

# Location 对象, URL 对象, URLSearchParams 对象

URL 是互联网的基础设施之一。浏览器提供了一些原生对象, 用来管理 URL。

## 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。如果端口不是协议默认的`80`和`433`, 则还会包括冒号 (:) 和端口。
- `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。

```
```javascript
// 当前网址为
// http://user:passwd@www.example.com:4097/path/a.html?x=111#part1
document.location.href
// "http://user:passwd@www.example.com:4097/path/a.html?x=111#part1"
document.location.protocol
// "http:"
document.location.host
// "www.example.com:4097"
document.location.hostname
// "www.example.com"
document.location.port
// "4097"
document.location.pathname
// "/path/a.html"
document.location.search
// "?x=111"
document.location.hash
// "#part1"
document.location.username
// "user"
document.location.password
```

```
// "passwd"  
document.location.origin  
// "http://user:passwd@www.example.com:4097"  
````
```

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。

```
````javascript  
// 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。

```
````javascript  
document.location.href = '#top';  
// 等同于  
document.location.hash = '#top';  
````
```

直接改写`location`，相当于写入`href`属性。

```
````javascript  
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（比如子窗口与父窗口）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

#### \*\* (1) Location.assign() \*\*

`assign`方法接受一个 URL 字符串作为参数，使得浏览器立刻跳转到新的 URL。如果参数不是有效的 URL 字符串，则会报错。

```
````javascript  
// 跳转到新的网址  
document.location.assign('http://www.example.com')  
````
```

#### \*\* (2) Location.replace() \*\*

``replace``方法接受一个 URL 字符串作为参数，使得浏览器立刻跳转到新的 URL。如果参数不是有效的 URL 字符串，则会报错。

它与``assign``方法的差异在于，``replace``会在浏览器的浏览历史``History``里面删除当前网址，也就是说，一旦使用了该方法，后退按钮就无法回到当前网页了，相当于在浏览历史里面，使用新的 URL 替换了老的 URL。它的一个应用是，当脚本发现当前是移动设备时，就立刻跳转到移动版网页。

```
```javascript
// 跳转到新的网址
document.location.replace('http://www.example.com')
```
```

### **\*\* (3) 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。

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

### **\*\* (4) Location.toString() \*\***

``toString``方法返回整个 URL 字符串，相当于读取``Location.href``属性。

## **## 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。

```
- `encodeURIComponent`  
- `encodeURIComponent`  
- `decodeURI`  
- `decodeURIComponent`
```

### encodeURIComponent()

`encodeURIComponent`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。

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

### encodeURIComponent()

`encodeURIComponent`方法用于转码 URL 的组成部分，会转码除了语义字符之外的所有字符，即元字符也会被转码。所以，它不能用于转码整个 URL。它接受一个参数，就是 URL 的片段。

```
```javascript  
encodeURIComponent('春节')  
// "%E6%98%A5%E8%8A%82"  
encodeURIComponent('http://www.example.com/q=春节')  
// "http%3A%2F%2Fwww.example.com%2Fq%3D%E6%98%A5%E8%8A%82"  
```
```

上面代码中，`encodeURIComponent`会连 URL 元字符一起转义，所以如果转码整个 URL 就会出错。

### decodeURI()

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

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

### ### decodeURIComponent()

`decodeURIComponent()`用于URL 片段的解码。它是`encodeURIComponent()`方法的逆运算。它接受一个参数，就是转码后的 URL 片段。

```
```javascript
decodeURIComponent('%E6%98%A5%E8%8A%82')
// "春节"
```
```

### ## URL 对象

`URL`对象是浏览器的原生对象，可以用来构造、解析和编码 URL。一般情况下，通过`window.URL`可以拿到这个对象。

`<a>`元素和`<area>`元素都部署了这个接口。这就是说，它们的 DOM 节点对象可以使用 URL 的实例属性和方法。

```
```javascript
var a = document.createElement('a');
a.href = 'http://example.com/?foo=1';

a.hostname // "example.com"
a.search // "?foo=1"
```
```

上面代码中，`a`是`<a>`元素的 DOM 节点对象。可以在这个对象上使用 URL 的实例属性，比如`hostname`和`search`。

### ### 构造函数

`URL`对象本身是一个构造函数，可以生成 URL 实例。

它接受一个表示 URL 的字符串作为参数。如果参数不是合法的 URL，会报错。

```
```javascript
var url = new URL('http://www.example.com/index.html');
url.href
// "http://www.example.com/index.html"
```
```

如果参数是另一个 URL 实例，构造函数会自动读取该实例的`href`属性，作为实际参数。

如果 URL 字符串是一个相对路径，那么需要表示绝对路径的第二个参数，作为计算基准。

```
```javascript
var url1 = new URL('index.html', 'http://example.com');
url1.href
```

```
// "http://example.com/index.html"

var url2 = new URL('page2.html', 'http://example.com/page1.html');
url2.href
// "http://example.com/page2.html"

var url3 = new URL('.', 'http://example.com/a/b.html')
url3.href
// "http://example.com/"
```

```

上面代码中，返回的 URL 实例的路径都是在第二个参数的基础上，切换到第一个参数得到的。最后一个例子里面，第一个参数是`..`，表示上层路径。

### ### 实例属性

URL 实例的属性与`Location`对象的属性基本一致，返回当前 URL 的信息。

- URL.href: 返回整个 URL
- URL.protocol: 返回协议，以冒号`:`结尾
- URL.hostname: 返回域名
- URL.host: 返回域名与端口，包含`:`号，默认的80和443端口会省略
- URL.port: 返回端口
- URL.origin: 返回协议、域名和端口
- URL.pathname: 返回路径，以斜杠`/`开头
- URL.search: 返回查询字符串，以问号`?`开头
- URL.searchParams: 返回一个`URLSearchParams`实例，该属性是`Location`对象没有的
- URL.hash: 返回片段识别符，以井号`#`开头
- URL.password: 返回域名前面的密码
- URL.username: 返回域名前面的用户名

```
```javascript
var url = new URL('http://user:passwd@www.example.com:4097/path/a.html?x=111#part1');

url.href
// "http://user:passwd@www.example.com:4097/path/a.html?x=111#part1"
url.protocol
// "http:"
url.hostname
// "www.example.com"
url.host
// "www.example.com:4097"
url.port
// "4097"
url.origin
// "http://www.example.com:4097"
url.pathname
// "/path/a.html"
```

```

```

url.search
// "?x=111"
url.searchParams
// URLSearchParams {}
url.hash
// "#part1"
url.password
// "passwd"
url.username
// "user"
```

```

这些属性里面，只有`origin`属性是只读的，其他属性都可写。

```

```javascript
var url = new URL('http://example.com/index.html#part1');

url.pathname = 'index2.html';
url.href // "http://example.com/index2.html#part1"

url.hash = '#part2';
url.href // "http://example.com/index2.html#part2"
```

```

上面代码中，改变 URL 实例的`pathname`属性和`hash`属性，都会实时反映在 URL 实例当中。

### ### 静态方法

**\*\* (1) URL.createObjectURL() \*\***

`URL.createObjectURL`方法用来为上传/下载的文件、流媒体文件生成一个 URL 字符串。这个字符串代表了`File`对象或`Blob`对象的 URL。

```

```javascript
// HTML 代码如下
// <div id="display"/>
// <input
//   type="file"
//   id="fileElem"
//   multiple
//   accept="image/*"
//   onchange="handleFiles(this.files)"
// >
var div = document.getElementById('display');

function handleFiles(files) {
  for (var i = 0; i < files.length; i++) {
    var img = document.createElement('img');
    img.src = window.URL.createObjectURL(files[i]);
    div.appendChild(img);
  }
}
```

```

```
}  
...
```

上面代码中，`URL.createObjectURL`方法用来为上传的文件生成一个 URL 字符串，作为`<img>`元素的图片来源。

该方法生成的 URL 就像下面的样子。

```
```javascript  
blob:http://localhost/c745ef73-ece9-46da-8f66-ebes574789b1  
```
```

注意，每次使用`URL.createObjectURL`方法，都会在内存里面生成一个 URL 实例。如果不再需要该方法生成的 URL 字符串，为了节省内存，可以使用`URL.revokeObjectURL()`方法释放这个实例。

**\*\* (2) URL.revokeObjectURL() \*\***

`URL.revokeObjectURL`方法用来释放`URL.createObjectURL`方法生成的 URL 实例。它的参数就是`URL.createObjectURL`方法返回的 URL 字符串。

下面为上一段的示例加上`URL.revokeObjectURL()`。

```
```javascript  
var div = document.getElementById('display');  
  
function handleFiles(files) {  
  for (var i = 0; i < files.length; i++) {  
    var img = document.createElement('img');  
    img.src = window.URL.createObjectURL(files[i]);  
    div.appendChild(img);  
    img.onload = function() {  
      window.URL.revokeObjectURL(this.src);  
    }  
  }  
}  
}
```

上面代码中，一旦图片加载成功以后，为本地文件生成的 URL 字符串就没用了，于是可以在`img.onload`回调函数里面，通过`URL.revokeObjectURL`方法卸载这个 URL 实例。

**## URLSearchParams 对象**

**### 概述**

`URLSearchParams`对象是浏览器的原生对象，用来构造、解析和处理 URL 的查询字符串（即 URL 问号后面的部分）。



它本身也是一个构造函数，可以生成实例。参数可以为查询字符串，起首的问号`?`有没有都行，也可以是对应查询字符串的数组或对象。

```
```javascript
// 方法一：传入字符串
var params = new URLSearchParams('?foo=1&bar=2');
// 等同于
var params = new URLSearchParams(document.location.search);

// 方法二：传入数组
var params = new URLSearchParams([[ 'foo', 1 ], [ 'bar', 2 ]]);

// 方法三：传入对象
var params = new URLSearchParams({ 'foo' : 1 , 'bar' : 2 });
```
```

`URLSearchParams`会对查询字符串自动编码。

```
```javascript
var params = new URLSearchParams({ 'foo': '你好' });
params.toString() // "foo=%E4%BD%A0%E5%A5%BD"
```
```

上面代码中，`foo`的值是汉字，`URLSearchParams`对其自动进行 URL 编码。

浏览器向服务器发送表单数据时，可以直接使用`URLSearchParams`实例作为表单数据。

```
```javascript
const params = new URLSearchParams({foo: 1, bar: 2});
fetch('https://example.com/api', {
  method: 'POST',
  body: params
}).then(...)
```
```

上面代码中，`fetch`命令向服务器发送命令时，可以直接使用`URLSearchParams`实例。

`URLSearchParams`可以与`URL`接口结合使用。

```
```javascript
var url = new URL(window.location);
var foo = url.searchParams.get('foo') || 'somedefault';
```
```

上面代码中，URL 实例的`searchParams`属性就是一个`URLSearchParams`实例，所以可以使用`URLSearchParams`接口的`get`方法。

DOM 的`a`元素节点的`searchParams`属性，就是一个`URLSearchParams`实例。

```

```javascript
var a = document.createElement('a');
a.href = 'https://example.com?filter=api';
a.searchParams.get('filter') // "api"
```

```

`URLSearchParams` 实例有遍历器接口，可以用 `for...of` 循环遍历（详见《ES6 标准入门》的《Iterator》一章）。

```

```javascript
var params = new URLSearchParams({'foo': 1, 'bar': 2});

for (var p of params) {
  console.log(p[0] + ': ' + p[1]);
}
// foo: 1
// bar: 2
```

```

`URLSearchParams` 没有实例属性，只有实例方法。

### URLSearchParams.toString()

`toString` 方法返回实例的字符串形式。

```

```javascript
var url = new URL('https://example.com?foo=1&bar=2');
var params = new URLSearchParams(url.search);

params.toString() // "foo=1&bar=2"
```

```

那么需要字符串的场合，会自动调用 `toString` 方法。

```

```javascript
var params = new URLSearchParams({'version': 2.0});
window.location.href = location.pathname + '?' + params;
```

```

上面代码中，`location.href` 赋值时，可以直接使用 `params` 对象。这时就会自动调用 `toString` 方法。

### URLSearchParams.append()

`append` 方法用来追加一个查询参数。它接受两个参数，第一个为键名，第二个为键值，没有返回值。

```

```javascript
var params = new URLSearchParams({'foo': 1, 'bar': 2});
```

```

```
params.append('baz', 3);
params.toString() // "foo=1&bar=2&baz=3"
```

`append`方法不会识别是否键名已经存在。

```
```javascript
var params = new URLSearchParams({'foo': 1, 'bar': 2});
params.append('foo', 3);
params.toString() // "foo=1&bar=2&foo=3"
```
```

上面代码中，查询字符串里面`foo`已经存在了，但是`append`依然会追加一个同名键。

### URLSearchParams.delete()

`delete`方法用来删除指定的查询参数。它接受键名作为参数。

```
```javascript
var params = new URLSearchParams({'foo': 1, 'bar': 2});
params.delete('bar');
params.toString() // "foo=1"
```
```

### URLSearchParams.has()

`has`方法返回一个布尔值，表示查询字符串是否包含指定的键名。

```
```javascript
var params = new URLSearchParams({'foo': 1, 'bar': 2});
params.has('bar') // true
params.has('baz') // false
```
```

### URLSearchParams.set()

`set`方法用来设置查询字符串的键值。

它接受两个参数，第一个是键名，第二个是键值。如果是已经存在的键，键值会被改写，否则会被追加。

```
```javascript
var params = new URLSearchParams('?foo=1');
params.set('foo', 2);
params.toString() // "foo=2"
params.set('bar', 3);
params.toString() // "foo=2&bar=3"
```
```

上面代码中，`foo`是已经存在的键，`bar`是还不存在的键。

如果有多名的同名键，`set`会移除现存所有的键。

```
```javascript
var params = new URLSearchParams('?foo=1&foo=2');
params.set('foo', 3);
params.toString() // "foo=3"
```
```

下面是一个替换当前 URL 的例子。

```
```javascript
// URL: https://example.com?version=1.0
var params = new URLSearchParams(location.search.slice(1));
params.set('version', 2.0);

window.history.replaceState({}, '', location.pathname + `?` + params);
// URL: https://example.com?version=2.0
```
```

### URLSearchParams.get(), URLSearchParams.getAll()

`get`方法用来读取查询字符串里面的指定键。它接受键名作为参数。

```
```javascript
var params = new URLSearchParams('?foo=1');
params.get('foo') // "1"
params.get('bar') // null
```
```

两个地方需要注意。第一，它返回的是字符串，如果原始值是数值，需要转一下类型；第二，如果指定的键名不存在，返回值是`null`。

如果有多名的同名键，`get`返回位置最前面的那个键值。

```
```javascript
var params = new URLSearchParams('?foo=3&foo=2&foo=1');
params.get('foo') // "3"
```
```

上面代码中，查询字符串有三个`foo`键，`get`方法返回最前面的键值`3`。

`getAll`方法返回一个数组，成员是指定键的所有键值。它接受键名作为参数。

```
```javascript
var params = new URLSearchParams('?foo=1&foo=2');
params.getAll('foo') // ["1", "2"]
```
```

上面代码中，查询字符串有两个`foo`键，`getAll`返回的数组就有两个成员。

```
### URLSearchParams.sort()
```

`sort`方法对查询字符串里面的键进行排序，规则是按照 Unicode 码点从小到大排列。

该方法没有返回值，或者说返回值是`undefined`。

```
```javascript
var params = new URLSearchParams('c=4&a=2&b=3&a=1');
params.sort();
params.toString() // "a=2&a=1&b=3&c=4"
```
```

上面代码中，如果有两个同名的键`a`，它们之间不会排序，而是保留原始的顺序。

```
### URLSearchParams.keys(), URLSearchParams.values(), URLSearchParams.entries()
```

这三个方法都返回一个遍历器对象，供`for...of`循环遍历。它们的区别在于，`keys`方法返回的是键名的遍历器，`values`方法返回的是键值的遍历器，`entries`返回的是键值对的遍历器。

```
```javascript
var params = new URLSearchParams('a=1&b=2');

for(var p of params.keys()) {
  console.log(p);
}
// a
// b

for(var p of params.values()) {
  console.log(p);
}
// 1
// 2

for(var p of params.entries()) {
  console.log(p);
}
// ["a", "1"]
// ["b", "2"]
```
```

如果直接对`URLSearchParams`进行遍历，其实内部调用的就是`entries`接口。

```
```javascript
for (var p of params) {}
// 等同于
for (var p of params.entries()) {}
```
```

## 参考链接

- [Location](<https://developer.mozilla.org/en-US/docs/Web/API/Location>), by MDN
- [URL](<https://developer.mozilla.org/en-US/docs/Web/API/URL>), by MDN
- [URLSearchParams](<https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams>), by MDN
- [Easy URL Manipulation with URLSearchParams](<https://developers.google.com/web/updates/2016/01/urlsearchparams?hl=en>), by Eric Bidelman