**一、相關完整程式碼檔案清單 Dir structure of Lotus**

**lotus-v1/**

├── crawler/

│ ├── fetch\_\*.js # 各新聞來源爬蟲（如 fetch\_macaodaily\_ws\_cb.js）

│ ├── modules/ # 共用後端模組

│ │ ├── logger.js # 全域日誌模組

│ │ └── historyManager.js # 歷史與 last\_updated 處理

│ └── ...

├── cardjs/ # 前端卡片核心程式

│ ├── \_card\_core.js # 卡片渲染共用邏輯

│ ├── macaodaily\_card.js # 各新聞來源卡片

│ ├── gcs\_card.js

│ └── ...

├── data/

│ ├── fetch\_\*.json # 主新聞資料

│ ├── his\_fetch\_\*.json # 各來源歷史資料

│ ├── HIS\_ALL.json # 全部來源的新聞彙總

│ ├── last\_updated.json # 更新狀態記錄

│ ├── overalllog.json # 日誌記錄

│ └── ...

├── css/ #前端樣式

│ └── \_common\_card.css # 卡片共用樣式

├── index.html #主頁面

├── server.js # 簡易 API 伺服器

└── package.json

## 2. Naming rule

* **爬蟲邏輯檔**：fetch\_xxxx\_ws.js
* **爬蟲主檔 (整合前端)**：fetch\_xxxx\_ws\_cb.js
* **主新聞 JSON**：fetch\_xxxx.json
* **歷史新聞 JSON**：his\_fetch\_xxxx.json
* **全域歷史檔**：HIS\_ALL.json
* **卡片 JS**：xxxx\_card.js
* **last\_updated.json key**：xxxx（與來源同名小寫）

## 3. 路徑/JSON 對照表

| **來源** | **fetch 檔** | **主 JSON** | **歷史 JSON** | **HIS\_ALL** |
| --- | --- | --- | --- | --- |
| macaodaily | fetch\_macaodaily\_ws\_cb.js | fetch\_macaodaily.json | his\_fetch\_macaodaily.json | HIS\_ALL.json |
| gcs | fetch\_gcs\_rss\_cb.js | fetch\_gcs.json | his\_fetch\_gcs.json | HIS\_ALL.json |
| aamacau | fetch\_aamacau\_ws\_cb.js | fetch\_aamacau.json | his\_fetch\_aamacau.json | HIS\_ALL.json |
| exmoo | fetch\_exmoo\_ws\_cb.js | fetch\_exmoo.json | his\_fetch\_exmoo.json | HIS\_ALL.json |
| … | … | … | … | HIS\_ALL.json |

**規則**：his\_fetch\_來源.json、fetch\_來源.json、fetch\_來源\_ws\_cb.js

## 4. 格式說明

### (A) overalllog.json

[

"[2025-07-06 11:29] [預覽] 👀 fetch\_jtm\_ws\_cb.js {\"title\":\"xxx\",...}",

"[2025-07-06 11:29] [成功] ✅ fetch\_jtm\_ws\_cb.js 共新增 0 條，已寫入 ./data/his\_fetch\_jtm.json",

"[2025-07-06 11:29] [成功] ✅ fetch\_jtm\_ws\_cb.js 共 13 則新聞已存至 ./data/fetch\_jtm.json",

"[2025-07-06 11:29] [INFO] ℹ️ fetch\_jtm\_ws\_cb.js 下載 Markdown：https://xxx"

]

### (B) last\_updated.json

*{*

*"gcs": {*

*"fetch": "gcs",*

*"lastRun": "2025-07-06 13:02:00",*

*"lastSuccess": "2025-07-06 13:02:00",*

*"lastAdded": "2025-07-06 13:02:00",*

*"lastOperation": "scheduled",*

*"lastManual": null*

*},*

*...*

*}*

### (C) Terminal 日誌格式

直接與 overalllog.json 完全同步，彩色版（如支持）

## 使用者流程步驟

|  |  |  |  |
| --- | --- | --- | --- |
| Action **1. OPEN THE INDEX.HTML** | | | |
| 1.1瀏覽器取得 index.html，並依序載入 CSS、Day.js、核心與來源模組。1.2 初始化卡片 DOM: \_card\_core.js 呼叫 cardInit()，在對應的 <div id="…-card"> 中插入卡片結構（標題、🔄 按鈕、.news-list、.card-status 等）1.3載入歷史資料（his\_<source>.json） | | | |
| 1.3- caseA: **若檔案存在且非空**  **Then,**  1.3A.1 顯示歷史新聞列表（最多 show 15 筆）  1.3A.2 Check the fetch condition (判斷是否立即抓取) by計算 *diff = now − lastSuccess:* | | 1.3- case B: **若檔案不存在或空** | |
| **Case** diff ≥ autoRefresh  then, 立即觸發 fetchAndUpdate(force=true) | **Case** diff < autoRefresh  then, startCountdown((autoRefresh − diff)×60 秒) | 1.3B.1留空列表，狀態欄顯示：“等待新聞中”  1.3B.2立即觸發 fetchAndUpdate(force=true)  1.3B.3 load and show the news in the fetch\_\*\*\_cb.json | |
| **Action 2: 手動觸發 fetch (click the refresh bottom)/ 自動觸發 fetch (倒數完成)** | | | |
| 立即觸發 fetchAndUpdate(force=true) | | | |
|  | **Case手動觸發 fetch (click the refresh bottom)/ 自動觸發 fetch (倒數完成)** | **Case 瀏覽器刷新**  then, Check the fetch condition (判斷是否立即抓取) by計算 *diff = now − lastSuccess:* | |
|  | then, 立即觸發 fetchAndUpdate(force=true)  lastupdatetime (**最後更新**) reset  nextupdatetime (下次**更新**) reset | **Case** diff ≥ autoRefresh  then, 立即觸發 fetchAndUpdate(force=true)  lastupdatetime (**最後更新**) reset  nextupdatetime (下次**更新**) reset | **Case** diff < autoRefresh  then, startCountdown((autoRefresh − diff)×60 秒)  lastupdatetime (**最後更新**) remain  nextupdatetime (下次**更新**) remain |

## 使用者流程步驟（詳細版）

1. **啟動伺服器／開啟前端頁面**
   * 使用者執行 node server.js 啟動後端 API。
   * 在瀏覽器開啟 index.html，自動載入所有卡片模組。
2. **卡片自動初始化**
   * 前端每張卡片（news card）在載入後即刻呼叫 cardInit。
   * 各卡片自動讀取對應的 JSON 檔，顯示最新新聞與狀態。
3. **自動倒數／自動刷新**
   * 每張卡片有自己的自動倒數計時器，倒數至「下次更新」。
   * 倒數結束後自動呼叫對應爬蟲（如 /run-fetch?script=fetch\_xxx\_ws\_cb.js），後端執行新聞抓取、資料存檔、狀態更新。
4. **手動刷新操作**
   * 使用者可點擊任一卡片的 [手動刷新] 按鈕，立即觸發該卡片的新聞爬蟲。
   * 手動刷新時，更新倒數計時，狀態顯示「手動刷新中」。
5. **更新流程與例外處理**
   * 成功抓取：更新主 JSON、歷史 JSON、HIS\_ALL、last\_updated、overalllog。
   * 抓取失敗（如 DNS、HTTP 錯誤）：顯示錯誤訊息，5 分鐘後自動重試，且不覆蓋主檔。
   * 所有操作（自動／手動）皆會記錄於 overalllog.json，方便日後查錯。
6. **頁面即時更新**
   * 當資料有更新時，卡片會即時刷新內容與「最後更新」時間標籤。
   * 狀態欄即時顯示「剛剛更新」、「幾分鐘前更新」、「DNS 失敗」等狀態。

## Fetch mechanism

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **fetch 成功後 (有新新聞)** | **fetch 成功後 (沒有新新聞)** | | | **fetch 失敗（DNS 等）** |
| **新聞列表顯示** | **先**顯示歷史（his），再顯示最新（fetch） | | | | 保留原有清單 |
| nextupdatetime (下次**更新**) | reset | | | | DNS 失敗，於 05:00 後重試 |
| lastupdatetime (**最後更新**) reset | reset並顯示「剛剛更新」 | | | | 保留原有顯示 |
| In fetch\_”source”.json | Record the | | **沒有**更新 |  | |
| IN HIS\_ALL.json | 更新 | | **沒有**更新 | **沒有**更新 | |
| IN his\_”source”.json | 更新 | | **沒有**更新 | **沒有**更新 | |
| IN overalllog.json | Record every steps | | Record every steps | Record every steps | |

* **自動排程**
  + 每個 fetch\_xxx\_ws\_cb.js 依據 autoRefresh 參數，於固定時間由前端自動呼叫後端 /run-fetch 端點，執行資料抓取。
  + 根據 last\_updated.json 記錄，自動跳過「剛抓過」的來源（避免重複頻繁觸發）。
* **手動刷新**
  + 使用者按下「刷新」時即時呼叫，**無論上次更新時間為何**。
* **去重／歷史補足**
  + 每次抓取先抓主檔，若不足則補歷史檔 (his\_fetch\_xxx.json)。
  + 新聞寫入時自動去重，歷史資料最多保留 1000 條。
* **錯誤備援**
  + 主站連不上（DNS、403、timeout）自動切換備援方式（如 HTML→Markdown fallback）。
  + 異常時不更新主檔，只顯示錯誤並重試。
* **日誌同步**
  + 所有步驟自動記錄到 overalllog.json，格式統一。
  + 實時顯示在 Terminal 及前端（若需要）。
* **更新記錄**
  + 每次新增或寫入都同步至 last\_updated.json，分別紀錄 lastRun, lastSuccess, lastAdded, lastManual, lastOperation。

**需求文件（Requirement Document）**

## 一、專案概覽

* **目的**：定期自動與手動抓取多個新聞來源，並在前端以「卡片」形式展示最新新聞。
* **重點功能**：
  1. 各來源自動定時抓取
  2. 每來源有獨立 UI 卡片
  3. 支援一鍵刷新/自動倒數刷新
  4. 主/歷史/全域匯總/狀態/日誌多檔資料結構
  5. 具備 DNS/來源失敗備援，容錯與提示
  6. 所有關鍵行為皆有日誌記錄

## 二、後端爬蟲（crawler/fetch\_\*\_cb.js）

1. **檔名規則**
   * fetch\_<source>\_cb.js（cb = callback/API 觸發 者）
   * 歷史檔：his\_fetch\_<source>\_cb.json
   * 主資料檔：fetch\_<source>\_cb.json
2. **時區、時間格式**
   * Asia/Macau，所有日期標準化 YYYY-MM-DD HH:mm:ss
3. **自動 vs 手動**
   * **自動排程**：每 N 分鐘（默認 N=5）由排程器或定時呼叫 node fetch\_\*.js 執行。
   * **手動刷新**：前端按鈕呼叫後端 /run-fetch?script=...&force=1 強制執行。
   * **排程與手動刷新分流**：明確標記是自動還是手動
4. **UI/前端規範**
   * 每卡片獨立，狀態顏色與訊息格式統一
5. **執行前檢查**
   * 對「首次載入」或「瀏覽器重載」：比較 last\_updated.json 中 lastSuccess 與當前時間，< N 分鐘則跳過爬蟲，> N 分鐘則執行。
   * 按鈕或排程觸發時永遠強制執行。
6. **DNS 失敗處理**
   * 若主站（或 RSS）fetch/goto 出現 ERR\_NAME\_NOT\_RESOLVED，
     + 不更新主 JSON、history、last\_updated。
     + 日誌記錄「DNS 失敗；於 MM:SS 後重試，原排程延至 N+5 分鐘」。
     + 在前端卡片以倒數形式顯示「DNS 失敗，於 MM:SS 後重試」。
7. **資料擷取**
   * **澳門日報**：使用 Puppeteer 抓取動態頁面，取出前 MAX\_NEWS（19）則新聞。
   * **GCS RSS**：以 fetch 取得純 RSS XML，解析 <item>、<title>、<link>、<pubDate>。
8. **檔案寫入**
   * 主 JSON：fetch\_<source>\_cb.json
   * 歷史 JSON：his\_fetch\_<source>\_cb.json（保留最新 1000 筆）
   * 執行紀錄：last\_updated.json
   * 全域日誌：overalllog.json
9. **日誌格式**
   * 時區固定：Asia/Macau
   * 檔頭註解：// <filename> -GPT-<version>-<MMDD-HHmm>
10. **維護性**：所有程式必有註解、易於擴充

## 三、資料格式

* **主 JSON**（fetch\_<source>\_cb.json）

*[*

*{ "title": "...", "pubDate": "...", "link": "..." },*

*...*

*]*

* **歷史 JSON**（his\_fetch\_<source>\_cb.json）
  + 最新 newCount 筆放最前，最多 1000 筆。
* **最後更新**（last\_updated.json）

*{*

*"<source>": {*

*"lastRun": "YYYY-MM-DD HH:mm:ss",*

*"lastSuccess":"YYYY-MM-DD HH:mm:ss",*

*"lastManual": "YYYY-MM-DD HH:mm:ss" // 按鈕觸發才更新*

*},*

*...*

*}*

* **全域日誌**（overalllog.json）
  + 陣列，每行為一條字串日誌，與 Console 同步。

## 四、後端 Server & 排程

1. **server.js**
   * 提供 /data 靜態目錄服務。
   * 提供 /run-fetch API:
   * **G**ET /run-fetch?script=fetch\_<source>\_cb.js&force=01&interval=N
   * 觸發對應的 fetch\_\*\_cb.js 子程序並回傳執行結果。
2. **排程**
   * 使用 node-cron 或外部排程（如 PM2、Heroku Scheduler）呼叫 /run-fetch HTTP 端點。
   * 內部也可用 setTimeout 在每次執行後自動排下次。

## 五、前端卡片（cardjs/\*.js + css/\*.css）

1. **卡片參數**

*import { cardInit } from './\_card\_core.js';*

*cardInit({*

*cardId: 'aamacau-card',*

*key: 'aamacau',*

*title: '論盡澳門',*

*jsonPath: '/data/fetch\_aamacau.json',*

*fetchScript: 'fetch\_aamacau\_ws\_cb.js',*

*show: 15,*

*max: 30,*

*autoRefresh: 15,*

*tag: '#澳門 #即時',*

*backgroundColor: '#B5495B',*

*scrollThumb: '#F2C7C7',*

*scrollTrack: '#B5495B'*

*});*

1. **功能**
   * **載入時**：讀 last\_updated.json 判斷是否要自動 fetch。
   * **手動刷新**：點擊按鈕後強制 fetch，更新畫面並重設倒數。
   * **自動倒數**：每秒更新「還剩約 MM:SS 將更新」，到 0 自動呼叫 fetch。
   * **錯誤顯示**：若 fetch 失敗（含 DNS），顯示「更新失敗」或「DNS 失敗，於 MM:SS 後重試」。
2. **UI 要求**
   * **標題**
     + 顯示網站名稱（如「澳門日報」、「政府新聞網聚合」）
     + 字體「花園明朝體」，純白色，字重 Bold，大小預設 20px，之後可依需求縮放。
   * **字體：**
     + 全站採用「花園明朝體」（HanaMin）＋「Noto Sans TC」
     + 卡片標題字重 800、大 100%
     + 新聞列表文字字重 normal，字體大小為標題的 ~70%
   * **顯示：**
     + 卡片標題 + 刷新按鈕 + 圓圈數字
     + 最後更新時間
     + tag
     + 新聞列表（排行、自動/手動鏈結、日期格式化）
     + 狀態列（倒數或提示文字）
   * **版面：**
     + 背景色、圓形數字、標題/清單字重、字體、捲軸樣式依設計圖。
     + 每張卡片底部皆有 .card-status 顯示：倒數或錯誤提示。

╭─────────────────────────────╮

│ 澳門日報         │

│ #澳門 #即時              1 分鐘前更新│

│    │

───────────────────────────────│

│ 1 新聞標題 …      剛剛  │

───────────────────────────────│

│ 2 新聞標題 …            2小時前│

…（可滑到最多22則）                                     │

───────────────────────────────│

│ 還剩約 03:47 將更新                                   │

╰─────────────────────────────╯

1. **通用 Core 或 獨立**
   * 可用單一 \_card\_core.js 或每張卡 macaodaily\_card.js、gcs\_card.js 獨立實作。
   * 獨立時彼此無相依，互不影響。

**六**、**功能行為（Behavior）**

* + **自動刷新週期**：每 N 分鐘（預設 5）自動觸發一次爬蟲。
  + **手動刷新**：點擊「🔄」強制執行爬蟲，不受時間限制，並重設倒數。
  + **首次載入 / 瀏覽器重整**：
    - 若距上次成功時間 < N 分鐘 → 只載入現有 JSON，不執行爬蟲，直接啟動剩餘倒數
    - 否則 → 先載入現有 JSON→ 觸發一次強制爬蟲→ 重設倒數
  + **DNS／網路失敗處理**：
    - 若爬蟲遇到 ERR\_NAME\_NOT\_RESOLVED → 保留舊 JSON、不更新檔案
    - 於「MM:SS 後重試」顯示倒數；於日誌標明「DNS 失敗，於 MM:SS 後重試，原排程延至 N+5 分鐘」
  + **狀態欄顯示**：
    - 正常自動倒數：還剩約 MM:SS 將更新
    - 手動刷新中：正在抓取最新新聞…
    - DNS／API 失敗：DNS 失敗，於 MM:SS 後重試 或 更新失敗

1. **後端爬蟲（fetch\_\*\_cb.js）**
   * **輸出 JSON**：data/fetch\_<source>\_ws\_cb.json
   * **歷史檔**：改名為 data/his\_fetch\_<source>\_ws\_cb.json，用於判斷新增數
   * **時間記錄**：集中寫入 data/last\_updated.json，包含 lastRun、lastSuccess、lastManual
   * **整合日誌**：data/overalllog.json 記錄每次 INFO/SUCCESS/ERROR，均以「Asia/Macau」時區
   * **備援機制**：對於 RSS 類型，若主站失敗自動切 Jina 轉送
2. **前端整合**
   * cardjs/\_card\_core.js 提供共用初始化函式 cardInit({...})
   * 於 index.html 依序呼叫兩個卡片：

*cardInit({*

*cardId:    'macaodaily-card',*

*jsonPath:  '/data/fetch\_macaodaily\_ws\_cb.json',*

*show:      10,*

*max:       19,*

*autoRefresh: 5,*

*tag:       '#澳門 #即時'*

*});*

*cardInit({*

*cardId:    'gcs-card',*

*jsonPath:  '/data/fetch\_gcs\_rss\_cb.json',*

*show:      10,*

*max:       35,*

*autoRefresh: 5,*

*tag:       '#官方 #政府'*

*});*

* + 同步讀取 last\_updated.json 計算「最後更新」與判斷是否自動觸發。
  + 點擊刷新按鈕觸發 /run-fetch?script=… API 呼叫，後端執行對應爬蟲。

**Global Rule**

1. **時區統一**
   * 1. 後端日誌與記錄（logger.js、historyManager.js、last\_updated.json、overalllog.json）皆以「Asia/Macau (GMT+8)」為基準。
     2. 前端卡片所有時間計算與顯示，也同樣使用 Macao 時區。
2. **檔案命名規則**
   * 1. 主資料 JSON：fetch\_<source>\_cb.json
     2. 歷史資料 JSON：his\_<source>\_cb.json
     3. 最後執行時間：last\_updated.json
     4. 綜合日誌：overalllog.json
3. **爬蟲執行邏輯**
   * 1. **開頁／手動刷新** → 強制執行一次爬蟲，無視上次時間；執行後重設倒數計時。
     2. **自動排程** → 每 N 分鐘觸發（N 由卡片設定，預設 5 分鐘）；觸發前會比對 lastSuccess，若距離低於 N 分鐘不跑爬蟲，直接重啟倒數。
     3. **DNS 失敗重試** → 若抓取主站失敗，立即排程「5 分鐘後重試」，並將原自動排程往後推 N+5 分鐘。卡片顯示「DNS 失敗，於 MM:SS 後重試」。
4. **後端模組化**
   * 1. logger.js：提供 logInfo、logSuccess、logError、logPreview，所有日誌附上 [YYYY-MM-DD HH:mm:ss]、狀態標籤、Script 名稱，並寫入 overalllog.json。
     2. historyManager.js：提供 saveHistoryAndUpdateLast(data, key, historyPath, lastUpdatedPath)，同時更新歷史與 last\_updated.json。
5. **程式碼註解與版本**
   * 1. 每支 JS/Node/ESM 模組文件最上方需加：

// <filename> - <MODEL\_VERSION>-<MMDD-HHMM>

* + 1. 檔尾需加：

// the end of <filename> -GPT-<MODEL\_VERSION>-<MMDD-HHMM>

1. **前端卡片結構**
   * 1. 每張卡片獨立一支 cardjs/<source>\_card.js、一份 css/<source>\_card.css。
     2. index.html 只要引入對應 HTML 結構與 cardInit({...}) 呼叫即可，不會互相干擾。
2. **前端時間顯示規則**
   * 1. **最後更新**：X 分鐘前更新（<1 分鐘 顯示「剛剛更新」）。
     2. **新聞發佈時間**（GCS 卡）：
        1. <24 小時 → X 小時前
        2. ≥24 小時 → MM-DD
3. **CSS 樣式守則**
   * 1. 卡片主背景色由各來源自行決定（澳門日報 #B5495B、GCS #FFB11B …）。
     2. 全站字體統一「花園明朝體」或「Noto Sans TC」；新聞列表文字用正常字重，標題比正文大 100%，列表文字縮小 30%。
     3. 卡片狀態欄 .card-status 必須貼在卡片底部，顯示倒數、抓取中或錯誤提示。
4. **依賴管理**
   * 1. 後端：puppeteer、dayjs（含 utc、timezone）。
     2. 前端：直接透過 <script type="module"> 引入同專案內 /lib/dayjs.min.js、utc.js、timezone.js 作時區運算，不再依賴 CDN。
5. **不破壞既有功能**
   * 1. 所有新增或修改絕不移除舊有功能。
     2. 任何變更前皆須思考對先前卡片、爬蟲腳本與日誌機制的影響。

**global development rule**

1. **時區統一**
   * 所有前後端顯示、記錄、日誌時間均以「Asia/Macau」（GMT+8）為準。
2. **日誌機制**
   * 後端所有腳本（fetch\_\*\_cb.js）必須使用 logger.js：
     + logInfo(script, msg)、logSuccess(script, msg)、logError(script, msg)、logPreview(script, item)。
   * 日誌輸出格式：[YYYY-MM-DD HH:mm:ss] [LEVEL] [ICON] script.js 訊息，並追加至 data/overalllog.json。
3. **歷史資料**
   * 歷史檔一律命名 his\_<script-base>.json（例：his\_fetch\_macaodaily\_ws\_cb.json），存放最新 1000 筆。
   * 透過 saveHistoryAndUpdateLast(data, key, hisPath, lastPath) 寫入歷史與更新時間。
4. **更新時間紀錄**
   * last\_updated.json 結構：

{

  "<key>": {

    "lastRun": "YYYY-MM-DD HH:mm:ss",

    "lastSuccess": "YYYY-MM-DD HH:mm:ss",

    "lastManual": "YYYY-MM-DD HH:mm:ss"  // 手動刷新專用

  }

}

* + 僅在成功抓取後更新 lastRun/lastSuccess；手動刷新時更新 lastManual。

1. **爬蟲排程邏輯**
   * 定期自動排程間隔 N 分鐘。
   * **手動刷新（點擊按鈕）**：無視時間差，**強制**執行爬蟲 → 重設倒數。
   * **自動排程**：若距離 lastSuccess ≥ N 分鐘則執行，否則略過，繼續倒數至下次。
2. **DNS 失敗處理**
   * 若主站 DNS 解析失敗，**不更新**主/歷史 JSON，日誌記錄「DNS 失敗」＋「於 MM:SS 後重試；原排程延後 N+5 分鐘」。
   * 卡片顯示狀態「DNS 失敗，於 MM:SS 後重試」。
3. **前端模組化**
   * 每張新聞卡獨立一支 cardjs/<name>\_card.js 並配對 <name>\_card.css。
   * index.html 只需引入對應 CSS 路徑與呼叫 cardInit({ … })。
4. **前端倒數 & 更新**
   * 每張卡：倒數計時用 setInterval（秒級更新）；到 0 後自動觸發 fetch。
   * 點擊刷新按鈕：觸發 fetch → 重新載入 JSON → 重設倒數 & 更新「最後更新」顯示。
5. **命名與註解規範**
   * **檔案開頭**：// <filename> -GPT-<version>-<MMDD-HHMM>。
   * **檔案結尾**：// ///// the end of <filename> -GPT-<version>-<MMDD-HHMM>.
   * 版本規則：X.Y.Z，X：大改、Y：功能新增、Z：小修。
6. **程式品質**
   * 不刪減既有功能，所有改動皆應向下相容。
   * JS 模組採用 ES module；CSS 使用明確命名空間，避免全域衝突。
   * 重要參數（URL、檔案路徑、間隔秒數）集中定義為常數。

**advanced logic**

1. **觸發來源**
   * **自動排程**：依照 INTERVAL\_MIN（N 分鐘）週期觸發
   * **手動刷新**：使用者點擊卡片上 🔄 按鈕
   * **頁面初次開啟/瀏覽器重整**：當作「自動排程」的一次觸發

1. **觸發時機**

| **操作／事件** | **自動排程** | **瀏覽器開啟／重整** | **手動刷新按鈕** |
| --- | --- | --- | --- |
| 是否檢查上次抓取 | ✔️ | ✔️ | ❌（不檢查，強制抓取） |
| 若距離 < N 分鐘 | ❌（不抓） | ❌（不抓） | — |
| 若距離 ≥ N 分鐘 | ✔️（抓取） | ✔️（抓取） | — |
| DNS 失敗時 | ✔️（抓取→失敗→排程） | ✔️（抓取→失敗→排程） | ✔️（抓取→失敗→排程） |

* + N：自動刷新間隔分鐘（預設 5 分鐘，可為各卡片獨立設定）。
  + 「瀏覽器開啟／重整」同「自動排程」，判斷是否時間到後才抓，避免一打開就強制抓。

1. **抓取流程**
2. **判斷是否要抓取**
   * 讀取 last\_updated.json[cardId].lastSuccess
   * 計算 diff = now – lastSuccess（分鐘）
   * 若為「手動刷新」或「自動排程到點」或「瀏覽器開啟且 diff ≥ N」，進入抓取；否則直接跳過。
3. **向後端呼叫 /run-fetch?script=...&force=...&interval=...**
   * force=1：手動或開啟時一旦判定要抓即強制抓。
   * force=0：自動排程且 diff < N 時跳過；否則等同強制抓。
4. **後端腳本執行**
   * Puppeteer 或 fetch() 抓資料
   * 成功：寫入主 JSON、更新歷史 his\_\*.json、更新 last\_updated.json，並在 overalllog.json 記錄完整日誌
   * 失敗（例如 DNS）：不改 JSON，不更新 lastSuccess；寫入錯誤日誌，並排程「5 分鐘後重試」＋把原自動排程延後 N＋5 分鐘
5. **執行條件**
   * **自動排程/頁面開啟**：
     1. 先讀取 last\_updated.json 裡該卡片的 lastSuccess 時間
     2. 計算距今差值 diffMin
     3. 若 diffMin ≥ N → **強制執行**爬蟲

否則 → **跳過**，直接讀取現有 JSON，並啟動倒數計時 (N−diffMin)

* + **手動刷新**：
    1. **不受時間限制**，每次都強制執行爬蟲

1. **DNS／網路失敗處理**
   * 在爬蟲階段若偵測到 ERR\_NAME\_NOT\_RESOLVED 或 Fetch 失敗：
     1. **不更新**主 JSON 與 last\_updated.json（保留舊資料）
     2. 在日誌中註明「DNS 解析失敗，使用舊 JSON；於 MM:SS 後重試」
     3. **立即啟動**一條「5 分鐘後重試」的單次排程，並將原有週期排程向後推 N+5 分鐘
     4. **日誌範例：**

[錯誤] ❌ fetch\_xxx.js DNS 解析失敗，使用舊 JSON；於 05:00 後重試，原排程延至 10 分鐘後

1. **成功抓取後更新步驟**
   1. 寫入主檔（fetch\_\*.json）
   2. 更新歷史檔（his\_\*.json）並回傳 newCount
   3. 更新 last\_updated.json 中的該卡片 lastRun、lastSuccess（手動刷新時同時更新 lastManual）
   4. 在終端列出 [預覽]（第一筆）或「無法顯示新聞」
   5. 重設倒數計時為 N 分鐘
2. **前端卡片行為**
   * **初始化**：
     1. 先呼叫 loadNews() 顯示現有 JSON
     2. 更新「最後更新」欄（X 分鐘前更新）
     3. 根據 diffMin 決定是否要先執行 fetchAndUpdate(true)
     4. 若不執行，則啟動倒數計時 (N−diffMin) \* 60 秒
   * **卡片狀態欄**
     1. 成功計時時：還剩約 MM:SS 將更新
     2. DNS 失敗後：DNS 失敗，於 MM:SS 後重試
     3. 正在抓取時：正在更新中... 或 正在抓取最新新聞...
   * **倒數計時**：
     1. 每秒更新「還剩約 MM:SS 將更新」
     2. 成功後自動重設為 N:00 → N−1:59 → … → 00:00 → 自動呼叫 fetch
     3. 手動抓完後立即重設
   * **最後更新**
     1. 根據 last\_updated.json[cardId].lastSuccess 計算分鐘差顯示
     2. <1 分鐘 顯示「剛剛更新」，其他顯示「X 分鐘前更新」
   * **手動刷新**：
     1. 立即呼叫 fetchAndUpdate(true)
     2. 重設倒數為 N 分鐘
   * **自動刷新觸發**（倒數到 0 時）：等同手動刷新行為
3. **後端日誌格式**
   * o**veralllog.json 每筆記錄：**

[YYYY-MM-DD HH:mm:ss] [LEVEL] <Emoji> <script> <訊息>

* + **saveHistoryAndUpdateLast(data, key, hisPath, lastPath) 會：**
    1. 更新歷史檔 his\_<key>.json
    2. 更新 last\_updated.json[key].lastRun 與 .lastSuccess
    3. 回傳 { newCount }
* **卡片完全解構**：每張卡片 JS、CSS、HTML DOM id 完全獨立，前端只需呼叫 cardInit。
* **自動/手動排程判斷**：自動只在距離上次成功 N 分鐘後觸發，手動刷新即刻強制觸發。
* **補歷史機制**：主檔若少於顯示數量，自動補上歷史，且不重覆。
* **異常備援**：如遇 HTML 擋爬/超時自動 fallback 到 Markdown、RSS、或其他格式。
* **狀態切換提示**：自動根據「距離上次成功」自動計算「還剩幾分將更新」，狀態即時顯示。
* **更新延遲與補救**：如遇失敗，排程自動延後，錯誤時顯示「DNS 失敗，5分鐘後重試」。
* **日誌與資料同步**：所有新增、寫入、異常、補全行為同步記錄於 overalllog.json，方便全局追蹤。

### 進階補救 fallback 邏輯（完整邏輯流程）

#### **HTML 失敗 → Markdown/RSS fallback**

1. 先以 cheerio/puppeteer 等爬取 HTML 頁面，嘗試正常擷取新聞。

2. 若遇下列情況即轉為備援：

- HTTP 403/404

- DNS 失敗

- Timeout 超時

- 抓不到任何新聞項

3. Fallback 步驟依據來源配置：

- Markdown 備援（經 Jina.ai 轉 Markdown，正報/力報/澳門日報等）

- RSS Feed（如 GCS）

- 其他爬蟲備援方式

4. 若備援仍失敗，則不覆蓋舊 JSON、僅記錄錯誤，排程自動延後，下次自動重試。

#### **ASCII** **補救流程圖**

╔══════════╗

║ 抓取 HTML║

╚═══╤══════╝

│

│成功

▼

[寫入主檔]

│

├───────────┐

▼ │

[補歷史/日誌] │

│ │

▼ │

[完成] [失敗]

│ │

│ ┌─┴─────────────┐

▼ ▼ │

[END] 補救 fallback │

│ (Markdown、RSS)│

▼ │

[成功?]─────────否──┤

│ │

是 │

│ │

▼ │

[寫入主檔/日誌] [只記錄錯誤]

│ │

▼ │

[END] [END]

**Terminal 日誌補救範例**

[2025-07-06 13:17] [INFO] ℹ️ fetch\_chengpou\_ws\_cb.js [HTML] 加载第 1 页 → https://chengpou.com.mo/newstag/Macao.html

[2025-07-06 13:17] [錯誤] ❌ fetch\_chengpou\_ws\_cb.js [HTML] 第 1 页加载失败：HTTP 403

[2025-07-06 13:17] [INFO] ℹ️ fetch\_chengpou\_ws\_cb.js HTML 資料不足，啟用 Markdown 備援

[2025-07-06 13:17] [INFO] ℹ️ fetch\_chengpou\_ws\_cb.js [MD] 加载第 1 页 → https://r.jina.ai/https://chengpou.com.mo/newstag/Macao.html

[2025-07-06 13:17] [成功] ✅ fetch\_chengpou\_ws\_cb.js 共 30 則新聞已存至 ./data/fetch\_chengpou.json

**API/爬蟲流程圖圖片（ASCII 圖＋流程圖腳本）**

**（A）純 ASCII 流程圖（可直接放 Markdown）**

使用者

│

▼

[前端卡片倒數/手動]

│

▼

fetch('/run-fetch?script=xxx.js')

│

▼

Express API (server.js)

│

▼

spawn node crawler/xxx.js

│

▼

[爬蟲 fetch\_xxx\_ws\_cb.js]

│

│──> [主要來源（HTML/Cheerio/Puppeteer）]

│ │

│ ├─ 成功 → 寫 JSON → 寫歷史/日誌 → 回傳 success

│ │

│ └─ 失敗 → fallback

│ │

│ ├─ Markdown

│ ├─ RSS

│ └─ 其他

│ │

│ ├─ 成功 → 寫 JSON/歷史/日誌

│ └─ 失敗 → 僅記錄錯誤/自動延後

▼

前端收到結果→更新卡片內容/顯示錯誤

（B）Mermaid 語法流程圖（直接放 docs 支援 Markdown Mermaid 的平台！）

<details>

<summary>點開看 Mermaid flowchart</summary>

*flowchart TD*

*A[使用者/自動刷新] --> B[前端 fetch /run-fetch]*

*B --> C[Express API]*

*C --> D[spawn node 爬蟲]*

*D --> E{主要來源成功?}*

*E -- 是 --> F[寫主檔/歷史/日誌]*

*F --> G[回傳成功]*

*E -- 否 --> H{有 fallback 嗎?}*

*H -- 有 --> I[切換 Markdown/RSS/備援]*

*I --> J{成功?}*

*J -- 是 --> F*

*J -- 否 --> K[記錄錯誤/自動延後]*

*K --> G*

*H -- 沒有 --> K*

</details>

**// cardjs/gcs\_card.js -GPT-1.1.1-0708-1100**

import dayjs from '../lib/dayjs.min.js';

import utc from '../lib/utc.js';

import timezone from '../lib/timezone.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const TIMEZONE = 'Asia/Macau';

const AUTO\_REFRESH\_MIN = 5;

export async function cardInit({

cardId,

jsonPath,

show = 10,

max = 35,

autoRefresh = AUTO\_REFRESH\_MIN,

tag = ''

}) {

const card = document.getElementById(cardId);

const refreshBtn = card.querySelector('.card-refresh');

const newsCountIcon = card.querySelector('.news-count-icon');

const lastUpdateEl = card.querySelector('.card-last-update');

const statusEl = card.querySelector('.card-status');

const newsList = card.querySelector('.news-list');

const tagEl = card.querySelector('.macau-tag');

tagEl.textContent = tag;

let countdown = autoRefresh \* 60;

let countdownTimer = null;

// 更新倒數欄

function updateCountdown() {

if (countdown <= 0) {

statusEl.textContent = '正在更新中...';

clearInterval(countdownTimer);

fetchAndUpdate(true);

} else {

const m = String(Math.floor(countdown/60)).padStart(2,'0');

const s = String(countdown%60).padStart(2,'0');

statusEl.textContent = `還剩約 ${m}:${s} 將更新`;

countdown--;

}

}

// 更新「最後更新」欄

async function updateLastUpdated() {

try {

const res = await fetch('/data/last\_updated.json?t=' + Date.now());

const j = await res.json();

const info = j[cardId]?.lastSuccess;

if (!info) {

lastUpdateEl.textContent = '-- 分鐘前更新';

return;

}

const now = dayjs().tz(TIMEZONE);

const then= dayjs(info).tz(TIMEZONE);

const diffMin = Math.floor(now.diff(then, 'minute'));

lastUpdateEl.textContent = diffMin < 1

? '剛剛更新'

: `${diffMin} 分鐘前更新`;

} catch {

lastUpdateEl.textContent = '-- 分鐘前更新';

}

}

// 從 JSON 載入並顯示新聞

async function loadNews() {

try {

const res = await fetch(jsonPath + '?t=' + Date.now());

if (!res.ok) throw new Error('FetchFailed');

const data = await res.json();

newsCountIcon.textContent = data.length > max ? max : data.length;

newsList.innerHTML = '';

data.slice(0, show).forEach((item, idx) => {

const pub = dayjs(item.pubDate).tz(TIMEZONE);

const now = dayjs().tz(TIMEZONE);

const diffH = Math.floor(now.diff(pub, 'hour'));

const dateStr = diffH < 24

? `${diffH} 小時前`

: pub.format('MM-DD');

const li = document.createElement('li');

li.innerHTML = `

<span class="news-rank">${idx + 1}</span>

<a class="news-title" href="${item.link}" target="\_blank" rel="noopener">${item.title}</a>

<span class="date">${dateStr}</span>

`;

newsList.appendChild(li);

});

if (newsList.children.length === 0) {

newsList.innerHTML = '<li>載入失敗</li>';

}

return true;

} catch {

newsList.innerHTML = '<li>載入失敗</li>';

return false;

}

}

// 觸發後端爬蟲並更新

async function fetchAndUpdate(force = false) {

statusEl.textContent = '正在抓取最新新聞...';

try {

const url = `/run-fetch?script=${encodeURIComponent(cardId)}.js&force=${force?1:0}&interval=${autoRefresh}`;

const res = await fetch(url);

if (!res.ok) throw new Error('FetchAPIFailed');

await loadNews();

await updateLastUpdated();

resetCountdown();

statusEl.textContent = '';

} catch {

statusEl.textContent = '更新失敗';

}

}

// 重設倒數

function resetCountdown() {

clearInterval(countdownTimer);

countdown = autoRefresh \* 60;

countdownTimer = setInterval(updateCountdown, 1000);

}

// 手動刷新

refreshBtn.addEventListener('click', () => {

fetchAndUpdate(true);

});

// 初始：先讀新聞、最後更新，判斷是否自動抓取

async function init() {

await loadNews();

await updateLastUpdated();

const res = await fetch('/data/last\_updated.json?t=' + Date.now());

const j = res.ok ? await res.json() : {};

const last = j[cardId]?.lastSuccess

? dayjs(j[cardId].lastSuccess).tz(TIMEZONE)

: null;

const now = dayjs().tz(TIMEZONE);

const diffMin = last ? Math.floor(now.diff(last, 'minute')) : Infinity;

if (diffMin >= autoRefresh) {

await fetchAndUpdate(true);

} else {

countdown = (autoRefresh - diffMin) \* 60;

resetCountdown();

}

}

init();

}

// ///// the end of cardjs/gcs\_card.js -GPT-1.1.1-0708-1100

**// fetch\_gcs\_rss\_cb.js -GPT-1.1.1-0707-1700**

import fs from 'fs';

import dayjs from 'dayjs';

import utc from 'dayjs/plugin/utc.js';

import timezone from 'dayjs/plugin/timezone.js';

import {

logInfo,

logSuccess,

logError,

logPreview

} from './modules/logger.js';

import { saveHistoryAndUpdateLast } from './modules/historyManager.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const TZ = 'Asia/Macau';

const SCRIPT = 'fetch\_gcs\_rss\_cb.js';

const RSS\_URL = 'https://govinfohub.gcs.gov.mo/api/rss/n/zh-hant';

const FALLBACK\_URL = 'https://r.jina.ai/https://govinfohub.gcs.gov.mo/api/rss/n/zh-hant';

const OUTPUT\_JSON = './data/fetch\_gcs\_rss\_cb.json';

const HISTORY\_JSON = './data/his\_fetch\_gcs\_rss\_cb.json';

const LAST\_UPDATED\_JSON = './data/last\_updated.json';

const RETRY\_DELAY\_SEC = 5 \* 60;

const INTERVAL\_MIN = 5;

async function fetchGcsRss() {

logInfo(SCRIPT, '啟動爬取 GCS RSS');

let xml, items = [];

try {

logInfo(SCRIPT, `GET RSS：${RSS\_URL}`);

let res = await fetch(RSS\_URL);

if (!res.ok) throw new Error(`Status ${res.status}`);

xml = await res.text();

logSuccess(SCRIPT, '主站 RSS 取得成功');

} catch (err) {

// DNS 或其他失敗

const mm = String(Math.floor(RETRY\_DELAY\_SEC/60)).padStart(2,'0');

const ss = String(RETRY\_DELAY\_SEC%60).padStart(2,'0');

logError(

SCRIPT,

`主站 RSS 失敗(${err.message})，使用備援 ${FALLBACK\_URL}`

);

try {

let res2 = await fetch(FALLBACK\_URL);

if (!res2.ok) throw new Error(`Status ${res2.status}`);

xml = await res2.text();

logSuccess(SCRIPT, '備援 RSS 取得成功');

} catch (e2) {

logError(

SCRIPT,

`備援也失敗(${e2.message})；於 ${mm}:${ss} 後重試，原排程延至 ${INTERVAL\_MIN+5} 分鐘後`

);

return;

}

}

// 解析 <item>

logInfo(SCRIPT, '解析 <item> 條目');

const regex = /<item>[\s\S]\*?<title>(?:<!\[CDATA\[)?([\s\S]\*?)(?:\]\]>)?<\/title>[\s\S]\*?<link>([\s\S]\*?)<\/link>[\s\S]\*?<pubDate>([\s\S]\*?)<\/pubDate>/g;

let m;

while ((m = regex.exec(xml)) !== null) {

items.push({ title: m[1].trim(), link: m[2].trim(), pubDate: m[3].trim() });

}

logSuccess(SCRIPT, `共解析 ${items.length} 則新聞`);

// 寫入 JSON

fs.writeFileSync(OUTPUT\_JSON, JSON.stringify(items, null, 2), 'utf-8');

logSuccess(SCRIPT, `共 ${items.length} 則新聞已存至 ${OUTPUT\_JSON}`);

// 更新歷史與時間

const { newCount } = saveHistoryAndUpdateLast(

items,

'gcs\_rss\_cb',

HISTORY\_JSON,

LAST\_UPDATED\_JSON

);

logSuccess(SCRIPT, `共新增 ${newCount} 條，已寫入 ${HISTORY\_JSON}`);

// 預覽

if (items.length > 0) logPreview(SCRIPT, items[0]);

else logPreview(SCRIPT, '無法顯示新聞');

}

fetchGcsRss();

// ///// the end of fetch\_gcs\_rss\_cb.js -GPT-1.1.1-0707-1700

**// fetch\_macaodaily\_ws\_cb.js -GPT-1.1.1-0708-1015**

import puppeteer from 'puppeteer';

import fs from 'fs';

import dayjs from 'dayjs';

import utc from 'dayjs/plugin/utc.js';

import timezone from 'dayjs/plugin/timezone.js';

import {

logInfo,

logSuccess,

logError,

logPreview

} from './modules/logger.js';

import { saveHistoryAndUpdateLast } from './modules/historyManager.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const TZ = 'Asia/Macau';

const SCRIPT = 'fetch\_macaodaily\_ws\_cb.js';

const URL = 'https://www.modaily.cn/amucsite/web/index.html';

const OUTPUT\_FILE = './data/fetch\_macaodaily\_ws\_cb.json';

const HISTORY\_FILE = './data/his\_fetch\_macaodaily\_ws\_cb.json';

const LAST\_UPDATED\_FILE = './data/last\_updated.json';

const MAX\_NEWS = 19;

const RETRY\_DELAY\_SEC = 5 \* 60; // 5 分鐘後重試

const INTERVAL\_MIN = 5; // 自動排程間隔

async function fetchNews() {

logInfo(SCRIPT, '啟動');

let news = [];

try {

logInfo(SCRIPT, `載入頁面：${URL}`);

const browser = await puppeteer.launch({ headless: true });

const page = await browser.newPage();

await page.goto(URL, { waitUntil: 'domcontentloaded', timeout: 60000 });

await page.setViewport({ width:1280, height:800 });

await page.waitForSelector('#mainContents div.conWidth.mianConLeft > div:nth-child(1)', { timeout:10000 });

news = await page.evaluate((limit) => {

const out = [];

document.querySelectorAll(

'#mainContents div.conWidth.mianConLeft > div'

).forEach((b, i) => {

if (out.length >= limit) return;

const t = b.querySelector('h3');

const d = b.querySelector('ul>li:nth-child(2)');

const img = b.querySelector('img');

if (!t || !d || !img) return;

const title = t.innerText.trim();

const pubDate = d.innerText.trim();

const src = img.getAttribute('src') || '';

const m = src.match(/\/(\d{7,})\_/);

const id = m ? m[1] : '';

const link = id

? `https://www.modaily.cn/amucsite/web/index.html#/detail/${id}`

: '#';

out.push({ title, pubDate, link });

});

return out;

}, MAX\_NEWS);

await browser.close();

// 正常取得，寫主檔

fs.writeFileSync(OUTPUT\_FILE, JSON.stringify(news, null, 2), 'utf-8');

logSuccess(SCRIPT, `共 ${news.length} 則新聞已存至於 ${OUTPUT\_FILE}`);

// 更新歷史與 last\_updated

const { newCount } = saveHistoryAndUpdateLast(

news,

'macaodaily\_ws\_cb',

HISTORY\_FILE,

LAST\_UPDATED\_FILE

);

logSuccess(SCRIPT, `共新增 ${newCount} 條，已寫入 ${HISTORY\_FILE}`);

// 預覽

if (news.length > 0) logPreview(SCRIPT, news[0]);

else logPreview(SCRIPT, '無法顯示新聞');

} catch (err) {

// DNS 失敗時

if (/ERR\_NAME\_NOT\_RESOLVED/.test(err.message)) {

const mm = String(Math.floor(RETRY\_DELAY\_SEC/60)).padStart(2,'0');

const ss = String(RETRY\_DELAY\_SEC%60).padStart(2,'0');

logError(

SCRIPT,

`DNS 解析失敗，使用舊 JSON；於 ${mm}:${ss} 後重試，原排程延至 ${INTERVAL\_MIN+5} 分鐘後`

);

return;

}

logError(SCRIPT, `抓取/寫入失敗：${err.message}`);

console.error(err);

}

}

fetchNews();

// ///// the end of fetch\_macaodaily\_ws\_cb.js -GPT-1.1.1-0708-1015

**// historyManager.js -GPT-1.1.1-0707-1620**

import fs from 'fs';

import dayjs from 'dayjs';

import utc from 'dayjs/plugin/utc.js';

import timezone from 'dayjs/plugin/timezone.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const TZ = 'Asia/Macau';

const LIMIT = 1000;

// 同時寫歷史與更新 last\_updated.json

export function saveHistoryAndUpdateLast(data, key, hisPath, lastPath) {

// --- 歷史檔 \*\*\*

let his = [];

if (fs.existsSync(hisPath)) {

try { his = JSON.parse(fs.readFileSync(hisPath, 'utf-8')); } catch {}

}

const seen = new Set(his.map(i => i.link));

const newItems = data.filter(i => !seen.has(i.link));

const updated = [...newItems, ...his].slice(0, LIMIT);

fs.writeFileSync(hisPath, JSON.stringify(updated, null, 2), 'utf-8');

// --- last\_updated.json \*\*\*

let last = {};

if (fs.existsSync(lastPath)) {

try { last = JSON.parse(fs.readFileSync(lastPath, 'utf-8')); } catch {}

}

const nowStr = dayjs().tz(TZ).format('YYYY-MM-DD HH:mm:ss');

last[key] = { lastRun: nowStr, lastSuccess: nowStr, lastManual: last[key]?.lastManual || null };

fs.writeFileSync(lastPath, JSON.stringify(last, null, 2), 'utf-8');

return { newCount: newItems.length };

}

// ///// the end of historyManager.js -GPT-1.1.1-0707-1620

**// logger.js -GPT-1.1.1-0707-1620**

import fs from 'fs';

import path from 'path';

import dayjs from 'dayjs';

import utc from 'dayjs/plugin/utc.js';

import timezone from 'dayjs/plugin/timezone.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const TZ = 'Asia/Macau';

const LOGFILE = path.resolve('./data/overalllog.json');

function appendLog(line) {

let arr = [];

if (fs.existsSync(LOGFILE)) {

try { arr = JSON.parse(fs.readFileSync(LOGFILE, 'utf-8')); } catch {}

}

arr.push(line);

fs.writeFileSync(LOGFILE, JSON.stringify(arr, null, 2), 'utf-8');

}

function now() {

return dayjs().tz(TZ).format('YYYY-MM-DD HH:mm:ss');

}

export function logInfo(script, msg) {

const line = `[${now()}] [INFO] ℹ️ ${script} ${msg}`;

console.log(line);

appendLog(line);

}

export function logSuccess(script, msg) {

const line = `[${now()}] [成功] ✅ ${script} ${msg}`;

console.log(line);

appendLog(line);

}

export function logError(script, msg) {

const line = `[${now()}] [錯誤] ❌ ${script} ${msg}`;

console.error(line);

appendLog(line);

}

export function logPreview(script, item) {

const preview = typeof item === 'string'

? item

: item.title

? `${item.title}（更新時間：${item.pubDate}）`

: JSON.stringify(item);

const line = `[${now()}] [成功] ✅ [預覽] ${preview}`;

console.log(line);

appendLog(line);

}

// ///// the end of logger.js -GPT-1.1.1-0707-1620

**// macaodaily\_card.js -GPT-1.1.1-0708-1105**

import dayjs from '../lib/dayjs.min.js';

import utc from '../lib/utc.js';

import timezone from '../lib/timezone.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const TIMEZONE = 'Asia/Macau';

const AUTO\_REFRESH\_MIN = 5;

export async function cardInit({

cardId,

jsonPath,

show = 10,

max = 19,

autoRefresh = AUTO\_REFRESH\_MIN,

tag = ''

}) {

const card = document.getElementById(cardId);

const refreshBtn = card.querySelector('.card-refresh');

const newsCountIcon = card.querySelector('.news-count-icon');

const lastUpdateEl = card.querySelector('.card-last-update');

const statusEl = card.querySelector('.card-status');

const newsList = card.querySelector('.news-list');

const tagEl = card.querySelector('.macau-tag');

tagEl.textContent = tag;

let countdown = autoRefresh \* 60;

let countdownTimer = null;

// 更新倒數顯示

function updateCountdown() {

if (countdown <= 0) {

statusEl.textContent = '正在更新中...';

clearInterval(countdownTimer);

fetchAndUpdate(true);

} else {

const m = String(Math.floor(countdown/60)).padStart(2,'0');

const s = String(countdown % 60).padStart(2,'0');

statusEl.textContent = `還剩約 ${m}:${s} 將更新`;

countdown--;

}

}

// 更新「最後更新」顯示

async function updateLastUpdated() {

try {

const res = await fetch('/data/last\_updated.json?t=' + Date.now());

const j = await res.json();

const info = j[cardId]?.lastSuccess;

if (!info) {

lastUpdateEl.textContent = '-- 分鐘前更新';

return;

}

const now = dayjs().tz(TIMEZONE);

const then= dayjs(info).tz(TIMEZONE);

const diffMin = Math.floor(now.diff(then, 'minute'));

lastUpdateEl.textContent = diffMin < 1

? '剛剛更新'

: `${diffMin} 分鐘前更新`;

} catch {

lastUpdateEl.textContent = '-- 分鐘前更新';

}

}

// 載入 JSON 顯示新聞

async function loadNews() {

try {

const res = await fetch(jsonPath + '?t=' + Date.now());

if (!res.ok) throw new Error('FetchFailed');

const data = await res.json();

newsCountIcon.textContent = data.length > max ? max : data.length;

newsList.innerHTML = '';

data.slice(0, show).forEach((item, idx) => {

const li = document.createElement('li');

li.innerHTML = `

<span class="news-rank">${idx + 1}</span>

<a class="news-title" href="${item.link}" target="\_blank" rel="noopener">${item.title}</a>

<span class="date">${item.pubDate}</span>

`;

newsList.appendChild(li);

});

if (newsList.children.length === 0) {

newsList.innerHTML = '<li>載入失敗</li>';

}

return true;

} catch {

newsList.innerHTML = '<li>載入失敗</li>';

return false;

}

}

// 呼叫後端 API 觸發爬蟲

async function fetchAndUpdate(force = false) {

statusEl.textContent = '正在抓取最新新聞...';

try {

const url = `/run-fetch?script=${encodeURIComponent(cardId)}\_ws\_cb.js&force=${force?1:0}&interval=${autoRefresh}`;

const res = await fetch(url);

if (!res.ok) throw new Error('FetchAPIFailed');

await loadNews();

await updateLastUpdated();

resetCountdown();

statusEl.textContent = '';

} catch {

statusEl.textContent = '更新失敗';

}

}

// 重設倒數

function resetCountdown() {

clearInterval(countdownTimer);

countdown = autoRefresh \* 60;

countdownTimer = setInterval(updateCountdown, 1000);

}

// 手動刷新

refreshBtn.addEventListener('click', () => {

fetchAndUpdate(true);

});

// 初始化

async function init() {

await loadNews();

await updateLastUpdated();

try {

const res = await fetch('/data/last\_updated.json?t=' + Date.now());

const j = await res.json();

const last = j[cardId]?.lastSuccess

? dayjs(j[cardId].lastSuccess).tz(TIMEZONE)

: null;

const now = dayjs().tz(TIMEZONE);

const diffMin = last ? Math.floor(now.diff(last, 'minute')) : Infinity;

if (diffMin >= autoRefresh) {

await fetchAndUpdate(true);

} else {

countdown = (autoRefresh - diffMin) \* 60;

resetCountdown();

}

} catch {

// 若讀不到 last\_updated，直接啟動一次爬蟲

await fetchAndUpdate(true);

}

}

init();

}

// ///// the end of cardjs/macaodaily\_card.js -GPT-1.1.1-0708-1105

**// server.js -GPT-1.1.1-0707-1105**

import express from 'express';

import cors from 'cors';

import path from 'path';

import { exec } from 'child\_process';

import dayjs from 'dayjs';

import utc from 'dayjs/plugin/utc.js';

import timezone from 'dayjs/plugin/timezone.js';

dayjs.extend(utc);

dayjs.extend(timezone);

const app = express();

const PORT = 3000;

const TZ = 'Asia/Macau';

// CORS 與靜態檔案服務

app.use(cors());

app.use(express.static(path.resolve('./'), { index: 'index.html' }));

/\*\*

\* 觸發後端爬蟲：

\* GET /run-fetch?script=<filename>&force=<0|1>&interval=<minutes>

\*/

app.get('/run-fetch', (req, res) => {

const { script, force = '0', interval = '5' } = req.query;

if (!script) {

return res.status(400).json({ error: 'Missing script parameter' });

}

const scriptPath = path.resolve('./crawler', script);

const cmd = `node "${scriptPath}"`;

// 立即執行爬蟲腳本

exec(cmd, (err, stdout, stderr) => {

if (err) {

console.error(stderr || err.message);

return res.status(500).json({ error: stderr || err.message });

}

// 回傳當下 Macau 時間為 lastSuccess

const lastSuccess = dayjs().tz(TZ).format('YYYY-MM-DD HH:mm:ss');

res.json({ success: true, lastSuccess });

});

});

// 啟動

app.listen(PORT, () => {

console.log(`[SERVER] ✅ Server started on http://localhost:${PORT}`);

});

// ///// the end of server.js -GPT-1.1.1-0707-1105

### index.html

<!DOCTYPE html>

<html lang="zh-Hant">

<head>

<meta charset="UTF-8">

<title>Lotus 新聞展示系統</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- 花園明朝體 -->

<link href="https://path.to/flower-font.css" rel="stylesheet">

<!-- 公用字型 -->

<link href="https://fonts.googleapis.com/css2?family=Noto+Sans+TC&display=swap" rel="stylesheet">

<!-- 卡片樣式 -->

<link rel="stylesheet" href="./css/macaodaily\_card.css">

<link rel="stylesheet" href="./css/gcs\_card.css">

<style>

body {

background-color: #f4f7fb;

font-family: "HanaMin", "Noto Sans TC", sans-serif;

padding: 40px;

display: flex;

flex-wrap: wrap;

gap: 32px;

justify-content: flex-start;

}

</style>

</head>

<body>

<!-- 澳門日報 卡片 -->

<div class="macaodaily-card" id="macaodaily-card">

<div class="card-header">

<div class="card-title-row">

<span class="card-title">澳門日報</span>

<button class="card-refresh" title="手動刷新">🔄</button>

<span class="news-count-icon">0</span>

</div>

<div class="card-last-update">-- 分鐘前更新</div>

<div class="macau-tag">#澳門 #即時</div>

</div>

<div class="news-list-area">

<ul class="news-list">

<li>載入中...</li>

</ul>

</div>

<div class="card-status">等待更新...</div>

</div>

<!-- 政府新聞網聚合 卡片 -->

<div class="gcs-card" id="gcs\_rss\_cb-card">

<div class="card-header">

<div class="card-title-row">

<span class="card-title">政府新聞網聚合</span>

<button class="card-refresh" title="手動刷新">🔄</button>

<span class="news-count-icon">0</span>

</div>

<div class="card-last-update">-- 分鐘前更新</div>

<div class="macau-tag">#官方 #政府</div>

</div>

<div class="news-list-area">

<ul class="news-list">

<li>載入中...</li>

</ul>

</div>

<div class="card-status">等待更新...</div>

</div>

<!-- 呼叫卡片初始化 -->

<script type="module">

import { cardInit as mcInit } from './cardjs/macaodaily\_card.js';

import { cardInit as gcsInit } from './cardjs/gcs\_card.js';

mcInit({

cardId: 'macaodaily-card',

jsonPath: '/data/fetch\_macaodaily\_ws\_cb.json',

show: 10,

max: 19,

autoRefresh: 5,

tag: '#澳門 #即時'

});

gcsInit({

cardId: 'gcs\_rss\_cb-card',

jsonPath: '/data/fetch\_gcs\_rss\_cb.json',

show: 10,

max: 35,

autoRefresh: 5,

tag: '#官方 #政府'

});

</script>

</body>

</html>