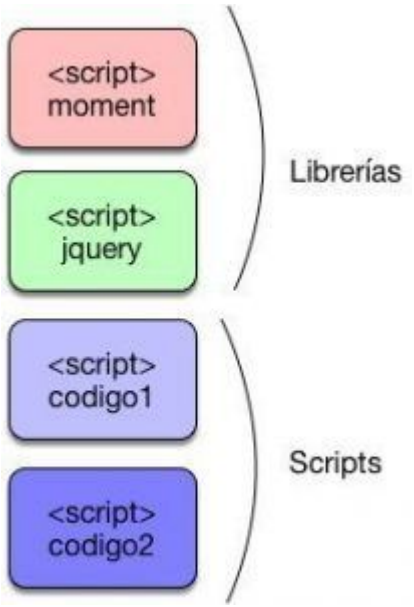
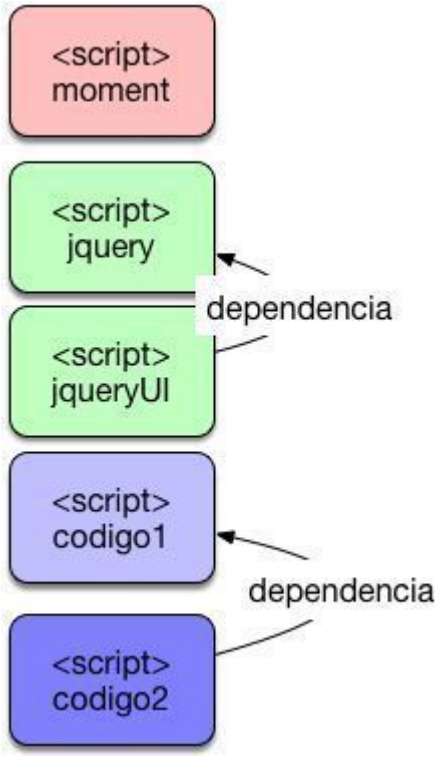
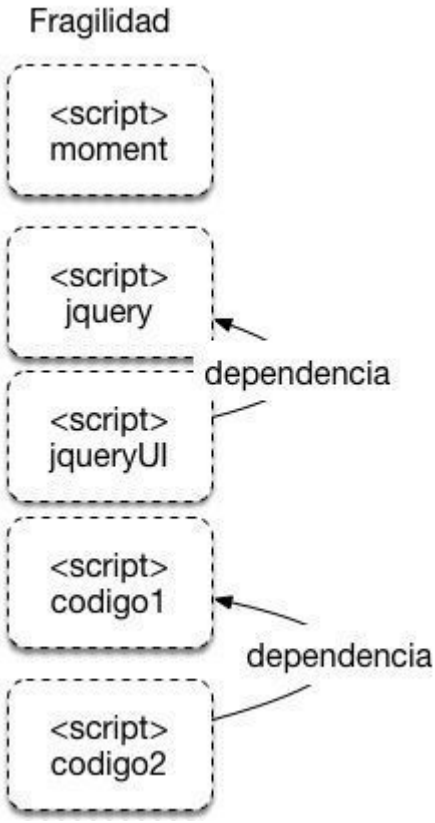
Webpack es un Module Bundler. Esto en un primer momento puede ser que no lo entendamos muy bien. Ahora bien todos necesitamos cuando programamos con JavaScript un Module Bundler . Otra cosa es que le usemos o no. Cuando uno trabaja en JavaScript se encuentra que usa una serie de librerías a nivel de navegador , jQuery ,Lodash etc. Para usarlas simplemente las añadimos como scripts en la página y una vez añadidas añadimos nuestros ficheros javascript con nuestro código.



Hay situaciones que se producen muy habitualmente de dependencias entre nuestras librerias . Es decir tendremos que añadir primero el script de jQuery que el de jQuery UI. De igual forma deberemos añadir primero nuestra librería A ya que la necesita nuestra librería B.



Cuando se trata de usar una librería y un pequeño código de JavaScript , resulta pesado pero no es un gran problema. Los problemas comienzan cuando nuestra aplicación tiene mucho código de JavaScript y muchas dependencias. Entonces el tema se complica sobre manera y todo parece muy frágil.



Las librerías tienen que ir en orden muy concreto para que todo funcione y con el paso del tiempo nadie se acuerda muy bien de porqué. No solo eso sino que a veces hay que descargarse 15 ficheros de JavaScript por parte del navegador lo cual va mas lento

Utilizando Webpack

Para solventar este problema ,necesitamos un module bundler como WebPack. Este tipo de herramientas nos permite trabajar con JavaScript de una forma natural definiendo las dependencias que tenemos entre las diferentes librerías y generando un empaquetado final con un único fichero de JavaScript que se carga en el navegador. Para ello se apoya en alguno de los sistema modulares de JavaScript .Vamos a ver un ejemplo apoyándonos en node. Para ello vamos a instalar jquery , moment y webpack con node.js.

```
npm install jquery

npm install moment

npm install webpack
```

Acabamos de instalar las tres librerías es momento de usarlas utilizando la gestión de dependencias de node.js. Para ello nos vamos a crear un módulo (usamoment.js) que nos devuelve una fecha usando moment.js.

```
var moment=require("moment");

function unaFecha() {

return moment("10-10-2017", "MM-DD-YYYY");
}

exports.unaFecha=unaFecha;
```

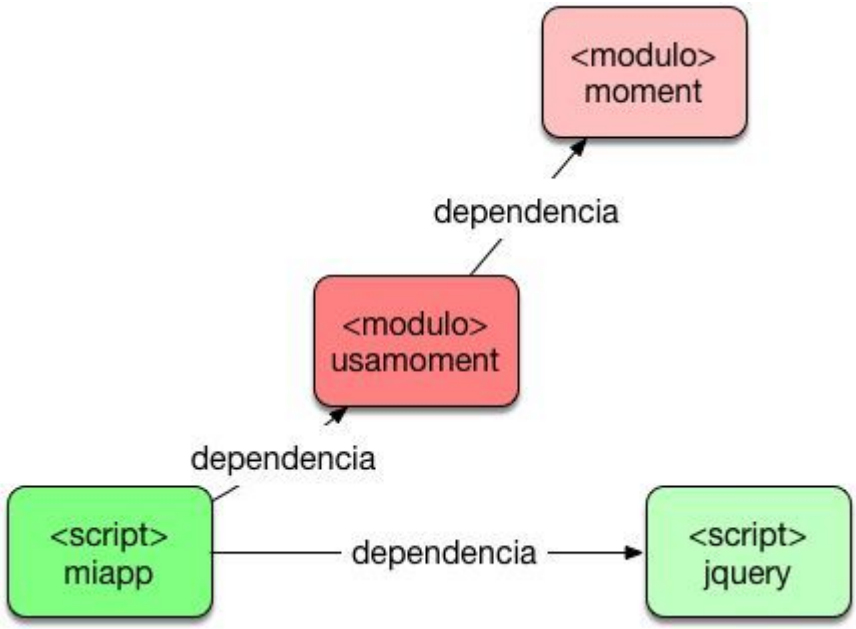
Luego nos vamos a crear un programa principal (app.js) que se apoye en este módulo y en jQuery.

```
var $=require("jquery");
var usamoment=require("./usamoment.js");

$(document).ready(function() {
```

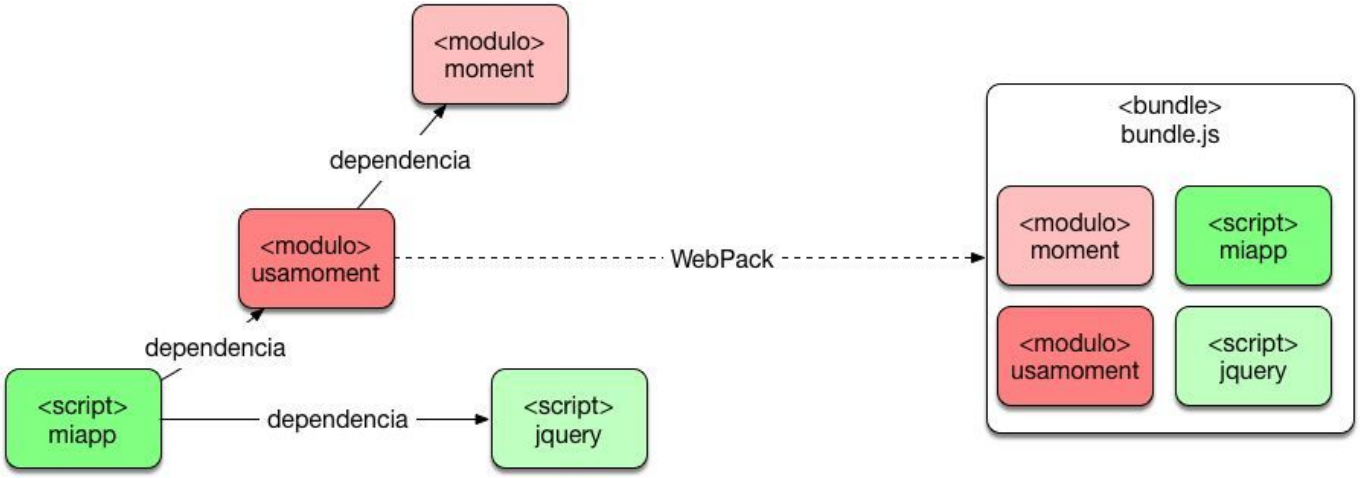
```
alert(usamoment.unaFecha().date());

});
```

Acabamos de usar el sistema de gestión de dependencias de node.js para crear nuestro programa principal . En el se define la dependencia de usamoment y la de jQuery.

.



El problema que tenemos es que un navegador no es capaz de entender este sistema de módulos y gestionar las dependencias. Para poder ejecutar nuestro pequeño programa en un navegador necesitamos utilizar webpack y generar un bundle.



Para ello nos crearemos un fichero de configuración de webpack (webpack.config.js) :

```
module.exports = {
entry: './miapp.js',
output: {
filename: 'bundle.js'
}
}
```

Este fichero define el fichero de javascript de entrada como el bundle de salida. Invocamos a webpack desde la linea de comandos

webpack

Nos generará un un bundle

```
Hash: 8744f16973ef2cda9d50
Version: webpack 3.6.0
Time: 711ms
    Asset    Size  Chunks                    Chunk Names
bundle.js  738 kB       0  [emitted]  [big]  main
 [116] ./app.js 141 bytes {0} [built]
 [118] ./usamoment.js 130 bytes {0} [built]
 [119] (webpack)/buildin/module.js 517 bytes {0} [built]
 [120] ./node_modules/moment/locale ^\.\/.*$ 2.79 kB {0} [optional] [built]
    + 117 hidden modules
```

El cual podemos cargar desde una página HTML sin problema .

```
<html>
<body>
<script type="text/javascript" src="bundle.js">
</script>
</body>
</html>
```

El resultado nos mostrará en un mensaje de alert el día 10 que contenía la fecha del módulo.

Acabamos de generar un bundle con nuestro código apoyandonos en Webpack.

Otros artículos relacionados:

1. JavaScript High Order Functions y su manejo
2. JavaScript Array Prototype y extensibilidad
3. JavaScript currying y funciones parciales.
4. Referencia