NODE.JS

rortegag

Sinopsis

Un ejemplo de un servidor web escrito con Node que responde con 'Hola Mundo'.
Para ejecutar el servidor, pon el código en un archivo llamado example.js y ejecútalo con el programa node.

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

http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(3000);

console.log('Server running at http://127.0.0.1:3000/');
```

Console

Imprime en stdout con nueva línea.
`console.log([data], [...]);`
Igual que console.log.
`console.info([data], [...]);`
Igual que console.log pero imprime en stderr.
`console.error([data], [...]);`
Igual que console.error.
`console.warn([data], [...]);`
Utiliza util.inspect en obj e imprime la cadena resultante en stdout.
`console.dir(obj);`
Marca la hora.
`console.time(label);`
Finaliza el temporizador, graba la salida.
`console.timeEnd(label);`
Imprime un seguimiento de pila en stderr de la posición actual.
`console.trace(label);`
Igual que assert.ok() donde si la expresión se evalúa como falsa lanza un AssertionError con mensaje.
`console.assert(expression, [message]);`

Objetos Globales

En los navegadores, el ámbito de nivel superior es el ámbito global.
Eso significa que en los navegadores si estás en el ámbito global var algo definirá una variable global.
En Node esto es diferente.
El ámbito de nivel superior no es el ámbito global; var algo dentro de un módulo Node será local a ese módulo.

El nombre del archivo del código que se está ejecutando. (ruta absoluta)
`__filename;`
El nombre del directorio en el que reside el script que se está ejecutando actualmente.
(ruta absoluta)
`__dirname;`
Una referencia al módulo actual. En particular module.exports se utiliza para definir lo que un módulo exporta y pone a disposición a través de require().
`module;`
Una referencia al módulo.exports que es más corto que el tipo.
`exports;`
El objeto proceso es un objeto global y se puede acceder a él desde cualquier lugar. Es una instancia de EventEmitter.
`process;`
La clase Buffer es un tipo global para tratar directamente con datos binarios.
`Buffer;`

Temporizadores

Para programar la ejecución de una llamada de retorno única tras un retardo de milisegundos.
Opcionalmente, también puede pasar argumentos a la llamada de retorno.
`setTimeout(callback, delay, [arg], [...]);`
Detiene un temporizador creado previamente con setTimeout().
`clearTimeout(t);`
Para programar la ejecución repetida del callback cada milisegundos de retardo. Opcionalmente también puede pasar argumentos a la devolución de llamada.
`setInterval(callback, delay, [arg], [...]);`
Detiene un temporizador creado previamente con setInterval().
`clearInterval(t);`
Detiene un temporizador creado previamente con setImmediate().
`clearImmediate(immediateObject);`
Permite crear un temporizador que esté activo pero si es el único elemento que queda en el bucle de eventos, el nodo no mantendrá el programa en ejecución.
`unref();`
Si previamente ha desreferenciado un temporizador, puede llamar a ref() para solicitar explícitamente que el temporizador mantenga abierto el programa.
`ref();`

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

Modules

Carga el módulo module.js en el mismo directorio.
```
var module = require('./module.js');
```
Cargar otro_modulo.js como si require() fuera llamado desde el propio módulo.
```
module.require('./otro_modulo.js');
```

El identificador del módulo. Normalmente es el nombre de archivo completamente resuelto.
```
module.id;
```
El nombre de archivo completamente resuelto del módulo.
```
module.filename;
```
Si el módulo ha terminado de cargarse o está en proceso de carga.
```
module.loaded;
```
El módulo que requería éste.
```
module.parent;
```
Los objetos del módulo requeridos por éste.
```
module.children;
```

```
exports.area = function (r) {
  return Math.PI * r * r;
};
```
Si quieres que la raíz de la exportación de tu módulo sea una función (como un constructor) o si quieres exportar un objeto completo en una asignación en lugar de construirlo propiedad a propiedad, asígnalo a module.exports en lugar de a exports.
```
module.exports = function(width) {
  return {
    area: function() {
      return width * width;
    }
  };
}
```

Child Process

Node provee una facilidad popen tri-direccional a través del módulo child_process.
Es posible transmitir datos a través de stdin, stdout y stderr de un hijo de forma totalmente no bloqueante.

Clase. ChildProcess es un EventEmitter.
```
ChildProcess;
```
Un flujo escribible que representa la entrada estándar del proceso hijo.
```
child.stdin;
```
Un flujo legible que representa la salida estándar del proceso hijo.
```
child.stdout;
```
Un flujo legible que representa el stderr del proceso hijo.
```
child.stderr;
```
El PID del proceso hijo.
```
child.pid;
```

MÁS ↓

Process

Emitido cuando el proceso está a punto de salir.
```
process.on('exit', function(code) {});
```
Emitido cuando una excepción burbujea hasta el bucle de eventos. (no debe utilizarse)
```
process.on('uncaughtException', function(err) {});
```
Un stream escribible a stdout.
```
process.stdout;
```
Un stream escribible a stderr.
```
process.stderr;
```
Un stream leíble para stdin.
```
process.stdin;
```

Un array que contiene los argumentos de la línea de comandos.
```
process.argv;
```
Objeto que contiene el entorno de usuario.
```
process.env;
```
Es la ruta absoluta del ejecutable que inició el proceso.
```
process.execPath;
```
Es el conjunto de opciones de línea de comandos específicas del nodo del ejecutable que inició el proceso.
```
process.execArgv;
```
En qué arquitectura de procesador se está ejecutando: 'arm', 'ia32', o 'x64'.
```
process.arch;
```
Un objeto que contiene la representación JavaScript de las opciones de configuración que se utilizaron para compilar el ejecutable del nodo actual.
```
process.config;
```
El PID del proceso.
```
process.pid;
```
En qué plataforma estás corriendo: 'darwin', 'freebsd', 'linux', 'sunos' o 'win32'.
```
process.platform;
```
Getter/setter para establecer lo que se muestra en 'ps'.
```
process.title;
```
Una propiedad compilada que expone NODE_VERSION.
```
process.version;
```
Una propiedad que expone las cadenas de versión del nodo y sus dependencias.
```
process.versions;
```

Esto hace que node emita un abort. Esto hará que nodo para salir y generar un archivo de núcleo.
```
process.abort();
```
Cambia el directorio de trabajo actual del proceso o lanza una excepción si falla.
```
process.chdir(dir);
```

MÁS ↓

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

Child Process

Si .connected es falso, ya no es posible enviar mensajes.
child.connected;
Enviar una señal al proceso hijo.
child.kill([signal]);
Cuando se utiliza child_process.fork() se puede escribir al child utilizando child.send(message, [sendHandle]) y los mensajes se reciben mediante un evento 'message' en el child.
child.send(message, [sendHandle]);
Cierra el canal IPC entre el padre y el hijo, permitiendo que el hijo salga con gracia una vez que no hay otras conexiones que lo mantengan vivo.
child.disconnect();
Inicia un nuevo proceso con el comando dado, con los argumentos de la línea de comandos en args. Si se omite, args es por defecto un array vacía.
child_process.spawn(command, [args], [options]);
Ejecuta un comando en un intérprete de comandos y almacena la salida.
child_process.exec(command, [options], callback);
Ejecuta un comando en un intérprete de comandos y almacena la salida.
child_process.execFile(file, [args], [options], [callback]);
Este es un caso especial de la funcionalidad spawn() para generar procesos Node. Además de tener todos los métodos de una instancia ChildProcess normal, el objeto devuelto tiene un canal de comunicación incorporado.
child_process.fork(modulePath, [args], [options]);

Util

Devuelve una cadena formateada utilizando el primer argumento como formato tipo printf (%s, %d, %j)
util.format(format, [...]);
Una función de salida sincrónica. Bloqueará el proceso y la cadena de salida inmediatamente a stderr.
util.debug(string);
Igual que util.debug() excepto que esta opción mostrará todos los argumentos inmediatamente en stderr.
util.error([...]);
Una función de salida sincrónica. Bloqueará el proceso y enviará todos los argumentos a stdout con nuevas líneas después de cada argumento.
util.puts([...]);
Una función de salida sincrónica. Bloqueará el proceso, convertirá cada argumento en una cadena y lo enviará a stdout. (sin nuevas líneas)
util.print([...]);
Salida con fecha y hora en stdout.
util.log(string);
Devuelve una representación de cadena del objeto, útil para depuración. (opciones: showHidden, depth, colors, customInspect)
util.inspect(object, [opts]);

MÁS

Process

Devuelve el directorio actual del proceso.
process.cwd();
Finaliza el proceso con el código especificado. Si se omite, exit utiliza el código de 'success' 0.
process.exit([code]);
Obtiene la identidad de grupo del proceso.
process.getgid();
Establece la identidad de grupo del proceso.
process.setgid(id);
Obtiene la identidad del usuario del proceso.
process.getuid();
Devuelve un array con los IDs de los grupos suplementarios.
process.getgroups();
Establece los ID de grupo suplementarios.
process.setgroups(grps);

Lee /etc/group e inicializa la lista de acceso a grupos, utilizando todos los grupos de los que el usuario es miembro.
process.initgroups(user, extra_grp);
Envía una señal a un proceso. pid es el id del proceso y signal es la cadena que describe la señal a enviar.
process.kill(pid, [signal]);
Devuelve un objeto que describe el uso de memoria del proceso Nodo medido en bytes.
process.memoryUsage();
En el siguiente bucle alrededor del bucle de eventos llama a este callback.
process.nextTick(callback);
Las callbacks pasadas a process.nextTick normalmente serán llamadas al final del flujo actual de ejecución, y por lo tanto son aproximadamente tan rápidas como llamar a una función de forma sincrónica.
process.maxTickDepth;
Establece o lee la máscara de creación de modo de archivo del proceso.
process.umask([mask]);
Número de segundos que ha estado funcionando el Nodo.
process.uptime();
Devuelve el tiempo real actual de alta resolución en un array de tupla [segundos, nanosegundos].
process.hrtime();

Devuelve true si el "objet" dado es un Array. false en caso contrario.
util.isArray(object);
Devuelve true si el "objet" dado es un RegExp. false en caso contrario.
util.isRegExp(object);
Devuelve true si el "objet" dado es un Date. false en caso contrario.
util.isDate(object);

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

Util

Devuelve true si el "objet" dado es un Error. false en caso contrario.
util.isError(object);
Toma una función cuyo último argumento es una devolución de llamada y devuelve una versión que devuelve promesas.
util.promisify(fn);
Hereda los métodos prototipo de un constructor en otro.
util.inherits(constructor, superConstructor);

Eventos

Todos los objetos que emiten eventos son instancias de events.EventEmitter. Puedes acceder a este módulo haciendo: require("events");
Para acceder a la clase EventEmitter, require('events').EventEmitter.
Todos los EventEmitters emiten el evento 'newListener' cuando se añaden nuevos listener y 'removeListener' cuando se elimina un listener.

Añade un oyente al final del array del listener para el evento especificado.
emitter.addListener(event, listener);
Igual que emitter.addListener().
emitter.on(event, listener);
Añade un receptor único para el evento. Este receptor sólo se invocará la próxima vez que se active el evento, tras lo cual se eliminará.
emitter.once(event, listener);
Elimina un oyente del array del listener para el evento especificado.
emitter.removeListener(event, listener);
Elimina todos los listener, o los del evento especificado.
emitter.removeAllListeners([event]);
Por defecto EventEmitters imprimirá una advertencia si se añaden más de 10 listener para un evento en particular.
emitter.setMaxListeners(n);
Devuelve un array de listeners para el evento especificado.
emitter.listeners(event);
Ejecuta cada uno de los listener en orden con los argumentos suministrados. Devuelve true si el evento tiene listener, false en caso contrario.
emitter.emit(event, [arg1], [arg2], [...]);
Devuelve el número de listener de un evento determinado.
EventEmitter.listenerCount(emitter, event);

Stream

Un stream es una interfaz abstracta implementada por varios objetos en Node. Por ejemplo, una solicitud a un servidor HTTP es un stream, al igual que stdout.
Los streams son legibles, escribibles o ambos. Todos los streams son instancias de EventEmitter.

La interfaz stream Readable es la abstracción para una fuente de datos de la que se está leyendo.
En otras palabras, los datos salen de un stream legible.
Un stream legible no empezará a emitir datos hasta que usted indique que está listo para recibirlos.
Ejemplos de streams legibles son: respuestas http en el cliente, peticiones http en el servidor, stream de lectura fs, zlib streams, crypto streams, tcp sockets, child process stdout y stderr, process.stdin.

MÁS

```
var readable = getReadableStreamSomehow();
```

Cuando se pueda leer un trozo de datos del stream, éste emitirá un evento 'readable'.
readable.on('readable', function() {});
Si adjuntas un listener de eventos de datos, entonces cambiará el stream a modo de flujo, y los datos se pasarán a tu manejador tan pronto como estén disponibles.
readable.on('data', function(chunk) {});

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

Stream

Este evento se dispara cuando no hay más datos para leer.
```
readable.on('end', function() {});
```
Se emite cuando el recurso subyacente (por ejemplo, el descriptor de archivo de respaldo) se ha cerrado. No todos los streams lo emiten.
```
readable.on('close', function() {});
```
Hereda los métodos prototipo de un constructor en otro.
```
readable.on('error', function() {});
```

El método read() extrae algunos datos del buffer interno y los devuelve. Si no hay datos disponibles, entonces devolverá null.
Este método sólo debe invocarse en modo no fluido. En el modo de flujo, este método se llama automáticamente hasta que se vacía el búfer interno.
```
readable.read([size]);
```

Llama a esta función para hacer que el stream devuelva cadenas de la codificación especificada en lugar de objetos Buffer.
```
readable.setEncoding(encoding);
```
Este método hará que el stream legible reanude la emisión de eventos de datos.
```
readable.resume();
```
Este método hará que un stream en modo flujo deje de emitir eventos de datos.
```
readable.pause();
```
Este método extrae todos los datos de un stream legible y los escribe en el destino suministrado, gestionando automáticamente el flujo para que el destino no se vea desbordado por un stream de lectura rápida.
```
readable.pipe(destination, [options]);
```
Este método eliminará los hooks establecidos para una llamada anterior a pipe(). Si no se especifica el destino, se eliminarán todas las tuberías.
```
readable.unpipe([destination]);
```
Esto es útil en ciertos casos en los que un stream está siendo consumido por un analizador sintáctico, que necesita "des-consumir" algunos datos que ha extraído de forma optimista de la fuente, para que el stream pueda ser pasado a alguna otra parte.
```
readable.unshift(chunk);
```

La interfaz Writable stream es una abstracción para un destino en el que se escriben datos.
Ejemplos de streams de escritura son: peticiones http en el cliente, respuestas http en el servidor, flujos de escritura fs, zlib streams, crypto streams, tcp sockets, child process stdin, process.stdout, process.stderr.

```
var writer = getWritableStreamSomehow();
```

Este método escribe algunos datos en el sistema subyacente, y llama al callback suministrada una vez que los datos han sido completamente manejados.
```
writable.write(chunk, [encoding], [callback]);
```
Si una llamada a writable.write(chunk) devuelve false, el evento drain indicará cuándo es apropiado empezar a escribir más datos en el stream.
```
writer.once('drain', write);
```

Llama a este método cuando no se van a escribir más datos en el stream.
```
writable.end([chunk], [encoding], [callback]);
```
Cuando el método end() ha sido llamado, y todos los datos han sido vaciados al sistema subyacente, este evento es emitido.
```
writer.on('finish', function() {});
```
Se emite cada vez que se llama al método pipe() en un stream legible, añadiendo este escribible a su conjunto de destinos.
```
writer.on('pipe', function(src) {});
```
Se emite cada vez que se llama al método unpipe() en un stream legible, eliminando este escribible de su conjunto de destinos.
```
writer.on('unpipe', function(src) {});
```

MÁS ↓

NODE.JS

rortegag

Stream

Se emite si se ha producido un error al escribir o transferir datos.
writer.on('error', function(src) {});
Los streams dúplex son streams que implementan tanto la interfaz de lectura como la de escritura.
Ejemplos de streams dúplex son: sockets tcp, zlib streams, crypto streams.

Los streams de transformación son streams dúplex en los que la salida se calcula de algún modo a partir de la entrada. Implementan las interfaces Readable y Writable. Para más información sobre su uso, véase más arriba.
Algunos ejemplos de stream de transformación son: zlib streams, crypto streams.

Sistema de archivos

Para usar este módulo haga require('fs').
Todos los métodos tienen formas asíncronas y síncronas.

Renombrar asíncrono. No se dan más argumentos que una posible excepción al callback de callback.Asynchronous ftruncate. El callback no recibe más argumentos que una posible excepción.
fs.rename(oldPath, newPath, callback);
Renombrado síncrono.
fs.renameSync(oldPath, newPath);

ftruncate asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.ftruncate(fd, len, callback);
ftruncate síncrono.
fs.ftruncateSync(fd, len);
truncate asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.truncate(path, len, callback);
truncate síncrono.
fs.truncateSync(path, len);

chown asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.truncateSync(path, len);
chown síncrono.
fs.chownSync(path, uid, gid);
fchown asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.fchown(fd, uid, gid, callback);
fchown síncrono.
fs.fchownSync(fd, uid, gid);
lchown asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.lchown(path, uid, gid, callback);
lchown síncrono.
fs.lchownSync(path, uid, gid);

chmod asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.chmod(path, mode, callback);
chmod síncrono.
fs.chmodSync(path, mode);
fchmod asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.fchmod(path, mode, callback);
fchmod síncrono.
fs.fchmodSync(path, mode);
lchmod asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.lchmod(path, mode, callback);
lchmod síncrono.
fs.lchmodSync(path, mode);

MÁS

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

Sistema de archivos

stat asíncrono. El callback recibe dos argumentos (err, stats) donde stats es un objeto fs.Stats.
fs.stat(path, callback);
stat síncrono. Devuelve una instancia de fs.Stats.
fs.statSync(path);
lstat asíncrono. El callback recibe dos argumentos (err, stats) donde stats es un objeto fs.Stats.
lstat() es idéntica a stat(), excepto que si path es un enlace simbólico, entonces el enlace en sí es categorizado, no el fichero al que se refiere.
fs.lstat(path, callback);
lstat síncrono. Devuelve una instancia de fs.Stats.
fs.lstatSync(path);
fstat asíncrono. El callback recibe dos argumentos (err, stats) donde stats es un objeto fs.Stats. fstat() es idéntica a stat(), excepto que el fichero al que se le va a aplicar la estadística se especifica mediante el descriptor de fichero fd.
fs.fstat(fd, callback);
fstat síncrono. Devuelve una instancia de fs.Stats.
fs.fstatSync(fd);

link asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.link(srcpath, dstpath, callback);
link síncrono.
fs.linkSync(srcpath, dstpath);
symlink asíncrono. El callback no recibe más argumentos que una posible excepción. El argumento type puede ser 'dir', 'file', o 'junction' (por defecto es 'file') y sólo está disponible en Windows (ignorado en otras plataformas).
fs.symlink(srcpath, dstpath, [type], callback);
symlink síncrono.
fs.symlinkSync(srcpath, dstpath, [type]);
readlink asíncrono. El callback recibe dos argumentos (err, linkString).
fs.readlink(path, callback);
readlink síncrono. Devuelve el valor de la cadena del enlace simbólico.
fs.readlinkSync(path);
unlink asíncrona. No se dan más argumentos que una posible excepción al callback de finalización.
fs.unlink(path, callback);
unlink síncrono.
fs.unlinkSync(path);

realpath asíncrono. El callback recibe dos argumentos (err, resolvedPath).
fs.realpath(path, [cache], callback);
realpath síncrono. Devuelve la ruta resuelta.
fs.realpathSync(path, [cache]);

rmdir asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.rmdir(path, callback);
rmdir síncrono.
fs.rmdirSync(path);
mkdir asíncrono. El callback no recibe más argumentos que una posible excepción. mode por defecto es 0777.
fs.mkdir(path, [mode], callback);
mkdir síncrono.
fs.mkdirSync(path, [mode]);
readdir asíncrono. Lee el contenido de un directorio. El callback recibe dos argumentos (err, files) donde files es un array de los nombres de los ficheros del directorio excluyendo '.' y '..'.
fs.readdir(path, callback);
readdir síncrono. Devuelve un array de nombres de archivo excluyendo '.' y '..'.
fs.readdirSync(path);

MÁS

NODE.JS

rortegag

Sistema de archivos

close asíncrono. No se dan más argumentos que una posible excepción a la llamada de callback.
fs.close(fd, callback);
close síncrono.
fs.closeSync(fd);
open asíncrono. No se dan más argumentos que una posible excepción a la llamada de callback.
fs.open(path, flags, [mode], callback);
Versión síncrona de fs.open().
fs.openSync(path, flags, [mode]);
Cambia las timestamps del archivo al que hace referencia la ruta suministrada.
fs.utimes(path, atime, mtime, callback);
Versión síncrona de fs.utimes().
fs.utimesSync(path, atime, mtime);
Cambia las timestamps de un archivo referenciado por el descriptor de archivo suministrado.
fs.futimes(fd, atime, mtime, callback);
Versión síncrona de fs.futimes().
fs.futimesSync(fd, atime, mtime);
fsync asíncrono. No se dan más argumentos que una posible excepción al callback de finalización.
fs.fsync(fd, callback);
fsync síncrono.
fs.fsyncSync(fd);

Escribir búfer en el archivo especificado por fd.
fs.write(fd, buffer, offset, length, position, callback);
Versión sincrónica de fs.write(). Devuelve el número de bytes escritos.
fs.writeSync(fd, buffer, offset, length, position);
Leer datos del archivo especificado por fd.
fs.read(fd, buffer, offset, length, position, callback);
Versión sincrónica de fs.read(). Devuelve el número de bytes leídos.
fs.readSync(fd, buffer, offset, length, position);
Lee de forma asíncrona todo el contenido de un fichero.
fs.readFile(filename, [options], callback);
Versión sincrónica de fs.readFile(). Devuelve el contenido del nombre de archivo. Si se especifica la opción de codificación, esta función devuelve una cadena. En caso contrario, devuelve un búfer.
fs.readFileSync(filename, [options]);

Escribe datos de forma asíncrona en un archivo, sustituyendo el archivo si ya existe. Los datos pueden ser una cadena o un búfer.
fs.writeFile(filename, data, [options], callback);
La versión síncrona de fs.writeFile().
fs.writeFileSync(filename, data, [options]);
Añade datos a un archivo de forma asíncrona, creando el archivo si aún no existe. Los datos pueden ser una cadena o un búfer.
fs.appendFile(filename, data, [options], callback);
La versión síncrona de fs.appendFile().
fs.appendFileSync(filename, data, [options]);
Busca cambios en el nombre de fichero, donde nombre de fichero puede ser un fichero o un directorio.
El objeto devuelto es un fs.FSWatcher. El evento puede ser 'rename' o 'change', y filename es el nombre del fichero que ha disparado el evento.
fs.watch(filename, [options], [listener]);
Comprueba si la ruta dada existe o no consultando con el sistema de ficheros. Luego llama al argumento callback con true o false. (no debe utilizarse)
fs.exists(path, callback);
Versión síncrono de fs.exists(). (no debe utilizarse)
fs.existsSync(path);

MÁS

NODE.JS

rortegag

Sistema de archivos

fs.Stats: los objetos devueltos por fs.stat(), fs.lstat() y fs.fstat() y sus homólogos síncronos son de este tipo.
stats.isFile();
stats.isDirectory();
stats.isBlockDevice();
stats.isCharacterDevice();
stats.isSymbolicLink(); (sólo es válido con fs.lstat())
stats.isFIFO();
stats.isSocket();

Devuelve un nuevo objeto ReadStream.
fs.createReadStream(path, [options]);
Devuelve un nuevo objeto WriteStream.
fs.createWriteStream(path, [options]);

HTTP

Para utilizar el servidor y el cliente HTTP es necesario require('http').

Una colección de todos los códigos de estado de respuesta HTTP estándar, y la breve descripción de cada uno.
http.STATUS_CODES;
Esta función permite emitir solicitudes de forma transparente.
http.request(options, [callback]);
Establece el método a GET y llama a req.end() automáticamente.
http.get(options, [callback]);

Devuelve un nuevo objeto servidor web.
El requestListener es una función que se añade automáticamente al evento 'request'.
server = http.createServer([requestListener]);
Comienza a aceptar conexiones en el puerto y nombre de host especificados.
server.listen(port, [hostname], [backlog], [callback]);
Inicia un servidor de sockets UNIX a la escucha de conexiones en la ruta indicada.
server.listen(path, [callback]);
El objeto handle puede ser un servidor o un socket (cualquier cosa con un miembro _handle subyacente), o un objeto {fd: <n>}.
server.listen(handle, [callback]);
Impide que el servidor acepte nuevas conexiones.
server.close([callback]);
Establece el valor de timeout para los sockets, y emite un evento 'timeout' en el objeto Servidor, pasando el socket como argumento, si se produce un timeout.
server.setTimeout(msecs, callback);

MÁS

Path

Use require('path') para usar este módulo.
Este módulo contiene utilidades para manejar y transformar rutas de archivos.
Casi todos estos métodos sólo realizan transformaciones de cadenas.
No se consulta el sistema de archivos para comprobar si las rutas son válidas.

Normaliza una ruta de cadena, teniendo en cuenta las partes '..' y '.'.
path.normalize(p);
Une todos los argumentos y normaliza la ruta resultante.
path.join([path1], [path2], [...]);
Resuelve 'to' en una ruta absoluta.
path.resolve([from ...], to);
Resuelve la ruta relativa desde 'from' a 'to'.
path.relative(from, to);
Devuelve el nombre del directorio de una ruta. Similar al comando dirname de Unix.
path.dirname(p);
Devuelve la última parte de una ruta. Similar al comando basename de Unix.
path.basename(p, [ext]);
Devuelve la extensión de la ruta, desde el último '.' hasta el final de la cadena en la última parte de la ruta.
path.extname(p);

El separador de archivos específico de la plataforma. '\\' o '/'.
path.sep;
El delimitador de ruta específico de la plataforma, ';' o ':'.
path.delimiter;

URL

Este módulo contiene utilidades para la resolución y el análisis de URL. Llame a require('url') para usarlo.

Toma una cadena URL y devuelve un objeto.
url.parse(urlStr, [parseQueryString], [slashesDenoteHost]);
Toma un objeto URL analizado y devuelve una cadena URL formateada.
url.format(urlObj);
Tome una URL base y una URL href y resuélvalas como lo haría un navegador para una etiqueta de anclaje.
url.resolve(from, to);

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

HTTP

Limita el número máximo de cabeceras entrantes, igual a 1000 por defecto. Si se establece en 0, no se aplicará ningún límite.
server.maxHeadersCount;
El número de milisegundos de inactividad antes de que se suponga que un socket ha expirado.
server.timeout;

Se emite cada vez que hay una solicitud.
server.on('request', function (request, response) { });
Cuando se establece un nuevo stream TCP.
server.on('connection', function (socket) { });
Se emite cuando se cierra el servidor.
server.on('close', function () { });
Se emite cada vez que se recibe una petición con un http Expect: 100-continue.
server.on('checkContinue', function (request, response) { });
Se emite cada vez que un cliente solicita un método http CONNECT.
server.on('connect', function (request, socket, head) { });
Emitido cada vez que un cliente solicita una actualización http.
server.on('upgrade', function (request, socket, head) { });
Si una conexión cliente emite un evento 'error' - se reenviará aquí.
server.on('clientError', function (exception, socket) { });

Envía una parte del cuerpo.
request.write(chunk, [encoding]);
Finaliza el envío de la petición. Si alguna parte del cuerpo no se ha enviado, se vaciará en el stream.
request.end([data], [encoding]);
Cancela una solicitud.
request.abort();
Una vez que se asigna un socket a esta petición y se conecta se llamará a socket.setTimeout().
request.setTimeout(timeout, [callback]);
Una vez que se asigna un socket a esta petición y se conecta se llamará a socket.setNoDelay().
request.setNoDelay([noDelay]);
Una vez que se asigna un socket a esta petición y se conecta se llamará a socket.setKeepAlive().
request.setSocketKeepAlive([enable], [initialDelay]);

Se emite cuando se recibe una respuesta a esta solicitud. Este evento sólo se emite una vez.
request.on('response', function(response) { });
Se emite después de asignar un socket a esta petición.
request.on('socket', function(socket) { });

MÁS

Query String

Este módulo proporciona utilidades para tratar con cadenas de consulta. Llama a require('querystring') para usarlo.

Serializa un objeto a una cadena de consulta. Anula opcionalmente los caracteres separadores ('&') y de asignación ('=') predeterminados.
querystring.stringify(obj, [sep], [eq]);
Deserializa una cadena de consulta a un objeto. Anula opcionalmente los caracteres separadores ('&') y de asignación ('=') predeterminados.
querystring.parse(str, [sep], [eq], [options]);

Assert

Este módulo se utiliza para escribir pruebas unitarias para sus aplicaciones, puede acceder a él con require('assert').

Lanza una excepción que muestra los valores de real y esperado separados por el operador proporcionado.
assert.fail(actual, expected, message, operator);
Prueba si el valor es verdadero, es equivalente a assert.equal(true, !!value, message);
assert(value, message);
assert.ok(value, [message]);
Prueba la igualdad superficial y coercitiva con el operador de comparación igual (==).
assert.equal(actual, expected, [message]);
Prueba la no igualdad superficial y coercitiva con el operador de comparación no igual (!=).
assert.notEqual(actual, expected, [message]);
Pruebas de igualdad en profundidad.
assert.deepEqual(actual, expected, [message]);
Pruebas de cualquier desigualdad profunda.
assert.notDeepEqual(actual, expected, [message]);
Prueba la igualdad estricta, determinada por el operador de igualdad estricta (===).
assert.strictEqual(actual, expected, [message]);
Comprueba la no igualdad estricta, determinada por el operador no igual estricto (!==).
assert.notStrictEqual(actual, expected, [message]);
El error puede ser un constructor, una RegExp o una función de validación.
assert.throws(block, [error], [message]);
Espera que el bloque no lance un error, ver assert.throws para más detalles.
assert.doesNotThrow(block, [message]);
Prueba si el valor no es un valor falso, lanza si es un valor verdadero. Útil cuando se comprueba el primer argumento, error en devoluciones de llamada.
assert.ifError(value);

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

HTTP

Emitido cada vez que un servidor responde a una petición con un método CONNECT. Si no se está escuchando este evento, los clientes que reciban un método CONNECT verán cerradas sus conexiones.
request.on('connect', function(response, socket, head) { });
Emitido cada vez que un servidor responde a una petición con una actualización. Si no se escucha este evento, se cerrarán las conexiones de los clientes que reciban una cabecera de actualización.
request.on('upgrade', function(response, socket, head) { });
Se emite cuando el servidor envía una respuesta HTTP '100 Continue', normalmente porque la petición contenía 'Expect: 100-continue'. Se trata de una instrucción para que el cliente envíe el cuerpo de la solicitud.
request.on('continue', function() { });

Envía un fragmento del cuerpo de la respuesta. Si se llama a este método y no se ha llamado a response.writeHead(), cambiará al modo de cabecera implícita y vaciará las cabeceras implícitas.
response.write(chunk, [encoding]);
Envía un mensaje HTTP/1.1 100 Continue al cliente, indicando que el cuerpo de la petición debe ser enviado.
response.writeContinue();
Envía una cabecera de respuesta a la solicitud.
response.writeHead(statusCode, [reasonPhrase], [headers]);
Establece el valor del tiempo de espera del socket en msecs. Si se proporciona una callback, entonces se añade como oyente en el evento 'timeout' en el objeto de respuesta.
response.setTimeout(msecs, callback);
Establece un único valor de cabecera para las cabeceras implícitas. Si esta cabecera ya existe en las cabeceras a enviar, su valor será reemplazado. Utilice un array de cadenas si necesita enviar varias cabeceras con el mismo nombre.
response.setHeader(name, value);
Lee una cabecera que ya ha sido puesta en cola pero no enviada al cliente. Tenga en cuenta que el nombre no distingue entre mayúsculas y minúsculas.
response.getHeader(name);
Elimina una cabecera que está en cola para su envío implícito.
response.removeHeader(name);
Este método añade cabeceras HTTP trailing (una cabecera pero al final del mensaje) a la respuesta.
response.addTrailers(headers);
Este método indica al servidor que se han enviado todas las cabeceras y el cuerpo de la respuesta; el servidor debe considerar este mensaje completo. El método response.end() DEBE invocarse en cada respuesta.
response.end([data], [encoding]);

MÁS ↓

OS

Proporciona algunas funciones básicas relacionadas con el sistema operativo.
Utilice require('os') para acceder a este módulo.

Devuelve el directorio por defecto del sistema operativo para los archivos temporales.
os.tmpdir();
Devuelve el ancho de banda de la CPU. Los valores posibles son "BE" o "LE".
os.endianness();
Devuelve el nombre de host del sistema operativo.
os.hostname();
Devuelve el nombre del sistema operativo.
os.type();
Devuelve la plataforma del sistema operativo.
os.platform();
Devuelve la arquitectura de la CPU del sistema operativo.
os.arch();
Devuelve la versión del sistema operativo.
os.release();
Devuelve el tiempo de actividad del sistema en segundos.
os.uptime();
Devuelve una matriz que contiene las medias de carga de 1, 5 y 15 minutos.
os.loadavg();
Devuelve la cantidad total de memoria del sistema en bytes.
os.totalmem();
Devuelve la cantidad de memoria libre del sistema en bytes.
os.freemem();
Devuelve un array de objetos que contienen información sobre cada CPU/núcleo instalado: modelo, velocidad (en MHz) y tiempos (un objeto que contiene el número de milisegundos que la CPU/núcleo pasó en: user, nice, sys, idle e irq).
os.cpus();
Obtener una lista de interfaces de red.
os.networkInterfaces();
Una constante que define el marcador de fin de línea apropiado para el sistema operativo.
os.EOL;

Buffer

Buffer se utiliza para tratar datos binarios.
Buffer es similar a una matriz de enteros, pero corresponde a una asignación de memoria sin procesar fuera de la pila de V8.

MÁS ↓

Para más información: https://nodejs.org/es/docs/

NODE.JS

rortegag

HTTP

Cuando se utilizan cabeceras implícitas (no se llama a response.writeHead() explícitamente), esta propiedad controla el código de estado que se enviará al cliente cuando se vacíen las cabeceras.
response.statusCode;
Booleano (sólo lectura). true si se enviaron las cabeceras, false en caso contrario.
response.headersSent;
Cuando es true, la cabecera Date se generará automáticamente y se enviará en la respuesta si no está ya presente en las cabeceras. Por defecto es true.
response.sendDate;

Indica que la conexión subyacente fue terminada antes de que response.end() fuera llamado o capaz de flush.
response.on('close', function () { });
Se emite cuando se ha enviado la respuesta.
response.on('finish', function() { });

En caso de petición al servidor, la versión HTTP enviada por el cliente. En caso de respuesta del cliente, la versión HTTP del servidor al que se ha conectado.
message.httpVersion;
El objeto de cabeceras de solicitud/respuesta.
message.headers;
El objeto de seguimiento de la solicitud/respuesta. Sólo se rellena después del evento 'end'.
message.trailers;
El método de solicitud como cadena. Sólo lectura. Ejemplo: 'GET', 'DELETE'.
message.method;
Cadena de URL de la solicitud. Sólo contiene la URL presente en la solicitud HTTP real.
message.url;
El código de estado de respuesta HTTP de 3 dígitos. Ejemplo: 404.
message.statusCode;
El objeto net.Socket asociado a la conexión.
message.socket;

Llama a message.connection.setTimeout(msecs, callback).
message.setTimeout(msecs, callback);

Buffer

Asigna un nuevo búfer de tamaño octeto.
Buffer.from(size);
Asigna un nuevo búfer utilizando una matriz de octetos.
Buffer.from(array);
Asigna un nuevo búfer que contiene la cadena dada. La codificación por defecto es 'utf8'.
Buffer.from(str, [encoding]);

Devuelve true si la codificación es un argumento de codificación válido, o false en caso contrario.
Buffer.isEncoding(encoding);
Comprueba si obj es un Buffer.
Buffer.isBuffer(obj);
Devuelve un búfer que es el resultado de concatenar todos los búferes de la lista.
Buffer.concat(list, [totalLength]);
Indica la longitud en bytes de una cadena.
Buffer.byteLength(string, [encoding]);

Escribe una cadena en la memoria intermedia utilizando la codificación dada.
buf.write(string, [offset], [length], [encoding]);
Decodifica y devuelve una cadena a partir de los datos del búfer codificados con encoding (por defecto 'utf8') comenzando en start (por defecto 0) y terminando en end (por defecto buffer.length).
buf.write(string, [offset], [length], [encoding]);
Devuelve una representación JSON de la instancia del Buffer, que es idéntica a la salida de los arrays JSON.
buf.toJSON();
Realiza copias entre buffers. Las regiones de origen y destino pueden superponerse.
buf.copy(targetBuffer, [targetStart], [sourceStart], [sourceEnd]);
Devuelve un nuevo búfer que hace referencia a la misma memoria que el anterior, pero desplazado y recortado por los índices de inicio (por defecto 0) y fin (por defecto buffer.length).
Los índices negativos comienzan desde el final del búfer.
buf.slice([start], [end]);
Rellena el búfer con el valor especificado.
buf.fill(value, [offset], [end]);
Obtener y establecer el octeto en el índice.
buf[index];
El tamaño del búfer en bytes, Tenga en cuenta que esto no es necesariamente el tamaño de los contenidos.
buf.length;

Cuántos bytes se devolverán cuando se llame a buffer.inspect(). Esto puede ser anulado por módulos de usuario.
buffer.INSPECT_MAX_BYTES;

Para más información: https://nodejs.org/es/docs/