

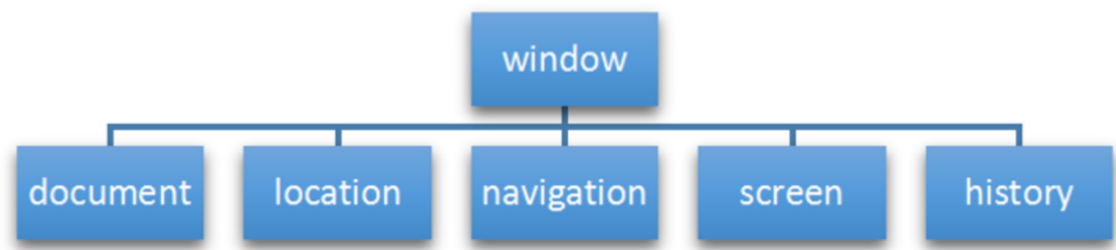
# BOM浏览器对象模型

## BOM概述

- BOM = Browser Object Model 浏览器对象模型
- 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
- 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	BOM
文档对象模型	浏览器对象模型
DOM 就是把 文档 当作一个对象来看待	把 浏览器当作一个对象来看待
DOM 的顶级对象是 document	BOM 的顶级对象是 window
DOM 主要学习的是操作页面元素	BOM 学习的是浏览器窗口交互的一些对象
DOM 是 W3C 标准规范	BOM 是浏览器厂商在各自浏览器上定义的，兼容性较差

## 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
- 它是 JS 访问浏览器窗口的一个接口
- 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，前面学习的对话框都属于 window 对象方法，如 `alert()`、`prompt()` 等。
- **注意：** window下的一个特殊属性 `window.name`

例子：

```

1      <script>
2          var num1 = 10;
3          console.log(num1);          //10
4          console.log(window.num1);    //10
5
6          function fn1(){
7              console.log('函数');
8          }
9          fn1();    //函数
10         window.fn1();    //函数
11
12         console.log(window);    //输出一个window对象的集合，里面包含了
alert()方法和许多我们需要后面学习的方法
13     </script>

```

## window 对象的常见事件

### 窗口加载事件 onload

概念：`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。

在没有接触 `window.onload` 事件之前，我们必须严格遵守代码读写的先后顺序，从上往下执行，如果js内容放置的位置不对，就会出现代码加载不成功或者报错的状态。

例子-----js内容放置的位置不对，代码加载不成功

```

1      <script>
2          var btn1 = document.querySelector('button');
3          btn1.onclick = function(){
4              alert('弹窗');
5          }
6      </script>
7      <button>按钮</button>

```

当使用了 `window.onload` 事件后，我们可以不考虑代码执行的先后顺序来使用 JavaScript 代码

例子-----使用 `window.onload` 事件后，js代码可以放在 `<head>` 中，也可以放在 `<html>` 标签的最初始位置，使用如下

```

1      <script>
2          window.onload = function(){
3              var btn1 = document.querySelector('button');
4              btn1.onclick = function(){
5                  alert('按下按钮出现弹窗');
6              }
7          }
8          //
9          window.onload = function(){

```

```

10         var btn1 = document.querySelector('button');
11         btn1.onclick = function(){
12             alert('因为 window.onload 是传统注册事件，只能写一次，如果有多个，会
按照最后一个 window.onload 为准');
13         }
14     }
15 </script>
16 <button>按钮</button>

```

`window.onload` 窗口加载事件语法如下：重点

```

1 //winow.onload 属于传统注册事件，只能写一次，如果写入多个，会按照最后一个
window.onload 为准
2 window.onload = function(){
3     //执行代码语句
4 }
5
6 //使用window.addEventListener 添加事件,因为addEventListener(),是一个方法监听事件，
可以在一个元素中设置多个事件
7 window.addEventListener('load',function(){
8     //执行代码语句
9 })

```

例子——使用 `window.addEventListener` 在同一个元素中添加多个事件

```

1 <script>
2     // 使用window.addEventListener 添加事件,因为addEventListener(),是一个方法
监听事件，可以在一个元素中设置多个事件
3     window.addEventListener('load',function(){
4         var btn1 = document.querySelector('button');
5         btn1.addEventListener('click',function(){
6             alert('使用window.addEventListener 添加事件,因为
addEventListener(),是一个方法监听事件，可以在一个元素中设置多个事件');
7         })
8     })
9     window.addEventListener('load',function(){
10         var btn1 = document.querySelector('button');
11         btn1.addEventListener('click',function(){
12             alert('使用window.addEventListener 添加事件,看，这就是同一个元素中
的第二个事件');
13         })
14     })
15 </script>
16 <button>按钮</button>

```

概念： `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。

总结：

- 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
- `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
- 如果使用 `addEventListener` 则没有限制，可以在一个元素添加多个事件

## 窗口加载事件 DOMContentLoaded

概念：窗口（页面）加载事件，当DOM加载完毕，不包含图片 flash css 等就可以执行。

语法：

```
1 window.addEventListener('DOMContentLoaded',function(){})
```

例子-----使用 `DOMContentLoaded` 和 `window.onload` 进行比较 3

```
1      <script>
2          window.addEventListener('load',function(){
3              var btn1 = document.querySelector('button');
4              btn1.addEventListener('click',function(){
5                  alert('我用了 window.addEventListener, 中的 load 事件');
6              })
7          })
8          window.addEventListener('load',function(){
9              var btn1 = document.querySelector('button');
10             btn1.addEventListener('click',function(){
11                 alert('我用了 window.addEventListener, 中的 load 事件');
12             })
13         })
14         window.addEventListener('DOMContentLoaded',function(){
15             var btn1 = document.querySelector('button');
16             // DOMContentLoaded 是DOM 加载完毕后，不包图片 flash css等就可以执行的
窗口加载事件，加载速度比load更改一些
17             btn1.addEventListener('click',function(){
18                 alert('DOMContentLoaded 是DOM 加载完毕后，不包图片 flash css等就
可以执行的窗口加载事件，加载速度比load更改一些');
19             })
20         })
21     </script>
22     <button>按钮</button>
```

总结：

- 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, 此时可以将优先显示的 JavaScript 函数提前。
- 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。

## onload 和 DOMContentLoaded 的区别

- `load` 等页面内容全部加载完毕，包括页面dom元素，图片，flash，css等
- `DOMContentLoaded` 是DOM加载完毕，不包含图片 flash css 等就可以执行，加载速度比load更快一些

## 调整窗口大小事件 window.onresize

概念：`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

语法；

```
1 window.onresize = function() {}  
2  
3 // 或者  
4 window.addEventListener('resize',function(){});
```

- 只要窗口大小发生像素变化，就会触发这个事件
- 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

例子-----利用resize事件，模拟CSS中的多媒体查询

```
1 <script>  
2     window.addEventListener('load',function(){  
3         var div1 = document.querySelector('div');  
4         // resize 事件，窗口大小发生像素变化，就会触发这个事件  
5         window.addEventListener('resize',function(){  
6             // innerwidth ， window中的一个属性，获取屏幕对应的宽度  
7             console.log(window.innerWidth);  
8             // 利用resize事件，模拟CSS中的多媒体查询，检测屏幕当前宽度是否小于  
9             900px，如果小于，则 div 由 aqua 变为 blue ,大于则为 aqua  
10            if(window.innerWidth <= 900){  
11                div1.style.backgroundColor = 'blue';  
12            }else{  
13                div1.style.backgroundColor = 'aqua';  
14            }  
15        })  
16    })  
17 </script>  
18 <div style="width: 100px;height: 100px;background-color: aqua;"></div>
```

例子-----利用resize事件，模拟CSS中的多媒体查询

```

1      <div style="width:100px;height: 100px;background-color: aqua;"></div>
2      <script>
3          window.onresize = function(){
4              var div1 = document.querySelector('div');
5              // console.log(window.innerWidth);
6              if(window.innerWidth < 900){
7                  div1.style.backgroundColor = 'blue';
8              }else{
9                  div1.style.backgroundColor = 'aqua';
10             }
11         }
12     </script>

```

## window 对象 定时器

- `setTimeout()`
- `setInterval()`

### setTimeout()定时器（单次触发定时器）

概念: `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。

语法:

```
1 window.setTimeout(调用函数,[延迟的毫秒数]);
```

例子——练习`setTimeout()`

```

1      <script>
2          // 1、语法规范， window.setTimeout(调用的函数，延时时间)
3          window.setTimeout(function(){
4              console.log('3秒过后，我会被输出');
5          },3000)
6          // 2、window 在代码中可以被省略，
7          setTimeout(function(){
8              console.log('3秒过后，我会被输出');
9          },3000)
10         // 3、延时时间也可以省略，如果没有设置，默认是 0 ，
11         setTimeout(function(){
12             console.log('0秒过后，我会被输出');
13         })
14         // 4、调用函数可以直接写函数，也可以写函数名，外部调用
15         setTimeout(fn1,5000);
16         function fn1(){
17             console.log('5秒过后，我会被输出，使用的是函数外部调用');
18         }
19         // 4.1另外一种写法是 '函数名()',知道写法就可以了，不推荐。
20         setTimeout('fn1()',1000);
21         function fn1(){
22             console.log("1秒过后，我会被输出，使用的是'函数名()'调用");
23         }

```

```
1 <script>
2 // 5、常规网页中会有很多定时器，我们需要给定时器加名字，进行区别。
3 function fn2(){
4     console.log('一个函数，出现两次输出');
5 }
6 var time1 = setTimeout(fn2,1000);
7 var time2 = setTimeout(fn2,2000);
8 </script>
```

总结：

- 1、语法规则，window.setTimeout(调用的函数，延时时间)
  - 2、window 在代码中可以被省略，
  - 3、延时时间也可以省略，如果没有设置，默认是 0，
  - 4、调用函数可以直接写函数，也可以写函数名，外部调用
- 4.1另外一种写法是 '函数名()',知道写法就可以了，不推荐。
- 5、常规网页中会有很多定时器，我们需要给定时器加名字，进行区别。

### setTimeout()定时器 —— 回调函数

- `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，时间到了才会去调用这个函数，因此称为回调函数。

提问例子——利用`setTimeOut()`函数，实现弹窗广告自动关闭

```
1 
2 <script>
3     var ad1 = document.querySelector('img');
4     setTimeout(function(){
5         ad1.style.display = 'none';
6     },2000);
7 </script>
```

### clearTimeout() 停止 setTimeout() 定时器

概念： `clearTimeout()` 方法用于取消先前通过调用 `setTimeout()` 建立的定时器

注意：

- `window` 可以省略
- 里面的参数就是定时器的标识符

例子——利用clearTimeout()练习 停止广告弹窗案例

```
1 
2 <button>我不看广告</button>
3 <script>
4     var ad1 = document.querySelector('img');
5     var btn1 = document.querySelector('button');
6
7     var adtime = setTimeout(function(){
8         ad1.style.display = 'block';
9     },3000);
10
11     // 设置停止定时器
12     btn1.addEventListener('click',function(){
13         clearTimeout(adtime);
14     })
15 </script>
```

## setInterval() 定时器（重复定时器）

概念：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

语法：

```
1 window.setInterval(回调函数,[间隔的毫秒数]);
```

注意：

- 这个回调函数：
  - 可以直接写函数
  - 或者写函数名
  - 或者采取字符 '函数名()'
- 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

例子

```
1 <script>
2     setInterval(function(){
3         console.log('每隔1s显示一次');
4     },1000)
5 </script>
```

例子——使用setInterval() 完成倒计时效果

```
1 <div>
2     <span class="day">0</span><b>天</b>
3     <span class="hour">0</span><b>小时</b>
4     <span class="minute">0</span><b>分钟</b>
5     <span class="second">0</span><b>秒</b>
6 </div>
```



```

7
8     <script>
9         // 获取元素
10        var day1 = document.querySelector('.day');
11        var hour1 = document.querySelector('.hour');
12        var minute1 = document.querySelector('.minute');
13        var second1 = document.querySelector('.second');
14
15        function demo(time) {
16            var nowTime = +new Date();           //获取现在距离1970年的总毫秒数
17            var input = +new Date('2022-5-1 18:00:00'); //获取活动截止
日期距离1970年的总毫秒数
18            var times = (input - nowTime) / 1000;    //活动截止日期 - 现在时间 =
距离活动截止的倒计时
19            var d = parseInt(times/60/60/24); //计算天数
20            d = d < 10 ? '0' + d : d;
21            day1.innerHTML = d; //将原先span中的值替换
22            var h = parseInt(times/60/60%24); //计算小时
23            h = h < 10 ? '0' + h : h;
24            hour1.innerHTML = h;
25            var m = parseInt(times/60%60); //计算分数
26            m = m < 10 ? '0' + m : m;
27            minute1.innerHTML = m;
28            var s = parseInt(times%60); //计算当前秒数
29            s = s < 10 ? '0' + s : s;
30            second1.innerHTML = s;
31        }
32        //解决方法就是单独设置函数先调用一下
33        demo();
34        // 开启定时器 setInterval(), 每隔1s触发一次,但是单独调用会有一个延迟效果, 延迟
效果根据你设置的时间而定, 因为setInterval 是回调函数。解决方法就是单独设置函数先调用一下
35        setInterval(demo, 5000);
36    </script>

```

## clearInterval() 停止定时器

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

注意:

- `window` 可以省略
- 里面的参数就是定时器的标识符

例子——创建全局变量, 完成`clearInterval()`事件

```

1    <button class="start">开启重复定时器</button>
2    <button class="stop">关闭重复定时器</button>
3
4    <script>
5        // 获取元素
6        var start1 = document.querySelector('.start');
7        var stop1 = document.querySelector('.stop');
8

```

```

9  /*      // 添加事件,, 使用 clearInterval ('定时器的名字') 方法 ,但是在运行中会报
      错。因为停止计时器clearInterval ('定时器的名字') , 中的「定时器名字」, 和开始的定时器变
      量不在同一个作用域, 所以会报错。
10      strat1.addEventListener('click',function(){
11          var time = setInterval(function(){
12              console.log('开启定时器时, 每1s加载一次');
13          },1000);
14      })
15      stop1.addEventListener('click',function(){
16          clearInterval(time);
17      }) */
18
19      // 解决 因为停止计时器clearInterval ('定时器的名字') , 中的「定时器名字」, 和
      开始的定时器变量不在同一个作用域, 导致报错的解决方法
20      // 定义一个全局变量为空的对象
21      var time = null;
22      strat1.addEventListener('click',function(){
23          time = setInterval(function(){
24              console.log('开启定时器时, 每1s加载一次');
25          },1000)
26      })
27      stop1.addEventListener('click',function(){
28          clearInterval(time);
29      })
30  </script>

```

## this指向 了解

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

目前所学阶段, 我们先需要了解如下几个 this 指向

- 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)
- 方法调用中谁调用 this 指向谁
- 构造函数中 this 指向构造函数实例

例子——研究 this 指向的相关问题

```

1      <button>点击</button>
2      <script>
3          // 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)
4          console.log(this);      // window
5
6          function fn1(){
7              console.log(this); // window
8          }
9          fn1();
10
11          setTimeout(function(){
12              console.log(this);
13          },1000)
14
15          // 2、方法调用中谁调用 this 指向谁

```

```

16     var a = {
17         SayHi: function(){
18             console.log(this);    // this 指向的是 a 这个对象
19         }
20     }
21     a.SayHi();
22
23     var btn1 = document.querySelector('button');
24     //传统点击事件
25     /* btn1.onclick = function(){
26         console.log(this); //this 指向的是 btn 按这个按钮元素
27     } */
28
29     btn1.addEventListener('click',function(){
30         console.log(this); //this 指向的是 btn 按这个按钮元素
31     })
32
33     // 3、构造函数中 this 指向构造函数实例
34     function Fun(){
35         console.log(this); // this 指向的是 fun 实例的对象
36     }
37     var fun1 = new Fun();
38 </script>

```

## location对象

- window 对象给我们提供了一个 `location` 属性用于获取或者设置窗体的url，并且可以解析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。

## url

统一资源定位符（uniform resouce locator）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：

```

1 protocol://host[:port]/path/[?query]#fragment
2
3 http://www.baidu.com/index.html?name=andy&age=18#link

```

组成	说明
protocol	通信协议 常用的http,ftp,mailto等
host	主机(域名) <a href="http://www.baidu.com">www.baidu.com</a>
port	端口号, 可选
path	路径 由零或多个 '/' 符号隔开的字符串
query	参数 以键值对的形式, 通过 & 符号分隔开来
fragment	片段 # 后面内容 常见于链接 锚点

## location对象属性

location对象属性	返回值
location.href	获取或者设置整个URL
location.host	返回主机 (域名) <a href="http://www.baidu.com">www.baidu.com</a>
location.port	返回端口号, 如果未写返回空字符串
location.pathname	返回路径
location.search	返回参数
location.hash	返回片段 #后面内容常见于链接 锚点

重点记住: href 和 search

例子

```

1  <script>
2      console.log(location.href);
3      console.log(location.host);
4      console.log(location.port);
5  </script>

```

### 案例—— location.href 做 404 页面跳转, 在当前页面5秒钟跳转页面

```

1  <!-- 使用 location.href 做 404 页面跳转-->
2  <div>404页面, 还有5秒后跳转页面</div>
3  <script>
4      var div1 = document.querySelector('div');
5      var times = 5;
6      fn1();
7      function fn1(){
8          if(times == 0){
9              location.href = 'http://www.baidu.com';
10         }else{
11             div1.innerHTML = '这个网页还有' + times + '秒之后跳转到首页';

```

```

12         times-- ;
13     }
14 }
15     setInterval(fn1,1000);
16 </script>

```

## location对象方法

location对象方法	返回值
location.assign()	跟href一样，可以跳转页面（也称为重定向页面）
location.replace()	替换当前页面，因为不记录历史，所以不能后退页面
location.reload()	重新加载页面，相当于刷新按钮或者 f5，如果参数为true 强制刷新 ctrl+f5

### 例子

```

1     <button>按钮</button>
2     <script>
3         var btn1 = document.querySelector('button');
4         btn1.addEventListener('click',function(){
5             // 跟href一样，可以跳转页面（也称为重定向页面），记录浏览历史，所以可以实现后
    退功能
6             // location.assign('http://www.baidu.com');
7             // 替换当前页面，因为不记录历史，所以不能后退页面
8             // location.replace('http://www.baidu.com');
9             // 重新加载页面，相当于刷新按钮或者 f5，如果参数为true 强制刷新 ctrl+f5
10            location.reload(true);
11        })
12    </script>

```

## navigator对象

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

### 例子

```

1     <script>
2         console.log(navigator);
3     </script>

```

下面前端代码可以判断用户是用哪个终端打开页面的，如果是用 PC 打开的，我们就跳转到 PC 端的页面，如果是用手机打开的，就跳转到手机端页面

```

1      <script>
2
3      if((navigator.userAgent.match(/(phone|pad|pod|iPhone|iPod|ios|iPad|Android|Mo
4      bile|BlackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|WebOS|Sym
5      bian|Windows Phone)/i))) {
6          window.location.href = "http://www.baidu.com";    //手机
7      } else {
8          window.location.href = "http://www.jingdong.com";    //电脑
9      }
10     </script>

```

## 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。

history对象方法	作用
back()	可以后退功能
forward()	前进功能
go(参数)	前进后退功能，参数如果是 1 前进1个页面 如果是 -1 后退1个页面

a\_index.html

```

1      <a href="/S2017history_list.html">点击前往list详情页</a>
2      <button>前进</button>
3      <script>
4          var btn1 = document.querySelector('button');
5          btn1.addEventListener('click',function(){
6              // history.forward();
7              history.go(1);
8          })
9      </script>

```

b\_list.html

```

1      <a href="/S2016history_index.html">点击前往首页</a>
2      <button>后退</button>
3      <script>
4          var btn1 = document.querySelector('button');
5          btn1.addEventListener('click',function(){
6              // history.back();
7              history.go(-1);
8          })
9      </script>

```

