55. 浏览器中的同源策略与 JS 的关系

1. 核心概念 (Core Concept)

浏览器的同源策略（Same-Origin Policy）是一种重要的安全机制，它限制了来自一个源（Origin）的文档或脚本如何与来自另一个源的资源进行交互。这里的“源”通常指协议（protocol）、域名（domain）和端口号（port）三者的组合。同源策略主要针对的是那些可能具有潜在危险的跨源读写操作。

2. 为什么需要它？ (The "Why")

- **防止恶意脚本读取敏感数据:** 同源策略是防止跨站脚本攻击（XSS）和跨站请求伪造（CSRF）等攻击的关键防御层。它阻止了恶意网站上的 JavaScript 在用户不经意间读取用户在其他敏感网站（如银行网站）上的 Cookies、本地存储或其他私密信息。
- **隔离不同应用的资源:** 它确保了运行在浏览器中的不同 Web 应用在没有用户明确授权的情况下，无法互相访问或修改对方的数据和资源，维护了 Web 环境的隔离性。
- **阻止恶意网站模拟用户操作:** 在某些场景下，同源策略也限制了脚本发起跨域的 HTTP 请求，防止恶意网站利用用户在另一网站上的登录状态去执行非预期的操作。

3. API 与用法 (API & Usage)

同源策略是浏览器内置的安全机制，它不是一个可以通过 JavaScript API 直接“使用”或“关闭”的功能。它是在浏览器层面**默默地**对脚本发起的各种跨源操作进行限制。

主要受同源策略限制的**跨源操作**包括：

- **XMLHttpRequest 或 Fetch API 发起的 HTTP 请求:** 这是最常见的受限场景。一个源的脚本无法直接通过 `XMLHttpRequest` 或 `fetch` 读取来自另一个源的响应数据（除非目标源明确允许，如通过 CORS）。
- **DOM 操作:** 难以或不可能通过 JavaScript 直接访问和操作加载自不同源的 `iframe` 中的 DOM 元素或脚本。
- **Cookies, LocalStorage, IndexDB:** 存储在浏览器中的数据（如 Cookies、localStorage、IndexDB）是**域隔离**的，一个源的脚本无法直接访问另一个源存储的数据。

代码示例（受限场景演示 - 伪代码概念）：

这是一个概念性的示例，说明在没有 CORS 的情况下，跨源的 `fetch` 请求会因同源策略而失败（虽然请求可能发出，但脚本无法读取响应）。

```
// 假设当前页面源是: https://site-a.com
// 尝试从另一个源获取数据
fetch('https://site-b.com/api/data')
  .then(response => {
```

```
    // 如果没有 CORS，同源策略会阻止脚本读取 response.json() 的内容
    // 即使网络请求成功，这里也可能出现跨域错误
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => {
    console.log('获取的数据:', data); // 这行通常不会在跨域失败时执行
  })
  .catch(error => {
    console.error('获取数据失败:', error); // 大部分情况下，跨域错误会在这里捕获
  });

// 注意：这只是演示同源策略的影响，实际错误处理取决于具体场景和浏览器实现。
// 解决跨域请求通常需要目标服务器配置 CORS 响应头。
```

不受同源策略限制的跨源加载（只读，不涉及敏感数据读取）：

有些跨源加载操作是不受同源策略严格限制的，但这通常是**只读**或不直接暴露敏感数据的形式：

- `<script src="...">` 标签加载外部脚本。
- `<img>` 标签加载图片。
- `<link rel="stylesheet" href="...">` 标签加载 CSS。
- `<video>` 和 `<audio>` 标签加载媒体文件。
- `<object>`、`<embed>` 和 `<applet>` 标签。
- 通过 `@font-face` 引用字体。
- `<iframe src="...">` 加载页面，但无法直接通过脚本读取 iframe 内容。

```
<!-- 跨源加载不受限（基本资源类型） -->
<script src="https://another-domain.com/script.js"></script>
<img src="https://another-domain.com/image.png" alt="Cross-origin Image">
<link rel="stylesheet" href="https://another-domain.com/style.css">
<iframe src="https://another-domain.com/page.html"></iframe>
```

4. 关键注意事项 (Key Considerations)

- **源（Origin）的定义:** 记住源由协议、域名和端口号共同决定。`http://example.com` 和 `https://example.com` 是不同源，`http://example.com:80` 和 `http://example.com:8080` 也是不同源（默认端口 80/443 可省略但在判断时仍需考虑）。
- **跨域解决机制:** 同源策略带来的跨域限制可以通过一些机制来解决或规避，最常见和推荐的是 **CORS (Cross-Origin Resource Sharing)**，由服务器端配置响应头来告知浏览器允

许来自哪些源的跨域访问。其他方法还包括 JSONP（仅限 GET 请求，已较少使用）、代理、WebSocket 等。

- **document.domain:** 在某些特定场景下，如果两个页面的域名相同但子域名不同（例如 `a.example.com` 和 `b.example.com`），可以通过将它们的 `document.domain` 手动设置为相同的祖先域（如 `example.com`）来尝试放宽部分同源策略限制，但这存在安全风险且有局限性。
- **postMessage API:** 对于不同源窗口之间的安全通信，可以使用 `window.postMessage()` API，它提供了一种受控的跨窗口消息传递机制，避免了直接的 DOM 和数据访问。

5. 参考资料 (References)

- **MDN Web Docs:** 同源策略 (Same-origin policy)
  - https://developer.mozilla.org/zh-CN/docs/Web/Security/Same-origin_policy
- **MDN Web Docs:** 跨源资源共享 (CORS)
  - https://developer.mozilla.org/zh-CN/docs/Web/HTTP/CORS
- **MDN Web Docs:** Window.postMessage()
  - https://developer.mozilla.org/zh-CN/docs/Web/API/Window/postMessage
- **HTML Standard:** Browsing the web (Contains definition of origin and cross-origin restrictions)
  - https://html.spec.whatwg.org/multipage/browsing-the-web.html#browsing-the-web (需阅读相关章节，如 "Origins" 和 "Security")