Node.js HTTP Module

- Definition, Methods, Properties, Events, and Examples
- By Mitesh Prajapati

Introduction

- - Node.js is a server-side JavaScript runtime.
- - HTTP module allows communication over HTTP protocol.
- - Can create servers or make HTTP requests.
- - Built-in module, no installation required.

Definition

- HTTP Module:
- The `http` module in Node.js is a built-in module that allows Node.js to transfer data over HTTP, create servers, and make client requests.
- Key Points:
- - Event-driven and asynchronous.
- - Can handle multiple requests simultaneously.
- - Core for web applications in Node.js.

How to Include HTTP Module

-
  ```js
- const http = require('http');
- ```
- - Now `http` can be used to create servers or send requests.
  ```

Server-Side Methods: createServer()

- Creates an HTTP server.

-
  ```js
- const server = http.createServer((req, res) => {
-     res.writeHead(200, {'Content-Type':'text/plain'});
-     res.end('Hello World');
- });
- server.listen(3000);
- ```
- - Use Case: Serve webpages or APIs.
  ```

Server Listening

- server.listen(port[, hostname][, backlog][, callback])

-
  ```js
- server.listen(3000, () => {
-     console.log('Server running on port 3000');
- });
- ```
- - Starts server on specific port.
  ```

Server Closing

- server.close([callback])

-
  ```js
- server.close(() => {
-     console.log('Server closed');
- });
- ```
- - Stops server from accepting new requests.
  ```

Server Events

Event	Description
request	Triggered when client makes a request
connection	Triggered when a TCP connection opens
close	Triggered when server closes
error	Triggered on server errors

Client-Side Methods: request()

- Makes HTTP requests (GET, POST, etc.)

-
  ```js
- const options = {hostname:'example.com', port:80, path:'/', method:'GET'};
- const req = http.request(options, res => { res.on('data', d => process.stdout.write(d)) });
- req.end();
- ```
- - Use Case: Fetch API data.
  ```

Client-Side Methods: get()

- Shortcut for GET requests. Automatically ends request.

-
  ```js
- http.get('http://example.com', res => {
-     res.on('data', chunk => console.log(chunk.toString()));
- });
- ```
- - Use Case: Fetch public APIs easily.
  ```

HTTP Properties

Property	Description
http.STATUS_CODES	Object with all HTTP status codes
http.METHODS	Array of HTTP methods (GET, POST, etc.)

Request & Response Objects

- req (Request)
- - req.method → GET / POST
- - req.url → Requested URL

- res (Response)
- - res.write(data) → Send data to client
- - res.end(data) → End response
- - res.setHeader(name, value) → Set header

Use Cases of HTTP Module

- 1. Create web servers and APIs
- 2. Make HTTP requests to other servers
- 3. Build proxy servers
- 4. Stream data to clients

Example – Complete Server

```js
const http = require('http');
const server = http.createServer((req, res) => {
  if(req.url === '/home') {
    res.writeHead(200, {'Content-Type':'text/html'});
    res.end('<h1>Welcome Home</h1>');
  } else {
    res.writeHead(404, {'Content-Type':'text/plain'});
    res.end('Page Not Found');
  }
});
server.listen(3000, () => console.log('Server running on port 3000'));
```

- Output:
- - `/home` → "Welcome Home"
- - Other → "Page Not Found"

Summary

- - HTTP module → Core module for HTTP servers/clients
- - Server methods → createServer(), listen(), close()
- - Client methods → request(), get()
- - Useful for websites, APIs, and data streaming

Diagram (Optional)

- Server-Client Flow Diagram:
- Client → HTTP request → Server → Response → Client