Storage 接口

1. 概述

`Storage` 接口用于脚本在浏览器保存数据。这个对象部署了两个接口：`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。

> 1. 当使用 `<a target="_blank" />`, `window.location.href`，`window.open` 打开新的页面时，新页面会复制之前同地址页面的 `sessionStorage`。
> 2. 尽管两个页面 `sessionStorage` 相同，但它们是相互独立的，不会改变对方。

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。

这个接口很像 `Cookie` 的强化版，能够使用大得多的存储空间。目前，每个域名的存储上限视浏览器而定，`Chrome` 是 `2.5MB`，`Firefox` 和 `Opera` 是 `5MB`，`IE` 是 `10MB`。其中，`Firefox` 的存储空间由一级域名决定，而其他浏览器没有这个限制。也就是说，`Firefox` 中，`a.example.com` 和 `b.example.com` 共享 `5MB` 的存储空间。另外，与 `Cookie` 一样，它们也受同域限制。某个网页存入的数据，只有同域下的网页才能读取，如果跨域操作会报错。

2. 属性

`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
```

3. 方法

3.1. 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');
```

3.2. 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。

3.3. 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');
```

3.4. 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();
```

3.5. Storage.key()

`Storage.key()` 方法接受一个整数作为参数（从零开始），返回该位置对应的键名。

```
window.sessionStorage.setItem('key', 'value');
window.sessionStorage.key(0); // "key"
```

结合使用 `Storage.length` 属性和 `Storage.key()` 方法，可以遍历所有的键。

```
for (var i = 0; i < window.localStorage.length; i++) {
  console.log(localStorage.key(i));
}
```

4. storage 事件

`Storage` 接口储存的数据发生变化时，会触发 `storage` 事件，可以指定这个事件的监听函数。

```
window.addEventListener('storage', onStorageChange);
```

监听函数接受一个 `event` 实例对象作为参数。这个实例对象继承了 `StorageEvent` 接口，有几个特有的属性，都是只读属性。

- `StorageEvent.key`：字符串，表示发生变动的键名。如果 `storage` 事件是由 `clear()` 方法引起，该属性返回 `null`。
- `StorageEvent.newValue`：字符串，表示新的键值。如果 `storage` 事件是由 `clear()` 方法或删除该键值对引发的，该属性返回 `null`。
- `StorageEvent.oldValue`：字符串，表示旧的键值。如果该键值对是新增的，该属性返回 `null`。
- `StorageEvent.storageArea`：对象，返回键值对所在的整个对象。也说是说，可以从这个属性上面拿到当前域名储存的所有键值对。
- `StorageEvent.url`：字符串，表示原始触发 `storage` 事件的那个网页的网址。

下面是 StorageEvent.key 属性的例子。

```
function onStorageChange(e) {
  console.log('key', e.key);
  console.log('storageArea', event.storageArea);
}

window.addEventListener('storage', onStorageChange);
```

注意，该事件有一个很特别的地方，就是它不在导致数据变化的当前页面触发，而是在同一个域名的其他窗口触发。也就是说，如果浏览器只打开一个窗口，可能观察不到这个事件。比如同时打开多个窗口，当其中的一个窗口导致储存的数据发生改变时，只有在其他窗口才能观察到监听函数的执行。可以通过这种机制，实现多个窗口之间的通信

在一个窗口中添加 `storage`

```
localStorage.setItem('c', 'ccc333');
```

在另一个同域名窗口中查看控制台：

```
key c
storageArea Storage {b: "bbb222", c: "ccc333", a: "aaa111", length: 3}
```