Das interessiert dich bestimmt auch

Einst abfällig als Spielzeug bezeichnet, ist JavaScript heute die am heißesten umkämpfte Sprache auf dem Markt. Mit Mozilla, Google, Microsoft und Apple ringen derzeit vier Schwergewichte um die beste Implementierung. Abseits der Arena profitiert Node.js von der rasanten Entwicklung und zeigt völlig neue Ansätze für das Entwickeln von Web- und Echtzeitanwendungen.

Die Idee, JavaScript auf dem Server auszuführen, ist fast so alt wie die Sprache selbst. Bereits 1996 wagte Netscape die ersten Schritte mit LiveWire – und viele andere folgten. Der Durchbruch blieb jedoch lange Zeit aus. Dies änderte sich am 8. November 2009, als Ryan Dahl sein Projekt Node.js [1] einem begeisterten Publikum auf der Konferenz JSConf.eu vorstellte. Inzwischen ist aus dem jungen Projekt eine interessante Plattform geworden, die sich vor allem durch Performance und Flexibilität auszeichnet. Mit mehr als 2.500 Mitgliedern in der schnell wachsenden Mailingliste ist Node.js inzwischen auf dem besten Weg, in den Mainstream vorzudringen und selbst eingefleischte Java-, PHP-, Ruby- und Python-Enthusiasten zu überzeugen.

Blitzschnelles JavaScript

Mit Googles V8-Engine setzt Node.js auf eine der derzeit schnellsten JavaScript-Implementierungen. Neben vielen Optimierungen zeichnet sich V8 vor allem dadurch aus, dass es JavaScript-Code vor dem Ausführen direkt in Maschinencode übersetzt. Dadurch katapultiert es JavaScript in eine bis dato für Skriptsprachen unerreichbare Performance-Region und ermöglicht den Einsatz in zahlreichen Bereichen, die bis dato C/C++ vorbehalten waren. Dies zeigt sich vor allem darin, dass Node.js zu großen Teilen selbst in JavaScript geschrieben ist.

Programmieren mal anders

Das Interesse an Node.js beruht im Kern auf einer für eine Plattform einzigartigen Eigenschaft. Obwohl im Laufe der Jahre verschiedenste Sprachen und Plattformen mit unterschiedlichsten Konzepten entwickelt wurden, hatten sie doch alle eine Eigenschaft gemein: I/O, das heißt der Zugriff auf Festplatte und Netzwerk, blockiert in allen gängigen Umgebungen standardmäßig den Programmablauf.

Dies führt dazu, dass zum Beispiel das Ausführen einer Datenbankabfrage zum kompletten Stillstand

des Programms führt. Die meisten Anwendungen verbringen daher große Teile ihrer Zeit mit dem Warten auf I/O-Operationen, die tausend bis eine Million Mal langsamer als CPU- oder RAM-Operationen ablaufen.

Die bisherige Lösung für dieses Problem hieß in der Regel Threading. Dabei wird das Programm in mehrere parallel laufende Unterprogramme zerteilt, so genannte Threads, die sich den gleichen Speicherbereich teilen. Das Problem bei diesem Modell ist allerdings der sichere Umgang bei der Programmierung. Durch das parallele Ablaufen einzelner Threads muss stets garantiert sein, dass bestimmte Speicherbereiche von nur jeweils einem Thread manipuliert werden, da anderenfalls Datenkorruption eintritt. Dies hat dazu geführt, dass die korrekte Benutzung von Threads selbst Experten erhebliche Schwierigkeiten bereitet.

Das schlichte Logo täuscht: Unter Early Adoptern gilt Node.js als die spannendste neue Technologie seit dem erscheinen von Ruby on Rails.

Node.js greift dieses Problem auf, indem es auf so genanntes „non-blocking I/O“ setzt. Das bedeutet in der Praxis, dass alle Funktionen, die I/O ausführen, eine Callback-Funktion als Parameter erwarten. Diese wird erst dann aufgerufen, wenn die Operation im Hintergrund ausgeführt wurde. Der Programmablauf als solcher wird dadurch nicht zum Stillstand gebracht. Entwickler von AJAX-Anwendungen werden dieses Konzept sehr vertraut finden, da es sich im Grunde kaum von der Programmierung von clientseitigen JavaScript-Anwendungen unterscheidet.

Um den Reiz dieses Ansatzes zu verstehen, muss man wissen, dass JavaScript immer in nur einem einzigen Thread ausgeführt wird. Callback-Funktionen werden also nur ausgeführt, wenn gerade kein anderer Code an der Reihe ist. Entwickler von JavaScript- und Node.js-Anwendungen müssen sich daher nie Gedanken um den sicheren Zugriff auf ihre Objekte machen.

Ungelöst bleibt hierbei die Verteilung einer Anwendung auf mehrere CPU-Kerne. Hierfür setzt

Node.js darauf, mehrere Prozesse zu starten, die über klassische Unix-Domain-Sockets miteinander kommunizieren.

Schlanker Kern und viele Module

Im Gegensatz zu anderen Umgebungen kommt Node.js mit einem sehr schlanken Paket ohne externe Abhängigkeiten aus, denn nur das Nötigste wird mitgeliefert. Dazu zählen vor allem Module zur Verwendung von TCP, UDP, Kryptographie, HTTP, DNS sowie des Dateisystems.

Darüber hinaus kann man mit Hilfe von npm [2] inzwischen auf eine große Auswahl von Community-Modulen zurückgreifen. Darunter finden sich unter anderem Treiber für diverse Datenbanken, Frameworks sowie zahlreiche andere Module, die Entwicklern bei der Programmierung von Webanwendungen hilfreich sein können.

Zum Testen von Node.js benötigt man eine Unix-artige Umgebung. Dazu eignet sich am besten Linux oder Mac OS X, unter Windows muss man auf cygwin zugreifen. Zur Installation muss man lediglich das heruntergeladene Paket kompilieren, anschließend kann es losgehen. Wenn alles funktioniert, sollte man eine Datei namens „hello.js“ mit dem Inhalt „console.log('Hello World')“ mit dem Befehl „node hello.js“ ausführen können.

Das geht

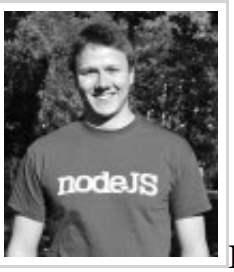
Prinzipiell eignet sich Node.js für die Entwicklung jeglicher Web- und Serveranwendungen. Frameworks wie Express [3] oder fab.js [4] erleichtern den Einstieg und stellen die nötigsten Hilfsmittel zur Verfügung. Besonders interessant wird Node.js aber vor allem bei den Themen Streaming und Echtzeit. So kann man mit Node.js zum Beispiel wunderbar Datei-Uploads als Stream empfangen – der Parser [5] schafft bis zu 500 MB/s – und diesen dann direkt an ein anderes Ziel weiterleiten (z. B. den Standardinput einer Kommandozeilenanwendung).

Außerdem kann man mit Node.js bewusst Code zwischen Browser und Server wiederverwenden. So läuft zum Beispiel die aus jQuery bekannte Selector-Engine Sizzle [6] in Node.js und mit Hilfe anderer Projekte [7] lassen sich einzelne Node.js-Module wiederum im Browser laden. Damit lässt sich zum Beispiel die lästige Duplizierung von Validierungscode komplett vermeiden. Die Homepage [8] verweist auf weitere Beispiele sowie Dokumentationen.

Obwohl Node.js eine noch recht junge Technologie ist, reihen sich schon erste größere Projekte in die Liste der Fallbeispiele. So verkündete Palm kürzlich, dass Node.js fester Teil des neuen webOS 2.0 ist und damit allen Anwendungsentwicklern zur Verfügung steht. Außerdem entstehen derzeit weitere

spannende Node.js-Anwendungen bei Teams wie Yahoo!Mail, GitHub und Joyent.

Der Autor



Felix Geisendörfer ist Mitbegründer von transloadit.com, einem auf Node.js basierenden Service für den Upload und die Enkodierung von Bildern und Videos. Darüber hinaus beteiligt er sich seit 2009 aktiv an der Entwicklung von Node.js und bietet mit seiner Berliner Firma Debuggable Dienstleistungen rund um Node.js an.

Original URL:
http://t3n.de/magazin/nodejs-javascript-server-revolutioniert-schubrakete-226177/