History 对象

1. 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 4 个网址。

```
window.history.length; // 4
```

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。

```
// 后退到上一个网址
history.back();

// 等同于
history.go(-1);
```

```
// 前进到下一个网址
history.forward();

// 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。

2. 属性

`History` 对象主要有两个属性。

- `History.length`：当前窗口访问过的网址数量（包括当前网页）
- `History.state`：`History` 堆栈最上层的状态值（详见下文）

```
// 当前窗口访问过多少个网页
window.history.length; // 1
```

```
// History 对象的当前状态
// 通常是 undefined，即未设置
window.history.state; // undefined
```

3. 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()`：接受一个整数或 `0` 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。
```

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。

3.1. History.pushState()

History.pushState() 方法用于在历史中添加一条记录。

`window.history.pushState(state, title, url)` 该方法接受三个参数，依次为：

- `state`：一个与添加的记录相关联的状态对象，主要用于 `popstate` 事件。该事件触发时，该对象会传入回调函数。也就是说，浏览器会将这个对象序列化以后保留在本地，重新载入这个页面的时候，可以拿到这个对象。如果不需要这个对象，此处可以填 `null`。
- `title`：新页面的标题。但是，现在所有浏览器都忽视这个参数，所以这里可以填空字符串。
- `url`：新的网址，必须与当前页面处在同一个域。浏览器的地址栏将显示这个网址。假定当前网址是 `example.com/1.html`，使用 `pushState()` 方法在浏览记录（`History` 对象）中添加一个新记录。

```
let stateObj = { foo: 'bar' };
history.pushState(stateObj, 'page 2', '2.html');
```

添加新记录后，浏览器地址栏立刻显示 `example.com/2.html`，但并不会跳转到 `2.html`，甚至也不会检查 `2.html` 是否存在，它只是成为浏览历史中的最新记录。这时，在地址栏输入一个新的地址(比如访问 `google.com`)，然后点击了倒退按钮，页面的 `URL` 将显示 `2.html`；你再点击一次倒退按钮，URL 将显示 `1.html`。

总之，`pushState()` 方法不会触发页面刷新，只是导致 `History` 对象发生变化，地址栏会有反应。

使用该方法之后，就可以用 `History.state` 属性读出状态对象。

```
let stateObj = { foo: 'bar' };
history.pushState(stateObj, 'page 2', '2.html');
```

```
history.state; // {foo: "bar"}
```

如果 pushState 的 URL 参数设置了一个新的锚点值（即 hash），并不会触发 hashchange 事件。反过来，如果 URL 的锚点值变了，则会在 History 对象创建一条浏览记录。

如果 pushState() 方法设置了一个跨域网址，则会报错。

```
// 报错
// 当前网址为 http://example.com
history.pushState(null, '', 'https://twitter.com/hello');
```

上面代码中，pushState 想要插入一个跨域的网址，导致报错。这样设计的目的是，防止恶意代码让用户以为他们是在另一个网站上，因为这个方法不会导致页面跳转。

3.2. History.replaceState()

History.replaceState() 方法用来修改 History 对象的当前记录，其他都与 pushState() 方法一模一样。

假定当前网页是 https://wangdoc.com/javascript/bom/history.html。

```
history.pushState({ page: 1 }, 'title 1', '?page=1');
// URL 显示为 `https://wangdoc.com/javascript/bom/history.html?page=1`

history.pushState({ page: 2 }, 'title 2', '?page=2');
// URL 显示为 `https://wangdoc.com/javascript/bom/history.html?page=2`

history.replaceState({ page: 3 }, 'title 3', '?page=3');
// URL 显示为 `https://wangdoc.com/javascript/bom/history.html?page=3`

history.back();
// URL 显示为 `https://wangdoc.com/javascript/bom/history.html?page=2`

history.back();
// URL 显示为 `https://wangdoc.com/javascript/bom/history.html?page=1`

history.go(2);
// URL 显示为 `https://wangdoc.com/javascript/bom/history.html?page=3`
```

3.3. popstate 事件

每当同一个文档的浏览历史（即 history 对象）出现变化时，就会触发 popstate 事件。

注意，仅仅调用 pushState() 方法或 replaceState() 方法 ，并不会触发该事件，只有用户点击浏览器倒退按钮和前进按钮，或者使用 JavaScript 调用 History.back()、History.forward()、History.go()方法时才会触发。另外，该事件只针对同一个文档，如果浏览历史的切换，导致加载不同的文档，该事件也不会触发。

使用的时候，可以为 popstate 事件指定回调函数。

```
window.onpopstate = function (event) {
  console.log('location: ' + document.location);
  console.log('state: ' + JSON.stringify(event.state));
};

// 或者
window.addEventListener('popstate', function (event) {
  console.log('location: ' + document.location);
  console.log('state: ' + JSON.stringify(event.state));
});
```

回调函数的参数是一个 `event` 事件对象，它的 `state` 属性指向 `pushState` 和 `replaceState` 方法为当前 `URL` 所提供的状态对象（即这两个方法的第一个参数）。上面代码中的 `event.state`，就是通过 `pushState` 和 `replaceState` 方法，为当前 `URL` 绑定的 `state` 对象。

这个 `state` 对象也可以直接通过 `history` 对象读取。

```
let currentState = history.state;
```

注意，页面第一次加载的时候，浏览器不会触发 `popstate` 事件。