React Concurrent Mode，獻給被 Deadline 緊緊相逼，仍不願跟 UX Say Goodbye 的你

撰文｜叡揚資訊 前端設計部 前端工程師 蘇子翔

今年十月，React Conf 2019 傳出了令人興奮的消息——Concurrent Mode、Suspense for Data Fetching 終於面世。React Fiber 計畫鋪的路，要開始綻放光芒了。

在前後端分離的現在，前端工程師們需要頻繁地呼叫 API 拿取資料，衍生出各種呼叫 API 的模式，以及大量的 Loading 狀態和 Race Condition 的風險。

沒經驗的工程師會被各種詭異的顯示結果要得團團轉，而就算是有經驗的工程師，每個人都有各自習慣的拿取資料模式，並沒有統一規範，造成一個問題有多種解法。不同解法帶來的使用者體驗不盡相同（例如資料要一次全拿，還是分批拿）最終導致產品的功能體驗缺乏一致性。

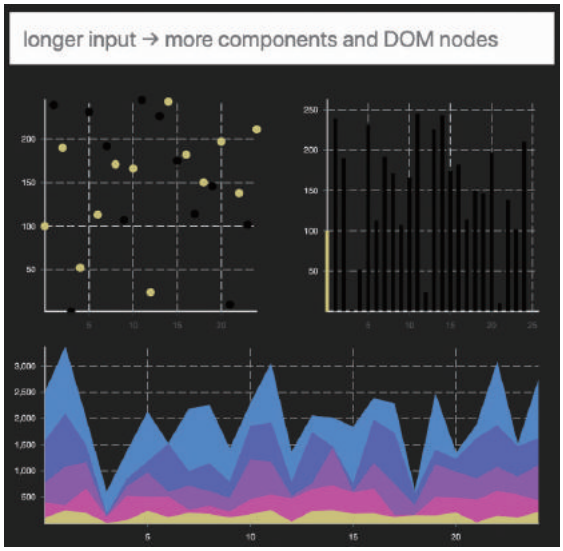
在眾多狀態中，Loading 狀態最容易被忽略，卻對使用者體驗至關重要。從資料回來前，該怎麼保持畫面的完整性、降低使用者的感知時間，到資料回來後，要如何不造成畫面的跳動或閃爍，其中有許多細節需要注意。甚至好不容易抓回資料，卻在使用者進行操作時，因資料筆數過多，導致運算速度跟不上而感到卡頓。這時該用防抖抑或節流，又是一個典型的前端回答，It depends.。

追求良好使用者體驗的道路上充滿荊棘，於是 React 射出了三發銀色子彈：

畫面不卡頓

畫面卡頓的起因，通常是因為 State 改變造成的渲染，阻止了畫面立即更新。例如下圖，當使用者輸入文字時，下方的圖表會跟著變化，大量的 DOM 變更，於是阻斷了輸入框即時更新文字。以往我們會有 setTimeout、防抖、節流等方式優化使用者體驗，但仍能感受到卡頓，而 Concurrent Mode 能透過中斷、暫停渲染解決這個問題。

背後的原理是呼叫原生的 Web API——requestIdleCallback，但這 API 本身支援度不高，因此 React 自己去做 Polyfill，來實現 Concurrent 的效果。



資料和畫面同步渲染

要讓資料和畫面同步渲染，需要 Suspense 和這次的新 Hook——useTransition 配合才能做到。

分成兩個步驟：首先是拿資料，接著則是處理等待狀態（Loading）。

第一個步驟需要 Suspense for Data Fetching——之前 Suspense 只能載入程式碼，現在可以載入資料了。

Data Fetching 的方式有三種

- **Fetch-on-Render：**畫面渲染之後才去呼叫 API，體驗和效率最差，因為會有 Waterfall（第一支API 回來之後，State 改變，引起 component 渲染，再呼叫下一支 API，等待 API 的時間會累加）和 Race Condition（畫面的渲染和發出的 request 脫鉤，若 request 回來前就發生下一次渲染，會產生 Race Condition）。

- **Fetch-Then-Render：**資料全部拿完再渲染，GraphQL 生態系比較容易做到，不會有 Waterfall，但第一次獲取資料的時間會比較久。

- **Render-as-You-Fetch (使用 Suspense)：**在 Render 之前儘早開始抓資料，並立刻開始 Render 下一個頁面。這時資料若還沒回來，就會進入 Suspense 的狀態；等資料回來後，React 會重新渲染。這種方式不會有 Race Condition，推薦用這種方式拿取資料。

Data Fetching 後，接著要處理等待狀態，此時就需要 useTransition。這是因為在拿資料的過程中，不可避免地需要等待，因此通常需要處理以下問題：

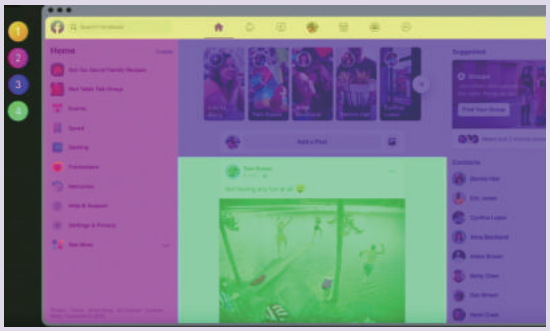
- **改善換頁體驗：**在完整頁面點擊連結換頁，畫面會瞬間變成 Loading 狀態，此時就算資料很快就回來，頁面還是會出現閃爍。這時

我們可以透過 useTransition 讓畫面停在原本的頁面，等下一頁的資料回來再渲染，這樣就不需要處理中間的 Loading 狀態和閃爍。

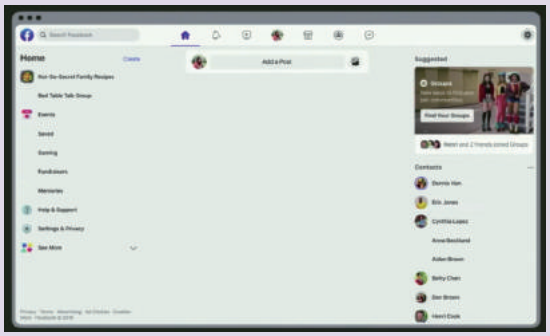
- **改善 Loading 畫面：**當頁面上有許多 Component 同時 Loading，會導致畫面過於瑣碎，此時透過 Suspense Boundary 能結合複數個 Loading 區塊，讓範圍內的 Component 同時出現，組合成有意義的區塊。
- **改善資料顯示順序：**同一頁面呼叫的 API 會在不同的時間回來並造成頁面閃爍，可以傳給 SuspenseList 一個 revealOrder 參數，讓每個 Component 同時或按照順序出現，減少畫面抖動。

透過 Suspense 和 useTransition 的配合，可以讓 Component 在所有資料到達之前就開始在記憶體中渲染，減少 Loading 狀態，避免畫面不停跳動，提升使用者體驗。

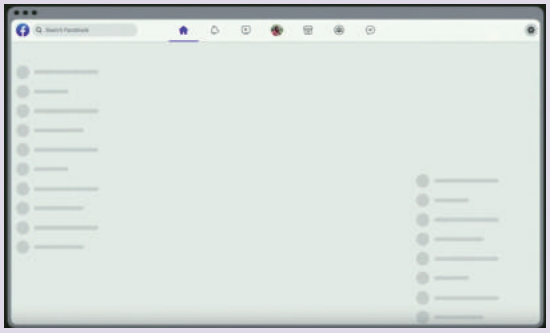
以 FB 的新介面為例



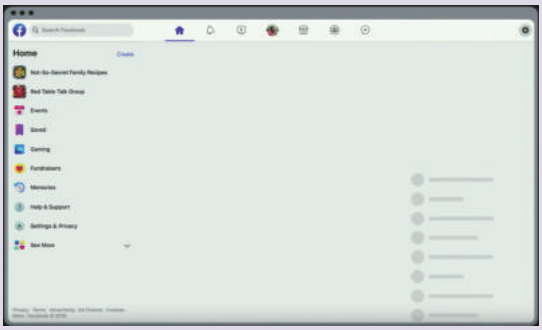
1 由於我們的閱讀習慣是由左到右，由上到下，所以理想的渲染順序是 1→2→3→4，並且是接續出現——在前一個區塊完全載入之前，先暫時不顯示下一個區塊的內容；我們希望 Component 能按照我們所編排的順序出現，而不是從頁面的各個角落隨機出現，分散使用者注意力，中斷閱讀體驗。



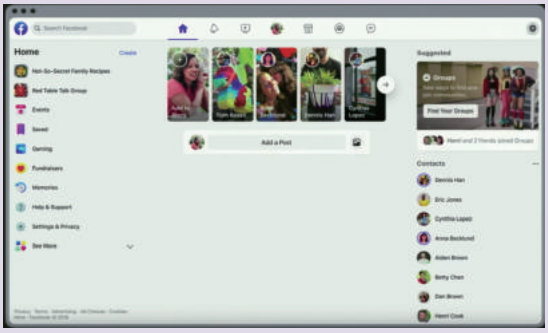
2 以往資料回來的順序無法控制，造成 Component 隨機出現在畫面上，使用者必須經歷破碎、閃爍和跳動的畫面才能看到完整內容，閱讀體驗不斷被突然出現的 Component 打斷，心中充滿了不知道會不會再有 Component 出現的不安定感。



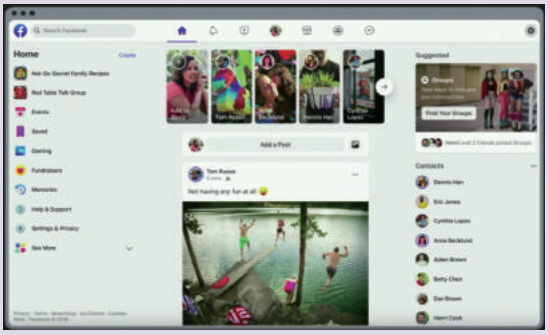
3 在有了 Suspense 之後，讓我們開始優化的第一步——避免使用者看到破碎的頁面。透過 Suspense Boundary 讓側邊欄的資料全都取得後再顯示，在那之前統一用 Loading 狀態包起來。既可以避免各個小型 Component 都出現自己的 Loading 狀態，讓畫面變得瑣碎，還可以讓使用者知道這個區塊的資料會一起出現，提升安定感。



4 從左圖我們可以看到，在左側邊欄的內容全部回來之前，右側邊欄持續保持在 Loading 狀態，使用者可以舒服地按照閱讀順序從左到右閱讀，而不會被忽左忽右出現的 Component 干擾。



5 再來要避免畫面跳動。如果下方的 Post 內容先出現，當上方動態消息出現時，畫面會產生跳動；我們必須確保下方的 Post 不早於上方的動態消息出現。所以我們傳入 revealOrder，編排區塊載入順序，讓編號 3 的區塊渲染完之後，才顯示編號 4 的區塊，以避免畫面向下跳動。



6 讓我們重看一次優化後的畫面渲染過程。首先，我們把小區域的 Loading 狀態整合成區域 Loading，讓畫面不再有破碎感；其次，我們讓左側邊欄渲染完成後，再開始渲染右側邊欄，減少畫面左右同時有 Component 出現的不安定感；最後我們編排畫面出現順序，確保編號 3 不早於編號 4 出現，降低畫面跳動。完成了以上優化之後，我們終於有信心說：我們帶來了滑順的使用者體驗！

完整的解決方案

以往要避免畫面卡頓，並讓資料、畫面同步渲染，需要各式各樣的特殊解法，而這些解法往往都只能針對特定情境做優化，或是添加冗長的程式碼，無法廣泛使用到各個專案中。

之前若要做到畫面按照編排順序出現，我們可能需要使用大量的 promise，當編號 1 的區塊資料回來後，再開始拿編號 2 區塊的資料，這不僅造成使用者等待時間增加，程式碼也冗長。但 React 這次的更新，讓開發者可以用簡潔的語法，就能達到效果，讓使用者體驗的優化可以跨專案且長期存在，也不會造成維護上的困難。

結語

“A great developer experience only matters if it’s in service of delivering a great user experience.” - Tom Occhino

現代前端開發日趨成熟且複雜，想用心優化使用者體驗，往往意味著注意力要分散到構建配置和優化，而不是業務需求，在排優先順序時多被忽略或推遲；但這次 React 的更新，把使用者體驗研究的成果融入框架中，同時提升開發者和使用者體驗。雖然目前的生態系要跟上 Concurrent Mode 還有漫漫長路要走，但至少我們往正確的方向邁出了一步。