【名詞解釋】

- 【名詞解釋】SSR、ISR、API Routes

【名詞解釋】SSR、ISR、API Routes

這三個名詞（**SSR、ISR、API Routes**）是 Next.js 的核心特性，常常讓人混淆。下面逐一拆解，說明它們的概念、運作方式、優缺點與應用場景。

1⃣ SSR（Server-Side Rendering）

Server-Side Rendering = 伺服端渲染

- **運作方式**
  每次使用者請求頁面時，Next.js 在 **伺服器端** 執行 React 元件，生成完整 HTML，再回傳給瀏覽器。
- **關鍵點**
  - 每次請求都會「即時」生成頁面內容。
  - 適合需要最新數據的頁面（例如：即時報價、使用者 Dashboard）。

程式碼範例：

```
// pages/index.tsx
export async function getServerSideProps() {
  const data = await fetch('https://api.example.com/data').then(res => res.json())
  return { props: { data } }
}

export default function Home({ data }) {
  return <div>最新數據：{data.value}</div>
}
```

優點

- SEO 友好（HTML 完整輸出）。
- 確保每次都是最新資料。

缺點

- 每次請求都要跑後端 → 伺服器壓力大。
- 延遲較高，不適合高流量純靜態內容。

適用場景

- 會員頁面、個人化內容、後台管理介面。

2⃣ ISR（Incremental Static Regeneration）

ISR = 增量式靜態再生

- **運作方式**
  - 頁面第一次請求時，Next.js 會生成靜態 HTML 並快取。
  - 之後相同頁面會直接回傳快取內容（像 SSG 一樣快）。
  - 在設定的 `revalidate` 時間後，Next.js 會在背景「重新生成」頁面，並更新快取。

程式碼範例：

```
// pages/blog/[id].tsx
export async function getStaticProps() {
  const post = await fetch('https://api.example.com/post/1').then(res => res.json())
  return {
    props: { post },
    revalidate: 60, // 每 60 秒重新生成一次
  }
}
```

優點

- 讀取速度快（靜態 HTML）。
- 自動更新內容，避免手動重建整個網站。
- 減輕伺服器壓力。

缺點

- 更新不是即時的，而是取決於 `revalidate` 時間。
- 若內容更新頻繁，可能仍需要 SSR。

適用場景

- 部落格文章、商品列表、新聞頁面。

---

3 API Routes

API Routes = 內建後端 API

- **運作方式**
  在 Next.js 專案內建立 `pages/api/*` 或 `app/api/*/route.ts`，即可定義一個 API endpoint。
  - 這些 Route 會在 **伺服器端執行**，輸出 JSON 或其他格式。
  - 不需要額外架設 Express/Koa，直接用 Next.js 提供的 Node Runtime。

程式碼範例：

```
// pages/api/hello.ts
export default function handler(req, res) {
  res.status(200).json({ message: 'Hello API!' })
}
```

或（App Router 格式）：

```
// app/api/hello/route.ts
export async function GET() {
  return Response.json({ message: 'Hello API!' })
}
```

優點

- 與前端同專案，部署簡單。
- 適合輕量後端邏輯（驗證、整合第三方 API、Form 提交）。

缺點

- 不適合複雜業務邏輯（相比 Spring Boot / NestJS）。
- 在 Serverless 環境（Vercel、Lambda）會有冷啟動問題。

適用場景

- Contact Form API、登入驗證、前端代理 API。

---

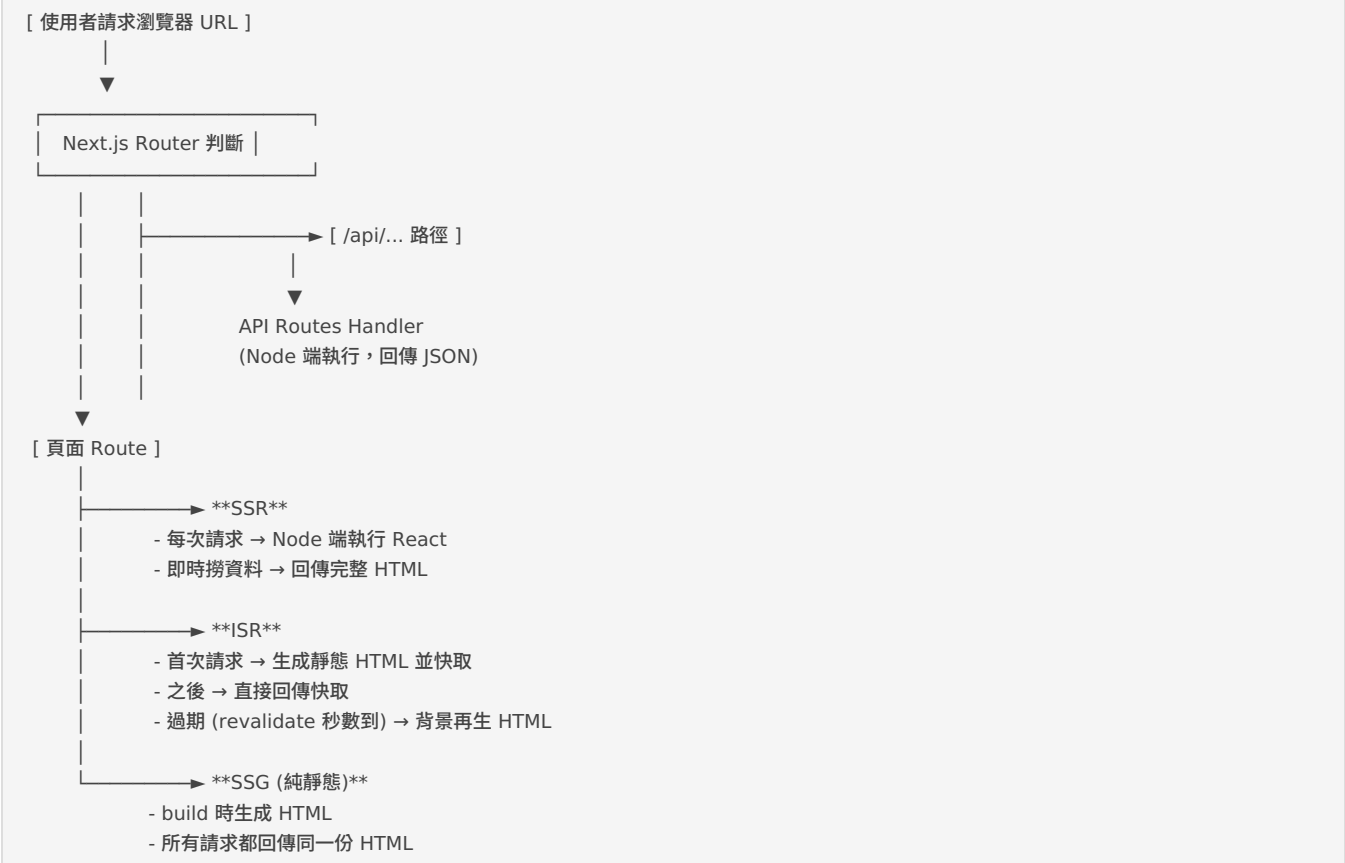
4 對比表

特性	SSR	ISR	API Routes
用途	動態頁面渲染	靜態頁面快取 + 定時更新	提供後端 API
執行時機	每次請求	首次請求生成，之後快取，背景再生	請求時執行（類似 REST endpoint）
效能	中等（依賴伺服器）	快（靜態快取）	視邏輯而定
SEO			不適用（API）
適合場景	即時內容、個人化頁面	部落格、商品頁、新聞	驗證、表單提交、聚合 API

總結

- **SSR**：即時資料，伺服器壓力大，適合動態頁面。
- **ISR**：快取靜態 + 背景更新，效能最佳，適合多內容但非即時需求。
- **API Routes**：Next.js 內建小型後端，適合輕量 API 或 BFF。

Next.js 三種模式請求流程圖



圖示解讀

- **API Routes**：
  - 請求 `/api/*` → 進入 Next.js 的 API Handler
  - 回傳 JSON 或其他格式（例如登入驗證、送表單）
- **SSR (getServerSideProps)**：
  - 每個請求都跑一次伺服端邏輯 → 最新資料
  - 缺點：伺服器壓力大
- **ISR (getStaticProps + revalidate)**：
  - 初次請求生成快取 HTML
  - 之後皆回傳快取 → 非常快
  - 到期時在背景重新生成，更新快取
- **SSG (Static Site Generation)**：
  - build 時就生成 HTML，完全靜態
  - 適合固定內容（FAQ、About Us）

使用建議

- **SSR** → 用於 **即時資料**（會員中心、儀表板、即時價格）。
- **ISR** → 用於 **高流量、定期更新內容**（新聞、部落格、商品頁）。
- **API Routes** → 適合 **輕量 API**（登入、表單處理、資料聚合）。
