此页面由社区从英文翻译而来。了解更多并加入 MDN Web Docs 社区。

浏览器的同源策略

同源策略是一个重要的安全策略，它用于限制一个origin的文档或者它加载的脚本如何能与另一个源的资源进行交互。它能帮助阻隔恶意文档，减少可能被攻击的媒介。

同源的定义

如果两个 URL 的 protocol、port (en-US) (如果有指定的话)和 host 都相同的话，则这两个 URL 是*同源*。这个方案也被称为“协议/主机/端口元组”，或者直接是 “元组”。（“元组” 是指一组项目构成的整体，双重/三重/四重/五重/等的通用形式）。

下表给出了与 URL `http://store.company.com/dir/page.html` 的源进行对比的示例:

URL	结果	原因
`http://store.company.com/dir2/other.html`	同源	只有路径不同
`http://store.company.com/dir/inner/another.html`	同源	只有路径不同
`https://store.company.com/secure.html`	失败	协议不同
`http://store.company.com:81/dir/etc.html`	失败	端口不同 (`http://` 默认端口是80)
`http://news.company.com/dir/other.html`	失败	主机不同

源的继承

在页面中通过 `about:blank` 或 `javascript:` URL 执行的脚本会继承打开该 URL 的文档的源，因为这些类型的 URLs 没有包含源服务器的相关信息。

例如， about:blank 通常作为父脚本写入内容的新的空白弹出窗口的 URL（例如，通过 Window.open() ）。 如果此弹出窗口也包含 JavaScript，则该脚本将从创建它的脚本那里继承对应的源。

注意：在Gecko 6.0之前，如果用户在位置栏中输入 data URLs， data URLs 将继承当前浏览器窗口中网页的安全上下文。

data ：URLs 获得一个新的，空的安全上下文。

IE 中的特例

Internet Explorer 的同源策略有两个主要的差异点：

- **授信范围**（Trust Zones）：两个相互之间高度互信的域名，如公司域名（corporate domains），则不受同源策略限制。
- **端口**：IE 未将端口号纳入到同源策略的检查中，因此 https://company.com:81/index.html 和 https://company.com/index.html 属于同源并且不受任何限制。

这些差异点是不规范的，其它浏览器也未做出支持，但会助于开发基于window RT IE的应用程序。

源的更改

满足某些限制条件的情况下，页面是可以修改它的源。脚本可以将 document.domain 的值设置为其当前域或其当前域的父域。如果将其设置为其当前域的父域，则这个较短的父域将用于后续源检查。

例如：假设 http://store.company.com/dir/other.html 文档中的一个脚本执行以下语句：

```
document.domain = "company.com";
```

这条语句执行之后，页面将会成功地通过与 http://company.com/dir/page.html 的同源检测（假设 http://company.com/dir/page.html 将其 document.domain 设置为“ company.com ”，以表明它希望允许这样做 - 更多有关信息，请参阅 document.domain ）。然而， company.com 不能设置 document.domain 为 othercompany.com ，因为它不是 company.com 的父域。

端口号是由浏览器另行检查的。任何对document.domain的赋值操作，包括 `document.domain = document.domain` 都会导致端口号被重写为 `null` 。因此 `company.com:8080` **不能**仅通过设置 `document.domain = "company.com"` 来与 `company.com` 通信。必须在他们双方中都进行赋值，以确保端口号都为 `null` 。

> 注意：使用 `document.domain` 来允许子域安全访问其父域时，您需要在父域和子域中设置 document.domain 为相同的值。这是必要的，即使这样做只是将父域设置回其原始值。不这样做可能会导致权限错误。

跨源网络访问

同源策略控制不同源之间的交互，例如在使用 `XMLHttpRequest` 或 `<img>` 标签时则会受到同源策略的约束。这些交互通常分为三类：

- 跨域***写操作***（*Cross-origin writes*）一般是被允许的。例如链接（links），重定向以及表单提交。特定少数的HTTP请求需要添加 preflight。
- 跨域***资源嵌入***（*Cross-origin embedding*）一般是被允许（后面会举例说明）。
- 跨域***读操作***（*Cross-origin reads*）一般是不被允许的，但常可以通过内嵌资源来巧妙的进行读取访问。例如，你可以读取嵌入图片的高度和宽度，调用内嵌脚本的方法，或availability of an embedded resource .

以下是可能嵌入跨源的资源的一些示例：

- `<script src="..."></script>` 标签嵌入跨域脚本。语法错误信息只能被同源脚本中捕捉到。
- `<link rel="stylesheet" href="...">` 标签嵌入CSS。由于CSS的松散的语法规则 ，CSS的跨域需要一个设置正确的 HTTP 头部 `Content-Type` 。不同浏览器有不同的限制： IE , Firefox , Chrome , Safari (跳至CVE-2010-0051)部分 和 Opera 。
- 通过 `<img>` 展示的图片。支持的图片格式包括PNG,JPEG,GIF,BMP,SVG,...
- 通过 `<video>` 和 `<audio>` 播放的多媒体资源。
- 通过 `<object>` 、 `<embed>` 和 `<applet>` 嵌入的插件。
- 通过 `@font-face` 引入的字体。一些浏览器允许跨域字体（cross-origin fonts），一些需要同源字体（same-origin fonts）。
- 通过 `<iframe>` 载入的任何资源。站点可以使用 X-Frame-Options 消息头来阻止这种形式的跨域交互。

如何允许跨源访问

可以使用 CORS 来允许跨源访问。CORS 是 HTTP 的一部分，它允许服务端来指定哪些主机可以从这个服务端加载资源。

如何阻止跨源访问

- 阻止跨域写操作，只要检测请求中的一个不可推测的标记(CSRF token)即可，这个标记被称为 Cross-Site Request Forgery (CSRF) 标记。你必须使用这个标记来阻止页面的跨站读操作。
- 阻止资源的跨站读取，需要保证该资源是不可嵌入的。阻止嵌入行为是必须的，因为嵌入资源通常向其暴露信息。
- 阻止跨站嵌入，需要确保你的资源不能通过以上列出的可嵌入资源格式使用。浏览器可能不会遵守 `Content-Type` 头部定义的类型。例如，如果您在HTML文档中指定 `<script>` 标记，则浏览器将尝试将标签内部的 HTML 解析为JavaScript。 当您的资源不是您网站的入口点时，您还可以使用CSRF令牌来防止嵌入。

跨源脚本API访问

JavaScript 的 API 中，如 `iframe.contentWindow`、 `window.parent`、 `window.open` 和 `window.opener` 允许文档间直接相互引用。当两个文档的源不同时，这些引用方式将对 Window 和 Location 对象的访问添加限制，如下两节所述。

为了能让不同源中文档进行交流，可以使用 `window.postMessage`。

规范: HTML Living Standard § Cross-origin objects 。

Window

允许以下对 `Window` 属性的跨源访问：

方法

`window.blur`

`window.close`

`window.focus`

`window.postMessage`

属性

`window.closed`	只读.

属性

`window.frames`	只读.
`window.length`	只读.
`window.location`	读/写.
`window.opener`	只读.
`window.parent`	只读.
`window.self`	只读.
`window.top`	只读.
`window.window`	只读.

某些浏览器允许访问除上述外更多的属性。

Location

允许以下对 `Location` 属性的跨源访问：

方法

`location.replace`

属性

`URLUtils.href`	只写.

某些浏览器允许访问除上述外更多的属性。

跨源数据存储访问

访问存储在浏览器中的数据，如 localStorage 和 IndexedDB，是以源进行分割。每个源都拥有自己单独的存储空间，一个源中的 JavaScript 脚本不能对属于其它源的数据进行读写操作。

Cookies 使用不同的源定义方式。一个页面可以为本域和其父域设置 cookie，只要是父域不是公共后缀（public suffix）即可。Firefox 和 Chrome 使用 Public Suffix List 检测一个域是否是公共后缀（public suffix）。Internet Explorer 使用其内部的方法来检测域是否是公共后缀。不管使用哪个协议（HTTP/HTTPS）或端口号，浏览器都允许给定的域以及其任何子域名(sub-domains) 访问 cookie。当你设置 cookie 时，你可以使用 `Domain`、 `Path`、 `Secure`、和 `HttpOnly` 标记来限定其可访问性。当你读取

cookie 时，你无法知道它是在哪里被设置的。 即使您只使用安全的 https 连接，您看到的任何 cookie 都有可能是使用不安全的连接进行设置的。

参见

- Same-Origin Policy at W3C
- http://web.dev/secure/same-origin-policy

原始文件资料

- Author(s): Jesse Ruderman

Last modified: 2022年3月10日, by MDN contributors