Aplicaciones Web

DHTML

David Cabrero Souto

Grupo MADS (http://www.grupomads.org/)
Universidade da Coruña

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Introducción

- Dynamic HTML.
- Posibilidad de ejecutar parte de la aplicación en el cliente.
- La parte cliente puede cambiar el documento HTML.
- Beneficios:
  - Es el sitio adecuado para la parte de IU.
  - Evitar circulaciones petición-respuesta.
- Problemas:
  - Portabilidad: no todos los navegadores lo soportan.
  - Implementación deficiente de estándares.
  - Aumenta la complejidad del cliente.
  - Accesibilidad.
- No confundir con:
  - Plugins (Flash, . . .).
  - Extensiones del navegador (p.e. Mozilla).

Javascript

- Lenguaje empleado en DHTML.
- Se compone de tres partes:
  - ECMAScript: lenguaje programación.
  - DOM: Modelo de Objetos del Documento (Document Object Model).
    API para manipular los documentos HTML.
  - BOM: Modelo de Objetos del Navegador (Browser Object Model).
    API para acceder a algunas funcionalidades del navegador.

Javascript

- Lenguaje empleado en DHTML.
- Se compone de tres partes:
  - ECMAScript: lenguaje programación.
  - DOM: Modelo de Objetos del Documento (Document Object Model).
    API para manipular los documentos HTML.
  - BOM: Modelo de Objetos del Navegador (Browser Object Model).
    API para acceder a algunas funcionalidades del navegador.

Javascript

- Lenguaje empleado en DHTML.
- Se compone de tres partes:
  - ECMAScript: lenguaje programación.
  - DOM: Modelo de Objetos del Documento (Document Object Model).
    API para manipular los documentos HTML.
  - BOM: Modelo de Objetos del Navegador (Browser Object Model).
    API para acceder a algunas funcionalidades del navegador.

Javascript

- Lenguaje empleado en DHTML.
- Se compone de tres partes:
  - ECMAScript: lenguaje programación.
  - DOM: Modelo de Objetos del Documento (Document Object Model).
    API para manipular los documentos HTML.
  - BOM: Modelo de Objetos del Navegador (Browser Object Model).
    API para acceder a algunas funcionalidades del navegador.

Javascript

- Lenguaje empleado en DHTML.
- Se compone de tres partes:
  - ECMAScript: lenguaje programación.
  - DOM: Modelo de Objetos del Documento (Document Object Model).
    API para manipular los documentos HTML.
  - BOM: Modelo de Objetos del Navegador (Browser Object Model).
    API para acceder a algunas funcionalidades del navegador.

ECMAScript

- Sintaxis con “sabor” a Java, C, perl.
- No incluye entrada/salida.
- El estándar ECMA define:
  - Sintaxis.
  - Tipos.
  - Instrucciones.
  - Palabras clave.
  - Palabras reservadas.
  - Operadores.
  - Objetos.
- Orientado a Objetos, pero no baso en clases, sino en prototipos.
- Sirve de core a otros lenguajes: Javascript, Actionscript, ...

ECMAScript

- Sintaxis con “sabor” a Java, C, perl.
- No incluye entrada/salida.
- El estándar ECMA define:
  - Sintaxis.
  - Tipos.
  - Instrucciones.
  - Palabras clave.
  - Palabras reservadas.
  - Operadores.
  - Objetos.
- Orientado a Objetos, pero no baso en clases, sino en prototipos.
- Sirve de core a otros lenguajes: Javascript, Actionscript, ...

ECMAScript

- Sintaxis con “sabor” a Java, C, perl.
- No incluye entrada/salida.
- El estándar ECMA define:
  - Sintaxis.
  - Tipos.
  - Instrucciones.
  - Palabras clave.
  - Palabras reservadas.
  - Operadores.
  - Objetos.
- Orientado a Objetos, pero no baso en clases, sino en prototipos.
- Sirve de core a otros lenguajes: Javascript, Actionscript, ...

ECMAScript

- Sintaxis con “sabor” a Java, C, perl.
- No incluye entrada/salida.
- El estándar ECMA define:
  - Sintaxis.
  - Tipos.
  - Instrucciones.
  - Palabras clave.
  - Palabras reservadas.
  - Operadores.
  - Objetos.
- Orientado a Objetos, pero no baso en clases, sino en prototipos.
- Sirve de core a otros lenguajes: Javascript, Actionscript, ...

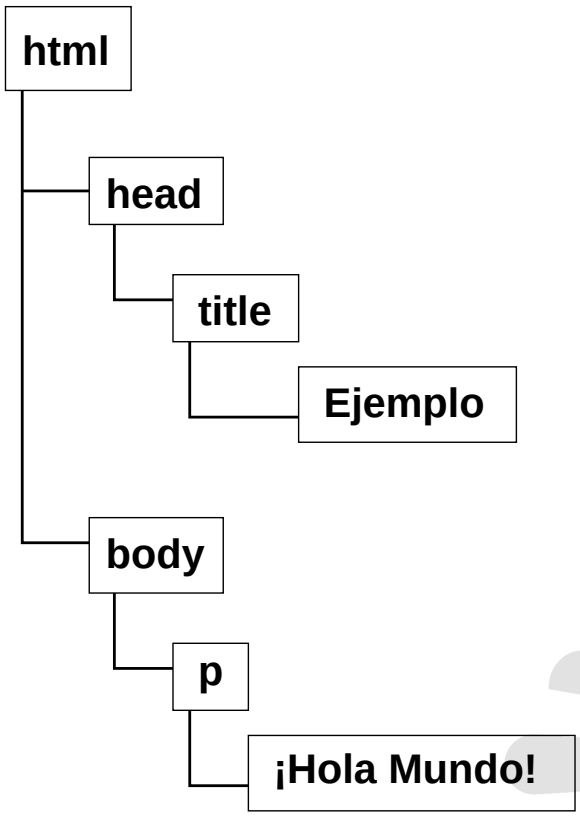
ECMAScript

- Sintaxis con “sabor” a Java, C, perl.
- No incluye entrada/salida.
- El estándar ECMA define:
  - Sintaxis.
  - Tipos.
  - Instrucciones.
  - Palabras clave.
  - Palabras reservadas.
  - Operadores.
  - Objetos.
- Orientado a Objetos, pero no baso en clases, sino en prototipos.
- Sirve de core a otros lenguajes: Javascript, Actionscript, . . .

DOM

- API para HTML, XML, XHTML, SVG, MathML, SMIL, . . .
- El documento se representa como una jerarquía de nodos.
- En HTML cada nodo representa un elemento y sus hijos estarán representados por los correspondientes nodos hijo.

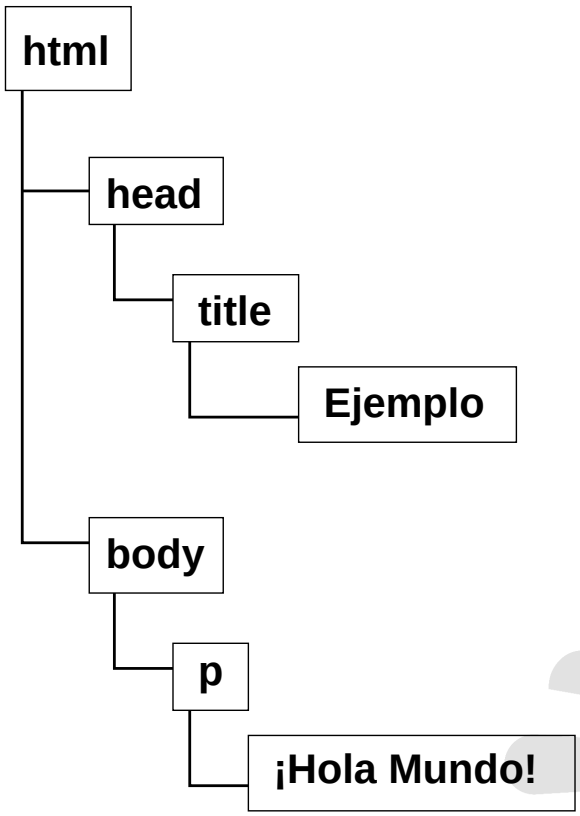
```
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
    <p>¡Hola Mundo!</p>
  </body>
</html>
```



DOM

- API para HTML, XML, XHTML, SVG, MathML, SMIL, . . .
- El documento se representa como una jerarquía de nodos.
- En HTML cada nodo representa un elemento y sus hijos estarán representados por los correspondientes nodos hijo.

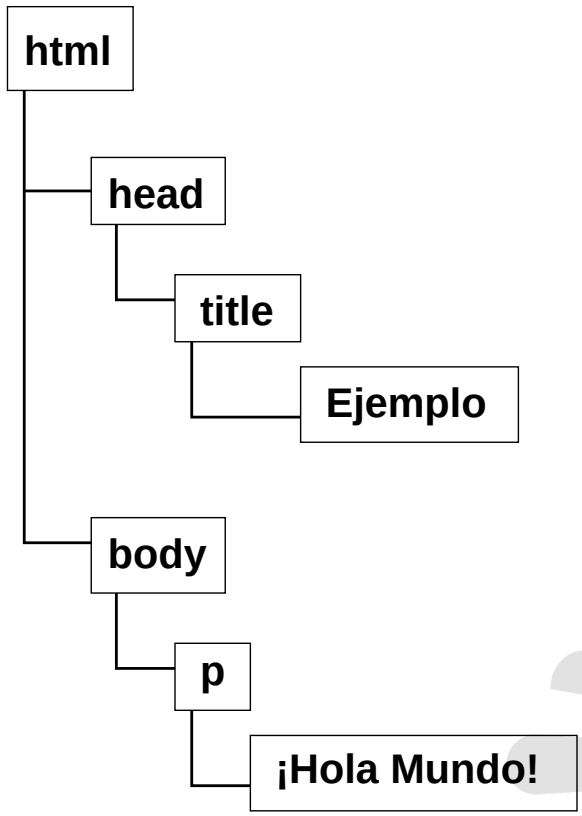
```
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
    <p>¡Hola Mundo!</p>
  </body>
</html>
```



DOM

- API para HTML, XML, XHTML, SVG, MathML, SMIL, . . .
- El documento se representa como una jerarquía de nodos.
- En HTML cada nodo representa un elemento y sus hijos estarán representados por los correspondientes nodos hijo.

```
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
    <p>¡Hola Mundo!</p>
  </body>
</html>
```



DOM (cont.)

- Estándares W3C.
  - Nivel 1. DOM Core (XML) + DOM HTML.
  - Nivel 2.
    - DOM Views. Documento antes y después de aplicar CSS.
    - DOM Events.
    - DOM Style. API para manipular CSS.
    - DOM Traversal and Range. Recorrido y manipulación del árbol.
  - Nivel 3. DOM Load and Save, DOM Validation, XML 1.0 (Xpath, XML Infoset, XML Base).

DOM (cont.)

- Estándares W3C.
  - Nivel 1. DOM Core (XML) + DOM HTML.
  - Nivel 2.
    - DOM Views. Documento antes y después de aplicar CSS.
    - DOM Events.
    - DOM Style. API para manipular CSS.
    - DOM Traversal and Range. Recorrido y manipulación del árbol.
  - Nivel 3. DOM Load and Save, DOM Validation, XML 1.0 (Xpath, XML Infoset, XML Base).

DOM (cont.)

- Estándares W3C.
  - Nivel 1. DOM Core (XML) + DOM HTML.
  - Nivel 2.
    - DOM Views. Documento antes y después de aplicar CSS.
    - DOM Events.
    - DOM Style. API para manipular CSS.
    - DOM Traversal and Range. Recorrido y manipulación del árbol.
  - Nivel 3. DOM Load and Save, DOM Validation, XML 1.0 (Xpath, XML Infoset, XML Base).

DOM (cont.)

- Estándares W3C.
  - Nivel 1. DOM Core (XML) + DOM HTML.
  - Nivel 2.
    - DOM Views. Documento antes y después de aplicar CSS.
    - DOM Events.
    - DOM Style. API para manipular CSS.
    - DOM Traversal and Range. Recorrido y manipulación del árbol.
  - Nivel 3. DOM Load and Save, DOM Validation, XML 1.0 (Xpath, XML Infoset, XML Base).

BOM

- No está estandarizado.
- Estándar *de facto*.
- Posibilidades habituales:
  - Abrir, mover, cambiar tamaño, y cerrar ventanas.
  - Objeto Navigator.
  - Objeto Location.
  - Objeto Screen.
  - Manejo de cookies.

BOM

- No está estandarizado.
- Estándar *de facto*.
- Posibilidades habituales:
  - Abrir, mover, cambiar tamaño, y cerrar ventanas.
  - Objeto Navigator.
  - Objeto Location.
  - Objeto Screen.
  - Manejo de cookies.

Utilidad de javascript

- Cosas útiles.
  - Evitar viajes ida-vuelta (round-trips) al servidor.
  - P.e.: validación de datos, menús.
- Cosas superfluas.
  - P.e.: menús animados.
- Cosas molestas/indeseables.
  - Vetanas emergentes.
  - Cambiar la barra de estado.
  - ...
- Cosas estúpidas.
  - Reimplementar links.

Utilidad de javascript

- Cosas útiles.
  - Evitar viajes ida-vuelta (round-trips) al servidor.
  - P.e.: validación de datos, menús.
- Cosas superfluas.
  - P.e.: menús animados.
- Cosas molestas/indeseables.
  - Vetanas emergentes.
  - Cambiar la barra de estado.
  - ...
- Cosas estúpidas.
  - Reimplementar links.

Utilidad de javascript

- Cosas útiles.
  - Evitar viajes ida-vuelta (round-trips) al servidor.
  - P.e.: validación de datos, menús.
- Cosas superfluas.
  - P.e.: menús animados.
- Cosas molestas/indeseables.
  - Vetanas emergentes.
  - Cambiar la barra de estado.
  - ...
- Cosas estúpidas.
  - Reimplementar links.

Utilidad de javascript

- Cosas útiles.
  - Evitar viajes ida-vuelta (round-trips) al servidor.
  - P.e.: validación de datos, menús.
- Cosas superfluas.
  - P.e.: menús animados.
- Cosas molestas/indeseables.
  - Vetanas emergentes.
  - Cambiar la barra de estado.
  - ...
- Cosas estúpidas.
  - Reimplementar links.

Utilidad de javascript

- Cosas útiles.
  - Evitar viajes ida-vuelta (round-trips) al servidor.
  - P.e.: validación de datos, menús.
- Cosas superfluas.
  - P.e.: menús animados.
- Cosas molestas/indeseables.
  - Vetanas emergentes.
  - Cambiar la barra de estado.
  - ...
- Cosas estúpidas.
  - Reimplementar links.

Sintaxis y tipos de datos de Javascript

```
/* Los puntos y coma al final de la línea son opcionales */

// Las variables no están tipadas
var color = "rojo";

// Las llaves indica bloques de código
if (jugador == "rojo") {
  jugador == "azul"
}

// Tipos de datos básicos
var aValores = new Array();
var bFound = true;
var fFlotante = 3.12;
var fnFuncion = funcionA;
var iEntero = 5;
var oObjeto = new Object();
var sCadena = ":Hola Mundo!"
```

Objetos en Javascript (I)

- ECMA-262:

  Un objeto es una colección sin ordenar de propiedades que contienen un valor primitivo, un objeto o una función.

- Las variables almacenan referencias a objetos.
- Más cerca de un LOO basado en prototipos que un LOO basado en clases.
  - La clases es simplemente una plantilla.
  - Es posible modificar los atributos y la interfaz de un objeto una vez creado.
- Objetos nativos.
  - Object
  - Boolean
  - String
  - Date
  - RegExp
  - Array
  - Function
  - URIError
  - EvalError
  - RangeError
  - SyntaxError
  - ReferenceError

Objetos en Javascript (I)

- ECMA-262:

  Un objeto es una colección sin ordenar de propiedades que contienen un valor primitivo, un objeto o una función.

- Las variables almacenan referencias a objetos.
- Más cerca de un LOO basado en prototipos que un LOO basado en clases.
  - La clases es simplemente una plantilla.
  - Es posible modificar los atributos y la interfaz de un objeto una vez creado.
- Objetos nativos.
  - Object
  - Boolean
  - String
  - Date
  - RegExp
  - Array
  - Function
  - URIError
  - EvalError
  - RangeError
  - SyntaxError
  - ReferenceError

Objetos en Javascript (I)

- ECMA-262:

  Un objeto es una colección sin ordenar de propiedades que contienen un valor primitivo, un objeto o una función.

- Las variables almacenan referencias a objetos.
- Más cerca de un LOO basado en prototipos que un LOO basado en clases.
  - La clases es simplemente una plantilla.
  - Es posible modificar los atributos y la interfaz de un objeto una vez creado.
- Objetos nativos.
  - Object
  - Boolean
  - String
  - Date
  - RegExp
  - Array
  - Function
  - URIError
  - EvalError
  - RangeError
  - SyntaxError
  - ReferenceError

Objetos en Javascript (I)

- ECMA-262:

  Un objeto es una colección sin ordenar de propiedades que contienen un valor primitivo, un objeto o una función.

- Las variables almacenan referencias a objetos.
- Más cerca de un LOO basado en prototipos que un LOO basado en clases.
  - La clases es simplemente una plantilla.
  - Es posible modificar los atributos y la interfaz de un objeto una vez creado.
- Objetos nativos.
  - Object
  - Boolean
  - String
  - Date
  - RegExp
  - Array
  - Function
  - URIError
  - EvalError
  - RangeError
  - SyntaxError
  - ReferenceError

Objetos en Javascript (III)

- Objetos incorporados.
  - Existe una instancia, sin necesidad de crearla.
  - Global. No es referenciable. Contiene las funciones globales. `encodeURI`, `parseFloat()`, ...
  - Math.
- No existe ámbito público/privado.

  Posible convención: Los atributos privados comienzan por "_".

- Los objetos se pueden crear añadiendo atributos y métodos.

```
var oCoche = new Object;
oCoche._color = "negro";
oCoche._conductores = new Array("Alonso", "Pedro");
oCoche.getColor = function() {
  return this._color;
}
```

Objetos en Javascript (III)

- Objetos incorporados.
  - Existe una instancia, sin necesidad de crearla.
  - Global. No es referenciable. Contiene las funciones globales. `encodeURI`, `parseFloat()`, ...
  - Math.
- No existe ámbito público/privado.

  Posible convención: Los atributos privados comienzan por "_".

- Los objetos se pueden crear añadiendo atributos y métodos.

```
var oCoche = new Object;
oCoche._color = "negro";
oCoche._conductores = new Array("Alonso", "Pedro");
oCoche.getColor = function() {
  return this._color;
}
```

Objetos en Javascript (III)

- Objetos incorporados.
  - Existe una instancia, sin necesidad de crearla.
  - Global. No es referenciable. Contiene las funciones globales. `encodeURI`, `parseFloat()`, ...
  - Math.
- No existe ámbito público/privado.

  Posible convención: Los atributos privados comienzan por "_".

- Los objetos se pueden crear añadiendo atributos y métodos.

```
var oCoche = new Object;
oCoche._color = "negro";
oCoche._conductores = new Array("Alonso", "Pedro");
oCoche.getColor = function() {
  return this._color;
}
```

Objetos en Javascript (IV)

- Clase = plantilla (Factory).
- Usamos una función como constructor.

```
function Coche(sColor) {
  this._color = "negro";
  this._conductores = new Array("Alonso", "Pedro");
  this.getColor = function() {
    return this._color;
  }
}

var oCoche1 = new Coche("negro");
var oCoche2 = new Coche("azul");
```

- Problema del constructor: cada vez se crea una instancia nueva de los métodos.

Objetos en Javascript (IV)

- Clase = plantilla (Factory).
- Usamos una función como constructor.

```
function Coche(sColor) {
  this._color = "negro";
  this._conductores = new Array("Alonso", "Pedro");
  this.getColor = function() {
    return this._color;
  }
}

var oCoche1 = new Coche("negro");
var oCoche2 = new Coche("azul");
```

- Problema del constructor: cada vez se crea una instancia nueva de los métodos.

Objetos en Javascript (IV)

- Podemos usar prototipos para instanciar objetos.

```
function Coche() {
}

Coche.prototype._color = "red";
Coche.prototype._conductores = new Array("Alonso", "Pe
Coche.prototype.getColor = function() {
    return this._color;
  }

var oCoche1 = new Coche();
var oCoche2 = new Coche();
```

- Las funciones (métodos) se comparten.
- Problemas:
  - El "constructor" no puede tener argumentos.

Objetos en Javascript (IV)

- Podemos usar prototipos para instanciar objetos.

```
function Coche() {
}

Coche.prototype._color = "red";
Coche.prototype._conductores = new Array("Alonso", "Pe
Coche.prototype.getColor = function() {
    return this._color;
  }

var oCoche1 = new Coche();
var oCoche2 = new Coche();
```

- Las funciones (métodos) se comparten.
- Problemas:
  - El "constructor" no puede tener argumentos.

Objetos en Javascript (IV)

- Podemos usar prototipos para instanciar objetos.

```
function Coche() {
}

Coche.prototype._color = "red";
Coche.prototype._conductores = new Array("Alonso", "Pe
Coche.prototype.getColor = function() {
    return this._color;
  }

var oCoche1 = new Coche();
var oCoche2 = new Coche();
```

- Las funciones (métodos) se comparten.
- Problemas:
  - El “constructor” no puede tener argumentos.

Objetos en Javascript (IV)

- Solución habitual: usar constructores y prototipos.

```
function Coche(sColor) {
  this._color = sColor;
  this._conductores = new Array("Alonso", "Pedro");
}
Coche.prototype.getColor = function() {
    return this._color;
  }

var oCoche1 = new Coche("negro");
var oCoche2 = new Coche("azul");
```

Herencia en Javascript (I)

- Los objetos nativos, incorporados o definidos por el navegador no se pueden derivar.
- La herencia es emulada.
- Con constructores: usar `call` para invocar el constructor de la *clase* padre.
- Con prototipos: la primera operación es asignar al prototipo una instancia de la *clase* padre, y a continuación extendemos el prototipo.

Herencia en Javascript (I)

- Los objetos nativos, incorporados o definidos por el navegador no se pueden derivar.
- La herencia es emulada.
- Con constructores: usar `call` para invocar el constructor de la *clase* padre.
- Con prototipos: la primera operación es asignar al prototipo una instancia de la *clase* padre, y a continuación extendemos el prototipo.

Herencia en Javascript (I)

- Los objetos nativos, incorporados o definidos por el navegador no se pueden derivar.
- La herencia es emulada.
- Con constructores: usar `call` para invocar el constructor de la *clase* padre.
- Con prototipos: la primera operación es asignar al prototipo una instancia de la *clase* padre, y a continuación extendemos el prototipo.

Herencia en Javascript (I)

- Los objetos nativos, incorporados o definidos por el navegador no se pueden derivar.
- La herencia es emulada.
- Con constructores: usar `call` para invocar el constructor de la *clase* padre.
- Con prototipos: la primera operación es asignar al prototipo una instancia de la *clase* padre, y a continuación extendemos el prototipo.

Herencia en Javascript (I)

- Caso híbrido. Ejemplo:

```
function Coche(sColor) {
  this._color = sColor;
}
Coche.prototype.getColor = function() {
    return this._color;
  }

function F1(sColor, sPiloto) {
  Coche.call(this, sColor);
  this._pilot = sPiloto;
}
F1.prototype.getPiloto = function () {
  return this._piloto;
}
```

Añadir javascript a los documentos html

- Varias formas.
- Elemento *script* en la cabecera.

```
<head>
  ...
  <script type="text/javascript">
    var n=6;
    alert("N vale "+n);
  </script>
  ...
</head>
```

- Enlace a rescurso externo.

```
<head>
  ...
  <script type="text/javascript"
          src="externo.js"></script>
```

Añadir javascript a los documentos html

- Varias formas.
- Elemento *script* en la cabecera.

```
<head>
  ...
  <script type="text/javascript">
    var n=6;
    alert("N vale "+n);
  </script>
  ...
</head>
```

- Enlace a rescurso externo.

```
<head>
  ...
  <script type="text/javascript"
          src="externo.js"></script>
```

Añadir javascript a los documentos html

- Varias formas.
- Elemento *script* en la cabecera.

```
<head>
  ...
  <script type="text/javascript">
    var n=6;
    alert("N vale "+n);
  </script>
  ...
</head>
```

- Enlace a rescurso externo.

```
<head>
  ...
  <script type="text/javascript"
          src="externo.js"></script>
```

Código Javascript en XHTML

- XHTML $\Rightarrow$ XML.

```
<script type="text/javascript">
// <![CDATA[

    var n=6;
    alert("N vale "+n);

// ]]>
</script>
```

Concurrencia

- Los browsers ejecutan algunas acciones concurrentemente o en un orden inconveniente. Típicamente: crear el documento y ejecutar el código javascript de la cabecera.
- El objeto `window` tiene un *callback* en el atributo `onload`.
- Un *idioma* típico:

```
<head>
  <script type="text/javascript">
  // <![CDATA[

  function init() { ... }

  window.onload = init;

  // ]]>
  </script>
</head>
```

Concurrencia

- Los browsers ejecutan algunas acciones concurrentemente o en un orden inconveniente. Típicamente: crear el documento y ejecutar el código javascript de la cabecera.
- El objeto `window` tiene un *callback* en el atributo `onload`.
- Un *idioma* típico:

```
<head>
  <script type="text/javascript">
  // <![CDATA[

  function init() { ... }

  window.onload = init;

  // ]]>
  </script>
</head>
```

DOM (I)

- Documento = jerarquía de nodos.
- Objeto *document*.
  - `window.document == document`.
  - Elemento *html*: `document.documentElement`.

```
var oHthml = document.documentElement;
var oHead = oHtml.childNodes.item(0);
var oBody = oHtml.childNodex.item(1);
```

DOM (I)

- Documento = jerarquía de nodos.
- Objeto *document*.
  - `window.document == document`.
  - Elemento *html*: `document.documentElement`.

```
var oHthml = document.documentElement;
var oHead = oHtml.childNodes.item(0);
var oBody = oHtml.childNodex.item(1);
```

DOM (II)

- La interface `Node`:

```
nodeName   : String            previousSibling : Node
nodeValue  : String            nextSibling     : Node
nodeType   : Number            hasChildNodes() : Boolean
ownerDocument : Document
firstChild     : Node          appendChild(node) : Node
lastChild      : Node          removeChild(node) : Node
childNodes     : NodeList
attributes     : NamedNodeMap
replaceChild(oldNode,newNode) : Node
insertBefore(newNode, refNode) : Node
```

  - Ejemplo: recorrer los hijos de un nodo.

```
var children = aNode.childNodes;
for(int i=0; i<children.length; i++) {
  some_op(children[i]);
}
```

DOM (II)

- La interface `Node`:

```
nodeName   : String            previousSibling : Node
nodeValue  : String            nextSibling     : Node
nodeType   : Number            hasChildNodes() : Boolean
ownerDocument : Document
firstChild    : Node           appendChild(node) : Node
lastChild     : Node           removeChild(node) : Node
childNodes    : NodeList
attributes    : NamedNodeMap
replaceChild(oldNode,newNode) : Node
insertBefore(newNode, refNode) : Node
```

- Ejemplo: recorrer los hijos de un nodo.

```
var children = aNode.childNodes;
for(int i=0; i<children.length; i++) {
  some_op(children[i]);
}
```

DOM (III)

- Los nodos *element* (Elementos XHTML) tiene atributos.
- La propiedad `attributes` contiene una lista de nodos de tipo atributo.

```
getNamedItem(name)        setNamedItem(node)
removeNamedItem(name)     item(position)
```

```
getAttribute(name) ==
    attributes.getNamedItem(name).value
setAttribute(name,value) ==
    attributes.getNamedItem(name).value = value
removeAttribute(name) ==
    attributes.removeNamedItem(name)
```

- Ejemplo: `oP` referencia el nodo de elemento:

```
<p class="rojo" id="p1">¡Hola Mundo!</p>
```

```
var sID = oP.attributes.getNamedItem("id").nodeValue;
```

DOM (III)

- Los nodos *element* (Elementos XHTML) tiene atributos.
- La propiedad `attributes` contiene una lista de nodos de tipo atributo.

```
getNamedItem(name)        setNamedItem(node)
removeNamedItem(name)     item(position)
```

```
getAttribute(name) ==
    attributes.getNamedItem(name).value
setAttribute(name,value) ==
    attributes.getNamedItem(name).value = value
removeAttribute(name) ==
    attributes.removeNamedItem(name)
```

- Ejemplo: `oP` referencia el nodo de elemento:

```
<p class="rojo" id="p1">¡Hola Mundo!</p>
```

```
var sID = oP.attributes.getNamedItem("id").nodeValue;
```

DOM (III)

- Los nodos *element* (Elementos XHTML) tiene atributos.
- La propiedad `attributes` contiene una lista de nodos de tipo atributo.

```
getNamedItem(name)        setNamedItem(node)
removeNamedItem(name)     item(position)
```

```
getAttribute(name) ==
    attributes.getNamedItem(name).value
setAttribute(name,value) ==
    attributes.getNamedItem(name).value = value
removeAttribute(name) ==
    attributes.removeNamedItem(name)
```

- Ejemplo: `oP` referencia el nodo de elemento:

```
<p class="rojo" id="p1">¡Hola Mundo!</p>
```

```
var sID = oP.attributes.getNamedItem("id").nodeValue;
```

DOM (IV)

- Otros métodos de acceso a los nodos.
  - `getElementsByName()`
  - `getElementsById()`
- Crear nodos.
  - `createElement()`
  - `createTextNode()`
  - `createAttribute(),createComment,...`

DOM (IV)

- Otros métodos de acceso a los nodos.
  - `getElementsByName()`
  - `getElementsById()`
- Crear nodos.
  - `createElement()`
  - `createTextNode()`
  - `createAttribute()`, `createComment`, ...

DOM (V)

- Ejemplo: formulario.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3c.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3c.org/199/xhtml">

<head>
<title>Ejemplo de DOM</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1" />
</head>
```

DOM (VI)

- Ejemplo: formulario (cont.).

```
<body>
  <form method="post" action="hacer.cgi">
    <fieldset>
      <legend>¿Qué personaje te gusta más?</legend>
        <p><input type="radio" name="personaje" vaue="LAdama">
            Lee Adama </input></p>
        <p><input type="radio" name="personaje" vaue="StarBuck"
            checked="checked">
            StarBuck </input></p>
        <p><input type="radio" name="personaje" vaue="Baltar">
            Dr. Baltar </input></p>
        <p><input type="radio" name="personaje" vaue="Cylon6">
            Cylon nb. 6 </input></p>
    </fieldset>
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

DOM (VII)

- Añadimos javascript para que el *foco* pase al primer elemento de la lista de opciones.

```
<head>
...
<script type="text/javascript">
//<![CDATA[
function init() {
    var oRadios = document.getElementsByName("personaje");
    oRadios.item(0).focus();
}

window.onload = init;
//]]>
</script>
</head>
```

DOM y CSS

- El estilo CSS asociado a un nodo se accede a traves de su atributo `style`.
- Ejemplo:

```
oP.sytle.color = "red";
```

- La clase se accede a traves del atrituto `className`.

DOM y CSS

- El estilo CSS asociado a un nodo se accede a traves de su atributo `style`.
- Ejemplo:

```
oP.sytle.color = "red";
```

- La clase se accede a traves del atrituto `className`.

Interactividad

- Es posible asociar manejadores a cierto eventos de usuario.
- Existen varias maneras.
- Implementación deficiente de estandares (IE).

Interactividad

- Es posible asociar manejadores a cierto eventos de usuario.
- Existen varias maneras.
- Implementación deficiente de estandares (IE).

Interactividad

- Es posible asociar manejadores a cierto eventos de usuario.
- Existen varias maneras.
- Implementación deficiente de estandares (IE).

Eventos

- Atributos de las etiquetas:

```
<a href="fallback.cgi"
   onClick="hacer_algo(); return false;">
  ...
</a>
```

- Atributos de los elementos:

```
var enlace = ...
enlace.onclick = hacer_algo;
```

- DOM W3C:

```
var enlace = ...
enlace.addEventListener('click',hacer_algo,false);
```

Eventos

- Atributos de las etiquetas:

```
<a href="fallback.cgi"
   onClick="hacer_algo(); return false;">
  ...
</a>
```

- Atributos de los elementos:

```
var enlace = ...
enlace.onclick = hacer_algo;
```

- DOM W3C:

```
var enlace = ...
enlace.addEventListener('click',hacer_algo,false);
```

Eventos

- Atributos de las etiquetas:

```
<a href="fallback.cgi"
   onClick="hacer_algo(); return false;">
  ...
</a>
```

- Atributos de los elementos:

```
var enlace = ...
enlace.onclick = hacer_algo;
```

- DOM W3C:

```
var enlace = ...
enlace.addEventListener('click',hacer_algo,false);
```

El objeto `event`

- Argumento de los manejadores.
- Contiene información del evento.

```
function hacer_algo(e) {
  alert("El tipo de evento es" + e.type);
}
```

- Pero, en Microsoft sólo tenemos `window.event`:

```
function hacer_algo(e) {
  alert("El tipo de evento es" + window.event.type);
}
```

- Cross-browser.

```
function hacer_algo(e) {
    if (!e) var e = window.event;
    // e está disponible en todos los browsers
```

El objeto `event`

- Argumento de los manejadores.
- Contiene información del evento.

```
function hacer_algo(e) {
  alert("El tipo de evento es" + e.type);
}
```

- Pero, en Microsoft sólo tenemos `window.event`:

```
function hacer_algo(e) {
  alert("El tipo de evento es" + window.event.type);
}
```

- Cross-browser.

```
function hacer_algo(e) {
    if (!e) var e = window.event;
    // e está disponible en todos los browsers
```

El objeto `event`

- Argumento de los manejadores.
- Contiene información del evento.

```
function hacer_algo(e) {
  alert("El tipo de evento es" + e.type);
}
```

- Pero, en Microsoft sólo tenemos `window.event`:

```
function hacer_algo(e) {
  alert("El tipo de evento es" + window.event.type);
}
```

- Cross-browser.

```
function hacer_algo(e) {
    if (!e) var e = window.event;
    // e está disponible en todos los browsers
```

Ejemplo (I)

- Validación en formularios.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional/
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html lang="en">
<head>
<title>W3C DOM - Form error messages</title>
...

<script type="text/javascript">
<!--
...
-->
</script>

</head>
```

Ejemplo (II)

- Validación en formularios.

```
<body>
...
<form name="example" action="/cgi-bin/show_params.cgi">
<p><label for="name">name</label>
<input size="20" name="name" id="name" /></p>
<p><label for="address">address</label>
<input size="20" name="address" id="address" /></p>
<p><label for="city">city</label>
<input size="20" name="city" id="city" /></p>
<p><label for="email">e-mail</label>
<input size="20" name="email" id="email" /></p>

<p><input type="submit" value="Submit form" /></p>
</form>
...
</body>
```

Ejemplo (III)

- Javascript de validación:

```
<script type="text/javascript">
<!--

var W3CDOM = (document.getElementsByTagName && document.cre

function init ()
{
        document.forms[0].onsubmit = function () {
                return validate()
        }
}

...

window.onload = init
// -->
```

Ejemplo (IV)

- Función de validación:

```
function validate()
{
        validForm = true;
        firstError = null;
        errorstring = '';
        var x = document.forms[0].elements;
        for (var i=0;i<x.length;i++)
        {
                if (!x[i].value)
                        writeError(x[i],'This field is required')
        }
        if (x['email'].value.indexOf('@') == -1)
                writeError(x['email'],'This is not a valid email
        if (!W3CDOM)
                alert(errorstring);
        if (firstError)
                firstError.focus();
        if (validForm)
```

Ejemplo (V)

- Función de marcado de errores:

```
function writeError(obj,message)
{
        validForm = false;
        if (obj.hasError) return;
        if (W3CDOM)
        {
                obj.className += ' error';
                obj.onchange = removeError;
                var sp = document.createElement('span');
                sp.className = 'error';
                sp.appendChild(document.createTextNode(message));
                obj.parentNode.appendChild(sp);
                obj.hasError = sp;
        }
        else
        {
                errorstring += obj.name + ': ' + message + '\n';
                obj.hasError = true;
```

Ejemplo (VI)

- Función de borrado de marcas de error:

```
function removeError()
{
        this.className = this.className.substring(0,this.classNam
        this.parentNode.removeChild(this.hasError);
        this.hasError = null;
        this.onchange = null;
}
```

Ejemplo (I)

- Crear elementos

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional/
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3c.org/199/xhtml">
<head>
<title>Ejemplo de DOM</title>
...

<script type="text/javascript">
<!--
...
--!>
</script>

</head>
```

Ejemplo (II)

- La página inicial.

```
<div id="imagenes">
<h1>Imagenes</h1>
</div>

<form action="fallback.cgi">
  <input type="submit" name="botonAñadir" value="Añadir" />
</form>
</body>
```

Ejemplo (III)

- Código de inicialización.

```
function init() {
  var boton = document.getElementsByName("botonAñadir").item(0);
  boton.addEventListener('click', addImage, false);
}
window.onload = init;
```

- Manejador de evento.

```
function addImage(e) {
    var capa = document.getElementById('imagenes');
    var img = document.createElement('img');
    img.setAttribute('src','imagen.jpg')
    img.setAttribute('class', 'imagen');
    capa.appendChild(img);
    e.preventDefault();
}
```

Ejemplo (III)

- Código de inicialización.

```
function init() {
  var boton = document.getElementsByName("botonAñadir").item(0);
  boton.addEventListener('click', addImage, false);
}
window.onload = init;
```

- Manejador de evento.

```
function addImage(e) {
    var capa = document.getElementById('imagenes');
    var img = document.createElement('img');
    img.setAttribute('src','imagen.jpg')
    img.setAttribute('class', 'imagen');
    capa.appendChild(img);
    e.preventDefault();
}
```