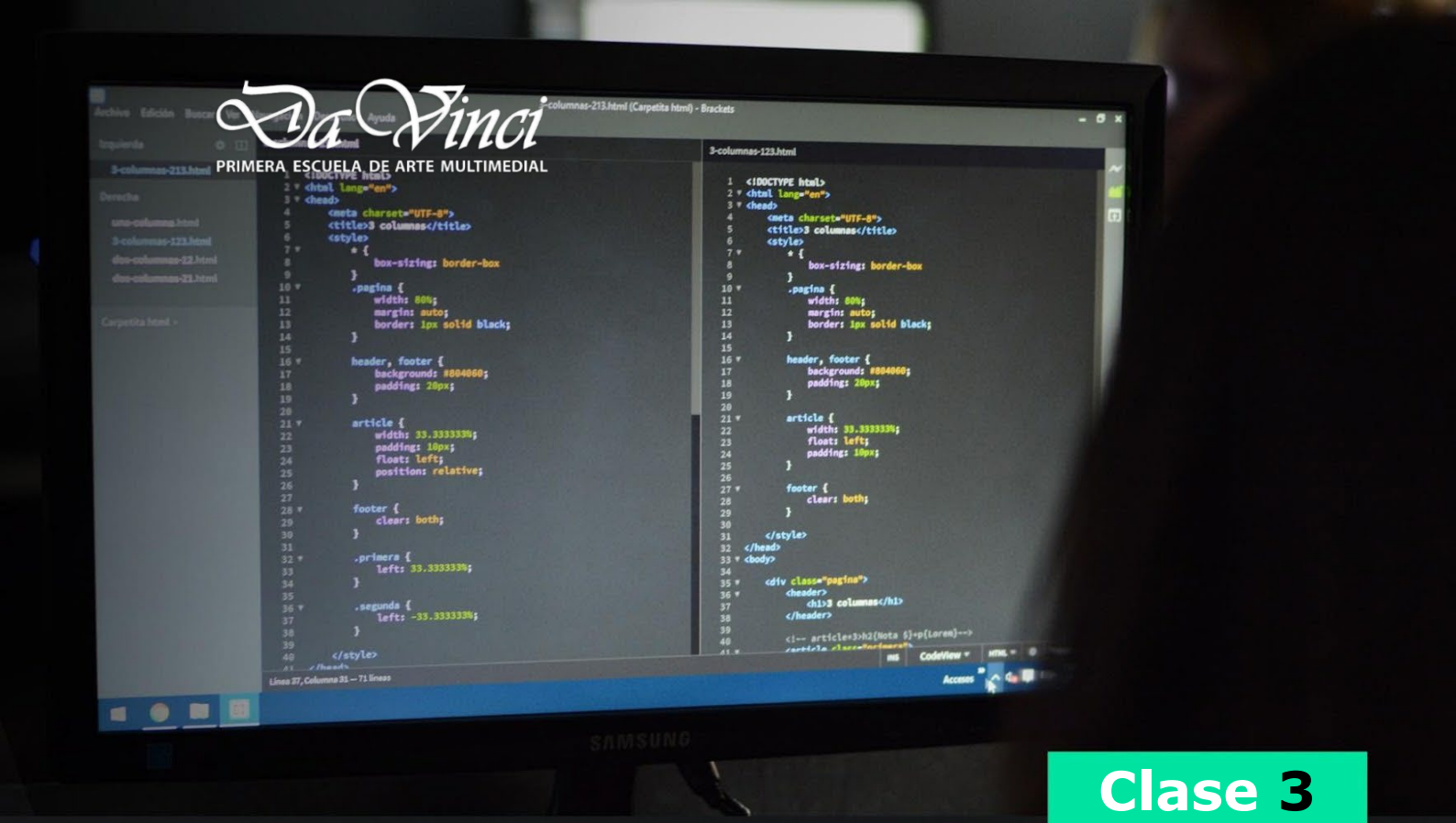

Clase 3

Diseño y Programación Web

Materia:
Aplicaciones Híbridas

Docente contenidista: MARCOS GALBÁN, Camila Belén

Módulos Nativos

Los módulos nativos en Node.js son componentes fundamentales del entorno de ejecución de Node.js que proporcionan funcionalidades básicas necesarias para construir aplicaciones. Estos módulos están escritos en C++ y JavaScript, y están integrados en el binario de Node.js.

Funcionamiento

Carga y Requerimiento:

Los módulos nativos se cargan utilizando la función require().

Encapsulamiento:

Cada módulo está encapsulado en su propio espacio de nombres para evitar conflictos con otras partes del programa. Esto se logra utilizando un patrón de módulos común en JavaScript.

Implementación en C++ y JavaScript:

Algunos módulos nativos están implementados en C++ para mejorar el rendimiento. Estos módulos C++ se exponen a JavaScript utilizando las APIs de Node.js y V8 (el motor de JavaScript de Chrome).

APIs Asíncronas y Síncronas:

Muchos módulos nativos proporcionan tanto APIs asíncronas como síncronas. Las APIs asíncronas utilizan devoluciones de llamada (callbacks), promesas (promises) o async/await para manejar operaciones que pueden tardar algún tiempo en completarse sin bloquear el hilo de ejecución principal.

Eventos y Flujos (Streams):

Los módulos como events y stream son fundamentales para el modelo de programación orientado a eventos de Node.js. Los flujos proporcionan una forma eficiente de manejar operaciones de entrada y salida (I/O).

Módulo OS

El módulo os en Node.js proporciona métodos para interactuar con el sistema operativo subyacente. Permite obtener información sobre el sistema operativo, la arquitectura de la CPU, el uso de memoria, y más. Es útil para obtener datos del entorno de ejecución y adaptar el comportamiento del programa según las características del sistema operativo.

Funcionamiento

Carga del módulo:

Se carga utilizando la función require():

```
const os = require('os');
```

Métodos Principales:

- os.arch(): Devuelve la arquitectura de la CPU.
- os.cpus(): Devuelve información sobre cada CPU/core del sistema.
- os.freemem(): Devuelve la cantidad de memoria libre del sistema.
- os.homedir(): Devuelve el directorio home del usuario.
- os.hostname(): Devuelve el nombre del host del sistema.
- os.platform(): Devuelve la plataforma del sistema operativo.
- os.release(): Devuelve la versión del sistema operativo.
- os.totalmem(): Devuelve la cantidad total de memoria del sistema.
- os.type(): Devuelve el nombre del sistema operativo.
- os.uptime(): Devuelve el tiempo de actividad del sistema en segundos.
- os.userInfo(): Devuelve información sobre el usuario actual.

Ejemplo:

Obtener información básica del sistema operativo

```
const os = require('os');

console.log('Arquitectura de la CPU:', os.arch());
console.log('Información de CPUs:', os.cpus());
console.log('Memoria libre:', os.freemem());
console.log('Directorio Home del usuario:', os.homedir());
console.log('Nombre del host:', os.hostname());
console.log('Plataforma del sistema operativo:', os.platform());
console.log('Versión del sistema operativo:', os.release());
console.log('Memoria total:', os.totalmem());
console.log('Nombre del sistema operativo:', os.type());
console.log('Tiempo de actividad del sistema (segundos):', os.uptime());
console.log('Información del usuario actual:', os.userInfo());
```

Uso en Aplicaciones Reales

El módulo os es especialmente útil en aplicaciones que necesitan adaptarse a diferentes entornos de ejecución. Por ejemplo, una aplicación puede cambiar su comportamiento si se está ejecutando en un servidor con poca memoria o en una plataforma específica (Windows, Linux, etc.).

Módulo path

El módulo path en Node.js proporciona utilidades para trabajar con rutas de archivos y directorios. Permite manipular rutas de archivos de manera independiente del sistema operativo, lo que ayuda a crear aplicaciones multiplataforma que funcionan correctamente tanto en Windows como en Unix/Linux.

Funcionamiento

Carga del módulo:

Se carga utilizando la función require():

```
const path = require('path');
```

Métodos Principales:

- path.basename(path, [ext]): Devuelve la última parte de una ruta.
- path.dirname(path): Devuelve el directorio de una ruta.
- path.extname(path): Devuelve la extensión de una ruta.
- path.isAbsolute(path): Determina si una ruta es absoluta.
- path.join([...paths]): Une todas las partes de una ruta utilizando el separador específico del sistema operativo.
- path.normalize(path): Normaliza una ruta, resolviendo .. y ..
- path.parse(path): Devuelve un objeto con las distintas partes de una ruta.
- path.relative(from, to): Calcula la ruta relativa entre dos rutas.
- path.resolve([...paths]): Resuelve una secuencia de rutas o segmentos de rutas en una ruta absoluta.
- path.sep: Proporciona el separador específico del sistema operativo ('/' en POSIX y '\\' en Windows).

Ejemplo

Obtener el nombre de archivo y la extensión

```
const path = require('path');

const filePath = '/user/local/bin/node';
console.log('Basename:', path.basename(filePath)); // node
console.log('Extname:', path.extname(filePath)); // ''
```

Uso en Aplicaciones Reales

El módulo path es especialmente útil para aplicaciones que manejan archivos y directorios, como herramientas de línea de comandos, servidores web, y sistemas de gestión de archivos. Permite manipular rutas de manera segura y consistente sin preocuparse por las diferencias entre sistemas operativos.

Módulo url

El módulo url en Node.js proporciona utilidades para el análisis y manipulación de URLs. Permite descomponer una URL en sus componentes y también construir URLs a partir de diferentes partes. Este módulo es muy útil para aplicaciones web y servicios que necesitan manejar URLs.

Funcionamiento

Carga del módulo:

Se carga utilizando la función require():

```
const url = require('url');
```

Principales funcionalidades:

- url.parse(urlString, [parseQueryString], [slashesDenoteHost]): Analiza una URL y devuelve un objeto con sus partes.
- url.format(urlObject): Ensambla una URL a partir de un objeto.
- url.resolve(from, to): Resuelve una URL relativa en una absoluta.

Ejemplo

Analizar una URL

```
const url = require('url');

const urlString =
'https://www.example.com:8080/path/name?query=string#hash';
const parsedUrl = url.parse(urlString);

console.log(parsedUrl);
// {
//   protocol: 'https:',
//   slashes: true,
//   auth: null,
//   host: 'www.example.com:8080',
//   port: '8080',
//   hostname: 'www.example.com',
//   hash: '#hash',
```

```
//   search: '?query=string',
//   query: 'query=string',
//   pathname: '/path/name',
//   path: '/path/name?query=string',
//   href: 'https://www.example.com:8080/path/name?query=string#hash'
// }
```

Módulo fs

El módulo fs (filesystem) en Node.js proporciona una API para interactuar con el sistema de archivos de manera tanto síncrona como asíncrona. Permite realizar operaciones como leer, escribir, abrir, cerrar, eliminar y renombrar archivos y directorios.

Funcionamiento

Carga del módulo:

Se carga utilizando la función require():

```
const fs = require('fs');
```

Métodos

El módulo fs en Node.js ofrece una amplia variedad de métodos para realizar operaciones de entrada/salida en el sistema de archivos. Estos métodos se pueden clasificar en métodos asíncronos, que utilizan devoluciones de llamada (callbacks) o promesas (promises), y métodos síncronos.

Métodos Asíncronos

Lectura de Archivos

- fs.readFile(path, options, callback): Lee el contenido de un archivo.
- fs.read(fd, buffer, offset, length, position, callback): Lee desde un archivo abierto.

Escritura de Archivos

- fs.writeFile(file, data, options, callback): Escribe datos en un archivo, reemplazando el archivo si ya existe.

- fs.appendFile(file, data, options, callback): Añade datos al final de un archivo.
- fs.write(fd, buffer, offset, length, position, callback): Escribe en un archivo abierto.

Apertura y Cierre de Archivos

- fs.open(path, flags, mode, callback): Abre un archivo.
- fs.close(fd, callback): Cierra un archivo.

Información de Archivos

- fs.stat(path, callback): Devuelve información sobre un archivo o directorio.
- fs.lstat(path, callback): Igual que fs.stat(), pero no sigue enlaces simbólicos.
- fs.fstat(fd, callback): Igual que fs.stat(), pero opera en un archivo abierto.

Operaciones de Directorio

- fs.readdir(path, options, callback): Lee el contenido de un directorio.
- fs.mkdir(path, options, callback): Crea un directorio.
- fs.rmdir(path, callback): Elimina un directorio.

Eliminar Archivos

- fs.unlink(path, callback): Elimina un archivo.

Renombrar y Mover Archivos

- fs.rename(oldPath, newPath, callback): Renombra o mueve un archivo o directorio.

Enlaces

- fs.symlink(target, path, type, callback): Crea un enlace simbólico.
- fs.link(existingPath, newPath, callback): Crea un enlace duro.

Cambiar Permisos

- fs.chmod(path, mode, callback): Cambia los permisos de un archivo.
- fs.chown(path, uid, gid, callback): Cambia el propietario y grupo de un archivo.

Flujos (Streams)

- fs.createReadStream(path, options): Crea un flujo de lectura desde un archivo.

- fs.createWriteStream(path, options): Crea un flujo de escritura a un archivo.

Métodos Síncronos

Lectura de Archivos

- fs.readFileSync(path, options): Lee el contenido de un archivo.
- fs.readSync(fd, buffer, offset, length, position): Lee desde un archivo abierto.

Escritura de Archivos

- fs.writeFileSync(file, data, options): Escribe datos en un archivo, reemplazando el archivo si ya existe.
- fs.appendFileSync(file, data, options): Añade datos al final de un archivo.
- fs.writeSync(fd, buffer, offset, length, position): Escribe en un archivo abierto.

Apertura y Cierre de Archivos

- fs.openSync(path, flags, mode): Abre un archivo.
- fs.closeSync(fd): Cierra un archivo.

Información de Archivos

- fs.statSync(path): Devuelve información sobre un archivo o directorio.
- fs.lstatSync(path): Igual que fs.statSync(), pero no sigue enlaces simbólicos.
- fs.fstatSync(fd): Igual que fs.statSync(), pero opera en un archivo abierto.

Operaciones de Directorio

- fs.readdirSync(path, options): Lee el contenido de un directorio.
- fs.mkdirSync(path, options): Crea un directorio.
- fs.rmdirSync(path): Elimina un directorio.

Eliminar Archivos

- fs.unlinkSync(path): Elimina un archivo.

Renombrar y Mover Archivos

- fs.renameSync(oldPath, newPath): Renombra o mueve un archivo o directorio.

Enlaces

- fs.symlinkSync(target, path, type): Crea un enlace simbólico.
- fs.linkSync(existingPath, newPath): Crea un enlace duro.

Cambiar Permisos

- fs.chmodSync(path, mode): Cambia los permisos de un archivo.
- fs.chownSync(path, uid, gid): Cambia el propietario y grupo de un archivo.

Ejemplo

Leer un archivo de forma asíncrona

```
const fs = require('fs');

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error(err);
    return;
  }
  console.log(data);
});
```

Uso en Aplicaciones Reales

El módulo fs es fundamental para cualquier aplicación Node.js que necesite interactuar con el sistema de archivos. Se utiliza en una variedad de aplicaciones, desde servidores web que sirven archivos estáticos hasta herramientas de línea de comandos que manipulan archivos y directorios.

Módulo http

El módulo http en Node.js es uno de los módulos nativos más fundamentales, que permite la creación de servidores HTTP y realizar solicitudes HTTP. Es esencial para construir aplicaciones web, servidores de APIs y servicios de red. El módulo proporciona una API basada en eventos para manejar el ciclo de vida de una solicitud y respuesta HTTP.

Funcionamiento

Carga del módulo:

Se carga utilizando la función require():

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

Creación de un servidor HTTP:

Utilizando el método http.createServer(), se puede crear un servidor que escuche solicitudes entrantes y envíe respuestas.

Métodos y Propiedades Clave:

- http.createServer([requestListener]): Crea un nuevo servidor HTTP.
- server.listen(port, [hostname], [backlog], [callback]): Hace que el servidor comience a aceptar conexiones en el puerto especificado.
- server.close([callback]): Cierra el servidor.
- request.method: Método HTTP utilizado (GET, POST, etc.).
- request.url: URL de la solicitud.
- request.headers: Encabezados de la solicitud.
- response.writeHead(statusCode, [statusMessage], [headers]): Envía una respuesta de cabecera al solicitante.
- response.write(chunk, [encoding], [callback]): Envía un fragmento de la respuesta.
- response.end([data], [encoding], [callback]): Finaliza la respuesta.

Ejemplos

Crear un servidor HTTP básico

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

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
```

```
});

server.listen(3000, '127.0.0.1', () => {
  console.log('Server running at http://127.0.0.1:3000/');
});
```

Manejar diferentes rutas

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

const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/plain');

  if (req.url === '/') {
    res.statusCode = 200;
    res.end('Welcome to the homepage\n');
  } else if (req.url === '/about') {
    res.statusCode = 200;
    res.end('Welcome to the about page\n');
  } else {
    res.statusCode = 404;
    res.end('Page not found\n');
  }
});

server.listen(3000, '127.0.0.1', () => {
  console.log('Server running at http://127.0.0.1:3000/');
});
```

Servir archivos estáticos

Para servir archivos estáticos, como HTML, CSS, o imágenes, podemos usar el módulo fs junto con http:

```
const http = require('http');
const fs = require('fs');
const path = require('path');

const server = http.createServer((req, res) => {
  if (req.url === '/') {
    fs.readFile(path.join(__dirname, 'index.html'), (err, content) => {
      if (err) {
        res.statusCode = 500;
```

```
        res.setHeader('Content-Type', 'text/plain');
        res.end('Server Error\n');
        return;
      }
      res.statusCode = 200;
      res.setHeader('Content-Type', 'text/html');
      res.end(content);
    });
  } else {
    res.statusCode = 404;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Page not found\n');
  }
});

server.listen(3000, '127.0.0.1', () => {
  console.log('Server running at http://127.0.0.1:3000/');
});
```

Uso en Aplicaciones Reales

El módulo http es la base sobre la cual se construyen muchos frameworks web de Node.js, como Express. Permite manejar solicitudes HTTP de manera detallada y controlar todos los aspectos del ciclo de vida de una solicitud y respuesta.