

【NextJS】相關

- [【Next.js】 建立新專案](#)
- [【Next.js】 專案結構與檔案說明](#)
- [【Next.js】 路由機制](#)
- [【Next.js】 路由機制\(動態\)](#)
- [【Next.js】 not-found \(404\)](#)
- [【Next.js】 Private Folders](#)
- [【Next.js】 Route Groups](#)
- [【Next.js】 Layout](#)
- [【Next.js】 Multiple Root Layouts](#)
- [【Next.js】 Link 元件](#)
- [【Next.js】 metadata 設定總覽](#)
- [【Next.js】 常用 scripts 說明](#)
- [【Next.js】 部署模式總覽](#)

【Next.js】建立新專案

```
# npx create-next-app 建立新專案
$ npx create-next-app
Need to install the following packages:
create-next-app@15.4.3
Ok to proceed? (y) y

✓ What is your project named? ... demo-next
✓ Would you like to use TypeScript? ... No / Yes #Yes
✓ Would you like to use ESLint? ... No / Yes #Yes
✓ Would you like to use Tailwind CSS? ... No / Yes #Yes
✓ Would you like your code inside a `src/` directory? ... No / Yes #Yes
✓ Would you like to use App Router? (recommended) ... No / Yes #Yes
✓ Would you like to use Turbopack for `next dev`? ... No / Yes #No
✓ Would you like to customize the import alias (`@/*` by default)? ... No / Yes #Yes
✓ What import alias would you like configured? ... @/*
Creating a new Next.js app in /Users/srou/Documents/vscode_workspace/demo/nextjs-demo/demo-next.

Using npm.

Initializing project with template: app-tw

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- @tailwindcss/postcss
- tailwindcss

added 59 packages, and audited 60 packages in 20s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Initialized a git repository.

Success! Created demo-next at /Users/srou/Documents/vscode_workspace/demo/nextjs-demo/demo-next
```

```
# 啟動專案 npm run dev
$ npm run dev

> demo-next@0.1.0 dev
> next dev

△ Warning: Found multiple lockfiles. Selecting /Users/srou/Documents/vscode_workspace/yarn.lock.
  Consider removing the lockfiles at:
  * /Users/srou/Documents/vscode_workspace/demo/demo-next/package-lock.json

▲ Next.js 15.4.3
  - Local:    http://localhost:3000
  - Network:  http://192.168.1.1:3000

✓ Starting...
✓ Ready in 2.1s
```

當你使用 `npx create-next-app` 建立 Next.js 專案時，CLI 會引導你一步一步選擇要開啟哪些功能。以下是每一個步驟的說明，以及如果最後選擇 **不設定 import alias** (例如 `@/*`)，日後如何補上設定。

□ 建立 Next.js 專案步驟說明

1. What is your project named?

“ 輸入專案資料夾名稱，例如 `demo-next` 。”

2. Would you like to use TypeScript?

“ 是否使用 TypeScript (推薦選 Yes，提供型別安全) 。”

- 選 Yes → 自動生成 `tsconfig.json`
- 選 No → 使用 JavaScript + `jsconfig.json`

3. Would you like to use ESLint?

“ 是否啟用 ESLint 做程式碼檢查 。”

- 選 Yes → 建立 `.eslintrc.json` 等設定，避免潛在錯誤。

4. Would you like to use Tailwind CSS?

“ 是否加入 Tailwind CSS 做 UI 樣式設計 。”

5. Would you like your code inside a src/ directory?

“ 是否將程式碼放入 `src/` 子資料夾中 。”

- 選 Yes → 專案結構更清晰，建議開啟。

6. Would you like to use App Router? (recommended)

“ 啟用 Next.js 13+ 的新 **App Router** 架構 (基於 `app/` 資料夾) 。”

- 選 Yes → 使用新路由系統，支援 Server Components 等現代功能。
-

7. Would you like to use Turbopack for next dev?

“ 是否使用新一代 Turbopack 開發模式（取代 Webpack 開發時編譯器）。

- 只會影響 `npm run dev` 時的速度
- 生產環境仍使用 Webpack

8. Would you like to customize the import alias (@/* by default)?

“ 是否要自訂 import 路徑別名 alias ?

選 Yes :

- CLI 會再問你輸入別名（例如 `@/*`），
- 會幫你自動建立 `paths` 設定（在 `tsconfig.json` 或 `jsconfig.json`）

選 No :

- 不會設定別名，之後你手動寫 `../../components/...` 等麻煩路徑。

若選擇 "No"，之後如何補上 import alias ?

▶ 1. 找出設定檔

如果你是：

- 用 TypeScript → 修改 `tsconfig.json`
- 用 JavaScript → 修改 `jsconfig.json`

▶ 2. 修改範例如下（代表 `@` 指向 `src` 資料夾）：

```
{
  "compilerOptions": {
    "baseUrl": ".",      // 專案根目錄
    "paths": {
      "@/*": ["src/*"]
    }
  }
}
```

▶ 3. 如果你沒使用 `src/`，就寫：

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@/*": ["*"]
    }
  }
}
```

□ 目錄結構對照範例

若你有勾選使用 `src/`，專案結構可能如下：

```
demo-next/  
├─ src/  
│  └─ app/  
│  └─ components/  
│  └─ ...  
└─ tsconfig.json
```

則你就能寫這樣的路徑：

```
import { Header } from '@components/Header'
```

□ 總結：建立流程與 alias 設定對照

功能	說明
是否設定 alias (<code>@</code>)	<code>create-next-app</code> 最後一個選項
若選否，日後可透過 <code>ts/jsconfig</code> 設定	<input type="checkbox"/> 很簡單，加一段 <code>"paths"</code> 即可
是否需要設定 <code>next.config.js</code>	通常不需要（除非使用 webpack plugin）

【Next.js】專案結構與檔案說明

以下是根據你提供的 Next.js 專案結構，整理出各資料夾與檔案的用途與說明：

專案結構與說明表格

路徑 / 檔案	類型	說明
<code>.next/</code>	資料夾	Next.js build 輸出資料夾，會在 <code>next build</code> 時產生。內含編譯後的 server 與靜態資源。可部署至伺服器。
<code>server/</code>	資料夾	Server-side compiled files (如 SSR 頁面、API Routes)。
<code>static/</code>	資料夾	靜態產出資源 (例如圖片、CSS、JS 等)。
<code>cache/</code>	資料夾	編譯過程中的 cache，提高二次編譯速度。
<code>build-manifest.json</code>	JSON 檔案	描述每個頁面對應的 JS chunk，供 runtime 載入使用。
<code>routes-manifest.json</code>	JSON 檔案	儲存頁面與路由資訊的映射表。
<code>prerender-manifest.json</code>	JSON 檔案	與預先產生的頁面 (SSG) 有關，例如 <code>getStaticProps</code> 結果。
<code>node_modules/</code>	資料夾	安裝的 npm 套件。
<code>public/</code>	資料夾	放置靜態資源 (如圖片、SVG)，透過 <code>/</code> 路徑對外公開，例如： <code>public/globe.svg</code> → <code>/globe.svg</code> 。
<code>src/app/</code>	資料夾	使用 App Router 架構的主要入口，代表新架構 (取代 <code>pages/</code> 目錄)。
<code>layout.tsx</code>	React 元件	每個 route 的共同 Layout 元件。App Router 中不可或缺。
<code>page.tsx</code>	React 元件	對應 <code>/</code> 路由的頁面內容 (Home)。
<code>globals.css</code>	CSS 檔案	全域 CSS，需在 <code>layout.tsx</code> 中引入。
<code>favicon.ico</code>	圖示檔案	網站 favicon，會自動用在 <code><head></code> 中。
<code>.gitignore</code>	設定檔	Git 忽略檔案清單。通常會包含 <code>.next/</code> 、 <code>node_modules/</code> 。
<code>next-env.d.ts</code>	TS 宣告	Next.js 產生的 type 設定檔，自動加入必要的 TypeScript 型別。
<code>next.config.ts</code>	設定檔	Next.js 專案設定檔，支援自訂建置、路徑、Plugin 等功能。
<code>package.json</code> / <code>package-lock.json</code>	設定檔	npm 套件定義與鎖定檔。記錄所有依賴與版本。
<code>postcss.config.mjs</code>	設定檔	PostCSS 設定，通常與 TailwindCSS 搭配使用。
<code>README.md</code>	說明檔	專案說明、安裝方式、開發資訊等。
<code>tsconfig.json</code>	設定檔	TypeScript 專案設定，例如路徑 alias、編譯選項等。

重點說明

分類	重點
重要目錄	<code>src/app/</code> (使用 App Router 架構)、 <code>public/</code> (靜態資源)、 <code>.next/</code> (build 輸出)
重要設定檔	<code>next.config.ts</code> 、 <code>tsconfig.json</code> 、 <code>postcss.config.mjs</code>
App Router 必備	<code>layout.tsx</code> 是必要頁面組件，類似傳統 <code>_app.tsx</code> + <code>_document.tsx</code>
部署時可忽略	<code>node_modules/</code> 、 <code>.next/cache/</code> 、 <code>.gitignore</code>

說明 layout.tsx 與 page.tsx 關係

`layout.tsx` 與 `page.tsx` 是 Next.js 13+ (App Router 架構) 中，兩個核心檔案，分別扮演不同但互補的角色。

□ 簡單說明

檔案	用途	載入頻率	比喻
layout.tsx	定義頁面結構與共用 UI (像是外框)	只載入一次 (不會重複重建)	「建築物的骨架」
page.tsx	定義實際頁面內容	每個頁面都會重新載入	「房間的內容、家具、住戶」

□ 目錄架構範例

```
app/
├── layout.tsx  # 所有頁面的共用 Layout
├── page.tsx   # / 路徑的首頁內容
├── about/
│   ├── layout.tsx  # 僅 /about 使用的 layout
│   └── page.tsx   # /about 的內容
```

□ 關係與行為

- 一個 **layout** 對應多個 **page**
 - 例如 `app/layout.tsx` 包住所有頁面：`/`, `/about`, `/contact`
- 嵌套 layout**
 - 子資料夾可有自己的 `layout.tsx`，會包在上層 **layout** 裡
- layout 不會因為切換 page 被卸載**
 - 可以放在 `Header`, `Sidebar`, `Footer` 等共用 UI 元件，效能更佳
- page.tsx 是實際的頁面入口**
 - 每個 URL 都需要一個 `page.tsx` 來當作入口點

□ 實際範例

app/layout.tsx

```
export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en">
      <body>
        <header>共用導覽列</header>
        <main>{children}</main>
        <footer>共用頁尾</footer>
      </body>
    </html>
  )
}
```

app/page.tsx

```
export default function HomePage() {
  return <h1>這是首頁內容</h1>
}
```

□ 此時 `HomePage` 內容會被注入進 `layout.tsx` 的 `{children}` 中。

一圖總結

```
layout.tsx
├── Header
├── <main>
│   └── page.tsx 的內容
└── Footer
```

補充：layout.tsx 特性

特性	是否支援	備註
支援 <code>loading.tsx</code>	☐	當子頁面在載入時顯示 loading UI
支援 <code>error.tsx</code>	☐	包裹此 layout 的錯誤畫面
支援 <code>metadata</code> 設定	☐	可以定義 title、description 等
自動 caching	☐	layout 有自動快取，非動態頁效能極佳

需要我幫你畫出 layout/page 的巢狀結構流程圖或建立範本專案也可以說一聲。

【Next.js】路由機制

Next.js 使用檔案結構 (File System Routing) 搭配 `page.tsx` (或 `page.jsx`) 自動產生路由，這是 Next.js 的一大特色，讓你不需手動寫 `route config`，只要建立檔案與資料夾就能對應 URL。

□ Next.js App Router 的路由機制 (以 `app/` 為根)

```
app/
├─ page.tsx      → /
├─ about/
│  └─ page.tsx   → /about
├─ blog/
│  └─ page.tsx   → /blog
│     └─ [slug]/
│        └─ page.tsx → /blog/hello、/blog/123
├─ dashboard/
│  └─ layout.tsx  → 包住 dashboard 所有子路由
│  └─ page.tsx   → /dashboard
│     └─ settings/
│        └─ page.tsx → /dashboard/settings
```

□ 路由對應邏輯

檔案名稱	對應路徑 (URL)	說明
<code>app/page.tsx</code>	<code>/</code>	首頁
<code>app/about/page.tsx</code>	<code>/about</code>	靜態路由
<code>app/blog/[slug]/page.tsx</code>	<code>/blog/:slug</code>	動態路由
<code>app/dashboard/page.tsx</code>	<code>/dashboard</code>	支援巢狀路由
<code>app/dashboard/settings/page.tsx</code>	<code>/dashboard/settings</code>	巢狀子頁面

□ 動態路由 (Dynamic Routes)

範例：

```
app/product/[id]/page.tsx
```

訪問 `/product/123`、`/product/abc` 都會進到這個頁面。

如何在 component 中取得參數：

```
import { useParams } from 'next/navigation'

export default function ProductPage() {
  const params = useParams()
  return <div>商品 ID：{params.id}</div>
}
```

□ 巢狀路由與 Layout 配合

```
app/
├── layout.tsx    // 全站 layout
├── dashboard/
│   ├── layout.tsx // dashboard 專屬 layout
│   └── settings/
│       └── page.tsx // 會套用兩層 layout
```

→ 這樣 `/dashboard/settings` 的畫面會被兩層 layout 包起來。

□ 注意事項

限制	說明
<code>page.tsx</code> 為必要頁面進入點	每個路由至少需要一個 <code>page.tsx</code>
不支援動態 catch-all *	請使用 <code>[...slug]</code>
<code>app/</code> 下不能使用 <code>pages/</code>	App Router 與 Pages Router 不能共存

□ 小技巧

目的	實作方式
動態路由 (product/:id)	<code>app/product/[id]/page.tsx</code>
可選參數 (可有可無)	<code>app/product/[id?]/page.tsx</code>
Catch-all 路由	<code>app/docs/[...slug]/page.tsx</code>
自定 metadata (如 title)	使用 <code>export const metadata</code>

□ 結論

Next.js 的路由系統基於檔案結構：

- 目錄名代表 URL 路徑
- `page.tsx` 是每個路由的「入口」
- 巢狀結構代表巢狀路由
- 使用 `[]` 定義動態路由

如果你有具體的頁面結構想設計，我可以幫你列出完整檔案路徑與對應 URL 對照表。

【Next.js】路由機制(動態)

在 Next.js 中，**Dynamic Routes (動態路由)** 允許你根據 URL 中的參數來動態建立頁面，這對於像是部落格文章、使用者頁面、商品詳細頁等非常有用。

□ 基本概念

在 Next.js 中，**每個檔案都對應一個路由**。當你在 `app/` 或 `pages/` 目錄中使用 **方括號 ([param])** 定義檔案名稱時，就代表建立了一個動態路由。

例如：

```
pages/  
└─ blog/  
   └─ [slug].tsx
```

這樣的結構可以對應：

- `/blog/hello-world`
- `/blog/my-second-post`

在 `app/` 目錄中則會是：

```
app/  
└─ blog/  
   └─ [slug]/  
      └─ page.tsx
```

□ Dynamic Routes 實作 (`app/`)

1 □ 建立檔案：`app/post/[id]/page.tsx`

```
interface PageProps {  
  params: { id: string }  
}  
  
export default function Post({ params }: PageProps) {  
  return <h1>Post ID: {params.id}</h1>  
}  
  
// { params }: PageProps  
// { params } => 解構，取出物件的 params 屬性  
// :PageProps typescript宣告型別  
// 直接簡化成 { params }: { params: { id: string } }
```

□ 多層動態路由

支援巢狀結構，例如：

```
app/  
└─ user/  
   └─ [userId]/  
      └─ post/  
         └─ [postId]/  
            └─ page.tsx
```

訪問路徑 `/user/1/post/10` 對應的 `params` 為：

```
params: {
  userId: '1',
  postId: '10'
}
```

☐ Catch-all 路由（捕捉所有）

你可以使用 `[...slug]` 來捕捉多層級路徑：

- `app/products/[...slug].tsx` 會匹配 `/products/a`、`/products/a/b/c`

```
interface PageProps {
  params: {
    slug?: string[];
  };
}

export default function ProductPage({ params }: PageProps) {
  return (
    <div>
      <h1>Products Page</h1>
      <pre>{JSON.stringify(params.slug)}</pre>
    </div>
  );
}
```

如果要可選，可用 `[[...slug]]` 表示非必要。

☐ 小結

類型	說明	範例
<code>[param]</code>	基本動態路由	<code>/post/1</code> → <code>[id].tsx</code>
<code>[...param]</code>	捕捉多層路徑（Catch-all）	<code>/docs/a/b</code> → <code>[...slug].tsx</code>
<code>[[...param]]</code>	可選 catch-all（Optional）	<code>/docs</code> or <code>/docs/a</code>

若你想看實際範例或搭配 `getStaticPaths`、`generateStaticParams` 使用，也可以提出，我可以補上。需要嗎？

【Next.js】not-found (404)

在 Next.js 中，`not-found` 頁面是用來顯示 **404 頁面**（找不到頁面）用的。從 Next.js 13（App Router）開始，你可以更靈活地自訂與觸發 `not-found`。

□ `not-found` 頁面用途

功能	說明
預設 404 頁面	訪問不存在的路由時自動顯示
自訂 <code>not-found</code>	你可以定義自訂的樣式與內容
手動觸發 404	在伺服器邏輯中，條件不符時主動觸發 <code>notFound()</code>

□ App Router (`app/` 目錄) 中的使用方式

□ 1. 建立 `not-found.tsx` 頁面

放在你想處理的路由資料夾中，例如：

```
app/
├─ blog/
│  └─ [slug]/
│     └─ page.tsx
│     └─ not-found.tsx  □ 對應 `/blog/[slug]` 找不到時
```

□ `app/blog/[slug]/not-found.tsx`

```
export default function NotFound() {
  return (
    <div>
      <h2>這篇文章不存在</h2>
      <a href="/">回首頁</a>
    </div>
  );
}
```

□ 2. 在 `page.tsx` 中觸發 `notFound()`

```
import { notFound } from 'next/navigation';

export default function BlogPost({ params }: { params: { slug: string } }) {
  const validSlugs = ['a', 'b', 'c'];

  if (!validSlugs.includes(params.slug)) {
    notFound(); // □ 主動觸發 404 頁面
  }

  return <h1>Blog: {params.slug}</h1>;
}
```

□ Global Not Found Page

如果你在 `app/not-found.tsx` 建立，這就是全站共用的 404 頁面。

```
app/  
├── not-found.tsx  □ 對應任何未定義頁面
```

□ `app/not-found.tsx` 範例

```
export default function GlobalNotFound() {  
  return (  
    <div style={{ textAlign: 'center', padding: '2rem' }}>  
      <h1>404 - 頁面找不到</h1>  
      <p>您訪問的頁面可能已被移除或從未存在。</p>  
      <a href="/">回首頁</a>  
    </div>  
  );  
}
```

□ 測試方式

測試路徑	預期結果
<code>/blog/xxx</code>	若 <code>xxx</code> 不存在 → 觸發 <code>notFound()</code>
<code>/some/random/page</code>	找不到任何匹配路由 → global 404

□ 小結

功能	用法
定義 not-found 頁面	在 <code>app/**/not-found.tsx</code> 中建立
全域 fallback 頁面	在 <code>app/not-found.tsx</code>
手動觸發 404	使用 <code>notFound()</code> 函式

【Next.js】Private Folders

在 Next.js 中，**Private Folders (私有資料夾)** 是指：

“ 不會被當作路由、自動產生頁面或靜態資源的資料夾。
這些資料夾的名稱 開頭是底線 或雙底線 ，也就是說：

資料夾名稱開頭為 或 ⇒ Next.js 會自動忽略它

什麼是 Private Folders ?

這是 Next.js 的一項慣例 (convention)，用來 隔離你不想當作頁面處理的輔助程式碼或元件檔案。

使用範例

假設你有以下結構：

```
app/
├─ page.tsx
├─ _utils/      ←  不會當成路由
│  └─ formatter.ts
├─ _tests_/    ←  不會當成路由
│  └─ page.test.tsx
```

這些 `_utils/`、`_tests_/` 都是 **Private Folders**：

- 不會產生 `/utils` 路由
- 不會被包含在 build 出來的頁面中
- 適合放測試、工具、常數、hooks 等程式碼

實用用途

資料夾名稱	用途範例
<code>_utils/</code>	放工具函式 (e.g. dateFormat)
<code>_hooks/</code>	放自訂 React Hook
<code>_tests_/</code>	放單元測試
<code>_mocks_/</code>	放 Jest mock 資料

注意事項

行為	說明
<input type="checkbox"/> 允許 <code>import</code> 使用	Private folders 裡的檔案仍然可以被引用
<input type="checkbox"/> 不會產生 URL	不會出現像 <code>/_tests_</code> 這種頁面
<input type="checkbox"/> 適用於 <code>pages/</code> 與 <code>app/</code> 架構	兩種路由系統都支援
△ 無法自訂規則	Next.js 只會根據 <input type="checkbox"/> 或 <input type="checkbox"/> 開頭自動判斷

□ 範例：使用 `_utils/format.ts`

```
// app/_utils/format.ts
export function formatCurrency(amount: number) {
  return `$$${amount.toFixed(2)}`
}
```

```
// app/products/page.tsx
import { formatCurrency } from '../_utils/format'

export default function ProductPage() {
  return <p>{formatCurrency(123.456)}</p>
}
```

□ 小結

項目	說明
命名規則	資料夾名稱開頭為 <code>_</code> 或 <code>__</code>
功能	不產生頁面、不加入 build 路由
適合作用途	工具、hook、測試、mock、常數
是否能 import	<code>_</code> 可以被引用使用

這是一個非常好也非常細緻的問題，牽涉到 JavaScript / TypeScript 社群和工具生態中的 **命名慣例 (naming conventions)**，並不是 Next.js 特有的規定，而是社群共識。

□ TL;DR 結論

命名形式	用途 / 慣例	來源
<code>_folder</code>	□ 私有資料夾，Next.js 慣例 (前綴式)	Next.js、社群通用慣例
<code>__tests__</code>	□ 測試資料夾，Jest 標準結構 (雙底線前後)	Jest 官方
<code>__mocks__</code>	□ Mock 資料夾，Jest 標準	Jest
<code>_filename.ts</code>	私有或輔助用檔案 (非主模組)	Node / TypeScript 社群

□ 單底線 `_`：一般表示私有或特殊用途

- `_utils/`、`_hooks/`、`_internal/`
- 表示：這不是一個路由，也不是 API 端點，只是輔助邏輯用

這種用法來自許多框架 (Django、Rails、Node.js) 與社群習慣。

□ 雙底線 `__`：通常是工具/測試框架的保留命名

來自 **Jest**、**React Native**、**Metro Bundler** 等工具：

- `tests` → Jest 會自動找出來執行單元測試
- `mocks` → Jest 用來定義自動 mock 資料
- `__snapshots__` → Jest 的 snapshot 測試檔案

□ **雙底線前後一致**，通常代表「這是特殊目的目錄」，會被工具偵測與處理。

命名語意總整理

命名	語意	常見用途
<code>_名稱</code>	私有 / 忽略 / 特殊用途 (Next.js 慣例)	<code>_utils</code> , <code>_internal</code> , <code>_api</code>
<code>__名稱__</code>	測試工具專用	<code>__tests__</code> , <code>__mocks__</code>
無底線	公開模組 / 頁面 / API 等	<code>products</code> , <code>user</code> , <code>api</code>

Next.js 怎麼處理這些命名？

資料夾名稱	是否會轉成路由？	用途建議
<code>products/</code>	<input type="checkbox"/> 是	頁面
<code>_products/</code>	<input type="checkbox"/> 否	工具 / 非頁面模組
<code>__tests_/</code>	<input type="checkbox"/> 否	單元測試、快照測試用

延伸知識：為何不是 `_tests/` 而是 `__tests__`？

這來自 **Unix / Bash** 與正規表達式處理的傳統：

- `__tests__` 這種明確結構更容易在工具中用 glob 匹配，如：

```
find . -name '__tests__'  
jest "**/__tests__/**/*.test.tsx"
```

這是 Jest 與 Babel 的官方推薦格式。

總結

命名類型	意圖 / 工具支援
<code>_xxx/</code>	Next.js 私有目錄，僅開頭底線
<code>__xxx_/</code>	Jest 風格目錄，工具自動識別處理
<code>_xxx.ts</code>	單一檔案私有語意

此外通常 `utils` 其他分類資料夾只要不在 `app` 下就不會產生路由，也是一種常用的方法

```
src
├── app
├── assets
├── businessLogic
├── components
├── enums
├── hooks
├── interface
├── pages
├── utils
├── d.ts
├── middleware.ts
├── .env.development
├── .env.production
```

【Next.js】Route Groups

在 Next.js 的 **App Router** 架構中，`Route Groups` 是一個非常實用的功能，用來：

“ `Route Groups` 分組路由檔案，但不會影響 URL 路徑或頁面路由。

Route Groups 是什麼？

語法：用括號 `()` 包住資料夾名稱

例如：

```
app/
├── (dashboard)/
│   └── analytics/
│       └── page.tsx ← URL: /analytics (不會變成 /dashboard/analytics)
```

路徑中的 `(dashboard)` 不會出現在網址上

但它可以让你組織頁面、套用 layout。

使用場景

使用目的	說明
組織頁面結構	讓你的 app 資料夾更乾淨、更有結構
不同區段共用不同 layout	例如 (marketing)、(admin)、(auth) 區塊使用不同 layout
不污染 URL	URL 不包含 group 名稱，例如 (admin)/users → /users

範例 1：基本 Route Group

```
app/
├── (marketing)/
│   ├── home/
│   └── page.tsx
```

訪問路徑：`/home`

不會是：`/marketing/home`

範例 2：共用 layout 的 Route Group

```
app/
├── (auth)/
│   ├── layout.tsx ← auth 區塊專用 layout
│   ├── login/
│   │   └── page.tsx ← `/login`
│   └── register/
│       └── page.tsx ← `/register`
```

範例 3：巢狀 Route Groups

```
app/
├── (admin)/
│   ├── layout.tsx
│   └── users/
│       └── page.tsx ← `users`
├── (marketing)/
│   └── about/
│       └── page.tsx ← `about`
```

每個 group 都可以有自己專屬的 `layout.tsx`，但網址仍保持扁平化。

搭配 `not-found.tsx`、`error.tsx`、`loading.tsx`

你可以在 group 中設：

```
app/
├── (admin)/
│   ├── layout.tsx
│   ├── not-found.tsx
│   ├── error.tsx
│   └── dashboard/
│       └── page.tsx
```

這樣 group 中的所有路由都會共用這些錯誤/狀態處理頁。

Route Groups 的注意事項

問題	說明
Group 名稱不能用 <code>/</code> 或 <code>-</code>	只能用括號與英文小寫（無實際路徑名稱）
不能與實體 URL 混用	<code>app/(admin)/users</code> 是 <code>/users</code> ，不是 <code>/admin/users</code>
沒有 <code>generateStaticParams()</code>	Route Group 自身不是 route，不能設 <code>page.tsx</code>

小結

特性	說明
語法	<code>(group-name)/</code>
是否會影響 URL？	<input type="checkbox"/> 不會，純粹是分組與結構管理用途
支援 layout 嗎？	<input type="checkbox"/> 支援，可針對 group 套用專屬 layout
適合什麼用途？	管理大型專案，區分功能區域（如 <code>admin/auth/marketing</code> ）

【Next.js】Layout

太好了！以下是為「Next.js 初學者」量身打造的教學文章，主題為：

□ Next.js App Router 中 layout.tsx、page.tsx、children 的邏輯與實作

□ 為什麼要了解這三個東西？

在 Next.js 的 App Router 架構中，整個畫面的內容是由許多層「外殼 (layout)」與「內頁 (page)」所組合而成。而這個組合的核心，就是這三個角色：

- `layout.tsx`：整體框架的「容器」
- `page.tsx`：實際對應到網址的「頁面內容」
- `children`：layout 內部接收的下一層內容（可能是 page，也可能是下一層 layout）

□ 這三者的關係是什麼？

你可以想像畫面組成就像是便當盒：

- `layout.tsx` 就是便當盒（框架）
- `page.tsx` 是便當裡的菜色（內容）
- `children` 是 `layout.tsx` 中的佔位符，用來「放內容的地方」

□ 換句話說：

“ `page.tsx` 的畫面內容，會自動傳給上層 `layout.tsx` 中的 `children` 來顯示 ”

□ 結構範例

```
app/
├─ layout.tsx    ← 全站框架 Layout
├─ page.tsx     ← 首頁 ` / `
├─ about/
└─   └─ page.tsx ← ` /about ` 頁面
```

□ layout.tsx 實際程式碼

```
export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en">
      <body>
        <header>這是全站共用的 Header</header>
        <main>{children}</main> { /* 這裡會被 page.tsx 的內容填進來 */ }
        <footer>這是共用的 Footer</footer>
      </body>
    </html>
  )
}
```

```
</body>
</html>
);
}
```

□ page.tsx 實際程式碼

```
export default function HomePage() {
  return <div>這是首頁的內容</div>;
}
```

□ 瀏覽 / 時會發生什麼事？

1. Next.js 根據 URL 找到 `app/page.tsx`
2. 找到對應的 `layout.tsx` 作為頁面外框
3. 把 `page.tsx` 的內容當作 `children` 傳給 `layout.tsx`
4. 畫面最終呈現：

```
<html lang="en">
  <body>
    <header>這是全站共用的 Header</header>
    <main>
      <div>這是首頁的內容</div>
    </main>
    <footer>這是共用的 Footer</footer>
  </body>
</html>
```

□ 多層 layout 怎麼辦？

可以有巢狀 layout 結構，例如：

```
app/
├── layout.tsx      ← 全站 layout
├── (admin)/
│   ├── layout.tsx ← admin 專區 layout
│   └── dashboard/
│       └── page.tsx ← `dashboard` 頁面
```

流程如下：

1. `dashboard/page.tsx` 被載入
2. 它被傳給 `admin/layout.tsx` 的 `children`
3. 然後再傳到最外層 `app/layout.tsx` 的 `children`
4. 最終包成一整頁畫面

□ 重點整理

元素	說明
<code>layout.tsx</code>	定義頁面外框、接收 <code>children</code>
<code>page.tsx</code>	對應 URL 的實際頁面內容
<code>children</code>	<code>layout.tsx</code> 的佔位符，用來顯示下層內容
傳遞邏輯順序	<code>page.tsx</code> → 傳給 → <code>layout.tsx</code> → 再往上層

□ 初學者常見疑問

□ Q1: `children` 是誰傳給 `layout.tsx` 的？

A: 是 Next.js 自動傳進來的，不需要自己傳。

□ Q2: 可以有多個 `layout.tsx` 嗎？

A: 可以。每一層目錄都可以定義自己的 `layout.tsx`，形成巢狀包裝結構。

□ Q3: 如果我不寫 `layout.tsx` 會怎樣？

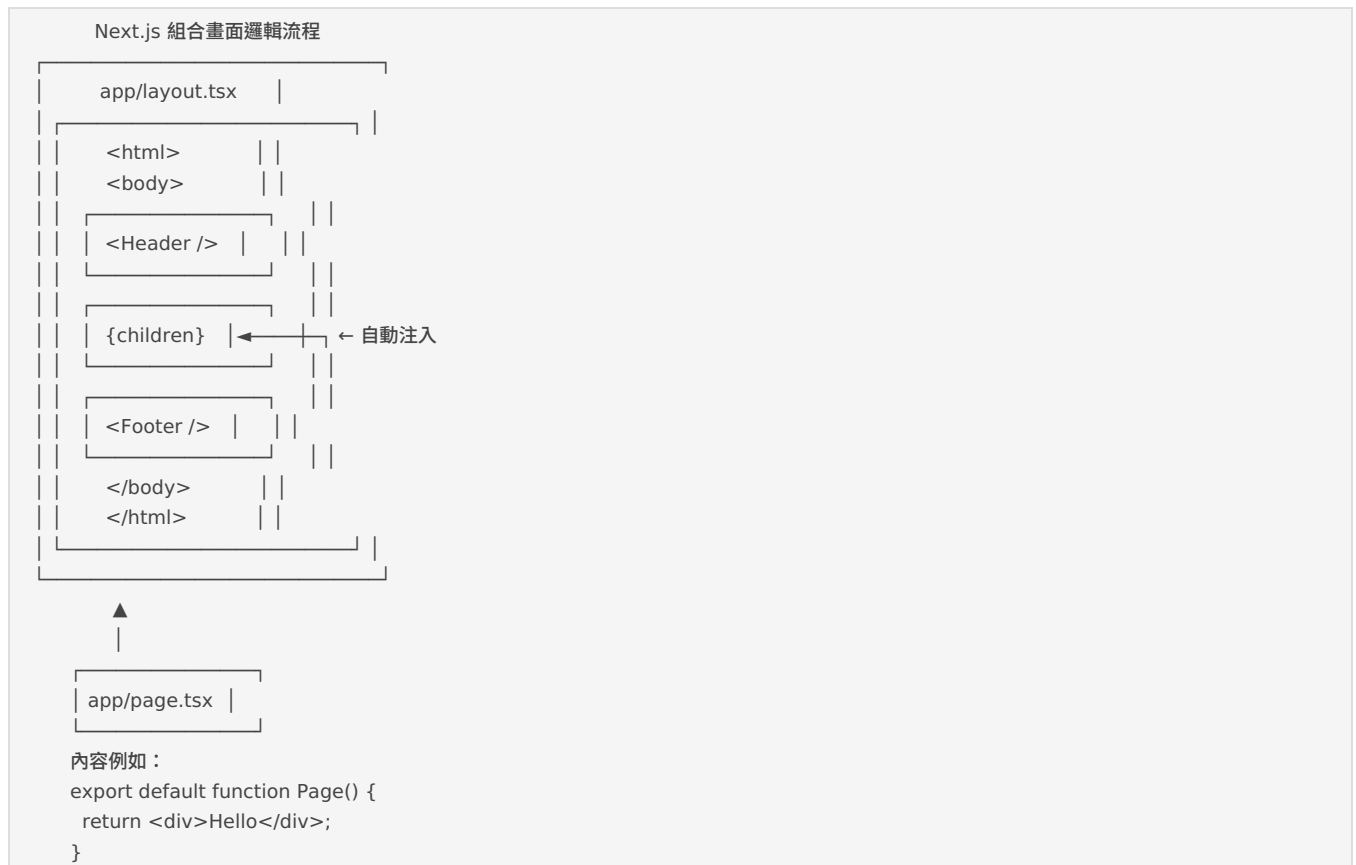
A: 畫面會照樣顯示，但你就無法包共用的元件（像是 Header、Footer、Provider）。

□ 建議實作順序

1. 建立 `app/layout.tsx`：放 header/footer
2. 建立 `app/page.tsx`：顯示首頁
3. 建立 `app/about/page.tsx`：新增新頁面
4. 觀察 `children` 如何把 page 內容塞進 layout

以下是這篇教學的視覺化圖解，讓初學者更清楚了解 `layout.tsx`、`page.tsx`、`children` 在 Next.js App Router 中是如何串接的。

□ 圖解：`layout.tsx` 如何包住 `page.tsx`



□ 圖解說明：

- `layout.tsx` 是整個頁面的外框（包含 `<html>`，`<body>`，Header, Footer）
- `{children}` 是一個「插槽」，Next.js 會自動把 `page.tsx` 的內容放進去
- 你不需要自己寫 props 傳入 `children`，Next.js 幫你處理好
- 如果你有巢狀 layout，每一層都是這樣一層一層包下去

個頁面不要套用全站的 `layout.tsx`，該怎麼做？

這是一個非常常見的情境：

“「在 Next.js App Router 中，某個頁面不要套用全站的 `layout.tsx`，該怎麼做？」

□ 簡短結論

在 App Router 中：

“□ 無法直接讓某個頁面「跳過」`app/layout.tsx`，因為它是全域的根 layout，每一頁都會套用。

但有兩個實務解法可以達到「看起來沒有 layout」的效果：

□ 解法 1：用 Route Group 分離 layout

<https://bookstack.treemanou.com/books/treemanreact/page/nextjsmultiple-root-layouts>

□ 結構：

```
app/
├── layout.tsx          ← 全站 layout (預設套用)
├── (no-layout)/       ← □ 特殊：不繼承全站 layout
│   ├── login/
│   │   ├── layout.tsx ← 空 layout
│   │   └── page.tsx   ← `login`
```

□ `(no-layout)/layout.tsx`

```
export default function NoLayout({ children }: { children: React.ReactNode }) {
  return children; // □ 直接回傳 children，不包任何框架
}
```

“□ 這樣 `/login` 就不會套到 `app/layout.tsx` 了
□ 它只會用 `(no-layout)/layout.tsx`，你可以視為「不使用 layout」

□ 解法 2：在 `layout.tsx` 裡判斷 route 隱藏部

分元件

如果你不想拆資料夾，也可以：

```
'use client'

import { usePathname } from 'next/navigation'
import Header from '@components/Header'
import Footer from '@components/Footer'

export default function RootLayout({ children }: { children: React.ReactNode }) {
  const pathname = usePathname()
  const isPlainPage = pathname.startsWith('/login') || pathname.startsWith('/print')

  return (
    <html>
      <body>
        {!isPlainPage && <Header />}
        <main>{children}</main>
        {!isPlainPage && <Footer />}
      </body>
    </html>
  )
}
```

“ `/login` 和 `/print` 等頁面就會不顯示 Header / Footer
 但 layout 本身還是存在，只是條件渲染內容

比較小結

方式	優點	缺點
<input type="checkbox"/> 用 Route Group 拆 layout	完全不繼承任何 layout	需要多一層目錄
<input type="checkbox"/> layout 判斷條件顯示元件	不需要改目錄結構	layout 實際還是存在，略為複雜

建議

使用情境	建議方式
登入頁、列印頁等需要極簡畫面	<input type="checkbox"/> 用 Route Group 分離 layout
特定頁只想隱藏 header/footer 等元件	<input type="checkbox"/> layout 內用 <code>usePathname()</code> 判斷

如果你需要範例專案或要把這部分整合進教學文章，我可以幫你補上 Markdown + 圖解，要嗎？

【Next.js】Multiple Root Layouts

當你想在 **Next.js App Router** 中讓不同的頁面使用 **不同的根 layout (Multiple Root Layouts)**，你可以透過「**路由分組 (Route Groups)**」來達成。

這是一種官方推薦的做法，可以讓你做到：

“不同路由區段有不同的外框架 (Header、Sidebar、樣式主題...)，互不影響。”

為什麼需要 Multiple Root Layouts ?

你可能有這些需求：

區塊	特性
/	使用全站基本 layout
/admin	使用管理後台 layout (含側邊欄)
/auth	不需要 layout，例如登入頁

解法：使用 Route Groups 分離 layout

語法關鍵：(group-name) ← 這是 **Route Group**

它：

- 不會出現在 URL 中
- 可以定義自己的 layout.tsx
- 可以做到多個「Root Layout」共存

範例結構：三個 Root Layouts

```
app/
├── (site)/
│   ├── layout.tsx    ← 公開區域 layout
│   ├── page.tsx     ← ` `
│   └── about/
│       └── page.tsx  ← `/about`
├── (admin)/
│   ├── layout.tsx   ← 管理區 layout (有側邊欄)
│   └── dashboard/
│       └── page.tsx ← `/dashboard`
├── (auth)/
│   ├── layout.tsx  ← 登入註冊 layout (極簡)
│   ├── login/
│   └── page.tsx    ← `/login`
```

每個 layout.tsx 都是自己的根 layout

app/(site)/layout.tsx :

```
export default function SiteLayout({ children }: { children: React.ReactNode }) {
  return (
    <html>
      <body>
        <header>網站主選單</header>
        <main>{children}</main>
        <footer>頁尾</footer>
      </body>
    </html>
  );
}
```

app/(admin)/layout.tsx :

```
export default function AdminLayout({ children }: { children: React.ReactNode }) {
  return (
    <html>
      <body>
        <aside>後台選單</aside>
        <section>{children}</section>
      </body>
    </html>
  );
}
```

app/(auth)/layout.tsx :

```
export default function AuthLayout({ children }: { children: React.ReactNode }) {
  return (
    <html>
      <body>
        {children} {/* 沒有 header/footer */}
      </body>
    </html>
  );
}
```

□ 注意事項

問題	說明
Group 名稱會出現在網址嗎？	<input type="checkbox"/> 不會出現，只是用來分開 layout 結構
可以混用 layout 嗎？	<input type="checkbox"/> 每個 group 是獨立 root layout
可以共用 components 嗎？	<input type="checkbox"/> 沒問題，只要透過 import 即可

□ 小結

技術	說明
Route Group <code>(xxx)</code>	幫你在不改網址的前提下拆分 layout 結構
Multiple Layouts	實現不同頁面區塊有各自獨立外觀、結構
頁面掛在哪個 group 就用哪個 layout	完全分離，互不影響

【Next.js】Link 元件

當然可以！

□ Next.js 中的 `<Link>` 元件完整說明 (針對 App Router)

□ 什麼是 `<Link>` ？

`<Link>` 是 Next.js 提供的**內部導頁元件**，用來在不同頁面之間切換（而不會重新載入整個網頁）。

它的作用類似於 HTML 的 `<a>` 標籤，但它是：

- 加上 **預先載入 (prefetch)**
- 使用 **Client-side navigation (客戶端跳轉)**
- 避免整頁重新載入，更快更流暢

□ 基本用法

```
import Link from 'next/link'

export default function HomePage() {
  return (
    <div>
      <h1>首頁</h1>
      <Link href="/about">前往 About 頁面</Link>
    </div>
  );
}
```

□ 對應效果：

```
<a href="/about">前往 About 頁面</a>
```

但點擊時會走 Next.js 的路由系統，不會整頁 reload。

□ 為什麼不用 `` ？

因為 `<a>` 是瀏覽器原生跳轉：

- □ 整頁重新載入
- □ 丟失 React 狀態
- □ 破壞 Single Page App 流暢性

而 `<Link>` 則使用 **client-side routing**：

- < 更快，不會重載頁面
- □ 保留 layout 與資料
- □ 支援 App Router 的預先載入

□ 支援的 props (常見)

Prop	說明
<code>href</code>	□ 必填，跳轉目標
<code>prefetch</code>	是否預先載入 (預設 <code>true</code>)
<code>replace</code>	不加入瀏覽器歷史紀錄 (像 <code>location.replace</code>)
<code>scroll</code>	是否自動捲動到頂部 (預設 <code>true</code>)
<code>locale</code>	多語系路由時指定語言

□ 帶參數用法

```
<Link href="/product/123">商品詳情</Link>
```

對應 `app/product/[id]/page.tsx`

□ 包含其他元素 (像按鈕)

```
<Link href="/about">
  <button>了解更多</button>
</Link>
```

□ 搭配動態路由 (含物件參數)

```
<Link href={{ pathname: '/product/[id]', query: { id: '123' } }}>
  查看商品
</Link>
```

□ 注意事項

規則	說明
<code><Link></code> 只能用於內部路由	外部網站請用 <code></code>
不需要手動加 <code>as</code>	App Router 不需要像舊版 Pages Router 那樣加 <code>as</code>
可以搭配 <code>usePathname()</code>	用來比對當前路徑，製作「選單高亮」

□ 實務建議

使用場景	建議寫法
切換內部頁面	用 <code><Link href="/about"></code>
導向外部網站	用 <code></code>
高亮當前路徑	搭配 <code>usePathname()</code>

□ 小結

功能	<Link>
路由切換	□ Client-side navigation
預先載入	□ 支援 prefetch
整頁 reload	□ 不會 reload，保持 SPA 流暢性
搭配 layout	□ Layout 不會重新渲染

在 Next.js 中，**Active Link (目前頁面導覽高亮)** 是一個常見的 UI 需求。當使用者正在某個頁面時，對應的導覽列連結 (nav bar) 應該加上「目前頁面樣式」來提示使用者。

□ Active Link 是什麼？

就是讓當前頁面的導覽項目加上特殊樣式，例如：

```
<a href="/about" class="active">About</a>
```

讓使用者知道「我現在在這個頁面」。

□ Next.js 怎麼做 Active Link？

Next.js 並不自帶 active class 功能，需要我們手動處理。但 App Router 搭配 `usePathname()` 非常方便。

□ 使用 `usePathname()` 判斷當前路徑

□ 範例：ActiveLink.tsx 元件

```
// components/ActiveLink.tsx
"use client";

import Link from "next/link";
import { usePathname } from "next/navigation";
import clsx from "clsx"; // optional, 幫助管理 class

type Props = {
  href: string;
  children: React.ReactNode;
};

export default function ActiveLink({ href, children }: Props) {
  const pathname = usePathname();
  const isActive = pathname === href;

  return (
    <Link
      href={href}
      className={clsx("px-4 py-2", {
        "text-blue-600 font-bold underline": isActive,
        "text-gray-600": !isActive,
      })}
    >
      {children}
    </Link>
  );
}
```

```
);  
}
```

□ 使用方式

```
// app/layout.tsx 或其他 Nav 元件中  
import ActiveLink from "@components/ActiveLink";  
  
export default function NavBar() {  
  return (  
    <nav className="flex gap-4">  
      <ActiveLink href="/">Home</ActiveLink>  
      <ActiveLink href="/about">About</ActiveLink>  
      <ActiveLink href="/products">Products</ActiveLink>  
    </nav>  
  );  
}
```

□ 小技巧

- 路徑可以使用 `startsWith()` 判斷「巢狀頁面」也高亮

```
const isActive = pathname.startsWith(href)
```

- 可以抽出 `activeClassName` 與 `className` 讓元件更彈性

□ 顯示效果

當你瀏覽 `/products` 頁面時：

```
<a href="/products" class="text-blue-600 font-bold underline">
```

□ 若你使用 Tailwind，可用 `clsx` 管理 class：

```
pnpm add clsx  
# or  
npm install clsx
```

□ 小結

工具	用途
<code>usePathname()</code>	取得目前 URL 路徑
<code>clsx</code>	根據條件切換 class
自訂 <code>ActiveLink</code>	建立可重用的 active 判斷元件

【Next.js】metadata 設定總覽

以下是 Next.js App Router 中 `metadata` 常用設定的整理表，適合初學者參考與實作時使用。

□ Next.js App Router `metadata` 設定總覽

“用於自動產出 SEO-friendly 的 `<title>`，`<meta>`，`<link>` 等標籤

□ 基本結構

```
export const metadata: Metadata = {
  title: "單一頁面標題",
  description: "這是這一頁的描述",
};
```

會輸出：

```
<title>單一頁面標題</title>
<meta name="description" content="這是這一頁的描述">
```

□ 多層 Layout 組合使用：template + default

```
// app/layout.tsx
export const metadata = {
  title: {
    default: "My Site",
    template: "%s | My Site",
  },
};
```

```
// app/about/page.tsx
export const metadata = {
  title: "About",
};
```

□ 實際輸出：

```
<title>About | My Site</title>
```

□ 絕對標題覆蓋 template

```
export const metadata = {
  title: {
    absolute: "登入中，請稍候...",
  },
};
```

```
},  
};
```

□ 實際輸出：

```
<title>登入中，請稍候...</title>
```

□ 設定 Open Graph、Twitter Card

```
export const metadata = {  
  title: "About",  
  description: "關於我們的介紹",  
  openGraph: {  
    title: "About Page",  
    description: "OpenGraph 專用描述",  
    url: "https://example.com/about",  
    siteName: "My Site",  
    images: [  
      {  
        url: "https://example.com/og-image.jpg",  
        width: 1200,  
        height: 630,  
      },  
    ],  
  },  
  twitter: {  
    card: "summary_large_image",  
    title: "About Page",  
    description: "Twitter 用的描述",  
    images: ["https://example.com/twitter-image.jpg"],  
  },  
};
```

□ 禁止搜尋引擎索引 (noindex)

```
export const metadata = {  
  robots: {  
    index: false,  
    follow: false,  
  },  
};
```

□ 實際輸出：

```
<meta name="robots" content="noindex, nofollow">
```

□ 設定 favicon 與 manifest

```
export const metadata = {  
  icons: {  
    icon: "/favicon.ico",  
    shortcut: "/shortcut-icon.png",  
    apple: "/apple-touch-icon.png",  
  },  
  manifest: "/site.webmanifest",  
};
```

☐ 常見欄位對照表

欄位名稱	說明
title	頁面標題，可以是 string 或物件
description	meta 描述，用於搜尋結果
openGraph	Facebook/LinkedIn 等分享資訊
twitter	Twitter 分享資訊
robots	是否讓搜尋引擎索引這頁
icons	頁籤圖示、apple icon 等
manifest	PWA 設定檔

☐ 使用建議

使用情境	建議設定
一般頁面	title + description 即可
分享卡片 / 社群預覽	加上 openGraph 與 twitter
後台或登入頁	可用 title.absolute + noindex

當然可以！下面我將清楚說明 Next.js `metadata` 中的 `openGraph` 與 `twitter` 欄位的用途、格式與實戰例子，幫助你製作具有社群分享預覽功能的頁面。

☐ openGraph 與 twitter 是什麼？

當你把一個網頁連結貼到 **Facebook**、**Line**、**Slack**、**Twitter** 等社群平台，它會自動產生「預覽卡片」：

“☐ 圖片
☐ 標題
☐ 描述
☐ 網站名稱

這些內容的來源就是：

- Facebook、Line：讀取 **Open Graph meta tags**
- Twitter：讀取 **Twitter Card meta tags**

Next.js 的 `metadata.openGraph` 與 `metadata.twitter` 就是自動產生這些 tags 的設定方法。

☐ openGraph 結構（用於 Facebook / Line / LinkedIn）

```
export const metadata = {
  openGraph: {
    title: "產品名稱 - 超好用的工具",
    description: "這是一款改變你人生的產品。",
    url: "https://yourdomain.com/product/123",
    siteName: "My Cool App",
    type: "website", // 常見還有: 'article', 'product', 'profile'
    locale: "zh_TW",
    images: [
```

```
{
  url: "https://yourdomain.com/images/product-123.jpg",
  width: 1200,
  height: 630,
  alt: "產品圖說",
},
],
},
};
```

□ 對應輸出的 HTML meta :

```
<meta property="og:title" content="產品名稱 - 超好用的工具" />
<meta property="og:description" content="這是一款改變你人生的產品。" />
<meta property="og:url" content="https://yourdomain.com/product/123" />
<meta property="og:site_name" content="My Cool App" />
<meta property="og:type" content="website" />
<meta property="og:locale" content="zh_TW" />
<meta property="og:image" content="https://yourdomain.com/images/product-123.jpg" />
<meta property="og:image:width" content="1200" />
<meta property="og:image:height" content="630" />
<meta property="og:image:alt" content="產品圖說" />
```

□ twitter 結構 (用於 Twitter)

```
export const metadata = {
  twitter: {
    card: "summary_large_image", // summary / summary_large_image / app / player
    title: "這是商品標題",
    description: "這是商品描述",
    images: ["https://yourdomain.com/images/product-123.jpg"],
    creator: "@your_twitter_id", // 可選
  },
};
```

□ 對應輸出的 HTML meta :

```
<meta name="twitter:card" content="summary_large_image" />
<meta name="twitter:title" content="這是商品標題" />
<meta name="twitter:description" content="這是商品描述" />
<meta name="twitter:image" content="https://yourdomain.com/images/product-123.jpg" />
<meta name="twitter:creator" content="@your_twitter_id" />
```

□ 圖片尺寸建議

平台	尺寸建議
Facebook	1200×630 px (1.91:1 長方形)
Twitter	1200×628 px (summary_large_image)
LINE	同 Facebook (讀 og:image)

□ 實戰檢查工具

工具名稱	連結
Facebook Debugger	https://developers.facebook.com/tools/debug/
Twitter Card Validator	https://cards-dev.twitter.com/validator

工具名稱	連結
OpenGraph Preview	https://www.opengraph.xyz

□ 小結

項目	openGraph	twitter
使用平台	Facebook, Line, LinkedIn, Slack 等	Twitter 專用
支援圖片	□ 多張圖片	□ 單張或多張
文字自訂	□ title, description, siteName 等	□ title, description, creator 等
SEO 關聯	□ 有助於點擊率與品牌識別	□ Twitter 分享時更具吸引力

如果你需要我產出一個 Next.js 頁面範例，實作一頁完整的 SEO + 分享卡片（含圖片），我可以幫你快速產一份喔！要嗎？

【Next.js】常用 scripts 說明

這段 "scripts" 通常定義在你的 `package.json` 中，是 Next.js 專案開發流程的基本指令集。下面我會逐一說明每個指令的用途、情境，以及何時使用：

□ "scripts" 說明

```
"scripts": {  
  "dev": "next dev",  
  "build": "next build",  
  "start": "next start",  
  "lint": "next lint"  
}
```

1. "dev": "next dev"

□ 本地開發環境使用，啟動熱重載伺服器。

- □ 支援 hot reload (你改檔案會自動重新載入)
- □ 開啟 <http://localhost:3000>
- □ 用於：開發、除錯、畫面確認、檢查 API

□ 情境：

```
npm run dev  
# 或  
pnpm dev
```

2. "build": "next build"

□ 打包應用程式，產生 `.next/` 目錄，準備部署。

- 會做：
 - 頁面預先產生 (SSG)
 - Server-side 轉譯編譯 (SSR)
 - Minify, tree shaking, code split...
- □ 不會啟動伺服器

□ 情境：

```
npm run build  
# -> 打包專案供 production 用
```

3. "start": "next start"

□ 啟動 Production 模式伺服器

- 需要先執行過 `next build`
- 適合用於正式環境或模擬正式環境 (staging)
- 沒有 hot reload (因為是 production)

□ 情境：

```
npm run build  
npm start  
# -> 用於正式上線或部署伺服器中
```

4. "lint": "next lint"

☐ 使用 Next.js 的內建 ESLint 設定檢查程式碼品質。

- 檢查：變數命名、語法錯誤、不一致縮排
- 如果你有設定 ESLint 設定檔（如 `.eslintrc.js`）會套用規則

☐ 情境：

```
npm run lint
# -> 在部署前自動檢查程式碼品質
```

☐ 小結（對照表）

指令	用途	執行時機	是否正式用途
<code>dev</code>	啟動開發伺服器	開發階段	☐
<code>build</code>	建立正式環境可執行檔	上線前打包	☐
<code>start</code>	啟動正式伺服器	上線後／模擬上線	☐
<code>lint</code>	檢查語法與格式	開發中、CI 中自動化流程檢查	☐

如果你想再加上測試、自動部署或 CI/CD 的 script，我也可以幫你補上建議範本。需要嗎？

【Next.js】部署模式總覽

下面整理 Next.js 的主要部署類型、每種模式的特性與常見應用場景，並特別解釋 standalone 是什麼以及它適合的情境。

1 Next.js 部署模式總覽

部署類型	next.config.js / 指令	產物	特性與適用場景
1. Node Server (傳統模式)	<code>next build && next start</code>	<code>.next/</code>	預設，完整支援 SSR、ISR、API Routes。部署到任何 Node 環境 (PM2、Docker、K8s、AWS EC2、GCP、Heroku...)。
2. Standalone	<code>output: 'standalone'</code> + <code>next build</code>	<code>.next/standalone</code> + <code>.next/static</code>	只輸出最小可執行檔 (server.js + 必要檔案)，方便 Docker/容器化。功能與 Node Server 相同 (SSR/ISR/API Routes 皆可)。
3. Static Export	<code>next export</code> 或 <code>output: 'export'</code>	<code>out/</code>	完全靜態 HTML/CSS/JS。無 SSR/ISR/API Routes。適合純內容網站、部落格、文件站。
4. Edge Runtime / Vercel	<code>runtime: 'edge'</code> 或自動判斷	Vercel Edge Functions (或 Cloudflare Workers)	以 Web 標準 API 為基礎，無 Node built-ins。啟動極快、全球節點分布，適合 Middleware、即時驗證、A/B 測試。
5. Serverless Functions	Vercel / AWS Lambda	Lambda zip	每個頁面/Route 轉成獨立 Serverless Function。可自動擴展，適合不維護長駐伺服器的架構。
6. Custom Server	自行建立 <code>server.js</code> 並 <code>next()</code>	<code>.next/</code>	自定義 Node 伺服器，整合 Koa/Express/Fastify，做複雜路由或中介層。

註：模式之間可以組合，例如 Node Server + Standalone、Vercel Serverless + Edge Middleware。

模式	需 Node.js Runtime	常見部署基底	代表場景
Node Server	☐	Node image / VM	全功能 SSR/ISR + API
Standalone	☐ (但包體精簡)	Node image / VM	Docker/K8s 輕量部署
Static Export	☐	Nginx、GitHub Pages、S3+CDN	純靜態內容
Edge Runtime	☐ (Web Worker API)	Vercel Edge / Cloudflare Workers	全球低延遲運算、Middleware
Serverless Functions	☐ (平台提供)	AWS Lambda / Vercel Functions	無伺服器、自動擴展
Custom Server	☐	Node image / VM	自訂路由、混合架構

2 各模式詳解

① Node Server (預設)

- 啟動：`next build && next start`
- 產物：`.next/` 目錄
- 特性：完整支援 SSR、ISR、API Routes。
- 適用：自行管理伺服器 (EC2、Kubernetes、Docker Swarm、Heroku...)。
- 優缺：最彈性，但部署包體積較大，需要完整 `node_modules`。

② Standalone ☐

- 設定：

```
// next.config.js
module.exports = {
  output: 'standalone',
}
```

- 產物：

```
.next/standalone # 精簡 Node 專案，可直接 node server.js
.next/static     # 靜態資源
```

- **原理：**
 - `next build` 會分析依賴，只把執行需要的檔案、`node_modules` 打包進 `standalone`。
 - 不再需要完整專案原始碼，也不必把整個 `node_modules` 打進 Docker。
- **適用場景：**
 - Docker / 容器化：
只複製 `.next/standalone` + `.next/static`，image 更小、build 更快。
 - Kubernetes / ECS：
配合多階段 Docker build，把 `RUN npm ci` 階段與最終執行環境分離。
- **功能：**與 Node Server 相同 (SSR / ISR / API Routes 全支援)。

③ Static Export

- **指令：**`next build && next export`
- **產物：**`out/` 完全靜態
- **限制：**無 SSR、無 API Routes、無 ISR。
- **適用：**
 - 部落格、說明文件、Landing Page
 - 部署到任何靜態空間 (S3+CloudFront、GitHub Pages、Netlify)。

④ Edge Runtime / Middleware

- **用法：**在頁面或 route handler 中 `export const runtime = 'edge'`
- **部署：**Vercel Edge Functions、Cloudflare Workers、Deno Deploy。
- **特性：**全球節點極低延遲；不能用 Node built-ins。
- **適用：**
 - 驗證/權限檢查
 - A/B 測試、URL 轉向
 - 即時 Header/Cookie 操作

⑤ Serverless Functions (Vercel / AWS Lambda)

- **部署：**Vercel 自動化，或 `serverless-http` 將 Next 整包轉為 Lambda。
- **特性：**
 - 每個頁面/路由轉成獨立 Lambda。
 - 自動擴展、免維運。
- **適用：**
 - 高併發但流量不穩定。
 - 只需 Node 短期計算、不需長連線。

⑥ Custom Server

- **方式：**建立 `server.js`

```
const express = require('express')
const next = require('next')
const app = next({ dev: false })
const handle = app.getRequestHandler()
app.prepare().then(() => {
  const server = express()
  server.get('/health', (req, res) => res.send('ok'))
  server.all('*', (req, res) => handle(req, res))
  server.listen(3000)
})
```

- **適用：**
 - 需要與既有 Node 架構整合。
 - 自訂路由、中介層或 WebSocket。

3 如何選擇

需求	建議模式
企業內部、自管 Docker/K8s	Standalone (Node Server) : 產物小、部署快
純靜態內容、無後端邏輯	Static Export
全球極低延遲、近使用者運算	Edge Runtime / Middleware
彈性擴展、免維運	Serverless Functions (Vercel / AWS Lambda)
需與自訂 Node 架構整合	Custom Server

□ 總結

- **Standalone** 是 **Node Server** 模式的輕量化打包，非常適合容器化部署。
- 如果要保留完整 SSR、ISR、API Routes，又希望 **映像檔小、啟動快**，選 **Standalone** 幾乎是最佳解。
- 其他模式 (Static Export、Edge、Serverless) 則依你的應用是否需要即時運算、全球分佈或純靜態內容來決定。

□ Next.js 部署模式功能支援度

部署模式	需要 Node Runtime	SSR	ISR	API Routes	適用場景
Node Server (next build && next start)	☐	☐	☐	☐	傳統部署，功能完整。適合自管伺服器、Docker、K8s。
Standalone (output: 'standalone')	☐	☐	☐	☐	與 Node Server 相同，但打包精簡，適合容器化。
Static Export (next export)	☐	☐	☐	☐	只能輸出純 HTML/JS/CSS。適合靜態網站 (部落格、Landing Page)。
Serverless Functions (Vercel / AWS Lambda)	☐ (平台提供)	☐	☐	☐	拆分成獨立 Functions，自動擴展。適合流量不穩的專案。
Edge Runtime (runtime: 'edge')	☐ (Web API 而非 Node)	△ 部分支援 (僅 Server Components, 可撈 API)	☐	△ 部分支援 (需 Web API, 不支援 Node drivers)	全球低延遲運算、Middleware、A/B 測試。
Custom Server (Express/Koa/Fastify + Next)	☐	☐	☐	☐	自訂路由/中介層，整合既有 Node 架構。

□ 補充說明

- **Static Export** : 因為沒有 Node 伺服器 → **無法即時渲染** (SSR)、**無法再生** (ISR)、**無 API Routes**。
- **Edge Runtime** :
 - 可以做 SSR (例如 Server Components 即時抓資料)，但**無 Node API** (不能用 fs、原生 DB drivers)。
 - 適合處理 Header、Cookie、權限驗證、低延遲資料讀取。
- **Serverless Functions** : Vercel/AWS 會把 SSR/ISR/API Route 拆成獨立 Function → 自動擴展，但要注意冷啟動與 DB 連線池。

□ 總結

- 要完整功能 (SSR/ISR/API Routes) → 用 **Node Server** 或 **Standalone**。
- 要純靜態內容 → 用 **Static Export**。
- 要全球低延遲 → 用 **Edge Runtime** (但功能有限制)。
- 要免維運，自動擴展 → 用 **Serverless Functions** (Vercel、AWS Lambda)。