# window对象

## 概述

浏览器里面，window对象（注意，w为小写）指当前的浏览器窗口。它也是当前页面的顶层对象，即最高一层的对象，所有其他对象都是它的下属。一个变量如果未声明，那么默认就是顶层对象的属性。

a = 1;window.a // 1

上面代码中，a是一个没有声明就直接赋值的变量，它自动成为顶层对象的属性。

window有自己的实体含义，其实不适合当作最高一层的顶层对象，这是一个语言的设计失误。最早，设计这门语言的时候，原始设想是语言内置的对象越少越好，这样可以提高浏览器的性能。因此，语言设计者 Brendan Eich 就把window对象当作顶层对象，所有未声明就赋值的变量都自动变成window对象的属性。这种设计使得编译阶段无法检测出未声明变量，但到了今天已经没有办法纠正了。

## window 对象的属性

### window.name

window.name属性是一个字符串，表示当前浏览器窗口的名字。窗口不一定需要名字，这个属性主要配合超链接和表单的target属性使用。

window.name = 'Hello World!';console.log(window.name)// "Hello World!"

该属性只能保存字符串，如果写入的值不是字符串，会自动转成字符串。各个浏览器对这个值的储存容量有所不同，但是一般来说，可以高达几MB。

只要浏览器窗口不关闭，这个属性是不会消失的。

### 全局对象属性

全局对象属性指向一些浏览器原生的全局对象。

* window.document：指向document对象。
* window.location：指向Location对象，用于获取当前窗口的 URL 信息。它等同于document.location属性。
* window.navigator：指向Navigator对象，用于获取环境信息。
* window.history：指向History对象，表示浏览器的浏览历史。
* window.localStorage：指向本地储存的 localStorage 数据。
* window.sessionStorage：指向本地储存的 sessionStorage 数据。
* window.console：指向console对象，用于操作控制台。

## window 对象的方法

### window.alert()，window.prompt()，window.confirm()

window.alert()、window.prompt()、window.confirm()都是浏览器与用户互动的全局方法。它们会弹出不同的对话框，要求用户做出回应。注意，这三个方法弹出的对话框，都是浏览器统一规定的式样，无法定制。

**（1）window.alert()**

window.alert()方法弹出的对话框，只有一个“确定”按钮，往往用来通知用户某些信息。

window.alert('Hello World');

用户只有点击“确定”按钮，对话框才会消失。对话框弹出期间，浏览器窗口处于冻结状态，如果不点“确定”按钮，用户什么也干不了。

window.alert()方法的参数只能是字符串，没法使用 CSS 样式，但是可以用\n指定换行。

alert('本条提示\n分成两行');

**（2）window.prompt()**

window.prompt()方法弹出的对话框，提示文字的下方，还有一个输入框，要求用户输入信息，并有“确定”和“取消”两个按钮。它往往用来获取用户输入的数据。

var result = prompt('您的年龄？', 25)

上面代码会跳出一个对话框，文字提示为“您的年龄？”，要求用户在对话框中输入自己的年龄（默认显示25）。用户填入的值，会作为返回值存入变量result。

window.prompt()的返回值有两种情况，可能是字符串（有可能是空字符串），也有可能是null。具体分成三种情况。

1. 用户输入信息，并点击“确定”，则用户输入的信息就是返回值。
2. 用户没有输入信息，直接点击“确定”，则输入框的默认值就是返回值。
3. 用户点击了“取消”（或者按了 ESC 按钮），则返回值是null。

window.prompt()方法的第二个参数是可选的，但是最好总是提供第二个参数，作为输入框的默认值。

**（3）window.confirm()**

window.confirm()方法弹出的对话框，除了提示信息之外，只有“确定”和“取消”两个按钮，往往用来征询用户是否同意。

var result = confirm('你最近好吗？');

上面代码弹出一个对话框，上面只有一行文字“你最近好吗？”，用户选择点击“确定”或“取消”。

confirm方法返回一个布尔值，如果用户点击“确定”，返回true；如果用户点击“取消”，则返回false。

var okay = confirm('Please confirm this message.');if (okay) {

// 用户按下“确定”

} else {

// 用户按下“取消”

}

### window.scroll(), window.scrollTo(), window.scrollBy()

窗口滚动功能

window.scroll(x, y)

区别：

scroll()和scrollTo()一样。

scrollBy()会在之前的数据上累加。

# History对象

## 概述

window.history属性指向 History 对象，它表示当前窗口的浏览历史。

History 对象保存了当前窗口访问过的所有页面网址。

下面代码表示当前窗口一共访问过3个网址。

window.history.length // 3

由于安全原因，浏览器不允许脚本读取这些地址，但是允许在地址之间导航。

// 后退到前一个网址

history.back()

// 等同于

history.go(-1)

浏览器工具栏的“前进”和“后退”按钮，其实就是对 History 对象进行操作。

## 方法

### history.back()、history.forward()、history.go()

这三个方法用于在历史之中移动。

* history.back()：移动到上一个网址，等同于点击浏览器的后退键。对于第一个访问的网址，该方法无效果。
* history.forward()：移动到下一个网址，等同于点击浏览器的前进键。对于最后一个访问的网址，该方法无效果。
* history.go()：接受一个整数作为参数，以当前网址为基准，移动到参数指定的网址，比如go(1)相当于forward()，go(-1)相当于back()。如果参数超过实际存在的网址范围，该方法无效果；如果不指定参数，默认参数为0，相当于刷新当前页面。

history.back();

history.forward();

history.go(-2);

history.go(0)相当于刷新当前页面。

history.go(0);

注意，移动到以前访问过的页面时，页面通常是从浏览器缓存之中加载，而不是重新要求服务器发送新的网页。

# Location 对象

Location对象是浏览器提供的原生对象，提供 URL 相关的信息和操作方法。通过window.location和document.location属性，可以拿到这个对象。

## 属性

Location对象提供以下属性。

* location.href：整个 URL。
* location.protocol：当前 URL 的协议，包括冒号（:）。
* location.host：主机，包括冒号（:）和端口（默认的80端口和443端口会省略）。
* location.hostname：主机名，不包括端口。
* location.port：端口号。
* location.pathname：URL 的路径部分，从根路径/开始。
* location.search：查询字符串部分，从问号?开始。
* location.hash：片段字符串部分，从#开始。
* location.username：域名前面的用户名。
* location.password：域名前面的密码。
* location.origin：URL 的协议、主机名和端口。

这些属性里面，只有origin属性是只读的，其他属性都可写。

注意，如果对Location.href写入新的 URL 地址，浏览器会立刻跳转到这个新地址。

// 跳转到新网址document.location.href = 'http://www.example.com';

这个特性常常用于让网页自动滚动到新的锚点。

document.location.href = '#top';

直接改写location，相当于写入href属性。

document.location = 'http://www.example.com';// 等同于document.location.href = 'http://www.example.com';

另外，location.href属性是浏览器唯一允许跨域写入的属性，即非同源的窗口可以改写另一个窗口的location.href属性，导致后者的网址跳转。location的其他属性都不允许跨域写入。

## 方法

**location.reload()**

reload方法使得浏览器重新加载当前网址，相当于按下浏览器的刷新按钮。

它接受一个布尔值作为参数。如果参数为true，浏览器将向服务器重新请求这个网页，并且重新加载后，网页将滚动到头部（即scrollTop === 0）。如果参数是false或为空，浏览器将从本地缓存重新加载该网页，并且重新加载后，网页的视口位置是重新加载前的位置。

// 向服务器重新请求当前网址window.location.reload(true);

## URL 的编码和解码

网页的 URL 只能包含合法的字符。合法字符分成两类。

* URL 元字符：分号（;），逗号（,），斜杠（/），问号（?），冒号（:），at（@），&，等号（=），加号（+），美元符号（$），井号（#）
* 语义字符：a-z，A-Z，0-9，连词号（-），下划线（\_），点（.），感叹号（!），波浪线（~），星号（\*），单引号（'），圆括号（()）

除了以上字符，其他字符出现在 URL 之中都必须转义，规则是根据操作系统的默认编码，将每个字节转为百分号（%）加上两个大写的十六进制字母。

比如，UTF-8 的操作系统上，http://www.example.com/q=春节这个 URL 之中，汉字“春节”不是 URL 的合法字符，所以被浏览器自动转成http://www.example.com/q=%E6%98%A5%E8%8A%82。其中，“春”转成了%E6%98%A5，“节”转成了%E8%8A%82。这是因为“春”和”节“的 UTF-8 编码分别是E6 98 A5和E8 8A 82，将每个字节前面加上百分号，就构成了 URL 编码。

JavaScript 提供四个 URL 的编码/解码方法。

* **encodeURI()**
* encodeURIComponent()
* **decodeURI()**
* decodeURIComponent()

### encodeURI()

encodeURI()方法用于转码整个 URL。它的参数是一个字符串，代表整个 URL。它会将元字符和语义字符之外的字符，都进行转义。

encodeURI('http://www.example.com/q=春节')// "http://www.example.com/q=%E6%98%A5%E8%8A%82"

### decodeURI()

decodeURI()方法用于整个 URL 的解码。它是encodeURI()方法的逆运算。它接受一个参数，就是转码后的 URL。

decodeURI('http://www.example.com/q=%E6%98%A5%E8%8A%82')// "http://www.example.com/q=春节"

// "春节"

# Cookie

## 概述

Cookie 是服务器保存在浏览器的一小段文本信息，每个 Cookie 的大小一般不能超过4KB。浏览器每次向服务器发出请求，就会自动附上这段信息。

Cookie 主要用来分辨两个请求是否来自同一个浏览器，以及用来保存一些状态信息。它的常用场合有以下一些。

* 对话（session）管理：保存登录、购物车等需要记录的信息。
* 个性化：保存用户的偏好，比如网页的字体大小、背景色等等。
* 追踪：记录和分析用户行为。

有些开发者使用 Cookie 作为客户端储存。这样做虽然可行，但是并不推荐，因为 Cookie 的设计目标并不是这个，它的容量很小（4KB），缺乏数据操作接口，而且会影响性能。客户端储存应该使用 Web storage API 和 IndexedDB。

Cookie 包含以下几方面的信息。

* Cookie 的名字
* Cookie 的值（真正的数据写在这里面）
* 到期时间
* 所属域名（默认是当前域名）
* 生效的路径（默认是当前网址）

举例来说，用户访问网址www.example.com，服务器在浏览器写入一个 Cookie。这个 Cookie 就会包含www.example.com这个域名，以及根路径/。这意味着，这个 Cookie 对该域名的根路径和它的所有子路径都有效。如果路径设为/forums，那么这个 Cookie 只有在访问www.example.com/forums及其子路径时才有效。以后，浏览器一旦访问这个路径，浏览器就会附上这段 Cookie 发送给服务器。

浏览器可以设置不接受 Cookie，也可以设置不向服务器发送 Cookie。window.navigator.cookieEnabled属性返回一个布尔值，表示浏览器是否打开 Cookie 功能。

// 浏览器是否打开 Cookie 功能window.navigator.cookieEnabled // true

document.cookie属性返回当前网页的 Cookie。

// 当前网页的 Cookiedocument.cookie

不同浏览器对 Cookie 数量和大小的限制，是不一样的。一般来说，单个域名设置的 Cookie 不应超过30个，每个 Cookie 的大小不能超过4KB。超过限制以后，Cookie 将被忽略，不会被设置。

浏览器的同源政策规定，两个网址只要域名相同和端口相同，就可以共享 Cookie

注意，这里不要求协议相同。

## Cookie 的属性

### Expires，Max-Age

Expires属性指定一个具体的到期时间，到了指定时间以后，浏览器就不再保留这个 Cookie。它的值是 UTC 格式，可以使用Date.prototype.toUTCString()进行格式转换。

Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;

如果不设置该属性，或者设为null，Cookie 只在当前会话（session）有效，浏览器窗口一旦关闭，当前 Session 结束，该 Cookie 就会被删除。另外，浏览器根据本地时间，决定 Cookie 是否过期，由于本地时间是不精确的，所以没有办法保证 Cookie 一定会在服务器指定的时间过期。

Max-Age属性指定从现在开始 Cookie 存在的秒数，比如60 \* 60 \* 24 \* 365（即一年）。过了这个时间以后，浏览器就不再保留这个 Cookie。

如果同时指定了Expires和Max-Age，那么Max-Age的值将优先生效。

如果Set-Cookie字段没有指定Expires或Max-Age属性，那么这个 Cookie 就是 Session Cookie，即它只在本次对话存在，一旦用户关闭浏览器，浏览器就不会再保留这个 Cookie。

### document.cookie

document.cookie属性用于读写当前网页的 Cookie。

读取的时候，它会返回当前网页的所有 Cookie

document.cookie // "foo=bar;baz=bar"

上面代码从document.cookie一次性读出两个 Cookie，它们之间使用分号分隔。必须手动还原，才能取出每一个 Cookie 的值。

var cookies = document.cookie.split(';');

for (var i = 0; i < cookies.length; i++) {

  console.log(cookies[i]);

}// foo=bar// baz=bar

document.cookie属性是可写的，可以通过它为当前网站添加 Cookie。

document.cookie = 'fontSize=14';

写入的时候，Cookie 的值必须写成key=value的形式。注意，等号两边不能有空格。另外，写入 Cookie 的时候，必须对分号、逗号和空格进行转义（它们都不允许作为 Cookie 的值），这可以用encodeURIComponent方法达到。

但是，document.cookie一次只能写入一个 Cookie，而且写入并不是覆盖，而是添加。document.cookie读写行为的差异（一次可以读出全部 Cookie，但是只能写入一个 Cookie），与 HTTP 协议的 Cookie 通信格式有关。浏览器向服务器发送 Cookie 的时候，Cookie字段是使用一行将所有 Cookie 全部发送；服务器向浏览器设置 Cookie 的时候，Set-Cookie字段是一行设置一个 Cookie。

写入 Cookie 的时候，可以一起写入 Cookie 的属性。

document.cookie = "foo=bar; expires=Fri, 31 Dec 2020 23:59:59 GMT";

上面代码中，写入 Cookie 的时候，同时设置了expires属性。属性值的等号两边，也是不能有空格的。

# Web Storage

## 概述

Storage 接口用于脚本在浏览器保存数据。两个对象部署了这个接口：window.sessionStorage和window.localStorage。

sessionStorage保存的数据用于浏览器的一次会话（session），当会话结束（通常是窗口关闭），数据被清空；localStorage保存的数据长期存在，下一次访问该网站的时候，网页可以直接读取以前保存的数据。除了保存期限的长短不同，这两个对象的其他方面都一致。

保存的数据都以“键值对”的形式存在。也就是说，每一项数据都有一个键名和对应的值。所有的数据都是以文本格式保存。

## 属性和方法

Storage 接口只有一个属性。

* Storage.length：返回保存的数据项个数。

window.localStorage.setItem('foo','a');

window.localStorage.setItem('bar','b');

window.localStorage.setItem('baz', 'c');

window.localStorage.length // 3

该接口提供5个方法。

### Storage.setItem()

Storage.setItem()方法用于存入数据。它接受两个参数，第一个是键名，第二个是保存的数据。如果键名已经存在，该方法会更新已有的键值。该方法没有返回值。

window.sessionStorage.setItem('key','value');

window.localStorage.setItem('key', 'value');

注意，Storage.setItem()两个参数都是字符串。如果不是字符串，会自动转成字符串，再存入浏览器。

window.sessionStorage.setItem(3,{ foo:1 });

window.sessionStorage.getItem('3') // "[object Object]"

上面代码中，setItem方法的两个参数都不是字符串，但是存入的值都是字符串。

如果储存空间已满，该方法会抛错。

写入不一定要用这个方法，直接赋值也是可以的。

// 下面三种写法等价

window.localStorage.foo = '123';

window.localStorage['foo'] = '123';

window.localStorage.setItem('foo', '123');

### Storage.getItem()

Storage.getItem()方法用于读取数据。它只有一个参数，就是键名。如果键名不存在，该方法返回null。

window.sessionStorage.getItem('key')

window.localStorage.getItem('key')

键名应该是一个字符串，否则会被自动转为字符串。

### Storage.removeItem()

Storage.removeItem()方法用于清除某个键名对应的键值。它接受键名作为参数，如果键名不存在，该方法不会做任何事情。

sessionStorage.removeItem('key');

localStorage.removeItem('key');

### Storage.clear()

Storage.clear()方法用于清除所有保存的数据。该方法的返回值是undefined。

window.sessionStorage.clear()

window.localStorage.clear()