跨來源資源共用 Cross-Origin Resource Sharing (CORS)

- 同源政策
  1. 用 JavaScript 透過 fetch API 或 XMLHttpRequest 等方式發起 request，必須遵守同源政策 (same-origin policy)

  2. 用 javascript 存取資源
     (1) 同源：存取不受限
     (2) 非同源：基於安全性考量，request 會受到限制
     =>若不遵守 CORS 的規範，瀏覽器會讓 request 失敗

  3. 同源：須滿足 3 條件
     (1) 相同的通訊協定 (protocol)： http/https
     (2) 相同的網域 (domain)
     (3) 相同的通訊埠 (port)

  4. 非同源=>產生跨來源 http 請求

```
try {
  fetch('https://othersite.com/data')
} catch (err) {
  console.error(err);
}
```

若伺服器設定錯誤=>違反 CORS

```
Access to fetch at *** from origin *** has been blocked
by CORS policy: No 'Access-Control-Allow-Origin' header
is present on the requested resource. If an opaque
response serves your needs, set the request's mode to
'no-cors' to fetch the resource with CORS disabled.
```

- CORS
  1. 針對非同源 request 的規範
  2. 透過 JavaScript 存取非同源資源時，server 必須明確告知瀏覽器允許何種請求，只有 server 允許的請求能夠被瀏覽器實際發送，否則會失敗。
  3. 種類
     (1) 簡單跨來源請求
     (2) 一般跨來源請求

- 簡單跨來源請求
  1. 只能是 HTTP 方法的 GET、POST、 HEAD
  2. 自訂的 request header 只能是 Accept、Accept-Language、Content-Language、Last-Event-ID、DPR、Save-Data、Viewport-Width、Width 或 Content-Type（值只能是 application/x-www-form-urlencoded、multipart/form-data 或 text/plain）

```
const response = await fetch('https://othersite.com/data', {
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
    'X-CUSTOM-HEADER': '123'
  }
});
```

  //上例 [redacted] 為不合法處，非簡單跨來源請求
  3. 沒有事件監聽器被註冊到任何用來發出請求的 XMLHttpRequestUpload 物件
  4. 請求中沒有 ReadableStream (en-US) 物件被用於上傳

- 非簡單跨來源請求
    1. 不符合簡單跨來源請求
    2. 瀏覽器在發送請求之前會先發送一個預檢請求，當瀏覽器看到跨來源請求的方法和 header 都有被列在允許的方法和 header 中=> 瀏覽器收到正確的 preflight response=>CORS 的驗證通過，可以送出跨來源請求

```
fetch('https://othersite.com/data/', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-CUSTOM-HEADER': '123'
  }
})
.then(response => response.json())
.then(json => {
  console.log(json);
});
```

request header 會長得會像這樣：

```
POST /data/
Host: othersite.com
Origin: https://shubo.io
Content-Type: application/json
X-MY-CUSTOM-HEADER: 123
```

瀏覽器幫我們發送 preflight request 就會像這樣：

```
OPTIONS /data/
Host: othersite.com
Origin: https://shubo.io
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-MY-CUSTOM-HEADER, Content-Type
```

- Origin (來源)
  1. 包含：通訊協定、網域、通訊埠
  2. 瀏覽器發送跨來源請求時，會帶一個 Origin header，表示這個請求的來源
  3. 當 server 端收到這個跨來源請求時，它可以依據「請求的來源」，亦即 Origin 的值，決定是否要允許這個跨來源請求。如果 server 允許這個跨來源請求，它可以「授權」給這個來源的 JavaScript 存取這個資源
  4. 當瀏覽器收到回應時，會檢查請求中的 Origin header 是否符合回應的 Access-Control-Allow-Origin header
     (1) 相符：瀏覽器讓這個請求成功，我們也可以順利地用 JavaScript 讀取到回應
     (2) 不相符：瀏覽器會將這個 request 視為是不安全的而讓他失敗，即便 server 確實收到請求也成功地回應了，但基於安全性的理由 JavaScript 中沒有辦法讀到回應。

所以從 `https://shubo.io` 發出的往 `https://othersite.com/data` 的請求會像這樣：

```
GET /data/
Host: othersite.com
Origin: https://shubo.io
...
```

授權的方法是在 response 裡加上 `Access-Control-Allow-Origin` header：

```
Access-Control-Allow-Origin: https://shubo.io
```

如果 server 允許任何來源的跨來源請求，那可以直接回 `*`：

```
Access-Control-Allow-Origin: *
```

- 解決基本的 CORS 錯誤
  1. 請後端設置 CORS header
     (1) 若為「簡單」的跨來源請求：
     在後端 GET/POST/HEAD 方法本身加上 Access-Control-Allow-Origin header
     (2) 若為「非簡單」跨來源請求：
     在後端 OPTIONS 加上 Access-Control-Allow-Methods 及 Access-Control-Allow-Headers header。
     另外，在後端方法本身加上 Access-Control-Allow-Origin header

  2. 使用 proxy server
     - proxy server 代理伺服器
       1. 想拿 A 網站的資料，但是它沒有提供 Access-Control-Allow-Origin 這個 header
       =>自己寫 proxy server，從後端去拿 A 網站的資料，再把資料丟回給自己的前端
       2. 步驟
          (1) 瀏覽器發 request 到 proxy，拿網站的資料
          (2) proxy server 去跟網站拿資料（後端，不是瀏覽器所以沒有跨來源限制）
          (3) 網站回傳資料給 proxy（同上，沒有跨來源限制）
          (4) proxy 回傳資料給瀏覽器，並加上 CORS header（所以前端不會被擋）