DOM (ELEMENT)

INTRODUCCIÓN

Una de las clases más importantes que tiene Mootools es la clase **Element**, la cual nos brinda métodos para manipular el DOM de nuestras páginas de manera muy sencilla.

El Document Object Model (una traducción al español no literal, pero apropiada, podría ser Modelo en Objetos para la representación de Documentos), abreviado DOM, es esencialmente un modelo computacional a través del cual los programas y scripts pueden acceder y modificar dinámicamente el contenido, estructura y estilo de los documentos HTML y XML. Su objetivo es ofrecer un modelo orientado a objetos para el tratamiento y manipulación en tiempo real (o de forma dinámica) a la vez que de manera estática de páginas de Internet.

[CORE: element.js] Creando un Elemento

```
var miElemento = new Element(tipoElemento, {opciones});
```

Donde tipoElemento es el tipo de elemento que vamos a crear, por ejemplo ‘a’ (un enlace).

```
var miEnlace = new Element('a');
```

Las opciones pueden ser propiedades, eventos y estilos.

Propiedades

Las propiedades tales como id, class, href se agregan como propiedades de un objeto, así:

```
var miEnlace = new Element('a', {
      'class': 'mi-clase',
      'id': 'mi-id',
      'href': 'http://sitio.com'
});
```

Estilos
Los estilos se agregan de manera muy similar a las propiedades, pero en una llave styles.

```
var miEnlace = new Element('a', {
      'href': 'http://sitio.com',
      'styles': {
            'border': '1px solid #ff0000',
            'padding': '5px',
      }
});
```

Eventos

Esta vez en la llave events.

```
var miEnlace = new Element('a', {
      'href': 'http://sitio.com',
      'styles':       {
            'border': '1px solid #ff0000',
            'padding': '5px',
      },
      'events':       {
            'mouseenter': function() { alert('mouseEnter'); },
            'mouseleave': function() { alert('mouseLeave'); }
      }
});
```

En todos los ejemplos los elementos son creados, pero para que puedan ser visualizados deben ser insertados en algún sitio.

[CORE: element.js] Utilizar elementos existentes

Si vamos a utilizar elementos que ya existen en nuestra pagina podemos hacer uso de las funciones $ y $$ para buscar y utilizar los elementos que necesitamos.

$(id)

Utilizamos $ cuando buscamos un solo elemento por su id. Retorna el elemento o false en caso de no encontrarlo.
También podemos usar la referencia al elemento

```
<a href="#" id="uno">Mi Enlace</a>
var miEnlace = $('uno');
```

```
//Referencia:
var miEnlace2 = $(miEnlace.childNodes[0]);
```

$$(selector [, otro selector, y mas selectores])

Cuando buscamos varios elementos debemos utilizar $$ quien nos devuelve un array con los elementos que especificamos.

```
//Seleccionamos todos los enlaces de nuestra página
var misEnlaces = $$('a');
```

```
//Seleccionamos todos los enlaces y h1.
var misElementos = $$('a', 'h1');
```

```
//Seleccionamos todos los enlaces y los elementos con id uno, dos y tres.
var misElementos = $$('a', ['uno', 'dos', 'tres']);
```

```
//Seleccionamos todos los enlaces con class enlace
var misEnlaces = $$('a.enlace');
```

```
//Seleccionamos todos los enlaces que están dentro de id elemento.
var misEnlaces = $$('#elemento a');
```

```
//Seleccionamos todos los enlaces, h1 y elementos con class texto.
var misElementos = $$('a', 'h1', '.texto');
```

OJO: Se recomienda usar una vez $$ por cada búsqueda, en lugar de encadenar búsquedas del tipo $$('a', 'href').

[CORE: element.js] CLASE ELEMENTS

Cómo funciona

Se lanza sobre un elemento ($) o un array de elementos ($$)

Ejemplo:

```
/* Todos los párrafos a color rojo */
$$('p').setStyle('color','red');
/* Buff que feo… vuelve a ponerlo como estaba */
(function(){$$('p').setStyle('color','#000')}).delay(1000);
```

OJO:
Cuando ejecuta un método sobre una colección de elementos, esto iterará por todos los elementos con lo que ejecutará ese método sobre cada uno. Vale cuando se tenga que ejecutar un método, pero NO si se debe ejecutar más que uno. El método múltiple hace un loop por cada uno de ellos y esto será muy lento.

```
$$('p').setStyle('color','red');
$$('p').effect('opacity').start(0,1);
//Esto es lo mismo, igual de mal:
$$('p').setStyle('color','red').effect('opacity').start(0,1);
```

Esto lo hace uno a uno (bien):

```
$$('p').each(function(p){
  p.setStyle('color','red').effect('opacity').start(0,1);
});
```

Set

A través de este método se pueden poner eventos, estilos y propiedades. Tiene la misma sintaxis que el segundo argumento de "new Element(tipoElemento, {opciones})". Cualquier característica(propiedad) natal está disponible (src, href, el valor).

```
$(el).set({events: ..., styles: ..., etc.});

//O lo que es lo mismo
$(el).setStyles({...}).addEvents({...}).etc.
$(myImg).set('src', newUrl);
$(myImg).set('alt', 'Pastel de carne');
```

get

El contrario de Element.set es Element.get, que recupera las propiedades de un elemento.

```
$ (myImg) .get ('src'); // http: //....
```

Element.get ('el valor') devolverá el valor de un elemento, pero sólo esto. No devolverá el valor de la opción seleccionada mySelect.get ('el valor') ya que el valor no es una característica(propiedad) de un select.

Set / Get opciones

Por defecto, a set y get se le puede poner como argumento:

* HTML consigue/pone HTML interior
* Text consigue/pone el valor de texto de un nodo
* Tag consigue la etiqueta de un nodo
* Value consigue/pone el valor de un input.

Con esto eliminamos los métodos de la versión 1.11 de setHTML, setText…

getSelected

Devuelve la opción seleccionada en una select. Siempre devuelve un array.

```
$(mySelect).getSelected(); //un array de opciones que son seleccionadas
```

toQueryString

Devuelve un string con los nombres/valores de los elementos de un formulario.

```
$(myForm).toQueryString(); //foo=bar&something=else...
```

Store / retrieve

Element.Storage es totalmente nuevo en MooTools 1,2. Es básicamente un Hash externo que almacena todas las propiedades y eventos personalizados para cada uno de los elementos con los que interactúa.

Es decir, podemos convertir al elemento en una especie de clase con propiedades y métodos sin que se altere el DOM.

Store almacena y retrieve almacena y recupera.

Ejemplos:

```
    var element = $('dos');
    element.store('effectInstance',new Fx.Tween(element, {
        property: 'opacity',
           duration: 500}));

    element.store('customProperty', 'someProperty');

    element.retrieve('effectInstance').start(1,0);
    //La instancia Fx.Tween +  lanza la instancia con start


    escribe(element.retrieve('customProperty'));
    //'someProperty'
```

```
var element = $('myElement');
var data = element.retrieve('galleryData', {});
data.id = 16;
data.source = '/images/16.jpg';
data.title = 'Some Title';
//later
$('myElement').retrieve('galleryData');
//{ id: 16, source: '/images/16.jpg', title: 'Some Title' }
$('myElement').retrieve('galleryData').id;
//16
```

Más información en: http://bezerik.es/me/category/mootools/

injectBefore / injectAfter

(injectBefore) Inserta un elemento antes del elemento que pasamos por parámetro. Dicho parámetro puede ser el id de un elemento o directamente su referencia.

```
<div id="otroElemento"></div>

var elemento = new Element('div', {'id':'MiElemento'});
elemento.injectBefore('otroElemento');

<div id="MiElemento"></div>
<div id="otroElemento"></div>
```

En este caso creamos un elemento (div) y lo insertamos antes que nuestro otro div.

También podríamos haber tomado cualquier otro elemento de nuestra pagina y hacer lo mismo, sin tener que crearlo como en el ejemplo anterior.

```
$('MiElemento').injectBefore('otroElementoMas');
```

En este caso, MiElemento desaparecerá de donde estaba y se colocará por delante de otroElementoMas.

injectInside

Como injectBefore, pero inserta dentro del elemento pasado por parámetro, al final.

```
<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>

var elemento = new Element('div', {'id':'MiElemento'});
elemento.injectInside('otroElemento');

<div id="otroElemento">
       <div id="otroElementoMas"></div>
       <div id="MiElemento"></div>
</div>
```

injectTop

Como injectInside, pero inserta dentro del elemento pasado por parámetro, al principio.

Inject

Inserta al elemento referenciado el pasado como argumento.

Pasando de arriba abajo (before/after).

```
html:
<div id="myElement"></div>
<div id="mySecondElement"></div>

<script>$('mySecondElement').inject('myElement', 'before');</script>

resultado html:
<div id="mySecondElement"></div>
<div id="myElement"></div>
```

Como hijo.

```
html:
<div id="myElement"></div>
<div id="mySecondElement"></div>

<script>$('mySecondElement').inject('myElement');</script>

resultado html:
<div id="myElement">
       <div id="mySecondElement"></div>
</div>
```

Moviendo nodos (top / bottom):

```
html:
<div id="myElement">
       <div id="childOne"></div>
       <div id="childTwo"></div>
       <div id="childThree"></div>
</div>

<script>$('childTwo').inject('myElement', 'top');</script>

resultado html:
<div id="myElement">
       <div id="childTwo"></div>
       <div id="childOne"></div>
       <div id="childThree"></div>
</div>
```

adopt

Adopt funciona similar a injectInside, con la diferencia que nos permite insertar uno o varios elementos dentro de otro. Se puede indicar que elementos insertar con un id, un array de ids, un array de elementos o lo que es lo mismo usando $$ y selectores CSS.

HTML ORIGINAL

```
<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>
<div id="destino"></div>
```

DIRECTO

```
var elemento = new Element('div', {'id':'MiElemento'});
$('otroElemento').adopt(elemento);

//indicamos directamente el elemento a insertar

<div id="otroElemento">
       <div id="otroElementoMas"></div>
       <div id="MiElemento"></div>
</div>
<div id="destino"></div>
```

CON ID'S

```
$('destino').adopt(['otroElementoMas', 'MiElemento']);

<div id="destino">
       <div id="otroElementoMas"></div>
       <div id="MiElemento"></div>
</div>
```

Un ejemplo usando selectores:

HTML ORIGINAL

```
<ul id="origen">
   <li>Uno</li>
   <li>Dos</li>
</ul>

<ul id="destino">
</ul>
```

```
$('destino').adopt($$('#origen li'));
//Todos los li dentro de #origen

<ul id="origen">
</ul>

<ul id="destino">
   <li>Uno</li>
   <li>Dos</li>
</ul>
```

dispose (ant conocido como remove)

Nos permite borrar elementos.

```
<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>

var elm_borrado = $('otroElementoMas').dispose();


<div id="otroElemento">
</div>
```

Permanece en memoria, con lo que más tarde se podrá volver a referenciar.

```
Elm_borrado.inject('otroElemento');
```

destroy

Borra, incluso de memoria, el nodo referenciado.

clone

Nos permite copiar elementos.
Podemos indicar mediante un parámetro si deseamos que se copien los elementos *hijos* del elemento que estamos copiando(true) o no(false). Predeterminado, true.

HTML ORIGINAL

```
<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>
<div id="destino">
</div>
```

TRUE

```
var elemento = $('otroElemento').clone();
$('destino').adopt(elemento);


<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>
<div id="destino">
       <div id="otroElemento">
              <div id="otroElementoMas"></div>
       </div>
</div>
```

FALSE:

```
var elemento = $('otroElemento').clone(false);
$('destino').adopt(elemento);


<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>
<div id="destino">
       <div id="otroElemento"></div>
</div>
```

Ojo: No se deben repetir los ids.

Wraps

Mueve el elemento de su localización actual a padre del elemento referenciado.

```
<div id="child"></div>
<hr/>
<div id="newParent"></div>

$('newParent').wraps('child');

Resultado:

<div id="newParent">
      <div id="child"></div>
</div>
<hr/>
```

replaces (ant conocido como replaceWith)

Reemplazamos un elemento con otro.

```
<div id="otroElemento">
       <div id="otroElementoMas"></div>
</div>

var elemento = new Element('div', {'id':'MiElemento'});
$('otroElementoMas').replaces(elemento);


<div id="otroElemento">
       <div id="MiElemento"></div>
</div>
```

appendText(texto)

Nos permite agregar **solo texto** a un elemento. Si intentamos insertar HTML con *appendText* no sera interpretado, es pasado a entidades HTML para ser mostrado como texto.

```
<div id="elemento">Elemento</div>
$('elemento').appendText(' con mucho texto');

Resultado:
<div id="elemento">Elemento con mucho texto</div>

Y si intentamos insertar HTML:
$('elemento').appendText(' con algunos <tags></tags>');

Resultado:
<div id="elemento">Elemento con algunos &lt;tags&gt;&lt;/tags&gt;</div>
```

hasClass(nombreClase)

Devuelve true si nuestro elemento contiene la clase indicada o false en caso contrario.

```
<div id="elemento" class="casa">Elemento</div>
var resultado = $('elemento').hasClass('departamento');
//resultado es false

var resultado = $('elemento').hasClass('casa');
//resultado es true
```

addClass(nombreClase)

Agrega una clase a un elemento.

```
<div id="elemento">Elemento</div>
$('elemento').addClass('casa');

Resultado:
<div id="elemento" class="casa">Elemento</div>

$('elemento').addClass('departamento');

Resultado:
<div id="elemento" class="casa departamento">Elemento</div>
```

removeClass(nombreClase)

Quita una clase de un elemento.

```
<div id="elemento" class="casa departamento">Elemento</div>
$('elemento').removeClass('casa');

Resultado:
<div id="elemento" class="departamento">Elemento</div>
$('elemento').removeClass('departamento');

Resultado:
<div id="elemento" class="">Elemento</div>
```

toggleClass(nombreClase)

Si la clase indicada está presente en nuestro elemento, la quita, de lo contrario, la agrega.

```
<div id="elemento">Elemento</div>
$('elemento').toggleClass('casa');

Resultado:
<div id="elemento" class="casa">Elemento</div>

$('elemento').toggleClass('casa');

Resultado:
<div id="elemento" class="">Elemento</div>
```

match

Devuelve true si lo que pasamos está incluido en el elemento. En caso de poner espacios se leen como “o”.

```
$(el).match('div'); //returns true if el is a div

$(el).match('div#someId p.someClass'); //true of el matches
```

getPrevious / getNext / getFirst / getLast / getParent / getChildren

Dado un elemento, *getPrevious* devuelve el elemento inmediatamente anterior.

```
<div id="contenedor">
      <div id="elemento1">Elemento1</div>
      <div id="elemento2">Elemento2</div>
      <div id="elemento3">Elemento3</div>
</div>

var miElemento = $('elemento2').getPrevious();
//miElemento es ahora una referencia al div con id="elemento1"

var miElemento = $('elemento2').getNext();
//miElemento es ahora una referencia al div con id="elemento3"

var miElemento = $('contenedor').getFirst();
//miElemento es ahora una referencia al div con id="elemento1"

var miElemento = $('contenedor').getLast();
//miElemento es ahora una referencia al div con id="elemento2"

var miElemento = $('elemento1').getParent();
//miElemento es ahora una referencia al div con id="contenedor"
```

Dado un elemento, *getChildren* devuelve sus elementos hijos (no los “nietos” ni otros anidados).

```
<div id="contenedor">
      <div id="elemento1">Elemento1</div>
      <div id="elemento2">Elemento2
            <div id="elemento3">Elemento3</div>
      </div>
</div>
var miElemento = $('contenedor').getChildren();

//miElemento es ahora un vector de dos posiciones con las referencias a los
elementos        id="elemento1" y id="elemento2"
```

hasChild(otroElemento)

Dado un elemento, *hasChild* devuelve *true* si dicho elemento es padre del elemento pasado por parámetro, de lo contrario, devuelve *false*.

```
<div id="contenedor">
      <div id="elemento1">Elemento1</div>
      <div id="elemento2">Elemento2</div>
</div>

var esHijo = $('contenedor').hasChild($('elemento1'));
//true
```

getProperty(propiedad)

Devuelve la propiedad especificada de un elemento.

```
<div id="elemento" class="miClase"></div>

var clase = $('elemento').getProperty('class');
//miClase
```

removeProperty(propiedad)

Borra la propiedad especificada de un elemento.

```
<div id="elemento" class="miClase"></div>
$('elemento').removeProperty('class');
```

```
<div id="elemento"></div>
```

getProperties(propiedad, otraPropiedad, masPropiedades)

Devuelve todas las propiedades especificadas de un elemento.

```
<div id="elemento" class="miClase"></div>

var propiedades = $('elemento').getProperties('id', 'class');
alert(propiedades.id); //elemento
alert(propiedades.class); //miClase
```

setProperties(objeto)

Cambia las propiedades especificadas por medio de un objeto de tipo *llave => valor*.

```
<div id="elemento" class="miClase"></div>
$('elemento').setProperties({
	'class': 'otraClase',
	'id': 'otroId'
});
```

```
<div id="otroId" class="otraClase"></div>
```

empty()

Borra todo el contenido de un elemento.

```
<div id="contenedor">
	<div id="elemento1">Elemento1</div>
		<div id="elemento3">Elemento3</div>
	</div>
</div>
$('contenedor').empty();
```

```
<div id="contenedor"></div>
```

ELEMENT ESTILOS

setStyle(propiedad, valor)

Nos permite cambiar una propiedad css.

```
<div id="elemento">Elemento</div>
$('elemento').setStyle('background-color', '#cccccc');

Resultado:
<div id="elemento" style="background-color:#cccccc">Elemento</div>
```

setStyles(objeto o string)

Nos permite cambiar varias propiedades css en un solo paso. Se puede indicar un objeto con los estilos y sus correspondientes valores o bien todos los estilos en un *string*. En este ultimo caso, se sobrescriben los estilos que posee el elemento.

```
<div id="elemento" style="width:300px">Elemento</div>

$('elemento').setStyles({
     'background-color': '#ccc',
     'color': '#333'
});

Resultado:
<div id="elemento"
       style="width: 300px;
       background-color: rgb(204, 204, 204);
       color: rgb(51, 51, 51);">Elemento</div>
```

Si hubieramos utilizado un string:

```
$('elemento').setStyles('background-color:#ccc;color:#333');

Resultado:
<div id="elemento" style="background-color: rgb(204, 204, 204); color: rgb(51,
51, 51);">Elemento</div>
<!-- La propiedad width fue borrada -->
```

getStyle(propiedad)

Devuelve el valor de la propiedad CSS indicada.

```
<div id="elemento" style="width:300px">Elemento</div>
var width = $('elemento').getStye('width');
//width vale "300px"
```

getStyles(propiedad, otra propiedad, mas propiedades)

Devuelve todas las propiedades CSS indicadas en forma de objeto.

```
<div id="elemento" style="width:300px;height:100px">Elemento</div>
var estilos = $('elemento').getStyes('width', 'height');
//estilos es {width: "300px", height: "100px"}
//estilos.width es 300px
//estilos.height es 100px
```

EVENTOS

Element.addEvent, addEvents

Atención: Los nombres de los eventos pierden el “on”. Por ejemplo, onClick será "click".

```
$('addEventExample').addEvent('click', function() {  alert('click!');  });
```

¿Porqué no usar **$('addEventExample').onclick = alert('click');** ?

Usando mootools podemos tener varias funciones colgando de un evento, aunque las creemos en distintos momentos. Si usamos do.onclick = myFunction() entonces solo puedes tener una función asociada a ese click.

```
//Aquí se ejecutarán ambos eventos.
    $("orden").addEvent("click",function(){alert(';-)')});
    $("orden").addEvent("click",function(){alert(';-(')});

//Aquí sólo se ejecutará el último.
    $("orden").onclick = function() {alert(';-)')};
    $("orden").onclick = function() {alert(';-(')};
```

Element.addEvents es el mismo simplemente que el argumento es un objeto:

```
$('addEventExample').addEvents({
      mouseover: function() {this.setStyle('color','#f00')},
      mouseout: function() {this.setStyle('color','#000')}
});
```

Element.removeEvent, removeEvents

Puedes usar Element.removeEvent para eliminar un evento de un elemento. Con removeEvents podrás eliminar todos los eventos de un elemento. Para que removeEvent funcione debes pasar la función que le pasaste al crearla. Si ese es tu intención no uses funciones anónimas.

```
$(el).addEvent('click', function(){alert(':->')});

//Esto no hace nada
$(el).removeEvent('click', function(){alert(':->')});

//Mejor con una instancia
var sayClicked = function(){alert(':->')};
$(el).addEvent('click', sayClicked);
$(el).removeEvent('click', sayClicked);
```

Element.fireEvent

Dispara el evento espeficicado en un elemento.

```
$('fireEventExample').addEvent('click', function(){alert(':->')});
$('fireEventExample').fireEvent('click'); //alerts ':->'
```

Element.cloneEvents

Copia los eventos asociados a un elemento.

```
$(new).cloneEvents(old); //copies the events from old to new
$(new).cloneEvents(old, 'click');//copies only the click events from old to
```

```
new
```

domready / onload

Quien conozca las bases de Javascript sabrá que existe un evento "onload", que sirve para ejecutar acciones cuando se termina la carga de la página. Pero este evento se ejecuta cuando TODOS los elementos de la página han terminado de cargarse, es decir, la propia página y todo lo que tenga, como imágenes, banners y otro tipo de recursos externos. Así que, si nuestra página tiene, por ejemplo, muchas imágenes muy pesadas, puede que onload tarde demasiado en ejecutarse y hasta entonces no se mostrarían los elementos que se desean insertar dinámicamente.

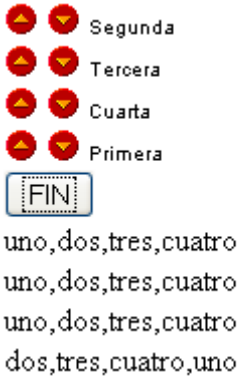
Pero mucho antes que finalice esa carga de elementos externos, la página podría haber estado lista para ejecutar acciones Javascript que alteren el DOM. En concreto, cuando una página termina de cargarse y procesarse, aunque se continúen descargando imágenes o elementos externos, el DOM estará listo para realiza acciones.

```
window.addEvent('domready', function(){...});
```

```
window.addEvent('load', function(){...});
```

EJERCICIOS

Objetivo.
A partir de un listado de “divs” cualquiera colocar botones de subir / bajar para cambiar la ordenación. Se pide control de errores y obtención de resultado final.



1.- Genera este HTML:

```
<div id="orden">
    <div id="uno" class='item'>Primera </div>
    <div id="dos"  class='item'>Segunda </div>
    <div id="tres"  class='item'>Tercera </div>
    <div id="cuatro"  class='item'>Cuarta </div>
</div>
```

2.- En Windows.onload comprueba que todos los divs hijos del div “orden” tienen un identificativo, En caso de que no sea así alert de error y acabas.

(Muy útil en este caso getChildren y $A).

3.- Por cada hijo del div “orden” consigue por DOM obtener esto:

```
<div id="orden">
    <div id="uno" class="item">
        <div class="img UP"/>
        <div class="img DW"/>
        <div class="texto">Primera </div>
    </div>
    <div id="dos" class="item">
        <div class="img UP"/>
```

Es decir, tenemos por cada hijo que (elementos.each(function elemento, indice){}) :

3.A.- Capturar el texto de cada hijo y encapsularlo en un div con un class “texto” get(“text”). Eliminar el texto original set(“text”,””)

3.B.- Añadir delante dos imágenes o div con imagen de fondo: una para subir y otra para bajar new Element(“div”,{opciones})

4.- Añade un evento a las opciones anteriores (aprovecha el new Element de 3B) de manera que cuando pulse sobre el botón arriba el nodo se desplace arriba o cuando pulse abajo se desplace abajo. Ojo a los errores de los extremos: si subo más arriba de la lista disponible me dará error al intentar intercambiar un valor nulo.

Para este punto vendrán muy bien
getFirst / getLast / getPrevious / getNext
getProperty(“id”)
inject

Ejemplo: $("orden").getFirst().getProperty("id")

5.- Controla el primero y el último, de manera que cuando sobrepasemos el primero este se vaya al último y si sobrepasamos el último se irá al primero.

Muy útil inject:
Ejemplo: el.inject($("orden"),"bottom");

6.- Para acabar inserta un botón el código de manera que al pulsarlo nos diga el orden de los id’s que tenemos en este momento.

Aquí vendrá muy bien $A o, si se prefiere, each

Ejemplo: $A($("orden").getChildren().getProperty("id"));