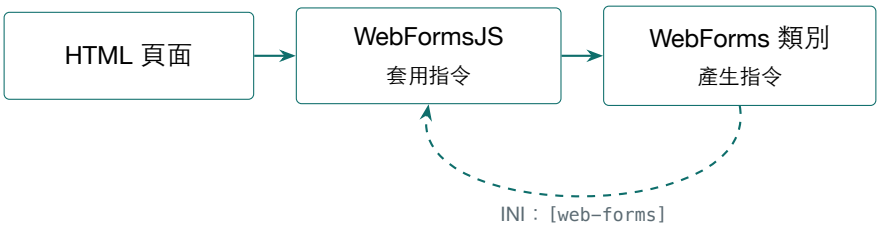
WebForms Core .NET

圖解學習手冊

伺服器下指令，瀏覽器改畫面



本書怎麼用

這不是某一家產品的官方手冊複本。它把 2026 年 10 月還能公開讀到的資料，整理成一條可以跟著做的學習路徑。

主線是 Elanat 的 **WebForms Core**：伺服器上的 `WebForms` 類別寫出 Action Control，瀏覽器裡的 WebFormsJS 照著改 HTML。在.NET 上，最完整的載體是 CodeBehind；同一套類別也能放進 Razor Pages，或只搭配一個靜態 HTML。

同名的另外兩條路放在最後兩章，而且會先講清楚它們不是同一件事：

- **WebFormsForCore**：把.NET Framework 的 `System.Web` / Web Forms 移植到.NET 10，讓舊的 `.aspx` 繼續跑，包含 Linux。
- **WebFormsCore**：另一個以 Web Forms 為靈感、針對 ASP.NET Core 10 重寫的頁面框架，公開 README 仍在早期。

整理基準：2026-10-16

CodeBehind 4.8.1 · .NET 10

WebFormsJS Action Control 公開說明

WebFormsForCore 1.6.8

怎麼讀這本手冊

你如果寫過 ASP.NET Web Forms，會覺得名詞很熟：頁面、回傳、控制項、Code-Behind。熟的是感覺，不是實作。WebForms Core 沒有 `ViewState`，也不把整頁重新畫出來。伺服器只回一小段文字，告訴瀏覽器「把這個標籤的字改掉、把那個按鈕關掉」。

這一節要記住

讀完第 1 章以前，先記住一句話：**WebForms Core** 是一套通訊協定，加上兩邊的實作。一邊是任何語言都能寫的 `WebForms` 類別，一邊是瀏覽器裡的 `web-forms.js`。C# 是這套類別的母本，所以這本手冊以.NET 為主。

建議順序

1. 第 1–2 章先把名字和歷史放對位置。跳過這兩章，後面的 `PostBack` 會和舊版 Web Forms 的 PostBack 纏在一起。
2. 第 3–5 章把一條請求走完：點按鈕、表單怎麼送、伺服器回什麼、畫面哪裡變。
3. 第 6–8 章是工具箱：怎麼指到正確的標籤、一行 Action Control 怎麼讀、C# 方法怎麼對上那一行。
4. 第 9–11 章拿來做：一般欄位、檔案、延遲刪除、瀏覽器上一頁，最後做一頁聯絡表單。
5. 第 12 章才看「把舊專案搬上.NET」和另一個同名框架。那是遷移手冊，不是這條主線的前置知識。

版本與界線

公開頁面會改。這本手冊鎖定下面這些可以核對的來源，不把部落格裡的行銷句當成 API。

項目	這本手冊採用的依據
CodeBehind	NuGet `CodeBehind` 4.8.1，目標框架 `net10.0`，頁面上的 `Program.cs` 與 MVC 範例
WebForms Core 教學	Elanat 文件〈Using WebForms Core in CodeBehind Framework〉
Action Control	`elanatframework/Web_forms` 的 README：代碼表、定位語法、`PostBack Options`、多重回應
類別方法	CodeBehind 教學裡出現的 C# 呼叫，並以各語言類別的母本行為交叉對照
WebFormsForCore	官方站 2026-10 的移植步驟，套件版本 `1.6.8`

先停一下

Go 語言移植的 `AssignDelay` 簽章，和 CodeBehind 教學裡的單參數呼叫不完全一樣。手冊遇到這種落差會寫出來，不會假裝只有一種多載。你專案裡的 IntelliSense 優先於這本書。

程式範例刻意只用 ASCII。中文解釋放在程式外面，這樣用 XeLaTeX 排版時，程式字體不必承擔中日韓字形。你自己的頁面當然可以用中文標籤與中文訊息。

目录

怎麼讀這本手冊 **ii**

第一章　三個容易搜到的名字 **1**
- 1.1　Elanat 的 WebForms Core . . . 1
- 1.2　WebFormsForCore：讓舊站在.NET 10 上跑 . . . 1
- 1.3　WebFormsCore：受 Web Forms 啟發的新框架 . . . 1
- 1.4　對照 . . . 2

第二章　舊版 **Web Forms** 留下的直覺 **3**
- 2.1　一頁應用程式，長在伺服器上 . . . 3
- 2.2　它沒有跟著進 ASP.NET Core . . . 3
- 2.3　WebForms Core 保留了哪一種感覺 . . . 3
- 2.4　它比較像什麼 . . . 4

第三章　一條請求長什麼樣子 **5**
- 3.1　四個角色 . . . 5
- 3.2　Action Control 的形狀 . . . 5
- 3.3　兩種回應，兩種包裝 . . . 6
- 3.4　CodeBehind 在這張圖的哪一層 . . . 6

第四章　把第一個專案跑起來 **8**
- 4.1　你需要的工具 . . . 8
- 4.2　View 與 Controller 各放什麼 . . . 9
- 4.3　最小的一頁 . . . 9
- 4.4　不使用 CodeBehind 的時候 . . . 10

第五章　走完官方那個歡迎頁 **11**
- 5.1　控制器 . . . 11
- 5.2　瀏覽器送出什麼 . . . 12
- 5.3　伺服器回什麼 . . . 12
- 5.4　三種把請求送回去的函式 . . . 12
- 5.5　用戶端可以調的開關 . . . 13

第六章　指到正確的那一個節點 **15**
- 6.1　一棵拿來指的樹 . . . 15
- 6.2　六種寫法 . . . 15
- 6.3　路徑，不只是一個節點 . . . 16
- 6.4　一組會踩到的情況 . . . 16

第七章 **Action Control** 代碼 **17**
7.1 一行裡面還能有前置 . . . 17
7.2 加、設、插入、刪 . . . 17
7.3 外觀、尺寸與表單狀態 . . . 18
7.4 增減 . . . 18
7.5 事件 . . . 18
7.6 先存起來，等一下再用 . . . 19
7.7 多段回應 . . . 20

第八章 **C#** 類別上你會呼叫的方法 **21**
8.1 產生正文 . . . 21
8.2 和歡迎頁同一族的方法 . . . 21
8.3 表單控制項 . . . 22
8.4 延遲是改寫已經寫好的那一行 . . . 22
8.5 網址、重新載入、標題 . . . 22

第九章 表單、檔案與頁面上的訊息 **24**
9.1 讀取時只信 name . . . 24
9.2 訊息寫在頁面上，而不是寫進記錄 . . . 24
9.3 檔案 . . . 25
9.4 第一次載入就綁好事件 . . . 25

第十章 沒有 **ViewState** 時，狀態放哪裡 **27**
10.1 三種放法 . . . 27
10.2 用查詢字串推進歷史 . . . 27
10.3 快取是用戶端的捷徑 . . . 28
10.4 重新整理之後什麼還在 . . . 28

第十一章 練習：一頁聯絡表單 **29**
11.1 頁面 . . . 29
11.2 控制器的判斷 . . . 29
11.3 你要在網路面板裡看到的 . . . 30
11.4 再改三處，才算做完 . . . 31

第十二章 另外兩條同名的路 **32**
12.1 什麼時候不該用 WebForms Core . . . 32
12.2 WebFormsForCore：舊站遷到.NET 10 . . . 32
12.2.1 套件對照 . . . 33
12.2.2 組件衝突 . . . 34
12.3 WebFormsCore 框架 . . . 34

代碼速查 **35**

公開來源 **36**

讀完之後 **39**

第一章 三個容易搜到的名字

搜尋「WebForm Core .NET」會同時掉進三個專案。它們都提到 Web Forms，也都跑在新的.NET 上，解決的問題卻不一樣。先把地圖畫出來，後面才不會把 A 的安裝步驟套到 B 的頁面上。

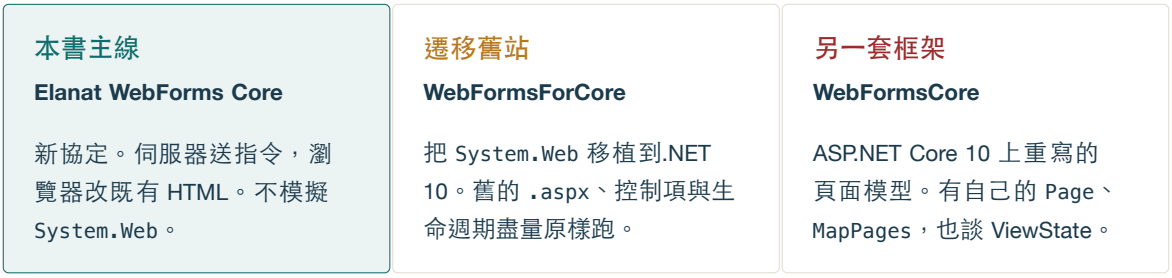


圖 1.1: 同一次搜尋裡的三個專案。卡片高度只是示意，中間那張與右邊那張的公開成熟度並不相同。

1.1 Elanat 的 WebForms Core

Elanat 把「伺服器上的 `WebForms` 類別」和「瀏覽器上的 WebFormsJS」之間的這段通訊，命名為 WebForms Core。官方教學寫得很直接：用戶端由 WebFormsJS 管理，伺服器端的 `WebForms` 類別把 Action Control 送給它。

它被包在 CodeBehind 裡，但協定本身不綁定 CodeBehind。GitHub 上的 `Web_forms_classes` 說明，各語言的類別都從 C# 類別衍生，NuGet 上也有獨立套件 `WFC`。所以你可以：

- 在 CodeBehind 的 `.aspx` 與 Controller 裡呼叫 `new WebForms()`；
- 在 ASP.NET Core Razor Pages 裡把指令寫進回應或 HTML 註解；
- 只用一個靜態 HTML，加上 `web-forms.js`，由任何後端回傳以 `[web-forms]` 開頭的文字。

畫面上的 HTML 保持你寫的那樣。函式庫不會為了自己再生成一套控制項標籤。官方把這點稱為對 HTML 忠實：除了在 `<head>` 引入腳本，頁面不必為了這套技術改寫結構。

1.2 WebFormsForCore：讓舊站在.NET 10 上跑

WebFormsForCore 是另一個開源專案，目標寫在首頁第一段：把.NET Framework 的 `System.Web` 程式庫移植到.NET 10，讓既有 Web Forms 網站直接在 ASP.NET Core 裡跑，也包含 Linux。作者用它把 FuseCP 控制台遷上去，動機是不想把整套畫面改寫成 Blazor。

你會在這裡看到 `app.UseWebForms()`、`.aspx`、Session、`*.designer.cs`。這些都是舊生命週期的延續。它不使用 `[web-forms]` 這份 INI 協定。

1.3 WebFormsCore：受 Web Forms 啟發的新框架

GitHub 組織 `WebFormsCore/WebFormsCore` 的公開說明把它定義成 ASP.NET Core 10 以上的框架，靈感來自 Web Forms，但不是直接移植，而且有許多破壞性變更。模板指令是：

```
dotnet new install WebFormsCore.Templates
dotnet new wfc -n MyWebFormsApp
```

註冊方式是 `AddWebFormsCore`、`UseWebFormsCore`、`MapPages`。它討論 `[ViewState]`、一頁多表單、範圍表單。這些詞彙更接近經典 Web Forms，而不是 Elanat 的 Action Control。

先停一下

這個儲存庫的公開頁面很短，星標也少。第 12 章只根據 README 裡已經寫出來的安裝與頁面宣告來介紹，不補上文件沒有的生命週期細節。

1.4 對照

	WebForms Core	WebFormsForCore	WebFormsCore
想解決的事	用伺服器指令更新 HTML	舊 Web Forms 繼續跑	用熟悉的頁面模型寫新站
畫面技術	你自己的 HTML + WebFormsJS	伺服器控制項、`.aspx`	`.aspx` 與控制項樹
典型回應	以 `[web-forms]` 開頭的文字	整頁 HTML 或局部更新	框架自己的頁面輸出
ViewState	官方教學明確說不使用	沿用 Web Forms 模型	README 提供 `[ViewState]`
和 ASP.NET Core 的關係	Middleware 之上的協定	取代 `System.Web` 的移植層	自己的 endpoint 對應

表 1.1: 三個專案不要互換安裝步驟。

微軟官方的遷移文件走的是第四條路，而且不叫 WebForms Core：把 Web Forms 改寫到 Blazor、Razor Pages 或 MVC，或以 YARP 並存、逐頁替換。那條路解決的是「離開 `System.Web`」。本書主線解決的是「還想用伺服器來驅動畫面，但不要 ViewState 與整頁重繪」。

這一節要記住

後面若沒有特別標出專案全名，「WebForms Core」都指 Elanat 的協定與 `WebForms` 類別。移植舊站時會寫全名 WebFormsForCore。

第二章 舊版 Web Forms 留下的直覺

WebForms Core 的命名是故意的。CodeBehind 的介紹寫著，結構和命名會讓人想起以前的 Microsoft Web Forms。官方教學的結論也說，功能感覺相近，實作是另一套更有彈性的做法。要學會新的那套，得先知道舊的直覺從哪裡來，又是哪裡斷掉。

2.1 一頁應用程式，長在伺服器上

經典 ASP.NET Web Forms 把網頁做成 `System.Web.UI.Page`。你在設計工具裡放 `TextBox`、`Button`、`GridView`，框架產生 `.aspx` 與 `.aspx.designer.cs`。按鈕有 `Click` 事件，事件處理常式寫在 code-behind。對許多人來說，這和 WinForms 幾乎是同一種心智模型：控制項、屬性、事件。

使用者按下按鈕時，瀏覽器把整個表單 POST 回同一支 URL。伺服器重建頁面物件，還原上一輪存下來的狀態，引發事件，再把完整 HTML 送回去。瀏覽器整頁替換。這就是 PostBack。

狀態要活過這一輪，靠的是 ViewState：一段藏在隱藏欄位裡的序列化資料。控制項樹一深，這個欄位就變大。你也比較難保證畫面上的 HTML 和自己手寫的標記一致，因為很多標籤是控制項渲染出來的。

HTTP 模組與處理常式掛在 `web.config`。應用程式有一條固定的生命週期事件。這些都綁在 `System.Web` 與 Windows 上的.NET Framework。

2.2 它沒有跟著進 ASP.NET Core

ASP.NET Core 拿掉了 `System.Web`。管線改成 middleware，設定改成 `Program.cs` 與相依性注入，跨平台執行。微軟沒有提供「把 Web Forms 專案改個目標框架就能跑」的官方路徑。

Microsoft Learn 上的遷移建議可以收成三句：

- 業務邏輯先從頁面拆出去，頁面技術才換得動。
- 想保留「元件 + 事件」的手感，優先看 Blazor。
- 大型站點用絞殺者模式：新站與舊站並存，一次搬一頁。Visual Studio 的升級助理可以用 YARP 把兩邊接在一起。

Blazor 的免費電子書《Blazor for ASP.NET Web Forms Developers》把元件、事件和單向綁定，逐項對回 Web Forms 的控制項與 PostBack。那是微軟給的對照。Web Forms 本身會隨著.NET Framework 繼續存在於 Windows，只是它不是.NET 10 的一部分。

2.3 WebForms Core 保留了哪一種感覺

Elanat 的做法不是把 `Page` 類別搬過來。它保留的是這句工作方式：人在伺服器上決定畫面要怎麼變。

差在「變」的單位。舊模型重建控制項樹，再渲染整頁。新模型產生幾行指令，例如把某個 `form` 的字級改成 28 像素、把送出鈕停用、在表單末尾加一個 `h3`。瀏覽器已經握有 DOM，不需要再收一次整頁。

官方列出的差異，值得當成設計約束而不是口號：

- 伺服器只產生協定文字，不在伺服器上組出整份新 DOM。
- 不使用 ViewState，View 頁面的 HTML 由你掌控。
- 送出資料走 AJAX，頁面本身不重新載入。
- 可以沒有 View。一支靜態 HTML 也能對伺服器發請求。
- 仍然可以呼叫 JavaScript。不做前端，是選項，不是封印。



圖 2.1: 同樣是「按按鈕之後伺服器決定結果」。左邊換掉整頁，右邊只改被點名的節點。

先停一下

官方教學有一句容易誤讀：「沒有 PostBack 機制」。指的是沒有 Web Forms 那種整頁 PostBack 與 ViewState。WebFormsJS 仍然提供名為 `PostBack` 的 JavaScript 函式，用來以 AJAX 送出表單。同名，不同機制。第 5 章會把這個函式拆開。

2.4 它比較像什麼

如果要用你已經會的東西來定位：它接近「伺服器渲染一小段操作腳本」，而不是第二個 React。你不描述虛擬 DOM，你描述對真實 DOM 的編輯動作：加標籤、設文字、設顏色、刪節點、綁事件。

頻寬上的好處來自這件事。一次成功的點擊，回應可能只有幾行：

```
[web-forms]
fs<form>=28px
bc<form>=violet
sd(btn_SetBodyValue)=1
nt<form>=h3
st<h3>=Welcome Nicolas!
```

瀏覽器不需要一份新的版面、一份新的腳本、一份新的隱藏欄位。代價是：你要會描述「改哪一個節點」。指錯節點，指令會作用在別的標籤上，而且不會有控制項樹在編譯期幫你擋下來。第 6 章就是在練這件事。

第三章　一條請求長什麼樣子

這一章把參與者固定下來。後面的程式都只是在填這張圖的某一格。

3.1　四個角色

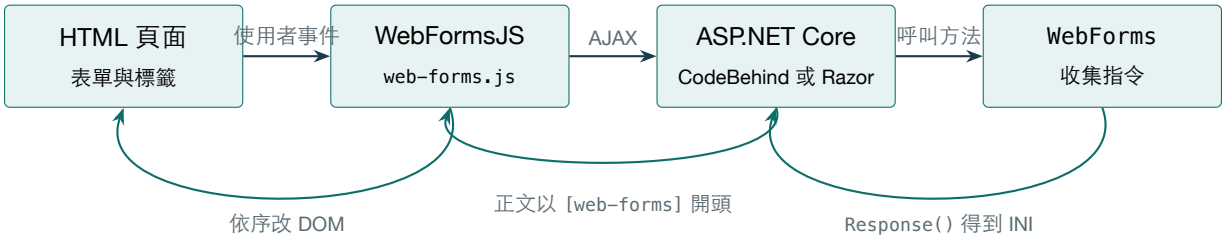


圖 3.1: 指令在伺服器組好，在瀏覽器執行。DOM 不送回伺服器重建。

HTML 頁面擁有真實的標籤。`input` 的 `name`、按鈕、你後來用指令加上的 `h3`，都在這裡。

WebFormsJS 做三件事。第一，送出時把表單序列化：一般表單用查詢字串形式，`multipart` 表單用 `FormData`。第二，看回應正文是不是 Action Control；是就執行，不是就把正文當成 AJAX 結果放進頁面（預設可放進一個 `div`，見 `PostBackOptions.SetResponseInsideDivTag`）。第三，在第一次整頁輸出時，若伺服器把指令放在 `<web-forms ac="...">`，它也會執行。

宿主是任何能讀取 `Request.Form` 並寫出正文的.NET 程式。CodeBehind 的 Controller 是文件裡的預設宿主。Razor Pages 的 `PageModel` 也行。

`WebForms` 類別本身幾乎不碰 HTTP。你呼叫 `SetText`、`AddTag`，它把對應的兩碼指令累積成一份緩衝。`Response()` 在前面加上 `[web-forms]` 與換行。真正寫進 HTTP 回應的是宿主。

這一節要記住

類別不認識你的資料庫，也不認識你的路由。它只負責把「對哪個節點、做哪種動作、值是什麼」編成一行文字。驗證、存檔、權限，仍寫在 `PageLoad` 或按鈕處理常式裡。

3.2　Action Control 的形狀

每一行都是同一種句子。官方 README 把第一個等號前面拆成「兩個字元的動作碼」加上「六種定位之一」。

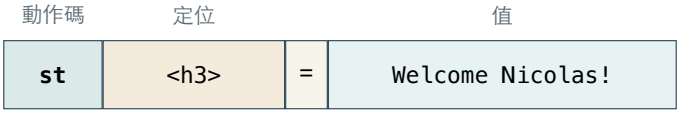


圖 3.2: `st` 是 Set Text。定位 `<h3>` 表示頁面上第一個 `h3`。等號右邊是要寫入的文字。

六種定位在第 6 章展開。這裡先記三個最常用的：

你想指到	寫法	例子
`id`	動作碼後面直接接 id	`deMyTagId=1`
`name`	圓括號	`sd(btn_Send)=1`
標籤名	角括號，索引從 0	`nt<form>=h3`

指令由上往下執行。若上一行把某個元素的 `id` 改掉，下一行不能再用舊的 `id`。負索引從尾端數：`de<li>-1=1` 刪除最後一個 `li`。

換行、等號、雙引號在值裡面有跳脫寫法，因為這份文字既要當 INI，有時還要塞進 HTML 屬性：

想表達	在值裡寫成
換行	`$[ln];`
等號	`$[eq];`
雙引號	`$[dq];`
多行指令折成單行	`$[sln];`

3.3 兩種回應，兩種包裝

WebFormsJS 區分「瀏覽器第一次打開這頁」和「頁面已經開著、使用者又打了一次請求」。

第一次整頁輸出時，指令可以放在頁尾的標籤裡，而不是單獨一份 INI。`ac` 屬性裡的多行指令用 `$[sln];` 接起來。類別對應的方法是 `ExportToWebFormsTag`。雙引號不能直接出現在 `ac` 裡。

```
<input name="TextBox" id="TextBox" type="text" />
<web-forms ac="bc<body>=green"></web-forms>
```

頁面已經在瀏覽器裡、這次是 AJAX 時，回應正文就是 INI。若你要換成一整塊新的內部 HTML，官方做法是讓第一行先 `SetText` 到主要容器（通常是 `body`），後面的行再做細修。

怎麼判斷這次是 AJAX？WebFormsJS 會加一個請求標頭：名稱 `Post-Back`，值 `true`。伺服器看到它，就知道不該再回完整版面。

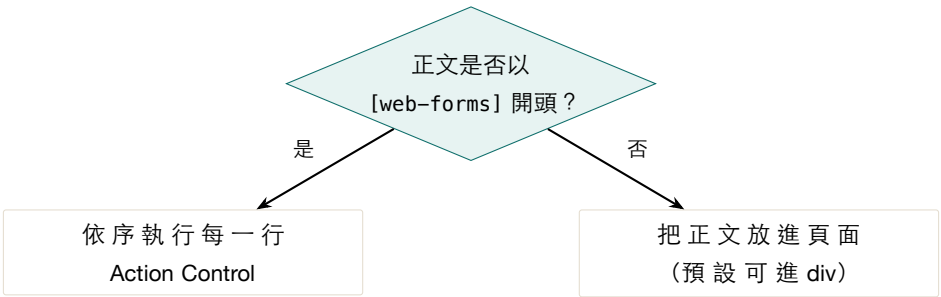


圖 3.3: WebFormsJS 的分支只有這一個判斷。沒有魔法定界，就是看第一段是不是 `[web-forms]`。

3.4 CodeBehind 在這張圖的哪一層

CodeBehind 是 ASP.NET Core 上的頁面框架，`.aspx` 是它的 View。標記支援兩種寫法：Razor 風格的 `@page`、`@model`，以及舊式的 `<%= …%>`。NuGet 上的範例把兩者並列。WebForms Core

是它的一項能力，不是它的全部。你仍然可以只做 MVC：Controller 組 Model，View 把 Model 印出來，完全不碰 `WebForms`。

和本書主線有關的三個接點：

1. 版面的 `<head>` 會帶 `/script/web-forms.js`。這支檔案由框架產生。引入之後，`type="submit"` 的輸入會自動得到 `onclick="PostBack(this)"`。
2. Controller 繼承 `CodeBehindController`。`PageLoad(HttpContext)` 是預設進入點。
3. 要把回應收成只有指令、不要再輸出 View，教學裡使用 `Control(form)` 接著 `IgnoreAll()`。另一段公開範例使用 `Write(form.Response())` 並把 `IgnoreViewAndModel` 設為 `true`。兩者都是在說：這次 HTTP 正文就是那份 INI。

先停一下

`Control` 與 `IgnoreAll` 屬於 CodeBehind 的 Controller，不屬於可移植的 `WebForms` 類別。換到 Razor Pages 時，改成 `return Content(form.Response(), "text/html")`，或在第一次繪製時用 `ExportToHtmlComment` 把指令嵌進頁面。

第四章 把第一個專案跑起來

這一章只建立宿主，還不寫業務。目標是：瀏覽器開得到一頁 HTML，標頭裡有 WebFormsJS，伺服器進得了 `PageLoad`。

4.1 你需要的工具

- .NET SDK 10。CodeBehind 4.8.1 的 NuGet 頁面寫明它對準 `net10.0`。
- 一個可以看網路回應的瀏覽器。開發者工具的 Network 面板比任何日誌都重要，因為你要核對的是表單正文與那份 INI。

下面的指令用 SDK 內建的空網站當起點，不使用 Visual Studio 專用的精靈。介面不同沒有關係，最後要出現的檔案是一樣的。

```
dotnet new web -n HelloWebForms
cd HelloWebForms
dotnet add package CodeBehind --version 4.8.1
```

`dotnet new web` 在.NET 10 會產生最小的 `Program.cs`。把官方套件頁上的註冊貼進去。開發環境不要在啟動時做正式編譯快取，所以 `AddCodeBehind` 的參數用「不是 Development 才啟用」：

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCodeBehind(!builder.Environment.IsDevelopment());

var app = builder.Build();

app.UseCodeBehind();

app.Run();
```

套件說明還列出兩個變體，先知道名字即可。`UseCodeBehindAsync()` 改為非同步執行。`UseCodeBehindNextNotFound()` 讓這個中介軟體找不到頁面時繼續往後送，而不是自己結束請求。第一個練習用同步的 `UseCodeBehind()` 就夠。

動手

執行 `dotnet run`，用瀏覽器打開終端機印出的位址。CodeBehind 的文件說，第一次設定並執行時，預設模板會進到 `wwwroot`。若目錄已經存在，而且選項裡的 `view_path` 不是 `wwwroot`，模板不一定會再生成。看到版面之後，檢視原始碼，確認 `head` 裡有：

```
<script type="text/javascript" src="/script/web-forms.js"></script>
```

這一行是後續所有自動 `PostBack` 的前提。沒有它，提交按鈕就是普通的整頁 POST。

4.2 View 與 Controller 各放什麼

CodeBehind 的 View 是 `.aspx`。官方 MVC 範例把控制器名字寫在頁首，而不是寫死在路由表。Razor 風格像這樣：

```
@page
@controller MyController
@model MyModel
```

標準風格則是：

```
<%@ Page Controller="MyController" Model="MyModel" %>
```

WebForms Core 的入門教學更短，只用控制器與版面，不強制 Model：

```
@page
@controller MyController
@layout "/layout.aspx"
@{
    ViewData.Add("title", "WebForms Core page");
}
```

控制器是一個部分類別，進入點固定叫 `PageLoad`：

```
using CodeBehind;

public partial class MyController : CodeBehindController
{
    public void PageLoad(HttpContext context)
    {
    }
}
```

這一節要記住

View 負責標記與欄位名稱。Controller 負責讀 `Request.Form`、決定要不要動資料、然後交出一份 `WebForms`。不要在 `.aspx` 裡寫事件處理。CodeBehind 的設計就是讓伺服器程式留在類別裡。

4.3 最小的一頁

把 `wwwroot` 裡的 `Default.aspx` 換成下面這份。欄位 `name` 會原樣出現在 POST 正文裡，所以控制器待會用同樣的字串去讀。

```
@page
@controller MyController
@layout "/layout.aspx"
@{
    ViewData.Add("title", "WebForms Core page");
}
```

```
<form method="post">
  <label>Your name
    <input type="text" name="txt_Name" />
  </label>
  <label>Font size
    <input type="number" name="txt_FontSize" value="28" />
  </label>
  <label>Background
    <input type="text" name="txt_BackgroundColor" value="violet" />
  </label>
  <input type="submit" name="btn_SetBodyValue" value="Send" />
</form>
```

版面檔沿用模板。你要確認的只有腳本那一行還在 `head` 裡。提交鈕不需要手寫 `onclick`。`PostBackOptions.AutoSetSubmitOnClick` 的預設是開啟，函式庫會自己補上 `PostBack(this)`。

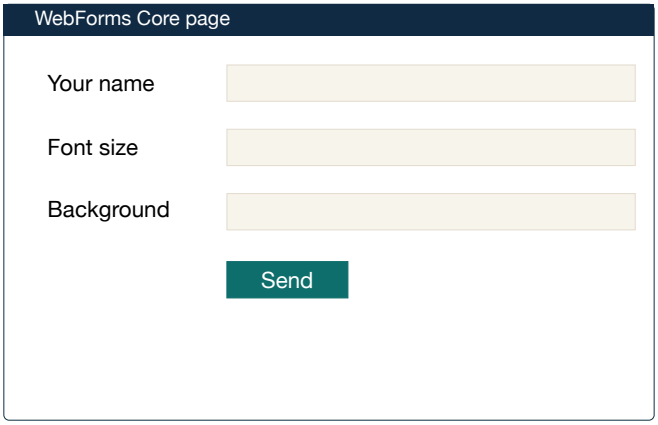


圖 4.1: 第一頁在點擊之前。三個欄位都在同一個 `form` 裡，按鈕是 `submit`。

4.4 不使用 CodeBehind 的時候

協定不要求 `.aspx`。獨立套件 `WFC` 提供同一套 `WebForms` 類別，給一般的 ASP.NET Core 使用。靜態頁面只要能載入 `web-forms.js`。腳本本體在 `elanatframework/Web_forms` 儲存庫，檔名就是 `web-forms.js`。

這種宿主裡沒有 `Control()` 與 `IgnoreAll()`。你自己把 `form.Response()` 寫進 HTTP 正文，`Content-Type` 用 `text/plain` 或 `text/html` 都可以，WebFormsJS 看的是正文開頭，不是媒體類型的細分。第一次整頁則把 `ExportToHtmlComment(true)` 放進 Razor 的 `ViewData`，讓指令以 HTML 註解的形式跟著頁面下來。公開的 Razor 範例使用的鍵是 `ViewData["WebForms"]`。

先停一下

不要同時啟用兩套「也叫 Web Forms」的中介軟體來處理同一條路由。CodeBehind 的 `UseCodeBehind`、WebFormsForCore 的 `UseWebForms`、WebFormsCore 的 `MapPages` 是三個不同管線。這章只留 CodeBehind。

第五章 走完官方那個歡迎頁

第 4 章的表單已經在頁面上。這一章補上控制器，然後對著網路面板把請求與回應讀完。範例的欄位名稱與官方教學一致，方便你對回文件。

5.1 控制器

`PageLoad` 先看這次 POST 有沒有帶來名為 `btn_SetBodyValue` 的欄位。有，才代表送出鈕被按過。WebFormsJS 送出的正文與傳統 HTML 表單相同，多出來的是 `Post-Back: true` 這個標頭，而不是一堆框架專用的隱藏欄位。

```
using CodeBehind;

public partial class MyController : CodeBehindController
{
    public void PageLoad(HttpContext context)
    {
        if (!string.IsNullOrEmpty(
                context.Request.Form["btn_SetBodyValue"]))
        {
            btn_SetBodyValue_Click(context);
        }
    }

    private void btn_SetBodyValue_Click(HttpContext context)
    {
        string name = context.Request.Form["txt_Name"];
        string background = context.Request.Form["txt_BackgroundColor"];
        int fontSize = context.Request.Form["txt_FontSize"].ToNumber();

        WebForms form = new WebForms();
        form.SetFontSize(InputPlace.Tag("form"), fontSize);
        form.SetBackgroundColor(InputPlace.Tag("form"), background);
        form.SetDisabled(InputPlace.Name("btn_SetBodyValue"), true);
        form.AddTag(InputPlace.Tag("form"), "h3");
        form.SetText(InputPlace.Tag("h3"), "Welcome " + name + "!");
        Control(form);
        IgnoreAll();
    }
}
```

`ToNumber()` 是 CodeBehind 對字串的擴充，教學用它把字級從表單文字變成整數。`InputPlace` 是定位的建構方式，下一章會對照它的字串形式。五個方法各自變成回應裡的一行，順序與呼叫順序相同。

5.2 瀏覽器送出什麼

在姓名欄填 `Nicolas`、字級 `28`、背景 `violet`，按下 Send。官方教學記錄到的正文是：

```
txt_BackgroundColor=violet&txt_FontSize=28&txt_Name=Nicolas&btn_SetBodyValue=Send
```

欄位順序依表單裡的 DOM 順序，教學頁上的按鈕值是他們的英文句子；你若把 `value` 改成 `Send`，正文最後一段就會是 `btn_SetBodyValue=Send`。伺服器只檢查這個名稱在不在，不檢查顯示文字。

同時看請求標頭，應該有：

```
Post-Back: true
```

沒有這個標頭，通常是腳本沒載入，瀏覽器改走傳統導覽。那時候你收到的會是整頁 HTML，而不是下面的 INI。

5.3 伺服器回什麼

```
[web-forms]
fs<form>=28px
bc<form>=violet
sd(btn_SetBodyValue)=1
nt<form>=h3
st<h3>=Welcome Nicolas!
```

對照呼叫，逐行讀：

呼叫	回應行	畫面上的結果
`SetFontSize(..., 28)`	`fs<form>=28px`	第一個 `form` 的字級變成 28 像素。整數多載會補上單位。
`SetBackgroundColor(..., "violet")`	`bc<form>=violet`	同一個表單的背景色。
`SetDisabled(..., true)`	`sd(btn_SetBodyValue)=1`	依 `name` 找到按鈕並停用。`1` 是真，`0` 是假。
`AddTag(..., "h3")`	`nt<form>=h3`	在表單內部末尾新增 `h3`。`nt` 是 Add Tag。
`SetText(...)`	`st<h3>=Welcome Nicolas!`	寫進頁面上第一個 `h3`。

`IgnoreAll()` 是這次回應能這麼短的原因。若忘記呼叫它，CodeBehind 仍會把 View 和版面渲染進正文。WebFormsJS 看不到開頭的 `[web-forms]`，就會把整頁 HTML 塞進頁面裡的某個容器。症狀很固定：版面嵌進版面。

5.4 三種把請求送回去的函式

README 把用戶端入口收成三個函式。

`PostBack` 送出表單欄位，而且應該出現在 `form` 裡面。提交鈕會自動變成 `PostBack(this)`。它有一個輸出位置的多載：

- `PostBack(this, true)`：保留頁面，把回應加到 `body` 內部的開頭。



圖 5.1: 指令套用之後。表單帶著背景色與較大的字，按鈕停用，歡迎句是新加的 h3，頁面沒有整頁重載。

- `PostBack(this, "<div>2")`：放進第三個 `div`（索引從 0）。
- `PostBack(this, "MyTagId")`：放進這個 `id`。

只有在回應不是 Action Control 時，這個輸出位置才決定正文被插到哪。若正文是指令，WebFormsJS 改的是指令點名的節點，不是這個輸出位置。

`GetBack` 只請求 URL，不帶表單欄位。它不需要包在 `form` 裡。三個多載：

- `GetBack("YourURL")`：請求你給的路徑，路徑可以帶查詢字串。
- `GetBack(this)`：在表單內則用該表單的 `action`，否則用頁面上第一個表單的 `action`。
- `GetBack()`：請求瀏覽器現在這條 URL。

`TagBack` 不再打伺服器。它渲染頁面上已經存在的某個 `<web-forms>` 標籤裡的 `ac`。例如 `TagBack("(my-class)-1")` 會處理 class 為 `my-class` 的最後一個 `web-forms` 標籤。這是把指令先夾在 HTML 裡、等事件發生再執行的做法。

這一節要記住

`PostBack` 帶資料，`GetBack` 只帶網址，`TagBack` 連網址都不打。三者最後若拿到的是 `[web-forms]`，都會走進同一個指令執行器。

5.5 用戶端可以調的開關

`web-forms.js` 開頭有一份 `PostBackOptions`。改的是函式庫的行為，不是某一頁的指令。公開 README 列出的預設值如下。

選項	預設在做的事
`UseProgressBar`	表單裡有檔案欄位時，顯示上傳進度。
`UseConnectionErrorMessage`	連線失敗時顯示 `ConnectionErrorMessage`，預設文字是 `Connection Error`。
`AutoSetSubmitOnClick`	自動把提交鈕接上 `PostBack`。
`SendDataOnlyByPostMethod`	預設 `false`。設成 `true` 就忽略表單的 `method`，一律 POST。
`SetResponseInsideDivTag`	非指令的回應要不要包進 `div`。
`WebFormsTagsBackgroundColor`	`web-forms` 標籤渲染前先塗一塊底色，預設 `#eee`。

進度條的三個樣式鍵是 `ProgressBarStyle`、`ProgressBarPercentLoadedStyle`、`ProgressBarValueStyle`。它們是 CSS 字串，不是伺服器屬性。

第六章　指到正確的那一個節點

Action Control 沒有控制項 ID 系統。你每次都在描述「現在這份 DOM 裡的哪一個節點」。指錯了，指令仍會成功執行，只是執行在別處。這一章把六種定位，和 C# 裡的兩種寫法，放在同一棵樹上。

6.1　一棵拿來指的樹

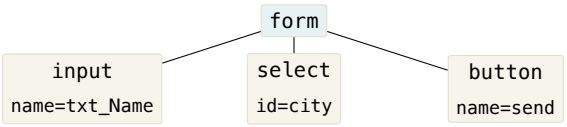


圖 6.1: 後面的例子都指這棵樹。索引從 0 開始；負索引從尾端數，`-1` 是最後一個。

6.2　六種寫法

README 規定，動作碼之後、等號之前，用符號決定定位種類。

種類	符號	說明
id	無，直接接在動作碼後	`svcity=taipei` 會改 `id="city"` 的值。id 在一頁裡應唯一，所以沒有索引。
name	`(name)` 再加索引	`sd(send)=1` 停用第一個 name 為 `send` 的元素。第三個要寫 `(send)2`。
標籤	`<tag>` 再加索引	`bc<form>=violet` 與 `bc<form>0=violet` 相同。第二個表單是 `<form>1`。
class	`{class}` 再加索引	`ta{my-textbox}1=right` 作用在 class 含該名稱的第二個元素。
query	`*` 開頭	一個節點。等號若出現在查詢裡，要寫成 `$[eq];`。
query all	`[` 開頭	同一條指令套到所有命中的節點。

表 6.1: 定位寫在動作碼與等號之間。預設索引是 0。

官方教學用 `InputPlace` 產生這些字串，也允許你直接傳字串。兩邊是同一條指令：

```
form.AddTag(InputPlace.Tag("form"), "h3");
form.AddTag("<form>", "h3");

form.AddTag("<form>1", "h3");

form.DeleteOptionTag(InputPlace.Id("city"), "taipei");
form.DeleteOptionTag("city", "taipei");

form.IncreaseValue(InputPlace.Name("hdn_Count"), 1);
form.IncreaseValue("(hdn_Count)", 1);
form.IncreaseValue("(hdn_Count)2", 1);
```

`InputPlace.Tag` 的第二個參數可以是索引。教學在檔案上傳範例裡用 `InputPlace.Tag("h3", -1)` 指到剛剛加上去的最後一個 `h3`。這比再寫一次標籤名安全，因為頁面上可能已經有別的 `h3`。

6.3 路徑，不只是一個節點

定位可以串起來。README 的例子：

```
at<ul>|{my-class}1|<li>2=My text string
```

讀的方向是由外往內：class 為 `my-class` 的元素，裡面的第一個 `ul`（索引 0，所以 `<ul>` 後面沒寫數字），再進去的第三個 `li`。`at` 是 Add Text，所以文字加在那個 `li` 裡。

也可以先找節點再往上爬。斜線表示父節點，多寫一條就再上一層：

```
bc<p>4/=violet
```

這一行把第五個 `p` 的父元素背景設成紫色。`query all` 不能用在這種巢狀寫法裡。

這一節要記住

指令是對「執行到這一行時」的 DOM 說話。`nt<form>=h3` 執行完，下一個 `st<h3>` 才看得到新的 `h3`。若你在加標籤之前就 `SetText` 到 `h3`，而頁面上又沒有別的 `h3`，那一行沒有目標。

6.4 一組會踩到的情況

索引是文件順序，不是視覺順序。`display:none` 的節點仍然佔一個索引。用標籤名定位之前，先在開發者工具裡用同樣的選擇器數一次。

id 定位中間沒有括號。`sd(send)=1` 找的是 name。`sdsend=1` 找的是 id。多一對括號，目標就換了一種屬性。

改 **id** 之後，舊 **id** 失效。README 特別寫了這條：後面的行不能再使用改掉之前的 id。要連續編輯同一個新節點時，用標籤加負索引，或先設好新 id 再用新 id。

query 裡的等號。CSS 屬性選擇器常有 `=`。它必須寫成 `$[eq];`，否則會被看成指令的等號。

動手

在歡迎頁的控制器裡，把 `SetText` 的目標從 `InputPlace.Tag("h3")` 改成 `InputPlace.Tag("h3", -1)`。再在版面裡故意放一個空的 `h3`。兩種寫法的歡迎句會落在不同節點。把兩次回應的 INI 存下來，對照索引。

第七章 Action Control 代碼

你平常呼叫的是 C# 方法。除錯時讀到的是這張代碼表。兩套名字要對得起來，否則網路面板裡的一行會變成無法翻譯的縮寫。代碼取自 WebFormsJS 的公開 README，這裡按學習順序重排，不是按原始檔的段落順序。

7.1 一行裡面還能有前置

在動作碼前面可以加前置，README 稱為 Pre Runner。每一段前置以) 結束。

前置	寫法	意義
延遲	: 秒數)	等這麼多秒再執行這一行
週期	(秒數)	之後每這麼多秒再執行一次

```
(4)+w<b>1=10px
:10)(4)+w<b>1=10px
```

第一行：每 4 秒把第二個 b 的寬度加 10 像素。第二行：先等 10 秒，再進入同一個週期。前置放在佇列裡，可以串。+w 是 Increase Width，下一節的表裡有。

7.2 加、設、插入、刪

Add 是在原值上追加。Set 是換成新值。Insert 只在目前沒有值時才寫入。Delete 拿掉。這個區分比死記兩個字母重要。

對象	Add	Set	Insert	Delete
標籤內部末尾	nt			de（值 1）
標籤之前	bt			dp 刪父標籤
標籤之後	ft			
標籤前方（往上）	ut			
文字	at	st	it	dt
往上加文字	pt			
title	al	sl	il	dl
屬性	aa	sa	ia	da（值是屬性名）
class	ac	sc	ic	dc
style	as	ss	is	ds（只寫樣式名）
value	av	sv	iv	dv
name	an	sn	in	dn
id	ai	si	ii	di

表 7.1: 空格表示 README 的該組沒有對應代碼。nt 的值是 TagName 或 TagName|Id。

屬性與選項用豎線分欄。saCheckBoxInput=onchange|MyNewFunc() 表示：對 id 為 CheckBoxInput 的元素，把 onchange 設成 MyNewFunc()。aoSelectInput=OptionValue|Option Text|1 表示加一個 option，第三段 1 是選取。

核取方塊的加入代碼是 `ak`，值同樣是 `Value|Text|1` 或 `0`。Set 對應 `sk`：若目標本身就是核取方塊，值只是 `1` 或 `0`；若是容器，值改為 `Value|Text|1` 或 `0`。

7.3 外觀、尺寸與表單狀態

代碼	方法語意	值
bc / tc	背景色 / 文字色	CSS 顏色
fs / fn / fb	字級 / 字型 / 粗體	尺寸、名稱、`1` 或 `0`
sw / sh	寬 / 高	CSS 長度
ta	文字對齊	例如 `right`
vi	可見	`1` 或 `0`
sd / sr / sf	停用 / 唯讀 / 聚焦	`1` 或 `0`
mx / mn	最大長度 / 最小長度	整數
ts / ti	選取的值 / 索引	下拉選單
ks / ki	核取的值 / 索引	值再加 `|1` 或 `|0`
cu	改瀏覽器網址	URL

可見性代碼是 `vi`，不是把元素 `Delete`。刪掉的節點下一行就指不到；隱藏的節點還在，索引也還在。

7.4 增減

增減只作用在數字。代碼是符號加一個字母，不是兩個英文字母。

對象	增加	減少
value	`+v`	`-v`
height	`+h`	`-h`
width	`+w`	`-w`
font size	`+f`	`-f`
max length	`+x`	`-x`
min length	`+n`	`-n`

`+v(hdn_Count)=1` 把該隱藏欄位的值加 1。減少的寫法對稱。README 把減少這組的標題拼成 Descrease，代碼本身仍是減號。

7.5 事件

事件代碼讓一個 HTML 事件再去打伺服器，或去改頁面上的另一個位置。入門最常用的是 Post 與 Get。

代碼	方向	值的形狀
Ep / EP	設 Post 事件 / 監聽器	事件 |* 或輸出位置。* 表示把回應加進頁面。
Rp / RP	移除 Post	HTML 事件名稱
Eg / EG	設 Get 事件 / 監聽器	事件 | 路徑 | 輸出位置。路徑寫 # 表示目前這條路徑。
Rg / RG	移除 Get	HTML 事件名稱
Et / ET	只改頁面、不打伺服器	事件 | 輸出位置
Rt / RT	移除這種監聽	事件名稱

小寫代碼綁的是 HTML 事件屬性（例如 `onclick`），大寫代碼綁的是 `addEventListener` 那種監聽器。公開說明裡 Get 這組有「在表單內」與「自帶路徑」兩種值的形狀，差別在路徑那一段寫不寫。C# 端對應 `SetPostEvent`、`SetGetEvent` 以及帶 `Listener`、`To`、`WithOutput` 的多載。第 10 章的 Razor 範例用的就是 `SetGetEvent`。

同一份類別還有 Patch、Delete、Head、WebSocket、SSE、WASM 等事件方法。那些是進階通道。第一個能上線的表單不需要它們。知道「事件也是一行 Action Control，不是另一套前端框架」就夠。

7.6 先存起來，等一下再用

Save 這組把節點的現值存進一把鍵，稍後再用。鍵用過可以丟掉。代碼都以 `@` 開頭，避免和普通的兩個字母相撞。

代碼	存什麼	代碼	存什麼
@gi	id	@gn	name
@gv	value	@ge	value 的長度
@gt	文字	@gg	文字長度
@gc	class	@gs	style
@gl	title	@ga	屬性（值是 鍵 | 屬性名）
@gw	寬	@gh	高
@gr	唯讀	@gx	選取索引
@ta	對齊	@nl	子節點數量
@vi	可見		

取回時用另一組，寫在等號右邊，因為它們是「值的函式」，不是動作碼：

寫在值的位置	取出
`@cd`、`@ct`	快取；`@ct` 取出後刪除。值是 鍵 | 取不到時的替代。
`@cs`、`@cl`	同上，但是只活在這個瀏覽器工作階段裡的快取。
`@co`	Cookie。
`@dy` `@dm` `@dd`	年、月、日。
`@dh` `@di` `@ds` `@dl`	時、分、秒、毫秒。
`@mr`	亂數，值是 最大 | 最小。
`@_`	執行一段腳本，把結果當值。

另外還有不帶定位、自己佔一行的指令：`_` 執行腳本，`cd` / `cs` 設定快取秒數（`*` 表示永久），`rd` / `rs` 刪除快取（`*` 表示全部）。

7.7　多段回應

一份正文可以含有多組指令，組與組之間用單獨一行的 `#` 分開。請求的 URL 在路徑後面加上 `# 索引` 來選組，索引從 0 開始。你也可以寫 `#=MyIndex` 給某一組命名，之後用這個名字選它。

```
[web-forms]
bcTextBox1=green
#
sv(email)=myemail@gmail.com
#=Saved
as<body>=margin:0
```

請求 `/page.aspx` 或 `/page.aspx#0` 只執行第一組。`/page.aspx#1` 執行第二組。`/page.aspx#Saved` 與 `/page.aspx#2` 都執行第三組。

README 建議這項功能用在可以放進瀏覽器快取的離線回應，而不是每一次點擊都回傳一大包用不到的組。長時間快取時，記得讓這類回應帶上適合的 HTTP 快取標頭。

先停一下

`#` 在這裡是協定語法，不是 URL 片段的一般用途。伺服器若先做前端路由、把井字號截掉，多段回應的選組會失效。井字號通常不會送到伺服器；WebFormsJS 是在用戶端挑選已經下載的那一組。

第八章　C# 類別上你會呼叫的方法

WebForms 的公開方法很多。這一章只排你在前 11 章會用到、而且能在公開教學或類別清單裡對上代碼的那些。方法的第一個參數幾乎都是定位：InputPlace 或它的字串形式。

8.1　產生正文

```
string ini = form.Response();
string tag = form.ExportToWebFormsTag("");
string comment = form.ExportToHtmlComment(true);
```

Response() 在緩衝區前加上 [web-forms] 與換行。緩衝區是空的時，它仍回傳這一行標頭。ExportToWebFormsTag 做成 <web-forms ac="..."> ，多行已折成 $[sln];。ExportToHtmlComment(true) 做成 HTML 註解，參數表示要不要先換行。Razor Pages 第一次繪製時用註解，AJAX 往返時用 Response()。

CodeBehind 控制器不要自己把字串寫進 HttpContext 的情況，教學用的是：

```
Control(form);
IgnoreAll();
```

聯絡表單那份公開範例用的是另一組，效果同樣是壓掉 View：

```
Write(form.Response());
IgnoreViewAndModel = true;
```

選一組，不要兩組各寫一半。

8.2　和歡迎頁同一族的方法

下面左欄是歡迎頁已經出現過的呼叫，右欄是同一族、簽名類似、只換代碼的方法。寬與高、字級都有字串多載與整數多載；整數多載會帶像素單位。

方法	代碼	你傳入的值
SetText / AddText	st / at	文字，換行會被編成 $[ln];
AddTag	nt	標籤名，或標籤名與 id
AddTagBefore / After / ToUp	bt / ft / ut	同上
SetBackgroundColor	bc	顏色
SetTextColor	tc	顏色
SetFontSize	fs	字級
SetFontName / SetFontBold	fn / fb	名稱 / 布林
SetWidth / SetHeight	sw / sh	長度
SetVisible	vi	布林，編成 1 或 0
SetDisabled / SetReadOnly / SetFocus	sd / sr / sf	布林
SetAttribute / AddAttribute	sa / aa	屬性名與值
Delete / DeleteParent	de / dp	只有定位

`AddTag` 的 id 多載會把值編成 標籤 `|id`。沒有 id 時，值就只有標籤名。歡迎頁用的是後者，所以回應是 `nt<form>=h3`，新標題沒有 id。

8.3 表單控制項

```
form.AddOptionTag(InputPlace.Id("city"), "Taipei", "TPE", false);
form.SetSelectedValue(InputPlace.Id("city"), "TPE");
form.DeleteOptionTag(InputPlace.Id("city"), "TPE");
form.DeleteAllOptionTag(InputPlace.Id("city"));

form.SetChecked(InputPlace.Name("agree"), true);
form.AddCheckBoxTag(place, "I agree", "yes", false);
form.IncreaseValue(InputPlace.Name("hdn_Count"), 1);
form.SetMaxLength(InputPlace.Name("txt_Name"), 40);
```

`AddOptionTag` 的參數順序是定位、顯示文字、值、是否選取。對應 `ao`，三段用豎線接起來。`DeleteOptionTag` 的第二個參數是 option 的 value；刪光用 `DeleteAllOptionTag`，代碼值為 `*`。

8.4 延遲是改寫已經寫好的那一行

CodeBehind 的聯絡範例在 `Delete` 之後呼叫單參數的 `AssignDelay(3)`。對應的回應行以 `:3)` 開頭，也就是第 7 章的延遲前置，單位是秒。

各語言類別的母本在演化。Go 移植裡看得到的簽名是 `AssignDelay(毫秒, 行索引)`：它讀取緩衝區裡已經存在的那一行，把 `: 毫秒)` 加到動作碼前面。索引指的是緩衝區的行，不是 DOM。

先停一下

呼叫 `AssignDelay` 之前先看你安裝的多載。若參數是 `(int, int)`，第一個數的單位要對一下是秒還是毫秒，第二個數是「已經 `Add` 進去的那一行」。單參數的教學寫法 `AssignDelay(3)` 不要直接抄到另一種簽名上。

週期前置沒有出現在入門教學裡。需要「每四秒加寬」這種效果時，先確認類別有沒有對應的 `Assign` 方法；沒有的話，最後手段才是直接組那一行字串。平常不要繞過方法去拼 INI，否則跳脫規則會漏。

8.5 網址、重新載入、標題

類別清單裡還有一組不針對某個輸入欄位的方法：`ChangeUrl`、`SetHeadTitle`、`Redirect`、`ReloadPage`、`HistoryGo`、`ScrollTo`、`ClipboardWriteText`。它們仍然是回應裡的一行，由 WebFormsJS 在瀏覽器執行。`Redirect` 與 `ReloadPage` 會弄掉目前這份 DOM，和「只改一個 `h3`」的目標相反。能用 `SetText` 解決的，就不要重新載入。

`AddState(path, title)` 會產生代碼 `AS`，值是 路徑 `|` 標題。它讓瀏覽器的上一頁、下一頁對得上你加進去的狀態。第 10 章有一個完整的 Razor Pages 例子。`DeleteState` 與 `DeleteAllState` 用來收回。

這一節要記住

方法是代碼的型別安全包裝。除錯時以網路上的 INI 為準，不以你記得的方法名為準。若 INI 裡的定位和你想的不同，先回到第 6 章，不要先懷疑 WebFormsJS 沒執行。

第九章 表單、檔案與頁面上的訊息

歡迎頁只有三個文字欄位。真實表單還有核取、下拉、多行文字與檔案。這一章的模式都一樣：用 name 讀取，用 WebForms 寫回頁面，不要把整頁換掉。

9.1 讀取時只信 name

WebFormsJS 序列化的是表單控制項的 name 與值，和傳統 HTML 相同。id 用來讓你的指令找節點，不參與提交。一個欄位通常兩個都要：id 給回寫，name 給讀取。

CodeBehind 教學在「送出表單資料」那一節，把各種 input 類型各讀一次：文字、顏色、日期、時間、datetime-local、週、月、數字、電子郵件、電話、網址、搜尋、密碼、範圍、隱藏欄位、select、核取、單選、textarea，以及檔案。讀取方式只有兩條：

```
string email = context.Request.Form["txt_Email"];
IFormFile file = context.Request.Form.Files["upd_File"];
```

核取方塊沒被勾選時，該名稱不會出現在正文裡。用「是否有值」判斷，不要假設一定是 "false" 字串。單選按鈕會送出被選中那一顆的 value。

動手

做一份只有 select 與一顆核取方塊的頁面。控制器把兩者的原始字串用 AddText 寫進一個 pre。先不要美化。你要看到的是框架沒有幫你改寫欄位名稱。

9.2 訊息寫在頁面上，而不是寫進記錄

使用者看得到的結果應該是 DOM 的變化。公開的聯絡範例做法是：加一個 h3，設背景色，寫入句子，幾秒後刪掉，並停用按鈕。

```
private void Button_Click(HttpContext context)
{
    string name = context.Request.Form["name"];

    WebForms form = new WebForms();
    form.AddTag("<form>", "h3");
    form.SetBackgroundColor("<h3>", "green");
    form.SetText("<h3>", name + "! Your message was sent.");
    form.Delete("<h3>");
    form.AssignDelay(3);
    form.SetDisabled("(button)", true);

    Write(form.Response());
    IgnoreViewAndModel = true;
}
```

這段裡 Delete 寫在 SetText 之後，所以指令順序是：建立、上色、寫字、刪除。AssignDelay(3) 在這份教學裡跟在 Delete 後面，讓刪除延後發生，使用者才看得到那一行字。若你的多

載需要行索引，就把延遲套在刪除那一行，而不是套在 SetText 上。套錯行的話，標題會等三秒才出現，或出現後立刻消失。

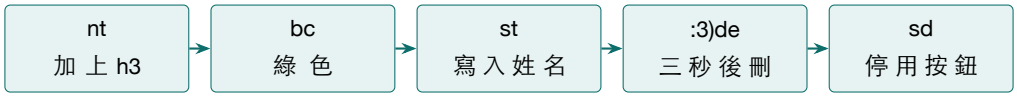


圖 9.1: 同一份回應裡的時間差只發生在帶有前置的那一行。其餘行仍依序立刻執行。

伺服器在這之前若要拒絕這次送出，就不要呼叫刪除。改成黃底或紅底的 h3，把按鈕留著，讓人改完再送。上傳教學就是這樣分出「不是圖片」與「已寫入」兩條路。

9.3 檔案

有 input type="file" 時，表單要能送出 multipart。WebFormsJS 會改用 FormData，並在 Use ProgressBar 為真時顯示進度條。控制器從 Request.Form.Files 拿 IFormFile。

公開範例的檢查順序值得照做，因為順序本身就是安全邊界：

1. 檔案物件是否存在、長度是否大於 0。
2. 副檔名是否落在你允許的清單，例如 .jpg、.jpeg、.png、.gif。不符合就寫提示、改背景色，然後 return。
3. 通過之後才 CopyTo 到你打算公開的目錄。
4. 用 AddTag 加一個 img，SetAttribute 設定 src，SetWidth 限制寬度。

```
form.AddTag("<form>", "img");
form.SetAttribute("<form>|<img>-1", "src", filePath);
form.SetWidth("<form>|<img>-1", 400);
```

這裡用巢狀定位加上 -1，避免頁面上另一張舊圖被改到。src 應該是網站內的路徑。不要把使用者提供的檔名直接拼進可以跳出目錄的位置；教學範例為了短，直接使用上傳檔名，正式環境要換成你產生的名稱。

先停一下

進度條只描述「位元組送出了多少」。它不表示檔案已經通過副檔名檢查，也不表示已經寫進磁碟。成功與否仍要看這次回應裡的 h3 文字。

9.4 第一次載入就綁好事件

有些按鈕不在 form 的提交路徑上。Razor 範例在 OnGet 裡先下 SetGetEvent，把點擊綁到一條查詢字串，指令則用 HTML 註解放進第一次的頁面：

```
WebForms form = new WebForms();
form.SetGetEvent("Button1", HtmlEvent.OnClick, "?add_state_1");
ViewData["WebForms"] = form.ExportToHtmlComment(true);
```

使用者按下按鈕時，瀏覽器對目前路徑發出帶有 add_state_1 的 Get。PageModel 看到查詢鍵，改走另一個處理方法，這次回 Content(form.Response(), "text/html")。

HtmlEvent.OnClick 是類別提供的事件名稱常數。若你的套件版本裡沒有這個類型，就改用該版本接受的事件字串，例如文件裡的 onclick。先讓編譯通過，再看 INI 裡的事件代碼是 Eg 還是 EG。

第十章 沒有 ViewState 時，狀態放哪裡

舊版 Web Forms 把控制項狀態塞進隱藏欄位，所以重新整理與回傳之後，文字框裡的字還在。WebForms Core 的官方教學把「不使用 ViewState」列成優點。優點的背面是：你要自己決定哪些狀態活在 DOM、哪些活在伺服器、哪些活在瀏覽器歷史。

10.1 三種放法

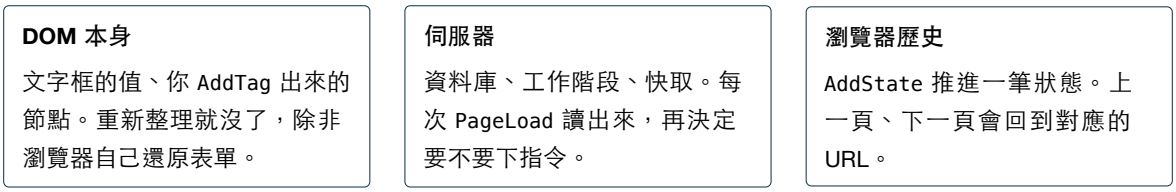


圖 10.1: 不要找一個隱藏欄位把三邊都裝進去。WebForms Core 沒有那條自動通道。

DOM 適合「使用者正看著的東西」：錯誤訊息、剛加進清單的一列、按鈕是否停用。伺服器適合「重新整理之後仍必須為真的東西」：訂單已經成立、檔案已經寫入。歷史適合「這次點擊是頁面故事的一步」：精靈的下一步、可以退回的預覽。

10.2 用查詢字串推進歷史

下面的流程來自 Elanat 公開的 Razor Pages 狀態範例，收成你可以對照實作的步驟。重點不是 Razor 語法，是 Get 與 `AddState` 的搭配。

1. 第一次 `OnGet` 沒有動作查詢鍵。建立 `WebForms`，對兩顆按鈕 `SetGetEvent`，路徑分別是 `?add_state_1` 與 `?add_state_2`。把 `ExportToHtmlComment` 放進 `ViewData`，回傳 `Page()`。

2. 按鈕被按下後，同一個 `OnGet` 看到查詢鍵，改呼叫對應的方法。

3. 該方法 `new WebForms()`，呼叫 `AddState` 寫入這一步的路徑與標題，再用 `AddText` 在頁面上留一句話。

4. 回傳 `Content(form.Response(), "text/html")`，不要回傳整頁。

`AddState` 的兩個參數是路徑與標題，編成 `AS`，值為 路徑 | 標題。瀏覽器的上一頁會回到上一筆路徑。伺服器若在那條路徑上又依據查詢鍵重做一次指令，畫面才跟得上歷史；只在用戶端堆文字、伺服器不認得那條路徑，上一頁會打開一個沒有對應畫面的網址。

這一節要記住

每一次點擊都是一次新的請求。伺服器預設不記得上一輪 `WebForms` 物件裡有什麼，那個物件在 `Response()` 之後就可以丟掉。需要累積的清單，要嘛寫在頁面的 DOM（下一輪表單會把它送回來），要嘛寫在伺服器的儲存體。

10.3 快取是用戶端的捷徑

第 7 章的 `cd`、`cs`、`@cd`、`@cs` 是 WebFormsJS 的快取，不是 ASP.NET Core 的輸出快取。`cd` 的值是秒數，`*` 表示永久。`cs` 只在這個瀏覽器工作階段有效。

它適合「同一段指令在短時間內會被要很多次」，例如多段回應裡那些離線組。它不適合當授權。永久快取一份會改 DOM 的指令，等於讓舊指令在你修完伺服器之後還繼續生效。公開說明建議多段回應搭配長期快取標頭時要有意識地開，而不是每一支 API 都開。

10.4 重新整理之後什麼還在

用歡迎頁做一次檢查，比記規則快。

操作	你應該觀察到
按下 Send	表單變色、出現歡迎句、按鈕停用。網址可以不變。
重新整理	回到 View 原本的 HTML。歡迎句消失，按鈕恢復。伺服器沒有把這些寫進 ViewState。
按上一頁	若你沒呼叫 `AddState`，上一頁是進入這站之前的文件，不是「送出之前的那一幕」。送出之前的那一幕根本沒有被推進歷史。

若產品要求重新整理後歡迎句還在，就把姓名寫進資料庫或工作階段，並在沒有 `Post-Back` 標頭的第一次 `PageLoad` 用 `ExportToWebFormsTag` 把 `SetText` 夾進頁面。那是伺服器狀態，不是協定自動幫你留的。

第十一章 練習：一頁聯絡表單

這一章把前面的零件裝成一頁。做完時，你應該能不看網路教學，獨立解釋回應裡的每一行。

11.1 頁面

```
@page
@controller ContactController
@layout "/layout.aspx"
<form method="post">
  <label>Name
    <input type="text" name="name" id="name" maxlength="80" />
  </label>
  <label>Message
    <textarea name="message" id="message"></textarea>
  </label>
  <input type="submit" name="button" value="Send" />
</form>
```

`maxlength` 寫在 HTML 上，是給瀏覽器的限制。伺服器仍要再檢查一次，因為請求可以不經過這個頁面。

11.2 控制器的判斷

```
using CodeBehind;

public partial class ContactController : CodeBehindController
{
    public void PageLoad(HttpContext context)
    {
        if (context.Request.Form["button"].Has())
            Button_Click(context);
    }

    private void Button_Click(HttpContext context)
    {
        string name = context.Request.Form["name"];
        string message = context.Request.Form["message"];
        WebForms form = new WebForms();

        if (string.IsNullOrWhiteSpace(name) ||
            string.IsNullOrWhiteSpace(message))
        {
            form.AddTag("<form>", "h3");
            form.SetBackgroundColor("<h3>-1", "yellow");
```

```
            form.SetText("<h3>-1", "Name and message are required.");
            Write(form.Response());
            IgnoreViewAndModel = true;
            return;
        }

        form.AddTag("<form>", "h3");
        form.SetBackgroundColor("<h3>-1", "green");
        form.SetText("<h3>-1", name + "! Your message was sent.");
        form.Delete("<h3>-1");
        form.AssignDelay(3);
        form.SetDisabled("(button)", true);

        Write(form.Response());
        IgnoreViewAndModel = true;
    }
}
```

`Has()` 是 CodeBehind 教學用來判斷表單鍵是否有值的擴充。若你的版本沒有它，改回 `!string.IsNullOrEmpty(...)`。失敗的那條路不要停用按鈕，也不要排刪除。成功的那條路才停用，並讓綠色標題在數秒後消失。

這裡的 `<h3>-1` 假設這一輪指令剛剛才加入頁面上最後一個 `h3`。若版面裡已經有別的 `h3`，負索引仍然指到全頁最後一個，不一定是你剛加的那個。更穩的寫法是 `AddTag` 時帶上 id，之後都用 id 定位。

11.3 你要在網路面板裡看到的

失敗時，正文類似：

```
[web-forms]
nt<form>=h3
bc<h3>-1=yellow
st<h3>-1=Name and message are required.
```

成功時，在單參數 `AssignDelay(3)` 的教學簽名下，刪除那一行會帶延遲前置。行的先後仍是建立、上色、寫字、延遲刪除、停用：

```
[web-forms]
nt<form>=h3
bc<h3>-1=green
st<h3>-1=Ada! Your message was sent.
:3)de<h3>-1=1
sd(button)=1
```

若你的 `AssignDelay` 把單位編成毫秒，前置裡的數字會是 `3000` 這一類，而不是 `3`。以實際回應為準，回頭改你對「使用者能看幾秒」的預期。

動手

故意在版面放兩個 `form`。不改控制器，再送出一次。歡迎句或錯誤句會出現在哪一個表單？用第 6 章的索引把 `AddTag` 的目標改成第二個表單，讓 INI 裡出現 `<form>1`。

11.4 再改三處，才算做完

1. 姓名超過 80 個字元時走失敗路徑。不要只靠 HTML 的 `maxlength`。
2. 成功時不要把訊息正文回寫到頁面。這份練習沒有存檔；若你加上存檔，頁面上只留「已送出」，避免把使用者輸入再嵌進 HTML 而不做編碼。`SetText` 會把文字放進節點。含有 HTML 的姓名不應該被當成標記。
3. 把 `AssignDelay` 拿掉，重新整理與不重新整理各試一次。確認標題是被指令刪掉，而不是被重新載入清掉。

這一節要記住

做完的標準：你能蓋住這章的程式，只看一份 INI，說出使用者會看到什麼、三秒後會看到什麼、哪一行是定位錯誤時最危險的一行。

第十二章　另外兩條同名的路

主線到第 11 章已經可以寫新頁面。這一章處理搜尋時一定會碰到的另外兩個專案。它們的安裝步驟不能和第 4 章混用。

12.1　什麼時候不該用 WebForms Core

協定適合「頁面結構大致固定，互動是改文字、改樣式、加幾列、停用按鈕」。下面這些情況，公開文件沒有把它說成目標，你也不該硬做：

- 整個畫面每秒都換版面。指令列會變得比 HTML 還長，除錯成本高過直接回傳一段片段。
- 團隊已經用 Blazor 或前端框架描述元件狀態。再加一套 DOM 指令，同一顆按鈕會有兩個主人。
- 你需要的是舊 Web Forms 的 `GridView`、設計工具與 `System.Web` 相容。那是下一節的移植，不是把 `GridView` 重寫成一百行 `AddTag`。

微軟給 Web Forms 開發者的現代預設仍是 Blazor：元件、事件、以及一份把舊概念逐項對過去的電子書。Razor Pages 適合表單不多、頁面為單位的站。這些是替換技術。WebForms Core 是另一種「伺服器描述畫面變化」的技術，不是微軟遷移指南裡的一步。

12.2　WebFormsForCore：舊站遷到.NET 10

官方站把步驟寫成一條直線。下面保留那些會讓專案失敗的約束，省略贊助與支援資訊。

1. 先備份非 SDK 風格的舊專案。用遷移工具轉成 SDK 專案。安裝與執行分開下：

   ```
   dotnet tool install -g Project2015To2017.Migrate2019.Tool
   dotnet migrate-2019 wizard
   ```

   若工具抱怨專案類型，先拿掉它點名的那個屬性再轉。

2. 目標框架改為 `net10.0`。若要和.NET Framework 雙重編譯，可以留著 `net48`。

3. `net10.0` 的輸出目錄必須是專案底下的直接子目錄。文件要求設成 `bin_dotnet`，並把 `AppendTargetFrameworkToOutputPath` 與 `AppendRuntimeIdentifierToOutputPath` 設為 `false`。原因是這套移植沿用經典 ASP.NET 對輸出路徑的假設。`net48` 仍輸出到 `bin`，而且 `OutputType` 維持程式庫；`net10.0` 則是可執行檔，`StartupObject` 為 `Program`。

4. 引用 `WebFormsForCore.Web`，文件頁面上的版本是 1.6.8。舊的 `Reference` 只留給 `net48`。

5. 在 `Program.cs` 用 `#if NETCOREAPP` 包住最小主機，呼叫 `app.UseWebForms()`。

```
#if NETCOREAPP
using Microsoft.AspNetCore.Builder;

public class Program
```

```
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);
        var app = builder.Build();
        app.UseWebForms();
        app.Run();
    }
}
#endif
```

預設只接住 Web Forms 自己的請求，例如 `.aspx`。其餘交給 ASP.NET Core。若站點用了友善網址與路由，改成：

```
app.UseWebForms(opt => opt.HandleAllRequestsWithWebForms());
```

工作階段要改用 ASP.NET Core 的 session，再告訴移植層使用它：

```
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession();

app.UseSession();
app.UseWebForms(opt => opt.UseAspNetCoreSessionProvider());
```

記憶體快取是單一行程。多機時換成文件所說的其他提供者，例如 Redis。

12.2.1 套件對照

舊組件名稱要換成移植套件，不能同時引用原來的 `System.Web`。

原來的組件	改引用
`System.Web`	`WebFormsForCore.Web`
`System.Web.Services`	`WebFormsForCore.Web.Services`
`System.Web.Extensions`	`WebFormsForCore.Web.Extensions`
`System.Web.Optimization` 與 Microsoft 的同名套件	`WebFormsForCore.Web.Optimization`
`System.Configuration`	`WebFormsForCore.Configuration`
`System.Drawing`	`WebFormsForCore.Drawing`
AjaxControlToolkit 與其 Sanitizer、靜態資源	同名前綴改成 `WebFormsForCore.`

`System.Drawing` 這份移植只實作屬性，好讓網站在沒有 `System.Drawing.Common` 的 Linux 上仍能啟動。不要把它當成完整的繪圖程式庫。

需要自動產生 `*.designer.cs` 時，再加 `WebFormsForCore.Build`，並把 `ExcludeAssets` 設成 `runtime`。文件說明它只處理 C#，不處理 Visual Basic，Visual Studio 裡的 Web 控制項視覺設計器也不能用。Build 套件還會在建置後拿掉舊組件上不相容的設計器屬性，讓某些參考了 `System.Web` 的第三方程式庫能載入型別。那些程式庫仍是對著.NET Framework 編譯的，載入成功不等於行為正確。

同一套 Build 提供類似 `AspNetCompiler` 的 MSBuild 工作 `AspNetCoreCompiler`，並有全域工具 `WebFormsForCore.AspNetCompiler`。

12.2.2 組件衝突

移植組件會取代 `System.Web`、`System.Drawing`，以及 `System.Configuration.ConfigurationManager`。文件要求在 `csproj` 裡加一個在參考組件解析之前執行的目標，把這三個檔名從 `ReferencePath` 移除，避免 NuGet 又把官方套件帶進來。條件用檔名與路徑比對。這段 XML 以官方站的範本為準，版本一變，正則也要跟著對。

先停一下

WebFormsForCore 跑起來之後，頁面仍是舊的控制項樹與 PostBack。不要在同一個按鈕上再掛 Elanat 的 `PostBack(this)`。兩套回傳會搶同一張表單。

12.3 WebFormsCore 框架

這一段只複述 `WebFormsCore/WebFormsCore` 公開 README 裡已經寫出的啟動方式。它是 ASP.NET Core 10 以上的框架，受 Web Forms 啟發，不是 `System.Web` 的二元相容層，也沒有 `[web-forms]` 協定。

手動建立時，專案 SDK 使用 `WebFormsCore.SDK.AspNetCore`。`Program.cs` 的形狀是：

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddWebFormsCore();
var app = builder.Build();
app.UseWebFormsCore();
app.MapPages();
app.Run();
```

`MapPages` 會從組件裡登記頁面。頁面檔用 `.aspx`，頁首帶 `Route`、`CodeBehind` 與 `Inherits`。README 寫明：沒有 `Route` 的頁面不會被 `MapPages` 公開，也可以改用 `MapPage<T>()` 或 `MapPage("/path", typeof(Page))` 手動對應。

README 另外點名兩項和經典模型相關的能力：用 `[ViewState]` 標在欄位或屬性上，由原始碼產生器寫入狀態；一頁可以有多個表單，各自帶自己的 view state；範圍表單在回傳時只處理該表單內的控制項。這些都說明它走的是「頁面與控制項」模型。

這一節要記住

三條路的第一個檔案就能分辨。看到 `web-forms.js` 與 `[web-forms]`，你在主線。看到 `UseWebForms` 與 `bin_dotnet`，你在移植舊站。看到 `AddWebFormsCore` 與 `MapPages`，你在 WebFormsCore。

代碼速查

除錯時先把回應第一行對上這一頁，再回去看方法名稱。定位符號的完整規則在第 6 章。

代碼	分組	值
nt, bt, ft, ut	加標籤	`TagName` 或 `TagName|Id`
at, pt, st, it	文字	文字；換行寫 `$[ln];`
aa, sa, ia, da	屬性	`名稱 | 值`；刪除時只寫名稱
ac, sc, ic, dc	class	類別名稱
as, ss, is, ds	style	樣式；刪除時只寫樣式名
av, sv, iv, dv	value	值；刪除時值為 `1`
an, sn, in, dn	name	名稱
ai, si, ii, di	id	id
al, sl, il, dl	title	標題
ao, so, io, do	option	`值 | 文字 |0 或 1`；刪除時可寫 `*`
ak, sk, ik, dk	checkbox	同上
bc, tc, fn, fs, fb	外觀	顏色、字型、字級、粗體 `0/1`
sw, sh, ta, vi	盒與可見	寬、高、對齊、可見 `0/1`
sd, sr, sf	表單狀態	停用、唯讀、聚焦
mx, mn, ts, ti	輸入限制與選取	長度、選取值、選取索引
+v +h +w +f +x +n	增加	數字
-v -h -w -f -x -n	減少	數字
: 秒) 與 (秒)	前置	延遲、週期，寫在動作碼前
Ep, Eg, Et	事件	見第 7 章
AS	歷史	`路徑 | 標題`
_	腳本	腳本原文

公開來源

下面這些頁面是 2026-10-16 整理時實際讀過的。版本列在前面的章節裡；頁面若已更新，以頁面為準。

参考文献

[1] Elanat, *Using WebForms Core in CodeBehind Framework*.
https://elanat.net/page_content/documentation/code_behind_framework_tutorial_series/using_web_forms_core

[2] Elanat, CodeBehind tutorial series, form submit and file fields.
https://elanat.net/page_content/documentation/code_behind_framework_tutorial_series/send_data_by_submitting_form

[3] elanatframework/Web_forms, *WebFormsJS* README: options, action codes, selectors, multiple responses, PostBack and GetBack.
https://github.com/elanatframework/Web_forms

[4] elanatframework/Web_forms_classes, protocol overview and the C# class as the parent implementation.
https://github.com/elanatframework/Web_forms_classes

[5] NuGet, *CodeBehind* 4.8.1 package page: `Program.cs`, view syntax, `net10.0`.
https://www.nuget.org/packages/CodeBehind/

[6] NuGet, *WFC*, server-side package used outside CodeBehind.
https://www.nuget.org/packages/WFC/

[7] Elanat, *State Management in WebForms Core 2 (Razor Pages Example)*.
https://dev.to/elanatframework/state-management-in-webforms-core-2-razor-pages-example-204f

[8] WebFormsForCore documentation site, migration steps and package list, package version 1.6.8 on the fetched page.
https://webformsforcore.github.io/

[9] webformsforcore/WebFormsForCore source repository.
https://github.com/webformsforcore/WebFormsForCore

[10] WebFormsCore/WebFormsCore README: template, `MapPages`, view state notes.
https://github.com/WebFormsCore/WebFormsCore

[11] Microsoft Learn, *Migrate from ASP.NET Web Forms to Blazor*.
https://learn.microsoft.com/dotnet/architecture/blazor-for-web-forms-developers/migration

[12] Microsoft Learn, *Upgrade an ASP.NET Framework Web Forms app to ASP.NET Core MVC*.
https://learn.microsoft.com/aspnet/core/migration/web_forms

[13] .NET Blog, *Blazor for ASP.NET Web Forms Developers*, free e-book announcement.
https://devblogs.microsoft.com/dotnet/blazor-aspnet-webforms-ebook/

讀完之後

若只能留下三個習慣：

1. 每次互動都打開網路面板，先讀 `Post-Back` 標頭，再讀正文第一行是不是 `[web-forms]`。
2. 指令的定位用 id，或用剛剛建立的節點配負索引。少用「頁面上第一個 `form`」這種會被版面改動打亂的寫法。
3. 重新整理會回到 View 的 HTML。要留下的狀態請寫進伺服器，或明確呼叫 `AddState`。不要假設有 ViewState。

下一份你自己的頁面，建議仍從第 11 章的聯絡表單複製，而不是從移植舊站的專案複製。等這頁的 INI 你能脫稿解釋，再去看 WebFormsForCore 的 `bin_dotnet` 約束。那時候兩套 PostBack 就不會再混成同一個詞。