What is Node.js and why would you use it?

Node.js is an open-source, cross-platform runtime environment that executes JavaScript code outside of a web browser. It was built on Chrome's V8 JavaScript engine, and it allows developers to build fast, scalable network applications. Node.js is event-driven, which means that it can handle a large number of connections with minimal resources. It is particularly useful for building real-time web applications, such as chat applications, gaming platforms, or collaborative tools.

How does Node.js handle asynchronous I/O?

Node.js uses a single-threaded, event-driven architecture to handle asynchronous I/O. Instead of creating a new thread for every incoming request, Node.js uses an event loop to manage incoming requests and responses. When a request comes in, Node.js adds it to the event loop, which then processes the request asynchronously. When the response is ready, Node.js sends it back to the client.

What is event-driven programming in Node.js?

Event-driven programming is a programming paradigm that allows developers to write code that responds to specific events or signals. In Node.js, event-driven programming is used to handle asynchronous I/O. When an event occurs, such as an incoming request or a completed database query, Node.js uses an event emitter to notify the rest of the application. The application can then respond to the event by executing a callback function or emitting its own events.

What is callback hell and how do you avoid it in Node.js?

Callback hell is a term used to describe the situation where callback functions are nested several levels deep, making the code difficult to read and maintain. This can happen in Node.js applications that rely heavily on asynchronous I/O. To avoid callback hell, developers can use techniques such as promises, async/await, or the use of libraries like async.js. These techniques can simplify the code and make it more readable.

What is the difference between require and import statements in Node.js?

In Node.js, the require statement is used to load a module, while the import statement is used in ES6 modules to import functionality from other modules. The require statement is a CommonJS module system, which is a module system used in Node.js, while the import statement is a new syntax for loading modules introduced in ES6.

How do you handle errors in Node.js?

In Node.js, errors can be handled using try/catch blocks or by passing errors as the first argument in a callback function. Developers can also use the error event to handle errors that occur in event emitters. It is important to handle errors appropriately in Node.js applications to prevent crashes and ensure proper error reporting.

What is middleware in Node.js and how do you use it?

Middleware is a function that sits between the server and the application logic, processing requests and responses. In Node.js, middleware can be used to handle authentication, logging, caching, or other functions that need to be applied to multiple routes or endpoints. Middleware is typically used in conjunction with the Express framework, and it can be added to a route using the use() method.

How do you handle authentication in a Node.js application?

In a Node.js application, authentication can be handled using techniques such as cookies, JSON Web Tokens (JWT), or OAuth. Authentication middleware can be added to routes to ensure that only authenticated users can access certain parts of the application. It is important to store passwords securely using techniques such as hashing and salting to prevent data breaches.

What is the purpose of the process object in Node.js?

The process object is a global object in Node.js that provides information about the current Node.js process, as well as methods to interact with it. It can be used to access environment variables, command line arguments, and other information about the running process. The process object also has methods for exiting the process, setting up signal handlers, and sending messages between processes.

What is the difference between synchronous and asynchronous programming in Node.js?

Synchronous programming is a programming model where each operation blocks the execution until it completes. Asynchronous programming, on the other hand, allows multiple operations to be executed simultaneously without blocking the execution of the program. In Node.js, asynchronous programming is used extensively to handle I/O operations, which can be time-consuming and block the event loop if executed synchronously.

How does Node.js handle multi-threading?

Node.js is single-threaded, which means that it can only execute one task at a time. However, Node.js can handle multiple requests simultaneously by using an event-driven architecture and a non-blocking I/O model. In addition, Node.js can take advantage of multi-core CPUs by using the cluster module, which allows developers to create child processes to handle incoming requests.

What is the difference between a buffer and a stream in Node.js?

A buffer is a temporary storage area in memory that holds data while it is being transferred between two endpoints. A stream, on the other hand, is a continuous flow of data that is processed in chunks as it is received. In Node.js, buffers are used to store binary data, such as images or video, while streams are used to transfer data between different parts of an application or between a server and a client.

What is the difference between setTimeout and setImmediate in Node.js?

setTimeout and setImmediate are both used to schedule code to be executed at a later time. However, setTimeout schedules the code to be executed after a certain amount of time has passed, while setImmediate schedules the code to be executed in the next iteration of the event loop. This means that setImmediate can be used to execute code immediately after the current code block has completed, without waiting for the next event loop iteration.

How do you handle file uploads in a Node.js application?

In a Node.js application, file uploads can be handled using middleware such as multer. Multer is a middleware that allows developers to handle file uploads in multiple formats, such as binary data or multipart/form-data. The middleware can be used to handle file uploads for single or multiple files, and it can be configured to limit the size or type of the files that can be uploaded.

What is clustering in Node.js?

Clustering is a technique used in Node.js to take advantage of multi-core CPUs. It allows developers to create child processes that can handle incoming requests, spreading the load across multiple CPUs. Clustering can be implemented using the cluster module, which allows developers to create a master process that spawns multiple worker processes to handle requests.

What is the purpose of the util module in Node.js?

The util module is a built-in module in Node.js that provides a set of utility functions that can be used to perform common tasks, such as formatting strings, logging, and working with objects. The util module also provides a set of functions for working with streams, buffers, and events.

What is the difference between process.nextTick and setImmediate in Node.js?

process.nextTick and setImmediate are both used to schedule code to be executed in the next iteration of the event loop. However, process.nextTick schedules the code to be executed before any I/O operations, while setImmediate schedules the code to be executed after any I/O operations. This means that process.nextTick is used to execute code as soon as possible, even if it means delaying I/O operations, while setImmediate is used to execute code after any I/O operations have completed.

How do you handle errors in Node.js?

In Node.js, errors can be handled using try-catch blocks or by attaching an error event listener to an object that emits events. Errors that occur during asynchronous operations can be handled using callbacks or promises. Node.js also provides a global process.on('uncaughtException') event listener that can be used to catch unhandled exceptions.

What is the purpose of the crypto module in Node.js?

The crypto module is a built-in module in Node.js that provides cryptographic functionality, such as creating secure hash algorithms, generating random numbers, and encrypting and decrypting data. The module supports various cryptographic algorithms, including AES, RSA, and SHA.

How do you debug a Node.js application?

Node.js provides several built-in debugging tools, including the debugger statement, which can be used to pause code execution and inspect the state of the program. Node.js also provides a command-line debugger tool called node inspect, which allows developers to step through code and set breakpoints. In addition, third-party debugging tools, such as VS Code and WebStorm, provide integrated debugging capabilities for Node.js applications.

What is the purpose of the http module in Node.js?

The http module is a built-in module in Node.js that provides functionality for creating HTTP servers and clients. It allows developers to create servers that listen for incoming HTTP requests and respond to them with HTML pages, JSON data, or other content. The http module also provides methods for sending HTTP requests to remote servers.

What is the purpose of the url module in Node.js?

The url module is a built-in module in Node.js that provides functionality for parsing and formatting URLs. It can be used to extract information from URLs, such as the host, port, pathname, and query parameters. The url module also provides methods for formatting URLs and resolving relative URLs.

What is the purpose of the fs module in Node.js?

The fs module is a built-in module in Node.js that provides functionality for working with the file system. It allows developers to read and write files, create and delete directories, and perform other operations related to file management. The fs module also provides methods for working with streams and buffers.

What is the purpose of the child_process module in Node.js?

The child_process module is a built-in module in Node.js that provides functionality for spawning child processes. It allows developers to execute shell commands, run scripts, and perform other tasks in a separate child process. The child_process module also provides methods for communicating with child processes through pipes and streams.

What is the purpose of the os module in Node.js?

The os module is a built-in module in Node.js that provides functionality for working with the operating system. It allows developers to access information about the system, such as the CPU architecture, memory usage, and network interfaces. The os module also provides methods for working with the file system, processes, and users.

What is the purpose of the events module in Node.js?

The events module is a built-in module in Node.js that provides functionality for creating and handling events. It allows developers to create custom events and emit them when certain actions occur. The events module also provides methods for registering event listeners that can respond to events when they are emitted.

What is the purpose of the stream module in Node.js?

The stream module is a built-in module in Node.js that provides functionality for working with streams of data. It allows developers to read and write data from and to streams, such as files, network sockets, and HTTP requests and responses. The stream module also provides methods for transforming and manipulating streams of data.

What is the purpose of the cluster module in Node.js?

The cluster module is a built-in module in Node.js that provides functionality for creating child processes that share the same server port. It allows developers to take advantage of multi-core systems by creating multiple worker processes that can handle incoming requests. The cluster module also provides methods for managing and monitoring worker processes.

What is the purpose of the process object in Node.js?

The process object is a global object in Node.js that provides functionality for interacting with the current Node.js process. It allows developers to access information about the current process, such as the command-line arguments, environment variables, and memory usage. The process object also provides methods for controlling the current process, such as exiting the process, setting the process title, and sending signals to the process.

What is the purpose of the net module in Node.js?

The net module is a built-in module in Node.js that provides functionality for creating TCP servers and clients. It allows developers to create servers that listen for incoming TCP connections and respond to them with data. The net module also provides methods for creating TCP clients that can connect to remote servers.

Example of creating a simple HTTP server using the http module:

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello, World!");
});

server.listen(3000, () => {
  console.log("Server listening on port 3000");
});
```

Example of creating a custom module:

```
// logger.js

function log(message) {
  console.log(message);
}

module.exports = log;
```

~~~
const logger = require('./logger');

logger('Hello, World!');

Example of using the fs module to read a file:

```javascript
Copy code
const fs = require('fs');

fs.readFile('file.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
~~~

Example of using the path module to work with file paths:

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

```
const fullPath = path.join(__dirname, 'folder', 'file.txt');
console.log(fullPath);
```

Example of creating a simple TCP server using the net module:

```
Copy code
const net = require('net');

const server = net.createServer((socket) => {
  socket.write('Echo server\r\n');
  socket.pipe(socket);
});
server.listen(1337, '127.0.0.1', () => {
  console.log('Server listening on port 1337');
});
```

Example of using the events module to create and emit custom events:

```
Copy code
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();

myEmitter.on('event', () => {
  console.log('An event occurred');
});

myEmitter.emit('event');
```

Example of using the stream module to read data from a file and write it to another file:

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

const readStream = fs.createReadStream('input.txt');
const writeStream = fs.createWriteStream('output.txt');

readStream.pipe(writeStream);
```

Example of using the cluster module to create worker processes:

```
Copy code
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  console.log(`Master ${process.pid} is running`);

  // Fork workers
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`Worker ${worker.process.pid} died`);
  });
} else {
  // Workers can share any TCP connection
  // In this case it is an HTTP server
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('Hello, World!');
  }).listen(8000);

  console.log(`Worker ${process.pid} started`);
}
```

Example of using the process object to get information about the current process:

```
Copy code
console.log(`Node.js version: ${process.version}`);
console.log(`Current working directory: ${process.cwd()}`);
console.log(`Command-line arguments: ${process.argv}`);
console.log(`Environment variables: ${process.env}`);
console.log(`Memory usage: ${process.memoryUsage()}`);
```