使用 HTTP cookie

HTTP **cookie**（网络 cookie、浏览器 cookie）是服务器发送给用户网络浏览器的一小段数据。浏览器可能会存储 cookie 并将其与以后的请求一起发送回同一服务器。通常，HTTP cookie 用于判断两个请求是否来自同一浏览器——例如，保持用户登录。它会记住无状态HTTP 协议 的有状态信息。

Cookie 主要用于三个目的：

会话管理

登录名、购物车、游戏分数或服务器应记住的任何其他内容

个性化

用户偏好、主题和其他设置

追踪

记录和分析用户行为

Cookie 曾经用于一般的客户端存储。虽然当它们是在客户端上存储数据的唯一方式时这是有道理的，但现在建议使用现代存储 API。每次请求都会发送 Cookie，因此它们会降低性能（尤其是对于移动数据连接）。用于客户端存储的现代 API 是Web Storage API（`localStorage` 和 `sessionStorage`）以及IndexedDB。

> **注意：** 要查看存储的 cookie（以及网页可以使用的其他存储），您可以在开发人员工具中启用存储检查器 并从存储树中选择 Cookie。

创建 cookie

收到 HTTP 请求后，服务器可以 `Set-Cookie` 随响应发送一个或多个标头。浏览器通常会存储 cookie 并将其与在 `Cookie` HTTP 标头内对同一服务器发出的请求一起发送。您可以指定一个到期日期或时间段，超过该日期或时间段不应发送 cookie。您还可以对特定的域和路径设置额外的限制，以限制 cookie 的发送位置。关于下面提到的header属性的详细信息，请参考 `Set-Cookie` 参考文章。

Set-Cookie 和标题 Cookie_

HTTP `Set-Cookie` 响应标头将 cookie 从服务器发送到用户代理。一个简单的cookie是这样设置的：

```
Set-Cookie: <cookie-name>=<cookie-value>
```

这指示服务器发送标头告诉客户端存储一对 cookie：

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

然后，对于服务器的每个后续请求，浏览器都会使用标头将所有先前存储的 cookie 发送回服务器 `Cookie` 。

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

> **注意：** 以下是如何 `Set-Cookie` 在各种服务器端应用程序中使用标头：
>
> - PHP

- 节点JS
- Python
- Rails 上的 Ruby

定义 cookie 的生命周期

cookie 的生命周期可以通过两种方式定义：

- 当前会话结束时，*会话cookie 将被删除*。浏览器定义“当前会话”何时结束，一些浏览器在重启时使用*会话恢复*。这可能会导致会话 cookie 无限期地持续下去。
- *永久*cookie 在属性指定的日期 `Expires` 或在属性指定的时间段后删除 `Max-Age` 。

例如：

```
Set-Cookie: id=a3fWa; Expires=Thu, 31 Oct 2021 07:28:00 GMT;
```

> **注意：** 当您设置 `Expires` 日期和时间时，它们是相对于设置 cookie 的客户端，而不是服务器。

如果您的站点对用户进行身份验证，则无论何时用户进行身份验证，它都应该重新生成并重新发送会话 cookie，即使是已经存在的 cookie。这种方法有助于防止第三方可以重复使用用户会话的会话固定攻击。

限制访问 cookie

您可以通过以下两种方式之一确保安全地发送 cookie，并且不会被意外方或脚本访问：使用属性 `Secure` 和 `HttpOnly` 属性。

具有该属性的 cookie `Secure` 仅通过 HTTPS 协议通过加密请求发送到服务器。它从不使用不安全的 HTTP 发送（本地主机除外），这意味着中间人攻击者无法轻松访问它。不安全的站点（ `http:` 在 URL 中）不能设置带有

该 Secure 属性的 cookie。但是，不要假设这 Secure 会阻止对 cookie 中敏感信息的所有访问。例如，有权访问客户端硬盘（或 JavaScript，如果 HttpOnly 未设置该属性）的人可以读取和修改信息。

HttpOnly JavaScript API 无法访问具有该属性的 cookie Document.cookie ；它只发送到服务器。例如，在服务器端会话中持续存在的 cookie 不需要对 JavaScript 可用，并且应该具有该 HttpOnly 属性。此预防措施有助于减轻跨站点脚本 (XSS) 攻击。

这是一个例子：

```
Set-Cookie: id=a3fWa; Expires=Thu, 21 Oct 2021 07:28:00 GMT;
Secure; HttpOnly
```

定义发送 cookie 的位置

和属性定义了cookie 的*范围*：应该将 cookie 发送到哪些 URL Domain 。 Path

域属性

该 Domain 属性指定哪些主机可以接收 cookie。如果服务器未指定，则浏览器将域默认为设置 cookie 的 Domain 同一主机，*不包括子域*。如果 Domain *指定*，则始终包含子域。因此，指定 Domain 比省略它限制更少。但是，当子域需要共享有关用户的信息时，它会很有帮助。

例如，如果您设置 Domain=mozilla.org ，则 cookie 可用于子域，例如 developer.mozilla.org .

路径属性

该 Path 属性指示 URL 路径，该路径必须存在于所请求的 URL 中才能发送 Cookie 标头。("/")字符 %x2F 被视为目录分隔符，子目录也匹配。

例如，如果您设置 Path=/docs ，则这些请求路径匹配：

- /docs

- `/docs/`
- `/docs/Web/`
- `/docs/Web/HTTP`

但是这些请求路径不会：

- `/`
- `/docsets`
- `/fr/docs`

SameSite 属性

该 `SameSite` 属性允许服务器指定是否/何时使用跨站点请求发送 cookie（其中Site由可注册域和*方案定义*：http 或 https）。这提供了一些针对跨站点请求伪造攻击 (CSRF) 的保护。它采用三个可能的值： `Strict` 、 `Lax` 和 `None` 。

使用 `Strict`，浏览器仅发送带有来自 cookie 源站点的请求的 cookie。 *类似，只是当用户导航* `Lax` 到 cookie 的源站点时浏览器也会发送 cookie（即使用户来自不同的站点）。例如，通过访问来自外部站点的链接。指定 cookie 在原始请求和跨站点请求上发送，但*仅在安全上下文中发送*（即，如果还必须设置该属性）。如果未设置任何属性，则将 cookie 视为.
`None` `SameSite=None` `Secure` `SameSite` `Lax`

这是一个例子：

```
Set-Cookie: mykey=myvalue; SameSite=Strict
```

> **注意：** 与最近更改相关的标准 `SameSite` （MDN 记录了上面的新行为）。有关如何在特定浏览器版本中处理该属性的信息， 请参

> - `SameSite=Lax` `SameSite` 如果未指定，则为新的默认值。以前，默认情况下会为所有请求发送 cookie。

- 带有的 Cookie `SameSite=None` 现在还必须指定 `Secure` 属性（它们需要安全上下文）。
- `http:` 如果使用不同的方案（或 `https:`）发送，来自同一域的 Cookie 将不再被视为来自同一站点。

Cookie 前缀

由于 cookie 机制的设计，服务器无法确认 cookie 是从安全来源设置的，甚至无法分辨cookie 最初设置的*位置*。

子域上的易受攻击的应用程序可以设置具有该属性的 cookie `Domain`，从而可以访问所有其他子域上的该 cookie。这种机制可以在*会话固定*攻击中被滥用。有关主要缓解方法，请参阅会话修复。

但是，作为纵深防御措施，您可以使用 *cookie 前缀*来断言有关 cookie 的特定事实。有两个前缀可用：

`__Host-`

如果 cookie 名称具有此前缀，则 `Set-Cookie` 仅当它也标有 `Secure` 属性、从安全来源发送、不*包含*属性 `Domain` 且 `Path` 属性设置为 时，才会在标头中接受它 `/`。这样，这些 cookie 可以被视为“域锁定”。

`__Secure-`

如果 cookie 名称具有此前缀，则 `Set-Cookie` 仅当它标有该 `Secure` 属性并从安全来源发送时，它才会在标头中被接受。这比 `__Host-` 前缀弱。

The browser will reject cookies with these prefixes that don't comply with their restrictions. Note that this ensures that subdomain-created cookies with prefixes are either confined to the subdomain or ignored completely. As the application server only checks for a specific cookie name when determining if the user is authenticated or a CSRF token is correct, this effectively acts as a defense measure against session fixation.

> **Note:** On the application server, the web application *must* check for the full cookie name including the prefix. User agents *do not* strip the prefix from the cookie before sending it in a request's `Cookie` header.

For more information about cookie prefixes and the current state of browser support, see the Prefixes section of the Set-Cookie reference article.

JavaScript access using Document.cookie

You can create new cookies via JavaScript using the `Document.cookie` property. You can access existing cookies from JavaScript as well if the `HttpOnly` flag isn't set.

```
document.cookie = "yummy_cookie=choco";
document.cookie = "tasty_cookie=strawberry";
console.log(document.cookie);
// logs "yummy_cookie=choco; tasty_cookie=strawberry"
```

Cookies created via JavaScript can't include the `HttpOnly` flag.

Please note the security issues in the Security section below. Cookies available to JavaScript can be stolen through XSS.

Security

> **Note:** When you store information in cookies, keep in mind that all cookie values are visible to, and can be changed by, the end user. Depending on the application, you may want to use an opaque identifier that the server looks up, or investigate alternative authentication/confidentiality mechanisms such as JSON Web Tokens.

Ways to mitigate attacks involving cookies:

- Use the `HttpOnly` attribute to prevent access to cookie values via JavaScript.
- Cookies that are used for sensitive information (such as indicating authentication) should have a short lifetime, with the `SameSite` attribute set to `Strict` or `Lax`. (See SameSite attribute, above.) In browsers that support SameSite, this ensures that the authentication cookie isn't sent with cross-site requests. This would make the request effectively unauthenticated to the application server.

Tracking and privacy

Third-party cookies

A cookie is associated with a particular domain and scheme (such as `http` or `https`), and may also be associated with subdomains if the `Set-Cookie` `Domain` attribute is set. If the cookie domain and scheme match the current page, the cookie is considered to be from the same site as the page, and is referred to as a *first-party cookie*.

If the domain and scheme are different, the cookie is not considered to be from the same site, and is referred to as a *third-party cookie*. While the server hosting a web page sets first-party cookies, the page may contain images or other components stored on servers in other domains (for example, ad banners) that may set third-party cookies. These are mainly used for advertising and tracking across the web. For example, the types of cookies used by Google.

A third-party server can create a profile of a user's browsing history and habits based on cookies sent to it by the same browser when accessing multiple sites. Firefox, by default, blocks third-party cookies that are known to contain trackers. Third-party cookies (or just tracking cookies) may also be blocked by other browser settings or extensions. Cookie blocking can cause some third-party components (such as social media widgets) not to function as intended.

There are some useful features available for developers who wish to respect user privacy, and minimize third-party tracking:

- Servers can (and should) set the cookie SameSite attribute to specify whether or not third-party cookies may be sent.
- Cookies Having Independent Partitioned State (CHIPS) enables developers to opt-in their cookies to partitioned storage, with a separate cookie jar per top-level site. This enables valid non-tracking uses of third-party cookies to continue working in browsers that do not allow cookies to be used for third-party tracking.

Cookie-related regulations

Legislation or regulations that cover the use of cookies include:

- The General Data Privacy Regulation (GDPR) in the European Union
- The ePrivacy Directive in the EU
- The California Consumer Privacy Act

These regulations have global reach. They apply to any site on the *World Wide* Web that users from these jurisdictions access (the EU and California, with the caveat that California's law applies only to entities with gross revenue over 25 million USD, among things).

These regulations include requirements such as:

- Notifying users that your site uses cookies.
- Allowing users to opt out of receiving some or all cookies.
- Allowing users to use the bulk of your service without receiving cookies.

There may be other regulations that govern the use of cookies in your locality. The burden is on you to know and comply with these regulations. There are companies that offer "cookie banner" code that helps you comply with these regulations.

Other ways to store information in the browser

Another approach to storing data in the browser is the Web Storage API. The window.sessionStorage and window.localStorage properties correspond to session and permanent cookies in duration, but have larger storage limits than cookies, and are never sent to a server. More structured and larger amounts of data can be stored using the IndexedDB API, or a library built on it.

There are some techniques designed to recreate cookies after they're deleted. These are known as "zombie" cookies. These techniques violate the principles of user privacy and user control, may violate data privacy regulations, and could expose a website using them to legal liability.

See also

- `Set-Cookie`
- `Cookie`
- `Document.cookie`
- `Navigator.cookieEnabled`
- Inspecting cookies using the Storage Inspector
- Cookie specification: RFC 6265
- HTTP cookie on Wikipedia
- Cookies, the GDPR, and the ePrivacy Directive