第 3 章 - 檔案結構

Pages(頁面)

Pages(頁面)在 Next.js 中是 React 的元件，會自動以檔案路由的方式執行。

頁面需要放在`pages`目錄中預設輸出(default exports)，可以支援的檔案類型為`.js`, `.jsx`, `.ts`, or `.tsx`。

以下是一個典型的 Next.js 應用的目錄結構，在最上層的目錄有 `public`, `pages` , `styles`幾個，如下圖所示:

```
next-app
├── node_modules
├── pages
│   ├── index.js // path: base-url (/)
│   ├── about.js // path: /about
│   └── products.js // path: /products
├── public
├── styles
├── .gitignore
├── package.json
└── README.md
```

每個頁面都是 React 的元件，它扮演的角色類似於"路由處理器"(route handlers)，每個頁面都會有對應的路由，由它的檔案名稱決定，`index.js`代表是該目錄中的基礎網址(或稱為"預設路由")。

```
// 位置: /pages/index.js
// <Home /> is just a basic React component
export default function Home() {
  return <h1>Welcome to Next.js</h1>
}
```

> 注意: React 中函式型元件的名稱命名按規定一定要"英文開頭大寫"。

> 注意: 在`pages`目錄中只能用於儲放頁面，其它周邊的樣式、組合用子女元件或版面元件，**請勿放置**在這個目錄中，需要在另外的目錄中放置。主要原因是 Next 會將所有在此目錄中的 React 元件，視作是某個單獨頁面。

Custom Pages(自訂頁面)

在`pages`目錄中有幾個檔案是有特殊的用途，可讓開發者針對需求調整，尤其是針對所有頁面使用的功能部份。在這目錄中建立後可以覆蓋原先內建預設的，所以稱為"自訂頁面"(Next 內部有預設的，開發者可以自訂需要的)。

`_app.js`

說明: Next 使用它作為全網站的各種初始化工作

用途:

- 在各頁面中維持排版
- 當在導覽各頁面時保持狀態(context)
- 注入額外的資料到各頁面中
- 加入全域的 CSS

```
// 位置: pages/_app.js
export default function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />
}
```

_document.js

> 註: 這個自訂頁面只有少數情況會自訂程式碼，通常見到會是用來整合特定的第三方 UI 庫

說明:

Next 使用它在頁面渲染時，更新<html> 與 <body>標記。因為這個檔案只會在伺服器端渲染，因此像事件處理函式例如 onClick 是不能在這裡面使用。

在這個檔案中的用途會像是，例如設定 HTML 語言(lang)，載入字體(fonts)、在網頁開始互動前載入自訂 script 檔案，或是集中 CSS 樣式(例如 Styled Components 解決方案使用)

範例:

```
// 位置: pages/_document.js
import { Html, Head, Main, NextScript } from 'next/document'

export default function Document() {
  return (
    <Html>
      <Head />
      <body>
        <Main />
        <NextScript />
      </body>
    </Html>
  )
}
```

404

404 錯誤頁面為"找不到網頁錯誤"，這個是常見的錯誤頁面。注意此頁面將會在打包後產生靜態頁面。

```
// 位置: pages/404.js
export default function Custom404() {
```

```
  return <h1>404 - Page Not Found</h1>
}
```

500

500 錯誤頁面為"內部伺服器錯誤"，這個是常見的錯誤頁面。注意此頁面將會在打包後產生靜態頁面。

```
// 位置: pages/500.js
export default function Custom500() {
  return <h1>500 - Server-side error occurred</h1>
}
```

註: [進階]500 錯誤自訂頁面可以透過next/error元件與自訂pages/_error.js來進行進階的覆蓋，但這只能在營運階段運行。

_error

自訂錯誤頁面為，這只能在營運階段運行。它通常可以在測試期間進行除錯用。以下為官網的範例:

```
// 位置: pages/_error.js
function Error({ statusCode }) {
  return (
    <p>
      {statusCode
        ? `An error ${statusCode} occurred on server`
        : 'An error occurred on client'}
    </p>
  )
}

Error.getInitialProps = ({ res, err }) => {
  const statusCode = res ? res.statusCode : err ? err.statusCode : 404
  return { statusCode }
}

export default Error
```

_app.js與_document.js的比較與使用情況

_document.js的使用情況是關乎全站，除了上述說的設定語言、字體外，另外設定全域使用的 meta 標記也是可以，但要注意無法在其中設定 <title>。通常要設定頁面中的 <head> 標記的內容，需要動態依照每個頁面產生，Next 提供了額外的next/head元件提供更多的彈性設定。

結論是，官方建議在這裡面只設定<html> 與 <body>的 css 類別即可，減少對_document.js這檔案的設定。甚至可以不用這個檔案設定<head>標記，另外為了要相容於 React 18 版本後功能改變 ，也盡量避免在這裡面使用getInitialProps與 renderPage方法。

_app.js相較於_document.js提供了更多初始頁面的設定彈性，以下以範例來說明:

設定頁面標題(title)、圖示、meta 標記(全站使用):

```
// 位置: pages/_app.js
import Head from 'next/head'

function MyApp({ Component, pageProps }) {
  return (
    <>
      <Head>
        <title>My new cool app</title>
        <meta
          name="description"
          content="This is a desription for My Next App"
        />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <Component {...pageProps} />
    </>
  )
}
```

導入全域 css:

```
// 位置: pages/_app.js
import '../styles/globals.css'

function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />
}

export default MyApp
```

載入第三方的 script 檔案:

```
// 位置: pages/_app.js
import Script from 'next/script'

export default function MyApp({ Component, pageProps }) {
  return (
    <>
      <Component {...pageProps} />
      <Script src="https://example.com/script.js" />
    </>
  )
}
```

Google 字體套用:

```
// 位置: pages/_app.js
import { Inter } from 'next/font/google'

const inter = Inter({ subsets: ['latin'] })

export default function MyApp({ Component, pageProps }) {
  return (
    <main className={inter.className}>
      <Component {...pageProps} />
    </main>
  )
}
```

動態套用 meta 標記與 title 要透過排版(layouts)元件來處理經常是最適合解決方案:

```
// 位置: components/layout.js
const Layout = ({
  children,
  title = 'default title',
  description = 'default description',
}) => {
  return (
    <>
      <Head>
        <meta name="description" content={description} />
        <title>{title}</title>
      </Head>
      <Navbar />
      <main>{children}</main>
      <Footer />
    </>
  )
}

// 位置: pages/my-page.js
const MyPage = (props) => {
  return (
    <Layout title="the-dynamic-title" description="the-dynamic-
description">
      {/* 其它頁面的內容置於此 */}
    </Layout>
  )
}
```

各資料夾說明

以下分類僅供參考，實際上這已經足夠針對各種中小型規模的專案使用。要依需求調整也可遵守幾個原則:

- 閱讀性: 命名對照要清楚明確，容易一眼看出是什麼為主
- 共用性: 有共用、共享的函式、樣式...等等，儘量獨立出來一個檔案或元件，特別對小組開發有幫助
- 維護管理擴充性

`assets`

程式碼內部要用導入(import)的各圖片、字體、媒體檔。

> 註: 如果使用的圖片(或媒體檔)的來源(src)是網站運作的相對連結，例如`<Image src="/vercel.svg"/>`，這種圖片是放在`public`資料夾中

> 註: 如果使用例如像 svg 直接寫入程式碼原始碼，以 jsx 語法寫成元件的方式，直接放置在`components/icons`或`components/svg-images`會較佳，不需要放在這個資料夾中。

`components`目錄

此目錄儲放所有 React 相關元件，主要有幾種常用的:

- common: 通常的。共享的、未分類的、常用的、測試用的小元件等等...評分元件(Rating)、按鈕...
- layout: 與排版有關。各頁面用排版元件，導覽列(Navbar)、選單列(Menu)、工具列(Toolbar)、側邊列(Sidebar)、頁尾(Footer)、版權區塊(CopyRight)、麵包屑(Breadcrumb)等等，這些會與排版(Layout)共同組成，所以放置在同一資料夾中
- 對應各 page 取名專用資料夾(shop, cart, user, member, admin, dashboard...): 對應各 page 的專用元件

> 註: 如果有使用 css module 技術，或某種用來樣式化單一元件的 css 或 sass 檔案，應和元件檔案放入一起，用資料夾儲放較佳。不要獨立放到 styles 資料夾。

`context`

React 的上下文(context)的專門存放目錄，context 是用來共享各元件狀態用的功能。

> 註: 有些勾子裡面也有使用 context，放於 `hooks` 即可。

`data`

範例、模擬資料、資料庫 sql 檔案存放處

`lib`

對應 Next 中 API router 用的函式庫，用於 SSR 為主，通常會用來與第三方函式庫溝通互動用的各函式，例如第三方認証、雲端硬碟、資料庫連接存取...等等。可再分`server`與`client`資料夾區分前後台使用。

`hooks`

React 的勾子(hooks)存放的目錄。勾子是一種能在 React 元件間，重覆利用各種程式邏輯的特殊擴充樣式，僅能用在函式型元件上。

`services`

針對外連伺服器的、RESTful 用、fetch 用的的 api 或函式放置處，通常各頁面或元件也能在裡面獨立寫，並不需要拆分到這裡。會放在這裡通常是共用的，或是要特別拆分出來，在這裡維護才使用。裡面可再區分

`server`與`client`資料夾作前後台使用。

> 註: 有些勾子也有這種類似功能，放於 `hooks` 即可。

`styles`

樣式檔存放處，通常是整個全站用，或是輔助各頁面(pages)用的`.css`、`.scss`、`.module.css`。

> 註: 各元件或版面的自己使用的專用樣式，放於元件對應的資料夾為較佳。

`utils`

一些程式碼片段，輔助用工具函式等等，例如讀取範例檔案、轉換貨幣時區，會放置於此是因為有數個元件會共享使用為主，只有某個元件用到，並不需要特別放在這裡，主要選擇放在這裡的原則，還是會不會共用到，或是日後好不好維護管理等原因。