08 - BOM 浏览器对象模型 LUO JIN 2020-08-26

学习内容：BOM 概述，window 对象的常见事件，定时器，JS 执行机制，location 对象，navigator 对象，history 对象

**1. BOM 概述**

**1.1 什么是 BOM**

BOM（Browser Object Model）即浏览器对象模型，它提供了独立于内容而与浏览器窗口进行交互的对象，其核心 对象是 window。页面前进后退页面，刷新页面，窗口大小变化，滚动条滚动

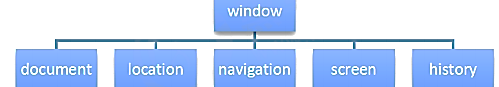
BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

BOM 缺乏标准，JavaScript 语法的标准化组织是 ECMA，DOM 的标准化组织是 W3C，BOM 最初是Netscape 浏览器标准的一部分。

|  |  |
| --- | --- |
| **DOM**  文档对象模型  DOM 就是把「文档」当做一个「对象」来看待  DOM 的顶级对象是 document  DOM 主要学习的是操作页面元素  DOM 是 W3C 标准规范 | **BOM**  浏览器对象模型：整个浏览器  把「浏览器」当做一个「对象」来看待  BOM 的顶级对象是 window  BOM 学习的是浏览器窗口交互的一些对象  BOM 是浏览器厂商在各自浏览器上定义的，兼容性较差 |

**1.2 BOM 的构成**

BOM 比 DOM 更大，它包含 DOM。



**window 对象是浏览器的顶级对象，**它具有双重角色。

1. 它是 JS 访问浏览器窗口的一个接口。通过这个接口可以改变窗口大小等等

2. 它是一个全局对象。定义在全局作用域中的变量、函数都会变成 window 对象的属性和方法。 在调用的时候可以省略 window，对话框都属于 window 对象方法，如 alert()、prompt() 等。

注意：window下的一个特殊属性 **window.name**

**2. window 对象的常见事件**

**2.1 窗口加载事件**



window.onload 是窗口 (页面）加载事件,当文档内容完全加载完成会触发该事件(包括图像、脚本文件、CSS 文件等), 就调用的处理函数。

注意：

1. 有了 window.onload 就可以把 JS 代码写到页面元素的上方，因为 onload 是等页面内容全部加载完毕， 再去执行处理函数。

2. window.onload 传统注册事件方式 只能写一次，如果有多个，会以最后一个 window.onload 为准。

3. 如果使用 addEventListener 则没有限制



**DOMContentLoaded** 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等等。

Ie9以上才支持

如果页面的图片很多的话, 从用户访问到onload触发可能需要较长的时间, 交互效果就不能实现，必然影响用 户的体验，此时用 **DOMContentLoaded** 事件比较合适。

【代码演示】

|  |
| --- |
| <script>          // window.onload = function() {          //     var btn = document.querySelector('button');          //     btn.addEventListener('click', function() {          //         alert('点击我');          //     })          // }          // window.onload = function() {          //     alert(22);          // }          window.addEventListener('load', function() {              var btn = document.querySelector('button');              btn.addEventListener('click', function() {                  alert('点击我');              })          })          window.addEventListener('load', function() {              alert(22);          })          document.addEventListener('DOMContentLoaded', function() {                  alert(33);              })              // load 等页面内容全部加载完毕，包含页面dom元素 图片 flash  css 等等              // DOMContentLoaded 是DOM 加载完毕，不包含图片 falsh css 等就可以执行 加载速度比 load更快一些      </script>  </head>  <body>      <button>点击</button>  </body> |

**2.2 调整窗口大小事件**



**window.onresize** 是调整窗口大小加载事件, 当触发时就调用的处理函数。

**注意：** 1. 只要窗口大小发生像素变化，就会触发这个事件。

2. 我们经常利用这个事件完成**响应式布局**。 **window.innerWidth 当前屏幕的宽度**

应用：淘宝 窗口大小变化，一列会隐藏

【代码演示】

|  |
| --- |
| <script>          window.addEventListener('load', function() {              var div = document.querySelector('div');              window.addEventListener('resize', function() {                  console.log(window.innerWidth);                  console.log('变化了');                  if (window.innerWidth <= 800) {                      div.style.display = 'none';                  } else {                      div.style.display = 'block';                  }              })          })      </script>      <div></div> |

**3. 定时器**

**3.1 两种定时器**

window 对象给我们提供了 2 个非常好用的方法-定时器。

* setTimeout()
* setInterval()

**3.2 setTimeout() 定时器**

window.setTimeout(调用函数, [延迟的毫秒数])

setTimeout() 方法用于设置一个定时器，该定时器在定时器到期后执行调用函数

**注意：**

1. window 可以省略。

2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串‘函数名()'三种形式。第三种不推荐

3. 延迟的毫秒数省略默认是 0，如果写，必须是毫秒。

4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

setTimeout() 这个调用函数我们也称为回调函数 callback

普通函数是按照代码顺序直接调用。

而这个函数，需要等待时间，时间到了才去调用这个函数，因此称为回调函数。

简单理解： 回调，就是回头调用的意思。上一件事干完，**再回头再调用这个函数**。

**以前的 element.onclick = function(){} 或者 element.addEventListener(“click”, fn);** 里面的 函数也是回调 函数

|  |
| --- |
| <script>          // 1. setTimeout          // 语法规范：  window.setTimeout(调用函数, 延时时间);          // 1. 这个window在调用的时候可以省略          // 2. 这个延时时间单位是毫秒 但是可以省略，如果省略默认的是0          // 3. 这个调用函数可以直接写函数 还可以写 函数名 还有一个写法 '函数名()'          // 4. 页面中可能有很多的定时器，我们经常给定时器加标识符 （名字)          // setTimeout(function() {          //     console.log('时间到了');          // }, 2000);          function callback() {              console.log('爆炸了');          }          var timer1 = setTimeout(callback, 3000);          var timer2 = setTimeout(callback, 5000);  // setTimeout(callback, 3000);          // setTimeout('callback()', 3000); // 我们不提倡这个写法      </script> |

【案例】5秒后自动关闭的广告

【案例分析】

1. 核心思路：5秒后，隐藏广告
2. 用定时器setTimeout()

|  |
| --- |
| <img src="images/ad.jpg" alt="" class="ad">      <script>          var ad = document.querySelector('.ad');          setTimeout(function() {              ad.style.display = 'none';          }, 5000);      </script> |

3.3 停止 setTimeout() 定时器

window.clearTimeout(timeoutID)

clearTimeout()方法取消了先前通过调用 setTimeout() 建立的定时器。

**注意：** 1. window 可以省略。 2. 里面的参数就是定时器的标识符 。

|  |
| --- |
| <button>点击停止定时器</button>      <script>          var btn = document.querySelector('button');          var timer = setTimeout(function() {              console.log('爆炸了');          }, 5000);          btn.addEventListener('click', function() {              clearTimeout(timer);          })      </script> |

**3.4 setInterval() 定时器**

window.setInterval(回调函数, [间隔的毫秒数]);

setInterval() 方法重复调用一个函数，每隔这个时间，就去调用一次回调函数。

**注意：**

1. window 可以省略。

2. 这个调用函数可以直接写函数，或者写函数名 或者采取字符串 '函数名()' 三种形式。

3. 间隔的毫秒数省略默认是 0，如果写，必须是毫秒，表示每隔多少毫秒就自动调用这个函数。

4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

5. 第一次执行也是间隔毫秒数之后执行，之后每隔毫秒数就执行一次。

|  |
| --- |
| <script>          // 1. setInterval          // 语法规范：  window.setInterval(调用函数, 延时时间);          setInterval(function() {              console.log('继续输出');          }, 1000);          // 2. setTimeout  延时时间到了，就去调用这个回调函数，只调用一次 就结束了这个定时器          // 3. setInterval  每隔这个延时时间，就去调用这个回调函数，会调用很多次，重复调这个函数      </script> |

【案例】倒计时，京东秒杀

【案例分析】

① 这个倒计时是不断变化的，因此需要定时器来自动变化（**setInterval）**

② 三个黑色盒子里面分别存放**时分秒**

③ 三个黑色盒子利用**innerHTML** 放入计算的**小时分钟秒数**

④ 第一次执行也是间隔毫秒数，因此刚刷新页面会有空白

⑤ 最好采取 封装函数 的方式， 这样可以先调用一次这个函数，防止刚开始刷新页面有空白问题

|  |
| --- |
| <style>          div {margin: 200px;}          span {              display: inline-block;              width: 40px;              height: 40px;              background-color: #333;              font-size: 20px;              color: #fff;              text-align: center;              line-height: 40px;          }      </style>  </head>  <body>      <div>          <span class="hour">1</span>          <span class="minute">2</span>          <span class="second">3</span>      </div>      <script>          // 1. 获取元素          var hour = document.querySelector('.hour'); // 小时的黑色盒子          var minute = document.querySelector('.minute'); // 分钟的黑色盒子          var second = document.querySelector('.second'); // 秒数的黑色盒子          var inputTime = +new Date('2019-5-1 18:00:00'); // 返回的是用户输入时间总的毫秒数          countDown(); // 我们先调用一次这个函数，防止第一次刷新页面有空白          // 2. 开启定时器          setInterval(countDown, 1000);          function countDown() {              var nowTime = +new Date(); // 返回的是当前时间总的毫秒数              var times = (inputTime - nowTime) / 1000; // times是剩余时间总的秒数              var h = parseInt(times / 60 / 60 % 24); //时              h = h < 10 ? '0' + h : h;              hour.innerHTML = h; // 把剩余的小时给 小时黑色盒子              var m = parseInt(times / 60 % 60); // 分              m = m < 10 ? '0' + m : m;              minute.innerHTML = m;              var s = parseInt(times % 60); // 当前的秒              s = s < 10 ? '0' + s : s;              second.innerHTML = s;          }  </script> |

**3.5 停止 setInterval() 定时器**

window.clearInterval(intervalID);

clearInterval()方法取消了先前通过调用 setInterval()建立的定时器。

**注意：** 1. window 可以省略。 2. 里面的参数就是定时器的标识符 。

|  |
| --- |
| <button class="begin">开启定时器</button>      <button class="stop">停止定时器</button>      <script>          var begin = document.querySelector('.begin');          var stop = document.querySelector('.stop');          var timer = null; // 全局变量  null是一个空对象, 其他地方函数就可以访问了          begin.addEventListener('click', function() {              timer = setInterval(function() {                  console.log('ni hao ma');              }, 1000);          })          stop.addEventListener('click', function() {              clearInterval(timer);          })      </script> |

【案例】发送短信，点击按钮之后，该按钮60秒之内不能再次点击，防止重复发送信息



【案例分析】

① 按钮点击之后，会禁用 disabled = true

② 同时按钮里面的内容会变化， 注意 button 里面的内容通过 innerHTML修改

③ 里面秒数是有变化的，因此需要用到定时器

④ 定义一个变量，在定时器里面，不断递减

⑤ 如果变量为0 说明到了时间，我们需要 停止定时器，并且复原按钮初始状态。

|  |
| --- |
| <body>      手机号码： <input type="number"> <button>发送</button>      <script>          // 按钮点击之后，会禁用 disabled 为true          // 同时按钮里面的内容会变化， 注意 button 里面的内容通过 innerHTML修改          // 里面秒数是有变化的，因此需要用到定时器          // 定义一个变量，在定时器里面，不断递减          // 如果变量为0 说明到了时间，我们需要停止定时器，并且复原按钮初始状态          var btn = document.querySelector('button');          var time = 3; // 定义剩下的秒数          btn.addEventListener('click', function() {              btn.disabled = true;              var timer = setInterval(function() {                  if (time == 0) {                      // 清除定时器和复原按钮                      clearInterval(timer);                      btn.disabled = false;                      btn.innerHTML = '发送';                  } else {                      btn.innerHTML = '还剩下' + time + '秒';                      time--;                  }              }, 1000);          })      </script>  </body> |

**3.6 this**

this的指向在函数定义的时候是确定不了的，**只有函数执行的时候才能确定this到底指向谁**，一般情况下**this 的最终指向的是那个调用它的对象**

现阶段，我们先了解一下几个this指向

1. 全局作用域或者普通函数中this指向全局对象window（注意定时器里面的this指向window）

2. 方法调用中谁调用this指向谁

3. 构造函数中this指向构造函数的实例

|  |
| --- |
| <button>点击</button>      <script>          // this 指向问题 一般情况下this的最终指向的是那个调用它的对象          // 1. 全局作用域或普通函数this指向全局对象window（ 注意定时器里面的this指向window）          console.log(this);          function fn() {              console.log(this);          }          window.fn();          window.setTimeout(function() {              console.log(this);          }, 1000);          // 2. 方法调用中谁调用this指向谁          var o = {              sayHi: function() {                  console.log(this); // this指向的是 o 这个对象              }          }          o.sayHi();          var btn = document.querySelector('button');          // btn.onclick = function() {          //     console.log(this); // this指向的是btn这个按钮对象          // }          btn.addEventListener('click', function() {                  console.log(this); // this指向的是btn这个按钮对象              })          // 3. 构造函数中this指向构造函数的实例          function Fun() {              console.log(this); // this 指向的是fun 实例对象          }          var fun = new Fun();      </script> |

【案例】电子时钟，显示当前年月日，自动变化

**4. JS 执行机制**

**4.1 JS 是单线程**

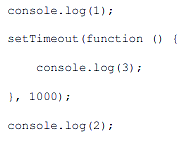
JavaScript 语言的一大特点就是**单线程**，也就是说，同一个时间只能做一件事。这是因为 Javascript 这门脚本语言诞生的使命所致——JavaScript 是为处理页面中用户的交互，以及操作 DOM 而诞生的。比如我们对 某个 DOM 元素进行添加和删除操作，不能同时进行。 应该先进行添加，之后再删除

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。这样所导致的问题是：如果 JS 执行的时间过长，这样就会造成页面的渲染不连贯，导致页面渲染加载阻塞的感觉

**4.2 一个问题**

以下代码执行的结果是什么？

1 2 3

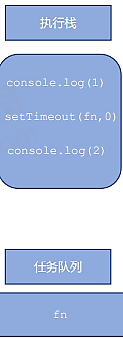


**4.3 同步和异步**

为了解决这个问题，利用 多核CPU 的计算能力，HTML5 提出 Web Worker 标准，允许 JavaScript 脚本创建多个线程。于是，JS 中出现了同步和异步。

**1）同步：**前一个任务结束后再执行后一个任务，程序的执行顺序与任务的**排列顺序是一致的**、同步的。比如做饭的同步做法：我们要烧水煮饭，等水开了（10分钟之后），再去切菜，炒菜。

**2）异步：** 你在做一件事情时，因为这件事情会花费很长时间，在做这件事的同时，你还可以去处理其他事情。比如做饭的异步做法，我们在烧水的同时，利用这10分钟，去切菜，炒菜。

他们的本质区别： 这条流水线上各个流程的执行顺序不同。

**3）同步任务**：同步任务都在**主线程**上执行，形成一个**执行栈**。

**4）异步任务：** JS 的异步是通过**回调函数**实现的。

一般而言，**异步任务**有以下三种类型:

1、普通事件，如 **click、resize** 等

2、资源加载，如 **load、error** 等

3、定时器，包括 **setInterval、setTimeout** 等

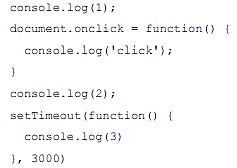
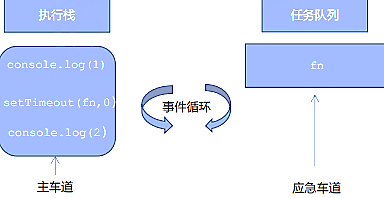
异步任务相关**回调函数**添加到**任务队列**中（任务队列也称为消息队列）。

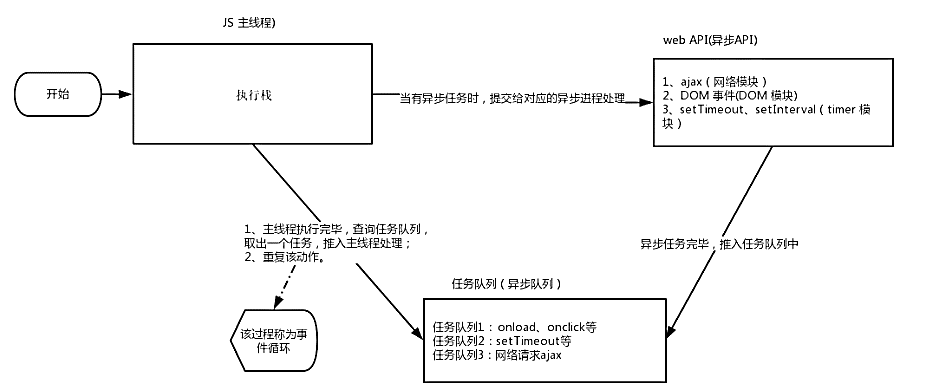
**4.4 JS 执行机制**

1. 先执行执行栈中的同步任务。

2. 异步任务（回调函数）放入任务队列中。

3. 一旦执行栈中的所有同步任务执行完毕，系统就会按次序读取**任务队列**中的异步任务，于是被读取的异步任 务结束等待状态，进入执行栈，开始执行。

123

由于主线程不断的重复获得任务、执行任务、再获取任务、再执行，所以这种机制被称为**事件循环**（ event loop）。

**5. location 对象**

**5.1 什么是 location 对象**

window 对象给我们提供了一个 location 属性用于 获取或设置窗体的 URL，并且可以用于解析 URL 。 因为 这个属性返回的是一个**对象**，所以我们将这个属性也称为 location 对象。

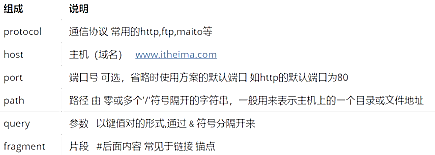
**5.2 URL**

统一资源定位符 (Uniform Resource Locator, URL) 是互联网上标准资源的地址。互联网上的每个文件都有 一个唯一的 URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。

URL 的一般语法格式为：

protocol://host[:port]/path/[?query]#fragment

<http://www.itcast.cn/index.html?name=andy&age=18#link>



**5.3 location 对象的属性**



**重点记住： href 和 search**

【案例】5秒钟之后自动跳转页面

【案例分析】

① 利用 **定时器** 做倒计时效果

② 时间到了，就跳转页面。 使用 location.href

|  |
| --- |
| <button>点击</button>      <div></div>      <script>          var btn = document.querySelector('button');          var div = document.querySelector('div');          btn.addEventListener('click', function() {              // console.log(location.href);              location.href = 'http://www.itcast.cn';          })          var timer = 5;          setInterval(function() {              if (timer == 0) {                  location.href = 'http://www.itcast.cn';              } else {                  div.innerHTML = '您将在' + timer + '秒钟之后跳转到首页';                  timer--;              }          }, 1000);      </script> |

【案例】 获取 URL 参数数据，主要练习数据在不同页面中的传递。

【案例分析】

① 第一个登录页面，里面有提交表单， action 提交到 index.html页面

② 第二个页面，可以使用第一个页面的参数，这样实现了一个数据不同页面之间的传递效果

③ 第二个页面之所以可以使用第一个页面的数据，是利用了URL 里面的 location.search参数

④ 在第二个页面中，需要把这个参数提取。

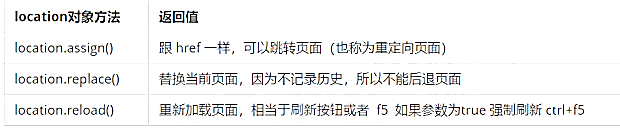
⑤ 第一步去掉？ 利用 **substr**

⑥ 第二步 利用=号分割 **键 和 值** **split(‘=‘)**

⑦ 第一个数组就是键 第二个数组就是值

|  |
| --- |
| <div></div>      <script>          console.log(location.search); // ?uname=andy          // 1.先去掉？  substr('起始的位置'，截取几个字符);          var params = location.search.substr(1); // uname=andy          console.log(params);          // 2. 利用=把字符串分割为数组 split('=');          var arr = params.split('=');          console.log(arr); // ["uname", "ANDY"]          var div = document.querySelector('div');          // 3.把数据写入div中          div.innerHTML = arr[1] + '欢迎您';      </script> |

**5.4 location 对象的方法**

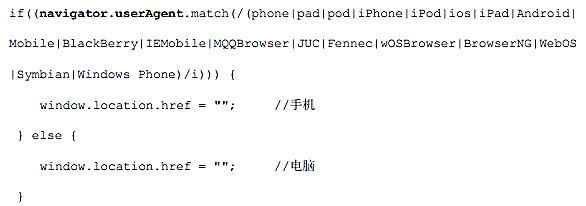


|  |
| --- |
| <button>点击</button>      <script>          var btn = document.querySelector('button');          btn.addEventListener('click', function() {              // 记录浏览历史，所以可以实现后退功能              // location.assign('http://www.itcast.cn');              // 不记录浏览历史，所以不可以实现后退功能              // location.replace('http://www.itcast.cn');              location.reload(true);          })      </script> |

**6. navigator 对象（了解）一般服务端做**

**navigator 对象**包含有关**浏览器的信息**，它有很多属性，我们最常用的是 **userAgent**，该属性可以**返回由客户机发送服务器的 user-agent 头部的值**。

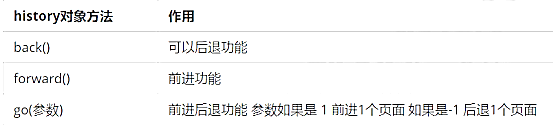
下面前端代码可以**判断用户那个终端(手机，电脑)打开页面**，实现跳转



|  |
| --- |
| <script>          if ((navigator.userAgent.match(/(phone|pad|pod|iPhone|iPod|ios|iPad|Android|Mobile|BlackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|WebOS|Symbian|Windows Phone)/i))) {              window.location.href = "../H5/index.html"; //手机          }      </script> |

**7. history 对象**

window 对象给我们提供了一个 history 对象，与浏览器历史记录进行交互。该对象包含用户（在浏览器窗口中） 访问过的 URL。



|  |
| --- |
| <a href="list.html">点击我去往列表页</a>      <button>前进</button>      <script>          var btn = document.querySelector('button');          btn.addEventListener('click', function() {              // history.forward();              history.go(1);          })      </script> |
| <a href="index.html">点击我去往首页</a>      <button>后退</button>      <script>          var btn = document.querySelector('button');          btn.addEventListener('click', function() {              // history.back();              history.go(-1);          })      </script> |

history 对象一般在实际开发中比较少用，但是会在一些 OA 办公系统中见到。