node.js

A quick tour

by Felix Geisendörfer

Who is talking?

- node.js hacker
- Cofounder of Debuggable
- CakePHP core alumnus

Why Node?

Why?

Node's goal is to provide an easy way to build scalable network programs.

-- nodejs.org

How?

Keep slow operations from blocking other operations.

Traditional I/O

```
var data = file.read('file.txt');
doSomethingWith(data);
```

Something is not right here

Traditional I/O

```
var data = file.read('file.txt');

// zzzZZzzz

doSomethingWith(data);
```

FAIL!

Don’t waste those cycles!

Async I/O

```
file.read('file.txt', function(data) {
  doSomethingWith(data);
});

doSomethingElse();
```

WIN ✓

No need to wait for the disk,
do something else meanwhile!

The Present

Quality components

- V8 (developed for google chrome)
- libev (event loop)
- libeio (non-block posix, thread pool)

CommonJS Modules

hello.js

```
exports.world = function() {
  return 'Hello World';
};
```

main.js

```
var hello = require('./hello');
var sys = require('sys');
sys.puts(hello.world());
```

```
$ node main.js
Hello World
```

Child processes

child.js

```
var child = process.createChildProcess('sh',
['-c', 'echo hello; sleep 1; echo world;']);
child.addListener('data', function (chunk) {
  p(chunk);
});
```

```
$ node child.js
"hello\n"
# 1 sec delay
"world\n"
null
```

Http Server

```
var http = require('http');
http.createServer(function(req, res) {
  setTimeout(function() {
    res.writeHeader(200, {'Content-Type': 'text/plain'});
    res.write('Thanks for waiting!');
    res.close();
  }, 1000);
}).listen(4000);
```

```
$ curl localhost:4000
# 1 sec delay
Thanks for waiting!
```

Tcp Server

```
var tcp = require('tcp');
tcp.createServer(function(socket) {
  socket.addListener('connect', function() {
    socket.write("Hi, How Are You?\n> ");
  });
  socket.addListener('data', function(data) {
    socket.write(data);
  });
}).listen(4000);
```

```
$ nc localhost 4000
Hi, How Are You?
> Great!
Great!
```

DNS

dns.js

```
var dns = require('dns');
dns.resolve4('nodejs.org', function(err, addr, ttl, cname) {
 p(addr, ttl, cname);
});
```

```
$ node dns.js
[ '97.107.132.72' ]
84279
'nodejs.org'
```

Watch File

watch.js

```
process.watchFile(__filename, function() {
  puts('You changed me!');
  process.exit();
});
```

```
$ node watch.js
# edit watch.js
You changed me!
```

ECMAScript 5

- Getters / setters

```
var a = {};
a.__defineGetter__('foo', function() {
  return 'bar';
});
puts(a.foo);
```

- Array: filter, forEach, reduce, etc.

- JSON.stringify(), JSON.parse()

& more [1]

There is only 1 thread

```
file.read('file.txt', function(data) {
  // Will never fire
});

while (true) {
  // this blocks the entire process
}
```

Good for conceptual simplicity
Bad for CPU-bound algorithms

The Future

Web workers

- Multiple node processes that do interprocess communication
- CPU-bound algorithms can run separately
- Multiple CPU cores can be used efficiently

Streams

- Node is working towards a unified data stream interface
- Stream can be readable, writable or both

see [2]

Readable Streams

- events: ‘data’, ‘end’
- methods: pause(), resume()

Writeable Streams

- events: ‘drain’, ‘close’
- methods: write(), close()

Stream Redirection

```
http.createServer(function (req, res) {
  // Open writable file system
  var temp = fs.openTemporaryFile();
  // Pump the request into the temp file.
  stream.pump(req, temp, function (err) {
    if (err) throw err;

    p('sweet!');
  });
});
```

Better Socket Support

- Support for unix sockets, socketpair(), pipe()
- Pass sockets between processes ➠ load balance requests between web workers

Debugger

- V8 support debugging
- Node has a few bugs with exposing the debugger, those need fixing
- Command line node-debug REPL tool

Readline and Curses

- Bindings for JavaScript
- Would allow to build better command line tools
- Goal should be to write a screen clone in node

HTML and XML parsing

- HTML is a major protocol
- Node should be able to parse dirty XML/HTML
- Should be a SAX-style parser in pure JS

Support for Windows

- Patches welcome! :)

Hot code reloading

(maybe)

- Reload module during runtime
- Update code without taking server offline

Suitable Applications

- Web frameworks
- Real time
- Crawlers

More Applications

- Process monitoring
- File uploading
- Streaming

Let’s write a chat

Http Chat in 14 LoC

```
var
  http = require('http'),
  messages = [];

http.createServer(function(req, res) {
  res.writeHeader(200, {'Content-Type' : 'text/plain'});
  if (req.url == '/') {
    res.write(messages.join("\n"));
  } else if (req.url !== '/favicon.ico') {
    messages.push(decodeURIComponent(req.url.substr(1)));
    res.write('ok!');
  }
  res.close();
}).listen(4000);
```

Production ready?

- For small systems, yes.
- Perfect example: Comet server
- Usually few bugs, but API is still changing

Questions?

☝

✎ @felixge

$ http://debuggable.com/

Links

[1]: http://wiki.github.com/ry/node/ecma-5mozilla-features-implemented-in-v8
[2]: http://wiki.github.com/ry/node/streams

Bonus Slides!

Dirty

CouchDB
relax

JavaScript Views
Disk Persistence

❤

↓

Dirty

Redis

Memory Store
Speed > Safety

A scriptable key-value store

- Let your business logic and your data share the same memory / process
- Network = OVERHEAD - Avoid whenever possible
- V8 makes it very fast

How fast?

- Set: 3-5 million docs / sec

- Get: 40-50 million docs / sec

(on my laptop - your milage may vary)

Benchmarks

Do your own!

Disk persistence

- Append-only log
- Writes happen every x-Sec or every x-Records
- Callbacks fire after disk write succeeded

Dirty Hello World

hello.js

```
var
  Dirty = require('dirty').Dirty,
  posts = new Dirty('test.dirty');

posts.add({hello: 'dirty world!'});
posts.set('my-key', {looks: 'nice'});
```

```
$ node hello.js
$ cat test.dirty
{"hello":"dirty world!","_key":"3b8f86..."}
{"looks":"nice","_key":"my-key"}
```

Reloading from Disk

hello.js

```
var
  Dirty = require('dirty').Dirty,
  posts = new Dirty('test.dirty');

posts.load(function() {
  p(posts.get('my-key'));
});
```

```
$ node hello.js
{"looks": "nice", "_key": "my-key"}
```

Use Cases

- Small projects (db < memory)
- Rapid prototyping
- Add HTTP/TCP interface and scale

http://github.com/felixge/node-dirty

(or google for “dirty felixge”)