JavaScript 程式設計新手村

單元20 - Ajax / JSON / Promise / Fetch API

@kdchang

Outline

1. Ajax 非同步處理基礎概念
2. JSON
3. Promise
4. Fetch API

Ajax 非同步處理基礎概念

HTTP 協定

HTTP 是一個用戶端終端（用戶）和伺服器端（網站）請求和應答的標準（TCP），常見請求方法：`GET`、`POST`、`DELETE`、`PUT` 等

```
GET / HTTP/1.1
Host: www.google.com
```

```
HTTP/1.1 200 OK
Content-Length: 3059
Server: GWS/2.0
Date: Sat, 11 Jan 2003 02:44:04 GMT
Content-Type: text/html
Cache-control: private
Set-Cookie: PREF=ID=73d4aef52e57bae9:TM=1042253044:LM=104225304
X9j; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.goo
Connection: keep-alive
```

HTTP 協定狀態碼

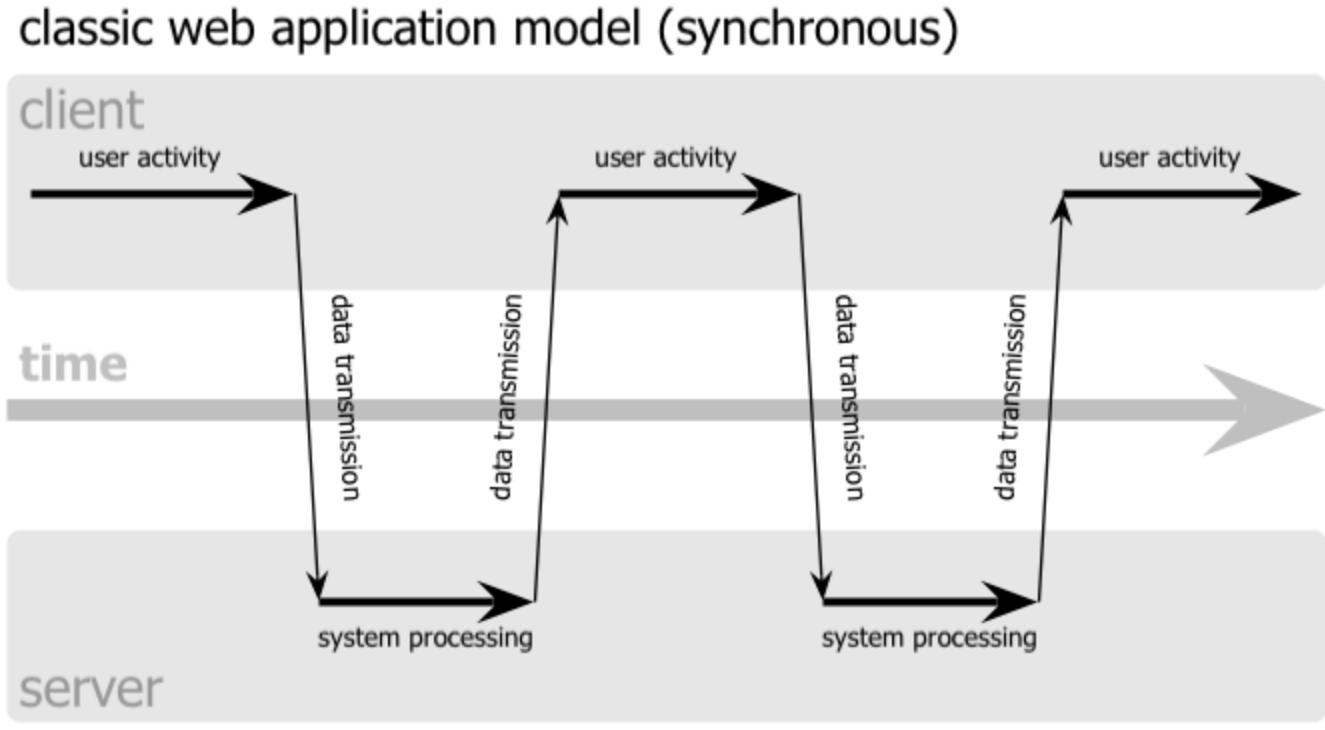
1xx訊息——請求已被伺服器接收，繼續處理
2xx成功——請求已成功被伺服器接收、理解、並接受
3xx重新導向——需要後續操作才能完成這一請求
4xx請求錯誤——請求含有詞法錯誤或者無法被執行
5xx伺服器錯誤——伺服器在處理某個正確請求時發生錯誤

什麼是同步/非同步？

- 非同步係指程式不會因為上一個函數尚未執行完（例如：回傳結果）就卡住，會往下執行下一個任務
- 同步就是要等到上一個函數任務執行完才能執行下一個，是依序執行

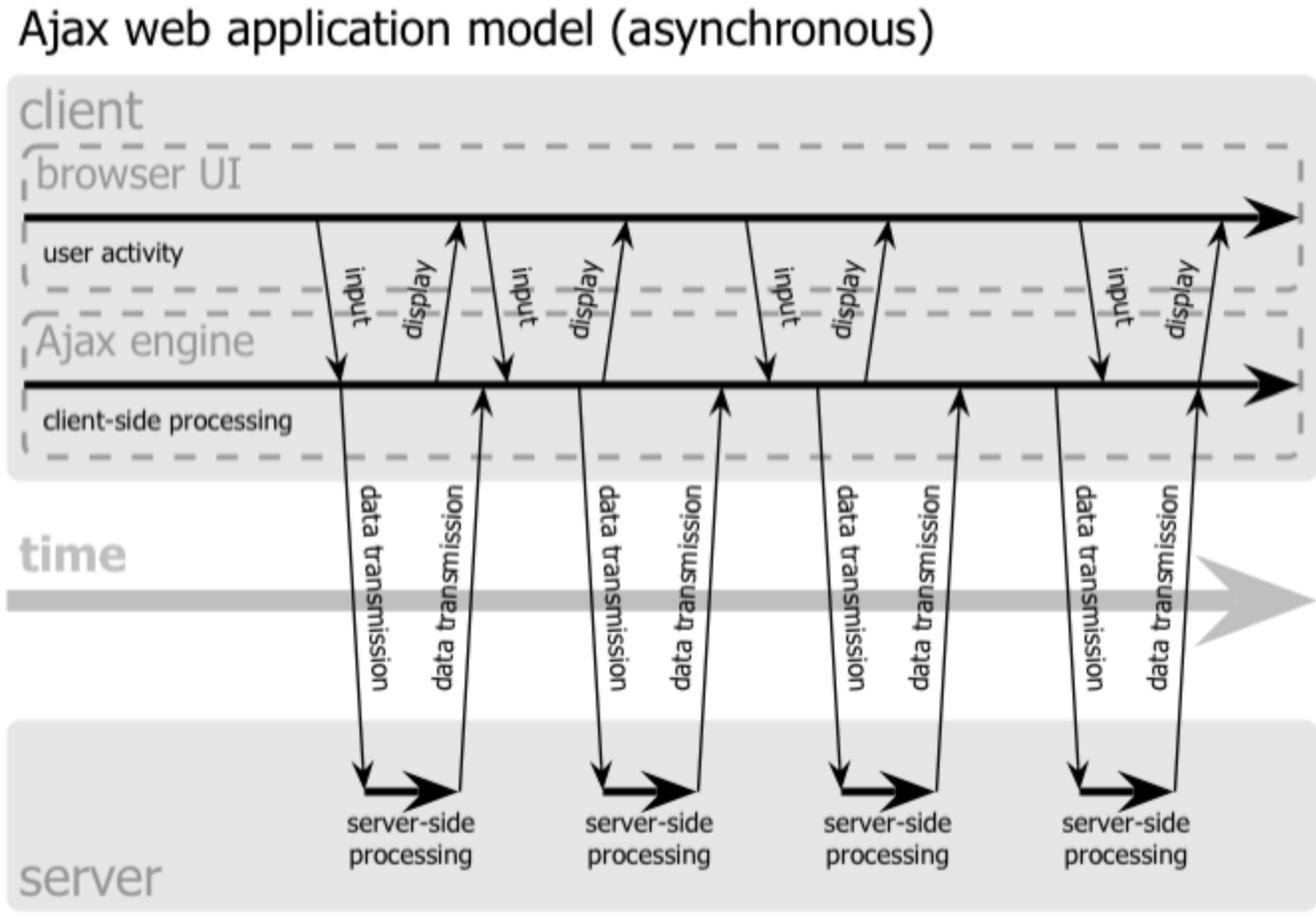
由於 `DOM 事件處理` 和 `Ajax 呼叫` 是非同步處理，所以大部分人會為 JavaScript 貼上非同步程式設計的標籤

同步 vs. 非同步



依序執行，等到上一個函數任務執行完才能執行下一個

同步 vs. 非同步



不會因為上一個函數尚未執行完（例如：回傳結果）就卡住，會往下執行下一個任務

什麼是 Ajax？

- Ajax 全名：`Asynchronous Javascript And XML`，指的是一套綜合了多項技術的瀏覽器端網頁開發技術
- 雖然 Ajax 中使用 XML 為名，不過 Ajax 不是指一種單一的技術。現在許多應用都使用更輕量的 JSON 進行資料傳輸
- 可以完成不刷頁局部更新應用，使用者體驗較好。不過要小心回調地獄（callback hell）

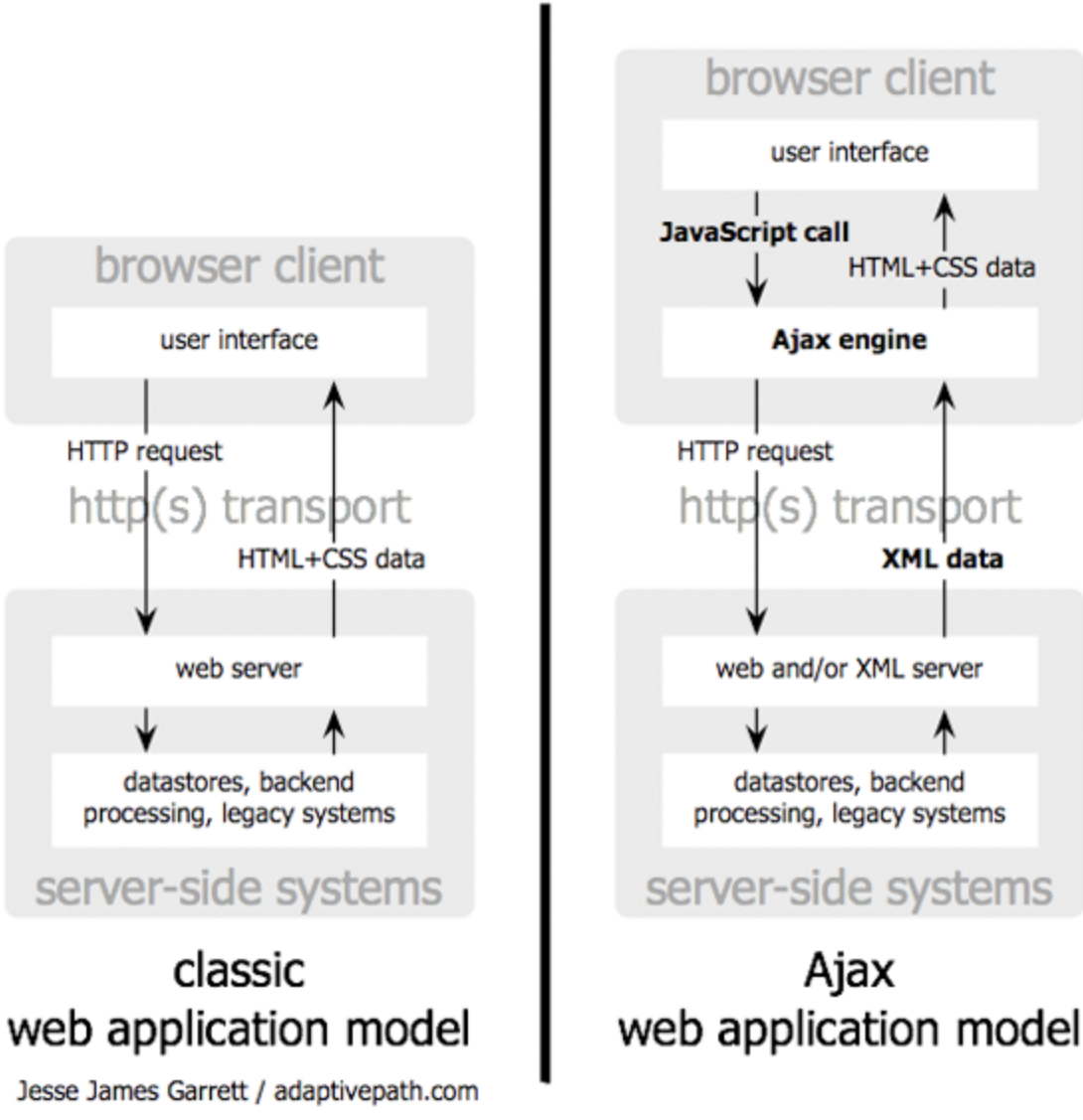
什麼是 XML？

可延伸標記式語言（英語：Extensible Markup Language，簡稱：XML），是一種標記式語言。標記指電腦所能理解的資訊符號，通過此種標記，電腦之間可以處理包含各種資訊的文章等

XML 定義結構、儲存資訊、傳送資訊。下例為郭靜傳送給黃茸的短信，儲存為XML

```
<?xml version="1.0"?>
<短信>
 <收件人>黃茸</收件人>
 <發件人>郭靜</發件人>
 <主題>路上盤纏夠嗎？</主題>
 <具體內容>早啊，飯吃了沒？靈蛇島路途遙遠，千萬別餓上了 </具體內容>
</短信>
```

傳統 http request 和 Ajax



傳統 http request 和 Ajax

- 傳統上我們會使用 `<form>` 表單和後端程式作互動，然而每次提交表單送出請求給伺服器，伺服器接收並處理傳來的表單，然後送回一個新的網頁
- 使用 Ajax 應用可以僅向伺服器發送並取回必須的數據，並在客戶端採用JavaScript 處理來自伺服器的回應，不僅減少伺服器負擔也加快反應速度

簡易 Ajax 實作

```
// 若需要支援跨瀏覽器，還需要額外檢驗
if (typeof XMLHttpRequest != 'undefined') {
    // 一般使用 XMLHttpRequest 物件
    const xhr = new XMLHttpRequest();
    const REQUEST_URL = 'http://163.29.157.32:8080/dataset/6a3e

    // 監聽是否完成
    xhr.onreadystatechange = function() {
        if (xhr.readyState === XMLHttpRequest.DONE) {
            console.log(xhr.responseText);
        }
    }

    xhr.open('GET', REQUEST_URL);
    xhr.send();
}
```

JSON

JSON 基礎概念

- JSON（JavaScript Object Notation）是一種由Douglas Crockford 構想設計、輕量級的資料交換語言，以文字為基礎，且易於讓人閱讀
- JSON 雖然起於 JavaScript，但資料格式與語言無關，目前很多程式語言都支援 JSON 格式資料的生成和解析
- JSON 的官方 MIME 類型是 `application/json`，其副檔名是 `.json`
- 基本格式 `{ "key": "value" }`、`{ "key": ["value1", "value2"] }`

JSON 長這樣

```
{
      "name": "John Smith",
      "address":
      {
          "streetAddress": "21 2nd Street",
          "city": "New York",
          "state": "NY",
          "postalCode": "10021"
      },
      "phoneNumber":
      [
          {
            "type": "home",
            "number": "212 555-1234"
          },
          {
            "type": "fax",
            "number": "646 555-4567"
          }
      ]
 }
```

Promise

回調地獄

```
getData(function(x){
    getMoreData(x, function(y){
        getMoreData(y, function(z){
            ...
        });
    });
});
```

Promise 基礎概念

過去我們在執行非同步的處理時很容易就掉入了回調地獄過多嵌套函數的窘境，而 ES6 提供原生的 Promise 的出現很大程度就能解決這個問題

```
// executor 是一個帶有 resolve, reject 回調函數參數的函數
new Promise(executor);
new Promise(function(resolve, reject) { ... });
```

- pending：初始狀態，不是 fulfilled 也不是 rejected
- fulfilled：成功的操作
- rejected：失敗的操作

Promise in jQuery

過去我們在使用 jQuery 時不知不覺就使用到了 Promise 的概念 ，如今 ES6 已經內建了原生 Promise 可以使用

```
$.getJSON('http://example.com/api/').
    done(function(data) {
        // ...
    }).

    fail(function() {
        // ...
    }).

    always(function() {
        // ...
    });
```

有了 Promise 後，回調函數擁有清晰次序，可以保證了完成後執行 done()，失敗執行 fail()，不管如何總是執行 always()

Promise 基礎

建立 Promise：

```
const comments = [
  { title: 'Comment 1', content: 'content1'},
  { title: 'Comment 2', content: ' content2'},
];

function getComments() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      resolve(comments);
    } , 1000);
  });
}
```

Promise 基礎

使用 Promise：

```
// 狀態 fulfilled 時做 then，可鏈結
function printCommentsToConsole() {
  getComments()
    .then(comments => console.log(comments))
    .catch(err => console.log(err));
}

printCommentsToConsole();
```

Fetch API

Fetch API 基本概念

Fetch API 讓原本的 XMLHttpRequest 更簡單（簡便 Options Object）和可維護（其回傳的是 ES6 `Promise` 物件）

Fetch API 簡易使用流程：
fetch() 主要使用兩個參數，請求地址和選項。請求完成後 return response（`Promise` 物件）

Fetch API 基本概念

```
const myHeaders = new Headers();

if (self.fetch) {
    // run my fetch request here
    const myInit = { method: 'GET',
                headers: myHeaders,
                mode: 'cors',
                cache: 'default' };
    const REQUEST_URL = 'http://163.29.157.32:8080/dataset/6a3e

    fetch(REQUEST_URL, myInit)
      .then(function(response) {
          return response.json();
      })
      .then(function(data) {
          console.log(data);
      });
} else {
    // do something with XMLHttpRequest ?
}
```

補充：Async/Await

ES7（ES2016）更提供了 Async/Await 來處理非同步問題

```
async function printPostsToConsole() {
  const posts = await getPosts();
  console.log(posts);
};

printPostsToConsole();
```

延伸閱讀：告別 JavaScript 的 Promise！迎接 Async/Await 的到來

總結

在這個章節中我們學會了：

1. Ajax 非同步處理基礎概念
2. JSON
3. Promise
4. Fetch API

參考圖片

1. adaptivepath
2. adaptivepath ajax