ななちゃんのIT教室

JavaScript 初級プログラマ禁断の Node.js の巻

by nara.yasuhiro@gmail.com

ななちゃんが
Node.js に挑戦するという お話

第 0.5 版 2017 年 5 月 11 日

フリー素材
http://freeillustration.net

もくじ

第1回 Node.js はなぜ初級プログラマ禁断？
第2回 Node.js ことはじめ
第3回 入出力
第4回 モジュールとパッケージ
第5回 ウェブサーバも作れる！
第6回 外部サービスの呼び出し
第7回 デバッグ支援機能
第8回 Buffer クラス

第1回 Node.js はなぜ初級プログラマ禁断？

なな: 今回のお題の「Node.js」って何？

先生: JavaScript 実行環境のひとつです。今まで勉強してきた JavaScript は、ブラウザに内蔵されている JavaScript 実行エンジンで実行されるものでした。Node.js は、ブラウザとは別の、パソコンなどに直接インストールする JavaScript 実行エンジンで実行します。2009年に Ryan Dahl さんが開発しました。正式名は Node ですが、一般名詞と混同しやすいので、通称として Node.js と呼ばれています。Google Chrome ブラウザのソースコードが無償公開されているんだけど、Node.js は、Google Chrome ブラウザのソースコードから「V8 JavaScript エンジン」だけを取り出したものをベースにして、いろいろな機能を追加したものです。Node.js では、これまでに勉強した命令の多くが実行できません。たとえば、alert、document.getElementById、canvas など。専門的に言えば、DOM (Document Object Model、ブラウザの資源) に関する命令は基本的に使えません。windows に関する命令も使えません。alert も、もともとは、windows.alert の省略表記です。

なな: 使えない命令が多いってことは、機能が低いってこと？

先生: そうでもないわ。Node.js には、ブラウザ上の JavaScript では使えない多数の命令が使えるの。たとえば、ファイルの入出力。ブラウザ上の JavaScript の前提は、ネットワーク接続先にある、他サイトの html/JavaScript プログラムを実行すること。悪意を持つクラッカーが書き換えたプログラムであることも想定しなければならないの。だからセキュリティを配慮した制限が多数あります。その典型が、ファイル入出力。ファイルが壊されないよう、個人情報が盗まれないよう、厳しい制限があります。

なな: Node.js は安全だということ？

先生: まあね。ブラウザだと、あやしいプログラムを気づかずに使ってしまう可能性があるけど、Node.js は、ユーザが、自分でセキュリティを考え、自分で作ったり、納得の上で自分でダウンロードしたプログラムを実行するという前提があります。だから、ファイル入出力がかなり自由に行える。

なな: でも、ブラウザも Node.js も、同じ JavaScript なの？

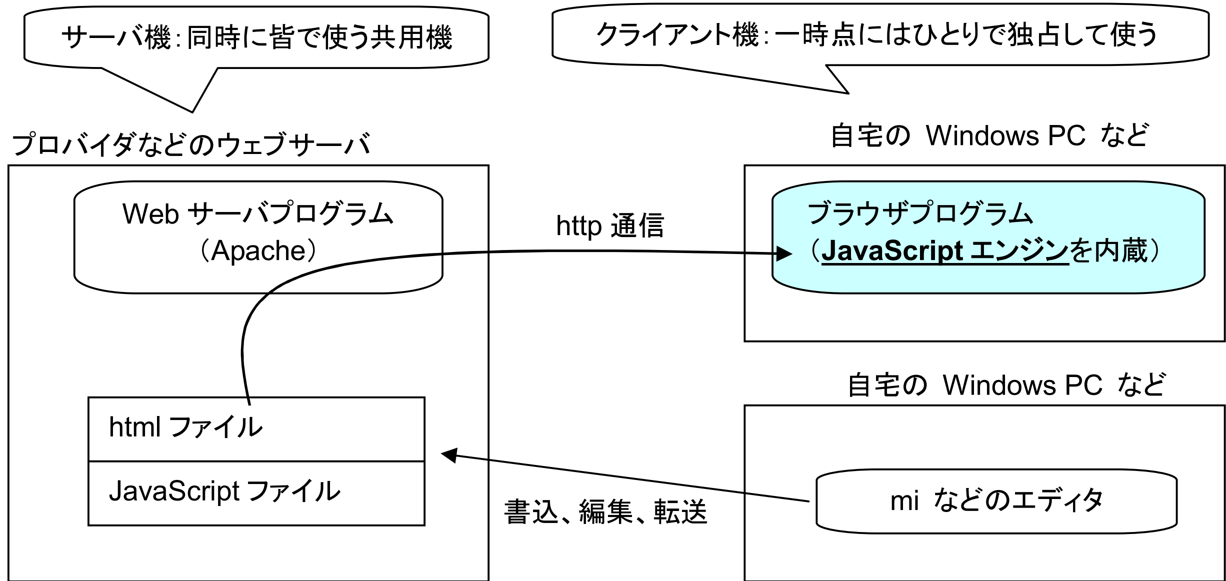
先生: たとえで言えば、アメリカ英語とイギリス、オーストラリア、フィリピンなどの英語との違いみたいなもの。英語としての文法は共通だけど、単語（語彙）に違いがある。地下鉄は、アメリカ英語で subway、イギリス英語で underground、tube。人間が話す英語なら、融通を利かして、すぐに理解できなくても意味を「想像」してくれるけど、まったく融通の利かないコンピュータが実行する JavaScript では、ずっと深刻です。Node.js の JavaScript エンジンは、ブラウザの JavaScript 用の JavaScript プログラムをまったく実行できず、エラーになってしまう。もっとも、人間が融通を利かして、ブラウザ用の JavaScript の命令を Node.js で読み替えるような「ライブラリ」を導入するという手はあるけどね。ようするに、ブラウザ用の JavaScript も十分理解できていないユーザが、Node.js をいきなり使うと、まるで分らなくなってしまうという危険があるわけ。だから「avaScript 初級プログラマ禁断」。

なな: たとえで言えば、アメリカで tube と言って、英語が通じないと悩むようなことが起き得るということね。英語のどの部分が共通で、どの部分が異なるということを知っていないと混乱するということね。

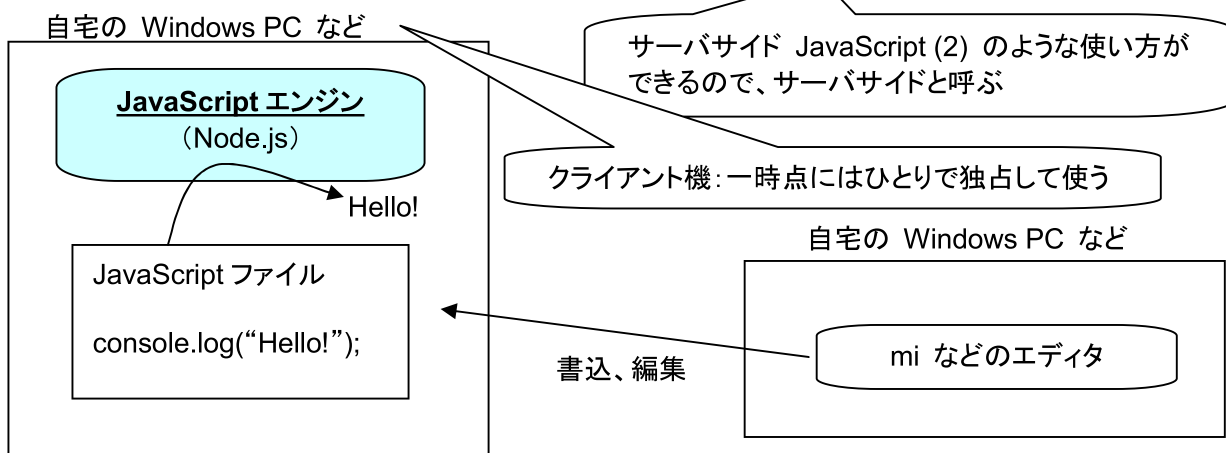
クライアントサイド JavaScript（ブラウザ環境）と、サーバサイド JavaScript（node.jp）の命令セットの違い。

	クライアントサイド JavaScript	サーバサイド JavaScript (node.jp)
var x = 1;	○	○
if 文	○	○
for 文	○	○
while 文	○	○
window.alert()	○	×
alert()	○	×
document.write()	○	×
document.getElementById()	○	×
console.log()	△（動作が異なる）	△（動作が異なる）
process.stdout.write()	×	○
process.stdin.resume()	×	○
process.stdin.setEncoding('utf8')	×	○
process.stdin.on('data', getString)	×	○
process.stdin.on('end', end)	×	○
require('http')	×	○
http.createServer()	×	○
require("fs");	×	○
writeFileSync("ファイル名", str);	×	○
readFileSync("ファイル名", code);	×	○

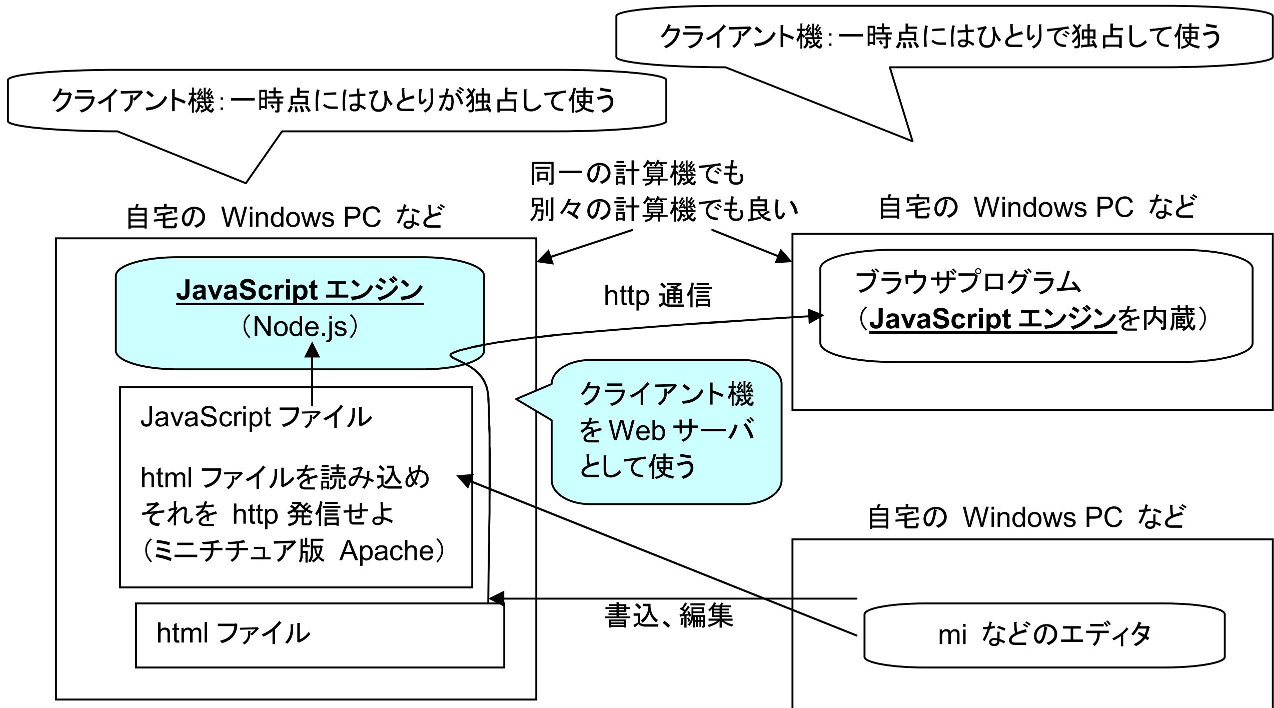
＜クライアントサイド JavaScript＞



＜サーバサイド JavaScript(1)＞ 「サーバサイド」と呼ぶのはちょっと変だけどガマン



＜サーバサイド JavaScript (2)＞ 第5回参照。クライアント機上に実現したウェブサーバ



第2回 Node.js ことはじめ

なな: Node.js を使うには、どうすれば良いの？

先生: まず、Node.js の実行環境をインストールする必要があります。ブラウザは、自分でインストールすることもできるけど、OS（Windows、MacOS、Linux など）に、はじめから用意されているものも利用できるわね。でも、Node.js は、OS にはじめから用意されていないので、はじめて使う時にインストールしないといけないの。https://nodejs.org/ja/ からインストーラをダウンロードして、インストールをします。

なな: ハードルが高そう。

先生: インストールが完了したら、コマンドウィンドウから利用します。Windows だったら、「コマンドプロンプト」（いわゆる DOS 窓）、MacOS や Linux だったら「kterm」などの「ターミナルソフト」を使います。

```
% node
> console.log("Hello, world!");
Hello, world!
undefined
>
```

「% 」は、ターミナルソフトが入力可能であることを示す印です。 コマンドプロンプトなら「C:¥Users¥User>」だったります。「node」 + 改行 で、Node.js が入力待ちになって「>」が表示されます。そこで、Node.js の命令「console.log("Hello, world!");」を入力すると、「Hello, world!」と表示されます。「undefined」は、「console.log("Hello, world!");」命令が、特に値を返してきていないことを示していますが、今は深く考える必要はありません。そして、次の命令入力を待つということで「>」が表示されます。

なな: 永遠に続くの？

先生: 終了するには、Control キーを押しながら d を入力します。こういうのを「ctrl-D を入力する」と言います。Node.js のバージョンによっては、ctrl-C を入力、ctrl-C を 2 回入力することで終了します。

なな: 毎回、プログラムをキーボードから入力するの？

先生: 下記のような内容を、エディタを使って、たとえば、sample1.js という名前のファイルに書き込んでおいて、

```
console.log("Hello, world!");
```

下記のように実行することもできます。

```
% node sample1.js
Hello, world!
%
```

なな: console.log は、ブラウザの JavaScript でも、コンソールに出力するために使ったわね。

先生: 「process.stdout.write();」も使えるけど、「window.alert()」や「document.write()」は Node.js では使えません。「process.stdout.write()」は、出力したあとに改行をしなけど、「console.log()」は改行をします。「process.stdout.write(x);」では、データ (x) が文字列でないとエラーになるけど、「console.log(x);」のデータは、文字列でなくても、文字列に自動変換されます。たとえば、「process.stdout.write(5);」はエラーになります。「process.stdout.write(String(5));」や、「process.stdout.write(5 + "");」にする必要があります。「console.log(5);」は、エラーにならずに「5 (改行)」と表示されます。単にメッセージを出力するだけなら「console.log()」を使用したほうが単純。です。「0、5、10、15」を縦に並べるなら、下記のようにします。

```
for (var i=0; i<20; i=i+5) {
   process.stdout.write(i + "¥n");
}
```

```
for (var i=0; i<20; i=i+5) {
   console.log(i);
}
```

なな: ラジャー！

第3回 入出力

なな: 表示は、console.log() や process.stdout.write() でできることが分かったけど、キーボード入力は？

先生: ちょっと複雑になってしまいます。下記のプログラムは、入力したものを画面にオウム返しするプログラム。

```
1   function getString(data) {
2     process.stdout.write("Input string: " + data);
3   }
4
5   function end() {
6     process.stdout.write("end¥n");
7   }
8
9   process.stdin.resume();
10  process.stdin.setEncoding("utf8");
11  process.stdin.on("data", getString);
12  process.stdin.on("end", end);
13  process.on("SIGINT", function () {
14    end(); process.exit();
15  });
```

1〜3行目で getString() 関数を、5〜7行目で end() 関数を定義しています。

9行目は標準入力からの入力を開始する、という意味。
10行目は、入力の文字コードが utf-8 であることを示しています。
11行目は、入力があったときに、getString() 関数を呼び出すことを登録しています。
12行目は、入力が終了した（最後まで読み込み終わった）時にend() 関数を呼び出すことを登録しています。
stdin（標準入力）は、node コマンドで Node.js を起動する際に、キーボードの代わりにファイルにつなぎ替えること（リダイレクション）が可能です(*)。その場合は、ファイルの末尾まで読んだら終了。キーボードの場合は、control-D (control キーを押しながら d を入力：入力の終了を示す) を入力すると終了します。MS Windows 環境では、control-D 入力を process.stdin.on(“end”,…) で捉えられない場合があります。そのような場合は、control-C 入力を入力し、process.on(“SIGINT”,…) で捉えるようにしています。

なな: リダイレクションではない、普通のファイルの入出力はどうすれば良いの？

先生: 下記のようになります。memo.txt ファイルを入力して、画面に表示するものです。

```
function response(err, txt) {
        if (err) {
            console.log("File not found.");
        } else {
            console.log(txt);
        }
}

fs = require("fs");
fs.readFile("memo.txt", "utf-8", response);
```

* リダイレクションの例：

```
% node input.js < memo.txt
 ここにファイルの内容が表示
%
```

先生: ファイルへの書き込みは下記のようになります。

```
fs = require("fs");
fs.writeFileSync('memo.txt', "test");
```

先生: ファイルを読んで、画面に表示する、別のプログラムです。

```
fs = require("fs");
console.log(fs.readFileSync('memo.txt', 'utf8'));
```

第4回　モジュールとパッケージ

なな：モジュールって？

先生：別のファイルに書いてある関数を利用する仕組みです。たとえば、foo.js というファイルの内容が

```
exports.printFoo = function(){ return "foo" }
```

の場合、別のファイルのプログラムから、このように呼び出すことができます。

```
var foo = require('./foo');
console.log(foo. printFoo());
```

なな：パッケージって？

先生：モジュールや、コマンドをまとめ、検索に必要な情報を付け加えたものです。「npm」 (Node package manager) という、Node.js のモジュールを管理するためのツール（便利プログラム）があります。 Node.js にはさまざまなモジュールがインターネット上に公開されていて、npmを使うことで簡単に検索、インストールができます。現在では、Node.js をインストールすると npm も一緒にインストールされるようになっています。次のように npm コマンドが実行できれば使える状態になっています。「1.3.2」は npm のバージョンで、別の数字でもOKよ。

```
% npm -v
1.3.2
```

モジュールをインストールするには、下記のように入力します。

```
% npm install モジュール名
```

「-g」オプションを付けるとグローバルインストールとなって、 npm のインストール場所にパッケージをインストールすます。パスが通ってモジュールのコマンドを実行できるようになります。

```
% npm install -g mocha
% mocha -h
```

「-g」を付けない場合は，カレントディレクトリの node_modules 不フォルダ内にインストールされます。個別のアプリケーションでしか利用しないライブラリモジュールなどはオプションなしでインストールします。

```
% npm install mocha
% ./mocha -h
```

npmは，インストール対象のパッケージが依存している（必要としている別の）ライブラリも，まとめてインストールしてくれます。また、パッケージに含まれるモジュール内の関数を、require を使って利用することもできます。

現在インストールしているパッケージの一覧を見たいときは，listコマンドで行います。

```
% npm list [-g]
```

アンインストールしたいときは，uninstall コマンドを使います。

```
% npm uninstall [-g] ＜パッケージ名＞
```

いずれも「-g」オプションでグローバルかカレントディレクトリかの切り替えができます。

なな：いろいろ試してみま～～す！

第5回 ウェブサーバも作れる！

なな: Node.js でウェブサーバを作れるって聞いたけど。

先生: はい。Node.js には簡単に Webサーバを実現する機能が備わっています。次のプログラムは、Node.js を使って作った簡単な Webサーバの例です。

```
1  function response(req, res) {
2     res.writeHead(200, {"Content-type": "text/plain"});
3     res.write('Hello World!¥n');
4     res.end();
5  }
6
7  var http = require("http");
8  var server = http.createServer();
9  server.on("request", response);
10 server.listen(1234);
11 console.log('Server running at http://' +
12                   require("os").hostname() + ':1234/');
```

1行目～5行目で、Webサーバにアクセスがあった場合の処理を response() 関数として定義しています。それ以下の行はサーバのセットアップ（準備や設定）です。

7行目で、http パッケージを呼び出し、http の機能（Webサーバ機能）を利用できるようにしています。
8行目で、サーバオブジェクトを生成しています。このとき、Webサーバが作られます。
9行目で、生成したサーバにアクセスがあったときの処理を定義しています。HTTP Request を受け取ると、response() 関数が呼び出されるように設定しています。
10行目で、サーバとしての待ち受けを始めています。TCP の 1234番ポート(*) で待ち受けます。
11行目で、サーバがスタートしたというメッセージを表示しています。
「res.write('Hello World!¥n');」を「res.write('Hello World!' + req.url + '¥n');」に変えると、「http://～:1234/abc」という URL でアクセスした場合、「/abc」の部分が req.url に設定され、ブラウザ画面に「Hello World!/abc」と表示されるようになります。

サーバオブジェクトが動いている限りプログラムは終了しません。プログラムを終了するためには、「Control-C」を入力します。ここで作ったWebサーバに、同じ計算機のブラウザからアクセスする際の URL は、「http://localhost:1234/」、または「http://127.0.0.1:1234/」になります。ブラウザに「Hello World!」と表示されたら成功です。

response() 関数の1行目は関数の定義です。仮引数 req と res があります。
req には、Webブラウザからの HTTP Request に入っている情報が格納されます。
res は HTTP Request に対する応答をするためのインターフェイスです。
2行目～4行目は応答処理です。
2行目は HTTP のヘッダ部分に対する指示で、200 は HTTP Request が正しく行われたという意味。
「{"Content-type": "text/plain"}」は、HTTP ヘッダ。ここでは、plain text を返すことを宣言しています。
3行目で、「Hello World!」という文字列を Webブラウザに返しています。
4行目で、応答を終了しています。

* ポート番号: TCP/IPでは、IPアドレスで通信相手のコンピュータを、そして、アプリケーション識別番号（ポート番号）で そのコンピュータ上で動いている複数のプログラムのひとつを指定します。IPアドレスを建物の住所にたとえるなら、ポート番号は部屋番号に相当します。指定できる番号の範囲はTCPやUDPなどの通信の種類毎にそれぞれ 0 から 65535（16ビット）と定められています。0～1023番のポートは、用途が決められています(予約ポート)。今回のように、実験的、あるいは、一時的に使う場合は、1024～65535番を使います。

さきほどの URL の、「localhost」は、「自分自身」という意味です。サーバとクライアント（ブラウザ）が同一計算機上で動作している場合に用いる省略記法です。

下記のプログラムで、自分の計算機の IP アドレスを調べることができます。

```
console.log(getLocalAddress()[0]);

function getLocalAddress() {
	var addr = [];
	var interfaces = require('os').networkInterfaces();
	for (var dev in interfaces) {
		interfaces[dev].forEach(function(details){
			if ((!details.internal) && (details.family == "IPv4"))
				addr.push(details.address);
		});
	}
	return addr;
};
```

調べた IP アドレス（例えば、192.168.11.2）を使って、「http://192.168.11.2:1234/」 のように指定することもできます。このような指定なら、別の計算機からのアクセスも可能です。

なな: わかりました。いろいろ実験してみようと思います。

追加：もっと単純なウェブサーバ

```
var http = require("http");
var server = http.createServer(function(req, res) {
   res.writeHead(200, {"Content-type": "text/html"});
   res.end('<h1>Hello world!</h1>');
});
server.listen(1234);
console.log('Server running at http://' +
			require("os").hostname() + ':1234/');
```

第6回 外部サービスの呼び出し

なな: 外部ウェブサイトへのアクセスはできるの？

先生: はい。下記は、JSONP データを送り返す、郵便番号検索サービスにアクセスする例です。

```
var http = require("http");
var zip = process.argv[2];
http.get({
    host: 'geoapi.heartrails.com',
    port: 80,
    path: '/api/json?method=searchByPostal&postal=' + zip + '&jsonp=callbackf'
}, function(res) {
    res.setEncoding('utf-8');
    res.on('data', function(chunk) {
        eval(chunk);
    });
}).on('error', function(e) {
    console.log('ERROR',e.message);
});

function callbackf(data) {
    console.log(data.response.location[0].prefecture +
        data.response.location[0].city +
        data.response.location[0].town);
}
```

実行結果です。

```
% node http.js 2530025
神奈川県茅ケ崎市松が丘一丁目

%
```

なな: すごい！

第7回 デバッグ支援機能

なな: デバッグ支援機能って？

先生: 具体例で説明しましょう。下記のような内容を sample.js というファイルに書き込みます。

```
var x = 1;
x++;
x++;
console.log(x);
```

下記がデバッグの様子です。

```
C:¥Users¥User¥Desktop¥nd>node debug sample.js
< Debugger listening on [::]:5858
connecting to 127.0.0.1:5858 ... ok
break in C:¥Users¥User¥Desktop¥nd¥sample.js:1
> 1 var x = 1;
  2 x++;
  3 x++;
debug> watch("x")
debug> n
break in C:¥Users¥User¥Desktop¥nd¥sample.js:2
Watchers:
  0: x = 1

  1 var x = 1;
> 2 x++;
  3 x++;
  4 console.log(x);
debug> n
break in C:¥Users¥User¥Desktop¥nd¥sample.js:3
Watchers:
  0: x = 2

  1 var x = 1;
  2 x++;
> 3 x++;
  4 console.log(x);
  5
debug> n
break in C:¥Users¥User¥Desktop¥nd¥sample.js:4
Watchers:
  0: x = 3

  2 x++;
  3 x++;
> 4 console.log(x);
  5
  6 });
debug> n
< 3
break in C:¥Users¥User¥Desktop¥nd¥sample.js:6
Watchers:
  0: x = 3

  4 console.log(x);
  5
> 6 });
debug>
C:¥Users¥User¥Desktop¥nd>
```

先生: こんどは、プログラムの中にブレークポイントを入れておく例。debugtest.js の内容。

```
var x = 1;
debugger;
x++;
x++;
console.log(x);
```

ブレークポイントを設定（debugger;）

下記が、デバッグの様子です。

```
C:¥Users¥User¥Desktop¥nodeJs¥node.js >node debugtest.js
3

C:¥Users¥User¥Desktop¥nodeJs¥node.js >node debug debugtest.js
< Debugger listening on port 5858
debug> . ok
break in C:¥Users¥User¥Desktop¥nodeJs¥node.js¥debugtest.js:1
> 1 var x = 1;
  2 debugger;
  3 x++;
debug> n
break in C:¥Users¥User¥Desktop¥nodeJs¥node.js¥debugtest.js:2
  1 var x = 1;
> 2 debugger;
  3 x++;
  4 x++;
debug> n
break in C:¥Users¥User¥Desktop¥nodeJs¥node.js¥debugtest.js:3
  1 var x = 1;
  2 debugger;
> 3 x++;
  4 x++;
  5 console.log(x);
debug> repl
Press Ctrl + C to leave debug repl
> x
1
debug> n
break in C:¥Users¥User¥Desktop¥nodeJs¥node.js¥debugtest.js:4
  2 debugger;
  3 x++;
> 4 x++;
  5 console.log(x);
  6
debug> repl
Press Ctrl + C to leave debug repl
> x
2
debug>
(^C again to quit)
debug>

C:¥Users¥User¥Desktop¥nodeJs¥node.js >
```

吹き出し:
- 普通に実行すると、debugger 文は無視される（node debugtest.js）
- デバッグモードで実行（node debug debugtest.js）
- ブレークポイントの手前で一時停止（> 1 var x = 1;）
- 「n」で 1 行実行（debug> n）
- 「repl」で 対話モードに（debug> repl）
- 「^C」で デバッグモードに
- 「^C ^C」で 終了

なな: 雰囲気は分かりました！

先生: デバッグモードで使えるコマンドをまとめておきます。

コマンド名	説明
cont、c	実行を再開する
next、n	次の命令を実行して一時停止する(ステップ実行)
step、s	次の関数の先頭で一時停止する(ステップイン)
out、o	実行中関数から抜けるまで実行して一時停止する(ステップアウト)
pause	コードの実行を一時停止する
setBreakpoint()、sb()	ブレークポイントを設定する
clearBreakpoint()、cb()	ブレークポイントを解除する
backtrace、bt	バックトレースを表示する
list(number)	次に実行する命令の前後のソースコードを number 引数で指定した行数ずつ表示する
watch(expr)	expr 引数で指定した式をウォッチする
unwatch(expr)	expr 引数で指定した式のウォッチを解除する
watcher	ウォッチしている式とその値を一覧表示する
repl	repl を開始する
run	プログラムの実行を開始する
restart	プログラムを先頭から再実行する
kill	スクリプトの実行を強制終了(KILL)する
scripts	ロードされているスクリプトを一覧表示する
version	V8(JavaScript 実行エンジン)のバージョンを表示する

なな: repl って何?

先生: Read-Eval-Print-Loop の頭文字です。Node.js の対話モードということです。

なな: わかりました。

第8回 Buffer クラス

なな: Buffer クラスって、何に使うの？

先生: 最初に使用例を見てね。ファイルの内容を 16 進数で表示するプログラムです。

```
require("fs").readFile(process.argv[2],function(err,data) { // コマンド引数＝ファイル名
    var buf = Buffer(data), subbuf = new Buffer(16);         // ファイル内容の文字列→バッファ
    for (var i=0; i<buf.length; i+=16) {                     // 16 バイトブロックを取り出す
        var o = i.toString(16);                              // 16 バイトブロックのアドレスを 16 進に
        while (o.length < 5) o = " " + o;                    // 5 桁に位置揃え
        o = o + ": ";                                        // アドレスの次に : を表示
        for (var j=i; j<i+16; j++) {                         // ブロックから 1 バイト取り出し
            if (j<buf.length) {                              // データが存在すれば
                var oo = buf[j].toString(16);                // データを 16 進数の文字列に
                if (oo.length == 1) oo = "0" + oo;           // 1 桁なら 前に 0 を付加
                o += oo + " ";                               // 表示データに追加し、スペースを付加
        } }
        buf.slice(i,i+16).copy(subbuf);                      // ブロックデータを再度コピー
        for (var k=0; k<16; k++) if (subbuf[k]<20) subbuf[k] = 46; // 非表示文字を「.」に取り換え
        console.log(o + " " + subbuf.toString("ascii"));     // ASCII 文字列として追加
    }
```

コマンドプロンプト

```
C:¥Users¥User¥Desktop¥nodeJs¥node.js>node hexdump.js hexdump.js
    0: ef bb bf 72 65 71 75 69 72 65 28 22 66 73 22 29   o:?require("fs")
   10: 2e 72 65 61 64 46 69 6c 65 28 70 72 6f 63 65 73   .readFile(proces
   20: 73 2e 61 72 67 76 5b 32 5d 2c 66 75 6e 63 74 69   s.argv[2],functi
   30: 6f 6e 28 65 72 72 2c 64 61 74 61 29 7b 0d 0a 20   on(err,data){..
   40: 20 20 76 61 72 20 62 75 66 20 3d 20 42 75 66 66     var buf = Buff
   50: 65 72 28 64 61 74 61 29 2c 20 73 75 62 62 75 66   er(data), subbuf
   60: 20 3d 20 6e 65 77 20 42 75 66 66 65 72 28 31 36    = new Buffer(16
   70: 29 3b 0d 0a 20 20 20 66 6f 72 20 28 76 61 72 20   );..   for (var
   80: 69 3d 30 3b 20 69 3c 62 75 66 2e 6c 65 6e 67 74   i=0; i<buf.lengt
   90: 68 3b 20 69 2b 3d 31 36 29 20 7b 0d 0a 20 20 20   h; i+=16) {..
   a0: 20 20 20 76 61 72 20 6f 20 3d 20 69 2e 74 6f 53      var o = i.toS
   b0: 74 72 69 6e 67 28 31 36 29 3b 0d 0a 20 20 20 20   tring(16);..
   c0: 20 20 77 68 69 6c 65 20 28 6f 2e 6c 65 6e 67 74     while (o.lengt
   d0: 68 20 3c 20 35 29 20 6f 20 3d 20 22 20 22 20 2b   h < 5) o = " " +
   e0: 20 6f 3b 0d 0a 20 20 20 20 20 20 6f 20 3d 20 6f    o;..      o = o
   f0: 20 2b 20 22 3a 20 22 3b 0d 0a 20 20 20 20 20 20    + ": ";..
  100: 66 6f 72 20 28 76 61 72 20 6a 3d 69 3b 20 6a 3c   for (var j=i; j<
  110: 69 2b 31 36 3b 20 6a 2b 2b 29 20 7b 0d 0a 20 20   i+16; j++) {..
  120: 20 20 20 20 20 20 20 69 66 20 28 6a 3c 62 75 66          if (j<buf
  130: 2e 6c 65 6e 67 74 68 29 20 7b 0d 0a 20 20 20 20   .length) {..
  140: 20 20 20 20 20 20 20 20 20 76 61 72 20 6f 6f 20            var oo
  150: 3d 20 62 75 66 5b 6a 5d 2e 74 6f 53 74 72 69 6e   = buf[j].toStrin
  160: 67 28 31 36 29 3b 0d 0a 20 20 20 20 20 20 20 20   g(16);..
  170: 20 20 20 20 20 69 66 20 28 6f 6f 2e 6c 65 6e 67        if (oo.leng
```

なな: 文字列データを、1 バイトずつ、16 進数で表示するのに使えるということね。

先生: まず、Buffer クラスのデータの作成から。

```
var buf = new Buffer(128);       // Buffer データをつくる。内容の初期値はゼロでなく、ランダム値。
buf.fill(0);                     // バッファに0を書き込む
buf.fill(1, 10);                 // バッファの10バイト目から最後までに1を書き込む
buf.fill(2, 20, 30);             // バッファの20バイト目から30バイト目までに2を書き込む
var buf = new Buffer([0, 1, 2]); // 配列からバッファへの変換
var buf = new Buffer(str, [encoding]); // 文字列からバッファへの変換
```

encoding: エンコーディングを文字列で指定。省略された場合「utf8」を使用。

ascii	ASCII
utf8	UTF-8
utf16le	リトルエンディアンUTF-16（UTF-16LE）
ucs2	utf16le と同じ
base64	BASE64

なな: 最初のプログラムは、ファイルのデータが、自動的に UTF-8 コードとして解釈されて文字列になっているので、これを、UTF-8 コードで Buffer クラスのデータにすれば、ファイルのデータに逆戻りするということね。
Buffer クラスのデータって、文字列データに似ているみたいね。

先生: 似ているけど異なる点で重要なのは、Stringクラスのオブジェクトはimmutable（文字単位の書き換え不可）。
「var str = 'あいうえお'; str[0] = 'か';」 では、「あ」 は書き換わりません。
Bufferクラスのオブジェクトはmutable（バイト単位の書き換え可）。
「var buf = new Buffer('あいうえお'); buf[0] = 0;」 で、「あ」の先頭バイトが 「0」 に書き換わります。
Stringクラスには、検索メソッドの indexOf、match、search、置換 replace、部分文字列取り出し substring があるけど、Bufferクラスではこのようなメソッドは用意されていません。でも、指定した位置のデータを取り出す slice メソッドは両方にあります。要注意なのは、Bufferクラスの slice メソッドで取り出したバッファを変更すると、元のバッファも変更されること。sliceメソッドではバイナリデータを格納しているメモリ領域をコピーせずに共有するからです。

なな: Bufferクラスのデータと、Stringクラスのデータの相互変換は？

先生: Bufferクラスに格納したデータ → 文字列データの変換は、
var str = buf.toString([encoding], [start], [end]);
文字列データ → Bufferクラスデータの変換は、Bufferクラスのコンストラクタに引数として文字列を与えて新たなBuffer型オブジェクトを作成すればよいです。既存のBufferオブジェクトに文字列を書き込みたい場合はwriteメソッドを使用します。
buf.write(string, [offset], [length], [encoding]);
offset、length のデフォルト値は 0、buffer.length − offset です。

なな: Bufferクラスのデータを 16 進表示するには？

先生: バッファのデータ全体をまとめて 16 進表示するには console.log(buf.toString("hex"));、
データを 1 バイトずつ取り出して 16 進表示するには、console.log(buf[i].toString(16));

なな: その他は？

先生: Bufferクラスデータのコピーは、
buf.copy(targetBuffer, [targetStart], [sourceStart], [sourceEnd])
targetBuffer引数には書き込み先となるBuffer型オブジェクトを指定。
targetStart、sourceStart のデフォルト値は0、sourceEnd のデフォルト値は buffer.length。
オブジェクトがBuffer型のオブジェクトかどうかの判別は、Buffer.isBuffer(obj)、
文字列のバイト長を計算するメソッドは、Buffer.byteLength(string, [encoding])、
複数のバッファを結合して新たなバッファを作成するメソッドは、Buffer.concat(list, [totalLength])
list: 結合する Buffer 型オブジェクト群を格納した配列。
totalLength: 結合結果を格納するBuffer型オブジェクトのサイズ。
指定されなかった場合、list引数で指定されたBuffer型オブジェクトのサイズの合計値。

なな: 先生、お疲れ様！