ATRESMEDIA

i3Televison Modules

Arquitectura de módulos Javascript en los portales de Atresmedia

Modularizando la Web

WEB 2.0 (Mootools, Prototype, jQuery)

En el origen sólo existía llamada y respuesta HTTP al servidor

La introduccion de las **llamadas AJAX**, abrieron la posibilidad a un nuevo modo de entender la web, donde no había que volver a pedir una nueva página HTML al servidor. Fue entonces donde el Javascript ganó fuerza, pero nada podía hacer competencia a aquellas fantasticas animaciones con audio y video que se conseguían realizar con FLASH, donde el negocio de la publicidad encontró un nicho que explotar y dominar durante mucho tiempo.
Malos tiempos para javascript.

HTML5, CSS3, V8, iPhone (jQuery, jQueryUI)

La llegada de **HTML5 y CSS3**, daba la posibilidad de llevar a los standares web parte de la potencia que hasta el momento sólo se conseguía con FLASH.

Se introducían **transiciones y animaciones** mediante css3, **contenido responsive** y se introducían elementos nuevos que harían que Flash fuese perdiendo fuerza, etiquetas como **<video> y <audio>**, todo ello con un fuerte contenido semántico, permitiendo hacer sites accesibles e indexables.

Dentro de este marco de cambio de la web, irrumpieron los **dispositivos móviles**, los cuales permitirían navegar por la web, mediante standares HTML5. Steve Jobs anunció que no permitiría la instalación de plugins de terceros en los iPhones, lo que supuso relegar el uso de Flash a la web y en concreto al vídeo.

Malos tiempos para Java y Flash.

MVC y Node.js (MVC)

La llegada del **motor de javascript de Google(V8)**, mas rápido y potente, rompió la hegemonía de Firefox e Internet Explorer, y poco a poco, en un corto periodo de tiempo, se convirtió en el navegador preferido de los usuarios. Su motor **Webkit** era el mismo que el que utilizaba Safari en la web y en los dispositivos Apple aceleraron los standares de la web.

La llegada de **Node.js** donde ya se podía escribir cliente y servidor en un mismo lenguaje, ademas de poder realizar utilidades **CLI(Command line Utilities)** dieron un nuevo impulso al mundo del Javascript, que había encontrado la forma de saltar de la web al Sistema Operativo.

Pronto empezaron a salir las primeras soluciones **MVC javascript** introduciendo un nuevo concepto en la web SPA + API REST, aprovechando que los navegadores cada vez eran mas potentes procesando Javascript, se empieza a tomar la decision de dejarle gran parte de la carga al cliente y aligerar las llamadas al servidor.

En éste marco, aparecen los primeros Frameworks y librerías tales como: **Backbone, AngularJS, Ember, Knockout, Marionete, ...**, y también aparecen los primeros motores de plantillas para javascript: **Mustache, Handlebars, EJS, ...**

Buenos tiempos para el desarrollo WEB y el Javascript

ES6, Web Components y React

Desde la introducción de HTML5, el futuro de HTML ya no pasa por crear un nuevo mundo de etiquetas nuevas que los navegadores tengan que implementar, ya que el ecosistema de etiquetas es ya bastante rico. En éste punto Javascript cobra más fuerza, y se abre el melón de **madurar Javascript** ya que el lenguaje había evolucionado poco desde sus origenes y se empieza a crear un ecosistema de lenguajes como **CoffeScript o typeScript**, que compilan en javascript intentando hacer de éste un lenguaje mas cercano a lenguajes clásicos.

Llega la revolucion del lenguaje y Javascript se abre a estos desarrolladores clásicos pero sin perder su potente esencia. Llega **ES6**, y se introducen mejoras en la sintaxis permitiendo hacerlo un lenguaje más modular, como por ejemplo, **el sistema de clases y la importación/exportacion de módulos**. También se introducen mejoras en el HTML y se eleva el componente HTML a nivel de clase, permitiendo la creación nuevos elementos HTML o la extensión de elementos HTML ya existentes, por parte del desarrollador; son los llamados **"Custom Elements"**, dando la posibilidad a os desarrolladores de crear elementos a la altura de cualquier otro elemento HTML, como un div, un button, un textarea o un input.

Aprovechando esa capacidad Google inventa el **Web Component** que se compone de 4 partes(Template / Imports /Custom Element / Shadow DOM), pero que sólo se implementa en Chrome, lo que no es un standard.

React surge en ese entorno de creación de componentes y lo potencia haciendo que su fuerte sea trabajar los cambios en el DOM sin tener que repintar todos los Nodos por ello se definen como la V de MVC, e introduce otro punto fuerte frente a otrso lenguajes y frameworks que el polimorfismo (la posibilidad de utilizar el mismo código en el cliente y en el servidor), solucionando un handicap que tenían todas las SPA's, ya que si todo el código se ejecuta en el cliente, no existe contenido para indexar ya que los robots no ejecutan javascript.

Spaghetti Module Code

Aquí no hay quien meta la mano

WTF!!

```
$(function(){
        // Módulo que hace una peticion Ajax cada 20 segundos y actualiza contenido
        function creaModulo(){
                var $modulo = $('.miModulo');
                setInterval(doAjax,20000);
                function doAjax(){
                        $.get('http://www.miweb.com/servicio', function(data){
                                $modulo.each(function(){
                                        $(this).text(data);
                                });
                        });
                }
```

Básicamente se convierte en una sobrepoblacion de variables globales y funciones encadenadas que hacen complicado su mantenimiento y su escalabilidad.

Cómo cargamos los módulos en la página

Si tenemos muchos módulos el sistema de carga de archivos de HTML nos obliga a incluir etiquetas "script" con la carga de todos los archivos que queramos incluir.

Podemos incluirlos todos en un único archivo, ya sea escribiendolos todo de una tacada o bien utilizando alguna utilidad que los concatene.

Surge la necesidad de importar módulos y aparecen dos standares

- **AMD (Asyncronous Module Definition)** - Usada por RequireJS
- **CJS (Common JS)** - Usada por Node.js

La necesidad y la polivalencia de poder utilizar cualquier módulo en cualquier sistema hace que surja un nuevo standard

- **UMD (Universal Module Definition)** - Surge a posteriori, para poder utilizar un mismo módulo en sistemas AMD, CJS, y en el sistema clásico del espacio global de nombres (window)

¿ Y que hay de ES6 ?

¡¡ Tenemos Babel !!
Transpila nuestro código para que funcione en todos los navegadores modernos

Node.js es javascript, ¿ puedo utilizar librerías de Node.js?

¡¡ Tenemos Browserify !!
Permite usar módulos de Node en el cliente

¡¡ SHOW ME THE CODE !!

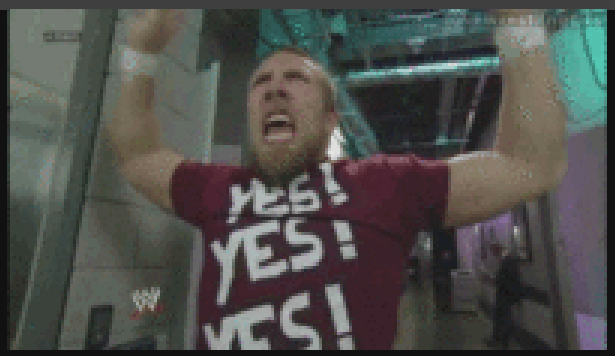
Cómo lo estamos haciendo en ATRESMEDIA

Si puedes hacerlo mejor, ...HAZLO!!!

Modulo ATRESMEDIA

- **Grunt** (Gestiona las tareas automatizadas del proyecto)
- **Browserify** (Genera el modulo final para ser consumido)
- **Nunjucks [nunjucksify]** (Motor de plantillas que usa el módulo)
- **Babel [babelify]** (Nos permite usar ES6 en nuestros módulos)

```
├── build/
│   ├── atresmedia_Hello.js
│   └── atresmedia_Hello..min.js
├── src/
│   ├── views/
│   │   └── MainView.js
│   ├── models/
│   │   └── SimpleModel.js
│   ├── templates/
│   │    └── MainTemplate.nunj
│   └── atresmedia_Hello.js
├── wwww-test/
```



Mi primer Módulo

Yeoman al rescate

Con el Generador de módulos de Yeoman tendrás todo lo necesario para empezar tu módulo con todo lo necesario
Sólo necesitarás **elegir un nombre**, por nomenclatura que empiece por "a3mod_" ó "atresmedia_".
a3mod_login, atresmedia_carrusel,...

El nombre del módulo se utilizará para la **generacion de archivos**, su uso en el **Window como nombre global** del módulo y la **carga dinámica de módulos**

```
# instalación de Yeoman
> npm install -g yo
# instalación de del generador de intergitlab
> npm install -g generator-backbone_module
# Creacion de carpeta del módulo -- nomenclarura de módulos [ a3mod_ | atresm
> mkdir atresmedia_Hello && cd atresmedia_Hello
# instalación de del generador de intergitlab
> yo backbone_module
```

¡Ya tenemos un scafolding de un módulo listo para empezar a programar!

Desmontando Gruntfile

Desde el Gruntfile.js vamos a controlar todo el proceso de creación del módulo y su generación en un módulo consumible

- **pkg** el package.json (El archivo madre)
- **config** Configuracion de variables (nombres de carpetasd de publicacion, test, y sources, puestos de connect,...)
- **jshint** Control de calidad del código
- **bower** Lanza la descarga de dependencias por bower
- **browserify** Desde el punto de entrada de nuestra aplicacion construye un archivo listo para usarse en el navegador y lo pone en la carpeta "build".
  Parte de la configuracion se encuentra en el ***package.json***
- **uglify** Minimificado del archivo generado por "browserify" en la carpeta "build"
- **concat** Concatenado de archivos de vendors, para probar en desarrollo
- **connect** Lanza un servidor de pruebas arrancando en la carpeta "www-test", expone toda la jerarquia como si fuese la raiz
- **watch** Escuchamos cambios en los archivos y lanzamos el livereload cuando hay cambios

Punto de entrada del módulo src/atresmedia_Hello.js

```
import MainView  from './views/MainView';

function moduleFactory(opts, pubsub){

        var MODTYPE = {
                'default'       : MainView
        };

        // Obtenemos los atributos datas del módulo
        var datas = opts.el.data();

        // Si viene el canal PUBSUB lo incluímos
```

- Factoría que devuelve una vista **Mainview** importada desde **./views/MainView**, en éste proceso tanto los "requires" como los "imports" son **"browserificados"** incluyendolos en el archivo final.
- Se suele utilizar el atributo **data-theme** del elemento html para gestionar distintas vistas.
- La vista que se importa suele ser una vista **Backbone**, por lo que el primer argumento (opts) es siempre un objecto que backbone puede interpretar.
- El segundo argumento (pubsub) es el canal de **publicación/subscripcion** que suele estar declarado en todos los sites en una etiqueta "script"

```
var evt_agr = _.clone(Backbone.Events);
```

Vista **src/views/MainView.js**

```
// Custom filters
var nunjucks = require( 'nunjucks' );
nunjucks.env = new nunjucks.Environment();
nunjucks.env.addFilter( 'test', function( test, text ) {
    return test + (text || ' (Nunjucks Test Filter)');
});

var MainView = Backbone.View.extend({
        template        : require('../templates/MainTemplate.nunj'),
        pubsub          : _.extend({}, Backbone.Events),

        initialize: function(opts){
```

- Es un módulo que expone la vista de Backbone **MainView**
- La propiedad "template" require la plantilla de nunjucks, que luego browserify sabe interpretar gracias al transformador de "nunjucksify"
- Finalmente renderiza el contenido de la plantilla en el interior del módulo mediante el método render de la vista

El módulo funcionando **"module_loader"** probando en **www-test/index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Module</title>
</head>
<body>
  <div class="container">
    <div data-mod="atresmedia_Hello"></div>
  </div>
  <script type="text/javascript" src="/public/vendors.js"></script>
  <script type="text/javascript" src="/build/atresmedia_Hello.js"></script>
```

- Incluímos el script de **vendors.js** y el del módulo "browserificado" **atresmedia_Hello.js**
- En los vendors se incluye el módulo **"module_loader"** que hace lo siguiente:
  - Buscar todos los elementos con el atributo **data-mod** (data-mod_type en OndaCero)
  - Ejecuta el valor del atributo en el espacio global de nombres **window["atresmedia_Hello"]** pasandole como parámetros el elemento html que tiene asociado, y el evt_agr (canal pub/sub de la página)
  - Si no existe, hace una **carga asíncrona del módulo (con SystemJS)**, buscando un archivo en la siguiente ruta **http://[jsDomain]/modules/atresmedia_Hello.min.js** y hace la misma ejecución que el método anterior en caso de que la carga del archivo sea satisfactoria

Probando el Módulo ATRESMEDIA

Modulo ATRESMEDIA

```
> grunt server
```

Este proceso inicia **bower**, **browserify**, genera los archivos del módulo en la carpeta **/build** y lanza un servidor de pruebas que se encuentra en la carpeta **www-test** en el puerto http://localhost:3000/
Cualquier cambio en el proyecto fuerza la recarga de la página gracias a **livereload** que se encuentra incluida en la tarea de **Grunt**

Si todo ha salido bien, deberíamos de ver el resultado de la plantilla en el/los elementos cuyo atributo sea **data-mod="atresmedia_Hello"**

Publicando el Módulo ATRESMEDIA

Gitlab: Es el lugar donde llevaremos el control de versiones y "tagearemos" el proyecto sincronizandolo con la version del modulo

Private Bower: Repositorio de paquetes de Bower.
Hace de proxy contra el gitlab para buscar las versiones del módulo.
Un site, por ejemplo www.antena3.com, incluye muchos módulos y los importamos mediante bower, recogiendo los archivos que exponemos en la carpeta **build** de cada módulo

Sinopia: Repositorio de paquetes de node.
Siempre que queramos utilizar algúna funcionalidad o plantilla de otro módulo en tiempo de desarrollo.

Configurando npm y bower

Configurar Sinopia

```
npm set registry http://intersinopia:4873
npm adduser --registry http://intersinopia:4873
```

Configurar Private Bower

```
{"registry": "http://intersinopia:5678","timeout": 300000}
```

Ya podemos publicar y consumir nuestros propios módulos

Publicando

1. Hacer la primera subida al GITLAB 

```
> git init
> git remote add origin [git://url_repositorio_git.git]
> git add --all
> git commit -am "Release"
> git push -u origin master
```

2. Publicar en Sinopia actualizando version npm

```
> npm version [ patch | minor | release]
> npm publish
```

3. Publicar en bower [Private Bower] 

```
> bower register [nombre_del_modulo] [http://url_repositorio_git]
```

4. Subir los cambios y el nuevo tag al GITLAB

```
> git push --follow-tags
```

Modificando un módulo ya creado y dado de alta en Private Bower

2. Publicar en Sinopia actualizando version npm

```
> npm version [ patch | minor | release]
> npm publish
```

4. Subir los cambios y el nuevo tag al GITLAB

```
> git push --follow-tags
```

Resumiendo el ciclo de vida de un Módulo

ATRESMEDIA

- Creación
  - Scafolding de Yeoman
- Desarrollo
  - GRUNT + BABEL +
- Publicacion
  - Crear repositorio en GITLAB y hacer la primera subida

    ```
    > git push --follow-tags
    ```
  - Versionado y publicación en Sinopia

    ```
    > npm version patch && npm publish
    ```
  - Registrar el módulo en Private bower

    ```
    > bower register [nombre del módulo] [http://repositorio_GIT]
    ```
  - Subir tageado en Gitlab siguiendo los tags del package.json

    ```
    > git push --follow-tags
    ```
- Actualización/Nuevas funcionalidades
  - Versionado y publicación en Sinopia

    ```
    > npm version patch && npm publish
    ```
  - Subir tageado en Gitlab siguiendo los tags del package.json

    ```
    > git push --follow-tags
    ```

JAM TIME

Ruegos, preguntas, súplicas, whatever...

See you later...