Webanwendungen mit Node.js

Tim Baumann
tim@timbaumann.info

26. März 2011

Wer hier benutzt PHP, Perl, Python oder Ruby für Webanwendungen?

Wer hier benutzt JavaScript?

Über Node.js

- Plattform für serverseitiges JavaScript auf Basis von Googles V8
- Das Projekt wurde 2009 von Ryan Dahl gestartet.
- Seitdem stark wachsende Popularität (momentan ca. 4000 User auf der Mailingliste).

nodejs.org

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

Hallo Welt!

Ein einfacher Webserver:

```
// http-Modul laden
var http = require('http');

var server = http.createServer(function(req, res) {
  // req = request  = Anfrageobjekt
  // res = response = Antwortobjekt
  res.writeHead(200,{'Content-Type':'text/plain'});
  res.end('Hallo Welt!');
});

// Auf Port 8000 lauschen
server.listen(8000);
```

Hallo Welt! (Forts.)

Ein Webserver, der erst nach einer Sekunde antwortet:

```
var http = require('http');

var server = http.createServer(function(req, res) {
  res.writeHead(200,{'Content-Type':'text/plain'});

  setTimeout(function() {
    // Fuehre diese Funktion nach 1s = 1000ms aus
    res.end('Welt!');
  }, 1000);
  res.write('Hallo ');
});

server.listen(8000);
```

Hallo Welt! (Forts.)

Ein einfacher Fileserver:

```
var http = require('http')
,   fs   = require('fs');

var server = http.createServer(function(req, res) {
  res.writeHead(200,
    { 'Content-Type': 'application/pdf' });

  // Lies die Datei "slides.pdf"
  fs.readFile('slides.pdf', function(err,contents){
    // Wirf einen eventuellen Lese-Fehler
    if (err) throw err;
    res.end(contents);
  });
});

server.listen(8000);
```

Blockierende I/O

```php
<?php

$schools  = mysql_query('SELECT * FROM schools ...');
$students = mysql_query('SELECT name FROM students ...');

// Mache irgendwas

?>
```

Die beiden Abfragen finden nacheinander statt. Die Gesamtzeit ist die Summe beider Abfragen.

Parallele Datenbank-Abfragen

```
// Lade das Datenbank-Modul
var db = require('example-db');

// Deklariere globale Variablen zum Speichern der Ergebnisse
var schools, students;

var processResults = function() {
  if (schools && students) {
    // Mache irgendwas
  }
};

// Starte die erste Abfrage
db.query('SELECT * FROM schools ...', function(err, rows) {
  // Fehlerbehandlung ...
  schools = rows;
  processResults();
});

// Starte die zweite Abfrage
db.query('SELECT name FROM students ...', function(err, rows) {
  students = rows;
  processResults();
});
```

Die Gesamtzeit ist das Maximum beider Abfragen.

Typische Dauer von I/O

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
...	...
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

Tabelle: Geklaut von „Teach Yourself Programming in Ten Years“

Warum?

- Gegenmodell: Ein Thread pro Verbindung
  - Zusätzlicher Speicherbedarf durch Thread Stacks
  - Performance-Overhead durch Context-Switching
- Volle Kontrolle über den Ablauf des Programms.
- Unblocking I/O ist leichter zu verstehen als Threads.

Probleme von „Unblocking I/O“

- Höhere Codekomplexität infolge der Event-basierten Programmierung
- Aufwendige Rechenoperationen blockieren den Event-Loop oder sollten auf mehrere Umläufe im Event-Loop verteilt werden. In Zukunft können Hintergrundprozesse mittels sogenannten „Worker“ stattfinden.

I/O-Aktionen

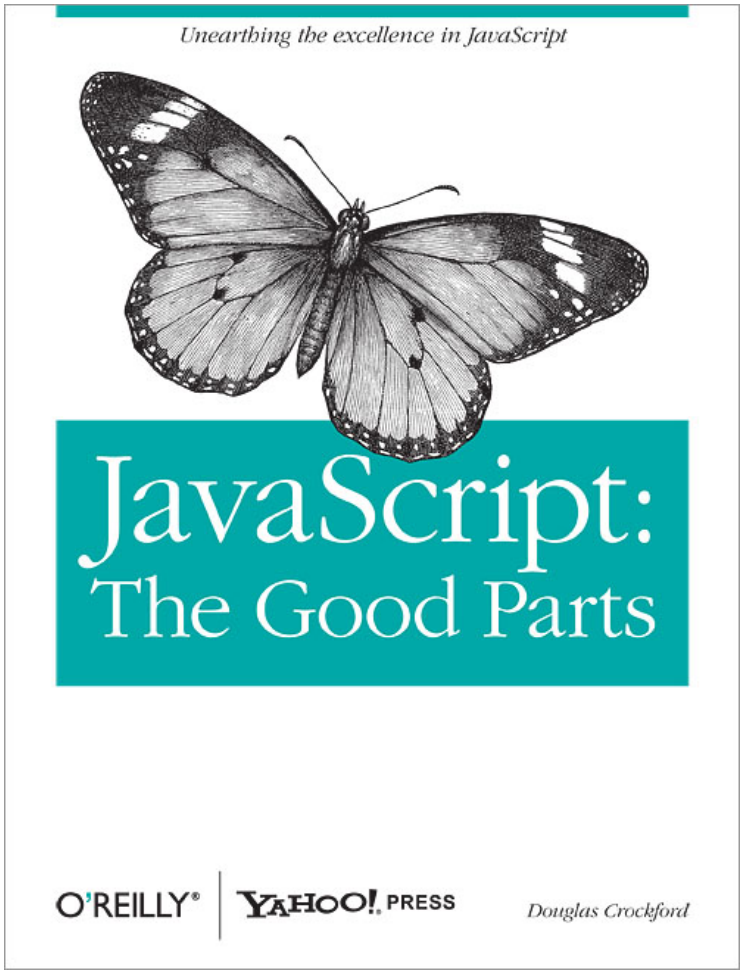
In der Standardbibliothek:

- Dateisystem
- TCP
- UDP
- HTTP(S)
- DNS
- Kind-Prozesse
- ...

Als Packete:

- MySQL
- FTP
- IRC
- IMAP
- XMPP
- PCAP
- ...

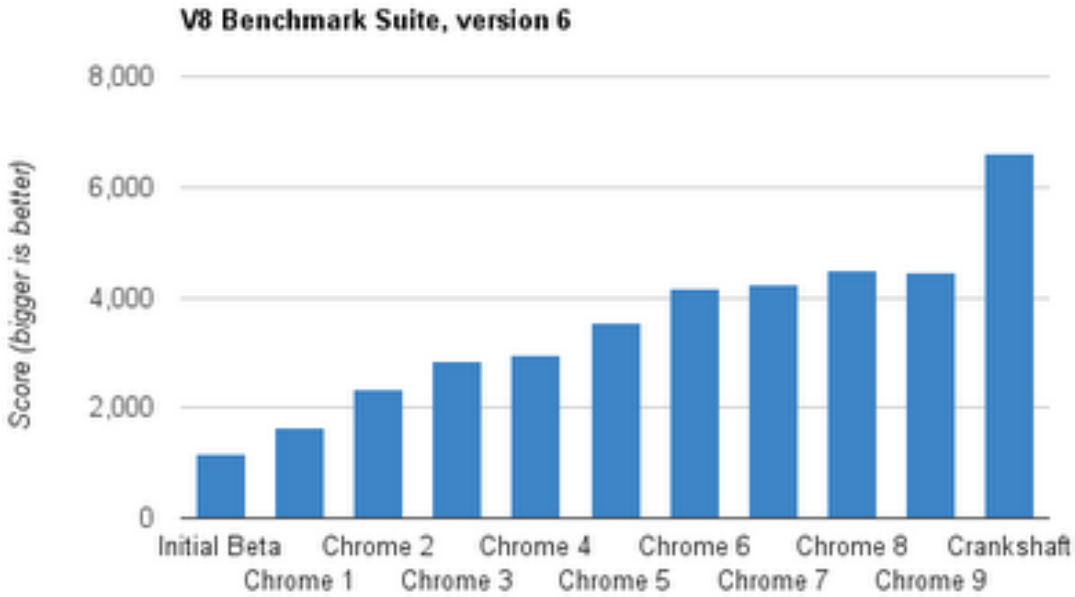
JavaScript hat seine guten Seiten!



- Das DOM ist gruselig, aber nicht Teil von JavaScript (ECMAScript).
- Closures
- First-class functions
- Prototypische Vererbung
- Douglas Crockford, JavaScript: The Good Parts (Das Beste an JavaScript)

Performance

- Browserhersteller: Kampf um die beste JS-Performance
- Immer noch einiges an Optimierungspotential



Bloß nicht blockieren!

```
var startTime = new Date().getTime();
setTimeout(function() {
  // soll eigentlich nach 50ms ausgefuehrt werden
  console.log(new Date().getTime() - startTime);
}, 50);
// Schleife, die fuer 100ms laeuft
while (new Date().getTime() < startTime + 100);
```

- Blockierender Code hält den Event-Loop auf.
- Vorteil von JavaScript: JavaScript hatte noch nie (blockierende) I/O.

Event-Handling im Browser


```
var button = document.getElementById('button');
button.addEventListener('click', function(event) {
  // mache irgendwas
}, false);
```

Wiederverwendbarkeit

- Don’t repeat yourself!
- JavaScript kann sowohl auf dem Client als auch auf dem Server ausgeführt werden.
- Beispiel: Formularvalidierung
- Der Browser auf dem Server: jsdom

Weiteres

- Buffer: Effiziente Speicherung von Binärdaten
- EventEmitter: Observer-Entwurfsmuster
- Stream:

```
var http = require('http');

http.createServer(function(req, res) {
  req.on('data', function(data) {
    process.stdout.write(data);
  });
  req.on('end', function() {
    res.end("The end.");
  });
}).listen(8000);
```

Weiteres

- Buffer: Effiziente Speicherung von Binärdaten
- EventEmitter: Observer-Entwurfsmuster
- Stream:

```
var http = require('http');

http.createServer(function(req, res) {
  req.pipe(process.stdout);
  req.on('end', function() {
    res.end("The end.");
  });
}).listen(8000);
```

CommonJS Modules

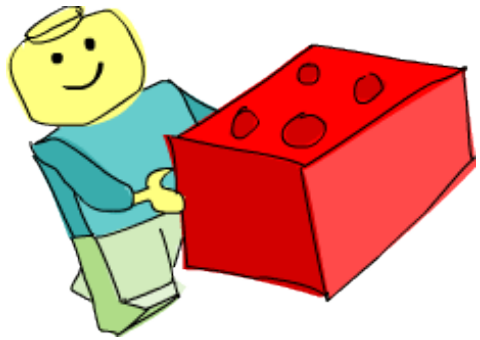
JavaScript kennt von sich aus keine Module. Die Initiative CommonJS definiert einen Standard:

```
var greeter = require('./greeter.js');
console.log(greeter.greet('LIT'));
```

Listing 1: foo.js

```
exports.greet = function(name) {
  return "Hallo, " + name + "!";
};
```

Listing 2: greeter.js



Beispiel: Onlinechat

Ajax

Asynchronous JavaScript and XML

Problem

- Anfragen können nur vom Browser aus initiiert werden.
- Man möchte eigentlich nicht nachfragen, sondern benachrichtigt werden, wenn etwas passiert.
- Wichtig für Online-Spiele und soziale Netzwerke

Long polling

1. Browser sendet eine Anfrage an den Server.
2. Server hält die Verbindung offen.
3. Wenn ein Ereignis eintritt, sendet der Server die relevanten Informationen und schließt die Verbindung.
4. Goto 1

$\Rightarrow$ Simulation von „Push“

Long polling

1. Browser sendet eine Anfrage an den Server.
2. Server hält die Verbindung offen.
3. Wenn ein Ereignis eintritt, sendet der Server die relevanten Informationen und schließt die Verbindung.
4. Goto 1

$\Rightarrow$ Simulation von „Push“

Long polling

1. Browser sendet eine Anfrage an den Server.
2. Server hält die Verbindung offen.
3. Wenn ein Ereignis eintritt, sendet der Server die relevanten Informationen und schließt die Verbindung.
4. Goto 1

$\Rightarrow$ Simulation von „Push“

Long polling

1. Browser sendet eine Anfrage an den Server.
2. Server hält die Verbindung offen.
3. Wenn ein Ereignis eintritt, sendet der Server die relevanten Informationen und schließt die Verbindung.
4. Goto 1

$\Rightarrow$ Simulation von „Push“

Long polling

1. Browser sendet eine Anfrage an den Server.
2. Server hält die Verbindung offen.
3. Wenn ein Ereignis eintritt, sendet der Server die relevanten Informationen und schließt die Verbindung.
4. Goto 1

$\Rightarrow$ Simulation von „Push“

Long polling

- Problem: Server muss mit viele Verbindungen gleichzeitig umgehen können.
- Mit einem Thread pro Verbindung geht das nur bedingt aufgrund von Context-Switching, Speicherbedarf von Threads etc.
- In Event-getriebenen Umgebungen braucht jede Verbindung nur etwas Speicherplatz.
- Zukunft: HTML5 WebSockets (bestehende Verbindung zwischen Client und Server)

index.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>Online-Chat</title>
</head>

<body>
  <ul id="messages"></ul>
  <input type="text" name="msg" id="input"
    placeholder="Was hast du zu sagen?">
  <script src="jquery.js"></script>
  <script src="chat.js"></script>
</body>

</html>
```

chat.js

```
$('#input').keydown(function(event) {
  // Wurde die Entertaste gedrueckt?
  if (event.keyCode == 13) {
    sendMessage($(this).val());
    $(this).val("");
  }
});

function sendMessage(msg) {
  $.ajax({
    url: '/sendmessage?msg='
         + encodeURIComponent(msg),
    dataType: 'text'
  });
}
```

chat.js

```
function getNextMessage() {
  $.ajax({
    url: '/nextmessage',
    dataType: 'text',
    success: function(msg) {
      if (msg) {
        $('<li />')
          .text(msg)
          .appendTo($('#messages'));
      }
      getNextMessage();
    }
  });
}

getNextMessage();
```

server.js

```
var http = require('http')
,   url  = require('url')
,   fs   = require('fs');

// Hier werden die offenen Verbindungen gespeichert
var connections = [];

http.createServer(function(req, res) {
  var parsed = url.parse(req.url, true)
  ,   path   = parsed.pathname
  if (path == '/') { path = '/index.html'; }
  console.log("Request: " + path);

  var staticFiles = ['/index.html', '/jquery.js', '/chat.js'];
  if (staticFiles.indexOf(path) != -1) {
    fs.createReadStream(__dirname + path).pipe(res);
  } else if (path == '/nextmessage') {
    connections.push({ time: new Date().getTime(), res: res });
  } else if (path == '/sendmessage') {
    sendMessage(parsed.query.msg);
    res.end('');
  }
}).listen(8000);
```

server.js

```
function sendMessage(msg) {
  connections.forEach(function(connection) {
    connection.res.end(msg);
  });
  connections = [];
}

// Alle 10 Sekunden alle Anfragen, die schon mehr
// als 30 Sekunden laufen, abbrechen, um Timeouts
// zu verhindern
setInterval(function() {
  var expiration = new Date().getTime() - 30000;
  connections = connections.filter(function(connection) {
    if (connection.time < expiration) {
      connection.res.end('');
      return false;
    }
    return true;
  });
}, 10000);
```

Demo

Beispiel: Steuerung einer Drone

Parrot AR.Drone

Parrot AR.Drone

Parrot AR.Drone

Technische Daten

- Ca. 12 Minuten Flugdauer
- Zwei Kameras nach vorne und nach unten
- Ultraschall-Höhenmesser
- Beschleunigungssensor
- iPhone-Steuerung
- Embedded Linux!

Steuerung

- Die Drone ist eine WLAN-Basisstation.
- Steuerung mit Ascii-Kommandos:

```
AT*REF=290718208
AT*PCMD=84,1,-1138501878,0,0,0
```

- Übertragung per UDP an Port 5556
- Am Besten mindestens 30 Mal pro Sekunde
- Die Drone sendet Sensordaten mit UDP-Multicast zurück.

Demo

```
$ npm install ardrone
$ npm install ardrone-web

$ git clone https://github.com/timjb/node-ardrone
```

Connect & Express

Connect: Middleware für Webserver:

- Static file serving
- Cookies
- Sessions
- GZIP
- Simple authentication
- Logging
- ...

Express: Sinatra-ähnliches Framework, das auf Connect aufbaut und vor allem flexibles Routing und Views bietet.

```
var express = require('express')
,   fs      = require('fs');

var app = express.createServer(
  express.logger(),
  express.static(__dirname+'/public')
);

app.configure(function() {
  app.set('views', __dirname + '/views');
});

app.get('/', function(req, res) {
  fs.readdir(__dirname+'/public', function(err,
      contents) {
    res.render('readdir.ejs', {contents:contents});
  });
});

app.listen(8000);
```

```
<ul>
<% contents.forEach(function(item) { %>
  <li>
    <a href="<%= item %>"><%= item %></a>
  </li>
<% }); %>
</ul>
```

Größere Anwendungen

- CoffeeScript: Sprache, die zu JavaScript kompiliert
- Cloud9: Webbasierte Entwicklungsumgebung
- StackVM: Virtuelle Maschinen im Web bedienen
- Nodester: Open Source Node.js Hosting
- ~~Palm~~ HP WebOS: Hintergrundservices
- scrabb.ly: Massive Multiuser Online Word Game

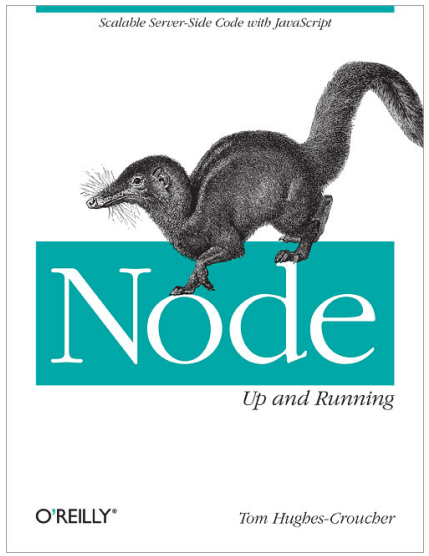
NPM – Node Packet Manager

Schon über 1400 Packete

Einstiegspunkte

- `http://nodejs.org/`
- **Wiki:** `https://github.com/joyent/node/wiki`
- **Zum Ausprobieren im Browser:** `http://jsapp.us/`
- **Buchankündigung: Up and running with Node.js, O'Reilly**
- **Installation:**

```
$ git clone https://github.com/joyent/node.git
$ cd node
$ ./configure
$ make && sudo make install
```



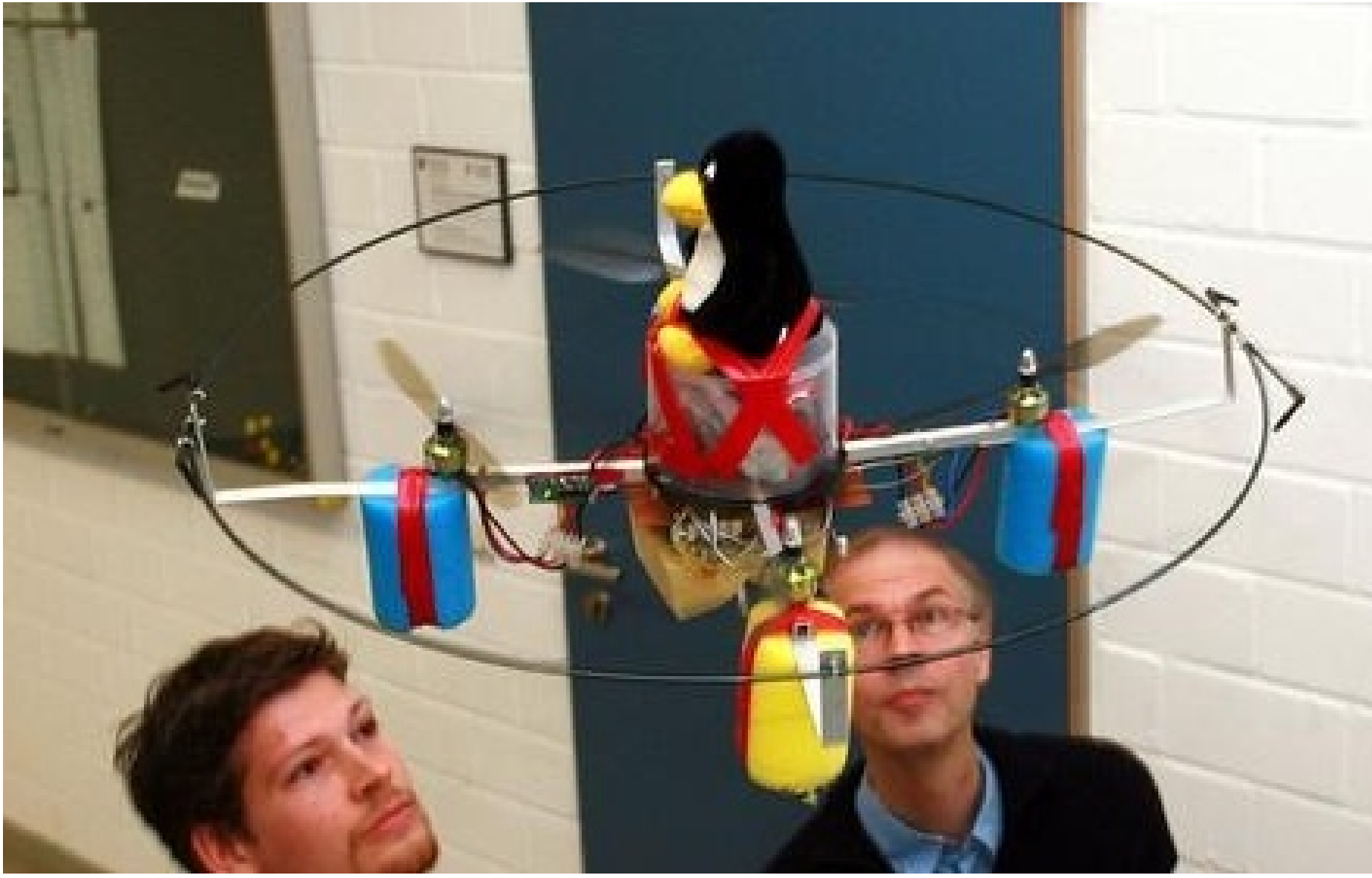
nodejs.org

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

Dankeschön!

Dankeschön!

Noch Fragen?



Unter der Haube

- libev: „full-featured and high-performance [. . .] event loop„ (benutzt epoll / kqueue / inotify intern)
- libeio: „full-featured asynchronous I/O library for C“
- c-ares: Asynchrone DNS-Auflösung

Für andere Programmiersprachen

- Python: Twisted
- Ruby: EventMachine
- Perl: IO::AIO
- Haskell: bald in GHC

Quellen

- http://norvig.com/21-days.html
- http://www.theregister.co.uk/2011/03/01/the_rise_and_rise_of_node_dot_js/page3.html
- http://ardrone.parrot.com/parrot-ar-drone/de/technologies

Bildquellen

- http://nodejs.org/logo.png
- http://www.whatbrowser.org/img/js-performance-over-time.png
- http://learnyouahaskell.com/input-and-output
- http://s3.amazonaws.com/lyah/modules.png
- http://covers.oreilly.com/images/9780596517748/lrg.jpg
- http://upload.wikimedia.org/wikipedia/commons/4/44/Recycle001.svg
- http://oktopoden.blogspot.com/2011/01/auf-ein-neues.html
- http://www.flickr.com/photos/tonynewell/582149665/
- http://c3connections.files.wordpress.com/2010/10/trafficjam.jpg
- http://covers.oreilly.com/images/0636920015956/lrg.jpg
- http://jashkenas.github.com/coffee-script/documentation/images/logo.png
- http://www.ajax.org/images/cloud9Logo.png
- http://stackvm.com/stackvm.png
- http://ardrone.parrot.com/press-photos/album-photo_jquery/parrot_ar.drone_05.jpg
- http://ardrone.parrot.com/press-photos/album-photo_jquery/parrot_ar.drone_01.jpg
- http://ardrone.parrot.com/press-photos/album-photo_jquery/parrot_ar.drone_02.jpg
- http://images.tecchannel.de/images/tecchannel/bdb/361900/361905/98222663D6FE9EA55EFFF46179D7C9B2_800x600.jpg