API Drag and Drop

1. Arrastrar y soltar en la web

Arrastrar un elemento desde un lugar y luego soltarlo en otro es algo que hacemos todo el tiempo en aplicaciones de escritorio, pero ni siquiera imaginamos hacerlo en la web. Esto no es debido a que las aplicaciones web son diferentes sino porque desarrolladores nunca contaron con una tecnología estándar disponible para ofrecer esta herramienta.

Ahora, gracias a la **API Drag and Drop**, introducida por la especificación HTML5, finalmente tenemos la oportunidad de crear software para la web que se comportará exactamente como las aplicaciones de escritorio que usamos desde siempre.

Nuevos eventos

Uno de los más importantes aspectos de esta API es un conjunto de **siete nuevos eventos** introducidos para informar sobre cada una de las **situaciones involucradas en el proceso**. Algunos de estos eventos son **disparados** por la **fuente** (el elemento que es arrastrado) y otros son disparados por el **destino** (el elemento en el cual el elemento arrastrado será soltado).

Eventos que dispara el elemento origen

Cuando el usuario realiza una operación de arrastrar y soltar (el que es arrastrado) genera 3 eventos:

- **dragstart** Este evento es disparado en el momento en el que **el arrastre comienza**. Los datos asociados con el elemento origen son definidos en este momento en el sistema.
- **drag** Este evento es similar al evento `mousemove`, excepto que será disparado durante una **operación de arrastre por el elemento origen**.
- **dragend** Cuando la operación de **arrastrar y soltar finaliza** (sea la operación exitosa o no) este evento es disparado por el elemento origen.

Eventos que dispara el elemento destino

Durante la operación de arrastrar y soltar, el elemento destino (donde el origen será soltado) genera los siguientes 4 eventos:

- **dragenter** Cuando **el puntero del ratón entra dentro del área** ocupada por los posibles elementos **destino** durante una operación de arrastrar y soltar, este evento es disparado.

- **dragover** Este evento es similar al evento `mousemove`, excepto que es disparado durante una operación de arrastre por posibles elementos destino, cuando el **ratón se mueve dentro del área** de dichos elementos **destino**.

- **drop** Cuando el **elemento origen es soltado** durante una operación de arrastrar y soltar, este evento es disparado por el elemento destino.

- **dragleave** Este evento es disparado cuando **el ratón sale del área ocupada por un elemento durante una operación de arrastrar y soltar**. Este evento es generalmente usado junto con `dragenter` para mostrar una ayuda visual al usuario que le permita identificar el elemento destino (donde soltar).

Antes de trabajar con esta nueva herramienta, existe un aspecto importante que debemos considerar. **Los navegadores realizan acciones por defecto durante una operación de arrastrar y soltar**.

Para obtener el resultado que queremos, necesitamos **prevenir** en algunas ocasiones este **comportamiento por defecto** y **personalizar las reacciones del navegador**. Para algunos eventos, como `dragenter`, `dragover` y `drop`, la prevención es **necesaria**, incluso cuando una acción personalizada ya fue especificada.

Veamos cómo debemos proceder usando un ejemplo simple.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
    Arrastre y suelte la imagen aquí
  </section>
  <section id="cajaimagenes">
     <img id="imagen"src="http://www.minkbooks.com/content/monster1.gif">
  </section>
</body>
</html>
```

El documento HTML incluye un elemento `<section>` identificado como `cajasoltar` y una imagen. El elemento `<section>` será usado como elemento destino y la imagen será el elemento a arrastrar. También incluimos dos archivos para estilos CSS y el código javascript que se hará cargo de la operación.

Fichero **dragdrop.css**:

```
#cajasoltar{
  float: left;
  width: 500px;
  height: 300px;
  margin: 10px;
  border: 1px solid #999999;
}
#cajaimagenes{
  float: left;
  width: 320px;
  margin: 10px;
  border: 1px solid #999999;
}
#cajaimagenes > img{
  float: left;
  padding: 5px;
}
```

Este código CSS simplemente otorga estilos a las cajas que nos servirán para identificar el elemento a arrastrar y el destino.

Fichero **dragdrop.js**:

```
function iniciar(){
  origen1=document.getElementById('imagen');

  origen1.addEventListener('dragstart', arrastrado, false);

  destino=document.getElementById('cajasoltar');

  destino.addEventListener('dragenter', function(e){
                          e.preventDefault(); }, false);

  destino.addEventListener('dragover', function(e){
                          e.preventDefault(); }, false);

  destino.addEventListener('drop', soltado, false);
}

function arrastrado(e){
  var codigo='<img src="'+origen1.getAttribute('src')+'">';
  e.dataTransfer.setData('Text', codigo);
}
```

```
function soltado(e){
  e.preventDefault();
  destino.innerHTML=e.dataTransfer.getData('Text');
}

window.addEventListener('load', iniciar, false);
```

Existen algunos atributos que podemos usar en los elementos HTML para configurar el proceso de una operación arrastrar y soltar, pero básicamente **todo puede ser hecho desde código Javascript**. En el Listado javascript del fichero **dragdrop.js** presentamos tres funciones:

- La función `iniciar()` agrega las **escuchas** para los **eventos** necesarios en esta operación.
- La función `arrastrado()` **genera** la **información transmitida** por este proceso.
- La función `soltado()` **recibe** la **información** que es **transmitida** por este proceso.

Para que una operación arrastrar y soltar se realice normalmente, debemos preparar la información que será compartida entre el elemento origen y el elemento destino. Para lograr esto, una **escucha para el evento** `dragstart` fue agregada. La escucha llama a la función `arrastrado()` cuando el evento es disparado y la **información** a ser **compartida** es **preparada** en esta función usando `setData()`.

La operación `soltar` **no es normalmente permitida en la mayoría de los elementos de un documento por defecto**. Por este motivo, para hacer esta operación disponible en nuestro elemento destino, debemos **prevenir el comportamiento por defecto del navegador**. Esto fue hecho agregando una escucha para los eventos `dragenter` y `dragover` y ejecutando el método `preventDefault()` cuando son disparados.

Finalmente, una escucha para el evento `drop` fue agregada para llamar a la función `soltado()` que **recibirá y procesará los datos enviados** por el elemento origen.

Para responder a los eventos `dragenter` y `dragover` usamos una **función anónima** y llamamos en su interior al método `preventDefault()` que **cancela el comportamiento por defecto del navegador**. La variable e fue enviada para referenciar al evento dentro de la función.

Cuando el elemento **origen comienza a ser arrastrado**, el evento `dragstart` es **disparado** y la función `arrastrado()` es **llamada**. En esta función obtenemos el valor del atributo `src` del elemento que está siendo arrastrado y **declaramos los datos que serán transferidos** usando el método `setData()` del objeto `dataTransfer`.

Desde el otro lado, cuando un **elemento es soltado dentro del elemento destino**,

el evento **drop** es disparado y la función **soltado()** es llamada. Esta función **modifica el contenido del elemento destino** con la información obtenida por el método **getData()**.

Los **navegadores** también realizan **acciones por defecto** cuando estos eventos son disparados (por ejemplo, a**brir un enlace o actualizar la ventana para mostrar la imagen que fue soltada**) por lo que debemos prevenir este comportamiento usando el método **preventDefault()**, como ya hicimos para otros eventos anteriormente.

dataTransfer

Este es el **objeto** que **contendrá la información en una operación arrastrar y soltar**. El objeto **dataTransfer** tiene varios métodos y propiedades asociados. Ya utilizamos los métodos **setData()** y **getData()** en nuestro ejemplo anterior. Junto con **clearData()**, estos son los métodos a cargo de la información que es transferida:

- **setData(tipo, dato)** Este método es usado para **declarar los datos a ser enviados y su tipo**. El método puede recibir tipos de datos regulares (como **text/plain**, **text/html** o **text/uri-list**), tipos de datos especiales (como **URL** o **Text**) o incluso tipos de datos personalizados. Un **método setData()** debe ser **llamado** por **cada tipo de datos** que queremos enviar en la misma operación.

- **getData(tipo)** Este método **retorna los datos enviados por el origen**, pero solo del **tipo especificado**.

- **clearData()** Este método **elimina los datos** del tipo especificado.

En la función **arrastrado()**, creamos un pequeño código HTML que incluye el valor del atributo **src** del elemento que comenzó a ser arrastrado, grabamos este código en la variable **codigo** y luego **enviamos esta variable como el dato a ser transferido** usando el método **setData()**. Debido a que **estamos enviando texto**, declaramos el tipo de dato como **Text**.

Podríamos haber usado un tipo de datos más apropiado en nuestro ejemplo, como **text/html** o incluso un tipo personalizado, pero varios navegadores solo admiten un número limitado de tipos en este momento, por lo que el **tipo Text** hace a nuestra pequeña aplicación **más compatible** y la deja lista para ser ejecutada.

Cuando **recuperamos** los **datos** en la función **soltado()** usando el método **getData()**, tenemos que **especificar** el **tipo** de datos a ser leído. Esto es debido a que **diferentes clases** de datos pueden ser enviados por el **mismo elemento**.

Por ejemplo, una imagen podría enviar la imagen misma, la URL y un texto describiendo la imagen. Toda esta información puede ser enviada usando **varias declaraciones** de **setData()** con diferentes tipos de valores y luego recuperada por **getData()** especificando los mismo tipos.

El objeto `dataTransfer` tiene algunos **métodos y propiedades** más que a veces podrían resultar útil para nuestras aplicaciones:

- **setDragImage(elemento, x, y)** Algunos navegadores muestran una **imagen en miniatura junto al puntero del ratón** que representa al elemento que está siendo arrastrado. Este método es usado para **personalizar esa imagen** y seleccionar la posición la posición en la que será mostrada relativa al puntero del ratón. Esta posición es determinada por los atributos x e y.

- **types** Esta propiedad retorna un `array` conteniendo los **tipos de datos que fueron declarados durante el evento** `dragstart` (por el código o el navegador). Podemos grabar este `array` en una variable (`lista=dataTransfer.types`) y luego leerlo con un bucle `for`.

- **files** Esta propiedad retorna un `array` conteniendo información acerca de los **archivos que están siendo arrastrados**.

- **dropEffect** Esta propiedad retorna el **tipo de operación actualmente seleccionada**. Los posibles **valores** son `none`, `copy`, `link` y `move`.

- **effectAllowed** Esta propiedad **retorna los tipos de operaciones que están permitidas**. Puede ser usada para cambiar las operaciones permitidas. Los posibles **valores** son: `none`, `copy`, `copyLink`, `copyMove`, `link`, `linkMove`, `move`, `all` y `uninitialized`.

Aplicaremos algunos de estos métodos y propiedades en los siguientes ejemplos.

dragenter, dragleave y dragend

Nada fue hecho aún con el evento `dragenter`. Solo cancelamos el comportamiento por defecto de los navegadores cuando este evento es disparado para prevenir efectos no deseados. Y **tampoco aprovechamos** los eventos `dragleave` y `dragend`. Estos son eventos importantes que nos permitirán ayudar al usuario cuando se encuentra arrastrando objetos por la pantalla.

```
function iniciar(){
  origen1=document.getElementById('imagen');
  origen1.addEventListener('dragstart', arrastrado, false);
  origen1.addEventListener('dragend', finalizado, false);

  soltar=document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', entrando, false);
  soltar.addEventListener('dragleave', saliendo, false);
  soltar.addEventListener('dragover', function(e){
                        e.preventDefault(); }, false);
```

```
  soltar.addEventListener('drop', soltado, false);
}
function entrando(e){
  e.preventDefault();
  soltar.style.background='rgba(0,150,0,.2)';
}

function saliendo(e){
  e.preventDefault();
  soltar.style.background='#FFFFFF';
}
function finalizado(e){
  elemento=e.target;
  elemento.style.visibility='hidden';
}
function arrastrado(e){
  var codigo='<img src="'+origen1.getAttribute('src')+'">';
  e.dataTransfer.setData('Text', codigo);
}
function soltado(e){
  e.preventDefault();
  soltar.style.background='#FFFFFF';
  soltar.innerHTML=e.dataTransfer.getData('Text');
}
window.addEventListener('load', iniciar, false);
```

En este ejemplo, agregamos dos funciones para el elemento destino y una para el elemento origen. Las funciones `entrando()` y `saliendo()` **cambiarán el color de fondo del elemento destino** cada vez que el puntero del ratón esté arrastrando **un objeto y entre o salga del área ocupada** por este elemento (estas acciones disparan los eventos `dragenter` y `dragleave`). Además, la función `finalizado()` será llamada por la escucha del evento `dragend` cuando el **objeto arrastrado es soltado**. Este evento o la función misma **no controlan si el proceso fue exitoso o no**. Este control lo **deberemos hacer nosotros** en el código.

Gracias a los eventos y funciones agregadas, cada vez que el ratón arrastra un objeto y entra en el área del elemento destino, este elemento se volverá verde, y **cuando el objeto es soltado** la **imagen original es borrada** de la pantalla. Estos cambios visibles no están afectando el proceso de arrastrar y soltar, pero sí están ofreciendo una guía clara para el usuario durante la operación.

Para **prevenir acciones por defecto del navegador**, tenemos que usar el método `preventDefault()` en cada función, incluso cuando acciones personalizadas fueron declaradas.

Seleccionando un origen válido

No existe ningún método específico para detectar si el elemento origen es válido o no. No podemos confiar en la información retornada por el método `getData()` porque incluso cuando podemos recuperar solo los datos del tipo especificado, otras fuentes podrían originar el mismo tipo y proveer datos que no esperábamos. Hay una propiedad del objeto `dataTransfer` llamada `types` que retorna un `array` con la lista de tipos configurados durante el evento `dragstart`, pero también es inútil para propósitos de validación.

Por esta razón, las **técnicas** para seleccionar y validar los datos transferidos en una operación arrastrar y soltar son **variadas**, y pueden ser tan **simples** o **complejos** como necesitemos.

```
 <!DOCTYPE html>
<html lang="es">
<head>
 <title>Drag and Drop</title>
 <link rel="stylesheet" href="dragdrop.css">
 <script src="dragdrop.js"></script>
</head>
<body>
 <section id="cajasoltar">
  Arrastre y suelte las imágenes aquí
 </section>
 <section id="cajaimagenes">
  <img id="imagen1"
       src="http://www.minkbooks.com/content/monster1.gif">
  <img id="imagen2"
       src="http://www.minkbooks.com/content/monster2.gif">
  <img id="imagen3"
       src="http://www.minkbooks.com/content/monster3.gif">
  <img id="imagen4"
       src="http://www.minkbooks.com/content/monster4.gif">
 </section>
</body>
</html>
```

Usando esta **nueva plantilla HTML** vamos a **filtrar los elementos a ser soltados dentro del elemento destino** controlando el atributo id de la imagen. El siguiente código Javascript indicará **qué imagen puede ser soltada y cuál no**:

```
function iniciar(){
 var imagenes=document.querySelectorAll('#cajaimagenes > img');
 for(var i=0; i<imagenes.length; i++){
  imagenes[i].addEventListener('dragstart', arrastrado, false);
 }
```

```
 soltar=document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', function(e){
                              e.preventDefault(); }, false);
  soltar.addEventListener('dragover', function(e){
                              e.preventDefault(); }, false);
  soltar.addEventListener('drop', soltado, false);
}
function arrastrado(e){
  elemento=e.target;
  e.dataTransfer.setData('Text', elemento.getAttribute('id'));
}
function soltado(e){
  e.preventDefault();
  var id=e.dataTransfer.getData('Text');
  if(id!="imagen4"){
    var src=document.getElementById(id).src;
    soltar.innerHTML='<img src="'+src+'">';
  }else{
    soltar.innerHTML='la imagen no es admitida';
  }
}
window.addEventListener('load', iniciar, false);
```

En este código usamos el método **querySelectorAll()** para **agregar una escucha para el evento dragstart a cada imagen dentro del elemento cajaimagenes**, enviando el valor del atributo **id** con **setData()** cada vez que una imagen es arrastrada, y controlando el valor de id en la función **soltado()** para **evitar** que el **usuario arrastre y suelte la imagen con el atributo** igual a "**imagen4**" (el mensaje "la imagen no es admitida" es mostrado dentro del elemento destino cuando el usuario intenta arrastrar y soltar esta imagen en particular).

Este es, por supuesto, un **filtro** extremadamente **sencillo**. Puede usar el método **querySelectorAll()** en la función **soltado()** para controlar que la imagen recibida es una de las que se encuentran dentro del elemento **cajaimagenes**, por ejemplo, o usar propiedades del objeto **dataTransfer** (como **types** o **files**), pero es siempre un proceso personalizado. En otras palabras, deberemos hacernos cargo nosotros mismos de realizar este control.

setDragImage()

Cambiar la imagen en miniatura que es mostrada junto al puntero del ratón en una operación arrastrar y soltar puede parecer inútil, pero en ocasiones nos evitará dolores de cabeza. El método **setDragImage()** no solo nos permite cambiar la imagen sino también recibe dos atributos, **x** e **y**, para especificar la **posición de esta imagen relativa al puntero**.

Algunos navegadores generan una imagen en **miniatura por defecto a**

partir del objeto original que es arrastrado, pero su posición relativa al puntero del ratón es determinada por la posición del puntero cuando el proceso comienza. El método `setDragImage()` nos permite declarar una posición específica que será la misma para cada operación arrastrar y soltar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
     <canvas id="lienzo" width="500" height="300"></canvas>
  </section>
  <section id="cajaimagenes">
    <img id="imagen1"
         src="http://www.minkbooks.com/content/monster1.gif">
    <img id="imagen2"
         src="http://www.minkbooks.com/content/monster2.gif">
    <img id="imagen3"
         src="http://www.minkbooks.com/content/monster3.gif">
    <img id="imagen4"
         src="http://www.minkbooks.com/content/monster4.gif">
  </section>
</body>
</html>
```

Veremos el uso del método `setDragImage()` utilizando un elemento `<canvas>` como el elemento destino.

```
  function iniciar(){
  var imagenes=document.querySelectorAll('#cajaimagenes > img');
  for(var i=0; i<imagenes.length; i++){
    imagenes[i].addEventListener('dragstart', arrastrado, false);
    imagenes[i].addEventListener('dragend', finalizado, false);
  }
  soltar=document.getElementById('lienzo');
  lienzo=soltar.getContext('2d');

 soltar.addEventListener('dragenter', function(e){
                          e.preventDefault(); }, false);
  soltar.addEventListener('dragover', function(e){
                          e.preventDefault(); }, false);
  soltar.addEventListener('drop', soltado, false);
}
function finalizado(e){
```

```
  elemento=e.target;
  elemento.style.visibility='hidden';
}
function arrastrado(e){
  elemento=e.target;
  e.dataTransfer.setData('Text', elemento.getAttribute('id'));
  e.dataTransfer.setDragImage(e.target, 0, 0);
}
function soltado(e){
  e.preventDefault();
  var id=e.dataTransfer.getData('Text');
  var elemento=document.getElementById(id);

 var posx=e.pageX-soltar.offsetLeft;
 var posy=e.pageY-soltar.offsetTop;

 lienzo.drawImage(elemento,posx,posy);
}
window.addEventListener('load', iniciar, false);
```

Probablemente, con este ejemplo, nos estemos **acercando** a lo que sería una aplicación de la vida real. El código javascript del listado anterior controlará tres diferentes aspectos del proceso. Cuando la **imagen es arrastrada**, la función `arrastrado()` es llamada y en su interior **una imagen en miniatura es generada** con el método `setDragImage()`. El código también crea el contexto para trabajar con el lienzo y dibuja la imagen soltada usando el método `drawImage()`, de la API `canvas`. Al final de todo el proceso la imagen original es ocultada usando la función `finalizado()`.

Para la **imagen miniatura personalizada usamos el mismo elemento que está siendo arrastrado**, pero declaramos la posición relativa al puntero del ratón como 0,0. Gracias a esto ahora sabremos siempre cual es exactamente la **ubicación de la imagen miniatura**. Aprovechamos este dato importante dentro de la función `soltado()`.

Calculamos dónde el objeto es soltado dentro del lienzo con las coordenadas `e.pageX` y `e.pageY`, junto con los valores de `offset` y dibujamos la imagen en ese lugar preciso. Si prueba este ejemplo en navegadores que ya aceptan el método `setDragImage()`, verá que la imagen es dibujada en el lienzo exactamente en la posición de la imagen miniatura que acompaña al puntero del ratón, haciendo fácil para el usuario seleccionar el lugar adecuado donde soltarla.

En este ejemplo el **evento** `dragend` se usa para **ocultar la imagen original cuando la operación termina**. Este evento es disparado por el elemento origen cuando una operación de arrastre finaliza, **incluso cuando no fue exitosa**. En nuestro ejemplo la imagen será ocultada en ambos casos, éxito o fracaso. Se deben crear los controles adecuados para actuar solo en caso de éxito.

Archivos

Posiblemente la característica más interesante de la API Drag and Drop es la habilidad de **trabajar con archivos**. La **API** no está solo disponible dentro del documento, sino también **integrada con el sistema**, permitiendo a los usuarios **arrastrar elementos desde el navegador hacia otras aplicaciones y viceversa**. Y normalmente los elementos más requeridos desde aplicaciones externas son **archivos**.

Como vimos anteriormente, existe una propiedad especial en el objeto `dataTransfer` que retornará un `array` conteniendo la **lista de archivos que están siendo arrastrados**. Podemos usar esta información para construir complejos códigos que trabajan con archivos o subirlos a un servidor.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Drag and Drop</title>
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
    Arrastre y suelte archivos en este espacio
  </section>
</body>
</html>
```

El documento HTML anterior genera simplemente una **caja para soltar los archivos arrastrados**. Los archivos serán arrastrados **desde una aplicación externa** (por ejemplo, el **Explorador de Archivos de Windows**). Los datos provenientes de los archivos serán procesados por el siguiente código:

```
  function iniciar(){
  soltar=document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', function(e){
                          e.preventDefault(); }, false);
  soltar.addEventListener('dragover', function(e){
                          e.preventDefault(); }, false);
  soltar.addEventListener('drop', soltado, false);
}
function soltado(e){
  e.preventDefault();
  var archivos=e.dataTransfer.files;
  var lista='';
  for(var f=0;f<archivos.length;f++){
    lista+='Archivo: '+archivos[f].name+' '+archivos[f].size+'<br>';
  }
  soltar.innerHTML=lista;
}
window.addEventListener('load', iniciar, false);
```

La **información** retornada por la propiedad `files` del objeto `dataTransfer` puede ser **grabada en una variable** y luego **leída por un bucle `for`**. En el código del javascript anterior, **solo mostramos el nombre y el tamaño del archivo** en el elemento destino usando las propiedades `name` y `size`. Para aprovechar esta información y construir aplicaciones más complejas, necesitaremos **recurrir a otras APIs** y técnicas de programación.