基礎網路知識

何謂HTTP?

- HTTP 全名是 超文本傳輸協定（HyperText Transfer Protocol），內容只規範了**客戶端請求與伺服器回應的標準**，實際上是藉由 TCP 作為資料的傳輸方式。
- 簡單來說，HTTP就是往返「瀏覽器（web端）」與「網頁伺服器（server端）」的通訊協議

HTTP

通訊協議

瀏覽器
Browser

網頁伺服器
Web Sever

何謂HTTP?

- 在HTTP協議中，Web端與Server端進行通訊是使用**明文的方式**，因為HTTP協議本身並**不具備加密的功能**，所以無法對請求端以及響應端的內容進行加密
- 為什麼HTTP**不使用加密協定**呢？
  - 加密會多消耗許多運算資源，
  - 佔用更多的傳輸頻寬
  - 緩存機制跟著會失效

何謂HTTPS?

- HTTPS 全名 超文本傳輸安全協定（ HyperText Transfer Protocol Secure ）

HTTPS → Secure 安全

HTTPS協定，以保密為前提為研發，可以算是HTTP的進階安全版

何謂HTTPS?

瀏覽器
Browser

HTTPS

傳輸過程中多了
一道加密的手續

網頁伺服器
Web Sever

若傳輸的資料中有敏感資料的話，可以使用HTTPS來加密資料，保障安全（如大型金融業）

延伸閱讀與參考資料

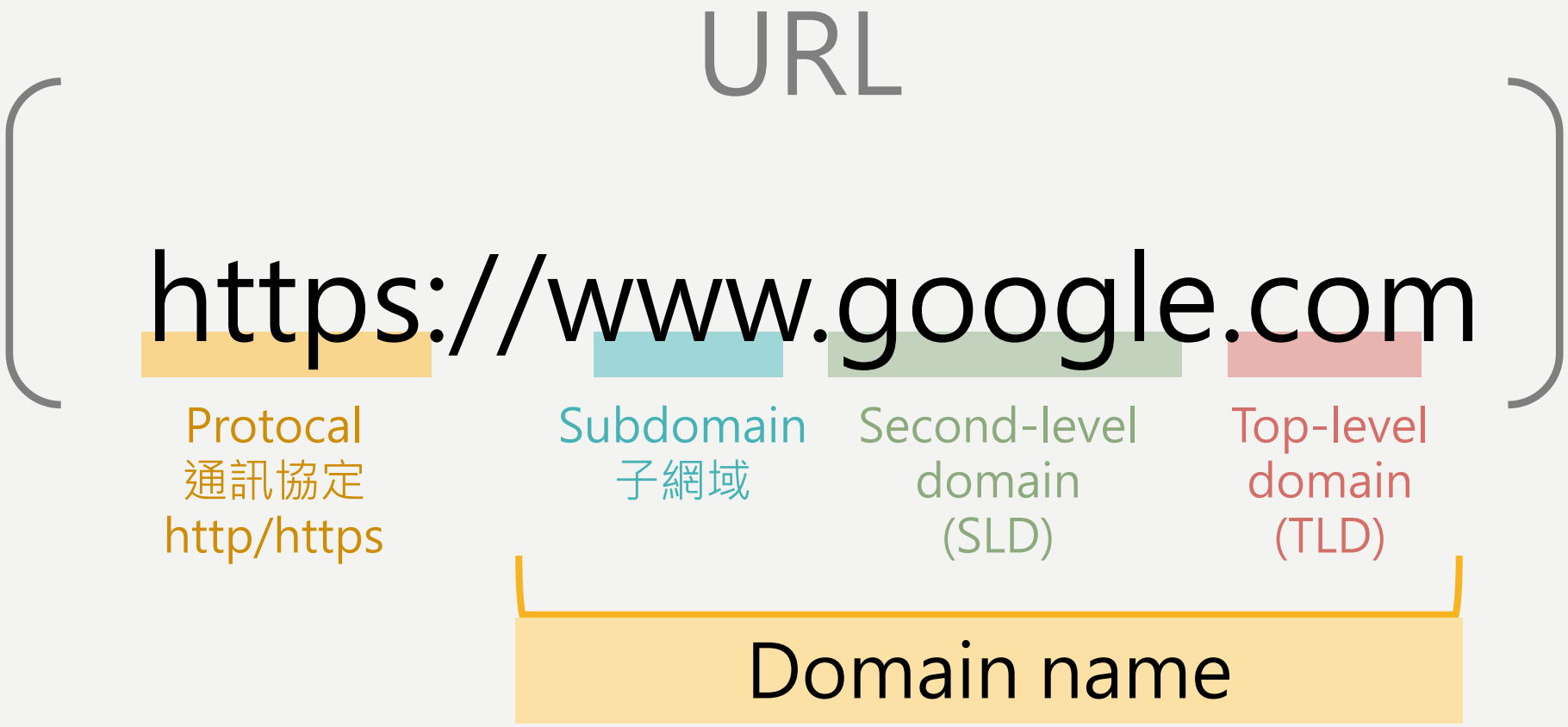
- 為什麼HTTPS比HTTP重要？一次帶你了解兩者的差異和重要性
- [不是工程師] 差一個字差很多，HTTP 不等於 HTTPS
- 一文搞懂 HTTP 和 HTTPS 是什麼？兩者有什麼差別

何謂DNS?

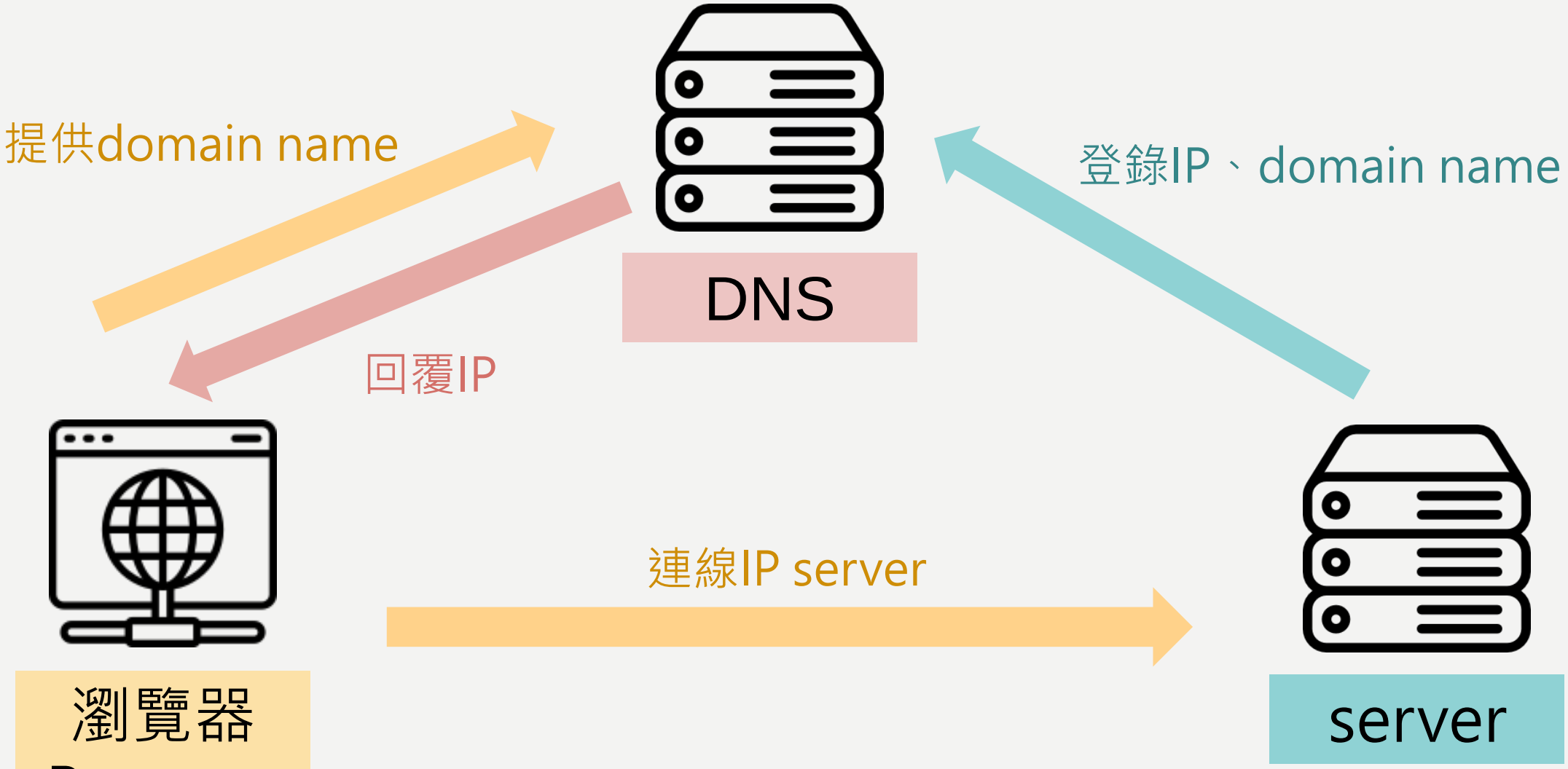
- DNS（Domain Name System）的中文名稱為網域名稱系統，主要負責維護網域名稱（domain name）與IP位址（IP address）的對應。
- 一般來說serve只能認得IP(比如：172.217.160.178)，而非https://www.google.com/這串文字，因此我們可以透過DNS進行轉譯

什麼是Domain name?

由於IP對於人來說相對難記住，因此有了Domain name來幫助我們更好記住



DNS 在做什麼呢?



圖片參考：http://www.tsnien.idv.tw/Internet_WebBook/chap13/13-2%20DNS%20%E7%B3%BB%E7%B5%B1%E5%8A%9F%E8%83%BD.html

延伸閱讀與參考資料

- 自媒體經營分享 | DNS篇 – 網站正確連線的三大要素入門知識
- DNS是什麼？認識DNS運作原理、伺服器類型及IP查詢流程
- What is DNS? | How DNS works

什麼是CORS？
(CROSS-ORIGIN RESOURCE SHARING)

- 簡單地說，CORS (Cross-Origin Resource Sharing) 是針對不同源的請求而定的規範
- 基於安全性問題，當瀏覽器透過 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

如何判斷是不是同源呢？（ SAME-ORIGIN POLICY ）

- ✓ 相同協定
- ✓ 主機位置
- ✓ 相同埠號 (如果有指定)

（IE 對於不同 port 會視為同源 ）

如何判斷是不是同源呢？（ SAME-ORIGIN POLICY ）

與 http://store.company.com/dir/page.html是不是同源呢？

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Same origin	Only the path differs
http://store.company.com/dir/inner/another.html	Same origin	Only the path differs
https://store.company.com/page.html	Failure	Different protocol
http://store.company.com:81/dir/page.html	Failure	Different port (http:// is port 80 by default)
http://news.company.com/dir/page.html	Failure	Different host

資料來源：https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

什麼是跨來源請求？

跨來源 http 請求
(cross-origin http request)

必須遵守 CORS 的規範

https://www.a.com

非同源

https://www.b.com

```
Access to XMLHttpRequest at 'https://google.com/' from origin 'chrome dashboard.html:1
-extension://laookkfknpbbblfpciffpaejjkokdgca' has been blocked by CORS policy: No
'Access-Control-Allow-Origin' header is present on the requested resource.
```

image source: Memegenerator.net

沒有遵守CORS規範的話會出現錯誤
這樣的情形可能會發生在串接API時

同源政策是瀏覽器專屬，所以才會發生用 postman 可以拿到 API 回應但放到網站上就是會失敗的狀況。

也有允許跨域的時候

跨來源讀取通常被禁止 (*reads*)
domain-a.com不能讀取

domain-b.com的

✕ cookie

✕ XMLHttpRequest

✕ Fetch API

特定情況下允許跨域

1. 跨來源嵌入(embed)
   - ✓ <script src=”…” ></script>
   - ✓ <link rel=” stylesheet” href=”…” >
   - ✓ <iframe >
   - ✓ <embed >
   - ✓ 圖片、影片

2. 跨來源寫入(writes)
   - ✓ 可以在藉由<form>在domain-a.com發 request 給domain-b.com，當然透過連結 links 或直接 redirect 到別的網站也是被允許的。

CORS規範與設定

由後端server進行設定，前端可以從Response Header裡看制定了哪一些規範

- 權限最大：Access-Control-Allow-Origin: *
- 針對特定網站開權限： Access-Control-Allow-Origin: www.fubon.com
- 允许 client side 帶 cookie 等驗證，default 是 false
  Access-Control-Allow-Credentials: true
- 可以設定允許哪些 method，default 是全部方法
  Access-Control-Request-Method: POST, GET

更多設定：MDN-Cross-Origin Resource Sharing (CORS)

CORS 請求種類：簡單請求

直接送出request，Server端收到request後，會先判斷Origin是否在許可範圍內，如許可則會回傳「Access-Control-Allow-Origin」資料串；如非許可，則會回傳錯誤（onerror）。
簡單請求條件如下，除此之外都屬於非簡單請求

- 請求方法是HEAD、GET、POST 其中一項
- HTTP的header為以下幾種（沒有其他自訂義需求）：
  - ✓ Accept
  - ✓ Accept-Language
  - ✓ Content-Language
  - ✓ Last-Event-ID
  - ✓ Content-Type：
    application/x-www-form-urlencoded、
    multipart/form-data、
    text/plain

```
GET /data/
Host: https://a.com
Origin:
https://fubon.com
```

CORS 請求種類：非簡單請求

非「簡單」的跨來源請求，例如：HTTP PUT/DELETE 方法，或是 Content-Type: application/json 等，瀏覽器在發送請求之前會先發送一個「preflight request（預檢請求）」，其作用在於先問伺服器：你是否允許這樣的請求？真的允許的話，我才會把請求完整地送過去。

- **Preflight Request (預檢請求)**

  Preflight request 是一個 http OPTIONS 方法，會帶有兩個 request header：Access-Control-Request-Method 和 Access-Control-Request-Headers。

  - Access-Control-Request-Method： 非「簡單」跨來源請求的 HTTP 方法。
  - Access-Control-Request-Headers 非「簡單」跨來源請求帶有的非「簡單」header。

CORS 請求種類：非簡單請求

前端發送非簡單請求

```
fetch('https://othersite.com/data/' ,
 {
   method: 'POST' ,
   headers: { 'Content-Type': 'application/json', 'X-CUSTOM- HEADER' :'123' }
 }
)
```

Request header

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

Preflight request

```
OPTIONS /data/
Host: xxx.com
Origin: https://myweb.com/
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Content-Type
```

範例來源： [教學] CORS 是什麼? 如何設定 CORS?

跨來源請求預設無法帶COOKIE

一般的http request 會帶有該網域底下的 cookie，**跨來源請求預設是不能帶 cookie ！**

Cookie可以攜帶與使用者相關的資訊，像是使用者登入資訊、購物車內容、瀏覽紀錄、seesion token，等於在請求裡就可能代表user的身份可以去存取機敏資料。

須要明確地標示「我要存取跨域 cookie」
如果是允許使用 cookie 的情況，「Access-Control-Allow-Origin不能用 * 」
必須明確標示哪些來源允許存取。

以axios為例：需設定 withCredentials: true

```
axios.defaults.withCredentials = true
axios.get(API_SERVER + '/todos', { withCredentials: true });
axios.interceptors.request.use(function (config) {
        config.withCredentials = true;
        return config
},
```

內容擷取： [教學] CORS 是什麼? 如何設定 CORS?

延伸閱讀與參考資料

- [教學] CORS 是什麼? 如何設定 CORS?
- 簡單弄懂同源政策 (Same Origin Policy) 與跨網域 (CORS)
- 關於CORS (Cross-Origin Resource Sharing)跨來源共享

何謂CONTENT SECURITY POLICY (CSP)?

- CSP 是瀏覽器提供網站設定白名單的機制，網站可以告知瀏覽器，該網頁有哪些位置可以連、哪些位置不能連。（預防被 XSS 攻擊）
- CSP 的設定方式：
  1. HTTP Header 加入 Content-Security-Policy: {Policy指令}
     當有不符合安全政策的情況，瀏覽器就會提報錯誤， 並終止該行為執行。
  2. HTTP Header 加入Content-Security-Policy-Report-Only: {Policy指令}
     當有不符合安全政策的情況，瀏覽器就會提報錯誤， 但會繼續執行 。
     ※主要用於測試用，怕網站直接套上 CSP 導致功能不正常。
  3. HTML 加入 <meta>
     在 HTML <head>區塊加入<meta http-equiv= “Content-Security-Policy” content= “{Policy指令}” >
     當有不符合安全政策的情況，瀏覽器就會提報錯誤， 並終止該行為執行。
     ※此方式不支援 Report-Only 的方式

- CSP 建議在 server 端加 ，雖然前端可以利用 meta http-equiv給瀏覽器一些額外資訊， 但所有文件還是 prefer 在 Server 那邊加 HTTP header

CSP 指令 (DIRECTIVES)

- 使用格式

```
Response Headers Content-Security-Policy: {CSP 指令} {位置}; {CSP 指令} {位置} {..位置..} {位置};
```

- 範例：允許與自己同來源與trusted.com與trusted.com同subdoamin

```
Content-Security-Policy: default-src 'self' trusted.com *.trusted.com
```

- 常用的 CSP 指令
  參考網站

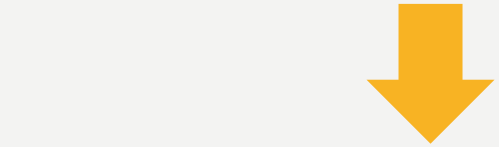
延伸閱讀與參考資料

- [Day27] ASP.NET Core 2 系列 - 網頁內容安全政策 (Content Security Policy)
- Content Security Policy (CSP) — 幫你網站列白名單吧
- MDN:Content Security Policy (CSP)

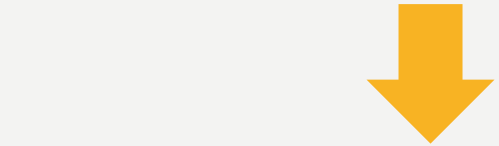
從輸入網址列到渲染畫面，過程經歷了什麼？

一般理解：

輸入網址



browser發送request給server

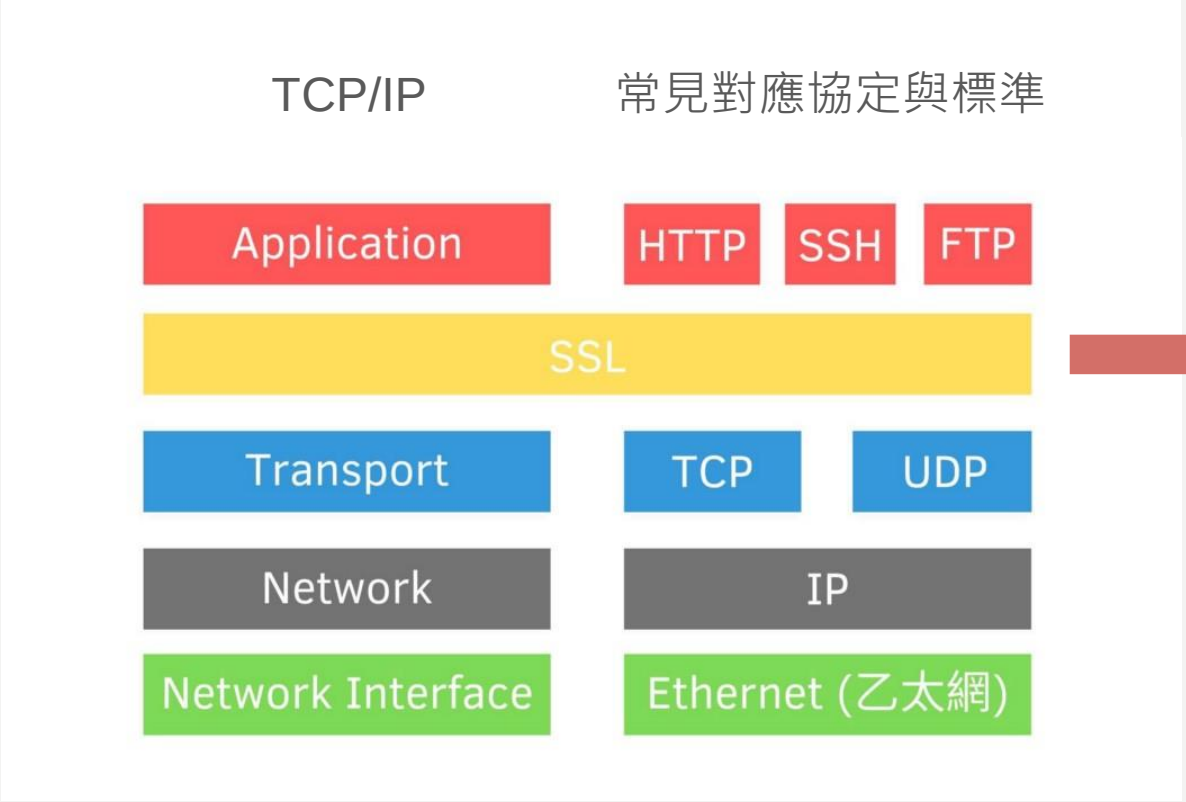


server回應response

實際流程

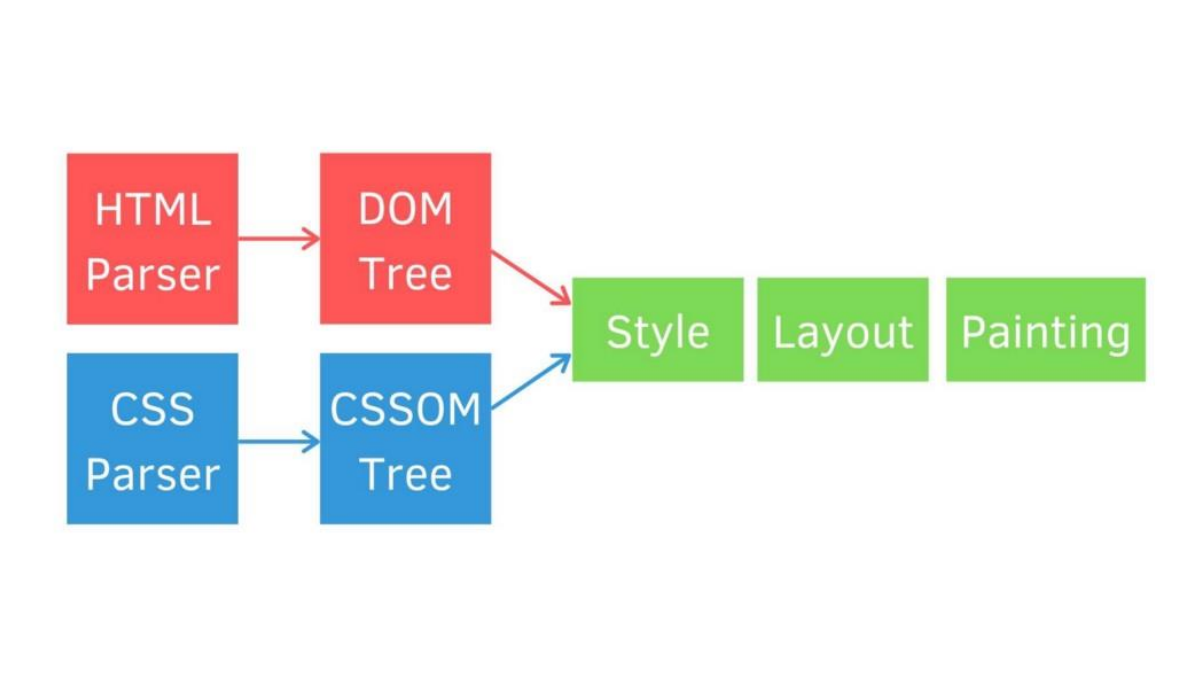
⚠ 事實上 browser 無法建立 HTTP 、 TCP 傳輸、從網址查找 IP，

這些是由 TCP/IP 傳輸協定做的而 TCP 包含四層(見下圖)



HTTPS 會在
Application 跟 Transport 中間
加了一層 SSL 加密憑證 (encryption)

那瀏覽器如何渲染出網頁?



HTML → DOM Tree & CSS
→ CSSOM Tress

結束會開始計算樣式該如何去套用到 HTML 上，
並產生 Render Tree

Layout 決定出每個元素在頁面上的位置

繪製在畫面上

⚠ HTML 一定會等到 CSS 解析完 (Parsing) 才會進行 render。
所以前端要避免讓 CSS 又肥又大包拖慢第一次 render 的速度

延伸閱讀與參考資料

- [熱門面試題] 從輸入網址列到渲染畫面，過程經歷了什麼事？