BOM

- BOM（Browser Object Model）浏览器对象模型
- BOM 使 JavaScript 有能力与浏览器"对话"
- 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

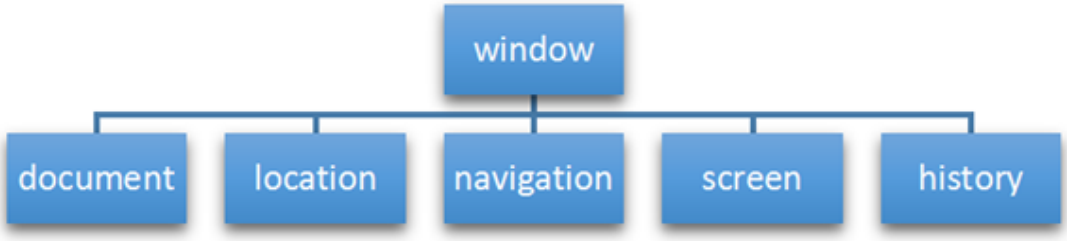
DOM

- 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
- 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

BOM的构成



window对象

- window对象表示浏览器窗口，是BOM模型中的顶层对象,，因此所有BOM模型中的对象都是该对象的子对象
- 所有 JavaScript 全局对象、函数以及变量均自动成为 window 对象的成员
- 全局变量是 window 对象的属性
- 全局函数是 window 对象的方法
- window对象调用属性、方法时可以省去window直接调用

window方法

方法名	说明
alert("信息内容")	弹出一个警告框
confirm("信息内容")	弹出一个确认对话框，返回true/false
prompt("信息内容",["默认输入内容"])	弹出一个提示对话框，返回输入内容

定时器

在项目中难免会碰到需要实时刷新，动画依次出现等等需求，这时候就需要定时器登场了

js 定时器有以下两种:

- setInterval
  周期性执行计时器（执行多次）
- clearInterval(timerId);
  根据定时器id停止定时器
- setTimeout
  定时执行计时器(只执行一次)
- clearTimeout(timerId);
  根据定时器id停止定时器

练习

❖发送短信倒计时

✓点击按钮后，该按钮30秒之内不能再次点击，防止重复发送短信。

window事件

window.onload事件

- onload
  - 窗口加载事件，当文档内容全部加载完成后(包括图片、CSS文件、Flash文件等)才会触发该事件
  - onload事件采用传统方式注册只能执行一次，如果有多个则只有最后一个生效
    - 通过addEventListener监听方式来注册则没有个数的限制（IE9以上支持）

```
window.onload = function() {
    alert('hello')
}
```

```
window.addEventListener('load',function(){
    alert('hello')
})
```

扩展：document.DOMContentLoaded事件

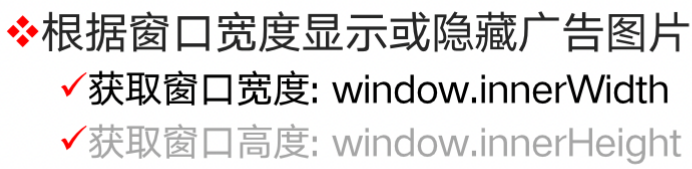
- DOMContentLoaded事件属于document对象
- DOMContentLoaded事件只有IE9以上版本支持
- DOMContentLoaded事件仅当文档内容加载完成之后就会触发，不包括图片、CSS文件、Flash文件等
- DOMContentLoaded加载速度更快，比onload有更好的用户体验

```
document.addEventListener('DOMContentLoaded',function() {
       alert('hello')
})
```

window.onresize事件

- 当窗口大小发生改变时会触发该事件
- 可以利用这个事件进行一些响应式布局

练习



location对象

location 对象用于获得当前页面的地址 (URL)，或把浏览器重定向到新的页面

location的属性

属性名	说明
pathname	返回当前页面的路径和文件名。
href	返回当前页面的 URL
hostname	返回域名
port	返回端口
protocol	返回协议
search	返回传值部分

location的方法

属性名	说明
reload([true|false])	从服务重新加载页面，true为绕过缓存,默认为false
replace("url")	跳转到新页面

navigator对象

navigator对象，也称为浏览器对象，该对象包含了浏览器的整体信息，如浏览器名称、版本号等

navigator属性

属性名	说明
appName	返回浏览器的名称
appVersion	返回浏览器的版本号
userAgent	返回浏览器用于HTTP请求的用户代理头的值
appCodeName	返回浏览器的代码名
platform	返回运行浏览器的操作系统或硬件平台
cookieEnabled	检测浏览器是否支持Cookie。该属性值为布尔类型，如果浏览器支持Cookie则返回true，否则返回false

练习

判断用户在哪个终端打开页面，实现跳转

```
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 = "";      //手机
 } else {
    window.location.href = "";      //电脑
 }
```

history对象

- history对象是JavaScript中的一种默认对象，该对象可以用来存储客户端浏览器窗口最近浏览过的历史网址。
- 通过history对象的方法，可以完成类似于浏览器窗口中的前进、后退等按钮的功能
- history对象一般在实际开发中比较少用，但是可能会在一些管理系统中使用

history属性

属性名	说明
length	浏览器窗口的历史列表中的网页个数

history方法

方法名	说明
go(num|url)	该方法可以直接跳转到某一个已经访问过的URL。该方法中可以包含两种参数，一种参数是要访问的URL在历史列表中的相对位置；另一种参数为要访问的URL的子串
forward()	该方法可以前进到下一个访问过的URL,等价于go(1)
back()	该方法可以返回到上一个访问过的URL,等价于go(-1)

Cookie&Storage

一、Cookie

Cookie保存在客户端，主要供服务器端存储数据。

Cookie是由服务器发送给客户端的信息，存储在客户端浏览器的内存或者硬盘上。常用于保存用户名，密码，个性化设置，个人偏好记录等。当用户访问服务器时，服务器可以设置和访问cookie的信息。

Cookie是一种会话跟踪技术，通过Cookie可以跟踪用户在不同页面的会话状态。

Cookie不能跨浏览器使用。

> 注意：即使是用document的cookie去设置也需要在服务器的环境下执行代码，否则无法操作cookie。

1、创建Cookie

- 语法：

```
document.cookie = "名=值" ;
```

```
// 例如：
document.cookie = "user=andy";
document.cookie = "password=123456"; // 这里“=”不代表重新赋值，而是表示继续添加
```

- 过期时间
  - cookie默认过期时间为浏览器关闭前永不过期，浏览器关闭后cookie失效。
  - 可以设置具体的过期时间(UTC 或 GMT 时间)，在浏览器关闭后仍保留cookie信息。

```
// 过期时间只能是UTC 或 GMT 时间
document.cookie = "名=值;expires=过期时间" ;
```

- 函数封装：创建可以设置过期时间的cookie

```
/**
* 设置cookie
* @param {string} 名称
* @param {mixed}  值
* @param {number} 过期时间(单位：秒)
*/
function setCookie(name, value, seconds) {
    if (seconds) {
        var date = new Date();
```

```
        date.setSeconds(date.getSeconds() + seconds);
        document.cookie = name + "=" + value + ";expires=" +
date.toUTCString();
    } else {
        document.cookie = name + "=" + value;
    }
}
```

2、获取cookie

- 语法:

```
// 获取全部cookie信息
document.cookie
```

- 函数封装：根据名称获取cookie的值

```
/**
* 根据名称获取cookie的值
* @param {string} name 名称
* @return {string} 值
*/
function getCookie(name) {
    var cookies = document.cookie.split(";");

    for (var i = 0; i < cookies.length; i++) {
        var cookie = cookies[i].split("=");
        if (cookie[0].trim() == name) {
            return cookie[1];
        }
    }
}
```

3、删除cookie

只需要将cookie的时间设置为已过期即可

- 语法:

```
document.cookie="名=值;expires=已过期的时间";
```

- 函数封装:删除指定名称的cookie

```
/**
* 删除指定名称cookie
* @param {string} 名称
*/
function removeCookie(name) {
    var date = new Date();
    date.setSeconds(date.getSeconds() - 1);
    document.cookie = name + "=" + null + ";expires=" + date.toUTCString();
}
```

- 函数封装:删除全部cookie

```
/**
* 删除全部cookie
*/
function clearCookie() {
    var cookies = document.cookie.split(";");

    for (var i = 0; i < cookies.length; i++) {
        var cookie = cookies[i].split("=");
        var date = new Date();
        date.setSeconds(date.getSeconds() - 1);
        document.cookie = cookie[0] + "=" + null + ";expires=" +
date.toUTCString();
    }

}
```

4、封装cookie函数

- setCookie() — 创建Cookie
- getCookie() — 获取指定Cookie
- removeCookie() — 删除指定Cookie
- clearCookie() — 清除所有Cookie

5、案例1：记住用户名、密码

```
<form action="">
    <input type="text" name="username" placeholder="请输入用户名">
    <input type="password" name="password" placeholder="请输入密码">
    <label for="remember">记住密码</label>
    <input type="checkbox" value="yes" name="remember" id="remember">
    <button type="button">登录</button>
</form>
<script src="../static/cookie.js"></script>
<script>
    var username = document.querySelector("input[name='username']");
    var password = document.querySelector("input[name='password']");
    var remember = document.querySelector("input[name='remember']");

    // 事件触发
    remember.onchange = function() {
        if (this.checked) {
            //记住密码
            setCookie("username", username.value, 7 * 86400);
            setCookie("password", password.value, 7 * 86400);
            setCookie("remember", remember.value, 7 * 86400);
        } else {
            //不记住密码
            removeCookie("username");
            removeCookie("password");
            removeCookie("remember");
        }
    };

    //显示用户密码信息
    window.onload = function() {
        username.value = getCookie("username") ? getCookie("username") : "";
```

```
        password.value = getCookie("password") ? getCookie("password") : "";
        remember.checked = getCookie("remember") ? true : false;
    }
</script>
```

6、案例2：自动登录

```
// express中服务端设置cookie
response.cookie(名,值,{maxAge: 过期时间毫秒});
// express中服务端获取cookie
request.cookies.名;
```

二、Storage

HTML5 提供了两种在客户端本地存储数据的新方法：localStorage和sessionStorage

1、Storage和Cookie的区别：

- 与服务器的交换方式不同：
  - cookie在浏览器与服务器之间来回传递，每次访问都需要携带cookie。
  - sessionStorage和localStorage不会把数据发给服务器,仅在本地保存。
- 过期时间不同：
  - cookie可以设置过期时间。
  - localStorage 没有过期时间，始终有效，长期保存。
  - sessionStorage 浏览器关闭后失效，自动销毁。
- 存储大小限制不同：
  - cookie数据不能超过4k
  - storage数据可以达到5M或更大
- 易用性不同：
  - cookie需要程序员自己封装，原生的操作方式不友好
  - storage原生操作方式简单易用

2、localStorage

2.1 设置数据

```
localStorage.setItem(名，值)
```

- 如果key已经存在，则覆盖key对应的value
- 如果不存在则添加key与value

2.2 获取数据

```
localStorage.getItem(名)
```

- 如果key不存在则返回null

2.3 删除数据

```
// 通过key来删除相应的数据
localStorage.removeItem(名)

// 将所有localStorage数据都清空
localStorage.clear();
```

2.4 获取数据长度

```
localStorage.length
```

2.5 storage事件

事件名：storage

当同源的localStorage有更改以后，会触发这个事件(sessionStorage也触发)

属性	含义
key	设置或删除或修改的键。调用clear()时，则为null。
oldValue	改变之前的旧值。如果是新增元素，则为null。
newValue	改变之后的新值。如果是删除元素，则为null。
storageArea	该属性是一个引用，指向发生变化的sessionStorage或localStorage对象
url	触发这个改变事件的页面的URL

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
    </body>
    <script>
        // window.onstorage = function(e) {
        //  //事件目标 输出：[object Window]对象（因为绑定在window上）
        //  console.log("target: " + e.target);
        //  //事件类型 输出：storage
        //  console.log("type : " + e.type);
        //  //事件是否冒泡 输出：false
        //  console.log("bubbles : " + e.bubbles);
        //  //事件是否可撤销 输出：false
        //  console.log("tarcancelable: " + e.cancelable);
        //  //键名
        //  console.log("key: " + e.key);
        //  //键值原值
        //  console.log("oldValue: " + e.oldValue);
        //  //键值新值
        //  console.log("newValue: " + e.newValue);
        //  //触发事件的url
        //  console.log("url: " + e.url);
```

```
        //  //受影响的存储空间 输出[object Storage]
        //  console.log("storageArea: " + e.storageArea);
        // }
        window.addEventListener("storage", function(e) {
            //事件目标 输出：[object Window]对象（因为绑定在window上）
            console.log("target: " + e.target);
            //事件类型 输出：storage
            console.log("type : " + e.type);
            //事件是否冒泡 输出：false
            console.log("bubbles : " + e.bubbles);
            //事件是否可撤销 输出：false
            console.log("tarcancelable: " + e.cancelable)
            //键名
            console.log("key: " + e.key);
            //键值原值
            console.log("oldValue: " + e.oldValue);
            //键值新值
            console.log("newValue: " + e.newValue);
            //触发事件的url
            console.log("url: " + e.url);
            //受影响的存储空间 输出[object Storage]
            console.log("storageArea: " + e.storageArea);
        });
    </script>
</html>
```

3、sessionStorage

- sessionStorage为临时性保存数据，当页面关闭就会消失，其他使用属性和方法与localStorage一样。
- sessionStorage不能跨页面访问，也不会触发跨标签页的storage事件，而localStorage可以。

```
// 设置数据
sessionStorage.setItem(名，值)

// 获取数据
sessionStorage.getItem(名)

// 通过key来删除相应的数据
sessionStorage.removeItem(名)

// 将所有数据都清空
sessionStorage.clear();

// 获取数据长度
sessionStorage.length
```

三、离线缓存-Application Cache

创建一个离线存储文件，引入到页面中。

文件名为manifest扩展名任意

```
CACHE MANIFEST
CACHE: 缓存内容
NETWORK:
    *
```

在页面中的html标签中用属性manifest指定该文件

```
<html manifest="manifest.manifest">

</html>
```

作业:

一、Cookie的作用及工作原理?

二、封装设置cookie函数

三、封装获取cookie函数

四、封装删除cookie函数

五、封装清除所有cookie函数

六、登录时记住用户名和密码

七、十天内自动登录

八、cookie和storage的区别

九、localStorage的设置、获取和删除操作

十、localStorage和sessionStorage的区别