

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá importado módulos de sistema o integrados.
- Habrá creado módulos personalizados.
- Habrá importado módulos personalizados.
- Habrá configurado la lista de directorios de búsqueda mediante la variable de entorno `NODE_PATH`.

Objetivos

El objetivo de la presente práctica es mostrar cómo trabajar con módulos. Por un lado, módulos ya definidos por otros y, por otra parte, módulos propios.

Uso de módulos integrados

Recordemos que un módulo integrado o de sistema es aquel que viene de fábrica con `Node`. Estos módulos se importan indicando únicamente su nombre, es decir, mediante rutas de búsqueda. Veamos cómo usar el módulo `fs` que permite el acceso a archivos y directorios de la máquina.

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él.
3. Crear el archivo `archivo.txt`:

```
$ echo "esto es el contenido" > archivo.txt
```
4. Abrir `node` en modo interactivo:

```
$ node  
>
```

5. Importar el módulo `fs`:

```
> const fs = require("fs");  
undefined  
>
```

De manera predeterminada, el intérprete importa los módulos integrados cuando lo abrimos en modo interactivo. Por lo que su importación no es necesaria. Habrá variables homónimas a los módulos integrados. Pero recuerde, sólo en modo interactivo. En modo *batch*, no ocurre, hay que hacer las importaciones explícitamente.

6. Leer el contenido de `archivo.txt` mediante la función `readFileSync()` del módulo `fs`:

```
> fs.readFileSync("./archivo.txt").toString()  
'esto es el contenido\n'  
>
```

7. Mostrar los nombres de los objetos reutilizables del módulo `fs`:

```
> Object.keys(fs)
```

Definición de módulos personalizados

Ahora, vamos a definir un módulo personalizado que exporte dos funciones `sum()` y `sub()` que realizan la suma y la resta de los valores pasados como argumentos:

1. Crear el archivo `calcul.js` en el directorio de la práctica:

```
//api
module.exports = exports = {
  sum: sum,
  sub: sub
};

//suma
function sum(...args) {
  var res = 0;
  for (let arg of args) res += arg;
  return res;
}

//resta
function sub(...args) {
  var res = 0;
  for (let arg of args) res -= arg;
  return res;
}
```

2. Ir a la consola de **node** interactivo.

3. Importar el módulo:

```
> var calcul = require("./calcul")
undefined
>
```

Cuando se importa un módulo personalizado definido en el propio proyecto, hay que usar *siempre* una ruta relativa.

4. Mostrar el objeto API del módulo:

```
> calcul
{ sum: [Function: sum], sub: [Function: sub] }
>
```

5. Calcular la suma de los valores 1, 3 y 5:

```
> calcul.sum(1, 3, 5)
9
>
```

6. Mostrar la caché de módulos:

```
> require.cache
```

Recuerde que todo módulo que se encuentre en la caché no se volverá a cargar cuando sea importado de nuevo.

Directorios de búsqueda

Recordemos que los módulos se pueden ubicar en directorios ajenos al proyecto. Para informar a **node** en qué directorios debe buscar los módulos cuando indicamos sólo su nombre, hay que utilizar la variable de entorno **NODE_PATH**.

1. Ir a la consola.

2. Mostrar el contenido de la variable de entorno **NODE_PATH**:

```
> process.env.NODE_PATH
```

3. Importar el módulo **calcul**, sin indicar su ruta, sólo su nombre:

```
> calcul = require("calcul")
Error: Cannot find module 'calcul'
    at Function.Module._resolveFilename (module.js:469:15)
    at Function.Module._load (module.js:417:25)
    at Module.require (module.js:497:17)
    at require (internal/module.js:20:19)
    at repl:1:10
    at sigintHandlersWrap (vm.js:22:35)
    at sigintHandlersWrap (vm.js:96:12)
    at ContextifyScript.Script.runInThisContext (vm.js:21:12)
    at REPLServer.defaultEval (repl.js:313:29)
    at bound (domain.js:280:14)
```

>
Esto es normal. Como sólo se indica el nombre del módulo, `node` lo busca en la lista de directorios de la variable de entorno `NODE_PATH`, la cual no contiene el directorio actual.

4. Salir del intérprete interactivo:

```
> .exit  
$
```

5. Actualizar la variable de entorno `NODE_PATH` para que `node` busque también en el directorio actual.

En **Linux**:

```
$ NODE_PATH=$NODE_PATH:.
```

En **Windows**:

```
> $env:NODE_PATH+=";."
```

6. Abrir `node` en modo interactivo.
7. Mostrar el contenido de la variable de entorno `NODE_PATH`:

```
> process.env.NODE_PATH
```

8. Importar el módulo `calcul`, por nombre, no por ruta:

```
> var calcul = require("calcul")  
undefined  
> calcul  
{ sum: [Function: sum], sub: [Function: sub] }  
>
```

No es buena práctica añadir el directorio actual (.) en la lista de búsqueda de la variable de entorno `NODE_PATH`. Pero sí es una buena manera de ver cómo funciona la búsqueda de módulos.

Módulo principal

El módulo principal es aquel que pasamos a `node` en la línea de comandos. Su objeto módulo se encuentra en la propiedad `require.main`.

1. Ir a la consola.
2. Mostrar el módulo principal:

```
> require.main  
undefined  
>
```

Todo normal. Hemos arrancado el intérprete en modo interactivo. No le hemos pasado ningún archivo como punto de entrada, o sea, ningún módulo principal.

3. Salir del intérprete.
4. Crear el archivo `main.js` con el siguiente contenido:

```
//imports  
const calcul = require("./calcul");  
  
//main  
if (require.main === module) {  
  console.log("Principal!");  
  console.log(calcul.sum(1, 3, 5, 7, 9));  
} else {  
  console.log("Secundario!");  
}
```

5. Invocar `node` en modo *batch* pasándole `main.js` como punto de entrada o módulo principal:

```
$ node main.js  
Principal!  
25  
$
```

Cuando le indicamos a `node` un módulo, éste se ejecuta como principal. Si siempre va a invocarse como principal, no hace falta incluir el código principal dentro de una sentencia `if`, tal como hemos hecho en el ejemplo. Lo hemos hecho para mostrar que se ejecuta una parte y otra

no lo hace.

6. Abrir **node** en modo interactivo:

```
$ node  
>
```

7. Importar el módulo main:

```
> var principal = require("./main")  
Secundario!  
undefined  
>
```