Profesor: Borja Rey Seoane

TEMA 3

MODELO DE OBXECTOS DA LINGUAXE

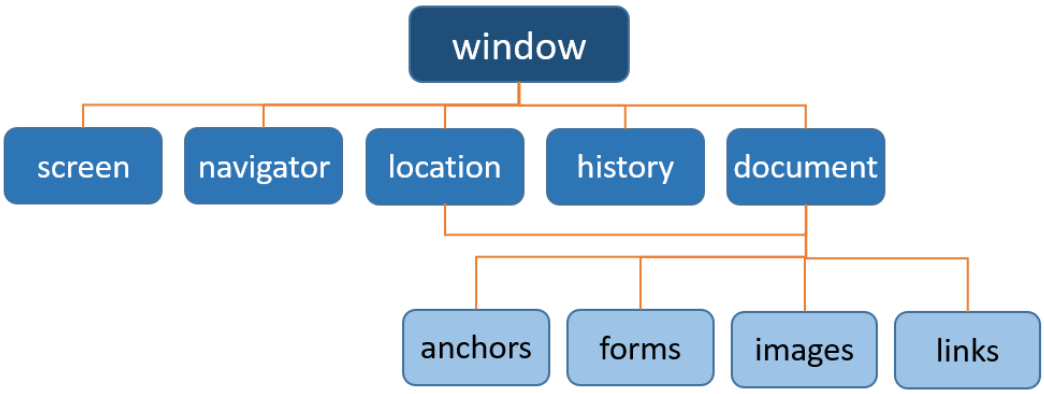
Módulo: **Desenvolvemento Web en Contorna Cliente**

Ciclo: **DAW**

Obxectos predefinidos

JS permite a comunicación co navegador co gaio de personalizar a contorna, de xeito que obteñamos aplicacións con maior interactividade. O anterior lógrase mediante o uso de BOM[1].

O devandito modelo non conta cun estándar que o defina e, polo tanto, os navegadores poden implementalo de distintos xeitos. Mesmo así, hai un certo consenso, existindo varios obxectos comúns aos navegadores principais, que forman a seguinte estrutura xerárquica:



Cando unha páxina web é cargada no navegador créanse os obxectos anteriores en base á estrutura da mesma, as características da aplicación cliente e a interacción do usuario.

A tarefa principal do BOM é a de xestionar as xanelas do navegador, permitindo a comunicación entre elas.

`Window`

Este obxecto, coma vimos no diagrama anterior, atópase na parte máis alta da xerarquía e considérase o obxecto por defecto, polo que non fai falta referencialo para acceder ás súas propiedades e métodos.

`Window` representa ao navegador, aínda que naqueles navegadores que permiten varias lapelas a cada unha delas correspóndelle un obxecto `window`, isto é, non se comparte o obxecto deste tipo entre as lapelas da mesma xanela de aplicación. De xeito semellante, se un documento contén `frames` (etiquetas `<iframe>`) a cada unha delas

[1] *Browser Object Model*.

corresponderá un obxecto `window` (ademais do principal que corresponda ao documento HTML que actúa coma contedor).

Ademais da área da xanela destinada a amosar a páxina web, este obxecto tamén permite traballar coas medidas da xanela, as barras de desprazamento, a barra de ferramentas, a barra de estado, etc.

Un aspecto importante a considerar cando programamos en JS é que todas as variables, obxectos e funcións globais que non sexan asignadas a ningún outro obxecto serán membros de `window`.

Referencia

Existen varias formas de facer referencia aos seus membros deste obxecto:

- **`window`:** coma con calquera obxecto, indícase o nome do mesmo e, separado por un punto (.), a propiedade ou método ao cal queremos acceder.
- **`self`:** pódese substituír o nome do obxecto pola palabra reservada `self`, de xeito que poidamos facer desde ao propio documento contido na xanela.
- **Por defecto:** aínda que fai menos comprensible o código, por tratarse do obxecto predefinido, pódese omitir a referencia ao mesmo e indicar simplemente a propiedade ou método a cal queremos acceder.

```
// Formas equivalentes de acceder á propiedade parent
window.parent;
self.parent;
parent;

// Formas equivalentes de acceder ao método close
window.close();
self.close();
close();
```

Propiedades

A continuación veremos unha enumeración das distintas propiedades de `window` e os valores que poden tomar:

- **Estado:**
  - `closed`: indica se a xanela foi pechada ou non.
  - `defaultStatus`: establece ou devolve o texto por defecto da barra de estado.
  - `name`: establece ou devolve o nome da xanela.
  - `status`: establece ou devolve o texto da barra de estado.

- **Tamaño:**
  - `innerHeight`: altura da área de visualización da xanela (incluíndo barras de desprazamento).
  - `innerWidth`: largura da área de visualización da xanela (incluíndo barras de desprazamento).
  - `outerHeight`: altura de toda a xanela do navegador, incluíndo a zona exterior á área de visualización (barras de ferramentas, de título, de estado, etc.).
  - `outerWidth`: largura de toda a xanela do navegador, incluíndo a zona exterior á área de visualización (barras de ferramentas, de título, de estado, etc.).
- **Posición:**
  - `pageXOffset`: cantidade de pixels desprazados horizontalmente dende a esquina superior-esquerda da xanela.
  - `pageYOffset`: cantidade de pixels desprazados verticalmente dende a esquina superior-esquerda da xanela.
  - `screenLeft`: coordenada horizontal da xanela relativa á pantalla.
  - `screenTop`: coordenada vertical da xanela relativa á pantalla.
  - `screenX`: coordenada horizontal da xanela relativa á pantalla.
  - `screenY`: coordenada vertical da xanela relativa á pantalla.
- **Frames:**
  - `frameElement`: elemento `<iframe>` no que se atopa a páxina actual.
  - `frames`: conxunto de elementos `<iframe>` que se atopan na páxina actual.
  - `length`: número de elementos `<iframe>` que se atopan na páxina actual.
- **Obxectos do BOM:**
  - `console`: referencia á consola do navegador que permite enviar mensaxes á mesma.
  - `document`: referencia ao obxecto `document` da xanela.
  - `history`: referencia ao obxecto `history` da xanela.
  - `location`: referencia ao obxecto `location` da xanela.
  - `navigator`: referencia ao obxecto `navigator` da xanela.
  - `screen`: referencia ao obxecto `screen` da xanela.
- **Almacenaxe no cliente:**
  - `localStorage`: permite almacenar pares clave/valor na aplicación cliente, sen data de expiración.

    - `sessionStorage`: permite almacenar pares clave/valor na aplicación cliente, durante a sesión actual.
- **Referencias:**
    - `opener`: referencia a xanela que abriu a actual.
    - `parent`: referencia a xanela nai do `frame` actual.
    - `self`: referencia a xanela actual.
    - `top`: referencia a xanela que está no tope da xerarquía.

```
// Obter información da xanela
var info = "Screen size = " + screen.availWidth +" x "+
screen.availHeight + " px";
info += "<br/>Outer size = " + window.outerWidth +" x "+
window.outerHeight + " px";
info += "<br/>Inner size = " + window.innerWidth +" x "+
window.innerHeight + " px";
info += "<br/>Position = " + window.screenLeft +" x "+
window.screenTop + " px";
document.getElementById("info").innerHTML = info;
```

Métodos

- **`alert()`:** amosa unha xanela cunha mensaxe e un botón de OK.
- **`atob()`:** decodifica unha *string* codificado en base 64.
- **`blur()`:** quita o foco da xanela actual.
- **`btoa()`:** codifica unha *string* en base 64.
- **`clearInterval()`:** borra o temporizador establecido con `setInterval()`.
- **`clearTimeout()`:** borra o temporizador establecido con `setTimeout()`.
- **`close()`:** pecha a xanela actual.
- **`confirm()`:** amosa unha xanela de diálogo cunha mensaxe e 2 botóns (OK e Cancelar).
- **`focus()`:** establece o foco na xanela actual.
- **`moveBy()`:** move a xanela relativa á posición actual.
- **`moveTo()`:** move a xanela a unha posición absoluta.
- **`open()`:** abre unha xanela do navegador nova.
- **`print()`:** imprime o contido da xanela actual.
- **`prompt()`:** amosa unha xanela de diálogo na que o usuario debe introducir información.
- **`resizeBy()`:** cambia o tamaño da xanela actual en base aos pixels da actual.
- **`resizeTo()`:** establece o tamaño (largo x alto) da xanela actual á cantidade de pixels indicada.

- **`scrollBy()`:** despraza o documento unha cantidade de pixels en base á posición actual.
- **`scrollTo()`:** despraza o documento ás coordenadas indicadas.
- **`setInterval()`:** chama a unha función ou avalía unha expresión a intervalos constantes (en milisegundos).
- **`setTimeout()`:** chama a unha función ou avalía unha expresión transcorridos uns milisegundos.
- **`stop()`:** detén a carga de contido na xanela.

```
// Pedimos un dato ao usuario
var seconds = window.prompt("Indica unha cantidade de segundos: ",
"5");
// Executa unha función cada 1000 ms
var myTimer = setInterval(showTimer, 1000);

function showTimer() {
  document.getElementById("info").innerHTML = new
Date().toLocaleTimeString();
  seconds--;
  if (seconds==0) {
    // Deixa de executar a función ao borrar o temporizador
    clearInterval(myTimer);
    // Amosa unha xanela emerxente cunha mensaxe
    window.alert("Rematou o tempo");
  }
}
```

Xestión

Desde o código JS nunca se poderá crear a xanela principal do navegador, xa que este código terá que estar cargado previamente nunha xanela inicial aberta polo usuario. A partires desa situación poderanse abrir subxanelas novas co método `open()`, o cal ten a seguinte sintaxe:

```
window.open(url, target, features, replace);
```

Parámetros

Todos os parámetros da función `open()` son opcionais:

- **`url`:** indica a URL do recurso que desexamos cargar na xanela. Se a deixamos en branco, abrirase unha xanela co contido baleiro `about:blank`.

- **`target`:** establece o destino no que se cargará a nova xanela ou o nome da mesma (para poder referila posteriormente):
    - `_blank`: a URL cárgase nunha nova xanela/lapela. É a opción por defecto.
    - `_parent`: a URL cárgase no marco pai da actual.
    - `_self`: a URL substitúe a páxina actual.
    - `_top`: a URL substitúe calquera conxunto de marcos que poidan estar cargados.
    - `name`: nome da nova xanela.
- **`features`:** trátase dunha listaxe de valores que permiten establecer propiedades da nova xanela. A listaxe debe ir delimitada por comiñas (""), separando cada par/valor mediante comas (,) e sen incluír espazos en branco entre elementos. A continuación algunhas das de uso máis habitual[2]:
    - **`height=pixels`:** alto da xanela en pixels. O mínimo é 100.
    - left=pixels. Posición con respecto á marxe esquerda da pantalla.
    - **`location=yes|no|1|0`:** amosar a barra de direccións (só en Opera).
    - **`menubar=yes|no|1|0`:** amosar a barra de menús.
    - **`resizable=yes|no|1|0`:** permitir redimensionar a xanela (só en Internet Explorer).
    - **`scrollbars=yes|no|1|0`:** amosar barras de desprazamento (só en Internet Explorer, Firefox e Opera).
    - **`status=yes|no|1|0`:** amosar a barra de estado.
    - **`titlebar=yes|no|1|0`:** amosar a barra de título.
    - **`toolbar=yes|no|1|0`:** amosar a barra de ferramentas (só en Internet Explorer e Firefox).
    - **`top=pixels`:** posición con respecto ao tope superior da pantalla.
    - **`width=pixels`:** largura da xanela en pixels. O mínimo é 100.
- **`replace`:** indica se creamos unha entrada nova no historial (`false`) ou se substitúe a actual (`true`).

[2] Algunhas das características anteriores non funcionan coa configuración por defecto de Chrome, Firefox, IE, Edge, Opera e Safari.

Exemplo

A seguinte instrución abriría unha xanela nova de 800x600 pixels, sen barra de estado e na que se carga o contido dunha páxina HTML xa creada que se atopa na mesma ubicación que a actual (new.html).

```
var  subWindow = window.open("new.html", "new",
"height=600,width=800");
```

Se desexamos omitir algún dos parámetros anteriores, basta con indicar comiñas dobres ("") no que corresponda.

Dentro do código pódese facer referencia á mesma mediante `new`, e así acceder ás súas propiedades e métodos co gaio manexala a través da variable `subWindow`. Así por exemplo, para pechar a xanela aberta, empregaríase o método `close()`:

```
subWindow.close();
```

Un exemplo máis elaborado sería:

```
// Abre unha páxina xa creada nunha subxanela dun tamaño concreto
var  subWindow = window.open("new.html", "_blank",
"width=400,height=300,status=no,location=yes,scrollbars=0,titlebar
=1", "true");

// Comproba se a subxanela foi creada (p.ex: non bloqueada por ser
emerxente)
if (!subWindow.closed) {
  // Redimensiona a xanela á metade do tamaño da pantalla do
usuario
  subWindow.moveTo(0, 0);
  subWindow.resizeTo(screen.availWidth/2, screen.availHeight/2);

  // Obtén información da subxanela
  var info = "Screen size = " + screen.availWidth +"x"+
screen.availHeight +" px"+
    "<br>Subwindow size = " + subWindow.outerWidth +"x"+
subWindow.outerHeight + " px";
  document.getElementById("info").innerHTML = info;
}
```

Document

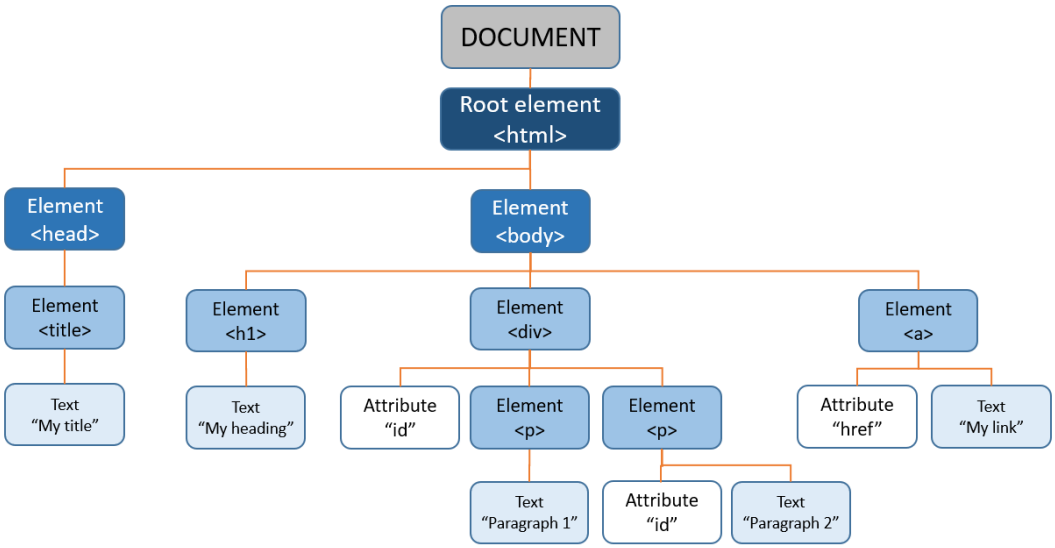
Cando se carga unha páxina HTML no navegador, este crea unha estrutura xerárquica en forma de árbore que a representa, o que se coñece como a árbore DOM. Como xa sabemos, DOM é un estándar do W3C (World Wide Web Consortium) que

fornece unha interface que permite, por medio de programación, acceder e actualizar dinamicamente o contido, estrutura e estilo dun documento, e que define:

- Os elementos HTML como obxectos.
- Propiedades dos elementos HTML.
- Métodos para manexar os elementos HTML.
- Eventos para os elementos HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      My title
    </title>
  </head>
  <body>
    <h1>My heading</h1>
    <div id="layer1">
      <p>Paragraph 1</p>
      <p class="p2">Paragraph 2</p>
    </div>
    <a href="#">My link</a>
  </body>
</html>
```

Para o documento HTML do exemplo anterior teriamos a seguinte árbore DOM:



Facendo cambios sobre os nodos da estrutura anterior mediante JS podemos fornecer capacidades dinámicas ao contido HTML, tales coma:

- Modificar os nodos da páxina HTML (elementos, atributos e textos).
- Modificar os estilos CSS da páxina.
- Engadir e eliminar nodos (elementos, atributos e textos).

- Reaccionar ante eventos producidos sobre a páxina HTML.
- Rexistrar eventos novos.

Propiedades

- **activeElement:** elemento que ten o foco actualmente no documento.
- **anchors:** colección de enlaces do documento (`<a>`) que teñen o atributo `name`.
- **applets:** colección de applets (`<applet>`) do documento.
- **body:** elemento `<body>` do documento.
- **cookie:** todos os pares nome/valor das cookies do documento.
- **characterSet:** xogo de carácteres en uso no documento.
- **defaultView:** obxecto `window` asociado ao documento.
- **doctype:** declaración do tipo de documento asociado ao actual.
- **documentElement:** elemento `<html>` do documento.
- **domain:** nome do dominio do servidor que serviu o documento.
- **embeds:** colección de elementos `<embed>` do documento.
- **forms:** colección de elementos `<form>` do documento.
- **head:** elemento `<head>` do documento.
- **images:** colección de elementos `<img>` do documento.
- **inputEncoding:** xogo de caracteres establecido para o documento.
- **links:** colección de elementos `<a>` e `<area>` do documento que teñan atributo `href`.
- **readyState:** estado do documento relativo á súa carga no navegador.
- **referrer:** URL do documento que cargou o actual.
- **scripts:** colección de elementos `<script>` do documento.
- **title:** elemento `<title>` do documento.
- **URL:** URL completa de acceso ao documento HTML.

Métodos

- **addEventListener():** asigna un manexador de eventos ao elemento especificado.
- **close():** pecha o fluxo de saída aberto previamente con `document.open()`.
- **createAttribute():** crea un nodo de tipo atributo.
- **createComment():** crea un nodo de tipo comentario co texto indicado.
- **createElement():** crea un nodo de tipo elemento.
- **createEvent():** crea un evento novo.
- **createTextNode():** crea un nodo de tipo texto.

- **`getElementById()`:** devolve o elemento cuxo atributo ID é igual ao valor indicado.
- **`getElementsByClassName()`:** devolve un `NodeList` que conterá todos os elementos co nome de clase igual ao valor indicado.
- **`getElementsByName()`:** devolve un `NodeList` que conterá todos os elementos co nome igual ao valor indicado.
- **`getElementsByTagName()`:** devolve un `NodeList` que conterá todos os elementos co nome da etiqueta igual ao valor indicado.
- **`hasFocus()`:** indica se o documento ten o foco (se fai click dentro ou se selecciona algún elemento do mesmo co tabulador).
- **`open()`:** abre un fluxo de saída HTML para captar a saída de `document.write()`.
- **`querySelector()`:** devolve o primeiro elemento do documento que coincida co selector CSS indicado.
- **`querySelectorAll()`:** devolve un `NodeList` con todos os elementos do documento que coinciden co selector CSS indicado.
- **`removeEventListener()`:** elimina un manexador de eventos do documento (asignado previamente con `document.addEventListener()`).
- **`renameNode()`:** renomea o nodo correspondente.
- **`write()`:** escribe código HTML/JavaScript no documento.
- **`writeln()`:** escribe código HTML/JavaScript no documento, engadindo un caracter de nova liña ao final.

```
document.write("<html><head><title>JavaScript</title></head><body>
<h1>Document</h1><p>Contido HTML xerado dende JS</p><button
onclick='crear()'>Crear!</button></body></html>");

function crear() {
  var par = document.createElement("P");
  var text = document.createTextNode("Nodo de parágrafo
dinámico.");
  par.appendChild(text);
  document.body.appendChild(par);
}
```

Exemplo

Segundo se viu no modelo do BOM, o obxecto `document` ten unha serie de *arrays* que actúan coma coleccións de elementos HTML. Para acceder a eles, pódense empregar tanto os métodos do DOM como os propios obxectos do BOM, do seguinte xeito:

- Se temos o seguinte documento HTML:

```
<html>
  <head>
    <title>Meu título</title>
  </head>
  <body>
    <h1>O meu título</h1>
    <a href="paxina_externa.html">O meu enlace</a>
    <img src="logo.png" name="logo" />
    <form method="post" name="rexistro">
      <input type="text" name="id" />
      <input type="submit" value="send" />
    </form>
  </body>
</html>
```

- Podemos acceder aos elementos dende JS, por exemplo así:

```
// Acceso empregando métodos do DOM
document.getElementByTagName("h1");

// Acceso empregando obxectos do BOM
document.links[0];
document.images[0];
document.images["logo"];
document.forms[0];
document.forms["rexistro"];

// Acceso ás propiedades dun obxecto
document.forms[0].method;
document.images["logo"].src;
```

Screen

Trátase dun obxecto que non posúe métodos de seu, senón que contén información refererente á aparencia e características de visualización da pantalla do usuario, isto é, manéxase exclusivamente a través de propiedades.

Propiedades

- **`availHeight`:** altura da pantalla excluíndo a barra de tarefas (en pixels).

- **availWidth:** largura da pantalla excluíndo a barra de tarefas (en pixels).
- **colorDepth**[3]**:** profundidade de cor da paleta de cores do navegador (en bits).
- **height:** altura total da pantalla (en pixels).
- **pixelDepth:** profundidade de cor da pantalla (en bits).
- **width:** largura total da pantalla (en pixels).

```
// Redimensionar a xanela ao tamaño máximo segundo a pantalla do
usuario
window.moveTo(0, 0);
window.resizeTo(screen.availWidth, screen.availHeight);
```

Navigator

O obxecto `navigator` fornece información relativa ao estado e identidade dos axentes de usuario (o navegador e o sistema operativo dende os cales é accedida a páxina actual).

Entre os seus usos habituais atopamos:

- Determinar o tipo/versión do navegador en aplicacións cuxo código difire segundo o cliente.
- Detectar se o navegador ten habilitadas características requiridas polas aplicacións: Java, *cookies*, plugins dispoñibles.

Hai que ter coidado coa información obtida a través desde obxecto posto que pode levar a conclusións erróneas polos seguintes motivos:

- Diferentes navegadores poden empregar o mesmo nome.
- A información do navegador pode ser cambiada polo propietario do mesmo.
- Algúns navegadores mudan a súa identificación intencionadamente para enganar aos servidores.
- Os navegadores non poden informar do uso de sistemas operativos novos, lanzados despois do xurdimento da versión que esteamos a empregar do navegador.

Propiedades

- **appCodeName:** nome en código do navegador[4].
- **appName:** nome que representa o tipo de navegador.

[3] Nos ordenadores modernos, o `colorDepth` e o `pixelDepth` son equivalentes.

[4] Por motivos de retrocompatibilidade, todos os navegadores modernos devolven o código Mozilla para se referir aos navegadores de tipo Firefox.

- **`appVersion`:** información da versión do navegador.
- **`cookieEnabled`:** indica se as *cookies* están habilitadas no navegador.
- **`geolocation`:** este obxecto pode ser empregado para determinar a localización xeográfica do usuario.
- **`language`:** idioma do navegador.
- **`onLine`:** indica se o navegador está conectado.
- **`platform`:** indica a arquitectura de procesador para a cal foi compilado o navegador.
- **`plugins`:** listaxe de *plugins* instalados no navegador.
- **`product`:** nome do motor do navegador.
- **`userAgent`:** cabeceira enviada polo axente de usuario ao servidor.

```
var x = navigator.userAgent;
var y = navigator.platform;
```

Métodos

- **`javaEnabled()`:** indica se está habilitada no navegador a execución de código Java.

```
var z = navigator.javaEnabled();
```

Location

Este obxecto permite acceder e manipular as partes da URL que corresponden á paxina actual, así coma redirixir o navegador a unha ubicación nova. Por mor da falta de estandarización, pódese acceder a este obxecto tanto a través de `window` como de `document`:

```
window.location;
document.location; // É equivalente ao anterior
```

Propiedades

Para entender mellor as propiedades deste obxecto, partiremos da seguinte URL de exemplo:

```
http://
www.dominio.gal:8080/ruta/subruta/paxina.php#seccion?key1=valor1
```

- **hash:** parte da URL que corresponde co *anchor* (`#`).

```
#seccion
```

- **host:** nome do *host* e número de porto.

```
www.dominio.gal:8080
```

- **hostname:** nome do *host*.

```
www.dominio.gal
```

- **href:** a URL completa. Se escribimos nesta propiedade, provocamos que o navegador se redirixa á nova ubicación.

```
http://
www.dominio.gal:8080/ruta/subruta/paxina.php#seccion?key1=valor1
```

- **origin:** protocolo, nome do *host* e número de porto da URL.

```
http://www.dominio.gal:8080
```

- **pathname:** ruta do recurso ao que se está a acceder no servidor.

```
/ruta/subruta/paxina.php
```

- **port:** número de porto da URL.

```
8080
```

- **protocol:** protocolo que se está a empregar para acceder ao recurso.

```
http:
```

- **search:** cadea de busca dentro da URL. Normalmente o que se indica despois dunha interrogación pechada (`?`).

```
?key1=valor1
```

Métodos

- **`assign()`:** carga un novo documento.

- **`reload()`:** recarga o documento actual, sendo equivalente a facer *click* no botón de actualizar do navegador.
- **`replace()`:** substitúe o documento actual cun novo. É semellante ao método `assign()`, pero o sitio substituído elimínase do historial da sesión.

```
location.assign("https:// www.xunta.gal/portada"); // Redirixe o navegador a outra URL
location = "https:// www.xunta.gal/portada"; // Máis un xeito de facer o mesmo
```

History

Este obxecto rexistra e permite xestionar o historial da sesión do navegador, isto é, as páxinas visitadas polo usuario na xanela ou `frame` na que está cargada a páxina actual.

Propiedades

- **`length`:** número de URL visitadas, incluída a actual.

Métodos

- **`back()`:** carga a URL previa do historial.
- **`forward()`:** carga a URL seguinte do historial.
- **`go()`:** vai a unha URL concreta, relativa á posición no historial da actual.

```
var x = history.length; // 1 (ao abrir a xanela por vez primeira)
history.back(); // Vai á páxina anterior no historial
history.go(-1); // Máis un xeito de facer o mesmo
```

Almacenaxe en cliente

Como xa sabemos, HTTP é un protocolo non orientado a conexión, o que quere decir que unha vez que o servidor web remata de enviar a páxina solicitada, este pecha a conexión e non lembrará ao usuario en peticións posteriores. Para resolver esta característica do protocolo predominante na Web (que tivo os seus motivos fundacionais pero que hoxe en día limita notablemente a funcionalidade do servizo) e, por exemplo, manter a sesión de usuario namentres este navega por un sitio web, fíxose precisa a implementación de mecanismos de almacenaxe no lado do cliente co gaio de permitir gardar pequenas porcións de información asociadas ás peticións que se fixeron dende un navegador concreto cara un certo servidor.

Co paso do tempo foron aparecendo distintos métodos para gardar información do usuario entre distintas solicitudes de páxinas web, como poden ser:

- **Cookies:** foron a primeira solución de almacenaxe no cliente por parte do navegador. Tamén constitúen a solución máis limitada e rudimentaria, aínda que se manteñen en uso até hoxe en día (sendo o mecanismo máis habitual deste tipo) grazas á súa simplicidade.
- **WebStorage:** considérase o substituto natural das *cookies*, xa que tamén permite almacenar información en pares clave/valor, pero ampliando a cantidade de información que se pode almacenar á orde dos MB, no canto dos KB que permitían as Cookies.
- **Atributos HTML5:** a última versión de HTML permite anexar atributos de datos as elementos do DOM, de xeito que estes poden ser incluídos dende o servidor para ser accedidos dende o código cliente (p.ex.: JavaScript).
- **Indexed DB:** permite almacenar datos estruturados e realizar procuras eficientes sobre eles. Tamén amplía aínda máis a cantidade de información que se pode almacenar (chegando á orde dos GB).
- **Websockets:** solución incluída en HTML5 que permite establecer con JavaScript unha comunicación orientada a conexión bidireccional entre o cliente e o servidor. Con eles resólvense principalmente as situacións nas que a información debe de se transmitir en resposta a eventos producidos no servidor.

Cookies

As *cookies* son pequenos ficheiros de texto asociados á comunicación dun navegador cun mesmo dominio aloxado nun servidor. Neses ficheiros gárdanse pares clave/valor que poden ser accedidos dende JS mediante a propiedade `document.cookie`.

O seu obxectivo é o de permitir lembrar información acerca do usuario (nome, preferencias, etc.), resolvendo así a desvantaxe derivada do feito de que o protocolo HTTP sexa non orientado a conexión.

Entre outras características, as *cookies* admiten o establecemento dunha data de expiración que, no caso de non ser fixada, indicaría que a mesma caducará no intre en que se peche o navegador.

Cando un navegador solicita unha páxina web a un servidor, as *cookies* asociadas ao documento solicitado son engadidas á devandita petición. Destea maneira o servidor obtén os recursos necesarios para recordar a información do usuario ao cal pertence a sesión.

Así mesmo, as *cookies* presentan unha serie de limitacións, tales coma:

- **Tamaño máximo por *cookie*:** na maioría de navegadores é de 4 KB (se ben o soporte a 8KB estase a estender na actualidade).
- **Número máximo de *cookies* por sitio web:** na maioría de navegadores é de 20 *cookies*.
- **Número máximo de *cookies* por dominio:** na maioría de navegadores é de 50 *cookies*.
- **Número máximo de *cookies* por navegador:** este dato varía notablemente entre navegadores e dispositivos, pero adoita estar en torno ás 3.000 *cookies*.

En relación co anterior, o uso de *cookies* para a conservación de información de sesión conleva tamén unha serie de desvantaxes:

- A acumulación de *cookies*, ademais de consumir espazo de almacenaxe, pode resultar na ralentización do navegador.
- As *cookies*, en tanto ficheiros de texto plano, poden ser facilmente accedidas e empregadas para desvelar información confidencial ou alterar os datos contidos nas mesmas se non son manexadas e almacenadas coidadosamente.
- O cifrado de *cookies*, se ben mellora o manexo seguro das mesmas, afecta de xeito notable ao rendemento do navegador que ten que manexalas.

Operacións

Accedendo á propiedade `document.cookie`, pódense levar a feito as operacións típicas de xestión deste tipo de elementos:

- **Creación:** faise asignando unha *string* que conteña a información axeitada coa seguinte sintaxe:

```
document.cookie = cookieString;
```

  O parámetro `cookieString` será un texto que conterá unha lista de pares clave/valor separados por punto e coma (`;`), ou un só par clave/valor con algún dos seguintes valores opcionais:

  - **`expires=data`:** indica a data en formato GMT (ver método `Date.toUTCString()`). Se non se inclúe, a *cookie* borrarase cando se peche o navegador.
  - **`max-age=segundos`:** establece a duración máxima en segundos. Se non se inclúe, a *cookie* borrarase cando sexa pechado o navegador. Este parámetro ten preferencia sobreo anterior (`expires`).
  - **`path=ruta`:** sinala o directorio no que se almacenará a *cookie* (p.ex: "`/`", "`/cookies`"). A ruta debe indicarse de xeito absoluto. Se non se indica a *cookie* pertence á páxina actual.
  - **`domain=nomeDominio`:** especifica o dominio do sitio (p.ex: "`domain.com`", "`subdomain.domain.com`"). Se non se indica, interprétase que será o dominio da páxina actual.
  - **`secure`:** especifica que vai empregarse o protocolo web seguro HTTPS para enviar a *cookie* ao servidor.

```
document.cookie="username=Xiao; expires=Tue, 9 Nov 2021 10:39:00
UTC; path=/";
```

A propiedade `document.cookie` pode parecer unha *string* normal, pero ao lela só se visualizan os pares clave/valor:

```
"username=Allen;"
```

Ao crear unha *cookie* nova non se sobrescriben a existentes senón que esta é engadida ao resto en `document.cookie`:

```
document.cookie="cookie1=valor1;";
document.cookie="cookie2=valor2;";
```

```
console.log(document.cookie); // cookie1=valor1; cookie2=valor2;
```

NOTA: o valor dunha *cookie* non pode conter comas (,), puntos e coma (;) ou espaczos en branco ( ). É útil empregar o método `encodeURIComponent()` para evitar conflitos neste sentido.

- **Acceso:** para obter as *cookies* dun documento operaremos así:

```
var c = document.cookie;
```

O anterior devolverá unha *string* do estilo:

```
"cookie1=valor1; cookie2=valor2; cookie3=valor3;"
```

Como xa mencionamos, accedendo á propiedade `document.cookie` obtéñense todos os pares clave/valor. Para acceder a unha *cookie* concreta haberá que empregar código JavaScript (p.ex: empregando o método `String.split()`) para fragmentar a *string* completa e poder acceder a cada par de modo individualizado.

- **Modificación:** pódese mudar o valor dunha *cookie* xa existente do mesmo xeito que se creou, xa que se sobrescribirá o valor orixinal:

```
document.cookie="username=Dombodán;  expires=Mon,  8  Nov  2021
12:05:57 UTC; path=/";
```

- **Borrado:** para eliminar unha *cookie* basta con coñecer o seu nome (non é necesario coñecer o seu valor) e, ou ben especificar unha data xa pasada no parámetro `expires`, ou ben unha duración de 0 segundos en `max-age`:

```
document.cookie  =  "username=;  expires=Thu,  01  Jan  1970  00:00:00
UTC; path=/;";
document.cookie = "username=; max-age=0; path=/;";
```

WebStorage

Trátase dunha API (*Application Programming Interface*) que xurdiu canda HTML5 como alternativa ás *cookies* para almacenar datos no cliente asociados á sesión do usuario ou permanentes.

O uso de `WebStorage` mellora a seguridade con respecto ás *cookies*, ampliando ademais a cantidade de información que se pode almacenar, sen afectar ao rendemento dos sitios web.

Consiste na emprega dun repositorio común por orixe (mesmo dominio e protocolo), o que supón que todas as páxinas da mesma orixe poderán almacenar e acceder aos mesmos datos.

Propiedades

`WebStorage` fornece dous obxectos de tipo `Storage` contidos en `Window`, para tratar a almacenaxe de distintos xeitos:

- **`sessionStorage`:** almacena os datos durante unha sesión (pérdense cando se pecha a ventá do navegador).
- **`localStorage`:** almacena os datos sen data de caducidade.

Denantes empregar `WebStorage` é preciso verificar que o navegador sopórtao:

```
if (typeof(Storage) !== "undefined") {
  // código usando WebStorage
} else {
  // Código se non hai soporte para WebStorage
}
```

Propiedades

- **`length`:** esta propiedade contén o número total de elementos almacenados no obxecto `Storage`.

Métodos

A continuación os métodos que o obxecto `WebStorage` pon á nosa disposición:

- **`setItem()`:** permite engadir un par clave/valor á almacenaxe cliente correspondente ou actualizala no caso de que a clave xa exista.
- **`getItem()`:** devolve o valor da clave indicada polo seu nome.
- **`key()`:** devolve o nome da clave indicada pola súa orde (comezando por 0).
- **`removeItem()`:** elimina o elemento cuxo nome coincide co indicado.
- **`clear()`:** baleira a almacenaxe cliente correspondente para a orixe actual.

```
localStorage.setItem("accesstime", new Date()); //
localStorage.accesstime = new Date();
localStorage.getItem("accesstime"); // Mon Nov 18 2019 14:06:42
GMT+0100
localStorage["accesstime"]; // Mon Nov 18 2019 14:06:42 GMT+0100
localStorage.accesstime; // Mon Nov 18 2019 14:06:42 GMT+0100
localStorage.key(0); // accesstime
localStorage.removeItem("accesstime");
localStorage.clear();
```

```
localStorage.length; // 0
```

Arrays

As estruturas de datos existen nas linguaxes de programación de alto nivel (JS incluída) para permitir, entre outras cousas, gardar información máis complexa e heteroxénea daquela que poden conter os tipos de datos elementais. Podemos, polo tanto, considerar as estruturas de datos coma coleccións de datos (elementais ou compostos) que se caracterizan pola súa organización interna e as operacións que se poden facer cos datos que almacenan.

Actividade

Cando desenvolvemos aplicacións é habitual que teñamos que almacenar listas de elementos relacionados. Un exemplo poderían ser as estacións do ano:

```
var estacion1 = "Primavera";
var estacion2 = "Verán";
var estacion3 = "Outono";
var estacion4 = "Inverno";
```

No caso anterior, atopariámonos cun problema se tiveramos, por exemplo, que facer comparacións con todas as variables anteriores para comprobar a coincidencia cun valor seleccionado. Este feito agravaríase se, no canto de estacións, fosen os meses do ano ou os días do mes.

Para facilitar o tratamento destes datos nos que pode haber varios elementos relacionados, as linguaxes de programación dispoñen dun tipo de dato especial, os *arrays*. A característica principal dos *arrays* é que poden almacenar varios valores baixo un mesmo nome, podendo estes ser do mesmo ou diferente tipo, e simples ou compostos (como outros *arrays*, obxectos, funcións, etc.).

```
var estacions; // Variable que será de tipo array
```

En resumo, un *array* ou matriz é unha colección de elementos ordenados nunha ou máis dimensións.

Creación

Hai 2 xeitos principais para crear *arrays*:

- **Empregando a palabra clave new:**
    - Xa sexa para crear un *array* baleiro:

```
var estacions = new Array();
estacions instanceof Array; // true
```

- Ou un *array* a partir dos elementos que van compoñelo:

```
var estacions = new Array("Primavera", "Verán", "Outono",
"Inverno");
```

- **De forma literal empregando corchetes (`[]`).**
  - Unha vez máis, xa para crear un *array* baleiro:

```
var estacions = [];
estacions instanceof Array; // true
```

- Ou un *array* co seu contido xa definido:

```
var estacions = ["Primavera", "Verán", "Outono", "Inverno"];
```

Empregando calquera das dúas formas anteriores conseguimos o mesmo.

Acceso

Os elementos dun *array* en JS son accedidos mediante o seu índice, isto é, o seu número de orde na colección que é o propio *array*. O devandito índice indícase entre corchetes, sendo sempre o primeiro índice o cero.

```
estacions[0]; // Primavera
estacions[3]; // Inverno
estacions[estacions.length-1]; // Inverno
```

Así, unha forma equivalente de crear o *array* de estacións do ano sería crear primeiro o *array* baleiro e, posteriormente, ir engadindo os elementos:

```
var estacions = new Array();
estacions[0] = "Primavera";
estacions[1] = "Verán";
estacions[2] = "Outono";
estacions[3] = "Inverno";
```

Desta forma tamén se poderán percorrer todos os elementos dun *array* empregando un bucle, sen máis que modificar o índice de acceso:

```
// Mediante un bucle WHILE
var i=0;
while (i<=estacions.length-1) {
  console.log(estacions[i]);
  i++;
}

// Mediante un bucle FOR
for (var i=0; i< estacions.length; i++) {
  console.log(estacions[i]);
}

// Mediante un bucle FOR-IN
for (i in estacions) {
  console.log(estacions[i]);
}

// Mediante un bucle FOR-OF
for (var estacion of estacions) {
  console.log(estacion);
}
```

Eliminación

A primeira forma empregada en JS para eliminar elementos de *arrays* era a asignación ás posicións correspondentes do valor `null`, o que non constitúe unha verdadeira eliminación, xa que a posición continúa a existir na colección (é o que se coñece en programación coma unha «eliminación preguiceira»). Este procedemento, que se ben ten a vantaxe da súa simplicidade, implica que a partires dese intre o dato da lonxitude do *array* non coincida co número de elementos realmente contidos na colección. Velaquí un exemplo:

```
estacions.length; // 4
estacions[2] = null;
estacions.length; // 4
estacions[2]; // null
```

Versións posteriores de JS incorporaron o operador `delete`, que permite ao intérprete da linguaxe liberar memoria cando a precise. Este método eliminará o índice do *array*, pero non reducirá a súa lonxitude:

```
delete estacions[2];
estacions.length; // 4
```

```
estacions [2]; // undefined
```

O máis axeitado é, pois, empregar os métodos destinados a tal efecto que trataremos no punto seguinte.

Tipos

Segundo a cantidade de dimensións e o xeito en que os empregamos podemos clasificar os arrays en:

- **Arrays unidimensionais:** aqueles que almacenan elementos simples, como os vistos ata o de agora:

```
var estacions = ["Primavera", "Verán", "Outono", "Inverno"];
```

- **Arrays paralelos:** esta técnica baséase en almacenar en varios *arrays* de unha única dimensión información relacionada entre si:

```
var estacions = ["Primavera", "Verán", "Outono", "Inverno"];
var temperatura = [15, 25, 20, 10]; // Temperatura máxima
var chuvia = [25, 10, 30, 20]; // Precipitación media en l/m2
```

  Deste xeito, empregando o mesmo índice conseguimos acceder aos datos da mesma estación do ano (p.ex: o índice 2 vainos dar en todos os *arrays* os datos correspondentes Outono):

```
estacions[2] +": "+ temperatura[2] +"°C, "+ chuvia[2] +"mm"; //
Outono: 20°C, 30mm
```

  Podemos, así mesmo, extrapolar unhas características comúns aos *array* membros dun mesmo grupo de *arrays* paralelos:
  - Deben ter todos a mesma lonxitude.
  - Débese manter unha orde estrita para que os datos non muden o seu significado.
  - Permiten simular *arrays* multidimensionais.
  - Son doados de interpretar visualmente.
  - Permiten facer un percorrido dos datos máis rápido.
- **Arrays multidimensionais:** en JS todos os *arrays* son teoricamente unidimensionais, pero como cada elemento dun *array* pode ser de calquera tipo (mesmo *arrays*), entón faise posible «simular» a multidimensionalidade.

```
var clima = new Array();
```

```
clima[0] = ["Primavera", "Verán", "Outono", "Inverno"];
clima[1] = [15, 25, 20, 10]; // Temperatura máxima
clima[2] = [25, 10, 30, 20]; // Precipitación media en l/m2
```

Neste caso, para simular unha estrutura bidimensional, tamén deben ser todos os *arrays* da mesma lonxitude e, para poder acceder aos datos da mesma estación do ano débese empregar o mesmo índice secundario (p.ex: o índice secundario 2 vainos dar, no *array* bidimensional, os datos correspondentes ao Outono):

```
clima[0][2] +": "+ clima[1][2] +"°C, "+ clima[2][2] +"mm"; //
Outono: 20°C, 30mm
```

Este tipo de *arrays* son moito máis recomendables para programar o acceso aos seus elementos (p.ex: para recorrer secuencialmente todos os elementos mediante un bucle aniñado):

```
for (var i=0; i<clima.length; i++) { // Percorre a 1ª dimensión
  for (var j=0; j< clima[i].length; j++) { // Percorre a 2ª
dimensión
    console.log(clima[i][j]);
  }
}
```

Métodos

Unha vez vistas as operacións básicas con *arrays*, o máis importante para traballar con eles é coñecer o funcionamento dos seus métodos, xa que van ser os que nos permitan aproveitar ao máximo as posibilidades dos mesmos:

- **Edición:** atopamos nesta categoría aqueles métodos que permiten inserir/extraer elementos do *array*, xa sexa modificando ou non o orixinal:
  - `pop()`: elimina o último elemento dun *array* modificando este, ao tempo que devolve o elemento eliminado.

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.pop(); // 24
console.log(idades); // [40, 17, 40, 85]
```

  - `push()`: engade un novo elemento no remate do *array*, devolvendo a nova lonxitude.

```
var idades = [40, 17, 40];
```

```
var resultado = idades.push(85, 24); // 5
console.log(idades); // [40, 17, 40, 85, 24]
```

- `shift()`: elimina o primeiro elemento do *array* modificando este. O seu valor de retorno é o propio elemento eliminado.

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.shift(); // 40
console.log(idades); // [17, 40, 85, 24]
```

- `unshift()`: engade un elemento ao principio do *array* ao tempo que devolve a nova lonxitude.

```
var idades = [40, 17, 40];
var resultado = idades.unshift(85, 24); // 5
console.log(idades); // [85, 24, 40, 17, 40]
```

- `slice()`: extrae un sub-*array* formado polos elementos indicados entre polas posicións dos argumentos inicio e fin (non incluído). Pódense indicar posicións negativas para comezar polo final do array.

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.slice(1,3); // [17, 40]
var resultado = idades.slice(-2); // [85, 24]
```

- `splice()`: permite eliminar elementos do *array* en posicións concretas e inserir secuencias de elementos no seu lugar. O primeiro parámetro recibido indica a posición e o segundo ao número de elementos a eliminar. Os parámetros subseguintes son os elementos que queremos engadir. O seu valor de retorno será un *array* cos elementos eliminados.

```
var idades = [40, 17, 24];
var resultado = idades.splice(2, 1, 40, 85); // [24]
console.log(idades); // [40, 17, 40, 85]
```

- `concat()`: une dous o máis *arrays* devolvendo o resultado tamén en forma de *array*.

```
var android = ["samsung", "huawei", "xiaomi"];
var outros = ["iphone"];
var smartphones = android.concat(outros);
```

- **Busca e conversión:** agrupamos aquí aqueles métodos que permiten procurar dentro do *array*, ordenalo, representalo en forma de *String* e obter información do mesmo. Son os seguintes:
    - `includes()`: determina se o array inclúe un valor concreto.

```
var idades = [40, 17, 85, 24];
var resultado = idades.includes(85); // true
```

    - `indexOf()`: procura no *array* o elemento indicado e devolve a súa posición (ou `-1` se non o atopa). Pódese indicar, opcionalmente, unha posición de inicio da procura.

```
var idades = [40, 17, 85, 40, 24];
var resultado = idades.indexOf(40, 2); // 3
```

    - `lastIndexOf()`: procura no *array* o elemento indicado comezando polo final ou, opcionalmente, nunha posición indicada *ad hoc*. Devolve a súa posición do elemento (ou `-1` se non o atopa).

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.lastIndexOf(40, 3); // 2
```

    - `toString()`: devolve unha *string* con todos os valores do *array* separados por comas (`,`).

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.toString(); // 40, 17, 40, 85, 24
```

    - `join()`: devolve os elementos dun *array* concatenados nunha *string*. Os elementos irán unidos por un separador (opcional), empregándose por defecto a coma (`,`).

```
var idades = [40, 17, 85, 40, 24];
var resultado = idades.join("|"); // "40|17|85|40|24"
```

- `sort()`: ordena os elementos do *array*, empregando ordenación alfabética ascendente por defecto. No caso de querermos organizar segundo un criterio diferente, deberemos fornecer unha función *ad hoc*, de xeito que a mesma reciba dous parámetros argumentos e devolva un número (positivo, cero ou negativo) que a relación entrambos (maior, igual ou menor).

```
var letras = ["beta", "gamma", "alpha", "delta"];
letras.sort(); // ["alpha", "beta", "delta", "gamma"]

var idades = [40, 17, 24, 5, 110];
idades.sort(); // [110, 17, 24, 40, 5]
idades.sort( function (a,b) {return a-b;} ); // [5, 17, 24, 40,
110]
```

- `reverse()`: inverte a orde dos elementos dun *array*, modificando o mesmo.

```
var idades = [40, 17, 40, 85, 24];
idades.reverse(); // [24, 85, 40, 17, 40]
```

- `keys()`: devolve un obxecto `Array Iterator` cos índices do *array*.

```
var idades = [40, 17, 85, 24];
var resultado = idades.keys(); // Array Iterator object
for (var i of resultado) { console.log(i); }
```

- `isArray()`: determina se un obxecto é un *array* (`true`) ou non (`false`).

```
var idades = [40, 17, 85, 40, 24];
var resultado = Array.isArray(idades); // true
```

- **Iteración:** neste grupo están aqueles métodos que aplican a cada elemento unha función, de forma automática e en secuencia**:**
  - `some()`: comproba se algún dos elementos do *array* pasa unha proba (fornecida por unha función pasada como parámetro). Este método executa a función para cada elemento e:
    - Se algún deles devolve `true`, o resultado será `true` e non seguirá comprobando.

- Se todos devolven `false`, o resultado será `false`.

```
var idades = [40, 17, 85, 24];
var resultado = idades.some(comprobarVoto); // true

function comprobarVoto(idade) { return idade>=18; }
```

- `every()`: comproba se todos os elementos do *array* pasan unha proba (fornecida por unha función pasada como parámetro). Este método executa a función para cada elemento e:
  - Se algún deles devolve `false`, o resultado será `false` e non seguirá comprobando.
  - Se todos devolven `true`, o resultado será `true`.

```
var idades = [40, 17, 85, 24];
var resultado = idades.every(podeVotar); // false

function podeVotar(idade) { return idade>=18; }
```

- `find()`: devolve o primeiro valor do *array* que pasa unha proba (fornecida por unha función pasada como parámetro). Se non atopa ningún devolve `undefined`.

```
var idades = [40, 17, 85, 24];
var resultado = idades.find(eMenor); // 17

function eMenor(idade) { return idade<18; }
```

- `findIndex()`: devolve o índice do primeiro valor do *array* que pasa unha proba (fornecida por unha función pasada como parámetro). Se non atopa ningún devolve `-1`.

```
var idades = [40, 17, 85, 24];
var resultado = idades.findIndex(eMenor); // 1

function eMenor(idade) { return idade<18; }
```

- `filter()`: devolve un *array* novo cos elementos do *array* fonte que pasan unha proba (fornecida por unha función pasada como parámetro).

```
var idades = [40, 17, 85, 24];
var resultado = idades.filter(eMenor); // [40, 85, 24]
```

```
function eMenor(idade) { return idade>=18; }
```

- `map()`: crea un novo *array* cos resultados de aplicar a cada elemento unha función (pasada como parámetro).

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.map(dobrar); // [80, 34, 80, 170, 48]

function dobrar(idade) { return idade*2; }
```

- `forEach()`: executa unha función (pasada como parámetro) unha vez por cada elemento do *array*.

```
var idades = [40, 17, 85, 24];
var resultado = 0;
idades.forEach(sumaIdades); // resultado = 166

function sumaIdades(idade) { resultado += idade; }
```

- `reduce()`: reduce os valores do *array* a un só, aplicando unha función (pasada como parámetro) a cada elemento de esquerda a dereita. O valor devolto pola función gárdase nun acumulador.

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.reduce(sumaIdades); // 206

function sumaIdades(total, idade) { return total + idade; }
```

- `reduceRight()`: reduce os valores do *array* a un só, aplicando unha función (pasada como parámetro) a cada elemento de dereita a esquerda. O valor devolto pola función gárdase nun acumulador.

```
var idades = [40, 17, 40, 85, 24];
var resultado = idades.reduce(sumaIdades); // 206

function sumaIdades(total, idade) { return total + idade; }
```

ÍNDICE