Module System – ¿Deberíamos usar index.js?

Este es un tema controvertido entre los desarrolladores de Node.

- A algunos les encanta por su capacidad para simplificar las importaciones y facilitar el uso de módulos complicados con miles de archivos.
- Otros desarrolladores sugieren que siempre se utilice la ruta directa al módulo, sin depender de ningún comportamiento especial en Node.
- Oros prefieren no usar index.js, para la mayoría de los proyectos. Es bueno entenderlo, en caso de encontrar un proyecto que lo use, pero podría añadir algo de confusión.

Si miramos la documentación de Node sobre como la función require encuentra el módulo solicitado, en realidad es bastante complejo.

```
1. If X is a core module,
   a. return the core module
   b. STOP
2. If X begins with '/'
   a. set Y to be the filesystem root
```

```
3. If X begins with './' or '/' or '../'
   a. LOAD_AS_FILE(Y + X)
   b. LOAD_AS_DIRECTORY(Y + X)
   c. THROW "not found"
4. If X begins with '#'
   a. LOAD_PACKAGE_IMPORTS(X, dirname(Y))
```

Ya hemos visto que hay un comportamiento diferente para los módulos incorporados en los que simplemente pasas el nombre del módulo frente a los módulos que hemos escrito nosotros mismos, e index.js se suma a esta lógica, podemos ver que hay una sección para cargar el index.js:

```
LOAD_INDEX(X)
1. If X/index.js is a file, load X/index.js as JavaScript text. STOP
2. If X/index.json is a file, parse X/index.json to a JavaScript object. STOP
3. If X/index.node is a file, load X/index.node as binary addon. STOP
```

Se añade otro caso especial a esta lógica ya bastante larga.

Puede que no parezca gran cosa, pero en realidad es uno de 10 arrepentimientos de Ryan Doll, creador de Node. En sus propias palabras, complicó innecesariamente el sistema de carga de módulos. En retrospectiva, una de esas cosas fue index.js y la lógica compleja que implica cargarlo.

La buena noticia es que no tenemos que usar index.js, podemos mantener las cosas muy simples si queremos.