**Create HTTP Server in Node JS**

Overview

Node.js has revolutionized the way we build web applications by allowing developers to create server-side applications using JavaScript. One of its core features is the ability to easily create an HTTP server. In this lesson, we'll dive into the process of creating a basic HTTP server using Node.js.

Importing the HTTP Module

To start our adventure, we need to import the built-in HTTP module. The HTTP module provides essential functionality for creating an HTTP server, including handling incoming requests and sending responses. To import the module, simply use the require() function like so:

const http = require('http');

Using the http.createServer() Method

Now that we've imported the HTTP module, it's time to use its createServer() method to create an instance of an HTTP server. This method returns an http.Server object, which we can configure to handle incoming requests and send responses.

Here's a basic example of how to create an HTTP server using the createServer() method:

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 running at <http://localhost:3000/>');

});

In this example, we pass a callback function to the createServer() method. This function will be called each time the server receives an incoming request. The callback function takes two arguments: req (the request object) and res (the response object).

We then call the listen() method on the server object to start listening for incoming connections on port 3000. When the server is ready, the provided callback function is executed, and a message is printed to the console.

Understanding the Request and Response Objects

The req and res objects are instances of the http.IncomingMessage and http.ServerResponse classes, respectively. They contain valuable information about the incoming request and provide methods for crafting the response.

* The req object (Request): This object contains information about the incoming request, such as the URL, headers, and body. Some of the useful properties of the req object include req.url, req.method, and req.headers.
* The res object (Response): This object represents the server's response to the incoming request. You can use it to set the status code, headers, and response body. Some of the useful methods of the res object include res.writeHead(), res.write(), and res.end().

Let's take a closer look at the example server code from earlier:

const server = http.createServer((req, res) => {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World!');

});

In this example, we use the res.writeHead() method to set the response's status code to 200 (indicating success) and the Content-Type header to 'text/plain'. We then use the res.end() method to send the response body ('Hello World!') and signal the end of the response.

By understanding the request and response objects, we can create more complex server logic to handle various types of requests and send appropriate responses.

Binding the Server to a Specific Port and Starting to Listen for Incoming Connections

To make your server accessible, you need to bind it to a specific port and start listening for incoming connections. This is done using the listen() method on the http.Server object. The listen() method takes two arguments: the port number and a callback function to be executed when the server starts listening.

Here's an example:

server.listen(3000, () => {

console.log('Server running at <http://localhost:3000/>');

});

In this example, we bind the server to port 3000 and start listening for incoming connections. When the server is ready, the provided callback function is executed, and a message is printed to the console.

Basic Example: Creating a "Hello, World!" HTTP Server

Now that we have covered the fundamental components of creating an HTTP server with Node.js, let's create a simple "Hello, World!" server as a practical example. This server will respond to every incoming request with the text "Hello, World!".

// Step 1: Import the HTTP module

const http = require('http');

// Step 2: Create the server using the http.createServer() method

const server = http.createServer((req, res) => {

// Step 3: Set the response headers

res.writeHead(200, {'Content-Type': 'text/plain'});

// Step 4: Send the response body

res.end('Hello, World!');

});

// Step 5: Bind the server to a specific port and start listening for incoming connections

server.listen(3000, () => {

console.log('Server running at <http://localhost:3000/>');

});

In this example, we import the HTTP module, create the server using the createServer() method, set the response headers, send the response body, and bind the server to port 3000.

With this basic example, you now have the foundation to build more complex HTTP servers using Node.js. As you continue to explore Node.js, you can experiment with handling different types of requests, routing, and integrating with databases or other services to create fully-fledged web applications.

Conclusion

In conclusion, Node.js has made it possible for developers to create server-side applications using JavaScript, and one of its core features is the ability to easily create an HTTP server. By importing the built-in HTTP module and using its createServer()method, we can configure an HTTP server to handle incoming requests and send responses. Understanding the request and response objects, as well as how to bind the server to a specific port and start listening for incoming connections, are crucial components to building a basic HTTP server. With this foundation, we can create more complex server logic and build fully-fledged web applications.