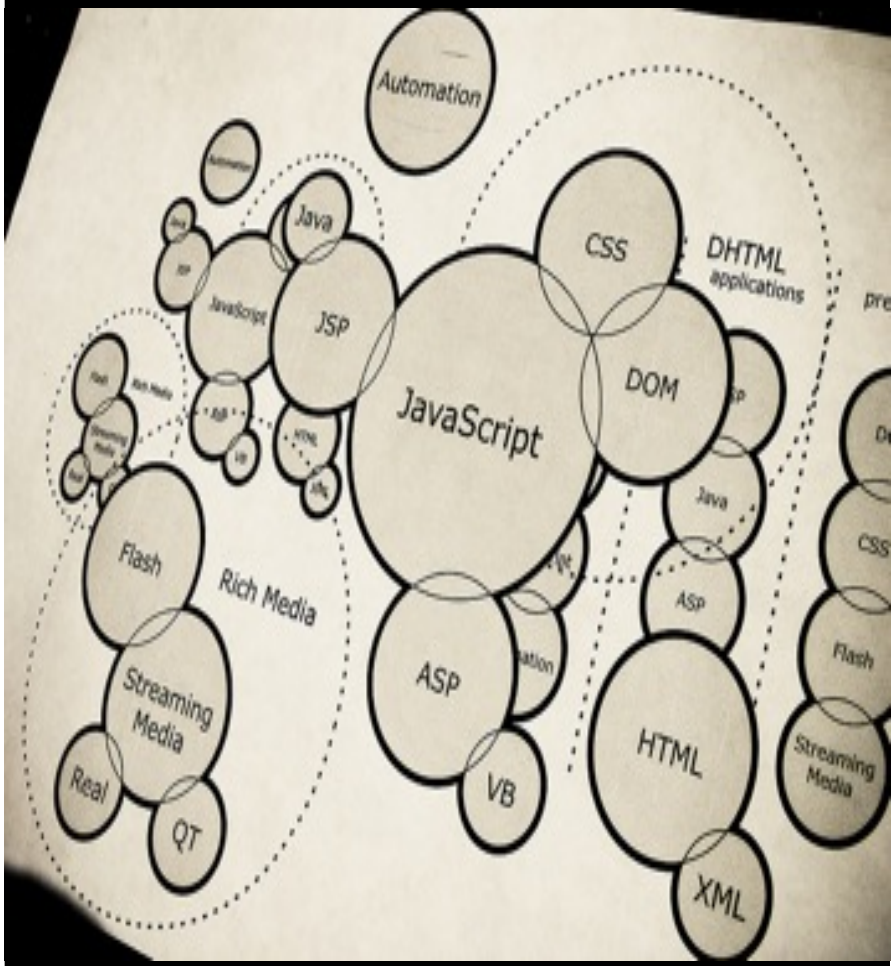
Javascript, parte IV

El DOM (Modelo de Objetos del Documento)



http://www.flickr.com/photos/jm3/362074444413

Javascript parte V

El DOM

1. Manejo del DOM

Conceptos básicos sobre el API DOM y la forma de manipular el HTML en dicho API

lexto

El DOM

- Como ya hemos visto, por cada etiqueta HTML existe un objeto Javascript equivalente
- Es decir, el navegador mantiene en memoria un **modelo orientado a objetos del documento** que refleja la estructura del HTML
  - El DOM es un **árbol**. Cada "componente" del HTML es un **nodo**
  - Los cambios en el DOM se reflejan en "tiempo real" en el HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo de DOM</title>
</head>
<body>
<!-- es un ejemplo un poco simple -->
<p style="color:red">Bienvenidos al <b>DOM</b></p>
</body>
</html>
```

Seleccionar nodos

- Por `id`

```
var noticias = document.getElementById("noticias")
```

- Por etiqueta

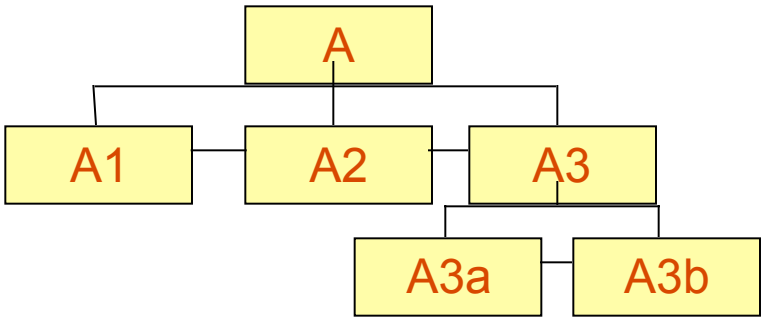
```
//Reducir el tamaño de todas las imágenes a la mitad
var imags = document.getElementsByTagName("img");
for(var i=0; i<imags.length; i++) {
      imags[i].width /= 2;
      imags[i].height /= 2;
}
```

- Usando selectores CSS

```
//Obtener el 1er nodo que cumple la condición
var primero = document.querySelector(".destacado");
//Obtenerlos todos
var nodos = document.querySelectorAll(".destacado");
//Cambiamos la clase. Nótese que es “className”, no “class”
for (var i=0; i<nodos.length; i++) {
    nodos[i].className = "normal";
}
//selectores un poco más complicados
var camposTexto = document.querySelectorAll('input[type="text"]');
var filasPares = document. querySelectorAll("tr:nth-child(2n)")
```

Navegar por el árbol

- Cada nodo tiene una serie de propiedades que reflejan el "parentesco" con otros, algunas de las cuales son
  - **childNodes**: array con los nodos hijos
  - **firstChild**: primer nodo hijo, **lastChild**: último nodo hijo
  - **parentNode**: nodo padre
  - **nextSibling**: siguiente hermano (nodo al mismo nivel) **prevSibling**: hermano anterior.



A.firstChild = A1
A.lastChild =A3
A.childNodes.length = 3
A.childNodes[0] = A1
A.childNodes[1] = A2
A.lastChild.firstChild = A3a
A3b.parentNode.parentNode = A

A1.nextSibling = A2
A3.prevSibling = A2
A3.nextSibling = null

Modificar el HTML

- `innerHTML`: propiedad que nos permite leer/modificar el código HTML que hay dentro de una etiqueta
- No es estándar en HTML4, pero sí en HTML5
- No se puede (debe) usar para editar tablas. Para eso existen métodos alternativos (ver p.ej. `http://msdn.microsoft.com/en-us/library/ms532998(v=vs.85).aspx`)

```
<input type="button" value="Pon texto" onclick="ponTexto()"/>
<div id="texto"></div>
<script type="text/javascript">
  function ponTexto() {
    var mensaje = prompt("Dame un texto y lo haré un párrafo")
    var miDiv = document.getElementById("texto")
    miDiv.innerHTML += "<p>" + mensaje + "</p>"
  }
</script>
```

API DOM estándar

- Muy **potente**, pero también algo **tedioso** de utilizar
- A diferencia de `innerHTML`, que permite manipular directamente el texto, se **manipulan los nodos** del árbol, lo que a su vez "modifica" el HTML

```
<input type="button" value="Añadir párrafo" id="boton"/>
<div id="texto"></div>
<script type="text/javascript">
 document.getElementById("boton").onclick = function() {
   var texto = prompt("Introduce un texto para convertirlo en párrafo");
   /* Nótese que la etiqueta <p> es un nodo, y el texto que contiene es OTRO
      nodo, de tipo textNode,  hijo del nodo <p> */
   var p = document.createElement("P");
   var nodoTexto = document.createTextNode(texto);
   p.appendChild(nodoTexto);
   document.body.appendChild(p);
 };
</script>
```

http://jsbin.com/Uwoduf/1/watch?html,js,output

Javascript, parte V

El DOM

4. DOM Templating

Insertar HTML de manera sencilla

Templates vs concatenar HTML

- Hasta ahora cuando queríamos generar en Javascript HTML con una parte "fija" y una "variable" lo hacíamos concatenando cadenas

```
var html = '<a href="' + miURL + '">' + miTexto + '</a>'
```

- Es mucho más "limpio" usar templates, como hacemos en el servidor con Mustache
  - Hay implementación Mustache para Javascript (¡varias, incluso!)

```
var tmpl = Mustache.compile('<a href="{{miURL}}"> {{miTexto}} </a>')
//tmpl se convierte en una función que al pasarle los datos nos //
//devuelve el HTML
var html = tmpl({"miURL":"http://www.ua.es", "miTexto":"UA"})
```

Dónde guardar los templates

- Si los fragmentos de HTML son muy grandes, sigue siendo engorroso pasárselos como argumento directamente a “compile”. Un “truco” muy usado es almacenarlos como un `<script>` de tipo no interpretable por el navegador

```
<script id="miTmpl" type="text/template">
  <h1>Bienvenido a {{sitio}}</h1>
  <h2>Querido {{nombre}}, .... </h2>
  <div> .... </div>
</script>
```

```
var tmpl = Mustache.compile(document.getElementById("miTmpl")).innerHTML
```

Algunas referencias

- Libros electrónicos accesibles solo **desde dentro de la UA**, en Safari O'Reilly (http://proquestcombo.safaribooksonline.com)
- **JavaScript: The Definitive Guide**, Sixth Edition, *David Flanagan*
  - *Referencia exhaustiva de Javascript, incluyendo DOM*
- **Pro Javascript Techniques**, *John Resig*
  - *DOM en Cap 5*
- **Javascript Cookbook**, *Shelley Powers*
  - *Poca teoría, básicamente ejemplos de código*
  - *DOM en Caps 11 y 12*