Restful API and Ajax

林新德

shinder.lin@gmail.com

0. 參考專案

```
# 本課程後端參考
https://bitbucket.org/lsd0125/mfee47-node.git

# 本課程前端參考
https://bitbucket.org/lsd0125/mfee47-next.git
```

```
# 網站使用 Google 帳號登入 (後端)
https://github.com/shinder/signin-google-try.git
```

1. Restful API

- REST為Representational State Transfer（表現層狀態轉換）的縮寫。
- 2000年由 Dr. Roy Thomas Fielding 在其博士論文中提出的 HTTP 資料交換風格。

1.1 什麼是 Restful API

- 要點：

1. 以 URI 指定資源，使用 HTTP 或 HTTPS 為操作協定。
2. 透過操作資源的表現形式來操作資料。
3. 就是以 HTTP 的 GET, POST, PUT, DELETE 方法對應到操作資源的 CRUD。
4. 資源的表現形式沒有限定，可以是 HTML, XML, **JSON** 或其它格式。
5. REST 是設計風格並**不是標準**，所以沒有硬性的規定。

- 實作 REST 的 後端 API 一般稱作 Restful API

1.2 以商品資料為說明

- 取得列表：
- http://my-domain/products (GET)
- 取得單項商品：
- http://my-domain/products/17 (GET)
- 新增商品：
- http://my-domain/products (POST)
- 修改商品：
- http://my-domain/products/17 (PUT)
- 刪除商品：
- http://my-domain/products/17 (DELETE)

1.3 管理端 URI (需要呈現表單)

- 呈現新增商品的表單：
- http://my-domain/products/add (GET)
- 呈現修改商品的表單：
- http://my-domain/products/17/edit (GET)
- 呈現刪除商品的表單：
- http://my-domain/products/17/delete (GET)

1.4 API 實作

- 商品 API：
  - 1. 列表
  - 2. 搜尋
  - 3. 單項商品

- 購物車 API：
  - 1. 讀取
  - 2.加入
  - 3.移除
  - 4.修改
  - 5.清空

2 前端 Ajax

- XMLHttpRequest
- Fetch API
- Axios（套件）
- jQuery.ajax（jQuery 附屬功能）

2.1 XMLHttpRequest

- XMLHttpRequest 屬性及方法參考： https://developer.mozilla.org/zh-TW/docs/Web/API/XMLHttpRequest
- 事件：onreadystatechange、onload、onerror、onprogress 、onabort 。
- 範例：https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest
- 必須在呼叫 open() 方法開啟請求連線之前，就註冊好事件監聽器，否則事件監聽器將不會被觸發。
- 可選擇同步或非同步。
- 上傳進度，使用XMLHttpRequest.upload 物件的事件： load、error、progress 、abort 。

ajax-xhr-01.html

```
function doAjax() {
    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function (event) {
        console.log(xhr.readyState, xhr.status);
        console.log(xhr.responseText);
        if(xhr.readyState===4 && xhr.status===200){
            info.innerHTML = xhr.responseText;
        }
    };
    // xhr.open('GET', 'data/sales01.json', true); // 非同步
    xhr.open('GET', 'data/sales01.json', false);   // 同步
    // XMLHttpRequest.open(method, url[, async[, user[, password]]])
    xhr.send();
}
```

2.2 Fetch API

- 參考 https://developer.mozilla.org/zh-TW/docs/Web/API/Fetch_API/Using_Fetch
- 使用 Promise 包裝的 API 。
- 只有網路錯誤或其他會中斷 request 的情況下，才會發生 reject 。
- 缺點：無法取得上傳或下載的進度過程。
- 在傳送少量資料時，是方便的選擇。
- 不會主動傳送 cookie，需設定 credentials 為 include。
- 放棄任務使用 AbortController 的 signal。

fetch: 使用 POST 方法

```
const url = 'https://example.com/profile';
const data = {username: 'example'};

fetch(url, {
    method: 'POST',
    body: JSON.stringify(data),
    headers: new Headers({
        'Content-Type': 'application/json'
    })
}).then(res => res.json())
    .catch(error => console.error('Error:', error))
    .then(response => console.log('Success:', response));
```

fetch: 上傳檔案及表單

```
const formData = new FormData(document.myForm);

fetch('https://example.com/profile/avatar', {
    method: 'PUT',
    body: formData
})
    .then(response => response.json())
    .catch(error => console.error('Error:', error))
    .then(response => console.log('Success:', response));
```

2.3 Axios

- Axios 工具: https://www.npmjs.com/package/axios

- Axios優點

  1. 方便使用，類似 jQuery 的 AJAX 方法
  2. Promise API 包裝
  3. 可以在後端 Node.js 中使用
  4. 體積輕量

axios: 上傳檔案及表單

```
axios.post('/try-upload3', formData, {
    headers: {
        'Content-Type': 'multipart/form-data'
    },
    onUploadProgress: function(progressEvent){
        const perc = Math.round(progressEvent.loaded/progressEvent.total*100);
        const t = new Date().toLocaleString();
        const str = `${perc} % : ${t}`;
        console.log(str);
        progress.innerHTML += str+'<br>';
    }

}).then(r=>{
    console.log(r.data);
    console.log(new Date());
});
```

3. JSON Web Token

- JSON Web Token 社群官網 https://jwt.io/
- 使用 https://www.npmjs.com/package/jsonwebtoken 套件。
- 先決條件：資料傳送過程必須在加密的環境中使用，如 HTTPS
- 優點：可在不同的用戶端環境使用，不局限於網站。
- 缺點：需存放在用戶端，由 JavaScript 發送，或其他前端技術發送。
- 過期時間 exp，也可以使用套件的設定 expiresIn。
- （使用 bcryptjs 套件加密用戶密碼）

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

3.1 JWT 編碼解碼

```
import jwt from "jsonwebtoken";
const JWT_KEY = "kjdgdk3453JYUGUYG57438"; // 自訂的密碼

// 將可以辨別用戶的資料加密為 JWT（編碼）
const data = {
  id: 26,
  account: "Shinder",
};
// 加密為 token
const token = jwt.sign(data, JWT_KEY);
console.log({token});
```

```
import jwt from "jsonwebtoken";

const JWT_KEY = "kjdgdk3453JYUGUYG57438"; // 自訂的密碼

// 接收到的 token
const token =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjYsImFjY291bn
QiOiJTaGluZGVyIiwiaWF0IjoxNzExMzMwMzE1fQ.YkdiG6utKCyhRAQHK4f
04YQ7nuxM9e0GHRNe61rrCcQ';

// 解密成為 JavaScript plain object
const payload = jwt.verify(token, JWT_KEY);
console.log(payload);
```

4. 建立 NextJS專案（pages router）

```
// next.config.mjs
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  // 可將 api server 名稱設定為環境變數
  env: {
    API_SERVER: 'http://localhost:3001'
  }
};
export default nextConfig;
```

4.1 API URL 可放在設定檔 /config/api-path.js

```
export const API_SERVER = "http://192.168.35.200:3001";
// address-book 取得列表資料
export const AB_LIST = `${API_SERVER}/address-book/api`;

// address-book 新增資料 POST
export const AB_ADD_POST = `${API_SERVER}/address-book/add`;

// address-book 刪除單筆 `${AB_DELETE}/${sid}`, 方法: DELETE
export const AB_DELETE = `${API_SERVER}/address-book`;

// address-book 讀取單筆 `${AB_ITEM}/${sid}`, 方法: GET
export const AB_ITEM = `${API_SERVER}/address-book`;

// address-book 修改單筆 `${AB_EDIT_PUT}/${sid}`, 方法: PUT
export const AB_EDIT_PUT = `${API_SERVER}/address-book/edit`;

// JWT 登入, 方法: POST, 欄位: account, password
export const JWT_LOGIN_POST = `${API_SERVER}/login-jwt`;
```

4.2 JSX 裡的類似 for 迴圈

```
{
  Array(11)
   .fill(1)
   .map(
      (v, i) => {}
   )
}
```

```
{
  [...Array(11)].map(
    (v, i) => {}
  )
}
```

4.3 CSS CDN 只能設定在 _document.js

```
import { Html, Head, Main, NextScript } from "next/document";

export default function Document() {
  return (
    <Html lang="zh">
      <Head>
        <link rel="stylesheet"
         href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" />
      </Head>
      <body>
        <Main />
        <NextScript />
        <script
          src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
          defer></script>
      </body>
    </Html>
  );
}
```

4.4 Layout 設定

```
import Head from "next/head";
import Navbar from "./navbar";
import Footer from "./footer";

export default function Layout1({title="小新的網站", children }) {
  return (
    <>
      <Head>
        <title>{title}</title>
      </Head>
      <Navbar />
      <div className="container">{children}</div>
      <Footer />
    </>
  );
}
```

使用 Layout

```
import Head from "next/head";
import Layout1 from "@/components/shared/layout1";

export default function Home() {
  return (
    <Layout1 title="首頁 - 小新的網站">
      <Head>
        <meta keyword="小新" />
      </Head>
      <h2>Home</h2>
    </Layout1>
  );
}
```

4.5 NextJS 路由架構

- Pages router 以 pages/ 為根目錄。

檔案路徑	對應的 URL
pages/login.js	http://localhost:3001/login
pages/products/index.js	http://localhost:3001/products
pages/products/[pid].js	http://localhost:3001/products/17
pages/slug1/[slug2]/[slug3].js	http://localhost:3001/slug1/aaa/bbb

```jsx
// *** pages/slug1/[slug2]/[slug3].js ***
import { useRouter } from "next/router";

export default function Slug3() {
  const router = useRouter();

  return <pre>
    { JSON.stringify(router.query, null, 4) }
  </pre>;
}
```

http://localhost:3001/slug1/aaa/bbb

```
{
    "slug2": "aaa",
    "slug3": "bbb"
}
```

4.6 SSR

- 安裝 Chrome 擴充功能「Quick source viewer」提供者為 Tomi Mickelsson
- 注意頁面來源：
  - 什麼狀況頁面由 Server-Side Rendering (SSR) 來？
  - 什麼狀況頁面由 Client-Side Rendering (CSR) 來？
- SSR 時不要使用到前端的 DOM 相關物件或功能。例如，在元件內不要直接使用 window 物件，若要使用應該放在 useEffect() 的 callback 內使用，以確保是在前端執行。

4.7 useRouter

- 使用 useRouter() 所取得的 router 物件資料為非同步，在第一次 render 時，可能沒有資料。
- router.**query** 取得 query string parameters。
- router.query 同時可取得動態路由的 slug 資料。
- query string 和 slug 使用的名稱應避免相同。
- router.**push**(url, as, options) 前端不刷新頁面跳轉。
- https://nextjs.org/docs/pages/api-reference/functions/use-router

5. JSX (參考資料)

- JSX 為 JS 延伸的語法，用來轉換成 DOM 的元素，本質上依然是 JS 不是 HTML。
- JSX 為 React.createElement() 的語法糖。
- JSX 通常視為 JavaScript XML 的簡稱，顧名思義 JSX 必須符合 XML 的規定。
- 由於 JSX 語法貼近 HTML，方便用來描述 HTML。
- JSX 的目的是呈現 HTML 頁面的內容，和呈現無關的操作不應該放在 JSX 裡。
- 會自動做 HTML 跳脫（HTML escape）。

JSX 的優點

參考來源：React Quickly, Manning Publications

1. 改善開發人員的體驗，和 **React.createElement()** 相比較。
2. 較好的錯誤訊息。
3. 快速的執行速度，**JSX** 轉換成 **JS** 語法後，執行速度依然很快速。
4. 非專業的程式設定師或視覺設計師也容易參與編輯。
5. 較少的錯誤發生，程式碼越少表示錯誤越少。

```
const el = <Card>
  <Title>Hello</Title>
  <Link href="/article/1">Article 1</Link>
</Card>;
```

```
const el = React.createElement(
  Card,
  null,
  React.createElement(Title, null, 'Hello'),
  React.createElement(Link, {href: "/article/1"}, 'Article 1'),
);
```

JSX 常用規則

1. 只能有一個根節點。
2. 有起始標籤，也要有結束標籤，若為空元素時（沒有子節點），使用空元素的表示方式。
3. 可以使用在 JSX 檔（JS 檔）的任何位置，視為一個類型的物件。
4. 自訂標籤（元件），必須大寫字母開頭如：Card、MyCard。
5. 不可以使用到 JS 關鍵字，如：for、class，應以 htmlFor 和 className 取代。
6. 標籤的屬性必須使用 Camel 的方式表達（aria- 開頭者為例外）。
7. 標籤的屬性值的大括號表示綁定值或參照，可以是 JS 的任何類型物件。
8. 屬性沒有設定值時，表示綁定 true。
9. 行內 style 屬性必須綁定 Object 類型的物件。
10. 標籤之間的大括號表示要**輸出**成 jsx 內容的運算區塊。
11. 標籤之間若換行，則生成 html 時，兩標籤中間將不會有空白。

```
// 規則 1, 2, 3
const myForm = <form>
  <img src="/img/a.png" alt="img1" />
  <hr/>
  <input type="text" name="account" />
  <br/>
  <input type="submit" name="送出"/>
</form>;
```

```jsx
// 規則 5, 6, 8
const jsx = <form>
  <div className="mb-3">
    <label htmlFor="email" className="form-label">Email address</label>
    <input type="email"
           className="form-control"
           name="email"
           aria-describedby="emailHelp"
           required
    />
  </div>
  <button type="submit" className="btn btn-primary">Submit</button>
</form>
```

```
// 規則 7
const dataObj = {name: "David", age: 25};
const jsx2 = <form onSubmit={(e) => e.preventDefault()}>
  <div className="mb-3">
    <label htmlFor="email" className="form-label">Email address</label>
    <input type="email"
           className="form-control"
           name="email"
           value={myState}
           onChange={(e) => setMyState(e.target.value)}
    />
    <MyInput {...dataObj} myValue={dataObj}/>
  </div>
  <button type="submit" className="btn btn-primary">Submit</button>
</form>
```

```
// 規則 10
const dataObj = {name: "David", age: 25};
const jsx3 = <ul>
  {Object.keys(dataObj).map(el => {
    return <li key={el}>{el}: {dataObj[el]}</li>;
  })}
</ul>
```

JSX 流程控制的轉換用法：

1. if 敘述使用 &&。
2. if/else 敘述使用三元運算子。
3. 迴圈敘述使用陣列的 map() 方法。
4. switch/case敘述則無對應的方式，可以採用 Object 物件 key-value 對應的性質。

6. Hooks 基本認識 (參考資料)

- Hooks（勾子）是因應 functional components 語法而生的。
- Components 使用 function 寫法時，基本上它就是一個 function，主要功能就是呈現（render）內容，沒有別的功能。
- 要像 class-based 元件有其它功能時，就必須借由 hooks 來賦予。
- 勾子的目的就是從 React 核心架構中勾一個功能到函式（元件）中來使用。
- Hooks 的名稱必須是以 **use** 為開頭，使用 Camel 的命名方式。
- Hooks 本身也是函式，可以組合基本的 hooks 來加以變化使用。
- Hooks 不可以放在 if/else 或者迴圈中使用。Hooks 使用時是跟著元件，並放在一個陣列中記錄，每次 render 時順序都必須一樣。
- Hooks 方便單元測試。

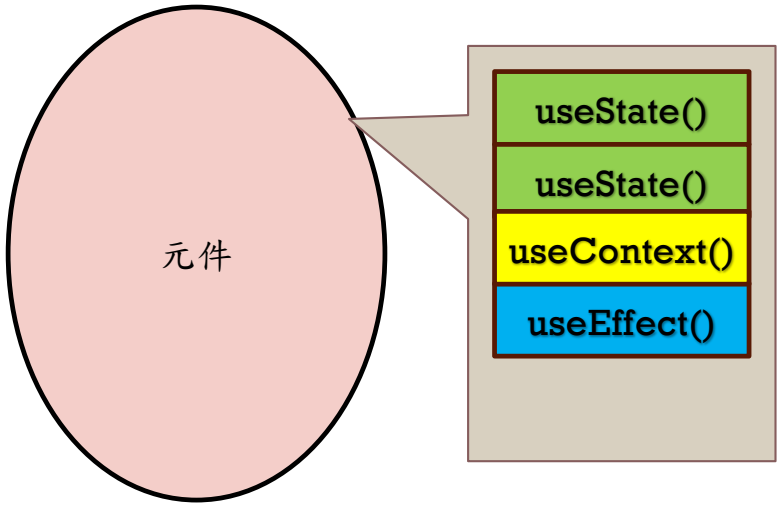
```jsx
// hooks 錯誤的使用方式
import {useState} from "react";

function MyButton() {
  if (Math.random() < .5) {
    return null;
  }
  const [val, setVal] = useState(0)
  return (
    <button>
      I'm a button {val}
    </button>
  );
}
```

Lint Error

React Hook "useState" is called conditionally. React Hooks must be called in the exact same order in every component render. Did you accidentally call a React Hook after an early return?

元件使用 hooks 示意圖



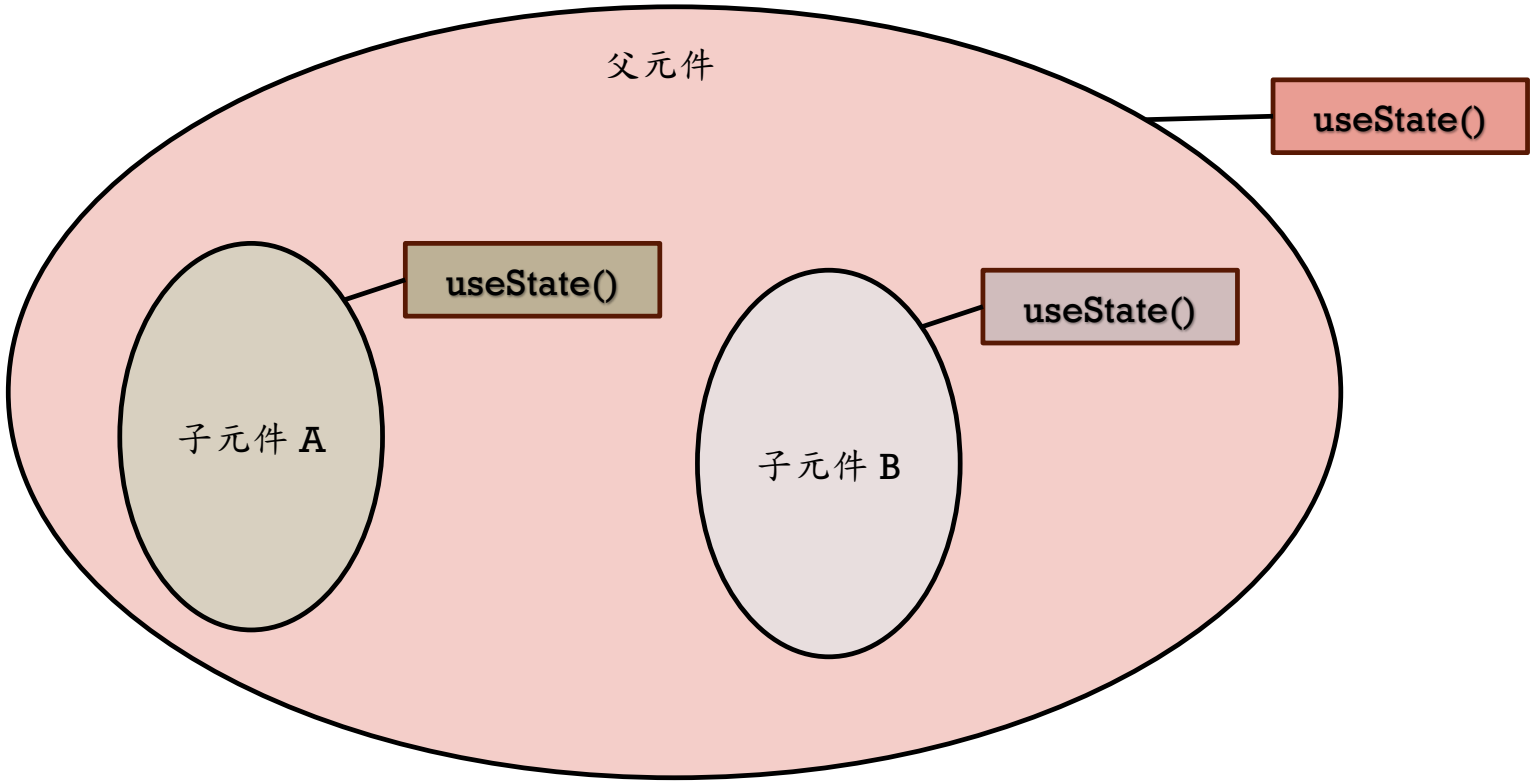
使用 hooks 的元件，相當於把使用的工具背在身上。
元件更新時（re-render），並不會影響 hooks 本身。

7. State（狀態）概念 (參考資料)

- useState() 將取得一個包含兩個參照的陣列，一個為**值**（getter），另一個為**變更函式**（setter）。
- **值**為狀態，視為「**唯讀**」，不可變更。（底層可能是使用 Proxy 實作的 getter）
- 變更狀態必須使用「**變更函式**」，而且其操作為**非同步**。意即呼叫**變更函式**後，值並沒有立即改變，而是必須等下次 render 時，**值**才能被觀察到變更。
- 狀態是跟在元件上的特殊屬性，當 state 變更時，表示元件需要更新（update），意即會觸發元件的 re-render 。
- 元件為函式時，re-render 相當於重新呼叫一次函式。函式內的所有區域變數將會被重新設定，包含存取狀態的**值**和**變更函式**。
- 狀態的**值**和**變更函式**，是由勾子處理，並不是在函式中處理。所以更新時，雖然**值**和**變更函式**被重新設定，但依然可以保有應有的狀態。

子元件 A 更新時的兩種情況

1. 子元件 A 自身的狀態改變
2. 父元件的狀態改變，以重新 render 子元件 A



模擬 useState - 1

```
<div id="app"></div>
<script>
  // *** 不能在嚴謹模式下測試此範例
  const app = document.querySelector('#app');
  const myMap = new Map(); // 儲存狀態用

  function MyApp() {
    const [val, setVal] = myUseState(5);
    // 模擬 render 行為
    app.innerHTML = `<div>
        <div>${val}</div>
        <div><button>click</button></div>
        </div>`;
    app.querySelector('button').onclick = () => {
      setVal(val + 1);
    }
  }
```

模擬 useState - 2

```
// 狀態類別
class MyState {
  constructor(init, caller) {
    this._value = init;
    this.belongTo = caller;
  }
  get value() {
    return this._value;
  }
  setValue = (v) => {
    this._value = v;
    this.belongTo(); // 重新 render
  }
}
```

模擬 useState - 3

```
 // 模擬 useState()
  function myUseState(init) {
    let item;
    if (myMap.has(myUseState.caller)) {
      item = myMap.get(myUseState.caller)
    } else {
      item = new MyState(init, myUseState.caller);
      myMap.set(myUseState.caller, item);
    }
    return [item.value, item.setValue];
  }

  MyApp();
</script>
```

** Function.prototype.caller 不建議使用，此範例純粹說明用。

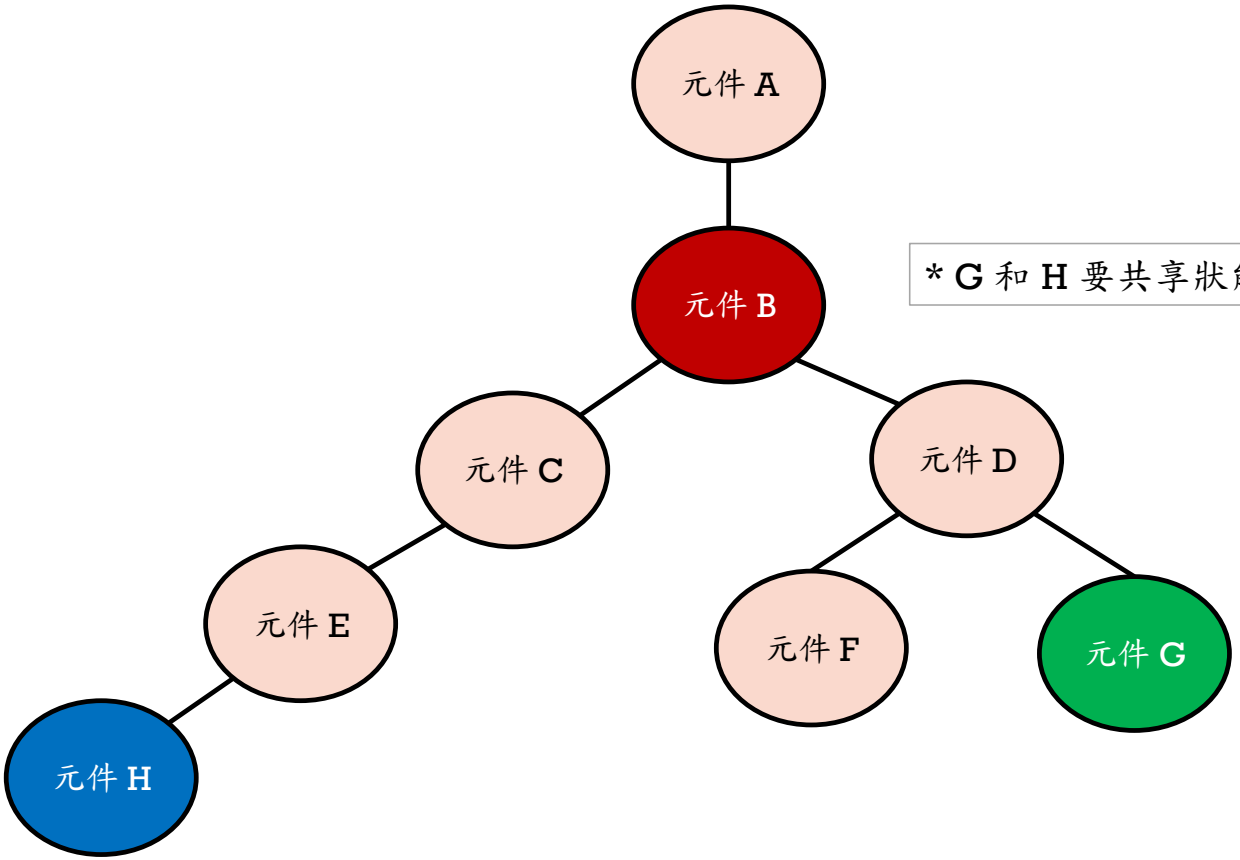
使用狀態時應注意的事項

- 父元件的狀態和子元件的狀態是各自獨立的。
- 各元件的狀態是各自獨立的。
- **單向資料流**，只有父元件傳屬性給子元件，沒有子元件傳給父元件。
- 兩元件要溝通時，應將狀態設定在「共同祖先元件」上。
- 共同祖先元件的「狀態變更函式」往下一層一層傳給下游元件，讓該元件有能力直接變更共同祖先元件的狀態，以瀑布更新的方式，讓下游元件更新。
- 要在全域共享資料時，狀態應該設定在最頂層元件（App 或 _app）。
- 越頂層的狀態，若變更太頻繁可能造成效能不佳的情況。
- Context.Provider 中的狀態不應該有太頻繁的變更。

* 父元件狀態改變時會 **render** 子元件，子元件（函式）會被呼叫。

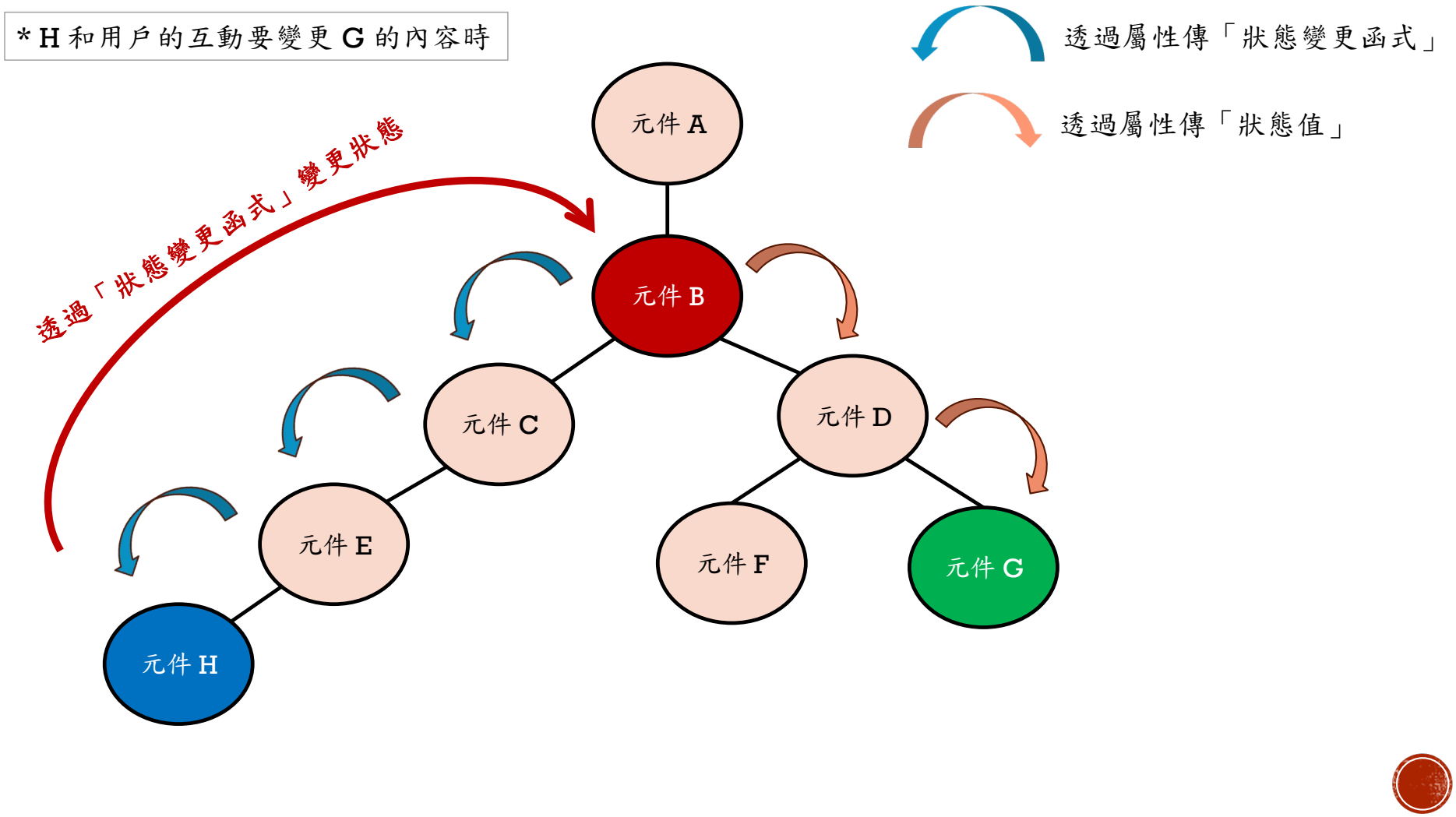
```
import {useState} from "react";
let renderCount = 0;
export default function MyApp() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>click</button>
      <ChildA />
    </div>
  );
}
```

```
function ChildA() {
  renderCount ++;
  console.log({renderCount})
  return (
    <div>
      {renderCount}
    </div>
  );
}
```



* G 和 H 要共享狀能時應該設定在 B

*H 和用戶的互動要變更 G 的內容時



8. 通訊錄 CRUD

- API Server 使用原 NodeJS 課程中的 Node/Express 伺服器

8.1 列表頁的狀態

```
// console.log(location.href); // 不要在這邊使用 BOM, DOM API

const { auth, getAuthHeader } = useAuth();

const router = useRouter();

// 狀態
const [listData, setListData] = useState({
  success: false,
  page: 0,
  rows: [],
  totalPages: 0,
});
```

8.2 取得列表資料

```
useEffect(() => {
  if (!router.isReady) return;
  const controller = new AbortController(); // 取消的控制器
  const signal = controller.signal;

  fetch(`${AB_LIST}${location.search}`, {
    signal,
    headers: { ...getAuthHeader() },
  })
    .then((r) => r.json())
    .then((result) => {
      setListData(result);
    })
    .catch((ex) => console.log({ ex })); // 用戶取消時會發生 exception
  return () => {
    controller.abort(); // 取消未完成的 ajax
  };
}, [router, auth]);
```

8.3 表格呈現內容

```
<tbody>
  {listData.rows.map((v, i) => {
    return (
      <tr key={i}>
        <td>{v.sid}</td>
        <td>{v.name}</td>
        <td>{v.email}</td>
        <td>{v.mobile}</td>
        <td>{v.birthday}</td>
        <td>{v.address}</td>
      </tr>
    );
  })}
</tbody>
```

8.4 分頁按鈕

```
<ul className=“pagination”>
  {Array(11)
    .fill(1)
    .map((v, i) => {
      const p = listData.page - 5 + i; // 從目前頁碼的前 5 個開始
      if (p < 1 || p > listData.totalPages) return null;
      const active = p === listData.page ? "active" : "";
      return (
        <li className={`page-item ${active}`} key={p}>
          <Link className="page-link" href={`?page=${p}`}>
            {p}
          </Link>
        </li>
      );
    })}
</ul>
```

8.5 有資料才呈現表格

```
<Layout1>
  {listData.success && listData.rows.length ? (
    <>
      {/* 呈現表格 */}
    </>
  ) : (
      <h5>沒有資料</h5>
  )}
</Layout1>
```

8.6 後端回應延遲模擬競爭的 Ajax

```
// ExpressJS 自訂的頂層的 middlewares 內

// ****** 測試用，模擬 2 sec 內的延遲 ***** 測試完記得註解掉
const waitMSec = Math.random() * 2000;
setTimeout(() => {
  next();
}, waitMSec);
```

** 可快速點擊分頁按鈕測試，此時可以看到 **effect cleanup** 的功能作用。

8.7 新增資料頁的表單（部份內容）

```
<h5 className="card-title">新增資料</h5>
<form name="form1" method="post" onSubmit={submitHandler}>
  <div className={"mb-3 " + (errors.name ? styles.error : "")}>
    <label htmlFor="name" className="form-label">
      姓名
    </label>
    <input
      type="text"
      className="form-control"
      id="name"
      name="name"
      value={myForm.name}
      onChange={changeHandler}
    />
    <div className="form-text">{errors.name}</div>
  </div>
```

8.8 新增資料頁的狀態

```
// 欄位檢查的規則
const schemaName = z.string().min(2, { message: "姓名需填兩個字以上" });
const schemaEmail = z.string().email({ message: "請填寫正確的電郵" });

  const router = useRouter();
  const [myForm, setMyForm] = useState({
    name: "",
    email: "",
    mobile: "",
    birthday: "",
    address: "",
  });
  // 呈現錯誤訊息的狀態
  const [errors, setErrors] = useState({
    hasErrors: false, // 判斷有沒有錯誤
    name: "",
    email: "",
  });
```

8.9 新增資料頁的輸入事件處理器

```
const changeHandler = (e) => {
  setMyForm({ ...myForm, [e.target.name]: e.target.value });
};
```

8.10 新增資料頁的表單發送前檢查

```
const submitHandler = async (e) => {
  e.preventDefault();
  let initErrors = {
    hasErrors: false, // 判斷有沒有錯誤
    name: "",
    email: "",
  };
  const r1 = schemaName.safeParse(myForm.name);
  if (!r1.success) {
    initErrors = {
      ...initErrors,
      hasErrors: true,
      name: r1.error.issues[0].message,
    };
  }
  // 略 ...
  if(initErrors.hasErrors) {
    setErrors(initErrors);
    return; // 欄位檢查時, 有錯誤的話, 就不發 AJAX
  }
```

8.11 新增資料頁的表單發送

```
const r = await fetch(AB_ADD_POST, {
  method: "POST",
  body: JSON.stringify(myForm),
  headers: {
    "Content-Type": "application/json",
  },
});
const result = await r.json();
console.log({ result });
if(result.success){
  //
  router.push(`/address-book`);
} else {
  //
  alert("資料新增發生錯誤")
}
```

8.12 列表頁中刪除資料的圖示

```
{listData.rows.map((v, i) => {
  return (
    <tr key={i}>
      <td>
        <a
          href="#"
          onClick={(e) => {
            e.preventDefault();
            removeOne(v.sid); // 移除 PK 是 v.sid 的資料
          }}
        >
          <FaRegTrashCan />
        </a>
      </td>
```

8.13 列表頁中刪除資料的函式

```
const removeOne = async (sid) => {
  const r = await fetch(`${AB_DELETE}/${sid}`, {
    method: "DELETE",
  });

  const result = await r.json();

  // URL 不變動的情況下, 改變 router 的狀態
  router.push(location.search);
};
```

8.14 列表頁中編輯資料的圖示

```
<td>
  <Link href={`/address-book/edit/${v.sid}`}>
    <FaFilePen />
  </Link>
</td>
```

8.15 編輯資料頁（從新增資料頁複製過來修改）

```
// 讀取欲編輯的資料項目
useEffect(() => {
  if (!router.isReady) return;

  fetch(`${AB_ITEM}/${router.query.sid}`)
    .then((r) => r.json())
    .then((result) => {
      if (result.success) {
        const { name, email, mobile, birthday, address } = result.data;
        setMyForm({ name, email, mobile, birthday, address });
      } else {
        router.push("/address-book");
      }
    })
    .catch((ex) => console.log({ ex }));
}, [router]);
```

8.16 表單上呈現失能的項目（提示用戶資料編號的值）

```
<div className="mb-3">
  <label htmlFor="mobile" className="form-label">
    編號
  </label>
  <input
    type="text"
    className="form-control"
    defaultValue={router.query.sid}
    disabled
  />
</div>
```

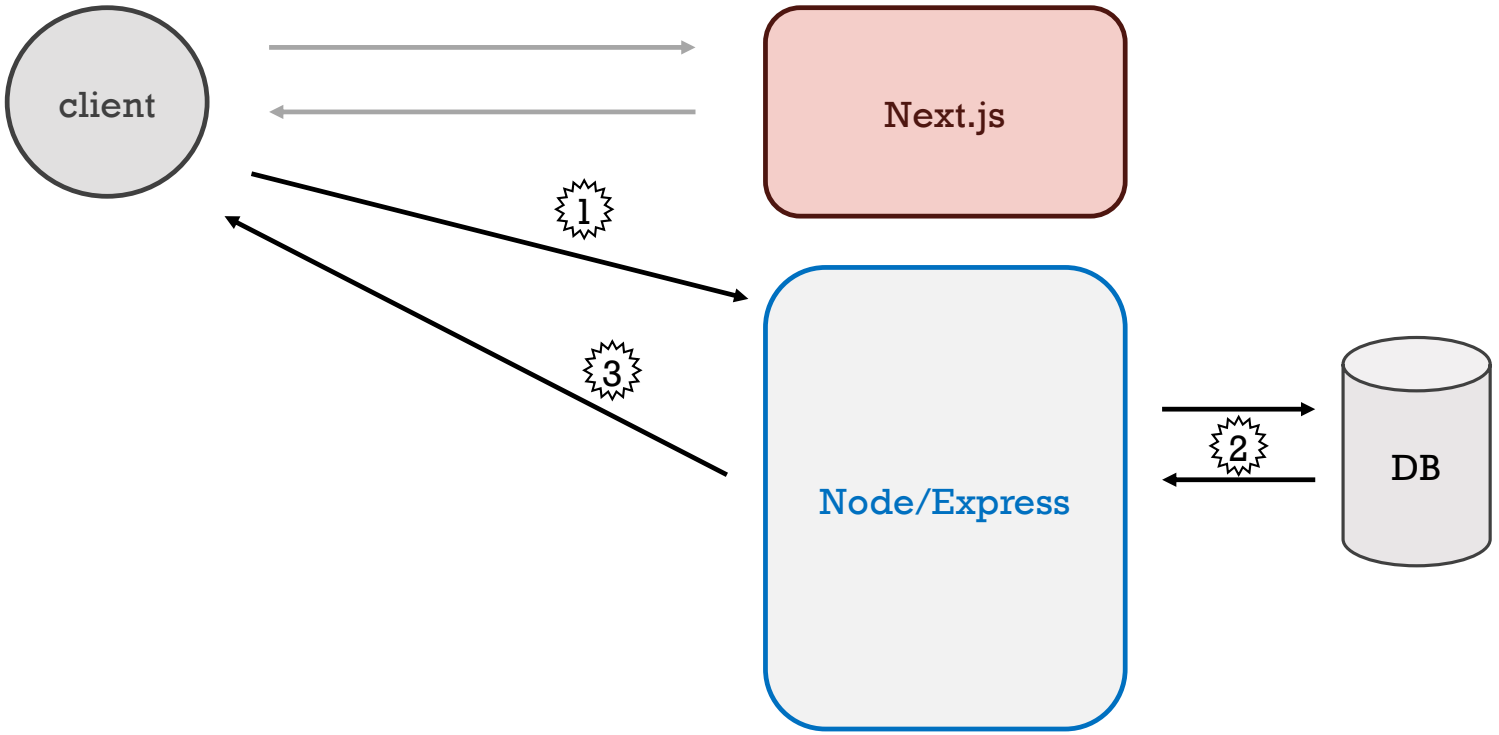
8.17 送出修改的表單

```
const r = await fetch(`${AB_EDIT_PUT}/${router.query.sid}`, {
  method: "PUT",
  body: JSON.stringify(myForm),
  headers: {
    "Content-Type": "application/json",
  },
});
const result = await r.json();
console.log({ result });
if (result.success) {
  alert("資料修改成功");
  router.back();
} else {
  alert("資料沒有修改!!!");
}
```

9. 登入使用 JWT

- 1. 用戶在前端登入頁面發送 account 和 password 資料。
- 2. 後端接收資料後比對帳號和密碼，若正確則往下一步。
- 3. 後端將用以識別用戶的資料包在 JWT 內，並送至前端。
- 4. 前端收到相關資料後存至 localStorage，並改變 AuthContextProvider 狀態，完成登入狀態。
- 5. 用戶刷頁面時（refresh），AuthContextProvider 會先判斷 localStorage 的 token 相關資料，是否為已登入的狀態。更好的作法，是讀取 token 後，再發 Ajax 確認 token 是否有效。

- ** 如果架構允許，token 存放在 http-only 的 cookie 裡，會比存放在 localStorage 裡安全，可以避免 XSS 攻擊。
- ** 在前後端分離的開發環境，token 存放在 localStorage 是比較容易實現的作法。


client
Next.js
1
3
Node/Express
2
DB

9.1 後端登入服務

```
app.post("/login-jwt", async (req, res) => {
  let { account, password } = req.body || {};
  const output = {
    success: false,
    error: "",
    code: 0,
    data: {
      id: 0,
      account: "",
      nickname: "",
      token: "",
    },
  };
  if (!account || !password) {
    output.error = "欄位資料不足";
    output.code = 400;
    return res.json(output);
  }
```

```
account = account.trim();
password = password.trim();

const sql = "SELECT * FROM members WHERE email=?";
const [rows] = await db.query(sql, [account]);

if (!rows.length) {
  // 帳號是錯的
  output.error = "帳號或密碼錯誤";
  output.code = 420;
  return res.json(output);
}
const row = rows[0];

const result = await bcrypt.compare(password, row.password);
```

```
 if (result) { // 帳號是對的, 密碼也是對的
    output.success = true;
    // 打包 JWT
    const token = jwt.sign({
        id: row.id,
        account: row.email,
      },
      process.env.JWT_SECRET
    );
    output.data = {
      id: row.id,
      account: row.email,
      nickname: row.nickname,
      token,
    };
  } else {
    output.error = "帳號或密碼錯誤";
    output.code = 450;
  }
  res.json(output);
});
```

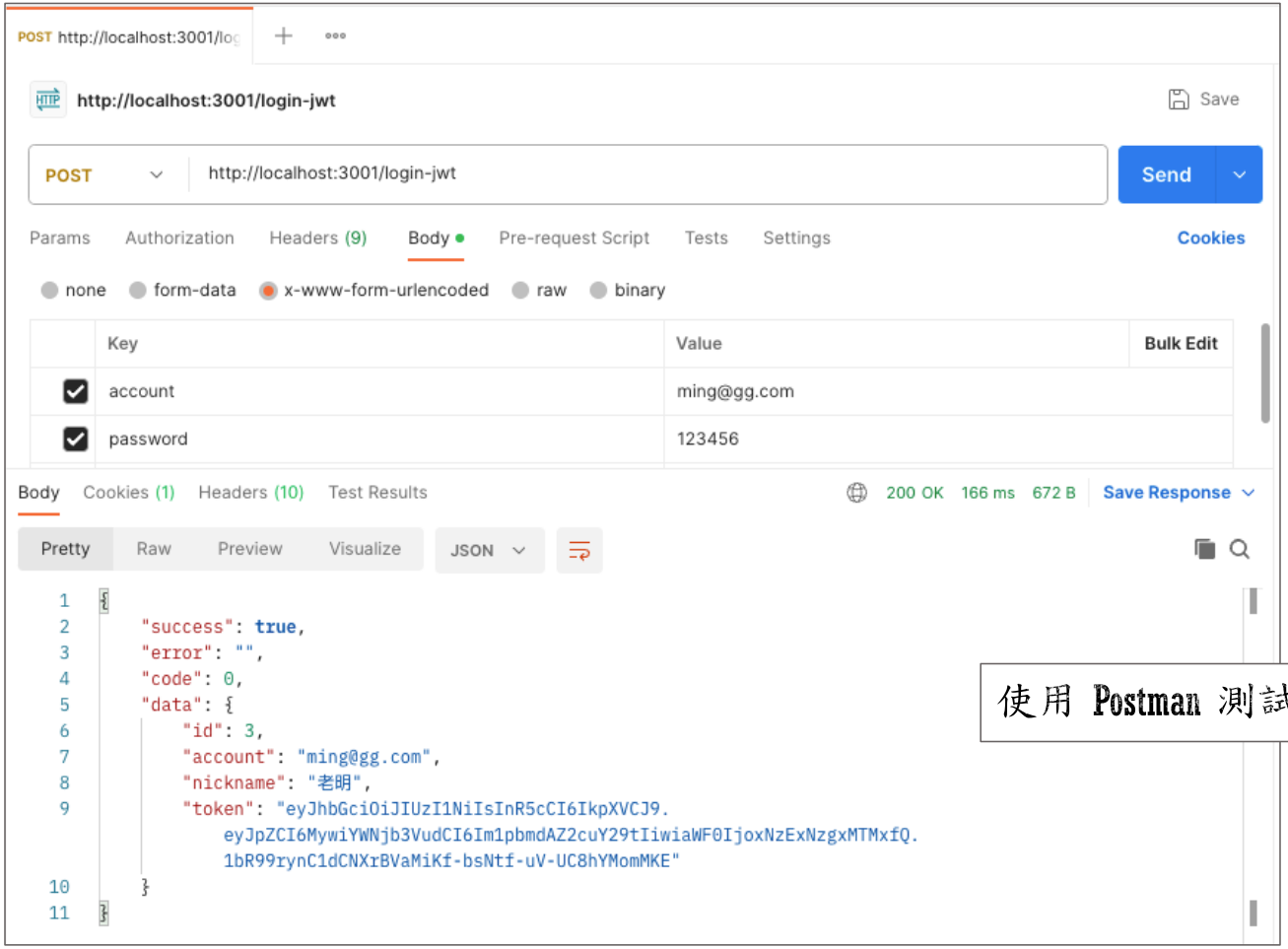
9.2 後端驗證 token

```
// 在 app top-level middleware 處理 JWT token
const auth = req.get("Authorization");
if (auth && auth.indexOf("Bearer ") === 0) {
  const token = auth.slice(7); // 去掉 "Bearer "
  try {
    // my_jwt 為我們決定的屬性名稱，勿與已存在的屬性重複
    req.my_jwt = jwt.verify(token, process.env.JWT_SECRET);
  } catch (ex) {}
}

/*
// ***** 只用在不發送 token 時，測試用戶
req.my_jwt = {
  id: 3,
  account: "ming@gg.com",
};
*/
```

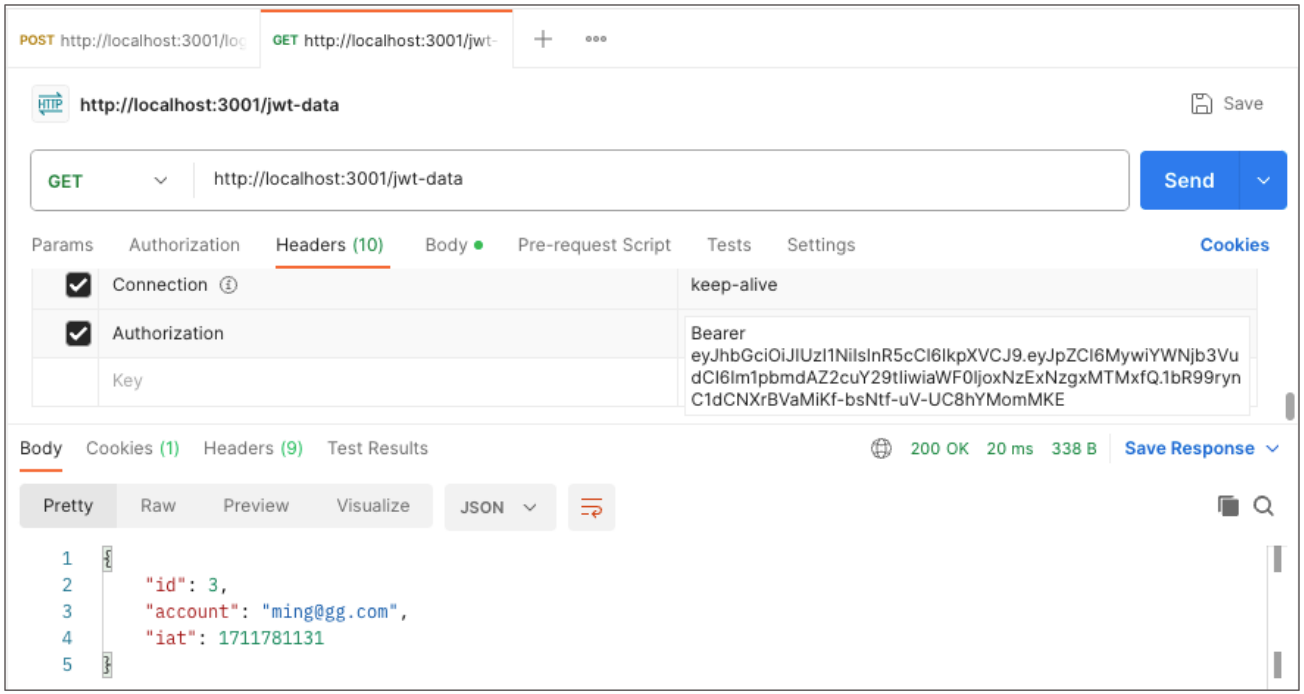
9.3 後端驗證 token 的測試路由

```
app.get("/jwt-data", async (req, res) => {
  res.json(req.my_jwt);
});
```



使用 Postman 測試登入功能

使用 Postman 測試授權要求，加入 Authorization 檔頭並放入 token



9.4 前端 AuthContext 檔案基本架構

```
// contexts/auth-context.js
const AuthContext = createContext();
// 保有登入的 "狀態": id, account, nickname, token
// login 功能
// logout 功能
// getAuthHeader() 取得包含 token 的 Authorization 檔頭

export function AuthContextProvider({ children }) {
  return (
    <AuthContext.Provider value={{auth, login, logout, getAuthHeader}}>
      {children}
    </AuthContext.Provider>
  );
}
// 自訂的 hook
export const useAuth = () => useContext(AuthContext);
export default AuthContext;
```

9.5 前端 AuthContext 檔案內全域變數

```
// 預設的狀態, 沒有登入
const emptyAuth = {
  id: 0,
  account: "",
  nickname: "",
  token: "",
};

const storageKey = "shinder-auth";
```

9.6 前端 AuthContextProvider 裡的登出功能

```
const [auth, setAuth] = useState(emptyAuth);

const logout = () => {
  localStorage.removeItem(storageKey);
  setAuth(emptyAuth);
};
```

9.7 前端 AuthContextProvider 裡的登入功能

```
const login = async (account, password) => {
  const r = await fetch(JWT_LOGIN_POST, {
    method: "POST",
    body: JSON.stringify({ account, password }),
    headers: {
      "Content-Type": "application/json",
    },
  });
  const result = await r.json();
  if (result.success) {
    // 把 token 記錄在 localStorage
    localStorage.setItem(storageKey, JSON.stringify(result.data));
    setAuth(result.data);
    return true;
  } else {
    return false;
  }
};
```

9.8 前端 getAuthHeader()

```
const getAuthHeader = () => {
  if (auth.token) {
    return { Authorization: "Bearer " + auth.token };
  } else {
    return {}
  }
};
```

9.9 前端判斷 localStorage 是否有登入的資料

```
useEffect(() => {
  const str = localStorage.getItem(storageKey);
  try {
    const data = JSON.parse(str);
    if (data) {
      setAuth(data);
    }
  } catch (ex) {}
}, []);
```

** 刷新頁面時，讀取 localStorage

9.10 前端快速登入的頁面

```
import Layout from "@/components/shared/layout1";
import { useAuth } from "@/contexts/auth-context";
export default function PageA() {
  const { auth, login } = useAuth();
  return (
    <Layout>
      <button
        onClick={() => {
          login("shin@gg.com", "234567").then((result) => {
              alert(result ? "登入成功" : "失敗");
          });
        }}
      >
        快速登入 shin@gg.com
      </button><br />
      <pre>{JSON.stringify(auth, null, 4)}</pre>
    </Layout>
  );
}
```

9.11 前端 Navbar

```
{auth.id ? (
  <>
    <li className="nav-item">
      <a className="nav-link">{auth.nickname}</a>
    </li>
    <li className="nav-item">
      <a className="nav-link" href="#"
        onClick={(e) => {
          e.preventDefault();
          logout();
        }} >登出</a>
    </li>
  </>
) : (
  <li className="nav-item">
    <Link className="nav-link" href="/login-quick">
      快速登入
    </Link>
  </li>
)}
```

10. 加入最愛使用 JWT

```sql
-- 資料表結構 `ab_likes`

CREATE TABLE `ab_likes` (
    `sid` int(11) NOT NULL,
    `member_sid` int(11) NOT NULL,
    `ab_sid` int(11) NOT NULL,
    `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `ab_likes` ADD PRIMARY KEY (`sid`);
ALTER TABLE `ab_likes` MODIFY `sid` int(11) NOT NULL AUTO_INCREMENT;
```

10.1 後端 toggle-like 功能

```
router.get("/toggle-like/:ab_sid", async (req, res) => {
  const output = {
    success: false,
    action: "", // add, remove
    error: "",
    code: 0,
  };

  if(! req.my_jwt?.id) {
    // 沒有授權
    output.code = 430;
    output.error = "沒有授權";
    return res.json(output);
  }
  const member_sid = req.my_jwt.id; // 從 JWT 來的
```

```
// 先確認有沒有這個項目
const sql1 = "SELECT sid FROM address_book WHERE sid=? ";
const [rows1] = await db.query(sql1, [req.params.ab_sid]);
if (!rows1.length) {
  output.code = 401;
  output.error = "沒有這個朋友";
  return res.json(output);
}

const sql2 = "SELECT * FROM `ab_likes` WHERE `member_sid`=? AND `ab_sid`=?";
const [rows2] = await db.query(sql2, [member_sid, req.params.ab_sid]);
```

```
if (rows2.length) {
  // 如果已經有這個項目, 就移除
  const sql3 = ` DELETE FROM ab_likes WHERE sid=${rows2[0].sid} `;
  const [result] = await db.query(sql3);
  if(result.affectedRows) {
    output.success = true;
    output.action = "remove";
  } else {
    // 萬一沒有刪除
    output.code = 410;
    output.error = "無法移除";
    return res.json(output);
  }
}
```

```
  else {
    // 如果沒有這個項目, 就加入
    const sql4 = "INSERT INTO `ab_likes` (`member_sid`, `ab_sid`) VALUES (?, ?)";
    const [result] = await db.query(sql4, [member_sid, req.params.ab_sid]);
    if(result.affectedRows) {
      output.success = true;
      output.action = "add";
    } else {
      // 萬一沒有新增成功
      output.code = 420;
      output.error = "無法加入";
      return res.json(output);
    }
  }
  res.json(output);
});
```

10.2 後端取得列表時的 SQL

```
-- 使用 SQL 子查詢
-- 子查詢結果必須使用別名
SELECT ab.*, li.sid like_sid
    FROM address_book ab
    LEFT JOIN (
        SELECT * FROM ab_likes WHERE member_sid=7
    ) li ON ab.sid=li.ab_sid
    ORDER BY ab.sid DESC
    LIMIT 30
```

10.3 後端 getListData()

```
const getListData = async (req) => {
  const member_sid = req.my_jwt?.id || 0; // 授權的用戶
  let page = +req.query.page || 1;
  let keyword = req.query.keyword || "";

  let where = " WHERE 1 ";
  if (keyword) {
    const keywordEsc = db.escape("%" + keyword + "%");
    where += ` AND (
      ab.\`name\` LIKE ${keywordEsc}
      OR
      ab.\`mobile\` LIKE ${keywordEsc}
      ) `;
  }
```

```
const dateFormat = "YYYY-MM-DD";
// 篩選: 起始生日的日期
let date_begin = req.query.date_begin || "";
if (dayjs(date_begin, dateFormat, true).isValid()) {
  where += ` AND ab.birthday >= '${date_begin}' `;
}

// 篩選: 結束生日的日期
let date_end = req.query.date_end || "";
if (dayjs(date_end, dateFormat, true).isValid()) {
  where += ` AND ab.birthday <= '${date_end}' `;
}

if (page < 1) {
  return { success: false, redirect: "?page=1" };
}
const perPage = 25;
const t_sql = `SELECT COUNT(1) totalRows FROM address_book ab ${where}`;
const [[{ totalRows }]] = await db.query(t_sql);
```

```
let rows = []; // 預設值
let totalPages = 0;
if (totalRows) {
  totalPages = Math.ceil(totalRows / perPage);
  if (page > totalPages) {
    // 包含其他的參數
    const newQuery = { ...req.query, page: totalPages };
    const qs = new URLSearchParams(newQuery).toString();
    return { success: false, redirect: `?` + qs };
  }
  const sql = `SELECT ab.*, li.sid like_sid
      FROM address_book ab
      LEFT JOIN (
        SELECT * FROM ab_likes WHERE member_sid=${member_sid}
      ) li ON ab.sid=li.ab_sid
      ${where}
      ORDER BY ab.sid DESC
      LIMIT ${(page - 1) * perPage}, ${perPage}`;
```

```
  [rows] = await db.query(sql);

    rows.forEach((r) => {
      if (r.birthday) {
        r.birthday = dayjs(r.birthday).format("YYYY-MM-DD");
      }
    });
  }
  return {
    success: true,
    totalRows,
    totalPages,
    page,
    perPage,
    rows,
    query: req.query,
    member_sid
  };
};
```

10.4 前端取得列表資料時要發送 token

```
useEffect(() => {
  if (!router.isReady) return;
  const controller = new AbortController(); // 取消的控制器
  const signal = controller.signal;

  fetch(`${AB_LIST}${location.search}`, {
    signal,
    headers: { ...getAuthHeader() },
  })
    .then((r) => r.json())
    .then((result) => {
      setListData(result);
    })
    .catch((ex) => console.log({ ex })); // 用戶取消時會發生 exception
  return () => {
    controller.abort(); // 取消未完成的 ajax
  };
}, [router, getAuthHeader]);
```

10.5 前端愛心圖示呈現

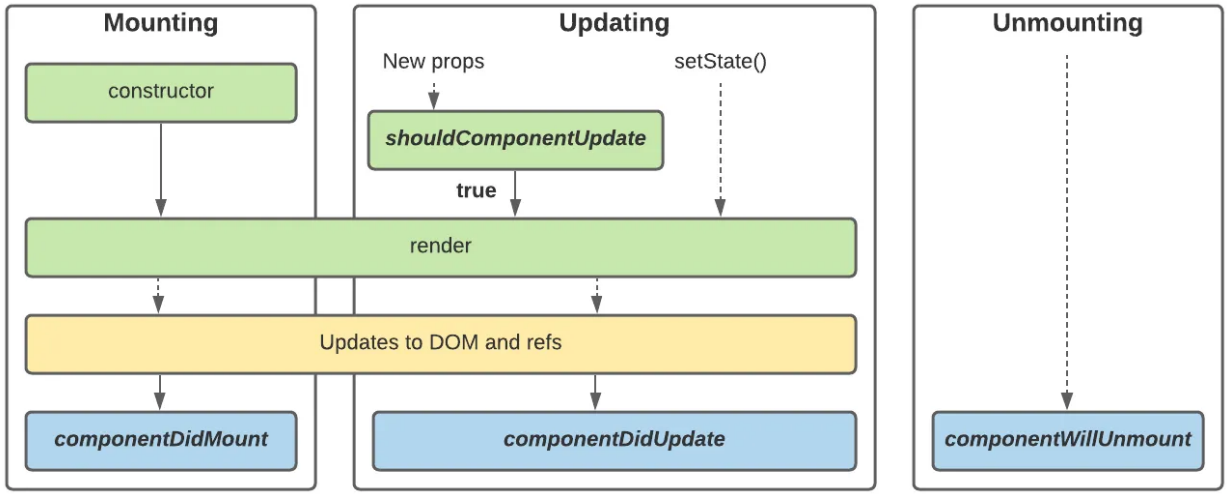
```
<td>
  {v.sid} {` `}
  <span
    onClick={(e) => {
      toggleLike(v.sid);
    }}
  >
    {v.like_sid ? <FaHeart /> : <FaRegHeart />}
  </span>
</td>
```

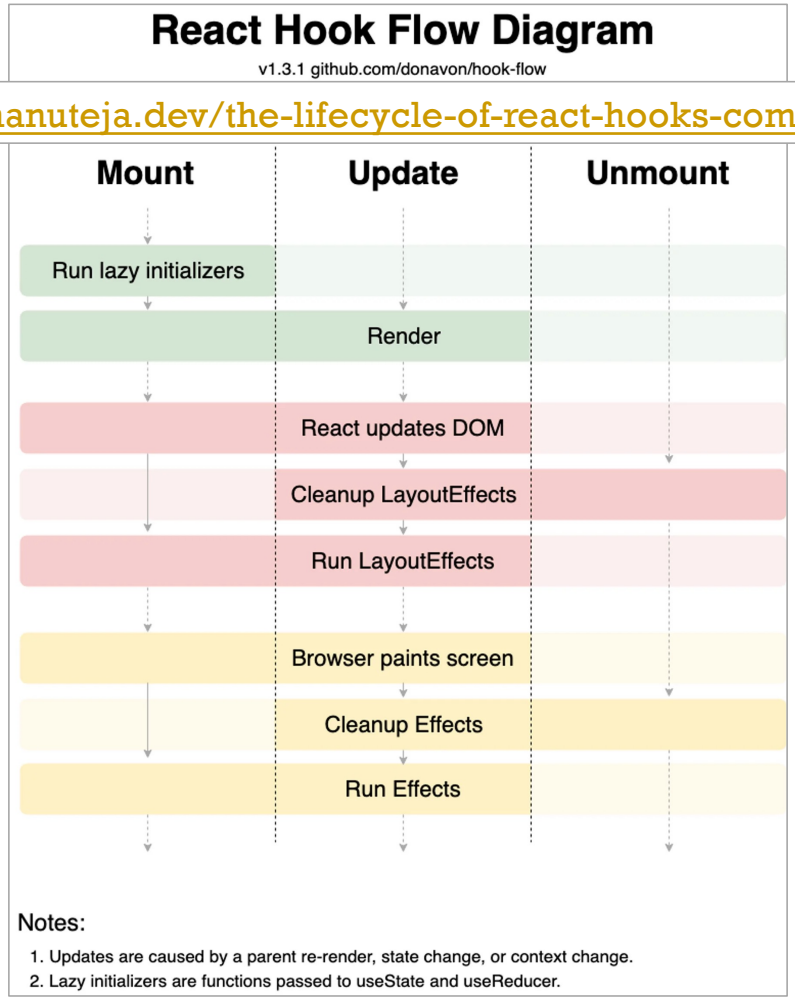
10.6 前端 toggleLike 功能

```
const toggleLike = async (sid) => {
  const r = await fetch(`${AB_LIKE}/${sid}`, {
    headers: { ...getAuthHeader() },
  });
  const result = await r.json();

  if (result.success) {
    const newListData = { ...listData };
    newListData.rows = listData.rows.map((item) => {
      if (item.sid === sid) {
        return { ...item, like_sid: !item.like_sid };
      } else {
        return { ...item };
      }
    });
    setListData(newListData);
  }
};
```

11. 元件生命週期（參考資料）





https://bhanuteja.dev/the-lifecycle-of-react-hooks-component

```
// *** pages/life-cycle.js
import {useState} from "react";
import LifeA from "@/components/LifeA";
export default function LifeCyCle() {
  const [show, setShow] = useState(false);
  return (
    <>
      <div className="container">
        <button
          onClick={() => {
            setShow((old) => !old);
          }}
        >
          toggle
        </button>
        {show && <LifeA/>}
      </div>
    </>
  );
}
```

```jsx
// *** components/LifeA.js 1/2
import {useEffect, useState} from "react";

export default function LifeA() {
  const [val, setVal] = useState(10);
  console.log("render----1");
  useEffect(() => {
    console.log("掛載之後");
    let n = 1;
    const interval_id = setInterval(() => {
      console.log(n++);
    }, 1000)
    return () => {
      console.log("將要卸載");
      clearInterval(interval_id)
    }
  }, []);
```

```
// *** components/LifeA.js 2/2

  useEffect(() => {
    console.log("已更新");
    return () => {
      console.log("將要更新");
    }
  }, [val]);
  return (
    <div>
      LifeA
      {console.log("render----2")}
    </div>
  );
}
```

Thank You