

BOM(Browser Object Model) 篇

概念： 浏览器对象模型，定义了与浏览器进行交互的编程接口，本质是用 Javascript 开发 Web 应用程序的核心。

window

概念： BOM 的核心，作为访问浏览器窗口的一个接口，也是一个 Global 对象

- 属性

- top

始终指向最外层窗口，即浏览器本身

- parent

始终指向当前窗口的父窗口

如果这两者是最上层窗口，则 `parent=top=window`，最上层的 window 如果不是通过 window

- self, 始终指向 window
 - screenLeft、screenTop 窗口向对于屏幕左侧和顶部的位置
 - innerWidth、innerHeight 窗口的内部宽高(包括滚动条)
 - outerWidth、outerHeight 浏览器窗口大小
 - clientWidth、clientHeight 页面视口宽度和高度
 - window.document === document

- 方法

- moveTo, moveBy(调整窗口位置)

`moveTo(坐标 x,坐标 y)`、`moveBy(x 方向距离,y 方向距离)`

`window.moveTo(0,0)` 移动到左上角

`window.moveBy(0,100)` 向下移动 100px

- resizeTo、resizeBy(调整窗口大小)

`resizeTo(宽度, 高度)`、`resizeBy(宽度缩放值, 高度缩放值)`

`resizeTo(100,100)` 缩放到 100x100

`resizeBy(100,50)` 缩放到 200x150

- onresize 监听调整窗口大小

- open 导航到指定 URL，也可以打开浏览器新窗口；能用于检测是否屏蔽弹窗(返回值 null)

- 语法
open(URL, target, 特性字符串, 新窗口在浏览器历史记录中是否替代当前加载页面的布尔值)

- 使用

```
window.open("http://www.wrox.com/",
"topFrame")
```

如果有一个窗口名叫"topFrame", 则这个窗口就会打开这个 URL; 否则就会打开一个新窗口并将其命名为"topFrame"。第二个参数也可以是一个特殊的窗口名, 比如_self、_parent、_top 或 blank。

- 特性字符串, 打开的新窗口的特性(窗口属性)
- 新创建窗口的 window 有一个 opener 属性, 指向打开它的窗口。指向调用 window.open() 打开它的窗口或窗格的指针。
- 任意窗口.close, 关闭自己; 如果引用还在可以用 closed 属性检查
- alert, 警告框
- confirm, 确认框; 有 bool 返回值
- prompt, 提示框; 有填写就有字符串返回值

location

概念: 提供窗口中加载文档的消息, 以及通常的导航功能, 也保存着把 URL 解析后的各个属性信息; window.location == document.location;

<http://wong:123@www.wrox.com:80/file/?q=javascript#contents>

- 属性(均为字符串)
 - hash, # 后面值, 修改 hash 值会添加历史记录, #contents
 - host, 服务器名及端口号, www.wrox.com:80
 - hostname, 服务器名, www.wrox.com
 - href, 完整 URL, <http://www.wrox.com:80/file/?q=javascript&name=123#contents>
 - pathname, 文件路径和文件名, file
 - port, 端口号, 80
 - protocol, 协议, http:
 - search, 查询字符串, ?q=javascript
 - origin, URL 的源地址。只读, <http://www.wrox.com>
 - username, 域名前指定的用户名, wong
 - password, 域名前指定的密码, 123

注意: hash、search、hostname、pathname 和 port 属性被设置为新值之后都会修改当前 URL; 并且页面重新加载 URL。

- 方法

- **assign** 修改浏览器的地址，会在浏览器中添加历史记录；与直接修改 **location** 和 **href** 相同
- **replace** 修改浏览器的地址，不会在浏览器中添加历史记录；
- **reload** 重新加载当前显示的页面。不传参，页面会以最有效的方式重新加载(可能从缓存中加载)；传 **true** 从服务器加载

navigator

概念： 包含有关浏览器的信息，常用于确定浏览器的类型。

screen

概念： 保存的浏览器窗口外面的客户端显示器的信息

history

概念： 当前窗口首次使用以来用户的导航历史记录虑，不会暴露用户访问过的 URL，可以在不知道 URL 的情况下前进和后退

- 属性
 - **length** 历史记录条目数
 - **state** 当前状态对象
- 方法
 - **go** 前进后退；简写，**back**(go(-1))、**forward**(go(1))
 - **pushState**

推送状态信息到历史记录中，浏览器地址栏也会改变以反映新的相对 URL。

- 语法

```
history.pushState(stateObject, '新状态标题', '可选的相对 URL')
```

- **replaceState** 传入与 **pushState** 同样的前两个参数来更新状态。不会创建新历史记录，只会覆盖当前状态
- **window.onpopstate** “后退”会触发 **window** 对象上的 **popstate** 事件；它的事件对象有 **state** 属性
- **window.onhashchange** 在页面 URL 变化时被触发

DOM(Document Object Model) 篇

概念：DOM 指的是文档对象模型，把整个文档当做一个对象，这个对象提供了处理网页内容的接口。

DOM 级别

- DOM0 级

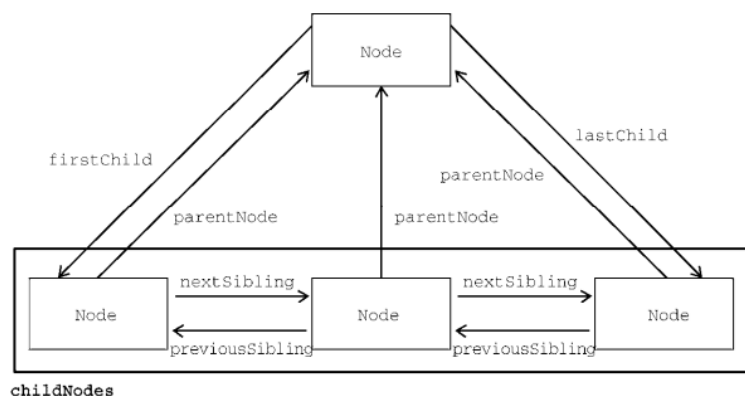
DOM 没有被 W3C 定为标准之前

- DOM1 级
成为 W3C 标准后的第一个级别
- DOM2 级
对 DOM1 的扩展。为节点添加更多方法和属性
- DOM3 级
进一步扩展了 DOM，添加了 XPath 模块
- DOM4 级
增加了 Mutation Observers(监视对 DOM 树所做更改)

DOM 事件级别

1. DOM1 级事件, `el = function() {}`, 在冒泡阶段发生。
2. DOM2 级事件, `el.addEventListener(eventName, callback, options|useCapture)`, 可以为同一事件添加多个事件处理程序，并且会以添加事件处理程序的顺序来触发。

DOM2 事件处理程序在 IE 浏览器中有兼容性问题，IE 中需要使用 `attachEvent`、`detachEvent`，事件处理程序会以添加它们的顺序反向触发。
3. DOM3 级事件, 在 DOM 2 级事件的基础上添加了更多的事件类型。
 - UI 事件，当用户与页面上的元素交互时触发，如：
`load`、`scroll`
 - 焦点事件，当元素获得或失去焦点时触发，如：`blur`、`focus`
 - 鼠标事件，当用户通过鼠标在页面执行操作时触发如：
`dblclick`、`mouseup`
 - 滚轮事件，当使用鼠标滚轮或类似设备时触发，如：
`mousewheel`
 - 文本事件，当在文档中输入文本时触发，如：`input`
 - 键盘事件，当用户通过键盘在页面上执行操作时触发，如：`keydown`、`keypress`
 - 合成事件，当为 IME（输入法编辑器）输入字符时触发，如：`compositionstart`
 - 变动事件，当底层 DOM 结构发生变化时触发，如：
`DOMSubtreeModified`
 - 同时 DOM3 级事件也允许使用者自定义一些事件。
4. DOM1 级中没有事件的相关内容，所以没有 DOM1 级事件。



EventTarget

- 概念：DOM 接口，接收事件、创建侦听器；常见如 Element、document、window，其他如 XMLHttpRequest
- addEventListener
 - 语法
 - `target.addEventListener(type, listener, options)`
 - `target.addEventListener(type, listener, useCapture)`
 - 参数
 - type，表示监听事件类型的字符串。
 - listener，当所监听的事件类型触发时，会接收到一个事件通知（实现了 Event 接口的对象）对象。listener 必须是一个实现了 EventListener 接口的对象，或者是一个函数。
 - options
 - capture, true, listener 会在该类型的事件捕获阶段传播到该元素时触发。
 - once, listener 在添加之后最多只调用一次。true, listener 会在其被调用之后自动移除。
 - passive, true, listener 永远不会调用 preventDefault()。如果 listener 仍然调用了这个函数，客户端会忽略它并抛出一个控制台警告。
 - useCapture
 - false 以冒泡形式触发事件，true 以捕获形式触发事件
- removeEventListener
 - 语法

- `target.removeEventListener(type, listener[, options])`
- `target.removeEventListener(type, listener[, useCapture])`

- 参数，类似于 `addEventListener`

`options` 只有 `capture` 属性，表示这个类型的事件将会被派遣到已经注册的侦听器，然后再派遣到 DOM 树中它下面的任何 `EventTarget`；`useCapture` 指定需要移除的 `listener` 是否为捕获。注意：如果同一个监听事件分别注册了捕获和冒泡，则需要分别移除。

- `dispatchEvent`

- 语法 `cancelled = !target.dispatchEvent(event)`
- 参数
 - `event`，事件对象
 - `target, EventTarget`
- 返回值，当该事件是可取消的(`cancelable` 为 `true`)，如果至少一个该事件的事件处理方法调用了 `Event.preventDefault()`，则返回值为 `false`；否则为 `true`。
- 注意点：与浏览器原生事件不同，`dispatchEvent` 是同步调用事件，原生浏览器通过 **eventloop** 异步调用。

Node

继承于 `EventTarget`

- 属性
 - `nodeType`, 用来区分不同类型的节点，比如 1 元素, 3 文本和 8 注释。
 - `nodeName` , 获取 HTML 元素名称，和 `tagName` 不同，`tagName` 只能获取节点类型为 1 的元素结点类型，属性、注释、文本等其他类型的节点只能用 `nodeName` 获得。
注释的 `nodeName` 以 # 开头，例如 `#text`
 - `childNodes`, 子节点集合，和 `children` 不同，`children` 只返回元素结点也就是 `nodeType` 为 1 的节点
 - `firstChild`、`lastChild`, 第一个子节点、最后一个子节点
 - `previousSibling`、`nextSibling`, 返回紧跟在它前面的兄弟节点、返回紧跟在它后面的兄弟节点
 - `parentNode`, 返回其父节点

- `textContent`,将所有子节点的 `textContent` 合并后返回,除了注释,在节点上设置 `textContent` 属性的话,会删除它的所有子节点,并替换为一个具有给定值的文本节点。可以防止 xss 攻击。
- 方法
 - `appendChild`,元素的所有后面添加节点
 - `insertBefore(newNode,refNode)`,在 `refNode` 前添加节点
 - `removeChild`,删除该元素中的指定子节点
 - `replaceChild(newChild, oldChild)`,替换元素中指定子节点
 - `cloneNode(deep)`,克隆节点,`deep` 是否采用深度克隆,如果为 `true`,则该节点的所有后代节点也都会被克隆,如果为 `false`,则只克隆该节点本身。
 - `contains`,在其 dom 树上查找是否存在该节点

Element

继承于 `Node`、`EventTarget`

- 属性
 - `children`,返回只读的子元素集合,为即时更新的集合
 - `attributes`,返回该元素所有属性节点的一个实时集合
 - `className`,获取或设置 `class` 属性
 - `id`,元素的 `id` 属性
 - `tagName`,标签名
 - `clientWidth`、`clientHeight`,只读属性,不包括滚动条、`border` 和 `margin` 的宽度和高度
 - `clientLeft`、`clientTop`,只读属性,
 - `scrollHeight`、`scrollWidth`,元素的内容高度、宽度包括由于溢出导致的视图中不可见内容。
 - `scrollLeft`、`scrollTop`,元素的卷曲宽度、高度,也就是左边和顶部因为滚动条导致的不可见的部分。
 - `innerText`,会触发回流。
 - `innerHTML`。
- 方法
 - `after`,在该元素后面添加节点。
 - `prepend`,在该元素父节点的第一个子节点之前插入节点。
 - `append`,在该元素的最后一个子节点之后插入一组节点。
 - `remove`,移除自己。
 - `getAttribute`,返回指定属性值。
 - `getAttributeName`,返回该元素所有的属性。
 - `removeAttribute`,移除属性。
 - `setAttribute`,设置属性及属性值。
 - `hasAttribute`,判断该元素是否有该属性。
 - `querySelector`、`querySelectorAll`。

- `scrollTo`,滚动到指定坐标。

```
scrollTo(0, 1000); // 滚动到指定坐标
scrollTo({
  top: 100,
  left: 100,
  behavior: "smooth"
});
```

- `scrollBy`,滚动指定距离。

```
scrollBy(300, 300); // 滚动指定距离
scrollBy({
  top: 100,
  left: 100,
  behavior: "smooth"
});
```

Event

- 概念:

`Event` 接口表示在 `DOM` 中发生的任何事件;一些是用户生成的,如鼠标、键盘事件,而其他由 `API` 生成,如指示动画已经完成运行的事件,视频被暂停;事件通常由外部源触发,同样也会以编程方式触发,例如执行一个 `Element` 的一个 `click` 方法,或者通过定义事件,然后使用 `EventTarget.dispatchEvent` 将其分派指定目标。

- 构造函数 `new Event(typeArg,eventInit)`

- `typeArg`, 事件名
- `eventInit`, 对象包括如下属性: `bubbles`、`cancelable`、`composed`

- 属性

- `target`, 触发事件的对象引用
- `currentTarget`, 绑定事件的对象引用
- `bubbles`, 是否冒泡
- `cancelable`, 是否可取消
- `type`, 事件类型
- `defaultPrevented`, 当前事件是否调用了 `preventDefault`
- `composed`, 指示该事件是否能从 `Shadow DOM` 传递到一般 `DOM`;

- 方法

- `Event.preventDefault`
- `Event.stopPropagation`, 阻止事件冒泡
- `Event.stopImmediatePropagation`, 阻止同一元素的其他相同事件被调用

- `Event.composedPath`，查看事件传播路径

总结：Event 对象用来管理事件触发的各种状态信息、以及事件特定行为

CustomEvent

- 构造函数

```
new CustomEvent(typeArg, customEventInit)
```

- `typeArg` 事件名
- `customEventInit`，对象包括如下属性：`detail`、`bubbles`、`cancelable`
可以在 `detail` 中添加任意属性

常见 events

- `DOMContentLoaded`

初始的 HTML 文档被完全加载和解析完成之后，`DOMContentLoaded` 事件被触发，而无需等待样式表、图像和 `iframe` 的完全加载。

属性：`target`(事件对象 DOM 树中最顶级的那个对象)、`type`(事件类型)、`bubble`(是否冒泡)、`cancelable`(是否可取消默认行为)，均为只读属性。

```
<script>
document.addEventListener("DOMContentLoaded",
function(event) {
    console.log("DOM fully loaded and parsed");
});

for(var i=0; i<1000000000; i++){
    // 这个同步脚本将延迟DOM的解析。
    // 所以DOMContentLoaded事件稍后将启动。
}
</script>
```

- `document.readyStatechange`

文档的 `readyState` 属性发生改变时，会触发 `readystatechange` 事件。

- `readyState`，`document.readyState` 属性描述了 `document` 的加载状态。
 - `loading`，dom 仍在加载
 - `interactive`，文档已被解析，`loading` 状态结束，但是如图像，样式表和框架之类的子资源仍在加载。
 - `complete`，文档和所有子资源已完成加载。`load` 事件即将被触发。

- **window.load**
整个页面及所有依赖资源如样式表和图片都已完成加载时，将触发 load 事件。
- **window.beforeunload**
浏览器窗口关闭或者刷新时，会触发 beforeunload 事件, 触发一个确认对话框，要显示确认对话框，事件处理程序需要在事件上调用 preventDefault()。当前页面不会直接关闭，可以点击确定按钮关闭或刷新，也可以取消关闭或刷新。
- **unload**
当文档或一个子资源正在被卸载时, 触发 unload 事件。在 beforeunload、pagehide 事件后被触发
- **window.pagehide**
当用户单击浏览器的“后退”按钮时，当前页面在显示上一页之前会收到一个 pagehide(页面隐藏)事件。

```

window.addEventListener(
  "pagehide",
  event => {
    if (event.persisted) {
      /* the page isn't being discarded, so it
      can be reused later */
    }
  },
  false
);

```

- focus、focusin,前者不冒泡，后者冒泡
- blur、focusout,前者不冒泡，后者冒泡
- click、dbclick
- keydown、keyup
- mouseenter、mouseleave,不会冒泡,指针离开元素及其所有后代时，会触发 mouseleave，而当指针离开元素或离开元素的后代（即使指针仍在元素内）时，会触发 mouseout。
- mouseover、mouseout,会冒泡
- mousemove
- fullscreenchange、fullscreenerror
- scroll
- select
- copy、paste、cut
- touchstart、touchend,移动端常用

扩充面试题：

DOMContentLoaded 与 load 区别(重点)

- DOMContentLoaded 仅在 HTML 解析完成后触发。因为 CSS 会阻塞 DOM 渲染和 js 的执行,而 js 会阻塞 DOM 的解析。所以, 如果脚本没有阻塞文档的解析(放在 body 尾部/开启了 defer),则浏览器解析完文档便会触发 DOMContentLoaded 事件; 如果文档的解析被 js 阻塞了,需要注意,因为 css 会阻塞 js 的执行,所运行 js 阻塞文档解析时 CSSOM 已经是构建完成的了。
- 任何情况下, DOMContentLoaded 的触发不需要等待图片和其他资源加载完成(谨记 html 解析完触发)。
- 当 onload 事件触发时,页面上所有的 DOM、样式表、脚本和图片资源等已经加载完毕。

DOM 事件模型和事件流

DOM 事件模型分为捕获和冒泡, 会在子元素和父元素之间传播;

传播分为捕获、目标、冒泡三个阶段。

- 捕获, 事件从 window 对象自上而下向目标节点传播的阶段
- 目标, 目标节点正在处理事件的阶段
- 冒泡, 事件从目标节点自下而上向 window 对象传播的阶段

事件流描述了页面接收事件的顺序。分为冒泡流和捕获流, 而早期的 IE 和 Netscape 开发团队提出了几乎完全相反的事件流方案, IE 支持了事件冒泡, 而 Netscape 支持事件捕获。

DOM 事件捕获流程

window -> document -> html -> body -> ... -> 目标元素

事件委托

概念: 由于事件会在冒泡阶段向上传播到父节点, 因此可以把子节点的监听函数定义在父节点上, 由父节点的监听函数统一处理多个子元素的事件。

优点:

- 减少内存消耗和 dom 操作就。添加到页面上的事件处理程序数量, 将直接关系到页面的整体运行性能, 因为需要不断的操作 dom, 那么引起浏览器重绘和回流的可能也就越多, 页面交互的时间也就变得越长, 这就是需要减少 DOM 操作 的原因。通过事件委托只对它的父级进行操作, 与 dom 的操作只需要交互一次, 大大减少了与 DOM 的交互次数, 提高了性能。
- 动态绑定事件, 因为事件绑定在父级元素, 所以新增元素同样可以触发相同的事件, 而无需给新增元素添加事件监听器。

例子:

```

<ul id="list">
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
  .....
  <li>item n</li>
</ul>
<script type="text/javascript">
  const ul = document.querySelector('#list');
  ul.addEventListener('click', function(e) {
    let event = e || window.event
    let target = event.target
    if(target.nodeName.toLocaleLowerCase() !== 'li')
  return
    console.log(target.innerHTML)
  });
</script>

```

封装事件监听与解绑的兼容写法

```

const addEvent = (ele, event, func) =>{
  //用能力检测进行跨浏览器兼容处理
  if (ele.addEventListener) {
    // false表示冒泡事件模型
    ele.addEventListener(event, func, false);
  } else if (ele.attachEvent) {
    ele.attachEvent("on" + event, func);
  } else {
    ele["on" + event] = func;
  }
}

const delEvent = (ele, event, func) => {
  if (ele.removeEventListener) {
    ele.removeEventListener(event, func, false);
  } else if (ele.detachEvent) {
    ele.detachEvent("on" + event, func); //IE
  } else {
    ele["on" + event] = null;
  }
}

};

```

textContent、innerText、innerHTML 区别

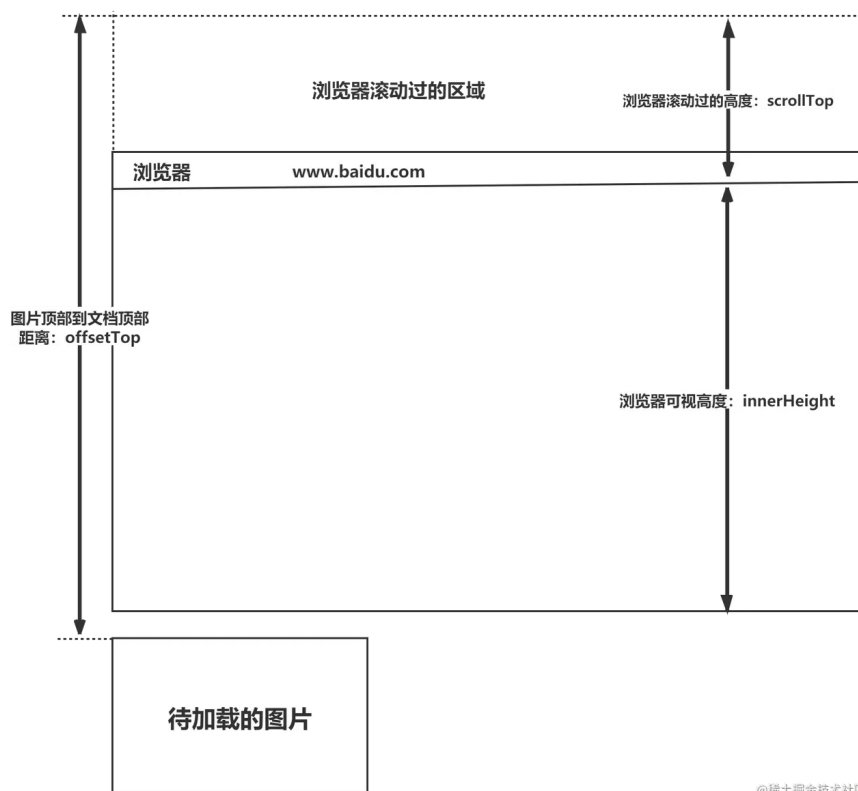
textContent、innerText、innerHTML 区别

url 参数提取及 decodeURIComponent 和 decodeURI 区别

[decodeURIComponent](#) 和 [decodeURI](#) 区别

判断元素是否到达可视区域

- `window.innerHeight` 是浏览器可视区的高度
- `document.documentElement.scrollTop` 是浏览器滚动的过的距离
- `imgs.offsetTop` 是元素顶部距离文档顶部的高度（包括滚动条的距离）
- 内容达到显示区域的： $\text{img.offsetTop} - \text{document.documentElement.scrollTop} < \text{window.innerHeight}$ (元素到文档顶部 高度 - 页面向上卷曲距离 < 视口高度)



mouseover/mouseout 和 mouseenter/mouseleave 区别

`mouseover` 支持事件冒泡，也就是当其本身或者它的子元素触发 `mouseover` 事件时会发生 `mouseover` 事件的冒泡，冒泡到父元素触发父元素的 `mouseover` 事件。

`mouseenter` 不支持事件冒泡，鼠标移入其子元素时并不会将 `mouseenter` 事件冒泡到父元素。



当鼠标移入一个元素时的事件触发顺序：

`mouseover` -> `mouseenter` -> `mousemove` (hover进去之后移动会触发) ->
`mouseout` -> `mouseleave`

touchstart touchend click mousedown mouseup mousemove mouseover 执行顺序

- 手指点触屏幕时触发顺序：

`touchstart` -> `touchend` -> `mousemove` -> `mousedown` -> `mouseup` ->
`click`

- 手指触摸屏幕移动时触发顺序：

`touchstart` -> `mouseover` -> `mouseenter` -> `mousemove` -> `mousedown` -
> `mouseup` -> `click` -> `touchend`

touchstart: 当手指触摸屏幕时发生

touchmove: 当手指在屏幕上滑动时连续触发

touchend: 手指从屏幕上移开时触发

touchcancel: 系统停止跟踪触摸时触发