
node.JS

result = query(SELECT foo FROM bar)

Was geschieht während der Abfrage?

nichts! → verschenkte Cycles

L1:	3 cycles
L2:	45 cycles
RAM:	250 cycles

→ non-blocking I/O

Disk: 41.000.000 cycles

Network: 240.000.000 cycles

→ blocking I/O

besser: Multithreading

- Kontextwechsel kostet
- Execution Stacks kosten Speicher
- ein System Thread pro Connection verbraucht viel Speicher

noch besser?

event-driven, non-blocking infrastructure

node.js

- Asynchrone I/O
- Eventloop
- serverseitiges JavaScript
- Google v8 (Garbage Collection, Inline Caching, …)
- ECMAScript v5

query(SELECT foo FROM bar, callback)

node.js kehrt sofort nach dem function call zur eventloop zurück.

Nachdem die I/O operation abgeschlossen ist wird die callback function aufgerufen.

Warum JavaScript?!

- anonyme Funktionen
- Closures
- JS fügt sich nahtlos in das event-basierte Konzept von node.js ein

http://developer.yahoo.com/yui/theater/video.php?v=crockonjs-1

Douglas Crockford — Crockford on JavaScript

wodurch zeichnet sich node.js aus?

umfangreiche API

FileSystem (POSIX), HTTP, DNS, Crypto, TCP, Timer, ...

Module

sehr einfache Möglichkeit libs und frameworks für node.js zu erstellen

Jeder der JS kann, kann auch für node.js entwickeln.

extrem schnell

aber …

momentan noch sehr viele API Änderungen

nicht alles ist non-blocking

z.B. require()

callbacks die lange laufen machen das Programm langsam

was ist mit Multithreading?

zukünftig Web Workers API

http://nodejs.org/
http://github.com/ry/node/wiki

some Examples

http://www.ranzwertig.de/nodewebsocket/examples/doodle.html

http://github.com/ranzwertig/nodewebsocket

Christian Ranz

twitter.com/ranzwertig
www.christianranz.com