第8章 浏览器对象模型

通过前面的学习，我们已经掌握了JavaScript这门语言的基础知识。但是正如在第一章在介绍JavaScript发展史时所提到的那样，JavaScript最初设计的目的就是用在浏览器里面的。所以从这一章开始，我们将一起来看一下如何使用JavaScript来与网页进行交互。

本章将学习如下内容：

- 在网页中引入JavaScript的方式
- 浏览器对象模型BOM
- window、navigator、location、history、screen以及document对象
- cookie的基础使用

8-1 浏览器引入JavaScript

要想利用JavaScript和网页实现互动，那么首先第一步就是应该了解如何向网页中引入JavaScript代码。实际上，向网页引入JavaScript代码的方式也是多种多样，下面我们一起来看一下。

8-1-1 直接在HTML文件中引入

首先第1种方式就是直接在HTML文档里面引入JavaScript代码。在维护一些老项目的时候，经常可以看到JavaScript代码是写在`<title>`标签下面的，并且通过一对`<script>`标签来引入代码。示例如下：

```
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        //JS代码
    </script>
</head>
```

但是，如果是直接在HTML文件中引入JavaScript代码，我们一般不写在`<title>`标签下面，更加优化的一种方式是将`<script>`标签写在`<body>`标签的最小面，这样可以更快的加载出页面效果。如下：

```
<body>
```

```
    <!-- HTML代码 -->
    <script>
        //JS代码
    </script>
</body>
```

8-1-2 调用外部JS文件

第2种方式是使用外部链接的方式来调用外部的JS文件。这种方式适用于当我们的JavaScript代码是单独的一个js文件时，就可以采用这种方式来引入JavaScript代码。示例如下：

```
<body>
    <!-- HTML代码 -->
    <script src="外部JS文件"></script>
</body>
```

8-1-3 使用a标记的href属性

我们还可以在 `<a>` 标签的 `href` 属性里面书写JavaScript代码，这样会改变 `<a>` 标签默认的超链接特性，而变为执行这段JavaScript代码，示例如下：

```
<body>
    <!-- HTML代码 -->
    <a href="javascript:alert('Hello World')">点击</a>
</body>
```

8-1-4 事件里面直接书写JS代码

在后面学习的事件中，我们也可以直接将JavaScript代码写在事件里面，例如我们书写一个点击事件，当点击button按钮时，会执行相应的JavaScript代码，示例如下：

```
<body>
    <!-- HTML代码 -->
    <button onclick="javascript:alert('Hello World!')">点击</button>
</body>
```

8-2 BOM基本介绍

8-2-1 什么是BOM

所谓BOM，英语全称为Browser Object Model，翻译成中文为浏览器对象模型。我们的浏览器，可以被看做是由各种各样的对象所组成的，就像我们这个可爱的世界一样。

一般来讲，在BOM中大致存在如下几个对象：

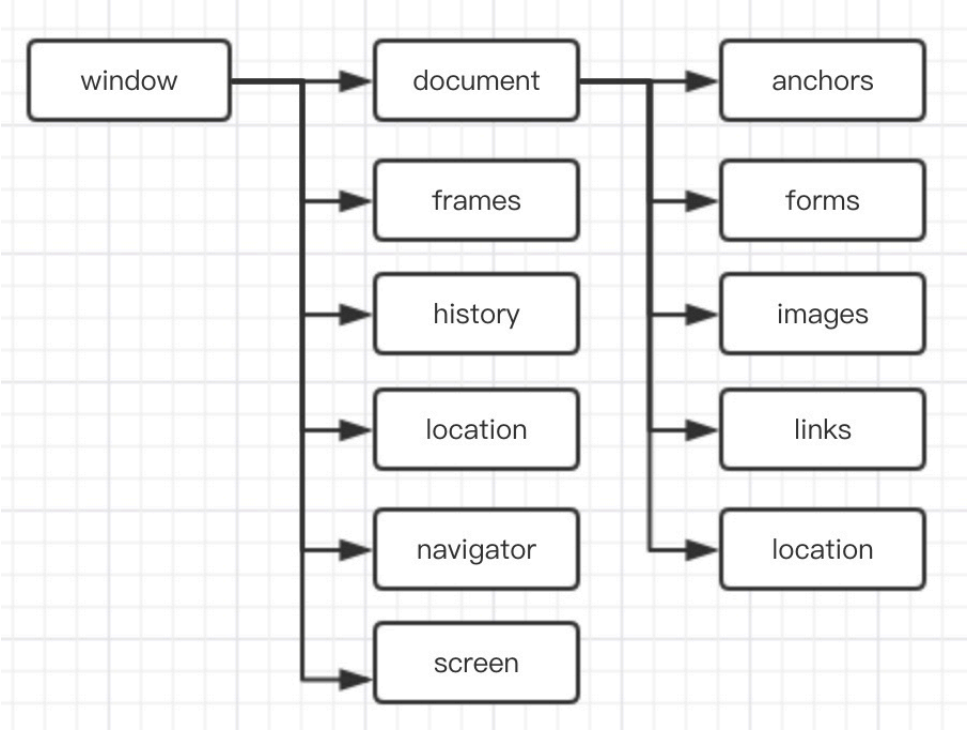
- window：表示窗口对象
- navigator：包含浏览器相关信息
- location：包含当前页面的位置信息
- history：包含用户访问页面的历史信息
- screen：包含客户端显示能力信息
- document：表示整个页面

整个BOM的核心对象就是是window对象，它代表的是浏览器的一个实例。window对象同时也是最顶层的对象。

8-2-2 BOM与DOM的关系

在下一章，我们还会介绍一个东西，被称之为DOM。很多初学者刚开始都搞不清楚BOM和DOM之间的关系，事实上很简单，DOM可以算是BOM的一个分支。因为BOM里面存在一个叫做Document的对象，但是这个对象的属性和方法太多了，所以W3C将其单独取了出来，做成了一套规范，这个就是DOM，英语全称document object model，翻译成中文就是文档对象模型。

接下来，我们可以用一张层级关系图来表示:



需要说明一下的是BOM目前还没有相应的规范。浏览器提供商会按照各自的想法去随意扩展它，于是浏览器之间共有的对象就成了事实上的标准。这些对象在浏览器中得以存在，很大程度上是由于它们提供了与浏览器的互操作性。目前，W3C已经将BOM的主要方面纳入了HTML5的规范中。

8-3 window对象

8-3-1 window对象基本介绍

我们先来看一下window对象。浏览器每打开一个窗口，就包含了一个window对象。window对象是整个BOM的核心对象，它代表着一个浏览器窗口的实例。

window对象扮演着在ES中的global对象的角色，因此所有在全局作用域中声明的变量以及函数都会成为该对象的属性和方法。也就是说全局变量是window对象的属性，全局函数是window对象的方法。我们来看下面的示例：

```
<body>
    <script>
        var a = 1;
        function test(){
            alert("this is a test");
        }
        alert(a);//1
        alert(window.a);//1
        test();//this is a test
        window.test();//this is a test
    </script>
</body>
```

可以看到，这里我们通过访问`window.a`和`window.test()`也可以打印出a变量的值和调用`test()`函数。

全局属性和window属性的区别

var和window对象的属性真的就是一模一样么?
也不是，还是有稍微不同的地方，那就是不能使用`delete`操作符删除用var声明的变量，但是如果是window的属性就可以使用`delete`操作符来进行删除。

```
<body>
    <script>
        var i = 10;
        window.j = 20;
        console.log(window.i);//10
        console.log(window.j);//20
        delete window.i;//并没有被删除掉
        delete window.j;//已经被删除掉了
```

```
        console.log(window.i);//可以打印出来
        console.log(window.j);//被删除了,所以是undefined
    </script>
</body>
```

let以及const所声明的变量

还需要注意的是，使用ES6新提供的let和const所声明的变量不会成为window对象的属性，证明如下：

```
<body>
    <script>
        var i = 10;
        let j = 20;
        const k = 30;
        alert(window.i);//10
        alert(window.j);//undefined
        alert(window.k);//undefined
    </script>
</body>
```

8-3-2 常见属性

既然是对象，那么肯定就有相应的属性和方法。这一小节，我们先来看一下window对象中常见的属性有哪些。

1. 窗口大小

关于窗口大小的属性有两组，`innerWidth`，`innerHeight`以及`outerWidth`，`outerHeight`区别在于：inner那一组表示的是页面视图区的大小，而outer那一组表示的是浏览器窗口本身的尺寸。

```
<body>
    <script>
        console.log(innerHeight);
        console.log(innerWidth);
        console.log(outerHeight);
        console.log(outerWidth);
    </script>
</body>
```

效果：

```
726
368
800
1280
```

但是，不同的浏览器，所表示的值略微有差异。

我们除了通过`innerWidth`，`innerHeight`来获取页面视图区的大小以外，还可以通过`document.documentElement.clientWidth`以及`document.documentElement.clientHeight`来获取页面视图区的大小，示例如下：

```
<body>
    <script>
        console.log('innerHeight:',innerHeight);
        console.log('innerWidth',innerWidth);
        console.log('outerHeight',outerHeight);
        console.log('outerWidth',outerWidth);
        console.log('document.documentElement.clientWidth:',document.documentElement.clientWidth);
        console.log('document.documentElement.clientHeight:',document.documentElement.clientHeight);
    </script>
</body>
```

效果：

```
innerHeight: 635
innerWidth 1280
outerHeight 709
outerWidth 1280
document.documentElement.clientWidth: 1280
document.documentElement.clientHeight: 635
```

那这两个属性有什么区别呢？具体如下表：

属性	描述
innerWidth	返回窗口的文档显示区的宽度(IE不支持)
innerHeight	返回窗口的文档显示区的高度(IE不支持)
document.documentElement.clientWidth	返回窗口的文档显示区的宽度(通用方法)
document.documentElement.clientHeight	返回窗口的文档显示区的宽度(通用方法)

2. 窗口位置

窗口位置的属性也是有两组，分别是`screenLeft`和`screenTop`，还有`screenX`和`screenY`这两组属性都是表示窗口相对于屏幕左边和上边的位置，如下：

```
<body>
    <script>
        console.log(window.screenLeft);
        console.log(window.screenTop);
        console.log("--------");
        console.log(window.screenX);
        console.log(window.screenY);
    </script>
</body>
```

效果：

```
22
45
--------
22
45
```

区别在于screenX和screenY属性最早是火狐浏览器里面特有的属性。但是上面的代码是在谷歌浏览器里面运行的，可以看到，现在这两组属性在很多浏览器里面都是通用的了。接下来我将上面两组属性的区别总结成下面的表格，如下：

属性	描述
screenX	返回浏览器相对于屏幕窗口的x坐标(IE不支持)
screenY	返回浏览器相对于屏幕窗口的y坐标，即距离浏览器最顶端(IE不支持)
screenLeft	返回浏览器相对于屏幕窗口的x坐标
screenTop	返回浏览器相对于屏幕窗口的y坐标(在IE中这个坐标包括了工具栏+菜单栏+地址栏的高度)

3. 元素位置

既然讲到了位置，那么这里我们提前来看一下如何获取一个元素在页面中的位置。我们可以通过`offsetLeft`以及`offsetTop`来得到一个元素在页面中的位置，位置的信息是不带单位的，示例如下：

```
<head>
```

```
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        div{
            width: 100px;
            height: 150px;
            background-color: pink;
            position: absolute;
            top: 300px;
            left: 200px;
        }
    </style>
</head>
<body>
    <div id="div"></div>
    <script>
        console.log(div.offsetLeft); // 200
        console.log(div.offsetTop); // 300
    </script>
</body>
```

4. 元素大小

除了获取到一个元素在页面中的位置，我们还可以通过`offsetWidth`和`offsetHeight`来得到一个元素的宽高，示例如下：

```
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        div{
            width: 100px;
            height: 150px;
            background-color: pink;
            position: absolute;
            top: 300px;
            left: 200px;
        }
    </style>
</head>
<body>
    <div id="div"></div>
    <script>
        console.log(div.offsetWidth); // 100
        console.log(div.offsetHeight); // 150
    </script>
```

```
</body>
```

5. 滚动位置

通过 `pageXOffset` 和 `pageYOffset` 属性我们可以获取文档在窗口左上角水平和垂直方向滚动的像素。

```
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        body {
            height: 5000px;
        }
    </style>
</head>
<body>
    <script>
        window.onscroll = function () {
            console.log(window.pageYOffset);
        }
    </script>
</body>
```

需要注意的是， `pageXOffset` 和 `pageYOffset` 属性相等于 `scrollX` 和 `scrollY` 属性。这些属性都是只读属性。

> 注：关于各浏览器对 `document.body.onscroll` ， `window.onscroll` ， `document.documentElement.onscroll` 的兼容性，可以参见下面这篇文档
> https://blog.csdn.net/qq_26445509/article/details/51153153

8-3-3 常用方法

接下来我们来一起看一下window对象中常见的方法有哪些。

1. 系统提示框

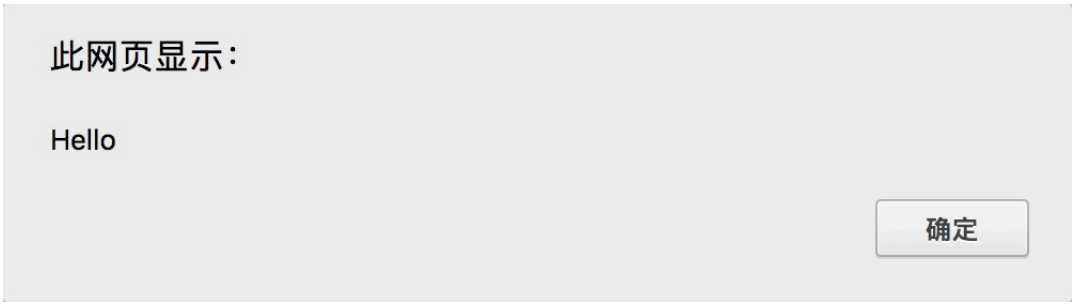
`alert()` ：用于显示带有一条指定消息和一个确定按钮的警告框。

> 注意：alert会终止JavaScript代码的执行

例如：

```
<body>
    <script>
        window.alert("Hello");
    </script>
</body>
```

运行结果：



`confirm()`：用于显示一个带有指定消息和 OK 及取消按钮的对话框。
例如：

```
<body>
    <script>
        let i = window.confirm("Are you sure?");
        console.log(i,typeof i);//true "boolean"
    </script>
</body>
```

注意返回的是一个布尔值，true或者false

`prompt()`：该对话框用于提示用户输入某些信息，得到的是一个字符串

```
<body>
    <script>
        let i = window.prompt("Please input something");
        console.log(i,typeof i);//用户输入内容  string
    </script>
</body>
```

2. 窗口大小调整

关于窗口大小，也有一组方法，`resizeTo()`和`resizeBy()`
`resizeTo()`：将浏览器窗口调整到指定的值
`resizeBy()`：相对于原来的浏览器窗口来进行调节

```
<body>
    <script>
        window.resizeTo(100,100);//调整到100x100
        window.resizeBy(100,50);//在原来的基础上增加100和50 变为200x150
    </script>
</body>
```

3. 窗口位置移动

关于窗口位置，我们如何来移动呢，在window对象里面提供了`moveTo()`和`moveBy()`
`moveTo()`方法是指定移动到哪个位置
`moveBy()`方法是在原来的位置上进行相对的偏移

```
<body>
    <script>
        window.moveTo(100,100);//移动到100x100的位置
        window.moveBy(100,50);//在原来的基础上向右移动100,向下移动50
    </script>
</body>
```

但是需要注意：这两组方法在浏览器里面基本上都是被禁用的，所以基本效果出不来。

4. 打开和关闭窗口

`open()`方法：使用`window.open()`方法，该方法有4个参数，如下：

```
<body>
    <script>
        //参数1:要加载的URL
        //参数2:指定target属性或窗口的名称
        //参数3:一个特性字符串,新窗口的大小
        //参数4:新窗口是否取代历史记录中的当前页面
        window.open("2.html","123","height=400,width=400,top=10,resizable=ye
s");
    </script>
</body>
```

效果：重新加载页面以后弹出新的窗口(2.html文件是提前准备好了的)

需要注意现在的浏览器一般都是会阻止弹窗的。一般需要我们点击允许弹出后，才弹出2.html页面

在调用了window对象的`open()`方法以后，他会返回一个对象，可以调用这些对象的方法，其中就有一个`close()`方法，示例如下：

```
<body>
    <button onclick="test()">关闭新窗口</button>
    <script>
        //参数1:要加载的URL
        //参数2:指定target属性或窗口的名称
        //参数3:一个特性字符串,新窗口的大小
        //参数4:新窗口是否取代历史记录中的当前页面
        let i = window.open("2.html","123","height=400,width=400,resizable=y
es");
        let test = function(){
            i.close();
        }
    </script>
</body>
```

还有一个`opener`属性，保存着打开它的原始窗口对象

```
<body>
    <script>
        //参数1:要加载的URL
        //参数2:指定target属性或窗口的名称
        //参数3:一个特性字符串,新窗口的大小
        //参数4:新窗口是否取代历史记录中的当前页面
        let i = window.open("2.html","123","height=400,width=400,resizable=y
es");
        console.log(i.opener === window);//true
    </script>
</body>
```

5. 定时函数

间歇调用

`setInterval()`与`clearInterval()`：这两个方法可以说是一组方法，前面是设定指定的时间周期调用某个函数，而后面的方法则是清除前面的设定。

`setInterval()`语法如下：
setInterval(函数名，间隔时间)
其中间隔时间以毫秒来计算，1000毫秒为1秒。该方法会返回一个id值，这个id值可以用于停止重复调用。

`clearInterval()`语法如下：
clearInterval(id)
作用是清除设置的间歇调用

示例：

```
<body>
    <script>
        let i = 1;
        let test = function(){
            console.log(i);
            i++;
            if(i == 10)
            {
                clearInterval(id);
            }
        }
        let id = setInterval("test()",1000);
    </script>
</body>
```

运行结果：依次打印出1-10

练习1：
制作一个时钟

```
<body>
    <span id="myTime"></span>
    <script>
        let myTime = document.getElementById("myTime");
        let showTime = function(){
            myTime.innerText = new Date().toLocaleString();
        }
        setInterval("showTime()",1000);
    </script>
</body>
```

运行结果：

2017/11/28 上午11:04:02

超时调用

`setTimeout()`和`clearTimeout()`：这两个也可以算是一组方法，前面是设定指定的时间周期后调用某个函数，而后面的方法是清除前面的设定。

`setTimeout()`语法如下：
setInterval(函数名，间隔时间)
其中间隔时间也是指代的毫秒数，但是这个毫秒数的含义是执行代码前需要等待的毫秒数。该方法也会返回一个id，可用于clearTimeout

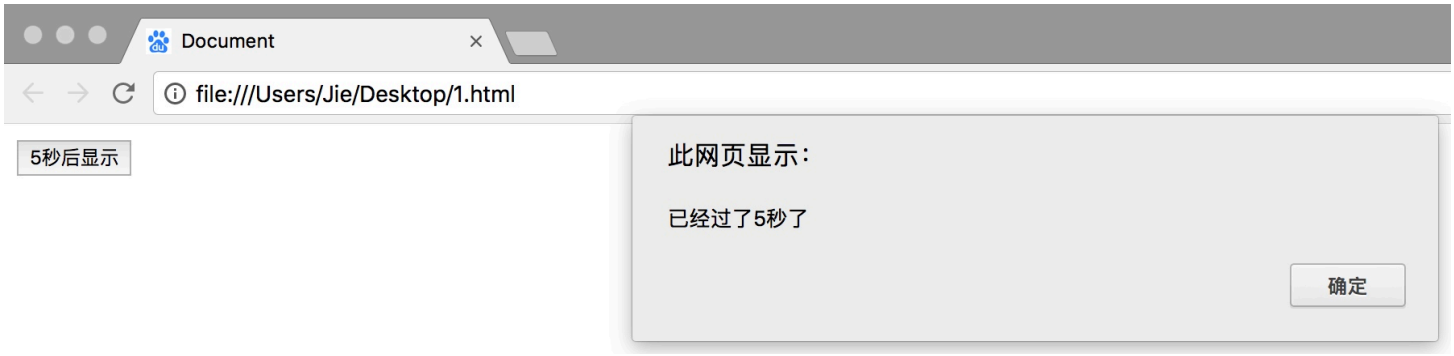
`clearTimeout()`语法如下：
clearInterval(id)
作用是清除设置的超时调用

示例：

```
<body>
    <button id="btn">5秒后显示</button>
    <script>
        let btn = document.getElementById("btn");
        btn.onclick = function(){
            setTimeout("alert('已经过了5秒了')",5000);
```

```
        }
    </script>
</body>
```

效果：点击按钮后过5秒钟弹出提示框



下面的例子显示了clearTimeout的用法，其实就是清除前面的设定

```
<body>
    <button id="btn">5秒后显示</button>
    <script>
        let btn = document.getElementById("btn");
        btn.onclick = function(){
            setTimeout("alert('已经过了5秒了')",5000);
            clearTimeout(i);
        }
    </script>
</body>
```

8-4 navigator对象

8-4-1 navigator对象介绍

navigator对象包含浏览器和运行浏览器的操作系统的大量信息。例如操作系统版本，浏览器类型和版本等信息。很多时候我们需要在判断网页所处的浏览器和平台，navigator对象为我们提供了便利。

8-4-2 navigator对象属性

navigator对象所包含的属性大致如下表：

属性	描述
appCodeName	返回浏览器的代码名
appMinorVersion	返回浏览器的次级版本
appName	返回浏览器名称
appVersion	返回浏览器的平台和版本信息
browserLanguage	返回当前浏览器的语言
cookieEnabled	返回指明浏览器中是否启动cookie的布尔值
cpuClass	返回浏览器系统的CPU等级
onLine	返回指明系统是否处于脱机模式的布尔值
platform	返回运行浏览器的操作系统平台
systemLanguage	返回OS使用的默认的语言
userAgent	返回由客户机发送服务器的user-agent头部的值
userLanguage	返回OS的自然语言设置

示例：

```
<body>
    <script>
        console.log("浏览器名称:",navigator.appName);
```

```
        console.log("浏览器版本:",navigator.appVersion);
        console.log("浏览器主语言:",navigator.language);
        console.log("产品名称:",navigator.product);
    </script>
</body>
```

效果：

浏览器名称: Netscape

浏览器版本: 5.0 (Macintosh; Intel Mac OS X 10_13_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36

浏览器主语言: zh-CN

产品名称: Gecko

8-5 location对象

location对象提供了当前窗口中加载的文档的有关信息。这个对象有点特别，它既是window对象的属性，也是document对象的属性，也就是说`window.location`和`document.location`调用的是同一个对象，我们也可以通过下面的方式来进行证明，如下：

```
<body>
    <script>
        console.log(window.location === document.location);//true
    </script>
</body>
```

8-5-1 常见属性

常见的location对象的属性如下表：

属性	描述
hash	返回一个URL的锚部分
host	返回一个URL的主机名和端口
hostname	返回URL的主机名
href	返回完整的URL
pathname	返回的URL路径名
port	返回一个URL服务器使用的端口号
protocol	返回一个URL协议
search	返回一个URL的查询部分

这里主要介绍一下`href`属性，可以查看或者设置location的`href`属性
查看当前页面的URL：

```
<body>
    <script>
        console.log(location.href);
        //file:///Users/Jie/Desktop/1.html
    </script>
```

```
</body>
```

设置当前页面的URL：

```
<body>
    <button onclick="test()">点击我们跳转</button>
    <script>
        let test = function(){
            location.href = "http://www.baidu.com";
        }
    </script>
</body>
```

效果：点击按钮后跳转到百度

接下来我们来看一下其他的属性：

- hash：如果URL中包含有“#”，该方法将返回该符号之后的内容 例如：`http://www.sunchis.com/index.html#welcome` 的hash是 `#welcome`
- host：服务器的名字，例如 `www.sunchis.com`
- hostname：通常等于 `host`，有时会省略前面的 `www`
- href：当前页面载入的完整URL。
- pathname：URL中主机名之后的部分。例如：`http://www.sunchis.com/html/js/jsbasic/2010/0319/88.html` 的pathname是 `/html/js/jsbasic/2010/0319/88.html`。
- port：URL中声明的请求端口。默认情况下，大多数URL没有端口信息（默认为80端口），所以该属性通常是空白的。像 `http://www.sunchis.com:8080/index.html` 这样的URL的port属性为 `8080`
- protocol：URL中使用的协议，即双斜杠（//）之前的部分。例如 `http://www.sunchis.com` 中的protocol属性等于 `http:`，`ftp://www.sunchis.com` 的protocol属性等于 `ftp:`
- search：执行GET请求的URL中的问号（?）后的部分，又称查询字符串。例如 `http://www.sunchis.com/search.html?tern=sunchis` 中search属性为 `?term=sunchis`

8-5-2 常见方法

location对象中常见的方法如下表：

方法	说明

assign()	载入一个新的文档
reload()	重新载入当前文档
replace()	用新的文档替换当前文档

接下来我们一个一个来看。

`reload()`：该方法用于重新加载当前文档。

示例：

```
<body>
    <button onclick="test()">刷下</button>
    <script>
        let test = function(){
            location.reload();
        }
    </script>
</body>
```

效果：点击按钮后会刷新一下页面

接下来的两个方法看上去是差不多的，分别是assign和replace方法。

`assign()`：该方法加载新的文档。

示例：

```
<body>
    <button onclick="test()">加载</button>
    <script>
        let test = function(){
            location.assign("http://www.baidu.com");
        }
    </script>
</body>
```

`replace()`：该方法可用一个新文档取代当前文档。

示例：

```
<body>
    <button onclick="test()">替换</button>
    <script>
        let test = function(){
            location.replace("http://www.baidu.com");
```

```
        }
    </script>
</body>
```

那么这两个方法的区别是什么呢?

`assign()`方法：加载 URL 指定的新的 HTML 文档。就相当于一个链接，跳转到指定的URL，当前页面会转为新页面内容，可以点击后退返回上一个页面。

`replace()`方法：通过加载 URL 指定的文档来替换当前文档，这个方法是替换当前窗口页面，前后两个页面共用一个窗口，所以是没有后退返回上一页的。

跳转的方式

最后我们来总结一下页面跳转的方式，抛开`<a>`标签，通过JavaScript代码来实现页面的跳转大致有下面几种方式:

- location.href
- window.location
- location
- location.assign()
- location.replace()

8-6 history对象

history对象用来管理当前窗口最近访问过的URL记录，这些URL记录被保存在history列表中，history对象使得脚本程序可以模拟浏览器工具栏的前进和后退按钮。

8-6-1 常见属性

history对象常见属性如下表：

属性	描述
length	返回浏览器历史列表中URL的数量

示例：

```
<body>
    <button onclick="test()">查看历史列表数量</button>
    <script>
        let test = function(){
            console.log("历史列表数:",history.length);
        }
    </script>
</body>
```

8-6-2 常见方法

history对象常见的方法如下表：

方法	描述
back()	加载history列表中的前一个URL
forward()	加载history列表中的下一个URL
go()	加载history列表中的某个具体页面

window.history.go()方法可以用来导航到指定的页面，0代表是当前页面

```
<script>
    window.history.go(1);//向前一个页面
```

```
    window.history.go(0);//重新加载当前页面
    window.history.go(-1);//回退一个页面
</script>
```

还有 `window.history.forward()` 和 `window.history.back()` 方法可以分别用来向前或者回退一个页面，就像浏览器的前进和后退按钮一样。

8-7 screen对象

有时脚本需要获取浏览器或者显示器的一些信息，例如分辨率，有效分辨率，DPI等。这个时候我们就可以使用screen对象。该对象提供了一组属性，供我们来获取到这些有用的信息。

> 注意：目前没有应用于screen对象的公开标准，不过所有浏览器都支持该对象。

screen对象属性如下表：

属性	说明
height	屏幕的像素高度
width	屏幕的像素宽度
availHeight	屏幕的像素高度减去系统部件高度之后的值(只读)
availWidth	屏幕的像素宽度减去系统部件宽度之后的值(只读)
left	当前屏幕距左边的像素距离[firefox返回0，chrome和IE不支持]
top	当前屏幕距上方的像素距离[firefox返回0，chrome和IE不支持]
availLeft	未被系统部件占用的最左侧的像素值(只读)[chrome和firefox返回0，IE不支持]
availTop	未被系统部件占用的最上方的像素值(只读)[chrome和firefox返回0，IE不支持]
bufferDepth	读、写用于呈现屏外位图的位数[IE返回0，chrome和firefox不支持]
colorDepth	用于表现颜色的位数(只读)[IE8-返回32，其他浏览器返回24]
pixelDepth	屏幕的位深(只读)[IE8-不支持，其他浏览器返回24]
deviceXDPI	屏幕实际的水平DPI(只读)[IE返回96，chrome和firefox不支持]
deviceYDPI	屏幕实际的垂直DPI(只读)[IE返回96，chrome和firefox不支持]
logicalXDPI	屏幕逻辑的水平DPI(只读)[IE返回96，chrome和firefox不支持]
logicalYDPI	屏幕逻辑的垂直DPI(只读)[IE返回96，chrome和firefox不支持]
updateInterval	读、写以毫秒表示的屏幕刷新时间间隔[IE返回0，chrome和firefox不支持]

fontSmoothingEnabled	是否启用了字体平滑(只读)[IE返回true，chrome和firefox不支持]

个别属性示例：

```
<body>
    <script>
        console.log("屏幕高度:",screen.height);
        console.log("屏幕宽度:",screen.width);
        console.log("调色板比特深度:",screen.colorDepth);
    </script>
</body>
```

效果：

```
屏幕高度: 800
屏幕宽度: 1280
调色板比特深度: 24
```

8-8 document对象

在后面的DOM中，我们会专门介绍document相关的大量的方法。这里就先介绍一个即可。`write()`方法，用于将一串文本写入页面。如果页面已经加载了，它将完全替换当前的文档。
示例：

```
<body>
    <p>Lorem ipsum dolor sit amet.</p>
    <button onclick="test()">按钮</button>
    <script>
        let test = function(){
            document.write("this is a test");
            document.write("this is a test,too");
        }
    </script>
</body>
```

效果：点击前

Lorem ipsum dolor sit amet.

按钮

点击后

this is a testthis is a test,too

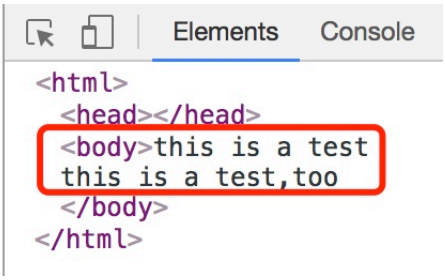
可以看到，使用`document.write()`书写页面的时候，原来的东西会完全被替换掉。并且我们书写了两行东西，这里两行东西是在一行里面显示的。与`document.write()`对应的有一个`document.writeln()`，这个方法就可以实现换行输出，但是这里的换行指的是代码里面会换行，页面上仍然是显示成一行的。不过多了一个空格，因为html存在空白折叠现象。书写的新的内容仍然会将之前的内容给替换掉。

```
<body>
    <p>Lorem ipsum dolor sit amet.</p>
    <button onclick="test()">按钮</button>
    <script>
        let test = function(){
            document.writeln("this is a test");
            document.writeln("this is a test,too");
        }
```

```
    </script>
</body>
```

效果：

this is a test this is a test,too



cookie

cookie是保存在用户计算机上的小文件，是网景公司发明的。之所以当时发明cookie这个东西，是因为HTTP协议是一种无状态的协议，这意味着从一个请求转到另一个请求后，浏览器不记得前一个请求的任何东西。于是每次用户访问一个页面，都记不住任何以前的访问。

所以网景公司就发明了cookie，用它来存储一些信息，从而避免了无状态这个问题。使用cookie我们可以实现个性化用户的浏览体验，例如存储用户偏好，跟踪用户选择(购物车)，身份验证和跟踪用户等。

近年来，用于追踪目的的cookie已经被滥用，所以它们在被用作数据存储的情况下渐渐的被新的HTML5 localStorage API所取代，因为它允许存储更多的数据。不过目前来讲，cookie对于保留状态信息(例如用户登录)仍然有用，因为每次HTTP请求发生时，它们都会在客户端和服务器端之间进行传递。

1. 创建cookie

要创建一个cookie，只需要将要保存的信息赋值给document.cookie属性即可，如下：

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <script>
        let setCookie = function(){
            document.cookie = "name = xiejie";
        }
        let showCookie = function(){
            console.log(document.cookie);
        }
    </script>
</body>
```

效果：

设置cookie　显示cookie

首先点击设置cookie进行设置，然后点击显示cookie

```
name=xiejie
```

> 需要注意谷歌浏览器下面需要搭建服务器环境才能看到效果(火狐浏览器不需要)，这里涉及到了事件的相关知识，关于事件我们会在第9章做详细的介绍。

可以追加cookie内容，只需要添入新的键值对即可

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <button onclick="addCookie()">追加cookie</button>
    <script>
        let setCookie = function(){
            document.cookie = "name = xiejie";
        }
        let showCookie = function(){
            console.log(document.cookie);
        }
        let addCookie = function(){
            document.cookie = "age = 18";
        }
    </script>
</body>
```

效果：先点击设置cookie，接下来显示cookie，然后点击追加cookie，然后再次显示

```
name=xiejie
name=xiejie; age=18
```

2. 修改cookie

如果要修改cookie的值也非常简单，只需要对已有的键重新赋值就可以了。示例如下：

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <button onclick="changeCookie()">修改cookie</button>
    <script>
        let setCookie = function(){
            document.cookie = "name = xiejie";
        }
```

```
        let showCookie = function(){
            console.log(document.cookie);
        }
        let changeCookie = function(){
            document.cookie = "name = yajing";
        }
    </script>
</body>
```

效果：先点击设置cookie，然后显示cookie，接下来点击修改cookie，再显示cookie

```
name=xiejie
name=yajing
```

3. 读取cookie

读取cookie只需要键入document.cookie即可，这个前面已经用过了。我们可以使用split()方法将cookie的字符串拆分成数组，然后使用for-of进行循环遍历

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <button onclick="changeCookie()">遍历cookie</button>
    <script>
        let setCookie = function(){
            //注意设置多个cookie的时候，只能这么多次赋值才进行设置
            document.cookie = "name = xiejie";
            document.cookie = "age = 18";
            document.cookie = "gender = male";
        }
        let showCookie = function(){
            console.log(document.cookie);
        }
        let changeCookie = function(){
            let cookies = document.cookie.split("; ");//分隔符为;加上一个空格
            for(let i of cookies)
            {
                let [key,value] = i.split("=");//解构
                console.log("cookie的键为:",key,"值为:",value);
            }
        }
    </script>
</body>
```

效果：

```
name=xiejie; age=18; gender=male
cookie的键为: name 值为: xiejie
cookie的键为: age 值为: 18
cookie的键为: gender 值为: male
```

如果要随意查看cookie的示例，可以访问任意网站，打开浏览器控制台，键入document.cookie即可。

4. cookie失效时间

cookie默认是会话的cookie，也就是说，一旦浏览器会话完成(也就是用户关闭浏览器标签页或者窗口时)，cookie就会被删除。
不过，cookie可以变成持久的，只需在设置cookie的时候，在cookie末尾添加'; expires=日期值'来设置失效日期即可，示例如下：

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <script>
        let setCookie = function(){
            //获取当前的日期时间
            let expiryDate = new Date();
            //在当前的日期时间上添加上1天
            let tomorrow = expiryDate.getTime() + 86400;
            //重新设置日期时间
            expiryDate.setTime(tomorrow);
            //设置该cookie的失效时间
            document.cookie = `name=xiejie;expires=${expiryDate.toUTCString(
)}`;
            document.cookie = "age = 18";
        }
        let showCookie = function(){
            console.log(document.cookie);
        }
    </script>
</body>
```

另一种方案是设置`max-age`值。这个值的单位为秒，不过IE10之前的版本不支持

```
document.cookie = "name = xiejie; max-age = 86400";//86400秒 = 1天
```

5. cookie的路径和域

默认情况下，cookie只能通过与设置它的文件在相同的目录和域中的页面来读取。这是出于安全的原因，所以访问cookie是有限制的。
路径可以进行修改。例如我们要让根目录中的任何页面都可以读取cookie。可以通过在设置cookie时，在cookie的末尾添加`; path=/`来实现，如下：

```
document.cookie = "name = xiejie; path = /";
```

除此之外，我们还可以在cookie的末尾添加`; domain = domainName`来设置域：

```
document.cookie = "name = xiejie; domain = syaketu.com";
```

原来的cookie，只能被创建它的域所读取，不过像上面这样我们手动设置了域之后，就可以让一个域下面的所有子域(例如：html.syaketu.com，ruby.syaketu.com)都可以读取这个cookie。

6. 保护cookie安全

这个也非常简单，在一个cookie的末尾添加上字符串'; secure'即可，这样可以确保它只能通过安全的HTTPS网络来进行传输

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <script>
        let setCookie = function(){
            //注意设置多个cookie的时候，只能这么多次赋值才进行设置
            document.cookie = "name = xiejie";
            document.cookie = "age = 18";
            document.cookie = "gender = male;secure";//只通过HTTPS网络传输
        }
        let showCookie = function(){
            console.log(document.cookie);
        }
    </script>
</body>
```

效果：由于gender=male添加了secure，所以只能通过HTTPS网络来传输

```
name=xiejie; age=18
```

7. 删除cookie

要删除一个cookie的方法非常简单，只需要将cookie的失效时间设置在过去的某个时间过期即

可。如下：

```
<body>
    <button onclick="setCookie()">设置cookie</button>
    <button onclick="showCookie()">显示cookie</button>
    <button onclick="deleCookie()">删除cookie</button>
    <script>
        let setCookie = function(){
            //注意设置多个cookie的时候，只能这么多次赋值才进行设置
            document.cookie = "name = xiejie";
            document.cookie = "age = 18";
            document.cookie = "gender = male";
        }
        let showCookie = function(){
            console.log(document.cookie);
        }
        let deleCookie = function(){
            document.cookie = "age = 18;expires = Thu,01 Jan 1990 00:00:01 G
MT";
        }
    </script>
</body>
```

效果：先点击设置，然后显示，接下来删除，再显示，可以看到键为name的cookie已被删除掉了

```
name=xiejie; age=18; gender=male
name=xiejie; gender=male
```

总结

1. 向浏览器引入JavaScript代码有多种方式。
2. BOM翻译成中文是浏览器对象模型，这里将整个浏览器看作了是一个对象。
3. window对象是BOM中的顶层对象。全局作用域下使用var声明的变量以及函数都是该对象的属性和方法。
4. navigator对象包含浏览器和运行浏览器的操作系统的大量信息。
5. location对象提供了当前窗口中加载的文档的有关信息。
6. history对象用来管理当前窗口最近访问过的URL记录。
7. screen对象提供了有关浏览器和显示器的一些信息，例如屏幕分辨率，DPI等。
8. cookie是保存在用户计算机上的小文件，用它来存储一些信息，从而避免了HTTP协议的无状态所带来的问题。