
nodeJS

Evented I/O for V8 JavaScript.

An example of a web server written in Node which responds with "Hello World" for every request.

```
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/');
```

To run the server, put the code into a file `example.js` and execute it with the `node` program:

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

Here is an example of a simple TCP server which listens on port 8124 and echoes whatever you send it:

```
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");
```

See the API documentation for more examples.

Download

git repo

2010.08.20 node-v0.2.0.tar.gz

Historical: versions, docs

Build

Node is tested on **Linux**, **Macintosh**, and **Solaris**. It also runs on **Windows/Cygwin**, **FreeBSD**, and **OpenBSD**. The build system requires Python 2.4 or better. V8, on which Node is built, supports only IA-32, x64, and ARM processors. V8 is included in the Node distribution. To use TLS, OpenSSL is required. There are no other dependencies.

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

Then have a look at the API documentation.

To run the tests

```
make test
```

About

Node's goal is to provide an easy way to build scalable network programs. In the "hello world" web server example above, many client connections can be handled concurrently. Node tells the operating system (through `epoll`, `kqueue`, `/dev/poll`, or `select`) that it should be notified when a new connection is made, and then it goes to sleep. If someone new connects, then it executes the callback. Each connection is only a small heap allocation.

This is in contrast to today's more common concurrency model where OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. See: this and this. Node will show much better memory efficiency under high-loads than systems which allocate 2mb thread stacks for each connection. Furthermore, users of Node are free from worries of dead-locking the process—there are no locks. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, less-than-expert programmers are able to develop fast systems.

Node is similar in design to and influenced by systems like Ruby's Event Machine or Python's Twisted. Node takes the event model a bit further—it presents the event loop as a language construct instead of as a library. In other systems there is always a blocking call to start the event-loop. Typically one defines behavior through callbacks at the beginning of a

script and at the end starts a server through a blocking call like `EventMachine::run()`. In Node there is no such start-the-event-loop call. Node simply enters the event loop after executing the input script. Node exits the event loop when there are no more callbacks to perform. This behavior is like browser javascript—the event loop is hidden from the user.

HTTP is a first class protocol in Node. Node's HTTP library has grown out of the author's experiences developing and working with web servers. For example, streaming data through most web frameworks is impossible. Node attempts to correct these problems in its HTTP parser and API. Coupled with Node's purely evented infrastructure, it makes a good foundation for web libraries or frameworks.

But what about multiple-processor concurrency? Aren't threads necessary to scale programs to multi-core computers? Processes are necessary to scale to multi-core computers, not memory-sharing threads. The fundamentals of scalable systems are fast networking and non-blocking design—the rest is message passing. In future versions, Node will be able to fork new processes (using the Web Workers API) which fits well into the current design.

See also:

- slides from JSConf 2009
- slides from JSConf 2010
- video from a talk at Yahoo in May 2010

Links

- A chat room **demo** is running at chat.nodejs.org. The source code for the chat room is at http://github.com/ry/node_chat. The chat room is not stable and might occasionally be down.
- For help and discussion, subscribe to the mailing list at http://groups.google.com/group/nodejs or send an email to nodejs+subscribe@googlegroups.com. For real-time discussion, check irc.freenode.net `#node.js`.
- IRC logs
- Projects/libraries which are using/for Node.js
- Node.js buildbot

Contributing

Patches are welcome. The process is simple:

```
git clone git://github.com/ry/node.git
cd node
(make your changes)
./configure --debug
make test-all # Check your patch with both debug and release builds
git commit -m "Good description of what your patch does"
git format-patch HEAD^
```

Be sure the your patch includes your full name and your valid email address. Git can be configured to do this like so:

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

Before your code your code can be accepted you have to sign the contributor license agreement.

The best way for your patch to get noticed is to submit it to the mailing list in form of a gists or file attachement.

You should ask the mailing list if a new feature is wanted before working on a patch.