前言

過去，Web的主流做法是運用靜態HTML或者伺服器端搭建HTML的方式來顯示在瀏覽器上。然而進入2000年代後，幾乎所有的瀏覽器都改採了動態頁面的顯示方式。HTML、CSS、JavaScript的演進跟生態系的多樣化，還有框架的問世，都為前端開發帶來了日新月異的進步。

在這進步的推波助瀾下，雖帶動了開發各種功能以及UI設計的潮流，卻也同時令我們不得不正視前端設計面臨高度資安風險的事實。筆者雖非資安專業人士，然而作為一介投身以前端開發為主、在Web應用程式開發上深耕多年的開發者的拙見來看，斗膽認為縱使在工作上已經長期接觸資安相關因應策略，卻也感受到一般民眾**很難有系統地來學習資安知識**。資安相關的書籍、網路上可查找的資訊，大多在講解的時候都將前端跟伺服器端混為一談，如果您是前端工程師，想必應該曾經遭遇過不知從何學起才好的困境吧？

本書旨在將焦點集中在有關前端設計的資安議題，並透過圖片、程式碼來進行講解。而針對資訊漏洞的機制與因應方式則是為了能讓各位讀者透過來學會。

由衷期待閱讀了本書的讀者，都能身為前端工程師跟Web工程師，學習到**避免出現漏洞的必備知識**。而雖然本書難以納入所有的資安知識，亦希望作為一本啟發前端工程師學習資安的契機，為開發確保資安品質的Web應用程式盡上一份心力。

本書適用對象

作為一本學習前端開發第一線的資安知識與對策入門書，特別想要協助那些日常當中擔任Web應用程式開發、並期望能夠更充實資安對策的工程師。

- **工作資歷在3年以內的前端工程師**
- **想開始學習Web資安的Web工程師**
- **想透過實作來學習資安漏洞機制的讀者**

相反地，本書可能不適合以下讀者。

- **想學習伺服器端的資安的讀者**
- **想學習通訊技術跟加密技術的機制等應用程式層面以外的範圍的讀者**
- **想知道Web標準跟技術規範的讀者**

此外，即便您已經學過資安，也可透過練習書中所提供的來複習如何因應資安漏洞。加上我們也探討了較新的議題，相信也能幫助讀者了解更多新資訊。由於本書是以專門講解用於開發第一線所需知識，因此刻意省略了通訊技術跟加密技術等原理上的解釋與技術規範。
書中依然會介紹延伸閱讀時的推薦書籍，所以讀者仍可透過其他書籍來獲取本書所未提及的資訊。

本書架構

本書分為多個章節來講述前端開發相關漏洞與資安風險機制及措施。從第3章到第7章也加入了用來複習講解內容的實作。

第1章
講解資安的必要性與近年來的趨勢。

第2章
建構進入第3章後運行程式所需的開發環境、使用Node.js搭建HTTP伺服器。

第3章
講解「HTTP」基本知識與「HTTPS」的機制及必要性。

● 第4章

講解Web資安基礎「同源政策」跟「跨來源資源共用」。

● 第5章

講解「跨站腳本攻擊」(cross-site scripting，XSS)。XSS是使用瀏覽器上執行的JavaScript的漏洞，跟前端的關係最為密切，因此本章在書中也佔據了最多的篇幅。

● 第6章

講解XSS之外的被動式攻擊：「跨站請求偽造」(cross-site request forgeries，CSRF)、「點擊劫持」(clickjacking)、「開放重定向」(open redirect)。

● 第7章

以Web應用程式不可或缺的登入功能為主，講解「驗證(authentication)」與「授權(Authorization)」。

● 第8章

講解運用JavaScript函式庫時會遭遇的風險、以及如何降低風險。

最後在附錄當中，會再次地講解讀完本書之後的學習方法、以及如何將Web應用程式改以HTTPS來進行傳輸。

檔案下載

請從下方網址下載範例程式碼「Hands-on sample code」

`https://github.com/shisama/security-handson`

中文版的「安全性檢查表」請於以下網址下載

`http://books.gotop.com.tw/download/ACL069400`

第 3 章

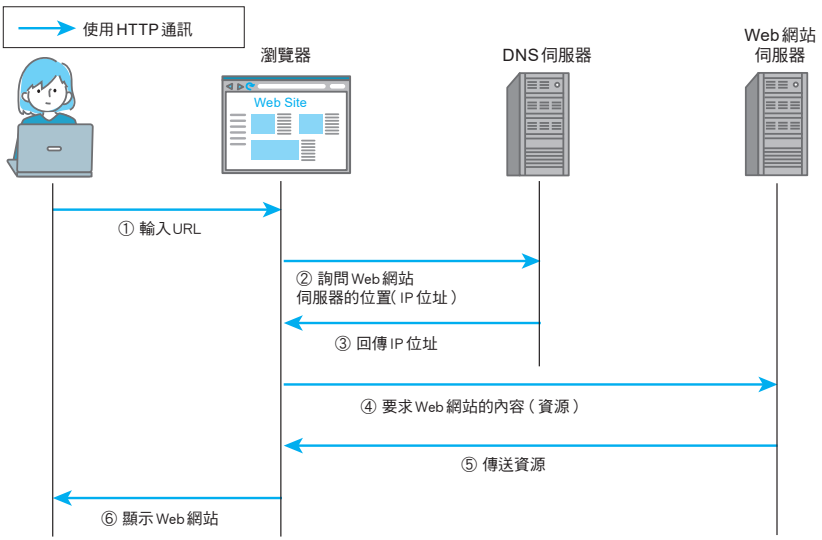
HTTP

第 3 章將介紹「HTTP」這個通訊協定（規範）。網頁瀏覽器依據與伺服器的通訊所獲取的資料，來顯示 Web 應用程式。如果通訊過程存在資安問題，將影響顯示和作動，並可能發生資訊外洩等意外。就算對通訊後的顯示和使用者操作盡可能做好資安措施，如果在通訊階段就出現問題，那些措施也可能會變得毫無意義。HTTP 知識是理解後續的攻擊技巧和對策的基礎。期待各位確實掌握 HTTP 基礎知識和安全通訊方法，並使用第 2 章創建的 HTTP 伺服器上實際演練來學會 HTTP 有哪些功能。

Section

3.1 HTTP 的基礎

Web 應用程式是由伺服器所下發的 HTML 跟、CSS 或圖像等被稱之為「資源（resouce）」的資料所構成。瀏覽器需要遵守 **HTTP** 通訊協定來跟伺服器進行通訊，以取得資源、執行建立、更新、刪除等處理（圖 3-1）。



▶ 圖 3-1 瀏覽器的通訊整體流程

瀏覽器為了要能夠在網際網路上找到 Web 應用程式伺服器，因此需要 **URL** 與 **DNS** 的機制。此外，DNS 跟 HTTP 則是使用了 **TCP/IP** 的機制。接著就來依序介紹「URL」、「DNS」、「TCP/IP」、「HTTP」。

注意

使用者用來與伺服器通訊的軟體、電腦設備稱為「用戶端（Client）」，這個名稱包含了Web瀏覽器、智慧型手機APP、IoT物聯網設備等，相當多元。前述的通訊流程並不限於瀏覽器，其他的用戶端（Client）也是透過相同流程來進行通訊。不過，由於本書以Web前端設計為主，因此如果有提到用戶端（Client）時皆是指瀏覽器。

3.1.1 URL

用來顯示網際網路上資源所在位置的字串，叫做 **URL**（Uniform Resource Locator），而瀏覽器就是透過 URL 去找到下發資源的伺服器、進行通訊。

URL 的結構如下（圖 3-2）。

▶ 圖 3-2 URL 的格式

● 通訊協定名稱（Scheme）

指出用來存取資源的通訊協定（Protocol），相關內容稍後會再提及。

● 主機名稱（Host）

指出伺服器的所在位置。

● 埠號（Port）

用於辨別伺服器內的服務的編號。有的伺服器是 Web 應用程式、有的是郵件伺服器，透過分配給各個服務不同的埠號，而得以提供多個服務。通常都會省略每個服務最常用的預設埠號。比方說，HTTP 的預設埠號是 80，因此就會將 http://site.example:80 的「:80」省略掉。

● 路徑名稱（Path）

指出伺服器內資源的所在位置。以圖 3-2 為例，就是存取 example.com 伺服器當中的 `/path/to/index.html` 資源。URL 相關規範都有定義在「URL Standard」[*3-1] 內，有興趣深入了解的讀者可以參閱。

※3-1 https://url.spec.whatwg.org/

3.1.2 DNS

接著講解 **DNS**（Domain Name System）如何透過 URL 來與伺服器連線。當使用者存取 Web 應用程式時，瀏覽器得先從 DNS 伺服器去找到與 URL 綁定的伺服器位置。

連線到網際網路上的所有機器都會被賦予「IP 位址」，這可以說是電腦的住址。一如我們會寫信寄到某人家一樣，電腦則是使用 IP 位址來將資料傳送到特定的電腦。IP 位址的樣子會像是 `192.0.2.0/24`，相當不好記，因此會轉換為容易記得住的主機名稱來進行運用。

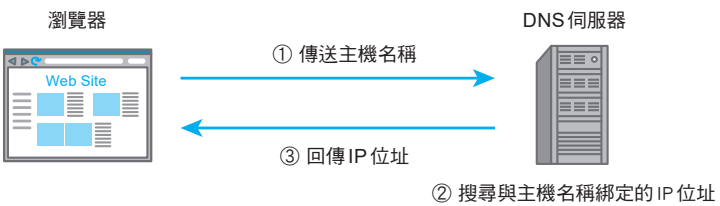
「主機名稱」這個詞有時候是指包含了網域名稱在內的「FQDN」（完整網域名稱，Fully Qualified Domain Name），但有時候則是不含網域名稱、單指 Hostname（圖 3-3）。本書所提到的所有主機名稱都是指 FQDN。

▶ 圖 3-3　完整網域名稱（FQDN）

瀏覽器要將 URL 當中的主機名稱轉換為 IP 位址才能連線到伺服器，但由於瀏覽器並不知道與主機名稱綁定的 IP 位址為何，因此才需要 DNS。

DNS 機制可以讓我們從主機名稱得知 IP 位址，宛如現實世界裡的通訊錄。我們在通訊錄裡去找到聯絡人、查看他的電話號碼，而 DNS 則是從主機名稱去找到 IP 位址。

搜尋 IP 位址的動作會在 DNS 伺服器內部進行，而瀏覽器會將主機名稱傳送給 DNS 伺服器的方式來取得 IP 位址、並且依據 DNS 伺服器所提供的 IP 位址來與伺服器連線，要求取得資源（圖 3-4）。



▶ 圖 3-4　向 DNS 伺服器要求 IP 位址

3.1.3 TCP/IP

電腦必須得要依照訂定的流程來進行通訊，方能將資料傳送給對方。倘若傳送資料的流程有誤，接收端就會不知道該如何接收資料。雙方都必須依照既定事項（規範）來互動，這就是**通訊協定**。

本章主題的「HTTP」也是一種通訊協定，而通訊協定的規範則是由名為 IETF 的標準化團體的 RFC 文件進行管理，每個文件都有著如「RFC 7231」的編號列管。這些編號是依據發布順序的流水號，迄今為止，已經發布了 9,000 份以上的文件。

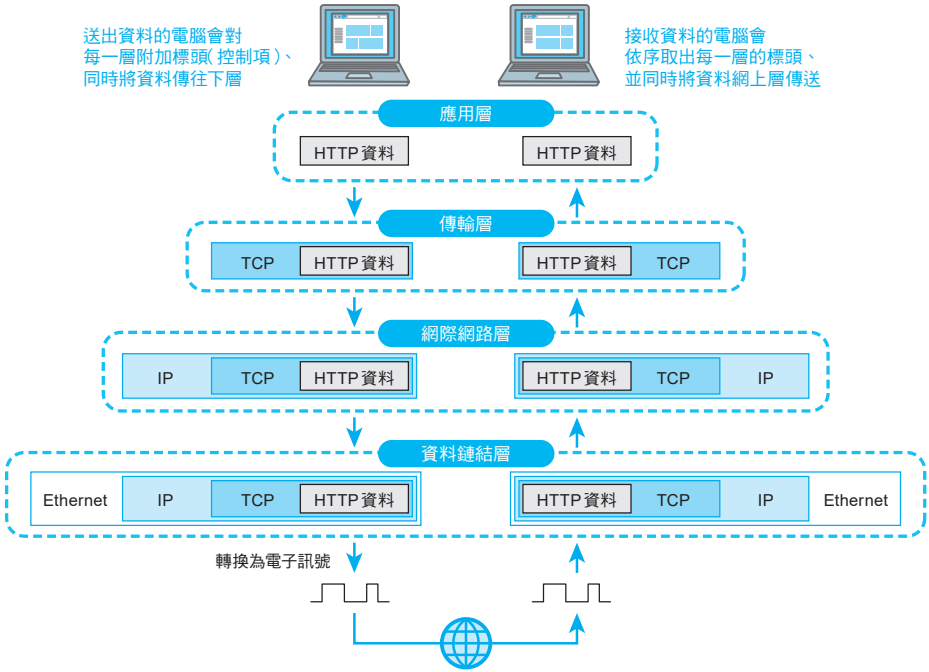
包含 TCP 與 IP 在內的通訊協定，統稱為 **TCP/IP**，且分為四個階層。

▶ 表 3-1 TCP/IP 的四個階層

階層	職責	主要的協議
應用層（Application Layer，Layer 4）	執行通訊服務作業。	HTTP：傳送與接收 Web 資料 SMTP：轉寄郵件
傳輸層（Transport Layer，Layer 3）	把從網際網路層傳過來的資料根據不同的用途提供給不同的應用層、偵測資料是否有誤等。	TCP：用於想要確實地將傳送的資料送達對方時 UDP：用於在乎即時通訊速度時
網際網路層（Internet Layer，Layer 2）	決定該將資料傳給哪台電腦。	IP：使用 IP 位址來確定要將資料傳給誰，選擇遞交資料的路由（Routing）。
資料鏈結層（Data-Link Layer，Layer 1）	由於無法直接將文字跟數字傳送給通訊裝置，所以需要轉換為電報傳輸。資料鏈結層會透過電報傳輸來傳送資料、檢測電控有無錯誤等。	Ethernet：有線網路 IEEE 802.11：無線網路

上層的通訊協定接收下層送上來的資料、並採取行動，各式各樣的應用層的通訊協定會在 TCP 之上運作。HTTP/1.1 跟 HTTP/2 是在 TCP 之上來運作，而 HTTP/3 則是在 UCP 上方來運作。另外，TCP 跟 UCP 則是在 IP 上方運作。

如圖 3-5 所示，傳送端為每一層添加標頭（Header）、同時將資料傳給下層的通訊協定，然後接收端會從每一層取出標頭、同時將資料傳到上層的通訊協定。



▶ 圖 3-5　TCP/IP 階層

3.1.4 HTTP 訊息

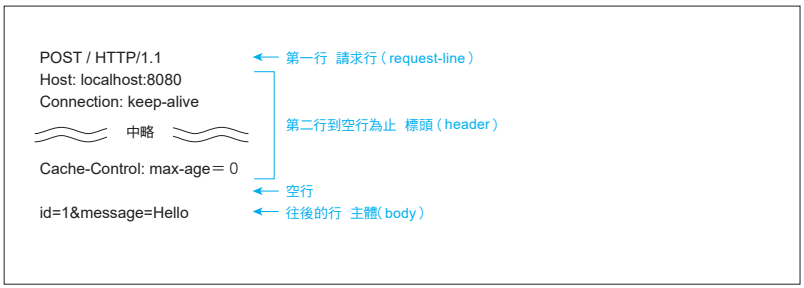
在 HTTP 當中，瀏覽器與伺服器是以既定的**HTTP 訊息**格式來傳送與接收資料（圖 3-6）。HTTP 有好幾種版本，本書會以 1.1 版（寫作 HTTP/1.1）來講解。

▶ 圖 3-6　HTTP 訊息的組成

HTTP 訊息分為「請求（HTTP request）」跟「回應（HTTP response）」，格式相同、內容不同。

● 請求（HTTP request）

與瀏覽器及伺服器彼此之間的 HTTP 通訊，開始於瀏覽器傳送請求給伺服器。瀏覽器使用 HTTP 將請求傳送給伺服器叫做 **HTTP request**（以下稱為請求）。請求的 HTTP 訊息是由請求行（request-line）、標頭（header）、主體（body）所組成（圖 3-7）。

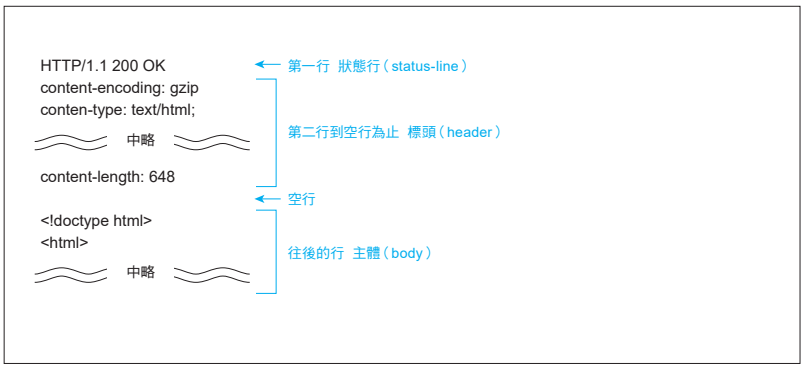


▶ 圖 3-7 HTTP request 的訊息

請求行包含了 GET 跟 POST 等 HTTP 方法（method）、欲請求資源的路徑名稱和 HTTP 版本。標頭則涵蓋與瀏覽器資訊或連線資訊相關的資料傳輸所需要的附加資訊。主體是請求的正文，這裡描述了想要取得的資訊跟關鍵字、或是打算註冊的資訊。根據某些請求的內容，其主體可能會是空值。

● 回應（HTTP response）

收到來自瀏覽器的請求後，伺服器所傳送的資訊稱為 **HTTP response**（以下稱為回應）。回應的 HTTP 訊息是由狀態行（status-line）、標頭（header）、主體（body）所組成（圖 3-8）。



▶ 圖 3-8 HTTP response 的訊息

狀態行會以 3 位數的數字來顯示伺服器依據請求所處理的結果，處理成功為 **200**、找不到請求對象的資源時則使用 **404** 來顯示錯誤。標頭包含了伺服器相關的資訊或傳送的資源的格式等附加資訊。主體則是回應的正文，記載了瀏覽器請求的資訊跟伺服器的處理結果。根據某些回應的內容，其主體可能會是空值。

下一節要跟各位詳細介紹組成 HTTP 訊息的「HTTP 方法（method）」、「狀態碼（status-code）」跟「HTTP 標頭（header）」。

3.1.5 HTTP 方法（method）

HTTP 方法的職責是告訴伺服器要對資源進行什麼樣的處理。比方說，請求行 **GET /index.html HTTP/1.1** 的意思就是請求取得（GET）**/index.html** 這個資源。在 HTTP/1.1 的規範（RFC 7231）裡定義了 8 個 HTTP 方法（表 3-2）。

▶ 表 3-2　RFC 7231 所定義的 HTTP 方法

名稱	簡介
GET	取得資源
HEAD	取得 HTTP 標頭。回應中不含主體
POST	註冊資料、建立資源
PUT	更新資源。當想更新的資源不存在時則新建資源
DELETE	刪除資源
CONNECT	在 HTTP 上預留給能夠將連線改為其他通訊協定的方式。主要用於使用 HTTP 代理伺服器進行通訊時
OPTIONS	確認通訊支援。也用於事先確認不同的 Web 應用程式之間是否可通訊（詳見第 4 章）
TRACE	伺服器直接回傳收到的資料。用於診斷或測試瀏覽器與伺服器之間的通訊路徑

請容筆者補充跟資安有關的部分。

GET、HEAD、OPTIONS、TRACE 是用來新建、更新、刪除資訊，這表示不會引發影響到伺服器資源的副作用，因此在 RFC 7231 規範中被視為是安全的 HTTP 方法。

相反地，POST、PUT、DELETE 則是會產生副作用的方法，一旦誤用將會導致伺服器狀態或資源遭受影響。

CONNECT 是為了讓位於請求端與回應端之間的代理伺服器看不見的加密資料時，可以直接通過的方法。使用 HTTP 通訊時，代理伺服器會從通訊內容來判斷回應端。但是，當資料被加密時，HTTP 通訊就無法知道回應端在哪，這部分後面會再提到。此時使用 CONNECT，就可以讓資料直接通過代理伺服器，其作用有點像是隧道，它是透過代理伺服器來執行 HTTPS 通訊時所需要的方法，但仍需留意如果沒有限制回應端的話，還是有可能會遭到惡意攻擊。

TRACE 已幾乎無人使用，由於它有可能遭受跨站請求追蹤（Cross-Site Tracing，XST）的攻擊，幾乎所有瀏覽器都不支援 TRACE 了。

3.1.6 狀態碼（status-code）

狀態碼是回應的狀態行中用來顯示結果的 3 位數字，例如 `HTTP/1.1 200 OK` 的狀態碼就是 `200`。RFC 7231 當中定義了所有狀態碼的含義，只要看最前面的數字就能判斷是哪種類型的狀態，下面就跟各位分享最具代表性的狀態碼有哪些。

1xx

通知正在處理資訊。

- 100 Continue：通知瀏覽器， 伺服器處理尚未完成、請求仍在持續中。

2xx

通知處理成功的結果。

- 200 OK：請求順利完成
- 201 Created：順利完成新建資源

3xx

通知轉址（redirect）相關資訊。

- 301 Moved Permanently：將指定資源移至他處
- 302 Found：暫時移動指定資源。用於臨時需要維護伺服器時

4xx

通知瀏覽器的請求有問題時。

- 400 Bad Request：請求的資訊有誤
- 404 Not found：請求所指定的資源不存在

5xx

通知伺服器處理有問題時。

- 500 Internal Server Error：伺服器內部發生錯誤
- 503 Service Unavailable：通知伺服器當機、或者正在維護伺服器等暫時無法執行處理的狀態

Web 應用程式在遇到問題時，透過狀態碼可以幫忙儘速查明原因。例如當圖像資料的狀態碼是 `404`、無法顯示圖像時，就可以推測可能是請求的 URL 不正確、或是圖像已經遭到刪除了。

3.1.7 HTTP 標頭（header）

HTTP 標頭放的是訊息主體的附加資訊、或者是傳輸資料所需的資訊，會以下方格式被放入 HTTP 訊息中。

```
Host: example.com
```

欄位名稱　　欄位值

▶ 圖 3-9　HTTP 標頭格式

請求跟回應都可以使用 HTTP 標頭，接著就來介紹幾個代表性的請求標頭（表 3-3）。

▶ 表 3-3 代表性的請求標頭

欄位名稱	簡介
HOST	指定接收請求的的伺服器主機名稱與埠號。預設的埠號會省略掉。例如要向 **example.com** 發出請求時，就會是 **Host: example.com**
User-Agent	傳遞請求端的資訊，如瀏覽器版本、作業軟體版本等資訊。瀏覽器不同、所傳遞的值也不同。
Referer	將所存取的 Web 應用程式的 URL 告訴伺服器。例如從 https://site-a.example 的頁面上的連結訪問了 https://site-b.example 時，https://site-b.example 的請求標頭上就會被附上 **Referer: https://site-a.example/**。也會被用在分析從哪裡存取 Web 應用程式的場合。

也介紹幾個代表性的回應標頭（表 3-4）。

▶ 表 3-4 代表性的回應標頭

欄位名稱	簡介
Server	將回應中所使用的伺服器軟體相關資訊告知瀏覽器。例如：當伺服器使用了 nginx 時，就會是 **Server: nginx**
Location	指定轉址的 URL

也有些 HTTP 標頭是可用於請求與回應兩者，稱之為**實體標頭（Entity header）**。下面兩個是最具代表性的例子（表 3-5）。

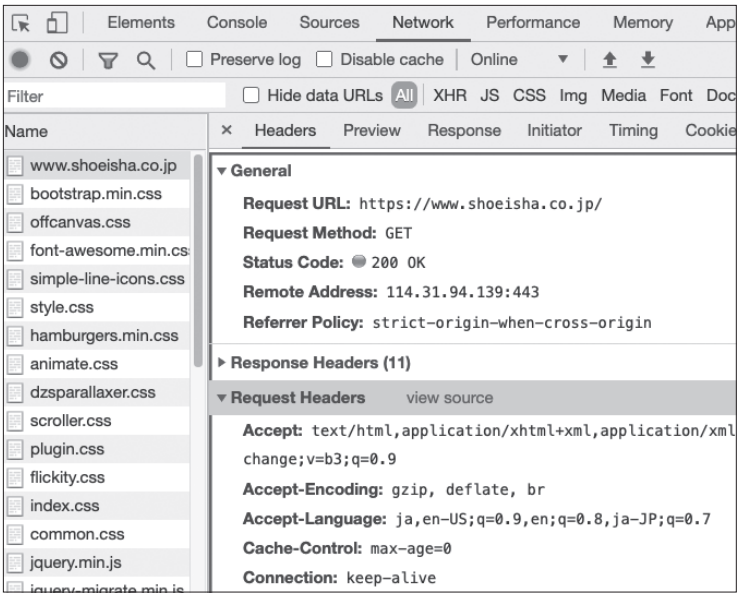
▶ 表 3-5 較有代表性的實體標頭

欄位名稱	簡介
Contene-Length	以 byte 為單位來表示資源的大小
Content-Type	資源的媒體類型。如 **Content-Type: text/html; charset=UTF-8** 就表示資源是使用 HTML、以 UTF-8 進行文字編碼。

● 使用開發者工具來確認 HTTP 標頭

使用瀏覽器的開發者工具，不僅可以查看 HTTP 標頭，連伺服器的位址（Request Address）、HTTP 方法（Request Method）、狀態碼（Status Code）這些先前介紹過的內容都能查看（圖 3-10）。這邊我們使用 Google Chrome，依照下列方法來演練。

1. 開啟 Network 面板
2. 重新載入頁面
3. 選擇任意的資源
4. 查看 Request Headers 跟 Response Headers



▶ 圖 3-10　查看 HTTP 標頭

以 HTTP 標頭為主的資安功能會在第 4 章的跨來源資源共用（cross-origin resource sharing，CORS）跟第 5 章的內容安全政策（content security policy，CSP）進行講解。雖說如果能正確運用 HTTP 標頭確實可以增強資安防護，但要是用錯方法也可能導致資安風險不減反增，因此使用時務必多加注意。

3.1.8 使用 Cookie 管理狀態

最早開發出 HTTP 是為了要能夠傳送文本，當時還不需要維持瀏覽器跟伺服器之間的狀態。但是，隨著 Web 普及，使用者希望在 HTTP 上面來傳輸資料，於是為了要固定瀏覽器與伺服器之間的狀態，創建了 **Cookie** 這個能將與伺服器往來的資訊儲存在瀏覽器內部的檔案的。

就像是當使用者登入過一次後，在該 Web 應用程式就能維持登入狀態。即便使用者已經去到其他頁面、或是關閉了瀏覽器，都會為了要能夠維持登入的狀態，而將登入資訊預先儲存在 Cookie。只要將登入資訊儲存在 Cookie，使用者就得以維持在登入狀態。

Cookie 會像下面這樣以「鍵：值」的格式來儲存資料。

```
SESSION_ID: 12345abcdef
```

若想將 Cookie 從伺服器儲存到瀏覽器時，可以在回應中加入 `Set-Cookie` 的標頭。

```
Set-Cookie: SESSION_ID=12345abcdef
```

當頁面切換時、或是傳送表單時，瀏覽器會自動將 Cookie 傳送給伺服器。也就是說開發人員毋需特地寫一段用來傳送 Cookie 的程式碼，就能很輕鬆地維持登入狀態了。