Cookie

- **概念:** Cookie 存储的是以 ';+空格' 连接的字符串。当用户访问了某个网站（网页）的时候，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。
- **工作：** 浏览器要发 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。
- **特征：**
  1. 不同浏览器存放 Cookie 位置不同，不能通用。
  2. Cookie 的存储以域名形式区分，不同的域下存储的 Cookie 是独立的。
  3. Cookie 生效的域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. 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 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
- **设置方式：**
  - 客户端可以设置 Cookie 下列选项：expires(过期时间)、domain(服务器域名)、path(域名下的哪些路径可以接受 Cookie)、secure(需要在 HTTPS 协议下,并且只是加密传输过程,保存在本地的 Cookie 并不加密),无法设置 **HttpOnly** 选项。

    ```
    document.cookie = "username=cfangxu; domain=baike.baidu.com"; //设置了生效域
    //在设置这些属性时，属性之间由'一个分号'和'一个空格'隔开。
    //当我们需要设置多个 cookie 时
    document.cookie = "name=Jonh";
    document.cookie = "age=12";
    ```

  - 服务端，不管是请求资源文件（如 html/js/css/图片），还是发送 ajax 请求，服务端都会返回响应。而响应头中有一项叫 Set-Cookie，用于服务端设置 Cookie 。多个 Cookie，添加多个 Set-Cookie 字段。

    ```
    Set-Cookie //消息头是一个字符串，其格式如下（中括号中的部分是可选的）：
    Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]
    ```

- 读取

  客户端通过 document.cookie 获取当前网站下的 cookie (只能获取非 HttpOnly 类型的 cookie)

- 修改

  只需要重新赋值，旧的值会被新的值覆盖。需要注意，设置新 cookie 时，name/domain/path 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。

- 补充

  cookie 虽然是个字符串，但这个字符串中逗号、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 encodeURI/decodeURI。

Session

- 概念
  - session 保存在服务器
  - session 中保存的是对象
  - session 不能区分路径，同一个用户访问一个网站期间，所有的 session 在任何一个地方都可以访问
  - session 需要借助 cookie 才能正常工作，如果禁用 cookie,session 则失效(服务器使用 URL 重写)
- 应用场景
  - session 上下文机制，针对每一个用户，通过 sessionid 来区分不同客户
  - session 是以 cookie 或 url 重写为基础的，默认使用 cookie 实现，系统会创造一个名为 jsessionid 的输出 cookie
  - 重要状态走 session,不重要走 cookie,登陆信息用 session，购物车用 cookie

localStorage

HTML5 新方法，仅 IE8 及以上浏览器兼容。

localStorage 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。

特点:

- 生命周期：持久化的本地存储，除非手动删除数据，否则数据是永远不会过期(永久存储)。
- 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：一般 5M（跟浏览器有关）
- 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

sessionStorage

- 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。
- 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。
- sessionStorage 要求在 协议、主机名、端口、同一窗口下，能读取/修改到同一份 sessionStorage 数据。

storage 事件(当 storage 发生改变时触发)

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

回调函数参数: StorageEvent 对象，具有 key、oldValue、newValue、url/uri 属性

总结

Cookie、localStorage、sessionStorage 都遵循同源策略；Cookie 可以设置过期时间默认在浏览器会话结束时清除，并且 Cookie 能存储的数据量较小，同一个源下能设置的 Cookie 数量也是有限的；localStorage 是永久存储除没有过期时间，除非手动删除都将一直保存在浏览器会话中，sessionStorage 在页面被关闭时数据会被清除，这三者只要是在当前源下都能访问到数据。Cookie 通过 document.cookie 读取与设置，localStorage 通过 getItem 和 setItem 读取与设置，sessionStorage 的存取方式与 localStorage 相同；Cookie 总是存储以`;`为间隔的字符串，localStorage、sesssionStorage 存储的数据总是字符串，如果不是字符串会自动转化为字符串类型。

session 是一种存储在服务器中的对象，session 不区分路径，同一个用户访问网站期间，session 在任何一个地方都能访问。session 的存储需要借助 Cookie ，在当前源的 Cookie 中保存一个 sesssionid 作为当前会话在服务器中的唯一标识，如果浏览器中需要访问 session 中的数据需要通过 sessionid 去服务器的内存中找，如果浏览器禁用了 Cookie 只能采用 URL 重写的方式使用 session。