ブラウザの枠を飛び越えろ！Node.js セミナー

Chapter02

Node.jsを動かして
サーバーを立ててみよう。

Node.jsはコマンドで動かす

Node.jsはコマンドプログラム。プログラムのソースコードはテキストファイルに記述することができるが実行するにはコマンドが必要。

なお、Node.jsは「インタラクティブモード」があり、その場でプログラムを書いて実行することもできる。

インタラクティブモードでNode.jsを実行

Terminal

```
node
```

インタラクティブモードの終了

Terminal

```
.exit
```

インタラクティブモード

「インタラクティブモード」では直接プラグラムを書いて実行できる。

node

```
console.log(出力するデータ)
```

テキストファイルのNodeプログラムを動かす

「インタラクティブモード」は直接プログラムを記述するモードだが、基本的に開発する際はテキストファイルにコードを書く。

テキストファイルに書いたNodeのプログラムを実行するには、nodeコマンドのあとに実行したいファイルを指定する。

スクリプトファイルの実行

Terminal

```
node 実行するファイル
```

例：「chapter02」フォルダ内の「hello.js」を実行する場合

chapter02/hello.js

Node

```
console.log("Hello Node.js");
```

Terminal

```
node chapter02/hello.js
```

グローバルオブジェクトの違い

Node.jsはJavaScriptの実行環境だが、Node.js上で使えるJavaScriptはブラウザ上で使えるものと全く一緒ではない。
いくつかの点で異なる箇所があるが、その一つとして「グローバルオブジェクト」がある。

ブラウザの場合

```JavaScript
var name = '柴田';
```

window

window オブジェクトに定義

Node.jsの場合

```Node
var name = '柴田';
```

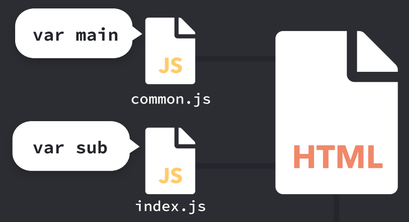
global

global オブジェクトに定義

グローバルオブジェクトの違い

Node.jsはJavaScriptの実行環境だが、Node.js上で使えるJavaScriptはブラウザ上で使えるものと全く一緒ではない。
いくつかの点で異なる箇所があるが、その一つとして「グローバルオブジェクト」がある。

ブラウザの場合



ブラウザにwindowオブジェクトが生成されるため
お互いのグローバル変数を参照、変更できる

Node.jsの場合

ファイルごとにglobalオブジェクトが生成されるため
モジュール間での参照、変更はできない

グローバルで使えるオブジェクトやメソッド

Node.jsのどのモジュールファイルでも使用できるオブジェクトやメソッドなどがある。

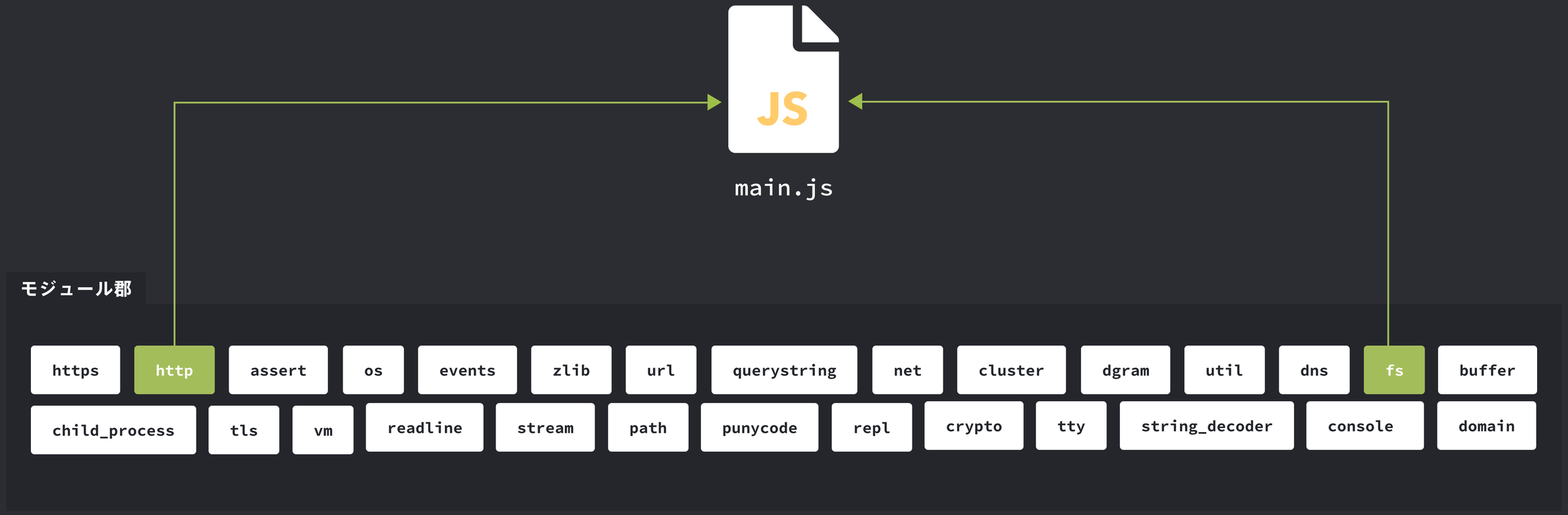
これらはNode.js固有のオブジェクトですが、この他にもJavaScriptの組み込みオブジェクトも利用できる。

- **console**
- **exports**
- **module**
- **process**
- **require()**

require()

Node.jsでは多数のオブジェクトを利用するが最初から全て使えるわけではない。

オブジェクトをモジュール（部品）化して管理し、必要なときに読み込んで利用する。



require()

Node.jsでは多数のオブジェクトを利用するが最初から全て使えるわけではない。

オブジェクトをモジュール化して管理し、必要なときに読み込んで利用する。

モジュールの読み込み

Node

```
const 変数名 = require( モジュールID );
```

例OSモジュールの読み込み

chapter02/require.js

Node

```
const os = require("os");
console.log(os.platform());
```

簡易サーバーを作る

Node.jsのhttpモジュールを使えば簡易的なサーバーを立てることができる。
このモジュールを利用してサーバーを立てることでHTTPを利用した通信を行える。

- 1.HTTPモジュールを読み込む
- 2.HTTPモジュールからサーバーオブジェクトを作る
- 3.サーバーオブジェクトを待ち受け状態にする

HTTPモジュールの読み込み

chapter02/server.js

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

サーバーオブジェクトを作る

httpモジュールを読み込んだら、createServer() でサーバーオブジェクトを生成する。

引数にはサーバーにアクセスがあった時に実行したい関数を入れておく。

その関数には、HTTPリクエストと、HTTPレスポンスを管理するための引数を設定する。

サーバーオブジェクトの作成

Node

```
const オブジェクト名 = http.createServer( サーバー実行時に処理を実行する関数 );
```

例:サーバーオブジェクトを作成

chapter02/server.js

Node

```
// HTTPモジュールの読み込み
const http = require("http");

// サーバーオブジェクトの作成
const server = http.createServer( (req, res) => {
  // サーバーにアクセスされた時に実行する関数


});
```

クライアントへの返信を終了する

引数に設定した res にはサーバーから送られた情報が格納されている。

res のメソッドを使ってクライアントへの返信を終了するには end を使ってメッセージを出しつつ終了することができる。

例:クライアントへの返信を終了する

chapter02/server.js

Node

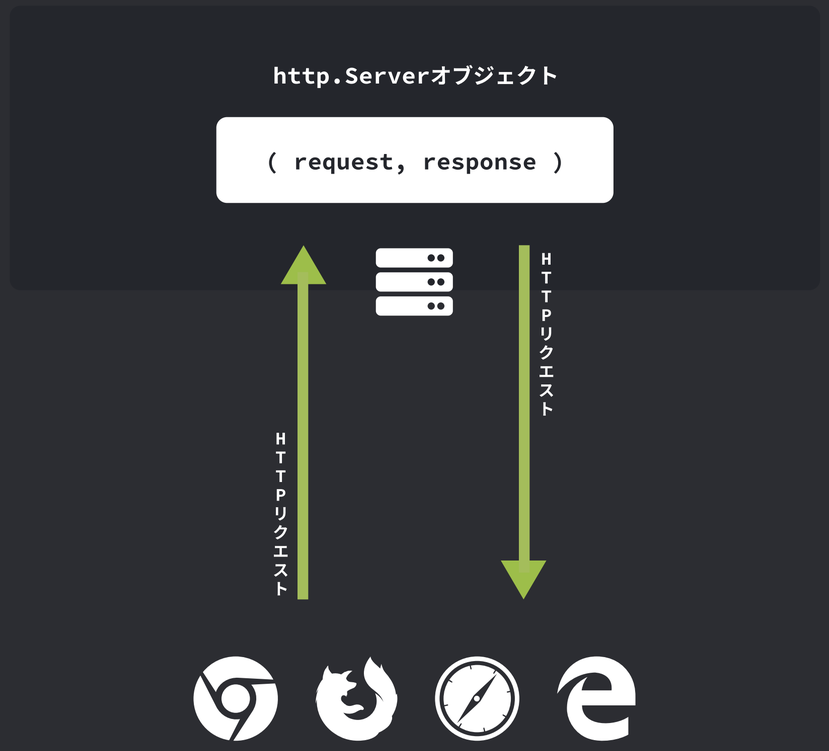
```
// HTTPモジュールの読み込み
const http = require("http");

// サーバーオブジェクトの作成
const server = http.createServer( (req, res) => {
  // サーバーにアクセスされた時に実行する関数

  // 文字列を出力させて終了
   res.end( "Hello Node.js" );

});
```

リクエストとレスポンス

http.createServerの引数で設定したリクエストとレスポンスの変数には、
以下のようなものが格納される。

- **http.ClientRequest**
  クライアントから送られてきたデータが管理されている

- **http.ServerResponse**
  サーバーから送られたデータが管理されている



待受状態にする

サーバーは外部からのアクセス（リクエスト）に応じて、結果を返す（レスポンス）必要がある。

よって、外部からアクセスが来ないか待受状態にする必要がある。

サーバー待受状態にする

Node

```
サーバーオブジェクト名.listen( ポート番号, ホスト, コールバック関数 );
```

例:クライアントへの返信を終了する

chapter02/server.js

Node

```
// HTTPモジュールの読み込み
const http = require("http");

// サーバーオブジェクトの作成
const server = http.createServer( (req, res) => {
  res.end( "Hello Node.js" );
});

// サーバーオブジェクトの作成
server.listen(3000, "127.0.0.1", () => {
  console.log("Server running at http://127.0.0.1:3000/");
});
```