
nodeJS

V8 JavaScript を用いたイベント駆動I/Oの実現

全てのリクエストに対して"Hello World"と返答するNodeで書かれたWEBサーバの例です。

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

このサーバを走らせるには、コードを `example.js`というファイル名で保存し、下記のように `node`コマンドを実行してください。

```
% node example.js
Server running at http://127.0.0.1:8124/
```

また、ポート番号8124をリッスンし、何でもエコーするシンプルなTCPサーバは以下のようになります。

```
var net = require('net');
net.createServer(function (socket) {
  socket.setEncoding("utf8");
  socket.write("Echo server\r\n");
  socket.on("data", function (data) {
    socket.write(data);
  });
  socket.on("end", function () {
    socket.end();
  });
}).listen(8124, "127.0.0.1");
```

APIドキュメントにもっと多くのサンプルがあります。

ダウンロード

gitリポジトリ

2010.09.17 node-v0.2.2.tar.gz

過去のバージョン: 各バージョン, ドキュメント

ビルド方法

NodeはLinuxやMacintosh、 Solaris上でテストされています。Windows/Cygwinや FreeBSD、OpenBSDでも動きます。ビルドには、Python 2.4以降が必要になります。 V8エンジン(Nodeがビルドします)は、IA-32とx64、ARMの各プロセッサのみをサポートしています。 V8エンジンはNodeのディストリビューションに含まれています。 TLSを使用するためにはOpenSSLが必要です。その他に依存関係はありません。

```
./configure
make
make install
```

あとは、APIドキュメントをご覧ください。

テストを走らせるには次のコマンドを実行します。

```
make test
```

概要

Nodeの目標は、簡単にスケーラブルなネットワークプログラムを作成する方法を提供することです。 上記の"Hello World"WEBサーバの例では、多数のクライアントとの接続を同時に扱うことができます。 NodeはOSに、(`epoll`や`kqueue`、`/dev/poll`や`select`を通じて)新しい接続が来たら知らせるように指示し、 スリープ状態になります。そして新しい接続があると、コールバックを実行します。 各接続にはほんの小さなヒープしか割り当てられません。

これは、OSのスレッドを用いる近頃のより一般的な同時実行モデルとは対照的です。 この手法に比べると、スレッドベースのネットワークプログラムは非効率的であり、またとても使いづらいのです。 ここや ここをご覧ください。 Nodeは、1接続ごとに2MBのスタックメモリを割り当てるスレッドベースのシステムに比べて、 高負荷時にとても良いメモリ効率を実現します。 さらに、Nodeのユーザはプロセスのデッドロックの心配から開放されました。Nodeにはロックがないのです。 Nodeには直接I/Oを実行する関数がほとんど無いため、プロセスは

決してブロックされません。 ブロックがないため、エキスパートではないプログラマであっても高速なシステムを開発することができるのです。

Nodeは、RubyのEvent Machine やPythonのTwistedの影響を受けており、 よく似たデザインになっています。 しかし、Nodeはイベントモデルをさらに推し進めています。 すなわち、イベントループをライブラリとして提供するのではなく、言語自体に組み込まれた構成として提供しています。 Node以外のシステムでは、イベントループを開始する際に必ずブロック呼び出しが存在します。 それらのシステムでは、一般的にスクリプトの先頭でコールバックを通じて行う動作を定義し、 スクリプトの最後で`EventMachine::run()` のようなブロック呼び出しを行ってサーバを開始させています。 Nodeでは、そのような「イベントループの開始」呼び出しはありません。 Nodeは入力されたスクリプトを実行し終えるとそのままイベントループに入ります。 そして動作すべきコールバックがなくなるとイベントループから抜け出します。 こういった挙動は、ブラウザ上のJavaSciprtとおんなじです。 すなわち、イベントループはユーザからは隠匿されているのです。

HTTPはNodeにおける第一級のプロトコルです。NodeのHTTPライブラリは 作者のWEBサーバ開発における経験を脱却させました。例えば、 ほとんどのWEBフレームワークでは、データをストリーミングさせることができません。 Nodeはこの問題をHTTPパーサ とAPIによって解決しようとしています。Nodeのイベント駆動な仕組みと連動させれば、 WEBライブラリやフレームワークのとてもよい基盤になることでしょう。

でも、マルチプロセッサ環境ではどうなるんだ？ マルチコアのコンピュータ上でプログラムをスケールさせるためには、スレッドを使う必要があるんじゃないのか？ マルチコアのコンピュータ上でスケールさせるのに必要なのは複数のプロセスであり、 共有メモリを利用するスレッドではありません。 スケーラブルなシステムの基本は高速なネットワーキングとブロックのないデザインであり、 あとはメッセージパッシングがあれば事足ります。 将来のバージョンで、Nodeは現在のデザインととてもよくフィットする形で子プロセスをforkする (Web Workers APIを使って) ことができるようになる予定です。

これらもご覧ください:

- スライド(2009年 JSConf)
- スライド(2010年 JSConf)
- ビデオ(2010年5月 Yahooにて)

リンク

- チャットルームの**デモ**がchat.nodejs.orgで動いています。 そのソースコードはhttp://github.com/ry/node_chatです。 チャットルームは安定してはいないので、たまにダウンしていることもあります。
- 助言を求めたり議論したりするために、http://groups.google.com/group/nodejs にあるメーリングリストに参加してください。 nodejs+subscribe@googlegroups.com にメールすることでも参加できます。リアルタイムで議論に参加したい場合には、 irc.freenode.netの `#node.js`をチェックしてみてください。
- IRCログ
- Node.jsを利用している、またはNode.jsのためのプロジェクトやライブラリ
- Node.jsのビルドボット

貢献するには

パッチは歓迎します。パッチ提出のプロセスはシンプルです:

```
git clone git://github.com/ry/node.git
cd node
(make your changes)
./configure --debug
make test-all #パッチをデバッグとリリースの両方のビルドでチェックしてください
git commit -m "パッチ内容のわかりやすい説明"
git format-patch HEAD^
```

パッチがあなたの氏名と有効なメールアドレスを含んでいることを確認してください。

```
git config --global user.email "ry@tinyclouds.org"
git config --global user.name "Ryan Dahl"
```

あなたのコードが受け入れられるためには、 貢献者使用許諾契約 にサインする必要があります。

パッチの存在を知らせる一番いい方法は、 メーリングリスト に gistsのURLや添付ファイルを添えて投稿することです。

もし新しい機能が欲しい場合には、パッチ作成に取りかかる前にメーリングリストで質問してください。