Desarrollo Plataforma Web/Programación Web
Ing. Tomás Eduardo Urbina

Lenguajes Scripts

Introducción a los scripts

Un *script* en el lado del cliente es un programa que puede acompañar a un documento HTML o que puede estar incluido en él. El programa se ejecuta en la máquina del cliente cuando se carga el documento, o en algún otro instante, como por ejemplo cuando se activa un vínculo. El soporte de scripts de HTML es independiente del lenguaje de scripts.

Los scripts ofrecen a los autores la posibilidad de extender los documentos HTML de maneras activas e interactivas. Por ejemplo:

- Pueden evaluarse los scripts a medida que se carga el documento para modificar los contenidos del documento dinámicamente.
- Los scripts pueden acompañar a un formulario para procesar los datos a medida que éstos se introducen. Los diseñadores pueden rellenar dinámicamente partes de un formulario según los valores de los otros campos. También pueden asegurarse de que los datos introducidos concuerden con rangos de valores predeterminados, de que los campos sean consistentes entre sí, etc.
- Los scripts pueden ser llamados por eventos que afecten al documento, como la carga, la descarga, el movimiento del foco sobre los elementos, los movimientos del ratón, etc.
- Los scripts pueden ser vinculados a controles de formulario (p.ej., botones) para producir elementos gráficos para la interfaz del usuario.

Hay dos tipos de scripts que los autores pueden asociar a un documento HTML:

- Aquellos que se ejecutan una sola vez cuando el agente de usuario carga el documento. Los scripts que aparecen dentro de un elemento `SCRIPT` se ejecutan cuando el elemento es cargado. Para los agentes de usuario que no puedan o que no vayan a ejecutar scripts, los autores pueden incluir contenido alternativo por medio del elemento `NOSCRIPT`.
- Aquellos que son ejecutados cada vez que ocurre un determinado evento. Estos scripts pueden ser asignados a varios elementos por medio de los atributos de eventos intrínsecos.

Nota. *Esta especificación incluye información más detallada sobre scripts en la sección sobre macros de scripts.*

Diseño de documentos para agentes de usuario que soporten scripts

Las siguientes secciones tratan sobre cuestiones que afectan a los agentes de usuario que soportan scripts.

18.2.1 El elemento `SCRIPT`

```
<!ELEMENT SCRIPT - - %Script;          -- sentencias de script -->
<!ATTLIST SCRIPT
  charset     %Charset;      #IMPLIED  -- codif. de caracteres del
recurso vinculado--
  type        %ContentType;  #REQUIRED -- tipo de contenido del
lenguaje de scripts --
  src         %URI;          #IMPLIED  -- URI del script externo --
  defer       (defer)        #IMPLIED  -- El AU puede retrasar la
ejecución  --
  >
```

Etiqueta inicial: ***obligatoria****, Etiqueta final:* ***obligatoria***

Definiciones de atributos

`src` = *uri* [CT]
: Este atributo especifica la localización de un script externo.

`type` = *tipo de contenido* [CI]
: Este atributo especifica el lenguaje de scripts de los contenidos del elemento y prevalece sobre el lenguaje de scripts por defecto. El lenguaje de scripts se especifica como un tipo de contenido (p.ej., "text/javascript"). Los autores deben proporcionar un valor para este atributo. No hay valor por defecto para este atributo.

`language` = *cdata* [CI]
: **Desaprobado.** Este atributo especifica el lenguaje de scripts de los contenidos de este elemento. Su valor es un identificador del lenguaje, pero debido a que estos identificadores no son estándar, este atributo ha sido desaprobado en favor de `type`.

`defer` [CI]
: Si está establecido, este atributo booleano indica al agente de usuario que el script no va a generar ningún contenido en el documento (p.ej., en javascript, cuando no hubiera ningún "document.write") y por lo tanto el agente de usuario puede seguir analizando y representando.

Atributos definidos en otros lugares

- `charset` (codificaciones de caracteres)

El elemento `SCRIPT` coloca un script dentro de un documento. Este elemento puede aparecer cualquier número de veces en el `HEAD` o en el `BODY` de un documento HTML.

El script puede estar definido dentro de los contenidos del elemento `SCRIPT` o en un fichero externo. Si el atributo `src` no está establecido, los agentes de usuario deben interpretar que los contenidos del elemento son el script. Si `src` tiene un valor URI, los agentes de usuario no deben tener en cuenta los contenidos del elemento y deben obtener el script mediante el URI. Obsérvese que el atributo `charset` se refiere a la codificación de caracteres del script designado por el atributo `src`; no afecta al contenido del elemento `SCRIPT`.

Los scripts son evaluados por *motores de scripts*, con los cuales deben poder comunicarse los agentes de usuario.

La sintaxis de los datos de scripts depende del lenguaje de scripts.

Especificación del lenguaje de scripts

Al no estar ligado el HTML a un lenguaje de scripts específico, los autores de los documentos deben decir explícitamente a los agentes de usuario el lenguaje de cada script. Esto puede hacerse o bien mediante una declaración por defecto o bien mediante una declaración local.

El lenguaje de scripts por defecto

Los autores deberían especificar el lenguaje de scripts por defecto de todos los scripts de un documento incluyendo la siguiente declaración `META` en el `HEAD`:

```
<META http-equiv="Content-Script-Type" content="type">
```

donde "type" es un tipo de contenido que se refiere al lenguaje de scripts. Como ejemplos de este valor tenemos "text/tcl", "text/javascript", "text/vbscript".

En ausencia de una declaración `META`, el valor por defecto puede ser establecido con un encabezado HTTP "Content-Script-Type".

```
    Content-Script-Type: type
```

donde "type" es nuevamente un tipo de contenido que se refiere al lenguaje de scripts.

Los agentes de usuario deberían determinar el lenguaje de scripts por defecto de un documento de acuerdo con los siguiente pasos (ordenados de prioridad más alta a más baja):

1. Si alguna declaración `META` especifica el "Content-Script-Type", la última de ellas en el flujo de caracteres determina el lenguaje de scripts por defecto.
2. En caso contrario, si algún encabezado HTTP especifica el "Content-Script-Type", el último de ellos en el flujo de caracteres determina el lenguaje de scripts por defecto.

Los documentos que no especifiquen información relativa al lenguaje de scripts por defecto y que contengan elementos que especifiquen un script de evento intrínseco son incorrectos. Los agentes de usuario aún pueden intentar interpretar scripts especificados incorrectamente, pero no se requiere que lo hagan. Las herramientas de creación deberían generar información sobre el lenguaje de scripts por defecto para ayudar a que los autores eviten la creación de documentos incorrectos.

Declaración local del lenguaje de un script

Se debe especificar el atributo `type` de todos los elementos `SCRIPT` de un documento. El valor del atributo `type` de un elemento `SCRIPT` prevalece sobre el languaje de scripts por defecto de ese elemento.

En este ejemplo, declaramos que el lenguaje de scripts por defecto es "text/tcl". Incluimos un SCRIPT en la cabecera, cuyo script se localiza en un fichero externo y que está en el lenguaje de scripts "text/vbscript". También incluimos un SCRIPT en el cuerpo, que contiene su propio script escrito en "text/javascript".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>Un documento con SCRIPT</TITLE>
<META http-equiv="Content-Script-Type" content="text/tcl">
<SCRIPT type="text/vbscript" src="http://algunsitio.com/progs/vbcalc">
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
...algo en JavaScript...
</SCRIPT>
</BODY>
</HTML>
```

Referencias a elementos HTML desde un script

Cada lenguaje de scripts tiene sus propias convenciones para referirse a objetos HTML desde dentro del script. Esta especificación no define un mecanismo estándar para referirse a objetos HTML.

Sin embargo, los scripts deberían hacer referencia a un elemento de acuerdo con su nombre asignado. Los motores de scripts deberían seguir las siguientes reglas de precedencia cuando identifiquen un elemento: un atributo name prevalece sobre un atributo id si ambos están establecidos. En caso contrario, se puede usar uno u otro.

Eventos intrínsecos

Definiciones de atributos

onload = *script* [CT]
: El evento onload ocurre cuando el agente de usuario finaliza la carga de una ventana o de todos los marcos de un FRAMESET. Este atributo puede utilizarse con los elementos BODY y FRAMESET.

onunload = *script* [CT]
: El evento onunload ocurre cuando el agente de usuario elimina un documento de una ventana o marco. Este atributo puede utilizarse con los elementos BODY y FRAMESET.

onclick = *script* [CT]

El evento `onclick` ocurre cuando se hace clic con el dispositivo apuntador sobre un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`ondblclick` = *script* [CT]

El evento `ondblclick` ocurre cuando se hace doble clic con el dispositivo apuntador sobre un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onmousedown` = *script* [CT]

El evento `onmousedown` ocurre cuando el botón del dispositivo apuntador se pulsa cuando está encima de un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onmouseup` = *script* [CT]

El evento `onmouseup` ocurre cuando el botón del dispositivo apuntador se suelta cuando está encima de un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onmouseover` = *script* [CT]

El evento `onmouseover` ocurre cuando el dispositivo apuntador se sitúa sobre un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onmousemove` = *script* [CT]

El evento `onmousemove` ocurre cuando el dispositivo apuntador se mueve mientras está sobre un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onmouseout` = *script* [CT]

El evento `onmouseout` ocurre cuando el dispositivo apuntador se aparta de un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onfocus` = *script* [CT]

El evento `onfocus` ocurre cuando el foco se dirige hacia un elemento, ya sea con el dispositivo apuntador o por navegación con tabulador. Este atributo puede utilizarse con los siguientes elementos: `A`, `AREA`, `LABEL`, `INPUT`, `SELECT`, `TEXTAREA` y `BUTTON`.

`onblur` = *script* [CT]

El evento `onblur` ocurre cuando el elemento pierde el foco ya sea con el dispositivo apuntador o por navegación con tabulador. Puede utilizarse con los mismos elementos que `onfocus`.

`onkeypress` = *script* [CT]

El evento `onkeypress` ocurre cuando se pulsa y se suelta una tecla encima de un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onkeydown` = *script* [CT]

El evento `onkeydown` ocurre cuando se pulsa una tecla encima de un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onkeyup` = *script* [CT]

El evento `onkeyup` ocurre cuando una tecla se suelta encima de un elemento. Este atributo puede utilizarse con la mayoría de los elementos.

`onsubmit` = *script* [CT]

El evento `onsubmit` ocurre cuando se envía un formulario. Sólo se aplica al elemento `FORM`.

`onreset` = *script* [CT]

El evento `onreset` ocurre cuando se reinicializa un formulario. Sólo se aplica al elemento `FORM`.

`onselect` = *script* [CT]

El evento `onselect` ocurre cuando un usuario selecciona texto de un campo de texto. Este atributo puede utilizarse con los elementos `INPUT` y `TEXTAREA`.

`onchange` = *script* [CT]

El evento `onchange` ocurre cuando un control pierde el foco de entrada *y* su valor ha sido modificado después de que el foco se dirigió hacia él. Este atributo se aplica a los siguientes elementos: `INPUT`, `SELECT` y `TEXTAREA`.

Es posible asociar una acción con un cierto número de eventos que ocurren cuando un usuario interacciona con un agente de usuario. Cada uno de los "eventos intrínsecos" recién enumerados toma como valor un script. El script se ejecuta cada vez que el evento ocurre para ese elemento. La sintaxis de los datos del script depende del lenguaje de scripts.

Los elementos de control tales como `INPUT`, `SELECT`, `BUTTON`, `TEXTAREA` y `LABEL` responden todos a ciertos eventos intrínsecos. Cuando estos elementos no aparecen dentro de un formulario, se pueden emplear para enriquecer la interfaz gráfica del usuario del documento.

Por ejemplo, los autores pueden querer incluir botones en sus documentos que no envíen un formulario pero que puedan comunicarse con un servidor cuando son activados.

Los siguientes ejemplos muestran posibles comportamientos de controles e interfaces de usuario basados en eventos intrínsecos.

En el siguiente ejemplo, `nombreUsuario` es un campo de texto obligatorio. Cuando un usuario intenta abandonar el campo, el evento `onblur` llama a una función JavaScript para confirmar que nombreUsuario tiene un valor aceptable.

```
<INPUT NAME="nombreUsuario" onblur="validarNombreUsuario(this.value)">
```

Aquí tenemos otro ejemplo en JavaScript:

```
<INPUT NAME="num"
       onchange="if (!checkNum(this.value, 1, 10))
           {this.focus();this.select();} else {thanks()}"
       VALUE="0">
```

Aquí tenemos un ejemplo en VBScript de un manejador de eventos para un campo de texto:

```
    <INPUT name="edit1" size="50">
    <SCRIPT type="text/vbscript">
      Sub edit1_cambiado()
        If edit1.value = "abc" Then
          button1.enabled = True
        Else
          button1.enabled = False
        End If
```

```
      End Sub
    </SCRIPT>
```

Aquí tenemos un ejemplo en JavaScript que asocia un evento con un script. En primer lugar vemos un manejador sencillo de clics:

```
<BUTTON type="button" name="miboton" value="10">
<SCRIPT type="text/javascript">
      function mi_onclick() {
         . . .
      }
    document.form.miboton.onclick = mi_onclick
 </SCRIPT>
 </BUTTON>
```

Aquí tenemos un manejador de ventanas más interesante:

```
<SCRIPT type="text/javascript">
      function mi_onload() {
         . . .
      }

      var ventana = window.open("algun/otro/URI")
      if (ventana) ventana.onload = mi_onload
</SCRIPT>
```

Obsérvese que un "document.write" o sus sentencias equivalentes en un manejador de eventos intrínsecos lo que hacen es crear un nuevo documento y escribir en él, no modificar el documento actual.

Modificación dinámica de documentos

Los scripts que se ejecutan cuando un documento es cargado pueden modificar los contenidos del documento dinámicamente. La capacidad de hacer esto depende del lenguaje de scripts en sí (p.ej., la sentencia "document.write" en el modelo de objetos de HTML no está soportada por algunas marcas).

La modificación dinámica de un documento puede ser modelizada de la siguiente manera:

1. Todos los elementos `SCRIPT` se evalúan en orden a medida que el documento es cargado.
2. Todas las construcciones de scripts contenidas en un elemento `SCRIPT` dado que generen datos CDATA SGML son evaluados. Su texto generado combinado se inserta en el documento sustituyendo al documento `SCRIPT`.
3. Los datos CDATA generados son evaluados nuevamente.

Los documentos HTML deben ser conformes con el DTD del HTML tanto antes como después del procesamiento de cualquiera de los elementos `SCRIPT`.

El ejemplo siguiente ilustra cómo puede un script modificar un documento dinámicamente. El siguiente script:

```
<TITLE>Documento de prueba</TITLE>
<SCRIPT type="text/javascript">
    document.write("<p><b>¡Hola Mundo!<\/b>")
</SCRIPT>
```

tiene el mismo efecto que este código HTML:

```
<TITLE>Documento de prueba</TITLE>
<P><B>¡Hola Mundo!</B>
```

Diseño de documentos para agentes de usuario que no soporten scripts

Las siguientes secciones tratan sobre cómo pueden los autores crear documentos que funcionen para agentes de usuario que no soporten scripts.

El elemento `NOSCRIPT`

```
<!ELEMENT NOSCRIPT - - (%block;)+
  -- contenedor de contenido alternativo para representación no basada
en scripts -->
<!ATTLIST NOSCRIPT
  %attrs;                              -- %coreattrs, %i18n, %events -
-
  >
```

Etiqueta inicial: ***obligatoria****, Etiqueta final:* ***obligatoria***

El elemento `NOSCRIPT` permite a los autores proporcionar contenido alternativo cuando un script no es ejecutado. El contenido de un elemento `NOSCRIPT` sólo debería ser representado por un agente de usuario capaz de reconocer scripts en los casos siguientes:

- El agente de usuario está configurado para no evaluar scripts.
- El agente de usuario no soporta un lenguaje de scripts invocado por un elemento `SCRIPT` anterior en el documento.

Los agentes de usuario que no soporten scripts en el lado del cliente deben representar los contenidos de este elemento.

Ocultar datos de scripts a agentes de usuario

Es probable que los agentes de usuario que no reconozcan el elemento `SCRIPT` representen los contenidos del elemento como texto. Algunos motores de scripts, incluyendo los de los lenguajes JavaScript, VBScript y Tcl, permiten que las sentencias de los scripts estén contenidas en un comentario SGML. Los agentes de usuario que no reconozcan el elemento `SCRIPT` ignorarán así el comentario, mientras que los motores

de scripts que funcionen correctamente entenderán que los scripts de los comentarios deberían ser ejecutados.

Otra solución al problema es mantener los scripts en documentos externos y hacer referencia a ellos con el atributo `src`.

Comentando scripts en Javascript
El motor de JavaScript permite que aparezca la cadena "<!--" al principio del elemento SCRIPT, e ignora el resto de los caracteres hasta el final de la línea. JavaScript interpreta "//" como el inicio de un comentario que se extiende hasta el final de la línea actual. Esta cadena es necesaria para ocultar la cadena "-->" al analizador JavaScript.

```
<SCRIPT type="text/javascript">
<!--  para ocultar los contenidos del script a los navegadores viejos
  function cuadrado(i) {
    document.write("La llamada pasó ", i ," a la función.","<BR>")
    return i * i
  }
  document.write("La función devolvió ",cuadrado(5),".")
// dejar de ocultar contenidos a los navegadores viejos  -->
</SCRIPT>
```

Comentando scripts en VBScript
En VBScript, un carácter de comilla simple hace que el resto de la línea actual sea tratada como un comentario. Puede usarse por tanto para ocultar a VBScript la cadena "-->", por ejemplo:

```
<SCRIPT type="text/vbscript">
  <!--
    Sub blabla()
     ...
    End Sub
  ' -->
 </SCRIPT>
```

Nota. *Algunos navegadores cierran los comentarios al encontrar el primer carácter ">", de modo que para ocultar el contenido de los scripts de estos navegadores, se pueden invertir los operandos de los operadores relacionales y de desplazamiento (p.ej., usar "y < x" en vez de "x > y") o se pueden usar caracteres de escape dependientes del lenguaje de scripts para ">".*