**1. Creating an HTTP Server**

The primary use of the http module is to create an HTTP server that can listen to incoming requests and send responses. You can do this using the http.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 is listening on port 3000');

});

* createServer(callback) takes a callback that is called whenever an HTTP request is made to the server.
* The callback function receives two arguments:
  + req: The request object, which represents the incoming HTTP request.
  + res: The response object, which allows you to send an HTTP response back.

**2. Handling Requests**

The req object contains information about the HTTP request, including:

* req.method: The HTTP method (GET, POST, PUT, etc.)
* req.url: The URL of the request
* req.headers: An object containing all the request headers
* req.body: The body of the request (in case of POST/PUT requests)

For example, handling different HTTP methods:

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

if (req.method === 'GET') {

res.write('You made a GET request!');

} else if (req.method === 'POST') {

res.write('You made a POST request!');

}

res.end();

}).listen(3000);

**3. Handling Responses**

The res object allows you to control the response sent to the client.

* res.writeHead(statusCode, headers): Sends an HTTP status code and a set of headers.
* res.write(data): Sends data to the client (this can be called multiple times before ending the response).
* res.end([data]): Ends the response and optionally sends the final data to the client.

For example, sending a JSON response:

const http = require('http');

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

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

const data = { message: 'Hello, World!' };

res.end(JSON.stringify(data));

}).listen(3000);

**4. Request Handling with URL**

You can parse the URL of the request using Node’s url module, or manually check it with req.url.

Example:

const url = require('url');

const http = require('http');

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

const parsedUrl = url.parse(req.url, true);

if (parsedUrl.pathname === '/greet') {

const name = parsedUrl.query.name || 'Guest';

res.write(`Hello, ${name}!`);

res.end();

} else {

res.write('Invalid path');

res.end();

}

}).listen(3000);

**5. Handling Different Content Types**

Node.js makes it easy to respond with different types of content by setting the Content-Type header in the response:

// JSON Response

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

res.end(JSON.stringify({ message: 'This is a JSON response' }));

// HTML Response

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

res.end('<h1>Hello, World!</h1>');

**6. HTTP Methods**

The http module allows you to handle various HTTP methods like GET, POST, PUT, DELETE, etc., by checking req.method.

Example of handling POST requests:

const http = require('http');

const querystring = require('querystring');

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

if (req.method === 'POST') {

let body = '';

req.on('data', chunk => {

body += chunk;

});

req.on('end', () => {

const parsedBody = querystring.parse(body);

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

res.end(JSON.stringify(parsedBody));

});

} else {

res.writeHead(405, { 'Content-Type': 'text/plain' });

res.end('Method Not Allowed');

}

}).listen(3000);

**7. Creating an HTTP Client (Making Requests)**

You can also use the http module to make HTTP requests to other servers.

const http = require('http');

const options = {

hostname: 'example.com',

port: 80,

path: '/path',

method: 'GET',

};

const req = http.request(options, (res) => {

let data = '';

res.on('data', chunk => {

data += chunk;

});

res.on('end', () => {

console.log(data);

});

});

req.on('error', (error) => {

console.error(`Problem with request: ${error.message}`);

});

req.end();

**8. Error Handling**

When working with the HTTP module, it is important to handle errors appropriately. For example, when the server can't bind to the port or when there is an issue with a client request.

Example:

const http = require('http');

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

try {

res.write('Handling request');

res.end();

} catch (err) {

res.writeHead(500, { 'Content-Type': 'text/plain' });

res.end('Internal Server Error');

}

});

server.listen(3000, (err) => {

if (err) {

console.error('Failed to start the server:', err);

} else {

console.log('Server running on port 3000');

}

});

**Summary of Key Methods**

* http.createServer(): Creates a new HTTP server.
* res.writeHead(statusCode, headers): Sends a response status and headers.
* res.write(data): Sends data to the client.
* res.end([data]): Ends the response and optionally sends final data.
* req.on('data', callback): Listens for incoming request data (useful for POST requests).
* req.on('end', callback): Triggered when the request data has been fully received.
* http.request(options, callback): Makes an HTTP request to another server.