用 Vite 開啟你的 Extension

套件用的好，讓你沒煩惱 | ~~套件，讓你與昨天說再見~~

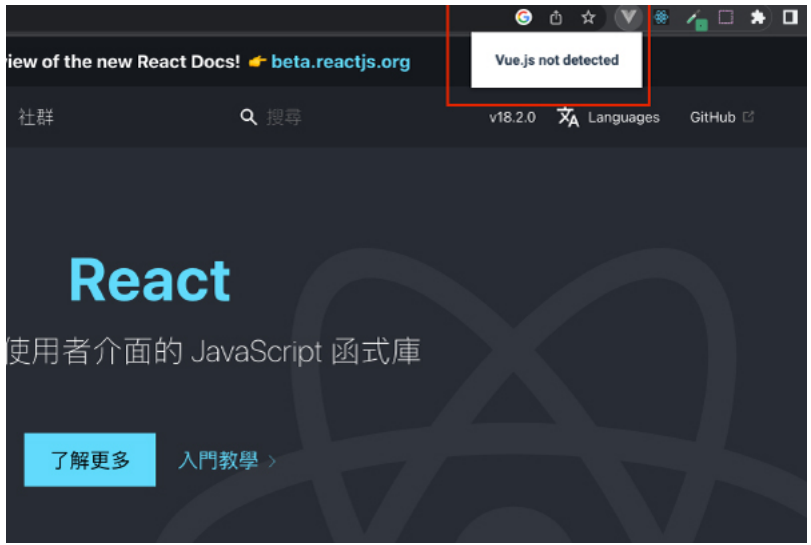
LearnWeb Taiwan x WINNIE WU

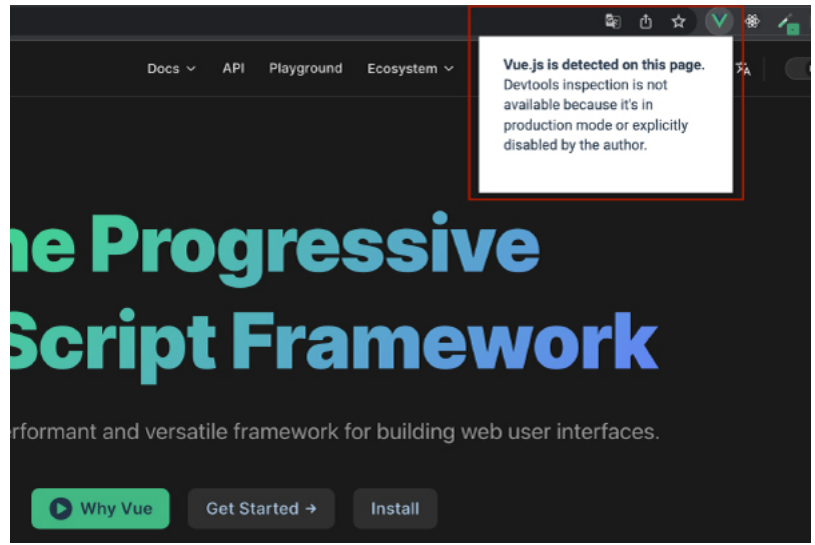
什麼是 Chrome Extension ?

Chrome Extension 為 Chrome 瀏覽器所提供的擴充工具，它能夠保持原始網站運作的狀態下，讓使用者可以透過瀏覽器所提供的權限來額外自定義附加功能，**『改變特定網站的外觀或者增加新的功能』**，主要使用 **『HTML、CSS、JavaScript』** 來開發。

以 Vue DevTool 為例

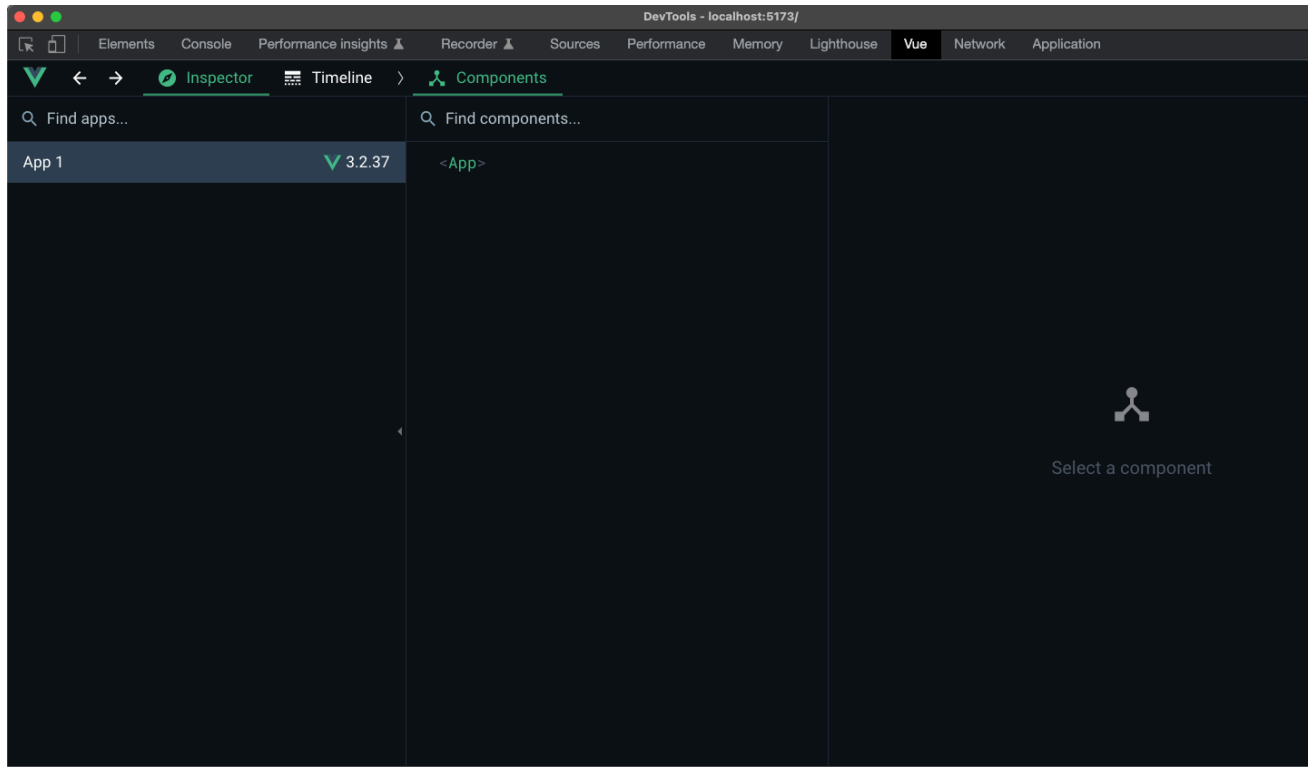
以 Vue DevTool 為例





Vue DevTool 會偵測當時網站有無使用 Vue 技術，如果該網站未使用到Vue開發，圖示呈現灰色狀態來表示
反之，有使用到則呈現原本的顏色。

以 Vue DevTool 為例



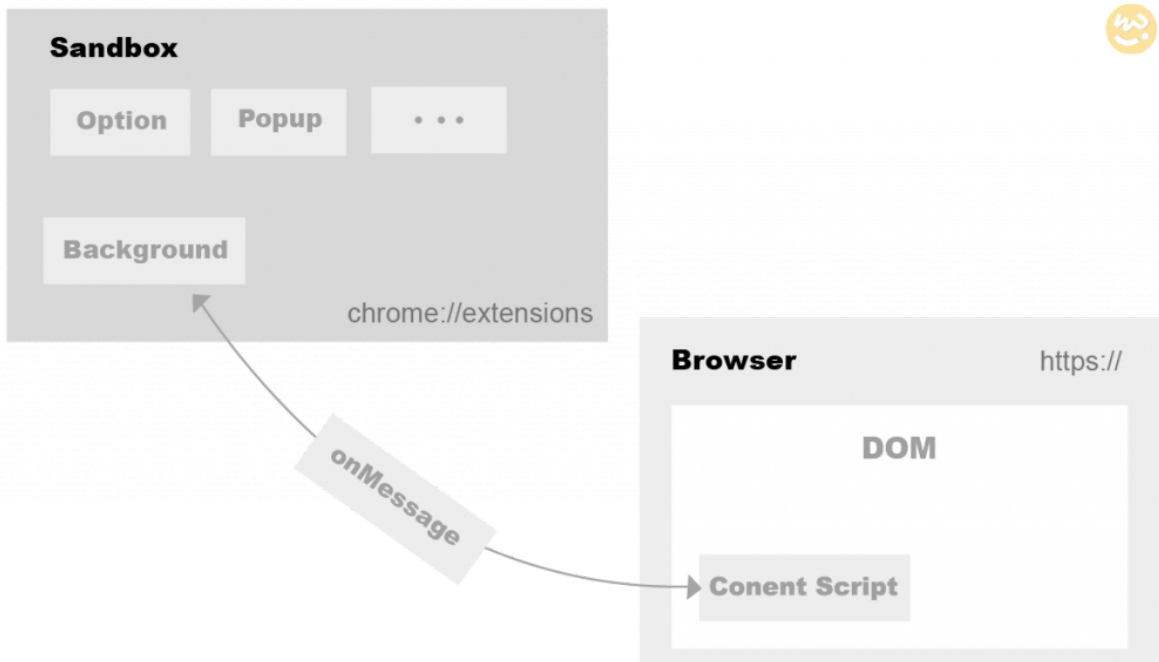
組成 與 運行環境

Chrome Extension 的 常見組成

在 Extension 中 主要是由一個 Manifest 組成，而其他的權限功能 Content Script 、 Action 及 Option 等則視需要可在 Manifest.json 自定義

運行環境

運行環境可以分為兩個為完全獨立空間，分別是 **Extension端 及 瀏覽器端**。因為是兩個互不影響，不能直接操作對方的相關方法，唯一溝通的方式只有透過 **onMessage 事件監聽方法** 來進行雙方資料溝通。



關於 Manifest

Manifest 為 套件主要程式設定檔，如基本名稱、版本、描述、各項權限都在此檔設定，所有功能的執行進入點

Manifest 必填及建議項目

必填項目

```
{
  "name": "Extension DEMO ",
  "version": "1",
  "manifest_version": 3
  //...略
}
```

建議項目

```
{
  "description": "DEMO Description",  // 建議
  "icons": {...},  // 建議
}
```

Manifest 選填項目

- permissions

  當 Extension 需要對瀏覽器 進行附加功能時，有可能會需要使用到 Browser 的功能(像是Cookie、Storage)協助，此時就需透過在 Permission 中進行使用宣告，才能進行 權限的使用。

```
{
    permissions": [
    "storage",
    "activeTab", //目前視窗頁
    "tabs",
    "cookies" // 網站的cookie
  ],
}
```

* 提醒: 建議沒有使用到的權限記得就不要宣告，要不然會導致最後 google 送審 時會未能通過

Manifest 選填項目

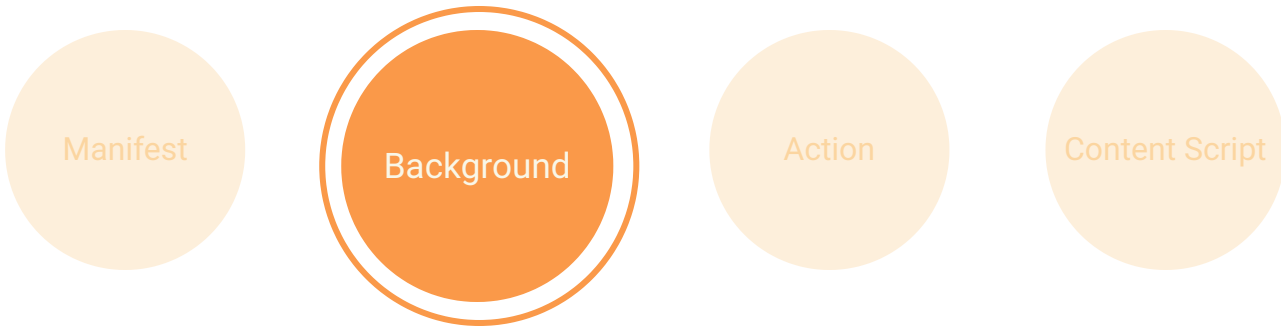
- host_permissions

  主要使用時機為需在 Extension 中 Cross-origin 請求對應API資料時，需在 host_permissions 內定義相關網域的網址。

```
{
  "host_permissions": [
    "https://www.google.com/", // 指定特定連結
    "https://www.google.com/*", // 指定在特定網域下所有連結
    "<all_urls>" //所有都可以使用
  ]
}
```

更多 Manifest 相關功能設定 請見 -> 官方文件

Chrome Extension 的 常見組成 Background



Background 為 Extension 中的 長時間執行腳本事件，通常 Extension 的相關的邏輯功能都會此進行定義，而其常見的功能主要為 監聽各項事件的發生、使『瀏覽器端』與『 Extension端 』進行資料溝通 及 跨域請求API資料 等...

Chrome Extension 的 常見組成 Background

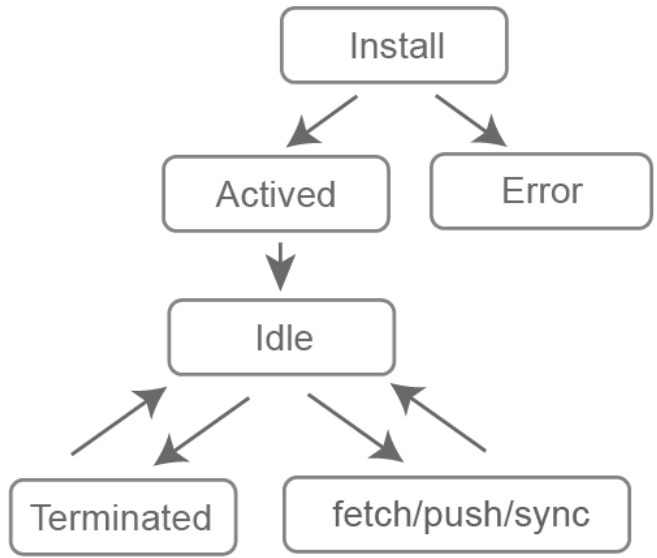
在 Manifest 3 中， Background 採用 Service Worker 來運行，如果需使用 background 相關功能，首先需在 Manifest 內 background 欄位中指定檔案路徑，使用 ES Module 只要 type 為 module 即可使用。

```
// manifest.json
{
  ...
  "background": {
    "service_worker": "background.js",
    "type": "module" //optional
  }
  ...
}
```

什麼是 Service Worker ?

為瀏覽器與網路之間的一種網路請求代理(Proxy)，為PWA核心技術之一，主要特性為 無法存取DOM元素，需透過 postMessage 方法與其他溝通 、 主要透過 事件監聽 觸發每個生命週期 及 需在 HTTPS 下執行，進行攔截請求

Service Worker 生命週期



關於 常見 事件 及 Chrome API

Background 常見監聽事件

『chrome.runtime.onInstalled』 為 Extension 第一次被安裝時，執行事件的觸發，永遠只會觸發一次

```
// background.js
const setStorage = (obj) => {
  chrome.storage.sync.set(obj, () => {});
};

chrome.runtime.onInstalled.addListener(async () => {
  console.log("Extension is Installed");
  //初始化資料
  await setStorage({
    isClear: false,
  });
});
```

Tab 頁籤 API及監聽事件

使用chrome.tabs API 可以與Browser的特定頁籤進行互動。像是創建、修改和重新排列頁籤順序等...

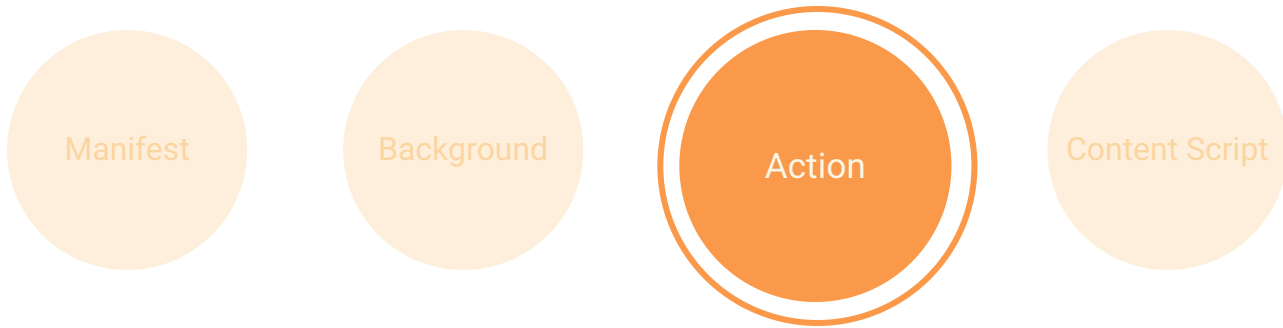
```
// 創建新Tab
chrome.tabs.create({ url: 'page.html' });

// 取出目前Tab
const getCurrentTab =()=>{
let queryOptions = { active:true,
  lastFocusedWindow: true };
   let [tab] = await chrome.tabs.query(queryOptions);
   return tab;
}

//已開啟tab切換時觸發
chrome.tabs.onActivated.addListener((tabId, windowId) => {});

//當tab更新時觸發
chrome.tabs.onUpdated.addListener((id, info, tab) => {});
```

Storage API及監聽事件

此時Storage主要存於Extension端，與 localStorage 主要區別為 使用者的相關資訊可以過使用storage.sync方法與Chrome自動同步，同時在Browser端也可以存取

```
chrome.storage.sync.set({ key: value }, () => {});

// 透過Key取出
chrome.storage.sync.get([key], (res) => res[key]);

//移除
chrome.storage.sync.remove(key, (val) => val);

//Storage 改變時觸發事件
chrome.storage.onChanged.addListener((changes, areaName) => {
  console.log(changes);
});
```

Chrome Extension 的 常見組成 Action



Action 為 設定 Chrome Extension UI介面的相關方法，提供許多定義方法讓開發者能依功能需求來自定義 Extension 樣式。

自定義 Popup 樣式

與Background相同，要自定義Extension Popup視窗一樣也需在 Manifest.json 中Action欄位的"default_popup"註冊相關HTML檔案路徑，如果需要自定義 Icon 也需要在 default_icon 欄位中指定對應圖片

```
// Manifest.json
"action": {
  "default_popup": "/popup/popup.html"
  "default_icon": {
    "16": "logo_16x16.png",
    "32": "logo_32x32.png"
  }
  ...
},
```

Action 相關API

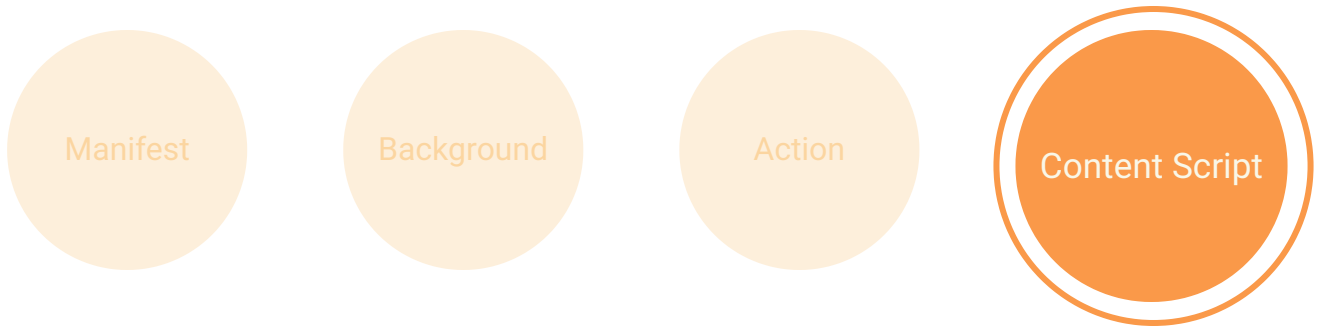
依照狀態 動態切換 popup 頁面

```
// popup.js
chrome.storage.local.get("signed_in", (data) => {
  if (data.signed_in) {
    chrome.action.setPopup({ popup: "popup.html" });
  } else {
    chrome.action.setPopup({ popup: "popup_sign_in.html" });
  }
});
```

依照狀態設定 popup 狀態提示 及 變更 Icon 圖示

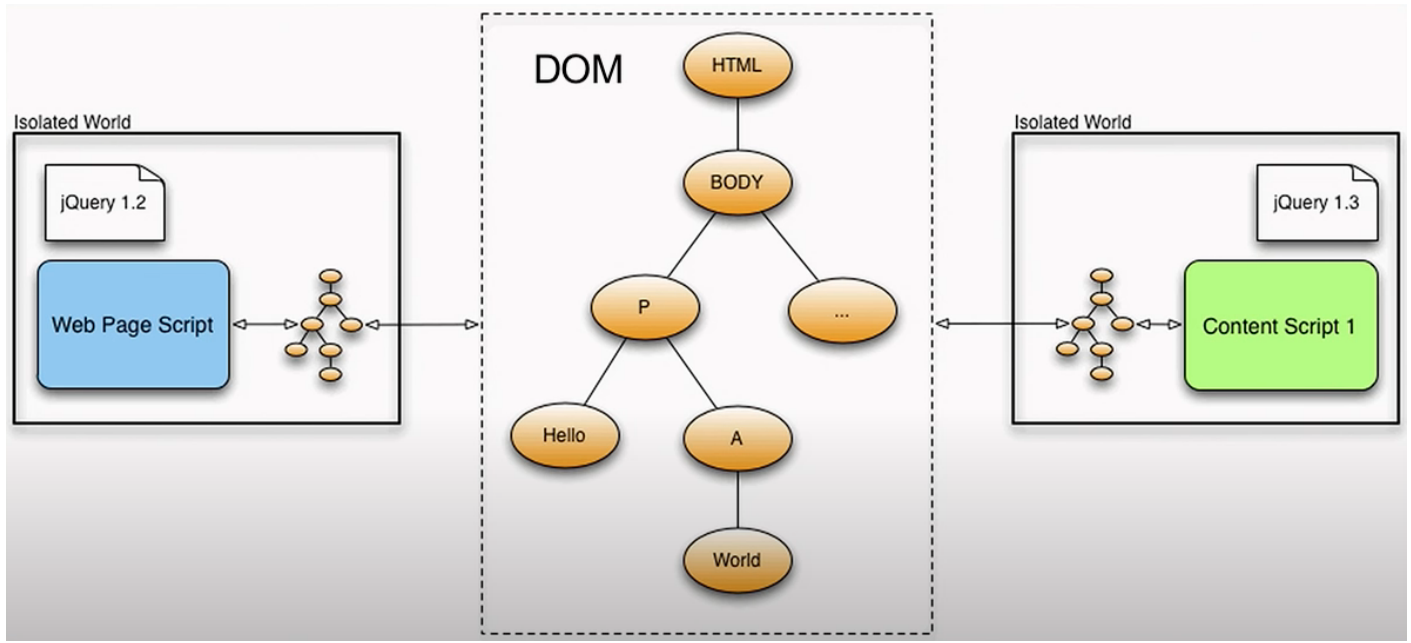
```
// popup.js
chrome.action.setBadgeText({ text: "ON" });
chrome.action.setBadgeBackgroundColor({ color: "#4688F1" });

const setIconStatus = async (iconPath) => {
  await chrome.action.setIcon({
    path: iconPath,
  });
};
```

Chrome Extension 的 常見組成 Content Script



Content Script 是 Extension 中唯一執行於 瀏覽器端 的 Context ，可以直接訪問 網頁中的資訊 且對DOM元素進行操作，同時又不會與頁面或 其他Extension 的內容腳本發生衝突，其中注入網站的方法又分為 『靜態 Inject Script 』 與 『動態 Inject Script』

為什麼不會與頁面其他 script 發生衝突呢？



因為 Content Script 主要運行於 Isolated Worlds，雖然都可以操作該頁面上的 DOM，但與 網頁上的 Script 實際上是兩個獨立的執行環境，所以不會相互影響

靜態注入 Script

需在manifest檔中content_script宣告對應檔案(js、css)

```
"content_scripts": [
  {
    "matches": ["https://*.xxx.com/*"], //指定特定網域
    "css": ["content.css"],
    "js": ["content.js"],
    "run_at": "document_start" // 注入的時機
  }
],
```

『run_at 欄位』 為指定何時將 Cotent Script 程式 注入於頁面中，分別有三個選項: document_idle、 document_start 或 document_end

更多 靜態 Inject 相關功能設定 請見 -> 官方文件

動態注入 Script

而動態 Inject Script 則需 『 Manifest.json 』 中的 『 Permission 』 欄位宣告 Scripting 權限

```
// manifest.json
{
  "permissions": ["scripting"]
}
```

透過 『 chrome.scripting.executeScript 』 方法將 JavaScript 檔案 注入 到 指定的 TabId 中。

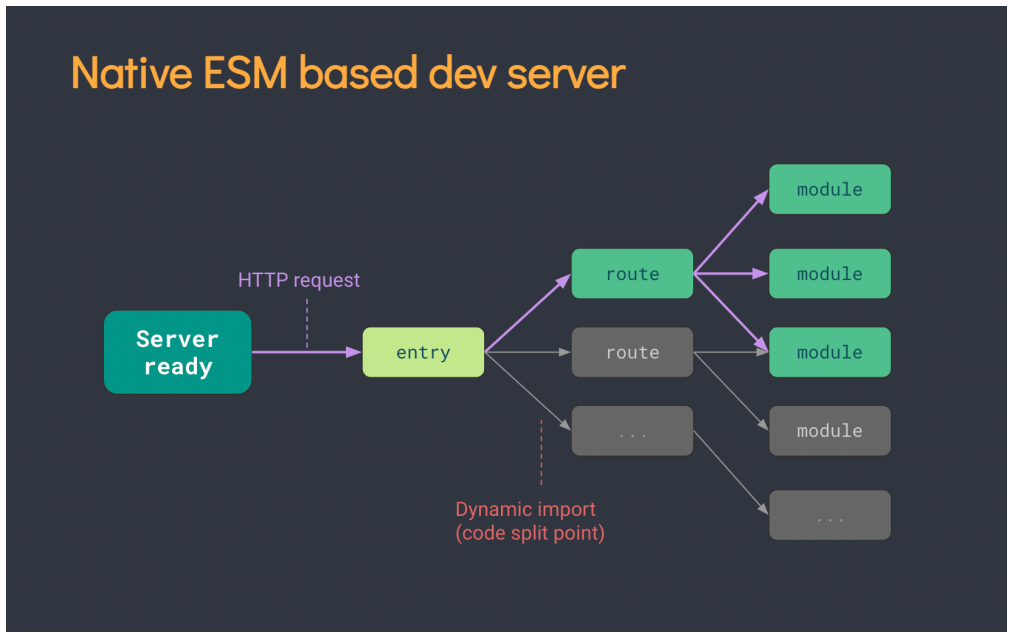
```
// background.js
chrome.scripting.executeScript(
    {
        target: { tabId: tab.id },
        matches: ["https://*.xxx.com/*"],
        function: injectedFunction // 也可以指定檔案路徑的形式 files: ['script.js'],
        args:['HiHi', 'from Winnie']
    },
    (d) => { console.log(d) //這裡可以接收回傳參數 }))
```

關於 Content Script 的限制

如何用 Vite 開發

什麼是 Vite

Vite 是一個打包工具，主要透過利用 瀏覽器支援 的 Native ESM 來作為核心運作，根據 Http的request 來載入模組進行處理，實現真正的依需求加載，讓開發速度及編譯可以變得更快。



快 ?! 我不喜歡太快!!

別急~示範一次給你看看

創建一個Vite專案

```
//npm：
npm create vite@latest extension-name
```

依照 Extension 功能需求，在『 src 』內建立相關檔案，如:background 等，而 manifest.json 不需編譯，所以可以放置在『 public 』中。

```
.
├── popup.html //根目錄
├── package.json
├── public
│   ├── logo.png
│   └── manifest.json
├── src
│   ├── background.js
│   ├── content_script
│   │   └── content.js
│   └── popup
│       └── popup.js
└── vite.config.js
```

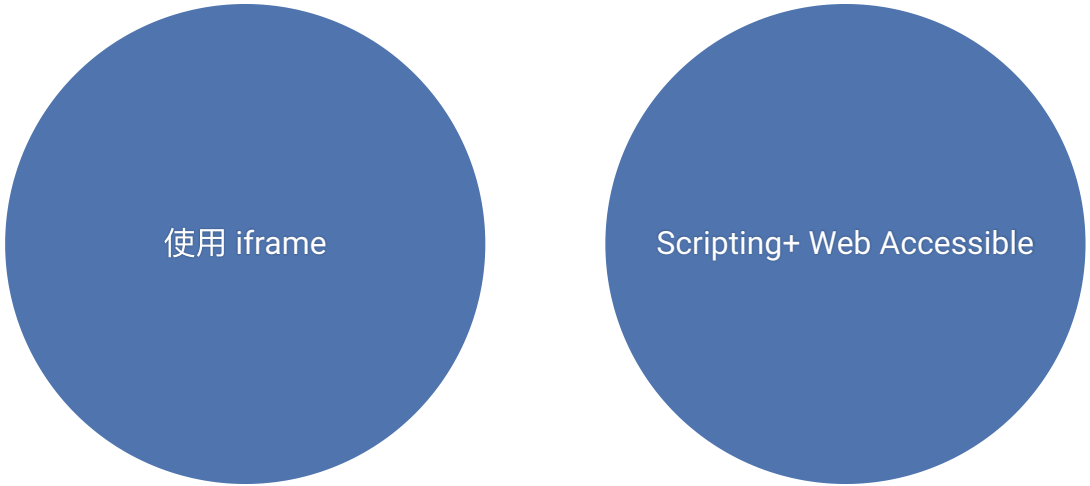
Vite config 設定

接著在 vite.config中設定檔案打包路徑，在build中透過 rollupOptions 屬性的input指定檔案的路徑位置，而output可以設置檔案輸出的名字。

```
export default defineConfig({
  ....
  build: {
    rollupOptions: {
      output: {
        entryFileNames: `[name].js`,
        chunkFileNames: `[name].js`,
        assetFileNames: `[name].[ext]`,
      },
      input: {
        popup: resolve(__dirname, 'popup.html'),
        background: resolve(__dirname, '/src/background.js'),
        content: resolve(__dirname, 'content.html')
      }
    }
  },
  ...
})
```

Content Script 與 ES Module的坑 !!

關於 Content Script 與 ES Module 的 坑



感謝大家!!