Universitat Oberta de Catalunya

Aula

M4.253 - Program. JavaScript programadores aula 1

Módulo 1.- Introducción a JavaScript

Inicio:	Fin:
02/03/20	**29/03/20**

Planteamiento

En la práctica de **Tooling** hemos desarrollado **JavaScript** sin habernos aventurado aun en las particularidades del lenguaje. Sin prácticamente darnos cuenta hemos usado para el desarrollo TDD la funcionalidad que nos va a servir de hilo argumental para esta unidad: **los módulos** .

Un **módulo** es en esencia un pieza de código reusable que exporta objetos específicos haciéndolos accesibles para ser importados y usados por otros módulos.

Esta funcionalidad que está intrínsecamente integrada en paradigmas como la programación orientada a objetos no existía en la especificación original de JavaScript puesto que su propósito no incluía un grado de complejidad suficiente para requerirlos.

El por qué, el cuándo y el cómo se añadió la funcionalidad de modulizar partes de código JavaScript es lo que va a articular el texto de esta primera unidad.

Introducción

JavaScript comparte muchos de sus rasgos y funcionalidades con otros lenguajes con los que habéis podido tener contacto previamente pero con características que lo hacen singular.

Lo que distingue a JavaScript es que es (casi) el único lenguaje de scripting de navegador. Su curva de aprendizaje es baja gracias a su flexibilidad y permisividad. Perfiles junior pueden ser completamente productivos a nivel laboral gracias a esto. Sin embargo es precisamente este exceso de flexibilidad la que ha condenado a JavaScript a ser considerado históricamente un lenguaje de segunda comparado con Java, PHP o .Net.

Nota: recordad que con vuestro usuario de la UOC tenéis acceso a la biblioteca técnica y a los cursos de formación de la plataforma Safari de O'Reilly. Os dejaré indicados los enlaces a dicha plataforma con el sufijo Safari en el nombre de los enlaces.

El ya clásico y fenomenal libro de Douglas Crockford , JavaScript The Good Parts [Safari] ha sido siempre una referencia desde su primera edición, cuando el lenguaje era mucho más inmaduro y era menos usado que actualmente.

Hasta ese momento nadie en la industria se tomaba en serio **JavaScript** , en una época en la que el contenido de las páginas web venían renderizados desde el servidor y la interacción en la parte de usuario se limitaba únicamente a los formularios.

El libro tiene un planteamiento así como un contenido brillante y permitió que la gente apreciara las partes realmente potentes del lenguaje, pero al ganar tracción provocó que se estigmatizara JavaScript como un lenguaje mal planificado, diseñado e implementado puesto que igual que tenía good parts seguía teniendo bad parts .

Vamos a intentar hacer un recorrido histórico para entender de dónde venimos, dónde estamos, a dónde vamos y comprender cada una de las transformaciones del lenguaje.

Historia y contextualización

Difícilmente, en 1995, Brendan Eich podía anticipar para lo que se estaría usando, 25 años más tarde, el lenguaje en el que estaba trabajando.

JavaScript no es perfecto pero todo lo que se especificó tenía un sentido aunque para algunos que vengan de otros lenguajes no les parezca evidente. Es aceptable valorar a nivel personal y señalar aspectos del lenguaje como bad parts pero sería injusto obviar partes del mismo sin realmente haber sometido a un sesudo juicio su posible potencial.

- Motivación inicial de Netscape para crear JavaScript
- Infografía sobre la cronología de JavaScript
- Explicación extendida sobre la cronología
- dotJS 2017 - Brendan Eich - A Brief History of JavaScript [Youtube]

Tened presente y valorad que cuando **Netscape** contrató a **Brendan Eich** en abril de 1995, se le dijo que tenía 10 días para crear y producir un prototipo funcional de un lenguaje de programación que se ejecutara en el navegador de Netscape.

Os recomiendo encarecidamente que veáis el vídeo de Brendan Eich que os he dejado en los enlaces más arriba puesto que nadie mejor que él puede entender y valorar el propósito inicial de su proyecto, su evolución, su situación actual y el futuro que le depara.

Las características fundacionales de JavaScript

Para entender la visión inicial de JavaScript debemos conocer las tres características que lo hacen tan especial:

- Lenguaje funcional
- Herencia
- Cadena de prototipo

Revisad esta explicación en MDN y ésta de Tyler McGinnis . Con 10 días para diseñar un lenguaje no pudo pensar todo desde cero y recibió influencias de muchos otros lenguajes que presentaban sintaxis y paradigmas que tuvo a bien considerar para la primera implementación de JavaScript.

Tanto **herencia** como la **cadena de prototipo** son dos aspectos básicos del funcionamiento y concepción de JavaScript. Ambas siguen siendo partes intrínsecas al lenguaje aunque en mi experiencia no se usan activamente para planificar y desarrollar proyectos.

Sin que ello signifique que no se puedan hacer propuestas válidas bajo estos paradigmas, **JavaScript** incluye actualmente herramientas y funcionalidades más convencionales e igualmente potentes para realizar programación de aplicaciones modernas.

El último ejemplo más o menos exitoso en el que recuerdo la aplicación de la cadena de prototipo de forma exhaustiva es Backbone . El primer framework de amplia adopción que abrió el camino para sus hermanos **Angular** , **Ember** y **React** . La propuesta era mucho más rudimentaria que las actuales APIs de estos últimos pero un paso necesario al fin y al cabo.

Evolución

JavaScript evoluciona por según un proceso de realimentación en el que las exigencias de las nuevas aplicaciones obligan a desarrollar de forma orgánica y espontánea nuevas funcionalidades como extensiones que los propios ingenieros implementan.

Cada cierto tiempo el grupo de mantenimiento y desarrollo **ECMAScript** (TS39) lanzan una nueva especificación con las funcionalidades más comúnmente usadas o requeridas como estándares oficiales.

Podéis comprobar la evolución de las diferentes versiones de ECMAScript en este enlace a la Wikipedia , siendo la última versión, **ES6** , la que usamos actualmente.

En este punto hay que puntualizar cómo funciona la adopción de estos estándares por parte de los diferentes navegadores: cuando una nueva versión de la especificación se hace pública ningún navegador lo soporta de forma oficial. Las nuevas versiones de los navegadores van incluyendo paulatinamente las nuevas funcionalidades en sus motores e intérpretes de JavaScript.

Por poner los dos ejemplos más claros:

- Mozilla Firefox Nightly
- Google Chrome Canary

Las diferentes plataformas ponen a disposición de los usuarios las últimas versiones de sus navegadores en las que trabajan en desarrollar las últimas funcionalidades para que los desarrolladores puedan probar la especificación de forma nativa.

Sin embargo, a pesar de lo satisfactorio y fluido de esto, hay que pensar que nuestros clientes, los usuarios finales de las páginas y aplicaciones del lado de cliente que creemos y que no necesariamente tienen que tener la última versión de nuestro navegador. Es por eso que nuestras herramientas de tooling deben someter nuestro código **ES6** (import / export) a un proceso de transpilación a un estándar previo mucho más extendido, **ES5** , para asegurar que la mayoría de nuestros usuarios tengan soporte en su navegador.

En **JavaScript** la herramienta que realizada este proceso más aceptada es Babel.js . Para que os hagáis una idea de un ejemplo real: la función flecha (arrow function) .

```
// Nuestro código ES6 con el setup de la práctica de Tooling
[ 1 , 2 , 3 ].map((n) => n + 1 );
// El código final transpilado a ES5
[ 1 , 2 , 3 ].map( function (n) {
  return n + 1 ;
});
```

Si queréis hacer pruebas sobre la transformación de código podéis usar el REPL de Babel.js .

Es interesante echar un vistazo a las reflexiones del creador de **JavaScript** para entender las cosas que quizás hubiera hecho diferente de haber anticipado el marco y uso actual de esta tecnología que desarrolló.

Situación actual

Al empezar el curso os hablé de lo cambiante y exigente que es el escenario **frontend** . Una industria que no para de avanzar y donde los estándares, tecnologías y paradigmas cambian, avanzan y mejoran a velocidades de vértigo.

Aterrizamos en la asignatura, el lenguaje y la tecnología en un momento de cambio y es eso precisamente junto con el incremento de la demanda de profesionales la que hace que cada día se abran más plazas en el sector:

- el incremento en valor absoluto de la demanda de profesionales
- los huecos dejados por profesionales que no se adaptan a los nuevos estándares

Aprovechar esta ventana de oportunidad requiere una constante inversión en aprendizaje y actualización y, en el caso de atacar por primera vez esta tecnología, la necesidad de contextualizar lo actual con lo pasado y lo futuro.

Os recomiendo que uséis **ES6** en las prácticas **.** Creo que hacer el esfuerzo de aprender los últimos estándares es lo más adecuado pero también es necesario que conozcáis su equivalente **ES5** para poder entrar a comprender cualquier base de código en **JavaScript** independientemente de la versión de la especificación usada.

Buceando en JavaScript

Nota: el trimestre anterior utilizamos Eloquent Javascript como hilo conductor y debido a los comentarios y valoraciones de los alumnos hemos sometido el temario a cambios y ajustes. Durante este trimestre usaremos dos materiales diferentes de forma simultánea por lo que os agradecería que me hagáis llegar cualquier feedback sobre el valor de uno sobre otro o vuestra percepción de facilidad de aprendizaje o conveniencia a nivel de aprendizaje. El contenido y los temarios se solapan así que si consideráis que con uno de los dos materiales tenéis suficiente no tenéis por qué revisar el otro siempre y cuando hayáis entendido y asimilados los conceptos asociados.

Como comentábamos al principio **Douglas Crockford** ayudó a dar empaque a **JavaScript** como lenguaje de programación a costa de separar las partes buenas de las malas.

No dudéis en revisar esta clase maestra sobre JavaScript, the Good Parts [Safari] .

Tiempo después Kyle Simpson argumentaba que no había tales partes malas sino una incapacidad de comprensión total de las motivaciones tras esas partes y por tanto se les daba un mal uso por condicionantes previos por otros lenguajes y metodologías.

Kyle Simpson nos presenta un JavaScript donde todas sus funcionalidades son buenas y malas en base al uso que se les dé y será su serie de libros You Don't Know JavaScript la que usaremos para aprenderlas.

La **PEC** de esta unidad consiste en una serie de ejercicios sencillos agrupados por temática y, en general, de variable complejidad que nos servirán como estiramiento y práctica de las peculiaridades de **JavaScript** . Es común en lo relativo a programación referirnos a estos ejercicios como katas, término propio de las artes marciales pero con el mismo propósito: practicar y aprender.

Nota: haremos la Kata #0 de forma guiada para que veáis el planteamiento a seguir usando la metodología TDD y las demás quedan a vuestro desarrollo personal.

A continuación os dejo la lista de recursos a revisar junto con los ejercicios de la **PAC1** a los que se corresponde:

Katas: #1 a #5

- [YDKJS] Conceptos básicos de programación
- [YDKJS] Conceptos básicos de JavaScript
- [Eloquent JS] Estructura de un programa
- [YDKJS] Tipos
- [Eloquent JS] Estructuras de datos: objetos y matrices
- [YDKJS] Valores
- [YDKJS] Nativas
- [Eloquent JS] Valores, tipos y operadores
- [YDKJS] Coerción
- [YDKJS] Gramática

Katas: #6 y #7

- [YDKJS] Scope y closure
- [Eloquent JS] Funciones
- [YDKJS] this y this

Katas: #8 a #10

- [YDKJS] Objetos, prototipo y clases (menos capítulos 1 y 2)

- [Eloquent JS] High order functions
- [Eloquent JS] La vida secreta de los objetos

Katas: #11 y #12

- [Eloquent JS] Bugs y errores
- [Eloquent JS] Expresiones regulares

Katas: #13

- [YDKJS] Nuevas funcionalidades ES6

Módulos

Llegados a este punto tenemos suficiente experiencia y conocimiento para entender la evolución de la modularización de JavaScript. Hemos usado módulos en el desarrollo TDD: implementamos la función en un módulo y la exportamos para más tarde importarla en su fichero spec y poder pasarle todos los tests.

Para ganar contexto os recomiendo las explicaciones del genial ingeniero de Google Addy Omani sobre:

- El patrón módulo (module pattern)
- El patrón de módulo "revelador" (revealing module pattern)

Estas dos maneras de organizar el código permite:

- Mejorar el **mantenimiento** del código al trabajar con unidades más pequeñas e independientes
- Separar funcionalidades y facilitar su **reusabilidad** en diferentes partes del código
- Permitir la **privacidad** de métodos y variables gracias al closure de las funciones

Esta funcionalidad tan obvia y útil no existía de inicio aunque los patrones de diseño mencionados anteriormente habilitaban los proyectos a tener cierta modularización.. En 2009 apareció la especificación de CommonJS que permitía reutilizar partes del código con una API más estricta y asentada.

Sin entrar en muchos detalles hay una versión asíncrona de CommonJS que se desarrolló para su integración en las peticiones del navegador llamada AMD. Conceptualmente es el mismo procedimiento pero con el añadido de la gestión asíncrona de las peticiones.

Por último y más reciente, la última revisión de ECMAScript introdujo dentro del estándar por primera vez el concepto de módulos que son los que hemos utilizado en nuestro desarrollo TDD y que volveremos a utilizar de formas exhaustiva en las próximas PECs:

- [YDKJS] Módulos ES5 y Módulos ES6
- [Eloquent JS] Módulo s
- [FreeCodeCamp] Módulos en JavaScript