1 Hello CoffeeScript

»Anyway, I know only one programming language worse than C and that is JavaScript.«

Robert Cailliau
einer der »Erfinder« des WWW

JavaScript erlebt zurzeit eine Renaissance, denn JavaScript ist überall. Auf nahezu jedem Computer dieser Welt gibt es mindestens einen JavaScript-Interpreter. Selbst Microcontroller wie den Espruino[1] kann man mittlerweile mit JavaScript programmieren. JavaScript treibt das Web 2.0 an. JavaScript wird zur Entwicklung für mobile Endgeräte verwendet. JavaScript ist auf dem Server angekommen. Aber JavaScript ist nicht schön. Und, wie Douglas Crockford schon bemerkte, eine der am häufigsten missverstandenen Programmiersprachen der Welt.

In zehn Tagen Mitte der Neunzigerjahre von Brendan Eich bei Netscape in aller Eile entwickelt, ist JavaScript tatsächlich eine missverstandene Sprache. Angefangen beim Namen, der eine (nicht vorhandene) Nähe zu Java vermuten lässt, über die C-ähnliche Syntax, die vermuten lässt, JavaScript sei eine prozedurale Sprache (dabei ist JavaScript funktionalen Sprachen wie Scheme oder Lisp deutlich näher), bis hin zu den vor allem anfangs äußerst fehlerbehafteten Implementierungen – JavaScript hat es Entwicklern nicht leicht gemacht.

Vieles hat sich seit den Anfängen verbessert, der Sprachstandard wird von den meisten Implementierungen mittlerweile eingehalten. Dennoch gibt es Raum für Verbesserungen – Raum, den CoffeeScript füllen will.

CoffeeScript ist eine relativ junge Sprache[2], die nach JavaScript kompiliert wird. CoffeeScript hat eine goldene Regel: Es ist einfach JavaScript. Der Code, der vom CoffeeScript-Compiler erzeugt wird, ist valides JavaScript, das die Tests von Douglas Crockfords JavaScript-Lint JSLint erfolgreich absolviert. Er läuft auf jeder JavaScript-Implementierung, und das sehr schnell, meistens sogar schneller, als selbst geschriebener JavaScript-Code ablaufen würde.

1. *http://www.espruino.com*
2. Die erste Version 0.1 erschien am 24. Dezember 2009, die Version 1.0 genau ein Jahr später.

Vor einigen Jahren ist ein Hype rund um CoffeeScript entstanden, der sich nicht zuletzt durch einen Tweet von David Heinemeier Hansson erklären lässt, der im April 2011[3] äußerte, dass die neue Ruby-on-Rails-Version 3.1 CoffeeScript enthalten würde. Mit der Veröffentlichung am 31. August 2011 kamen auf einmal viele Ruby-on-Rails-Entwickler mit dieser neuen Sprache in Berührung.

Heute, rund fünf Jahre nach seiner Entstehung, ist CoffeeScript in vielen großen Projekten angekommen, zum Teil wegen seiner Integration in Ruby on Rails, aber nicht nur, wie etwa die Online-Shopping-Plattform Shopify zeigt, die CoffeeScript in ihrem Frontend einsetzt.

Doch was macht CoffeeScript so besonders, dass immer mehr Entwickler beginnen, sich mit dieser Sprache zu beschäftigen?

Am einfachsten versteht man das, wenn man sich ein paar Beispiele dazu ansieht.

1.1 Appetithäppchen – oder: die Eleganz von CoffeeScript

Im Folgenden werde ich ein paar Codeschnipsel präsentieren und Ihnen zeigen, welchen JavaScript-Code der Compiler daraus erzeugt. Der erzeugte JavaScript-Code ist im Übrigen JavaScript, das, wie schon erwähnt, alle JavaScript-Lint-Tests positiv durchlaufen wird und so »gut« ist, dass es auf den meisten Plattformen laufen wird, also auch in älteren Browsern.

Doch nun zu den Beispielen. Es wird immer erst der generierte JavaScript-Code gezeigt, im Anschluss das CoffeeScript, aus dem dieser Code entstand.

Bitte lassen Sie sich nicht durch die Beispiele erschrecken, wenn Sie bisher noch nicht mit JavaScript gearbeitet haben. Es geht hier nicht um die Details, es geht darum zu zeigen, wie viel eleganter oder kompakter CoffeeScript im Vergleich zu JavaScript erscheint.

Gibt es dich? – Der Existential-Operator

Wie kann man in JavaScript überprüfen, ob es ein Objekt/eine Variable gibt, sprich, ob sie existiert?

```
if (typeof weihnachtsmann !== "undefined" && weihnachtsmann !== null) {
  console.log("Den Weihnachtsmann gibt es!");
}
```

Wie man sehen kann, ist es in JavaScript nicht ganz einfach, die Existenz zum Beispiel einer Variablen zu überprüfen, da sowohl der Typ als auch der Inhalt überprüft werden müssen.

3. *http://twitter.com/dhh/status/58207700672200704*

Nun zur Lösung in CoffeeScript:

```
console.log "Den Weihnachtsmann gibt es!" if weihnachtsmann?
```

Der aus Ruby bekannte »Existential-Operator« ist eine elegante Möglichkeit, die Existenz eines Objekts oder einer Variablen zu überprüfen.

Ebenso kann man an diesem kleinen Beispiel noch zwei weitere CoffeeScript-Eigenschaften erkennen: Sogenannter »Syntactic Noise« wie Klammern kann (meist) weggelassen werden und es gibt die ebenfalls aus modernen Skriptsprachen wie Python oder Ruby bekannte Postfix-Form des `if`, d.h. am Ende einer Zeile stehend.

Ich will nicht alles – bestimmte Elemente eines Arrays filtern

Wie kann man in JavaScript nur bestimmte Elemente eines Arrays filtern, also für jedes Element überprüfen, ob es ein bestimmtes Kriterium erfüllt, und nur im Ja-Fall dieses Element zurückliefern, um so ggf. ein neues – gefiltertes – Array zu erhalten?

In unserem Beispiel sollen alle Elemente eines mit Strings gefüllten Arrays zurückgegeben werden, die mit einem »S« beginnen. Es gäbe sicher auch noch elegantere Möglichkeiten, in JavaScript über ein Array zu iterieren, allerdings ist die aus CoffeeScript kompilierte Version wirklich auf nahezu jeder JavaScript-Umgebung lauffähig[4].

```
var name, names;

names = ["Schubert", "Schneider", "Müller", "Meier"];

names = (function() {
  var _i, _len, _results;
  _results = [];
  for (_i = 0, _len = names.length; _i < _len; _i++) {
    name = names[_i];
    if (name.indexOf("S") === 0) {
      _results.push(name);
    }
  }
  return _results;
})();

console.log(names);
```

Und nun die Variante in CoffeeScript:

```
names = ["Müller","Hubert","Schneider","Schmidt"]
names = ( name for name in names when name.indexOf("S") is 0 )
console.log names
```

4. Die Variante `Array.forEach(function(element))` läuft zum Beispiel nicht im Internet Explorer 8 und darunter.

Dieses Beispiel zeigt sehr schön, wie verständlich und kompakt CoffeeScript im Vergleich zu JavaScript wirkt. Auch ohne dass Sie bisher Kontakt zu CoffeeScript hatten, sollten die drei Zeilen CoffeeScript durchaus verständlich sein: Es werden in einem Array mit Namen nur die Elemente gesucht, die mit einem »S« beginnen.

Der resultierende JavaScript-Code ist mit 14 Zeilen deutlich länger und bei Weitem nicht so verständlich, vor allem für jemanden, der bisher noch kein JavaScript gesehen hat.

Sollten Sie diese kleinen Appetithäppchen ermuntert haben, mehr mit CoffeeScript zu experimentieren, dann soll es im Weiteren erst einmal darum gehen, was man benötigt, um mit CoffeeScript entwickeln zu können.

1.2 Aller Anfang ist leicht

Für das Nachvollziehen der beiden Codebeispiele auf den vorangegangenen Seiten oder die ersten Schritte mit CoffeeScript benötigen Sie nichts weiter als einen Webbrowser mit aktiviertem JavaScript.

Ein Besuch auf der Homepage von CoffeeScript gibt Ihnen neben einem Überblick über die Sprache auch die Möglichkeit, CoffeeScript interaktiv im Browser zu testen. Gehen Sie dazu einfach auf die Seite http://coffeescript.org und klicken Sie dort im Menü am oberen Seitenrand auf »Try CoffeeScript«. Sie gelangen dann auf eine Seite, auf der Sie im linken Teil CoffeeScript eingeben können und sofort und während des Tippens sehen, was für JavaScript-Code daraus generiert wird. Diesen JavaScript-Code können Sie dann durch einen Klick auf den »Run«-Button auch gleich ausführen lassen.

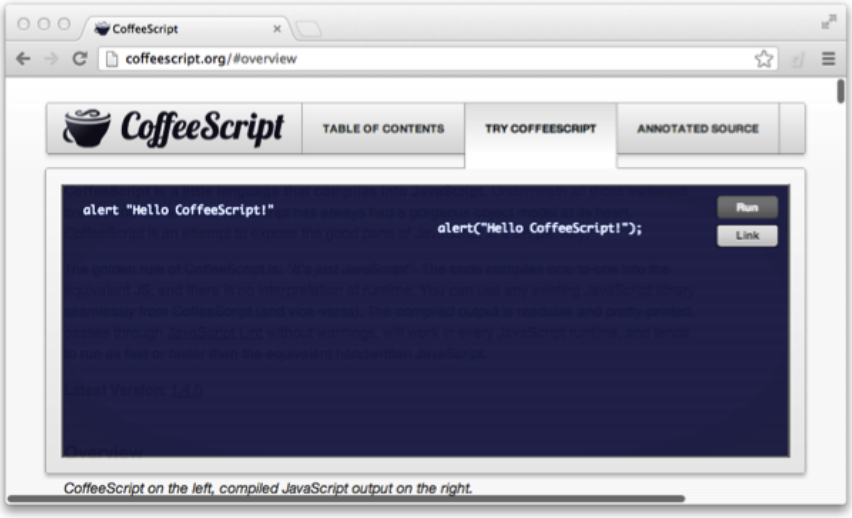

Abb. 1–1 *Try CoffeeScript auf der offiziellen Webseite*

CoffeeScript direkt in einer HTML-Datei

Die Funktionalität des »Try CoffeeScript« auf der offiziellen Webseite wird durch eine kleine Version des CoffeeScript-Compilers in JavaScript zur Verfügung gestellt, den man auch selbst verwenden kann:

```
<html>
<head>
   <script src="http://coffeescript.org/extras/coffee-script.js"
      type="text/javascript"></script>
<head>
<body>
   <script type="text/coffeescript">
      console.log "Hello CoffeeScript"
   </script>
</body>
</html>
```

Listing 1–1 *CoffeeScript direkt in der HTML-Datei*

Für erste Schritte oder auch ein Experiment zwischendurch sind dies sehr schöne Möglichkeiten, um mit CoffeeScript zu arbeiten. Für die ernsthafte Entwicklung benötigt man aber natürlich eine lokale Installation von CoffeeScript. Und um diese soll es nun gehen.

1.3 Den Kaffee auf den Tisch – oder: Wie installiere ich CoffeeScript?

CoffeeScript lässt sich am einfachsten über den Package-Manager für Node Packaged Modules (kurz npm) installieren. Er ist Bestandteil von Node.js, einer JavaScript-Plattform für serverseitiges JavaScript, um schnelle und skalierbare Netzwerkanwendungen oder Konsolenanwendungen zu entwickeln.

Als Erstes geht es also darum, Node.js zu installieren. Da sich die Installation für unterschiedliche Betriebssysteme unterschiedlich gestaltet, hier ein paar spezifische Hinweise.

Linux

Wenn Sie mit Linux arbeiten, dann empfiehlt es sich, den distributionseigenen Paketmanager zu verwenden, um Node.js zu installieren. Die meisten aktuellen Distributionen bringen eine Version von Node.js mit oder es gibt ein Repository von einem Drittanbieter.

Unter Ubuntu 11.4 reicht ein

```
sudo apt-get install nodejs
```

um Node.js zu installieren. Eine Übersicht mit Hinweisen zu anderen Distributionen ist auf der offiziellen Github-Seite von Node.js im Wiki[5] zu finden.

Windows

Unter Windows ist die Installation ganz einfach, da es von Node.js ein Installationsprogramm gibt, das einfach ausgeführt werden kann. Nach der Installation kann man sich einen sogenannten Node.js Command Prompt starten, eine Eingabeaufforderung mit aktivem Node.js.

Mac OS X

Unter Mac OS X empfehle ich nicht die Installation von Node.js über das aus der Webseite erhältliche Paket, vielmehr hat es sich als sehr praktikabel erwiesen, den alternativen Paketmanager Homebrew[6] zu verwenden.

Dieser Weg hat den Vorteil, dass Sie dann gleich einen Paketmanager installiert haben, der die Installation von vielen nützlichen Unix-/Open-Source-Tools erheblich erleichtert. Und bei der Arbeit mit CoffeeScript oder der Kommandozeile werden Sie sicher noch das eine oder andere Tool brauchen.

Um nun Homebrew zu installieren, reicht in einem Terminal die Eingabe von

```
ruby -e "$(curl -fsSkL raw.github.com/mxcl/homebrew/go)"
```

um die Installation zu starten.

Nachdem Homebrew erfolgreich installiert wurde, reicht ein

```
brew install node
```

um die Installation von Node.js anzustoßen.

Sollten Sie sich nicht für diesen Weg entscheiden, steht es Ihnen natürlich frei, den Installer der Webseite zu verwenden.

Und nun für alle Plattformen gleich: die Installation von CoffeeScript

Ein erfolgreich installiertes Node.js enthält ja, wie bereits erwähnt, den Package-Manager für Node Packaged Modules, den wir verwenden werden, um CoffeeScript zu installieren. Dabei würden eventuell benötigte Abhängigkeiten durch npm automatisch aufgelöst und ebenfalls installiert werden.

Die eigentliche Installation ist dann ein Einzeiler:

```
npm –g install coffee-script
```

---

5. *https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager*
6. *http://mxcl.github.com/homebrew/*

Das war's! Die Option `-g` sorgt dafür, dass npm das Paket global und Binaries unter einem allgemeinen Pfad (etwa `/usr/local/bin`) installiert. Dafür kann es nötig sein, dass Sie administrative Rechte benötigen (etwa durch Voranstellen von `sudo`).

Je nach Plattform und Voraussetzungen reichen also wenige Zeilen (meist weniger als drei) in einem Terminal, und schon ist CoffeeScript lokal in der aktuellen Version installiert. Auffälligste Änderung am System nach der Installation ist das Vorhandensein eines neuen Befehls: `coffee`. Dieser stellt die zentrale Schnittstelle zwischen Entwickler und Programmiersprache dar und bietet einige Optionen, die im folgenden Abschnitt dargestellt werden sollen. Ebenso darf natürlich ein Blick auf Editoren und Entwicklungsumgebungen nicht fehlen.

(Ih)git(t)? – am besten installieren Sie Git gleich dazu

Wir werden im Laufe des Buches immer wieder an den Punkt kommen, dass bestimmte Tools, Bibliotheken oder Beispiele aus dem Open-Source-Umfeld benötigt werden. Und viele dieser Projekte verwenden zur Versionskontrolle mittlerweile Git[7] als Tool der Wahl, sodass es eine gute Idee ist, dieses Tool ebenfalls zu installieren. Ich empfehle daher den Besuch der Git-Webseite und die Installation von Git. Wir werden immer wieder Gebrauch davon machen.

Die Installation ist für unterschiedliche Betriebssysteme ähnlich der Installation von CoffeeScript selbst. Wenn vorhanden, empfehle ich einen Paket-Manager (wie zum Beispiel Homebrew auf dem Mac[8]).

Für Windows gibt es ein Installationsprogramm. Für Unix-Systeme ohne eine vorkompilierte Version oder eigenen Paket-Manager gibt es natürlich auch die Möglichkeit, Git aus dem Quellcode zu übersetzen und zu installieren.

1.4 Kaffee – nicht nur zum Dessert

Für die allerersten Schritte nach der lokalen Installation empfiehlt sich eine interaktive CoffeeScript-Sitzung. Dazu geben Sie in einem Terminal einfach folgenden Befehl ein:

```
coffee
```

Danach ändert sich der Prompt in `coffee>` und Sie können nach Lust und Laune CoffeeScript-Befehle eingeben, die dann nach JavaScript kompiliert und gleich in der Node.js-Umgebung ausgeführt werden.

---

7. *http://git-scm.com*
8. Git lässt sich bei installiertem XCode auch darüber bequem installieren, falls Sie nicht den Weg über Homebrew gehen möchten.

Daher ist ein Hello-World-Programm unter dieser interaktiven Sitzung ein Einzeiler:

```
coffee> console.log "Hello World!"
```

Die interaktive CoffeeScript-Sitzung eignet sich für den Anfänger hervorragend, um kleine CoffeeScript-Schnipsel direkt auszuführen und so einen Eindruck der Sprache zu bekommen.

Allerdings kann man das Binary `coffee` noch mit deutlich mehr Optionen aufrufen. Grundsätzlich erwartet `coffee` nach den Optionen einen Datei- oder Verzeichnisnamen, der die auszuführenden oder zu kompilierenden CoffeeScript-Dateien angibt. Folgende Tabelle gibt einen kleinen Überblick über die wichtigsten Optionen:

Option	Funktion
-c, --compile	Kompiliert eine .coffee-Datei in eine .js-Datei gleichen Namens: coffee -c hello_world.coffee
-o, --output [DIR]	Schreibt alle kompilierten JavaScript-Dateien in das angegebene Verzeichnis: coffee -o js/ -c coffe/hello_wold.coffee
-j, --join [FILE]	Vor dem Kompilieren werden alle Skripte in der angegebenen Reihenfolge zusammengefügt und in die benannte Datei kompiliert. Nützlich, um größere Projekte zu bauen: coffee -j large_project.js -c 01.coffee 02.coffee 03.coffee Das Ergebnis ist eine Datei large_project.js
-w, --watch	Überwacht Dateien auf Änderungen und führt das angegebene Kommando bei Neuerungen durch: coffee -w hello_world.coffee Führt jedes Mal nach einer Änderung die Datei hello_wold.coffee aus. coffee -w -o js -c coffee/*.coffee Überwacht alle Dateien mit der Endung .coffee im Verzeichnis coffee und kompiliert sie in ein Verzeichnis js.
-e, --eval	Führt den angegebenen CoffeeScript-Code direkt aus: coffee -e "console.log 'Hello World'"
-h, --help	Zeigt alle Optionen des coffee-Binary an: coffee -h
--nodejs	Alle Optionen nach dieser Option werden direkt an das node-Binary weitergegeben. Nützlich, um node-Optionen setzen zu können.

Tab. 1–1 *Die wichtigsten Optionen des coffee-Binarys*

Es gibt noch einige weitere Optionen, um etwa das kompilierte JavaScript nicht in eine Datei, sondern auf `STDOUT` zu schreiben (`-p, --print`). Eine Übersicht aller Optionen finden Sie mithilfe der Option `-h` (bzw. `--help`).

Wie sag' ich's dir? Editoren/IDEs für CoffeeScript

Jeder Programmierer, der eine neue Sprache lernen soll, findet sich ziemlich schnell bei der Frage nach einem passenden Editor oder einer IDE, mit der er in der neuen Sprache programmieren soll.

Die aktuellste Übersicht bietet hierbei sicherlich das CoffeeScript-Wiki auf Github[9]. Für viele bekannte Editoren gibt es Plug-ins, etwa VIM, Emacs, TextMate oder jEdit. Ebenso sind einige IDEs vertreten, z.B. RubyMine oder Eclipse.

Ich persönlich verwende TextMate auf dem Mac mit entsprechendem Plug-in. Das funktioniert hervorragend.

Plattformübergreifend unter Windows, Mac OS X und Linux verfügbar ist jEdit oder Sublime Text 2.

Welchen Editor bzw. welche IDE Sie verwenden, ist sicher Ihrem persönlichen Geschmack und der verwendeten Entwicklungsplattform geschuldet. Werfen Sie einfach einen Blick auf die Wiki-Seite, vielleicht gibt es ja bereits für die von Ihnen bisher bevorzugte IDE ein Plug-in oder Ihr favorisierter Editor bietet schon Unterstützung für CoffeeScript.

1.5 Der (Fehler-)Teufel steckt im JavaScript

Gleich zu Beginn des Buches soll auch noch auf eine Schwierigkeit bei der Entwicklung mit CoffeeScript hingewiesen werden: die Fehlersuche.

Ein grundlegendes Problem von Sprachen, die nach JavaScript kompilieren, ist die Tatsache, dass Fehlermeldungen sich ja auf das kompilierte JavaScript beziehen, nicht auf den Quellcode.

Es gibt nun mehrere Möglichkeiten, mit diesem Problem umzugehen. Eine wäre, es überhaupt nicht als Problem zu sehen, sondern als Herausforderung. Also, wenn Sie von einer JavaScript-Umgebung zur Laufzeit einen Fehler bekommen, dann bezieht sich das ja auf das kompilierte JavaScript. Wenn Sie sich diesen Code anschauen (und vor allem die Zeilen rund um den Fehler), dann ist es im Normalfall und mit etwas Übung auch kein Problem, die entsprechende Zeile im CoffeeScript-Quellcode zu finden.

Nehmen wir dazu folgendes – zugegebenermaßen – einfache Beispiel: Es handelt sich dabei um das Array-Beispiel vom Anfang dieses Kapitels, bei dem sich nun ein einfacher Tippfehler eingeschlichen hat.

```
names = ["Müller","Hubert","Schneider","Schmidt"]
names = ( name for name in names when name.indeOf("S") is 0 )
console.log names
```

Listing 1–2 *fehlerhaft.coffee – ein fehlerbehaftetes Beispiel*

9. *https://github.com/jashkenas/coffee-script/wiki/Text-editor-plugins*

Anstelle von `name.indexOf` steht hier nun `name.indeOf`. Dieses Skript lässt sich mit `coffee -c fehlerhaft.coffee` anstandslos kompilieren. Ein Aufruf des Kompilats mittels `node fehlerhaft.js` liefert jedoch folgende (gekürzte) Fehlermeldung:

```
fehlerhaft.js:12
   if (name.indeOf("S") === 0) {
      ^
TypeError: Object Müller has no method 'indeOf'
   at fehlerhaft.js:12:16
   at Object.<anonymous>
   ...
```

Interessant ist dabei die markierte Stelle, die uns die Zeile (und Spalte) im JavaScript-Code zeigt, die fehlerhaft ist. Ein Blick in das kompilierte Programm zeigt uns folgendes Bild:

```
// Generated by CoffeeScript 1.7.1
(function() {
   var name, names;

   names = ["Müller", "Hubert", "Schneider", "Schmidt"];

   names = (function() {
      var _i, _len, _results;
      _results = [];
      for (_i = 0, _len = names.length; _i < _len; _i++) {
         name = names[_i];
         if (name.indeOf("S") === 0) {
            _results.push(name);
         }
      }
      return _results;
   })();

   console.log(names);

}).call(this);
```

Listing 1–3 *fehlerhaft.js – das kompilierte, aber fehlerhafte JavaScript-Beispiel*

Die fehlerhafte Zeile ist also die, in der `name.indeOf("S")` aufgerufen wird. Diese Zeile korrespondiert mit der zweiten Zeile unseres CoffeeScript-Skripts. Nach einer entsprechenden Korrektur läuft das Skript dann auch wieder einwandfrei.

Sie sehen also, dass es mit etwas Übung nicht allzu schwer ist, durch einen Blick in den kompilierten JavaScript-Code den Fehler und dann die korrespondierende Zeile im CoffeeScript-Code zu finden.

Allerdings ist diese Methode weder elegant noch das, was man aus anderen Programmiersprachen gewohnt ist. Da CoffeeScript nicht die einzige Sprache ist, die nach JavaScript kompiliert wird, und es somit einen gewissen Druck gibt,

wird auch schon an einer Lösung gearbeitet. Weiterhin stehen selbst JavaScript-Entwickler vor ähnlichen Problemen, denn meist wird JavaScript ja nicht in der gleichen Form ausgeliefert, wie es entwickelt wird. Der Code wird häufig minifiziert und zusammengefasst, um Ladezeiten und Performanz zu optimieren. Auch hier sind Fehlermeldungen dann natürlich auf Zeilennummern bezogen, die mit dem Original-Quellcode nichts gemeinsam haben. Hier kann man sich dann zwar dadurch helfen, dass der Fehler mit der originalen Version des JavaScript-Codes nachgestellt wird, aber auch das ist weder elegant noch immer praktikabel. Eine bessere Lösung wäre also wünschenswert.

Das Stichwort sind hierbei die sogenannten *Source-Maps*[10], einem von Google und Mozilla vorangetriebenen Vorschlag, genau dieses Problem zu lösen. Wenn Sie sich für diese Art des Debuggings näher interessieren, schauen Sie sich einfach das entsprechende Kapitel im Anhang genauer an.

Ein paar letzte Worte zu Beginn

Nachdem wir uns im ersten Kapitel mit einigen grundlegenden Aspekten der Sprache CoffeeScript beschäftigt haben – der Installation, ein paar ersten Schritten, Editoren/IDEs sowie der Fehlersuche –, werden wir uns im nächsten Kapitel eingehend mit der Sprache beschäftigen.

Dabei wird es um eine möglichst vollständige Beschreibung der Syntax und der Sprachfeatures gehen, die anhand vieler Beispiele beleuchtet werden sollen. Ich empfehle Ihnen, das zweite Kapitel vollständig durchzuarbeiten, ebenso wie das dritte Kapitel, in dem es um die Objektorientierung gehen wird.

In den darauffolgenden Kapiteln werden dann weitere Aspekte der Sprache vorgestellt, wie etwa das Testen oder das Zusammenspiel mit Ruby on Rails. Auch AngularJS wird in einem eigenen Kapitel behandelt.

Diese Kapitel bilden in sich abgeschlossene Einheiten, die nicht zwingend in der Reihenfolge des Buchs gelesen werden müssen. Es ist also Ihnen überlassen, sich die Kapitel auszusuchen, die Sie besonders interessieren.

Und nun: hinunter in den Kaninchenbau[11].

10. *http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/*
11. Frei nach »Alice im Wunderland« (*http://de.wikipedia.org/wiki/Alice_im_Wunderland*)