Node.js and WebSocket

Ver. 1.10

Datagram Ltd.

2014 年 12 月 2 日

概 要

CentOS 上に Node 環境を構築する方法を記述しています。そして簡単なチャットプログラムを配置します。また、PHP アプリケーションとの連携についても言及します。

改版履歴

Ver. 1.00 2014 年 12 月 2 日
新規作成

Ver. 1.10 2014 年 12 月 2 日
PHP アプリケーションとの連携

目 次

1 概要

Node.js はサーバ側の JavaScript です。JavaScript のオブジェクトは非同期で動くことができますが、シングルスレッドで実現されています。

1.1 手順

構築手順は次の通りです。

1. CentOS のインストール
2. Node.js と npm のインストール
3. express と socket.io のインストール

2 用意するもの

VM 環境を使用する前提[1]でお話しします。VM が使用できる 64bit の PC で OS は Windows7/8 です。

2.1 OS とハードウェア

OS は Windows7 または Windows8 です。PC（ハードウェア）は、“Intel-VT / Intel Virtualization Technology” 対応のもので VM（仮想マシン）が動作します。BIOS で Inetel-VT が有効であることを確認する必要があります。

2.2 ソフトウェア

VM とターゲット OS 関連のファイルをインターネット上からダウンロードしておきます。もしも PC が 32 ビットマシンであったならば 32bit 用の CentOS の iso を用意する必要があります。PC のコアメモリは 4GB 以上であれば問題ありません。また、ハードディスクの空き容量は 20GB 程度で OK です。

- VM（仮想マシン）: VMware-player-6.0.3-1895310.exe
- ターゲット OS：CentOS-6.5-x86_64-bin-DVD1.iso, CentOS-6.5-x86_64-bin-DVD2.iso

Node が動作する環境を持つことが目的ですので、多少のバージョンの違いは問題ありません。

3 CentOS のインストール

3.1 VM に CentOS をインストール

1. VMware-player-6.0.3-1895310.exe の実行。
2. デフォルトの設定でセットアップウィザードを完了させます。
3. VMware Player を起動して「新規仮想マシンの作成（N）」をクリック。
4. インストーラディスクイメージファイルに CentOS-6.5-x86_64-bin-DVD1.iso を指定し［次へ］。
5. 名前とユーザ名とパスワードを英数字で指定し、［次へ］。
6. 仮想マシン名（web また man）を指定し、［次へ］。
7. 後はデフォルトの設定で CentOS がインストールされます。
8. 自動的に再起動されます。

[1] しかし、VM ではなく PC に直接 OS をインストールする方がベターです。

3.2 OS の基本設定

1. CentOS インストール後にログイン画面になりますので、Other を指定し、root でログインします。
2. ここで root ログインに対するアラートが表示されるかもしれませんが、続行します。
3. GUI の左上のメニューから「System」＞「Preferences」＞「Keyboard」を選択します。
4. Layout で Japan/Japanese を追加（Add）します。
5. Default のラジオボタンを Japanese に設定し、Close します。
6. 「Applications」＞「System Tools」＞「Terminal」で端末ツールを起動。
7. 「vi /etc/selinux/config」で「SELINUX=disabled」に変更。
8. 「vi /etc/inittab」で「id:3:initdefault:」に変更。
9. 「ifconfig」で VM の IP アドレスを確認しておきます。[2]
10. 「shutdown -r now」でリブート。

3.3 サービスポートの公開

```
# vi /etc/sysconfig/iptables
<<< -A INPUT ... --dport 22 ... の直下に追加
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 3000 -j ACCEPT
>>>
# /etc/init.d/iptables restart
```

3.4 その他のインストール

一般的に以下のソフトウェアをインストールしておいた方がベターです。

```
# yum -y install make
# yum -y install man-pages
# yum -y install man
# yum -y install mailx
# yum -y install openssh-clients
# yum -y install bind-utils
# yum -y install nkf
# yum -y install wget
# yum -y install lynx
# yum -y install emacs
```

4 Node.js と npm のインストール

npm[3]は Node に必要不可欠です。

```
# rpm -ivh http://ftp.riken.jp/Linux/fedora/epel/6/x86_64/epel-release-6-8.noarch.rpm
# yum -y install nodejs npm --enablerepo=epel
```

5 express と socket.io のインストール

5.1 express のインストールと配置の用意

当資料では express 上で動く WebSocket を紹介します。myapp というディレクトリを用意しそこにサンプルプログラムを配置します。

```
# mkdir -p ~/dev/sample01
# cd ~/dev/sample01
# npm install -g express-generator
# express --css stylus myapp
# cd myapp
# npm install
```

[2]VM の IP アドレスは、後で使用しますので重要な情報です。
[3]Node Packaged Modules

ここまで来たら Node の動作確認ができます。サービスポート 3000 で動きます。

```
# DEBUG=myapp ./bin/www
```

http://VM の IP アドレス:3000

URL の確認が終わったら Ctrl/C で Node を中止します。

5.2 socket.io のインストール

WebSocket を実現するために、socket.io をインストールします。

```
# npm install socket.io
```

6 サンプルプログラム

簡単なチャットプログラムを配置します。

6.1 app.js

```
# cd ~/dev/sample01/myapp
# cp app.js app.js.org
# vi app.js
```

~/dev/sample01/myapp/app.js の内容は次の通りです。

```
 1  var express = require('express');
 2  var path = require('path');
 3  var favicon = require('serve-favicon');
 4  var logger = require('morgan');
 5  var cookieParser = require('cookie-parser');
 6  var bodyParser = require('body-parser');
 7
 8  var routes = require('./routes/index');
 9  var users = require('./routes/users');
10
11  var app = express();
12
13  // view engine setup
14  app.set('views', path.join(__dirname, 'views'));
15  app.set('view engine', 'jade');
16
17  // uncomment after placing your favicon in /public
18  //app.use(favicon(__dirname + '/public/favicon.ico'));
19  app.use(logger('dev'));
20  app.use(bodyParser.json());
21  app.use(bodyParser.urlencoded({ extended: false }));
22  app.use(cookieParser());
23  app.use(require('stylus').middleware(path.join(__dirname, 'public')));
24  app.use(express.static(path.join(__dirname, 'public')));
25
26  app.use('/', routes);
27  app.use('/users', users);
28
29  // catch 404 and forward to error handler
30  app.use(function(req, res, next) {
31      var err = new Error('Not Found');
32      err.status = 404;
33      next(err);
34  });
35
36  // error handlers
37
38  // development error handler
```

```
39  // will print stacktrace
40  if (app.get('env') === 'development') {
41      app.use(function(err, req, res, next) {
42          res.status(err.status || 500);
43          res.render('error', {
44              message: err.message,
45              error: err
46          });
47      });
48  }
49
50  // production error handler
51  // no stacktraces leaked to user
52  app.use(function(err, req, res, next) {
53      res.status(err.status || 500);
54      res.render('error', {
55          message: err.message,
56          error: {}
57      });
58  });
59
60
61  module.exports = app;
```

6.2 Node 起動部分

```
# cd ~/dev/sample01/myapp/bin
# cp www www.org
# vi www
```

~/dev/sample01/myapp/bin/www の内容は次の通りです。

```
1   #!/usr/bin/env node
2   var debug = require('debug')('myapp');
3   var app = require('../app');
4
5   app.set('port', 80);
6
7   var server = app.listen(app.get('port'), function() {
8     debug('Express server listening on port ' + server.address().port);
9   });
10
11  var socketIO = require('socket.io');
12  // クライアントの接続を待つ (IP アドレスとポート番号を結びつけます)
13  var io = socketIO.listen(server);
14
15  // クライアントが接続してきたときの処理
16  io.sockets.on('connection', function(socket) {
17
18    // メッセージを受けたときの処理
19    socket.on('message', function(data) {
20      // つながっているクライアント全員に送信
21      console.log("message");
22      io.sockets.emit('message', { value: data.value });
23    });
24
25    // クライアントが切断したときの処理
26    socket.on('disconnect', function(){
27      console.log("disconnect");
28    });
29  });
```

6.3 view/index

6.4 view/index.jade

```
# cd ~/dev/sample01/myapp/view
# cp index.jade index.jade.org
# vi index.jade
```

~/dev/sample01/myapp/views/index.jade の内容は次の通りです。

```
1  extends layout
2
3  block content
4    include _index.js
5    include _index.html
```

6.5 view/_index.js

```
# cd ~/dev/sample01/myapp/view
# vi _index.js
```

~/dev/sample01/myapp/views/_index.js の内容は次の通りです。

```
1  <script src="/socket.io/socket.io.js"></script>
2  <script type="text/javascript">
3
4  var socket = io.connect('http://VM の IP アドレス');
5
6  socket.on('connect', function() {
7    console.log("connet");
8    document.getElementById("status").innerHTML = 'connected';
9  });
10
11 // メッセージを受けたとき
12 socket.on('message', function(msg) {
13   // メッセージを画面に表示する
14   document.getElementById("receiveMsg").innerHTML = msg.value;
15 });
16
17 // メッセージを送る
18 function SendMsg() {
19
20   var msg = document.getElementById("message").value;
21
22   // メッセージを発射する
23   socket.emit('message', { value: msg });
24 }
25
26 // 切断する
27 function DisConnect() {
28
29   var msg = ' 切断されました。';
30
31   // メッセージを発射する
32   socket.emit('message', { value: msg });
33
34   // socket を切断する
35   socket.disconnect();
36
37   document.getElementById("status").innerHTML = msg;
38 }
39 </script>
```

6.6 view/_index.html

```
# cd ~/dev/sample01/myapp/view
# vi _index.html
```

~/dev/sample01/myapp/views/_index.html の内容は次の通りです。

```
1    <h1>socket.io のサンプルプログラム</h1>
2    <div id="status">Not connected</div>
3    <div id="connectId"></div>
4    <div id="type"></div>
5    <br>
6    <input type="text" id="message" value="">
7    <input type="button" value="メッセージを送る" onclick="SendMsg()">
8    <input type="button" value="切断する" onclick="DisConnect()">
9    <div id="receiveMsg"></div>
```

6.7 socket.io.js の用意

```
# cd ~/dev/sample01/myapp/public
# mkdir socket.io
# cd socket.io
# ln -s /root/dev/sample01/myapp/node_modules/socket.io/node_modules/\
socket.io-client/socket.io.js socket.io.js
```

6.8 サンプルプログラム起動

```
# cd ~/dev/sample01/myapp
# DEBUG=myapp ./bin/www
```

http://VM の IP アドレス

複数の WEB ブラウザから上記 URL を指定します。あるひとつの WEB ブラウザからの送信メッセージが全ての WEB ブラウザ上に表示されます。URL の確認が終わったら Ctrl/C で Node を中止します。

7 PHP アプリケーションとの連携

PHP アプリケーションとこのチャットのサンプルプログラムの連動を考えてみたいと思います。PHP のセッションデータは、ファイルや DB に保存されています。ログイン成功時にクッキーに PHP セッション ID が書かれます。それ以降、このセッション ID でセッションデータを使用することができます。

7.1 Node アプリケーションで PHP のセッションデータを利用する

Node アプリケーションで PHP のセッションデータを利用するには、少し工夫が必要です。明示的にセッション ID を渡す処理には問題があります。セキュリティ的に脆弱になります。

7.2 安全にセッション ID を渡す手順

PHP アプリケーションでデータを暗号化し、Node アプリケーションで復号化するという方式をとります。

1. PHP 側：リモート IP アドレス、セッション ID そしてログイン ID を暗号化して WEB クライアント側の JavaScript が使用できるようにしておく。
2. Node 側：処理を行う前にデータを復号化し、その内容をチェックし、OK であれば処理を進める。

リモート IP アドレスは Node 側でも個別に取得できますので、その値が同じか否かを判断できます。この IP アドレスが異なる場合、エラーとします。通常、セッションデータの中にログイン ID を識別できるものが入っていますので、このチェックも可能です。セッション ID とログイン ID は必ずペアになっています。

また、暗号化されたデータを使いまわすという手口も考えられますので、Node 側ではクッキーの値を参照し、最新のセッション ID であるかをチェックします。正しいセッション ID であると判断された場合、そのセッションデータを使います。この暗号化されたデータは。“ワンタイムトークン” とみることもできます。この暗号化されたデータは、必ずリモート IP アドレスとセッション ID に紐づいています。

7.3 暗号化手順

PHP 側と Node 側は、シェル経由で openssl を使用できます。

7.3.1 準備

RSA 秘密鍵・公開鍵方式を用います。秘密鍵（private.pem）や公開鍵（public.pem）は、予め生成しておきます。

```
# openssl genrsa -out private.pem 2048
# openssl rsa -in private.pem -pubout -out public.pem
```

7.3.2 暗号化

平文（file_hira.txt）を暗号化（file_enc.dat）します。公開鍵を使います。

```
# echo abc1234 > file_hira.txt
# cat file_hira.txt
abc1234
# openssl rsautl -encrypt -inkey public.pem -pubin -in file_hira.txt -out file_enc.dat
```

7.3.3 復号化

暗号化されたファイル（file_enc.dat）を復号化（file_dec.dat）します。秘密鍵を使います。

```
# openssl rsautl -decrypt -inkey private.pem -in file_enc.dat -out file_dec.dat
# cat file_dec.dat
abc1234
#
```