Scripting en el cliente: Javascript

Tecnologías Web

Funciones (avanzado)

- Funciones anidadas
  - A partir de JavaScript 1.2 y ECMAScript v3, se pueden definir funciones anidadas
  - Aunque podamos definir funciones anidadas, las funciones no se pueden definir en cualquier parte (e.j no podemos definirlas dentro de una instrucción condicional o dentro de un bucle)

Ejemplo de función anidada

```
function hipotenusa(a, b) {
  function cuadrado(x) { return x*x; }
  return Math.sqrt(cuadrado(a) + cuadrado(b));
}
```

- Constructor `Function()` (JavaScript 1.1+)
  - Sirve para crear funciones dinámicas
  - Se utiliza junto al operador `new`.

Declaración dinámica de función y uso

```
var f = new Function("x", "y", "return x*y;");
alert(f(10,2));
```

Declaración estática equivalente

```
function f(x, y) { return x*y; }
```

  - `Function()` recibe como argumentos un número de cadenas, donde la última de ellas es el cuerpo (instrucciones JavaScript) de la función
  - Hay que darse cuenta que al utilizar `Function()` no le estamos pasando por ninguna parte el nombre de la función.

Funciones (avanzado)

- Funciones como literales (JavaScript 1.2+)
  - Podemos construir las funciones de una manera adicional

Declaración función literal

```
var f = function(x) { return x*x; };
```

  - Este tipo de funciones son particularmente útiles cuando solo las vamos a utilizar una vez.

Usos

```
a.sort(function(a,b){return a-b;}); // Definición y paso a otra función
var tensquared = (function(x) {return x*x;})(10);  // Definición y llamada
```

  - La declaración literal tiene como ventaja la posibilidad de incluir el código JavaScript como tal y no utilizando una cadena de texto.
    - Por el contrario, la función literal es estática
  - Adicionalmente, utilizando funciones literales solo se realiza el proceso de compilación 1 vez y por el contrario con `Function()` se compila la función cada vez que se declara un valor.

- Funciones como datos
  - Las funciones pueden utilizarse como si fueran valores de variables

Uso de función como dato

```
function square(x) { return x*x; }
var a = square(4);  // a contiene el número 16
var b = square;     // b referencia a la función square
var c = b(5);       // c contiene el número 25
```

  - Las funciones también pueden asignarse propiedades de objetos en vez de a variables globales

Asignación a propiedades de objetos

```
var o = new Object;
o.square = new Function("x", "return x*x");
y = o.square(16);
```

  - Las funciones no necesitan necesariamente nombres

```
var a = new Array(3);
a[0] = function(x) { return x*x; }  // Función literal
a[1] = 20;
a[2] = a[0](a[1]);
```

- Objeto `arguments`
  - Está disponible dentro del cuerpo de una función
    - Este objeto es similar a un array (pero no lo es realmente), ya que permite obtener sus valores pasados a la función utilizando un índice.
  - Aunque el número de argumentos de una función en JavaScript es fijo, podemos pasarle cualquier número de parámetros al invocarla.
  - Podemos acceder a
    - Los argumentos utilizando la sintaxis `arguments[]`
    - El número de argumentos se accede mediante la propiedad `length`.

Ejemplo función máximo

```
function max( ){
  var m = Number.NEGATIVE_INFINITY;

  for(var i = 0; i < arguments.length; i++)
    if (arguments[i] > m) m = arguments[i];
  return m;
}
```

¿Qué es un objeto?

- JavaScript es un lenguaje basado en objetos
  - Un objeto en JavaScript es simplemente un conjunto de valores con nombre.
    - Estos valores son habitualmente denominados propiedades
    - Ej: `imagen.anchura; imagen.altura`
  - Estas propiedades pueden contener valores de cualquier tipo de datos, en particular: arrays, funciones y otros objetos
  - Cuando una propiedad contiene un valor de tipo función, esta propiedad recibe el nombre de método.
- Podemos diferenciar varios tipos de objetos
  - Objetos del navegador
    - Permiten manejar aspectos de la presentación de las páginas en el navegador
  - Objetos del lenguaje
  - Objetos definidos por el usuario

Creando objetos y propiedades

- La creación de un objeto se realiza invocando a una función especial denominada "constructor" utilizando el operador `new`

```
var o = new Object( );
var now = new Date( );
```

  - Una vez creado, podemos establecer las propiedades según queramos

```
var point = new Object( );
point.x = 2.3;
point.y = -1.2;
```

  - Para crear una propiedad simplemente se asigna un valor a la propiedad del objeto
  - Para eliminar la propiedad se utiliza el operador `delete`

```
delete point.x = 2.3;
```

- El constructor `Object()` permite crear un objeto sin ninguna propiedad

Acceso a propiedades y métodos de los objetos

- Para acceder a una propiedad de un objeto
  - `objeto.propiedad`
  - Algunas propiedades son de sólo lectura; otras se pueden modificar

```
imagen.anchura
imagen.altura = 50;
```

  - Cuando la propiedad no existe el acceso a dicha propiedad devuelve el valor `undefined`

- Un objeto también es considerado como un array asociativo

```
imagen["anchura"]; imagen["altura"]
```

- Para llamar a un método de un objeto:
  - `objeto.metodo(parametros)`
  - Existen métodos como operadores, e.j: +

- Bucle for ... in
  - Se utiliza para recorrer las propiedades disponibles de un objeto.

```
for (var aux in objeto ) // Sentencias
```

Ejemplo de creación, acceso y borrado de propiedades

Ejemplo de Objeto

```
<html>
  <body>
    <script type="text/javascript">
       var o = new Object();
       document.write("<h1>Objeto NO inicializado</h1>");
       document.write("<p>"+o.propiedad+"</p>");
       o.propiedad = "Valor";
       document.write("<h1>Objeto inicializado</h1>");
       for( p in o )
         document.write("<p>"+p+": "+o[p]+"</p>");
       o.propiedad = undefined;
       document.write("<h1>Asignación de valor indefinido</h1>");
       for( p in o )
         document.write("<p>"+p+": "+o[p]+"</p>");
       delete o.propiedad;
       document.write("<h1>Una vez eliminada la propiedad</h1>");
       for( p in o )
         document.write("<p>"+p+": "+o[p]+"</p>");
    </script>
  </body>
</html>
```

Constructores

- Ya hemos visto como crear objetos utilizando el operador `new` y una función y un constructor predefinido como `Object()`, `Date()`
- También es posible crear nuestros propios constructores de clase
- Un constructor en JavaScript es una función con dos características especiales:
  - Es invocada utilizando el operador `new`
  - Se le pasa una referencia al objeto (vacío) que se acaba de crear mediante la palabra reservada `this`.

Ejemplo de constructor de Rectángulo y su uso

```
function Rectangle( h, w){
  this.height = h;
  this.width = w;
}

var r1 = new Rectangle(10,40);
```

Métodos

- Los métodos son propiedades cuyos valores son funciones
  - Estas funciones pueden acceder a las propiedades del objeto a través de la palabra reservada `this`.

Declaración de métodos en el constructor

```
function Rectangle_area( ) { return this.width * this.height; }
function Rectangle_perimeter( ) { return 2*this.width + 2*this.height; }
function Rectangle_set_size(w,h) { this.width = w; this.height = h; }
function Rectangle_enlarge( ) { this.width *= 2; this.height *= 2; }
function Rectangle_shrink( ) { this.width /= 2; this.height /= 2; }

function Rectangle(w, h) {
    this.width = w;
    this.height = h;

    // Define methods for the object.
    this.area = Rectangle_area;
    this.perimeter = Rectangle_perimeter;
    this.set_size = Rectangle_set_size;
    this.enlarge = Rectangle_enlarge;
    this.shrink = Rectangle_shrink;
}
```

Clases y Objetos del lenguaje

- "Objeto Global"
  - Objeto que está siempre disponible y que contiene un conjunto de funciones útiles.

- Object
- Array
- Date
- Boolean
- Number
- Math
- String
- RegExp

Clases del Lenguaje

- navigator

- "Objeto global"
  - `eval("cadena")` → Permite evaluar una cadena de texto como si fuera un programa JavaScript
  - `parseInt("cadena", base)` → Convierte una cadena de texto a un número entero, si falla la conversión devuelve `NaN`.
    - Si no se especifica ninguna base se asume la base 10.
    - Si la cadena comienza por `"0x"` se asume base 16.
  - `parseFloat("cadena")` → Convierte una cadena de texto a un número en punto flotante, si falla devuelve `NaN`.
  - `isNan(arg)` → Devuelve `true` si su argumento no es un número (**N**ot **a** **N**umber), `false` en otro caso.
  - `isFinite(arg)` → Devuelve true si su argumento no es `NaN`, `+Infinity`, `-Infinity`.
  - `decodeURI('URICodificada')` → Descodifica una URI codificada.
  - `encodeURI('URI')` → Codifica una URI.

Los objetos del lenguaje (cont.)

- Arrays

Creación de Array

```
var array1 = new Array(); // Creación simple
var array2 = new Array(10); // Creación con tamaño
var array3 = new Array("cad1", "cad2"); // Creación e inicialización
```

Acceso y propiedades

```
for(var indice = 0; indice < array.length; i++ ){
  alert(array[i]);
}
```

- Métodos
  - `concat(valor1, ...)` → Añade nuevos valores y devuelve el nuevo array (el array sobre el que se llama no se modifica)
  - `join(separador)` → Crea una representación en forma de cadena de texto de los elementos que contiene el array, separándolos con 'separador'
  - `pop()` → Elimina el último elemento.
  - `push(valor1, ...)` → Añade nuevos valores al array.
  - `reverse()` → Invierte el orden de los elementos de un array.

Los objetos del lenguaje (cont.)

- `slice(posIni, posFin)` → Devuelve un nuevo array con los elementos entre la posición `'posIni'` y `'posFin'`.
  - Si no se especifica `'posFin'` se asume que es hasta el final.
- `sort(funcion)` → Ordena los elementos del array.
  - Si no se especifica ninguna función se ordena por orden alfabético (previa conversión de los valores almacenados a cadena)
  - Si se especifica una función ésta tiene que aceptar dos argumentos (e.j. `elem1` y `elem2`) y devolver un número.
    - -1 → `elem1` es menor que `elem2`
    - 0 → `elem1` es igual a `elem2`
    - 1 → `elem1` es mayor que `elem2`.

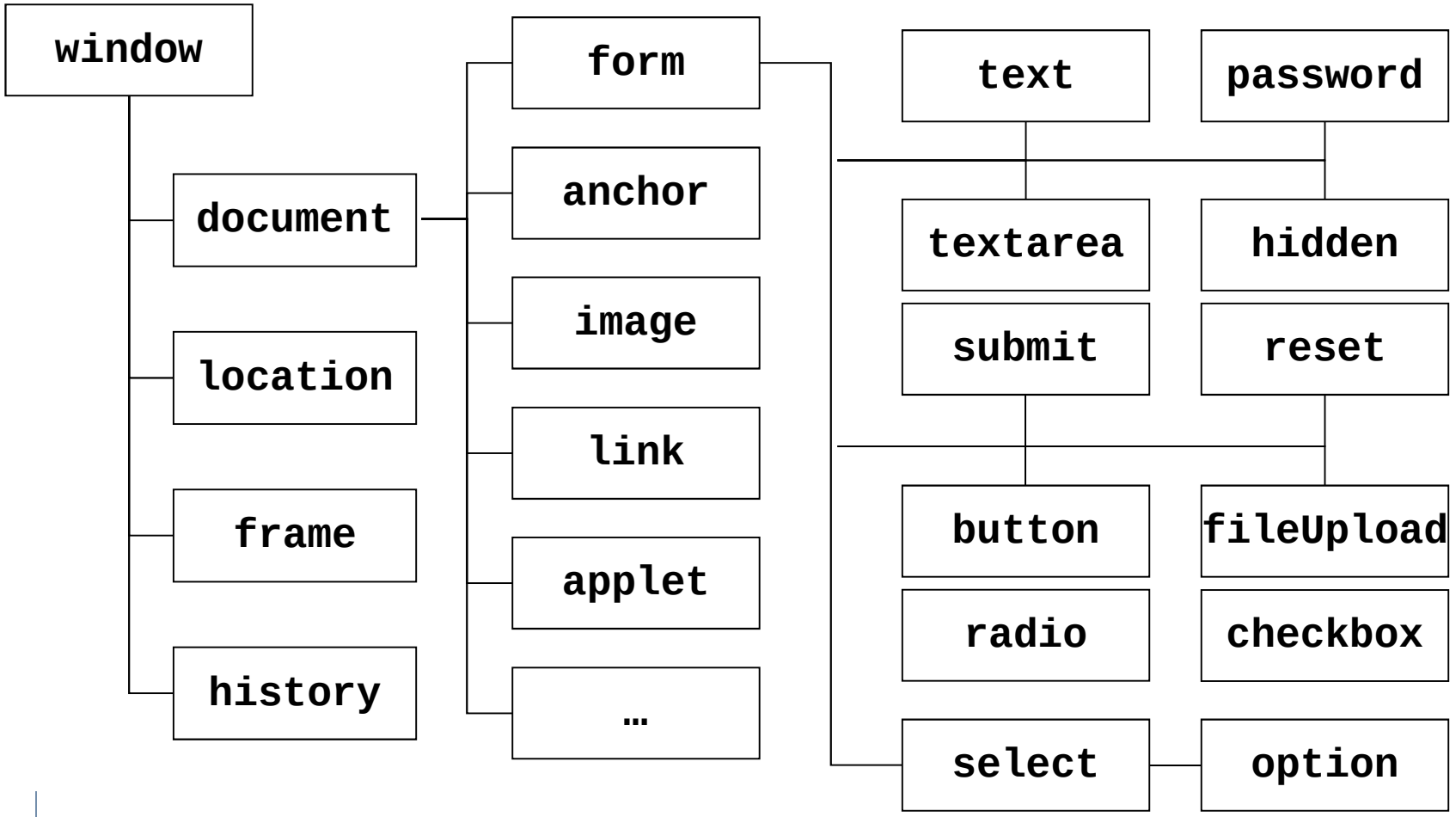
Ejemplo de ordenación

```
<script type="text/javascript" language="JavaScript"><!--
  function ord_creciente(elem1, elem2) { return (elem1-elem2); }
  var miArray = new Array(3, 5, 6, 2, -1);
  miArray.sort(ord_creciente);
//--></script>
```

Objetos del navegador

- En JavaScript existen clases que se corresponden con aspectos del navegador, del documento HTML que está mostrando, y de los elementos de dicho documento
- Algunos objetos siempre están presentes:
  - `window` (clase Window) → ventana del navegador
  - `document` (clase Document) → página cargada en la ventana
- Otros se crean según se van encontrando sus elementos correspondientes en el HTML
- Algunos objetos tienen propiedades que son otros objetos o colecciones de objetos
  - Todos los objetos del navegador están contenidos en propiedades de `window`, o en propiedades de objetos contenidos en `window` (recursivamente)
  - Se puede hablar de una jerarquía de objetos del navegador

Jerarquías de objetos del navegador



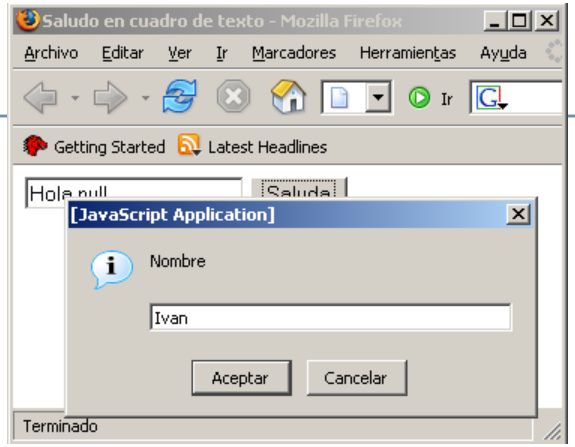
Acceso a los objetos del navegador

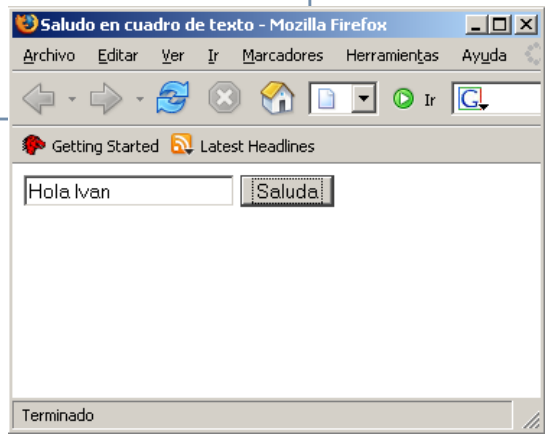
- En JavaScript hay varias maneras de acceder a un objeto del navegador
  - ¡Cuidado! En algunos casos, un método de acceso que funcione en un navegador podría no funcionar en otro
- La manera más sencilla es hacerlo a través de su nombre o identificador (propiedades *name* e *id* de HTML)
  - El objeto será una propiedad de su padre en la jerarquía; el nombre de la propiedad será el valor de su atributo *name* o *id*
  - Algunos navegadores pueden no soportar el atributo *id*; sin embargo la nueva especificación de HTML recomienda utilizarlo en lugar de *name*
    - Para evitar problemas de compatibilidad, lo mejor es especificar los dos con el mismo valor

Ejemplo

Ejemplo de evento y trabajo con formularios

```
<html>
  <head><title>Saludo en cuadro de texto</title>
  <script type="text/javascript" language="JavaScript"><!--
      function saluda() {
        var nombre;
        nombre=window.prompt("Nombre");
        window.document.formulario.texto.value="Hola "+nombre;
      }
    //--></script>
  </head>
  <body>
    <form id="formulario" name="formulario">
      <input type="text" name="texto" id="texto"/>
      <input type="button" onclick="saluda()" value="Saluda"/>
    </form>
  </body>
</html>
```





Acceso a los objetos del navegador (cont.)

- Algunos objetos tienen propiedades que son colecciones de los objetos que contienen
  - Esto proporciona otra manera de acceder a los objetos
- Para acceder a un elemento de una colección se puede especificar el índice en número o el nombre o identificador
  - `window.document.forms[0].elements[0]`
  - `window.document.forms["formulario"].elements["texto"]`
- También se puede acceder al objeto actual mediante this
  - En el código de un manejador de evento se refiere al objeto en el que se produce el evento.
  - Fuera de un manejador de evento se refiere a `window`

Colecciones de objetos

- Objeto `document`
  - `frames[ ]`
  - `applets[ ]`
  - `anchors[ ]`
  - `links[ ]`
  - `images[ ]`
  - `forms[ ]`
- Objeto `form`
  - `elements[ ]` → controles del formulario
- Objeto `select`
  - `options[ ]` → opciones del menú

Eventos

- Un evento es un suceso que le ocurre a un objeto
- En JavaScript, se puede asociar a un evento un código que se ejecutará cuando se produzca el evento
  - Esto se hace mediante los manejadores de eventos
  - Un manejador de eventos es similar a una propiedad, cuyo valor es el código asociado al evento
  - El nombre de los manejadores es *onEvento*, donde Evento es el nombre del evento concreto
  - La manera más habitual de establecer los manejadores de eventos es hacerlo en el código HTML
    - La mayoría de elementos HTML están asociados a un objeto de JavaScript
    - Mediante el atributo *onEvento* se establece el código asociado al manejador del evento

Ejemplo de Eventos

Asociación mediante atributo y mediante propiedad

```
<html>
  <head><title>Ejemplo de asociación de eventos</title>
  <script language="JavaScript" type="text/javascript"><!--
    function paginaCargada(){
      alert("La pagina ha sido cargada");
    }
    window.onload=paginaCargada;
  //--></script>
  </head>
  <body>
    <form>
      <input type="button" value="Saluda" onClick="window.alert('Hola')" />
    </form>
  </body>
</html>
```

Asociación mediante Propiedad

Asociación mediante atributo

Algunos Eventos

- Algunos ejemplos de evento
  - `onLoad` → Carga de una página o de una imagen.
  - `onUnLoad` → Se descarga la página.
  - `onFocus` → Cuando el foco se dirige a un objeto.
  - `onBlur` → Cuando el foco se dirige a otro objeto
  - onChange → Cuando el campo de un formulario cambia de valor.
  - `onSubmit` → Cuando se hace click sobre el botón de envío de un formulario.
  - `onClick` → Cuando se hace click sobre un objeto.
  - `onDblClick` → Cuando se hace doble click sobre un objeto.
  - `onKeyDown` → Cuando se pulsa un tecla
  - `onKeyUp` → Cuando se suelta una tecla
  - `onKeyPress` → Cuando se pulsa y suelta una tecla
  - `onMouseDown` → Cuando se pulsa un botón del ratón.
  - `onMouseUp` → Cuando se suelta el botón del ratón
  - `onMouseOver` → Cuando se mueve el puntero del ratón sobre un objeto.
  - `onMouseOut` → Cuando el puntero del ratón abandona un objeto

Interacción con el usuario

- Visualización de mensajes
  - `alert(mensaje);`
  - mensaje → Mensaje a mostrar
- Formulación de preguntas
  - `confirm(pregunta);`
  - pregunta → Mensaje que se desea mostrar al alumno.
  - La función devuelve true si se ha pulsado Aceptar o false si ha pulsado en Cancelar
- Obtención de datos
  - `prompt(pregunta, valorPredeterminado);`
  - pregunta → Mensaje a mostrar
  - valorPredeterminado → (Opcional) Valor predeterminado que aparecerá en el cuadro de texto.
  - La función devuelve el valor introducido por el usuario.

Referencias

- http://www.dannyg.com/
  - Autor de JavaScript Bible
- http://www.mozilla.org/docs/dom/domref/
  - Referencia de Objetos para Mozilla / Firefox
- http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml_node_entry.asp
  - Sección HTML / DHTML Reference, referencia para IExplorer
- http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/ (Sección Document Object Model HTML)
- http://www.w3.org/2002/11/DOM-Level-2-HTML-Results/
  - Nos da información acerca de qué métodos de la especificación de DOM están disponibles en los navegadores

Críticas, dudas y sugerencias...

Federico Peinado
www.federicopeinado.es