一、 為什麼會發生 CORS

CORS 全名為 Cross-Origin Resource Sharing(跨來源資料)，這個錯誤是出在「跨來源呼叫 API」。在寫 html 檔案有時候會需要用到 API，例如之前網頁作業的捷運票價及時間，還有這周上課的 Quiz 7。

大部分的 CORS 都不是前端的問題，純前端是無法解決這個問題的。跨來源[1]是指從來源 A 去拿來源 B 的東西，而 A 跟 B 是在不同 origin[2]，這個來源代表著「發送 request 的來源」，而 CORS 錯誤被擋住的是 response 而不是 request。。如果我們想避開跨來源的話，就會把前後端都放在同一個 origin 下。

跨來源呼叫 API 被擋住的原因為**安全方面**的問題，只要不肖人士在他自己的網頁用 AJAX 去連結到別的公司的網址拿到公司機密資料，就會造成機密外洩，而他們就可以分析這些網站找出漏洞攻擊這些公司的網站。

而當我們要引入的只是一個圖片 、CSS 樣式或是 script 卻不會被擋住是因為上述只是網頁資源的一部分，而我們取回來的資源是沒辦法用程式讀取它的，圖片資源就只有瀏覽器知道圖片的內容，在本機端這邊是沒辦法知道圖片內容的，也無法將結果傳到其他地方，資料外洩的問題就不存在。

二、 如何解決 CORS 問題

要如何解決 CORS 錯誤？第一個解法是關掉瀏覽器的安全設置，但是把安全機制關掉後，不只是 CORS，連其他的安全機制也一併被關掉了，而且這個方法是在自己的電腦上沒問題，但是別人的電腦上卻是有問題的。方法二是把 fetch mode 設成 no-cors，這個是適用在如果使用 fetch 去抓取資料，但是改成 no-cors 代表著跟瀏覽器說：**發送一個 request 到沒有 CORS header 的 url**。我們已經預想到可能會有錯誤，所以**避開了這個錯誤但是我們一定**

[1] 而 same origin 是指來源相同。兩個 URL A 跟 B 的 origin 是一樣的就稱 A 跟 B 為 same origin。
[2] origin 可當作成：sceme+host+port 的組合。以 https://www.cgu.edu.tw 為例，scheme 就是 https 或是 http，host 是 www.cgu.edu.tw，而 port 若沒有特別指定的話，http 預設的 port 是 80，https 則是 443。

拿不到 response。第三個解法非常直接：不要用 AJAX 拿資料，**改成利用 script 去獲取資料**，因為 script 標籤不會擋跨來源請求，讓 server 動態產生檔案的內容再利用呼叫 JS funcction 傳遞 JSON 格式的資料(就是 **JSONP**[3])。最正確的解法是：請後端設置 CORS header，因為後端才擁有權限，可以告訴 browser 允許這個 origin 跨來源來存取資源。例如下文：

> 這個 header 的名稱叫做 Access-Control-Allow-Origin ，內容就是你想要放行的 origin，例如說： Access-Control-Allow-Origin: http://localhost:8081，這樣就是允許 http://localhost:8081 的跨來源請求。

或是用 proxy server，代理服務器，如下圖。但是有些 proxy server 很不穩定。所以最正確的方法就是在後端設置 CORS header。

```
<script>
var dataUrl = "https://cors-anywhere.herokuapp.com/https://
wic.heo.taipei/OpenData/API/Rain/Get?
stationNo=&loginId=open_rain&dataKey=85452C1D";
```

三、 後端加上 header 解決 CORS 問題

跨來源請求可分為簡單請求與非簡單請求。要解決 CORS 有很多 header，如下：

1. 在後端加上 `Access-Control-Allow-Origin:*`[4]：代表來自任何 Origin 的網站 2 都可以用 AJAX 存取這資源。

   後端：

```
app.get('/', (req, res) => {
  res.header('Access-Control-Allow-Origin', '*')
  res.json({
    data: db.getFormOptions(),
  })
})
```

   前端：

[3] JSONP 的原理是透過 script 標籤傳遞資料跨過限制，而一般使用的 AJAX 都是用 XMLHttpRequest 或是 fetch，這兩種方法的原理相去甚遠，完全不一樣。

[4] 無論是簡單請求或是非簡單請求，後端都要給 `Access-Control-Allow-Origin` 這個 header。差別在於非簡單請求在發送正式的 request 之前會先發送一個 preflight request，如果這個沒通過，是不會發出正式的 request。

```
fetch('http://localhost:3000')
  .then(res => res.json())
  .then(res => console.log(res))
```

2. 在後端加上 `Access-Control-Allow-Headers`：當 CORS request 含有自訂的 header 時，需要用這個來表明願意接受這個 header，瀏覽器才會預檢通過 preflight request[5]，接著發送正式的 request。

   後端：

```
app.options('/form', (req, res) => {
  res.header('Access-Control-Allow-Origin', '*')
  res.header('Access-Control-Allow-Headers', 'X-App-Version, content-type')
  res.end()
})
```

   前端：

```
fetch('http://localhost:3000/form', {
      method: 'POST',
      headers: {
        'X-App-Version': "v0.1",
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    }).then(res => res.json())
      .then(res => console.log(res))
```

3. 帶上 Cookie：跨來源的請求預設是不會帶上 cookie 的。所以要先確保以下三點符合，才能帶 Cookie。[6]

   a. 後端 Response header 有 `Access-Control-Allow-Credentials: true`

   b. 後端 Response header 的 `Access-Control-Allow-Origin` 不能是`*`，要明確指定

   c. 前端 `fetch` 加上 `credentials: 'include'`

---

[5] Preflight 是一個驗證機制，驗證相容性跟安全性，確保後端知道前端要送出的 request 是預期的，瀏覽器才會通過。

[6] 不只是帶上 cookie，連設置 cookie 也要這三個條件。後端可以用 `Set-Cookie` 這個 header 讓瀏覽器設置 cookie。無論是否要存取 cookie 都會建議 `Access-Control-Allow-Origin` 不要設置成`*`而是要有一個明確的 origin。

後端：

```
const VALID_ORIGIN = 'http://localhost:8080'
app.post('/form', (req, res) => {
  res.header('Access-Control-Allow-Origin', VALID_ORIGIN) // 明確指定
  res.header('Access-Control-Allow-Credentials', true) // 新增這個
  res.json({
    success: true
  })
})

app.options('/form', (req, res) => {
  res.header('Access-Control-Allow-Origin', VALID_ORIGIN) // 明確指定
  res.header('Access-Control-Allow-Credentials', true) // 新增這個
  res.header('Access-Control-Allow-Headers', 'content-type, X-App-Version')
  res.end()
})
```

前端：

```
fetch('http://localhost:3000/form', {
  method: 'POST',
  credentials: 'include', // 新增這個
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
}).then(res => res.json())
  .then(res => console.log(res))
```

4. 在後端加上 `Access-Control-Expose-Headers`：若要存取 response 的 header，尤其是自定義的 header，要加上這個 header 才能存取成功。

   後端：

```
app.get('/', (req, res) => {
  res.header('Access-Control-Allow-Origin', '*')
  res.header('Access-Control-Expose-Headers', 'X-List-Version') // 加這個
  res.header('X-List-Version', '1.3')
  res.json({
    data: [
      {name: '1/10 活動', id: 1},
      {name: '2/14 特別活動', id: 2}
    ]
  })
})
```

前端：

```
fetch('http://localhost:3000')
  .then(res => {
    console.log(res.headers.get('X-List-Version'))
    return res.json()
  })
  .then(res => console.log(res))
```

5. 後端加上 `Access-Control-Allow-Methods`：如果前端要使用 `GET`、`HEAD`、`POST` 以外的 HTTP method 發送請求的話，必須要有這個 header 而且指定合法的 method，preflight 才會通過。

   後端：

```
// preflight
app.options('/form', (req, res) => {
  res.header('Access-Control-Allow-Origin', VALID_ORIGIN)
  res.header('Access-Control-Allow-Credentials', true)
  res.header('Access-Control-Allow-Methods', 'PATCH') // 多這個
  res.header('Access-Control-Allow-Headers', 'content-type, X-App-Version')
  res.end()
})
```

   前端：

```
fetch('http://localhost:3000/form', {
  method: 'PATCH',
  credentials: 'include',
  headers: {
    'X-App-Version': "v0.1",
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    token: 'test_token',
    content: 'new content'
  })
}).then(res => res.json())
  .then(res => console.log(res))
```

6. 快取 preflight request：用 `Access-Control-Max-Age: 數字`，讓後端把相同的 preflight 快取住，這樣同個瀏覽器重複發送 request 就不用再做預檢了。

   後端：

```
app.options('/form', (req, res) => {
  res.header('Access-Control-Allow-Origin', VALID_ORIGIN)
  res.header('Access-Control-Allow-Credentials', true)
  res.header('Access-Control-Allow-Headers', 'content-type, X-App-Version')
  res.header('Access-Control-Max-Age', 300)
  res.end()
})
```

四、 CORS 的 spec (更詳盡的介紹：fetch.spec.whatwg.org)

1. Origin

   Origin 的內容只會有兩種，一種是`"null"`，是一個字串；另一種就是 scheme + host + port 的組合。

2. CORS

   CORS protocol 的存在是為了讓網頁有除了 form 的元素以外，也可以抓取跨來源資源的方法，死 protocol 是建立在 HTTP5 之上的。

   CORS 是透過 header 來決定一個 response 是否能夠被跨來源共享 (CORS 就是藉由一堆的 response header 來跟瀏覽器說哪些東西是前端有權限存取的。)如果一個 request 超過 HTML 的 form 源物可以表達的範圍，那就會有一個 CORS-preflight request。CORS-preflight request 是利用 OPTIONS 來確認 server 是不是理解 CORS protocol。

   一個 HTTP 如果含有 origin 這個 header 就稱為 CORS request，但

是不代表 request 跟 CORS protocol 有關係。

在 CORS protocol 中的 preflight request 會帶著兩個 header：

a. `Access-Control-Request-Method`
b. `Access-Control-Request-Headers`

來表示之後的 CORS request 可能會用到的 method 跟 header。

而 response 則會用：

a. `Access-Control-Allow-Origin`：決定哪些 origin 合法
b. `Access-Control-Allow-Credentials`：決定是否允許帶上以及設置 cookie

CORS-preflight request 也是一種 CORS request，所以上述針對 CORS request 可以給的 response 也都可以用。除此之外還定義了：

a. `Access-Control-Allow-Methods`：可以使用哪些 method
b. `Access-Control-Allow-Headers`：可以使用哪些 header
c. `Access-Control-Max-Age`：前兩個 header 可以快取多長時間

而針對不是 preflight 的 CORS request 可以提供 `Access-Control-Expose-Headers` 用來指名哪些 header 可以存取，一定要明確指定，不然 response 還是沒辦法拿到 header。

3. Preflight request

若 request 的 method 不是 CORS-safelisted method 或是 header 裡面有 CORS-unsafe request-header name 的話，就會設置 CORS-preflight flag 並進行 HTTP fetch。

只有 GET、HEAD、POST 才不會觸發到 preflight。CORS-unsafe request-header name 會去檢查 headers 是不是都是 CORS-safelisted request-header，基本上只有以下幾種會過：

a. accept
b. accept-language
c. content-language
d. content-type
    i. application/x-www-from-urlencoded
    ii. multipart/form-data
    iii. text/plain

4. CORS check

要如何驗證 CORS 是過關的？若 `Access-Control-Allow-Origin` 裡面的 origin 是 null，這裡的 null 不是字串，就是失敗；再來，若 origin 是* 且 credentials mode 不是 include 就是成功；接著看 request 的 origin 跟 header 裡的如果不同就是回傳失敗，相同就在去看 credentials mode 不是 include 就給過。相反的，去檢查 `Access-Control-Allow-Credentials`，如果是 true 就是成功的。

五、 心得

在閱讀完作者寫的一系列有關 CORS 問題的種種，有更深刻的認識到 CORS，以及有粗淺的的概念去解決它。就像老師之前說的，還沒辦法用 dataUrl 的時候，就先自己寫一個假的資料，回傳看看是不是正確的，等到最後等到後端處理完就可以直接連上。

最後，報告心得真的比程式作業好很多，雖然查找跟閱讀上會花比較多時間，但是至少我還不會抱頭哀號。如果可以，教授之後上課的時候可以給我們多一些的例子去寫程式作業，因為我經過了 qz7，還是不會用直接用 JSON 檔去呈現資料 QQ。