

Web API

Web API介绍

API的概念

API (Application Programming Interface,应用程序编程接口) 是一些预先定义的函数, 目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力, 而又无需访问源码, 或理解内部工作机制的细节。

- 任何开发语言都有自己的API
- API的特征输入和输出(I/O)
 - `var max = Math.max(1, 2, 3);`
- API的使用方法(`console.log('adf')`)

Web API的概念

浏览器提供的一套操作浏览器功能和页面元素的API(BOM和DOM)

此处的Web API特指浏览器提供的API(一组方法), Web API在后面的课程中有其它含义

掌握常见浏览器提供的API的调用方式

[MDN-Web API](#)

JavaScript的组成

ECMAScript - JavaScript的核心

定义了JavaScript 的语法规范

JavaScript的核心, 描述了语言的基本语法和数据类型, ECMAScript是一套标准, 定义了一种语言的标准与具体实现无关

BOM - 浏览器对象模型

一套操作浏览器功能的API

通过BOM可以操作浏览器窗口, 比如: 弹出框、控制浏览器跳转、获取分辨率等

DOM - 文档对象模型

一套操作页面元素的API

DOM可以把HTML看做是文档树, 通过DOM提供的API可以对树上的节点进行操作

DOM

DOM的概念

文档对象模型 (Document Object Model, 简称DOM) , 是W3C组织推荐的处理可扩展标记语言的标准编程接口。它是一种与平台和语言无关的应用程序接口(API),它可以动态地访问程序和脚本, 更新其内容、结构和www文档的风格(目前, HTML和XML文档是通过说明部分定义的)。文档可以进一步被处理, 处理的结果可以加入到当前的页面。DOM是一种基于树的API文档, 它要求在处理过程中整个文档都表示在存储器中。

DOM又称为文档树模型

 image-20210510193634646

 1497154623955

- 文档: 一个网页可以称为文档
- 节点: 网页中的所有内容都是节点 (标签、属性、文本、注释等)
- 元素: 网页中的标签
- 属性: 标签的属性

DOM经常进行的操作

- 获取元素
- 对元素进行操作(设置其属性或调用其方法)
- 动态创建元素
- 事件(什么时机做相应的操作)

获取页面元素

为什么要获取页面元素

例如: 我们想要操作页面上的某部分(显示/隐藏, 动画), 需要先获取到该部分对应的元素, 才进行后续操作

根据id获取元素

```
1 var div = document.getElementById('main');
2 console.log(div);
3
4 // 获取到的数据类型 HTMLDivElement, 对象都是有类型的
```

注意: 由于id名具有唯一性, 部分浏览器支持直接使用id名访问元素, 但不是标准方式, 不推荐使用。

根据标签名获取元素

```
1 var divs = document.getElementsByTagName('div');
2 for (var i = 0; i < divs.length; i++) {
3     var div = divs[i];
4     console.log(div);
5 }
```

根据name获取元素

```
1 var inputs = document.getElementsByName('hobby');
2 for (var i = 0; i < inputs.length; i++) {
3     var input = inputs[i];
4     console.log(input);
5 }
```

根据类名获取元素

```
1 var mains = document.getElementsByClassName('main');
2 for (var i = 0; i < mains.length; i++) {
3     var main = mains[i];
4     console.log(main);
5 }
```

根据选择器获取元素

```
1 var text = document.querySelector('#text');
2 console.log(text);
3
4 var boxes = document.querySelectorAll('.box');
5 for (var i = 0; i < boxes.length; i++) {
6     var box = boxes[i];
7     console.log(box);
8 }
```

- 总结

```
1 掌握
2     getElementById()
3     getElementsByTagName()
4 了解
5     getElementsByName()
6     getElementsByClassName()
7     querySelector()
8     querySelectorAll()
```

事件

事件：触发-响应机制

事件三要素

- 事件源:触发(被)事件的元素
- 事件名称: click 点击事件
- 事件处理程序:事件触发后要执行的代码(函数形式)

事件的基本使用

```
1 var box = document.getElementById('box');
2 box.onclick = function() {
3     console.log('代码会在box被点击后执行');
4 };
```

案例

- 点击按钮弹出提示框
- 点击按钮切换图片

属性操作

非表单元素的属性

href、title、id、src、className

```
1 var link = document.getElementById('link');
2 console.log(link.href);
3 console.log(link.title);
4
5 var pic = document.getElementById('pic');
6 console.log(pic.src)
```

案例：

点击按钮显示隐藏div

美女相册

- innerHTML和innerText

```
1 var box = document.getElementById('box');
2 box.innerHTML = '我是文本<p>我会生成为标签</p>';
3 console.log(box.innerHTML);
4 box.innerText = '我是文本<p>我不会生成为标签</p>';
5 console.log(box.innerText);
```

- HTML转义符

```
1 "      &quot;;
2 '      &apos;;
3 &      &amp;;
4 <      &lt;; // less than 小于
5 >      &gt;; // greater than 大于
6 空格   &nbsp;;
7 ©      &copy;;
```

- innerHTML和innerText的区别
- innerText的兼容性处理

表单元素属性

- value 用于大部分表单元素的内容获取(option除外)
- type 可以获取input标签的类型(输入框或复选框等)
- disabled 禁用属性
- checked 复选框选中属性
- selected 下拉菜单选中属性

案例

- 给文本框赋值, 获取文本框的值
- 点击按钮禁用文本框
- 检测用户名是否是3-6位, 密码是否是6-8位, 如果不满足要求高亮显示文本框
- 设置下拉框中的选中项
- 搜索文本框
- 全选反选

自定义属性操作

- getAttribute() 获取标签行内属性
- setAttribute() 设置标签行内属性
- 使用.直接添加属性, 不会在html标签内形成属性, 但是可以在js中使用
- removeAttribute() 移除标签行内属性
- 与element.属性的区别: 上述三个方法用于获取任意的行内属性。

样式操作

- 使用style方式设置的样式显示在标签行内

```
1 var box = document.getElementById('box');
2 box.style.width = '100px';
3 box.style.height = '100px';
4 box.style.backgroundColor = 'red';
```

- 注意

通过样式属性设置宽高、位置的属性类型是字符串, 需要加上px

类名操作

- 修改标签的className属性相当于直接修改标签的类名

```
1 var box = document.getElementById('box');
2 box.className = 'show';
```

案例

- 开关灯
- 点击按钮改变div的背景颜色
- 图片切换二维码案例
- 当前输入的文本框高亮显示
- 点击按钮改变div的大小和位置
- 列表隔行变色、高亮显示
- tab选项卡切换

创建元素的三种方式

区分节点和元素

节点有许多种：元素节点、属性节点、文本节点等等，用nodeType不同的值来区分元素指的是元素节点，是节点中的一种

```
1  <body>
2    <ul id="ul">
3      <li>first</li>
4      <li>1111</li>
5      <li>1111</li>
6      <li>1111</li>
7      <li>1111</li>
8      <li>last</li>
9    </ul>
10
11
12  <script src="js/common.js"></script>
13  <script>
14    var ul = document.getElementById('ul');
15
16    // 获取第一个子节点
17    console.dir(ul.firstChild);
18    // 获取第一个子元素
19    console.dir(ul.firstChildElement);
20
21
22    // 获取最后一个子节点
23    console.dir(ul.lastChild);
24    // 获取最后一个子元素
25    console.dir(ul.lastElementChild);
26
27    var firstElement = getFirstElementChild(ul);
28    console.log(firstElement);
29  </script>
30 </body>
```

注意获取第一个元素（ul）和第一个节点（文本）的内容是不一样的，带element的是元素

document.write()

```
1 document.write('新设置的内容<p>标签也可以生成</p>');
```

innerHTML

```
1 var box = document.getElementById('box');
2 box.innerHTML = '新内容<p>新标签</p>';
```

document.createElement()

```
1 var div = document.createElement('div');
2 document.body.appendChild(div);
```

性能问题

- innerHTML方法由于会对字符串进行解析，需要避免在循环内多次使用。
- 可以借助字符串或数组的方式进行替换，再设置给innerHTML
- 优化后与document.createElement性能相近

案例

- 动态创建列表，高亮显示
- 根据数据动态创建表格

节点操作

元素节点操作

```
1 var body = document.body;
2 var div = document.createElement('div');
3 body.appendChild(div);
4
5 var firstEle = body.children[0];
6 body.insertBefore(div, firstEle);
7
8 body.removeChild(firstEle);
9
10 var text = document.createElement('p');
11 body.replaceChild(text, div);
```


案例：

选择水果

节点属性

- nodeType 节点的类型
 - 1 元素节点
 - 2 属性节点
 - 3 文本节点
- nodeName 节点的名称(标签名称)
- nodeValue 节点值
 - 元素节点的nodeValue始终是null

模拟文档树结构

1497165666684

```
1 function Node(option) {
2   this.id = option.id || '';
3   this.nodeName = option.nodeName || '';
```

```

4   this.nodeValue = option.nodeValue || '';
5   this.nodeType = 1;
6   this.children = option.children || [];
7 }
8
9 var doc = new Node({
10  nodeName: 'html'
11 });
12 var head = new Node({
13  nodeName: 'head'
14 });
15 var body = new Node({
16  nodeName: 'body'
17 })
18 doc.children.push(head);
19 doc.children.push(body);
20
21 var div = new Node({
22  nodeName: 'div',
23  nodeValue: 'haha',
24 });
25
26 var p = new Node({
27  nodeName: 'p',
28  nodeValue: '段落'
29 })
30 body.children.push(div);
31 body.children.push(p);
32
33 function getChildren(ele) {
34   for(var i = 0; i < ele.children.length; i++) {
35     var child = ele.children[i];
36     console.log(child.nodeName);
37     getChildren(child);
38   }
39 }
40 getChildren(doc);

```

节点层级

```

1 var box = document.getElementById('box');
2 console.log(box.parentNode);
3 console.log(box.childNodes);
4 console.log(box.children);
5 console.log(box.nextSibling);
6 console.log(box.previousSibling);
7 console.log(box.firstChild);
8 console.log(box.lastChild);

```

- 注意

childNodes和children的区别，**childNodes获取的是子节点，children获取的是子元素**

nextSibling和previousSibling获取的是节点，**获取元素对应的属性是nextElementSibling和previousElementSibling获取的是元素**

nextElementSibling和previousElementSibling有兼容性问题，IE9以后才支持

- 总结

```
1  节点操作，方法
2      appendChild()
3      insertBefore()
4      removeChild()
5      replaceChild()
6  节点层次，属性
7      parentNode
8      childNodes
9      children
10     nextSibling/previousSibling
11     firstChild/lastChild
```

事件详解

注册/移除事件的三种方式

add 添加 Event 事件 Listener 监听者（事件处理函数） IE9以后才支持

第一个参数 事件名称 不带on

第二个参数 事件处理函数

第三个参数 事件冒泡/事件捕获

```
1  btn.addEventListener('click', function () {
2      // alert('abc');
3      alert(this); // 触发事件的对象
4  }, false);
5
6  btn.addEventListener('click', function () {
7      // alert('hello');
8      alert(this);
9  }, false);
```

```
1  var box = document.getElementById('box');
2  box.onclick = function () {
3      console.log('点击后执行');
4  };
5  box.onclick = null;
6
7  box.addEventListener('click', eventCode, false);
8  box.removeEventListener('click', eventCode, false);
9
10 box.attachEvent('onclick', eventCode);
11 box.detachEvent('onclick', eventCode);
12
13 function eventCode() {
14     console.log('点击后执行');
15 }
```

兼容代码

```
1 function addEventListener(element, type, fn) {
2   if (element.addEventListener) {
3     element.addEventListener(type, fn, false);
4   } else if (element.attachEvent){
5     element.attachEvent('on' + type, fn);
6   } else {
7     element['on' + type] = fn;
8   }
9 }
10
11 function removeEventListener(element, type, fn) {
12   if (element.removeEventListener) {
13     element.removeEventListener(type, fn, false);
14   } else if (element.detachEvent) {
15     element.detachEvent('on' + type, fn);
16   } else {
17     element['on'+type] = null;
18   }
19 }
```

事件的三个阶段

1. 捕获阶段
2. 当前目标阶段
3. **冒泡阶段：如果父元素和子元素都有相同的事件，触发子元素事件时，会同时触发父元素事件**

事件对象.eventPhase属性可以查看事件触发时所处的阶段

事件委托

把子元素要做的事情，转交给父元素来做（事件冒泡）

事件对象的属性和方法

- event.type 获取事件类型
- clientX/clientY 所有浏览器都支持，窗口位置
- pageX/pageY IE8以前不支持，页面位置
- event.target || event.srcElement 用于获取触发事件的元素
- **event.preventDefault() 取消默认行为**

案例

- 跟着鼠标飞的天使

```
1 var img = document.getElementById('img');
2 document.onmousemove = function (e) {
3   console.log('abc');
4   e = e || window.event;
5
6   img.style.position = 'absolute';
7
8   // clientX / clientY 获取的是鼠标在可视区域中的位置
9   img.style.left = e.clientX + 'px';
```

```

10     img.style.top = e.clientY + 'px';
11
12     // 获取鼠标在页面中的位置
13     // img.style.left = e.pageX + 'px';
14     // img.style.top = e.pageY + 'px';
15 }

```

- 鼠标点哪图片飞到哪里
- 获取鼠标在div内的坐标

阻止事件传播的方式

- **标准方式** `event.stopPropagation();`

```

1     array.forEach(function (item) {
2
3         item.addEventListener('click', function (e) {
4             // eventPhase 事件的阶段
5             // 1 捕获阶段 2 目标阶段 3 冒泡阶段
6             // console.log(e.eventPhase);
7
8             // target 始终是点击的那个元素
9             console.log('target ' + e.target.id);
10            // currentTarget 调用事件处理函数的元素
11            console.log('currentTarget ' + e.currentTarget.id);
12            // this跟currentTarget一样 是调用事件处理函数的元素
13            console.log('this ' + this.id);
14
15            // 阻止事件冒泡
16            e.stopPropagation();
17        }, false);
18
19    });

```

- IE低版本 `event.cancelBubble = true;` 标准中已废弃

常用的鼠标和键盘事件

- `onmouseup` 鼠标按键放开时触发
- `onmousedown` 鼠标按键按下触发
- `onmousemove` 鼠标移动触发
- `onmouseover/onmouseout` 鼠标进入/离开 会触发事件冒泡
- `onmouseenter/onmouseleave` 鼠标进入/离开 不会触发事件冒泡
- `onkeyup` 键盘按键按下触发
- `onkeydown` 键盘按键抬起触发

BOM

BOM的概念

BOM(Browser Object Model) 是指浏览器对象模型，浏览器对象模型提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。BOM由多个对象组成，其中代表浏览器窗口的Window对象是BOM的顶层对象，其他对象都是该对象的子对象。

我们在浏览器中的一些操作都可以使用BOM的方式进行编程处理，

比如：刷新浏览器、后退、前进、在浏览器中输入URL等

BOM的顶级对象window

window是浏览器的顶级对象，当调用window下的属性和方法时，可以省略window

注意：window下一个特殊的属性 window.name

对话框

- alert()
- prompt()
- confirm()

页面加载事件

- onload：页面加载完毕执行（DOM元素加载完毕，当外部文件加载完毕）

```
1 window.onload = function () {
2     // 当页面加载完成执行
3     // 当页面完全加载所有内容（包括图像、脚本文件、CSS 文件等）执行
4     $('qwer').load(function () {
5         var x = document.getElementsByTagName('iframe')[0];
6         var y = (x.contentWindow || x.contentDocument);
7         var z = y.getElementsByTagName('video')[0];
8         console.log(z);
9     });
10 }
```

- onunload：当关闭网页的时候执行

```
1 window.onunload = function () {
2     // 当用户退出页面时执行
3 }
```

定时器

setTimeout()和clearTimeout()

在指定的毫秒数到达之后执行指定的函数，只执行一次

```
1 // 创建一个定时器，1000毫秒后执行，返回定时器的标示
2 var timerId = setTimeout(function () {
3     console.log('Hello world');
4 }, 1000);
5
6 // 取消定时器的执行
7 clearTimeout(timerId);
```

setInterval()和clearInterval()

定时调用的函数，可以按照给定的时间(单位毫秒)周期调用函数

```
1 // 创建一个定时器，每隔1秒调用一次
2 var timerId = setInterval(function () {
3     var date = new Date();
4     console.log(date.toLocaleTimeString());
5 }, 1000);
6
7 // 取消定时器的执行
8 clearInterval(timerId);
```

案例：

```
1 定时器
2 简单动画
```

location对象

location对象是window对象下的一个属性，使用的时候可以省略window对象

location可以获取或者设置浏览器地址栏的URL

location有哪些成员？

- 使用chrome的控制台查看
- 查MDN
[MDN](#)
- 成员
 - assign()/reload()/replace()
 - hash/host/hostname/search/href.....

URL

统一资源定位符 (Uniform Resource Locator, URL)

- URL的组成

```
1 scheme://host:port/path?query#fragment
2 http://www.itheima.com:80/a/b/index.html?name=zs&age=18#bottom
3 scheme:通信协议
4     常用的http,ftp,maito等
5 host:主机
6     服务器(计算机)域名系统 (DNS) 主机名或 IP 地址。
7 port:端口号
8     整数，可选，省略时使用方案的默认端口，如http的默认端口为80。
9 path:路径
10    由零或多个'/'符号隔开的字符串，一般用来表示主机上的一个目录或文件地址。
11 query:查询
12    可选，用于给动态网页传递参数，可有多个参数，用'&'符号隔开，每个参数的名和值用'='符号隔
    开。例如：name=zs
13 fragment:信息片断
14    字符串，锚点。
```

作业

解析URL中的query，并返回对象的形式

```
1 function getQuery(queryStr) {
2   var query = {};
3   if (queryStr.indexOf('?') > -1) {
4     var index = queryStr.indexOf('?');
5     queryStr = queryStr.substr(index + 1);
6     var array = queryStr.split('&');
7     for (var i = 0; i < array.length; i++) {
8       var tmpArr = array[i].split('=');
9       if (tmpArr.length === 2) {
10        query[tmpArr[0]] = tmpArr[1];
11      }
12    }
13  }
14  return query;
15 }
16 console.log(getQuery(location.search));
17 console.log(getQuery(location.href));
```

history对象

- back()
- forward()
- go()

navigator对象

- userAgent

特效

this.style 获取的仅仅是标签的style中设置的样式属性，如果标签没有设置style属性，此时获取到的都是空字符串

偏移量

- offsetParent用于获取定位的父级元素
- offsetParent和parentNode的区别
- offsetLeft是元素距离页面左边的偏移量，offsetTop是元素距离页面上部的偏移量

```
1 var box = document.getElementById('box');
2 console.log(box.offsetParent); //最近的具有定位父元素
3 console.log(box.offsetLeft); //获取当前元素到 具有定位父节点 的left方向的距离
4 console.log(box.offsetTop); //获取当前元素到 具有定位父节点 的top方向的距离
5 console.log(box.offsetWidth); //水平方向 width + 左右padding + 左右border-width
6 console.log(box.offsetHeight); //垂直方向 height + 上下padding + 上下border-width
```

客户区大小

```
1 var box = document.getElementById('box');
2 console.log(box.clientLeft); // 左border的宽度
3 console.log(box.clientTop); // 上border的宽度
4 console.log(box.clientWidth); // 水平方向 width + 左右padding
5 console.log(box.clientHeight); // 垂直方向 height + 上下padding
```

滚动偏移

```
1 var box = document.getElementById('box');
2 console.log(box.scrollLeft);
3 console.log(box.scrollTop);
4 console.log(box.scrollWidth); // 元素内容真实的宽度，内容不超出盒子高度时为盒子的
  clientWidth
5 console.log(box.scrollHeight); // 元素内容真实的高度，内容不超出盒子高度时为盒子的
  clientHeight
```

案例

- 拖拽案例
- 弹出登录窗口
- 放大镜案例
- 模拟滚动条
- 匀速动画函数
- 变速动画函数
- 无缝轮播图
- 回到顶部

附录

元素的类型



1497169919418