浏览器存储

三种本地存储方式和一些扩展

cookie

前言

网络早期最大的问题之一是如何管理状态。简而言之，服务器无法知道两个请求是否来自同一个浏览器。当时最简单的方法是在请求时，在页面中插入一些参数，并在下一个请求中传回参数。这需要使用包含参数的隐藏的表单，或者作为 URL 参数的一部分传递。这两个解决方案都手动操作，容易出错。cookie 出现来解决这个问题。

作用

cookie 是纯文本，没有可执行代码。存储数据，当用户访问了某个网站（网页）的时候，我们就可以通过 cookie 来向访问者电脑上存储数据，或者某些网站为了辨别用户身份、进行 session 跟踪而储存在用户本地终端上的数据（通常经过加密）

如何工作

当网页要发 http 请求时，浏览器会先检查是否有相应的 cookie，有则自动添加在 request header 中的 cookie 字段中。这些是浏览器自动帮我们做的，而且每一次 http 请求浏览器都会自动帮我们做。这个特点很重要，因为这关系到“什么样的数据适合存储在 cookie 中”。

存储在 cookie 中的数据，每次都会被浏览器自动放在 http 请求中，如果这些数据并不是每个请求都需要发给服务端的数据，浏览器这设置自动处理无疑增加了网络开销；但如果这些数据是每个请求都需要发给服务端的数据（比如身份认证信息），浏览器这设置自动处理就大大免去了重复添加操作。所以对于那种设置“每次请求都要携带的信息（最典型的就是身份认证信息）”就特别适合放在 cookie 中，其他类型的数据就不适合了。

特征

1. 不同的浏览器存放的 cookie 位置不一样，也是不能通用的。
2. cookie 的存储是以域名形式进行区分的，不同的域下存储的 cookie 是独立的。

3. 我们可以设置 cookie 生效的域（当前设置 cookie 所在域的子域），也就是说，我们能够操作的 cookie 是当前域以及当前域下的所有子域
4. 一个域名下存放的 cookie 的个数是有限制的，不同的浏览器存放的个数不一样,一般为 20 个。
5. 每个 cookie 存放的内容大小也是有限制的，不同的浏览器存放大小不一样，一般为 4KB。
6. cookie 也可以设置过期的时间，默认是会话结束的时候，当时间到期自动销毁

cookie 值既可以设置，也可以读取。

设置

客户端设置

注意： 客户端可以设置 cookie 的下列选项：expires、domain、path、secure（有条件：只有在 https 协议的网页中，客户端设置 secure 类型的 cookie 才能成功），但无法设置 HttpOnly 选项。

服务器端设置

不管你是请求一个资源文件（如 html/js/css/图片），还是发送一个 ajax 请求，服务端都会返回 response。而 response header 中有一项叫 set-cookie，是服务端专门用来设置 cookie 的。

注意： 一个 set-Cookie 字段只能设置一个 cookie，当你要想设置多个 cookie，需要添加同样多的 set-Cookie 字段。
服务端可以设置 cookie 的所有选项：expires、domain、path、secure、HttpOnly
通过 Set-Cookie 指定的这些可选项只会在浏览器端使用，而不会被发送至服务器端。

读取

我们通过 document.cookie 来获取当前网站下的 cookie 的时候，得到的字符串形式的值，它包含了当前网站下所有的 cookie（为避免跨域脚本(xss)攻击，这个方法只能获取非 HttpOnly 类型的 cookie）。它会把所有的 cookie 通过一个分号+空格的形式串联起来，例如 `username=chenfangxu; job=coding`

修改 cookie

要想修改一个 cookie，只需要重新赋值就行，旧的值会被新的值覆盖。但要注意一点，在设置新 cookie 时，path/domain 这几个选项一定要旧 cookie 保持一样。否则不会修改旧值，而是添加了一个新的 cookie。

删除

把要删除的 cookie 的过期时间设置成已过去的时间,path/domain/这几个选项一定要旧 cookie 保持一样。

注意

如果只设置一个值，那么算 cookie 中的 value; 设置的两个 cookie,key 值如果设置的相同，下面的也会把上面的覆盖。

cookie 的属性（可选项）

过期时间

如果我们想长时间存放一个 cookie。需要在设置这个 cookie 的时候同时给他设置一个过期的时间。如果不设置，cookie 默认是临时存储的，当浏览器关闭进程的时候自动销毁

一般设置天数：new Date().setDate(oDate.getDate() + 5); 比当前时间多 5 天

一个设置 cookie 时效性的例子

> expires 是 http/1.0 协议中的选项，在新的 http/1.1 协议中 expires 已经由 max-age 选项代替，两者的作用都是限制 cookie 的有效时间。expires 的值是一个时间点（cookie 失效时刻= expires），而 max-age 的值是一个以秒为单位时间段（cookie 失效时刻= 创建时刻+ max-age）。
> 另外，max-age 的默认值是 -1(即有效期为 session)；max-age 有三种可能值：负数、0、正数。
> 负数：有效期 session；
> 0：删除 cookie；
> 正数：有效期为创建时刻+ max-age

cookie 的域概念（domain 选项）

domain 指定了 cookie 将要被发送至哪个或哪些域中。默认情况下，domain 会被设置为创建该 cookie 的页面所在的域名，所以当给相同域名发送请求时该 cookie 会被发送至服务器。

浏览器会把 domain 的值与请求的域名做一个尾部比较（即从字符串的尾部开始比较），并将匹配的 cookie 发送至服务器。

客户端设置

`document.cookie = "username=cfangxu;path=/;domain=qq.com"`

如上：“www.qq.com" 与 "sports.qq.com" 公用一个关联的域名"qq.com"，我们如果想让

"sports.qq.com" 下的 cookie 被 "www.qq.com" 访问，我们就需要用到 cookie 的 domain 属性，并且需要把 path 属性设置为 "/"。

服务端设置

```
Set-Cookie: username=cfangxu;path=/;domain=qq.com
```

注：一定的是同域之间的访问，不能把 domain 的值设置成非主域的域名。

cookie 的路径概念（path 选项）

cookie 一般都是由于用户访问页面而被创建的，可是并不是只有在创建 cookie 的页面才可以访问这个 cookie。
因为安全方面的考虑,默认情况下，只有与创建 cookie 的页面在同一个目录或子目录下的网页才可以访问。 即 path 属性可以为服务器特定文档指定 cookie，这个属性设置的 url 且带有这个前缀的 url 路径都是有效的。

客户端设置

最常用的例子就是让 cookie 在根目录下,这样不管是哪个子页面创建的 cookie，所有的页面都可以访问到了。

```
document.cookie = "username=cfangxu; path=/"
```

服务端设置

```
Set-Cookie:name=cfangxu; path=/blog
```

如上设置：path 选项值会与 /blog，/blogrool 等等相匹配；任何以 /blog 开头的选项都是合法的。需要注意的是，只有在 domain 选项核实完毕之后才会对 path 属性进行比较。path 属性的默认值是发送 Set-Cookie 消息头所对应的 URL 中的 path 部分。

domain 和 path 总结：

domain 是域名，path 是路径，两者加起来就构成了 URL，domain 和 path 一起来限制 cookie 能被哪些 URL 访问。
所以 domain 和 path2 个选项共同决定了 cookie 何时被浏览器自动添加到请求头部中发送出去。如果没有设置这两个选项，则会使用默认值。domain 的默认值为设置该 cookie 的网页所在的域名，path 默认值为设置该 cookie 的网页所在的目录。

cookie 的安全性（secure 选项）

通常 cookie 信息都是使用 HTTP 连接传递数据，这种传递方式很容易被查看，所以 cookie 存储的信息容易被窃取。假如 cookie 中所传递的内容比较重要，那么就要求使用加密的数据传输。

secure 选项用来设置 cookie 只在确保安全的请求中才会发送。当请求是 HTTPS 或者其他安全协议时，包含 secure 选项的 cookie 才能被发送至服务器。

```
document.cookie = "username=cfangxu; secure"
```

把 cookie 设置为 secure，只保证 cookie 与服务器之间的数据传输过程加密，而保存在本地的 cookie 文件并不加密。就算设置了 secure 属性也并不代表他人不能看到你机器本地保存的 cookie 信息。机密且敏感的信息绝不应该在 cookie 中存储或传输，因为 cookie 的整个机制原本都是不安全的

注意：如果想在客户端即网页中通过 js 去设置 secure 类型的 cookie，必须保证网页是 https 协议的。在 http 协议的网页中是无法设置 secure 类型 cookie 的。

httpOnly

这个选项用来设置 cookie 是否能通过 js 去访问。默认情况下，cookie 不会带 httpOnly 选项(即为空)，所以默认情况下，客户端是可以通过 js 代码去访问（包括读取、修改、删除等）这个 cookie 的。当 cookie 带 httpOnly 选项时，客户端则无法通过 js 代码去访问（包括读取、修改、删除等）这个 cookie。

在客户端是不能通过 js 代码去设置一个 httpOnly 类型的 cookie 的，这种类型的 cookie 只能通过服务端来设置。

cookie 的编码

cookie 其实是个字符串，但这个字符串中等号、分号、空格被当做了特殊符号。所以当 cookie 的 key 和 value 中含有这 3 个特殊字符时，需要对其进行额外编码，一般会用 escape 进行编码，读取时用 unescape 进行解码；当然也可以用 encodeURIComponent/decodeURIComponent 或者 encodeURI/decodeURI

第三方 cookie

通常 cookie 的域和浏览器地址的域匹配，这被称为第一方 cookie。那么第三方 cookie 就是 cookie 的域和地址栏中的域不匹配，这种 cookie 通常被用在第三方广告网站。为了跟踪用户的浏览记录，并且根据收集的用户的浏览习惯，给用户推送相关的广告。

- **cookie 推荐资源**
  - 聊一聊 cookie (opens new window)

localStorage（本地存储）

HTML5 新方法，不过**IE8 及以上**浏览器都兼容。

特点

- 生命周期：持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。
- 存储的信息在同一域中是共享的。
- 当本页操作（新增、修改、删除）了 localStorage 的时候，本页面不会触发 storage 事件,但是别的页面会触发 storage 事件。
- 大小：据说是 5M（跟浏览器厂商有关系）
- 在非 IE 下的浏览中可以本地打开。IE 浏览器要在服务器中打开。
- localStorage 本质上是对字符串的读取，如果存储内容多的话会消耗内存空间，会导致页面变卡
- localStorage 受同源策略的限制

设置

```
localStorage.setItem('username','cfangxu');
```

获取

```
localStorage.getItem('username')
```

也可以获取键名

```
localStorage.key(0) #获取第一个键名
```

删除

```
localStorage.removeItem('username')
```

也可以一次性清除所有存储

```
localStorage.clear()
```

storage 事件

当 storage 发生改变的时候触发。

注意： 当前页面对 storage 的操作会触发其他页面的 storage 事件

事件的回调函数中有一个参数 event,是一个 StorageEvent 对象，提供了一些实用的属性,如下表：

Property	Type	Description
key	String	The named key that was added, removed, or moddified
oldValue	Any	The previous value(now overwritten), or null if a new item was added
newValue	Any	The new value, or null if an item was added
url/uri	String	The page that called the method that triggered this change

---

sessionStorage

其实跟 localStorage 差不多，也是本地存储，会话本地存储

特点：

- 用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此 sessionStorage 不是一种持久化的本地存储，仅仅是会话级别的存储。也就是说只要这个浏览器窗口没有关闭，即使刷新页面或进入同源另一页面，数据仍然存在。关闭窗口后，sessionStorage 即被销毁，或者在新窗口打开同源的另一个页面，sessionStorage 也是没有的。

---

cookie、localStorage、sessionStorage 区别

- 相同：在本地（浏览器端）存储数据
- 不同：localStorage、sessionStoragelocalStorage 只要在相同的协议、相同的主机名、相同的端口下，就能读取/修改到同一份 localStorage 数据。sessionStorage 比 localStorage 更严苛一点，除了协议、主机名、端口外，还要求在同一窗口（也就是浏览器的标签页）下。localStorage 是永久存储，除非手动删除。sessionStorage 当会话结束（当前页面关闭的时候，自动销毁）cookie 的数据会在每一次发送 http 请求的时候，同时发送给服务器而 localStorage、sessionStorage 不会。

扩展其他的前端存储方式（不常用）

web SQL database

先说个会被取代的，为什么会被取代，主要有以下几个原因：

1. W3C 舍弃 `Web SQL database` 草案,而且是在 2010 年年底，规范不支持了，浏览器厂商已经支持的就支持了，没有支持的也不打算支持了，比如 IE 和 Firefox。
2. 为什么要舍弃？因为 `Web SQL database` 本质上是一个关系型数据库，后端可能熟悉，但是前端就有很多不熟悉了，虽然 SQL 的简单操作不难，但是也得需要学习。
3. SQL 熟悉后，真实操作中还得把你要存储的东西，比如对象，转成 SQL 语句，也挺麻烦的。

indexedDB

> 来自 MDN 的解释： indexedDB 是一种低级 API，用于客户端存储大量结构化数据(包括, 文件/blobs)。该 API 使用索引来实现对该数据的高性能搜索。虽然 Web Storage 对于存储较少量的数据很有用，但对于存储更大量的结构化数据来说，这种方法不太有用。IndexedDB 提供了一个解决方案。

所以， `IndexedDB` API 是强大的，但对于简单的情况可能看起来太复杂了，所以要看你的业务场景来选择到底是用还是不用。