HTTP 基礎知識：網頁溝通的語言 🌐

🚀 HTTP 是什麼？

HTTP（HyperText Transfer Protocol）就像是網頁世界的「郵差」！

負責在瀏覽器和伺服器之間傳遞訊息，讓我們能夠看到網頁內容。

HTTP 的工作原理

想像一下網頁載入的過程：

1. **你**（用戶）：點擊連結或輸入網址
2. **瀏覽器**：發送 HTTP 請求給伺服器
3. **伺服器**：處理請求並回傳 HTTP 回應
4. **瀏覽器**：顯示網頁內容給你看

請求與回應的結構

HTTP 請求格式：

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 ...
Accept: text/html,application/xhtml+xml,...
```

HTTP 回應格式

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<html>
<head><title>網頁標題</title></head>
<body>網頁內容...</body>
</html>
```

🔗 TCP vs HTTP：連接方式差異

這裡是很多人容易混淆的地方！

TCP（傳輸控制協議）

- **有狀態連接**（Stateful）
- 建立連線後會保持「連線狀態」
- 就像電話接通後的持續對話
- 可靠傳輸，有確認機制
- 適用於需要持續溝通的應用

TCP 三次握手

```
# TCP 三次握手建立連線
1. 客戶端 → 伺服器: SYN (我想連線)
2. 伺服器 → 客戶端: SYN-ACK (好啊，我準備好了)
3. 客戶端 → 伺服器: ACK (連線建立成功！)
```

HTTP（超文字傳輸協議）

- **無狀態協議**（Stateless）
- 每個請求都是獨立的「交易」
- 就像寄信：寄出一封，收到回信，就結束了
- 沒有持續的連線狀態
- 每個請求都需要重新建立連線

Connectionless 的誤解澄清

很多人以為 HTTP 是「無連線」的，事實上是：

- **HTTP/1.0**: 每個請求確實是獨立的（真正無連線）
- **HTTP/1.1**: 支援連線複用（Keep-Alive），但仍是無狀態的
- **HTTP/2 與 HTTP/3**: 進一步優化連線效率

HTTP 的真實連線行為

```
# 實際上網頁載入時：
1. 建立 TCP 連線 (三次握手)
2. 發送 HTTP 請求 1 (首頁 HTML)
3. 收到 HTTP 回應 1
4. 發送 HTTP 請求 2 (CSS 檔案)
5. 收到 HTTP 回應 2
6. 發送 HTTP 請求 3 (JavaScript 檔案)
7. 收到 HTTP 回應 3
8. 關閉 TCP 連線
```

雖然每個 HTTP 請求是獨立的，但底層還是使用 TCP 連線！

🎯 HTTP 方法（HTTP Methods）

告訴伺服器「我想做什麼」的方式：

GET 方法

用途：請求資料（讀取）

```
GET /users/123 HTTP/1.1
```

GET 方法說明

- 用來取得資源
- 請求參數放在 URL 中
- 安全且可重複執行
- 例如：載入網頁、取得資料

POST 方法

用途：建立新資源（寫入）

```
POST /users HTTP/1.1
Content-Type: application/json

{
  "name": "張小明",
  "email": "user@example.com"
}
```

POST 方法說明

- 用來送出資料建立新內容
- 請求主體（body）包含資料
- 例如：註冊帳號、上傳檔案

PUT 方法

用途：更新完整資源（覆蓋更新）

```
PUT /users/123 HTTP/1.1
Content-Type: application/json

{
  "name": "李小華",
  "email": "newemail@example.com"
}
```

PUT 方法說明

- 完整替換資源內容
- 如果資源不存在會建立新資源

PATCH 方法

用途：部分更新資源（差異更新）

```
PATCH /users/123 HTTP/1.1
Content-Type: application/json

{
  "email": "newemail@example.com"
}
```

PATCH 方法說明

- 只更新指定的欄位
- 比 PUT 更有效率

DELETE 方法

用途：刪除資源

```
DELETE /users/123 HTTP/1.1
```

DELETE 方法說明

- 刪除指定的資源
- 成功後回傳 204 No Content

其他重要方法

方法	用途	安全	可快取
HEAD	取得資源的 header（不含主體）	✅	✅
OPTIONS	查詢伺服器支援的方法	✅	❌
TRACE	診斷請求路徑	✅	❌

HTTP 狀態碼（Status Codes）

伺服器回應「請求結果」的數字代碼：

2xx 成功狀態碼

200 OK - 請求成功！

```
HTTP/1.1 200 OK
Content-Type: text/html

網頁內容...
```

201 Created

201 Created - 資源建立成功

```
HTTP/1.1 201 Created
Location: /users/456

新用戶已建立！
```

204 No Content

204 No Content - 請求成功但無內容回傳

3xx 重新導向狀態碼

301 Moved Permanently - 永久重新導向

```
HTTP/1.1 301 Moved Permanently
Location: https://new-domain.com/
```

302 Found

302 Found - 暫時重新導向

304 Not Modified

304 Not Modified - 資源未修改（快取有效）

4xx 用戶端錯誤狀態碼

400 Bad Request - 請求語法錯誤

```
HTTP/1.1 400 Bad Request

請求格式不正確，請檢查語法
```

401 Unauthorized - 需要認證

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Secure Area"
```

403 Forbidden

403 Forbidden - 禁止存取

404 Not Found

404 Not Found - 資源不存在

5xx 伺服器錯誤狀態碼

500 Internal Server Error - 伺服器內部錯誤

502 Bad Gateway

502 Bad Gateway - 閘道錯誤

503 Service Unavailable

503 Service Unavailable - 服務暫時無法使用

☕ 特別的狀態碼：418 I'm a teapot

412 Precondition Failed 不是 teapot！

正確的 teapot 狀態碼是 **418**！

418 I'm a teapot 的故事

這是 1998 年的惡作劇狀態碼！

來自於一個有趣的 RFC 規格：

> "Any attempt to brew coffee with a teapot should result in the error code '418 I'm a teapot'."

```
HTTP/1.1 418 I'm a teapot

☕ 抱歉，我是茶壺，不是咖啡機！
```

雖然是個玩笑，但有些伺服器真的會回傳這個狀態碼！

🔒 HTTPS：安全的 HTTP

HTTP 透過 TLS/SSL 加密後就變成 HTTPS：

- **HTTP**: 明文傳輸，可能被竊聽
- **HTTPS**: 加密傳輸，保護隱私

HTTP vs HTTPS 範例

```
# 一般網站
http://example.com

# 安全網站
https://example.com
```

HTTP 版本演進

版本	特色	推出年份
HTTP/1.0	每個請求獨立連線	1996
HTTP/1.1	連線複用、壓縮支援	1997
HTTP/2	多工、二進位格式、伺服器推送	2015
HTTP/3	基於 QUIC 協議	2022

🎮 HTTP 動手玩

試試看這些指令：

使用 curl 發送請求

```
# 使用 curl 發送請求
curl -I https://httpbin.org/get           # HEAD 請求
curl -X POST https://httpbin.org/post     # POST 請求
curl -X PUT https://httpbin.org/put       # PUT 請求
curl -X DELETE https://httpbin.org/delete # DELETE 請求
```

查看狀態碼

```
# 查看狀態碼
curl -w "%{http_code}" https://httpbin.org/status/200
curl -w "%{http_code}" https://httpbin.org/status/404
```

🔍 開發時的檢查清單

- [ ] HTTP 方法選擇正確嗎？
- [ ] 狀態碼處理是否完整？
- [ ] 使用 HTTPS 而非 HTTP？
- [ ] 錯誤處理機制是否健全？
- [ ] 請求逾時設定是否合理？

📚 延伸學習資源

- MDN HTTP 指南
- HTTP Status Code 查詢
- REST API 設計指南
- HTTP/3 說明

恭喜你完成了 HTTP 基礎學習！

現在你能夠理解網頁溝通的機制，並且不會再混淆 TCP 和 HTTP 的差異了！

記住：HTTP 是應用層協議，建立在 TCP 連線之上，但每個請求仍是獨立的「交易」。