V、拖放操作 { Drag & Drop }

Drag & Drop

- 最早是IE5的正式標準
- 在使用拖放功能之前，必須先經由draggable屬性指定要拖曳的元素。
- 放下的地點沒有特別的限制，但拖放過程要透過事件來處理。
- 操作方法：
  1. 首先要指名開始處理拖曳**事件**
     -- 寫成拖曳事件的function
  2. 接著設定該物件是否可以被拖曳
     -- draggable設定為true | false
  3. 置放物件：default都是不接受該項操作，所以要先取消預設操作
     -- event.preventDefault()
  4. 拖曳時會產生的事件
     -- dragstart|drag|dragend

     置放物件會產生的事件

     -- dragenter|dragover|dragleave|drop

Drag & Drop：對於Video的控制

事件	說明
dragstart	拖曳開始
drag	拖曳中
dragend	拖曳結束
dragenter	拖進該範圍
dragover	停在裡面
dragleave	脫離該範圍
drop	放下

處理拖曳事件的物件為：**e.dataTransfer**	
dropEffect	指定置放時的游標形狀 copy|link|move|none
effectAllowed	指定拖曳時的游標形狀 copy|link|move|none copyLink|copyMove| linkMove|all
types	
clearData(type)	清除拖曳中的資料
setData(type,data)	開始拖曳時會呼叫此方法
getData(type)	置放時會呼叫此方法
files	
setDragImage(img,x,y)	指定要做為圖示的圖片
addElement(target)	指定非圖片的物件當作圖示

VI、檔案處理 { File API }

File API

- 何謂檔案？
- 讀取檔案資訊
- 讀取檔案內容
  1. 搭配<input type="file">選擇開啟的檔案

     常用事件：change

     文字檔案 | 圖檔 | 影片
  2. 直接從檔案總管將檔案拖曳到瀏覽器

     文字檔案 | 圖檔 | 影片

檔案

檔案的內建物件	
FileList	選取多重檔案 <input type="file" **multiple**>
File	
FileReader	讀取File和Blob的資訊
Blob	Binary Large Object 二進位檔案

讀取檔案資訊

name	檔案名稱
type	MIME類型，若無法對應則出現空白
size	檔案大小(單位：byte)
lastModifiedDate	最後修改日期時間
root.fullPath	

```
var file = document.getElementById('myFile').files[0];
……
var message = 'File Name : ' + file.name + '\n';
message += 'File Size : ' + file.size + '\n';
message += 'File Type : ' + file.type + '\n';
message += 'Last Modified : ' + file.lastModifiedDate;
```

讀取檔案內容：<input type="file">

fileReader = new FileReader();	
方法	
readAsText()	以純文字格式回傳結果
readAsDataURL()	以DataURL格式回傳結果，讀取圖檔和影片都使用此方法
abort()	中斷檔案讀取
屬性	
result	讀取到的內容
total	讀取的檔案大小
loaded	已讀到的檔案大小
事件	
loadstart | progress | load | abort | error | loadend	

讀取檔案內容：直接拖曳

使用物件：**e.dataTransfer**	
事件物件的方法	
preventDefault()	
拖曳事件的屬性	
draggable	[optional]
置放(drop)事件	
dragenter	拖進該範圍
dragover	停在裡面
drop	放下

VII、地理定位 { Geolocation }

Geolocation API

- Geolocation 是藉由多種類型的資料收集機制，識別使用者或運算裝置的地理位置。一般而言，大部分的地理定位服務利用網路路線規劃位置或利用內部 GPS 裝置，來判斷位置。

  Geolocation API 是否可使用需視裝置類型而定，部分瀏覽器/裝置支援此功能，其餘則不支援，因此，請記得並非所有的網路應用程式皆可使用地理定位。

- Geolocation 提供的方法：

  1. **getCurrentPosition()**--單次擷取目前的位置

  2. **watchPosition()**--能持續監控使用者位置，並定期確認是否有移動

  以上兩種方法都是以非同步方式確認使用者所在位置。

  ※若使用者第一次連到此網站，瀏覽器一定會強制限制固定交談窗，詢問是否願意公開位置。

  3. **clearWatch()**--清除監控

- getCurrentPosition() 與 watchPosition()共用功能，且接收參數也相同

  getCurrentPosition() 與 watchPosition()的第一個參數

  1．成功時的處理函數：

  物件：事件物件(position | e | evt | event)

  內含的兩個屬性：coords 物件和timestamp(時間戳記)

  coords 物件的屬性--第一層級

  ◦緯度：latitude (單位：double)

  ◦經度：longitude (單位： double)

  ◦準確度：accuracy (單位：公尺)

  coords 物件的屬性--第二層級

  ◦海拔高度：altitude (單位：double)

  ◦準確度： altitudeAccuracy (單位： double)

  ◦方向：heading (單位： double)

  ◦速度：speed (單位： double)

getCurrentPosition() 與 watchPosition()的第二個參數，非必要操作

2．錯誤時的處理函數：

物件：事件物件(error | e | evt | event)

屬性：

◦錯誤瑪：code (單位：unsigned short)

code的屬性值		
UNKNOWN_ERROR	0	未知錯誤
PERMISSION_DENIED	1	使用者不同意公開位置
POSITION_UNAVAILABLE	2	找不到使用者位置
TIMEOUT	3	逾時

◦錯誤訊息：message (單位： DOMString)

getCurrentPosition() 與 watchPosition()的第三個參數，非必要

3．設定地裡位置：

屬性：

- enableHighAccuracy (單位：bool , false by default)
  //是否啟用高精準度功能
- timeout (單位：毫秒, infinity/0 by default)
  //指定逾時的時間
- maximumAge (單位：毫秒, 0 by default)
  //可接受多久以前的資料

例如：若要取得高精準度功能，並設定10秒後逾時，且不使用舊的位置資料。

```
navigator.geolocation.getCurrentPosition(success,
    error, {
    enableHighAccuracy: true,
    timeout: 10000,
    maximumAge: 0
});
```

Google Maps API

參考網址：
https://developers.google.com/maps/documentation/javascript/?hl=zh-TW

寫在.html

```
<div id="message"
     style="width:1000px;height:800px;">
</div>
```

載入Google Maps API

```
<script src="http://maps.google.com/maps/api/js?sensor=false">
</script>
```

建立一個地圖物件

```
var map = new google.maps.Map(area,option);
        //area：網頁上呈現出的地圖區塊範圍
        //option：地圖資訊
```

地圖資訊	
zoom	地圖的比例，數字越大顯示的區域就越大
center	地圖的中心點，使用 google.maps.LatLng 物件來表示。 var latlng=new google.maps.LatLng(latitude,longitude);
mapTypeId	地圖形式，衛星圖或街道圖 google.maps.MapTypeId.ROADMAP 地圖樣式的常數： ROADMAP：平常看到的那樣 SATELLITE：地圖方塊 HYBRID：以上兩者的混合圖 TERRAIN：顯示實際起伏

目前位置：Marker

```
var marker = new google.maps.Marker({position:經緯度,
   map:地圖內容});
```

position	使用 google.maps.LatLng 物件來表示
map	使用 google.maps.Map 物件來表示
title	變換文字
icon	變換圖檔

```
var latlng=new google.maps.LatLng(lati, longi);
var map=new google.maps.Map(document.getElementById('message'),{
                zoom: 14,
                center: latlng,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        });
var image='../images/flag.png‘;
var marker=new google.maps.Marker({
        position: latlng,
        map: map,
        icon: image
});
```

地理位置編碼：Geocoder

```
var geocoder= new google.maps.Geocoder();
```

geocode(GeocoderRequest,callback(GeocoderResult,GeocoderStatus));	
GeocoderRequest	建立編碼的相關資訊
GeocoderResult	編碼的回傳值 results[0].geometry.location
GeocoderStatus	編碼的回傳狀態

```
function geocodeAddress(geocoder, resultsMap) {
  var address = document.getElementById('address').value;
  geocoder.geocode({'address': address}, function(results,status){
    if (status === google.maps.GeocoderStatus.OK) {
      resultsMap.setCenter(results[0].geometry.location);
      var marker = new google.maps.Marker({
        map: resultsMap,
        position: results[0].geometry.location
      });
    } else {
      alert('Geocode was not successful for the following reason:  '+ status);
    }
  });
}
```

VIII、資料儲存 { Web Storage }

Web Storage

- 網頁儲存區：cookie | web storage | IndexedDB
- 網頁儲存區是為了在client的磁碟上保存少量資料的儲存區。之前都是用cookie來處理。
- W3C將Web Storage定義為client端的 Javascript 環境中的一種實作的 Storage 介面的實體。
- 此介面提供了一組基於 key / value 的操作方法，隱藏了資料存續細節。也就是一個 storage 就是一個 hash table ，你只需要按照 hash table 的方式存取資料，client端的底層會幫忙處理資料存續的工作，完全不必知道資料如何存取資料庫。
- Web Storage 目前有兩種型態的儲存體：localStorage和sessionStorage
- 就 Javascript 來說，就是兩個全域變數：localStorage 和sessionStorage

·Local Storage(本機儲存區)

◦localStorage 的持續時間與存在範圍與 Cookie 類似。

它的持續時間由撰寫者指定，不會隨著瀏覽器關閉而自動終止。

它的存在範圍，同一個網站的所有網頁都會使用同一個 localStorage

◦把值放到儲存區中

```
localStorage.settings = 'ABC';
//或 localStorage["settings"] = 'ABC';
//或 localStorage.setItem("settings",'ABC');
```

◦取出值

```
var value = localStorage.settings ;
var value = localStorage["settings"];
var value = localStorage.getItem("settings");
```

◦刪除值

```
delete localStorage.settings ;
delete localStorage["settings"];
localStorage.removeItem("settings");
```

◦刪除所有資料

```
localStorage.clear();
```

- Session Storage(區段儲存區)
  - 在新分頁或新視窗中開啟連結時，client端程式將會為新開啟的視窗建立一個新的session，每一個session代表一組獨立的可用資源。
  - 而 sessionStorage 就是屬於session管理的資料項目。代表每個視窗都會有自己的 sessionStorage ；不同視窗的 sessionStorage 就是不同的內容。
  - 當網頁關閉時，表示此session結束了，所以此 sessionStorage 會被刪除。等到下次再開啟此網頁時，sessionStorage 的內容將會重新開始。
  - 因此，sessionStorage 只適合用於儲存暫時的資料。

  - 結論： sessionStorage物件是個暫存性的儲存區域，其中的資料以session為基礎，每個session有專屬的儲存區域，除了使用者對其進行異動之外，其間的資料會隨著session結束而結束。

- localStorage 和sessionStorage 的傳回值皆為Storage物件，用法完全相同

- 使用Javascript對WebStorage的控制(方法)

方法	傳回值	
`length`	`int`	回傳儲存的資料數目
`key(index)`	`string`	回傳index
`getItem(key)`	`value`	回傳index對應的資料
`setItem(key, value)`	`void`	儲存index的資料值
`removeItem(key)`	`void`	刪除key所對應的資料
`clear()`	`void`	刪除所有的資料

- 使用Javascript對WebStorage的控制(屬性)

屬性	
key	要變更的資料的key
oldValue	變更前的資料(複本)
newValue	變更後的資料(複本)
url	事件發生的來源
storageArea	要變更的儲存區的reference

/*customizeVideo.html*/

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <title>Video</title>
    <link rel="stylesheet" href="main.css">
    <script src="new.js"></script>
</head>
<body>
  <section id="skin">
    <video id="myMovie" width="640" height="360">
        <source src="videos/videoplayback.mp4">
    </video>
    <nav>
        <div id="buttons">
            <button id="playButton">Play</button>
        </div>
        <div id="defaultBar">
            <div id="progressBar"></div>
        </div>
        <div style="clear:both;"></div>
    </nav>
  </section>
</body>
</html>
```

```
// main.css
body{
    text-align:center;
}
section,nav{
    display:block;
}
#skin{
    width:700px;
    margin:10px auto;
    padding:5px;
    background:red;
    border: 3px dashed black;
    border-radius:10px;
}
nav{
    margin:5px auto;
}
#buttons{
    float:left;
    width:70px;
    height:22px;

}
#defaultBar{
    float:left;
    width:600px;
    height:16px;
    padding:4px;
    background:yellow;
    border: 2px solid black;
}
#progressBar{
    width:0px;
    height:16px;
    background:blue;
}
```

```
// new.js
function doFirst(){
    barSize=600;
    myMovie = document.getElementById('myMovie');
    playButton = document.getElementById('playButton');
    defaultBar = document.getElementById('defaultBar');
    progressBar = document.getElementById('progressBar');

    playButton.addEventListener('click',playOrPause,false);
    defaultBar.addEventListener('click',clickedBar,false);

}

function playOrPause(){
    if(!myMovie.paused && !myMovie.ended){
        myMovie.pause();
        playButton.innerHTML = 'Play';
        window.clearInterval(updateBar);
    }else{
        myMovie.play();
        playButton.innerHTML = 'Pause';
        updateBar = setInterval(update,500);
    }
}

function update(){
    if(!myMovie.ended){
        var size = parseInt(barSize/myMovie.duration*
myMovie.currentTime);
        progressBar.style.width = size + 'px';
    }else{
        progressBar.style.width =  '0px';
        playButton.innerHTML = 'Play';
        window.clearInterval(updateBar);
    }
}
```

```
function clickedBar(e){
    if(!myMovie.paused && !myMovie.ended){
        var mouseX = e.clientX - bar.offsetLeft;
        var newTime = mouseX * myMovie.duration / barSize;
        myMovie.currentTime = newTime;
        progressBar.style.width = mouseX + 'px';
    }
}
window.addEventListener("load",doFirst,false);
```