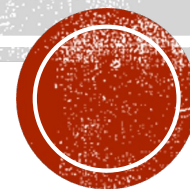


HTML5 JS API

林新德

shinder.lin@gmail.com

<https://github.com/shinder/html5-js-api-base>



0. Web APIs

- 主題列表：<https://developer.mozilla.org/en-US/docs/Web/API>
- 參考網站：<http://html5index.org/>。



0.1 將介紹的主題

- 1. 新增的標籤與屬性
- 2. Media: Audio and Video
- 3. Canvas
- 4. SVG
- 5. Drag and Drop
- 6. File and FileReader
- 7. Web Storage
- 9. **Server Sent Event**
- 10. Web Sockets
- 11. AJAX File Upload
- 12. Web Workers
- 13. **Chrome** 啟動圖示
- 14. 離線使用
- 15. Web Notifications
- 16. Geolocation



0.2 工具網站

- 查看某個功能在各瀏覽器的支援情況

<https://caniuse.com/>

<https://zh.wikipedia.org/wiki/W3C推薦標準>

- 工作草案 (WD, Working draft)
- 候選推薦標準 (CR, Candidate recommendation)
- 提案推薦標準 (PR, Proposed recommendation)
- W3C推薦標準 (REC, W3C recommendation)



0.3 環境準備

■ 部份內容會使用到後端環境，故會使用 `node/express` 環境

- 1. 在欲放置專案資料夾的路徑，建立專案資料夾。

```
mkdir html5-js-api-base
```

- 2. 切換工作目錄到專案資料夾內。

```
cd html5-js-api-base
```

- 3. 使用 `npm` 初始化專案。

```
npm init -y
```

- 4. 利用 `npm` 安裝所需套件。

```
npm i express  
npm i serve-index  
npm i --save-dev nodemon
```



- 5. 編輯 `package.json`，使用 ESM。

```
"type": "module",
```

- 6. 在專案目錄建立 `public` 資料夾，並放入一支 `html` 檔。

```
mkdir public
```

- 7. 修改 `package.json`，加入啟動的 `scripts`。

```
"scripts": {  
  "dev": "nodemon index.js"  
},
```

- 8. 在專案目錄新增檔案 `index.js`，並輸入以下內容。



```
import express from "express";
import serveIndex from "serve-index";

const web_port = 3031;
const app = express();

app.use(express.static("public"));
app.use("/", serveIndex("public", { icons: true }));

app.listen(web_port, () => {
  console.log(`伺服器啟動於通訊埠 : ${web_port}`);
});
```

- 9. 在專案目錄的命令列，輸入下式啟動伺服器，並於瀏覽器上查看 <http://localhost:3031>。

```
npm run dev
```





1. 新增的標籤與屬性

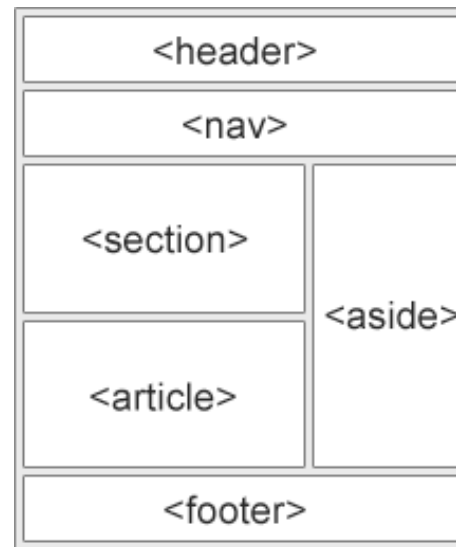
- Semantic Elements
- Form Elements
- Contenteditable



1.1 Semantic Elements

https://www.w3schools.com/html/html5_semantic_elements.asp

<article>
<aside>
<details>
<figcaption>
<figure>
<footer>
<header>
<main>
<mark>
<nav>
<section>
<summary>
<time>



1.2 表單的 input

https://www.w3schools.com/html/html_form_input_types.asp

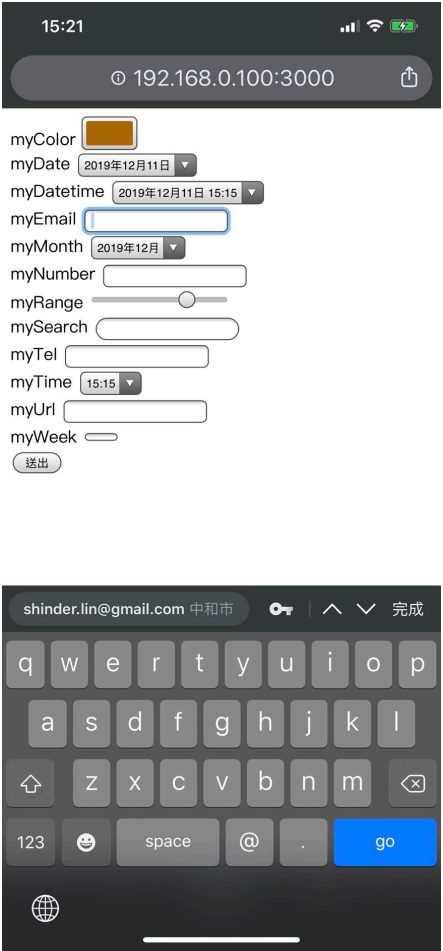
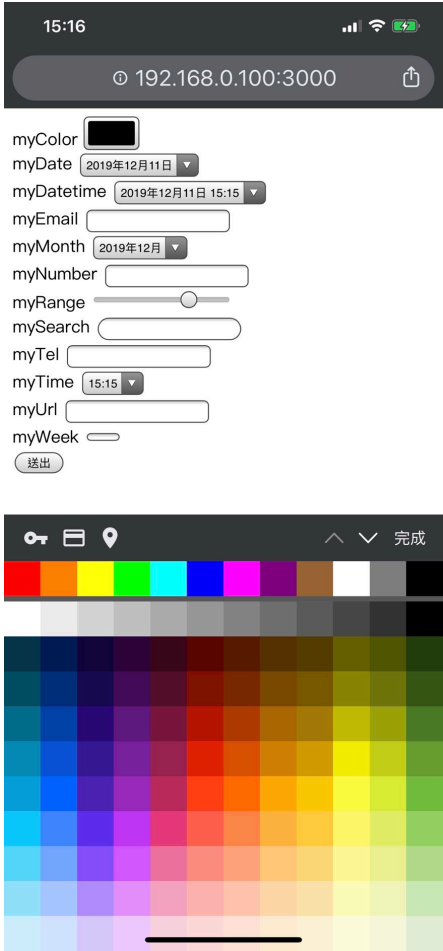
▪ input 新的 type 屬性值

```
<input type="button">  
<input type="checkbox">  
<input type="file">  
<input type="hidden">  
<input type="image">  
<input type="password">  
<input type="radio">  
<input type="reset">  
<input type="submit">  
<input type="text">
```

```
<input type="color">  
<input type="date">  
<input type="datetime-local">  
<input type="email">  
<input type="month">  
<input type="number">  
<input type="range">  
<input type="search">  
<input type="tel">  
<input type="time">  
<input type="url">  
<input type="week">
```



101-form-input-type.html



- https://www.w3schools.com/html/html_form_attributes.asp

- **input** 新的屬性

autocomplete
autofocus
height and width
list (搭配 datalist)
min and max
multiple
pattern (regexp)
placeholder
required
step

- **form** 新的屬性

autocomplete
novalidate



102-form-input-pattern.html

```
<form onsubmit="return false;" data-novalidate autocomplete="on">
  <label for="myName">姓名 : </label>
  <input type="text" name="myName" id="myName" required autofocus /><br />

  <label for="myEmail">電郵 : </label>
  <input type="email" name="myEmail" id="myEmail" /><br />

  <label for="myGroup">組別 : </label>
  <input type="text" name="myGroup" id="myGroup" list="myList" />
  <datalist id="myList">
    <option value="甲組"></option>
    <option value="乙組"></option>
    <option value="丙組"></option>
  </datalist><br />

  <label for="myTel">手機號碼 : </label>
  <input type="tel" name="myTel" id="myTel" pattern="09\d{2}-?\d{3}-?\d{3}" />
  <br />
  <input type="submit" />
</form>
```



- 自訂驗證訊息 103-form-custom-validity.html

```
const myPass = document.querySelector("#myPass");
const myPass2 = document.querySelector("#myPass2");

myPass2.addEventListener("blur", (event) => {
  console.log(event);
  if (myPass.value !== myPass2.value) {
    // 只要有設定，送出前就會顯示
    myPass2.setCustomValidity("密碼確認欄必須和密碼欄相同內容");
  } else {
    // 取消
    myPass2.setCustomValidity("");
  }
});
```



■ 自訂驗證訊息 104-form-custom-validity-2.html

```
let inputs = document.querySelectorAll("input");
const myValidate = function (event) {
  const t = event.target;
  const v = t.validity;
  let msg = "";
  switch (true) {
    case v.valueMissing:
      msg = "必填欄位";
      break;
    case v.typeMismatch:
      msg = "請填寫正確的類型";
      break;
    case v.patternMismatch:
      msg = "請填寫規定的格式";
      break;
  }
  t.setCustomValidity(msg);
};
inputs.forEach((field) => {
  field.addEventListener("blur", myValidate);
});
```



1.3 Contenteditable

105-contenteditable.html

```
<style>
  #info {
    background-color: #00b7ff;
  }
  #info:focus {
    background-color: #ffc44e;
  }
</style>
<div
  id="info"
  contenteditable="true"
  oninput="console.log(this.innerText);"
>
  123
</div>
```





2. Embed Media: Audio and Video

- 在頁面內嵌入多媒體內容。
- `<audio>` 標籤為 [HTMLAudioElement](#) 類型。
- `<video>` 標籤為 [HTMLVideoElement](#) 類型。
- `HTMLAudioElement` 和 `HTMLVideoElement` 皆繼承自 [HTMLMediaElement](#)。
- 特有的事件定義於 [HTMLMediaElement](#)。
- 由於聲音可以是沒有外觀的，可以用 **Audio** 類別建立。



2.1 Audio

201-audio-tag.html

```
<audio
  src="media/196381__minitauross__panflute-loop.mp3"
  data-src="mp3, wav, ogg"
  preload="auto"
  data-preload="auto, metadata, none"
  loop
  data-loop="循環播放"
  autoplay
  data-autoplay="自動播放，目前政策是無效果"
  controls
  data-controls="控制器"
>
  <a href="media/196381__minitauross__panflute-loop.mp3"
    >沒有支援 audio 標籤時顯示，下載音樂檔</a>
  <
</audio>
```



- 使用 **Audio** 物件動態載入聲音檔

202-audio-object.html

```
const doLoad = document.querySelector("#doLoad");
const doPlay = document.querySelector("#doPlay");
const au = new Audio();
const handler = (event) => {
  console.log(` ${event.type}
    duration: ${au.duration}, currentTime: ${au.currentTime},
    volume: ${au.volume}, muted: ${au.muted}, paused: ${au.paused} `);
};
au.addEventListener("loadedmetadata", handler);
au.addEventListener("timeupdate", handler);
au.addEventListener("ended", handler);

doLoad.onclick = () => {
  au.src = "media/196381__minitauross__panflute-loop.mp3";
  doLoad.style.display = "none";
  doPlay.style.display = "block";
};
doPlay.onclick = () => (au.paused ? au.play() : au.pause());
```



2.2 Video

- Demo 的影片檔：<https://www.webmfiles.org/demo-files/>

```
<video
  src="media/big-buck-bunny_trailer.webm"
  data-src="webm, mp4, avi, ogv"
  preload="metadata"
  data-preload="auto, metadata, none"
  loop data-loop="循環播放"
  autoplay data-autoplay="自動播放，目前政策是無聲才能自動播放"
  controls data-controls="控制器"
  poster="images/bunny-poster.png"
  data-poster="未播放時替代圖片"
  width="320" height="240" muted
>
  <p>沒有支援 video 標籤時顯示</p>
</video>
```



■ 動態載入影片檔 (一)

204-video-object.html

```
<video id="player" controls></video>
<div id="btnGroup">
  <button id="video1">video 1</button>
  <button id="video2">video 2</button>
  <button id="doPlay">play or pause</button>
</div>
<script>
  const videos = [
    "media/big-buck-bunny_trailer.webm",
    "media/elephants-dream.webm",
  ];
  const player = document.querySelector("#player");
  const playerListener = (event) => {
    console.log(` ${event.type}
      duration: ${player.duration}, currentTime: ${player.currentTime},
      volume: ${player.volume}, muted: ${player.muted},
      paused: ${player.paused} `);
  };

```



■ 動態載入影片檔 (承上頁)

204-video-object.html

```
player.addEventListener("loadedmetadata", playerListener);
player.addEventListener("timeupdate", playerListener);
player.addEventListener("ended", playerListener);
const btnsListener = (event) => {
  switch (event.target.id) {
    case "video1":
      player.src = videos[0];
      break;
    case "video2":
      player.src = videos[1];
      break;
    case "doPlay":
      player.paused ? player.play() : player.pause();
  }
};
document
  .querySelector("#btnGroup")
  .addEventListener("click", btnsListener);
</script>
```



- 字幕 WebVTT
- https://developer.mozilla.org/zh-TW/docs/Web/API/Web_Video_Text_Tracks_Format

205-video-subtitles.html

```
<video
  src="media/big-buck-bunny_trailer.webm"
  preload="auto"
  controls
  poster="images/bunny-poster.png"
>
  <track
    kind="subtitles"
    label="正體中文"
    src="media/bunny-zh.vtt"
    default
  />
  <track kind="subtitles" label="English" src="media/bunny-en.vtt" />
</video>
```

WEBVTT

NOTE 這是VTT註解

1
00:00:05.000 --> 00:00:08.000
This is the first subtitle.

2
00:00:12.000 --> 00:00:20.000
This is the second.

3
00:00:20.000 --> 00:00:25.000
Third





3. Canvas

- 線與多邊形
- 矩形
- 圓、弧、扇形
- 圖片
- 文字
- 影片截取
- Canvas lib: `pixi.js`, `three.js`, `frabric.js`



3.1 畫線與多邊形

301-canvas-line-1.html

```
<!-- 預設 300px * 150px -->
<canvas id="myCanvas" width="600" height="400"></canvas>
<script>
  const myCanvas = document.querySelector("#myCanvas");
  const ctx = myCanvas.getContext("2d");

  ctx.beginPath();      // 重置 path
  ctx.moveTo(50, 50);
  ctx.lineTo(200, 300);
  ctx.lineTo(400, 50);
  ctx.strokeStyle = "orange"; // 設定畫筆顏色
  ctx.lineWidth = 20;    // 設定畫筆粗細
  ctx.stroke();          // 在路徑上畫線
</script>
```



302-canvas-line-2.html

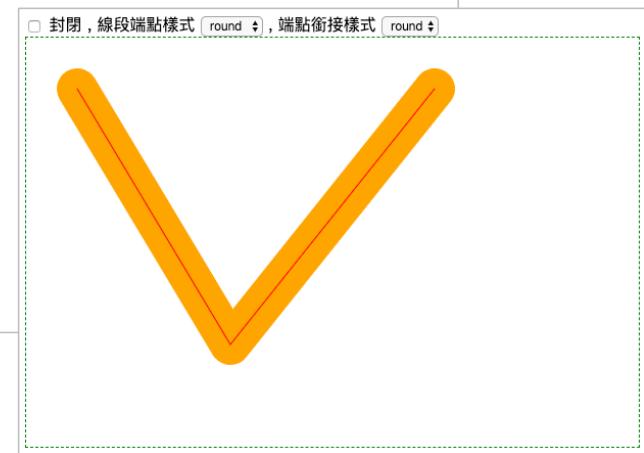
```
<form name="form1">
  <input type="checkbox" name="isClosedPath" id="isClosedPath" />
  <label for="isClosedPath">封閉</label> ,
  <label for="lineCap">線段端點樣式</label>
  <select name="lineCap" id="lineCap">
    <option value="butt" selected>butt</option>
    <option value="round">round</option>
    <option value="square">square</option>
  </select>
  ,
  <label for="lineJoin">端點銜接樣式</label>
  <select name="lineJoin" id="lineJoin">
    <option value="miter" selected>miter</option>
    <option value="round">round</option>
    <option value="bevel">bevel</option>
  </select>
</form>
<canvas id="myCanvas" width="600" height="400"></canvas>
```



302-canvas-line-2.html

```
let isClosedPath = document.form1.isClosedPath.checked;
let lineCap = document.form1.lineCap.value;
let lineJoin = document.form1.lineJoin.value;

ctx.clearRect(0, 0, myCanvas.width, myCanvas.height); // 清除畫布
ctx.beginPath(); // 重置 path
ctx.moveTo(50, 50);
ctx.lineTo(200, 300);
ctx.lineTo(400, 50);
isClosedPath ? ctx.closePath() : false;
ctx.strokeStyle = "orange";
ctx.lineWidth = 40;
ctx.lineCap = lineCap;
ctx.lineJoin = lineJoin;
ctx.stroke();
```



3.2 畫矩形

303-canvas-rect.html

```
const myCanvas = document.querySelector("#myCanvas");
const ctx = myCanvas.getContext("2d");

ctx.strokeStyle = "orange";           // 設定畫筆顏色
ctx.lineWidth = 20;                   // 設定畫筆粗細
ctx.lineJoin = "round";
ctx.strokeRect(50, 50, 100, 100);     // (x, y, width, height)
ctx.fillStyle = "lightblue";          // 填滿的顏色
ctx.fillRect(50, 200, 100, 100);
// 陰影效果
ctx.shadowOffsetX = 10;                // 位移
ctx.shadowOffsetY = 10;
ctx.shadowColor = "rgba(0,0,0,.5)";
ctx.shadowBlur = 4;                   // 模糊效果
ctx.strokeRect(350, 50, 100, 100);
ctx.fillRect(350, 200, 100, 100);
```



3.3 畫圓、弧、扇形

304-canvas-arc.html

```
const myCanvas = document.querySelector("#myCanvas");
const ctx = myCanvas.getContext("2d");

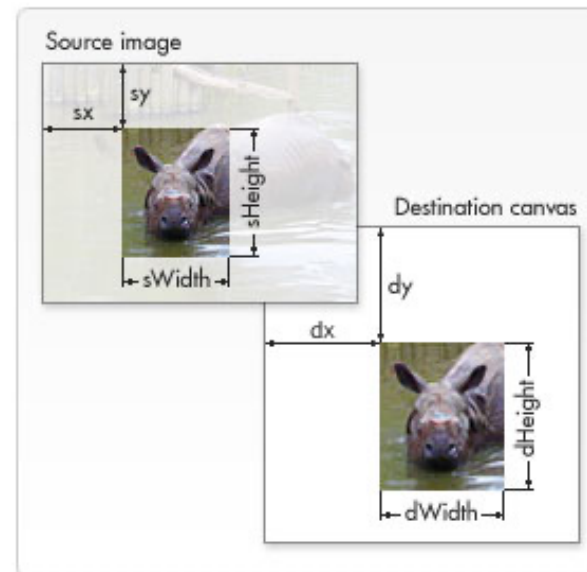
ctx.strokeStyle = "red";          // 設定畫筆顏色
ctx.lineWidth = 6;                // 設定畫筆粗細
ctx.fillStyle = "lightblue";     // 填滿的顏色

// arc(x, y, radius, startAngle, endAngle, anticlockwise);
// arc(圓心x, 圓心y, 半徑, 起始角度, 結束角度, 是否為逆時針方向);
ctx.arc(420, 60, 50, 0, Math.PI * 2); // 圓形
ctx.stroke();
```



3.4 畫圖片

```
// 定點繪製  
drawImage(image, x, y)  
  
// 縮放  
drawImage(image, x, y, width, height)  
  
// 切割影像  
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
```



https://developer.mozilla.org/zh-TW/docs/Web/API/Canvas_API/Tutorial/Using_images



- 等比例縮放：305-canvas-image-1.html

```
const myCanvas = document.querySelector("#myCanvas");
const ctx = myCanvas.getContext("2d");

const showImg = () => {
  ctx.clearRect(0, 0, myCanvas.width, myCanvas.height);
  let w = myCanvas.width;
  let h = myCanvas.height;
  switch (myWay.value) {
    case "1":
      h = (img.height * myCanvas.width) / img.width;
      break;
    case "2":
      w = (img.width * myCanvas.height) / img.height;
      break;
  }
  ctx.drawImage(img, 0, 0, w, h);
};

const img = new Image();
img.addEventListener("load", showImg);
img.src = "images/bunny-poster.png";
myWay.addEventListener("change", showImg);
```



- 從客戶端圖檔繪入：306-canvas-image-2.html

```
<canvas id="myCanvas" width="600" height="400"></canvas>
<br />
<input type="file" id="myFile" hidden accept="image/*" />
<button onclick="myFile.click()">繪製圖檔</button>
<script>
  const myCanvas = document.querySelector("#myCanvas");
  const ctx = myCanvas.getContext("2d");

  const img = new Image();
  img.addEventListener("load", (event) => {
    ctx.drawImage(img, 0, 0);
  });

  myFile.addEventListener("change", () => {
    img.src = URL.createObjectURL(myFile.files[0]);
  });
</script>
```



3.5 畫文字

307-canvas-text.html

```
const myCanvas = document.querySelector("#myCanvas");
const ctx = myCanvas.getContext("2d");
ctx.fillStyle = "red";
ctx.strokeStyle = "blue";
ctx.lineWidth = 2;

ctx.textAlign = "center";
ctx.textBaseline = "middle";

ctx.font = "bold 72px serif";
ctx.fillText("Hello Canvas", 300, 100);
ctx.font = "bold 72px arial";
ctx.strokeText("Hello Canvas", 300, 200);
ctx.fillText("哈囉", 300, 300);
```



3.6 截取影片

308-canvas-video.html

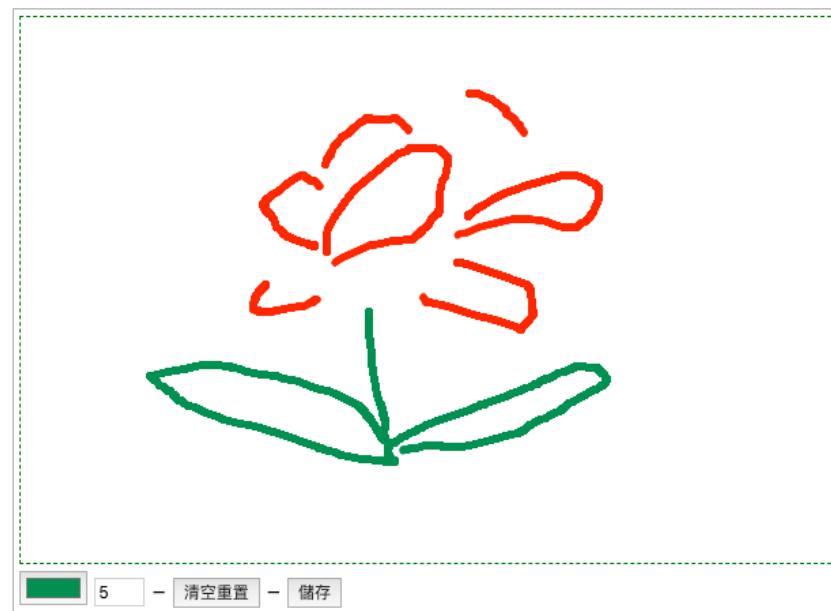
```
<video src="media/big-buck-bunny_trailer.webm" preload="auto"
  controls></video>
<button onclick="takePic()">截取</button>
<br />
<canvas id="myCanvas"></canvas>
<script>
  const myCanvas = document.querySelector("#myCanvas");
  const ctx = myCanvas.getContext("2d");
  const video = document.querySelector("video");

  const takePic = () => {
    myCanvas.setAttribute("width", video.videoWidth);
    myCanvas.setAttribute("height", video.videoHeight);
    ctx.drawImage(video, 0, 0);
  };
</script>
```



3.7 塗鴉練習

310-canvas-draw-app.html





4. SVG

- 網頁向量圖標準
- XML
- 有自己的標籤定義
- 可使用 **style**
- 可放入 **HTML** 檔內，可使用 **DOM API** 操作
- 方便使用繪畫軟體製作，例如：Adobe illustrator, [Inkscape](#)
- <https://developer.mozilla.org/en-US/docs/Web/SVG/Element>
- 有名的 lib：D3.js, snap.svg



4.1 常見的 SVG 標籤

- <svg>：頂層標籤，設定顯示區大小。
- <line>：線
- <rect>：矩形
- <circle>：圓形
- <ellipse>：橢圓
- <polygon>：多邊形
- <path>：路徑
- <g>：群組



- 401-svg-demo.html

```
<svg viewBox="0 0 300 200" xmlns="http://www.w3.org/2000/svg">
```

```
  <polygon
```

```
    class="c1"
```

```
    points="0,100 25,25 75,75 100,0" />
```

```
  <polygon
```

```
    points="100,150 125,75 175,125 200,50"
```

```
    fill="green"
```

```
    stroke="black"
```

```
    stroke-width="10"
```

```
  />
```

```
</svg>
```

```
<script>
```

```
  const polygon1 = document.querySelector("polygon");
```

```
  polygon1.onclick = () => {
```

```
    const newClass = polygon1.classList.contains("c1") ? "c2" : "c1";
```

```
    polygon1.setAttribute("class", newClass);
```

```
  };
```

```
</script>
```

```
<style>
```

```
  svg {
```

```
    width: 200px;
```

```
    height: 100px;
```

```
  }
```

```
  .c1 { fill: red; }
```

```
  .c1:hover { fill: blue; }
```

```
  .c2 { fill: orange; }
```

```
  .c2:hover { fill: purple; }
```

```
</style>
```





5. Drag and Drop (拖放功能)

- `dragstart`: 開始拖拉
- `drag`: 正在拖拉 (類似 `mousemove`)
- `dragend`: 結束拖拉
- `dragenter`: 有拖拉物進入時
- `dragover`: 有拖拉物在範圍內移動時
- `dragleave`: 有拖拉物離開時
- `drop`: 拖拉物在範圍內放開時
- `dragover` 的處理器內要呼叫 `preventDefault()` , 才能觸發 `drop` 事件。
- `<a>` 和 `` 原本就有預設拖拉的功能。



拖曳測試：501-drag.html

```
<style>
  .rect {
    position: absolute;
    width: 200px;
    height: 200px;
    background-color: #a2eff6;
    border: 1px solid gray;
  }
  .ball {
    width: 60px;
    height: 60px;
    border-radius: 50%;
    text-align: center;
    line-height: 60px;
    background-color: #ee6f2f;
    border: 1px solid black;
    color: white;
    font-weight: 800;
    cursor: pointer;
  }
</style>
```

```
<div class="rect">
  <div class="ball" draggable="true">77</div>
</div>

<script>
  const b = document.querySelector(".ball");
  const handler = (e) => {
    console.log(e.type);
  };
  // 開始拖曳
  b.addEventListener("dragstart", handler);

  // b.addEventListener('drag', handler); // 拖曳中

  // 結束拖曳
  b.addEventListener("dragend", handler);
</script>
```



拖曳放開測試：502-drag-drop.html

```
const b = document.querySelector(".ball");
const [rect1, rect2] = document.querySelectorAll(".rect");
const dragHandler = (e) => console.log(e.type);

b.addEventListener("dragstart", dragHandler); // 開始拖曳
b.addEventListener("dragend", dragHandler);   // 結束拖曳

rect2.addEventListener("dragenter", (e) => {
  e.currentTarget.style.backgroundColor = "#ffeff6";
});
rect2.addEventListener("dragover", (e) => e.preventDefault());
rect2.addEventListener("dragleave", (e) => {
  e.currentTarget.style.backgroundColor = "#a2eff6";
});
rect2.addEventListener("drop", (e) => {
  e.currentTarget.style.backgroundColor = "#a2eff6";
  console.log("drop");
});
```



使用 `dataTransfer` : 503-drag-drop.html

```
const b = document.querySelector(".ball");
const [rect1, rect2] = document.querySelectorAll(".rect");

b.addEventListener("dragstart", (e) => {
  // element id 當作資料
  e.dataTransfer.setData("text", e.currentTarget.id);
});
rect2.addEventListener("dragover", (e) => e.preventDefault());

rect2.addEventListener("drop", (e) => {
  const tid = e.dataTransfer.getData("text");
  // 透過 id 取得 element 再加入 rect2
  e.currentTarget.append(document.getElementById(tid));
});
```



使用變數記錄拖曳對象：504-drag-drop.html

```
let dragTarget = null; // 目前拖拉的對象
const balls = document.querySelectorAll(".ball");
const rects = document.querySelectorAll(".rect");

balls.forEach((b) => {
  b.addEventListener("dragstart", (e) => (dragTarget = e.currentTarget));
  b.addEventListener("dragend", (e) => (dragTarget = null));
});

rects.forEach((r) => {
  r.addEventListener("dragover", (e) => e.preventDefault());
  r.addEventListener("drop", (e) => {
    if (dragTarget) {
      r.append(dragTarget);
    }
  });
});
```





6. FileList, File and FileReader

- 由於安全性考量，不能直接存取檔案。
- 透過使用者的操作（允許）可有條件的讀取檔案。
- `<input type="file">` 可讓使用者決定要讀取或上傳的檔案。
- **File** 類型的物件用來表示檔案的實體，但無法使用 **JS** 取得確切的磁碟路徑。
- **FileReader** 類型的物件用以讀取檔案的內容。



6.1 FileList 和 File 類型

檔案欄位: 601-file-field.html

```
<input
  type="file"
  id="fileFiled"
  style="display: none"
  multiple
  accept="image/*"
/>
<div class="container" onclick="fileFiled.click()">點擊選取檔案</div>
<script>
  const container = document.querySelector(".container");
  const fileFiled = document.querySelector("#fileFiled");

  fileFiled.addEventListener("change", (e) => {
    for (let f of fileFiled.files) {
      container.innerText += `\n${f.name}`;
    }
  });
</script>
```



拖拉進來的檔案: 602-dnd-files.html

```
container.addEventListener("drop", function (e) {
  e.preventDefault(); // 避免預設行為 (瀏覽器開始圖檔)
  container.style.backgroundColor = "#b0e9ff";

  for (let f of e.dataTransfer.files) {
    // 判斷是否為圖檔
    if (f.type.indexOf("image/") === 0) {
      const url = URL.createObjectURL(f); // 同步方式取得 url
      container.innerHTML += `<div class="box">
        
        </div>`;
    }
  }
});
```



6.2 FileReader 類型

讀取文字檔內容: 603-read-text.html

```
<input type="file" id="fileFiled" hidden accept="text/*" />
<button onclick="fileFiled.click()">開啟一個文字檔</button><br />
<textarea class="container" rows="100" cols="50"></textarea>
<script>
  const container = document.querySelector(".container");
  const fileFiled = document.querySelector("#fileFiled");
  const readerLoaded = (event) => {
    container.append(event.target.result);
  };
  fileFiled.addEventListener("change", (event) => {
    if (!fileFiled.files.length) return;
    const file = fileFiled.files[0];
    const reader = new FileReader();
    reader.addEventListener("load", readerLoaded);
    reader.readAsText(file);
  });
</script>
```



讀取圖檔內容: 604-read-images.html

```
const readerLoaded = (event) => {  
  let image = new Image();  
  image.height = 150;  
  image.src = event.target.result; // 將結果設定給 image  
  container.append(image);  
};  
  
fileFiled.addEventListener("change", (e) => {  
  for (let file of fileFiled.files) {  
    const reader = new FileReader();  
    reader.addEventListener("load", readerLoaded);  
    reader.readAsDataURL(file); // 非同步  
  }  
});
```

** **FileReader.readAsDataURL()** 為**非同步**操作，結果為 **base64** 字串。

** **URL.createObjectURL()** 為**同步**操作，結果為暫存的 **url**。





7. Web Storage

- 前端常用的儲存資料技術：**Cookie**, **Web Storage**, **Indexed DB**。
- 採「**同源政策**」(**same-origin policy**)。
- **Web Storage** 包含兩個物件：**localStorage** 和 **sessionStorage**。
- 儲存格式：**key-value pair** 文字格式。
- 兩個物件的方法皆相同。
- 容量限制約 **5M ~ 10M**，依瀏覽器實作而定。
- **localStorage** 適用於整個網站。
- **sessionStorage** 只適用於單一個瀏覽器視窗或分頁。



7.1 Web Storage 方法

- **length** 取得資料筆數
- **key(n)** 取得索引值為 n 的 key
- **setItem(key, value)** 設定項目
- **getItem(key)** 讀取項目
- **removeItem(key)** 移除項目
- **clear()** 刪除所有項目



7.2 使用 localStorage

- 儲存資料：701-localStorage-setItem.html

```
const prefix = "friend-book;;";
document.form1.onsubmit = (e) => {
  e.preventDefault();
  const t = Date.now(); // 當下時間 timestamp 單位毫秒數
  let data = {
    name: document.form1.name.value,
    age: document.form1.age.value,
    mobile: document.form1.mobile.value,
    time: t,
  };
  localStorage.setItem(prefix + t, JSON.stringify(data));
  document.form1.reset();
  alert("儲存完成！");
};
```



- 讀取資料：702-localStorage-getItem.html

```
const prefix = "friend-book;";
const tbody = document.querySelector("#the-list");
const showList = () => {
  let str = "";
  for (let i = 0; i < localStorage.length; i++) {
    let key = localStorage.key(i);
    if (key.indexOf(prefix) === 0) {
      let item = localStorage.getItem(key);
      item = JSON.parse(item);
      str += `<tr>
        <td>${item.name}</td>
        <td>${item.age}</td>
        <td>${item.mobile}</td>
      </tr>`;
    }
  }
  tbody.innerHTML = str;
};
showList();
```



- 讀取列表加刪除功能：703-localStorage-list.html

```
const showList = () => {
  let str = "";
  for (let i = 0; i < localStorage.length; i++) {
    let key = localStorage.key(i);
    if (key.indexOf(prefix) === 0) {
      let item = localStorage.getItem(key);
      item = JSON.parse(item);
      str += `<tr>
        <td><button onclick="removeItem('${key}')">X</button></td>
        <td>${item.name}</td>
        <td>${item.age}</td>
        <td>${item.mobile}</td></tr>`;
    }
  }
  tbody.innerHTML = str;
};
showList();
const removeItem = (key) => {
  if (!confirm("確定要移除項目？")) return;
  localStorage.removeItem(key);
  showList();
};
```

** 問題：列表有按照新增順序嗎？如何解決？



- 儲存圖片：705-localStorage-draw-app.html

```
// 儲存項目
const saveCanvas = () => {
  const imgTxt = myCanvas.toDataURL("image/png");
  let key = "shinder-draw;;" + Date.now();
  localStorage.setItem(key, imgTxt);
  alert("儲存完成");
  renewList();
};

// 載入項目
const loadCanvas = (key) => {
  if (!key) return;
  clearCanvas();
  const img = new Image();
  img.onload = () => {
    ctx.drawImage(img, 0, 0);
  };
  img.src = localStorage.getItem(key);
};
```



7.3 使用 `sessionStorage`

- `localStorage` 的資料會一直存在用戶端，除非是手動或以 **JS** 去刪除。
- `sessionStorage` 的資料則只能在一個瀏覽器「視窗」或「分頁」存在。
- 在相同的分頁，不同頁面的轉變可以一直保有 `sessionStorage` 的資料。
- 當「視窗」或「分頁」被關閉，`sessionStorage` 的資料也會跟著消失。
- `sessionStorage` 通常不會超過容量限制。



711-sessionStorage-1.html

```
<div id="info"></div>
<br />
<a href="712-sessionStorage-2.html">next</a>
<script>
  const info = document.querySelector("#info");
  info.innerHTML = sessionStorage.getItem('s-key');
  const str = "Hello ~~";
  sessionStorage.setItem("s-key", str);
</script>
```

712-sessionStorage-2.html

```
<div id="info"></div>
<script>
  const info = document.querySelector("#info");
  const str = sessionStorage.getItem("s-key");
  info.innerHTML = str;
</script>
```





9. Server Sent Event

- 用戶端瀏覽器停留在該頁面（保持連線）的情況下，伺服器端可以間隔一段時間，將 **UTF-8** 資料推用到用戶端。
- 資料只能單向由伺服器端到用戶端。

- 伺服器端送出的檔頭：

```
Content-Type: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
```

- 資料格式：

```
: 可以放註解\n
event: myevent\n
id: 20\n
retry: 5000\n
data: 傳送的資料\n\n
```

```
: 註解
event: 自訂的事件名稱，預設為 message
id: 訊息id ( 可以是字串 )
retry: 重新連線毫秒數，預設為 5 秒
data: 傳送的資料
```



▪ SSE 測試 - 伺服器端

```
app.get("/try-sse", (req, res) => {  
  let id = 30;  
  res.writeHead(200, {  
    "Content-Type": "text/event-stream",  
    "Cache-Control": "no-cache",  
    Connection: "keep-alive",  
  });  
  setInterval(function () {  
    let now = new Date();  
    res.write(`id: ${id++}\n`);  
    res.write(`data: ${now.toLocaleString()}\n\n`);  
  }, 2000);  
});
```



- SSE 測試 - 客戶端 : 801-sse.html

```
<div id="info">123</div>
<script>
  const info = document.querySelector("#info");
  const es = new EventSource("/try-sse");

  es.addEventListener("open", (e) => {
    info.style.backgroundColor = "orange";
    console.log(new Date().toLocaleString(), e);
  });
  es.addEventListener("message", (e) => {
    info.innerHTML = `${e.lastEventId}: ${e.data}`;
  });
  es.addEventListener("error", (e) => {
    console.log(new Date().toLocaleString(), e);
  });
</script>
```





10. WebSocket

- 作 **client-server** 雙向的資料交換（溝通）。
- 有「狀態」的連線方式（**HTTP**為無狀態的連線方式）。
- 傳輸的資料以文字為主。
- 網路七層架構（**OSI** 模型 <https://zh.wikipedia.org/wiki/OSI模型>）中，**WebSocket** 屬於第7層（**HTTP**也是），**Socket** 則屬於第4層。
- 可以用 **Socket** 去實作上層的應用，但要處理的事務相當煩瑣。為了讓通訊更為簡便，所以有 **WebSocket** 的誕生。
- 常用來實作「聊天室」、「即時客服」等應用。
- 需要用 **WebSocket** 伺服器。
- **ws** 使用 port **80**，**wss** 使用 port **443**。



10.1 WebSocket 客戶端 API

- 使用 **WebSocket** 類別建立連線物件。
- **open** 事件：建立連線時觸發。
- **message** 事件：資料送達時觸發，**event.data** 可取得資料。
- **close** 事件：連線關閉時觸發。
- **error** 事件：發生錯誤時觸發。
- **bufferedAmount** 屬性：資料使用緩衝區的大小。使用 **send()** 方法時，資料會先放在緩衝區排隊送出。
- **readyState** 屬性：連線狀態（唯讀）。

常數	值	描述
CONNECTING	0	連線尚未打開。
OPEN	1	連線已打開，可以進行通訊。
CLOSING	2	連線正在進行關閉程序。
CLOSED	3	連線已關閉 / 連線不能打開。



10.2 ws: a Node.js WebSocket library

- 官網：<https://www.npmjs.com/package/ws>。
- 安裝：`npm i ws`
- 可以是獨立的 `server` 也可以和 Node 的 `http server` 使用相同的 `domain` 和 `port`。
- 說明文件：<https://github.com/websockets/ws/blob/master/doc/ws.md>



10.3 Map 和 Set 類型

- **Map** 和 **Set** 都是 ES6 加入的集合資料類型。
- **Map** 類型物件的功能類似 **Object** 物件，差別在於 **Map** 物件的 **key** 可以是參照類型。
- **Set** 物件相當於只有 **Map** 的 **key** 沒有 **value**，或者說 **key** 就是 **value**。
- **ws** 套件的 **WebSocketServer** 個體的 **clients** 屬性為 **Set** 類型，表示所有連線的用戶。



A01-Map.html

```
const obj = { name: "bill" }; // key: 字串, Symbol
const f1 = () => {};
const map = new Map();          // key: 可以是所有類型的物件

map.set({}, 123);
map.set({}, 456);
map.set(obj, "abc");
map.set(obj, [1, 2, 3]);
map.set(f1, "函式");

console.log(map.size);           // 鍵值對 數量
console.log(map.get(obj));       // 取得對應的值
console.log([...map.keys()]);    // 取得所有的 key 並轉換成陣列

// 以迴圈的方式取出 value 和 key
map.forEach((v, k) => {
  console.log({ v, k });
});
```



A02-Set.html

```
const set = new Set();           // key: 可以是所有類型的物件
const obj = { name: "bill" };
const f1 = () => {};

set.add(obj);
set.add(obj);
set.add(f1);
console.log(set.size);           // 數量

console.log([...set.keys()]);    // 取得所有的 key 並轉換成陣列

// 以迴圈的方式取出 value
set.forEach((v, k) => {
  console.log({ v, k });
});
```



10.4 回音程式

```
import "./routes/ws-echo.js";
```

後端主程式 index.js

```
import { WebSocketServer } from "ws";

const wsServer = new WebSocketServer({ port: 3070 });
wsServer.on("connection", (ws, req) => {
  const ip = req.socket.remoteAddress; // 用戶的 IP
  ws.on("message", (message) => {
    ws.send(message.toString());
  });

  ws.send(`連線了，你來自 ${ip}，連線人數：${wsServer.clients.size}`);
});

export default wsServer;
```

伺服器程式 routes/ws-echo.js



前端頁面 A03-WebSocket-echo.html

```
<!-- A03-WebSocket-echo.html 的 HTML -->
<button onclick="doConnect()">連線</button>
<br />
<input type="text" id="myInput" />
<button onclick="socket.send(myInput.value)" disabled id="sendBtn">
    送出
</button>
<br />
<div id="info"></div>
```



前端頁面 A03-WebSocket-echo.html

```
// A03-WebSocket-echo.html 的 JS
const info = document.querySelector("#info");
let socket;

const myOpen = (e) => sendBtn.removeAttribute("disabled");
const myMessage = (e) => (info.innerHTML += `${e.data}<br>`);
const myClose = (e) => sendBtn.setAttribute("disabled", "true");

const doConnect = () => {
  if (socket && socket.readyState === WebSocket.OPEN) return;
  socket = new WebSocket(`ws://${location.hostname}:3070/a`); // 開始連線
  socket.addEventListener("open", myOpen); // 連線時觸發
  socket.addEventListener("message", myMessage); // 有訊息傳入時
  socket.addEventListener("close", myClose); // 斷線時
};
```



10.4 簡易聊天室

```
import "./routes/ws-chat.js";
```

後端主程式 index.js

伺服器端程式 routes/ws-chat.js

```
import WebSocket, { WebSocketServer } from "ws";
const wsServer = new WebSocketServer({ port: 3071 });
const broadcastMsg = (msg) => {
  wsServer.clients.forEach((c) => {
    if (c.readyState === WebSocket.OPEN) {
      // 確認是有效連線，再送資料
      c.send(msg);
    }
  });
};
```



伺服器程式 routes/ws-chat.js

```
wsServer.on("connection", (ws, req) => {
  const dataObj = { name: "", ip: req.socket.remoteAddress }; // 記錄用戶資料
  ws.on("message", (message) => {
    const m = message.toString();
    let msg = ""; // 要回應的訊息
    if (dataObj.name) {
      msg = `${dataObj.name}: ${m}`;
    } else {
      // 沒有值，表示都還沒有傳任何訊息進來
      dataObj.name = m; // 第一次傳入的就是他的名字
      msg = `${dataObj.name}(${dataObj.ip}) 進入, 人數: ${wsServer.clients.size}`;
    }
    broadcastMsg(msg);
  });

  ws.on("close", () => {
    const msg = `${dataObj.name}(${dataObj.ip}) 離開, 人數: ${wsServer.clients.size}`;
    broadcastMsg(msg);
  });
});
export default wsServer;
```



前端頁面 A04-WebSocket-chat.html

```
<!-- A04-WebSocket-chat.html 的 HTML -->
<button onclick="doConnect()">連線</button>
<br />
<form onsubmit="event.preventDefault(); socket.send(myInput.value); ">
  <input type="text" id="myInput" />
  <button onclick="" disabled id="sendBtn">送出</button>
</form>
<br />
<div id="info"></div>
```



10.5 共用塗鴉板

```
import "./routes/ws-draw.js";
```

後端主程式 index.js

```
import WebSocket, { WebSocketServer } from "ws";
const wsServer = new WebSocketServer({ port: 3072 });
wsServer.on("connection", (ws, req) => {
  ws.on("message", (message) => {
    wsServer.clients.forEach((c) => {
      if (c.readyState === WebSocket.OPEN) {
        // 確認是有效連線，再送資料
        c.send(message.toString());
      }
    });
  });
});
export default wsServer;
```

伺服器端程式 routes/ws-draw.js



前端頁面 A05-WebSocket-draw.html (部份內容)

```
// {"x1":0,"y1":0,"x2":0,"y2":0,"c":"#","s":5}
const socket = new WebSocket(`ws://${location.hostname}:3072`);

socket.addEventListener("message", function (event) {
  let dataObj;
  try {
    dataObj = JSON.parse(event.data);
    ctx.beginPath();
    ctx.strokeStyle = dataObj.c;
    ctx.lineWidth = dataObj.s;
    ctx.moveTo(dataObj.x1, dataObj.y1);
    ctx.lineTo(dataObj.x2, dataObj.y2);
    ctx.stroke();
  } catch (ex) {}
});
```





11. Ajax files uploading

- 安裝 `multer` 套件
- 安裝 `uuid` 套件
- 建立 `/public/images` 資料夾，存放上傳的圖檔。



11.1 上傳圖檔的後端模組

routes/upload-img-module.js

```
import multer from "multer";
import { v4 } from "uuid";
// 篩選及副檔名對應物件
const extMap = {
  "image/jpeg": ".jpg", "image/png": ".png", "image/webp": ".webp",
};
const fileFilter = (req, file, cb) => {
  cb(null, !!extMap[file.mimetype]); // 對不到副檔名的檔案捨棄
};
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "public/images"); // 儲存檔案的資料夾位置
  },
  filename: (req, file, cb) => {
    cb(null, v4() + extMap[file.mimetype]); // 隨機決定檔名
  },
});
export default multer({fileFilter, storage});
```



11.2 處理上傳單一圖檔的表單

後端上傳的路由：index.js

```
import fs from "node:fs/promises";
import upload from "../routes/upload-img-module.js";

app.post("/uploads/profile", upload.single("avatar"), async (req, res) => {
  let data = {}; // 要存檔的資料
  try {
    const d = await fs.readFile("../public/profile.json");
    data = JSON.parse(d.toString());
  } catch (ex) {}
  data = { ...data, ...req.body }; // 變更資料

  if (req.file && req.file.originalname) { // 若有上傳檔案
    data.avatar = "/images/" + req.file.filename; // 儲存包含路徑
  }
  try {
    await fs.writeFile("../public/profile.json", JSON.stringify(data));
  } catch (ex) {
    return res.json({ success: false, data });
  }
  res.json({ success: true, data });
});
```



處理上傳單一圖檔的表單 - 前端

B01-upload-single.html

```
<form name="form1">
  <label for="user">姓名</label>
  <input type="text" name="user" id="user" /><br />
  <label for="avatar">大頭貼</label>
  <input type="file" name="avatar" id="avatar" accept="image/*" /><br />
  <img src="" alt="" id="avatar-img" width="300px" /><br />
  <label for="description">簡介</label><br />
  <textarea name="description" id="description" cols="50"></textarea><br />
  <button type="submit">上傳</button>
</form>
```



B01-upload-single.html

```
const avatarImg = document.querySelector("#avatar-img");
// 取得原有的內容
fetch("/profile.json")
  .then((r) => {
    return r.ok ? r.json() : {};
  })
  .then((obj) => {
    user.value = obj.user || "";
    avatarImg.src = obj.avatar || "";
    description.value = obj.description || "";
  })
  .catch((ex) => console.log("ex:", ex));
// 選擇檔案時預覽
avatar.addEventListener("change", (e) => {
  avatarImg.src = "";
  if (avatar.files[0]) {
    avatarImg.src = URL.createObjectURL(avatar.files[0]);
  }
});
```



B01-upload-single.html

```
// 以 AJAX 送出表單資料
document.form1.onsubmit = (e) => {
  e.preventDefault();
  const fd = new FormData(document.form1);
  fetch("/uploads/profile", {
    method: "POST",
    body: fd,
  })
    .then((r) => r.json())
    .then((obj) => alert("上傳完成"));
};
```



11.3 處理上傳多個圖檔

後端上傳的路由：index.js

```
app.post("/uploads/photos", upload.array("photos"), (req, res) => {  
  const output = [];  
  req.files.forEach((file) => {  
    output.push("/images/" + file.filename);  
  });  
  res.json(output);  
});
```



處理上傳多個圖檔

前端：B02-upload-multiple.html

```
<form action="" name="form1" onsubmit="return false;">
  <label for="photos">選了就上傳</label>
  <input
    type="file"
    name="photos"
    id="photos"
    accept="image/*"
    multiple
  /><br />
  <div id="img-container"></div>
  <textarea
    name="description"
    id="description"
    cols="80" rows="10"
    readonly
  ></textarea>
</form>
```



前端：B02-upload-multiple.html

```
const imgContainer = document.querySelector("#img-container");
photos.addEventListener("change", (event) => {
  const fd = new FormData(document.form1);
  fetch("/uploads/photos", {
    method: "POST",
    body: fd,
  })
    .then((r) => r.json())
    .then((ar) => {
      description.value = JSON.stringify(ar, null, 4);
      let str = "";
      ar.forEach((el) => {
        str += ``;
      });
      imgContainer.innerHTML = str;
    });
});
```





12. Web Worker

- 兩種型態：**Dedicated Worker**（針對某個頁面）和 **Shared Worker**（可在同源不同頁面分享運算的結果）。
- **Worker** 是以 JS 檔的方式存在，類型為 **Worker** 或 **SharedWorker**。
- **Workers** 雙方透過 **message** 事件接收資料，透過 **postMessage()** 方法傳送資料。
- 在 **Dedicated Worker** 的 JS 檔內，可定義 **onmessage()** 方法接收資料；直接呼叫 **postMessage()** 傳送資料。
- 在主程式，可定義 **dedicatedWorker.onmessage()** 方法接收資料；呼叫 **dedicatedWorker.postMessage()** 傳送資料。
- **Shared Worker** 的物件必須以其 **port** 子物件去啟動（**start**），及做資料的傳送及接收。
- 限制：不能操作 **DOM**。
- 參考資料：https://developer.mozilla.org/zh-TW/docs/Web/API/Web_Workers_API/Using_web_workers



12.1 Dedicated Worker

- 計算質數沒有使用 Worker：C01-no-dedicated-worker.html
- 計算質數使用 Worker：C02-with-dedicated-worker.html

```
<input type="text" placeholder="測試輸入欄" /><br />
<button onclick="start(event)">開始算質數</button>
<div id="info"></div>
<script>
  const worker = new Worker("js/dedicated-worker-prime-numbers.js");
  worker.onmessage = (event) => {
    let { primes, length, howLong } = event.data;
    info.innerHTML = `${length} 個, ${howLong} 毫秒`;
  };
  const start = (event) => {
    event.target.style.display = "none";
    worker.postMessage("start");
  };
</script>
```



▪ js/dedicated-worker-prime-numbers.js

```
const getPrimeNumbers = (num = 5e7) => {
  const startTime = new Date().getTime();
  const pn = [2]; // 存放質數的陣列

  // 略 . . .
  return {
    primes: pn,
    length: pn.length,
    howLong: new Date().getTime() - startTime,
  };
};

// this, self: DedicatedWorkerGlobalScope
self.onmessage = (event) => {
  if (event.data === "start") {
    postMessage(getPrimeNumbers());
  }
};
```



12.2 Shared Worker

- 只要有一個頁面使用 Shared Worker，Shared Worker 將會一直存在。
- Shared Worker 頁面：C03-shared-worker-1.html, C04-shared-worker-2.html

```
<input type="text" placeholder="測試輸入欄" /><br />
<div id="info"></div>
<script>
  const worker = new SharedWorker("js/shared-worker-prime-numbers.js");

  worker.port.onmessage = (event) => {
    const data = event.data;
    const pn = data.primeNumbers[data.primeNumbers.length - 1];
    info.innerHTML = `${data.action} -最大質數：${pn}， ${data.runningTime} msec`;
  };

  worker.port.postMessage("page-1");
</script>
```



- 使用 **Chrome** 測試 **Shared Worker** 時，請使用：<chrome://inspect/#workers>
- **js/shared-worker-prime-numbers.js**

```
const pn = [2]; // 存放質數的陣列
let runningTime;
const getPrimeNumbers = (num = 5e7) => {
  // 略 . . .
};
getPrimeNumbers();

self.onconnect = (event) => {
  const port = event.ports[0]; // MessagePort
  port.onmessage = (event) => {
    port.postMessage({
      primeNumbers: pn,
      runningTime: runningTime,
      action: event.data,
    });
  };
};
```





13. 啟動圖示

- Native app vs Web app
- Native app: 效能較好、啟動圖示、存取裝置、離線使用。
- Web app 如何追趕上 Native app ?
- PWA: Progressive Web App
- Web app manifest 為 JSON 文件，功能是將 Web app 安裝到設備的主畫面，其中包含名稱、圖示和描述等相關資訊。
- <https://developer.mozilla.org/zh-TW/docs/Web/Manifest>
- 由於 Chrome 和 Chrome for Android 的支援程度較好，故將以 Chrome 為主要環境。







13.1 Manifest 內容 MDN Manifest

```
{
  "name": "Web app 名稱",
  "short_name": "短名 · 啟動圖示上顯示的名稱",
  "start_url": "點擊圖示時開啟的網頁",
  "display": "常用 standalone",
  "background_color": "#fff",
  "description": "Web app 描述",
  "icons": [
    {
      "src": "images/touch/homescreen192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ],
  "related_applications": [
    {
      "platform": "play",
      "url": "https://play.google.com/store/apps/details?id=000"
    }
  ]
}
```



13.2 建立 App: Manifest

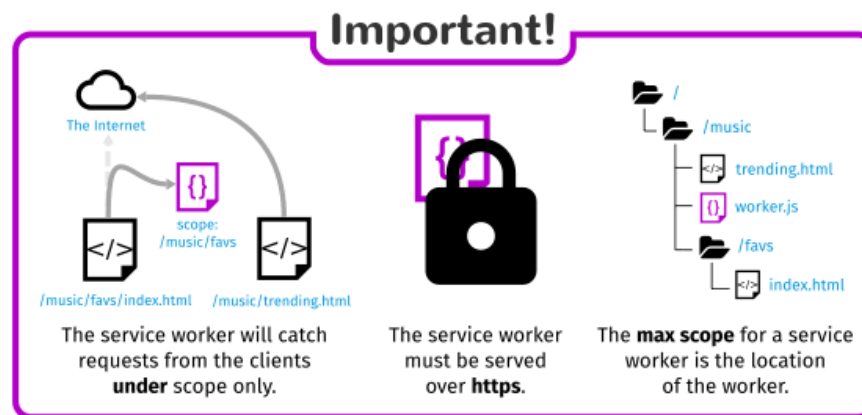
- 在 `/public` 資料夾內新增 `myApp` 資料夾。
- 將 `canvas-draw-app.html` 複製到資料夾內。
- 修改 `.html` 加入：
 - `<link rel="manifest" href="./manifest.json" />`
- 測試頁面，並使用開發人員工作，查看 **Application** 分頁

Application	App Manifest
 Manifest	manifest.json
 Service Workers	
 Clear storage	
Storage	
	Installability
	 No matching service worker detected. You may need to

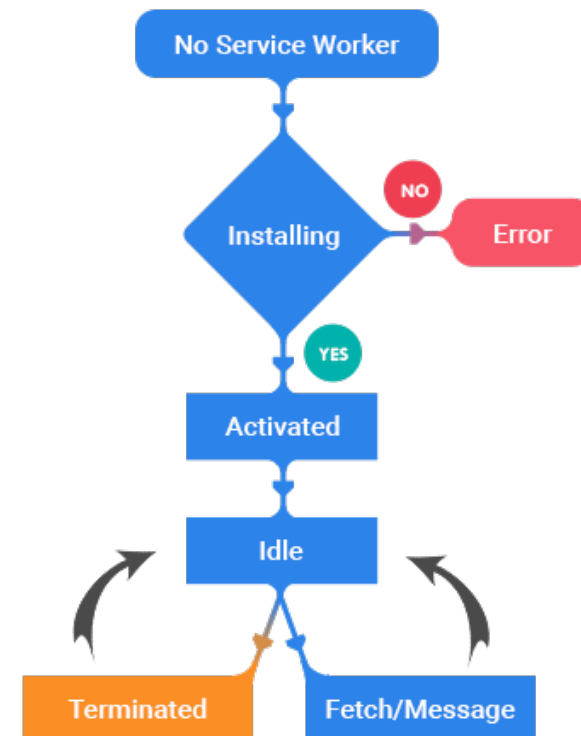


13.3 建立 App: Service Worker

- 使用 Service Worker: [Using Service Workers](#)



- <https://www.cronj.com/blog/browser-push-notifications-using-javascript/>



- 註冊 Service Worker

```
navigator.serviceWorker
  .register('service-worker.js')
  .then(reg=>{
    console.log('註冊成功：' + reg);
  })
  .catch(err=>{
    console.log('註冊失敗：' + err);
  })
```

- Service Worker 基本事件處理器

```
self.addEventListener('install', function(e) {});
self.addEventListener('activate', function(e) {});
self.addEventListener('fetch', function(e) {});
```

- 可以安裝但還不能離線使用





14. 離線使用

myApp/service-worker.js

```
let cacheName = "v1";
self.addEventListener("install", function (event) {
  console.log("install:", event);
  event.waitUntil(
    caches.open(cacheName).then((cache) => {
      return cache.addAll([
        "/myApp/canvas-draw-app.html",
        "/myApp/manifest.json",
        "/myApp/images/icons/icon-144x144.png",
      ]);
    })
  );
});
```

安裝時將檔案放入快取區，才能離線使用



```
self.addEventListener("fetch", function (event) {  
  console.log("fetch:", event); // 查看 chrome://inspect/#service-workers  
  event.respondWith(  
    caches.match(event.request).then((response) => {  
      return response || fetch(event.request);  
    })  
  );  
});
```

取得檔案資料時，先由快取區取得

```
self.addEventListener("activate", (event) => {  
  console.log("activate:", event);  
  event.waitUntil(  
    caches.keys().then((keyList) => {  
      return Promise.all(  
        keyList.map((key) => {  
          if (key !== cacheName) {  
            return caches.delete(key);  
          }  
        })  
      );  
    })  
  );  
});
```

啟動時可以移除舊版本的快取



15. Notifications

- Chrome 通知設定 <chrome://settings/content/notifications>
- 連結桌面的通知功能
- 測試網址 <https://tw.yahoo.com/>
- 使用 **Notification** 類型
- 檢視目前允許狀況 **Notification.permission**
 - **granted** : 允許
 - **denied** : 封鎖
 - **default** : 預設
- 要求權限 **Notification.requestPermission()**



```
const checkNotification = function () {  
  switch (Notification.permission) {  
    case "granted": // 允許  
      btn.style.display = "block";  
      break;  
    case "denied": // 封鎖  
      break;  
    default: // 未決定  
      // 要求權限  
      Notification.requestPermission(function () {  
        if (Notification.permission === "granted") {  
          btn.style.display = "block";  
        }  
      });  
  }  
};  
checkNotification();
```



F01-notification.html

```
const btn = document.querySelector("button");  
// 發送提示  
btn.onclick = function () {  
    let ntf = new Notification("MyTitle", {  
        body: "說明文字",  
        icon: "./myApp/images/icons/icon-144x144.png",  
    });  
};
```





16. Geolocation

- 使用 `navigator.geolocation.getCurrentPosition()`
- 由於屬於隱私權資料，所以會有詢問要求權限的對話框。
- 通常搭配 **Google Maps**
- 另一個地圖服務的選擇: [OpenStreetMaps](#)



取得經緯度座標: G01-geolocation.html

```
<pre id="info"></pre>
<script>
  const positionHandler = (position) => {
    const lat = position.coords.latitude; // 緯度
    const lng = position.coords.longitude; // 經度
    info.innerHTML = JSON.stringify({ lat, lng }, null, 4);
  };
  // 取得目前位置經緯度
  navigator.geolocation.getCurrentPosition(positionHandler, (error) => {
    console.log(error); // GeolocationPositionError
  });
</script>
```



座標搭配地圖: G02-geolocation-map.html

```
const positionHandler = (position) => {  
  const lat = position.coords.latitude; // 緯度  
  const lng = position.coords.longitude; // 經度  
  const latlng = new google.maps.LatLng(lat, lng);  
  const myOptions = {  
    zoom: 13,  
    center: latlng,  
    mapTypeId: google.maps.MapTypeId.ROADMAP,  
  };  
  const map = new google.maps.Map( //建立地圖  
    document.querySelector("#myMap"),  
    myOptions  
  );  
  //建立標記  
  const marker = new google.maps.Marker({  
    position: latlng,  
    title: "標記的名稱",  
  });  
  marker.setMap(map); // 把標記放入地圖  
};
```







Thank you for listening

