

Javascript

xbZhong

2025-03-15

[本页PDF](#)

Javascript的使用方式

- 行内式
 - `html css javascript <input type = 'button' value = '按钮' onclick = 'alert('OK')'>`
- 内嵌式
 - `html css javascript <script> // alert为弹出窗口 alert('NO') </script>`
- 外联式：写在文件里，引入文件进行使用

常见语句

- `alert`: 在网页弹出窗口
- `console.log`: 在控制台输出
- `prompt`: 提示用户输入信息
- `document.write`: 在body里面输入内容

```
// 弹出你好
alert("你好");
// 在控制台输出hello world
console.log("hello world");
// 提示用户输入信息
prompt('请输入你的姓名：');
// 在body输入一级标题
document.write('<h1>我是一级标题</h1>')
```

变量和数据类型

javascript是弱语言，不需要直接指定变量的类型

- 单行注释: `//`
- 多行注释: `/* */`
- 语句结束需要加`;`分号

定义变量的语法格式：

```
// 声明变量
var item = 45;
let item = 45;
const item = 45;
```

5种基本数据类型

1. number 数字类型
2. string 字符串类型
3. boolean 布尔类型 true或false
4. undefined 类型，变量声明未初始化
5. null 类型，表示空对象
6. object 复合类型
 - 复合类型定义用大括号
 - 内嵌的类型定义为： 变量名：变量值

使用typeof读取数据类型

```
// 声明变量
var num = 20;
var string = '黑马';
var unData;

// 声明复合类型
var operson = {
    name:'itcast',
    age:12
};

// 获取变量类型
alert(typeof(num)); //输出number
alert(typeof(string)); //输出string
alert(typeof(unData)); //输出undefined
alert(typeof(unData)); //输出object

// 获取数据 在控制台输出
console.log(operson.name);
console.log(operson.age);
```

字符串里插入变量

- 使用反引号`，变量名用\${}括起来

```
let name = 'whs';
let content = `大家好我是${name}`;
console.log(content) //输出: 大家好我是whs
```

函数的定义和调用

使用function定义函数

```
// 定义函数
function fnAlert(){
    alert('hello!');
};

function add(num1,num2){
    var irs = num1 + num2;
    return irs
};

// 函数调用
fnAlert();
var result = add(3,4);
alert(result); // 弹出result
```

条件语句

假如x = 5

比较运算符	描述	例子
==	等于	x == 8为false
====	全等 (值和类型)	x === 5为true;x === “5” 为false

```
var score = 79;
if(score < 60){
    alert('没及格');
}else{
    alert('及格');
}
```

逻辑运算符	描述
&&	且
	或
!	非

获取标签元素

命令	说明
getElementsByClassName()	获取具有指定类名的元素
getElementById()	获取具有特定Id的元素
getElementsByTagName()	获取具有特定标签名的所有元素
getElementByName()	获取具有指定name属性的所有元素

```
// 获取具有指定类名的元素
let class = document.getElementsByClassName('my_class');

// 获取具有特定Id的元素
let id = document.getElementById('my_id');

// 获取具有特定标签名的所有元素
let tagname = document.getElementsByTagName('div');

// 获取具有指定name属性的所有元素
let name = document.getElementsByName('my_name');
```

- 用内置对象**document**上的**getElementById方法**来获取页面上设置了id属性的标签元素，获取到的是一个html对象，然后把他赋值给一个变量
 - onload事件：页面数据加载完后触发
 - onclick事件：点击触发

- ```
// 获取id属性为btn的元素
var oBth = document.getElementById('btn');
alert(oBth);

// 定义一个函数方便随时调用
function fnload(){
 var oBth = document.getElementById('btn');
 alert(oBth);
}

// 使用onLoad方法
window.onLoad = fnload;
// 或者

window.onload = function(){
 var oBth = document.getElementById('btn');
 alert(oBth);
}

// 使用onClick来进行函数调用
<input type='button' value='按钮' id='bin' onclick='fnload()'>
```

## 使用CSS选择器获取元素

| 命令                 | 说明            |
|--------------------|---------------|
| querySelector()    | 获取一个元素        |
| querySelectorAll() | 获取多个元素，得到的是数组 |

```
// 获取一个元素
let div = document.querySelector('div');

// 用class选择器
let cla = document.querySelector('.class');

// 用id选择器
let id = document.querySelector('#id');

// 获取多个元素
let divs = document.querySelectorAll('div');
```

## 操作标签元素属性

### 标签属性和样式修改

```
// 获取标签对象
var obtn = document.getElementById('bin');
// 修改标签属性
obtn.value = 'user_name';
// 修改样式属性
obtn.style.background = 'red';

<input type='button' value='按钮' id='bin'>
```

## 属性名在js中的写法

1. html的属性和js里面的属性大多数都一样，但‘`class`’属性写成‘`className`’，
2. ‘`style`’属性里面的属性，有横杠的改成驼峰式，比如：‘`font-size`’改成‘`style.fontSize`’

## 读取或者设置标签包裹的内容

- **innerHTML**可以读取或者设置标签包裹的内容，包括html代码
- **innertext**可以读取不包含html代码的纯文本内容

```
window.onload = function(){
 // 获取标签对象
 var odiv = document.getElementById('mydiv');
 // 获取标签中的内容
 alert(odiv.innerHTML);
 // 修改标签的内容
 odiv.innerHTML = '你好';
}

<div id='mydiv'>我是一个标签</div>
```

```
// 修改纯文本内容
let content = document.querySelector('.class').innertext
content = '你好呀'

<div class='class'> 你好 </div>
```

## 数组及操作方法

数组里面的数据可以是**不同类型**的数据

- **数组的定义**

```
// 实例化对象方式创建
var alist = new Array(1,2,3);

// 字面量方式创建
var alist2 = [1,2,3, 'asd'];

// 多维数组
var alist3 = [[1,2,3],[4,5,6]];
```

- **获取数组长度**

```
var alist = [1,2,3,4];
alert(alist.length); // 输出4
```

- **根据下标取值**

```
var alist = [1,2,3,4];
alert(alist[0]); // 输出1
```

- **从数组最后添加和删除数据**

```
var alist = [1,2,3,4];
alist.push(5); // 追加数据
alert(alist); // 输出1, 2, 3, 4, 5
alist.pop(); // 删除数据
alert(alist) // 输出1, 2, 3, 4
```

- **根据下标添加和删除元素**

- `数组.splice(start,num,element1,...elementN)`
- `start`: 必需, 开始删除的索引
- `num`: 可选, 删除数组元素的个数
- `elementN`: 可选, 在`start`索引位置要插入的新元素

此方法会删除从`start`索引开始的`num`个元素，并将`elementN`参数插入到`start`索引位置

```
// 删除指定数据
var array = [10,20,30,40];
array.splice(0,1,'itcast');
console.log(array); // 输出'itcast',20, 30, 40
```

## 字符串拼接

使用“+”运算符

```
var inum1 = 10;
var inum2 = 11.1;
var str = 'abc';

result = inum1 + inum2;
alert(result); // 弹出21.1

result = inum2 + str; // 把数字转换为字符串再相加
alert(result); // 弹出11.1abc
```

## 定时器

**定时器的作用：**在一段特定的时间后执行某段程序代码

- **js定时器中有两种创建方式**
  - func：定时器要执行的函数名
  - delay：表示时间间隔，单位是毫秒，默认为0
  - param1：函数的参数

函数	作用
setTimeout(func[,delay,param1,param2])	以指定的时间间隔（以毫秒计）调用一次函数的定时器
setInterval(func[,delay,param1,param2])	以指定的时间间隔（以毫秒计）重复调用一个函数的定时器

- **清除定时器**
  - clearTimeout(timeoutID)：清楚只执行一次的定时器
  - clearInterval(timeoutID)：清楚反复执行的定时器
  - **timeoutID**为调用定时器函数获得的返回值

```
// 执行完后自动关闭定时器
function fnshow(name){
 alert(name);
 clearTimeout(Id);
}

var Id = setTimeout(fnshow,2000,'itcast');
```

# jQuery

是对JavaScript的封装，简化了JavaScript编程

## jQuery的引入

```
// 导入jQuery文件
<script src = 'jQuery的地址'> </script>
```

## jQuery的入口函数

jQuery提供了ready函数保证标签元素获取没有问题

两种写法

```
// 完整写法
$(document).ready(function(){
 ...
});

// 简化写法
$(function(){
 ...
});
```

## jQuery选择器

jQuery选择器的种类：

1. 标签选择器
2. 类选择器
3. id选择器
4. 层级选择器
5. 属性选择器

```
$('#myid') // 选择id为myid的标签
$('.myClass') // 选择class为myclass的标签
$('li') // 选择所有li标签
$('#ul1 li span') // 选择id为ul1标签下的所有li标签下的span标签
$('input[name=first]') // 选择name属性为first的input标签
```

使用变量接收时要在变量名前面加上 \$

```
$(function(){
 var $myp = $("p");
 alert($myp.length); // 弹出p标签的个数
});

$(function(){
 var $myobject = $("div p");
 alert($myobject.length); // 弹出div标签下的p标签的个数
};

$(function(){
 var $myclass = $('.myclass');
 alert($myclass.length); // 弹出class名为myclass的标签的个数
});
```

## 选择集过滤

- **has(选择器名称)方法：**表示选取包含指定选择器的标签
- **eq(索引)方法：**表示选取指定索引的标签

```
$(function(){
 // 获取标签对象
 var $myobject = $("div");
 // 进行样式修改
 $myobject.css({"height":"100px"});
 // has方法
 $myobject.has("#bin").css({"background":"red"});
 // eq方法
 $myobject.eq(0).css({"background":"red"}); //更改第1个div标签
});

<div> <input type = 'button' value = '按钮'> </div>
<div> <input type = 'button' value = '按钮_ '> </div>
```

# 选择集转移

以选择的标签为参照，然后获取转移后的标签

## 具体操作

- `$("#box").prev();` 表示选择id是box元素的上一个同级元素
- `$("#box").prevAll();` 表示选择id是box元素的上面所有的同级元素
- `$("#box").next();` 表示选择id是box元素的下一个同级元素
- `$("#box").nextAll();` 表示选择id是box元素的下面所有的同级元素
- `$("#box").parent();` 表示选择id是box元素的父级元素
- `$("#box").children();` 表示选择id是box元素的所有子元素
- `$("#box").siblings();` 表示选择id是box元素的其它同级元素
- `$("#box").find('.myclass');` 表示选择id是box元素的class为myclass的元素

```
// 获取标签元素
var $mydiv = $('#div');
// 获得上一个同级元素
alert($mydiv.prev());
// 获得上面的所有元素
alert($mydiv.prevAll());
// 获得下一个元素
alert($mydiv.next());
// 获得下面的所有元素
alert($mydiv.nextAll());
// 获得所有父级元素
alert($mydiv.parent());
// 获得所有同级元素
alert($mydiv.siblings());
// 获得所有子元素
alert($mydiv.children());
// 获得class为myclass的元素
alert($mydiv.find('.myclass'));
```

# 获取和设置元素内容

- **html方法的使用：**
  - `.html()`：获取和设置标签的内容，但会把之前的内容清空
  - `.append()`：追加内容

```
// 获取标签对象
var $mydiv = $('#div');
// 获取标签内容
var $result = $mydiv.html();
alert($result);
// 清空并修改标签内容
$mydiv.html('你好呀');
// 追加标签内容
$mydiv.append('我真帅');
```

## 获取和设置元素属性

- 标签样式: height, weight, font-size相关的
  - 使用 `css` 方法给标签设置标签样式
- 标签属性: type, value, id等标签中的属性
  - 使用 `prop` 方法设置标签属性

```
$(function(){
 // 获取标签对象
 var $my = $("p");
 // 获取标签样式
 var $style = $my.css('font-size');
 // 修改标签样式
 $my.css({"background":'red'});
 // 获得标签对象
 var $myt = $('#btn');
 // 获取标签属性
 var $result = $myt.prop('type');
 alert($result);
 // 设置标签属性
 $myt.prop({'value':'name'});
 // 或者
 $myt.val('name');
})

<p>itcast</p>
<input type = 'button' value = '按钮' id = 'btn'>
```

## jQuery事件

### 常用事件:

- `click()`: 鼠标点击
- `blur()`: 元素失去焦点
- `focus()`: 元素获得焦点
- `mouseover()`: 鼠标进入 (进入子元素也触发)
- `mouseout()`: 鼠标离开 (离开子元素也触发)

- ready(): DOM加载完成

```
// ready方法
$(function(){
 // 获取标签对象
 var $mytext = $('#text1');
 var $mybutton = $('#btn1');
 // click方法
 $mybutton.click(function(){
 alert('successfully')
 });
 // focus()元素获取焦点
 $mytext.focus(function(){
 // this指的是$mytext
 $(this).css({"background":"red"});
 });
 // blur()元素失去焦点
 $mytext.blur(function(){
 $(this).css({"background":"white"});
 });
 // mouseover()鼠标进入
 $mydiv = $('div');
 $mydiv.mouseover(function(){
 $(this).css({"background":"red"});
 });
 // mouseout(): 鼠标离开
 $mydiv.mouseout(function(){
 $(this).css({"background":"white"});
 });
})
<div>
 <input type = 'text' id = 'text1'>
 <input type = 'button' value = '按钮' id = 'btn1'>
</div>
```

## 事件代理

- 利用事件冒泡的原理(事件会向他的父级一级一级传递), 把事件加到父级上
- 使用父元素来代理子元素的事件, 可以**减少事件绑定的次数**
- 使用 `delegate`方法 来完成
  - `delegate(childSelector,event,function)`
    - childSelector: 子元素的选择器
    - event: 事件名称
    - function: 当事件触发执行的函数

```
$(function(){
 var $myul = $('ul')
 $myul.delegate('li','click',function(){
 $(this).css({"background":"red"})
 });
});
```

## javascript对象

创建方式：

- 通过顶级object类型来实例化一个对象

- ```
var person = new object();

// 添加属性
person.name = 'tom';
person.age = '25';

// 添加方法
person.sayname = function(){
    alert(this.name);
}

// 调用属性和方法
alert(perosn.age);
person.sayname()
```

- 通过对对象字面量创建一个对象

- ```
var person = {
 name: 'tom';
 age: '26';
 sayname:function(){
 alert(this.name);
 }
}

// 调用属性和方法
alert(person.name);
person.sayname();
```

## Json

数据交换语言(本质上是字符串), 浏览器和服务器之间交换数据时使用, 便于接收数据和发送数据

- **对象格式:**

- 对象格式的json数据, 使用一对大括号, 大括号里面放入key: value形式的键值对, 多个键值对使用逗号分隔

- **对象格式的JSON数据:**

- ```
{  
    "name": "tom",  
    "age": 18  
}
```

- **数组格式的JSON数据:**

- 用中括号括起来, 里面的数据用逗号分隔

- ```
[{"name": "老王", "age": 18}, {"name": "老李", "age": 23}]
```

- JSON数据转化成JavaScript对象

- 使用 `JSON.parse()` 方法

- ```
var sjson = '{  
    "name": "tom",  
    "age": 18  
}';  
var json = '[{"name": "老王", "age": 18}, {"name": "老李", "age": 23}]';  
  
// 浏览器获取时转换成JavaScript对象  
var operson = JSON.parse(sjson);  
  
// 浏览器获取时转换成数组  
var list = JSON.parse();
```

ajax

- 可以让JavaScript发送异步的http请求, 与后台进行数据的获取, 最大的优点是实现局部刷新

- 异步: 意味着页面在发出请求后不需要等待服务器响应, 用户仍然可以继续操作。

- **ajax的实现**

- 使用 `$.ajax()` 方法

Axios

封装在官方提供的Axios文件里

使用的时候要引入Axios的js文件

```
// 请求示例
axios({
  url: '',
  method: 'GET',
}).then((result) =>{
  console.log('你好');
}).catch((err) =>{
  console.log(err);
})

// 请求别名
// 格式: axios.请求方式(url[,data[,config]])
axios.get('').then((result) =>{
  console.log('调用成功');
}).catch((err)>{
  console.log(err);
})
```

async/await

- 将异步代码转化为同步代码，在声明方法的时候使用，通常需要和await一起使用
- 不阻塞主线程，只暂停当前代码的执行，等待Promise变成rejected或resolved才继续执行代码

示例

```
async search(){
  let result = await axios.get('');
  this.searchbox = result.data.data;
}
```

Vue生命周期

| 状态 | 阶段周期 |
|--------------|------|
| beforeCreate | 创建前 |
| created | 创建后 |
| beforeMount | 载入前 |

| 状态 | 阶段周期 |
|---------------|-------|
| mounted | 挂载完成 |
| beforeUpdate | 数据更新前 |
| updated | 数据更新后 |
| beforeUnmount | 组件销毁前 |
| unmounted | 组件销毁后 |
