CommonJS modules

CommonJS modules son la forma original de packagear código javascript para node.js. Node.js también soporta ECMAScript modules standard usado por buscadores y otros runtimes de javascript.

En node, cada archivo es tratado como un módulo separado. Por ejemplo, considera un archivo llamado foo.js:

```
const circle = require('./circle.js');
console.log(`The area of a circle of radius 4 is ${circle.area(4)}`);
```

En la primer línea, foo.js carga el módulo circle.js que se encuentra en el mismo directorio que foo.js

Contenido de circle.js:

```
const { PI } = Math;

exports.area = (r) => PI * r ** 2;

exports.circumference = (r) => 2 * PI * r;
```

El módulo circle.js exportó las funciones area() y circumference(). Las funciones y objetos son añadidas a la raíz del módulo especificando propiedades adicionales en el objeto especial ‘export’.

Las variables locales al módulo van a ser privadas. En este ejemplo, la variable PI es privada a circle.js

La propiedad module.exports puede ser asignada un nuevo valor (como una función u objeto).

En el siguiente código, bar.js usa el módulo square, el cual exporta una clase Square:

```
const Square = require('./square.js');
const mySquare = new Square(2);
console.log(`The area of mySquare is ${mySquare.area()}`);
```

El módulo square es definido en square.js:

```
// Assigning to exports will not modify module, must use module.exports
module.exports = class Square {
  constructor(width) {
    this.width = width;
  }
```

```
  area() {
    return this.width ** 2;
  }
};
```

Sintaxis: CommonJS vs ES Modules

Ejemplo de CommonJS module que exporta 2 funciones:

```
module.exports.add = function(a, b) {
        return a + b;
}

module.exports.subtract = function(a, b) {
        return a - b;
}
```

y así se importan las funciones públicas en otro script node.js usando require:

```
module.exports.add = function(a, b) {
        return a + b;
}

module.exports.subtract = function(a, b) {
        return a - b;
}
```

Se pueden activar ES modules cambiando la extensión de .js a .mjs.

Ejemplo de un ES module (.mjs) exportando 2 funciones de uso público:

```
// util.mjs

export function add(a, b) {
        return a + b;
}

export function subtract(a, b) {
        return a - b;
}
```

y así se importan ambas funciones con la sentencia import:

```
// app.mjs

import {add, subtract} from './util.mjs'

console.log(add(5, 5)) // 10
console.log(subtract(10, 5)) // 5
```

Formas de activar ES Modules

Se pueden activar cambiando la extensión de los scripts de .js a .mjs, pero la mejor forma es agregar un campo “type: module” en el package.json:

```
{
  "name": "my-library",
  "version": "1.0.0",
  "type": "module",
  // ...
}
```

global keyword

En node.js el “top-level scope” no es el scope global; var nombre dentro de un módulo de node.js va a ser local a ese módulo, sin importar si es un módulo CommonJS o ECMAScript.