浏览器安全方面的限制，大多数 "Ajax" 请求遵守同源策略；请求无法从不同的域、子域或协议成功地取回数据。

**示例**

$("#div1").load("demo\_test.txt");

$("#div1").load("demo\_test.txt **#p1**");

$("button").click(function(){

$("#div1").load("demo\_test.txt",function(responseTxt,statusTxt,xhr){

if(statusTxt=="success")

alert("外部内容加载成功！");

if(statusTxt=="error")

alert("Error: "+xhr.status+": "+xhr.statusText);

});

});

与上面的实例类似，但是以 POST 形式发送附加参数并在成功时显示信息：

$("#feeds").load("feeds.php", {limit: 25}, function(){

alert("The last 25 entries in the feed have been loaded");

});

1. jQuery $.get() 方法

**定义和用法**

get() 方法通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 $.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 $.ajax。

**语法**

$(*selector*).get(*url*,*data*,*success(response,status,xhr)*,*dataType*)

|  |  |
| --- | --- |
| **参数** | **描述** |
| *url* | 必需。规定将请求发送的哪个 URL。 |
| *data* | 可选。规定连同请求发送到服务器的数据。 |
| *success(response,status,xhr)* | 可选。规定当请求成功时运行的函数。  额外的参数：   * response - 包含来自请求的结果数据 * status - 包含请求的状态 * xhr - 包含 XMLHttpRequest 对象 |
| *dataType* | 可选。规定预计的服务器响应的数据类型。  默认地，jQuery 将智能判断。  可能的类型：   * "xml" * "html" * "text" * "script" * "json" * "jsonp" |

**详细说明**

该函数是简写的 Ajax 函数，等价于：

$.ajax({

url: *url*,

data: *data*,

success: *success*,

dataType: *dataType*

});

根据响应的不同的 MIME 类型，传递给 success 回调函数的返回数据也有所不同，这些数据可以是 XML root 元素、文本字符串、JavaScript 文件或者 JSON 对象。也可向 success 回调函数传递响应的文本状态。

对于 jQuery 1.4，也可以向 success 回调函数传递 XMLHttpRequest 对象。

1. jQuery post() get() 方法

$.post(url,data,function(data,status,xhr){},dataType);

$.get(url,data,function(data,status,xhr){},dataType);

$.post('post3.php',{

'name':$("#name").val(),

'txt':$("#txt").val()

},function(data,status){

//.....

},"json");

1. jQuery getScript() getJson() 方法

$.getJSON(url, function(data, status) {});

$.getScript('getScript.js',function(data,status){});

1. jQuery ajax() 方法

语法

$.ajax(*{name:value, name:value, ... }*)

该参数规定 AJAX 请求的一个或多个名称/值对。

|  |  |
| --- | --- |
| **名称** | **值/描述** |
| async | 布尔值，表示请求是否异步处理。默认是 true。 |
| beforeSend(xhr) | 发送请求前运行的函数。 |
| cache | 布尔值，表示浏览器是否缓存被请求页面。默认是 true。 |
| complete(xhr,status) | 请求完成时运行的函数（在请求成功或失败之后均调用，即在 success 和 error 函数之后）。 |
| contentType | 发送数据到服务器时所使用的内容类型。默认是："application/x-www-form-urlencoded"。 |
| context | 为所有 AJAX 相关的回调函数规定 "this" 值。 |
| data | 规定要发送到服务器的数据。 |
| dataFilter(data,type) | 用于处理 XMLHttpRequest 原始响应数据的函数。 |
| dataType | 预期的服务器响应的数据类型。 |
| error(xhr,status,error) | 如果请求失败要运行的函数。 |
| global | 布尔值，规定是否为请求触发全局 AJAX 事件处理程序。默认是 true。 |
| ifModified | 布尔值，规定是否仅在最后一次请求以来响应发生改变时才请求成功。默认是 false。 |
| jsonp | 在一个 jsonp 中重写回调函数的字符串。 |
| jsonpCallback | 在一个 jsonp 中规定回调函数的名称。 |
| password | 规定在 HTTP 访问认证请求中使用的密码。 |
| processData | 布尔值，规定通过请求发送的数据是否转换为查询字符串。默认是 true。 |
| scriptCharset | 规定请求的字符集。 |
| success(result,status,xhr) | 当请求成功时运行的函数。 |
| timeout | 设置本地的请求超时时间（以毫秒计）。 |
| traditional | 布尔值，规定是否使用参数序列化的传统样式。 |
| type | 规定请求的类型（GET 或 POST）。 |
| url | 规定发送请求的 URL。默认是当前页面。 |
| username | 规定在 HTTP 访问认证请求中使用的用户名。 |
| xhr | 用于创建 XMLHttpRequest 对象的函数。 |

下面的表格中列出了可能的名称/值：

$.ajax({

type:"post",//get post

url:"",

async:true//异步请求

});

1. jQuery 全局事件 ajaxStart ajaxStop

// 1.9 以前写法 链式操作风格

$("#loading").ajaxStart(function() {

$(this).show();

}).ajaxStop(function() {

$(this).hide();

})

// 1.9+ 之后写法

$(document).ajaxStart(function() {

$("#loading").show();

}).ajaxStop(function() {

$("#loading").hide();

})

// 1.9 以前

$("#loading").on("ajaxStart ajaxStop", function() {

$(this).toggle();

})

// 1.9+ 以后

$(document).on("ajaxStart ajaxStop", function() {

$("#loading").toggle();

})

全局事件：

说明：1.9版本前后代码格式不同，发现原来在jquery1.9+版本以后，ajax全局事件需

要绑定到document对象上才能触发。

.ajaxComplete()//请求完成时执行的函数

.ajaxError()//请求发生错误时执行的函数

.ajaxSend()//请求发送前执行的函数

.ajaxStart()//请求开始执行的函数

.ajaxStop()//请求结束时执行的函数

.ajaxSuccess()//请求成功时执行的函数

1. jQuery 插件中的闭包

关于匿名函数：

function abc(x, y) {

return x + y;

}

alert(typeof function(x,y){return x+y;});// "function"

alert((function(x,y){return x+y;})(2,3));// "5"

//匿名函数

function haha() {}; // --> function(){//放置代码}

//所以匿名函数 被()包裹后，会立即执行;

(function() {

//放置代码

})(); // -->haha();

一般形式：

(function() {

//放置代码

})();

兼容性形式：

//为了更好的兼容性，在开始前 先加一个 ，

;(function($) { //此处$是形参

//放置代码 ，可以在函数内部可以使用 $作为jquery的简称

})(jQuery); //此处jquery作为实参，传递到该匿名函数中

1. jQuery 插件编写

jQuery.extend();

jQuery.fn.extend();

1. jQuery.extend()

jQuery.extend( [deep ], target, object1 [, objectN ] );

常用于设置插件的默认参数

function foo(option) {

option = jQuery.extend({

//默认参数

"name": "河东勋",

"age": "33",

"sex": "man"

}, option);

return option;

}

var opt = {

"name": "xuefeicheng",

"age": "23"

}

console.log(foo(opt));

//用于封装全局函数插件

;(function($){

$.extend({

ltrim:function(text){

return (text || "").replace(/^\s+/g,"");

},

rtrim:function(text){

return (text || "").replace(/\s+$/g,"");

}

});

})(jQuery);

//用于封装自定义选择器的插件

//使用jquery提供的$.extend()方法 编写 自定义选择器 插件

;

(function($) {

var $index = -1;

$.extend($.expr[':'], {

group: function(element, index, matches, set) {

var num = parseInt(matches[3], 10);

var x = $(element).index();

if(isNaN(num)) {

return false;

}

return x <= num;

}

});

})(jQuery);

1. jQuery.fn.extend()

//用于编写jquery对象方法插件的方法

;(function($) {

//对 对象的 扩展 使用

//设置 字体颜色的插件

$.fn.extend({

"color": function(value) {

return this.css('color', value);

}

});

})(jQuery);

1. jQuery 自动隐藏或关闭元素

//以1.4版本为节点

//在一段时间自动隐藏或关闭元素

//之前版本

setTimeout(function(){

$('div').fadeIn(400)

},2000);

//之后可以使用 dely方法

$('div').slideUp(300).delay(2000).fadeIn(400);

1. jQuery 代码优化

循环时的DOM操作：

var dom ='';

for(var i = 0 ; i < data.files.length;i++){

dom += '<div class="scjl">'+

'<div>'+data.files[i].name+

'</div><span onclick="delfile('+data.files[i].id+')">X</span>'+

'</div>';

}

$("#updateNames").append(dom);

var elem = $('#elem'), arr = [];

for(var i=0,l=data.crkls.length;i<l;i++){

arr.push('<tr>'

+'<td>'+ data.crkls[i].carbh +'</td>'

+'<td>'+ data.crkls[i].insj +'</td>'

+'<td>'+ data.crkls[i].crkmc +'</td>'

+'<td><button>操作</button></td>'

+'</tr>');

}

elem.append(arr.join(''));

1. jQuery获取回车事件

$(function(){

//全局：

document.onkeydown = function(e){ ?

var ev = document.all ? window.event : e;

if(ev.keyCode==13) {

// 如（ev.ctrlKey && ev.keyCode==13）为ctrl+Center 触发

//要处理的事件

}

}

//某个控件：

$("#id").keydown(function(e){

if(e.keyCode==13){

$("#FormId").submit();//处理事件

}

});

1. jQuery ajax规范写法

$.ajax({

url: "http://1.1.1.1/api/v2/user/inviters", //接口

type: "post", //GET或POST

data: JSON.stringify({}), //数据

headers: {

'Content-Type': 'application/json'

},

//dataType: 'json',

//返回的数据格式： json/xml/html/script/jsonp/text

success: function(msg) {

if (msg.data && msg.data.inviters) {

addFimg(msg.data.inviters);

}

}, //回调成功方法

error: function(xhr, textStatus) {

console.log(xhr);

console.log(textStatus)

}

});

**Angular**

**css**

1.文本域textarea

* 1. 禁止拖动

textarea{ resize:none;}

chrome中文本域textarea既没有黄框又不能拖拽的办法

textarea{outline:none;resize:none;}

H5新特性

设置disabled属性值为disabled

2.渲染数据闪烁问题

Vue中存在类似问题。

在使用angularjs开发评教移动端的时候，我们经常会看见在Chrome这类快速解析的浏览器上会闪过{{express}}。这是由于Javascript去操作DOM，都会等待DOM加载完成之后。同样，anguarjs等DOM加载完成之后才回去解析html，所以浏览器上会出现闪烁的情况。

解决这个问题，其实angularjs给了一个指令——ng-cloak。我们可以在需要的地方加上这个指令就可以。

[html]

<span style="font-size:18px;"><!DOCTYPE html>

<html ng-app>

<head>

    <meta charset="utf-8">

    <title>ng-bind directive</title>

</head>

<body ng-controller="HelloController">

<div ng-cloak>

    <p>直接输出字符串字面值</p>

    Hello {{"World"}}

    <hr>

</div>

</body>

<script src="js/angular-1.3.0.js"></script>

</html></span>

原理：

Ng-cloak实现原理为：页面初始化是在DOM的header增加一行css代码。

[html]

<span style="font-size:18px;"><style type="text/css">@charset "UTF-8";[ng\:cloak],[ng-cloak],[data-ng-cloak],[x-ng-cloak],.ng-cloak,.x-ng-cloak,.ng-hide{display:none !important;}ng\:form{display:block;}.ng-animate-start{clip:rect(0,auto,auto,0);-ms-zoom:1.0001;}.ng-animate-active{clip:rect(-1px,auto,auto,0);-ms-zoom:1;}

</style></span>

Angualr将带有ng-cloak的元素设置为display:none。等到angularjs解析到带有ng-cloak节点的时候，会把元素上ng-cloak  attribute和calss同时remove掉，这样就防止了节点的闪烁。

**Filter**

1.实现截取字符串长度过滤器

angular.module('ng').filter('cut', function () {

return function (value, wordwise, max, tail) {

if (!value) return '';

max = parseInt(max, 10);

if (!max) return value;

if (value.length <= max) return value;

value = value.substr(0, max);

if (wordwise) {

var lastspace = value.lastIndexOf(' ');

if (lastspace != -1) {

value = value.substr(0, lastspace);

}

}

return value + (tail || '');

};

});

**Vue**

**交互请求**

1. 结合jquery

$.ajax({

url: "/demo/menu/list", //接口

type: "post", //GET或POST

success: function(msg) {

new Vue({

el: '#menuList',

data: {menulist: msg.menus}

})

}, //回调成功方法

error: function(xhr, textStatus) {

console.log(xhr);

console.log(textStatus)

}

});

1. 使用vue-resource.js插件

说明：vue本身不具备$http服务，需要引用其他插件，还有axios

首先在github下载插件，在项目中调用

new Vue({

el: '#menuList'

,data: {menulist: ''}

,created: function () { //created方法，页面初始调用

var url = "/demo/menu/list";

this.$http.get(url).then(function (data) {

//ajax请求封装

var json = data.bodyText;

var usedData= JSON.parse(json);

//我的json数据参考下面

this.menulist = usedData.menus;

}, function (response) { //返回失败方法调用，暂不处理

console.info(response);

})

}

,methods:{}

})

var url = "/demo/user/insert";

this.$http.post(url,{

'userName':this.user.userName,

'nickName':this.user.nickName,

'password':this.user.password

},{emulateJSON:true}).then(function (data) { //ajax请求封装

console.log(data.body);

}, function (response) { //返回失败方法调用，暂不处理

console.info(response);

});

**组件间的消息分发**

1. $emit、$on

/\*定义一个消息分发中间件\*/

var bus = new Vue();

Vue.component("page",{

template:"#page",

created:function(){

var that = this;

bus.$on("myevent", function(msg) {

//通过$on("事件名称",function(参数列表){})来监听一个事件是否进行处理

// 接收到数据之后，就可以更新组件中使用的数据，然后展示到页面上

console.log("接收到其他组件传递的数据：" + msg);

that.allpage = Math.ceil(msg/5);

});

},

data:function(){},

methods:{}

})

new Vue({

el:'#dataBox',

data:{totalCount:''},

created:function(){

var url = "/demo/user/userList";

this.$http.post(url,{'currPage':1,'pageSize':10,'key':''})

.then(function (data) { //ajax请求封装

var json = data.bodyText;

var usedData= JSON.parse(json);

//我的json数据参考下面

this.totalCount = usedData.totalCount;

bus.$emit("myevent", this.totalCount);

//$emit("事件名称",[参数列表])通过$emit在代码中触发一个事件

}, function (response) { //返回失败方法调用，暂不处理

console.info(response);

})

},

methods:{}

})

**css**

1.表格隔行变色

<tbody>

<tr v-for='(t,index) in tabList' :class="{'odd':index%2==0}">

<td><input name="" type="checkbox" value="" /></td>

<td>{{t.userName}}</td>

<td>{{t.nickName}}</td>

</tr>

</tbody>

2.侧边导航样式，动态展示是否为选中状态

如图：



思路：

点击时，将全部列表的样式置为unactive。之后在设置被点击的元素的样式为active。【此处有一个注意点：vue在数组中不能检测到元素的更新，需要手动更新视图】

代码：

Html结构：

<dl>

<dd v-for='m in menulist'>

<div class="title">

<span><img src="o01.png" /></span>{{m.title}}

</div>

<ul class="menuson" >

<li v-for='(mm,index) in m.permissionList' @click="selectStyle(mm,index) " :class="mm.active? 'active':''" >

<cite></cite>

<a v-bind:href="mm.url" target="rightFrame">{{mm.name}}</a>

<i></i>

</li>

</ul>

</dd>

</dl>

Js：

,methods:{

selectStyle:function(item, index){

//在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。

this.$nextTick(function() {

this.menulist.forEach(function(item) {

if(item.permissionList.length>0){

item.permissionList.forEach(function(i){

Vue.set(i,'active',false)

});

}

Vue.set(item,'active',false)

});

Vue.set(item,'active',true)

});

}

}

3.vue渲染数据出现表达式闪烁问题

【注：vue1.x和vue2.x有区别】

查找官网api得到解决办法：

[v-cloak](https://cn.vuejs.org/v2/api/" \l "v-cloak" \o "v-cloak)指令

不需要表达式

用法：

这个指令保持在元素上直到关联实例结束编译。和 CSS 规则

如 [v-cloak] { display: none } 一起用时，这个指令可以隐藏未编译的 Mustache 标

签直到实例准备完毕。

示例：

[v-cloak]{display:none;}

<div v-cloak>

{{message}}

</div>

不会显示，直到编译结束。

但是有时候会不起作用，原因有以下：

1：检查css代码是否错误，没有“：”

2：v-cloak的display属性被更高的层级给覆盖掉了，需要在后面加 !important

3：需要行内引入该样式 或者link引入，样式放在了@import引入的css文件中不起作用

Vue1.x与vue2.x的使用区别：

Vue1 中，允许将 Vue 实例挂载在 body 上，而 Vue2 是不允许的，想对整个页面实例化，需要另外用一个 div 来容纳整个页面内容，对其进行实例化

这样在使用 v-cloak 时，同样需要用到这种方法