actibyti

a netmind social project

DISEÑO MODULAR

Autor: Ricardo Ahumada

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"

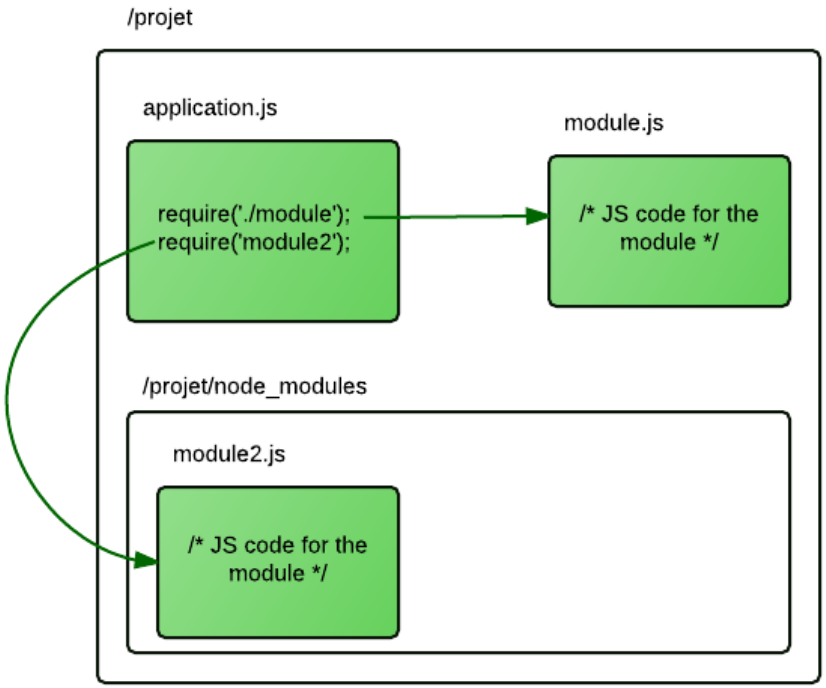
ÍNDICE DE CONTENIDOS

1 Diseño Modular

Diseño Modular

- El sistema modular de JavaScript se vuelve cada vez mas importante para los desarrolladores web

- Los módulos son grupos de código, los cuales son completamente independientes entre sí, con funcionalidades distintas, y que pueden ser mezclados, añadidos y eliminados, sin alterar el sistema en su conjunto.



Ventajas del diseño Modular

Su utilización es ventajosa, sobre todo cuando el sistema crece, y queremos independizar bloques de código.

- **Mantenimiento**, un modulo bien diseñado, disminuye al máximo las dependencias, por tanto su crecimiento o rediseño no afecta a los demás. Sería desastroso para un sistema si la modificación de una parte del código llegase a poner en riesgo su funcionamiento.

- **Espacios de nombre**, las variables fuera de las funciones se vuelven globales, es decir, cualquier modulo puede acceder a ellas, cuando esto sucede, se vuelve común que los módulos se contaminen de nombres de variables que ya están definidas, es decir, códigos completamente independientes compartiendo variables globales, lo cual es una mala practica. Cada modulo, es capaz de crear sus variables en un espacio privado.

- **Reutilización,** un modulo es susceptible de ser usado por diversas aplicaciones en distintos contextos, ya que expone una interface.

Patrón Module

- El patrón Module se usa para imitar el concepto de clase (ya que JavaScript no soporta clases), para poder almacenar métodos y poder trabajar con variables públicas y privadas dentro de un objeto –de manera similar a Java o Phyton.
- Esto nos permitirá crear una API para los métodos que queramos exponer al mundo, a la vez que encapsulamos variables y métodos privados en un contexto cerrado.
- Hay varias maneras de implementar el patrón module, la más básica es usar un **closure anónimo**.
- Pero también se pueden usar otras estrategias como **global import**, **Interfaz de objeto autocontenido** o el **patrón Revealing Module**.

Ejemplo - Closure anónimo

- La función anónima contiene su propio entorno de evaluación o "closure",
- Nos permite ocultar variables globales del contexto padre (global)
- Permite trabajar, de esta manera, con variables locales, pero aun teniendo acceso a el uso de las variables globales.

```
(function () {
  // We keep these variables private inside this closure scope
  var myGrades = [93, 95, 88, 0, 55, 91];

  var average = function() {
    var total = myGrades.reduce(function(accumulator, item) {
                return accumulator + item
                }, 0);

    return 'Your average grade is ' + total / myGrades.length + '.';
  }

  var failing = function(){
    var failingGrades = myGrades.filter(function(item) {return item < 70;});

    return 'You failed ' + failingGrades.length + ' times.';
  }

  console.log(failing());

}());

// 'You failed 2 times.'
```

Ejemplo – Global import

- Otro enfoque utilizado por librerías como jQuery es la importación global.
- Es similar al closure anónimo, pero ahora pasamos los globales como parámetros
- La ventaja de este método sobre las funciones anónimas es que se declaran las variables globales al principio, lo cual hace sencillo comprender el código.

JavaScript

```
(function (globalVariable) {

  // Keep this variables private inside this closure scope
  var privateFunction = function() {
    console.log('Shhhh, this is private!');
  }
```

http://jsbin.com/wajugenoco/edit?js,console

Ejemplo – Interfaz de objeto autocontenido

- Otra aproximación es crear un módulo usando una **interfaz de objeto autocontenido**.

```
var myGradesCalculate = (function () {

  // Keep this variable private inside this closure scope
  var myGrades = [93, 95, 88, 0, 55, 91];

  // Expose these functions via an interface while hiding
  // the implementation of the module within the function() block

  return {
    average: function() {
      var total = myGrades.reduce(function(accumulator, item) {
        return accumulator + item;
        }, 0);

      return'Your average grade is ' + total / myGrades.length + '.';
    },

    failing: function() {
      var failingGrades = myGrades.filter(function(item) {
          return item < 70;
        });

      return 'You failed ' + failingGrades.length + ' times.';
    }
  }
})();

myGradesCalculate.failing(); // 'You failed 2 times.'
myGradesCalculate.average(); // 'Your average grade is 70.33333333333333.'
```

Patrón Revealing module

- Es muy similar al enfoque anterior, excepto que asegura que todos los métodos y variables se mantengan privados hasta que estén explícitamente expuestos

```
var myGradesCalculate = (function () {

  // Keep this variable private inside this closure scope
  var myGrades = [93, 95, 88, 0, 55, 91];

  var average = function() {
    var total = myGrades.reduce(function(accumulator, item) {
      return accumulator + item;
      }, 0);

    return'Your average grade is ' + total / myGrades.length + '.';
  };

  var failing = function() {
    var failingGrades = myGrades.filter(function(item) {
        return item < 70;
      });
    return 'You failed ' + failingGrades.length + ' times.';
  };

  // Explicitly reveal public pointers to the private functions
  // that we want to reveal publicly
  return {
    average: average,
    failing: failing
  }
})();

myGradesCalculate.failing(); // 'You failed 2 times.'
myGradesCalculate.average(); // 'Your average grade is 70.33333333333333.'
```

Pongámoslo en práctica

- Crea un módulo que encapsule el procesado de strings para generar procesos:
  - Primera mayúscula
  - Tipo Oración
  - Minúsculas

Pongámoslo en práctica

- Usando el módulo anterior, crea un módulo que aplique las funcionalidades a un elemento concreto del DOM

2 CommonJS y AMD

CommonJS

- **CommonJS** (http://www.commonjs.org/) permite definir APIs que manejan muchas necesidades comunes de aplicaciones, haciéndolos disponibles a otros módulos.
- CommonJS es un grupo de voluntarios que se encargan de diseñar e implementar las API de JavaScript para declarar módulos.
- Para definir un modulo CommonJS, se hace de la siguiente manera:

```
function myModule() {
  this.hello = function() {
    return 'hello!';
  }

  this.goodbye = function() {
    return 'goodbye!';
  }
}

module.exports = myModule;
```

CommonJS

- Usamos el objeto especial del módulo y colocamos una referencia de la función en **module.exports**.
- Cuando se quiera utilizar *myModule*, se puede llamar usando **require**:

```
var myModule = require('myModule');

var myModuleInstance = new myModule();
myModuleInstance.hello();
myModuleInstance.goodbye();
```

- De este modo, se evita la contaminación de namespaces globales, y nuestras dependencias se vuelven mas explícitas.

Instalación

- Se puede descargar del sitio
  - http://umbraengineering.github.io/common.js/
- También se puede usar npm

```
npm install common.js
```

Usar CommonJS

1. Escribir el módulo y su cliente

```
/*foo.js*/
module.exports = function() {
    console.log('foo');
};
```

```
/*bar.js*/
var foo = require('foo');

exports.doTheThing = function() {
    foo();
};
```

2. Hacer build usando el procesador commonjs

```
commonjs --src ./path/to/javascripts --dest ./path/to/js --client
```

3. Incluir el módulo en la aplicación

```
<script src="js/common.js"></script>
<script src="js/foo.js"></script>
<script src="js/bar.js"></script>
<script>
    var bar = require('bar');

    bar.doTheThing();
</script>
```

AMD – Asynchronours Module Definition

- AMD (Definición de módulos asíncronos), como su nombre lo indica hace referencia a la carga de módulos de manera asíncrona
- La sintaxis es como sigue:

```
define(['myModule', 'myOtherModule'], function(myModule, myOtherModule) {
  console.log(myModule.hello());
});
```

  - La función **define** toma como su primer argumento una matriz de dependencias del módulo.
  - Estas dependencias se cargan en segundo plano (de forma no bloqueante),
  - Una vez cargadas define llama a la función de callback correspondiente.
  - La función callback toma, como argumentos, las dependencias que se cargaron permitiendo que la función use estas dependencias.
  - Finalmente, las propias dependencias también deben definirse utilizando la palabra clave define.

Módulos con nombre

- El módulo anterior no declara un nombre para sí mismo. Esto permite que el módulo sea muy portable:
  - Un desarrollador puede colocar el módulo en una ruta diferente para darle un ID/nombre diferente.
  - El cargador de AMD le dará al módulo un ID basado en cómo es referenciado por otros scripts.
- Sin embargo, las herramientas que combinan varios módulos necesitan una forma de dar nombres a cada módulo en el archivo optimizado final (bundle).
- AMD permite indicar el nombre del módulo en el primer argumento de **define**():

```
define('myModule', ['dep1', 'dep2'], function (dep1, dep2) {
    //Define the module value by returning a value.
    return function () {};
});
```

Ejemplo – MyModule AMD

```
define([], function() {

  return {
    hello: function() {
      console.log('hello');
    },
    goodbye: function() {
      console.log('goodbye');
    }
  };
});
```

Ejemplo - Inclusión de múltiples módulos

```
define([ "require", "jquery", "blade/object", "blade/fn", "rdapi",
        "oauth", "blade/jig", "blade/url", "dispatch", "accounts",
        "storage", "services", "widgets/AccountPanel", "widgets/TabButton",
        "widgets/AddAccount", "less", "osTheme", "jquery-ui-1.8.7.min",
        "jquery.textOverflow"],
function (require,$,object, fn, rdapi,
         oauth,jig, url, dispatch,accounts,
         storage,services,AccountPanel,TabButton,
         AddAccount,less,osTheme) {

});
```

Implementaciones de AMD

- Existen unas cuantas implementaciones
  - RequireJS (http://requirejs.org/)
  - curl (https://github.com/cujojs/curl)
  - lsjs (https://github.com/zazl/lsjs)
  - Dojo 1.7+ (http://dojotoolkit.org/)

- La más popular es requirejs
- Además, si se necesitan convertir varios módulos el proyecto **r.js** tiene una herramienta de build
  - https://github.com/requirejs/r.js

Instalación de require.js

- Descargar require de
  - http://requirejs.org/docs/download.html

- Añadir *require.js* a los scripts de la página

```
<script data-main="scripts/main" src="scripts/require.js"></script>
```

Instalar r.js

- Instalar con la orden

```
npm install -g requirejs
```

- Hacer build con

```
node r.js -convert path/to/commonjs/modules/ path/to/output
```

CommonJS vs AMD

- A diferencia de CommonJS, AMD posee un **enfoque asíncrono**.
- Otra ventaja es que los módulos a cargar pueden ser objetos, funciones JSON y otros muchos tipos. CommonJS solo es compatible con objetos de tipo modulo.
- AMD no es compatible con io, sistema de archivos y otras funciones orientadas al servidor disponibles en CommonJS.
- En AMD, la función que envuelve la sintaxis es un poco más detallada en comparación con una simple declaración ***require***.

Dependencias circulares

- Cuantas más referencias se realicen dentro del código, mas complicado será de comprender.
- Esto puede llevar a que muchas referencias creen dependencias circulares, los cuales llegan a causar error.
- Un error común sucede cuando dos módulos AMD se intentan cargar de manera síncrona entre ellos.

- Por ejemplo en el código:

```
define( [ 'b' ], function() {
    function a() {
        b();
        console.log( 'a' );
    };
    return a; } );
```

```
define( [ 'a' ], function() {
    function b() {
        a();
        console.log( 'b' );
    };
    return b; } );
```

Dependencias circulares

- Lo mismo puede pasar si tenemos otro tipo de relaciones entre módulos, donde la circularidad no es tan evidente
- En el siguiente ejemplo en ES6 se evidencia esto:

```
// a.js
import B from 'b.js';
export default class A {
    get b() {
        return new B();
    }
}
```

```
// b.js
import A from 'a.js';
export default class B extends A {}
```

Dependencias circulares

- En el ejemplo anterior, si desde un Módulo C se require el módulo B, funciona bien, pero si se requiere el A, dará un error: *B requiere sincronamente A, mientra A no requiere B de la misma manera*

- Pero si se requiere A desde C; a su vez A sincrónicamente requiere B.
- En tal caso B no puede requerir A de forma síncrona, ya que causaría dependencia circular:
  - tratará de cargar A de forma síncrona, ya que A se necesita sincrónicamente para crear B, como una clase base
  - por lo que obtendremos un error.

Deshaciendo dependencias circulares

- De hecho, no se trata de dependencias circulares. Es una cuestión de capas.
- Dividir el código en capas: una estará siempre arriba del todo y condicionará el resto.
- El objetivo es crear la aplicación de tal manera que las capas sean lo más delgadas posible.
- Los módulos de en la capa inferior no debería ser consciente de los que están por encima de él.
- Sólo entonces, está bien tener referencias dentro de una capa (incluso las circulares) siempre y cuando la capa no crezca demasiado.

Pongámoslo en práctica

- Modifica los módulos anteriores para usar
  - CommonJS
  - AMD

3 Dependencias asíncronas

Porqué AMD

- Proporciona una propuesta clara sobre cómo abordar la definición de módulos flexibles.
- Es significativamente más limpio que el namespace global y las soluciones de etiquetas <script>: Hay una manera limpia de declarar módulos independientes y las dependencias que puedan tener.
- Las definiciones de los módulos están encapsuladas, lo que nos ayuda a evitar la contaminación del namespace global.
- Funciona mejor que CommonJS. No tiene problemas con el dominio cruzado, local o de depuración; y no tiene dependencia de las herramientas del lado del servidor a ser usadas. La mayoría de los cargadores de AMD admiten cargar módulos en el navegador sin un proceso de compilación.
- Proporciona un enfoque de 'transporte' para incluir múltiples módulos en un solo archivo. CommonJS aún no ha acordado un formato de transporte.
- Si es necesario, es posible hacer lazy load de scripts.

AMD define()

- Los dos conceptos claves para comenzar con módulos son la idea de un "**define**", lo cual facilita la definición de un módulo, y un "**require**", el cual carga las dependencias.

- En el siguiente ejemplo, se indica la sintaxis de "**define**" para definir un módulo:

```
define(
    module_id /*optional*/,
    [dependencies] /*optional*/,
    definition function /*function for instantiating the module or object*/
);
```

AMD Require()

- Se utiliza normalmente para cargar código a nivel superior en un archivo JavaScript o dentro de un módulo para buscar dependencias dinámicamente.

- El siguiente ejemplo de **require()**, muestra la sintaxis del mismo:

```
require(['foo', 'bar'], function ( foo, bar ) {
        // rest of your code here
        foo.doSomething();
});
```

Dependencias cargadas dinamicamente

- Es posible cargar módulos de manera dinámica, como se muestra en el siguiente ejemplo

```
define(function ( require ) {
    var isReady = false, foobar;

    // note the inline require within our module definition
    require(['foo', 'bar'], function (foo, bar) {
        isReady = true;
        foobar = foo() + bar();
    });

    // we can still return a module
    return {
        isReady: isReady,
        foobar: foobar
    };
});
```

Plugins

- Con AMD, es posible cargar activos de casi cualquier tipo incluyendo archivos de texto y HTML.
- Esto nos permite tener dependencias de plantillas que pueden usarse para pintar componentes ya sea en carga de página o dinámicamente.

```
define(['./templates', 'text!./template.md','css!./template.css'],
   function( templates, template ){
      console.log(templates);
      // do some fun template stuff here.
   }
});
```

Módulos con dependencias diferidas

- Es posible cargar dependencias diferidas a través de promesas.
- Esto podría ser compatible con la implementación diferida de jQuery, futures.js (sintaxis ligeramente diferente) o cualquiera de una serie de otras implementaciones

```
define(['lib/Deferred'], function( Deferred ){
    var defer = new Deferred();
    require(['lib/templates/?index.html','lib/data/?stats'],
        function( template, data ){
            defer.resolve({ template: template, data:data });
        }
    );
    return defer.promise();
});
```

4 Empaquetado

Empaquetado

- JS no es un lenguaje compilado, de preferencia, es recomendable usar un proceso de build para construir librerías (incluso las sencillas).
- Aunque existen excepciones, la estructura general de los directorios de las librerías de JS es la siguiente:

```
dist/
  commonjs/fong.js
  amd/fong.js
  glob/fong.js

src/
  fong.js

test/

README.md
Makefile
package.json
```

- La carpeta "**dist/**" se genera mediante un proceso de build y no debe ser editado manualmente.
- Los archivos del directorio "**src/**" deben ser actualizados por los autores del paquete.

Los módulos en la actualidad

- Formatos de módulos
  - CommonJS (http://wiki.commonjs.org/wiki/Modules)
  - AMD (https://github.com/amdjs/amdjs-api/wiki/AMD)
  - UMD (https://github.com/umdjs/umd)
  - ES6 Modules (http://www.2ality.com/2014/09/es6-modules-final.html#an_overview_of_the_es6_module_syntax)
- Activos para bundling
  - Browserify (http://browserify.org/)
  - jspm (http://jspm.io/)
  - Webpack (http://webpack.github.io/)
  - Rollup (http://rollupjs.org/)
  - Brunch (http://brunch.io/) / Brocolli (http://broccolijs.com/)
  - Sprockets (https://github.com/rails/sprockets)
  - Build your own with Gulp (http://gulpjs.com/) / Grunt (http://gruntjs.com/)

Los módulos en la actualidad (II)

- Librerías para carga dinámica
  - Require.js (http://requirejs.org/)
  - System.js (https://github.com/systemjs/systemjs)
- Transpiladores
  - Babel for ES6 (https://babeljs.io/)
  - CoffeeScript (http://coffeescript.org/)
  - Typescript (http://www.typescriptlang.org/)

Usando require.js + Bower

Crea el módulo

- El módulo está escrito en UMD para soportar todos los posibles estándar de carga

```
// calculator.js
(function (name, context, definition) {
  if (typeof module != 'undefined' && module.exports)
    module.exports = definition();
  else if (typeof define == 'function' && define.amd)
    define(name, definition);
  else
    context[name] = definition();
}('calculator', this, function () {
  // your module here!
  return {
    sum: function(a, b) { return a + b; }
  };
});
```

Crea el archivo main

- El archivo main es el script de entrada del módulo, donde se referencia a los otros módulos

```
// main.js
requirejs.config({ baseUrl: '/scripts' });
requirejs(['app']);
```

Instala Bower

- Teniendo en cuenta que las operaciones de RequireJS están totalmente limitadas a la carga del módulo, es necesario que se empareje con un gestor de paquetes, como Bower.
- Si lo hace, tendrá acceso a una enorme base de datos (+20.000) de módulos y librerías front de terceros que nos permitirán gestionar las dependencias, los conflictos de versionado y las descargas de módulos locales.

- En una consola de comandos, usa npm para instalar bower

```
npm install -g bower
```

Configuración del proyecto y dependencias

- Crea el archivo de configuración del proyecto **bower.json**

```
{
 "name": "my-project",
 "private": true,
 "dependencies": {
  "rsvp": "~3.0.16"
 }
}
```

- Descarga las dependencias

```
bower install
```

Bundle con requiere

- Ejecuta el bundle del módulo usando require.js

```
node r.js -o name=main out=bundle.js baseUrl=.
```

  - El proceso analiza los archivos que comienzan en un punto de entrada especificado (en nuestro caso, main.js) para reconstruir las dependencias en la base de las llamadas define() que encuentra a lo largo del camino.
  - A continuación, puede organizar y concatenar todos los archivos necesarios en un único archivo: bundle.jst

- Añade el bundle al HTML

```
<script data-main="scripts/bundle" src="scripts/require.js"></script>
```

Resultado

- El archivo bundle es simplemente una concatenación de los archivos Javascript necesarios en tiempo de ejecución:

```
// bundle.js
define("calculator", [],function() {
  return {
    sum: function(a, b) { return a + b; }
  };
});

define("app", ["calculator"], function(calculator)
{
  console.log(calculator.sum(1, 2)); // => 3
});

requirejs(['app']);
```

Usando Browserify + Npm

Browserify

- Como RequireJS, Browserify posee una herramienta de línea de comandos, que permite analizar módulos a partir de un punto de entrada dado (en este caso, app.js) en busca de ocurrencias de llamadas a **require()**.

- Podemos instalarlo usando npm

```
npm install -g browserify
```

- Si tenemos dependencias en el módulo las instalaremos con npm en nuestro proyecto. Por ejemplo, si dependemos de **underscore**

```
npm install underscore
```

Crea y usa el bundle

- Para empaquetar

```
browserify app.js --outfile bundle.js
```

- Si queremos parsear el contenido antes de empaquetarlo (por ejemplo para uglificarlo y minificarlo) usaremos la opción transform

```
browserify app.js --transform coffeeify  --transform uglifify --outfile bundle.js
```

- Usarmos el paquete en el HTML

```
<script src="js/bundle.js"></script>
```

Resultado

```
// bundle.js
debundle({
  entryPoint: "./app",
  modules: {
    "./app": function(require, module) {
      var calculator = require('./calculator');
      console.log(calculator.sum(1, 2));
    },
    "./calculator": function(require, module) {
      module.exports = {
        sum: function(a, b) { return a + b; }
      };
    }
  }
});

function debundle(data) {
  var cache = {};
  var require = function(name) {
    if (cache[name]) { return cache[name]; }
    var module = cache[name] = { exports: {} };
    data.modules[name](require, module);
    return module.exports;
  };
  return require(data.entryPoint);
}
```

Haciendo BananaTube Modular

- Revisa nuevamente la arquitectura de Banana y divídela en módulos que puedan interactuar entre ellos.
- Esto facilitará la evolución de BananaTube y permitirá reutilizar funcionalidades

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"