



6. Webサーバの起動

Node.js の Webサーバ

JavaScript でプログラミング

通常、Webサーバを構築する場合、`Apache`、`Nginx` といったサービスを `Linux` にインストールして利用しますが、`Node.js`でWebサーバを独自に構築することもできます。サーバサイドのプログラミング言語は、`JavaScript` のため勉強コストが比較的低い傾向にあります。

Node.js サーバの特徴

`Node.js` の Webサーバは、軽量の処理を大量のアクセスに対応して、比較的サーバ負荷をかけずに動作させるために `ノンブロッキングI/O`、`イベントループ` といった機能が特徴的です。開発では `イベントドリブン（イベント駆動型）` といわれる手法でプログラミングします。

ノンブロッキングI/O

`ノンブロッキングI/O` は、ノンブロッキング処理ともいい、`シングルスレッドで複数の処理をする仕組み`です。`I/O` は、コンピュータの情報の入出力 (`Input / Output`) のことをさします。

ノンブロッキング処理とブロッキング処理

ノンブロッキング処理は、`I/O` の処理待ちが発生してもすぐに関数の反応があるのが特徴で、`複数処理を同時に進めながら他の処理が終わるまで待つ仕組み`です。逆にブロッキング処理は複数処理ではなく、`1つ1つ順番に処理を実行`します。

参考

イベントループ

イベントループとは

イベントループは、ノンブロッキングI/Oを効率よく実行するための仕組みです。

- タイマーのスケジュール設定
- イベントのコールバックをキューに追加
- コールバックを実行
- 次のコールバックのチェックやクローズ

サーバーが起動すると同時にイベントを待機する状態となり、`JavaScript` のイベントハンドラのリクエスト処理します。

非同期処理との違い

同じような仕組みに `非同期処理` がありますが、`非同期処理` は `I/O` が完了したタイミングで通知し、完了通知があるまでの間は他の処理を進めることができます。

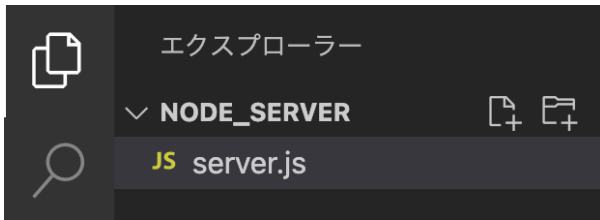
Node.js サーバ構築

`Node.js` に標準搭載されている `http` モジュールを使って、Webサーバのリクエスト&レスポンスを処理します。

ファイル構成

`node_server/` フォルダを作成して `VSCode` で開きます。また `server.js` も作成しておきます。

```
node_server
└─ server.js
```



http モジュール

`http` モジュールを読み込みます。

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

ポート & ホストの設定

ポート番号 `3000` とホスト名 `localhost` を定数に設定します。また、レスポンスで表示する文字も用意しておきます。

```
const port = 3000;
const host = 'localhost';
const message = "Hello Node Server!\n";
```

サーバの作成

`http.createServer()` メソッドでサーバを作成し、コールバック関数で `request` と `response` オブジェクトが利用できます。

```
const app = http.createServer(function (request, response) {

})
```

ヘッダの出力

ヘッダでは成功の `200` と、 `Content-Type` に `text/html` をレスポンスします。

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

レスポンスの書き込み

コールバック関数の中でレスポンスを出力します。

```
response.write(message);
```

レスポンスの終了

サーバーから返答するメソッド `end()` でレスポンスを終了します。

```
response.end();
```

サーバの待機

`listen()` メソッドでサーバを待機して、クライアントからのリクエストを待ちます。

```
app.listen(port, host)
```

ソース

`server.js` にサーバ起動のコードを記述します。 `http.createServer()` のコールバック関数の引数 `request`、`response` を使って、サーバのリクエスト&レスポンスの処理します。

```
const http = require('http');
const port = 3000;
const host = 'localhost';
const message = "Hello Node Server!\n";

const app = http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/html'});
  // レスポンス書き込み
  response.write(message);

  //レスポンスを閉じる
  response.end();

  console.log(`Method: ${request.method}`);
  console.log(`Response: ${message}`);
});

// ホストとポートを指定して監視
app.listen(port, host);
```

```
console.log(`Server listen: http://${host}:${port}`);
```

サーバの起動

VSCode のターミナルで **server.js** を実行してWebサーバを起動します。サーバが起動するとホストlocalhost、ポート3000の **localhost:3000** が待機アドレスになります。

```
% node server.js  
Server listen: http://localhost:3000
```

サーバの動作確認

ブラウザで **http://localhost:3000** でアクセスしてみましょう。ブラウザに文字が表示されれば成功です。

A screenshot of a web browser's address bar. It contains navigation icons (back, forward, refresh, home) and the text 'localhost:3000'.

Hello Node Server!

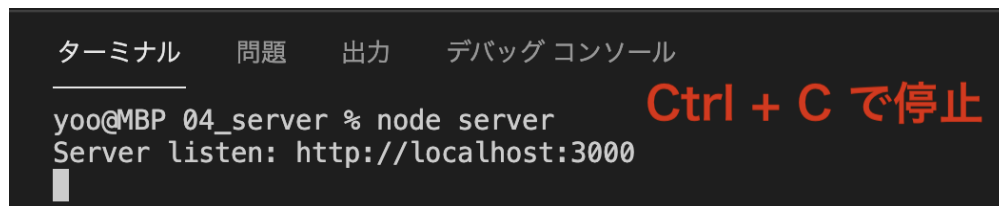
ターミナルログの確認

ターミナルのログを確認します。同じ内容の **GET** リクエストが2回表示されますが **favicon.ico** も取得するためです。

```
Server listen: http://localhost:3000  
Method: GET  
Response: Hello Node Server!  
  
Method: GET  
Response: Hello Node Server!
```

サーバの停止

サーバの停止は、サーバ起動中のターミナル上で **Ctrl + C** を入力します。



```
ターミナル 問題 出力 デバッグ コンソール
yoo@MBP 04_server % node server
Server listen: http://localhost:3000
```

Ctrl + C で停止

当サイトの教材をはじめとするコンテンツ（テキスト、画像等）の無断転載・無断使用を固く禁じます。これらのコンテンツについて権利者の許可なく複製、転用等する事は法律で禁止されています。尚、当ウェブサイトの内容をWeb、雑誌、書籍等へ転載、掲載する場合は「ロジコヤ」までご連絡ください。