

Javascript en Node.js

Desarrollo de Aplicaciones Web Avanzado



Capacidad Terminal

- Diseñan y crea aplicaciones interactivas en el lado del cliente, utilizando JavaScript

Competencia específica de la sesión

- Al finalizar la sesión, los estudiantes serán capaces de identificar las funcionalidades de Java Script en Node.js

En los inicios de Javascript

```
var form = document.forms[0];  
if (form.txtName.length == 0) {  
    alert("You forgot your name!");  
    return false;  
}
```

Hoy en día

```
function MobilePromo(e){"object"==typeof e&&(e.cooloffDays=e.cooloffDays||7,this.config=e,this.init
)}}function x(e,t){if(t in e)return t;for(var n=t[0].toUpperCase()+t.slice(1),r=t,i=Ge.length;i--;)
e="string"==typeof e?pe[e]||a(e):Q.extend({},e);var t,n,r,i,o,s,u=[],m=!e.once&&[],d=function(a){fo
null}}),Q.each(["tabIndex","readOnly","maxLength","cellSpacing","cellPadding","rowSpan","colSpan","
var n=v.methodMap[e];(console[n]||console.log).call(console,t)});t.logger=v;var b=v.log;t.log=b},f
},initTOSBanner:function(){var e=t(".j-tos-update-banner");if(e.length){var n=t(mobile_util.isMobil
"percent"===i["[[style]]"]&&(t*=100),n=se.call(i,"[[minimumSignificantDigits]]")&&se.call(i,"[[maxi
},{day:"numeric",month:"numeric",year:"numeric",hour:"2-digit",minute:"2-digit",pattern:"{day}/{mon
}})),IntlPolyfill.__addLocaleData({locale:"ja",date:{ca:["gregory"],hourNo0:!1,hour12:!1,formats:[{
negativePattern:"-{number}%"},symbols:{latn:{decimal:".",group:".",nan:"NaN",percent:"%",infinity:
pattern:"{weekday}"},{weekday:"short",pattern:"{weekday}"},{day:"numeric",pattern:"{day}日"}],calen
month:"long"},t["my.long"]=t.my,t["my.medium"]={year:"numeric",month:"short"},t["my.short"]=t["my.m
t.time={year:"numeric",month:"narrow",day:"numeric",hour:"numeric",minute:"numeric",hour12:!0},t["t
date:/((?:19|20)[0-9]{2})-(?:((?:0[1-9]|1[0-2])-(?:0[1-9]|1[0-9]|2[0-9])|((?:?!02)(?:0[1-9]|1[0-2])-(?
),center:function(){var n=e(t);return this.settings.$next_tip.css({top:(n.height()-this.settings.$n
m.on("click","."+a.prev_class,y.prev),a.next_on_click&&m.on("click","."+a.slides_container_class+"}
```



TECSUP
Pasión por la Tecnología

Código eficiente

- Es muy importante escribir código de manera eficiente sobre todo en el ámbito del desarrollo web, debido a que los segundos que ganamos de ejecución se verán multiplicados por la cantidad de usuarios que tengamos enganchados.

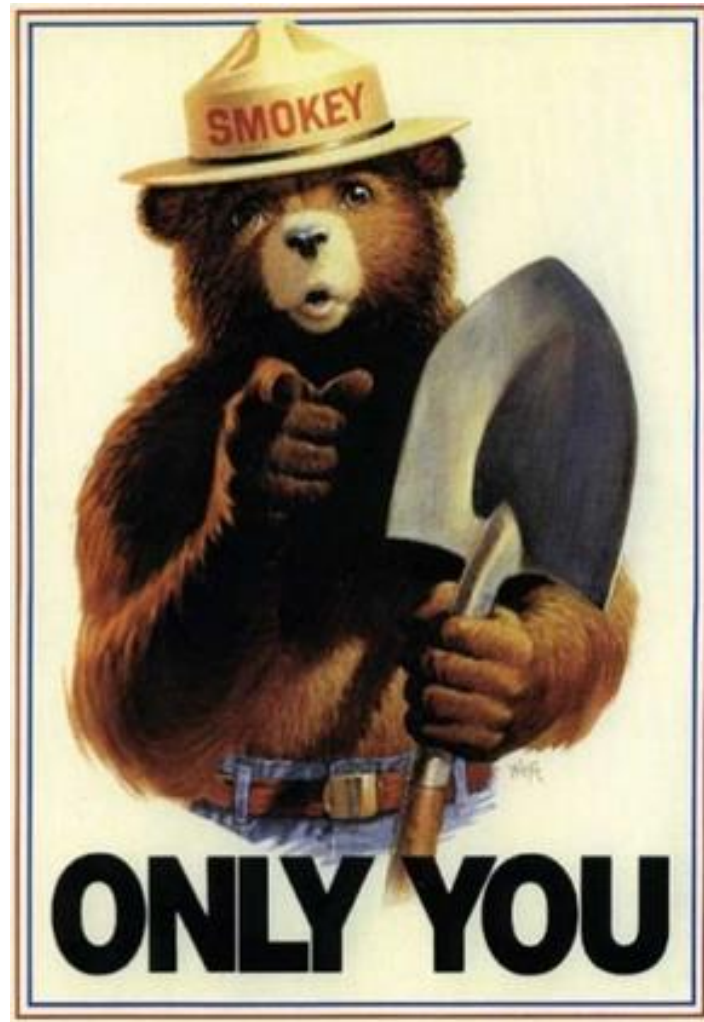
En los inicios de Javascript

- Solamente se utilizaba para validar formularios o efectos de imágenes (hover)
- Modelo Click-and-go (hacer click e ir)
- Cada página contenía poco código
- Conexiones lentas de internet (la gente estaba acostumbrada a esperar)

Hoy en día

- Ajax y la Web 2.0
- Más código Javascript que nunca
- Conexiones rápidas de Internet (La gente espera aún más velocidad)
- Aplicaciones que permanecen abierta mucho más tiempo (Facebook, Gmail)
- Descargas y ejecutas más código mientras más interactuas

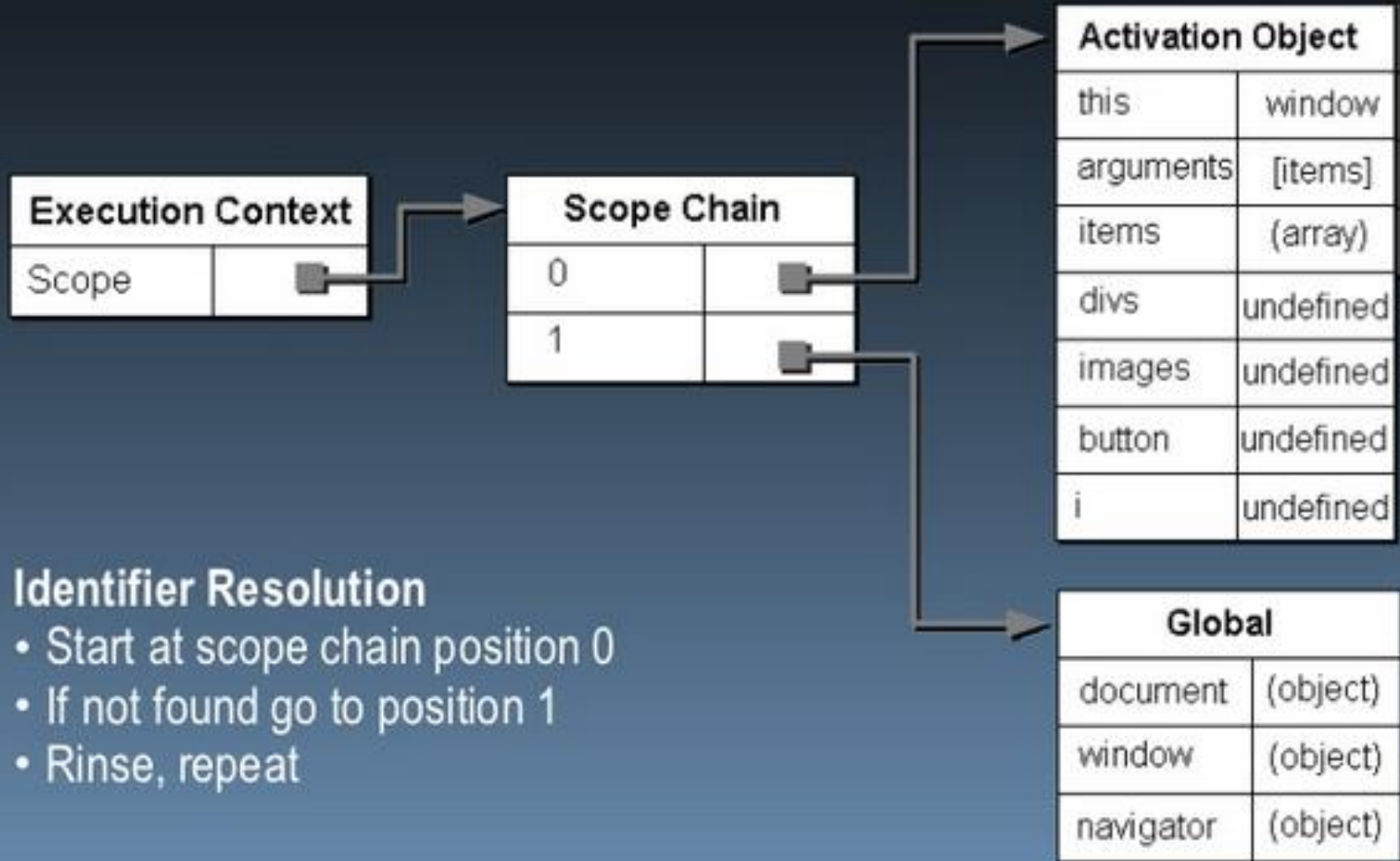
¿Quién ayudará a tu código?



Lo que sucede cuando una función se ejecuta

1. Un contexto de ejecución es creado
2. La cadena de eventos del ámbito de contexto es inicializada con los miembros de la colección de funciones
3. Un objeto de activación es creado, conteniendo todas las variables locales
4. El objeto de activación es empujado al inicio de la cadena de eventos del ámbito de contexto

Contexto de ejecución



Optimizar código

- Siempre debemos preguntarnos si una variable o función será llamada o ejecutada a veces inútilmente durante nuestro programa. Un ejemplo clásico es en los bucles.

```
var names = ['George', 'Ringo', 'Paul', 'John'];  
for(var i = 0; i < names.length; i++) {  
    doSomethingWith(names[i]);  
}
```

Optimizar código

- Una forma sencilla de optimizar el código anteriormente visto es declarar solamente una vez la variable que indica la longitud del array, de tal forma que se consultará una única vez dicha propiedad.

```
var names = ['George', 'Ringo', 'Paul', 'John'];  
var all = names.length;  
for(var i = 0; i < all; i++) {  
    doSomethingWith(names[i]);  
}
```

Optimizar código

- E incluso podemos optimizar aún más reduciendo las líneas de código innecesario o sobre todo, como en el ejemplo presentado, sin la necesidad de declarar una nueva variable, solamente una ligada al ámbito del bucle.

```
var names = ['George', 'Ringo', 'Paul', 'John'];  
for(var i = 0, j = names.length; i < j; i++) {  
    doSomethingWith(names[i]);  
}
```



Si el usuario puede equivocarse.. Se equivocará

- Y esta es una regla de oro para todo programador. Códigos como el presentado fallarán durante su uso debido a la falta de validación de las entradas a su función.

```
function buildMemberList(members) {  
    var all = members.length;  
    var ul = document.createElement('ul');  
    for(var i = 0; i < all; i++) {  
        var li = document.createElement('li');  
        li.appendChild(document.createTextNode(members[i].name));  
        ul.appendChild(li);  
    }  
    return ul;  
}
```


Código con validación de entradas recibidas

```
function buildMemberList(members) {  
    if (typeof members === 'object' &&  
        typeof members.slice === 'function') {  
        var all = members.length;  
        var ul = document.createElement('ul');  
        for(var i = 0; i < all; i++) {  
            var li = document.createElement('li');  
            li.appendChild(document.createTextNode(members[i].name));  
            ul.appendChild(li);  
        }  
        return ul;  
    }  
}
```

- Una de las formas más simples de testear código antes y después de haberlo optimizado, es utilizando la siguiente porción de código.

```
var test = ( new Date() ).getTime();  
// Código a testear  
console.log( ( new Date() ).getTime() - test );
```

Módulos en Javascript

- Llamamos módulo JavaScript a un código que de alguna manera es “auto contenido” y que expone una interfaz pública para ser usada. Esto no es realmente nuevo, el patrón de módulo ya hace bastantes años que se utiliza y no requiere más que algunos conocimientos de JavaScript para aplicarlo.
- El problema con los módulos en JavaScript no ha sido nunca el crearlos si no el de cargarlos. De hecho, ¿no se trata solo de poner un tag `<script />`? Pues la realidad es que no, porque cargar un módulo implica que antes deben estar cargadas sus dependencias y por lo tanto debemos tener un mecanismo para definir esas dependencias y otro mecanismo para cargarlas al tiempo que cargamos el módulo deseado.

CommonJS

- CommonJS es un sistema de módulos **síncrono**: es decir la carga de módulos es un proceso síncrono que empieza por un módulo inicial. Al cargarse este módulo se cargarán todas sus dependencias (y las dependencias de las dependencias, y las dependencias de las dependencias... y así hasta cualquier nivel de profundidad). Una vez finalicen todas esas cargas, el módulo inicial está cargado y empieza a ejecutarse.

Exportación de un módulo

```
var Complex = function (r, i) {  
  this.r = r instanceof Complex ? r.r : r;  
  this.i = r instanceof Complex ? r.i : (i || 0);  
}  
module.exports = Complex;
```

Requerimiento de un módulo

```
var Complex = require('./complex');
addComplex = function (ca, cb) {
    return new Complex(ca.r + cb.r, ca.i + cb.i);
}
var math = {
    add: function (a, b) {
        if (a instanceof Complex || b instanceof Complex) {
            return addComplex(new Complex(a), new Complex(b));
        }
        return a + b;
    }
}
module.exports = math;
```

- Nótese en el último ejemplo, si quisiéramos importar **math.js**, este archivo, al ya tener la declaración require, cargará su dependencia que es **complex.js**.
- Nodejs soporta módulos CommonJS de forma nativa, pero... ¿qué pasa con el navegador? Pues que necesitamos soporte de alguna herramienta externa. Una de las más conocidas es [browserify](#) que se instala como un paquete de node.



- AMD es otra especificación de módulos JavaScript, cuya principal diferencia con CommonJS es que es asíncrona (AMD significa Asynchronous Module Definition). La implementación más conocida para navegadores de AMD es requirejs. Al ser asíncrona permite escenarios con carga de módulos bajo demanda (es decir cargar un módulo sólo si se va a usar), lo que puede ser interesante en según que aplicaciones.


```
define([], function () {  
    console.log('complex loaded...');  
    var Complex = function (r, i) {  
        this.r = r instanceof Complex ? r.r : r;  
        this.i = r instanceof Complex ? r.i : (i || 0);  
    }  
  
    return Complex;  
});
```

```
define(['complex_amd'], function (Complex) {  
    addComplex = function (ca, cb) {  
        return new Complex(ca.r + cb.r, ca.i + cb.i);  
    }  
    var math = {  
        add: function (a, b) {  
            if (a instanceof Complex || b instanceof Complex) {  
                return addComplex(new Complex(a), new Complex(b));  
            }  
            return a + b;  
        }  
    }  
    return math;  
});
```



TECSUP
Pasión por la Tecnología






- Observa ahora como el módulo depende del módulo `complex_amd`. Eso significa que al cargarse este módulo, el módulo `complex_amd` (archivo `complex_amd.js`) debe estar cargado. Si no lo está `requirejs` lo cargará asincrónicamente, y cuando esta carga haya finalizado invocará la función que define el módulo. Observa ahora que la función tiene un parámetro. Este parámetro se corresponde con lo que **exporta (devuelve) el módulo `complex_amd` del cual dependíamos**. Básicamente, por cada elemento (dependencia) del array tendremos un parámetro en la función.

```
define(['complex_amd', 'math_amd'], function (Complex, math) {  
    console.log(math.add(40, 2));  
    var c1 = new Complex(40, 3);  
    console.log(math.add(c1, 2));  
});
```



TECSUP
Pasión por la Tecnología

- En el navegador podremos apreciar la carga por separado y asíncrona de todos los archivos requeridos por la aplicación.

Name	Status	Type	Initiator
 index.html	200	document	Other
 require.js	200	script	<u>index.html:4</u>
 main_amd.js	304	script	<u>require.js:1926</u>
 complex_amd.js	304	script	<u>require.js:1926</u>
 math_amd.js	304	script	<u>require.js:1926</u>

¿Cuál utilizar?

- La carga asíncrona y on-demand es mucho más natural en la web que la carga síncrona que tiene CommonJS. Lo que ocurre es que actualmente solemos siempre crear un bundle de todos nuestros JavaScript, porque sabemos que es más rápido descargarse un solo fichero de 100Ks que 10 ficheros de 10Ks cada uno.
- Una de las normas básicas de optimizar una página web consiste en minimizar la descarga de ficheros. Los bundles de JavaScript, de CSS, los sprite-sheets y el uso de data-uris van todos por ese camino: cargar un fichero más grande antes que varios de pequeños. Si seguimos esa tónica perdemos la característica de carga on-demand y asíncrona de AMD (porque *antes* de ejecutar la aplicación hemos tenido que generar ese bundle).

¿Cuál utilizar?

- Así parece que, actualmente, no haya una diferencia sustancial entre usar CommonJS y AMD si al final terminamos en un bundle. La cuestión puede reducirse a gustos personales o cantidad de módulos existentes en cada formato (a pesar de que es posible, con poco trabajo, usar módulos CommonJS bajo AMD) pero HTTP2 puede cambiar eso. HTTP2 convierte los bundles en no necesarios, ya que mejora el soporte para varias conexiones.
- Bajo ese nuevo prisma, AMD parece ser una mejor opción que CommonJS.

Bibliografía

- Optimize your JavaScript, CSS and HTML code <https://wet-boew.github.io/v4.0-ci/docs/opt-en.html>
- Módulos en JavaScript
<https://geeks.ms/etomas/2015/09/07/mdulos-en-javascript-amd-commonjs/>
- Writing efficient Javascript
https://www.slideshare.net/nzakas/writing-efficient-javascript/96-Browser_Limit_Causes_Too_much

GRACIAS POR SU ATENCIÓN